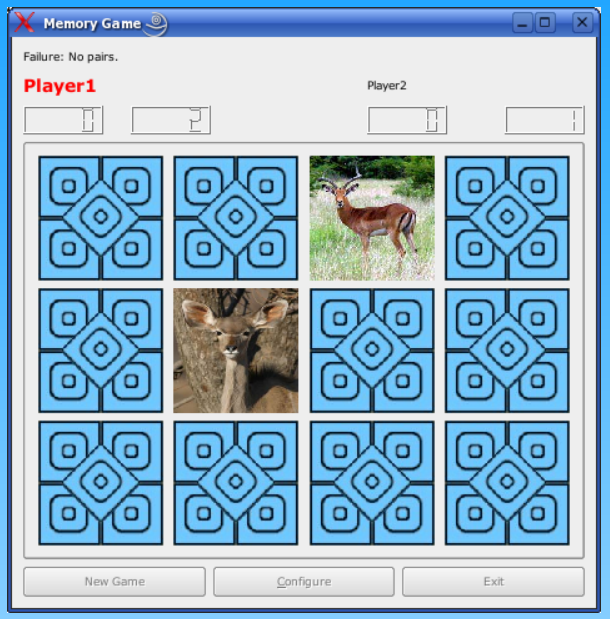
Elemi alkalmazások fejlesztése III.

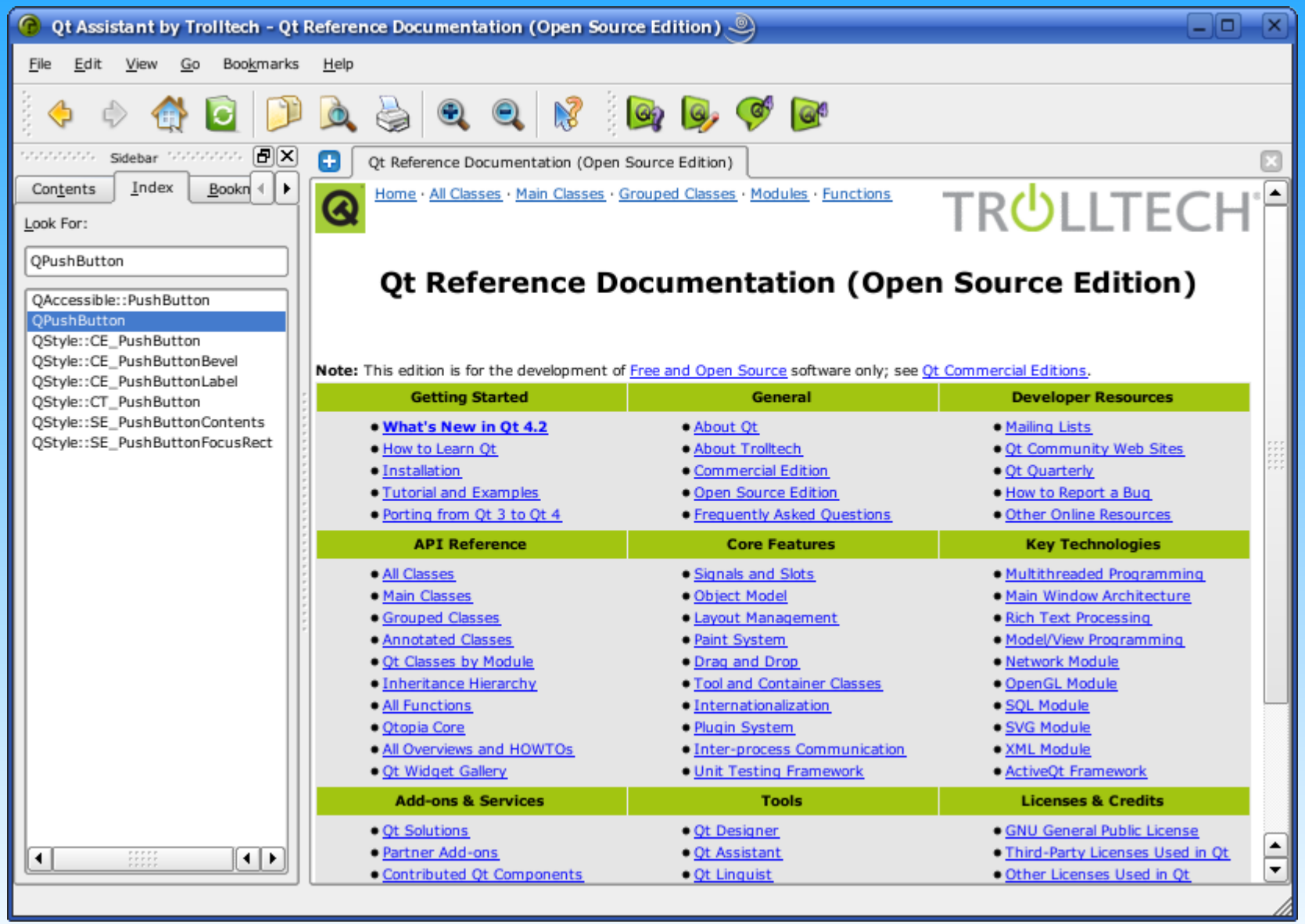


Bevezetés

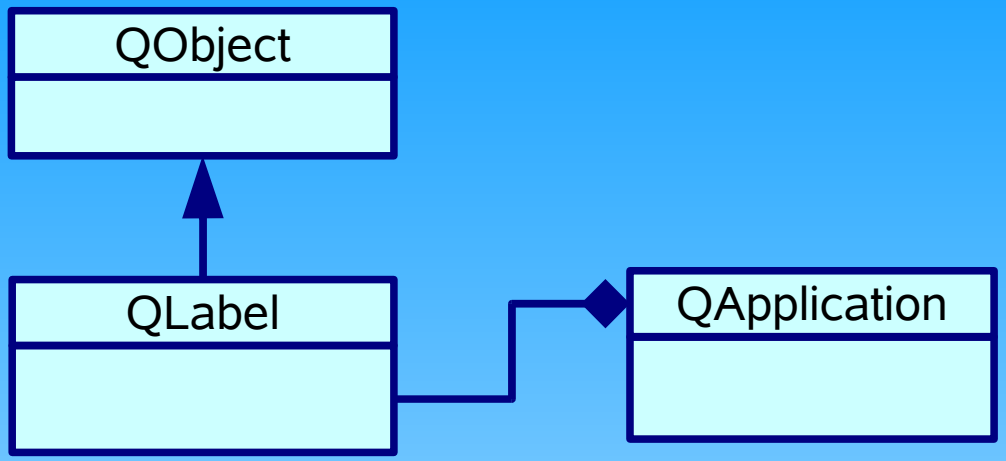
Készítette: Szabóné Nacsa Rozália
nacsa@inf.elte.hu
people.inf.elte.hu/nacsa/qt4/eaf3/
Qt 4
2007

A Qt assistant nyitó ablaka

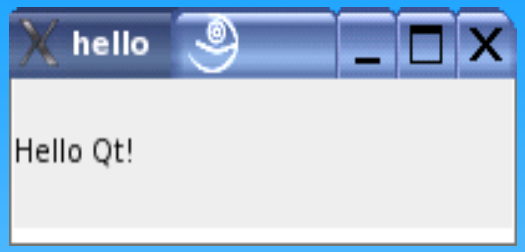
www.trolltech.com



Alkalmazás Qt osztályokkal: “Hello Qt”



main.cpp

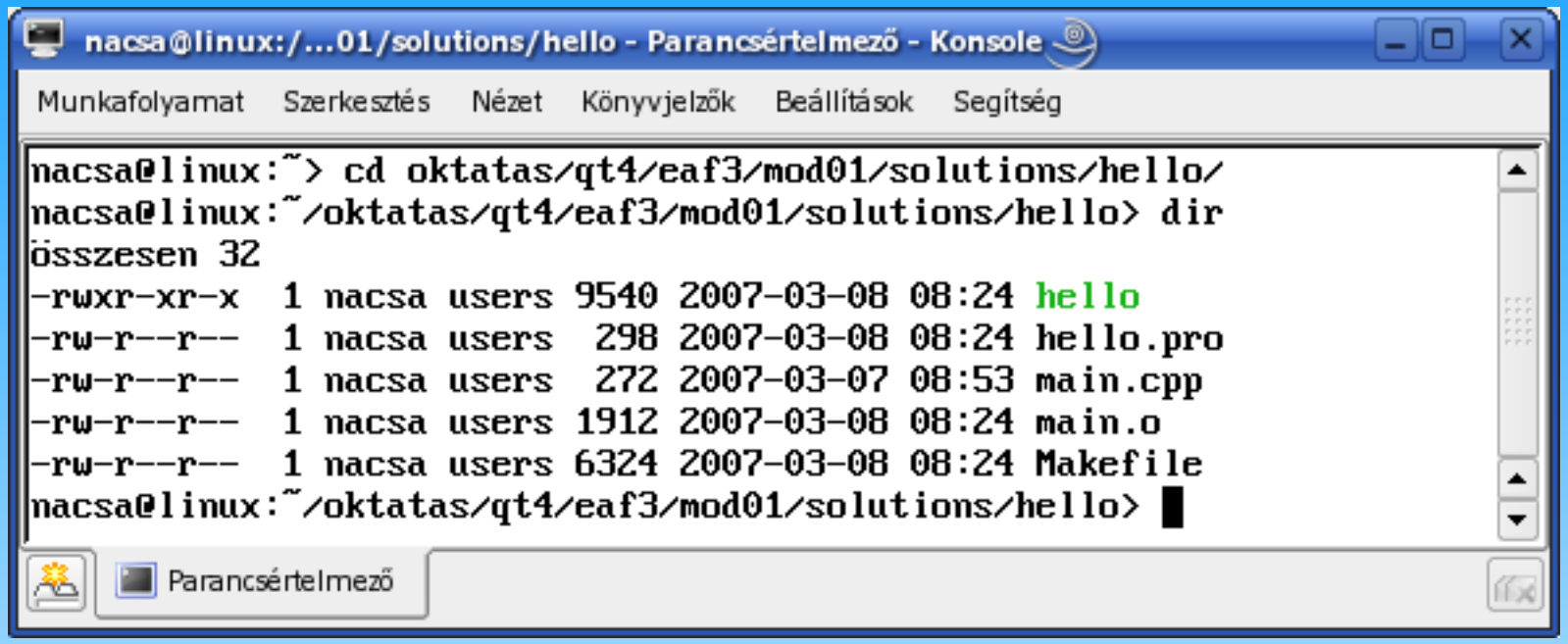



```
#include <QApplication>
#include <QLabel>

int main(int argc,char **argv)
{
        QApplication app(argc,argv);

        //QLabel *label = new QLabel("Hello Qt!");
        QLabel *label = new QLabel("<h1><i>Hello </i><font color=red>Qt!</font></h1>");
        label->show();

        return app.exec();
}
```

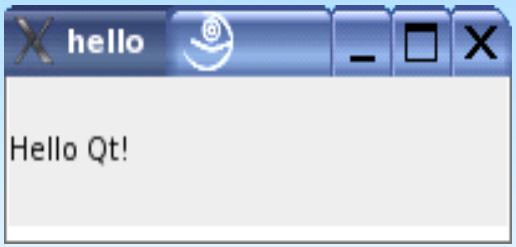
Használhatunk html formázást.

people.inf.elte.nacsa/qt4/eaf3/mod01/projects/hello

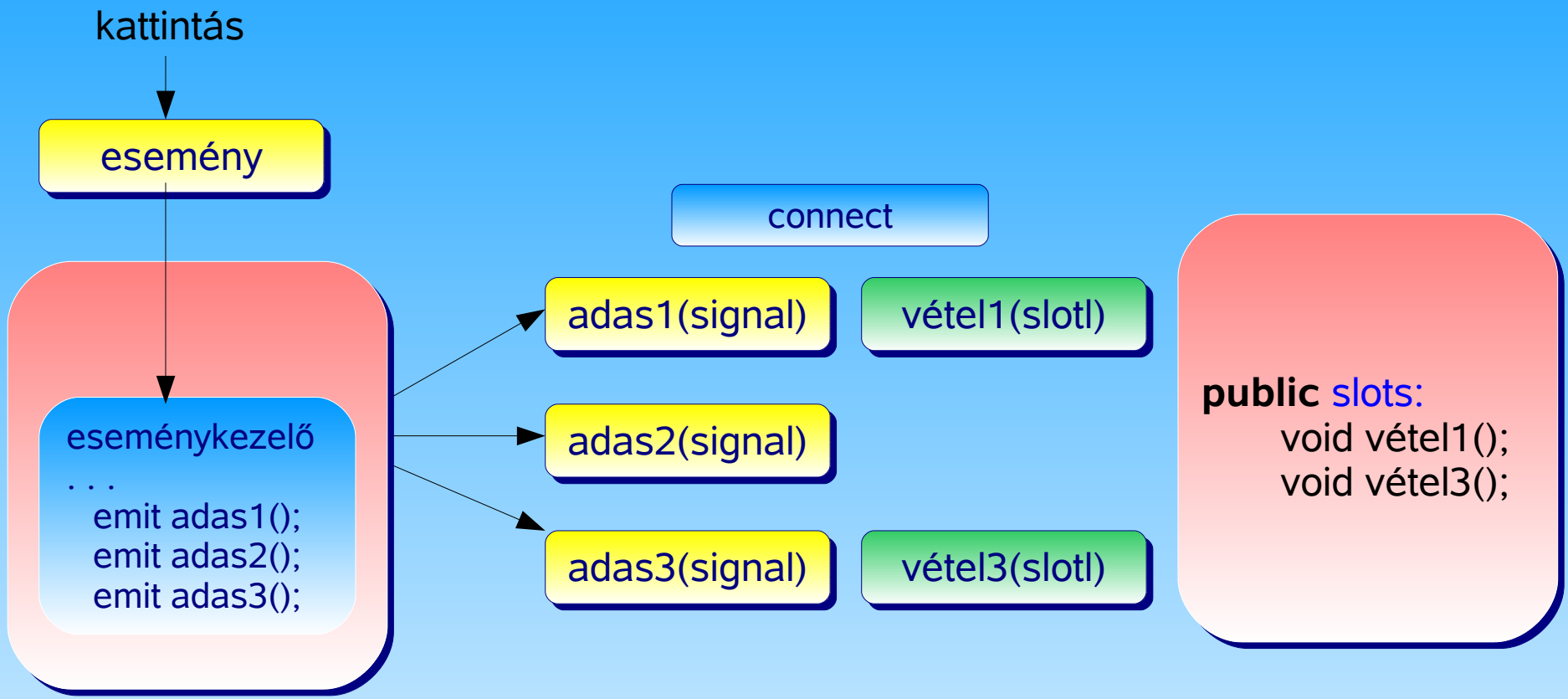
Fordítás/futtatás parancs sorból - Linux




```
qmake -project
qmake hello.pro
make
./hello
```

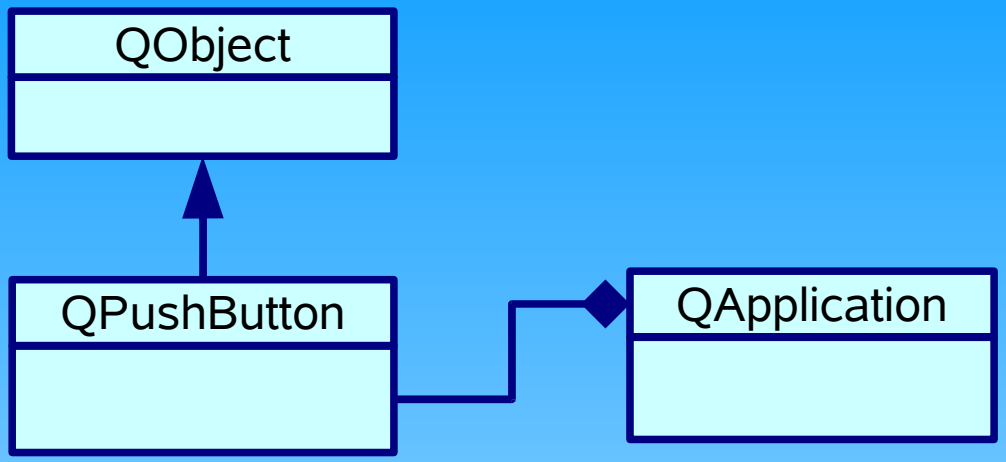


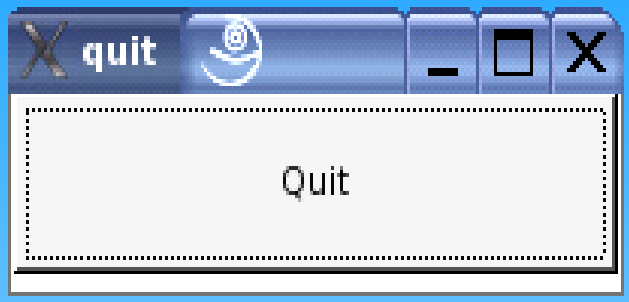
“Eseménykezelés” signal-slot mechanizmussal



Adás – vétel (pl. rádió)

Alkalmazás Qt osztályokkal: Jelek és jelvevők összekapcsolása





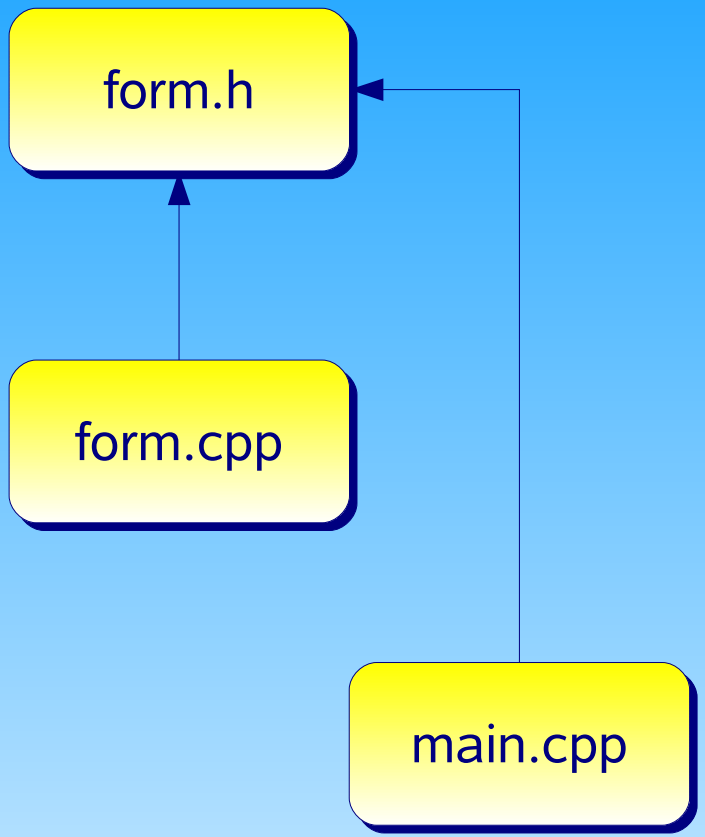
main.cpp

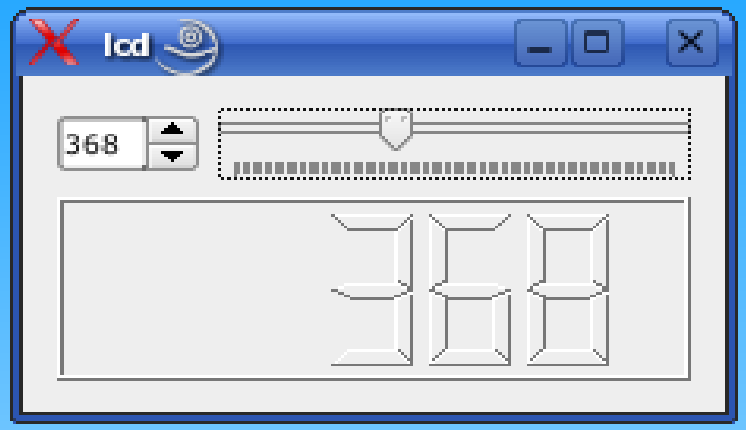
```
#include <QApplication>
#include <QPushButton>

int main(int argc, char *argv[])
{
        QApplication app(argc,argv);
        QPushButton *button = new QPushButton("Quit");

        QObject::connect(button,SIGNAL(clicked()),&app,SLOT(quit()));

        button->show();
        return app.exec();
}
```

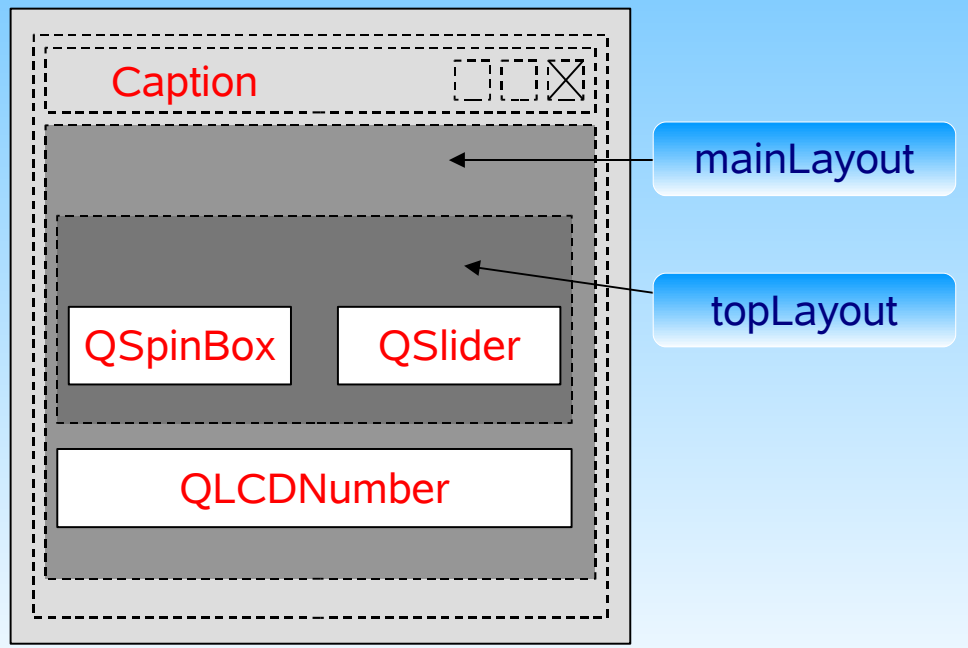
A gomb clicked() jelzését összekapcsoljuk az alkalmazás quit() jelvevő függvényével (slot).

people.inf.elte.nacsa/qt4/eaf3/mod01/projects/quit

Több vezérlő összekapcsolása, együttműködésük összehangolása





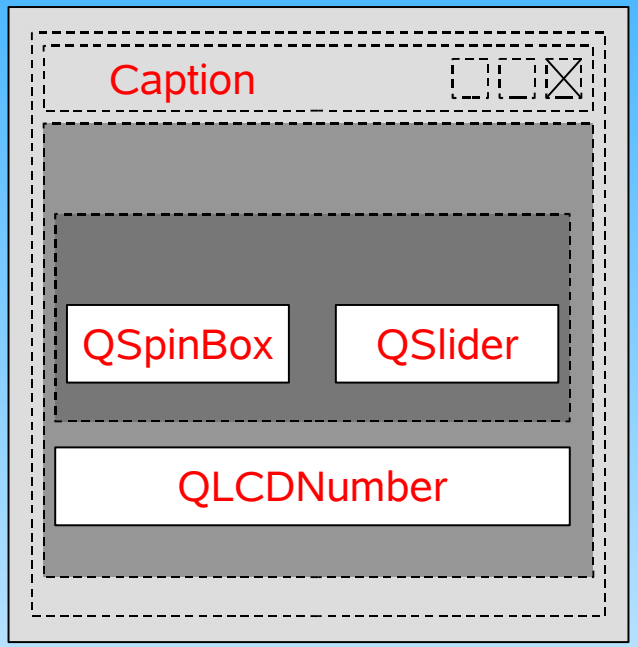


A grafikus-felület osztály definiálása (Form osztály)

form.h

```
class QSpinBox;
class QSlider;
class QLCDNumber;
class QHBoxLayout;
class QVBoxLayout;

class Form: public QWidget
{
    Q_OBJECT

public:
    Form(QWidget *parent=0);

private:
    QSpinBox *spinBox;
    QSlider *slider;
    QLCDNumber *lcd;

    QHBoxLayout *topLayout;
    QVBoxLayout *mainLayout;

};
```

```
#include <QWidget>
```

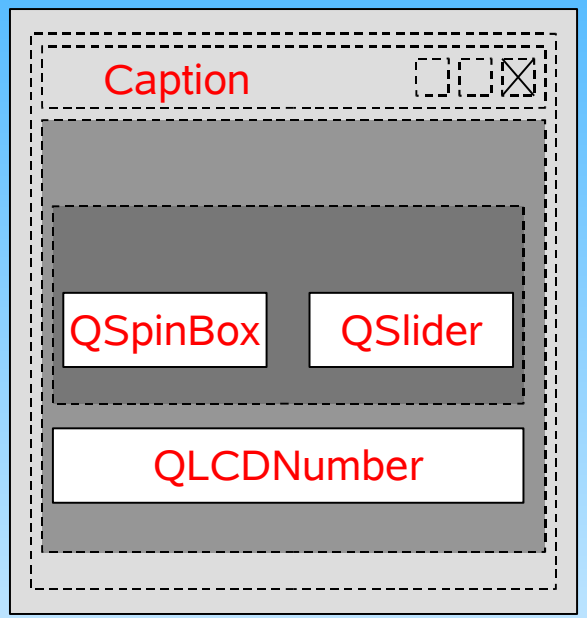


Form osztály: konstruktor

form.cpp

```
Form::Form(QWidget *parent) : QWidget(parent)
{
        spinBox = new QSpinBox;
        setObjectName(QString::fromUtf8("spinBox"));
        spinBox->setRange(0,100);

        slider = new QSlider(Qt::Horizontal);
        setObjectName(QString::fromUtf8("slider"));
        slider->setRange(0,100);
        slider->setSingleStep(10);
        slider->setOrientation(Qt::Horizontal);
        slider->setTickPosition(QSlider::TicksBelow);

        lcd = new QLCDNumber;
        setObjectName(QString::fromUtf8("lcd"));
      . . .
 }
```

```
#include <QLayout>
#include <QSpinBox>
#include <QSlider>
#include <QLCDNumber>

#include "form.h"
```



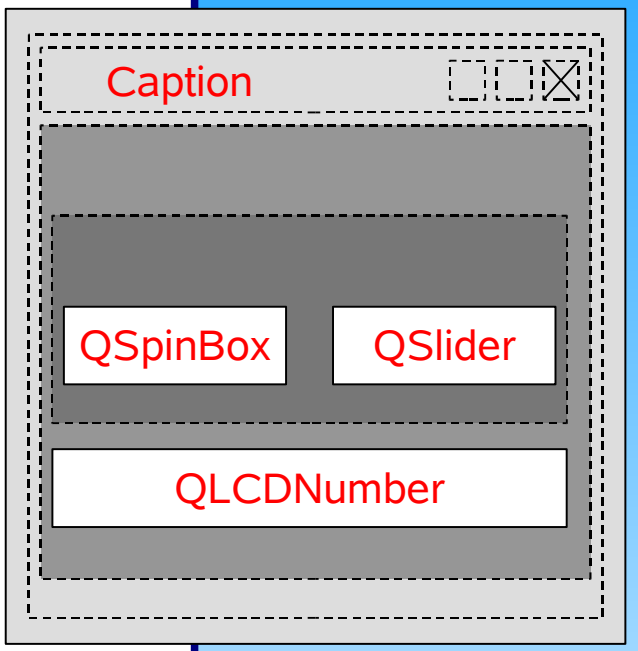
Vezérlők létrehozása és tulajdonságainak beállítása.

A Form osztály konstruktora - elrendezés

form.cpp

```
Form::Form(QWidget *parent) : QWidget(parent) {

    . . .

    topLayout = new QHBoxLayout;
    topLayout->addWidget(spinBox);
    topLayout->addWidget(slider);

    mainLayout = new QVBoxLayout;
    mainLayout->addLayout(topLayout);
    mainLayout->addWidget(lcd);
    mainLayout->setMargin(11);  //11 pixel margó az elrendező köé
    mainLayout->setSpacing(6);   //6 pixel helyköz a vezérlők között

    setLayout(mainLayout);


    connect(spinBox, SIGNAL(valueChanged(int)), lcd, SLOT(display(int)));
    connect(spinBox, SIGNAL(valueChanged(int)), slider, SLOT(setValue(int)));
    connect(slider, SIGNAL(valueChanged(int)), lcd, SLOT(display(int)));
    connect(slider, SIGNAL(valueChanged(int)), spinBox, SLOT(setValue(int)));

}
```

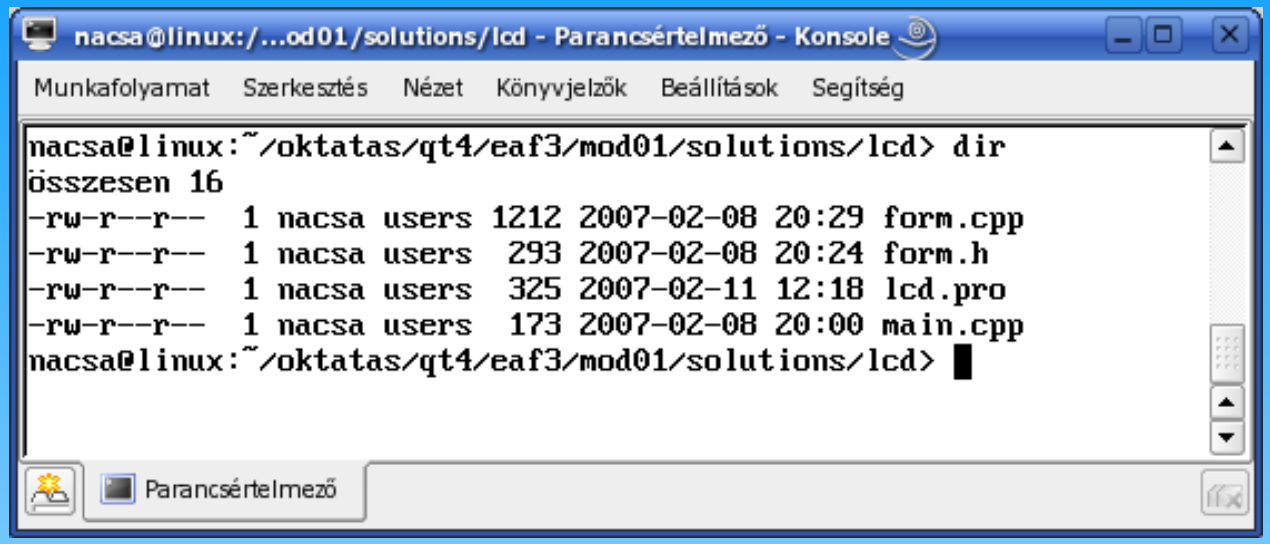
Elemek elrendezése.

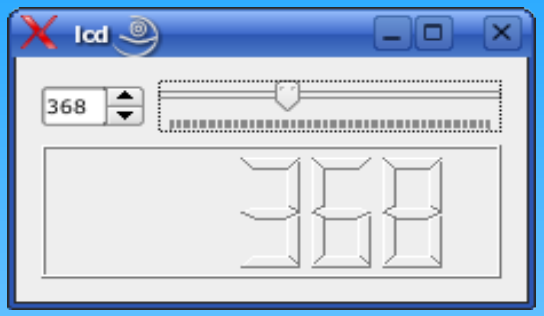


Jeladók és jelfeldolgozók összekapcsolása.

lcd projekt

Fordítás, futtatás





main.cpp

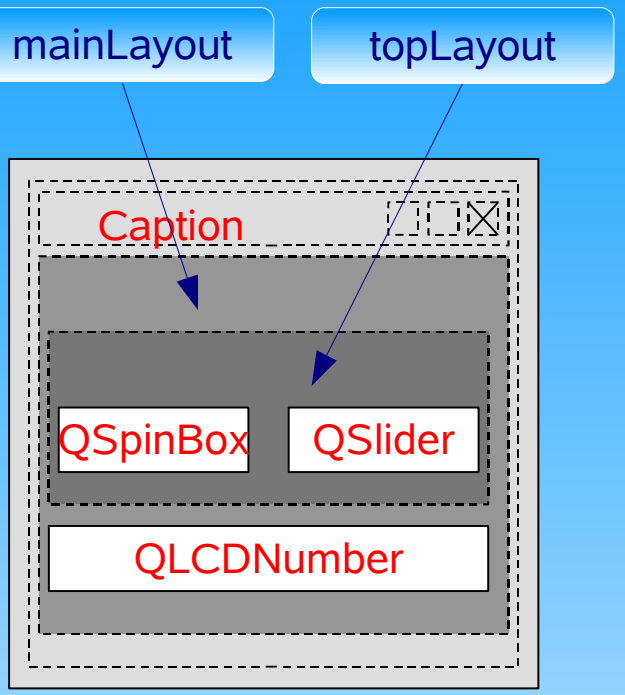
```
#include <QApplication>
#include "form.h"

int main(int argc, char *argv[])
{
        QApplication app(argc,argv);

        Form *form = new Form;
        form->show();

        return app.exec();
}
```

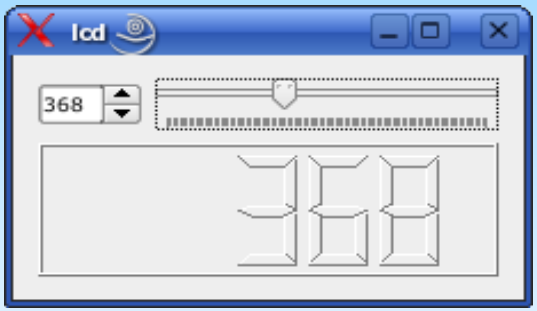
lcd.pro

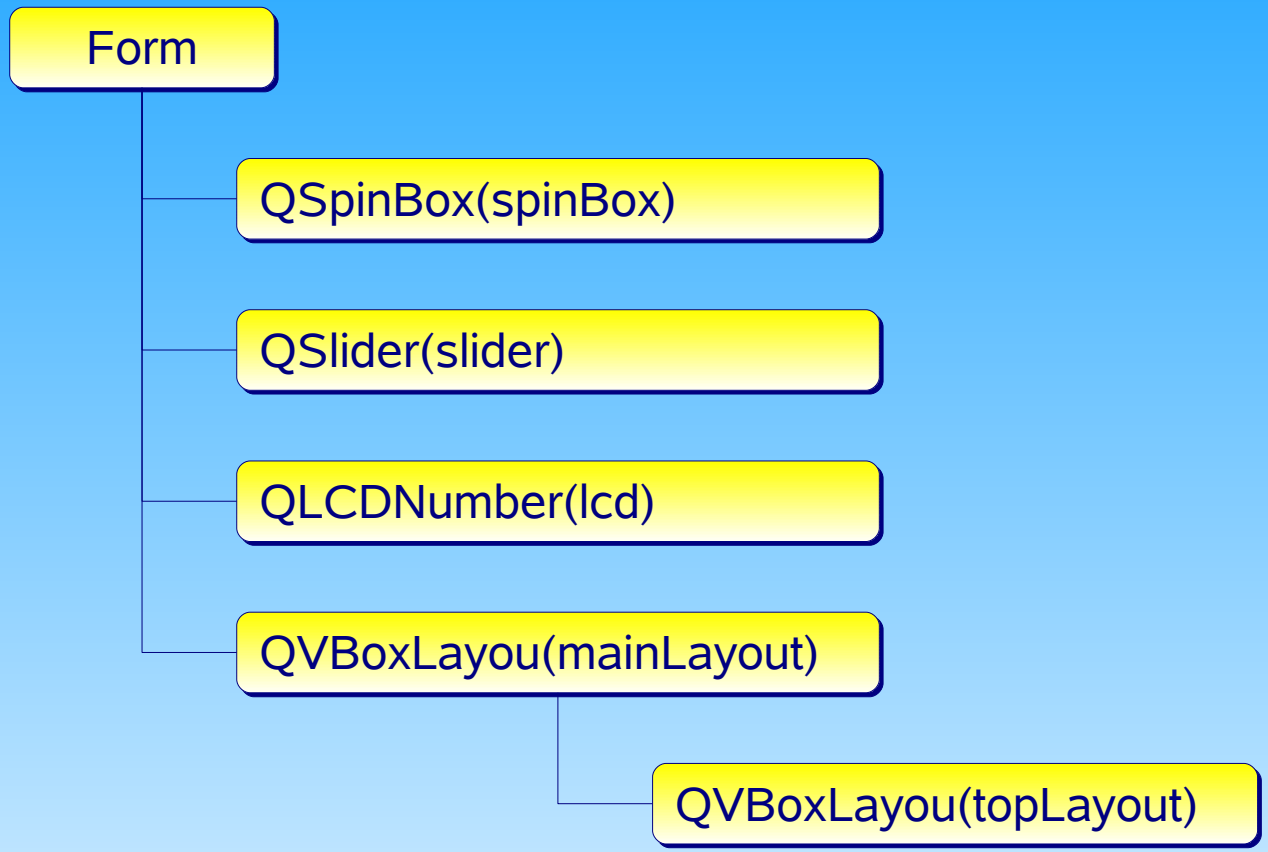
```
TEMPLATE = app
TARGET =
DEPENDPATH += .
INCLUDEPATH += .

# Input
HEADERS += form.h
SOURCES += form.cpp main.cpp
```

people.inf.elte.nacsa/qt4/eaf3/mod01/projects/lcd

Memóriagazdálkodás

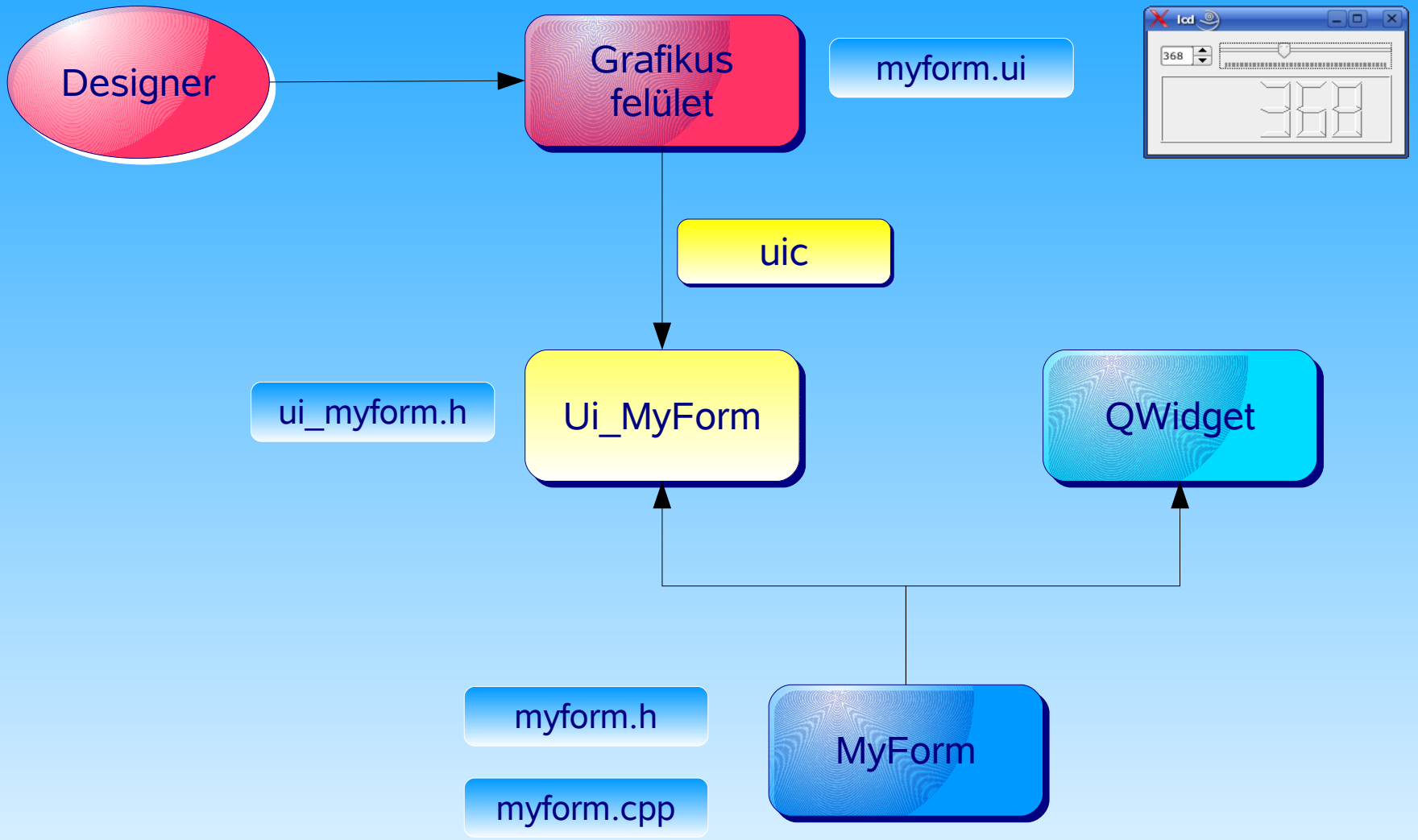




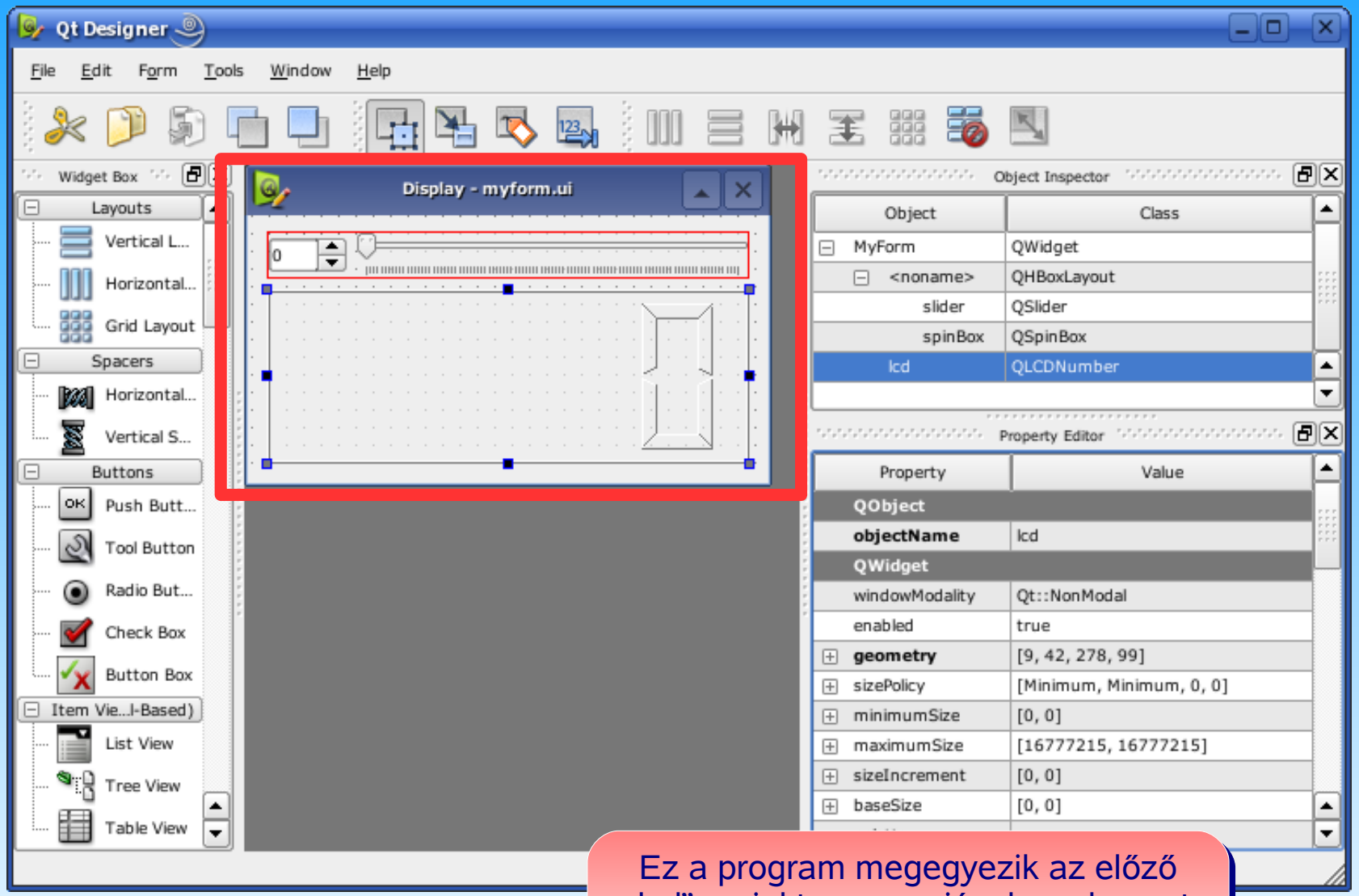


Csak azokat a widget-eket kell törölni, amelyeket **new** operátorral hozunk létre és „**nincs szülője**”.

A Qt Designer használata



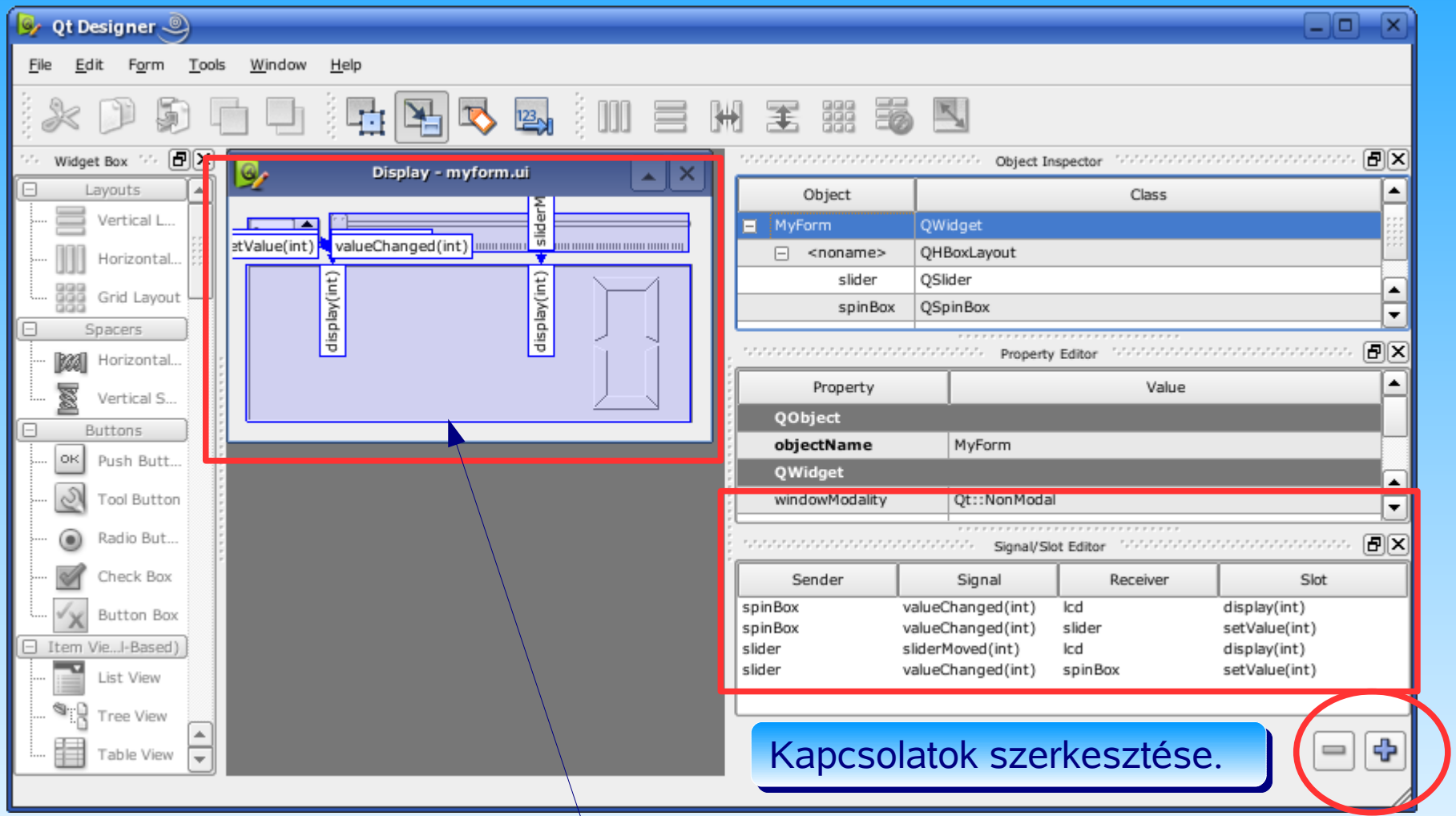
Qt Designer: felület tervezése



Ez a program megegyezik az előző „lcd” projekt programjával, csak most Qt Designer-t is használunk az elkészítéséhez.

Qt Designer: Jelek és jeladók összekapcsolása

Edit/Edit Signals and Slots (F4)



Kapcsolatok szerkesztése.

Ui_Myform osztály

A származtatott MyForm osztály megvalósítása

myform.h

```
#include <QWidget>

#include "ui_myform.h"

class MyForm : public QWidget, private Ui_MyForm
{
    Q_OBJECT
public:
    MyForm(QWidget *parent = 0);

};
```

main.cpp

```
#include <QApplication>
#include "myform.h"
int main (int argc, char *argv[])
{
    QApplication app(argc,argv);

    MyForm *form = new MyForm;
    form->show();

    return app.exec();
}
```

myform.cpp

```
#include "myform.h"

MyForm::MyForm(QWidget *parent)
    : QWidget(parent)
{
    setupUi(this);

}
```

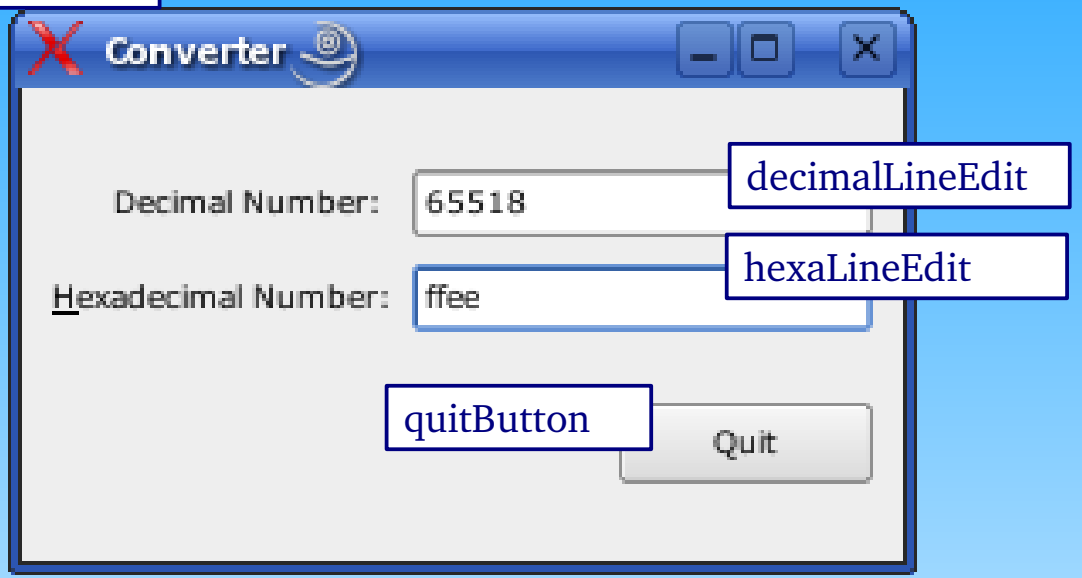
people.inf.elte.nacsa/qt4/eaf3/mod01/projects/display

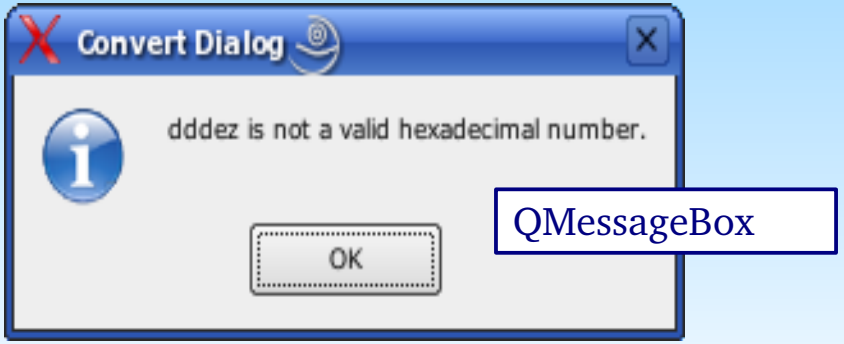
Saját eseménykezelő készítése

conver projekt

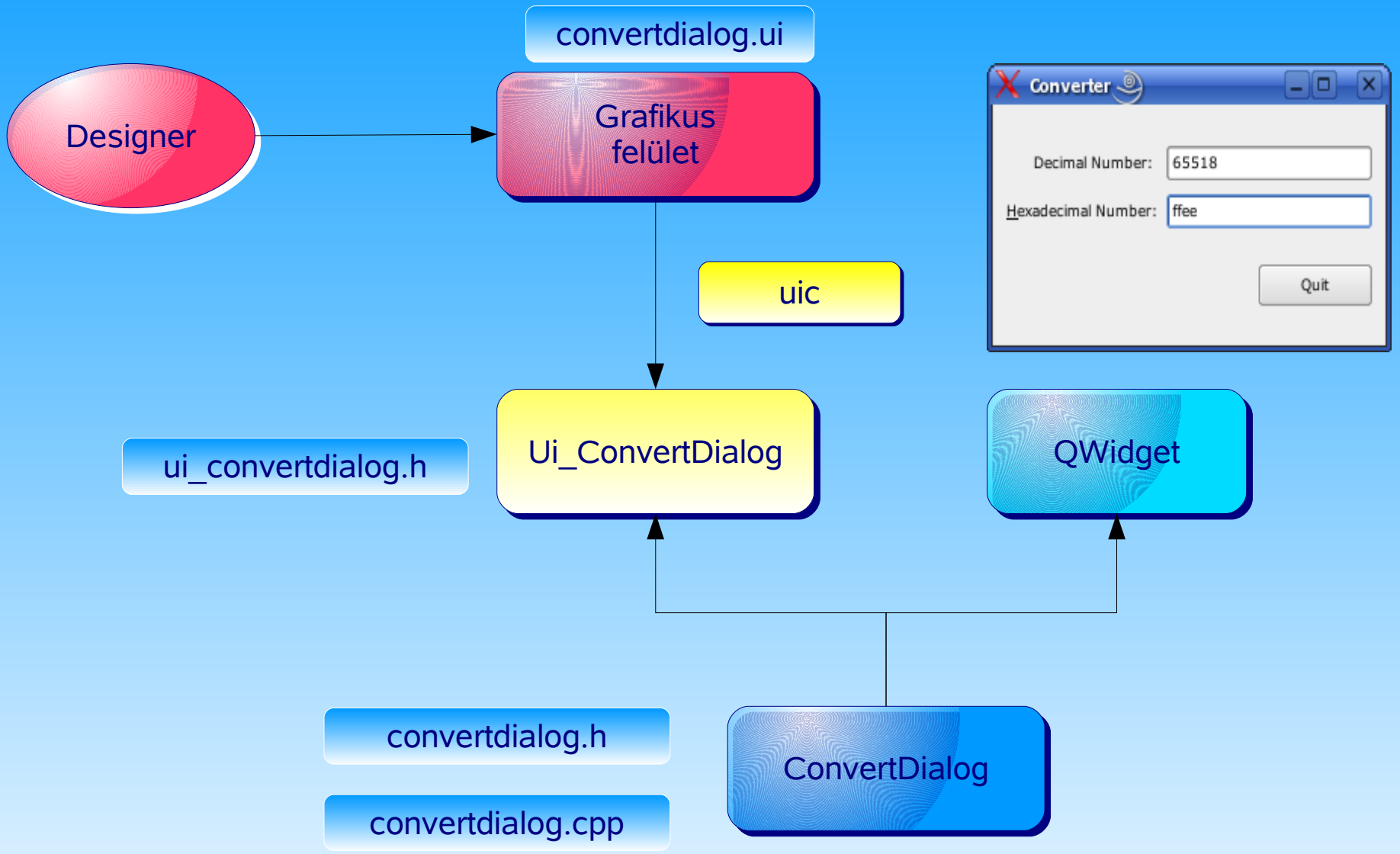
Konvertáló program

MainForm





A Qt Designer használata



Saját grafikus osztály megvalósítása származtatással

convert projekt

convertdialog.h

```
#ifndef _CONVERT_DIALOG_H_
#define _CONVERT_DIALOG_H_

#include <QWidget>

#include "ui_convertdialog.h"

class ConvertDialog : public QWidget, private Ui_ConvertDialog
{
    Q_OBJECT
public:
    ConvertDialog(QWidget *parent = 0);

public slots:

    void on_quitButton_clicked();
    void updateHexaLineEdit(const QString & str);
    void updateDecimalLineEdit(const QString & str);
private:
    QString convert(const QString& str, const int fromBase, const int toBase);
};

#endif // _CONVERT_DIALOG_H_
```

A Qt Designerrel készített felület implementációja.

Ha saját **signal**-t, vagy **slot**-ot használunk, akkor mindig meg kell adni a Q_OBJECT makrót.

Metanyelvi elem.
A moc (meta object compiler) készít belőle szabványos kódot.

ConvertDialog: konstruktor

convertdialog.cpp

```
#include <QMessageBox>

#include "convertdialog.h"

ConvertDialog::ConvertDialog(QWidget *parent)
        : QWidget(parent)
{
        setupUi(this);

        connect(decimalLineEdit, SIGNAL(textEdited(const QString &)),
                        this, SLOT(updateHexaLineEdit(const QString &)));
        connect(hexaLineEdit, SIGNAL(textEdited(const QString &)),
                        this, SLOT(updateDecimalLineEdit(const QString &)));
}

void ConvertDialog::on_quitButton_clicked() // a "névkonvenció" miatt nem kell connect
{
        qApp->quit();   //qApp az (egyetlen!) alkalmazásunkra mutató pointer
}
```

AsetupUi() inicializálja a tervezővel készített elemeket.

convert projekt

ConvertDialog: Eseménykezelők

convert projekt

convertdialog.cpp

```
void ConvertDialog::updateHexaLineEdit(const QString & str)
{
      hexaLineEdit->setText(convert(str,10,16).toUpper());
}

void ConvertDialog::updateDecimalLineEdit(const QString & str)
{
      decimalLineEdit->setText(convert(str,16,10));
}

QString ConvertDialog::convert(const QString& str, const int fromBase, const int toBase)
{
      QString ret = "";
      bool ok;
      int value = str.toInt(&ok, fromBase);
      if(ok)
          ret = QString::number(value,toBase);
      else
            QMessageBox::information(this, "Convert Dialog",
                                              QString::number(value) + " is not a valid number.");
      return ret;
}
```

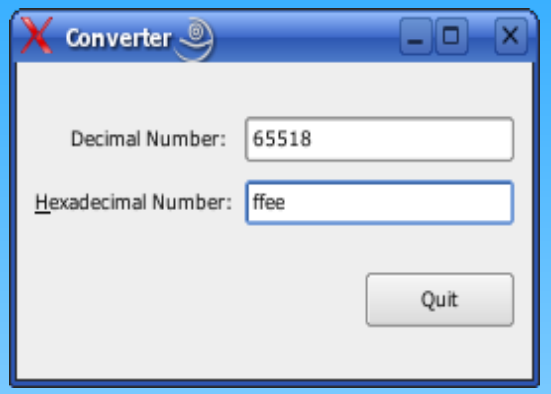
Az str stringet számként értelmezve a fromBase számrendszerből toBase számrendszerbeli számra konvertálja át, és visszaadja az így előállított számot szöveges formában.

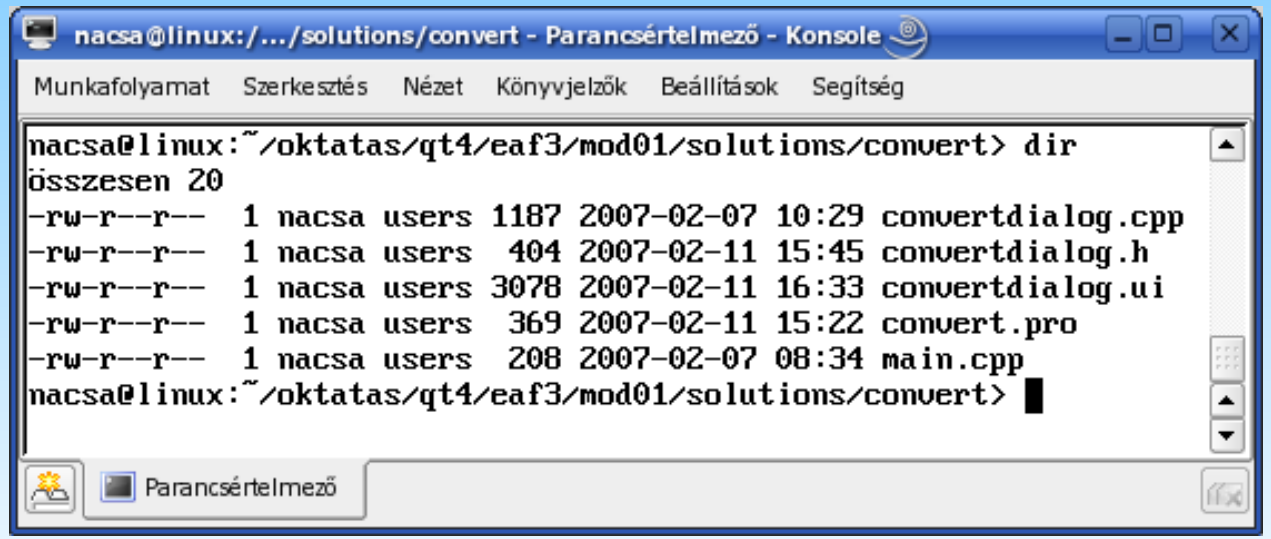
Konvertáló program - Fordítás/futtatás

convert projekt

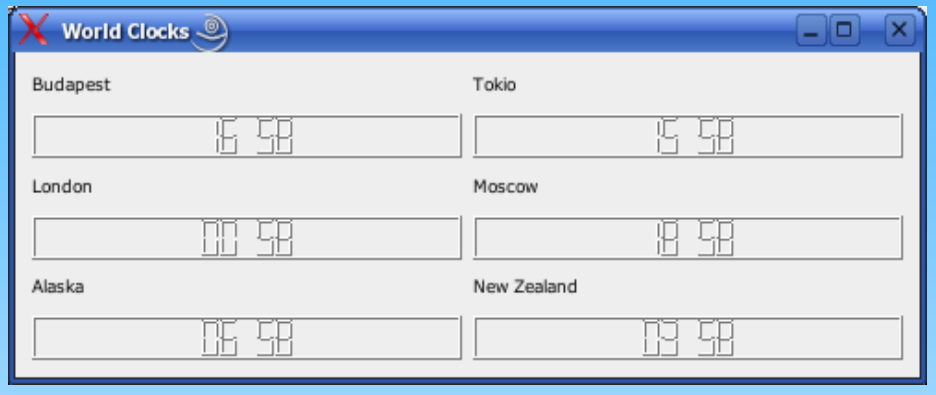
main.cpp

```
#include <QApplication>
#include "convertdialog.h"
int main (int argc, char *argv[])
{
        QApplication app(argc,argv);

        ConvertDialog *dialog = new ConvertDialog;
        dialog->show();

        return app.exec();
}
```





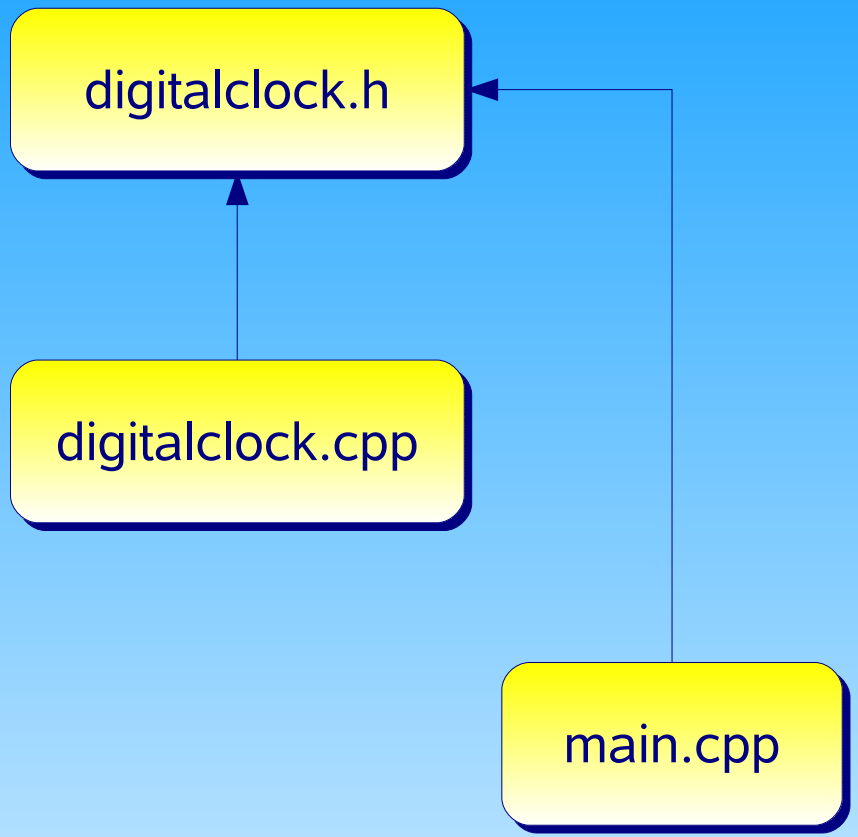
Saját vezérlő használata Qt Designerben

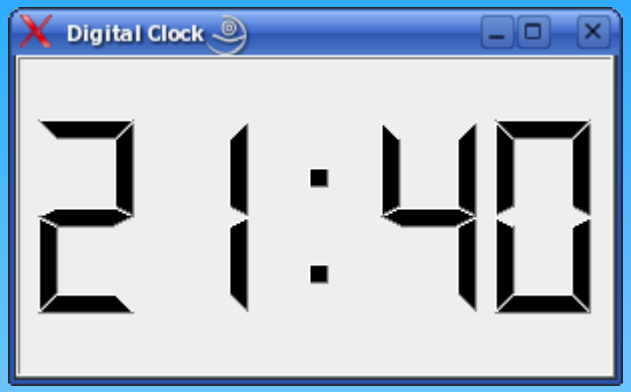


worldclock projekt

Ismétlő időzítő, egyszeri időzítő

Digitális óra készítése





Esemény kiváltása nem vizuális elemmel.

Digitális óra definíciója

worldclock projekt

digitalclock.h

```
#include <QLCDNumber>
class DigitalClock : public QLCDNumber{
    Q_OBJECT
public:
    DigitalClock(QWidget *parent = 0);
    void setTimeZone(int time) {mTimeZone = time;}

private slots:
    void showTime();

private:
        int mTimeZone;
};
```

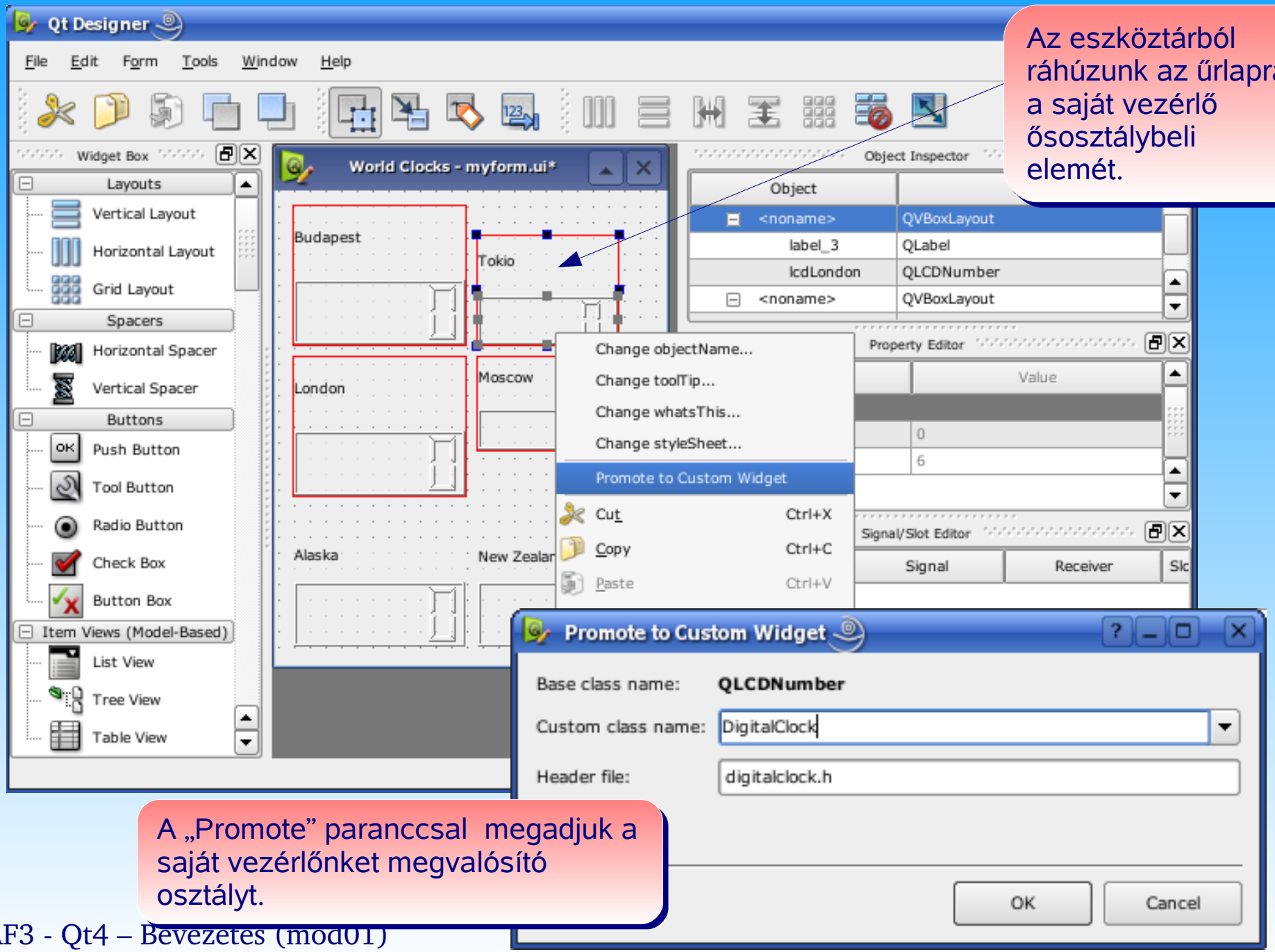
digitalclock.cpp

```
DigitalClock::DigitalClock(QWidget *parent)  : QLCDNumber(parent)
{
    setSegmentStyle(Filled);
    QTimer *timer = new QTimer(this);
    connect(timer, SIGNAL(timeout()), this, SLOT(showTime()));
    timer->start(1000);
    showTime();
    setWindowTitle(tr("Digital Clock"));
    resize(150, 60);
}
void DigitalClock::showTime()
{
    QTime time = QTime::currentTime().addSecs(mTimeZone *60 *60);
    QString text = time.toString("hh:mm");
    if ((time.second() % 2) == 0)
        text[2] = ' ';
    display(text);
}
```

```
#include <QTimer>
#include <QDateTime>
#include "digitalclock.h"
```

Beállítjuk az űrlap címkesorát és a kezdeti méretét.

Qt Designer saját vezérlő használata közben

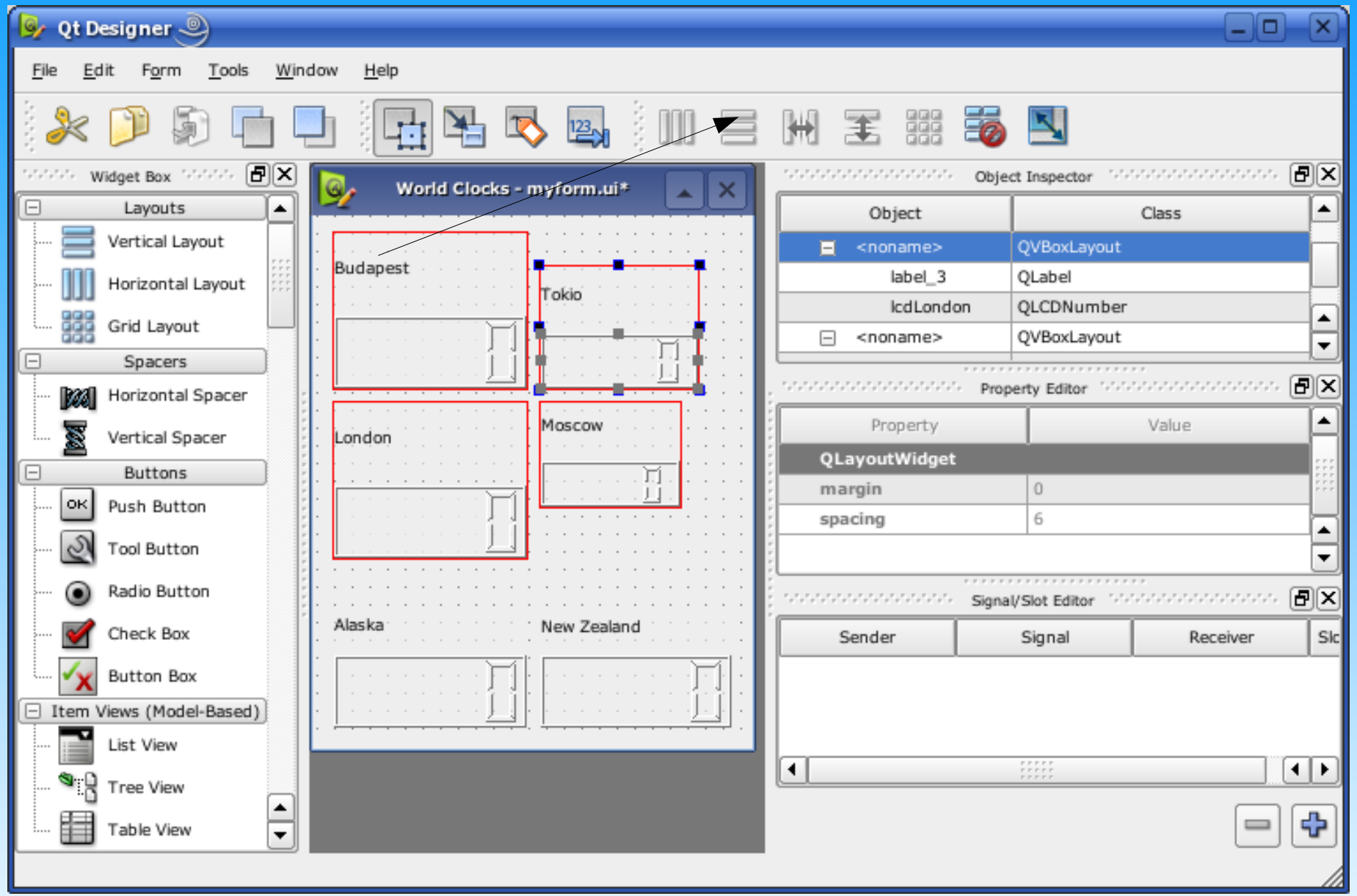


Az eszköztárból ráhúzunk az űrlapra a saját vezérlő ősosztálybeli elemét.

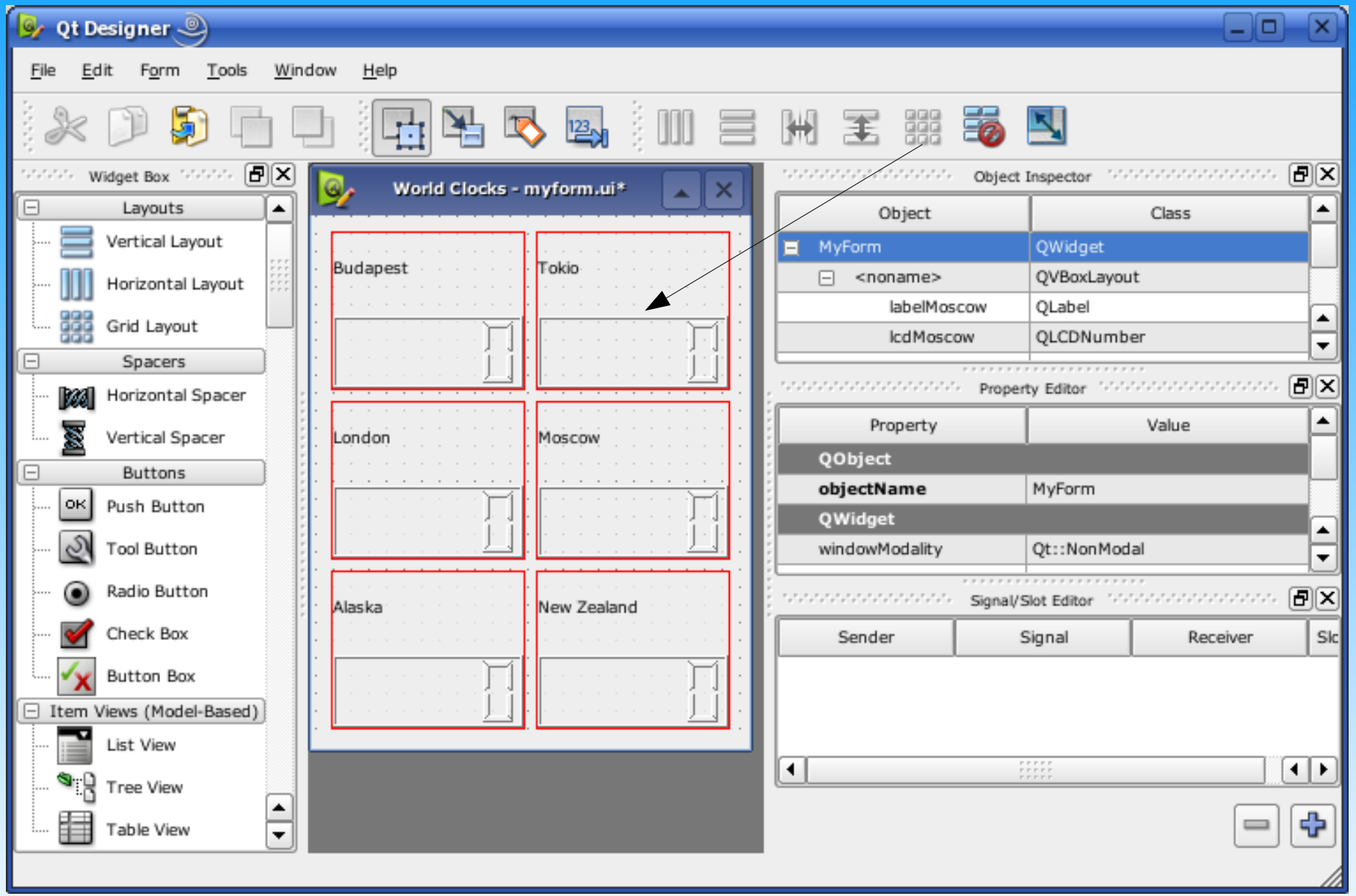
A „Promote” paranccsal megadjuk a saját vezérlőnket megvalósító osztályt.

worldclock projekt

Vezérlők elrendezése: Lay out Vertically



Vezérlők elrendezése: Lay out in Grid



worldclock projekt

MyForm osztály

worldclock projekt

myform.h

```
#include <QWidget>
#include "ui_myform.h"
class MyForm : public QWidget, private Ui_MyForm
{
    Q_OBJECT
 public:
    MyForm(QWidget *parent = 0);
};
```

myform.cpp

```
#include "myform.h"
MyForm::MyForm(QWidget *parent)
    : QWidget(parent)
{
    setupUi(this);

    lcdLondon->setTimeZone(-1);
    lcdTokio->setTimeZone(8);
    lcdMoscow->setTimeZone(2);
    lcdAlaska->setTimeZone(-10);
    lcdNewZealand->setTimeZone(11);
}
```

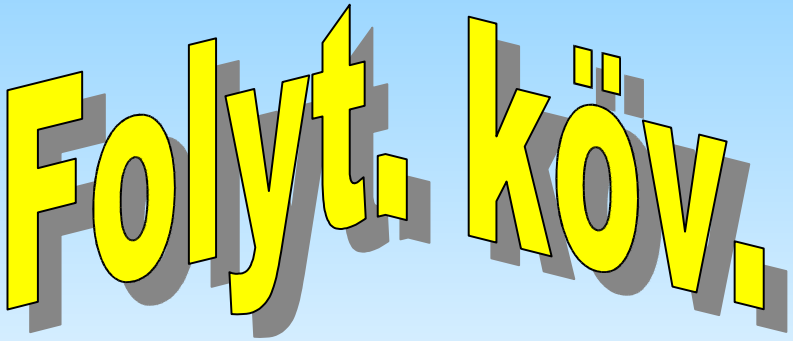
main.cpp

```
#include <QApplication>
#include "myform.h"

int main (int argc, char *argv[])
{
    QApplication app(argc,argv);

    MyForm *form = new MyForm;
    form->show();
    return app.exec();
 }
```

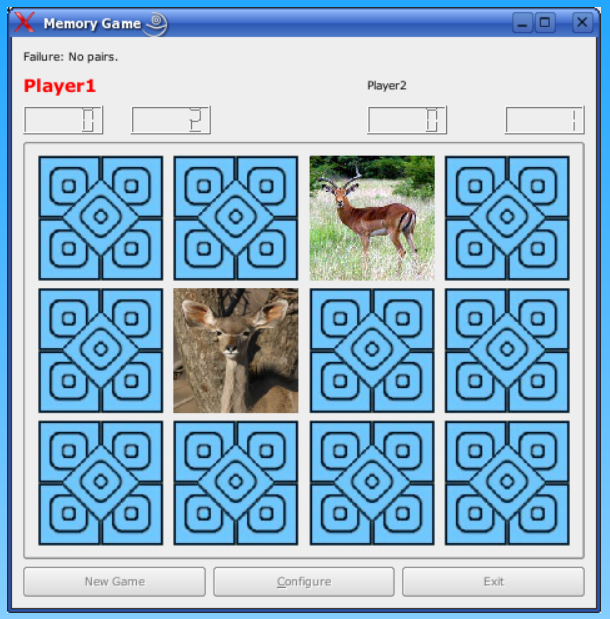
A bemutatott programok megtalálhatók a

people.inf.elte.nacsa/qt4/eaf3/mod01/projects/

címen.



Elemi alkalmazások fejlesztése III.



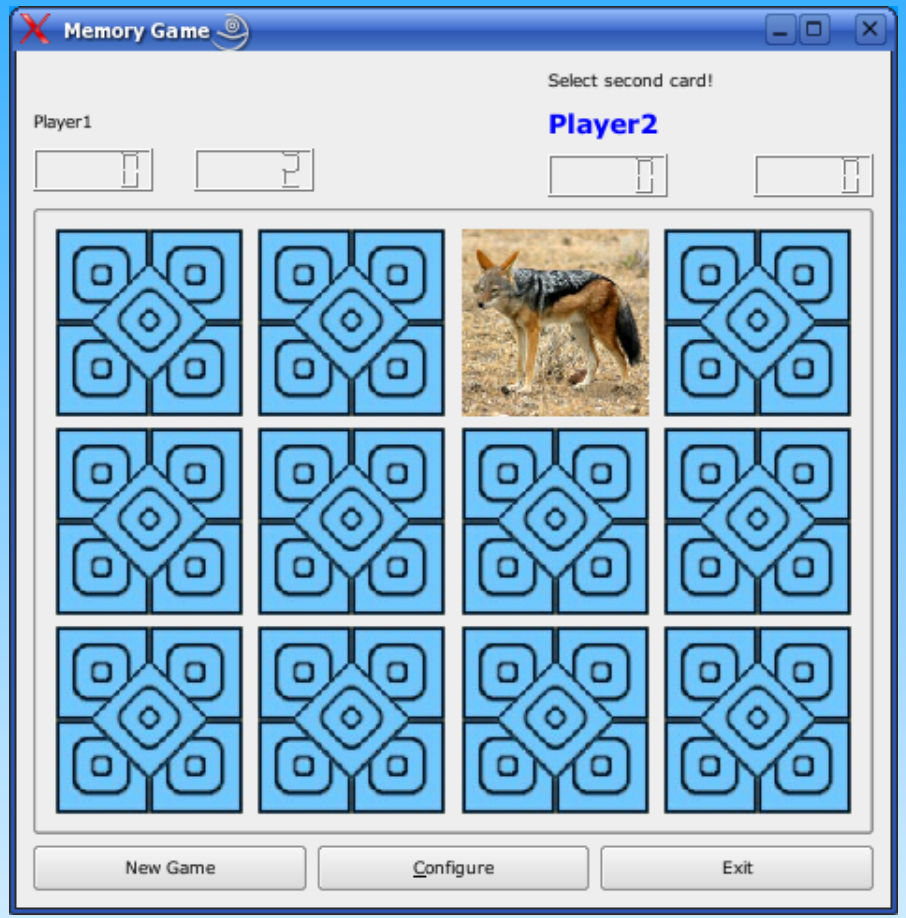
Memóriajáték 1.

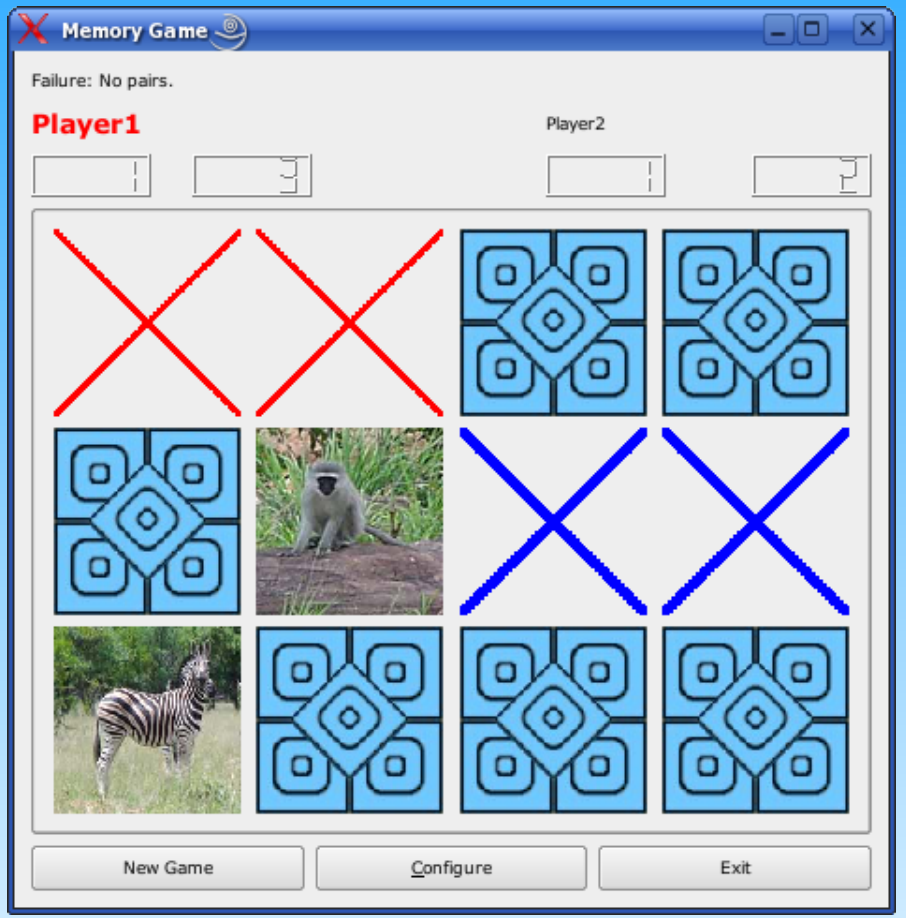
Készítette: Szabóné Nacsa Rozália
nacsa@inf.elte.hu
people.inf.elte.hu/nacsa/qt4/eaf3/

Qt 4

2007

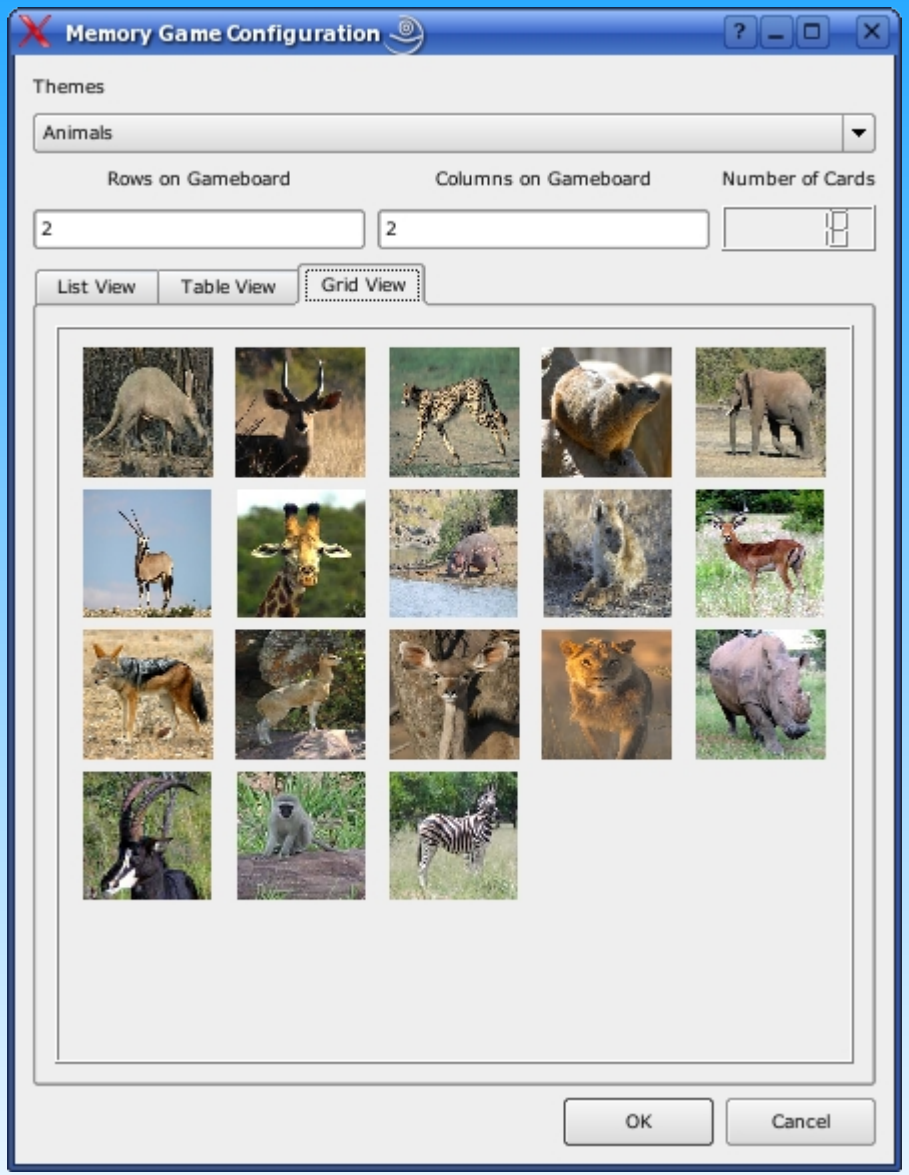
Játék közben

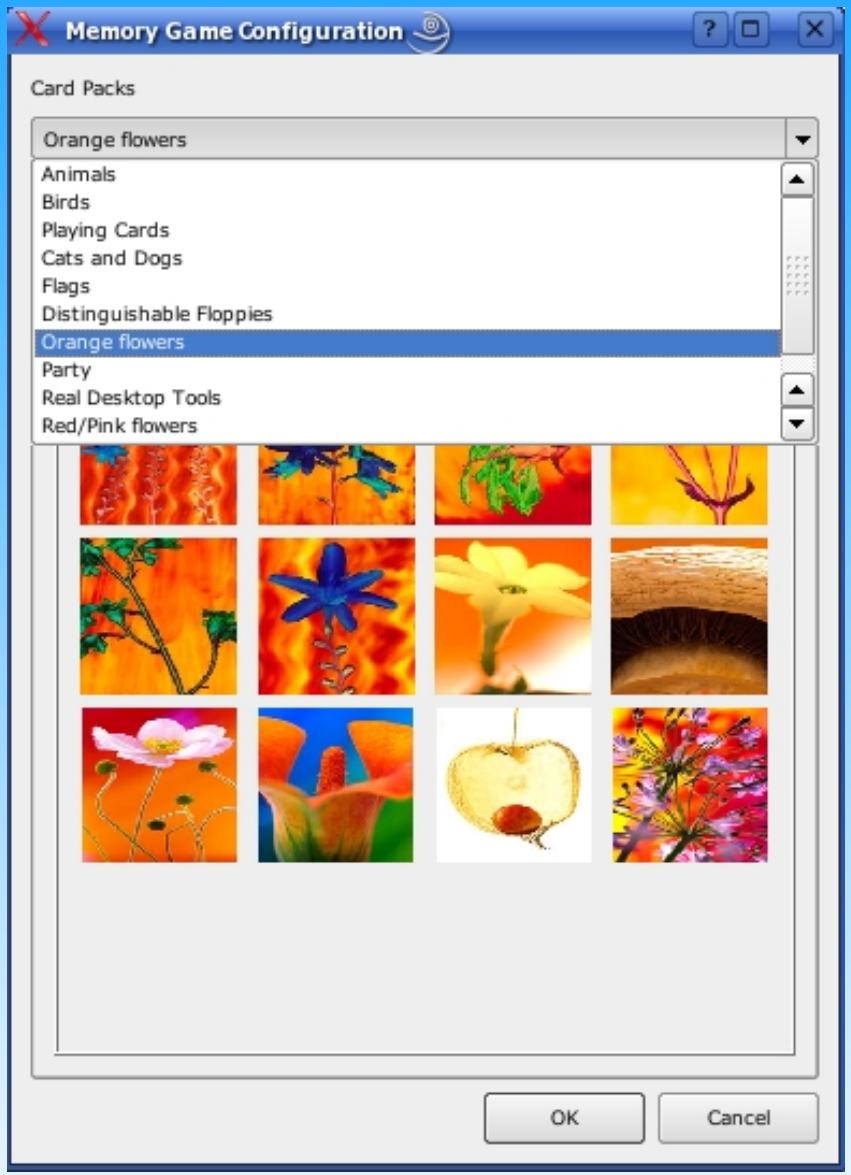




memorygame

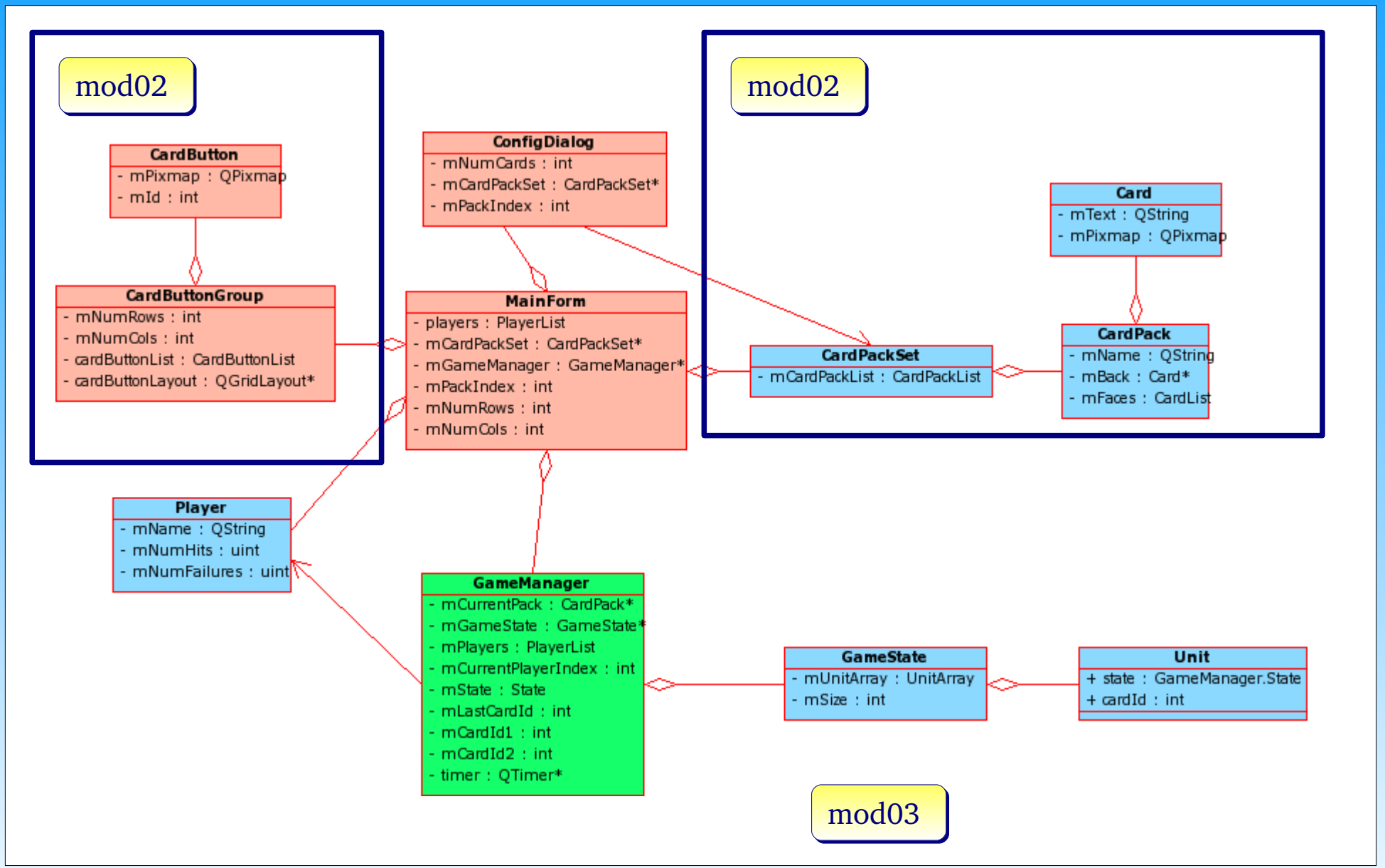
Kártyák kiválasztása





memorygame

Az alkalmazás osztálydiagramja



A modul tartalma

Látványelemek

- Saját kártya-vezérlő készítése (CardButton osztály)
- Saját „kártya csoport vezérlő” készítése (CardGroup osztály)

Adatok

- Kártya csomag kezelő osztály elkészítése (CardPack osztály)
- Kártya csomagokat (többet!) kezelő osztály elkészítése (CardPackSet osztály)

Alkalmazás

- Kártya csomagok megjelenítése (cardviewer, cardtree projektek)

Utils modul: segéd függvények

utils.h, utils.cpp

utils.h

```
#include <QString>

namespace Utils
{
	void initRandomGenerator(); //Véletlenszám generátor inicializálása
	int getNextRandomNumber(const int max); //[0..max] közé eső véletlenszám
	QString getApplicationDirPrefix(); //Az éppen futó alkalmazás elérési útvonala
};
```

utils.cpp

```
#include <QDir>
#include <QApplication>
#include <QTime>

#include "utils.h"
```

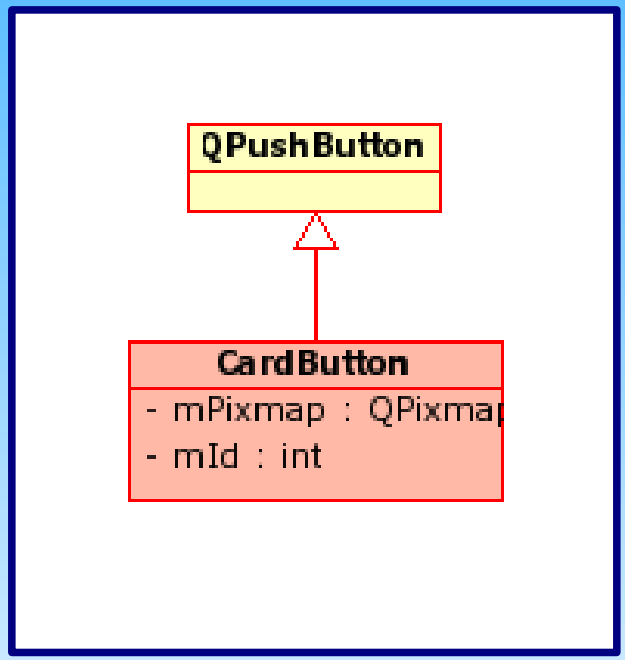
```
void Utils::initRandomGenerator()
{
	QTime t = QTime::currentTime();
	srand(t.hour()*12 + t.minute()*60 + t.s
}
int Utils::getNextRandomNumber(const int max)
{
	return (int) (((float) max)*rand()/(RAND_MAX + 0.1));
}
QString Utils::getApplicationDirPrefix()
{
	return qApp->applicationDirPath() + QDir::separator();
}
```

CardButton: definíció

cardbutton.h

```
class CardButton : public QPushButton
{
        Q_OBJECT

public:
    CardButton( QWidget * parent = 0);
    ~CardButton();
    void setPixmap (QPixmap pixmap) {mPixmap=pixmap;}
    int getId() {return mId;}
    void setId(int id) {mId = id;}
protected:
        void paintEvent(QPaintEvent * event);
public slots:
        void slotClicked();
signals:
        void cardButtonClicked(CardButton*);
        void cardButtonClicked(int);
 private:
     QPixmap mPixmap;
     int mId;
};
```

```
#include <QPushButton>
#include <QPixmap>
```



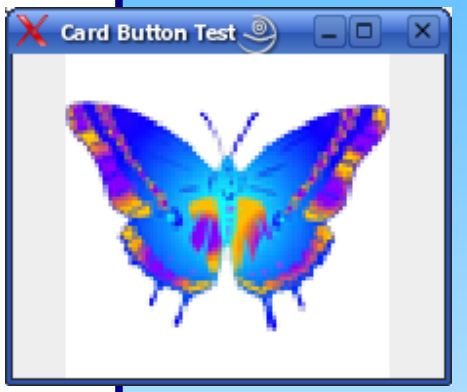
cardbutton - cardbutton.h

CardButton: implementáció

cardbutton.cpp

```
CardButton::CardButton( QWidget * parent)
        : QPushButton( parent )
{
        connect(this,SIGNAL(clicked()),this,SLOT(slotClicked()));
}

void CardButton::slotClicked()
{
        emit cardButtonClicked(this);
        emit cardButtonClicked(mId);
}

void CardButton::paintEvent (QPaintEvent * /*event*/)
{
        int side = qMin(size().width(), size().height());
        QRect rect((size().width()-side) / 2, (size().height()-side) / 2, side, side);
        QPainter painter;
        painter.begin(this);
        painter.drawPixmap(rect,mPixmap);
        painter.end();
}
```

```
#include <QPainter>
#include <QPaintEvent>
#include "cardwidget.h"
```



cardbutton - cardbutton.cpp

CardButtonGroup: definíció

cardbuttongroup.h

```
typedef QList<CardButton*> CardButtonList;
class CardButtonGroup : public QFrame{
    Q_OBJECT
public:
    CardButtonGroup(QWidget *parent = 0);
    void newCardButtonGroup(int row, int col);
    CardButton* getCardButton(int i);
public slots:
    void updateCardButtonPixmap(int cardButtonId, QPixmap pixmap);
    void setCardButtonEnabled(int cardButtonId, bool b);
signals:
    void cardButtonClicked(int i);
    void cardButtonGroupChanged(int numRows, int numCols);
private:
    void insertCardButtons(int row, int col );
    void removeCardButtons();
    void debugCardButtons();
private:
    int mNumRows;
    int mNumCols;
    CardButtonList cardButtonList;
    QGridLayout *cardButtonLayout;
};
```

```
#include <QFrame>
#include <QList>

class QGridLayout;
class QPixmap;
class CardButton;
```

cardbuttongroup - cardbuttongroup.h

CardButtonGroup: konstruktor, newCardButtonGroup

cardbuttongroup.cpp

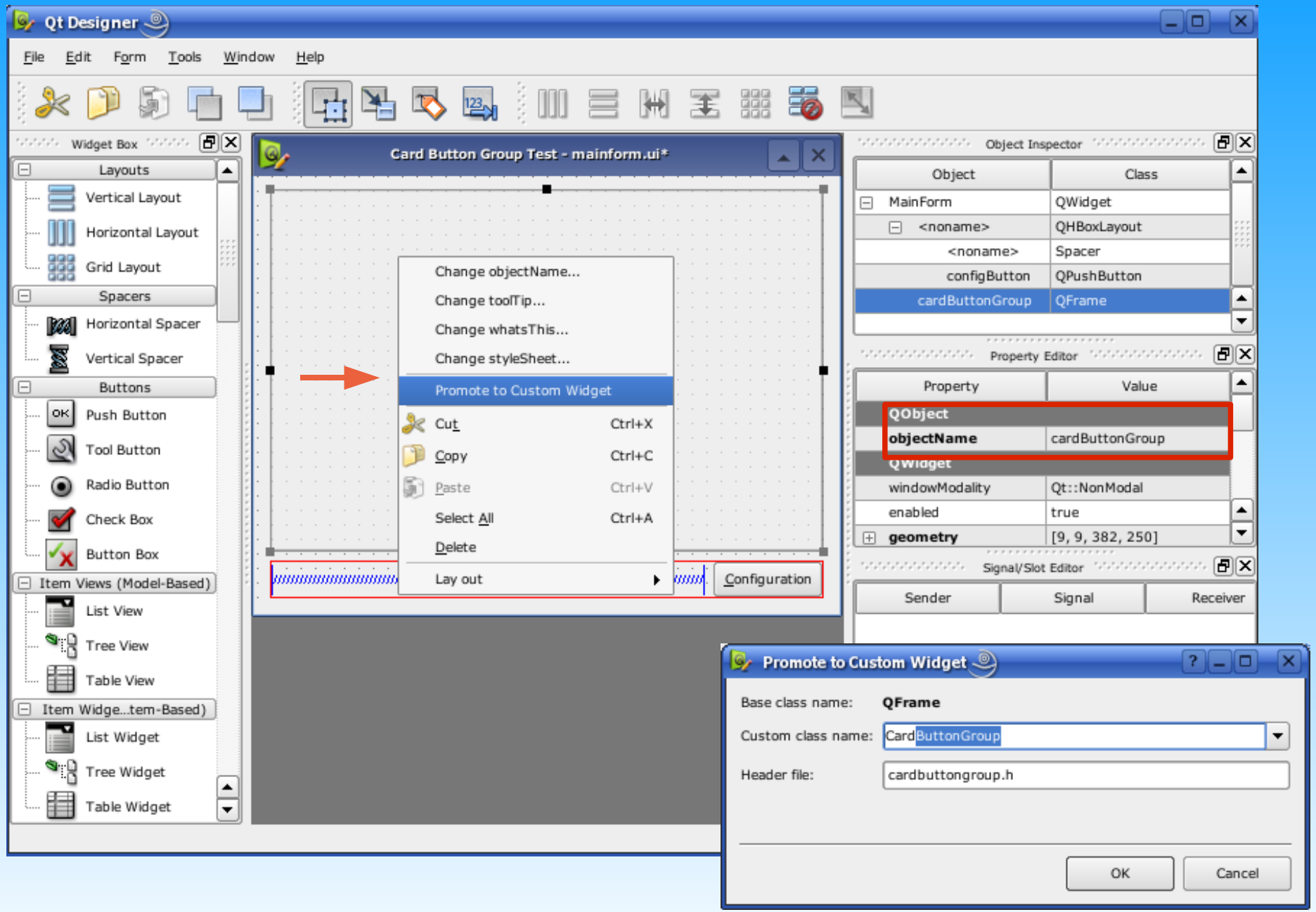
```
CardButtonGroup::CardButtonGroup(QWidget *parent)
        :QFrame(parent), mNumRows(0), mNumCols(0)
{
        cardButtonLayout = new QGridLayout;
        setLayout(cardButtonLayout);
}

void CardButtonGroup::newCardButtonGroup(int row, int col)
{
        removeCardButtons();
        insertCardButtons(row, col );
        mNumRows = row;
        mNumCols = col;
 }
```

CardButtonGroup: insert, remove, update

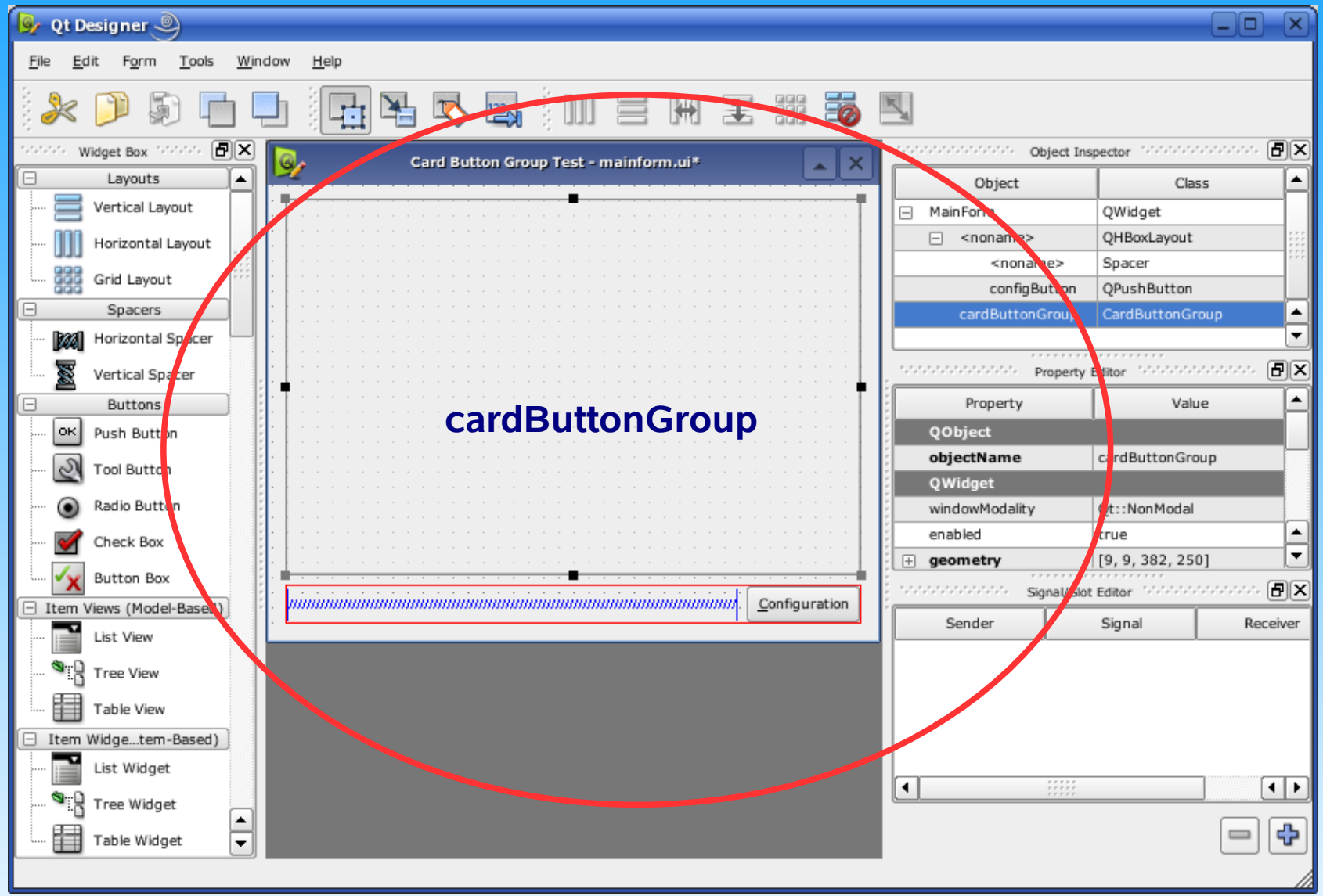
cardbuttongroup.cpp

```
void CardButtonGroup::insertCardButtons(int row, int col ) {
        CardButton *cardButton;
        for (int i = 0; i < row; i++) {
                for (int j = 0; j< col; j++) {
                        cardButton = new CardButton(this);
                        cardButton->setSizePolicy(
                                QSizePolicy( (QSizePolicy::Expanding),(QSizePolicy::Expanding)));
                        cardButton->setId(i*col+j);
                        cardButtonLayout->addWidget(cardButton,i,j);  //Registers cardButton to
                        cardButtonList.append(cardButton);      //Add cardButton to container
                        connect(cardButton, SIGNAL(cardButtonClicked(int)),
                                this, SIGNAL(cardButtonClicked(int)));
                }
        }
        show();
        emit cardButtonGroupChanged(row,col);
}
void CardButtonGroup::removeCardButtons( ) {
        while (!cardButtonList.isEmpty())
                delete  cardButtonList.takeFirst();
}
void CardButtonGroup::updateCardButtonPixmap(int cardButtonId, QPixmap pixmap) {
        cardButtonList.at(cardButtonId)->setPixmap(pixmap);
        cardButtonList.at(cardButtonId)->update();
}
```

Saját vezérlő használata Qt Designer-ben

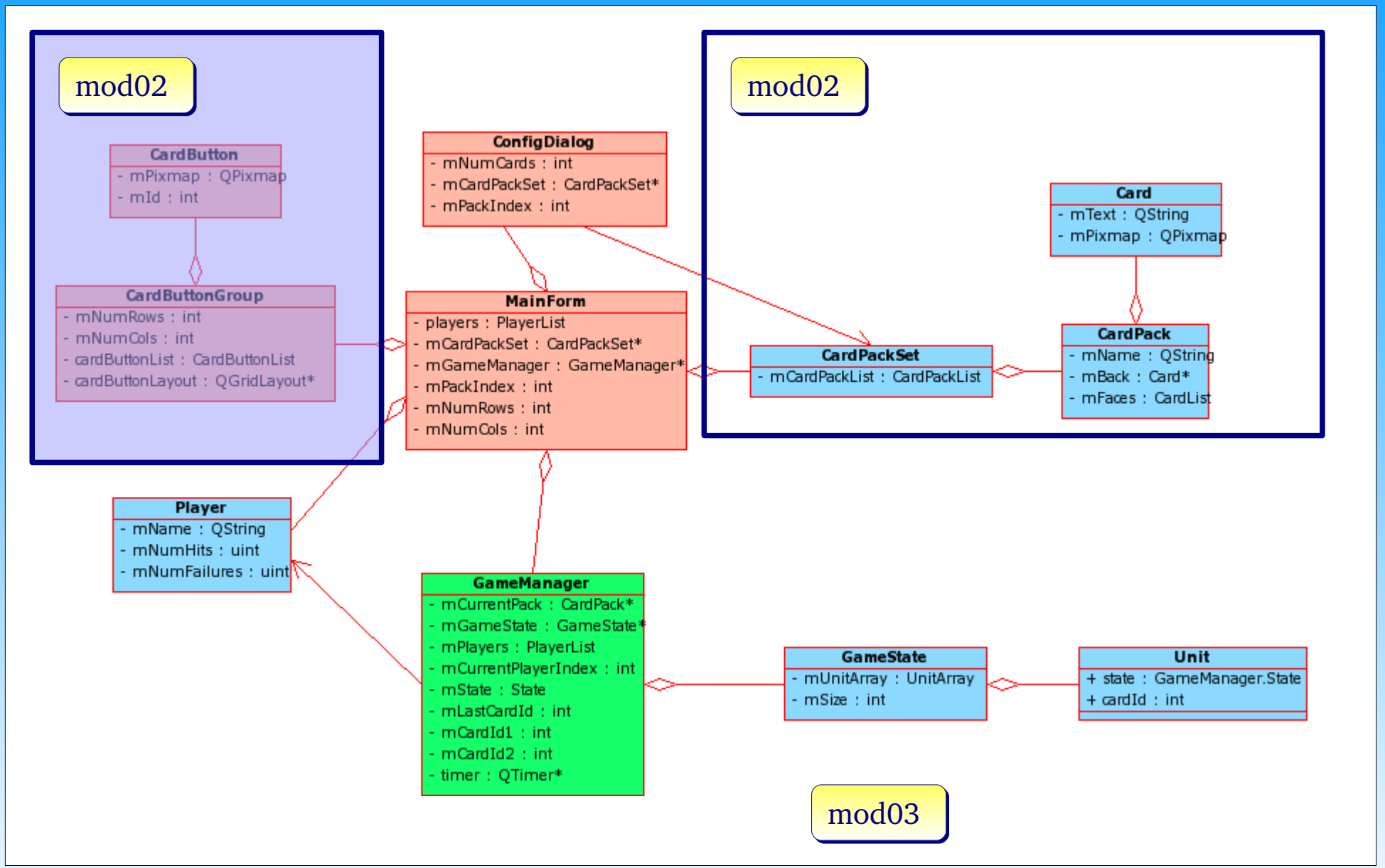


Saját vezérlő elhelyezése az űrlapon



cardgroup - cardgroup.ui

Az alkalmazás osztálydiagramja

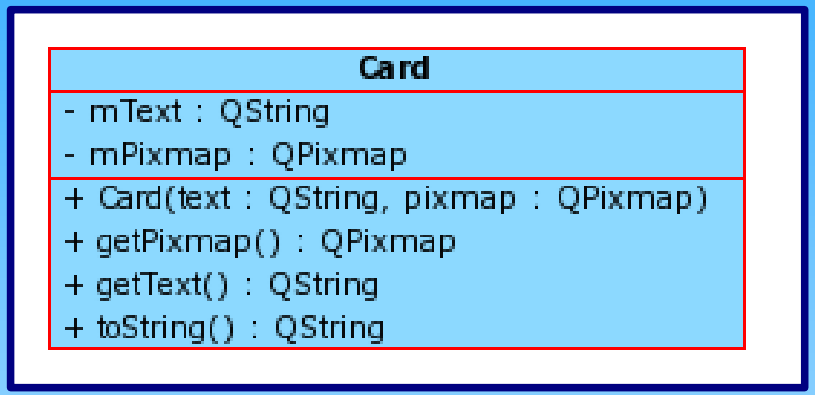


Card osztály

card– card.h, card.cpp

card.h

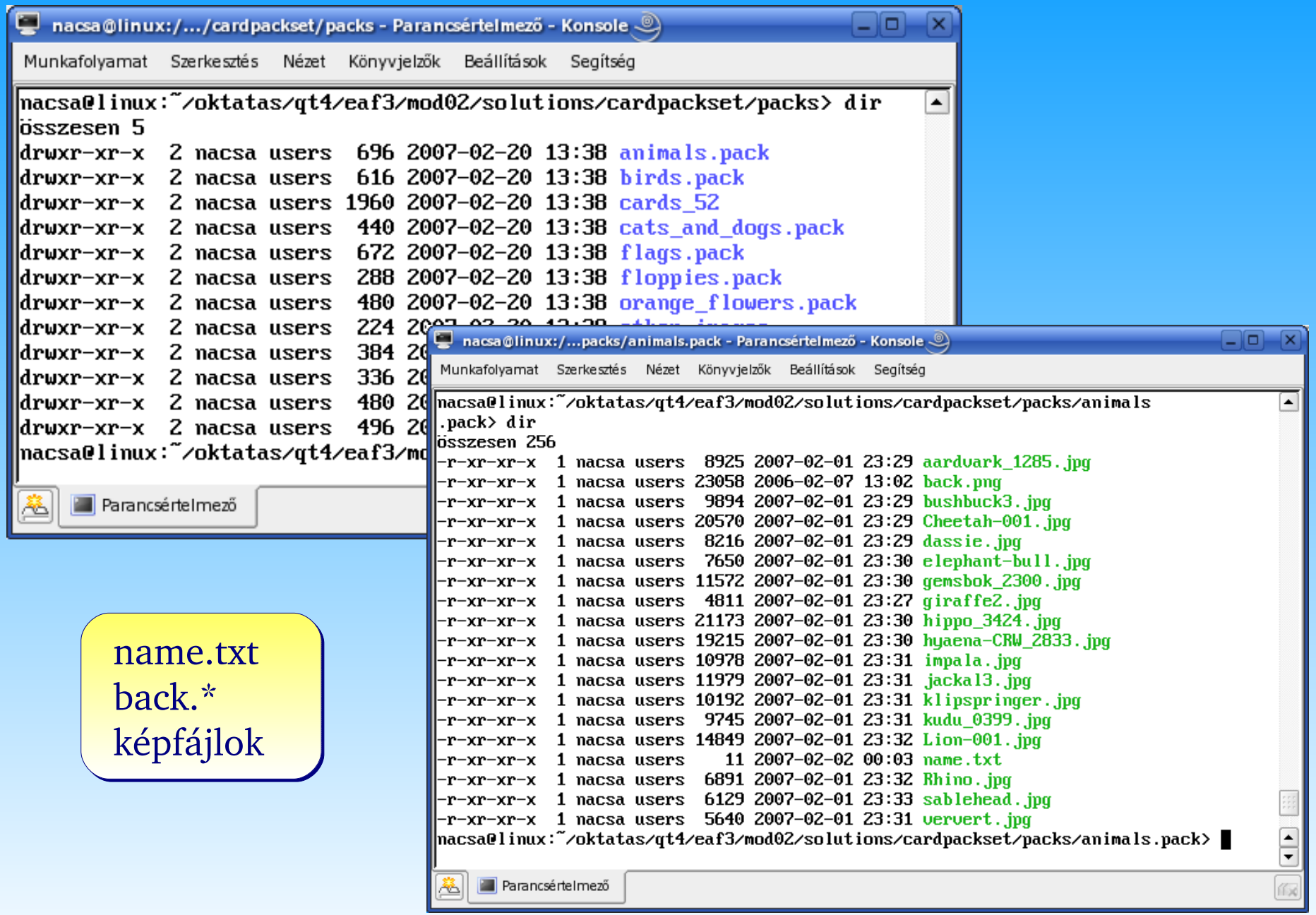
```
#include <Qpixmap>
class Card
{
public:
        Card(QString text, QPixmap pixmap);
        QPixmap getPixmap(){return mPixmap;}
        QString getText(){return mText;}
        QString toString();

private:
        QString mText;
        QPixmap mPixmap;
};
typedef QList<Card*> CardList;
```



card.cpp

```
Card::Card(QString text, QPixmap pixmap)
        : mText(text), mPixmap(pixmap)
{}
QString Card::toString() {
        QString ret = "{ Text=" + mText + " PixmapValid = "
            + QString::number(!mPixmap.isNull()) + " }";
        return ret;
}
```

Kártyacsomagok tárolása

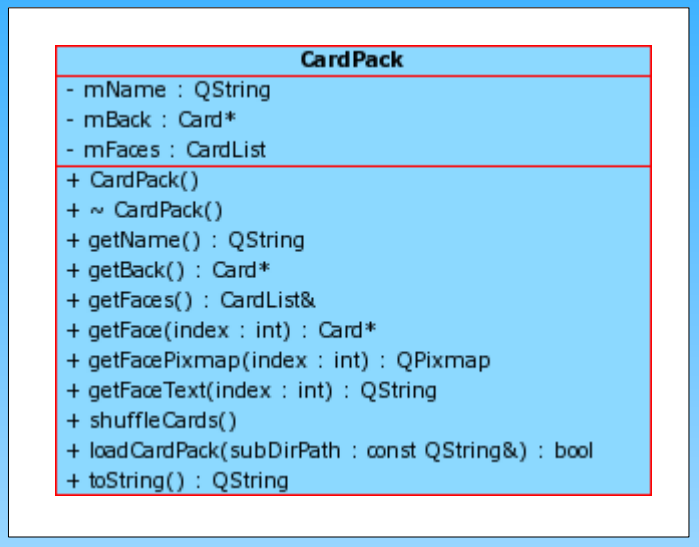


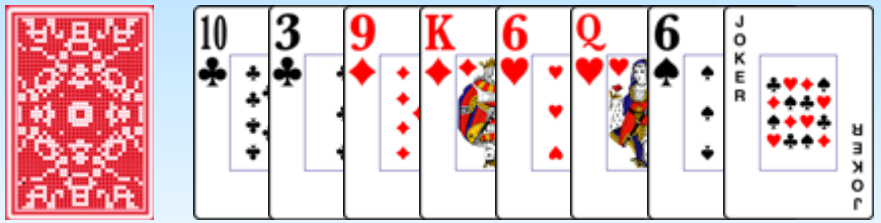
name.txt
back.*
képfájlok

CardPack osztály: definíció

cardpack.h

```
class CardPack
{
public:
        CardPack();
        ~CardPack();

        QString getName(){return mName;}
        Card* getBack(){return mBack;}
        CardList &getFaces(){return mFaces;}
        Card* getFace(int index);
        QPixmap getFacePixmap(int index);
        QString getFaceText(int index);
        void shuffleCards();
        bool loadCardPack(const QString &subDirPath) ;

        QString toString();

private:
        QString mName;
        Card* mBack; //updated by constructor
        CardList mFaces;
};

typedef QList<CardPack*> CardPackList;
```

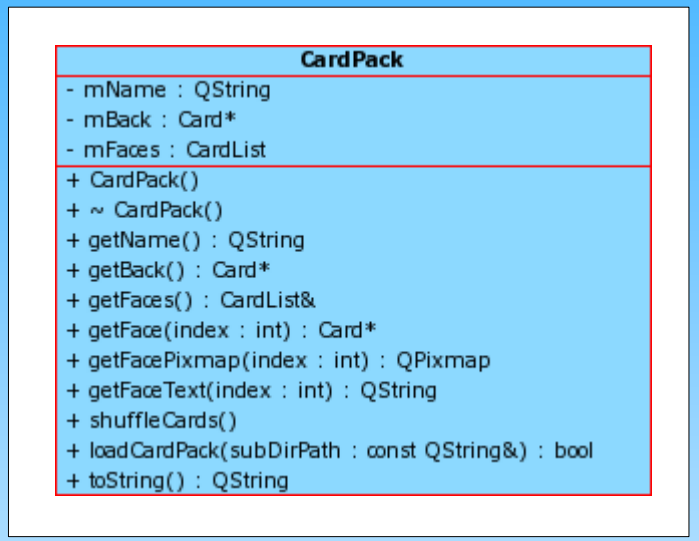




Kártya csomag osztály implementálása

cardpack.cpp

```
CardPack::CardPack() : mName(""), mBack(0)
{}

CardPack::~CardPack(){
        while (!mFaces.isEmpty())
                delete mFaces.takeFirst();
        delete mBack;
}

Card* CardPack::getFace(int index)
{
        return mFaces.at(index);
}

QPixmap CardPack::getFacePixmap(int index)
{
        return mFaces.at(index)->getPixmap();
}

QString CardPack::getFaceText(int index)
{
        return mFaces.at(index)->getText();
}
```



cardpack - cardpack.cpp

Kártyacsomag betöltése: csomag neve

cardpack.cpp

name.txt betöltése

```
bool CardPack::loadCardPack(const QString &subDirPath)
{
    //read name.txt
    QFile nameFile(subDirPath + "/name.txt");          //name.txt exists !!!
    if(!nameFile.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        qWarning(QString("Warning: ** name.txt file is missing or not a text file. ** \nsubDirPath \""
            +  subDirPath + "\"").toAscii());
        return false;
    }
    QTextStream in(&nameFile);
    while (!in.atEnd()) {
        mName += in.readLine();
    }

    . . .
```

Kártyacsomag hátlapjának betöltése, kártya létrehozása

cardpack.cpp

hátlap betöltése

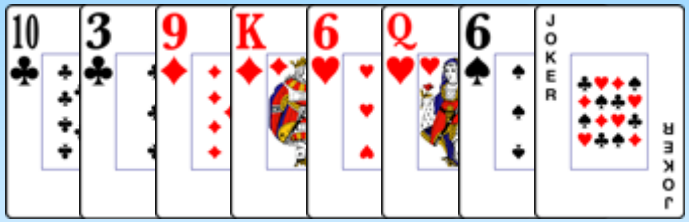
```
. . .
QStringList imageFilters;  //Create filters for images
foreach (QByteArray format, QImageReader::supportedImageFormats())
        imageFilters += "*." + format;

//Load back.*
QStringList allFiles;
foreach(QString file, QDir(subDirPath).entryList(imageFilters, QDir::Files))
        allFiles += QFileInfo(QDir(subDirPath),file).baseName();

if(!allFiles.contains("back"))
{
        qWarning(QString("Warning: ** Back file is missing from the \n\""
                + subDirPath + "\"\ndirectory.").toAscii());
        return false;
}

QStringList backFiles;
foreach(QString file, QDir(subDirPath).entryList(QStringList("back.*"), QDir::Files))
        backFiles += QFileInfo(QDir(subDirPath),file).fileName();

QString backFile = backFiles.at(0);     //Take the first back.*
mBack = new Card("Back",QPixmap(subDirPath + "/" + backFile));
. . .
```

Kártyacsomag előlapjainak betöltése

cardpack.cpp

előlapok betöltése

```
    . . .

    //Load faces
    //Add faces to subDir's pack but skip backs
    mFaces.clear();
    foreach(QString file, QDir(subDirPath).entryList(imageFilters, QDir::Files)){
        if (! (file.startsWith("back.")))
            mFaces.append(new Card(file, QPixmap(subDirPath + "/" + file)));
    }

    return true;
}
```



cardpack – cardpack.cpp : loadCardPack

Kártyacsomag előlapjainak keverése, “szövegesítése”

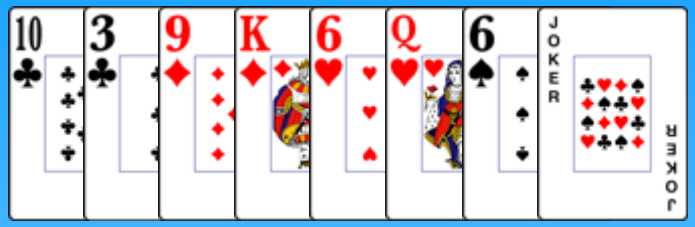
cardpack.cpp

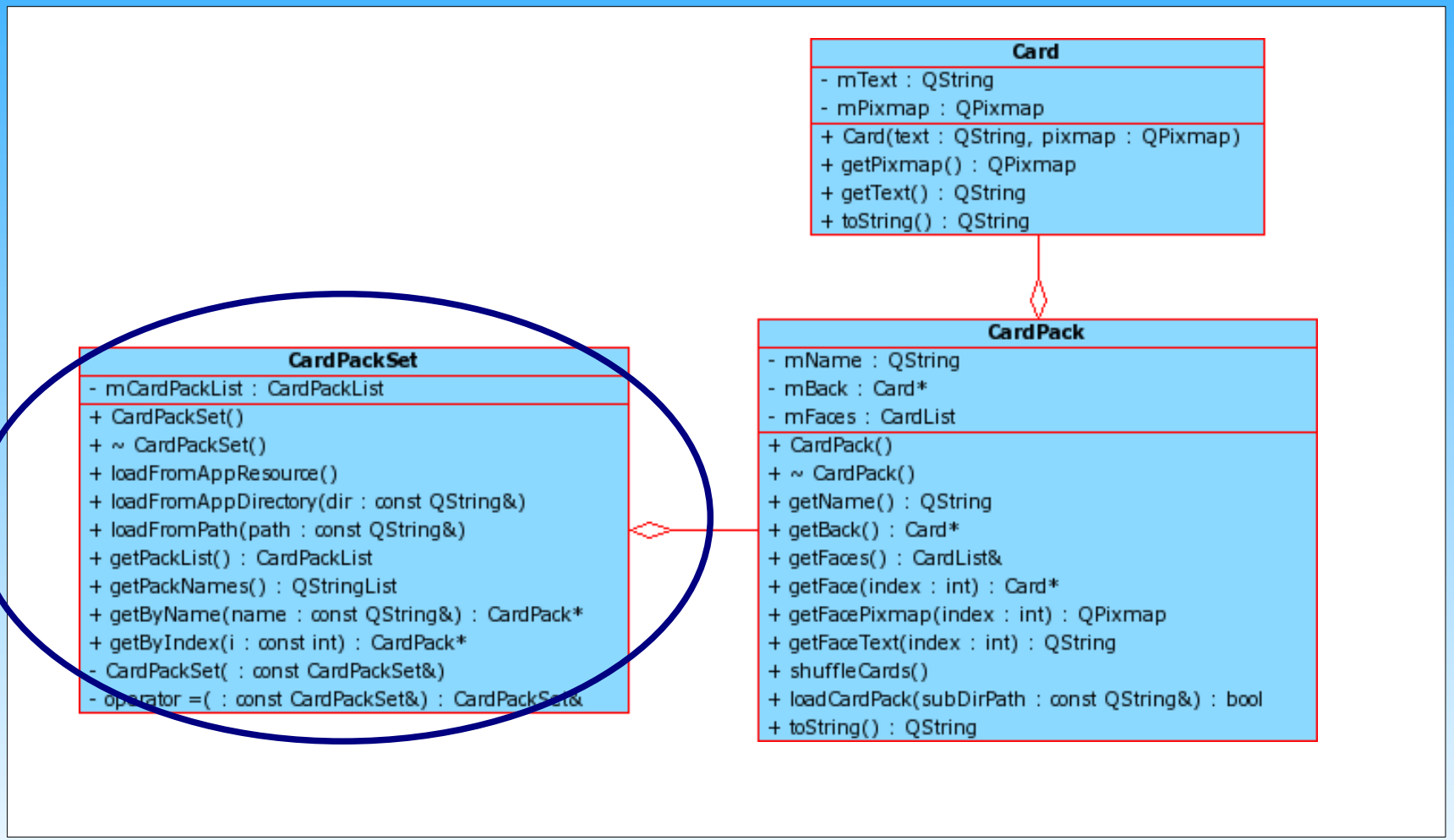
```
void CardPack::shuffleCards()
{
	for (int i= 0; i < mFaces.size() * mFaces.size(); ++i)
	{
		int ind1 = Utils::getNextRandomNumber(mFaces.size());
		int ind2 = Utils::getNextRandomNumber(mFaces.size());
		while(ind1 == ind2)
			ind2 = Utils::getNextRandomNumber(mFaces.size());
		qSwap(mFaces[ind1],mFaces[ind2] ); // <QtAlgorithms>
	}
}

QString CardPack::toString()
{
	QString ret = "\nPack name = " + mName + "\n";
	ret += QString("Back: %1\n").arg(mBack->toString());
	for (int i = 0; i < mFaces.size(); ++i) {
		ret += QString("\t   Face #%1: %2 \n").arg(i).arg(mFaces.at(i)->toString());
	}
	return ret;
}
```

előlapok keverése

előlapok „szövegesítése”

CardPackSet osztály



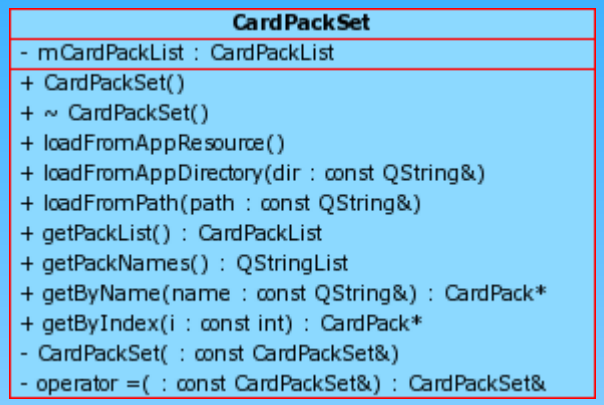


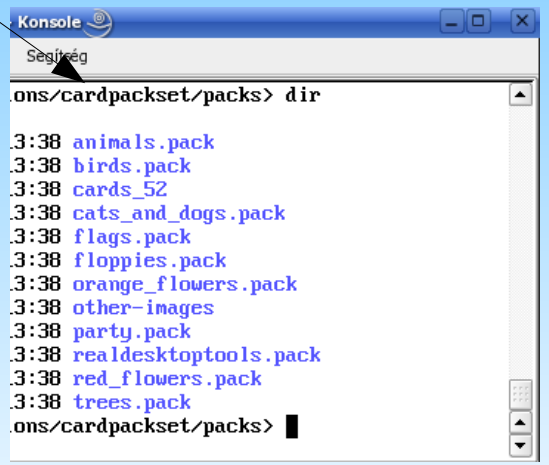
cardpackset - osztálydiagram

CardPackSet : definíció

cardpackset.h

```
#include "cardpack.h"

class CardPackSet
{
public:
        CardPackSet();
        ~CardPackSet();
        void loadFromAppResource();
        void loadFromAppDirectory(const QString &dir = "/packs");
        void loadFromPath(const QString &path);
        CardPackList getPackList(){return mCardPackList;};
        QStringList getPackNames();
        CardPack* getByName(const QString& name);
        CardPack* getByIndex(const int i);

private:
        CardPackSet(const CardPackSet&);
        CardPackSet& operator=(const CardPackSet&);

private:
        CardPackList mCardPackList;
};
```





CardPackSet : konstruktor, destruktor

cardpackset.cpp

```
CardPackSet::CardPackSet()
{}

CardPackSet::~CardPackSet()
{
        while (!mCardPackList.isEmpty())
                delete mCardPackList.takeFirst();
}
```

```
#include <QDir>
#include <QImageReader>
#include <QString>
#include <QFileInfo>
#include <QImage>
#include <QPixmap>
#include <QFile>
#include <QTextStream>
#include <QMessageBox>

#include "utils.h"
#include
"cardpackmanager.h"
#include "cardpack.h"
```

CardPackSet osztály: “getek”

cardpackset.cpp

```
QStringList CardPackSet::getPackNames()
{
	QStringList packNames;
	foreach(CardPack *pack, mCardPackList)
		packNames += pack->getName();
	return packNames;
}

CardPack* CardPackSet::getByName(const QString& name)
{
	CardPack *cardPack = NULL;
	foreach(CardPack *pack, mCardPackList)
		if (name == pack->getName())  cardPack = pack;
	return cardPack;
}

CardPack* CardPackSet::getByIndex(const int index)
{
	CardPack *cardPack = NULL;
	if(index < mCardPackList.size())
		cardPack = mCardPackList.at(index);
	return cardPack;
}
```

CardPackSet osztály: kártyacsomagok betöltése

cardpackset.cpp

```
void CardPackSet::loadFromAppResource()
{
	loadFromPath(QString(":/datafiles/packs"));
}

void CardPackSet::loadFromAppDirectory(const QString &dir)
{
	loadFromPath(Utils::getApplicationDirPrefix() + dir);
}
```

card.qrc

```
<!DOCTYPE RCC>
<RCC version="1.0">
<qresource>
	<file alias="datafiles/packs/default.pack/name.txt">datafiles/packs/default.pack/name.txt</file>
	<file alias="datafiles/packs/default.pack/back.png">datafiles/packs/default.pack/back.png</file>
	<file alias="datafiles/packs/default.pack/01.jpg">datafiles/packs/default.pack/01.jpg</file>
	. . .
	<file alias="datafiles/packs/default.pack/12.jpg">datafiles/packs/default.pack/12.jpg</file>
	<file alias="datafiles/packs/default.pack/13.jpg">datafiles/packs/default.pack/13.jpg</file>
	<file alias="datafiles/packs/default.pack/14.jpg">datafiles/packs/default.pack/14.jpg</file>
</qresource>
</RCC>
```

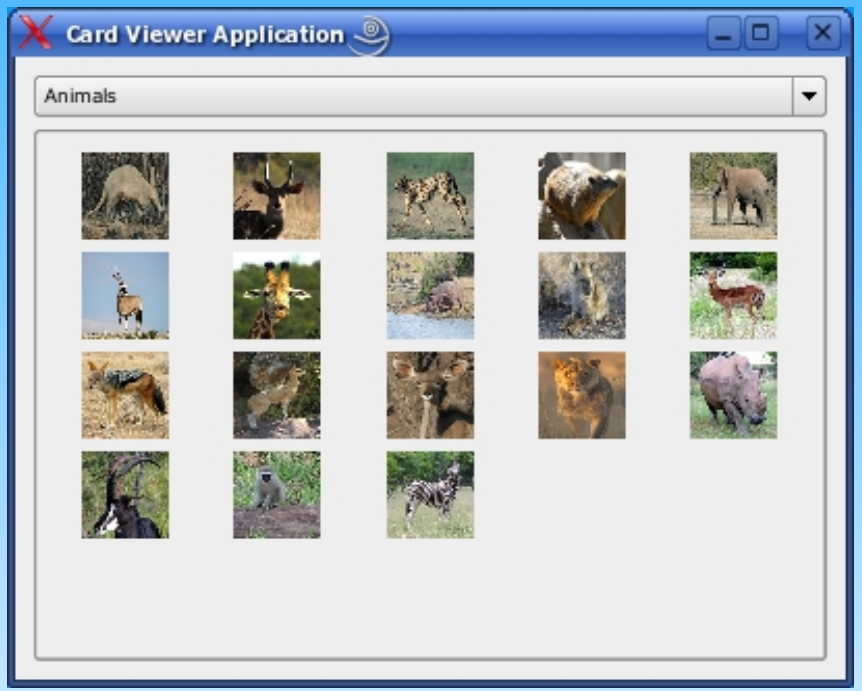
CardPackSet osztály: csomagok betöltése

cardpackset.cpp

```
void CardPackSet::loadFromPath(const QString &path)
{
   mCardPackList.clear();
   QDir packsDir(path);

    //Create subdirs of packs directory without . and ..
   QStringList subDirs;
   foreach (QString subDir, packsDir.entryList(QDir::Dirs|QDir::NoDotAndDotDot))
   {
       subDirs += subDir;
   }

   for (int i = 0; i < subDirs.size(); ++i)
   {
      QString subDirPath = path + "/" + subDirs.at(i);
      QDir subDir(subDirPath);

      CardPack *pack = new CardPack();
      if(pack->loadCardPack(subDirPath))
            mCardPackList.append(pack);
    }
 }
```

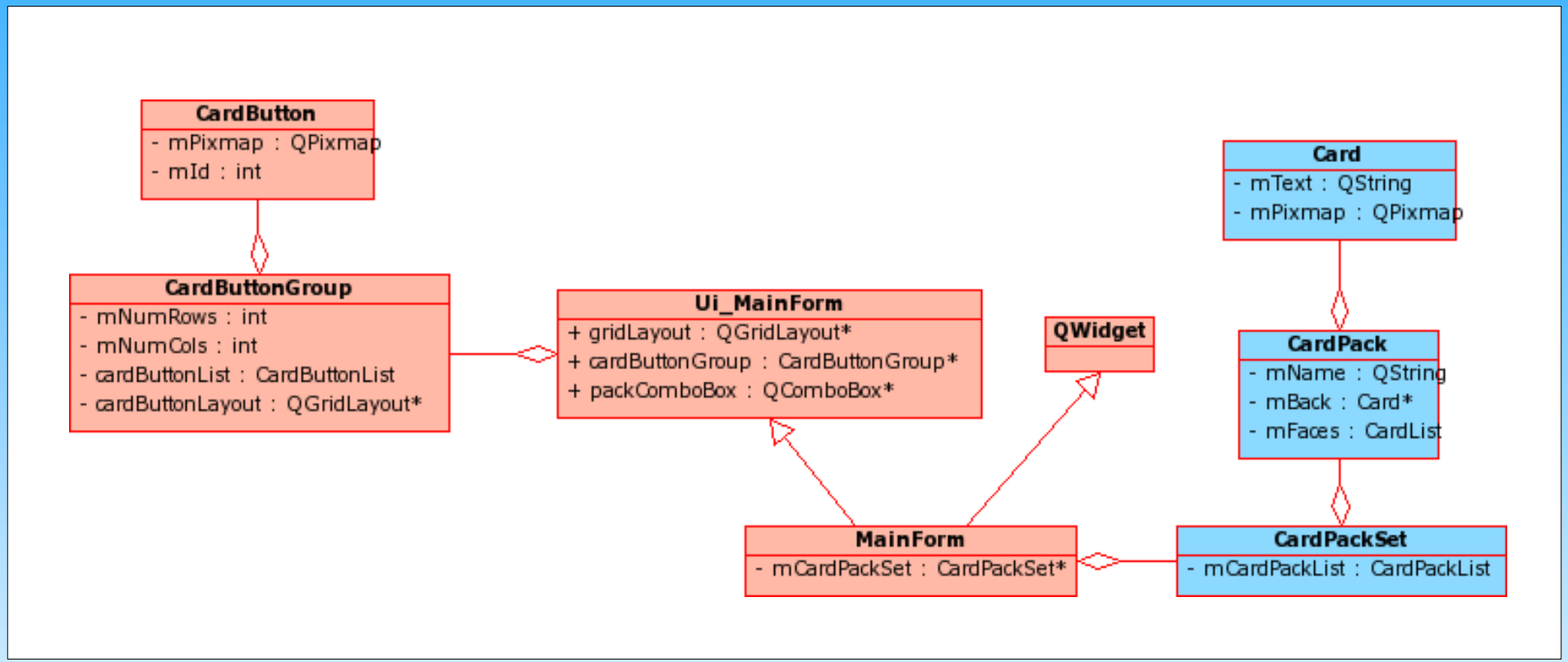
Kártya nézegető (cardviewer) alkalmazás

cardviewer

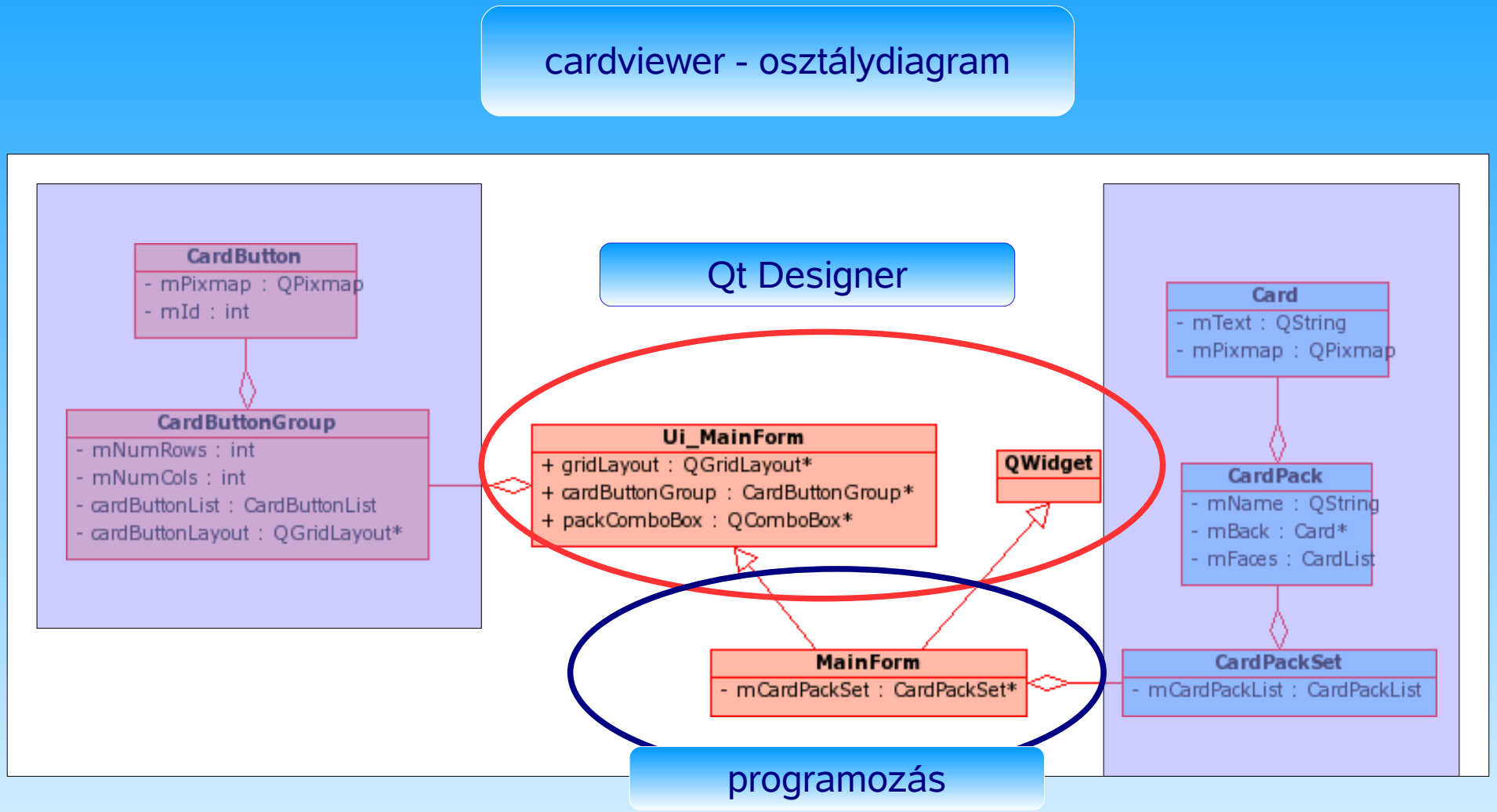


A kártya nézegető alkalmazás osztálydiagramja

cardviewer - osztálydiagram

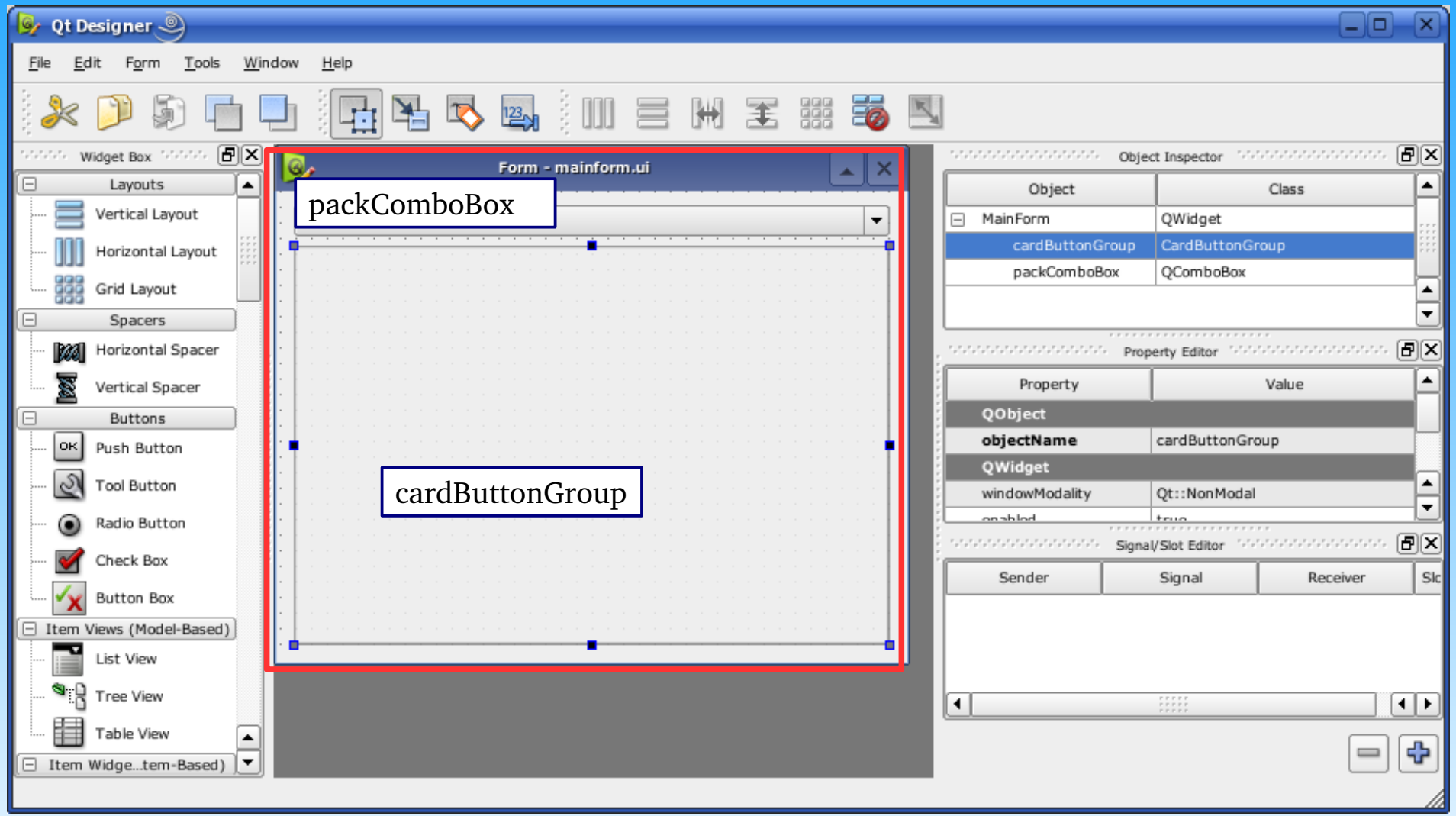


A kártya nézegető alkalmazás osztálydiagramja



Az alkalmazás felületterve

mainform.ui



CardViewer: mainform.h

mainform.h

```
#include <QWidget>

#include "ui_mainform.h"

class CardPackSet;

class MainForm : public QWidget, private Ui_MainForm
{
    Q_OBJECT
public:
    MainForm(QWidget *parent = 0);

public slots:
    void on_packComboBox_currentIndexChanged(int index);
    void slotCardClicked(int i);

private:
    CardPackSet* mCardPackSet;

};
```

CardViewer: mainform.cpp

mainform.cpp

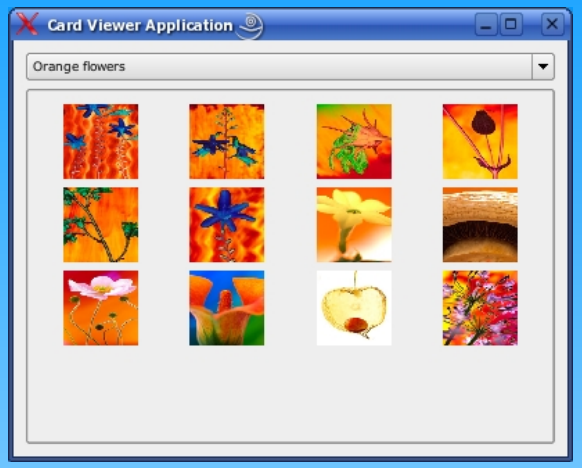
```
#include <QMessageBox>
#include <math.h>
#include "cardpackset.h"
#include "cardbutton.h"
#include "mainform.h"

MainForm::MainForm(QWidget *parent)
        : QWidget(parent)
{
        setupUi(this);

        mCardPackSet = new CardPackSet();
        mCardPackSet->loadFromAppDirectory();

        packComboBox->addItems(mCardPackSet->getPackNames());
        packComboBox->setCurrentIndex(0);

        connect(cardButtonGroup, SIGNAL(cardButtonClicked(int)), this, SLOT(slotCardClicked(int)));

}
```

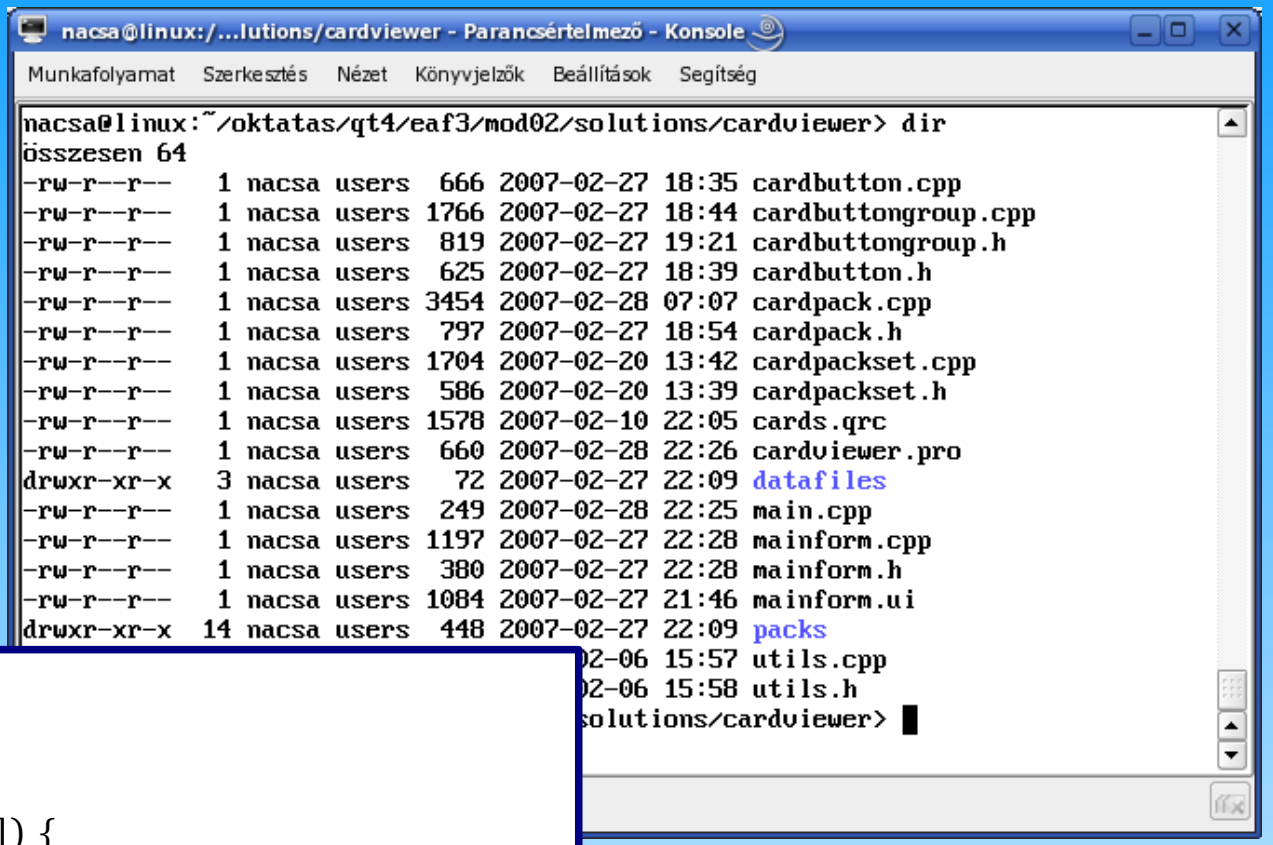
CardViewer: mainform.cpp

mainform.cpp

```
void MainForm::on_packComboBox_currentIndexChanged(int index)
{
      int side = (int) sqrt(mCardPackSet->getPackList().at(index)->getFaces().count()) + 1; //<math.h>
      cardButtonGroup->newCardButtonGroup(side,side);

      for (int i= 0; i< mCardPackSet->getPackList().at(index)->getFaces().count(); i++)
      {
            cardButtonGroup->updateCardButtonPixmap(i,
                        QPixmap(mCardPackSet->getPackList().at(index)->getFacePixmap(i)));
            QString text = mCardPackSet->getPackList().at(index)->getFaces().at(i)->getText();
            cardButtonGroup->getCardButton(i)->setToolTip(text.left(text.indexOf(".")));
      }
}

void MainForm::slotCardClicked(int i)
{
      QMessageBox::information(this, "Card Group Test Information", "Card Click \nCardId =  "
                        + QString::number(i) );
}
```

Fordítás, futtatás





main.cpp

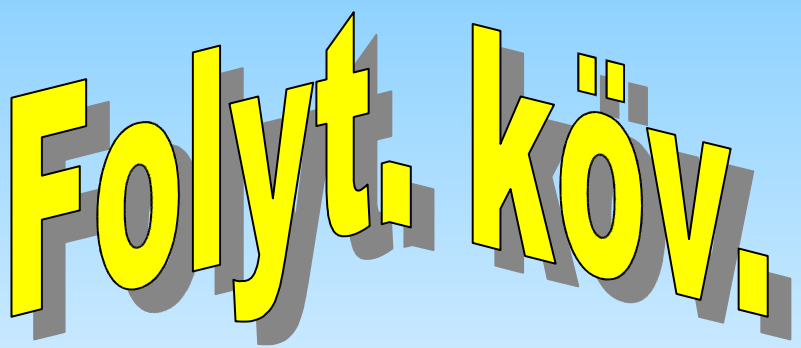
```
#include <QApplication>
#include "mainform.h"

int main (int argc, char *argv[]) {
        QApplication app(argc,argv);

        MainForm *mainForm = new MainForm;
        mainForm->setWindowTitle("Card Viewer Application");
        mainForm->show();

        return app.exec();
}
```

```
qmake -project
qmake
cardviewer.pro
make
./cardviewer
```

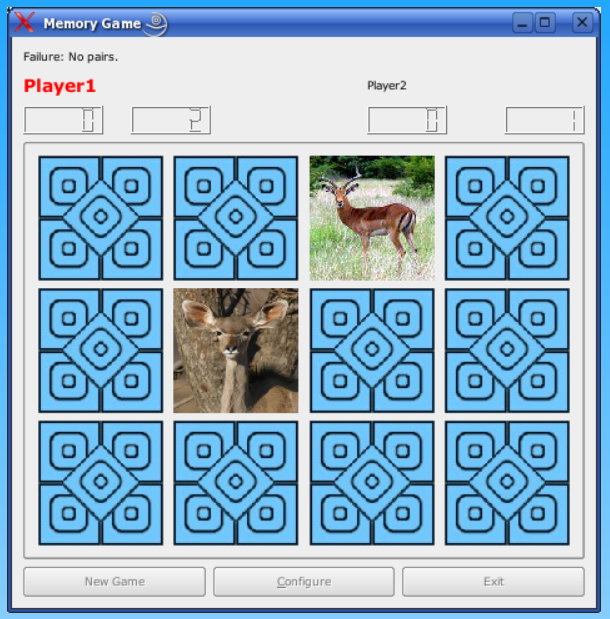
A bemutatott osztályok, az azokat tesztelő projektek, valamint a kártya nézegető alkalmazás programjai megtalálhatók a

people.inf.elte.nacsa/qt4/eaf3/mod02/projects/

címen.



Elemi alkalmazások fejlesztése III.



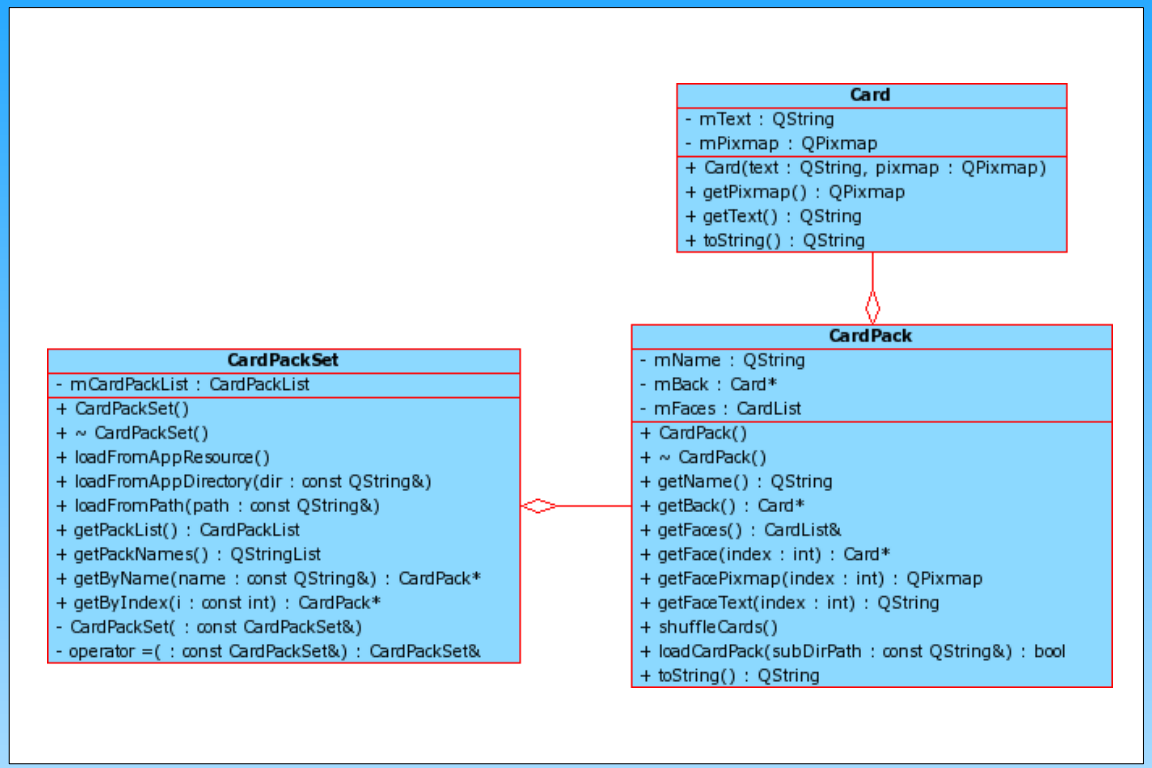
Memóriajáték 2.

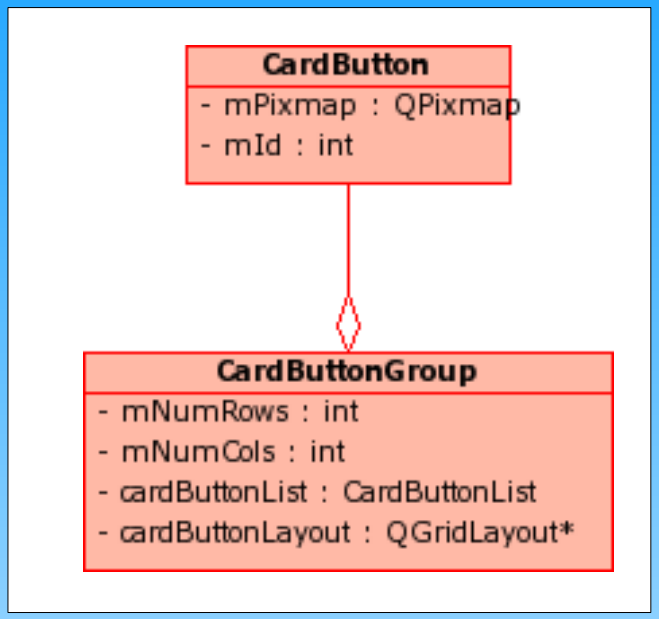
Készítette: Szabóné Nacsa Rozália
nacsa@inf.elte.hu
people.inf.elte.hu/nacsa/qt4/eaf3/

Qt 4

2007

Hol tartunk?



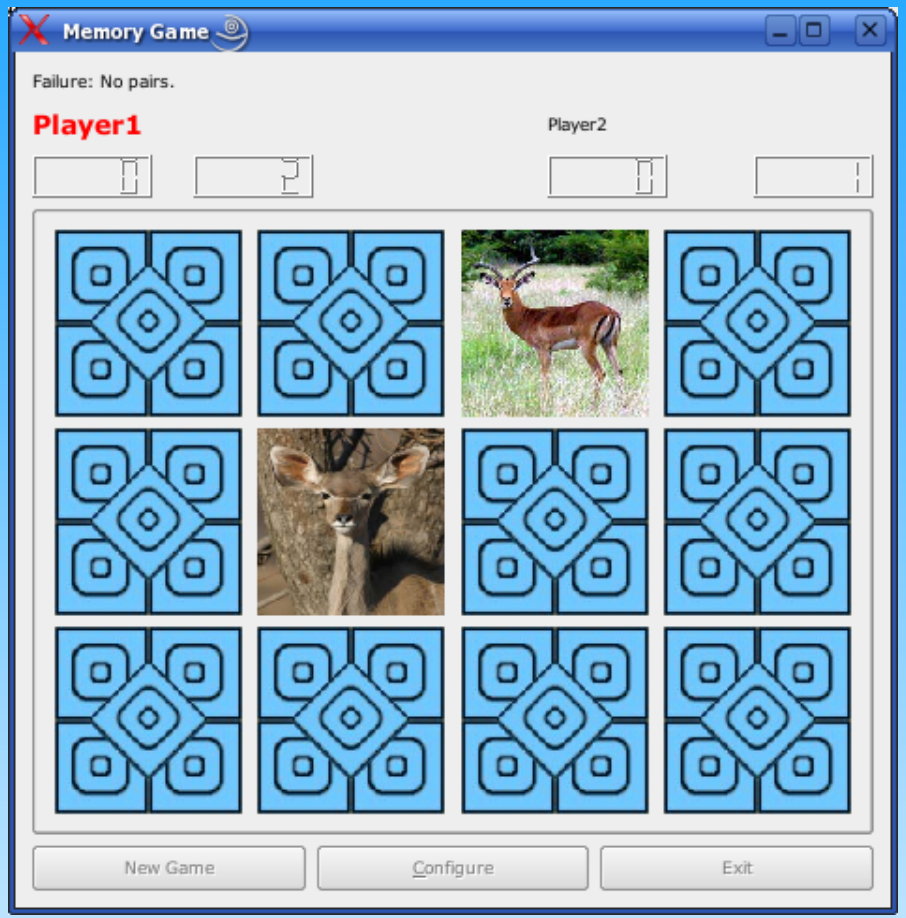


Adatokért felelős osztályok

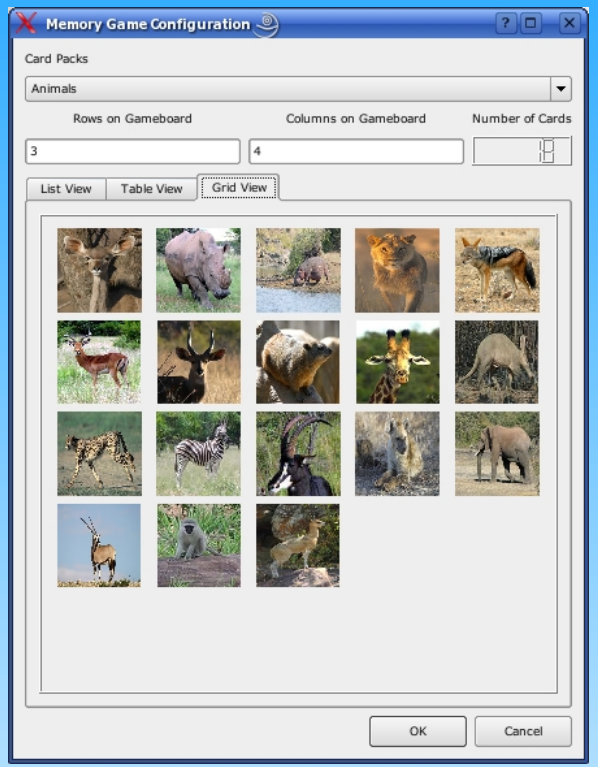
Megjelenésért felelős osztályok

Ezeket az osztályokat elkészítettük az előző modulban.

Mit szeretnénk?

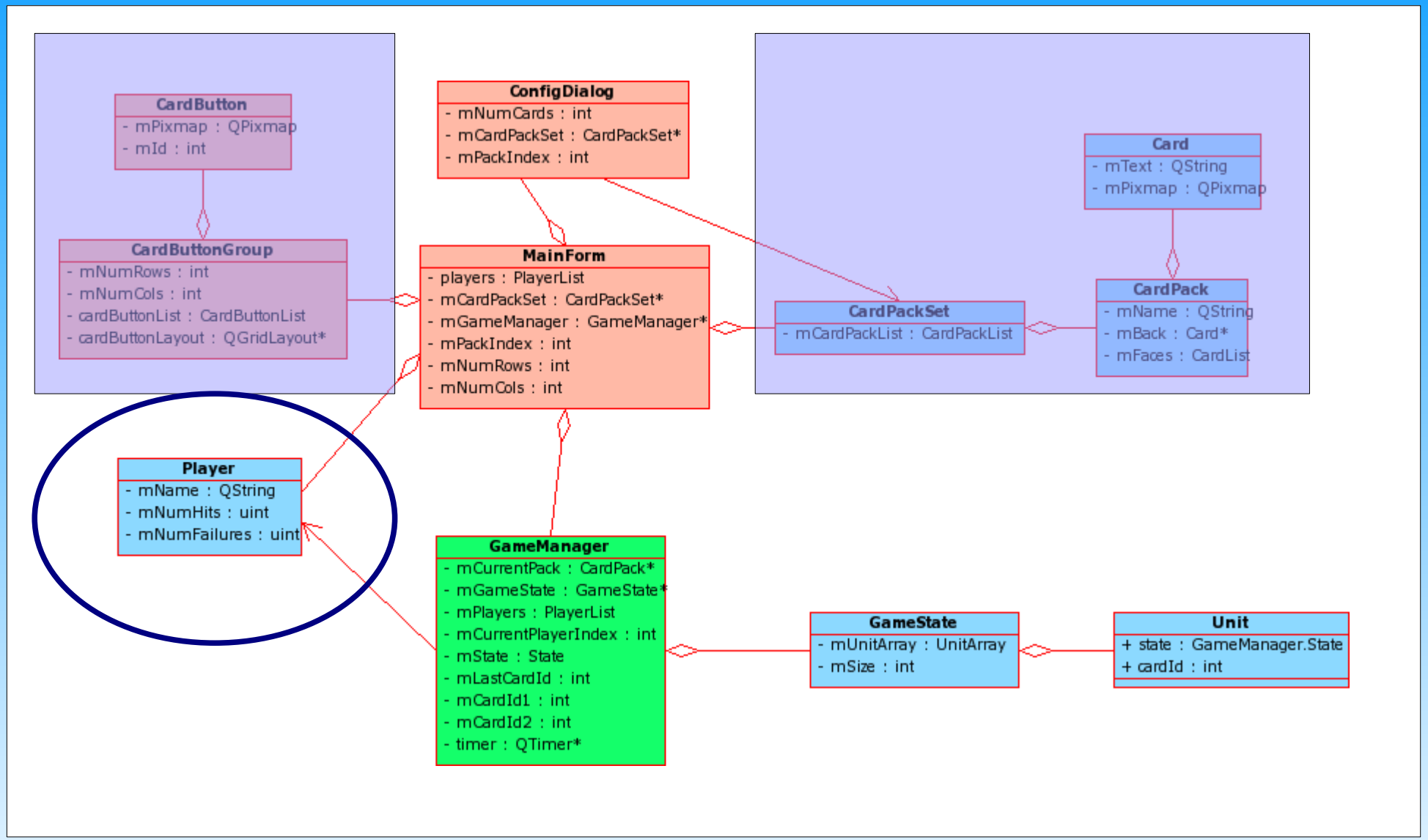


Memóriajáték



A játék beállítása

Osztálydiagram: Player osztály

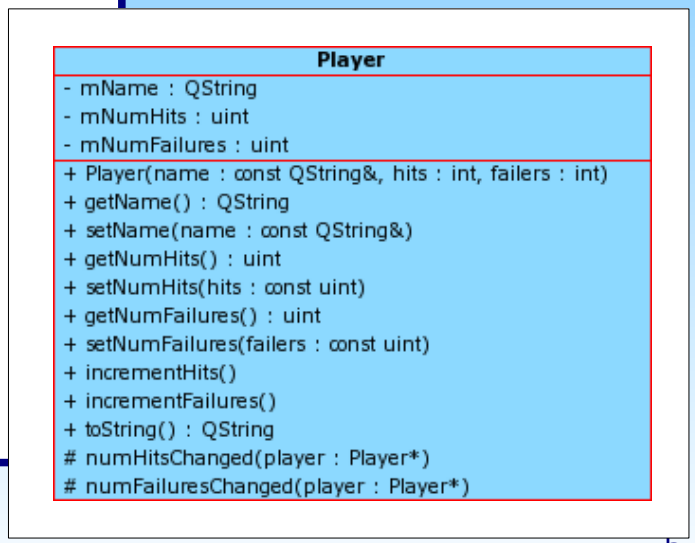


Player osztály: definíció

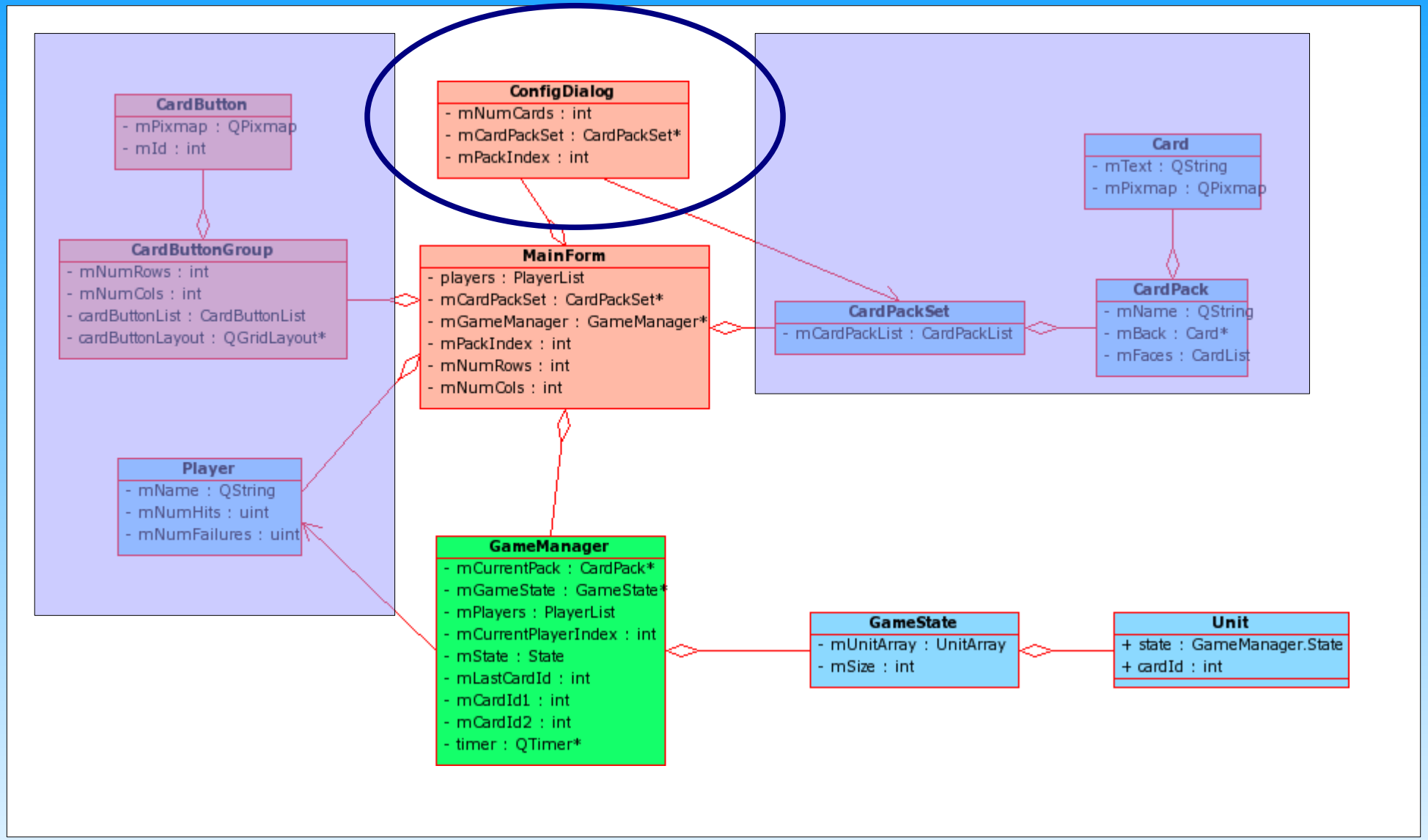
player.h

```
class Player : public QObject
{
    Q_OBJECT
public:
    Player(const QString& name, int hits = 0, int failers = 0);

    QString getName() const {return mName;}
    void setName(const QString& name) {mName = name;}
    uint getNumHits() const {return mNumHits;}
    void setNumHits(const uint hits){mNumHits = hits;}
    uint getNumFailures() const {return mNumFailures;}
    void setNumFailures(const uint failers){mNumFailures = failers;}
    void incrementHits();
    void incrementFailures();
    QString toString();
signals:
    void numHitsChanged(Player* player);
    void numFailuresChanged(Player* player);
private:
    QString mName;
    uint mNumHits;
    uint mNumFailures;
};
```

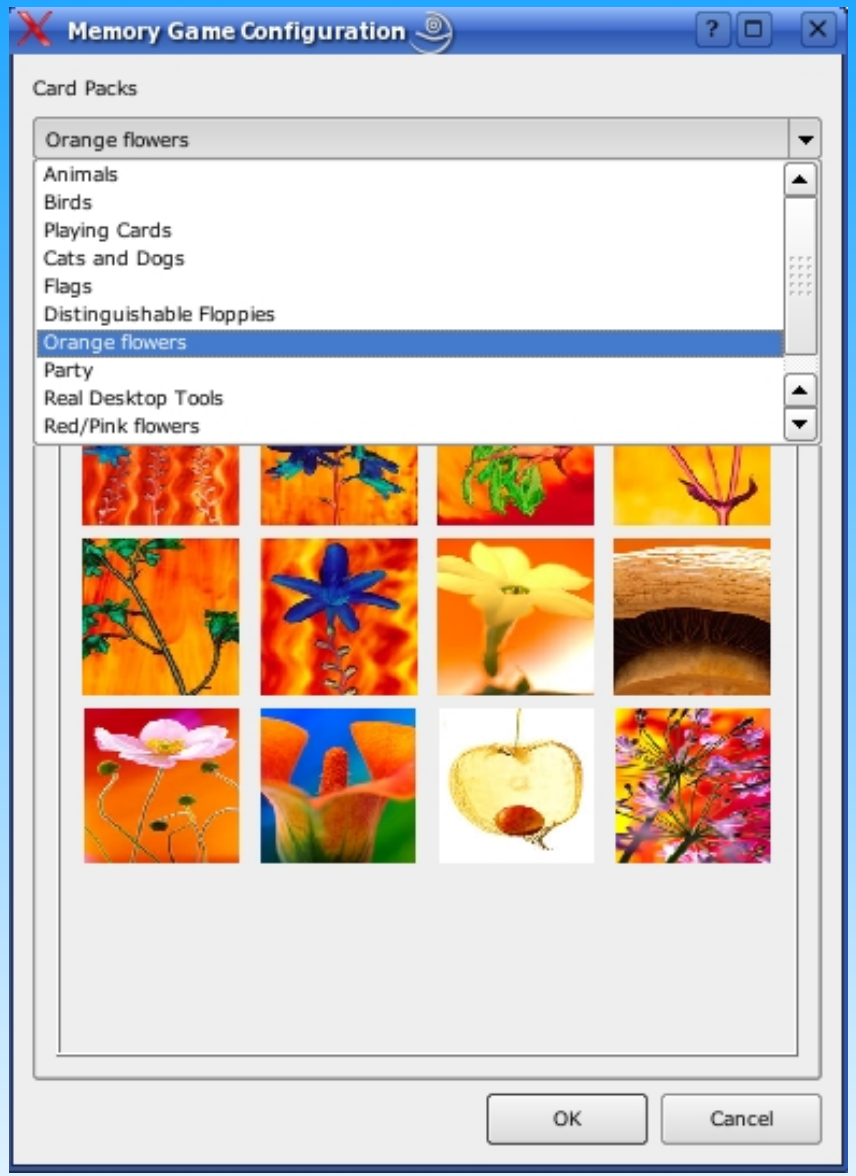
```
#include <QObject>
```



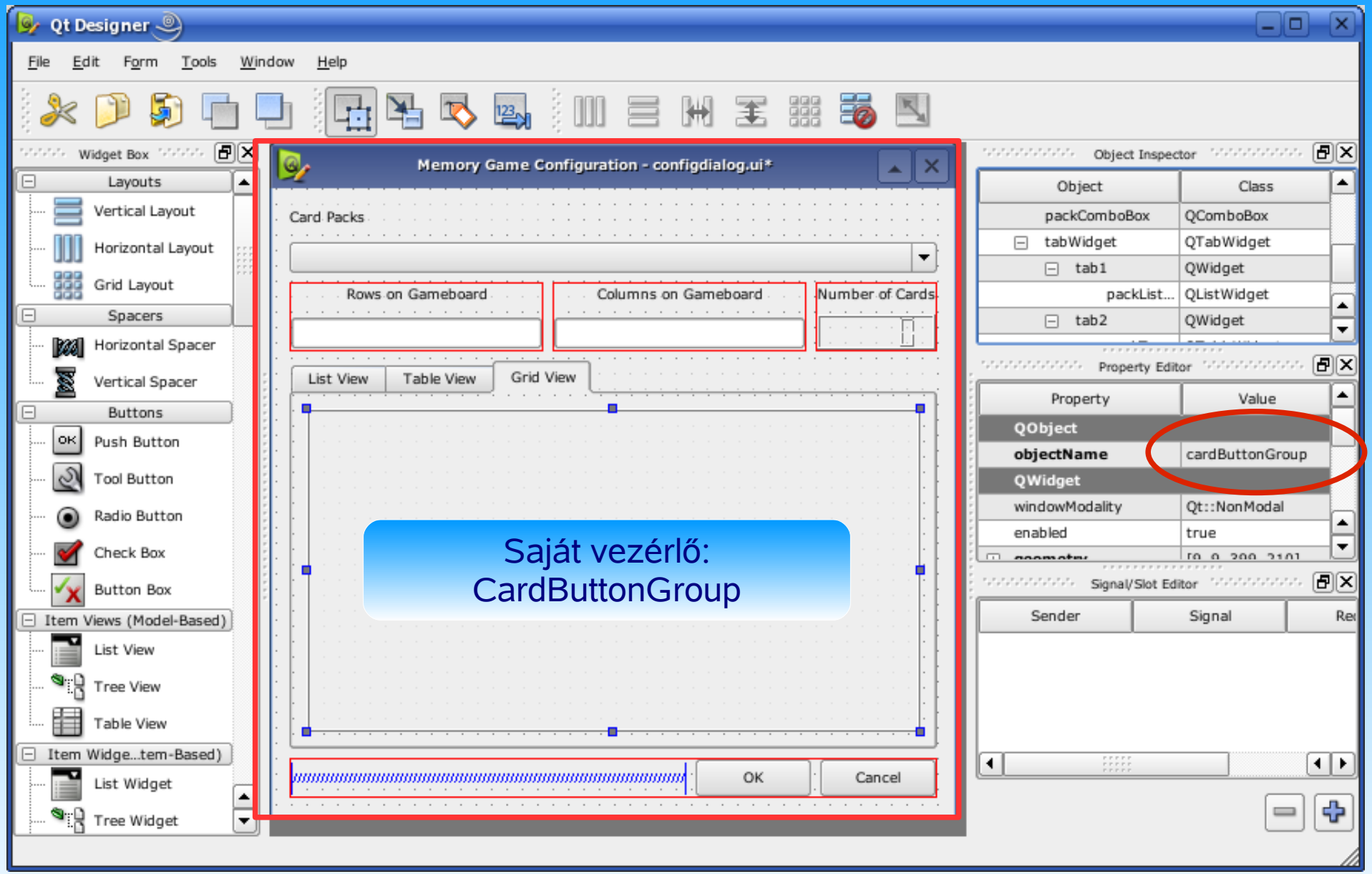
Osztálydiagram: ConfigDialog



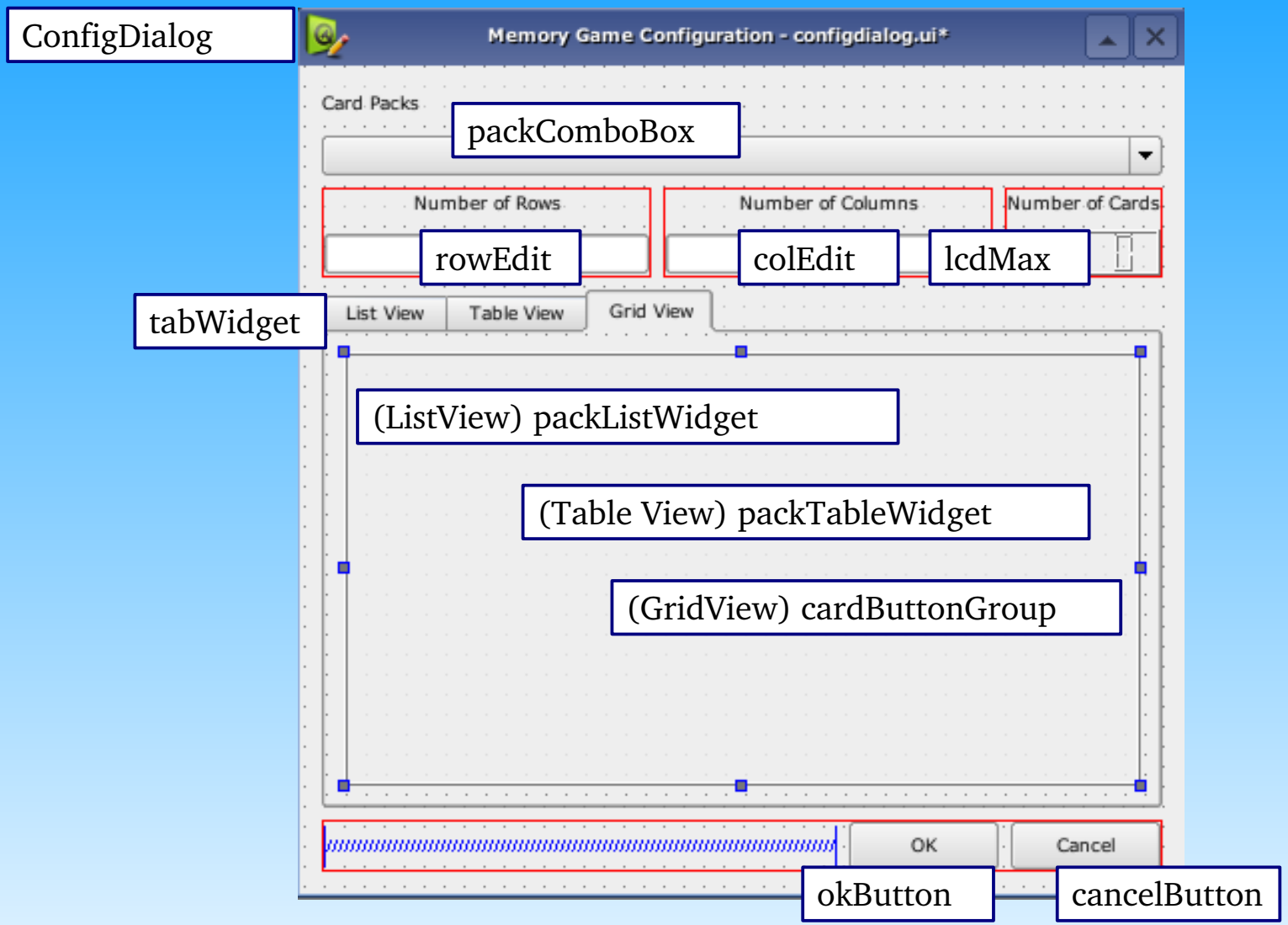
Cél: ConfigDialog elkészítése



A ConfigDialog megtervezése Qt Designerben



ConfigDialog: vezérlők



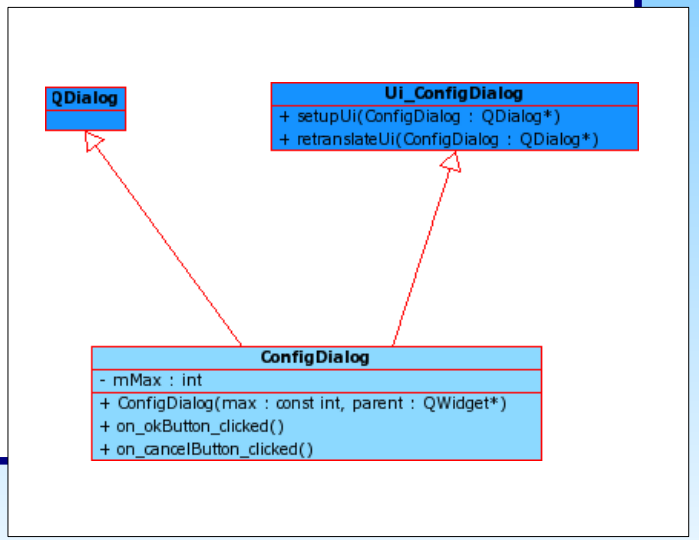
ConfigDialog: definíció

configdialog.h

```
#include <QWidget>
#include "ui_configdialog.h"
```

```
class CardPackSet;

class ConfigDialog : public QDialog, public Ui_ConfigDialog {
    Q_OBJECT
public:
    ConfigDialog(CardPackSet* cardPackSet, int packIndex, int row, int col, QWidget * parent=0);
public slots:
    void on_okButton_clicked();
    void on_cancelButton_clicked();
    void on_packComboBox_currentIndexChanged(int index);
private:
    void updatePackComboBox();
    void updateNumCards();
    void updatePackListWidget();
    void updatePackTableWidget();
    void updatePackCardGrid();
    void updateView();
private:
    CardPackSet *mCardPackSet;
    int mPackIndex;
};
```

Slotok automatikus elnevezése



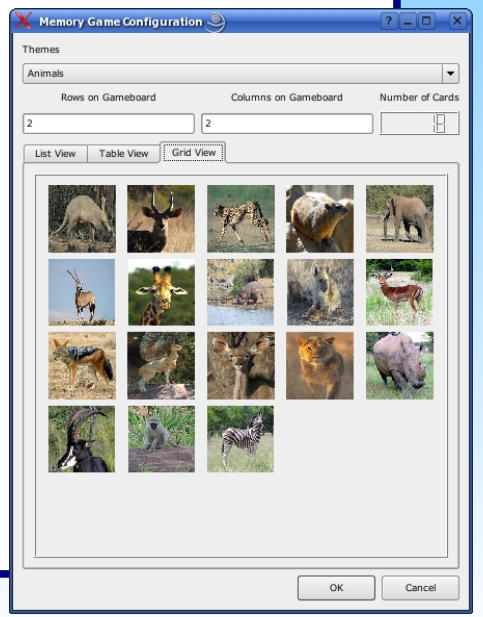
config - configdialog.h

ConfigDialog: konstruktor, eseménykezelők

configdialog.cpp

```
ConfigDialog::ConfigDialog(CardPackSet *cardPackSet, int packIndex, int row, int col, QWidget * parent)
        : QDialog(parent), mCardPackSet(cardPackSet), mPackIndex(packIndex)
{
        setupUi(this);
        rowEdit->setText(QString::number(row));
        colEdit->setText(QString::number(col));
        packComboBox->setCurrentIndex(mPackIndex);
        updateNumCards();
        updatePackComboBox();
}
void ConfigDialog::on_okButton_clicked()
{
        int i = rowEdit->text().toInt() * colEdit->text().toInt();
        int numCards = mCardPackSet->getPackLi st().at(mPackIndex)->getFaces().count() * 2;
        if(i == 0 || i > numCards || i%2 != 0)
                QMessageBox::information(0,"MainForm","Too big or not pair");
        else
                accept();
}
void ConfigDialog::on_cancelButton_clicked()
{
        reject();
}
void ConfigDialog::on_packComboBox_currentIndexChanged(int index){
        mPackIndex = index;
        updateView();
}
```

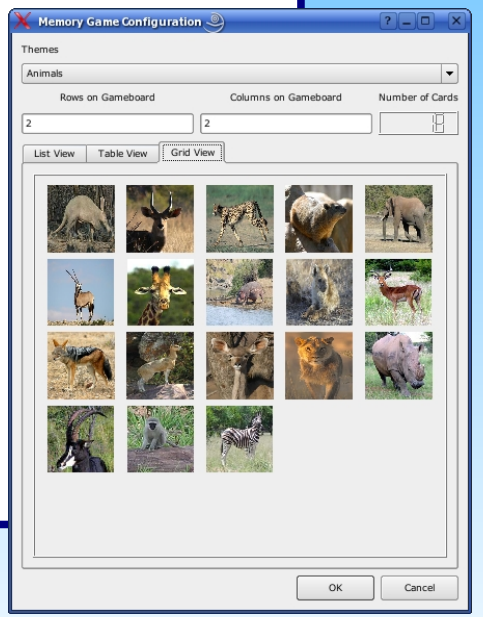
```
#include <QMessageBox>
#include <QLineEdit>
#include <QListWidget>
#include <QListWidgetItem>
#include <math.h>

#include "cardpackset.h"
#include "cardbutton.h"
#include "configdialog.h"
```



ConfigDialog: eseménykezelők

configdialog.cpp

```
void ConfigDialog::updatePackComboBox()
{
	packComboBox->addItems(mCardPackSet->getPackNames());
	packComboBox->setCurrentIndex(mPackIndex);
}

void ConfigDialog::updateNumCards()
{
	lcdMax->display(mCardPackSet->getPackList().at(mPackIndex)->getFaces().count());
}


void ConfigDialog::updateView()
{
	updateNumCards();
	updatePackListWidget();
	updatePackTableWidget();

	updatePackCardGrid();
}
```

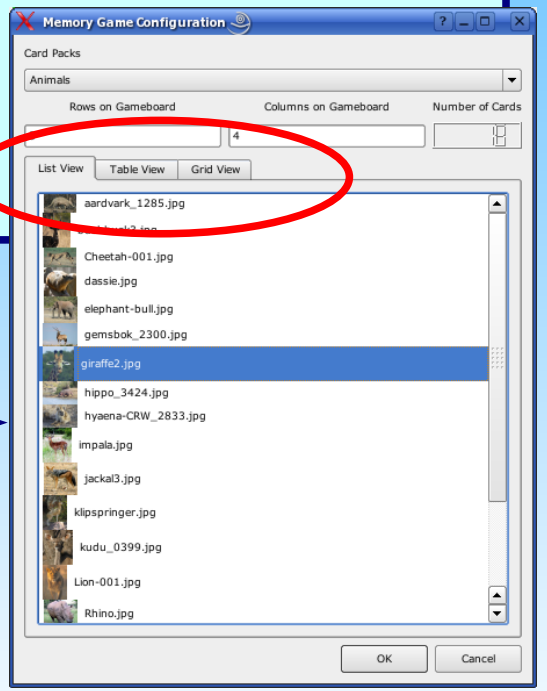


ConfigDialog: “Lista nézet”

configdialog.cpp

```
void ConfigDialog::updatePackListWidget()
{
	packListWidget->clear();
	for (int i= 0; i< mCardPackSet->getPackList().at(mPackIndex)->getFaces().count(); ++i)
	{
		QListWidgetItem *item = new QListWidgetItem();
		QPixmap pixmap(mCardPackSet->getPackList().at(mPackIndex)->getFacePixmap(i));
		item->setIcon(QIcon(pixmap));
		QString str = mCardPackSet->getPackList().at(mPackIndex)->getFaceText(i);
		item->setData(Qt::DisplayRole,str);
		packListWidget->insertItem(i,item);
	}
}
```

QListWidget



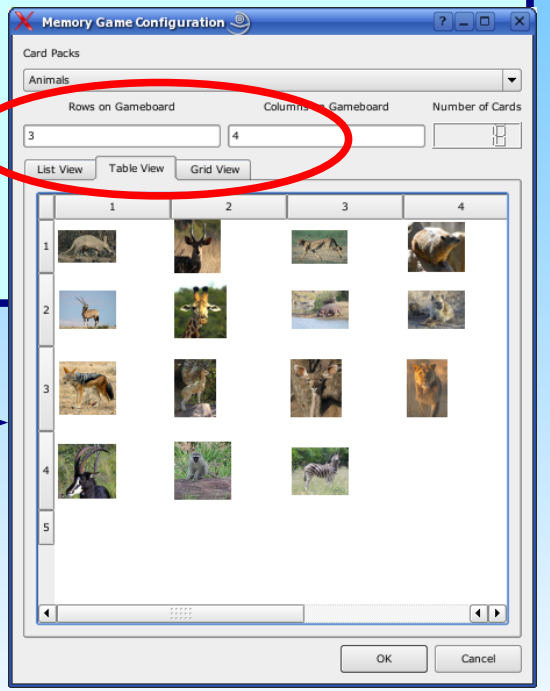
ConfigDialog: “Tábla nézet”

configdialog.cpp

```
void ConfigDialog::updatePackTableWidget()
{
        packTableWidget->clear();
        for (int i= 0; i< mCardPackSet->getPackList().at(mPackIndex)->getFaces().count(); ++i)
        {
                QTableWidgetItem *item = new QTableWidgetItem();
                int side = (int) sqrt(mCardPackSet->getPackList().at(mPackIndex)->getFaces().count()) + 1;
                packTableWidget->setRowCount(side);
                packTableWidget->setColumnCount(side);
                QPixmap pixmap(mCardPackSet->getPackList().at(mPackIndex)->getFacePixmap(i));
                item->setIcon(QIcon(pixmap));
                packTableWidget->setItem((int)(i/side), (int)(i%side),item);
        }
}
```

```
#include <math.h>
```

QTableWidget



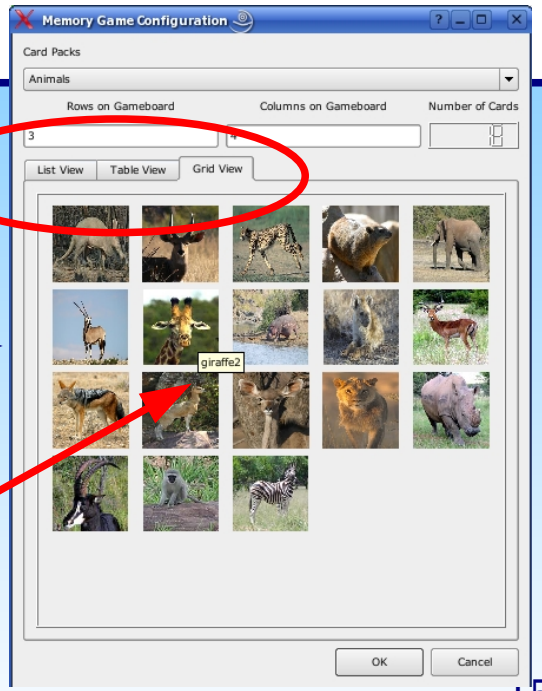
ConfigDialog: “Grid nézet”

configdialog.cpp

```
void ConfigDialog::updatePackCardGrid( ){
      int side = (int) sqrt(mCardPackSet->getPackList().at(mPackIndex)->getFaces().count()) + 1;
       cardButtonGroup->newCardButtonGroup(side,side);

      for (int i= 0; i< mCardPackSet->getPackList().at(mPackIndex)->getFaces().count(); ++i)
      {
            cardButtonGroup->updateCardButtonPixmap(i,
                  QPixmap(mCardPackSet->getPackList().at(mPackIndex)->getFacePixmap(i)));
            QString text = mCardPackSet->getPackList().at(mPackIndex)->getFaces().at(i)->getText();
            cardButtonGroup->getCardButton(i)->setToolTip(text.left(text.indexOf(".")));
      }
}
```

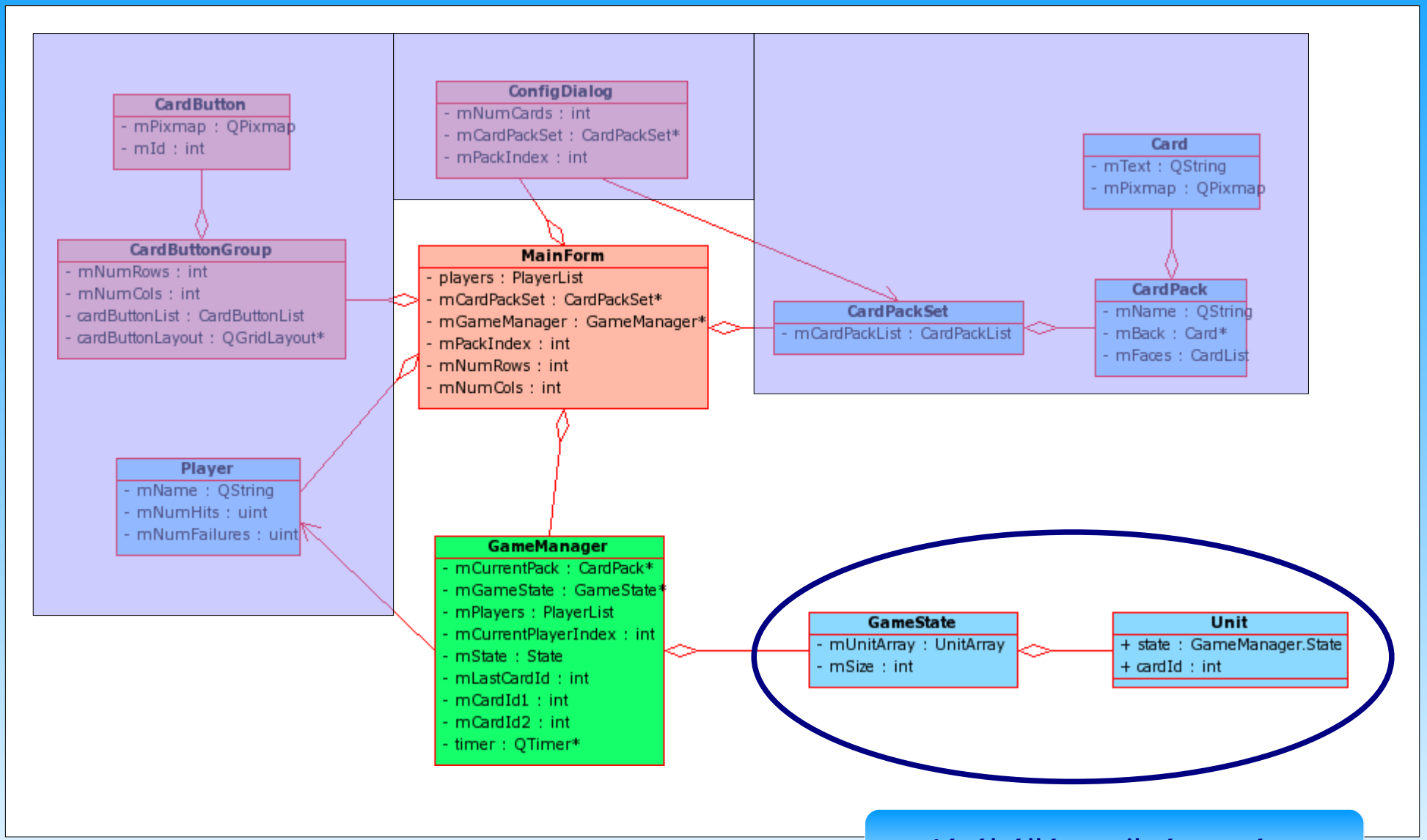
CardButtonGroup (saját vezérlő)

ToolTip



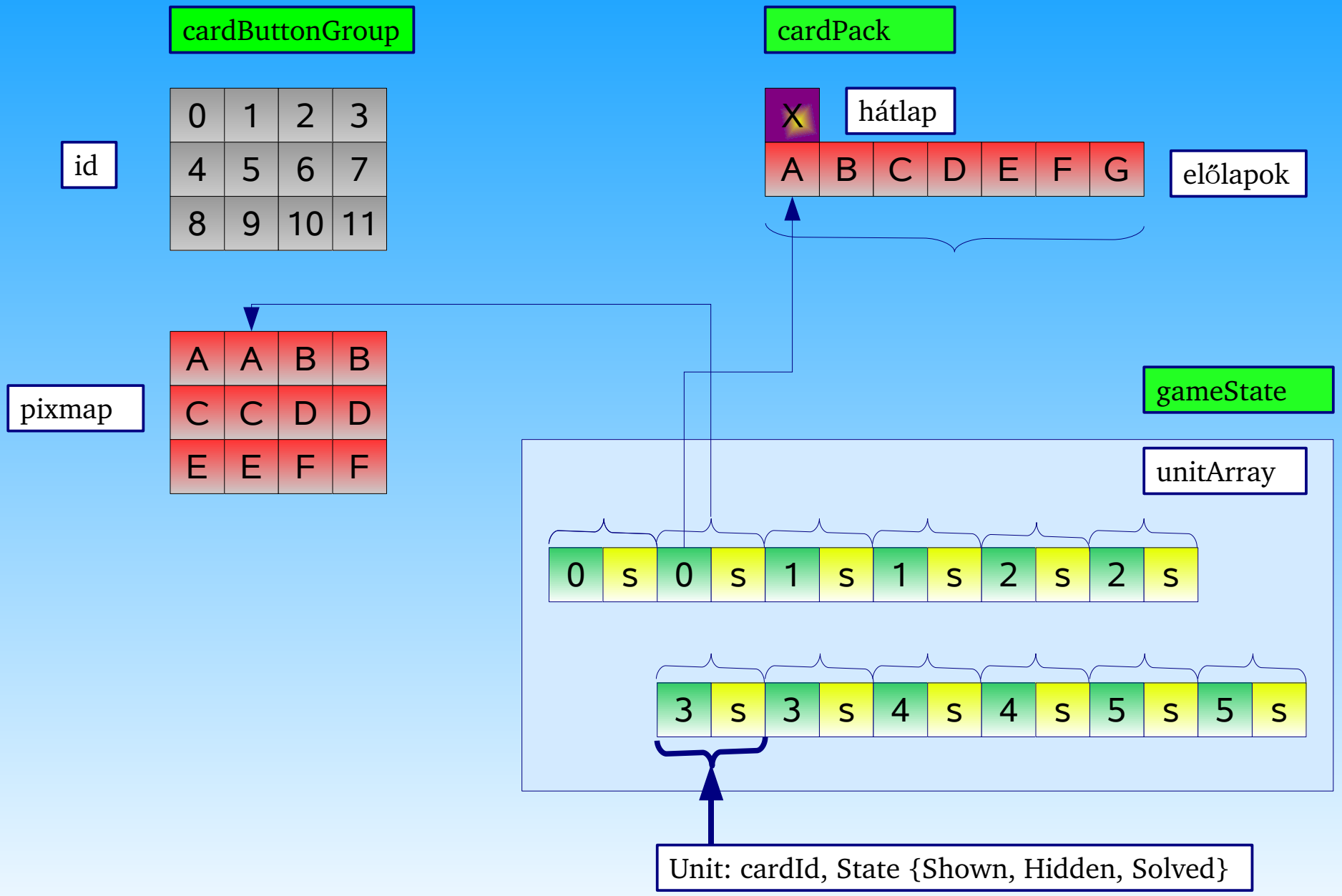
A játék

A memóriajáték osztálydiagramja

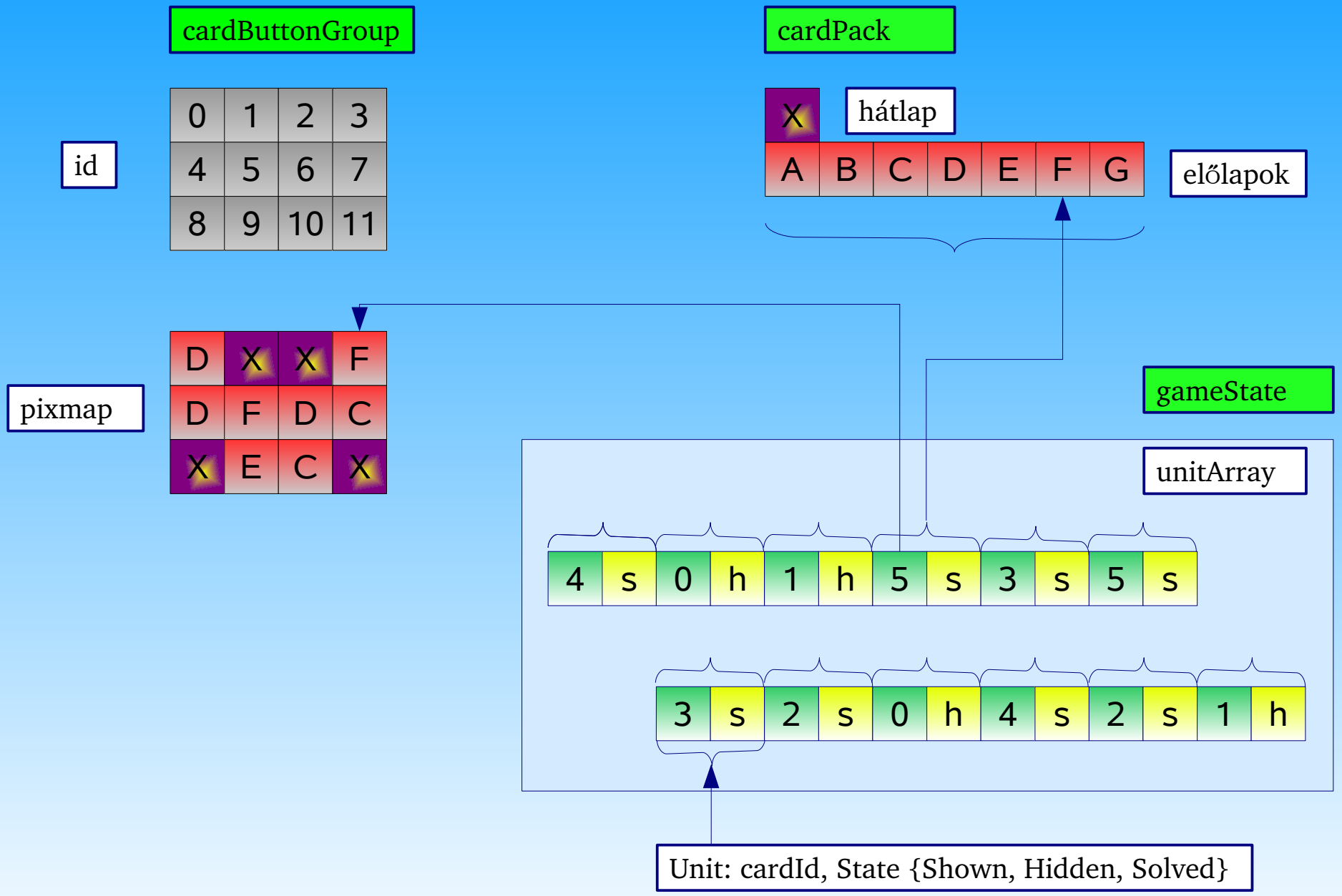


Játékállás nyilvántartása

Játékállás nyilvántartása (GameState) - kezdőállapot

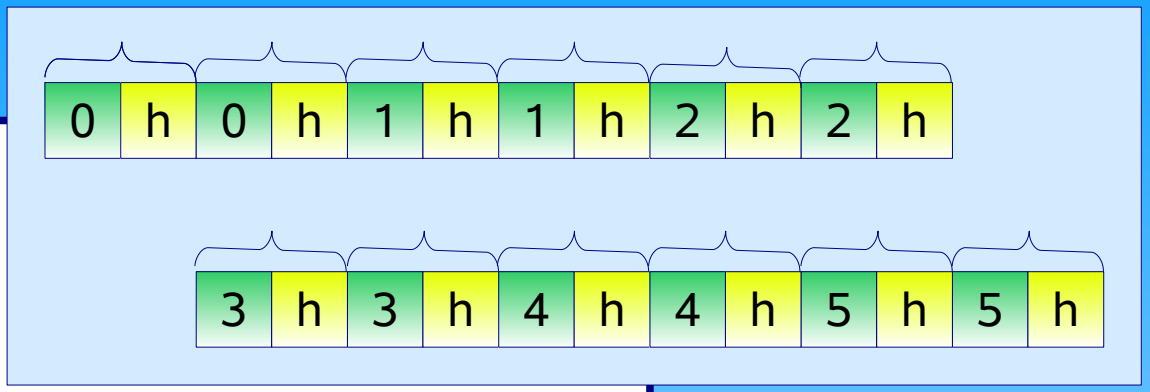


Játékállás nyilvántartása (GameState) – játékállapot



GameState: initUnitArray, shuffleUnits

gamestate.cpp




```
void GameState::initUnitArray()
{
	mUnitArray.resize(mSize);
	int cardId = 0;
	int i= 0;
	while(i < mUnitArray.size()-1)
	{
		mUnitArray[i].state = mUnitArray[i+1].state = Unit::Hidden;
		mUnitArray[i].cardId = mUnitArray[i+1].cardId = cardId;
		i += 2;
		++cardId;
	}
}
void GameState::shuffleUnits()
{
	for (int i= 0; i < mUnitArray.size() * mUnitArray.size(); ++i)
	{
		int ind1 = Utils::getNextRandomNumber(mUnitArray.size());
		int ind2 = Utils::getNextRandomNumber(mUnitArray.size());
		while(ind1 == ind2)
			ind2 = Utils::getNextRandomNumber(mUnitArray.size());
		qSwap(mUnitArray[ind1],mUnitArray[ind2] ); // <QtAlgorithms>
	}
}
```

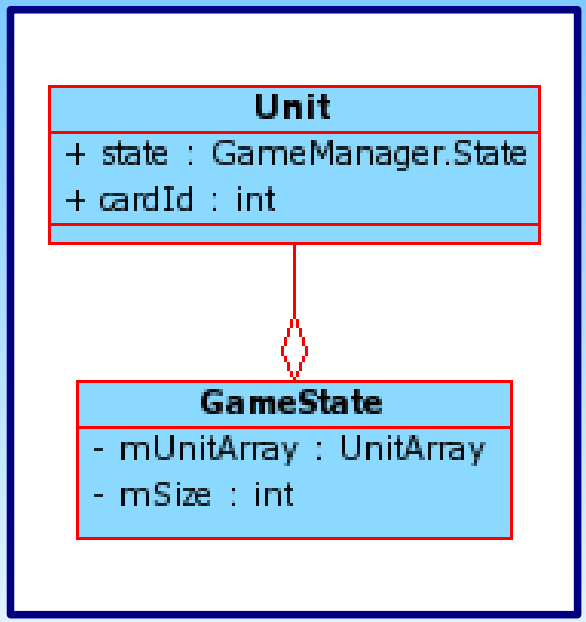
A játék kezdetén a kártyák hátlapját mutatjuk.

GameState: definíció

gamestate.h

```
#include <QVector>
class GameState {
public:

        GameState(int size);
        ~GameState();
        void newGameState();
        void newGameState(int size);
        Unit::State getState(int unitPos) const;
        void setState (int unitPos, Unit::State state);
        int getCardId(int unitPos) const;
        void setCardId (int unitPos, int cardId);
        int getSize()const {return mSize;}
        bool match(int unitPos1, int unitPos2);
        int numUnsolvedCards() const;
        QString toString();
private:
        void shuffleUnits();
        void initUnitArray();
private:
        UnitArray mUnitArray;
        int mSize;
};
```

```
struct Unit {
        enum State {Hidden, Shown, Solved};
        State state;
        int cardId;
};
typedef QVector<Unit> UnitArray;
```



GameState: konstruktor, newGameState

gamestate.cpp

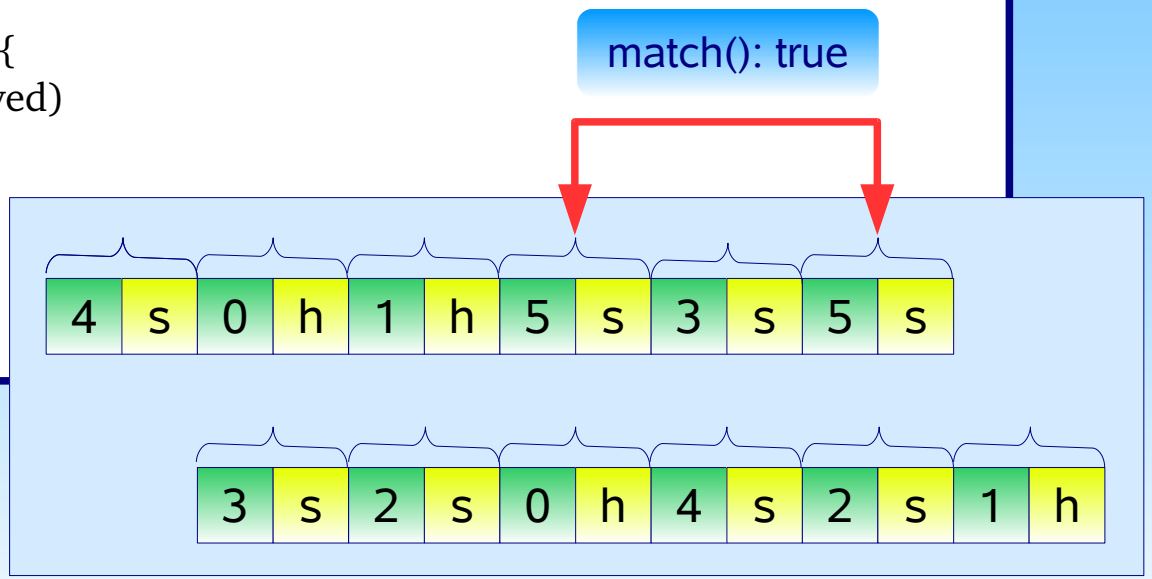
```
GameState::GameState(const int size)
        : mSize(size)
{
        Utils::initRandomGenerator();
        newGameState(mSize);
}

GameState::~GameState()
{}

void GameState::newGameState()
{
        newGameState(mSize);
}

void GameState::newGameState (int size)
{
        mSize = size;
        initUnitArray();
        shuffleUnits();
}
```

```
#include <QtAlgorithms>

#include "utils.h"
#include "gamestate.h"
```

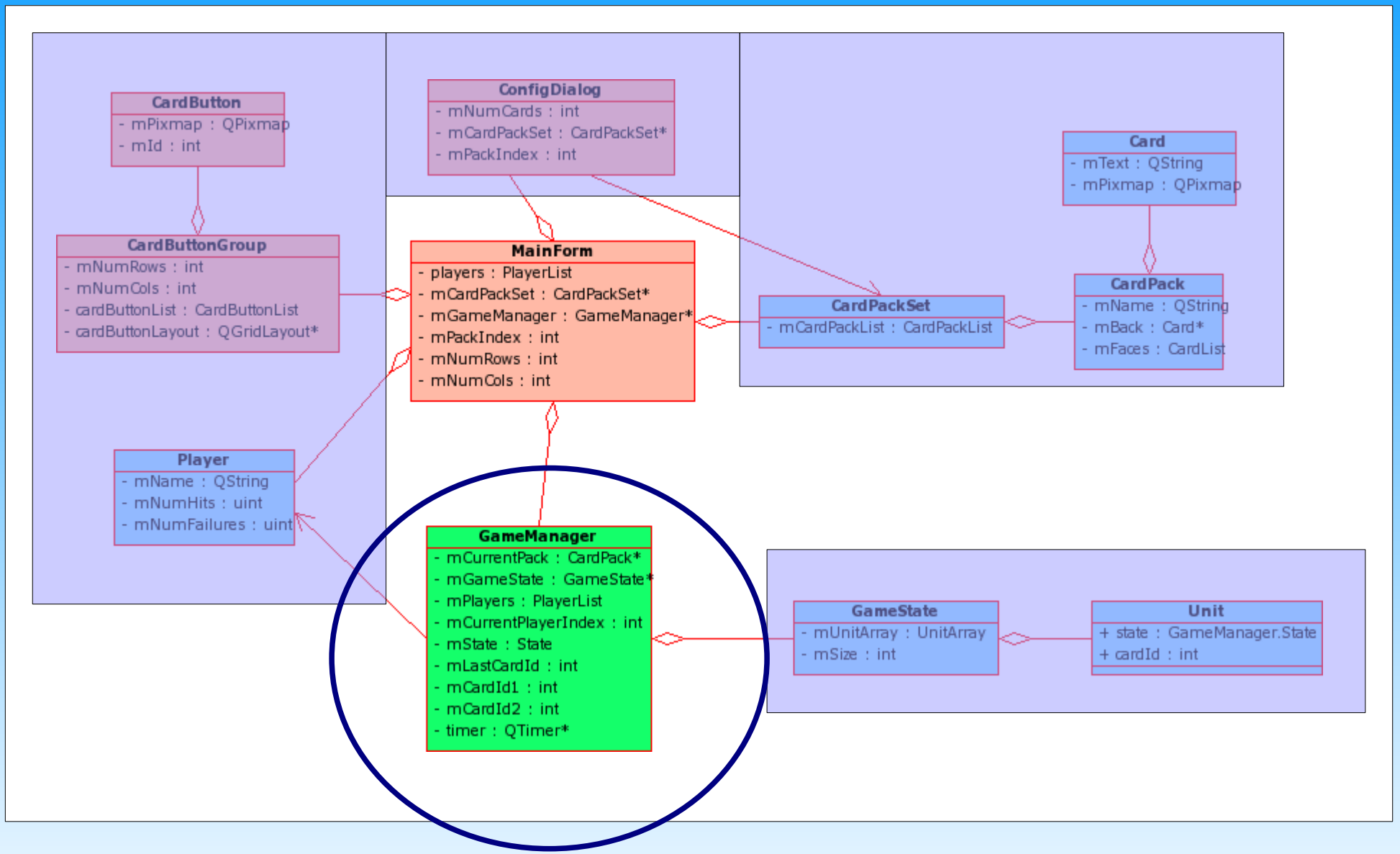
GameState: match, numUnsolvedCards

gamestate.cpp

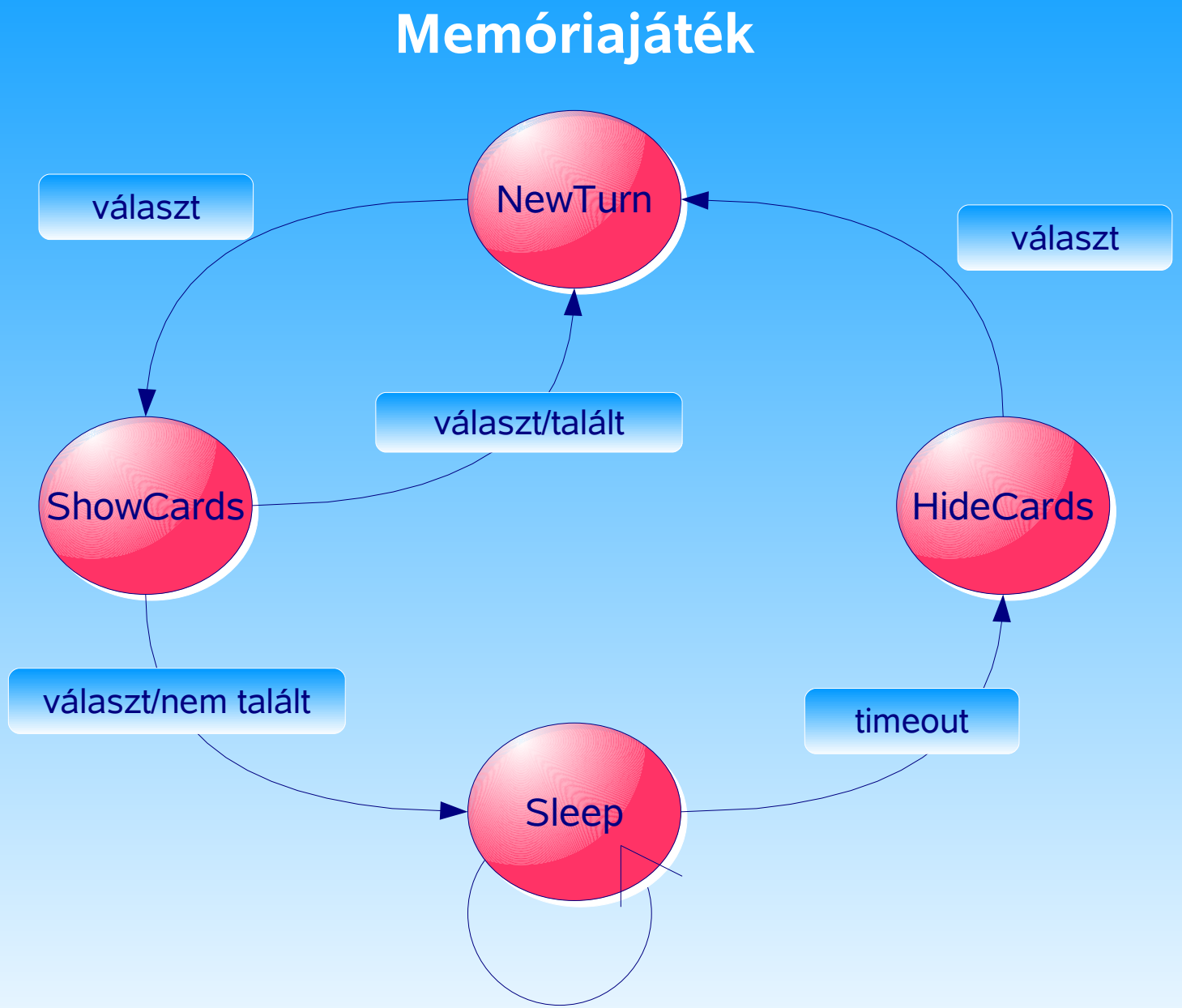
```
bool GameState::match(int unitPos1, int unitPos2)
{
	Q_ASSERT(unitPos1 < mSize);
	Q_ASSERT(unitPos2 < mSize);
	return (mUnitArray[unitPos1].cardId == mUnitArray[unitPos2].cardId) ? true :  false;
}


int GameState::numUnsolvedCards() const
{
	int unsolvedCards = 0;
	for (int i = 0; i < mSize; ++i) {
		if(getState(i) != Unit::Solved)
			++unsolvedCards;
	}
	return unsolvedCards/2;
}
```



GameManager: A játékot felügyelő osztály



A játékmenet állapotdiagramja



GameManager: Állapotátmenet (NewTurn, HideCards)

gamemanager.cpp

```
void GameManager::stateTransition()
{
        switch(mState) {

        case(NewTurn):
              . . .
        case (ShowCards):
              . . .
        case (HideCards):
              . . .
        }
}
```

```
case(NewTurn):
        mCardId1 = mLastCardId;
        mGameState->setState(mCardId1, GameState::Unit::Shown);
        updateCard(mCardId1);
        emit playerInfoChanged(getCurrentPlayer(),"Select second card!");
        mState = ShowCards;
        break;
```

```
case (HideCards):
        mGameState->setState(mCardId1, GameState::Unit::Hidden);
        mGameState->setState(mCardId2, GameState::Unit::Hidden);
        setCurrentPlayerIndex(nextPlayerIndex());
        updateCard(mCardId1);
        updateCard(mCardId2);
        emit playerInfoChanged(getCurrentPlayer(),"Select first card!");
        mState = NewTurn;
        break;
```

GameManager: Állapot átmenet (ShowCards)

gamemanager.cpp

```
void GameManager::stateTransition()
{
        switch(mState) {

        case(NewTurn):
                . . .
        case (ShowCards):
                . . .
        case (HideCards):
                . . .
        }
}
```

```
case (ShowCards):
        if (mCardId1 == mLastCardId) return;
        mCardId2 = mLastCardId;
        if(mGameState->match(mCardId1,mCardId2)){  //match
                mGameState->setState(mCardId1, GameState::Unit::Solved);
                mGameState->setState(mCardId2, GameState::Unit::Solved);
                updateCard(mCardId1);
                updateCard(mCardId2);
                getCurrentPlayer()->incrementHits();
                emit playerInfoChanged(getCurrentPlayer(),"Select first card!");
                mState = NewTurn;
                checkResult();
        }else {  //failure
                emit playerInfoChanged(getCurrentPlayer(),"Failure: No pairs.");
                mGameState->setState(mCardId2, GameState::Unit::Shown);
                updateCard(mCardId2);
                getCurrentPlayer()->incrementFailures();
                mState = Sleep;
                timer->start(1500);
        }
        break;
```

GameManager: definíció

gamemanager.cpp

```
#include <QObject>
#include <QPixmap>

class GameState;
class CardPack;
class QTimer;

#include "player.h"
```

```
class GameManager : public Qobject {
    Q_OBJECT

public:
    GameManager (int numRows, int numCols, PlayerList players);
    ~GameManager();

    void setCurrentPack(CardPack *pack) {mCurrentPack = pack;}
    PlayerList getPlayerList(){return mPlayers;}
    int getCurrentPlayerIndex(){return mCurrentPlayerIndex;}
    Player* getCurrentPlayer() {return mPlayers.at(mCurrentPlayerIndex);}
    void setCurrentPlayerIndex(int playerIndex);
    Ide jönnek a signal és slot definíciók (következő oldal)
private:
    void updateCard(int cardId);
    void checkResult();
    int nextPlayerIndex();
    QPixmap createSolvedPixmap(QPixmap pixmap);
    GameManager(const GameManager&);
    GameManager& operator=(const GameManager&);
};
```

```
private: //Data members
    CardPack* mCurrentPack;
    GameState* mGameState;
    PlayerList mPlayers;

    int mCurrentPlayerIndex;
    enum State {NewTurn, ShowCards,
                HideCards, Sleep};
    State mState;
    int mLastCardId;
    int mCardId1;
    int mCardId2;
    QTimer* timer;
```

gamemanager - gamemanager.h

GameManager: definíció (signalok és slotok)

gamemanager.cpp

```
public slots:
    void newGame(int numRows, int numCols);
    void handleCardButtonClick(int cardId);

protected slots:
    void stateTransition();
    void wakeUp();

signals:
    void cardButtonEnabledChanged(int cardId, bool enable = true);
    void cardButtonPixmapChanged(int cardId, QPixmap pixmap);
    void currentPlayerChanged(Player* player);
    void gameOver(PlayerList players);
    void playerInfoChanged(Player* player, QString info);
```

gamemanager - gamemanager.h

GameManager: konstruktor, newGame

gamemanager.cpp

```
GameManager::GameManager (int numRows, int numCols, PlayerList players)
        :mCurrentPack(NULL), mGameState(NULL), mPlayers(players), mCurrentPlayerIndex(0) {
        Q_ASSERT(players.count() >= 1);
        mGameState = new GameState(numRows*numCols);
        timer = new QTimer(this);
        timer->setSingleShot ( true );
        timer->stop();

         connect(timer, SIGNAL(timeout()), this, SLOT(wakeUp()));
}

GameManager::~GameManager() {
        delete mGameState;
}

void GameManager::newGame(int numRows, int numCols) {
        mState = NewTurn;
        if(mCurrentPack) mCurrentPack->shuffleCards();
        setCurrentPlayerIndex(0);
        emit playerInfoChanged(mPlayers.at(0),"Select first card!");

        mGameState->newGameState(numRows*numCols);
        for(int i = 0; i < mGameState->getSize(); ++i) updateCard(i);
}
```

```
#include <QPixmap>
#include <QTimer>
#include <QMessageBox>
#include <QPainter>

#include "utils.h"
#include "cardpack.h"
#include "gamestate.h"
#include "gamemanager.h"
```

GameManager: eseménykezelők

gamemanager.cpp

```
void GameManager::handleCardButtonClick(int cardId)
{
	if(mState == Sleep) return; //Ignore click in sleep state
	if(mGameState->getState(cardId) == GameState::Unit::Hidden) //Only hidden cards are valid
	{
		mLastCardId = cardId;
		stateTransition();
	}
}

void GameManager::wakeUp()
{
	mState = HideCards;
	stateTransition();
}
```

GameManager: updateCards

gamemanager.cpp

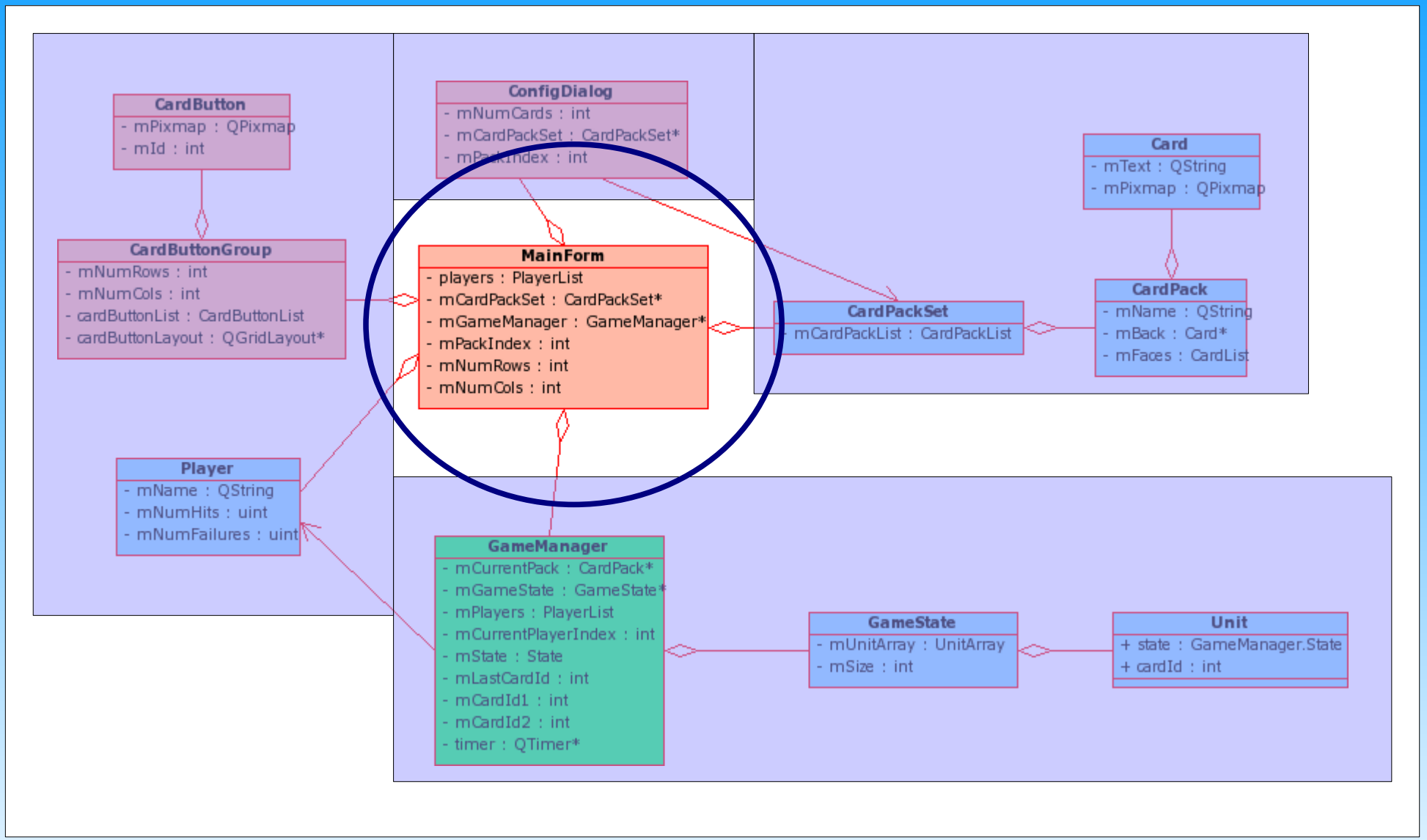
```
void GameManager::updateCard(int cardId) {
    Q_ASSERT(mCurrentPack != NULL);

    QPixmap pixmap;
    switch(mGameState->getState(cardId)) {

    case GameState::Unit::Shown:
        pixmap = mCurrentPack->getFacePixmap(mGameState->getCardId(cardId));
        emit cardButtonEnabledChanged(cardId, false);
        break;

    case GameState::Unit::Hidden:
        pixmap = mCurrentPack->getBack()->getPixmap();
        emit cardButtonEnabledChanged(cardId, true);
        break;

    case GameState::Unit::Solved:
        pixmap = createSolvedPixmap(
            mCurrentPack->getFacePixmap(mGameState->getCardId(cardId)));
        emit cardButtonEnabledChanged(cardId, false);
        break;
    }
    emit cardButtonPixmapChanged(cardId, pixmap);
}
```

GameManager: privát függvények

gamemanager.cpp

```
QPixmap GameManager::createSolvedPixmap(QPixmap pixmap) {
        QRect rect = pixmap.rect();
        QPainter painter(&pixmap);
        QPalette palette;
        QColor eraseColor = palette.color(QPalette::Background);
        painter.setBrush(eraseColor);
        painter.fillRect(rect, eraseColor);
        return pixmap;
 }
 void GameManager::checkResult() {
        if(mGameState->numUnsolvedCards() == 0)
                emit gameOver(mPlayers);
}
void GameManager::setCurrentPlayerIndex(int playerIndex){
        mCurrentPlayerIndex = playerIndex;
        emit currentPlayerChanged(getCurrentPlayer());
}
int GameManager::nextPlayerIndex(){
        ++mCurrentPlayerIndex;
        if(mCurrentPlayerIndex >= mPlayers.count())
                mCurrentPlayerIndex = 0;
        return mCurrentPlayerIndex ;
}
```

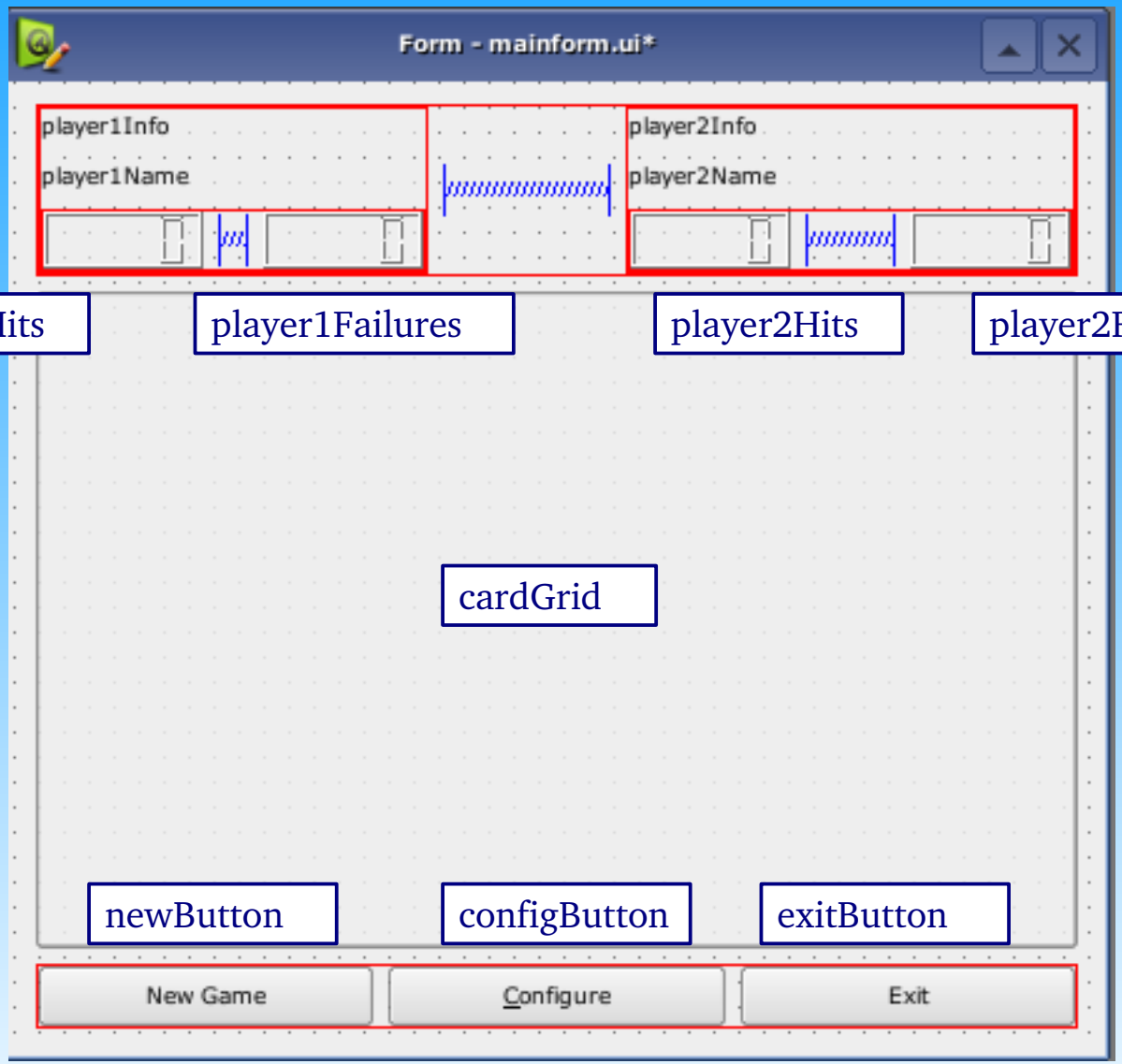
MainForm


CardButton
- mPixmap : QPixmap
- mId : int
ConfigDialog
- mNumCards : int
- mCardPackSet : CardPackSet*
- mPackIndex : int
Card
- mText : QString
- mPixmap : QPixmap
CardButtonGroup
- mNumRows : int
- mNumCols : int
- cardButtonList : CardButtonList
- cardButtonLayout : QGridLayout*
MainForm
- players : PlayerList
- mCardPackSet : CardPackSet*
- mGameManager : GameManager*
- mPackIndex : int
- mNumRows : int
- mNumCols : int
CardPackSet
- mCardPackList : CardPackList
CardPack
- mName : QString
- mBack : Card*
- mFaces : CardList
Player
- mName : QString
- mNumHits : uint
- mNumFailures : uint
GameManager
- mCurrentPack : CardPack*
- mGameState : GameState*
- mPlayers : PlayerList
- mCurrentPlayerIndex : int
- mState : State
- mLastCardId : int
- mCardId1 : int
- mCardId2 : int
- timer : QTimer*
GameState
- mUnitArray : UnitArray
- mSize : int
Unit
+ state : GameManager.State
+ cardId : int

MainForm

player1Info

player1Name



player2Info

player2Name

player1Hits

player1Failures

player2Hits

player2Failures

cardGrid

newButton

configButton

exitButton

MainForm - definíció


```
class MainForm : public QWidget, public Ui_MainForm{
    Q_OBJECT
public:
    MainForm(QWidget *parent = 0);
    ~MainForm();
public slots:
    void configure();
    void newGame();
    void updateCurrentPlayer(Player* player);
    void updatePlayerInfo(Player*player,QString info);
    void updateHits(Player* player);
    void updateFailures(Player* player);
    void handleGameOver(PlayerList players);
private:
    void initPlayers();
    MainForm(const MainForm&);
    MainForm& operator=(const MainForm&);
private:
    PlayerList players;
    CardPackSet* mCardPackSet;
    GameManager* mGameManager;
    int mPackIndex;
    int mNumRows;
    int mNumCols;
};
```

mainform.h

```
#include <QWidget>

#include "player.h"
#include "ui_mainform.h"

class CardPackSet;
class GameManager;
```

MainForm – konstruktor, destruktor

mainform.cpp

```
MainForm::MainForm(QWidget *parent)
    : QWidget(parent), mPackIndex(0), mNumRows(2), mNumCols(2)
{
    setupUi(this);  //Setup GUI elements created by designer

    initPlayers();

    mCardPackSet = new CardPackSet();
    mCardPackSet->loadFromAppDirectory();
    if(mCardPackSet->getPackList().empty()){
        QMessageBox::information(0,"pack name",
            "No \"packs\" directory for this application \nLoading default images.");
        mCardPackSet->loadFromAppResource();
  }
    mGameManager = new GameManager(mNumRows,mNumCols,players);

    Ide kell beírni a „connect” - eket. (következő dia)

    configure(); // Mus be given AFTER!! connects
}
MainForm::~MainForm() {
    delete mCardPackSet;
    delete mGameManager;
}
```

MainForm – signál-slot kapcsolatok

gamemanager - mainform.cpp

mainform.cpp

```
MainForm::MainForm(QWidget *parent)
    : QWidget(parent), mPackIndex(0), mNumRows(2), mNumCols(2)
{
    . . .

    connect(mGameManager, SIGNAL(playerInfoChanged(Player*,QString)),
            this,SLOT(updatePlayerInfo(Player*, QString)));
    connect(mGameManager, SIGNAL(gameOver(PlayerList)),
            this,SLOT(handleGameOver(PlayerList)));
    connect(mGameManager, SIGNAL(currentPlayerChanged(Player*)),
            this,SLOT(updateCurrentPlayer(Player*)));
    connect(mGameManager, SIGNAL(cardButtonPixmapChanged(int, QPixmap)),
            cardButtonGroup, SLOT(updateCardButtonPixmap(int, QPixmap)));
    connect(mGameManager, SIGNAL(cardButtonEnabledChanged(int, bool)),
            cardButtonGroup,SLOT(setCardButtonEnabled(int,bool)));
    connect(cardButtonGroup, SIGNAL(cardButtonClicked(int)),
            mGameManager, SLOT(handleCardButtonClick(int)));

    connect(exitButton, SIGNAL(clicked()), qApp, SLOT(quit()));
    connect(newButton, SIGNAL(clicked()), this, SLOT(newGame()));
    connect(configButton, SIGNAL(clicked()), this, SLOT(configure()));

    . . .
}
```

A felület aktualizálása a „szereplők” jelzései alapján.

MainForm – Új játék indítása, konfigurálás, játék vége

mainform.cpp

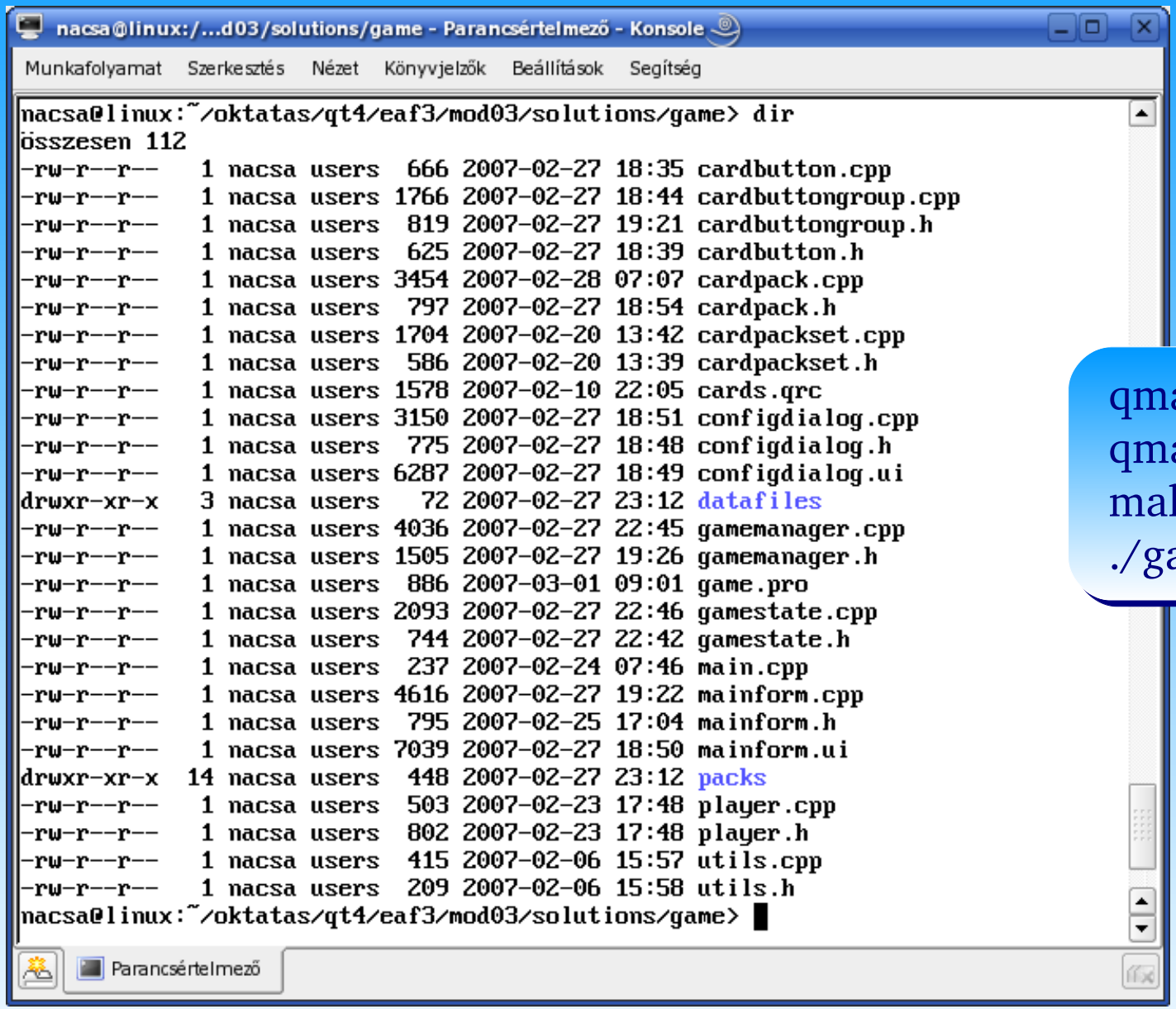
```
void MainForm::newGame() {
	cardButtonGroup->newCardButtonGroup(mNumRows,mNumCols);
	mGameManager->setCurrentPack(mCardPackSet->getByIndex(mPackIndex));
	mGameManager->newGame(mNumRows,mNumCols);
	newButton->setEnabled(false);
	configButton->setEnabled(false);
	exitButton->setEnabled(false);
}
void MainForm::configure() {
	ConfigDialog dialog(mCardPackSet, mPackIndex, mNumRows, mNumCols, this);
	if(dialog.exec())
	{
		mNumRows = dialog.rowEdit->text().toInt();
		mNumCols = dialog.colEdit->text().toInt();
		mPackIndex = dialog.packComboBox->currentIndex();
	}
	newGame();
}
void MainForm::handleGameOver(PlayerList players){
	player1Info->setText("");
	player2Info->setText("");
	player1Name->setText("<font color=black>" +  players.at(0)->getName() + "</font>");
	player2Name->setText("<font color=black>" +  players.at(1)->getName() + "</font>");
	QMessageBox::information(0, "Memory Game", "Game Over");
	newButton->setEnabled(true);
	configButton->setEnabled(true);
	exitButton->setEnabled(true);
}
```

game.pro – projekt leíró fájl

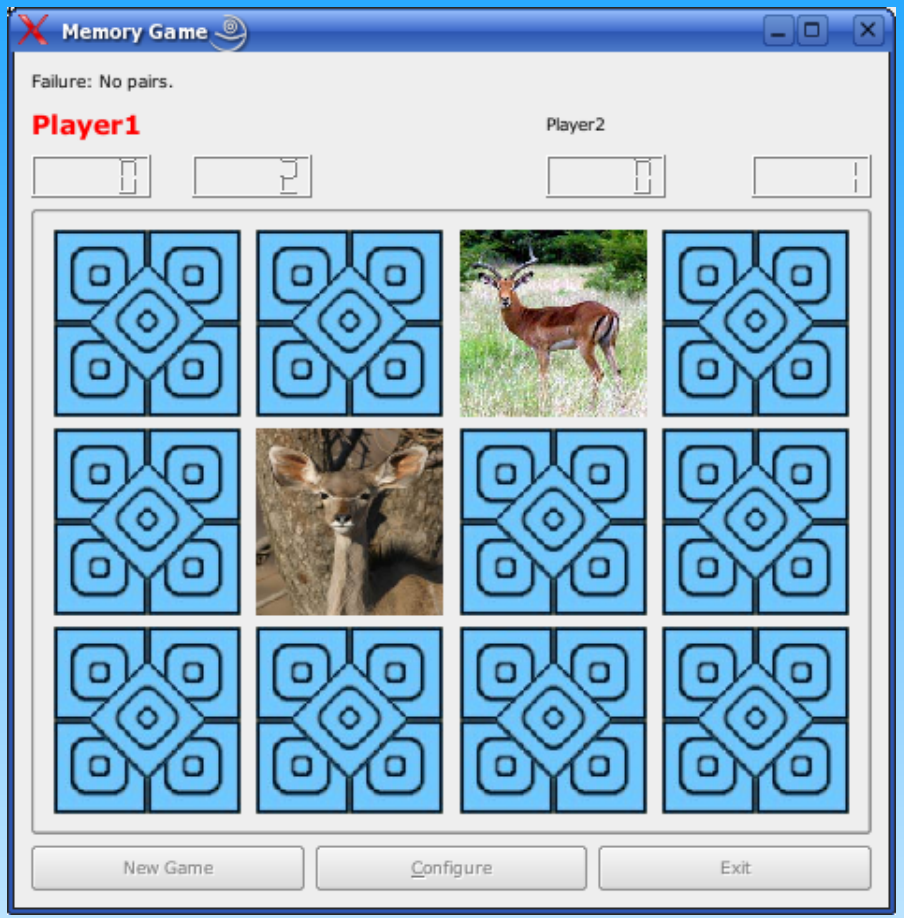
game.pro

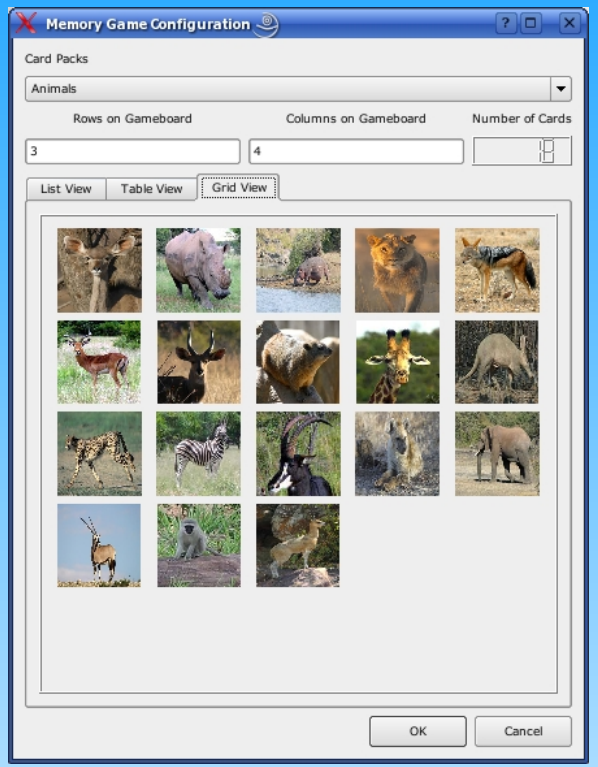
```
TEMPLATE = app
TARGET =
DEPENDPATH += .
INCLUDEPATH += .
# Input
HEADERS += cardbutton.h \
           cardbuttongroup.h \
           cardpack.h \
           cardpackset.h \
           configdialog.h \
           gamemanager.h \
           gamestate.h \
           mainform.h \
           player.h \
           utils.h
FORMS += configdialog.ui mainform.ui
SOURCES += cardbutton.cpp \
           cardbuttongroup.cpp \
           cardpack.cpp \
           cardpackset.cpp \
           configdialog.cpp \
           gamemanager.cpp \
           gamestate.cpp \
           main.cpp \
           mainform.cpp \
           player.cpp \
           utils.cpp
RESOURCES += cards.qrc
```

Fordítás, futtatás



qmake -project
qmake game.pro
make
./game

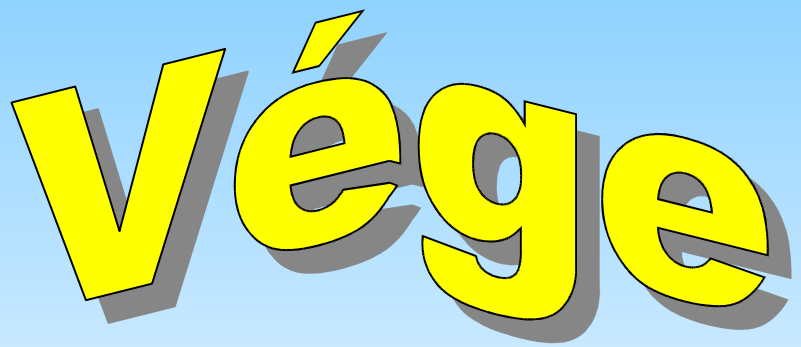

Memory Game
Failure: No pairs.
Player1
Player2
0
2
0
1
New Game
Configure
Exit


Memory Game Configuration
Card Packs
Animals
Rows on Gameboard
Columns on Gameboard
Number of Cards
3
4
18
List View
Table View
Grid View
OK
Cancel

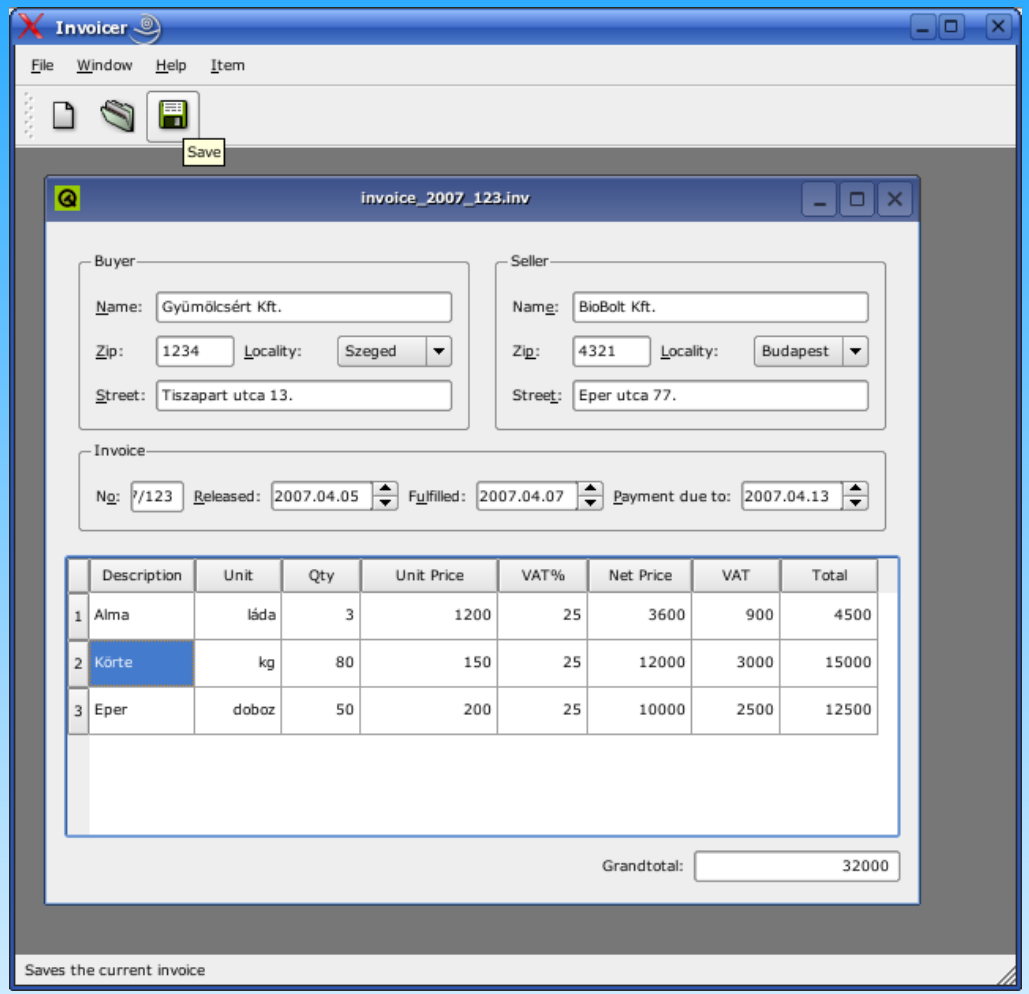
A bemutatott osztályok, az azokat tesztelő projektek, valamint a memóriajáték programjai megtalálhatók a

people.inf.elte.nacsa/qt4/eaf3/mod03/projects/

címen.



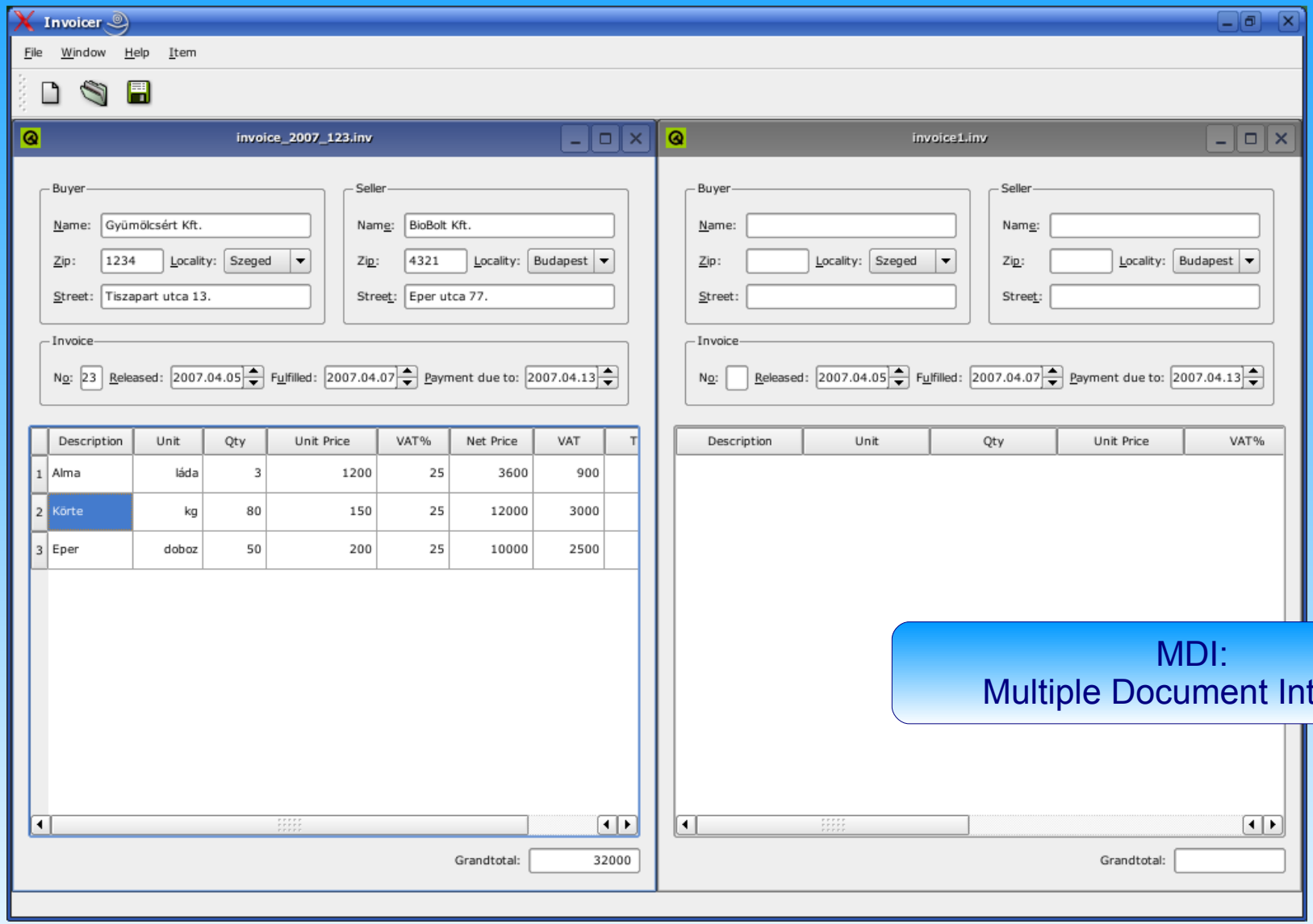
Elemi alkalmazások fejlesztése III.



MDI alkalmazás I.
(Számla)

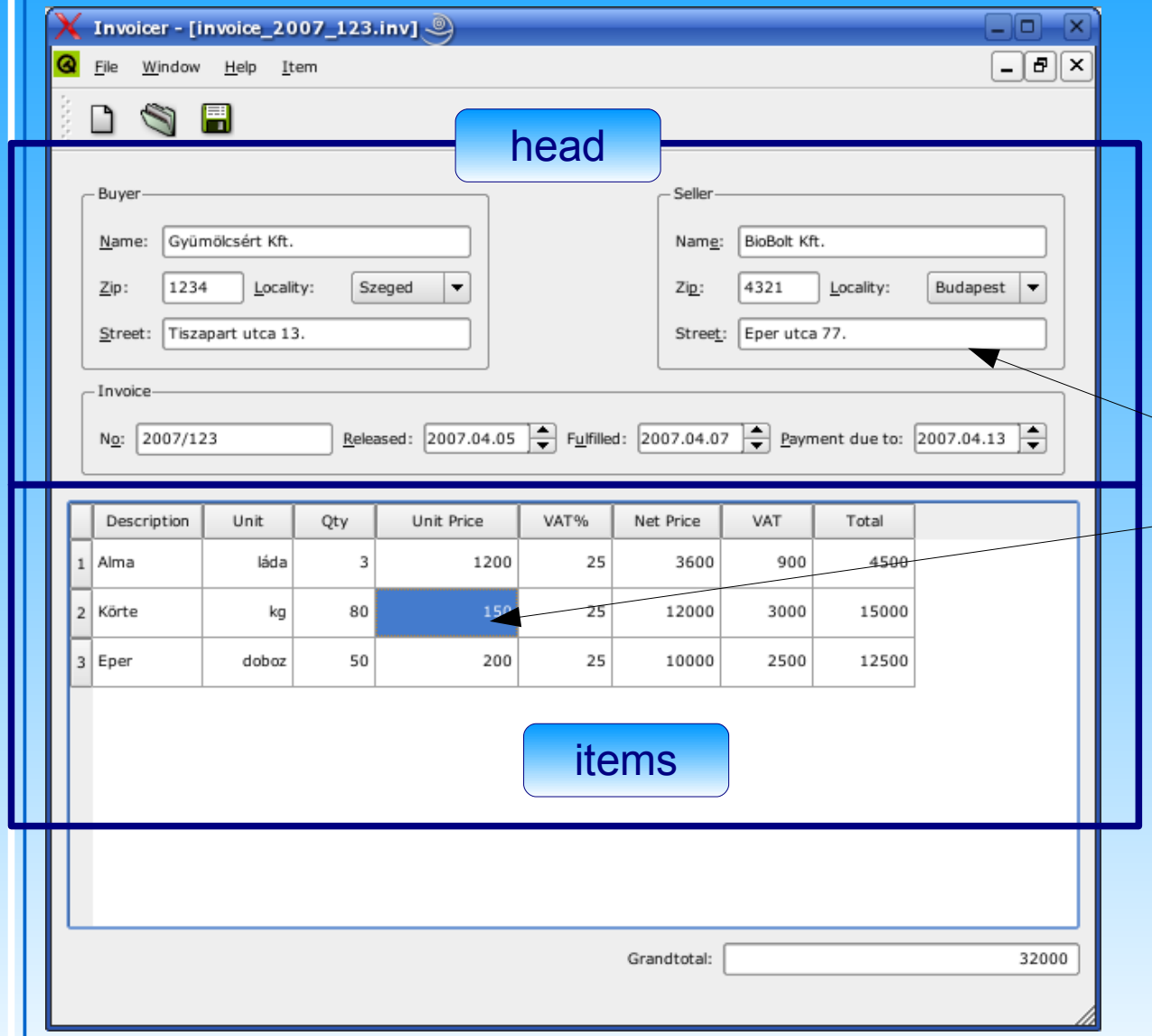
Készítette: Szabóné Nacsa Rozália
nacsa@inf.elte.hu
people.inf.elte.hu/nacsa/qt4/eaf3/
Qt 4
2007

Több dokumentum kezelése (MDI)



MDI:
Multiple Document Interface

Számla adatok tárolása

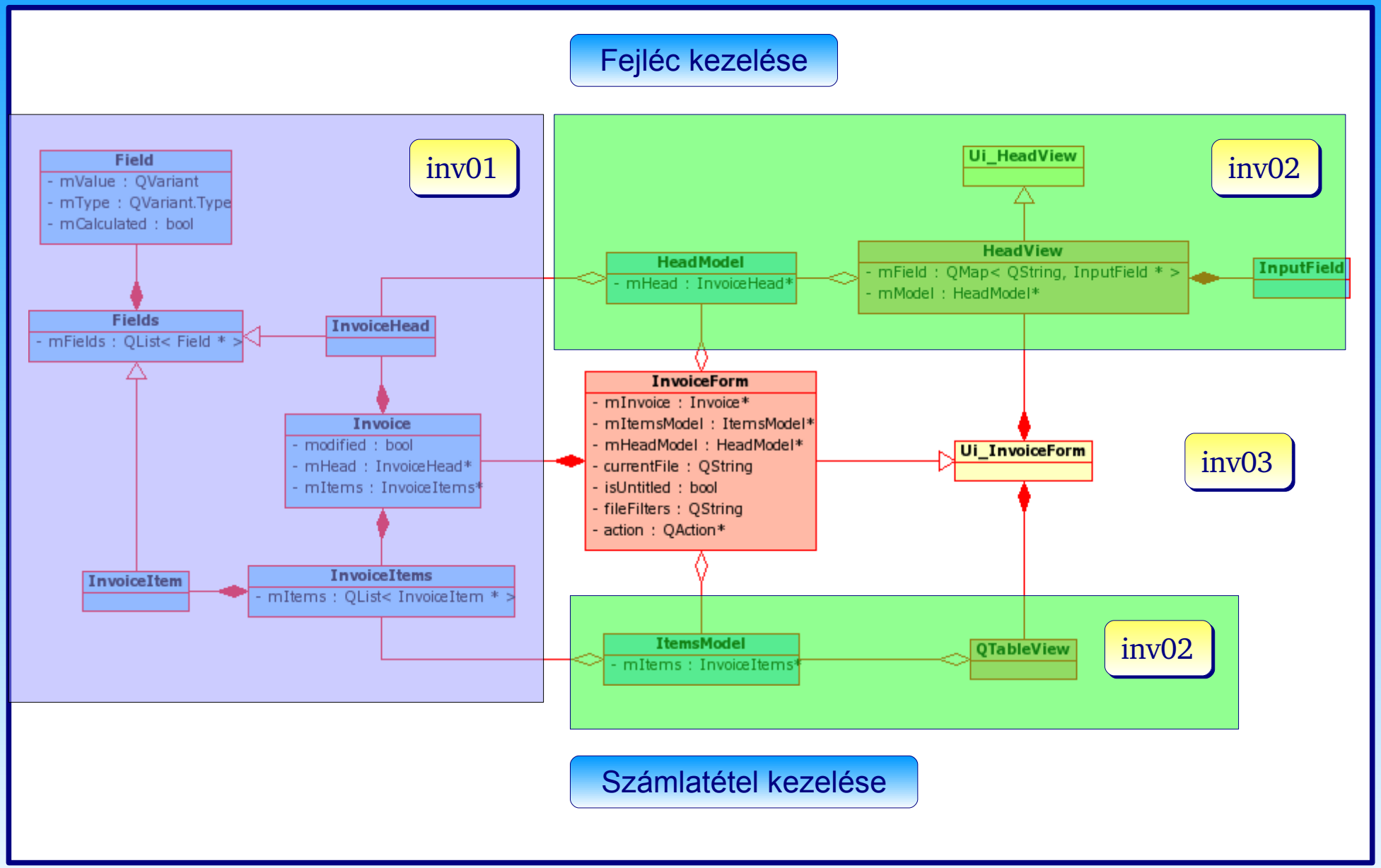


field

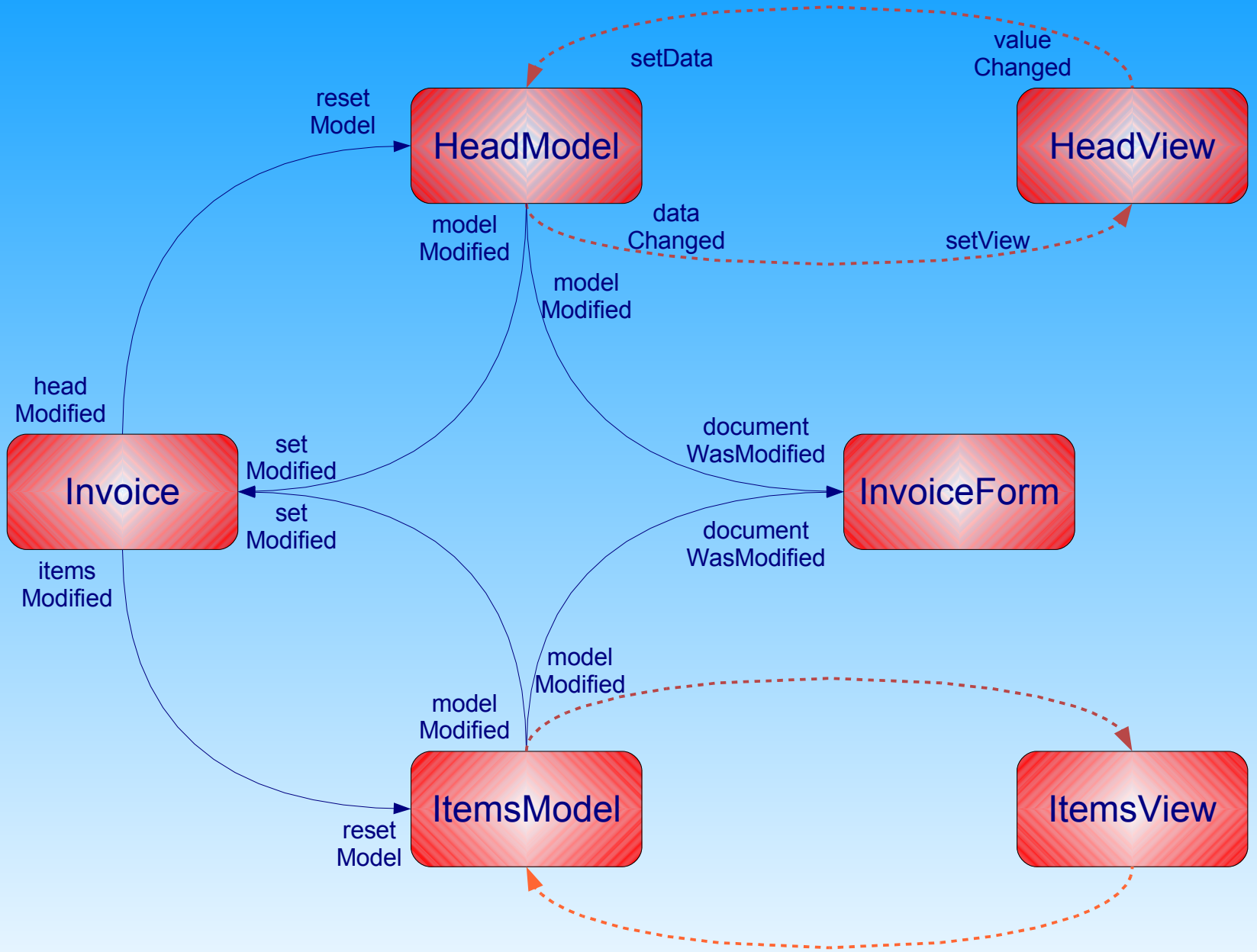
inv_2007_123.inv

```
12
Gyümölcsért Kft.
1234
Szeged
Tiszapart utca 13.
BioBolt Kft.
4321
Budapest
Eper utca 77.
2007/123
2007-04-05
2007-04-07
2007-04-13
3
Alma
láda
3
1200
25
Körte
kg
80
150
25
Eper
doboz
50
200
25
```

Az alkalmazás osztálydiagramja



Objektumok együttműködése



Az invoicer01 modul tartalma

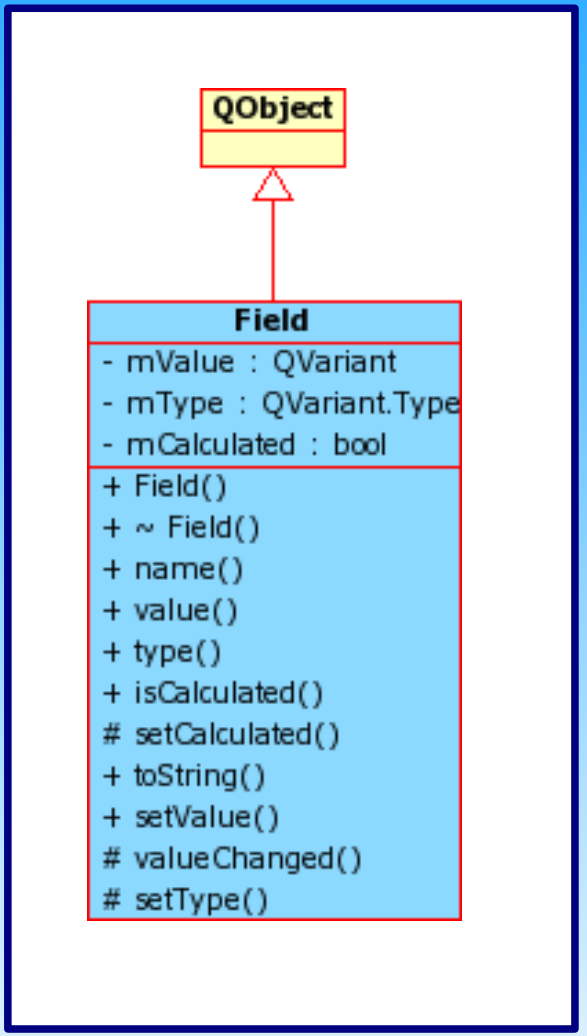
Adat kezelő osztályok elkészítése

- Adatmező osztály elkészítése (Field osztály)
- Adatmezők osztály elkészítése (Fields osztály)
- Számla tétel osztály elkészítése (InvoiceItem)
- Számla tételek osztály elkészítése (InvoiceItems)
- Számla fejléc osztály elkészítése (InvoiceHead)
- Számla osztály elkészítése (Invoice)
- Osztályok tesztelése (invoice projekt)

Field: definíció

field.h

```
class Field : public QObject {
    Q_OBJECT
public:
    Field ( const QString& name,
            QVariant value,
            QVariant::Type type = QVariant::String,
            bool calculated = false );
    virtual  ~Field() {}
    virtual  QString name() {return objectName();}
    virtual  QVariant value() {return mValue;}
    virtual  QVariant::Type type() {return mType;}
    bool     isCalculated() {return mCalculated;}

    QString toString() const;

public slots:
    virtual  bool setValue(QVariant newValue);

signals:
    void     valueChanged();

protected:
    void     setType(QVariant::Type type) {mType = type;}
    void     setCalculated(bool calculated = true)
                 {mCalculated = calculated;}

private:
    QVariant        mValue;
    QVariant::Type  mType;
    bool            mCalculated;

};
```



Field: implementáció

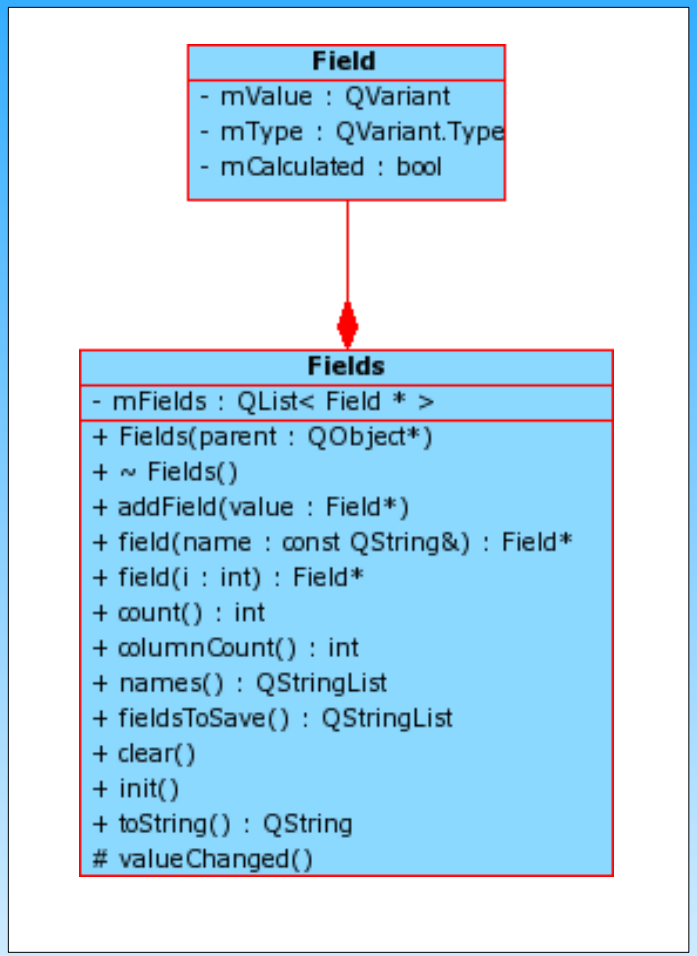
field.cpp

```
#include "field.h"

Field::Field(const QString& name, QVariant value, QVariant::Type type, bool calculated)
            : mValue(value), mType(type), mCalculated(calculated)
{
      setObjectName(name);
}

bool Field::setValue(QVariant newValue) {
   if(mValue==newValue) return true;
      mValue = newValue;
   emit valueChanged();
   return true;
}

QString Field::toString() const {
   return QString("%1(%2) = %3").
      arg(objectName()).arg(mType).arg(mValue.toString());
}
```

Fields: definíció

fields.h

```
class Fields : public QObject
{
        Q_OBJECT

public:
        Fields(QObject* parent = 0);
        ~Fields();
        void addField( Field* value);
        Field* field(const QString& name);
        Field* field(int i);
        int count() {return mFields.count();}
        int columnCount(){return count();}
        QStringList names();
        QStringList fieldsToSave();
        void clear();
        void init();
        QString toString();

signals:
        void valueChanged();

private:
        QList <Field*> mFields;

};
```



Fields: implementáció

fields.cpp

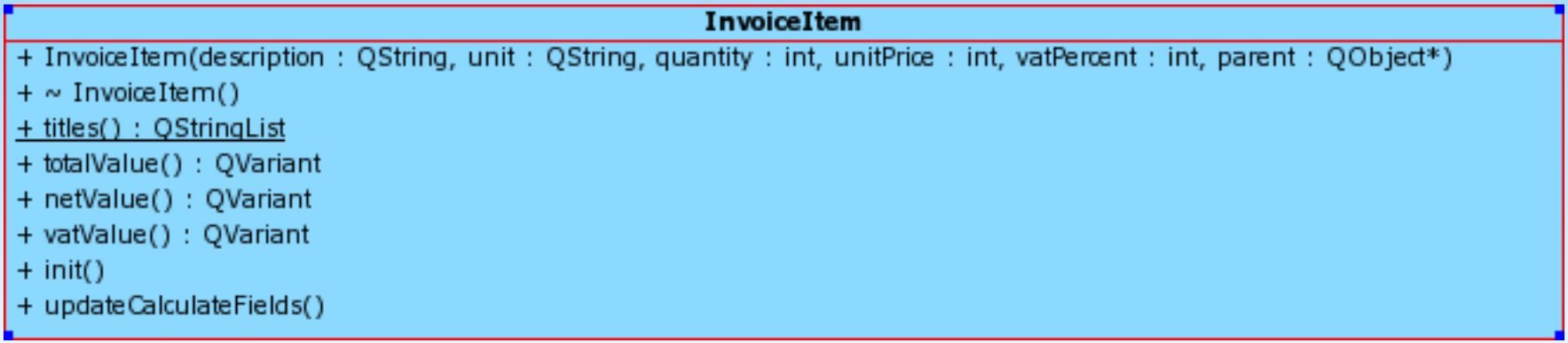
```
Fields::Fields(QObject* parent){
	setParent(parent);
}

Fields::~Fields(){
	clear();
}

void Fields::clear(){
	while (!mFields.isEmpty())
		delete mFields.takeFirst();
}

void Fields::init(){
	for (int i = 0; i < count(); ++i)
		field(i)->setValue(QVariant());
}

Field* Fields::field(const QString& name){
	return mFields.at(names().indexOf(name));
}

Field* Fields::field(int i){
	return mFields.at(i);
}
```

```
void Fields::addField(Field* field){
	mFields.append(field);
}

QStringList Fields::names(){
	QStringList ret;
	for (int i = 0; i < count(); ++i) {
		ret << field(i)->name();
	}
	return ret;
}

QStringList Fields::fieldsToSave(){
	QStringList ret;
	for (int i = 0; i < count(); ++i) {
		if (!field(i)->isCalculated())
			ret << field(i)->value().toString();
	}
	return ret;
}
```

InvoiceItem: definíció

invoiceitem.h

```
class InvoiceItem : public Fields {
    Q_OBJECT
public:
    InvoiceItem(QString description=QString::null, QString unit=QString::null,
        int quantity=0,int unitPrice=0, int vatPercent=0, QObject* parent = 0);
    virtual ~InvoiceItem();

    static QStringList titles();

    QVariant totalValue();
    QVariant netValue();
    QVariant vatValue();
    void init();

public slots:
    void updateCalculateFields();
};
```



InvoiceItem: implementáció

invoiceitem.cpp

```
InvoiceItem::InvoiceItem(QString description, QString unit,
            int quantity,int unitPrice, int vatPercent,QObject* parent)
      : Fields(parent)
{
      addField(new Field("description",description,QVariant::String));
      addField(new Field("unit",unit,QVariant::String));
      addField(new Field("quantity",quantity,QVariant::Int));
      addField(new Field("unitprice",unitPrice,QVariant::Int));
      addField(new Field("vatpercent",vatPercent,QVariant::Int));

      //Calculated fields
      addField(new Field("netvalue",netValue(),QVariant::Int,true));
      addField(new Field("vatvalue",vatValue(),QVariant::Int,true));
      addField(new Field("totalvalue",totalValue(),QVariant::Int,true));

      connect(field("quantity"),SIGNAL(valueChanged()),
                  this,SLOT(updateCalculateFields()));
      connect(field("unitprice"),SIGNAL(valueChanged()),
                  this,SLOT(updateCalculateFields()));
      connect(field("vatpercent"),SIGNAL(valueChanged()),
                  this,SLOT(updateCalculateFields()));
}
```

InvoiceItem: implementáció

invoiceitem.cpp

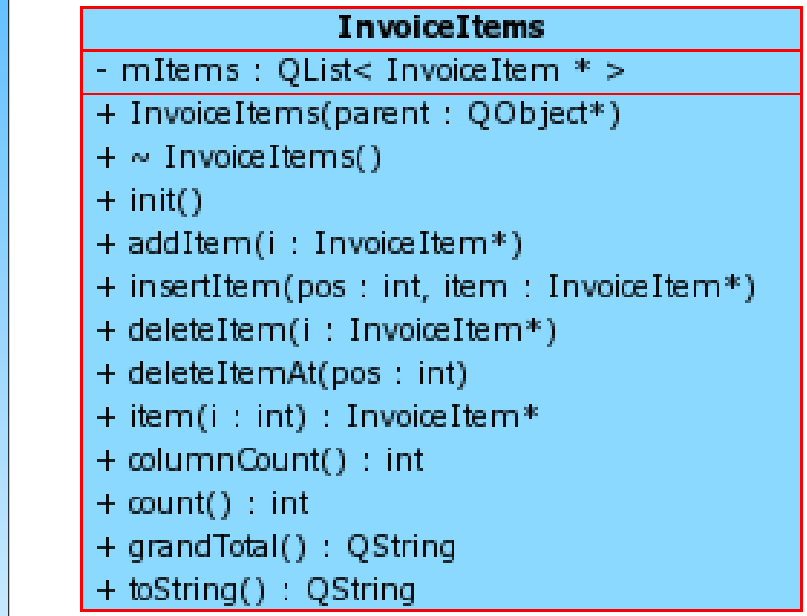
```
void InvoiceItem::init()
{
        field("description")->setValue("");
        field("unit")->setValue("db");
        field("quantity")->setValue(1);
        field("unitprice")->setValue(0);
        field("vatpercent")->setValue(20);
}

QStringList InvoiceItem::titles()
{
        QStringList titles;
        titles << QObject::trUtf8("Description")
          << QObject::trUtf8("Unit")
          << QObject::trUtf8("Qty")
          << QObject::trUtf8("Unit Price")
          << QObject::trUtf8("VAT%")
          << QObject::trUtf8("Net Price")
          << QObject::trUtf8("VAT")
          << QObject::trUtf8("Total");
        return titles;

}
```

```
void InvoiceItem::updateCalculateFields(){
        field("netvalue")->setValue(netValue());
        field("vatvalue")->setValue(vatValue());
        field("totalvalue")->setValue(totalValue());
}
QVariant InvoiceItem::totalValue()
{
        return field("quantity")->value().toInt()
                * field("unitprice")->value().toInt()
                * (100 +  field("vatpercent")->value().toInt())/100;
}
QVariant InvoiceItem::netValue()
{
        return field("quantity")->value().toInt()
                * field("unitprice")->value().toInt() ;
}
QVariant InvoiceItem::vatValue()
{
        return field("quantity")->value().toInt()
                * field("unitprice")->value().toInt()
                * field("vatpercent")->value().toInt()/100 ;
}
```

InvoiceItems: definíció

invoiceitems.h

```
class InvoiceItems : public QObject {
public:
    InvoiceItems(QObject* parent = 0);
    ~InvoiceItems();

    void init();

    void addItem(InvoiceItem *i);
    void insertItem(int pos, InvoiceItem *item);
    void deleteItem(InvoiceItem *i);
    void deleteItemAt(int pos);

    InvoiceItem* item(int i) {return mItems.at(i);}

    int columnCount();
    int count() {return mItems.count();}

    QString grandTotal();
    QString toString();

private:
        QList<InvoiceItem*> mItems;

};
```



InvoiceItems: implementáció

invoiceitems.cpp

```
InvoiceItems::InvoiceItems(QObject* parent){
        setParent(parent);
        setObjectName(QString::fromUtf8("InvoiceItems"));
}
InvoiceItems::~InvoiceItems(){
        init();
}
void InvoiceItems::init(){
        while (!mItems.isEmpty())
                delete mItems.takeFirst();
}
void InvoiceItems::addItem(InvoiceItem *item){
        mItems.append(item);
}
void InvoiceItems::insertItem(int pos, InvoiceItem *item){
        mItems.insert(pos,item);
}
void InvoiceItems::deleteItem(InvoiceItem *i){
        deleteItemAt(mItems.indexOf(i));
}
void InvoiceItems::deleteItemAt(int pos){
        if (0 <= pos < mItems.size())
                delete mItems.takeAt(pos);
}
```

InvoiceItems: implementáció

invoiceitems.cpp

```
QString InvoiceItems::toString(){
        QString ret = "\nInvoice: Item List";
        for (int i=0; i< count(); i++)
                ret += mItems.at(i)->toString();
        return ret;
}

int InvoiceItems::columnCount(){
        return mItems.count() > 0 ? mItems.at(0)->count(): 0;
}

QString InvoiceItems::grandTotal() {
        int total=0;
        for(int i = 0; i < count(); ++i)
                total += mItems.at(i)->totalValue().toInt();
        return QString::number(total);
}
```

InvoiceHead: definíció

invoicehead.h

```
#include <QObject>
#include <QVariant>
#include <QList>

#include "field.h"
#include "fields.h"

class InvoiceHead : public Fields
{
        Q_OBJECT

public:
        InvoiceHead(QObject* parent = 0);
        ~InvoiceHead();
        void init();
};
```

InvoiceHead: implementáció

invoicehead.cpp

```
InvoiceHead::InvoiceHead(QObject* parent)
        : Fields(parent)
{
        addField(new Field("buyer_name",QVariant(QString()),QVariant::String));
        addField(new Field("buyer_zip",QVariant(QString()),QVariant::String));
        addField(new Field("buyer_city",QVariant(QString()),QVariant::String));
        addField(new Field("buyer_street",QVariant(QString()),QVariant::String));

        addField(new Field("seller_name",QVariant(QString()),QVariant::String));
        addField(new Field("seller_zip",QVariant(QString()),QVariant::String));
        addField(new Field("seller_city",QVariant(QString()),QVariant::String));
        addField(new Field("seller_street",QVariant(QString()),QVariant::String));

        addField(new Field("invNo",QVariant(QString()),QVariant::String));
        addField(new Field("released",QVariant(QDate()),QVariant::Date));
        addField(new Field("fulfilled",QVariant(QDate()),QVariant::Date));
        addField(new Field("dueTo",QVariant(QDate()),QVariant::Date));
}
```

InvoiceHead: implementáció

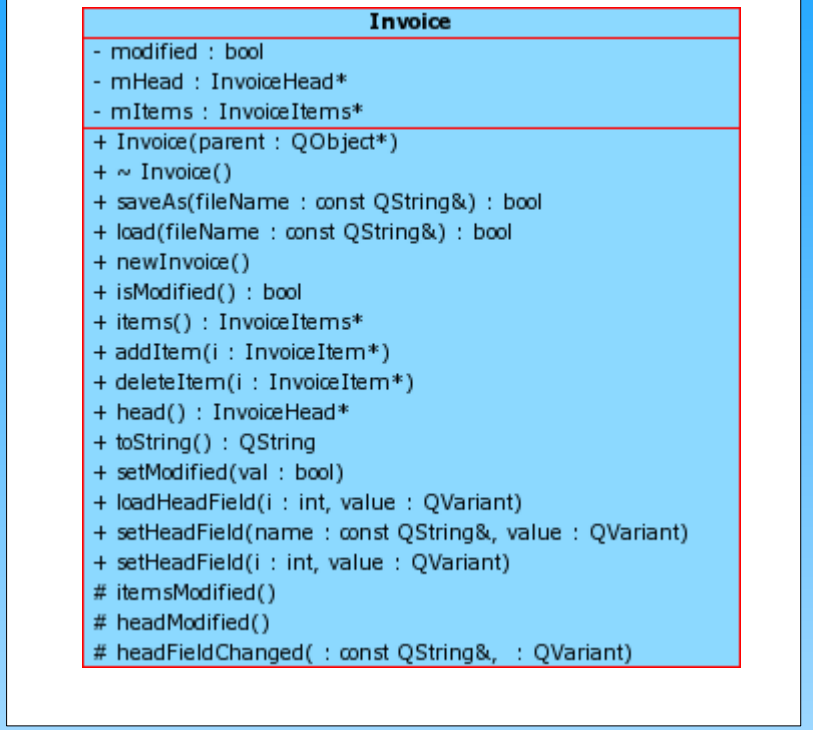
invoicehead.cpp

```
void InvoiceHead::init()
{
	field("buyer_name")->setValue("");
	field("buyer_zip")->setValue("");
	field("buyer_city")->setValue("Szeged");
	field("buyer_street")->setValue("");

	field("seller_name")->setValue("");
	field("seller_zip")->setValue("");
	field("seller_city")->setValue("Budapest");
	field("seller_street")->setValue("");

	field("invNo")->setValue("");
	field("released")->setValue(QDate::currentDate());
	field("fulfilled")->setValue(QDate::currentDate().addDays(2));
	field("dueTo")->setValue(QDate::currentDate().addDays(8));

}
```

Invoice: definíció

invoice.h

```
class Invoice : public QObject {
        Q_OBJECT
public:
    Invoice(QObject*parent = 0);
    ~Invoice() {};

    bool saveAs(const QString &fileName);
    bool load(const QString &fileName);
    void newInvoice();
    bool isModified() const;

    InvoiceItems* items() const { return mItems; }
    InvoiceHead* head()  const {return mHead;}

    void addItem(InvoiceItem *i);
    void deleteItem(InvoiceItem *i);

    QString toString();

private:
    bool modified;
    InvoiceHead* mHead;
    InvoiceItems* mItems;

};
```




```
public slots:
        void setModified(bool val = true) { modified = val; }
        void loadHeadField(int i, QVariant value);
        void setHeadField(const QString &name,QVariant value);
        void setHeadField(int i, QVariant value);
signals:
        void itemsModified();
        void headModified();
        void headFieldChanged(const QString &, QVariant);
```

Invoice: implementáció

invoice.cpp

```
Invoice::Invoice(QObject *parent)
{
	setParent(parent);
	setObjectName("Invoice");
	modified = false;
	mItems = new InvoiceItems(this);
	mHead = new InvoiceHead(this);
}

void Invoice::newInvoice()
{
	mHead->init();
	mItems->init();
	emit headModified();
	emit itemsModified();
	setModified(false);
}
```

Invoice: implementáció

invoice.cpp

```
bool Invoice::load(const QString & fileName)
{
	QFile invoiceFile(fileName);
	if(!invoiceFile.open(QIODevice::ReadOnly | QIODevice::Text))
	{
		qWarning(QString("Warning: ** " + fileName + " file is missing or not a text file. ** ").toAscii());
		return false;
	}
	QTextStream in(&invoiceFile);
	mHead->init();
	int headCount = in.readLine().toInt();
	for (int i=0; i< headCount; i++)
		mHead->field(i)->setValue(in.readLine());
	emit headModified();

	int itemsCount = in.readLine().toInt();
	for (int i=0; i< itemsCount; i++)
	{
		QString name = in.readLine();
		QString unit = in.readLine();
		int quantity = in.readLine().toInt();
		int unitPrice = in.readLine().toInt();
		int vatPercent = in.readLine().toInt();
		mItems->addItem(new InvoiceItem(name,unit,quantity,unitPrice,vatPercent));
	}
	emit itemsModified();
	setModified(false);
	return true;
}
```

Invoice: implementáció

invoice.cpp

```
bool Invoice::saveAs(const QString & fileName)
{
	QFile invoiceFile(fileName);
	if(!invoiceFile.open(QIODevice::WriteOnly))
	{
		qWarning((QString("Cannot open file for writing:"
			+ invoiceFile.errorString() + "\n")).toAscii());
		return false;

	} else {
		QTextStream out(&invoiceFile);

		out << QString::number(mHead->fieldsToSave().count()) << endl;
		for (int i=0; i < mHead->fieldsToSave().count(); ++i)
			out << mHead->fieldsToSave().at(i) << endl;

		out << QString::number(mItems->count()) << endl;
		for (int i=0; i< mItems->count(); ++i)
			for (int j=0; j < mItems->item(i)->fieldsToSave().count(); ++j)
				out << mItems->item(i)->fieldsToSave().at(j) << endl;

		return true;
	}
}
```

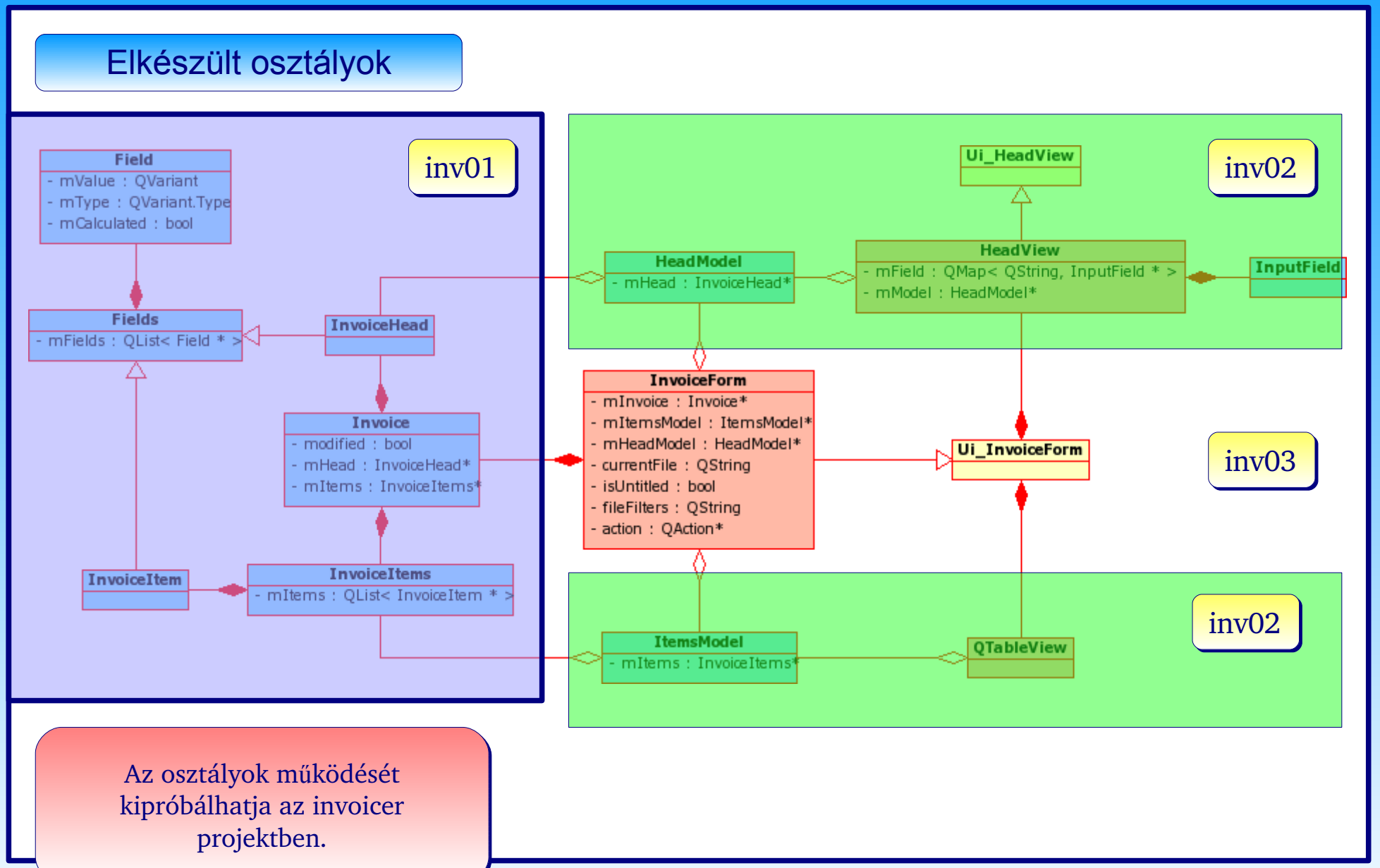
Invoice: implementáció

invoice.cpp

```
void Invoice::addItem(InvoiceItem *item){
        mItems->addItem(item);
        emit itemsModified();
        setModified(true);
}
void Invoice::deleteItem(InvoiceItem *i){
        mItems->deleteItem(i);
        emit itemsModified();
        setModified(true);
}
void Invoice::loadHeadField(int i,QVariant value){
        mHead->field(i)->setValue(value);
        setModified(true);
}
void Invoice::setHeadField(const QString &name, QVariant value){
        if (mHead->field(name)->value()==value) return;
        mHead->field(name)->setValue(value);
        setModified(true);
        emit headFieldChanged(name,value);
}
void Invoice::setHeadField(int i,QVariant value){
        if (mHead->field(i)->value()==value) return;
        mHead->field(i)->setValue(value);
        setModified(true);
        emit headFieldChanged(mHead->field(i)->name(),value);
}
```

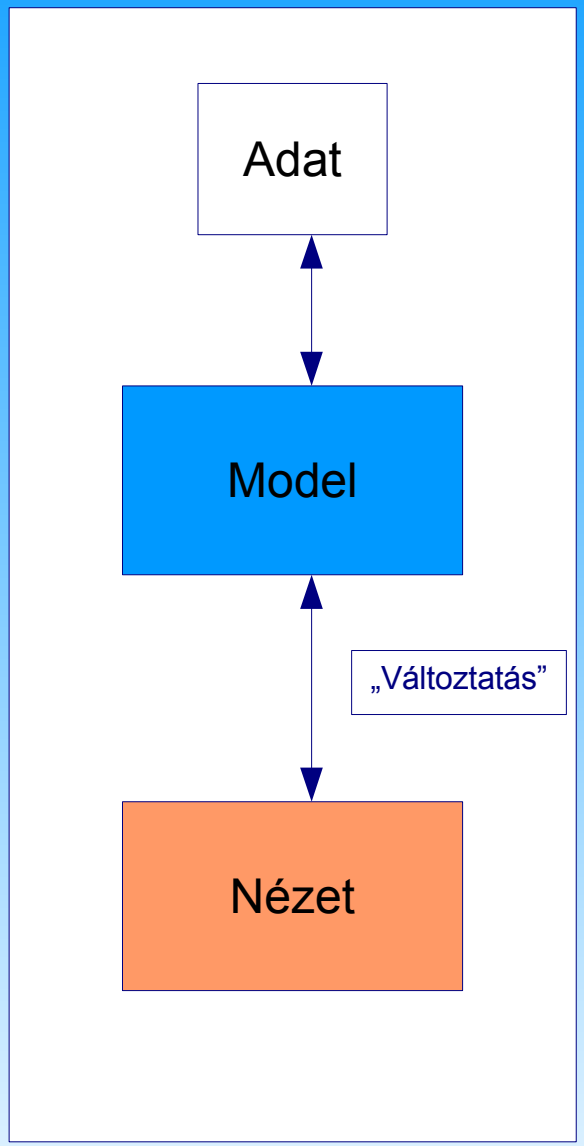
Invoicer projekt: adat osztályok

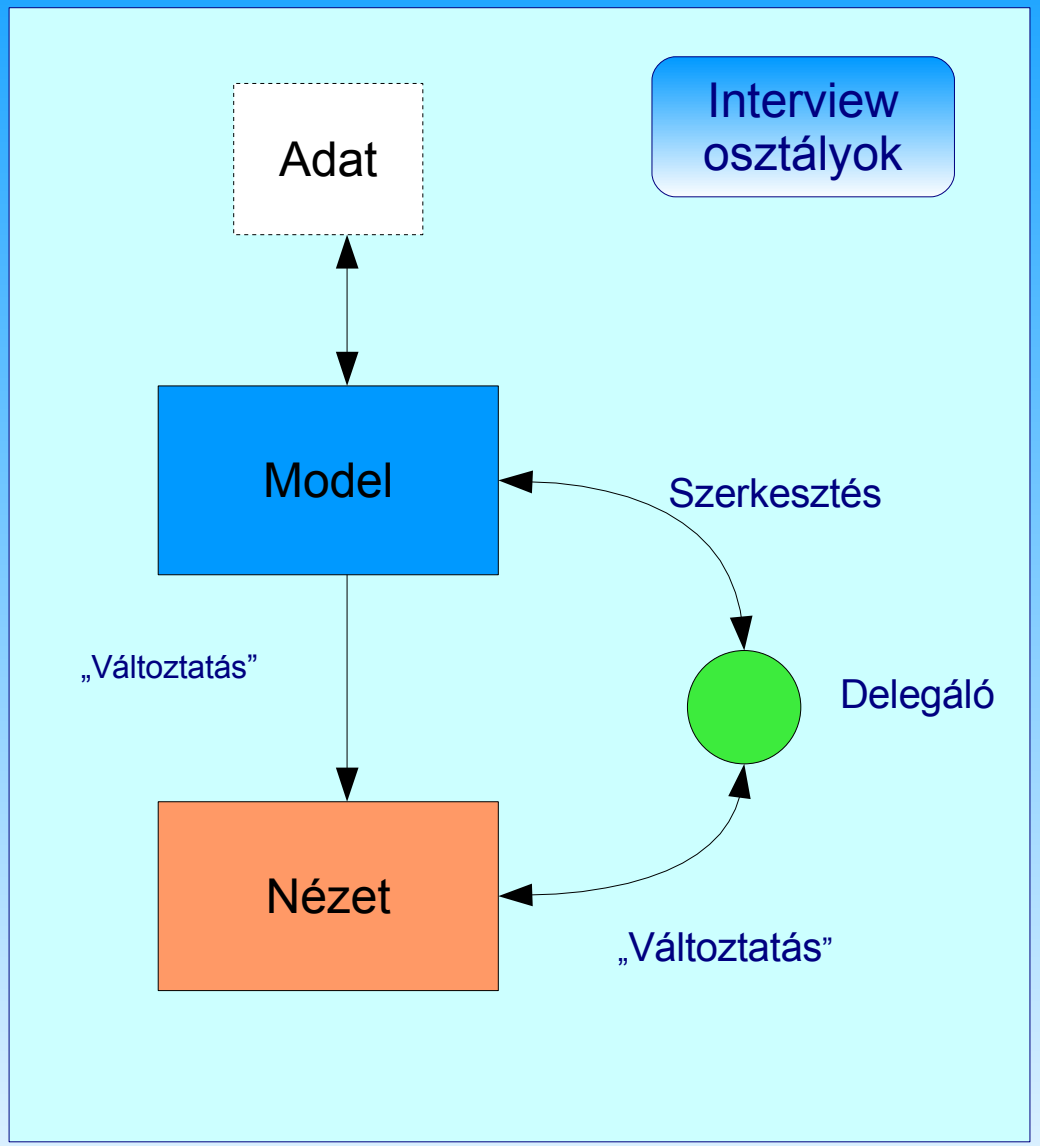
Elkészült osztályok



Az osztályok működését kipróbálhatja az invoicer projektben.

Előretekintés:modell-nézet tervezési minta





Elemhalmazt megjelenítő osztályok

Hagyományos vezérlők:

- QListWidget
- QTableWidget
- QTreeWidget

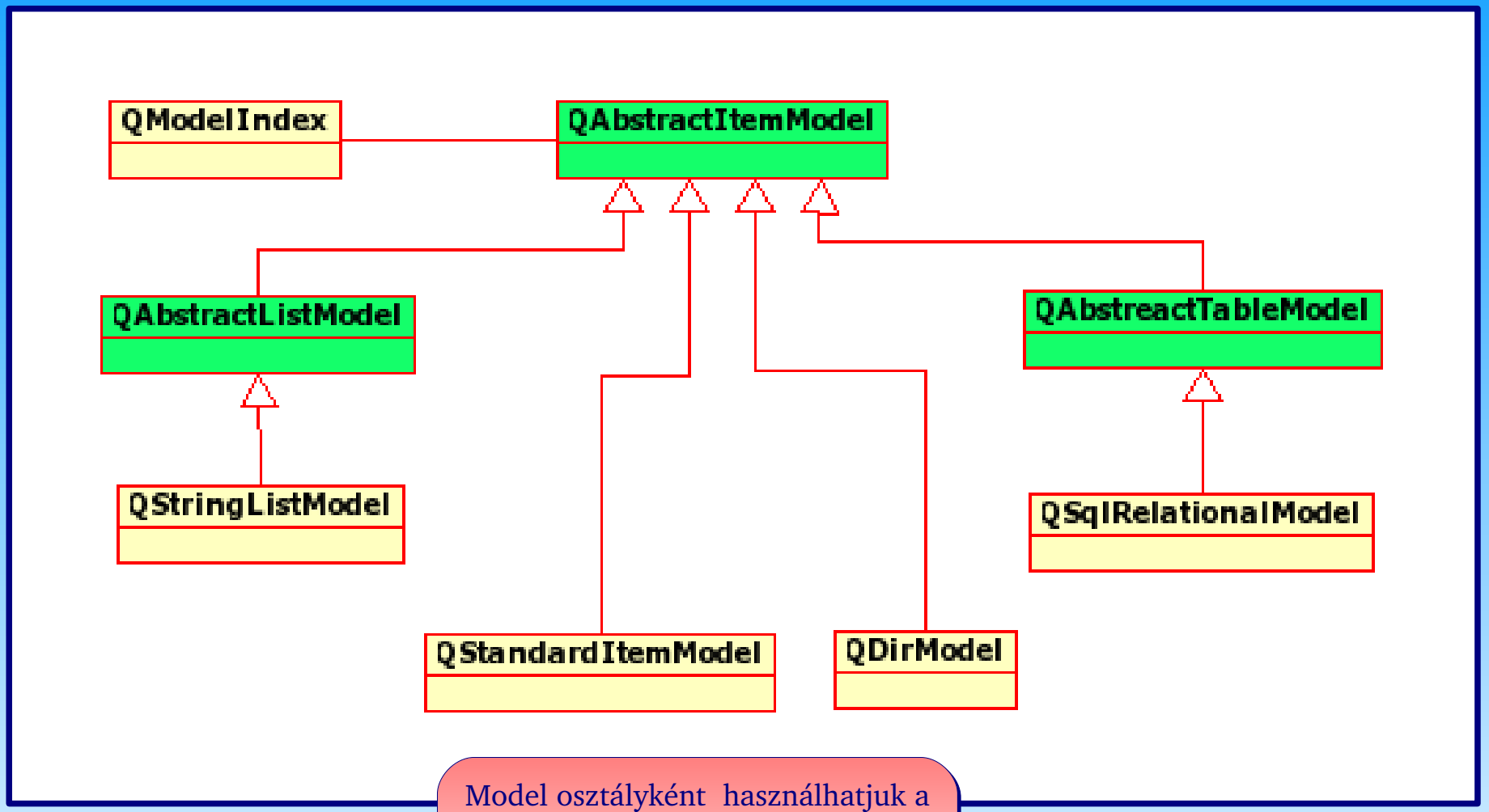
A hagyományos vezérlőkben az adatokat a vezérlőkben tároljuk.

„Nézet” vezérlők:

- QListView
- QTableView
- QTreeView

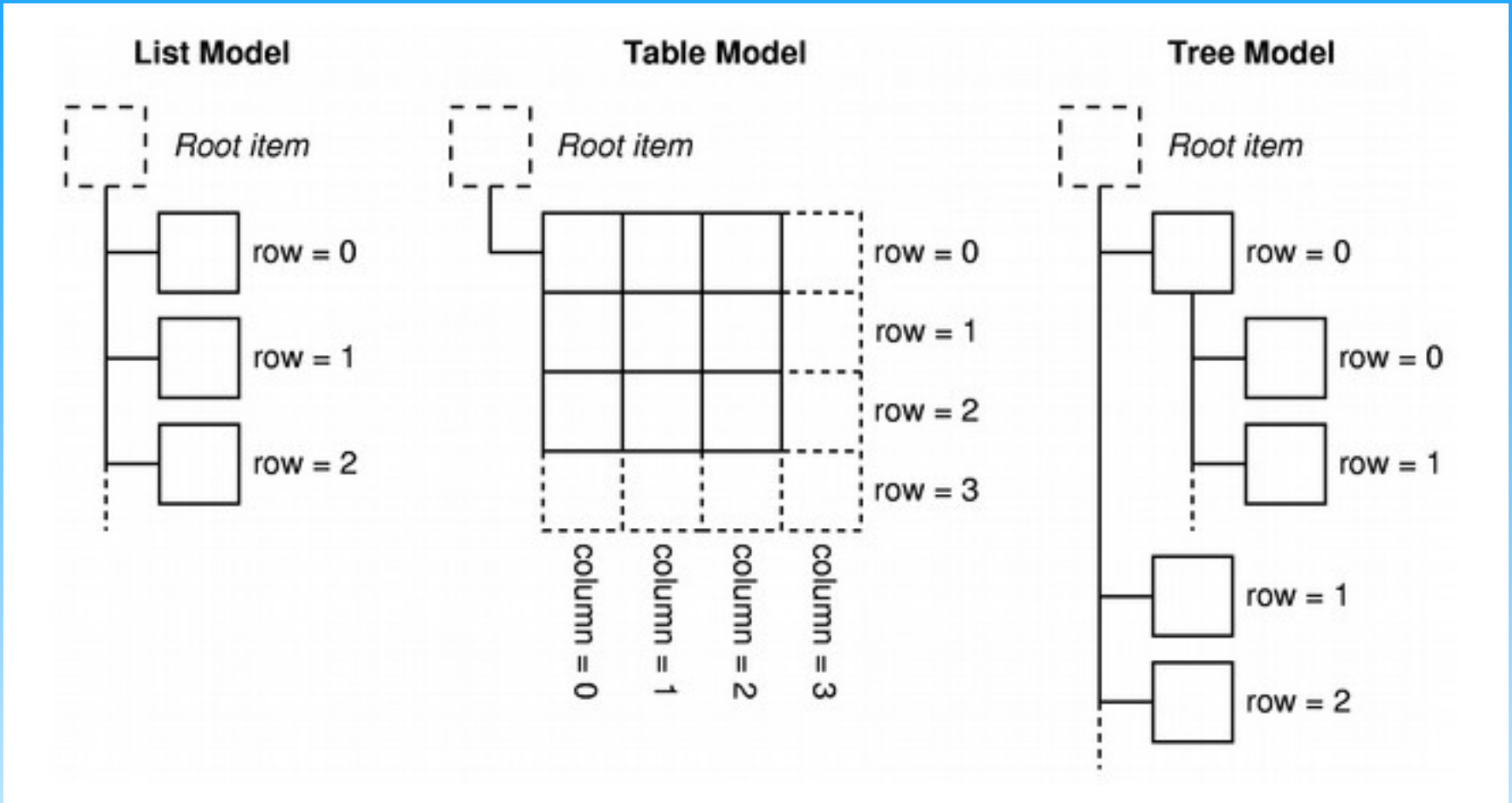
A model/view megközelítésben az adatokat egy önálló (model) osztály szolgáltatja.

Qt4 Model osztályok



Model osztályként használhatjuk a Qt konkrét model osztályait, de készíthetünk az abszrakt osztályokból származtatva saját model osztályt is.

Model/View: indexelés (QModelIndex)



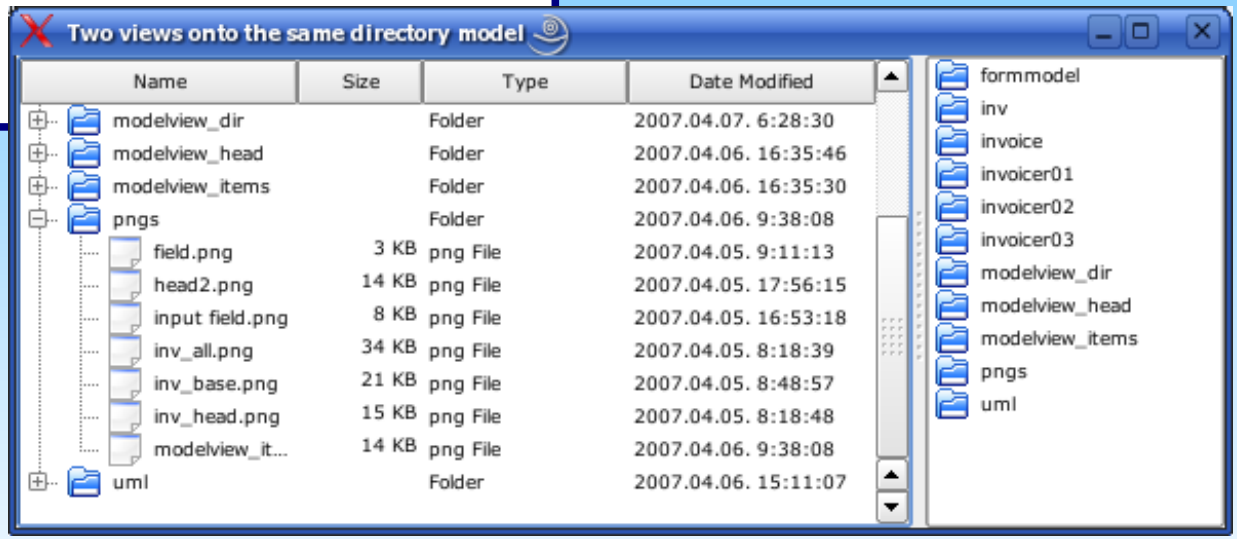
Index lekérdezése: QModelIndex index = model->index(row, column, ...);

Adatra (cellára) mutató pointer lekérdezése: QAbstractItemModel *model = index.model();

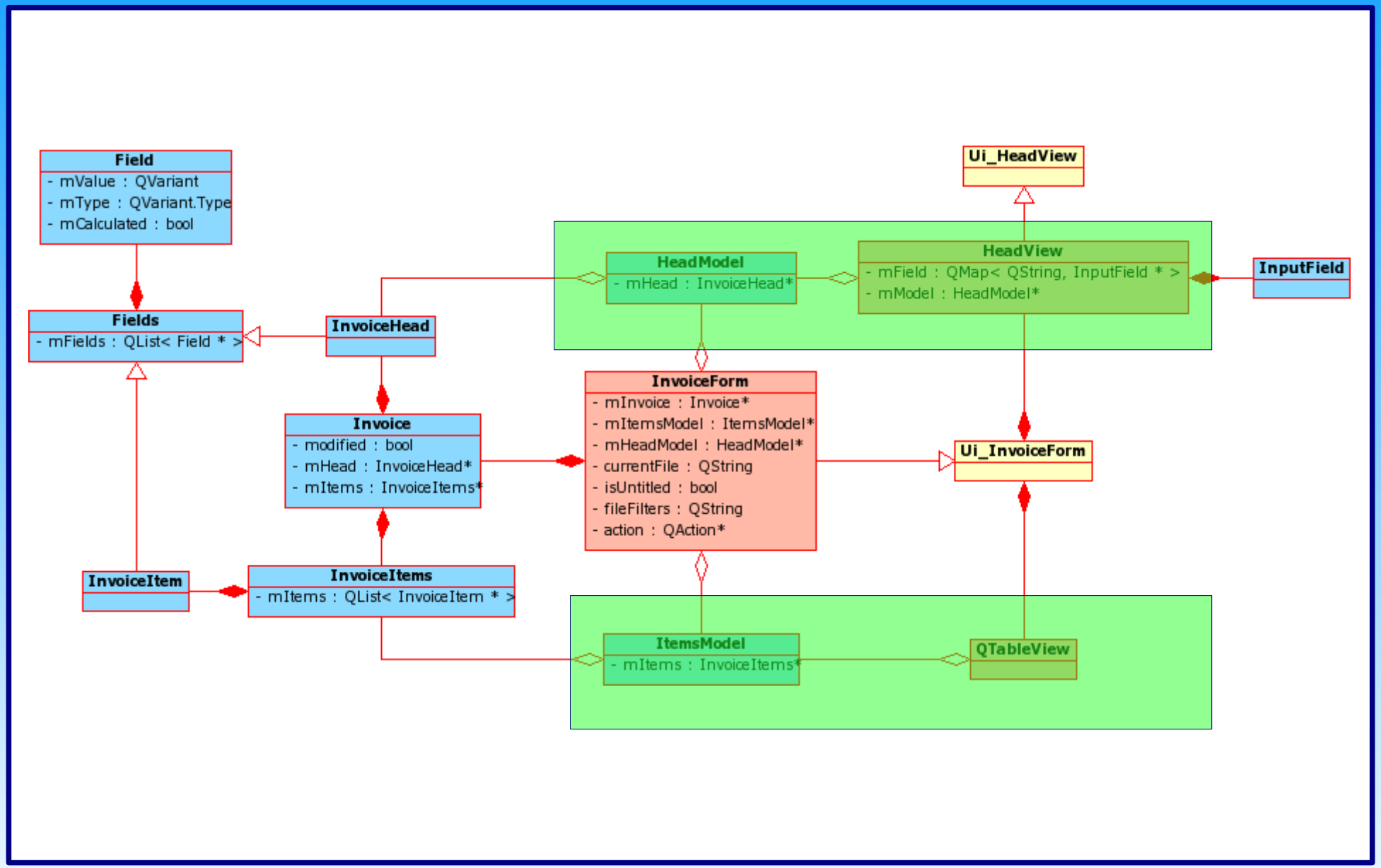
Model/View: példa

main.cpp

```
int main(int argc, char *argv[])  {
    QApplication app(argc, argv);
    QSplitter *splitter = new QSplitter;

    QDirModel *model = new QDirModel;

    QTreeView *tree = new QTreeView(splitter);
    tree->setModel(model);
    tree->setRootIndex(model->index(QDir::currentPath()));

    QListView *list = new QListView(splitter);
    list->setModel(model);
    list->setRootIndex(model->index(QDir::currentPath()));

    splitter->setWindowTitle("Two views onto the same directory model");
    splitter->show();
    return app.exec();
}
```



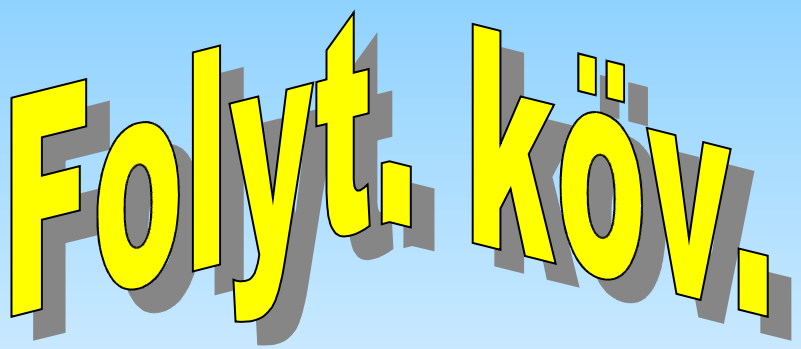
Modell-nézet minta alkalmazása a projektben



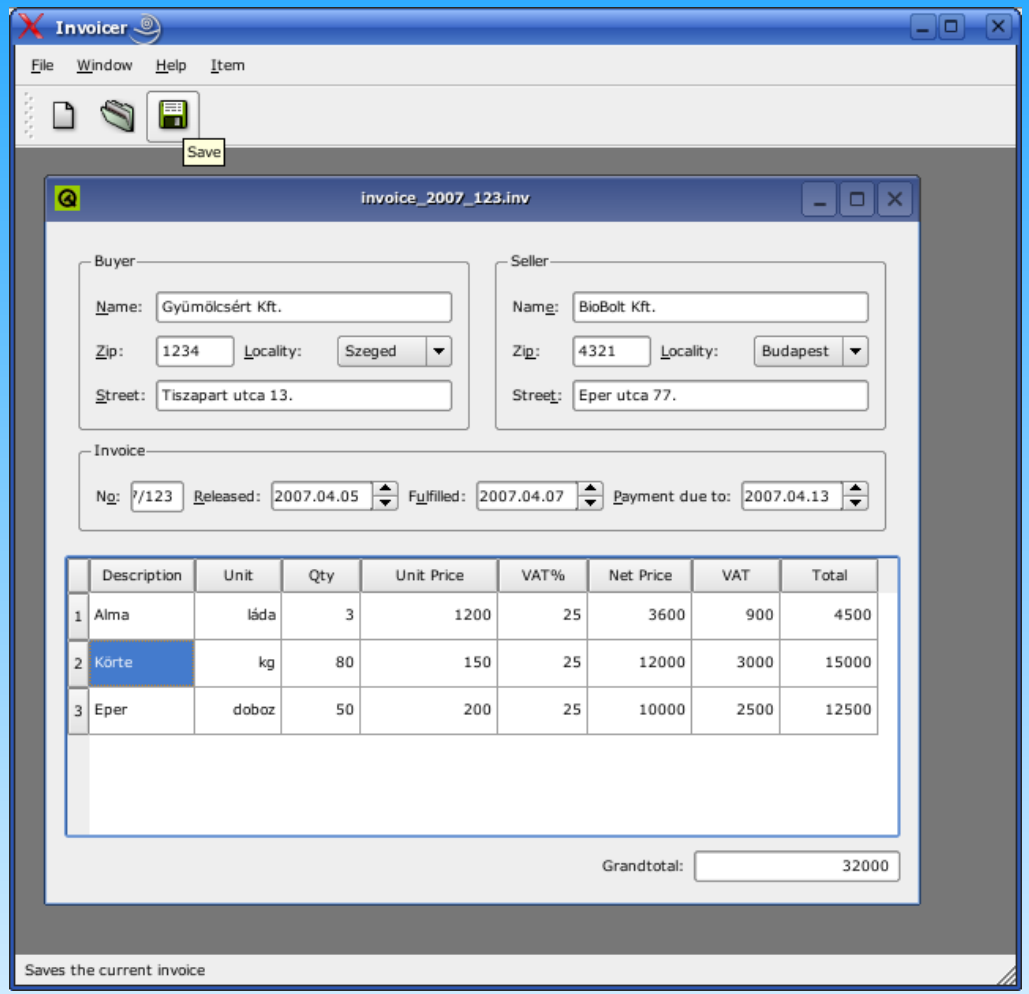
A bemutatott programok megtalálhatók a

people.inf.elte.nacsa/qt4/eaf3/inv01/projects/

címen.



Elemi alkalmazások fejlesztése III.



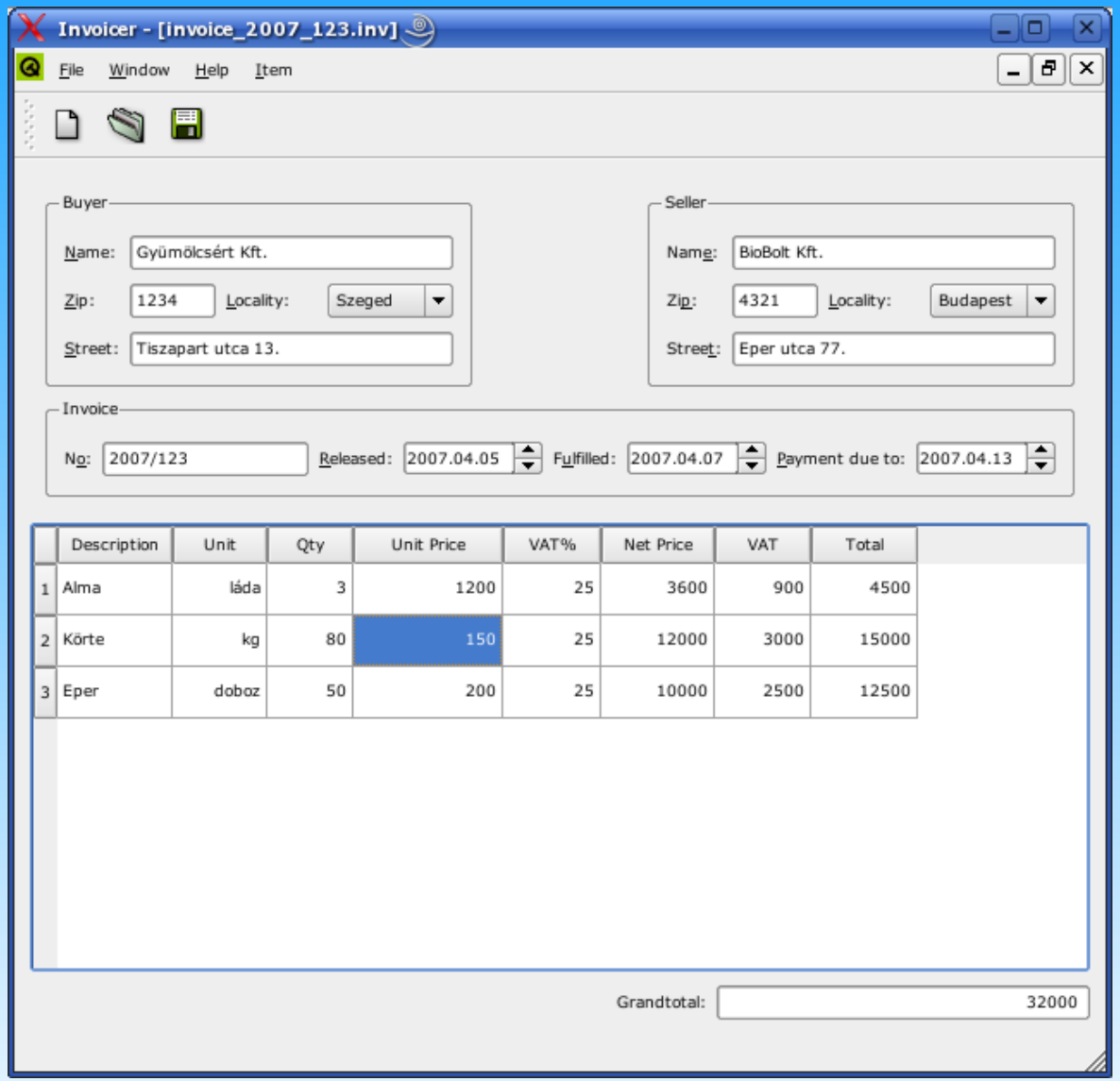
MDI alkalmazás II.
(Számla)

Készítette: Szabóné Nacsa Rozália
nacsa@inf.elte.hu
people.inf.elte.hu/nacsa/qt4/eaf3/
Qt 4
2007

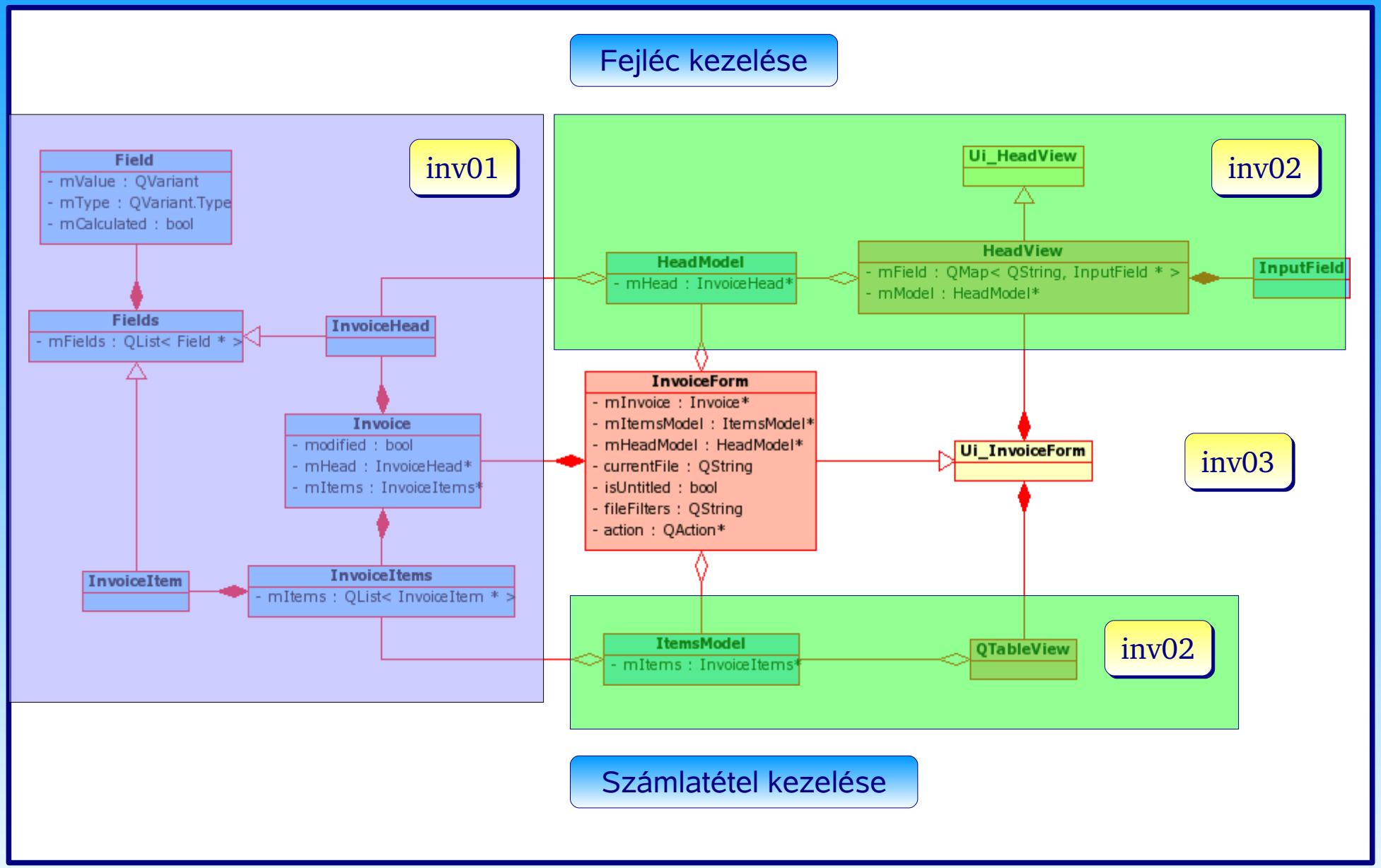
Számla felépítése

fejléc

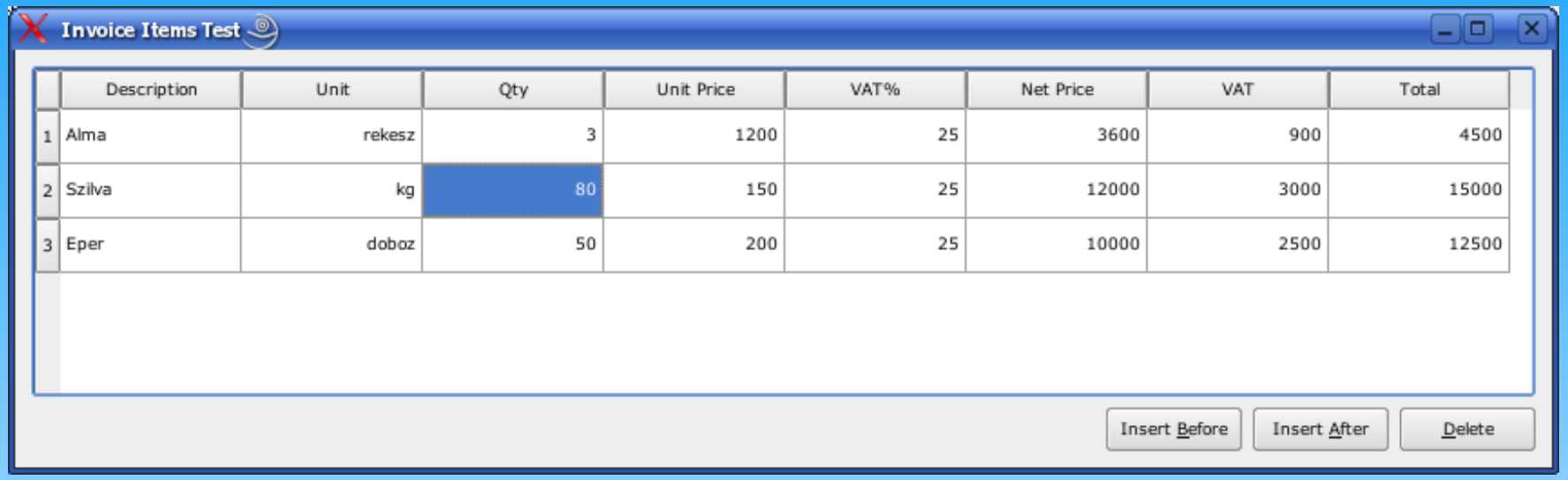
tábla



Az alkalmazás osztálydiagramja

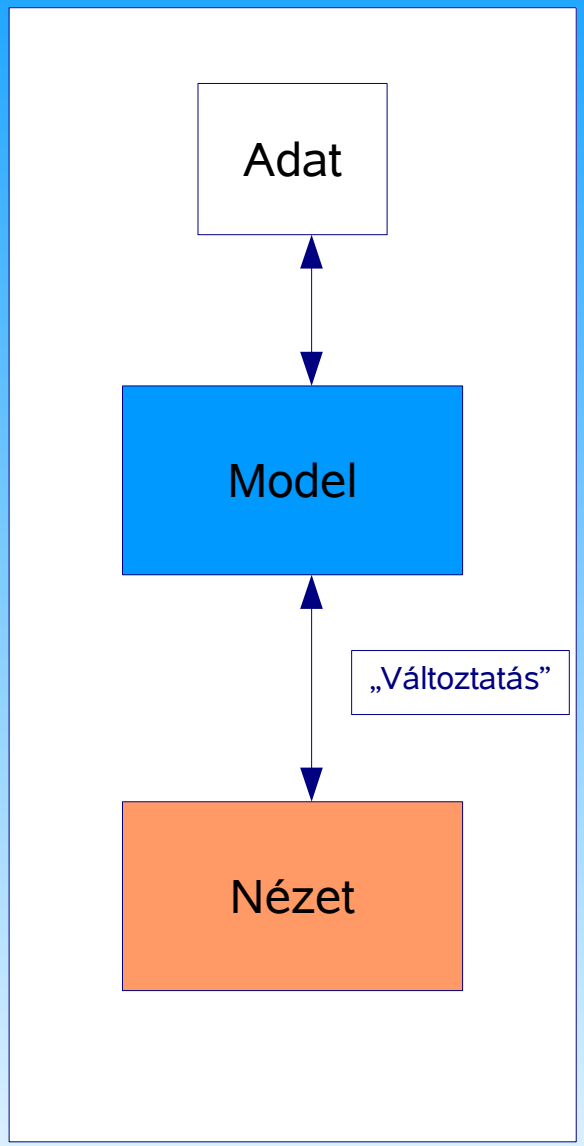


Számlatételek kezelése - “táblakezelés”



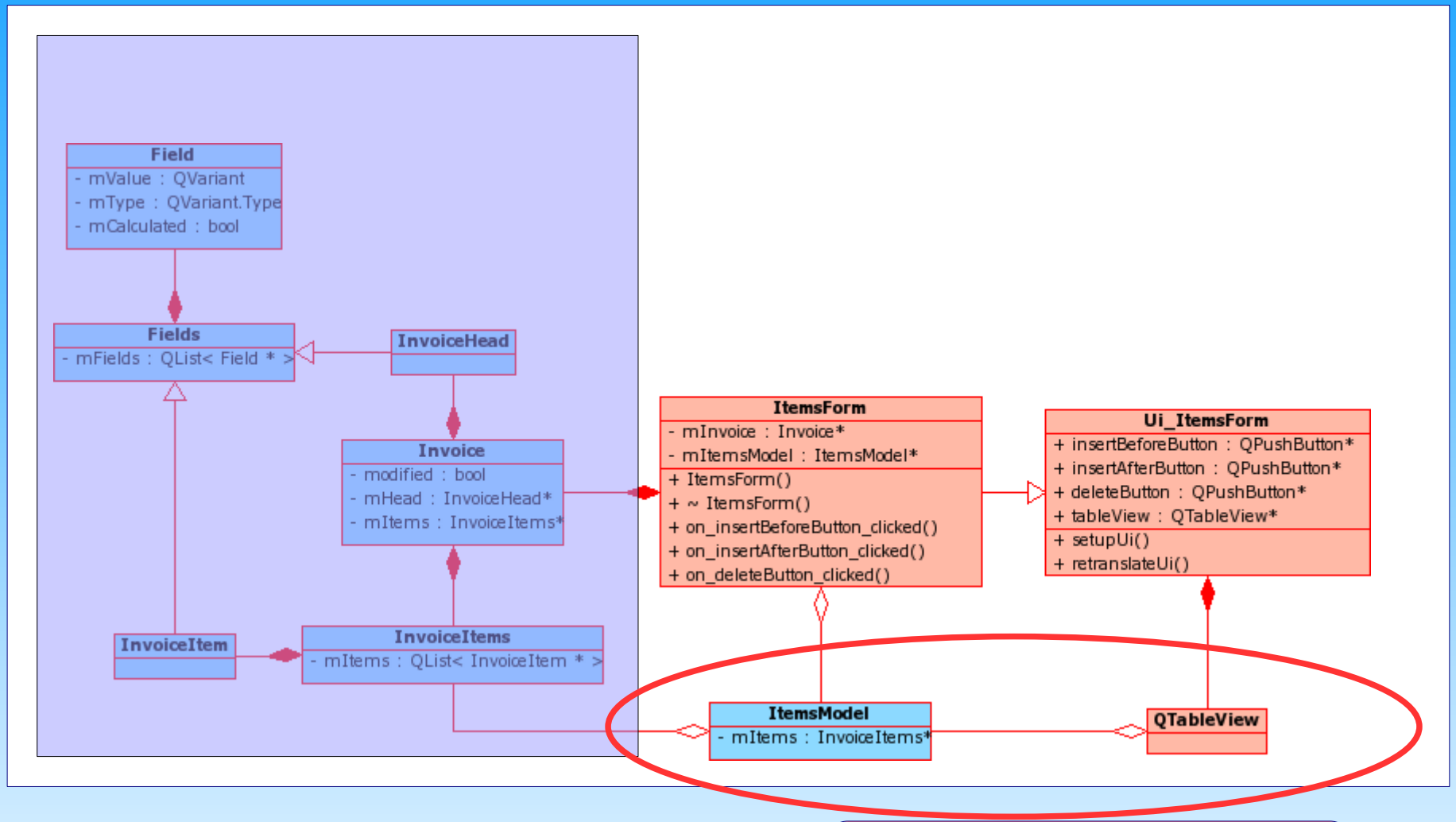
Készítünk egy olyan (egyszerűbb) alkalmazást, amelyben "csak" számlatételeket lehet szerkeszteni.

Model/View/Controller



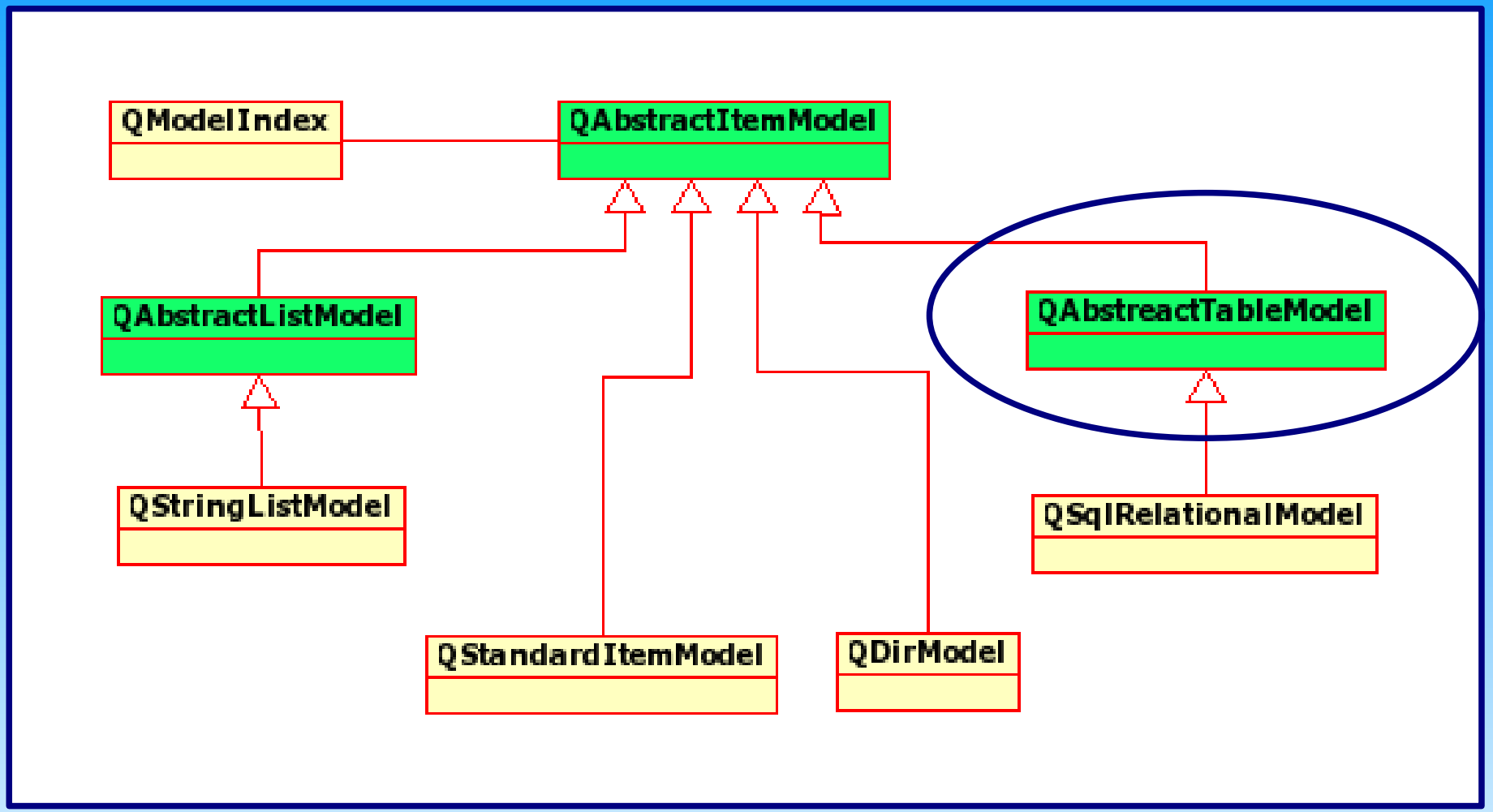

```
. . .

mItemsModel = new ItemsModel(parent);
tableView->setModel(mItemsModel);

. . .
```

ItemsForm űrlap



Az adatszolgáltatást és az adatmegjelenítést külön osztályokban valósítjuk meg.

A Qt4 Model osztályai



ItemsModel osztály: definíció

itemsmodel.h

```
class ItemsModel : public QAbstractTableModel{
    Q_OBJECT

public:
    ItemsModel(QObject* parent=0);
    ~ItemsModel(){}

     //Definiálandó  metódusok
    int rowCount(const QModelIndex & parent = QModelIndex()) const;
    int columnCount(const QModelIndex & parent = QModelIndex()) const;
    QVariant data(const QModelIndex & index, int role = Qt::DisplayRole ) const;
    QVariant headerData (int section, Qt::Orientation orientation, int role = Qt::DisplayRole) const;

     Qt::ItemFlags flags ( const QModelIndex & index ) const;
    bool setData ( const QModelIndex & index, const QVariant & value, int role = Qt::EditRole );
    bool insertRows(int position, int rows,const QModelIndex &index = QModelIndex());
    bool removeRows(int position, int rows,const QModelIndex &index = QModelIndex());

public slots:
    void resetModel();

signals:
    void dataChanged(QModelIndex,QModelIndex);
    void modelModified();

private:
    InvoiceItems* mItems;
};
```

Adat szolgáltatás

Szerkesztés

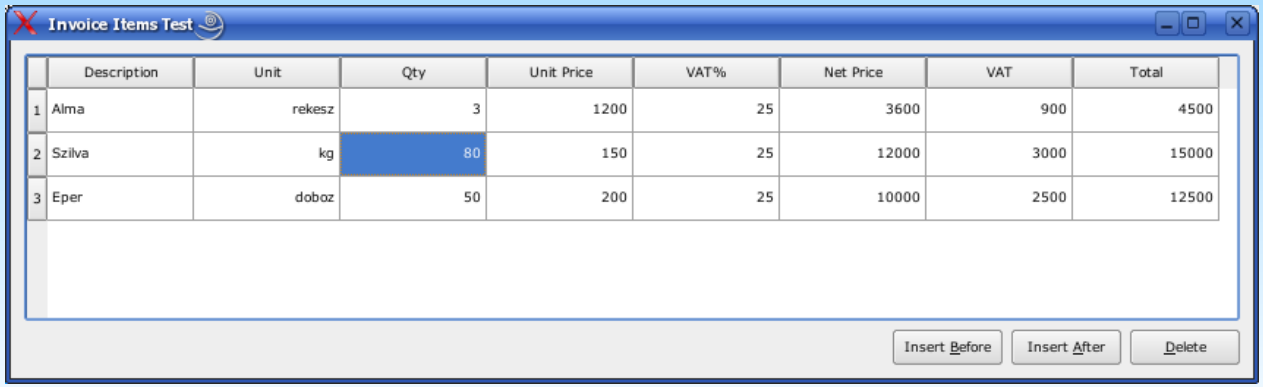
```
public:
    // Egyéb metódusok
    void addInvoiceItem(QModelIndex &index);
    void removeInvoiceItem(QModelIndex &index);
    void setItems(InvoiceItems* items);

    QString grandTotal();
    QString toString();
```

ItemsModel osztály: implementáció

```
QVariant ItemsModel::headerData (int section, Qt::Orientation orientation, int role) const
{

        if(role != Qt::DisplayRole) return QVariant();
        if (orientation == Qt::Horizontal)
                return (InvoiceItem::titles())[section];
        else
                return section + 1;
}

Qt::ItemFlags ItemsModel::flags(const QModelIndex &index) const {
    if (!index.isValid())
        return Qt::ItemIsEnabled;
        if(index.isValid() && index.column() > 4) //számított mező csak olvasható
            return QAbstractItemModel::flags(index) | Qt::ItemIsEnabled;
        else
            return QAbstractItemModel::flags(index) | Qt::ItemIsEditable;
}
```

Az oszlopok feliratát szolgáltató függvény.

A cellákon végezhető műveletek beállítása.



ItemsModel osztály: implementáció

itemsmodel.cpp

```
QVariant ItemsModel::data(const QModelIndex & index, int role) const
{
	if(!index.isValid()) return QVariant();

	if (role == Qt::TextAlignmentRole) {
		if (index.column() == 0)
			return int(Qt::AlignLeft | Qt::AlignVCenter);
		else
			return int(Qt::AlignRight | Qt::AlignVCenter);
	} else if (role == Qt::DisplayRole) {
			return mItems->item(index.row())->field(index.column())->value();
	}
	return QVariant();
}

bool ItemsModel::setData ( const QModelIndex & index, const QVariant & value, int role )
{
	if(index.isValid() && role == Qt::EditRole){
		mItems->item(index.row())->field(index.column())->setValue(value);
		emit dataChanged(index,index);
		emit modelModified();
		return true;
	} else
		return false;
}
```

Az index-szel azonosított elem értékét szolgáltatja.

Az index-szel azonosított elem értékét value értékre állítja.
Az index változó érvényességét ellenőrizni kell.

ItemsModel osztály: implementáció

itemsmodel.cpp

```
bool ItemsModel::insertRows(int position, int rows,const QModelIndex &parent)
{
        if (position==-1)
                position=rowCount()-1;
        beginInsertRows(parent, position, position + rows -1);
        for (int row = 0; row < rows; ++row)
                mItems->insertItem(position,new InvoiceItem());
        QModelIndex topLeft(index(position,0,parent));
        QModelIndex bottomRight(index(position+1,columnCount(),parent));
        emit dataChanged(topLeft,bottomRight);
        emit modelModified();
        endInsertRows();
        return true;
}
```

Model (adatok) aktualizálása sorok beszúrásakor.

```
bool ItemsModel::removeRows(int position, int rows, const QModelIndex &parent)
{
        if(rowCount() == 0) return false;
        beginRemoveRows(parent, position, position + rows -1);
        for (int row = 0; row < rows; ++row)
                mItems->deleteItemAt(position);
        QModelIndex topLeft(index(position,0,parent));
        QModelIndex bottomRight(index(position+1,columnCount(),parent));
        emit dataChanged(topLeft,bottomRight);
        emit modelModified();
        endRemoveRows();
        return true;
}
```

Model (adatok) aktualizálása sorok törlésekor.

ItemsModel osztály: implementáció

itemsmodel.cpp

```
void ItemsModel:: addInvoiceItem(QModelIndex &index)
{
      if(!index.isValid())     insertRows(rowCount(),1);
      else                     insertRows(index.row(),1);
}

void ItemsModel::removeInvoiceItem(QModelIndex &index)
{
      if(!index.isValid() || rowCount() == 0)      return;
      else                                         removeRows(index.row(),1);
}

void ItemsModel::setItems(InvoiceItems* items)
{
      mItems = items;
      resetModel();
}

void ItemsModel::resetModel()
{
      reset();
      emit modelModified();
}
```

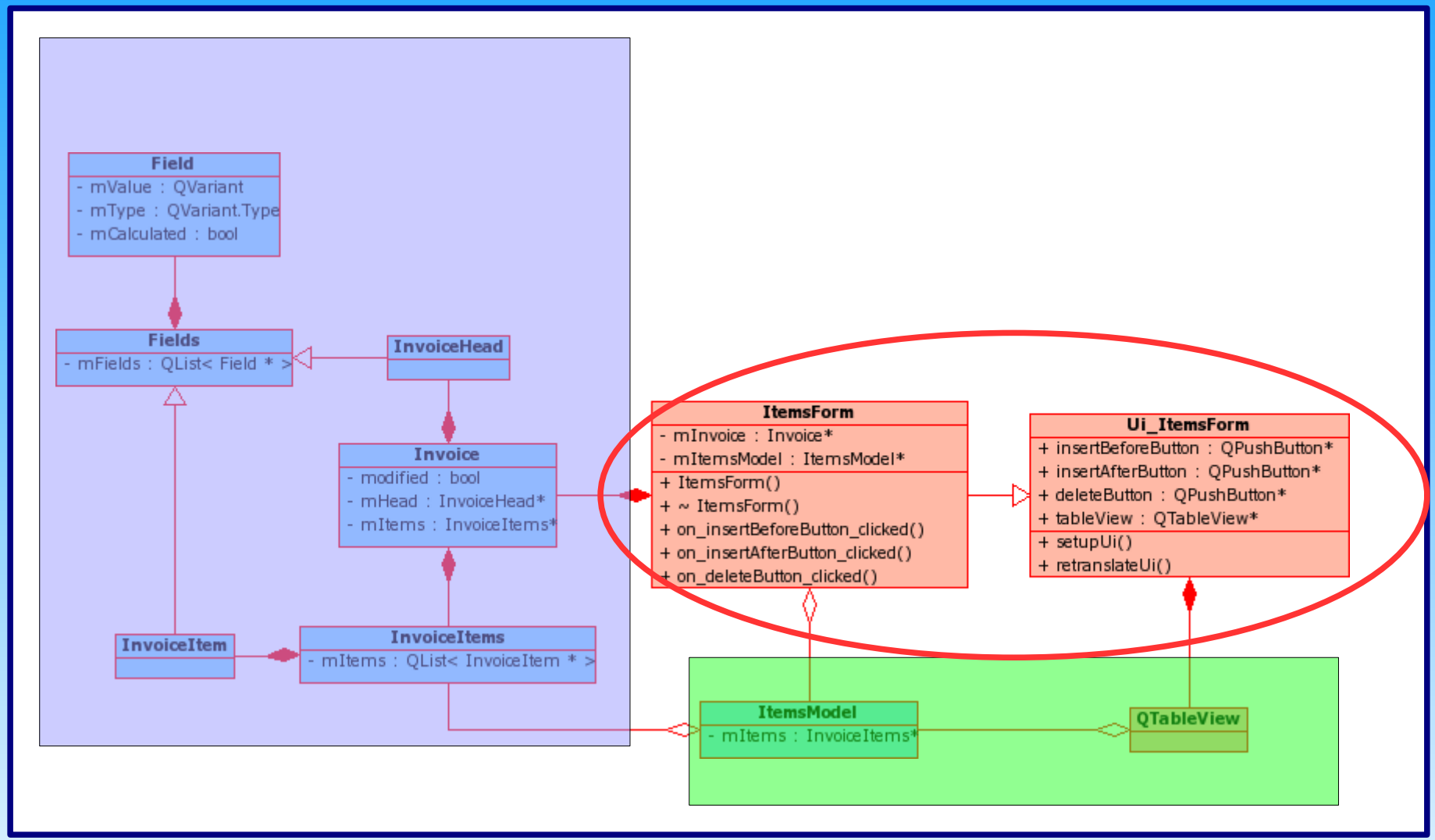
Új sor (számlatétel) beillesztése a táblába.

Sor (számlatétel) törlése a táblából.

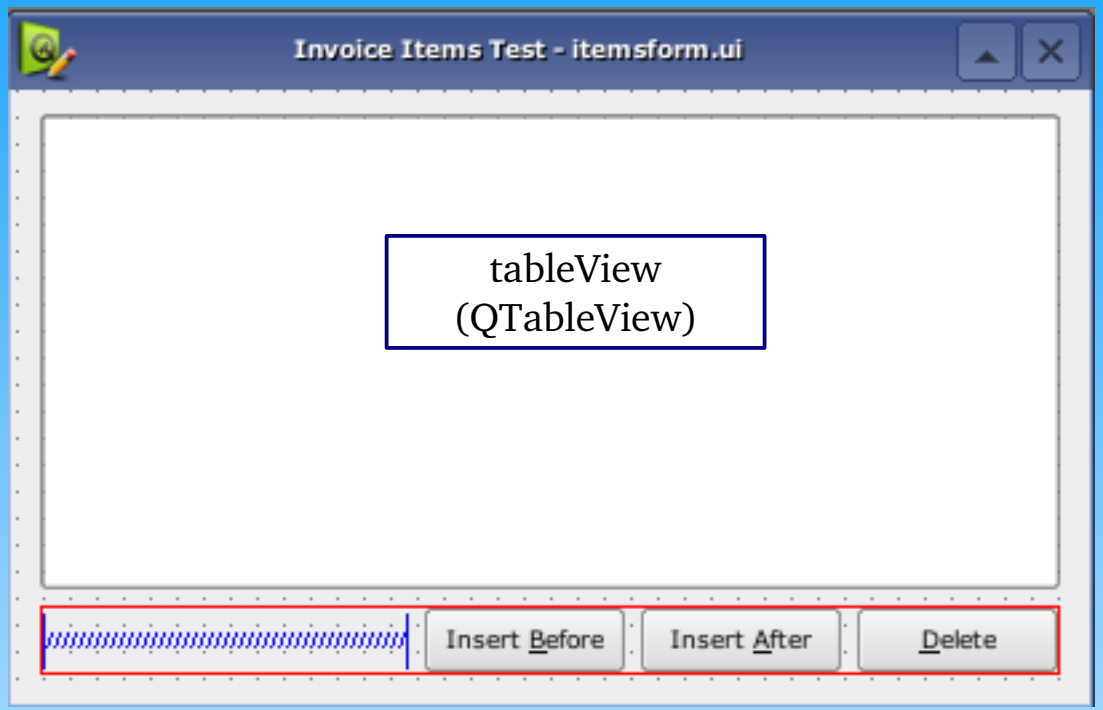
A számlatételeket tartalmazó listára mutató pointer beállítása.

A resetModel() "értesíti" a megjelenítő osztályokat az adatok megváltozásáról.

A számlatétel kezelő program osztálydiagramja



ItemsForm megtervezése Qt Designerrel



insertBeforeButton
(QPushButton)

insertAfterButton
(QPushButton)

deleteButton
(QPushButton)

ItemsForm osztály

itemsform.h

```
class ItemsForm : public QWidget, public Ui_ItemsForm
{
        Q_OBJECT

public:
         ItemsForm(Invoice* invoice, QWidget *parent=0);
        ~ItemsForm(){}

public slots:
        void on_insertBeforeButton_clicked();
        void on_insertAfterButton_clicked();
        void on_deleteButton_clicked();

private:
        Invoice* mInvoice;
        ItemsModel* mItemsModel;

};
```

itemsform.cpp

```
ItemsForm::ItemsForm(Invoice* invoice, QWidget *parent)
        : QWidget(parent), mInvoice(invoice)
{
        setupUi(this);

        mItemsModel = new ItemsModel(parent);
        mItemsModel->setItems(mInvoice->items());
        tableView->setModel(mItemsModel);

        if (mItemsModel->rowCount() > 0)
                tableView->setCurrentIndex(mItemsModel->index(0,0));
}
```

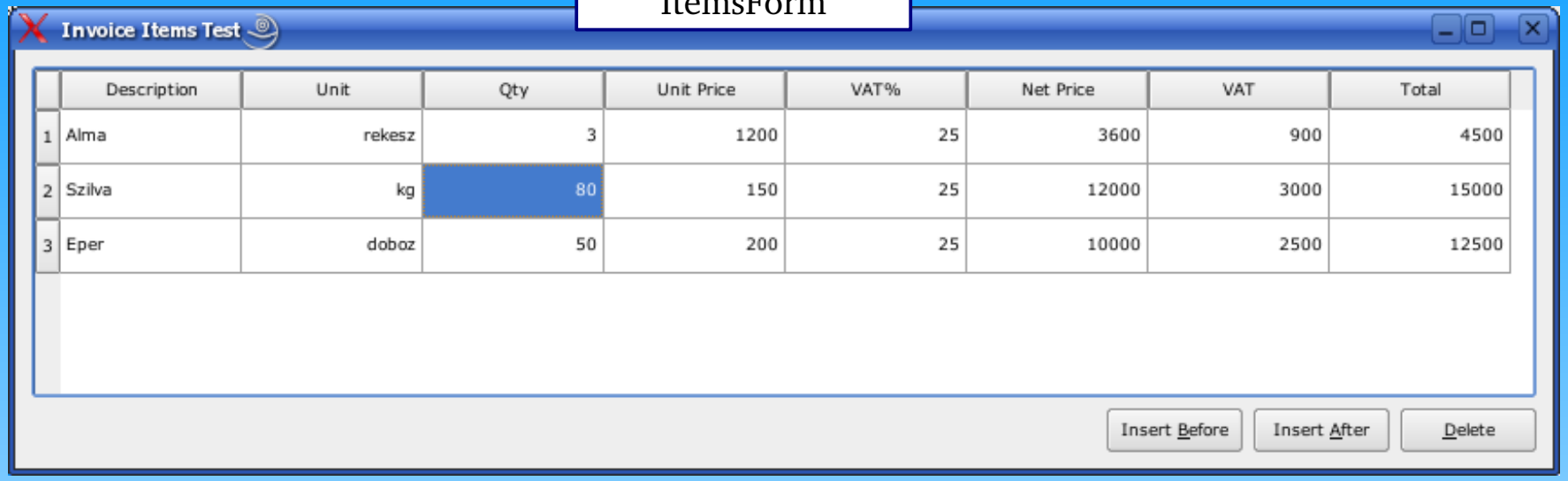
ItemsForm osztály

itemsform.cpp

```
void ItemsForm::on_deleteButton_clicked()
{
    QModelIndex index = tableView->currentIndex();
    if(!index.isValid()) return;

    mItemsModel->removeInvoiceItem(index);

    int nextRow=(index.row()<mItemsModel->rowCount())? index.row():mItemsModel->rowCount()-1;
    if(nextRow >= 0)
        tableView->setCurrentIndex(mItemsModel->index(nextRow,index.column()));
}

void ItemsForm::on_insertBeforeButton_clicked()
{
    QModelIndex index = tableView->currentIndex();

    mItemsModel->addInvoiceItem(index);

    if (mItemsModel->rowCount() == 1)
        index = mItemsModel->index(0,0);
    tableView->setCurrentIndex(index);
    if(index.isValid()) tableView->edit(index);
}
```

Számlatétel kezelése: főprogram

ItemsForm

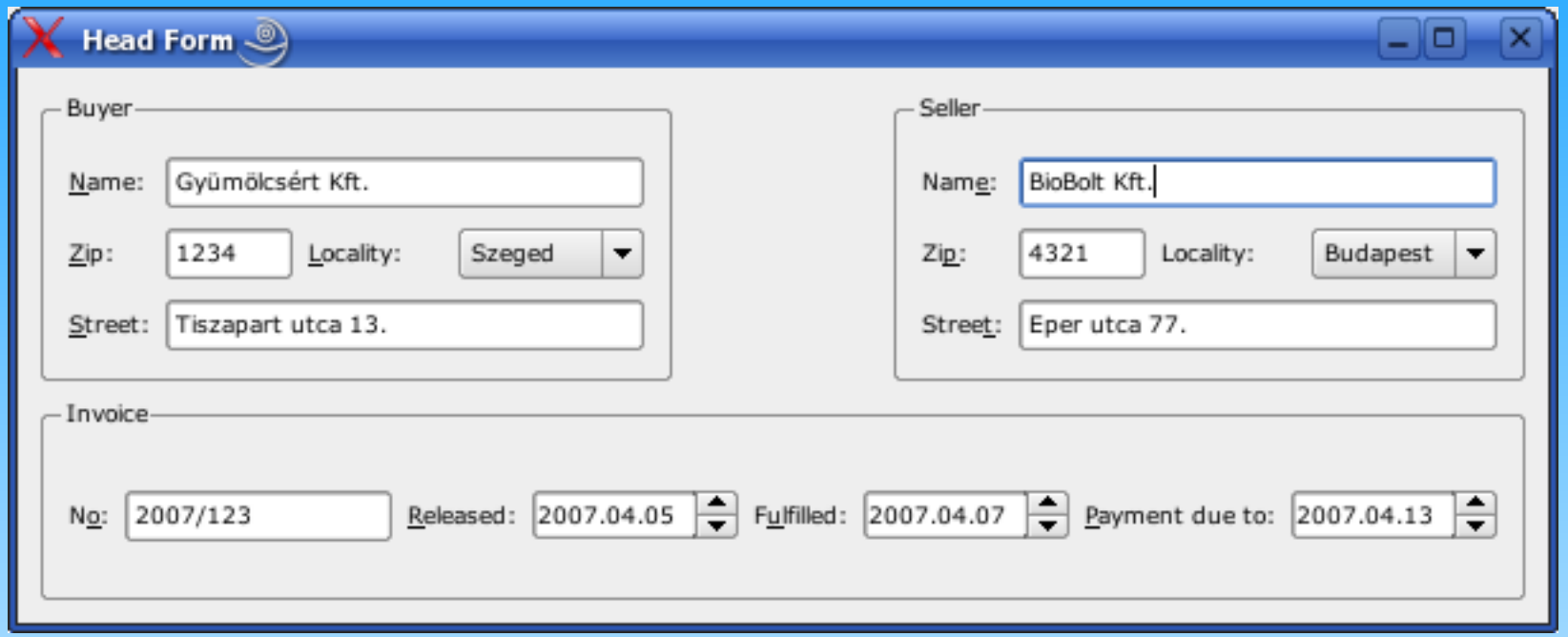


main.cpp

```
int main (int argc, char *argv[])
{
        QApplication app(argc,argv);
        Invoice* invoice = new Invoice();
        invoice->load("./invoices/2007/invoice_2007_123.inv");

        ItemsForm *itemsForm = new ItemsForm(invoice);
        itemsForm->show();

        bool ret = app.exec();
        invoice->saveAs("./invoices/2007/invoice_2007_567.inv");
        return ret;
}
```

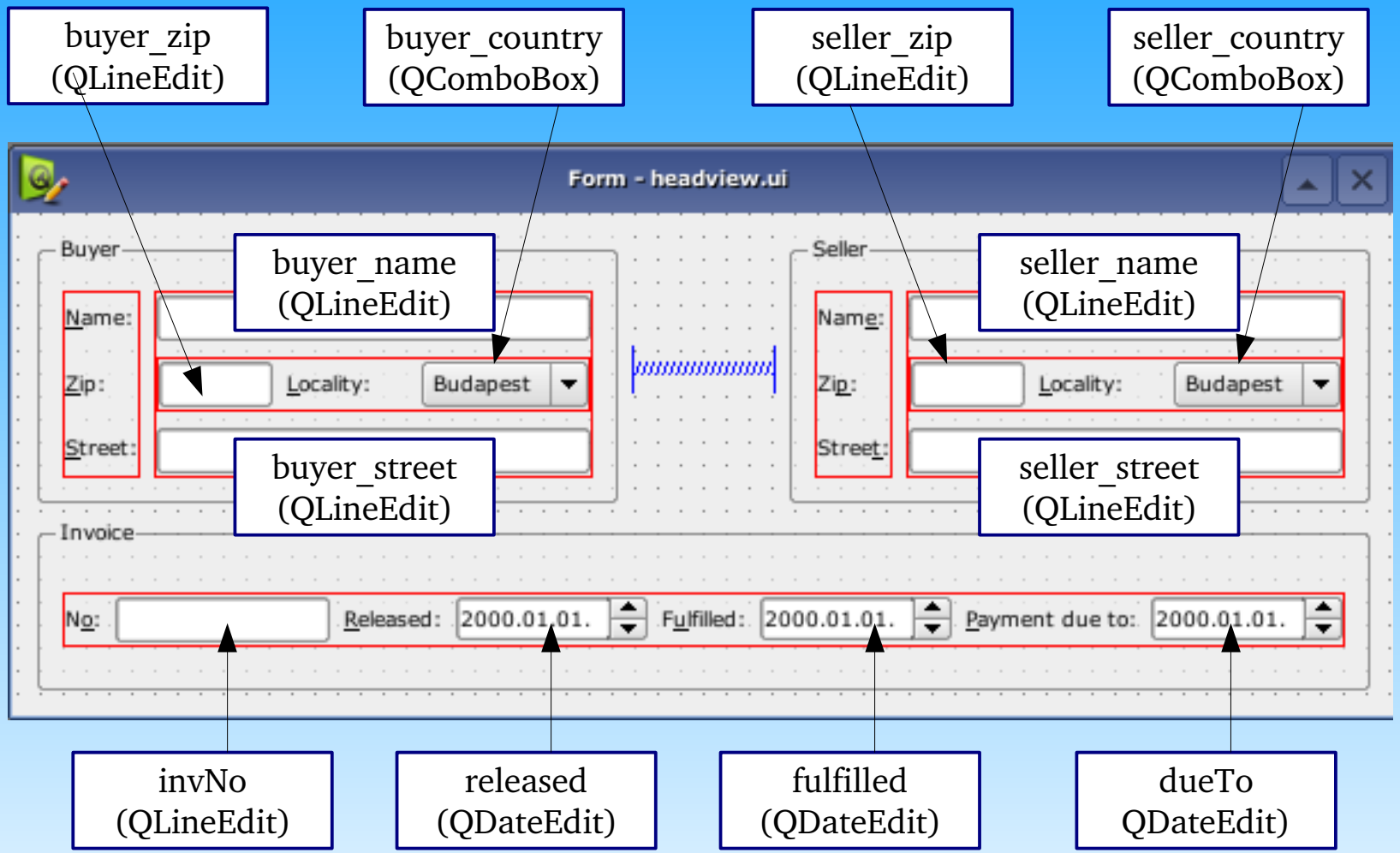
Ezzel a programmal a számlatételek (táblázat) szerkesztését teszteljük. A végleges számlaszerkesztő programot a harmadik modulban készítjük el.

Fejléc szerkesztése - “FormView”



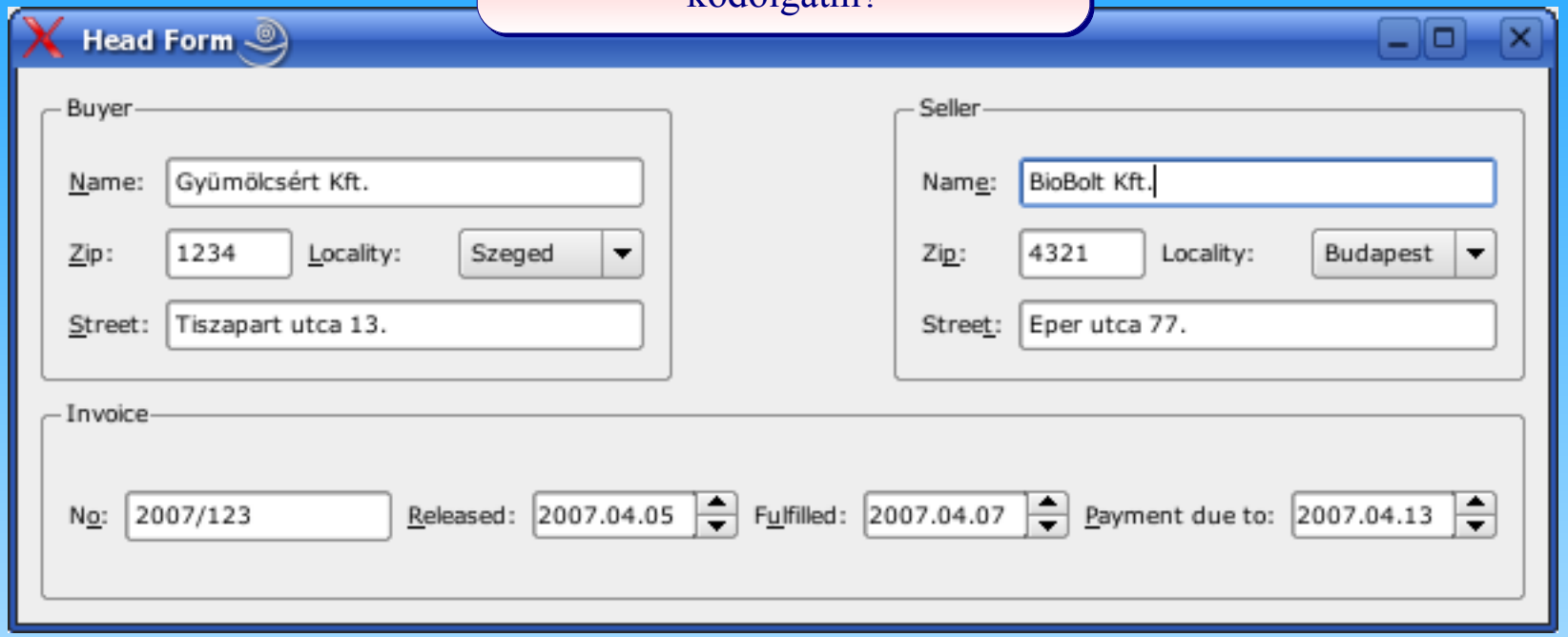
A modul második részében készítünk egy olyan (egyszerűbb) alkalmazást, amelyben csak a számla fejlécét lehet szerkeszteni.

Számla fejléc adatbeviteli mezői



Számla fejléc adatbeviteli mezői

Valóban minden adatbeviteli mezőt „egyedileg” kell kódolgatni?



buyer_zip (QLineEdit)

buyer_street (QLineEdit)

seller_street (QLineEdit)

buyer_name (QLineEdit)

invNo (QLineEdit)

released (QDateEdit)

fulfilled (QDateEdit)

dueTo QDateEdit)

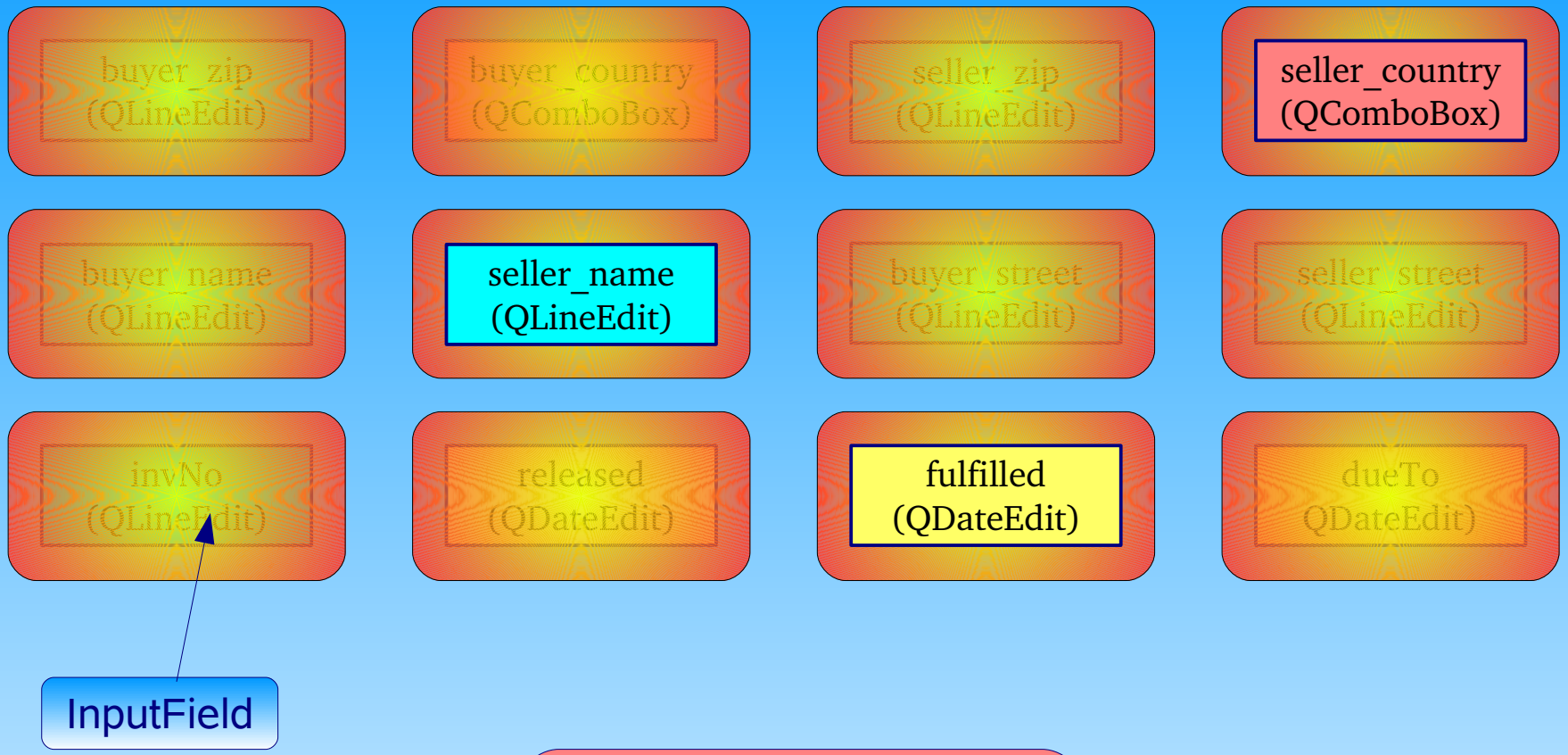
seller_country (QComboBox)

buyer_country (QComboBox)

seller_name (QLineEdit)

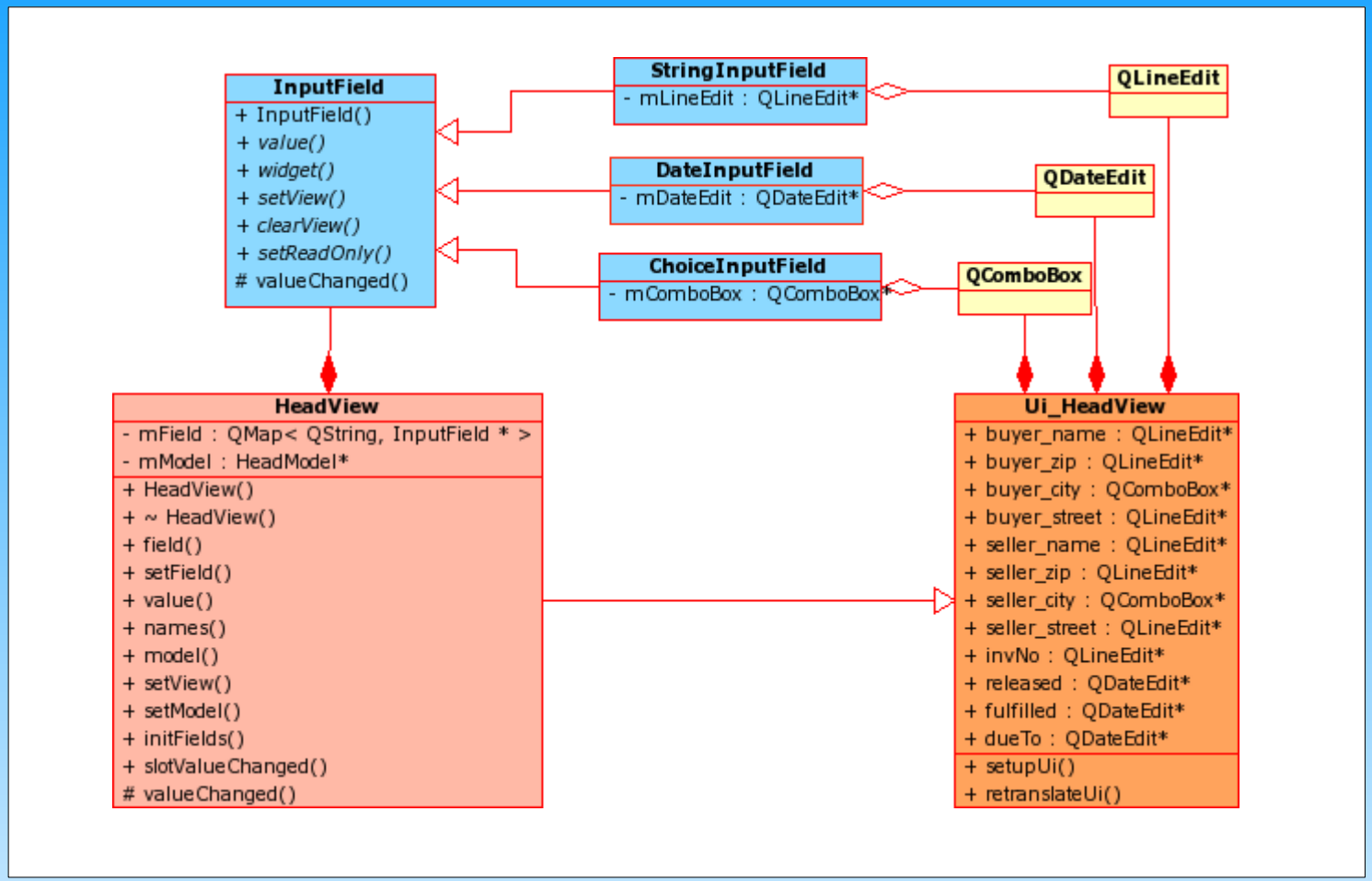
seller_zip (QLineEdit)

Számla fejléc adatbeviteli mezői



A számla fejlécén szereplő mezők egységes kezelésére bevezetünk egy absztrakt osztályt, és abból származtatjuk az egyedi osztályokat.

Adatbeviteli mezők egységes kezelése



1. A vezérlőket tervezővel helyezzük el az űrlapra.
2. Az objektumokat a tervezőben használt mezőnevekkel azonosítjuk.
3. A nézet osztályban a mezőket egységes formában (InputField) kezeljük
4. A mezőket mező nevekkel indexelhető tömbben (QMap) tároljuk.

InputField: absztrakt osztály

inputfield.h

```
#include <QVariant>
#include <QString>
#include <QObject>

class InputField: public QObject {
   Q_OBJECT

public:

   InputField(QString name, QObject* parent=0);
   virtual QVariant value() const =0;
   virtual QWidget* widget() const = 0;

public slots:
   virtual void setView(QVariant newValue) = 0;
   virtual void clearView() = 0;
   virtual void setReadOnly(bool) = 0;

signals:
   void valueChanged(QVariant val);

};
```

StringInputField: definíció

stringinputfield.h

```
#include "inputfield.h"

class QLineEdit;

class StringInputField : public InputField
{
   Q_OBJECT
public:
   StringInputField(QLineEdit* lineEdit,
         QString name, QWidget* parent = 0);
   QVariant value() const ;
   QWidget* widget() const ;
public slots:
   void setReadOnly(bool v);
   void setView(QVariant qv);
   void clearView();
   void slotTextChanged(const QString&);

private:
   QLineEdit *mLineEdit;

};
```

stringinputfield.cpp

```
StringInputField::StringInputField(QLineEdit* lineEdit,
            QString name, QWidget* parent)
      : InputField(name, parent), mLineEdit(lineEdit)
{
   connect(mLineEdit,SIGNAL(textChanged(const QString&)),
            this,SLOT(slotTextChanged(const QString&)));
}

void StringInputField::slotTextChanged(const QString& text)
{
   emit valueChanged(QVariant(text));
}

void StringInputField::setReadOnly(bool v) {
      mLineEdit->setReadOnly(v);
}

QVariant StringInputField::value()  const {
      return QVariant(mLineEdit->text());
}

void StringInputField::setView(QVariant qv) {
      mLineEdit->setText(qv.toString());
}
```

DateInputField: definíció

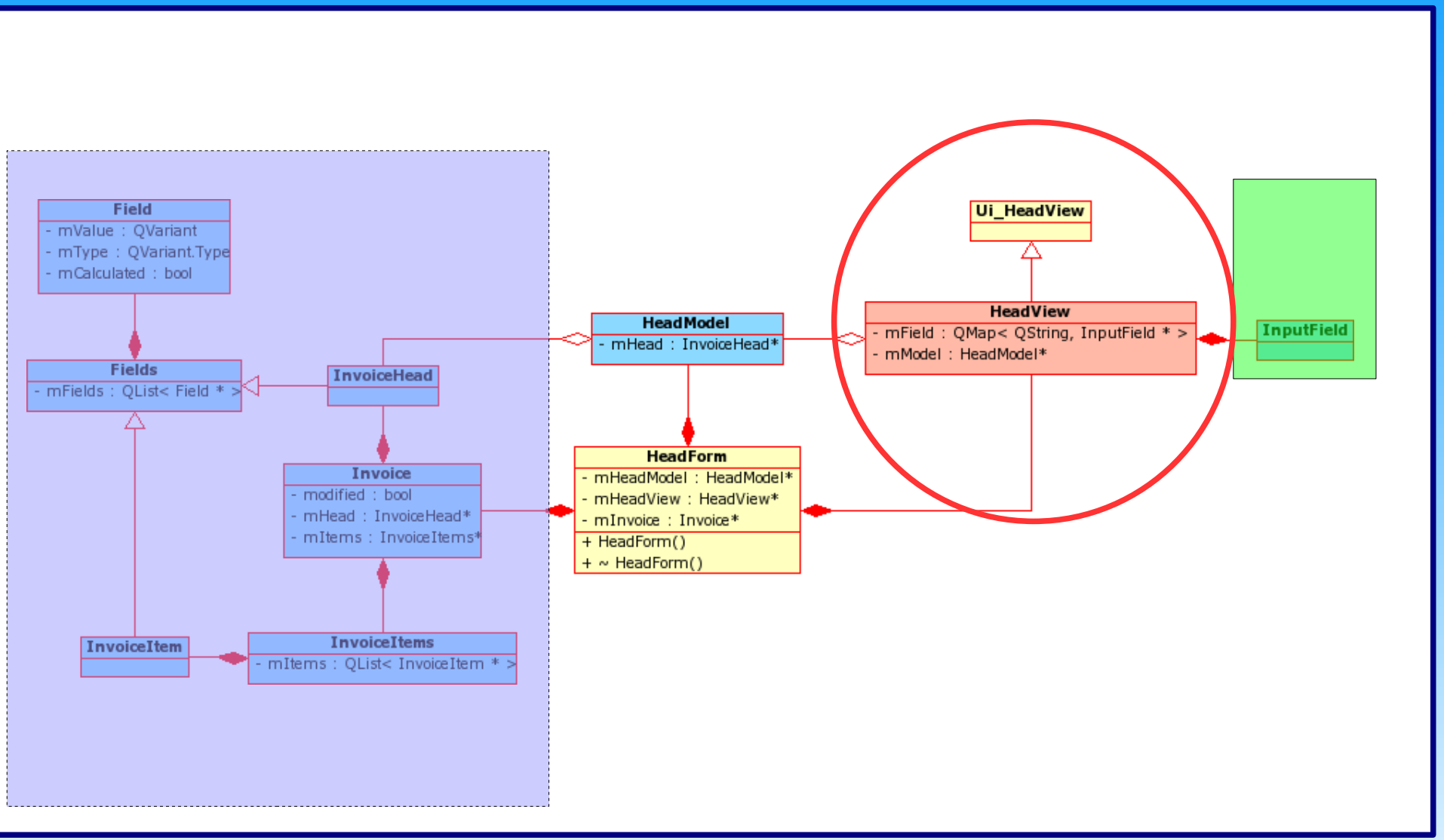
dateinputfield.h

```
#include "inputfield.h"

class QDateEdit;

class DateInputField : public InputField {
    Q_OBJECT
public:
    DateInputField(QDateEdit* dateEdit,
                   QString name, QWidget* parent=0);
    QVariant value() const ;

    QWidget* widget() const ;
public slots:
    void setReadOnly(bool v);
    void setView(QVariant qv);
    void clearView();
    void slotDateChanged(const QDate & date);

private:
    QDateEdit* mDateEdit;

};
```

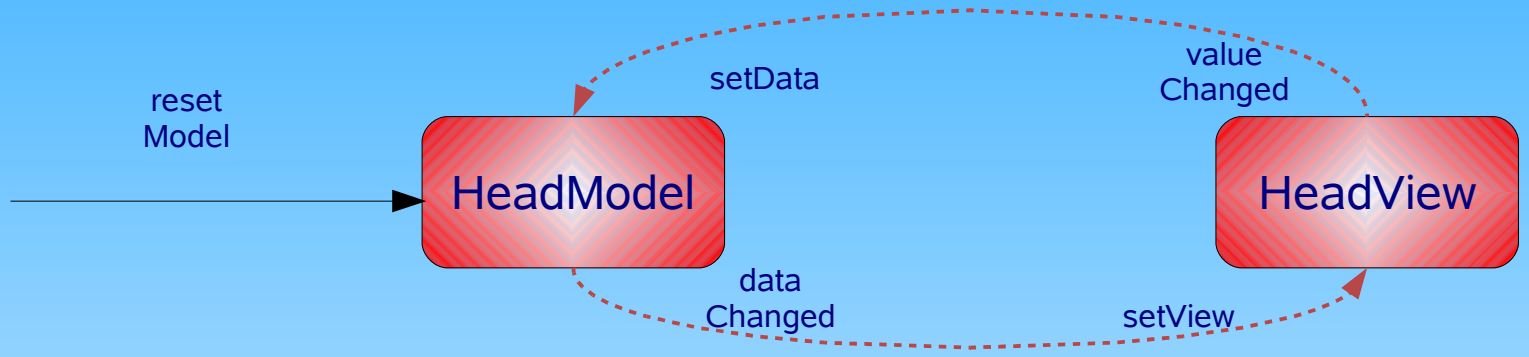
dateinputfield.cpp

```
DateInputField::DateInputField(QDateEdit* dateEdit,
                   QString name ,QWidget* parent)
        : InputField(name, parent), mDateEdit(dateEdit)
{
    mDateEdit->setDisplayFormat("yyyy.MM.dd");
    connect(mDateEdit,SIGNAL(dateChanged(const QDate &)),
                this,SLOT(slotDateChanged(const QDate &)));
}
void DateInputField::slotDateChanged(const QDate & date){
        emit valueChanged(QVariant(date));
}
void DateInputField::setReadOnly(bool v) {
        mDateEdit->setReadOnly(v);
}
QVariant DateInputField::value() const {
        return QVariant(mDateEdit->date());
}
void DateInputField::setView(QVariant qv) {
        mDateEdit->setDate(qv.toDate());
}
void DateInputField::clearView() {
        mDateEdit->setDate(QDate::currentDate());
}
QWidget* DateInputField::widget() const {
        return mDateEdit;
}
```

Fejléc kezelő program osztálydiagramja

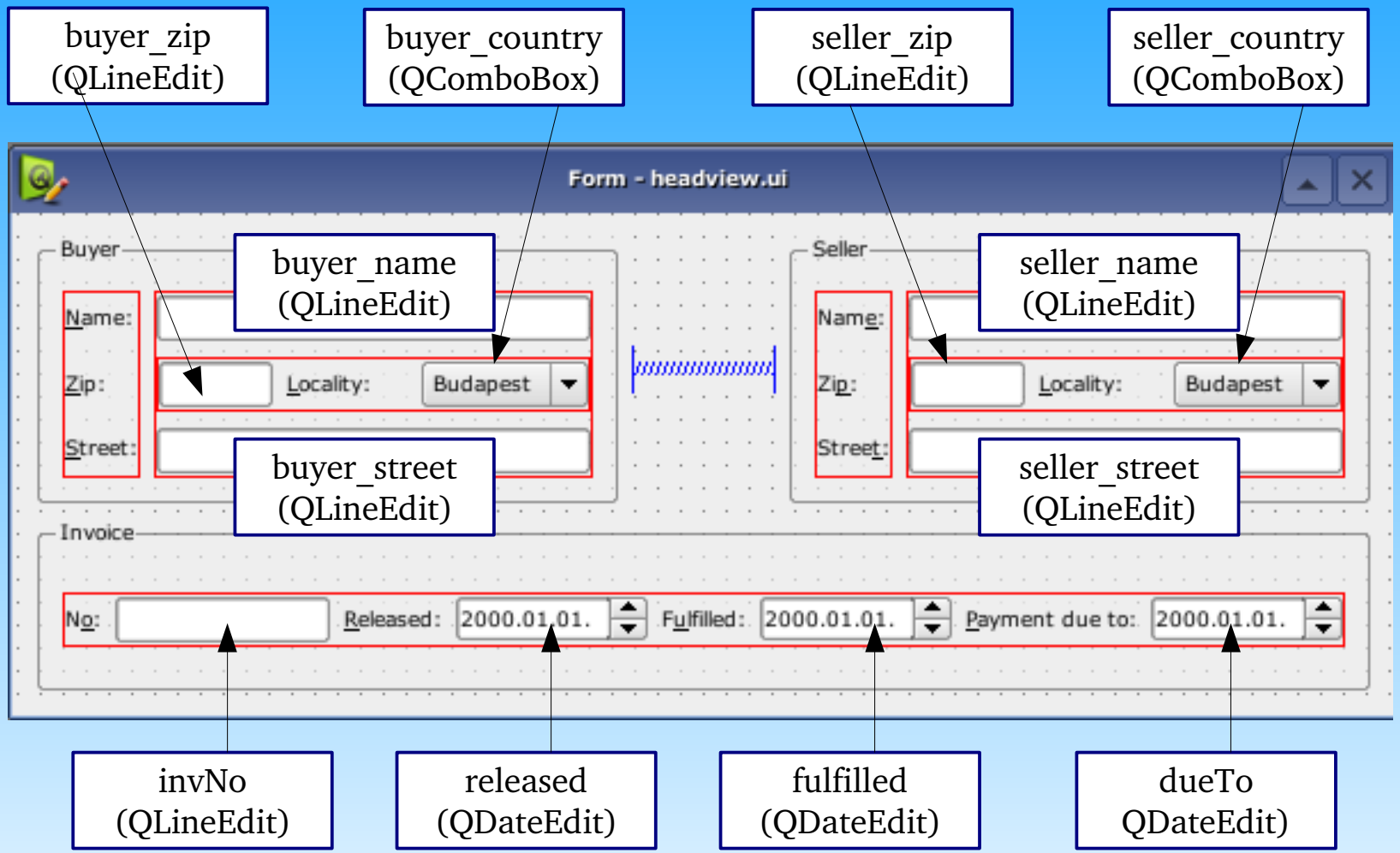


Objektumok együttműködése



Az objektumok közötti kommunikációt a Qt jeladó/jelvevő (signal/slot) mechanizmusával oldjuk meg.
jeladók: valueChanged(), dataChanged()
jelvevők: setData(), setView()

Számla fejléc megtervezése Designerrel



HeadView: definíció

headview.h

```
class HeadView : public QWidget, private Ui_HeadView
{
        Q_OBJECT
public:
        HeadView(QWidget *parent = 0);
        ~HeadView();

        InputField* field(const QString& name) {return mField[name];}
        void setField(const QString& name, InputField* field){mField[name] = field;}
        QVariant value(const QString& fieldName);
        QStringList names();

        HeadModel* model() {return mModel;}
         public slots:
        void setView(const QString& fieldName, QVariant newValue);
        void setModel(HeadModel* model);
        void initFields();
        void slotValueChanged(QVariant val);

signals:
        void valueChanged(const QString& fieldName,QVariant val);

private:
        QMap<QString, InputField*> mField;
        HeadModel* mModel;

};
```

A fejlécen található adatbeviteli mezőket (mezőkre mutató pointereket) a mezőnévvel indexelhető QMap gyüjteményben tároljuk.

HeadView: implementáció

```
HeadView::HeadView(QWidget *parent) : QWidget(parent)
{
	setupUi(this);
	initFields();
}

HeadView::~HeadView(){
	foreach(QString key, mField.keys())
		delete mField[key];
	mField.clear();
}

void HeadView::slotValueChanged(QVariant value){
	emit valueChanged(QObject::sender()->objectName(), value);
}

QStringList HeadView::names(){
	QStringList ret;
	foreach(QString key, mField.keys())
		ret << mField[key]->objectName();
	return ret;
}

void HeadView::setView(const QString& fieldName, QVariant value){
	if(field(fieldName)->value() == value) return;
	field(fieldName)->setView(value);
	emit valueChanged(fieldName, field(fieldName)->value());
}
```

```
QVariant HeadView::value(const QString& fieldName)
{
	return field(fieldName)->value();
}

void HeadView::setModel(HeadModel* model){
	if(!model) return;
	mModel = model;
}
```

Értesít egy adatbeviteli mező megváltozásáról. Továbbadja a megváltozott mező nevét és a módosított értéket.

Visszaadja a fejlécben található mezőneveket.

Megváltoztatja a mező értékét a paraméterben kapott értékre (ha más, mint a régi érték) és erről értesítést küld.

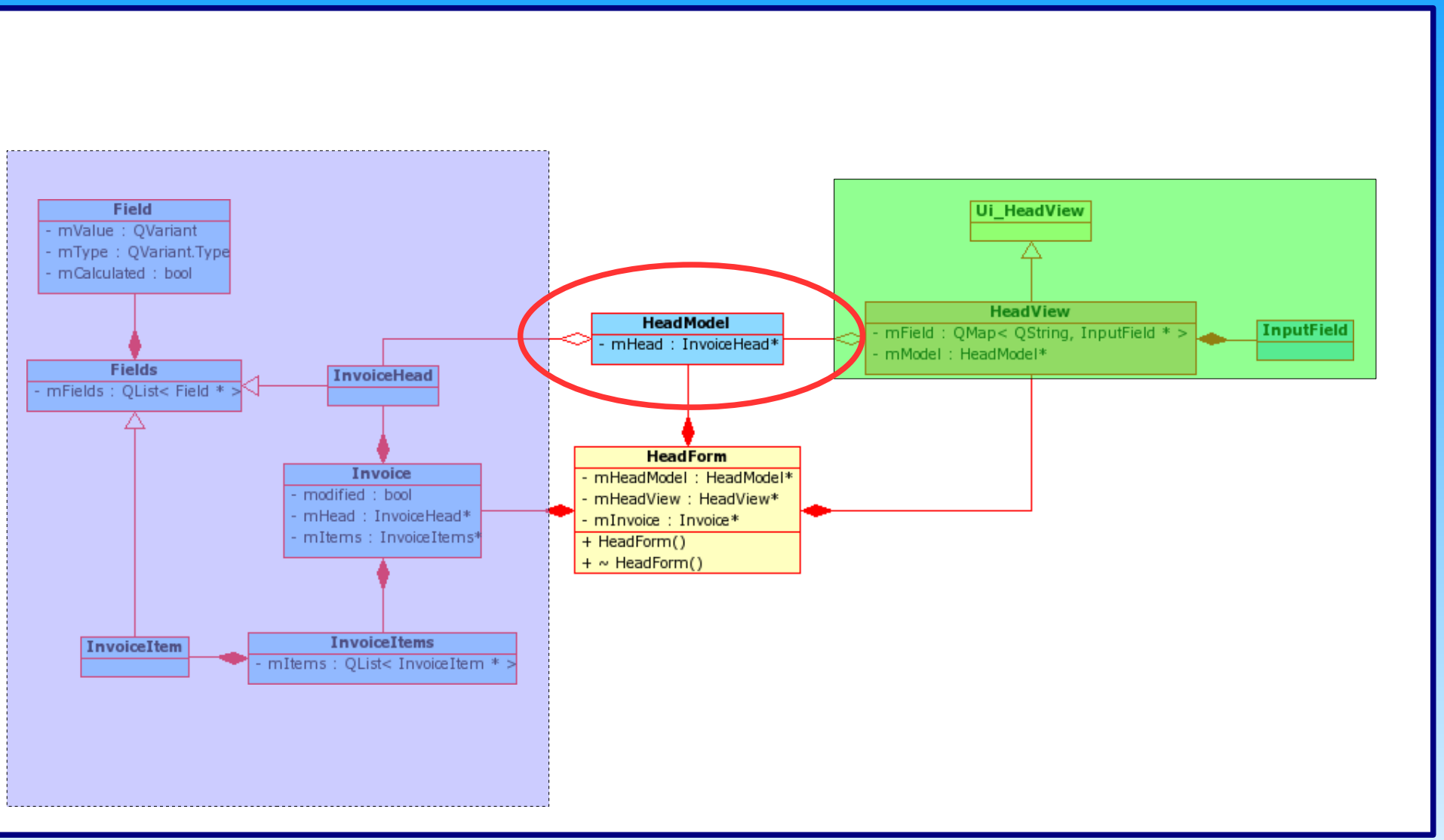
HeadView: implementáció

headview.cpp

```
void HeadView::initFields()
{
    //init QLineEdit fields
    QList<QLineEdit *> lineEditList = findChildren<QLineEdit *>();
    foreach(QLineEdit* inputField, lineEditList)
    {
        mField[inputField->objectName()] = new StringInputField(inputField,inputField->objectName(),this);
        connect(mField[inputField->objectName()],SIGNAL(valueChanged(QVariant)),
                this,SLOT(slotValueChanged(QVariant)));
    }
    //init QDateEdit fields
    QList<QDateEdit *> dateEditList = findChildren<QDateEdit *>();
    foreach(QDateEdit * inputField, dateEditList)
    {
        mField[inputField->objectName()] = new DateInputField(inputField,inputField->objectName(),this);
        connect(mField[inputField->objectName()],SIGNAL(valueChanged(QVariant)),
                this,SLOT(slotValueChanged(QVariant)));
    }
    //init QComboBox fields
    QList<QComboBox *> comboBoxList = findChildren<QComboBox *>();
    foreach(QComboBox * inputField, comboBoxList)
    {
        mField[inputField->objectName()] = new ChoiceInputField(inputField,inputField->objectName(),this);
        connect(mField[inputField->objectName()],SIGNAL(valueChanged(QVariant)),
                this,SLOT(slotValueChanged(QVariant)));
    }
}
```

Az mField QMap típusú gyüjteményben elhelyezzük az űrlapon található QLineEdit, QDateEdit és QComboBox típusú adatbeviteli mezőkre mutató pointereket és a mezőneveket.

A fejléc kezelő program osztálydiagramja



HeadModel: definíció

headmodel.h

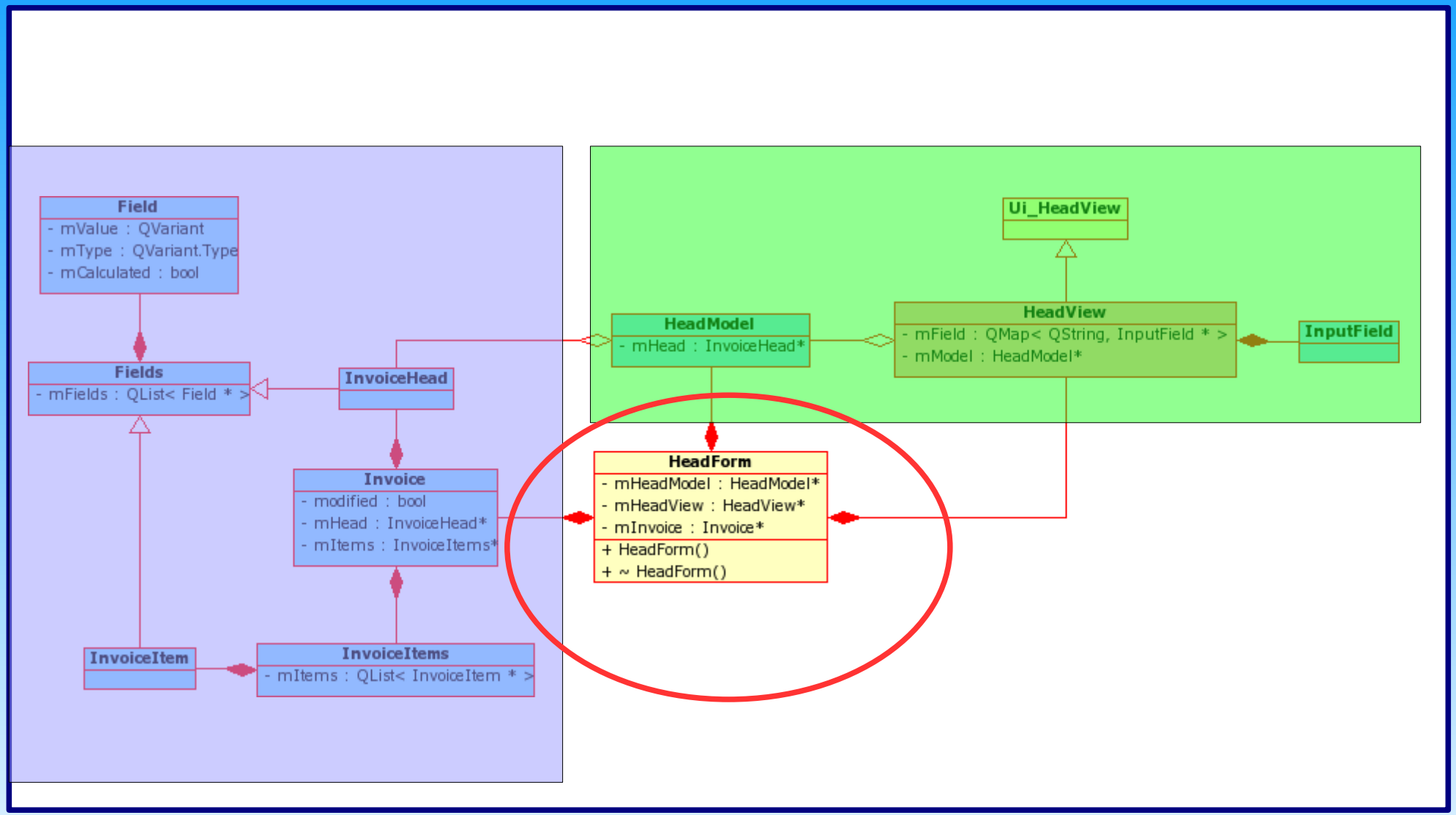
```
class HeadModel : public QObject {
    Q_OBJECT

public:
    HeadModel(QObject *parent=0);
    ~HeadModel() {}

    QVariant data(const QString& name);
    void setHead(InvoiceHead* head);
    QString toString();

public slots:
    void setData(const QString& name, QVariant data);
    void resetData(const QString& name, QVariant data);
    void resetModel();

signals:
    void dataChanged(const QString& name, QVariant data);
    void modelModified();

private:
    InvoiceHead* mHead;

};
```

HeadModel: implementáció

headmodel.cpp

```
QVariant HeadModel::data(const QString& name)
{
	return mHead->field(name)->value();
}
void HeadModel::setData(const QString& name, QVariant data)
{
	if(mHead->field(name)->value() == data) return;
	mHead->field(name)->setValue(data);
	emit dataChanged(name,data);
	emit modelModified();
}
void HeadModel::setHead(InvoiceHead* head)
{
	mHead = head;
	resetModel();
}
void HeadModel::resetData(const QString& name, QVariant data)
{
	emit dataChanged(name,data);
	emit modelModified();

}
void HeadModel::resetModel()
{
	for (int i = 0; i < mHead->count(); ++i)
		resetData(mHead->field(i)->name(),mHead->field(i)->value());
}
```

Fejléc kezelő program osztálydiagramja



HeadForm osztály és főprogram

headform.h

```
class HeadForm : public QWidget {
        Q_OBJECT

public:
        HeadForm(Invoice* invoice,
                        QWidget *parent = 0 );
        ~HeadForm(){};

private:
        HeadModel* mHeadModel;
        HeadView* mHeadView;
        Invoice* mInvoice;

};
```

headform.cpp

```
HeadForm::HeadForm(Invoice* invoice, QWidget *parent )
        : QWidget(parent), mInvoice(invoice)
{
        setWindowTitle("Head Form");

        mHeadModel = new HeadModel(this);
        mHeadModel->setHead(mInvoice->head());
        mHeadView = new HeadView(this);
        mHeadView->setModel(mHeadModel);
        mHeadModel->resetModel();
}
```

main.cpp

```
int main (int argc, char *argv[])
{
        QApplication app(argc,argv);
        Invoice* invoice = new Invoice();
        invoice->load("./invoices/2007/invoice_2007_123.inv");

        HeadForm *headForm = new HeadForm(invoice);
        headForm->show();

        bool ret = app.exec();
        invoice->saveAs("./invoices/2007/invoice_2007_567.inv");
        return ret;
}
```

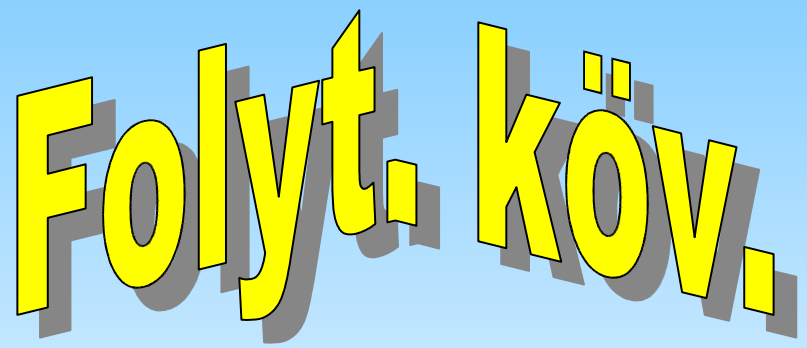
A bemutatott programok megtalálhatók a

people.inf.elte.nacsa/qt4/eaf3/inv02/projects/**modelview_items**

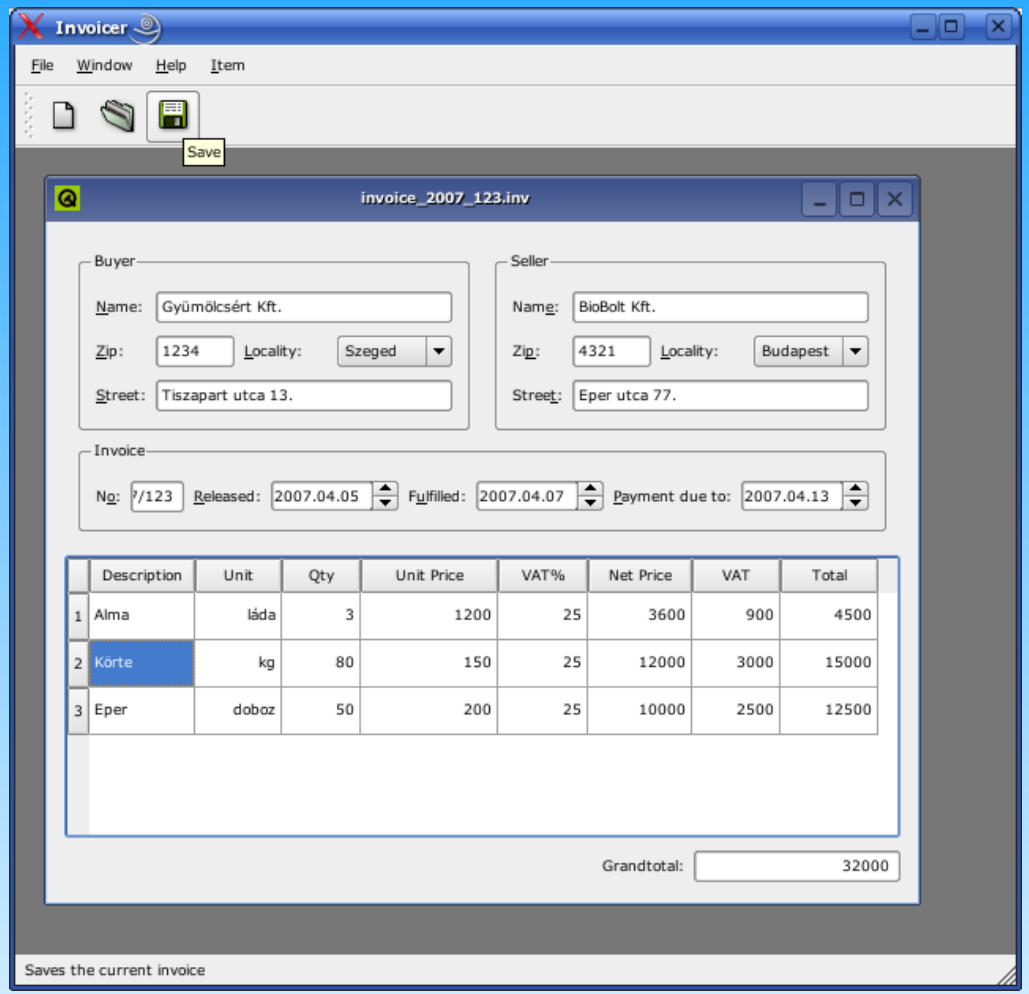
és a

people.inf.elte.nacsa/qt4/eaf3/inv02/projects/**modelview_head**

címen.



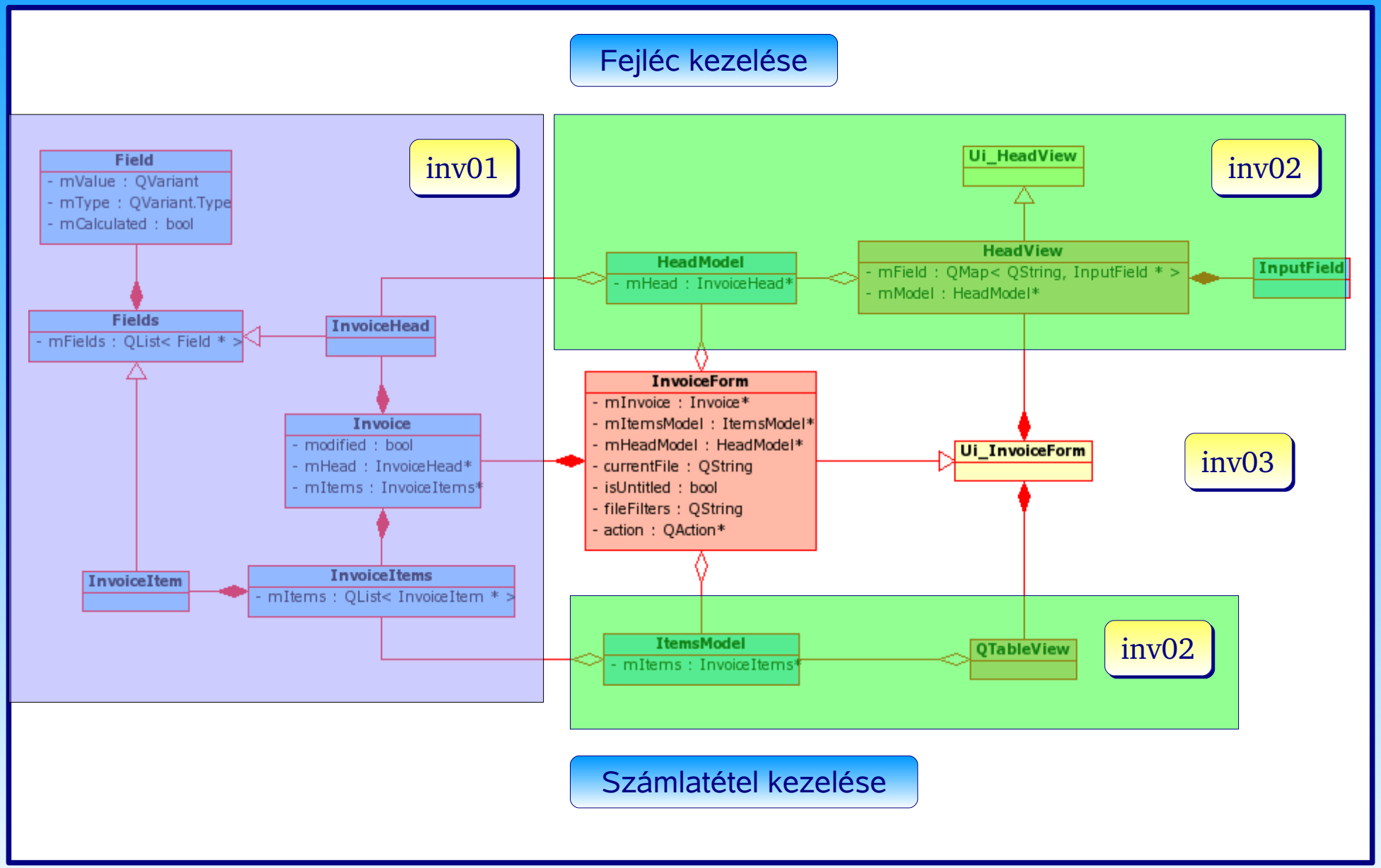
Elemi alkalmazások fejlesztése III.



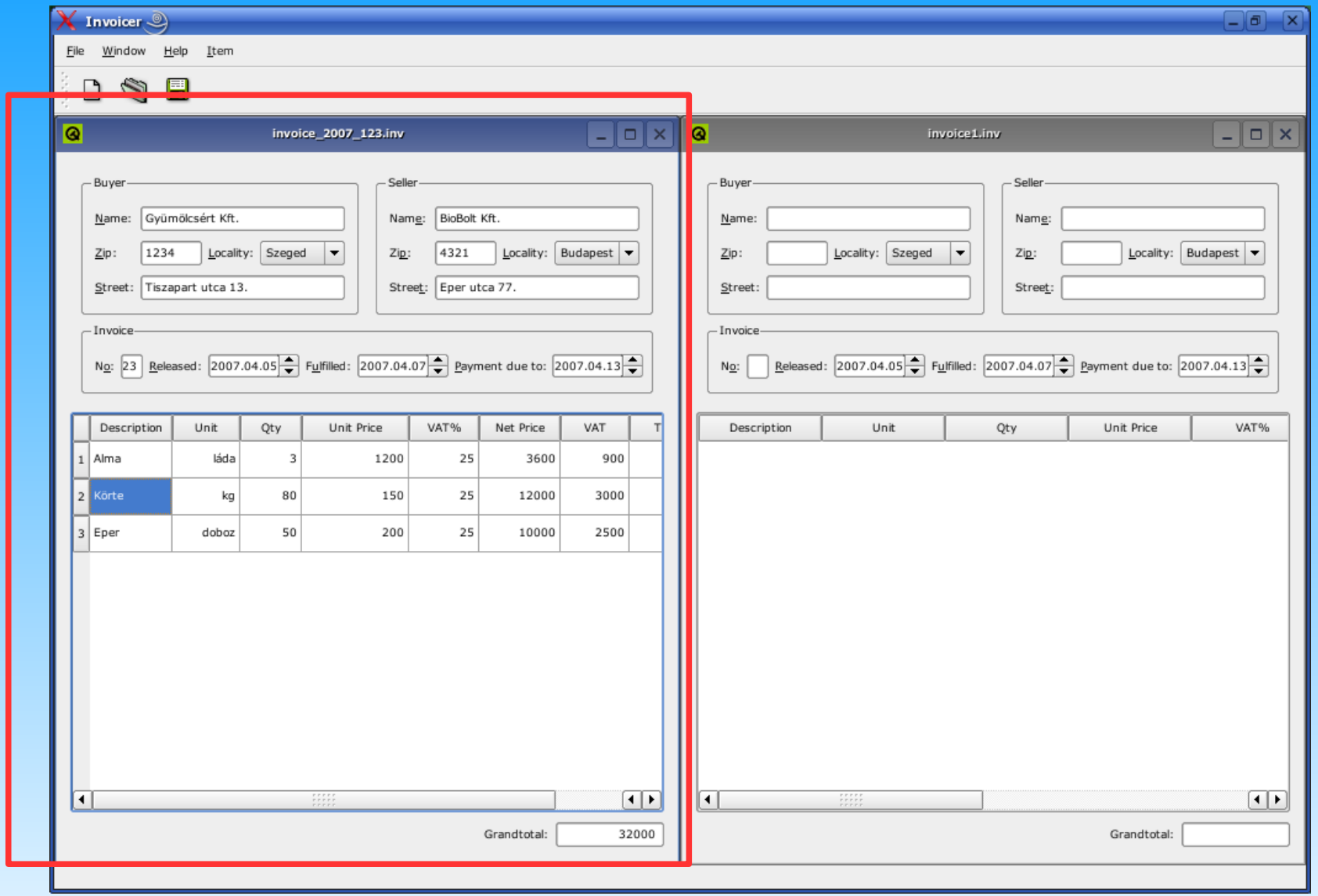
MDI alkalmazás III.
(Számla)

Készítette: Szabóné Nacsa Rozália
nacsa@inf.elte.hu
people.inf.elte.hu/nacsa/qt4/eaf3/
Qt 4
2007

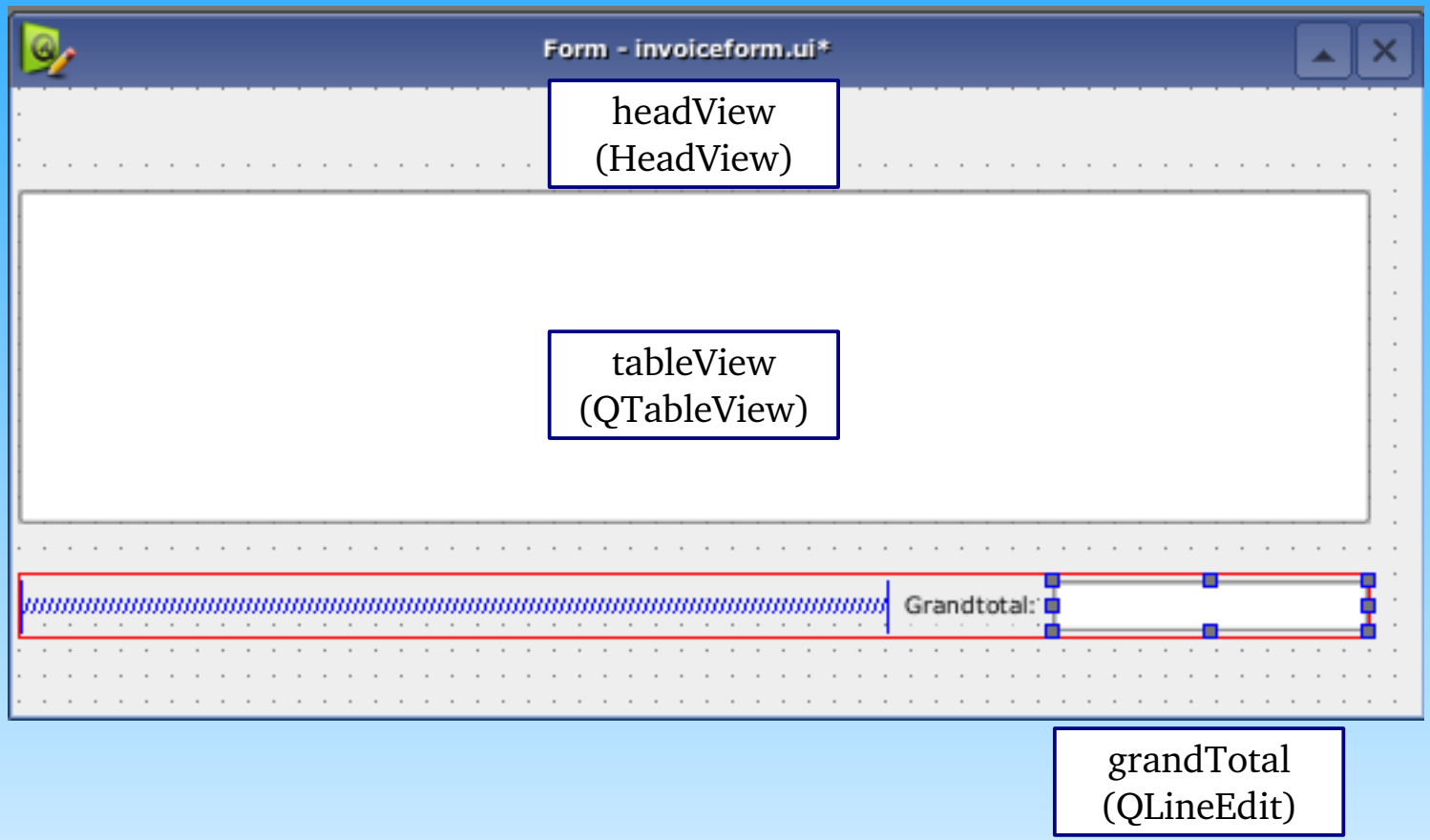
Az alkalmazás osztálydiagramja



Számlakezelő űrlap



InvoiceForm (számlakezelő űrlap) tervezése



InvoiceForm: definíció

```
class InvoiceForm : public QWidget, public Ui_InvoiceForm
{
    Q_OBJECT
public:
    InvoiceForm(QWidget *parent=0);
    ~InvoiceForm();

protected:
    void closeEvent(QCloseEvent *event);

public:
    void newFile();
    bool open();
    bool openFile(const QString &fileName);
    bool save();
    bool saveAs();

    QAction* windowMenuAction() const {return action;}
    int itemCount();

private:
    bool okToContinue();
    bool saveFile(const QString &fileName);
    void setCurrentFile(const QString &fileName);
    QString strippedName(const QString &fullFileName);
```

Az űrlap bezárásakor lefutó eseménykezelő.

invoiceform.h

```
private slots:
    void documentWasModified();

public slots:
    void insertItemBefore();
    void insertItemAfter();
    void deleteItem();
    void updateGrandTotal();

private:
    Invoice* mInvoice;
    ItemsModel* mItemsModel;
    HeadModel* mHeadModel;

    QString currentFile;
    bool isUntitled;
    QString fileFilters;
    QAction* action;

};
```

InvoiceForm: implementáció

```
InvoiceForm::InvoiceForm(QWidget *parent)  : QWidget(parent)
{
      setupUi(this);  //Setup GUI elements created by designer

      action = new QAction(this);
      action->setCheckable(true);
      connect(action, SIGNAL(triggered()), this, SLOT(show()));
      connect(action, SIGNAL(triggered()), this, SLOT(setFocus()));

      setWindowIcon(QPixmap(":/images/document.png"));
      setAttribute(Qt::WA_DeleteOnClose);
      setWindowTitle(strippedName(currentFile) + "[*]");
      isUntitled = true;
      fileFilters =  tr("Invoice files (*.inv);; Any files (*)");

      mInvoice=new Invoice(this);

      mItemsModel = new ItemsModel(this);
      mItemsModel->setItems(mInvoice->items());
      tableView->setModel(mItemsModel);

      mHeadModel = new HeadModel(this);
      mHeadModel->setHead(mInvoice->head());
      headView->setModel(mHeadModel);
      . . .

      mInvoice->newInvoice();
      setWindowModified(false);
}
```

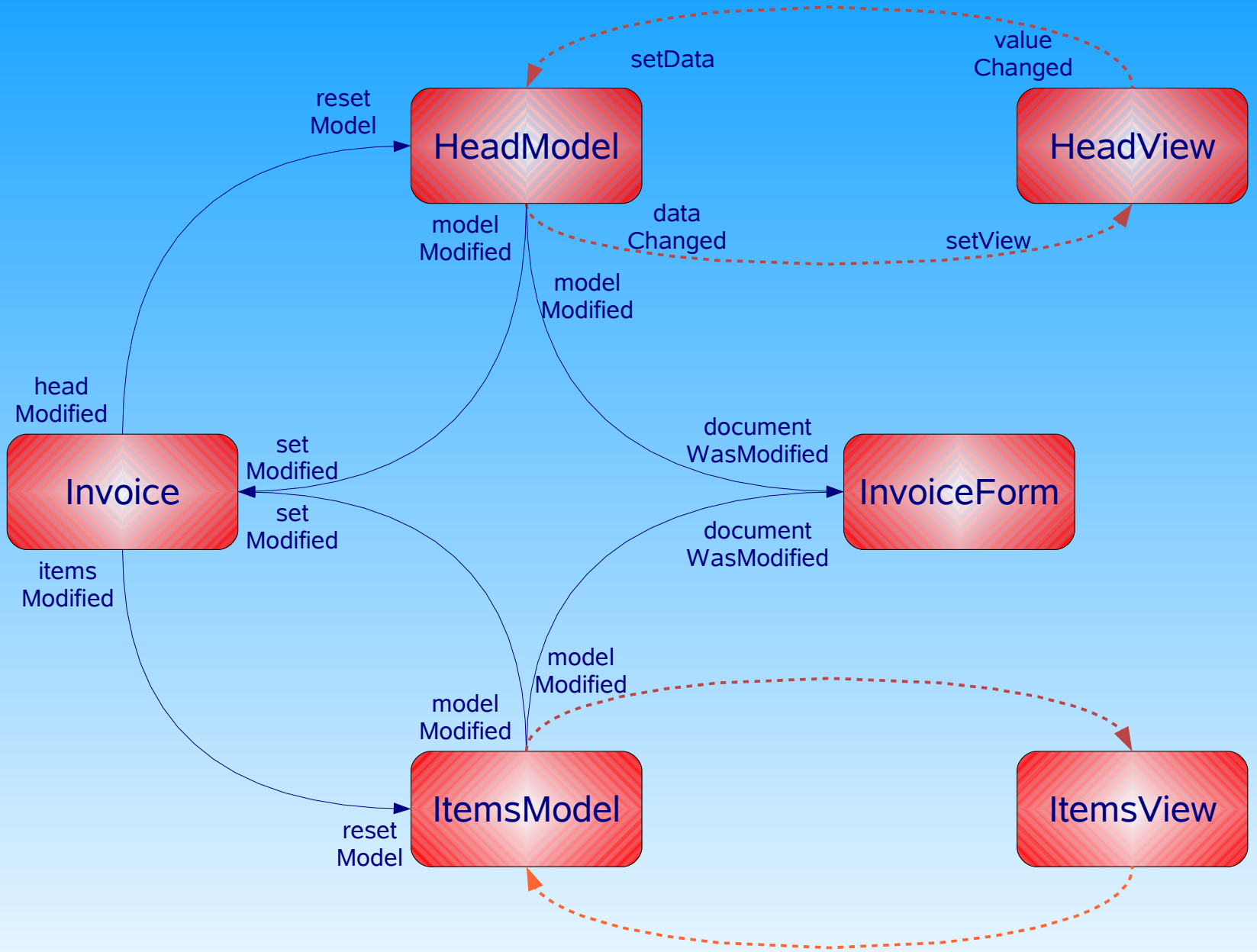
invoiceform.cpp

Számlaételek kezelése

Fejléc kezelése

Jelek, jelkezelők összekapcsolása (l. köv. dia)

Objektumok együttműködése



InvoiceForm: implementáció

invoiceform.cpp

```
InvoiceForm::InvoiceForm(QWidget *parent)
    : QWidget(parent)
{
    . . .
    connect(mInvoice, SIGNAL(itemsModified()),mItemsModel, SLOT(resetModel()));
    connect(mInvoice, SIGNAL(headModified()),mHeadModel, SLOT(resetModel()));

    connect(mHeadModel, SIGNAL(modelModified()),this, SLOT(documentWasModified()));
    connect(mItemsModel, SIGNAL(modelModified()),this, SLOT(documentWasModified()));

    connect(mHeadModel, SIGNAL(modelModified()), mInvoice, SLOT(setModified()));
    connect(mItemsModel, SIGNAL(modelModified()), mInvoice, SLOT(setModified()));

    connect(mItemsModel, SIGNAL(modelModified()),this, SLOT(updateGrandTotal()));

    . . .
}
```

InvoiceForm: implementáció

invoiceform.cpp

```
void InvoiceForm::deleteItem()
{
	QModelIndex index = tableView->currentIndex();
	if(!index.isValid()) return;

	mItemsModel->removeInvoiceItem(index);

	int nextRow=(index.row()<mItemsModel->rowCount())? index.row():mItemsModel->rowCount()-1;
	if(nextRow >= 0)
		tableView->setCurrentIndex(mItemsModel->index(nextRow,index.column()));
}

void InvoiceForm::insertItemBefore()
{
	QModelIndex index = tableView->currentIndex();
	mItemsModel->addInvoiceItem(index);
	if (mItemsModel->rowCount() == 1)
		index = mItemsModel->index(0,0);
	tableView->setCurrentIndex(index);
	tableView->edit(index);
}
```

InvoiceForm: implementáció

invoiceform.cpp

```
bool InvoiceForm::save()
{
        if(isUntitled)          return saveAs();
        else                    return saveFile(currentFile);
}

bool InvoiceForm::saveAs()
{
        QString fileName = QFileDialog::getSaveFileName(this, tr("Save Invoice"),
                                        Utils::getApplicationDirPrefix() + "invoices/" +
                                        QDate::currentDate().toString("yyyy"), fileFilters);
        if (fileName.isEmpty()) return false;
        if (!fileName.contains("."))    return saveFile(fileName + ".inv");
        else                            return saveFile(fileName);

}

bool InvoiceForm::saveFile(const QString &fileName)
{
        if(!mInvoice->saveAs(fileName)) return false;
        setCurrentFile(fileName);
        return true;
}
```

InvoiceForm: implementáció

invoiceform.cpp

```
void InvoiceForm::newFile()
{
	static int documentNumber = 1;
	currentFile = tr("invoice%1.inv").arg(documentNumber);
	setWindowTitle(currentFile + "[*]");
	action->setText(currentFile);
	isUntitled=true;
	++documentNumber;
}

bool InvoiceForm::open()
{
	QString fileName =
		QFileDialog::getOpenFileName(this, tr("Open Invoice"),
					Utils::getApplicationDirPrefix() + "invoices/" +
					QDate::currentDate().toString("yyyy"),fileFilters);
		if (fileName.isEmpty())
			return false;
		return openFile(fileName);
}

bool InvoiceForm::openFile(const QString &fileName)
{
	if(!mInvoice->load(fileName))
		return false;
	setCurrentFile(fileName);
	tableView->update();
	return true;
}
```

InvoiceForm: implementáció

invoiceform.cpp

```
void InvoiceForm::closeEvent(QCloseEvent *event)
{
	if(okToContinue())
		event->accept();
	else
		event->ignore();
}

bool InvoiceForm::okToContinue()
{
	if(isWindowModified()) {
		int ans = QMessageBox::warning(this, tr("Invoice"),
				tr("Invoice \"") + strippedName(currentFile)
					+ tr("\" has been modified.\n Do you want to save your changes?"),
				QMessageBox::Yes | QMessageBox::Default,
				QMessageBox::No,
				QMessageBox::Cancel|QMessageBox::Escape);
		if( ans == QMessageBox::Yes)
			return save();
		else if (ans == QMessageBox::Cancel)
			return false;
	}
	return true;
}
```

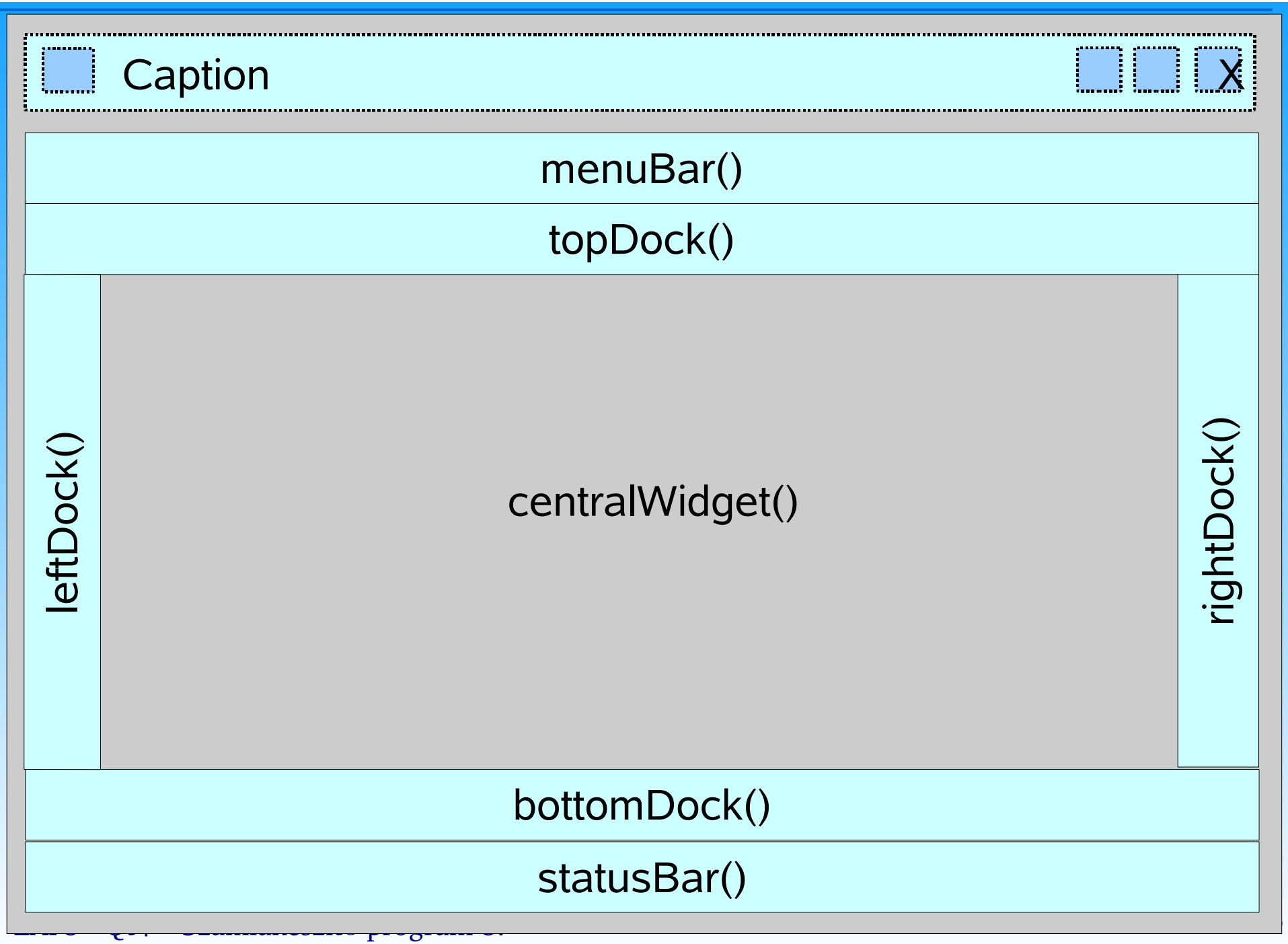
InvoiceForm: implementáció

```
void InvoiceForm::setCurrentFile(const QString &fileName)
{
        currentFile = fileName;
        isUntitled = false;
        action->setText(strippedName(currentFile));
        mInvoice->setModified(false);
        setWindowTitle(strippedName(currentFile) + "[*]");
        setWindowModified(false);
}

QString InvoiceForm::strippedName(const QString &fullFileName)
{
        return QFileInfo(fullFileName).fileName();
}

void InvoiceForm::documentWasModified()
{
        setWindowModified(true);
}
```

invoiceform.cpp

Főablak

QMainWindow osztály: ablak felépítése



QAction osztály

A felhasználói felülethez kötött tevékenységek absztrakciója. Megjelenhet menükben és eszközsoron is.

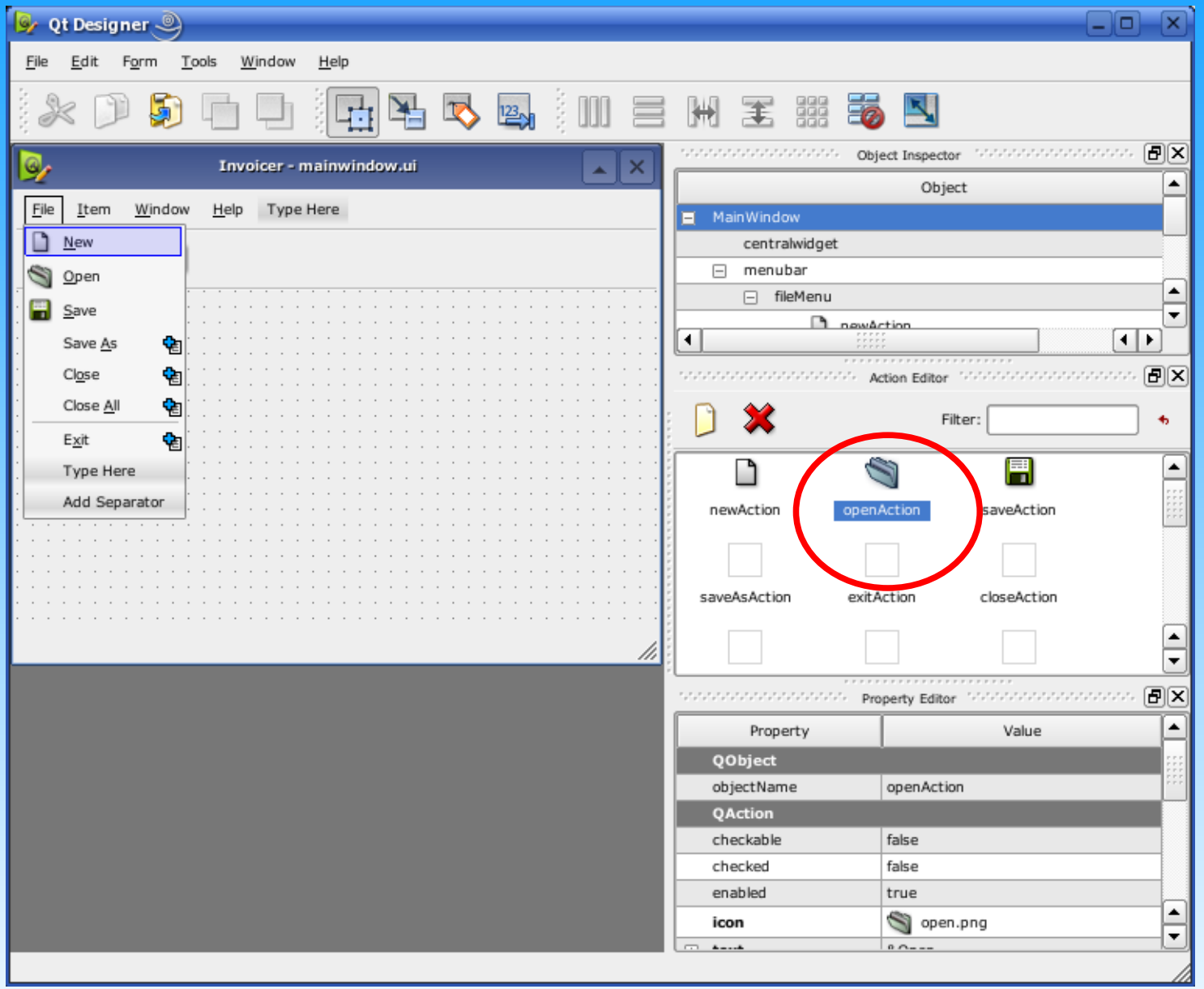
Command action

- tevékenység végrehajtására szolgál
- amikor a tevékenységet el kell végezni **triggered()** signalt küld

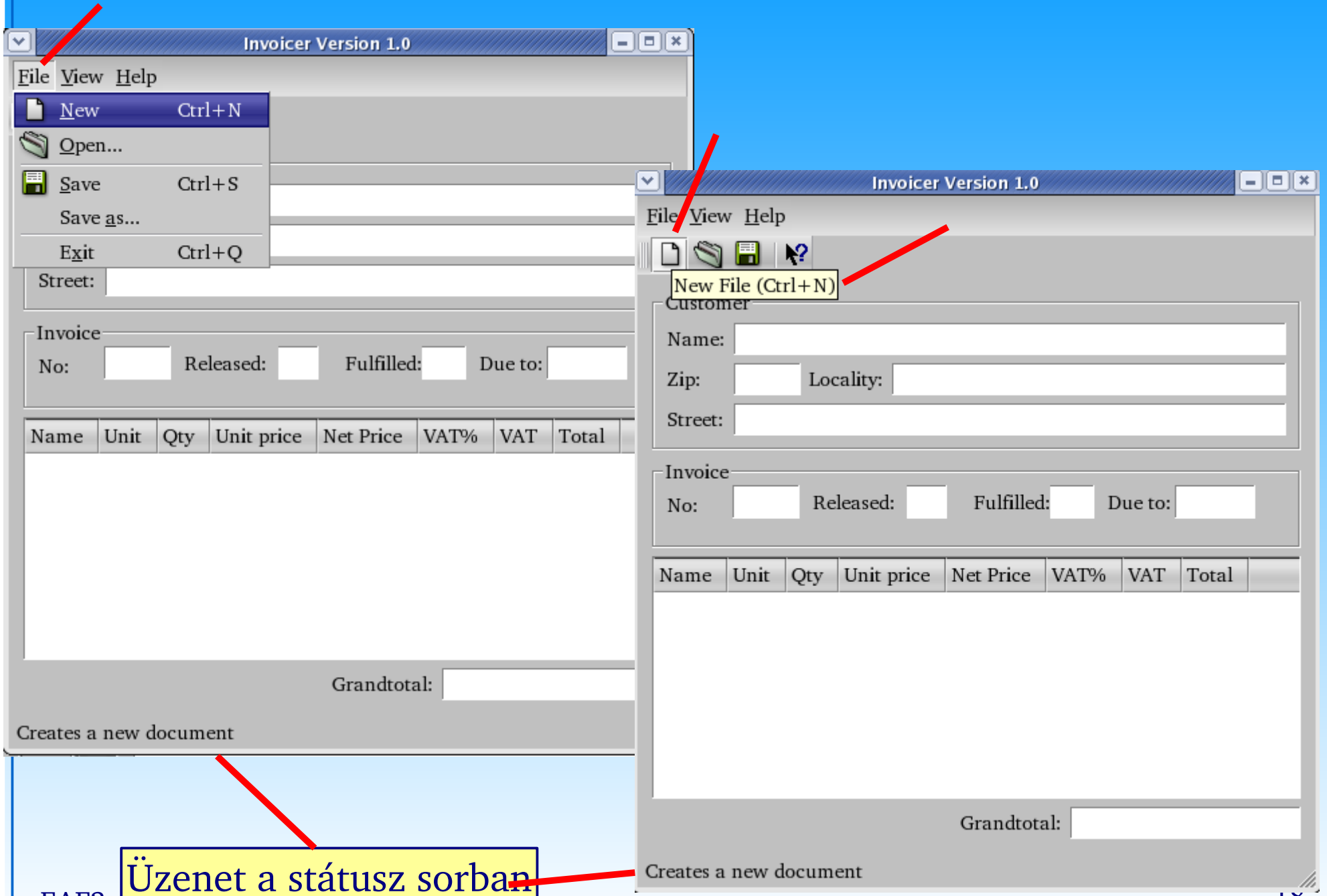
Toggle action

- flag (opció) beállítására szolgál
- az opció változásakor **toggled(bool)** signalt küld
- csoportosítható (QActionGroup)

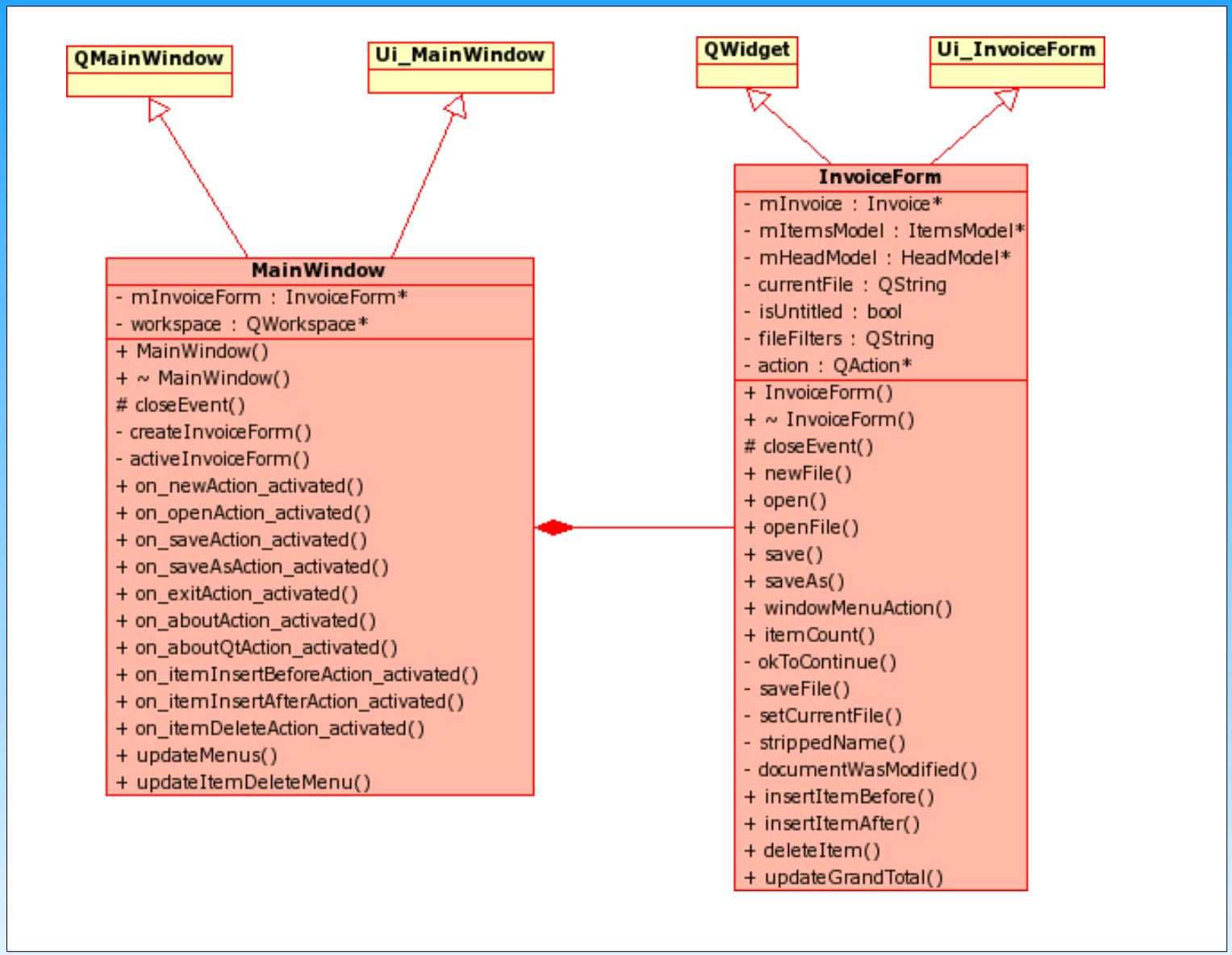
Menüpontok megtervezése Qt Designerben



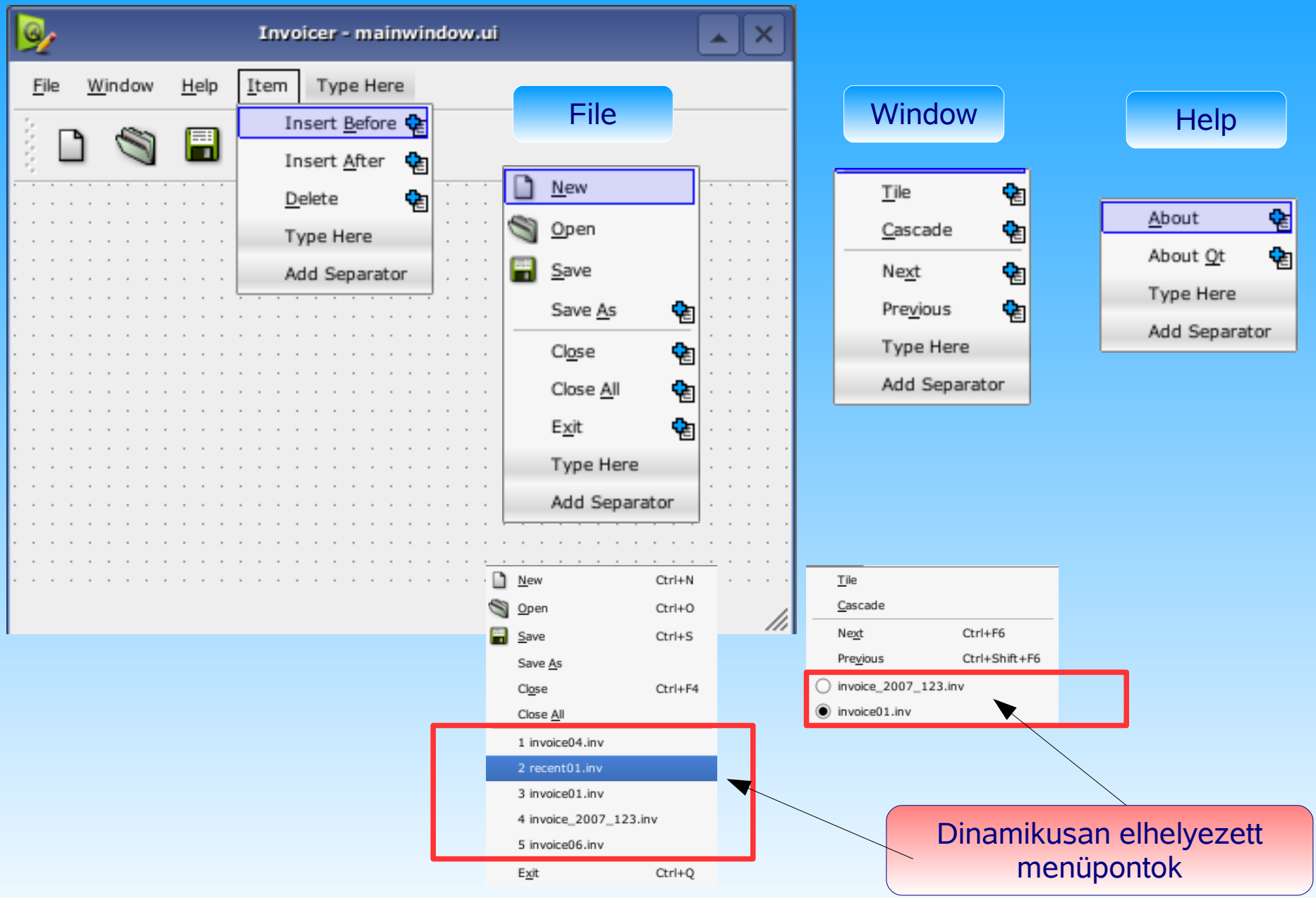
File/New “menüpont”



Osztálydiagram a főablak elkészítéséhez



Főablak elkészítése



MainWindow: definíció

mainwindow.h

```
class MainWindow : public QMainWindow, public Ui_MainWindow
{
    Q_OBJECT

 public:
    MainWindow(QMainWindow *parent = 0);
    ~MainWindow();
    void openFile(const QString &fileName);

protected:
    void closeEvent(QCloseEvent *event);

private:
    InvoiceForm* createInvoiceForm();
    InvoiceForm* activeInvoiceForm();
    void writeSettings();
    void readSettings();
```

```
private:
    InvoiceForm* mInvoiceForm;
    QWorkspace* workspace;
    QActionGroup* windowActionGroup;
    enum {MaxRecentFiles = 5};
    QAction* recentFileActions[MaxRecentFiles];
    QAction* separatorAction;
    QStringList recentFiles;
 };
```

```
private slots:
    void openRecentFile();
    void createRecentFileMenus();
    void updateRecentFileActions
        (const QString& fileName);
    void updateRecentFileActions();

public slots:
    void on_newAction_activated();
    void on_openAction_activated();
    void on_saveAction_activated();
    void on_saveAsAction_activated();
    void on_exitAction_activated();
    void on_aboutAction_activated();
    void on_aboutQtAction_activated();

    void on_itemInsertBeforeAction_activated();
    void on_itemInsertAfterAction_activated();
    void on_itemDeleteAction_activated();

    void updateMenus();
    void updateItemDeleteMenu();
```

MainWindow: implementáció

mainwindow.cpp

```
MainWindow::MainWindow(QMainWindow *parent)   : QMainWindow(parent)
{
      setupUi(this);  //Setup GUI elements created by designer

      workspace = new QWorkspace;
      setCentralWidget(workspace);

      windowActionGroup = new QActionGroup(this);
      createRecentFileMenus();

      connect(workspace,SIGNAL(windowActivated(QWidget*)),this,SLOT(updateMenus()));
      connect(closeAction, SIGNAL(triggered()),workspace, SLOT(closeActiveWindow()));
      connect(closeAllAction, SIGNAL(triggered()),workspace, SLOT(closeAllWindows()));

      connect(tileAction, SIGNAL(triggered()), workspace, SLOT(tile()));
      connect(cascadeAction, SIGNAL(triggered()), workspace, SLOT(cascade()));
      connect(nextAction, SIGNAL(triggered()),workspace, SLOT(activateNextWindow()));
      connect(previousAction, SIGNAL(triggered()),workspace, SLOT(activatePreviousWindow()));

      setWindowTitle(tr("MDI Invoicer"));
      readSettings();
      updateMenus();
}
```

Ősosztály metódusai

MainWindow: implementáció

mainwindow.cpp

```
InvoiceForm* MainWindow::createInvoiceForm()
{
    InvoiceForm* invoiceForm = new InvoiceForm;
    workspace->addWindow(invoiceForm);
    windowMenu->addAction(invoiceForm->windowMenuAction());
    windowActionGroup->addAction(invoiceForm->windowMenuAction());
    return invoiceForm;
}


void MainWindow::on_newAction_activated()
{
    statusBar()->showMessage(tr("Creating new file..."),2000);
    InvoiceForm *invoiceForm = createInvoiceForm();
    invoiceForm->newFile();
    invoiceForm->show();
    statusBar()->showMessage(tr("Ready."),2000);
}
```

MainWindow: implementáció

mainwindow.cpp

```
void MainWindow::on_openAction_activated()
{
	statusBar()->showMessage(tr("Opening file..."),2000);
	InvoiceForm *invoiceForm = createInvoiceForm();
	if(invoiceForm->open()) {
		invoiceForm->show();
		updateRecentFileActions(invoiceForm->getCurrentFile());
		statusBar()->showMessage(tr("File opened"),2000);
	} else {
		invoiceForm->close();
		statusBar()->showMessage(tr("File open failed"),2000);
	}
}

void MainWindow::openFile(const QString &fileName)
{
	statusBar()->showMessage(tr("Opening file..."),2000);
	InvoiceForm *invoiceForm = createInvoiceForm();
	if(invoiceForm->openFile(fileName)) {
	    updateRecentFileActions(invoiceForm->getCurrentFile());
		invoiceForm->show();
		statusBar()->showMessage(tr("File opened"),2000);
	} else {
		invoiceForm->close();
		statusBar()->showMessage(tr("File open failed"),2000);
	}
}
```

MainWindow: implementáció

mainwindow.cpp

```
void MainWindow::updateMenus()
{
    bool hasInvoiceForm = (activeInvoiceForm() != 0);
    saveAction->setEnabled(hasInvoiceForm);
    saveAsAction->setEnabled(hasInvoiceForm);
    closeAction->setEnabled(hasInvoiceForm);
    closeAllAction->setEnabled(hasInvoiceForm);
    tileAction->setEnabled(hasInvoiceForm);
    cascadeAction->setEnabled(hasInvoiceForm);
    nextAction->setEnabled(hasInvoiceForm);
    previousAction->setEnabled(hasInvoiceForm);

    itemInsertBeforeAction->setEnabled(hasInvoiceForm);
    itemInsertAfterAction->setEnabled(hasInvoiceForm);
    updateItemDeleteMenu();

    if(activeInvoiceForm())
        activeInvoiceForm()->windowMenuAction()->setChecked(true);
}

InvoiceForm* MainWindow::activeInvoiceForm()
{
    return qobject_cast<InvoiceForm*>(workspace->activeWindow());
}
```

MainWindow: implementáció

mainwindow.cpp

```
void MainWindow::on_saveAction_activated()
{
	statusBar()->showMessage(tr("Saving file..."),2000);
	if(activeInvoiceForm()){
		activeInvoiceForm()->save();
		updateRecentFileActions(activeInvoiceForm()->getCurrentFile());
	}
	statusBar()->showMessage(tr("File saved"),2000);
}
void MainWindow::on_saveAsAction_activated()

{
	statusBar()->showMessage(tr("Saving file..."),2000);
	if(activeInvoiceForm()) {
		activeInvoiceForm()->saveAs();
		updateRecentFileActions(activeInvoiceForm()->getCurrentFile());
   }
	statusBar()->showMessage(tr("File saved"),2000);
}
void MainWindow::closeEvent(QCloseEvent *event)
{

	workspace->closeAllWindows();
	if(activeInvoiceForm())
		event->ignore();
	else {
		writeSettings();
		event->accept();
	}
}
```

MainWindow: implementáció

```
void MainWindow::on_itemInsertBeforeAction_activated()
{
        activeInvoiceForm()->insertItemBefore();
        itemDeleteAction->setEnabled(activeInvoiceForm()
                        && (activeInvoiceForm()->itemCount() != 0));
}

void MainWindow::on_itemInsertAfterAction_activated()
{
        activeInvoiceForm()->insertItemAfter();
        updateItemDeleteMenu();
}

void MainWindow::on_itemDeleteAction_activated()
{
       activeInvoiceForm()->deleteItem();
       updateItemDeleteMenu();
}

void MainWindow::updateItemDeleteMenu()
{
       itemDeleteAction->setEnabled(activeInvoiceForm()
                       && (activeInvoiceForm()->itemCount() != 0));
}
```

mainwindow.cpp

MainWindow: implementáció

mainwindow.cpp

```
void MainWindow::createRecentFileMenus()
{
    //create actions
    for (int i=0; i < MaxRecentFiles; ++i) {
        recentFileActions[i] = new QAction(this);
        recentFileActions[i]->setVisible(false);
        connect(recentFileActions[i],SIGNAL(triggered()),this, SLOT(openRecentFile()) );
    }

    //update and create menus
    fileMenu->removeAction(exitAction);
    for (int i=0; i < MaxRecentFiles; ++i) {
        fileMenu->addAction(recentFileActions[i]);
    }
    separatorAction = fileMenu->addSeparator();
    fileMenu->addAction(exitAction);
}

void MainWindow::openRecentFile()
{
    QAction* action = qobject_cast<QAction*>(sender());
    if(action)
        openFile(action->data().toString());
}
```

MainWindow: implementáció

mainwindow.cpp

```
void MainWindow::updateRecentFileActions(const QString& fileName)
{
	recentFiles.removeAll(fileName);
	recentFiles.prepend(fileName);
	updateRecentFileActions();
}

void MainWindow::updateRecentFileActions()
{
	QMutableStringListIterator i(recentFiles);
	while(i.hasNext()) {
		if(!QFile::exists(i.next()))
			i.remove();
	}

	for(int j=0; j < MaxRecentFiles; ++j) {
		if (j < recentFiles.count()) {
			QString text = tr("%1 %2").arg(j+1).arg(QFileInfo(recentFiles[j]).fileName());
			recentFileActions[j]->setText(text);
			recentFileActions[j]->setData(recentFiles[j]);
			recentFileActions[j]->setVisible(true);
		} else {
			recentFileActions[j]->setVisible(false);
		}
		separatorAction->setVisible(!recentFiles.isEmpty());
	}
}
```

MainWindow: implementáció

mainwindow.cpp

```
void MainWindow::writeSettings()
{
    QSettings settings("ELTE IK EAF3","Invoicer" );
    settings.setValue("geometry",geometry());
    settings.setValue("recentFiles",recentFiles);
}

void MainWindow::readSettings()
{
    QSettings settings("ELTE IK EAF3","Invoicer" );

    QRect rect = settings.value("geometry", QRect(200,200,400,400)).toRect();
    move(rect.topLeft());
    resize(rect.size());

    recentFiles = settings.value("recentFiles").toStringList();
    updateRecentFileActions();
}
```

Főprogram

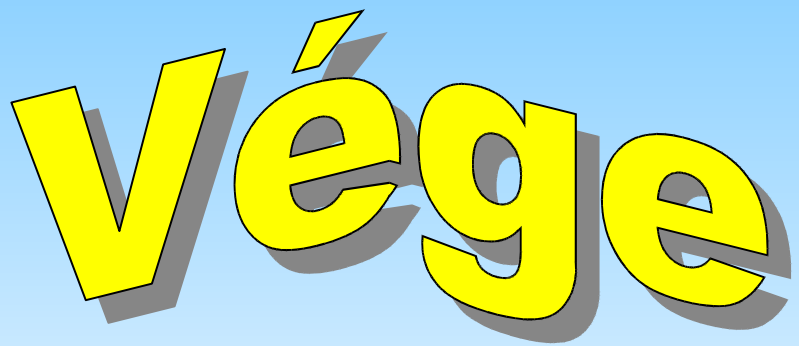
main.cpp

```
int main (int argc, char *argv[])
{
    QApplication app(argc,argv);

    QSplashScreen *splash = new QSplashScreen;
    splash->setPixmap(QPixmap("./images/splash.jpg"));
    splash->show();

    Qt::Alignment topRight = Qt::AlignRight | Qt::AlignTop;

    splash->showMessage(QObject::tr("Welcome ..."));

    MainWindow *mainWindow = new MainWindow;

    QStringList args = app.arguments();
    if (args.count() > 1) {
        for (int i=1; i < args.count(); i++)
            mainWindow->openFile(args[i]);
    }

    mainWindow->show();

    sleep(3);
    splash->finish(mainWindow);

    return app.exec();
}
```

A bemutatott program megtalálható a

people.inf.elte.nacsa/qt4/eaf3/inv03/projects/invoicer

címen.



Tartalomjegyzék

A munkafüzet programjai letölthetők a people.inf.elte.hu/nacsa/qt4/eaf3/mod01/projects/ címről.

A munkafüzetben bemutatott programok készítésekor a **Qt 4.2.2** verziót használtam.

Készítette: Szabóné Nacsa Rozália

email: nacsa@inf.elte.hu

honlap: people.inf.elte.hu/nacsa

Budapest, 2007. március

Alkalmazás készítés Qt osztályokkal

A **Qt** egy **keretrendszer** C++ kódú, platform független, többnyelvű alkalmazások fejlesztésére. A **Qt Designer** kimenete szabványos C++ kód, ezért mindenütt felhasználható, ahol bármely más C++ program. A **Qt** eszközök honlapja: **www.trolltech.com.** Részletes on-line segítséget kap, ha elindítja az **assistant** programot. A **Qt** ingyenes változatában nem működik kódkiegészítő szolgáltatás, ezért a fejlesztés során érdemes folyamatosan igénybe venni az ismeretek bőséges tárházát felsorakoztató „asszisztensünket”.

Ebben a modulban megismerkedünk a **Qt** néhány eszközével, és azok használatával.

A qmake eszköz

A **qmake** eszközzel könnyen és gyorsan összeállíthatjuk az alkalmazásunkat. A **qmake** a fejlesztés alatt álló alkalmazás fájljai alapján automatikusan elkészíti a projekt leíró fájlt (qmake -project), majd a projekt leíró fájlban található információk alapján automatikusan elkészíti a projekt összeállításához szükséges Makefile-t. (qmake -hello.pro). A Makefile-ba automatikusan bekerülnek az adott projekt elkészítéséhez szükséges fordítási, szerkesztési parancsok. Ha nyomkövetést is szeretnénk, akkor ezt a projekt leíró fájlban külön jelezni kell (CONFIG += qt debug).

“Hello Qt” (hello)

Feladat: Készítsünk egy futtatható “Hello Qt” alkalmazást, **Qt osztályok segítségével**, „kézi programozással”.

A “Hello Qt” alkalmazás Linux alatt

hello projekt: main.cpp

```
#include <QApplication>
#include <QLabel>

int main(int argc,char **argv)
{
    QApplication app(argc,argv);

    //QLabel *label = new QLabel("Hello Qt!");
    QLabel *label = new QLabel("<h1><i>Hello </i><font color=red>Qt!</font></h1>");
    label->show();

    return app.exec();
}
```

A fájl elejére beillesztettük a programban használt osztálydefiníciókat, majd megadtuk a grafikus alkalmazások készítésekor is kötelező *main()* függvényt. A *main()* függvényben először a verem tetején létrehoztunk egy **QApplication** (alkalmazás) objektumot (*app*). Ez az objektum kezeli az alkalmazás erőforrásait. Ezután létrehoztunk egy, a kívánt szöveget megjelenítő **QLabel** (címke) widget[1]-et (*label*), végül az így előkészített vezérlőt megjelenítettük a képernyőn (*show()*). A **QLabel** konstruktorában megadott szövegre alkalmazhatunk HTML stílusú formázást is. **Qt**-ben bármely vezérlő lehet főablak, mert minden vezérlője a **QWidget** osztályból származik.

1 *widget: a felhasználói felületen látható elem*

Alapértelmezésben az újonnan létrehozott vezérlők nem jelennek meg, erről nekünk külön gondoskodni kell. Ezzel elkerülhető a folyamatos frissítésből adódó villogás.

A program végén meghívtuk az alkalmazás *exec()* metódusát. Ezzel az alkalmazás "készenléti állapotba" kerül, és a neki szóló eseményekre várakozik ("*üzenet ciklus"*). Az így futtatott alkalmazás folyamatosan feldolgozza a hozzá érkező eseményeket mindaddig, amíg be nem zárjuk az alkalmazás főablakát. A főablak bezárásakor a főablakhoz rendelt vezérlők törlődnek a memóriából, és eltűnnek a képernyőről.

A projekt elkészítésének lépései

1. Hozza létre a **hello** alkönyvtárat.
2. Gépelje be a fenti programot a **main.cpp** fájlba, és mentse el azt a **hello** alkönyvtárba.
3. Legyen a **hello** alkönyvtárban.
4. A **qmake -project** paranccsal állítsa elő a platform független projekt leíró fájlt (*hello.pro*).
5. A **qmake hello.pro** paranccsal állítsa elő a projekt platform függő make[2] fájlját.
6. A **make** paranccsal szerkessze össze a projektet.
7. **Futtassa** a programot. **./hello**

A projekt leíró fájl tartalma:

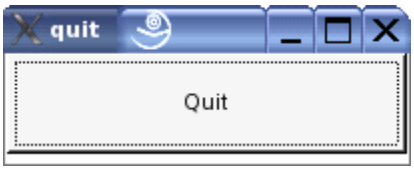
```
TEMPLATE = app
TARGET =
DEPENDPATH += .
INCLUDEPATH += .

# Input
SOURCES += hello.cpp
```

A program letölthető a people.inf.elte.hu/nacsa/qt4/eaf3/mod01/projects/hello címről.

Objektumok közötti kommunikáció (quit)

A **Qt** vezérlő elemei (*widget*-ek) állapotuk megváltozásakor vagy egy rájuk vonatkozó felhasználói esemény (pl. egér kattintás) bekövetkezésekor jelet (signál, „adás") adhatnak le. **Qt**-ben léteznek az ilyen jelek fogadására alkalmas (egyébként szabványos függvényként is meghívható) jelkezelő függvények (*slot*). Ha jelet (*signal*) és jelkezelő függvényeket (*slot*) összekapcsolunk, az összekapcsolás után a jel kibocsájtásakor a jelkezelő függvények automatikusan végrehajtódnak. Az összekapcsolást a **QObject** osztály *connect()* függvényével lehet megadni. Egy jelre több jelkezelő függvény is rákapcsolható. Ilyenkor a függvények végrehajtási sorrendje nem definiált. **Qt**-ben két jelet is össze lehet kötni.

Ebben a példában megmutatjuk, hogyan „válaszolhatunk" a felhasználó által kezdeményezett eseményre a jel--jelkezelés (*signal-slot)* mechanizmussal. Az alkalmazás egyetlen gombból áll. A gombra kattintva (*felhasználói esemény*) az alkalmazás ablaka bezárul (*az esemény feldolgozása*). A főablak bezárásakor a főablak automatikusan törlődik a memóriából, és eltűnik a képernyőről is. A példa nagyon hasonlít az „Hello Qt" programhoz, csak itt címke (**QLabel**) helyett gombbal (**QPushButton**) dolgozunk.



A "Quit" alkalmazás Linux alatt

2 : A Makefile egy olyan script fájl, amely az adott projekt elemeinek lefordítását, összeszerkesztését végzi.

quit projekt: main.cpp

```
#include <QApplication>
#include <QPushButton>

int main(int argc, char *argv[])
{
    QApplication app(argc,argv);
    QPushButton *button = new QPushButton("Quit");

    QObject::connect(button,SIGNAL(clicked()),&app,SLOT(quit()));

    button->show();
    return app.exec();
}
```

A példa *connect()* függvényében a *button* widget *clicked()* jelzését (signal) kötöttük össze az *app* objektum *quit()* jelkezelő függvényével (slot). Figyeljük meg, hogy paraméterként a *button* és az *app* objektum címét adtuk meg.

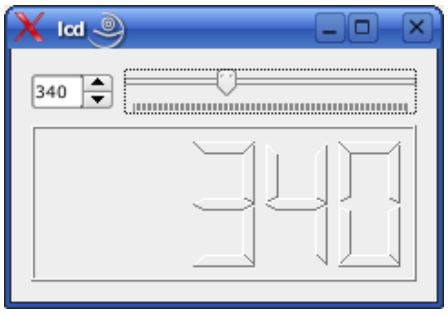
A projekt elkészítésének lépései

1. Hozza létre a **quit** alkönyvtárat.
2. Hozza létre a **main.cpp** fájlt, és mentse el a **quit** alkönyvtárba.
3. Legyen a **quit** alkönyvtárban.
4. A **qmake -project** paranccsal állítsa elő a platform független projekt leíró fájlt (**quit.pro**).
5. A **qmake quit.pro** paranccsal állítsa elő a projekt platform függő make fájlját.
6. A **make** paranccsal szerkessze össze a projektet.
7. **Futtassa** a programot. Linux: **./quit**

A program letölthető a people.inf.elte.hu/nacsa/qt4/eaf3/mod01/projects/quit címről.

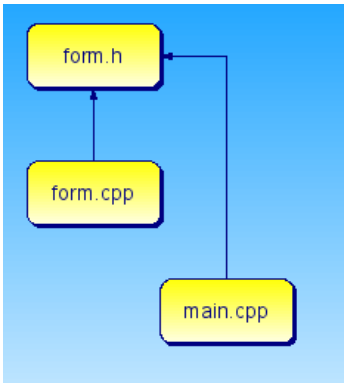
Grafikus felület létrehozása, vezérlők szinkronizálása „programozással" (lcd)

Feladat: Készítsük alkalmazást, amely egy számlálót, egy csúszkát és egy LCD kijelzőt tartalmaz. Ha bármelyiken beállítunk egy értéket, a többi vezérlő automatikusan mutassa a beállított értéket.

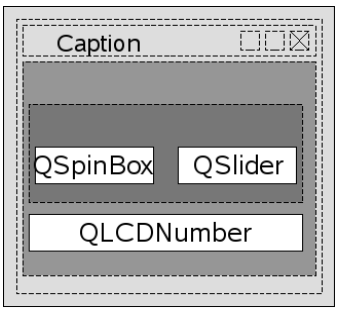


Először elkészítjük a grafikus felületet reprezentáló **Form** osztályt. A főprogramban (*main()*) létrehozzuk a **Form** osztály egy példányát, melyet megjelenítünk a képernyőn (*show()*), majd elindítjuk az alkalmazást (*exec()*).

A főablakot a **QWidget** osztályból származtatjuk. A főablakon elhelyezzük a **QSpinBox** (*spinBox*), **QSlider** (*slider*) és a **QLCDNumber** (*lcd*) vezérlőket.



Az alkalmazás modulszerkezete



Az alkalmazás felületterve

A felületen elhelyezett elemek

Típus	***Név***	***Beállítások***
QSpinBox	spinBox	minValue = 0, maxValue = 10
QSlider	slider	minValue = 0, maxValue = 10
QLCDNUmber	lcd	

A Form osztály definíciója

lcd projekt: form.h

```
#ifndef _FORM_H_
#define _FORM_H_

#include <QWidget>

class QSpinBox;
class QSlider;
class QLCDNumber;
class QHBoxLayout;
class QVBoxLayout;

class Form: public QWidget
{
    Q_OBJECT
 public:
    Form(QWidget *parent=0);
 private:
    QSpinBox *spinBox;      //számláló
    QSlider *slider;        //csúszka
    QLCDNumber *lcd;        //lcd kijelző

    QHBoxLayout *topLayout;    //vízszintes elrendező
    QVBoxLayout *mainLayout;  //függőleges elrendező

};
#endif // _FORM_H_
```

A Form osztály implementációja

lcd projekt: form.cpp

```
#include <QLayout>
#include <QSpinBox>
#include <QSlider>
#include <QLCDNumber>

#include "form.h"

Form::Form(QWidget *parent) : QWidget(parent)
{
    spinBox = new QSpinBox;
    setObjectName(QString::fromUtf8("spinBox"));
    spinBox->setRange(0,100);
```

Létrehoztuk a számláló objektumot (*spinBox*), és beállítottuk a tulajdonságait. A számláló 0 és 100 közötti értéket vehet fel.

```
    slider = new QSlider(Qt::Horizontal);
    setObjectName(QString::fromUtf8("slider"));
    slider->setRange(0,100);
    slider->setSingleStep(10);
    slider->setOrientation(Qt::Horizontal);
    slider->setTickPosition(QSlider::TicksBelow);
```

Létrehoztuk a csúszkát (*slider*), és beállítottuk a tulajdonságait. A csúszka 0 és 100 közötti értéket vehet fel. Ha a csúszkát nyíllal léptetjük, akkor egy leütés 10-et ér. A csúszkát vízszintesen szeretnénk elhelyezni. A csúszka alatt jelenjenek meg az értékeket jelző rovátkák.

```
    lcd = new QLCDNumber;
    setObjectName(QString::fromUtf8("lcd"));
```

Létrehoztuk az *lcd* kijelző objektumot.

```
    topLayout = new QHBoxLayout;
    topLayout->addWidget(spinBox);
    topLayout->addWidget(slider);
```

Létrehoztuk a *topLayout* vízszintes elrendezőnket, majd elhelyeztünk benne két vezérlőt, a számlálónkat és a csúszkánkat.

```
    mainLayout = new QVBoxLayout;
    mainLayout->addLayout(topLayout);
    mainLayout->addWidget(lcd);
```

Létrehoztuk a függőleges elrendezőt, amelybe két elemet helyeztünk el. A vízszintes elrendezőnket (*topLayout*) és az LCD kijelzőnket (*lcd*). Vegye észre, hogy az elrendezőbe betehetünk elrendezőt (*Layout*) is és vezérlőt (*widget*) is.

```
    mainLayout->setMargin(11);
    mainLayout->setSpacing(6);
```

Az elrendező margóját 11 pixelre, az elrendezőben elhelyezett vezérlő elemek közötti távolságot 6 pixelre állítottuk be.

```
    setLayout(mainLayout);
```

A *setLayout()* hatására minden elem, amely a *mainLayout* rendezőben van, az űrlapunk (Form) gyermeke lesz. Az űrlap megszűnésekor a hozzátartozó gyerekek is megszűnnek, és az általuk elfoglalt tárhely automatikusan felszabadul.

```
connect(spinBox, SIGNAL(valueChanged(int)), lcd, SLOT(display(int)));
connect(spinBox, SIGNAL(valueChanged(int)), slider, SLOT(setValue(int)));
connect(slider, SIGNAL(valueChanged(int)), lcd, SLOT(display(int)));
connect(slider, SIGNAL(valueChanged(int)), spinBox, SLOT(setValue(int)));
```

A *connect()* függvénnyel vezérlők által kibocsájtott jeleket (signal) kapcsolunk össze vezérlők jelfeldolgozó függvényeivel (slot). Így például, ha a számláló értéke megváltozott, akkor a csúszka automatikusan felveszi a megváltozott értéket. Esetünkben a *connect()* függvényhívásnál azért nem kell megadni, hogy ő a **QObject** osztály metódusa, mert az űrlapukat a **QWidget** osztályból származtattuk, a *connect()* függvény a **QObject** osztály függvénye, a **QObject** osztály pedig a **QWidget** osztály őse, ezért a fordító így is megtalálja ezt a függvényt.

```
}
```

Az alkalmazás főprogramja

lcd projekt: main.cpp

```
#include <QApplication>
#include "form.h"

int main(int argc, char *argv[])
{
    QApplication app(argc,argv);
    Form *form = new Form;
    form->show();
    return app.exec();
}
```

Az alkalmazás projekt fájlja

Nézzük meg a **qmake** eszközzel generált projekt leíró fájlt.

lcd projekt: lcd.pro

```
TEMPLATE = app
TARGET =
DEPENDPATH += .
INCLUDEPATH += .

# Input
HEADERS += form.h
SOURCES += form.cpp main.cpp
```

A projekt elkészítésének lépései

1. Hozza létre az **lcd** alkönyvtárat.
2. Hozza létre a **form.h**, **form.cpp**, **main.cpp** fájlokat, és tárolja azokat a **lcd** alkönyvtárban.
3. Legyen a **lcd** alkönyvtárban.
4. A **qmake -project** paranccsal állítsa elő a platform független projekt leíró fájlt (**lcd.pro**).
5. A **qmake -o Makefile nlcd.pro** paranccsal állítsa elő a projekt platform függő **make** fájlját.
6. A **make** paranccsal szerkessze össze a projektet.
7. **Futtassa** a programot.

A program letölthető a people.inf.elte.hu/nacsa/qt4/eaf3/mod01/projects/lcd címről.

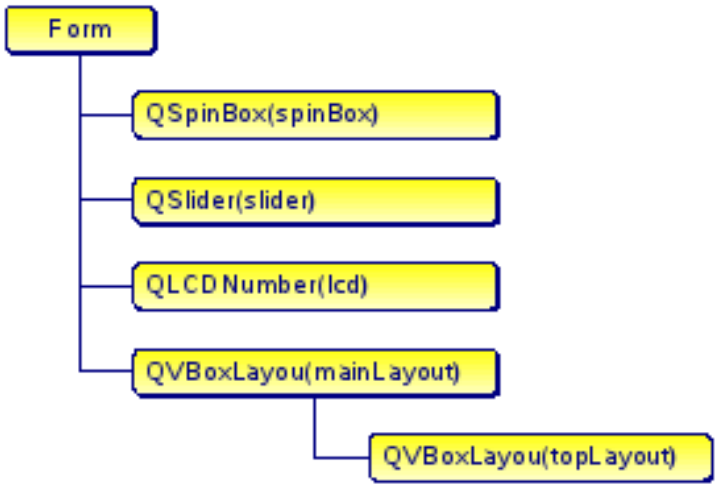
Memóriagazdálkodás, szülő-gyermek kapcsolat

Ha megnézzük a programunkat, akkor azt látjuk, hogy számos helyen használjuk a **new** operátort, és sehol sem szabadítottuk fel az így lefoglalt tárhelyet. Grafikus alkalmazások készítésekor nagyon sok elemet (widget) kell kezelni. Ezeket általában a szabad tárterületen (*heap*) **new** operátorral hozzuk létre. Az eddigi ismereteink szerint a **new** operátorral lefoglalt tárhelyet fel kell szabadítani. A vezérlők használat utáni megsemmisítését és az általuk elfoglalt tárhely felszabadítását segíti a **Qt** "**szülő-gyermek**" mechanizmusa, melynek lényege a következő:

- Ha egy widget-et törlünk, akkor törlésre kerül annak valamennyi gyermeke.
- Törölt widget eltűnik a képernyőről.
- A főablakként kezelt widget bezárásnál törlődik.

Tehát csak olyan widget-ek törléséről kell gondoskodnunk, melyeket **new** operátorral hoztunk létre, és **nem volt szülője**.

Az alábbi ábra az **lcd** program „szülő-gyermek[3]" kapcsolatát mutatja.



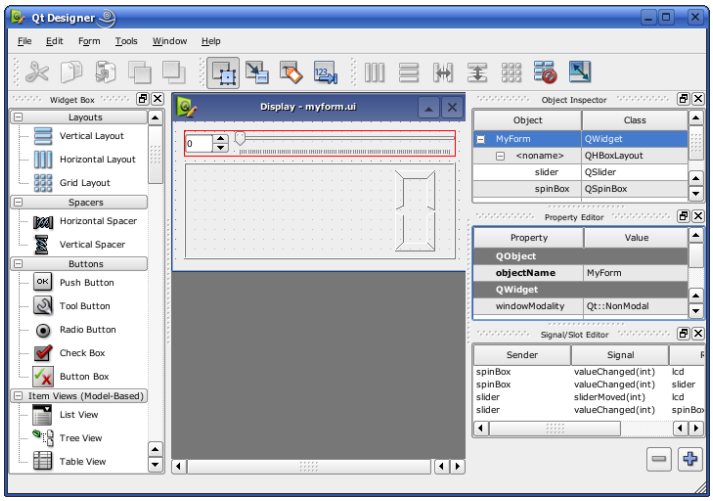
A **Form** osztály implementációjában található *setLayout(mainLayout)* függvényhívás hatására a vezérlők „szülő-gyermek" kapcsolata úgy épül fel, hogy a *mainLayout*-hoz rendelt valamennyi vezérlőnek a **Form** osztály lesz a szülője, így a **Form** ablak bezárásakor minden általunk létrehozott vezérlő számára lefoglalt tárhely felszabadul.

Grafikus felület tervezése Qt Designerrel (display)

Bonyolult felületek kialakítása „kézi programozással" nehézkes. Ilyenkor érdemes használni a **Qt** felület tervező eszközét, a **Qt Designer**-t.

A **Qt Designer -** melyet installálás után a **designer** paranccsal lehet elindítani - grafikus felületek tervezését támogató (*nem integrált*) vizuális eszköz. A **Qt** eszközök alkalmazásakor a felület megtervezése után a felület osztályból származtatott osztályban adhatjuk meg a felület alkalmazás-specifikus funkcióit.

3 A „szülő-gyermek" kapcsolat és az öröklődés nem azonos fogalmak.



Feladat: Készítsük el az előző programunkat **Qt Designer** segítségével is. Projekt neve legyen: **display**

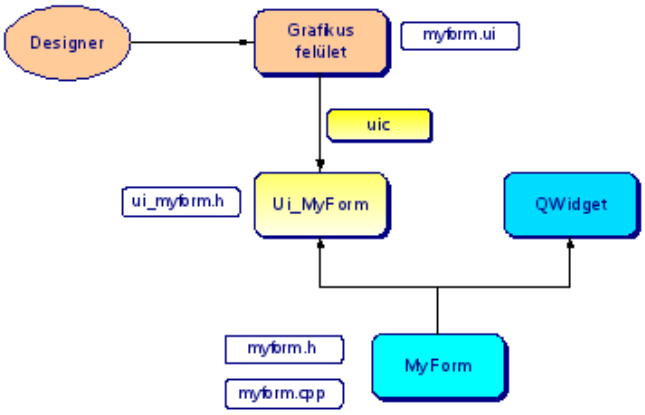
A grafikus felület megtervezése:

1. Indítsa el a **Qt Designert** (designer).
2. Open->Widget
3. A felbukkanó **NewForm** ablakban válassza ki a **Widget** sablont.
4. Create
5. Form kijelölése -> Property Editor -> objecName: **MyForm**
6. Az eszköztárból válassza ki a **QSpinBox** eszközt és húzza rá a felületre.
7. **QSpinBox** kijelölése – Property Editor – objectName: **spinBox**
8. Az eszköztárból válassza ki a **QSlider** eszközt és húzza rá a felületre.
9. **QSlider** kijelölése – Property Editor – objectName: **slider**
10. Az eszköztárból válassza ki a **QLCDNumber** eszközt és húzza rá a felületre.
11. **QLCDNumber** kijelölése – Property Editor – objectName: **lcd**
12. Elrendezés: Kattintson a **spinBox**-ra, majd a **Shift**-et lenyomva tartva a **slider**-re.
13. **Form** menü/**Lay out Horizontally**
14. **Form** (az űrlap(!)) kijelölése – Form menü/**Lay out Vertically**
15. **Edit**/ **Edit Signals/slots** menü
16. Kattintson a **spinBox**-ra, majd a nyilat húzza át a **slider**-re
    Kötögesse össze a vezérlők jeleit (signal) és jelkezelőit (slot) az alábbi tábla alapján:

Sender	***Signal***	***Receiver***	***Slot***
spinBox	valueChanged(int)	lcd	display(int)
spinBox	valueChanged(int)	slider	setValue(int)
slider	valueChanged(int)	lcd	display(int)
slider	valueChanged(int)	spinBox	setValue(int)

17. Az **Edit/Edit Widgets** paranccsal visszatérhet a *widget tervező* módra.
    (A Tools/Signal/Slots Editor menüpont bekapcsolásával teheti ki a képernyőre a Signal/Slot ablakot.)
18. Mentse el a felületet **myform.ui** néven a display alkönyvtárba.

Nézzük meg, hogyan illeszthető be programunkba a **Qt Designerrel** elkészített „felület”, vagyis hogyan lesz ebből a felületből C++ kód.



A **Qt Designer** "kimenete" egy **XML** alapú, **.ui** kiterjesztésű szöveges fájl, amely a grafikus felület definícióját tartalmazza (**myform.ui**). Az alkalmazás „összeállításakor” a felületet definiáló **.ui** fájlból egy speciális program, az **uic** (*user interface compiler*) segítségével készül el a felületet reprezentáló **Ui_MyForm** osztály C++ kódja, amely az **ui_mainform.h** fájlba kerül.

A grafikus felület megtervezése után készíthetjük el a felületet reprezentáló **MyForm** osztályt. A **MyForm** osztályt a **QWidget** és az **Ui_MyForm** (két!) osztályból, származtatással állítjuk elő. Az alkalmazás specifikus funkciókat ebben az osztályban kell elhelyezni. A **Qt Designerrel** megtervezett vezérlők a **MyForm** osztály konstruktorában kiadott *setupUi(this)* parancs hatására kerülnek rá az űrlapunkra.

MyForm osztály

display projekt: myform.h

```
#include <QWidget>
#include "ui_myform.h"

class MyForm : public QWidget, private Ui_MyForm
{
    Q_OBJECT
public:
    MyForm(QWidget *parent = 0);
};
```

display projekt: myform.cpp

```
#include "myform.h"

MyForm::MyForm(QWidget *parent)    : QWidget(parent)
{
    setupUi(this);
}
```

Az alkalmazás főprogramja

display projekt: main.cpp

```
#include <QApplication>
#include "myform.h"

int main (int argc, char *argv[])
{
    QApplication app(argc,argv);

    MyForm *form = new MyForm;
    form->show();

    return app.exec();
}
```

Az alkalmazás projekt fájlja

display projekt: dislay.pro

```
TEMPLATE = app

TARGET =
DEPENDPATH += .
INCLUDEPATH += .

# Input
HEADERS += myform.h
FORMS += myform.ui
SOURCES += main.cpp myform.cpp
```

A projekt elkészítésének lépései

1. Hozza létre az **display** alkönyvtárat.
2. Gépelje be a **myform.h**, **myform.cpp**, **main.cpp** fájlokat, és mentse el a **display** alkönyvtárban.
3. Legyen a **display** alkönyvtárban.
4. A **qmake -project** paranccsal állítsa elő a platform független projekt leíró fájlt (**display.pro**).
5. A **qmake -o Makefile display.pro** paranccsal állítsa elő a projekt platform függő **make** fájlját.
6. A **make** paranccsal szerkessze össze a projektet.
7. **Futtassa** a programot.

A program letölthető a people.inf.elte.hu/nacsa/qt4/eaf3/mod01/projects/display címről.

Pillantson be az *uic* által generált **ui_myform.h** fájlba. Nézze meg, milyen kódot generált az *uic*. Sokszor jó ötleteket kaphatunk a generált kód elemzésével. Figyelje meg a generált *connect()* függvényhívásokat.

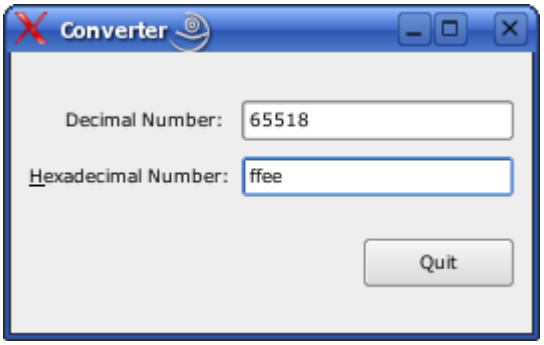
```
QObject::connect(spinBox, SIGNAL(valueChanged(int)), lcd, SLOT(display(int)));
QObject::connect(spinBox, SIGNAL(valueChanged(int)), slider, SLOT(setValue(int)));
QObject::connect(slider, SIGNAL(valueChanged(int)), lcd, SLOT(display(int)));
QObject::connect(slider, SIGNAL(valueChanged(int)), spinBox, SLOT(setValue(int)));
```

Vegye észre, hogy a jelek (*signals*) és a jelkezelő függvények (*slots*) paramétereiben **csak a paraméter típusa** szerepel, és nincs paraméter név.

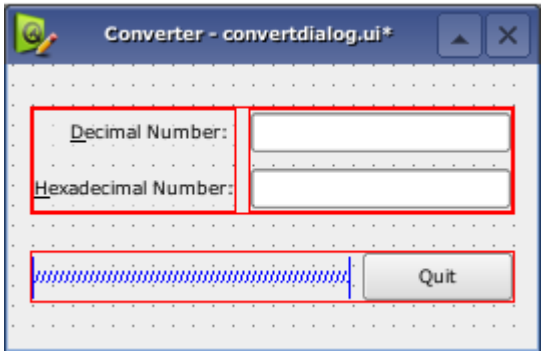
Saját jelkezelők (slotok) használata (convert)

Qt-ben megadhatunk saját jelkezelő függvényeket (slot). Erre mutatunk példát a következő programban.

Feladat: Készítsünk programot, amely egyszerre jeleníti meg egy szám decimális és hexadecimális értékét. Ha bármelyik értéket megváltoztatjuk, a másik érték is azonnal változzon meg.



Alkalmazás futás közben



Felület tervezése Qt Designerben

A grafikus felület elemei és beállításuk

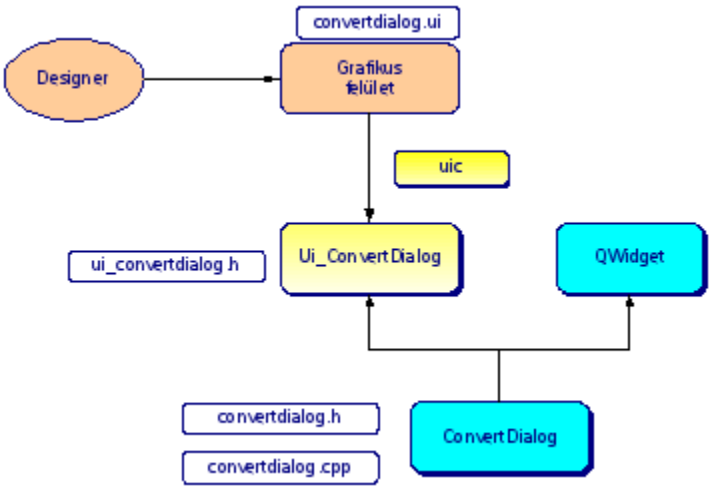
Típus	*Név*	*Beállítások*
QLabel	decimalLabel	text: &Decimal Number; buddy: decimalLineEdit; Alignement-Horizontal: Qt::AlignRight
QLabel	hexaLabel	text: &Hexadecimal Number; buddy: hexaLineEdit; Alignement-Horizontal: Qt::AlignRight
QLineEdit	decimalLineEdit	
QLineEdit	hexaLineEdit	
QPushButton	quitButton	text: &Quit

Megjegyzés: *Az &X megadásával az adott elemet Alt+X gyorsbillentyűvel is kiválaszthatjuk. A „buddy” beállításával az adatbeviteli mezőt megelőző címke gyorsbillentyűje a buddy-ként megjelölt mezőre ugrik.*

Indítsa el a **Qt Designert,** és készítse el a **ConvertDialog** osztály felülettervét. A tervezés végén mentse el a tervet a **convertdialog.ui** fájlba. A tervezett felülethez a **qmake** majd elkészíti az **Ui_ConvertDialog** osztályt.

ConvertDialog osztály

A **ConvertDialog** osztályunkat a **QWidget** és a **Qt Designerrel** elkészített **Ui_ConvertDialog** osztályokból, származtatással készítjük el.



convert projekt: convertdialog.h

```
#ifndef _CONVERT_DIALOG_H_
#define _CONVERT_DIALOG_H_

#include <QWidget>
#include "ui_convertdialog.h"

class ConvertDialog : public QWidget, private Ui_ConvertDialog {

    Q_OBJECT

public:
    ConvertDialog(QWidget *parent = 0);

public slots:
    void on_quitButton_clicked();
    void updateHexaLineEdit(const QString & str);
    void updateDecimalLineEdit(const QString & str);

private:
    QString convert(const QString& str, const int fromBase, const int toBase);

};
#endif //  _CONVERT_DIALOG_H_
```

convert projekt: convertdialog.cpp

```
#include <QMessageBox>
#include "convertdialog.h"
```

```
ConvertDialog::ConvertDialog(QWidget *parent)
    : QWidget(parent)
{
    setupUi(this);
```

A **Qt Designerrel** megtervezett vezérlőket elhelyeztük az űrlapon.

```
    connect(decimalLineEdit, SIGNAL(textEdited(const QString &)),
            this, SLOT(updateHexaLineEdit(const QString &)));
    connect(hexaLineEdit, SIGNAL(textEdited(const QString &)),
            this, SLOT(updateDecimalLineEdit(const QString &)));
}
```

Ha a decimális szám megváltozott, akkor a *decimalLineEdit* vezérlő jelet küld. A jel továbbítja a megváltozott szöveget, amelyet az általunk elkészített (az űrlaphoz tartozó) jelfeldolgozó függvény *(updateDecimalLineEdit(const QString &))* kap meg. Hasonlóan kezeljük le a hexadecimális szám megváltozását.

```
void ConvertDialog::on_quitButton_clicked()     // a "névkonvenció" miatt nem kell connect
{
    qApp->quit();  //qApp az (egyetlen!) alkalmazásunkra mutató pointer
}

void ConvertDialog::updateHexaLineEdit(const QString & str)
{
    hexaLineEdit->setText(convert(str,10,16).toUpper());
}
```

A jelfeldolgozó függvény (*slot*) meghívja a *convert()* privát függvényünket, melynek visszatérési értéke az átkonvertált szám szövege. Ezzel a szöveggel aktualizáltuk a *hexaLineEdit* vezérlőnk *text* adatát.

```
void ConvertDialog::updateDecimalLineEdit(const QString & str)
{
    decimalLineEdit->setText(convert(str,16,10));
}

QString ConvertDialog::convert(const QString& str, const int fromBase, const int toBase)
{
    QString ret = "";
    bool ok;
    int value = str.toInt(&ok, fromBase);
    if(ok)
      ret = QString::number(value,toBase);
    else
      QMessageBox::information(this, "Convert Dialog", QString::number(value) +
                                                    " is not a valid number.");
    return ret;
}
```

A *convert()* privát függvényben az *str* stringet számként értelmezve *fromBase* számrendszerből *toBase* számrendszerbeli számra konvertáltuk, majd visszaadtuk az így előállított számot szöveges formában. A Qstring osztály number() függvénye visszaadja az első paraméterben megadott számot szöveges formában a második paraméterben megadott számrendszert figyelembe véve. Ha nem adunk meg második paramétert, akkor 10-es számrendszerben számol. A QString egy igen gazdag szolgáltatásokkal bíró szövegkezelő osztály, ezért érdemes közelebbről is megismerni. Indítsa el az **assistant** programot, és keresse meg a Qt osztályt. Próbálja ki néhány

metódus működését.

Az alkalmazás főprogramja

convert projekt: main.cpp

```
#include <QApplication>
#include "convertdialog.h"

int main (int argc, char *argv[]) {
    QApplication app(argc,argv);

    ConvertDialog *dialog = new ConvertDialog;
    dialog->show();

    return app.exec();
}
```

Az alkalmazás projekt fájlja

convert projekt: convert.pro

```
TEMPLATE = app
TARGET =
DEPENDPATH += .
INCLUDEPATH += .

# Input
HEADERS += convertdialog.h
FORMS += convertdialog.ui
SOURCES += convertdialog.cpp main.cpp
```

A projekt elkészítésének lépései

1. Hozza létre az **convert** alkönyvtárat.
2. Gépelje be a **myform.h**, **myform.cpp**, **main.cpp** fájlokat, és mentse el a **convert** alkönyvtárban.
3. Legyen a **convert** alkönyvtárban.
4. A **qmake -project** paranccsal állítsa elő a platform független projekt leíró fájlt (**convert.pro**).
5. A **qmake -o Makefile convert.pro** paranccsal állítsa elő a projekt platform függő **make** fájlját.
6. A **make** paranccsal szerkessze össze a projektet.
7. **Futtassa** a programot.

A program letölthető a people.inf.elte.hu/nacsa/qt4/eaf3/mod01/projects/convert címről.

Saját vezérlő használata Qt Designerben (worldclocks)

Készítsünk programot, mely mutatja a helyi időt a világ különböző pontjaira.

DigitalClock osztály

A **Qt** osztályai között vannak olyanok is, melyek a felhasználó közreműködése nélkül tudnak jelet küldeni. Ilyen például a **QTimer** osztály, amely egy adott idő elteltével vagy adott időközönként küld jelet (*timeout()*). Az időzítő használatára mutatunk példát az alábbi **worldclocks** példában. Először létrehozzuk a **DigitalClock** osztályt, amelyet a **Qt Designer** segítségével több példányban is ráteszünk az űrlapunkra.

A **DigitalClock** osztályunkat a **QLCDNumber** osztályból, „kézi programozással", származtatással készítjük el.

worldclocks projekt: digitalclock.h

```
#ifndef _DIGITAL_CLOCK_H_
#define _DIGITAL_CLOCK_H_

#include <QLCDNumber>
class QTimer;

class DigitalClock : public QLCDNumber
{
   Q_OBJECT

public:
   DigitalClock(QWidget *parent = 0);
   void setTimeZone(int time) {mTimeZone = time;}

private slots:
   void showTime();

private:
   QTimer* timer;
   int mTimeZone;
};
#endif //_DIGITAL_CLOCK_H_
```

worldclocks projekt: digitalclock.cpp

```
#include <QTimer>
#include <QDateTime>

#include "digitalclock.h"

DigitalClock::DigitalClock(QWidget *parent)
   : timer(0),mTimeZone(0), QLCDNumber(parent)
{
```

```
    setSegmentStyle(Filled);

    timer = new QTimer(this);
    connect(timer, SIGNAL(timeout()), this, SLOT(showTime()));
    timer->start(1000);

    showTime();

    setWindowTitle(tr("Digital Clock"));
    resize(150, 60);
}
```

Beállítottuk az LCD kijelző megjelenését. Létrehoztuk az időzítőnket, majd összekapcsoltuk az időzítő *timeout()* jelét a **DigitalClock** osztály *showTime()* jelkezelő függvényével. Az időzítő *start()* függvényében azt az időt kell megadni, mely időközönként az időzítőnknek *timeout()* jelet kell küldeni (1000 milisecundum 1 másodperc). A grafikus felületen megjelenő szövegeinket *tr()* függvénybe „csomagoltuk", így, ha igény lesz rá, akkor programunk felületét a **Qt QLinguist** eszközzel könnyen átalakíthatjuk tetszőleges más nyelvre is.

```
void DigitalClock::showTime()
{
    QTime time = QTime::currentTime().addSecs(mTimeZone *60 *60);
    QString text = time.toString("hh:mm");
    if ((time.second() % 2) == 0)
        text[2] = ' ';
    display(text);
}
```

A **QTime** osztály *toString()* metódusával az aktuális időből elkészítjük az LCD kijelzőn megjelenítendő szöveget óó:pp alakban, ú időjelző szöveget. Minden második másodpercben a kettőspontot lecseréljük helyközzel.

A DigitalClock osztállyal definiált vezérlő használata Qt Designerben („Promote")

1. Indítsa el a **Qt Designer**t.
2. Open/Widget
3. A felbukkanó **NewForm** ablakban válassza ki a **Widget** sablont.
4. Create
5. Form kijelölése - Property Editor - objecName: **MyForm**
6. A **QLCDNumber** vezérlőt húzzá rá az ablakra.
7. Jelölje ki a vezérlőt, majd jobb kattintással hívja be a helyi menüt, és válassza ki a „**Promote to Custom Widget**" menüpontot.
8. Adja meg a **DigitalClock** nevet.

   (Ezzel a technikával csak az ősosztály tulajdonságait tudja a Property ablakban beállítani. A későbbiekben majd megtanulunk egy olyan technikát is, ahol már a saját vezérlőnkre bevezetett tulajdonságokat is állítani lehet a Property ablakban.)
9. Az előző pont alapján helyezze el az összes **DigitalClock** elemet az űrlapra.

Típus	***Név***	***Beállítások***
DigitalClock	lcdBudapest	Promote: QLCDNumber
DigitalClock	lcdTokio	Promote: QLCDNumber

Típus	***Név***	***Beállítások***
DigitalClock	lcdLondon	Promote: QLCDNumber
DigitalClock	lcdMoscow	Promote: QLCDNumber
DigitalClock	lcdAlaska	Promote: QLCDNumber
DigitalClock	lcdNewZealand	Promote: QLCDNumber

10. A szükséges elrendezők segítségével (*Layout*) rendezze el vezérlőket az űrlapon.
11. Mentse el a felületet **myform.ui** néven a **worldclocks** alkönyvtárba.

Az alkalmazás főprogramja

worldclocks projekt: main.cpp

```
#include <QApplication>
#include "myform.h"

int main (int argc, char *argv[])
{
    QApplication app(argc,argv);

    MyForm *form = new MyForm;
    form->show();

    return app.exec(n);
}
```

Az alkalmazás projekt fájlja

worldclock projekt: worldclock.pro

```
TEMPLATE = app

TARGET =
DEPENDPATH += .
INCLUDEPATH += .

# Input
HEADERS += digitalclock.h myform.h
FORMS += myform.ui
SOURCES += digitalclock.cpp main.cpp myform.cpp
```

A projekt elkészítésének lépései

1. Hozza létre az **worldclocks** alkönyvtárat.
2. Készítse el a **digitalclock.h** és **digitalclock.cpp** fájlokat.
3. Regisztrálja be a **Qt Designerbe** a **DigitalClock** osztályt.
4. **Qt Designer**rel tervezze meg a grafikus felületet, majd mentse el **myform.ui** néven.
5. Gépelje be a **myform.h**, **myform.cpp**, **main.cpp** fájlokat, és mentse el.
6. Legyen a **worldclocks** alkönyvtárban.
7. A **qmake -project** paranccsal állítsa elő a platform független projekt leíró fájlt (**worldclocks .pro**).
8. A **qmake -o Makefile worldclocks .pro** paranccsal állítsa elő a projekt platform függő **make** fájlját.
9. A **make** paranccsal szerkessze össze a projektet.
10. **Futtassa** a programot.

A program letölthető a people.inf.elte.hu/nacsa/qt4/eaf3/mod01/projects/worldclocks címről.

Tartalomjegyzék

A munkafüzet programjai letölthetők a people.inf.elte.hu/nacsa/qt4/eaf3/mod02/projects/ címről.

A munkafüzetben bemutatott programok készítésekor a Qt 4.2.2 verziót használtam.

Készítette: Szabóné Nacsa Rozália

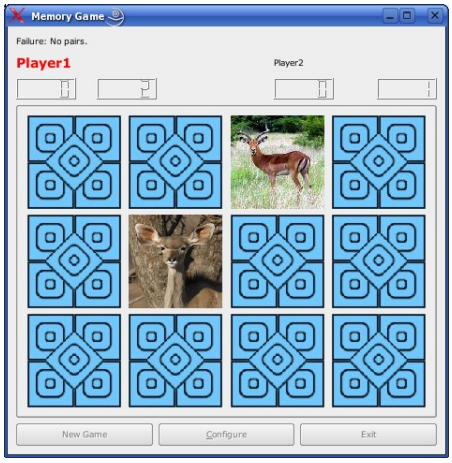
email: nacsa@inf.elte.hu

honlap: people.inf.elte.hu/nacsa

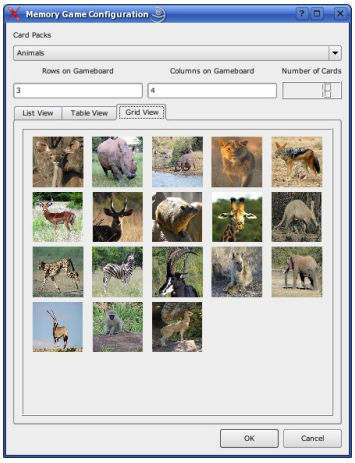
Budapest, 2007. március

Memóriajáték

Feladat: Készítsünk programot, mellyel játszhatjuk a jól ismert memóriajátékot. A játék során a játékmező legyen átméretezhető. A játékhoz szükséges képeket az alkalmazás könyvtárában található képfájlokból állítsuk elő.



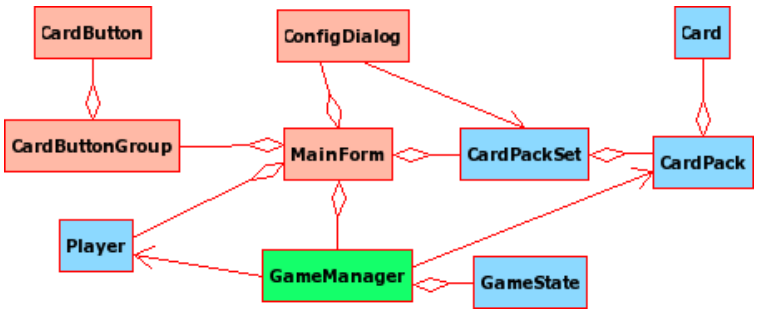
Memóriajáték – játék közben (MainForm)



A játékmező beállítása (ConfigDialog)

Az alkalmazást modularizáltan, jól meghatározható felelősséggel rendelkező **osztályok** segítségével szeretnénk elkészíteni. Ebben a munkafüzetben a játékprogram megvalósításához szükséges osztályok egy részét készítjük el. Bár ezeket az osztályokat a játékprogramhoz terveztük, a tanulás megkönnyítésére minden osztályhoz megadunk egy olyan kis próba projektet, amellyel az egyes osztályok működése, viselkedése azonnal tesztelhető.

A memóriajáték osztálydiagramja



A **CardButton** osztály a játékmezőn megjelenített képért (gomb) felelős osztály.

A **CardButtonGroup** osztály a játékmezőn megjelenített kép csoportért felelős osztály.

A **Player** osztály a játékosok adatait (név, nyeremény pont) kezeli.

A **ConfigDialog** a játékmező beállításáért (kártyacsomag kiválasztása, játékmező mérete) felel.

A **Card** osztály a kártya képet reprezentáló (adat) osztály.

A **CardPack** osztály egy kártyacsomagot megtestesítő (adat) osztály. A kártyacsomag adatai: a kártyacsomag neve, a hátlap képe és az előlapok képe.

A **CardPackSet** kártyacsomagokat (többet!) tartalmazó (adat) osztály.

A **GameState** a játék pillanatnyi állapotát tároló osztály: melyik mezőn, milyen kártyát kell megjeleníteni és az éppen milyen „állapotban” van: *mutatott, elrejtett, megoldott kártya*.

A **GameManager** osztály felügyeli, és irányítja a játékot (új játék, kártyák mutatása, kártyák elrejtése, pontszámok aktualizálása, játékosok tájékoztatása: ki következik, stb.).

A **MainForm** osztály az alkalmazás főablaka (űrlap). Az alkalmazás az űrlapon keresztül tartja a kapcsolatot a felhasználóval. Figyeli a felhasználói interakciókat (pl. kattintás az űrlapra az egérrel), és továbbítja azokat a megfelelő komponensekhez. Ebben az osztályban „születnek”, és „halnak meg” a projekthez tartozó komponensek, itt adjuk meg, hogyan kommunikálnak egymással az alkalmazás egyes komponensei (*connect()*).

A projektben használt „segéd függvények” (Utils)

A *Utils* modulban összegyűjtöttük a projekt különböző osztályaiból hívható segéd függvényeket.

utils.h

```
#include <QString>

namespace Utils
{
    void initRandomGenerator(); //Véletlenszám generátor inicializálása
    int getNextRandomNumber(const int max); //[0..max] közé eső véletlenszám
    QString getApplicationDirPrefix(); //Az éppen futó alkalmazás elérési útvonala
};
```

utils.cpp

```
#include <QDir>
#include <QApplication>
#include <QTime>

#include "utils.h"

void Utils::initRandomGenerator()
{
    QTime t = QTime::currentTime();
    srand(t.hour()*12 + t.minute()*60 + t.second()*60);
}
```

Az *initRandomGenerator()* függvénnyel inicializálhatjuk a véletlenszám generátort.

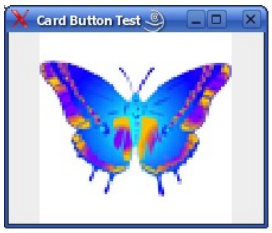
```
int Utils::getNextRandomNumber(const int max)
{
    return (int) (((float) max)*rand()/(RAND_MAX + 0.1));
}
```

A *getNextRandomNumber(const int max)* függvény visszatérési értéke egy [0..max] intervallumba eső véletlen egész szám.

```
QString Utils::getApplicationDirPrefix()
{
    return qApp->applicationDirPath() + QDir::separator();
}
```

A *getApplicationDirPrefix()* függvény visszatérési értéke az az útvonal, ahonnan az alkalmazást elindítottuk.

CardButton osztály



Az űrlapon (ablakon) megjelenő kártyák osztálya, melyet **QPushButton** osztályból állítunk elő származtatással.

cardbutton.h

```
#include <QPushButton>
#include <QPixmap>

class CardButton : public QPushButton
{
    Q_OBJECT
```

A Q_OBJECT makrót mindig meg kell adni, ha saját jel (*slot*) vagy jelkezelő (*signal*) függvényeket definiálunk az osztályban.

```
public:
  CardButton( QWidget * parent = 0);
  ~CardButton();
  void setPixmap (QPixmap pixmap) {mPixmap=pixmap;}
  int getId() {return mId;}
  void setId(int id) {mId = id;}

protected:
    void paintEvent(QPaintEvent * event);
```

Az ősosztály *paintEvent(QPaintEvent *event)* virtuális metódusát felüldefiniáljuk, mert magunk szeretnénk gondoskodni a kártyaképek kirajzolásáról. Az eseménykezelő akkor kapja meg a vezérlést, amikor valami miatt újra kell rajzolni az űrlapot. (pl. takarás után, átméretezésnél).

```
public slots:
    void slotClicked();
```

A *slotClicked()* függvény az osztály saját jelkezelő függvénye.

```
signals:
    void cardButtonClicked(CardButton*);    //Jeladás a kattintásról
    void cardButtonClicked(int);            //Jeladás a kattintásról

private:
   QPixmap mPixmap;
   int mId;
};
```

A **CardButton** osztálynak két adattagja (tulajdonsága) van. Az *mPixmap*-ben a kártyán megjelenítendő képet tároljuk. Az *mId* egy egész szám, amely a kártya azonosítására szolgál. A vezérlőnek két jel függvénye van. Ezt a két jelet kártyára kattintáskor adjuk le. A jelekkel adatokat továbbítunk a kártyánkról. A *cardButtonClicked(CardButton*)* jellel a kártyára mutató pointert, a *cardButtonClicked(int)* jellel az adott kártya azonosítóját továbbítjuk a jel fogadására bejelentkező (*connect()*) objektumoknak.

Az osztálynak fontos feladata a *paintEvent(QPaintEvent * event)* eseménykezelő újradefiniálása. Ez a metódus akkor hajtódik végre, amikor a kártyát valamilyen okból újra kell rajzolni. Azt szeretnénk, ha a kép minden esetben kitöltené a rendelkezésére álló teljes felületet, ezért magunkra vállaljuk a kártyák újrarajzolását. A felület geometriájának megfelelően átméretezzük a képet, és ez lesz az új kártyakép. (Az átméretezés miatt a játékhoz „négyzetes" képeket érdemes megadni.)

cardbutton.cpp

```
#include <QPainter>
#include <QPaintEvent>
```

A programban használt **Qt** osztályokat beillesztjük a programunkba.

```
#include "cardbutton.h"

CardButton::CardButton( QWidget * parent) : QPushButton( parent )
{
    connect(this,SIGNAL(clicked()),this,SLOT(slotClicked()));
}
```

A **CardButton** osztályt a **QPushButton** osztályból állítjuk elő származtatással. A kettőspont után megadott *QPushButton(parent)*-ben meghívjuk az ősosztály konstruktorát, és jelezzük, hogy az ősosztálynak is *parent* legyen a szülője.

```
CardButton::~CardButton()
{}
```

A destruktorban semmi dolgunk, hiszen a Qt „szülő-gyermek" mechanizmusa miatt az űrlapon elhelyezett valamennyi gyerek-vezérlő megszűnik az űrlap bezárásakor.

```
void CardButton::slotClicked()
{
    emit cardButtonClicked(this);
    emit cardButtonClicked(mId);
}
```

A származtatott gombra kattintva az objektum két jelet is küld. A gombot használó objektumok rákapcsolódhatnak erre az jelre.

```
void CardButton::paintEvent (QPaintEvent */*event*/)

{
    int side = qMin(size().width(), size().height());
    QRect rect((size().width()-side) / 2, (size().height()-side) / 2, side, side);
    QPainter painter;
    painter.begin(this);
    painter.drawPixmap(rect,mPixmap);
    painter.end();
}
```

A *PaintEvent()* metódus a gomb újrarajzolásakor hajtódik végre. Az újrarajzolást magunk is kezdeményezhetjük az *update()* vagy a *repaint()* metódussal. A két metódus között az a különbség, hogy az *update()* „összegyűjti" a rajzolási lépéseket, és egyszerre hajtja végre azokat, míg a *repaint()* azonnal frissít. A vezérlő aktuális méretéből (*size()*) meghatároztuk a kisebbik oldalméretet, és erre méreteztük a képet. Definiáltunk egy saját rajzoló eszközt (**QPainter**). A rajzoló eszközt hozzárendeltük a saját vezérlőnkhöz (*painter.begin(this)*), majd a vezérlőnköz tartozó téglalapra (*rect()*) rárajzoltuk a kártya képét (*drawPixmap()*).

Az osztály és az osztályt tesztelő program letölthető a
people.inf.elte.hu/nacsa/qt4/eaf3/mod02/projects/cardbutton címről.

CardButtonGroup osztály

A játékprogramhoz készítünk egy saját vezérlőt, melyben dinamikusan elhelyezhetünk szükséges számú, **CardButton** típusú képet. Ez lesz a **CardButtonGroup** osztály. Ezt az osztályt a **QFrame** osztályból származtatjuk. A **CardButtonGroup** osztály „tudni fogja magáról”, hogy a képeket hány sorban és oszlopban kell elhelyezni. Az osztály *cardButtonList* adattagjában a képekre mutató pointereket tároljuk. A dinamikus átméretezést a **Qt** osztálykönyvtár **QGridLayout** „rácsos elrendező”-jével valósítjuk meg.

cardbuttongroup.h

```
#include <QFrame>
#include <QList>

class QGridLayout[1];
class QPixmap;
class CardButton;
typedef QList<CardButton*> CardButtonList;

class CardButtonGroup : public QFrame {
    Q_OBJECT

public:
    CardButtonGroup(QWidget *parent = 0);
    void newCardButtonGroup(int row, int col);
    CardButton* getCardButton(int i);

public slots:
    void updateCardButtonPixmap(int cardButtonId, QPixmap pixmap);
    void setCardButtonEnabled(int cardButtonId, bool b);

signals:
    void cardButtonClicked(int i); //jel: rákattintottak az i azonosítójú gombra
    void cardButtonGroupChanged(int numRows, int numCols); //jel: megváltozott a gombcsoport

private:

    void insertCardButtons(int row, int col );
    void removeCardButtons();
    void debugCardButtons();

private:
    int mNumRows;
    int mNumCols;
```

1*Gyorsabb lesz a fordítás, ha azokat az osztályokat, amelyekre csak pointerrel hivatkozunk „***class** osztály;” formájában *deklaráljuk, és a teljes osztálydefiníciót az implementációban illesztjük be a programba.*

```
    CardButtonList cardButtonList;
    QGridLayout *cardButtonLayout;
};
```

Az osztályra két jelet definiálunk. A *cardButtonClicked(int i)* jelet akkor adja ki, amikor a gombcsoport valamelyik gombjára rákattintunk. Ez a jel továbbítja a kérdéses gomb azonosítóját. A *cardButtonGroupChanged(int numRows, int numCols)* jelet akkor adjuk ki, amikor a gombcsoport szerkezete (*sora, oszlopa)* megváltozott.

cardbuttongroup.cpp

```
#include <QGridLayout>
#include "cardbutton.h"
#include "cardbuttongroup.h"

CardButtonGroup::CardButtonGroup(QWidget *parent)
    :QFrame(parent), mNumRows(0), mNumCols(0)
{
    cardButtonLayout = new QGridLayout;
    setLayout(cardButtonLayout);
}
```

A konstruktor inicializáló sorában meghívtuk az ősosztály (**QFrame**) konstruktorát, mellyel létrehoztuk a példány ősosztálybeli részét, majd beállítottuk az adattagok kezdeti értékeit. A konstruktor törzsében létrehoztunk egy „rácsos elrendezőt” (*cardButtonLayout*), és beállítottuk, hogy ez az elrendező gondoskodjon az osztály elemeinek elhelyezéséről.

```
void CardButtonGroup::newCardButtonGroup(int row, int col)
{
    removeCardButtons();
    insertCardButtons(row, col );
    mNumRows = row;
    mNumCols = col;
}
```

Ezzel a metódussal az űrlapon új kártyacsoport szerkezetet alakítunk ki. A régi kártyákat leszedjük, az újakat ráhelyezzük az űrlapra.

```
void CardButtonGroup::insertCardButtons(int row, int col )
{
    CardButton *cardButton;
    for (int i = 0; i < row; ++i) {
        for (int j = 0; j< col; ++j) {
            cardButton = new CardButton(this);
            cardButton->setSizePolicy(
                        QSizePolicy( (QSizePolicy::Expanding),(QSizePolicy::Expanding)));
            cardButton->setId(i*col+j);
             cardButtonLayout->addWidget(cardButton,i,j); //Registers cardButton to
            cardButtonList.append(cardButton);     //Add cardButton to container
            connect(cardButton, SIGNAL(cardButtonClicked(int)),
                        this, SIGNAL(cardButtonClicked(int)));
        }
    }
    emit cardButtonGroupChanged(row,col);
    show();
}
```

Ezzel a metódussal helyezzük el az új kártyákat. A függvény meghívása előtt a régi kártyákat le kell szedni (*removeButtons()*). Egyesével létrehozzuk a **CardButton** osztály példányait, beállítjuk a megfelelő tulajdonságokat (azonosító, átméretezhetőség), majd beregisztráljuk a vezérlő elrendezőjénél. A létrehozott **CardButton** példányokra mutató pointereket megőrizzük egy listában. (*cardButtonList*). A *connect()* függvényben a *cardButton* objektum *cardButtonClicked()* jelét összekapcsoljuk a saját (*this*) *cardButtonClicked()* jelünkkel. (Érdemes megfigyelni, hogy **Qt**-ben két jelet is össze lehet kapcsolni.) A kártyacsoport elkészítése után a *show()* metódussal megjelenítjük a kártyákat a képernyőn.

```
void CardButtonGroup::removeCardButtons( )
{
    while (!cardButtonList.isEmpty())
            delete cardButtonList.takeFirst();
}
```

Ez a metódus „leszedi” a kártyákat. A *cardButtonList.takeFirst()* függvénnyel megkapjuk a lista első elemét, amely egy CardButton-ra mutató pointer. Az általa címzett tárhelyet felszabadítjuk (delete). Ezt mindaddig megtesszük, amíg van elem a listában. A tárhelyről kitörölt vezérlő automatikusan eltűnik a képernyőről is.

```
void CardButtonGroup::updateCardButtonPixmap(int cardButtonId, QPixmap pixmap)
{
    cardButtonList.at(cardButtonId)->setPixmap(pixmap);
    cardButtonList.at(cardButtonId)->update();
}
```

A gombok létrehozásakor még nem rendeltünk képet a gombokhoz. A képek hozzárendelése ezzel a metódussal történik. Mivel ez a metódus publikus, a gombcsoport felett rendelkező osztály ezzel a metódussal bármikor tetszőlegesen módosíthatja a gombokon megjelenő képeket.

```
CardButton* CardButtonGroup::getCardButton(int cardButtonId)
{
    return cardButtonList.at(cardButtonId);
}
```

Ez a függvény visszaadja a gombcsoport *cardButtonId*-ik gombjára mutató pointert.

```
void CardButtonGroup::setCardButtonEnabled(int cardButtonId, bool b)
{
    cardButtonList.at(cardButtonId)->setEnabled(b);
}
```

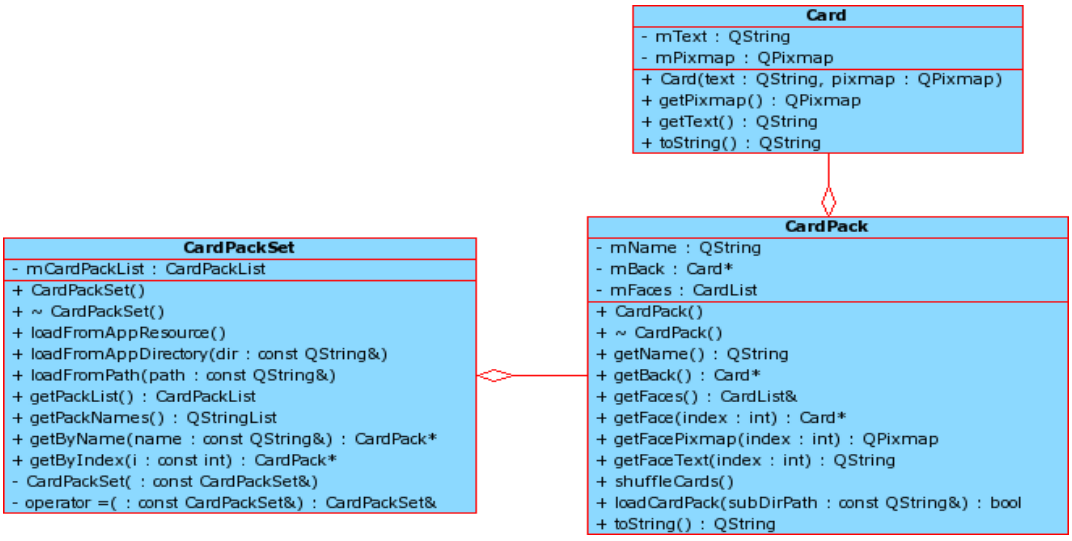
A gombcsoport felett rendelkező osztály ezzel a metódussal beállíthatja a gombok állapotát.

Az elkészített **CardButtonGroup** osztályt felvehetjük a **Qt Designer**-be (*lásd. előző munkafüzet*), és a továbbiakban a többi vezérlőhöz hasonlóan ezt is használhatjuk a felület megtervezésénél.

Az osztály és az osztályt tesztelő program letölthető a
people.inf.elte.hu/nacsa/qt4/eaf3/mod02/projects/cardbuttongroup címről.

Adattároló osztályok

A memóriajáték adatainak kezelésére is készítünk osztályokat. (Card, CardPack,CardPackSet).



A memóriajáték erőforrás fájlja

A kártyacsomag képeit elhelyezhetjük a projekt **pack** alkönyvtárában, de megadhatunk képfájlokat a projekt erőforrásai között is. Ha a képfájlokat erőforrásként adjuk meg, akkor meg kell adnunk egy erőforrás fájlt, (*card.qrc*), amely a projekt része. Az erőforrás megléte nélkül nem fordul le a program. Az erőforrásként megadott kép fájlok elérési útvonalára a programon belül kettősponttal kezdődő útvonallal hivatkozhatunk.

card.qrc

```
<!DOCTYPE RCC>

<RCC version="1.0">
<qresource>
    <file alias="datafiles/packs/default.pack/name.txt">datafiles/packs/default.pack/name.txt</file>
    <file alias="datafiles/packs/default.pack/back.png">datafiles/packs/default.pack/back.png</file>
    <file alias="datafiles/packs/default.pack/01.jpg">datafiles/packs/default.pack/01.jpg</file>
    <file alias="datafiles/packs/default.pack/02.jpg">datafiles/packs/default.pack/02.jpg</file>
    <file alias="datafiles/packs/default.pack/03.jpg">datafiles/packs/default.pack/03.jpg</file>

    . . .

    <file alias="datafiles/packs/default.pack/10.jpg">datafiles/packs/default.pack/10.jpg</file>
    <file alias="datafiles/packs/default.pack/11.jpg">datafiles/packs/default.pack/11.jpg</file>
    <file alias="datafiles/packs/default.pack/12.jpg">datafiles/packs/default.pack/12.jpg</file>
    <file alias="datafiles/packs/default.pack/13.jpg">datafiles/packs/default.pack/13.jpg</file>
    <file alias="datafiles/packs/default.pack/14.jpg">datafiles/packs/default.pack/14.jpg</file>
</qresource>
</RCC>
```

CardPack és Card osztályok

A **cardpack** modulban (**cardpack.h, cardpack.cpp**) két osztályt készítünk el, a **Card** és a **CardPack** osztályokat.

cardpack.h

```
include <QPixmap>

class Card
{
public:
    Card(QString text, QPixmap pixmap);

    QPixmap getPixmap(){return mPixmap;}
    QString getText(){return mText;}
    QString toString();

private:
    QString mText;
    QPixmap mPixmap;

};

typedef QList<Card*> CardList;
```

A **Card** osztály egy konkrét kártya képet reprezentáló osztály. A kép mellett egy szöveget is tárolunk a kártyáról. Vegye észre, hogy a **Card**-nak nincs hátlapja. A kártya hátlapját a kártyacsomagban adjuk meg, kihasználva, hogy egy kártyacsomagon belül minden kártyának ugyanolyan hátlapja van.

```
class CardPack
{
public:
    CardPack();
    ~CardPack();

    QString getName(){return mName;}
    Card* getBack(){return mBack;}
    CardList &getFaces(){return mFaces;}
    Card* getFace(int index);
    QPixmap getFacePixmap(int index);
    QString getFaceText(int index);
    void shuffleCards();
    bool loadCardPack(const QString &subDirPath) ;

    QString toString();

private:
    QString mName;
    Card* mBack; //updated by constructor
    CardList mFaces;
};

typedef QList<CardPack*> CardPackList;
```

A kártyacsomag egy hátlapból (*mBack)*, előlapokból (*mFaces*) és a kártyacsomag nevéből (*mName*) áll. A szokásos „*get-set*” metódusok mellett az osztály keveri a kártyákat, és olvassa be a képfájlokat.

cardpack.cpp

```
#include <QtAlgorithms>

#include <QDir>
#include <QImageReader>
#include <QString>
#include <QFileInfo>
#include <QImage>
#include <QPixmap>
#include <QFile>
#include <QTextStream>
#include <QMessageBox>
```

Beillesztjük a használt **Qt** osztályokat.

```
#include "utils.h"
```

A **utils.h, utils.cpp** modulokban, egy külön névtérben helyeztük el a projektben több helyen is használatos segéd függvényeket.

```
#include "cardpack.h"

Card::Card(QString text, QPixmap pixmap)
    : mText(text), mPixmap(pixmap)
{}
```

A konstruktor inicializáló részében beállítottuk az adattagok kezdeti értékeit.

```
QString Card::toString()
{
    QString ret = "{ Text=" + mText + " PixmapValid = " + QString::number(!mPixmap.isNull()) + " }";
    return ret;
}
```

Készítettünk egy *toString()* metódust a teszteléshez.

```
CardPack::CardPack()
    :mName(""), mBack(0)
{}
```

Az „újszülött” kártyacsomag nevének és a hátlap képnek a kezdőértékét üresre illetve nullára állítottuk.

```
CardPack::~CardPack()
{
    while (!mFaces.isEmpty())
            delete mFaces.takeFirst();
    delete mBack;
}
```

A destruktor a kártyacsomag előlapjainak képeire mutató pointereket tartalmazó lista alapján felszabadítja az előlapok (*mFaces*) és a hátlap által lefoglalt tárhelyet.

```
Card* CardPack::getFace(int index) {
    return mFaces.at(index);
}
```

A *getFace(int index)* függvény visszaadja a kártyacsomag *index*-edik előlapjára mutató pointert.

```
QPixmap CardPack::getFacePixmap(int index)
{
    return mFaces.at(index)->getPixmap();
}
```

A *getFacePixmap(int index)* függvény visszaadja a kártyacsomag *index*-edik előlapjának a képét.

```
QString CardPack::getFaceText(int index)
{
    return mFaces.at(index)->getText();
}
```

A *getFaceText(int index)* függvény visszaadja a kártyacsomag *index*-edik előlapjához megadott szöveget.

```
void CardPack::shuffleCards()
{
    for (int i= 0; i < mFaces.size() * mFaces.size(); ++i)
    {
        int ind1 = Utils::getNextRandomNumber(mFaces.size());
        int ind2 = Utils::getNextRandomNumber(mFaces.size());
        while(ind1 == ind2)
            ind2 = Utils::getNextRandomNumber(mFaces.size());
        qSwap(mFaces[ind1],mFaces[ind2] ); // <QtAlgorithms>
    }
}
```

A *shuffleCards()* metódussal keverhetjük meg a kártyacsomagot (előlapokat). A keveréshez véletlenszám generátort használtunk. Egy adott intervallumba eső szám előállításához a **Utils** modulban definiált *getNextRandomNumber()* függvényt hívtuk meg.

```
QString CardPack::toString()
{
    QString ret = "\nPack name = " + mName + "\n";
    ret += QString("Back: %1\n").arg(mBack->toString());
    for (int i = 0; i < mFaces.size(); ++i) {
        ret += QString("\t  Face #%1: %2 \n").arg(i).arg(mFaces.at(i)->toString());
    }
    return ret;
}
```

A *toString()* metódus előállítja egy kártyacsomag adatait, szöveges formában. Alapvetően adatokat tartalmazó osztályok esetén az ilyen metódusok segítségünkre lehetnek a tesztelésben.

```
bool CardPack::loadCardPack(const QString &subDirPath)
{
```

Ezzel a metódussal töltjük be a *subDirPath* alkönyvtárban található kártyacsomag adatait: név: **name.txt**; hátlap: **back.*** (képfájl); előképek: ***.*** (képfájlok). A kártya példány **text** adatát a fájl nevével töltjük ki.

```
    //read name.txt
    QFile nameFile(subDirPath + "/name.txt");
    if(!nameFile.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        qWarning(QString("Warning: ** name.txt file is missing or not a text file. ** \nsubDirPath \""
            + subDirPath + "\"").toAscii());
        return false;
    }
```

Ha nem létezik az adott alkönyvtárban **name.txt** fájl, akkor „nincs értelme” a kártyacsomagnak.

```
QTextStream in(&nameFile);

while (!in.atEnd()) {
    mName += in.readLine();
}
```

Ha megtaláltuk a **name.txt** fájlt, akkor olvassuk be. Ez a szöveg jeleník meg az űrlap vezérlőin a kártyacsomag azonosítására. (pl. a ComboBox-ban)

```
//Load back.*
 QStringList imageFilters;  //Create filters for images
foreach (QByteArray format, QImageReader::supportedImageFormats())
    imageFilters += "*." + format;
```

A *QImageReader::supportedImageFormats()* metódussal egy **QStringList** típusú listában visszakapjuk, milyen kiterjesztésű képfájlokat támogat a rendszer. A listán végigsétálva előállítunk egy szűrőt (*imageFilters*), melyet a képfájlok összegyűjtésére használunk. (pl. „*.gif” + „*.png” + „*.bmp”)

```
QStringList allFiles;
foreach(QString file, QDir(subDirPath).entryList(imageFilters , QDir::Files))
    allFiles += QFileInfo(QDir(subDirPath),file).baseName();
```

A *QDir(subDirPath).entryList(filters, QDir::Files)* egy olyan lista, amely a *subDirPath*-ban megadott könyvtárban található fájlokat (*QDir::Files* paraméter miatt) tartalmazza. A lista elemein végigsétálva az **allFiles** karakterlánc listában (QStringList) összegyűjtöttük a fájl nevek előtagjait (pont előtti részét).

```
if(!allFiles.contains("back"))
{
    qWarning(QString("Warning: ** Back file is missing from the \n\""
                     +  subDirPath + "\"\ndirectory.").toAscii());
    return false;
}
```

Ha az így összegyűjtött fájl-előnév listában nincs „back”, akkor a kártyacsomag hibás, mert nincs megadva hátlap. Ezt a hiányos csomagot kihagyjuk.

```
QStringList backFiles;
foreach(QString file, QDir(subDirPath).entryList(QStringList("back.*"), QDir::Files))
    backFiles += QFileInfo(QDir(subDirPath),file).fileName();

QString backFile = backFiles.at(0);    //Take the first back.*
mBack = new Card("Back",QPixmap(subDirPath + "/" + backFile));
```

Ha találtunk „back” képfájlt, akkor az első ilyen kép lesz a hátlap.

```
mFaces.clear();
foreach(QString file, QDir(subDirPath).entryList(imageFilters, QDir::Files)){
    if (! (file.startsWith("back.")))
        mFaces.append(new Card(file, QPixmap(subDirPath + "/" + file)));
}
```

Az *imageFilters* szűrővel előállított képfájlok listáján végighaladva előállítjuk az előlapok képeinek listáját (*mFaces*). (A „back” névkezdetű fájlokat kihagytuk.)

```
    return true;
}
```

A Card és a CardPack osztályok és az osztályokat tesztelő program letölthető a people.inf.elte.hu/nacsa/qt4/eaf3/mod02/projects/cardpack címről.

CardPackSet osztály

A **CardPackSet** osztály a projekthez tartozó összes kártyacsomagot (többet) reprezentáló osztály.

cardpackset.h

```
#include "cardpack.h"

class CardPackSet
{
public:
    CardPackSet();
    ~CardPackSet();
    void loadFromAppResource(); //Kártyacsomagok betöltése az erőforrás fájl alapján
    void loadFromAppDirectory(const QString &dir = "/packs"); //Betöltés a pack alkönyvtárból
    void loadFromPath(const QString &path); //A  path útvonalon található képek betöltése
    CardPackList getPackList(){return mCardPackList;}; //Visszaadja a kártyacsomagok listáját
    QStringList getPackNames(); //Visszaadja a kártyacsomagok nevét.
    CardPack* getByName(const QString& name); //Visszaadja egy adott nevű kártyacsomag pointerét
    CardPack* getByIndex(const int i); //Visszaadja egy adott indexű kártyacsomag pointerét

private:
    CardPackSet(const CardPackSet&);
    CardPackSet& operator=(const CardPackSet&);

private:
    CardPackList mCardPackList;
};
```

Az osztály egyetlen adattagja egy lista, amely kártyacsomagokra mutató pointereket tartalmaz. Az osztály, feladatának megfelelően, kényelmes metódusokat kínál a kártyacsomagok és egy-egy kártyacsomag kezeléséhez.

cardpackset.cpp

```
#include <QDir>
#include <QImageReader>
#include <QString>
#include <QFileInfo>
#include <QImage>
#include <QPixmap>
#include <QFile>
#include <QTextStream>
#include <QMessageBox>
```

Beillesztettük a programban használt **Qt** osztályok definícióját.

```
#include "utils.h"
#include "cardpackset.h"
#include "cardpack.h"
```

Beillesztettük a programban használt saját osztályaink definícióját.

```
CardPackSet::CardPackSet()
{}

CardPackSet::~CardPackSet()
{
```

```
        while (!mCardPackList.isEmpty())
                delete mCardPackList.takeFirst();
}
```

A destruktor felszabadítja a kártyacsomagok által lefoglalt tárhelyet. A kártyacsomag lista elemei kártyacsomagra mutató pointerek. A **while** ciklusban az első listaelemben található pointer által mutatott kártyacsomag számára lefoglalt tárhelyet szabadítgatjuk fel mindaddig, amíg van elem a listában.

```
void CardPackSet::loadFromAppResource()
{
        loadFromPath(QString(":/datafiles/packs"));
}
```

A *loadFromAppResource()* metódust akkor hívjuk meg, ha a *card.qrc* erőforrásban definiált képfájlokból szeretnénk elkészíteni a kártyacsomagot. Ennek meglétét a fordító ellenőrzi. Vegye észre az elérési útvonalban szereplő kettőspontot.

```
void CardPackSet::loadFromAppDirectory(const QString &dir)
{
        loadFromPath(Utils::getApplicationDirPrefix() + dir);
}

void CardPackSet::loadFromPath(const QString &path)
{
        mCardPackList.clear();
```

Kinulláztuk a kártyacsomagokat tartalmazó listát.

```
        QDir packsDir(path);
```

A paraméterül kapott alkönyvtárhoz (*path*) létrehoztunk egy **QDir** típusú *packsDir* példányt.

```
        //Create subdirs of packs directory without . and ..
        QStringList subDirs;
        foreach (QString subDir, packsDir.entryList(QDir::Dirs|QDir::NoDotAndDotDot)) {
            subDirs += subDir;
        }
```

Elkészítettük a kártyacsomagokat tartalmazó könyvtár alkönyvtárainak a listáját. A listába nem vettük bele a „.” (pont) és a „..” (pont,pont) nevű alkönyvtárakat.

```
        for (int i = 0; i < subDirs.size(); ++i)
        {
           QString subDirPath = path + "/" + subDirs.at(i);
           QDir subDir(subDirPath);

           CardPack* pack = new CardPack();
           if(pack->loadCardPack(subDirPath))
                   mCardPackList.append(pack);
        }
    }
```

A kártyacsomagokat tartalmazó könyvtár alkönyvtárait használva létrehoztuk a megfelelő kártyacsomag példányokat (*pack*), és hozzávettük őket a kártyacsomagokat tartalmazó listához (*mCardPackList.append(pack)*). Vegye észre, hogy a lista elemeiben nem kártyacsomagot, hanem kártyacsomagra mutató pointert tároltunk. (A pack változó Cardpack* típusú.)

```
QStringList CardPackSet::getPackNames()
{
    QStringList packNames;
    foreach(CardPack* pack, mCardPackList)
        packNames += pack->getName();
    return packNames;
}
```

A *getPackNames()* metódus visszaadja kártyacsomagok nevét.

```
CardPack* CardPackSet::getByName(const QString& name)
{
    CardPack *cardPack = 0;
    foreach(CardPack* pack, mCardPackList)
        if (name == pack->getName())
            cardPack = pack;
    return cardPack;
}
```

A *getByName(const QString& name)* metódus visszaad egy adott nevű (*name*) kártyacsomagra mutató pointert, ha van ilyen, egyébként 0 értékkel tér vissza.

```
CardPack* CardPackSet::getByIndex(const int index)
{
    CardPack *cardPack = 0 ;
    if(index < mCardPackList.size())
        cardPack = mCardPackList.at(index);
    return cardPack;
}
```

A *getByIndex(const int index)* metódus visszaad egy *index* indexű kártyacsomagra mutató pointert, ha van ilyen, egyébként 0 értékkel tér vissza.

Az osztály és az osztályt tesztelő program letölthető a
people.inf.elte.hu/nacsa/qt4/eaf3/mod02/projects/cardpackset címről.

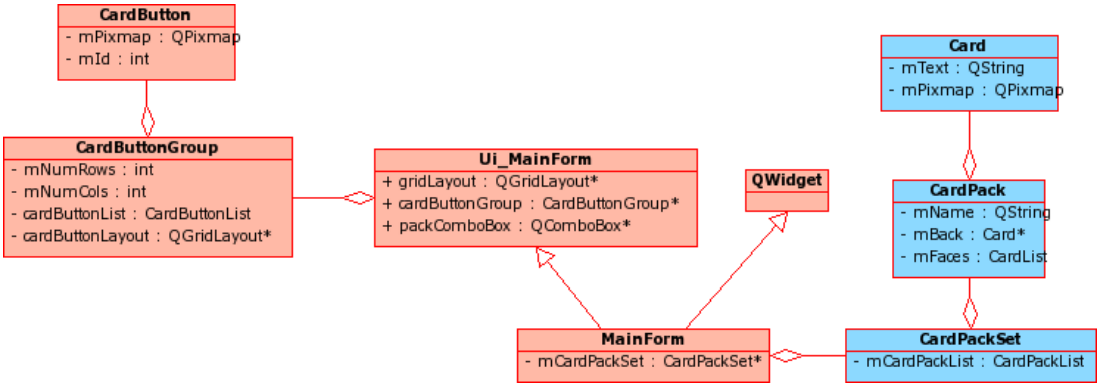
Kártya nézegető alkalmazás (CardViewer)

Készítünk egy programot, amely használja az imént elkészített osztályokat.

Feladat: Készítsünk olyan programot, amely „megkeresi” a projektünk kártyacsomagjait (ha van ilyen), majd megjelenít egy űrlapot, ahol egy legördülő listában megjelenített kártyanevek alapján kiválaszthatunk egy kártyacsomagot, és az űrlapon megjelennek a kiválasztott kártyacsomag előlapjainak képei.

Az űrlapon elhelyezünk egy ComboBox-ot (packComboBox) és a kártyákat tartalmazó kép csoportot (cardButtonGroup). Emlékeztetőül: a CardButtonGroup-ot Qt Designerben úgy tehetjük fel az űrlapra, hogy először rátesszük az ősosztály vezérlőt (QFrame), majd Jobb klikk/Promote: CardButtonGroup. (Ennek a technikának a részletes leírása megtalálható az előző munkafüzetben.)

Az alkalmazás osztálydiagramja



A felület megtervezése után készítse el a **MainForm** osztályt az alábbiak szerint:

mainform.h

```
#include <QWidget>
#include "ui_mainform.h"

class CardPackSet;

class MainForm : public QWidget,  private Ui_MainForm
{
    Q_OBJECT
public:
    MainForm(QWidget *parent = 0);

public slots:
    void on_packComboBox_currentIndexChanged(int index);
    void slotCardClicked(int i);

private:
    CardPackSet* mCardPackSet;

};
```

mainform.cpp

```
#include <QMessageBox>
#include <math.h>
#include "cardpackset.h"
#include "cardbutton.h"
#include "mainform.h"

MainForm::MainForm(QWidget *parent) : QWidget(parent)
{
    setupUi(this);
```

A *setupUi()* metódus helyezi el az űrlapra a **Qt Designerrel** megtervezett elemeket.

```
mCardPackSet = new CardPackSet();
mCardPackSet->loadFromAppDirectory();
if(mCardPackSet->getPackList().empty()){
        QMessageBox::information(0,"pack name",
            "No \"packs\" directory for this application \nLoading default images.");
        mCardPackSet->loadFromAppResource();
}
```

A konstruktor feladata a kártyacsomagokat tartalmazó mCardPackSet inicializálása. A kártyacsomag képeit az alkalmazás **pack** alkönyvtárában található (ha van ilyen) vagy az alkalmazás erőforrásainál megadott fájlok alapján készítjük el.

```
packComboBox->addItems(mCardPackSet->getPackNames());
packComboBox->setCurrentIndex(0);
```

Az mCardPackSet) objektum *getPackNames()* metódusával lekértük a kártyacsomagok névlistáját (**QStringList**), és a nevekkel feltöltöttük a *packComboBox*-ot.

```
connect(cardButtonGroup, SIGNAL(cardButtonClicked(int)), this, SLOT(slotCardClicked(int)));
```

Figyeljük meg, működik-e a képre kattintás. A kattintás jelre rákapcsolódunk a kártya csoportra (*cardButtonGroup*). A jelet a *slotCardClicked(int)* metódusban fogjuk feldolgozni.

```
}

void MainForm::on_packComboBox_currentIndexChanged(int index)
{
   int side = (int) sqrt(mCardPackSet->getPackList().at(index)->getFaces().count()) + 1; //<math.h>
   cardButtonGroup->newCardButtonGroup(side,side);

    for (int i= 0; i< mCardPackSet->getPackList().at(index)->getFaces().count(); i++)
    {
        cardButtonGroup->updateCardButtonPixmap(i,
            QPixmap(mCardPackSet->getPackList().at(index)->getFacePixmap(i)));
        QString text = mCardPackSet->getPackList().at(index)->getFaces().at(i)->getText();
        cardButtonGroup->getCardButton(i)->setToolTip(text.left(text.indexOf(".")));
    }
}
```

Ez a jelkezelő függvény akkor fut le, amikor a *packComboBox*-ból új sort választunk ki. Ehhez a függvényhez nem kellett külön *connect()*-et megadnunk, mert a Qt 4-ben bevezetett névkonvenció miatt (*on_vezérlőNeve_signal*) a **Qt Designer** már beillesztette ezt az összekapcsolást az **Ui_MainForm** osztályba, így csak magát a jelfogadó függvényt kell megírni. A *side* változóban a kártyacsomagban szereplő kártyák száma alapján meghatároztuk, hogy a kártyacsoportnak hány sora és oszlopa legyen, létrehoztuk a kártyacsoportot, majd feltöltöttük a kártyacsomagban található képekkel.

```
void MainForm::slotCardClicked(int i)
{
    QMessageBox::information(this, "Card Group Test Information", "Card Click \nCardId =  "
                + QString::number(i) );
}
```

A képre kattintva kiírjuk, milyen azonosítójú (sorszámú) kártyára kattintottunk.

Az alkalmazás főprogramja (main.cpp)

```
#include <QApplication>

#include "mainform.h"

int main (int argc, char *argv[])
{
	QApplication app(argc,argv);

	MainForm *mainForm = new MainForm;
	mainForm->setWindowTitle("Card Viewer Application");
	mainForm->show();

	return app.exec();
}
```

Az alkalmazás projekt leíró fájlja (cardviewer.pro)

```
TEMPLATE = app
TARGET =
DEPENDPATH += .
INCLUDEPATH += .

# Input
HEADERS += cardbutton.h \
	cardbuttongroup.h \
	cardpack.h \
	cardpackset.h \
	mainform.h \
	utils.h
FORMS += mainform.ui
SOURCES += cardbutton.cpp \
	cardbuttongroup.cpp \
	cardpack.cpp \
	cardpackset.cpp \
	main.cpp \
	mainform.cpp \
	utils.cpp
RESOURCES += cards.qrc
```

Az alkalmazás programjai letölthetők a people.inf.elte.hu/nacsa/qt4/eaf3/mod02/projects/cardviewer címről.

A memóriajáték elkészítését a következő munkafüzetben fejezzük be.

Tartalomjegyzék

A munkafüzet programjai letölthetők a people.inf.elte.hu/nacsa/qt4/eaf3/mod03/projects/ címről.

A munkafüzetben bemutatott programok készítésekor a Qt 4.2.2 verziót használtam.

Készítette: Szabóné Nacsa Rozália

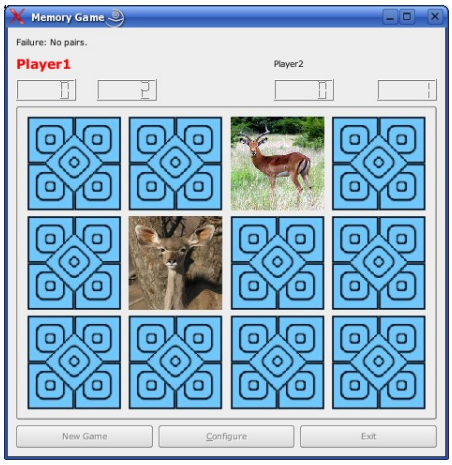
email: nacsa@inf.elte.hu

honlap: people.inf.elte.hu/nacsa

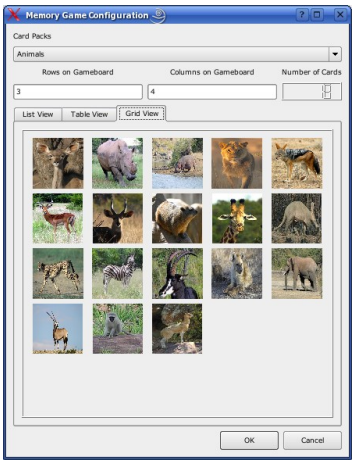
Budapest, 2007. március

Memóriajáték

Feladat: Készítsünk programot, mellyel játszhatjuk a jól ismert memóriajátékot. A játék során a játékmező legyen átméretezhető. A játékhoz szükséges képeket az alkalmazás könyvtárában található képfájlokból állítsuk elő.

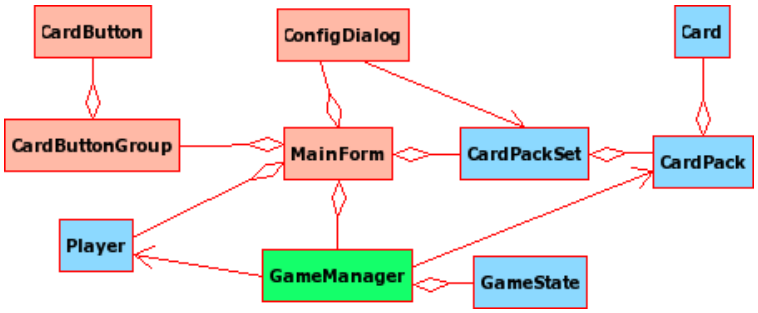


Memóriajáték – játék közben(MainForm)



A játékmező beállítása(ConfigDialog)

A memóriajáték osztálydiagramja



* A **CardButton** osztály a játékmezőn megjelenített képért (gomb) felelős osztály.

* A **CardButtonGroup** osztály a játékmezőn megjelenített kép csoportért felelős osztály.

A **Player** osztály a játékosok adatait (név, nyeremény pont) kezeli.

A **ConfigDialog** a játékmező beállításáért (kártyacsomag kiválasztása, játékmező mérete) felel.

* A **Card** osztály a kártya képet reprezentáló (adat) osztály.

* A **CardPack** osztály egy kártyacsomagot megtestesítő (adat) osztály. A kártyacsomag adatai: a kártyacsomag neve, a hátlap képe és az előlapok képe.

* A **CardPackSet** több kártyacsomagokat (többet!) tartalmazó (adat) osztály.

A **GameState** a játék pillanatnyi állapotát tároló osztály (melyik mezőn, milyen kártyát kell megjeleníteni, és az éppen milyen „állapotban” van: *mutatott, elrejtett, megoldott kártya*).

A **GameManager** osztály felügyeli, és irányítja a játékot. (új játék, kártya mutatás, kártya elrejtés, pontszámok aktualizálása, játékosok tájékoztatása: ki következik, stb.)

A **MainForm** osztály az alkalmazás főablaka (űrlap). Az alkalmazás az űrlapon keresztül tartja a kapcsolatot a felhasználóval. Figyeli a felhasználói interakciókat (pl. kattintás az űrlapra az egérrel), és továbbítja azokat a megfelelő komponensekhez. Ebben az osztályban „születnek", és „halnak meg" a projekthez tartozó komponensek, itt adjuk meg, hogyan kommunikálnak egymással az alkalmazás egyes komponensei (*connect()*).

A csillaggal megjelölt osztályokat az előző modulban már elkészítettük.

Ebben a modulban elkészítjük a még hiányzó osztályokat, és befejezzük az alkalmazást.

Player osztály

A játékosok adataiért felelős osztály.

player.h

```
#include <QObject>

class Player : public QObject
{
    Q_OBJECT

public:
    Player(const QString& name, int hits = 0, int failers = 0);

  QString getName() const {return mName;}
  void setName(const QString& name) {mName = name;}

  uint getNumHits() const {return mNumHits;}
  void setNumHits(const uint hits){mNumHits = hits;}

  uint getNumFailures() const {return mNumFailures;}
  void setNumFailures(const uint failers){mNumFailures = failers;}

  void incrementHits();
  void incrementFailures();

  QString toString();

signals:
    void numHitsChanged(Player* player);  //Jelzés: nyereménypont megváltozott
    void numFailuresChanged(Player* player);  //Jelzés: hibapont megváltozott

private:
    QString mName;
    uint mNumHits;
    uint mNumFailures;
};

typedef QList<Player*> PlayerList;
```

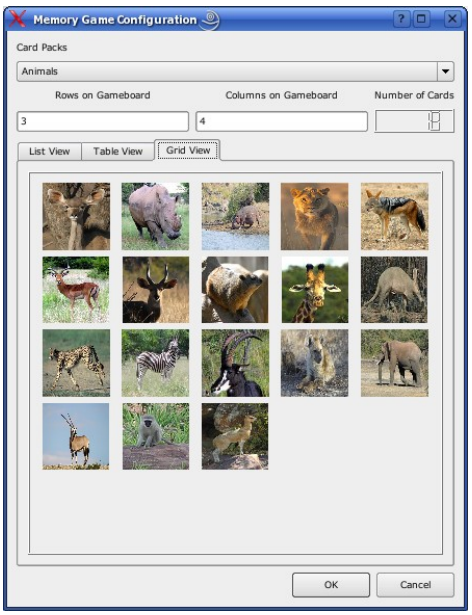
Az osztályban három adatot tárolunk: a játékos nevét, nyereménypontjait és hibapontjait. Az adatokat kezelő szokásos függvények mellett definiálunk két jelet (signals), melyekkel a játékos megváltozott nyereménypontját és hibapontját továbbítjuk.

player.cpp

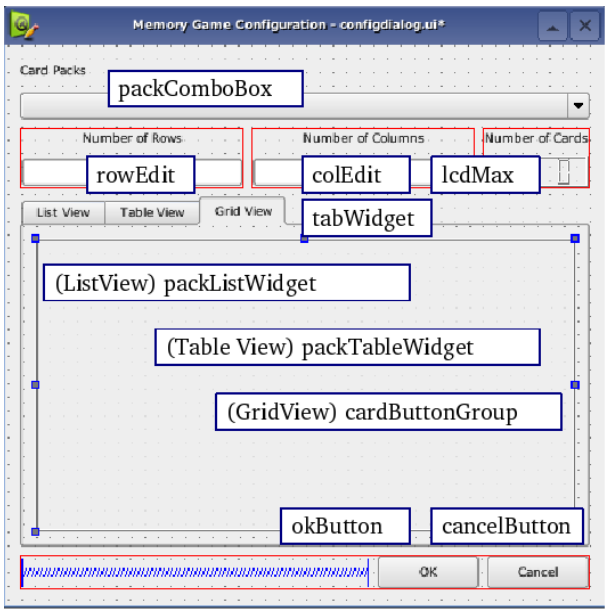
```
#include "player.h"

Player::Player(const QString& name, int hits, int failures)
    : QObject(), mName(name), mNumHits(hits), mNumFailures(failures)
{}

void Player::incrementHits()
{
    mNumHits++;
    emit numHitsChanged(this);
}

void Player::incrementFailures()
{
    mNumFailures++;
    emit numFailuresChanged(this);
}

QString Player::toString()
{
    QString ret = "";
    ret += "\n Name= " + mName + "\t Hits= " + QString::number(mNumHits) + "\t Failures= " +
QString::number(mNumFailures);
    return ret;
}
```

Játékfelület kialakítása (ConfigDialog osztály)

Készítünk egy olyan dialógus ablakot, melyben megadhatjuk, melyik kártyacsomaggal szeretnénk játszani, és mekkora legyen a játékmező (*sor,oszlop*).



A dialógus ablak használat közben



A dialógusablak tervezés közben

A felületen elhelyezett elemek

Típus	*Név(objectName)*	*Beállítások, megjegyzés*
QWidget	ConfigDialog	windowTitle: Memory Game Configuration
QComboBox	packComboBox	
QLineEdit	rowEdit	
QLineEdit	colEdit	
QLCDNumber	lcdMax	
QTabWidget	tabWidget	currentIndex = 0; currentTabText = List View; currentTabName = tab1
QListView	packListWidget	
QTabWidget	tabWidget	currentIndex = 1; currentTabText = Table View; currentTabName = tab2
QTableWidget	packTableWidget	
QTabWidget	tabWidget	currentIndex = 2; currentTabText = Grid View; currentTabName = tab3
CardButtonGroup	cardButtonGroup	QFrame->Jobb klikk->Promotion: CardButtonGroup
QPushButton	okButton	text = &OK
QPushButton	cancelButton	text = &Cancel

Először **Qt Designerrel** megtervezzük a felületet, melyet **configdialog.ui** néven mentünk el. Ezután a felülettervet használva, származtatással, „szövegszerkesztővel” elkészítjük a **ConfigDialog** osztályt.

configdialog.h

```
#include "ui_configdialog.h"

class CardPackSet;

class ConfigDialog : public QDialog, public Ui_ConfigDialog
```

A **ConfigDialog** osztályt a **QDialog** és a tervezővel elkészített **Ui_ConfigDialog** osztályokból származtatjuk.

```
{
    Q_OBJECT
```

Megadtuk a **Q_OBJECT** makrót, mert az osztály használ saját jelet (*signal*) és jelkezelőt (*slot*). Ha bármelyik modulunkban a **Q_OBJECT** makrót először adtuk meg, mindig végre kell hajtani a **qmake -project** parancsot a **make** előtt, mert csak így kerülnek be a makró által használt függvények a programunkba. Ha ezt elmulasztjuk, akkor a fordító „nem definiált függvény” hibaüzenetet küld.

```
public:
    ConfigDialog(CardPackSet* cardPackSet, int packIndex, int row, int col, QWidget * parent=0);

public slots:
    void on_okButton_clicked();
    void on_cancelButton_clicked();
    void on_packComboBox_currentIndexChanged(int index);
```

Deklaráltuk az osztály jelkezelő függvényeit.

```
private:
    void updatePackComboBox();
    void updateNumCards();
    void updatePackListWidget();
    void updatePackTableWidget();
    void updatePackCardGrid();
    void updateView();
```

Deklaráltuk a vezérlők aktualizálását végző privát függvényeket.

```
private:
    CardPackSet *mCardPackSet;
    int mPackIndex;
```

Az osztálynak két adattagja van. Az *mCardPackSet* a projekthez tartozó kártyacsomagokra mutat. Az *mPackIndex* az aktuális kártyacsomag indexe.

```
};
```

configdialog.cpp

```
#include <QMessageBox>
#include <QLineEdit>
#include <QListWidget>
#include <QListWidgetItem>
#include <math.h>

#include "cardpackset.h"
#include "cardbutton.h"
#include "configdialog.h"

ConfigDialog::ConfigDialog(CardPackSet *cardPackSet, int packIndex, int row, int col, QWidget * parent)
    : QDialog(parent),mCardPackSet(cardPackSet), mPackIndex(packIndex)
{
    setupUi(this);
```

A *setupUi()* metódussal elhelyeztük a Qt Designerrel tervezett felület elemeit a dialógusablakra.

```
    rowEdit->setText(QString::number(row));
    colEdit->setText(QString::number(col));
    packComboBox->setCurrentIndex(mPackIndex);
    updateNumCards();
    updatePackComboBox();
}

void ConfigDialog::on_okButton_clicked()
{
    int i = rowEdit->text().toInt() * colEdit->text().toInt();
    int numCards = mCardPackSet->getPackList().at(mPackIndex)->getFaces().count() * 2;
    if(i == 0 || i > numCards || i%2 != 0)
        QMessageBox::information(0,"MainForm","Too big or not pair");
    else
        accept();
}
```

Kilépéskor ellenőrizzük, hogy a játékmező mérete megfelel-e a kiválasztott kártyacsomagnak. Mivel párosával rakjuk le a kártyákat, így a játékmező mérete legyen páros, és ne legyen nagyobb, mint a kártyacsomagban található kártyaszám duplája.

```
void ConfigDialog::on_cancelButton_clicked()
{
    reject();
}

void ConfigDialog::on_packComboBox_currentIndexChanged(int index)
{
    mPackIndex = index;
    updateView();
}
```

Ha új elemet (új kártyacsomagot) választottunk ki a **ComboBox**-ban, akkor frissítsük az adatainkat.

```
void ConfigDialog::updatePackComboBox()
{
    packComboBox->addItems(mCardPackSet->getPackNames());
    packComboBox->setCurrentIndex(mPackIndex);
}
```

Ezzel a metódussal töltjük ki a **ComboBox**-ot a kártyacsomag nevekkel.

```
void ConfigDialog::updateView()
{
    updateNumCards();
    updatePackListWidget();
    updatePackTableWidget();
    updatePackCardGrid();

}
```

updateView() metódus frissíti a dialógusablakon található összes vezérlőket.

```
void ConfigDialog::updatePackListWidget()
{
    packListWidget->clear();
    for (int i= 0; i< mCardPackSet->getPackList().at(mPackIndex)->getFaces().count(); ++i)


    {
        QListWidgetItem *item = new QListWidgetItem();

        QPixmap pixmap(mCardPackSet->getPackList().at(mPackIndex)->getFacePixmap(i));

        item->setIcon(QIcon(pixmap));

        QString str = mCardPackSet->getPackList().at(mPackIndex)->getFaceText(i);
        item->setData(Qt::DisplayRole,str);
        packListWidget->insertItem(i,item);
    }
}
```

A Lista nézet frissítéséhez végigmentünk az aktuális kártyacsomag előlapjain. Minden előlaphoz létrehoztunk egy **QListWidgetItem** elemet. Létrehoztunk egy **QPixmap** példányt az előlaphoz, majd ikonként betöltöttük a képet a listaelembe. A listaelem szöveg részét kitöltöttük a kártyához rendelt szövegrésszel, és a most létrehozott listaelemet hozzáadtuk a listához.

```
void ConfigDialog::updatePackTableWidget()
{
    packTableWidget->clear();
    for (int i= 0; i< mCardPackSet->getPackList().at(mPackIndex)->getFaces().count(); ++i)
    {
        QTableWidgetItem *item = new QTableWidgetItem();
        int side = (int) sqrt(mCardPackSet->getPackList().at(mPackIndex)->getFaces().count()) + 1;
        packTableWidget->setRowCount(side);
        packTableWidget->setColumnCount(side);
        QPixmap pixmap(mCardPackSet->getPackList().at(mPackIndex)->getFacePixmap(i));
        item->setIcon(QIcon(pixmap));
        packTableWidget->setItem((int)(i/side), (int)(i%side),item);
    }
}
```

A Tábla nézet frissítéséhez végigmentünk az aktuális kártyacsomag előlapjain. Minden előlaphoz létrehoztunk egy **QTableWidgetItem** elemet. Kiszámoltuk, hány sora, és oszlopa legyen a táblának (*side*), és beállítottuk a tábla méreteit az így kiszámított értékre. Létrehoztunk egy **QPixmap** példányt az előlaphoz, majd ikonként betöltöttük a képet a táblaelembe. Ezután hozzáadtuk a frissen létrehozott táblaelemet a táblához.

```
void ConfigDialog::updateNumCards()
{
    lcdMax->display(mCardPackSet->getPackList().at(mPackIndex)->getFaces().count());
}
```

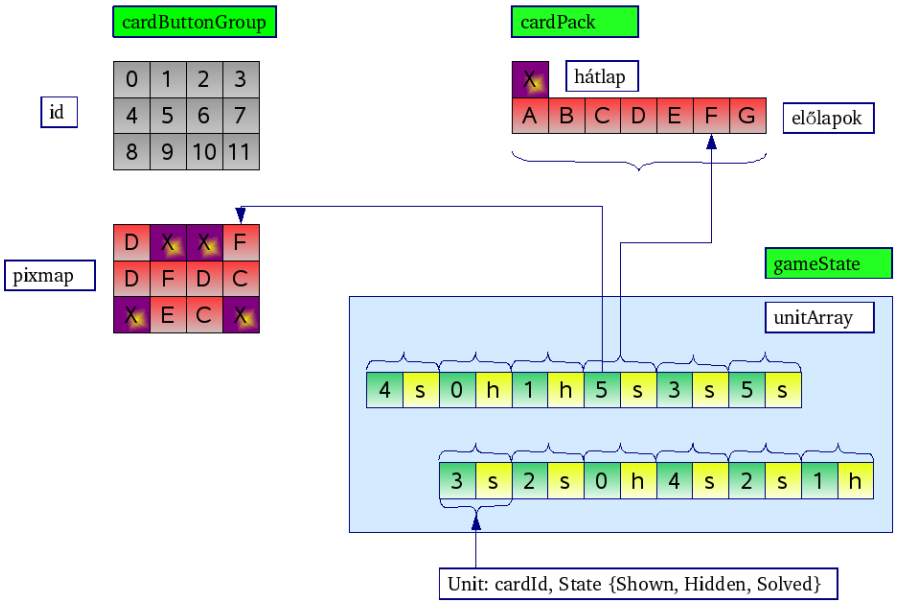
Az *lcd* kijelzőben megjelenítjük az aktuális kártyacsomagban található előlapok számát.

```
void ConfigDialog::updatePackCardGrid( )
{
    int side = (int) sqrt(mCardPackSet->getPackList().at(mPackIndex)->getFaces().count()) + 1;
    cardButtonGroup->newCardButtonGroup(side,side);

    for (int i= 0; i< mCardPackSet->getPackList().at(mPackIndex)->getFaces().count(); ++i)
    {
        cardButtonGroup->updateCardButtonPixmap(i,
            QPixmap(mCardPackSet->getPackList().at(mPackIndex)->getFacePixmap(i)));
        QString text = mCardPackSet->getPackList().at(mPackIndex)->getFaces().at(i)->getText();
        cardButtonGroup->getCardButton(i)->setToolTip(text.left(text.indexOf(".")));
    }
}
```

A Rács (Grid) nézet frissítéséhez először kiszámoltuk, mekkora legyen a rács oldala (*side*), majd a dialogusablakon található *cardButtonGroup* vezérlő méretét beállítottuk erre a méretre. Ezután végigmentünk az aktuális kártyacsomag előlapjain, és a képekkel frissítettük a *cardButtonGroup* egyes képeit. A képekhez hozzárendeltünk egy „mini súgót” (*tool tip*), így a képek fölött mozogva megjelenik a képeket tartalmazó fájl neve. Ha ügyesen adjuk meg a fájlok nevét, ez segíthet a képek felismerésében.

Az osztály és az osztályt tesztelő program letölthető a people.inf.elte.hu/nacsa/qt4/eaf3/mod03/projects/config címről.

Játékállás nyilvántartása (GameState osztály)



A játékállást egy tömbben (*unitArray*) adminisztráljuk. A tömb egyes elemeiben (*unit*) tároljuk, hogy az asztalon (*cardButttonGroup*) található kártyák közül a„tömb-index-edik” kártya képe a kártyacsomag (*cardPack*) melyik kártyájának (előlapnak) felel meg, és mi az állapota (Shown, Hidde, Solved).

A fenti ábra éppen azt a pillanatot mutatja, hogy az asztalon található kártyák közül a 3-as indexű kártya előlapjának megjelenítéséhez a kártyacsomagból az 5-ös indexű előlapot kell használni. A kártya állapota „Shown”, ezért az előlapot kell mutatni.

A játékállás kezelését a GameState osztály végzi.

gamestate.h

```
#include <QVector>

class GameState {

public:
    struct Unit {
        enum State {Hidden, Shown, Solved};
        State state;  /A kártya állapota
        int cardId;  //Az előlap kártyacsomagbeli sorszáma (azonosítója)
    };

    typedef QVector<Unit> UnitArray;

    GameState(int size);
    ~GameState();

    void newGameState();
    void newGameState(int size);
```

```
        Unit::State getState(int unitPos) const;
        void setState (int unitPos, Unit::State state);
        int getCardId(int unitPos) const;
        void setCardId (int unitPos, int cardId);

        int getSize()const {return mSize;}
        bool match(int unitPos1, int unitPos2);
        int numUnsolvedCards() const;

        QString toString();

    private:
        void shuffleUnits();
        void initUnitArray();


    private:
        UnitArray mUnitArray;
        int mSize;
};
```

gamestate.cpp

```
#include <QtAlgorithms>
#include "utils.h"
#include "gamestate.h"

GameState::GameState(const int size)  : mSize(size)
{
    Utils::initRandomGenerator();
    newGameState(mSize);
}

GameState::~GameState()
{}

void GameState::newGameState()
{
    newGameState(mSize);
}

void GameState::newGameState (int size)
{
    mSize = size;
    initUnitArray();
    shuffleUnits();
}

void GameState::initUnitArray()
{
    mUnitArray.resize(mSize);
    int cardId = 0;
    int i= 0;
    while(i < mUnitArray.size()-1)
    {
```

```
            mUnitArray[i].state = mUnitArray[i+1].state = Unit::Hidden;
            mUnitArray[i].cardId = mUnitArray[i+1].cardId = cardId;
            i += 2;
            ++cardId;
        }
    }
```

Az *initUnitArray()* a tömböt kártya párokkal tölti fel.

```
void GameState::shuffleUnits()
{
    for (int i= 0; i < mUnitArray.size() * mUnitArray.size(); ++i)
    {
        int ind1 = Utils::getNextRandomNumber(mUnitArray.size());
        int ind2 = Utils::getNextRandomNumber(mUnitArray.size());
        while(ind1 == ind2)
            ind2 = Utils::getNextRandomNumber(mUnitArray.size());
        qSwap(mUnitArray[ind1],mUnitArray[ind2] ); // <QtAlgorithms>
    }
}
```

A *shuffleUnits()* véletlenszerűen megkeveri a kártyát (*a kártyaállást rögzítő tömb elemeit*).

```
bool GameState::match(int unitPos1, int unitPos2)
{
    Q_ASSERT(unitPos1 < mSize);
    Q_ASSERT(unitPos2 < mSize);
    return (mUnitArray[unitPos1].cardId == mUnitArray[unitPos2].cardId) ? true : false;
}
```

A *match(int unitPos1, int unitPos2)* megnézi, hogy a paraméterül kapott két kártya azonos-e.

```
int GameState::numUnsolvedCards() const
{
    int unsolvedCards = 0;
    for (int i = 0; i < mSize; ++i) {
        if(getState(i) != Unit::Solved)
            ++unsolvedCards;
    }
    return unsolvedCards/2;
}
```

A *numUnsolvedCards()* visszaadja, hány kártya van még játékban.

```
GameState::Unit::State GameState::getState(int unitPos) const
{
    Q_ASSERT(unitPos < mSize);
    return (mUnitArray[unitPos]).state;
}
```

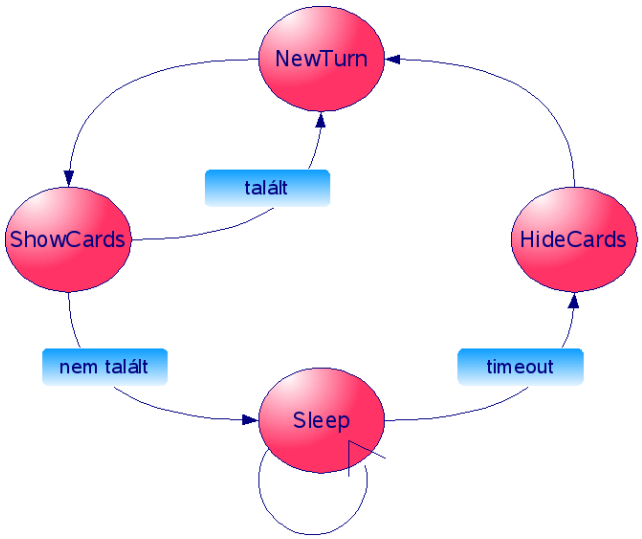
A *getState(int unitPos)* visszaadja, hogy egy adott pozícióban lévő kártya milyen állapotban van.

```
void GameState::setState (int unitPos, Unit::State state)
{
    Q_ASSERT(unitPos < mSize);
    mUnitArray[unitPos].state = state;
}
```

```
int GameState::getCardId(int unitPos) const
{
	Q_ASSERT(unitPos < mSize);
	return mUnitArray[unitPos].cardId;
}

void GameState::setCardId (int unitPos, int cardId)
{
	Q_ASSERT(unitPos < mSize);
	mUnitArray[unitPos].cardId = cardId;
}

QString GameState::toString()
{
	QString ret = "";
	for (int i = 0; i < mSize; ++i) {
		ret += QString("Card Look Id: %1, State %2
\n").arg(mUnitArray[i].cardId).arg(mUnitArray[i].state);
	}
	return ret;
}
```

Az osztály és az osztályt tesztelő program letölthető a
people.inf.elte.hu/nacsa/qt4/eaf3/mod03/projects/gamestate címről.

A „Játékmenedzser” (GameManager osztály)



A memóriajáték állapotdiagramja

A memóriajátéknak négy állapota van. A játék **NewTurn** állapotból indul. **NewTurn** állapotból kártyára kattintva jutunk át **ShowCards** állapotba. Újabb kattintásra **ShowCards** állapotból egyező kártyák esetén **NewTurn**, eltérő kártyák esetén **Sleep** (*a két kártyát egy kis időre felfordítva tartjuk*) állapotba jutunk. **Sleep** állapotból az időzítő *timeout()* jelzése viszi át a játékot **HideCards** állapotba. **HideCards** állapotból képre kattintással jutunk át **NewTurn** állapotba.

gamemanager.h

```
#include <QObject>
#include <QPixmap>

class GameState;
class CardPack;
class QTimer;

#include "player.h"

class GameManager : public QObject
{
    Q_OBJECT

public:
    GameManager (int numRows, int numCols, PlayerList players);
    ~GameManager();

    void setCurrentPack(CardPack *pack) {mCurrentPack = pack;}
    PlayerList getPlayerList(){return mPlayers;}
    int getCurrentPlayerIndex(){return mCurrentPlayerIndex;}
    Player* getCurrentPlayer() {return mPlayers.at(mCurrentPlayerIndex);}
    void setCurrentPlayerIndex(int playerIndex);

public slots:
    void newGame(int numRows, int numCols);
    void handleCardButtonClick(int cardId);

protected slots:
    void stateTransition();
    void wakeUp();

signals:
    void cardButtonEnabledChanged(int cardId, bool enable = true);
    void cardButtonPixmapChanged(int cardId, QPixmap pixmap);
    void currentPlayerChanged(Player* player);
    void gameOver(PlayerList players);
    void playerInfoChanged(Player* player, QString info);

private:
    void updateCard(int cardId);
    void checkResult();
    int nextPlayerIndex();
    QPixmap createSolvedPixmap(QPixmap pixmap);
    GameManager(const GameManager&);
    GameManager& operator=(const GameManager&);

private: //Data members

    CardPack* mCurrentPack;
    GameState* mGameState;
    PlayerList mPlayers;

    int mCurrentPlayerIndex;
    enum State {NewTurn, ShowCards, HideCards, Sleep};
    State mState;
```

```
    int mLastCardId;
    int mCardId1;
    int mCardId2;
    QTimer* timer;
};
```

gamemanager.cpp

```
#include <QPixmap>
#include <QTimer>
#include <QMessageBox>
#include <QPainter>

#include "utils.h"
#include "cardpack.h"
#include "gamestate.h"
#include "gamemanager.h"

GameManager::GameManager (int numRows, int numCols, PlayerList players)
    :mCurrentPack(NULL), mGameState(NULL), mPlayers(players), mCurrentPlayerIndex(0)
{

    mGameState = new GameState(numRows*numCols);

    timer = new QTimer(this);
    timer->setSingleShot (true);
    timer->stop();

    connect(timer, SIGNAL(timeout()), this, SLOT(wakeUp()));
}
```

A játékállás nyilvántartására létrehozzuk a *sor*oszlop* méretű **GameState** típusú *mGameState* objektumot. A kártya forgatásához (mutat, rejt) létrehozunk egy időzítőt. Az időzítő legyen „egyszer jelző”, és a GameManager létrejöttekor még nem kell jeleznie (*stop()*). A *connect()* függvényben az időzítő *timeout()* jelét összekapcsoljuk az *mGameState* objektum *wakeUp()* jelkezelő függvényével. Az összekapcsolás hatására a *wakeUp()* függvény a *timeout*() jelre automatikusan végrehajtódik.

```
GameManager::~GameManager()
{
    delete mGameState;
}
```

A destruktorban a **new** operátorral lefoglalt, szülővel nem rendelkező tárhelyet felszabadítjuk.

```
void GameManager::newGame(int numRows, int numCols)
{
    mState = NewTurn;
    if(mCurrentPack) mCurrentPack->shuffleCards();

    setCurrentPlayerIndex(0);
    emit playerInfoChanged(mPlayers.at(0),"Select first card!");

    mGameState->newGameState(numRows*numCols);
    for(int i = 0; i < mGameState->getSize(); ++i) updateCard(i);
}
```

A **GameManager** létrejöttekor **NewTurn** állapottal kezdünk. A kártyát megkeverjük. A kezdő játékos a játékos lista 0-dik indexű játékosa lesz. A játékosunkra vonatkozó tájékoztató szöveget jellel (signal) továbbítjuk. Erre a jelre a jelfeldolgozásában érdekelt objektumok kapcsolódhatnak rá.

```
void GameManager::handleCardButtonClick(int cardId)
{
    if(mState == Sleep) return; //Ignore click in sleep state
    if(mGameState->getState(cardId) == GameState::Unit::Hidden) //Only hidden cards are valid
    {
        mLastCardId = cardId;
        stateTransition();
    }
}
```

A handleCardButtonClick(int cardId) jelkezelő képre kattintáskor kerül végrehajtásra. Ha „alszunk”, (azaz Sleep állapotban vagyunk, a két kártyát éppen felfelé fordítva tartjuk, és a játékunk az időzítő *timeout()* jelzésére vár), akkor nem csinálunk semmit. Ha „nem alszunk”, akkor megjegyezzük, utoljára melyik kártyára kattintottak, és átadjuk a vezérlést az állapot átmenetet kezelő függvénynek (*stateTransition()*).

```
void GameManager::wakeUp()
{
    mState = HideCards;
    stateTransition();
}
```

Ez a „jelvevő” metódus akkor kerül végrehajtásra, amikor az időzítőnk (*timer*) *timeout()* jelzést küld. Ekkor a játék állapotát **Hidden** állapotra állítjuk, majd átadjuk a vezérlést az állapot átmenetet koordináló függvénynek (*stateTransition()*).

```
void GameManager::stateTransition()
{
    switch(mState) {

    case(NewTurn):

        mState = ShowCards;
        mCardId1 = mLastCardId;
        mGameState->setState(mCardId1, GameState::Unit::Shown);
        updateCard(mCardId1);
        emit playerInfoChanged(getCurrentPlayer(),"Select second card!");
        break;
```

A **NewTurn** állapotot a **ShowCards** állapot követi. Az *mLastCardId*-ben eltároljuk, hogy utoljára melyik kártyára kattintottak. A kártya állapotát „mutatott kártyára” (Shown) állítjuk. Az új állapotnak megfelelően frissítjük a kártya megjelenést, majd kibocsájtjuk azt a jelet, amely jelzi, hogy megváltozott a játékos tájékoztató szövege.

```
    case (ShowCards):
        if (mCardId1 == mLastCardId) return;
        mCardId2 = mLastCardId;
        if(mGameState->match(mCardId1,mCardId2)){ //match
            mState = NewTurn;
            mGameState->setState(mCardId1, GameState::Unit::Solved);
            mGameState->setState(mCardId2, GameState::Unit::Solved);
            updateCard(mCardId1);
            updateCard(mCardId2);
```

```
            getCurrentPlayer()->incrementHits();
            emit playerInfoChanged(getCurrentPlayer(),"Select first card!");
            checkResult();
```

Ha többször is ugyanarra a kártyára kattintottak, akkor ne csináljunk semmit (return). Ha a második kártya olyan, mint az előző (*match()*), akkor a két kártya kikerül a játékból (*Solved és update*), és ugyanaz a játékos játszhat még egy fordulót. Az újabb forduló elindítása előtt ellenőrízni kell, maradt-e még kártya az asztalon (*checkresult()*).

```
        }else { //failure
            emit playerInfoChanged(getCurrentPlayer(),"Failure: No pairs.");
            mState = Sleep;
            mGameState->setState(mCardId2, GameState::Unit::Shown);
            updateCard(mCardId2);
            getCurrentPlayer()->incrementFailures();
            timer->start(1500);
```

Ha a két kártya nem azonos (!*match()*), akkor egy rövid ideig (*1500 milisecundum*) felfordítva (*Shown)* hagyjuk a kártyát, Ehhez elindítjuk az időzítőnket (*timer*), és beállítjuk a **Sleep** állapotot. Ebből az állapotból az időzítő *timeout()* jelét feldolgozó *wakeUp()* függvény viszi át a játék állapotát **HideCards** állapotba.

```
            }
            break;

        case (HideCards):
            mState = NewTurn;
            mGameState->setState(mCardId1, GameState::Unit::Hidden);
            mGameState->setState(mCardId2, GameState::Unit::Hidden);
            setCurrentPlayerIndex(nextPlayerIndex());
            updateCard(mCardId1);
            updateCard(mCardId2);
            emit playerInfoChanged(getCurrentPlayer(),"Select first card!");
            break;
    }
```

Beállítjuk, hogy a **HideCard** állapotot majd a **NewTurn** állapot követi. A kártyákat lefordítjuk (**Hidden**), vesszük a következő játékost, és játékos tájékoztató szöveggel jelet bocsájtunk ki.

```
}

void GameManager::updateCard(int cardId)
{
    Q_ASSERT(mCurrentPack != NULL);

    QPixmap pixmap;
    switch(mGameState->getState(cardId)) {

    case GameState::Unit::Shown:
        pixmap = mCurrentPack->getFacePixmap(mGameState->getCardId(cardId));
        emit cardButtonEnabledChanged(cardId, false);
        break;
```

Ha a kártya „mutatott” (*Shown*) állapotban van, akkor az aktuális kártyacsomag előlapjai alapján elkészítjük a kártya képét (*pixmap*), és jelezzük, hogy a kártyára nem lehet kattintani.

```
    case GameState::Unit::Hidden:
```

```
            pixmap = mCurrentPack->getBack()->getPixmap();
            emit cardButtonEnabledChanged(cardId, true);
            break;
```

Ha a kártya „rejtett” (*Hidden*) állapotban van, akkor az aktuális kártyacsomag hátlapja alapján elkészítjük a kártya képét (*pixmap*), és jelezzük, hogy a a kártyára lehet kattintani.

```
        case GameState::Unit::Solved:
            pixmap = createSolvedPixmap(
                mCurrentPack->getFacePixmap(mGameState->getCardId(cardId)));
            emit cardButtonEnabledChanged(cardId, false);
            break;
```

Ha a kártya „megoldott” (*Solved*) állapotban van, akkor createSolvedPixmap() metódussal elkészítjük a kártya képét (*pixmap*), és jelezzük, hogy a a kártyára nem lehet kattintani.

```
        }
        emit cardButtonPixmapChanged(cardId, pixmap);
```

Leadjuk a jelet, hogy a kártyakép megváltozott. A jel segítségével továbbadjuk, hogy melyik kártya változott meg, és mi az új képe.

```
}

QPixmap GameManager::createSolvedPixmap(QPixmap pixmap)
{
   QRect rect = pixmap.rect();
      QPainter painter(&pixmap);
      QPalette palette;
   QColor eraseColor = palette.color(QPalette::Background);
   painter.setBrush(eraseColor);
   painter.fillRect(rect, eraseColor);
   return pixmap;
 }
```

Ez a függvény állítja elő az a képet, amelyet meg szeretnénk jeleníteni a játékból kikerült (megoldott) kártyákon. A kép elkészítéséhez beállítjuk az ecsetet a háttér „radírozó” (*erase*) színére, és egy ilyen színű téglalapot rajzolunk.

```
 void GameManager::checkResult()
{
      if(mGameState->numUnsolvedCards() == 0)
            emit gameOver(mPlayers);
}
```

Megnézzük, hogy kikerült-e minden kártya a játékból, mert akkor vége van a játéknak. Ha a játéknak vége, akkor erről jelet küldünk.

```
void GameManager::setCurrentPlayerIndex(int playerIndex) {
      mCurrentPlayerIndex = playerIndex;
      emit currentPlayerChanged(getCurrentPlayer());
}
int GameManager::nextPlayerIndex() {
      ++mCurrentPlayerIndex;
      if(mCurrentPlayerIndex >= mPlayers.count())
            mCurrentPlayerIndex = 0;
      return mCurrentPlayerIndex ;
```

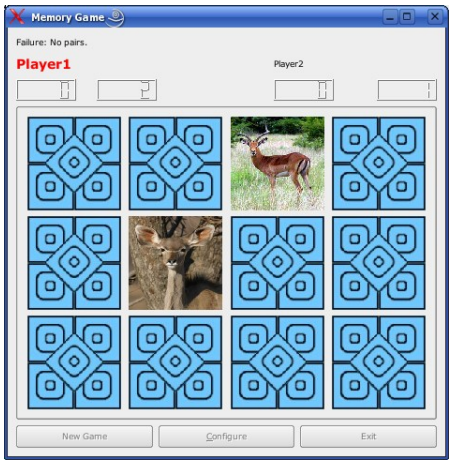
```
}
```

Programunk kihasználja, hogy pontosan két játékos van, és ennek ismeretében jelöli ki a következő játékost. Mivel a játékosokat listában tároljuk, így kis módosítással ez a program több játékosra is átírható.

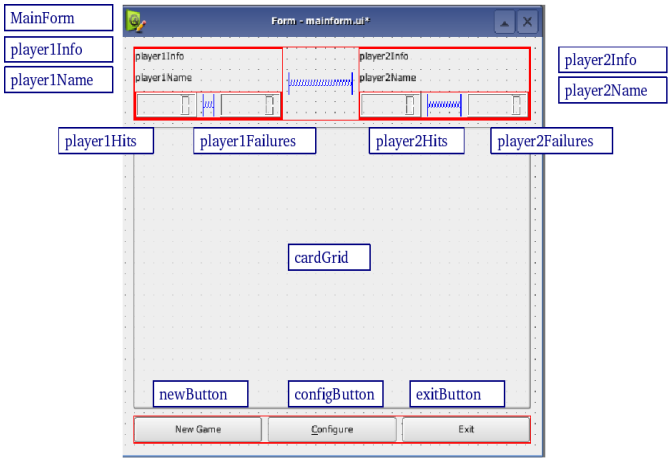
Miután minden szükséges osztályt elkészítettünk, elérkezett a pillanat, hogy elkészítsük alkalmazásunk főablakát.

Az alkalmazás főablak(MainForm osztály)

Először **Qt Designerrel** tervezzük meg a főablakot.



MainForm futás közben



MainForm tervezés közben

A felületen elhelyezett elemek

Típus	*Név(objectName)*	*Beállítások, megjegyzés*
QWidget	MainForm	windowTitle: Memory Game
QLabel	player1Info	
QLabel	player1Name	
QLabel	player2Info	
QLabel	player2Name	
QLCDNumber	player1Hits	
QLCDNumber	player1Failures	
QLCDNumber	player2Hits	
QLCDNumber	player2Failures	
CardButtonGroup	cardButtonGroup	QFrame->Jobb klikk->Promotion: CardButtonGroup
QPushButton	newButton	&New
QPushButton	configButton	&Configure
QPushButton	exitButton	&Exit

Először Qt Designerrel tervezze meg a felületet, majd mentse el **mainform.ui** néven. Ezután a felülettervet használva, származtatással, készítse el a **MainForm** osztályt.

mainform.h

```
#include <QWidget>

#include "player.h"
#include "ui_mainform.h"

class CardPackSet;
class GameManager;

class MainForm : public QWidget, public Ui_MainForm
{
    Q_OBJECT

public:
    MainForm(QWidget *parent = 0);
    ~MainForm();

public slots:
    void configure();
    void newGame();
    void updateCurrentPlayer(Player* player);
    void updatePlayerInfo(Player*player,QString info);
    void updateHits(Player* player);
    void updateFailures(Player* player);
    void handleGameOver(PlayerList players);

private:
    void initPlayers();
    MainForm(const MainForm&);
    MainForm& operator=(const MainForm&);

private:

    PlayerList players;
    CardPackSet* mCardPackSet;
    GameManager* mGameManager;
    int mPackIndex;
    int mNumRows;
    int mNumCols;
};
```

A **Memóriajáték** programnak vannak játékosai (**players***)*, kártyacsomagjai (**CardPackSet**), és a játékra felügyelő menedzsere (**GameManager**). A játékhoz tudnunk kell még, éppen melyik kártyacsomaggal játszunk (*mPackIndex*), és mekkora a játékmező (*mNumRows, mNumCols*).

A **MainForm** feladata a memóriajátékban résztvevő szereplők „létrehozása”, együttműködésük előkészítése, a felhasználói jelzések érzékelése és továbbítása a megfelelő komponesek felé, valamint a komponensek felől érkező, felhasználónak szóló információk megjelenítése.

mainform.cpp

```
#include <QMessageBox>

#include "cardpackset.h"
#include "gamemanager.h"
#include "configdialog.h"
#include "mainform.h"

MainForm::MainForm(QWidget *parent)
    : QWidget(parent), mPackIndex(0), mNumRows(2), mNumCols(2)
{
    setupUi(this); //Setup GUI elements created by designer

    initPlayers();

    mCardPackSet = new CardPackSet();
    mCardPackSet->loadFromAppDirectory();
    if(mCardPackSet->getPackList().empty()){
        QMessageBox::information(0,"pack name","
                                 No \"packs\" directory for this application \nLoading default images.");
        mCardPackSet->loadFromAppResource();
    }
```

Először betöltjük a kártyacsomagokat az alkalmazás **pack** alkönyvtárából (*loadFromAppDirectory()*). Ha a betöltés nem sikerült, vagy nem találtunk helyes kártyacsomagot (csomag neve: name.txt, hátlap: back.* (képfájl), előlapok: *.* (képfájlok)), akkor az alkalmazás erőforrásában elhelyezett kártyacsomagot olvassuk be. (*loadFromAppResource()*).

```
    mGameManager = new GameManager(mNumRows,mNumCols,players);
```

Létrehozunk egy GameManager példányt.

```
    connect(mGameManager, SIGNAL(playerInfoChanged(Player*,QString)),
            this,SLOT(updatePlayerInfo(Player*, QString)));
    connect(mGameManager, SIGNAL(gameOver(PlayerList)),
            this,SLOT(handleGameOver(PlayerList)));
    connect(mGameManager, SIGNAL(currentPlayerChanged(Player*)),
            this,SLOT(updateCurrentPlayer(Player*)));
    connect(mGameManager, SIGNAL(cardButtonPixmapChanged(int, QPixmap)),
            cardButtonGroup, SLOT(updateCardButtonPixmap(int, QPixmap)));
    connect(mGameManager, SIGNAL(cardButtonEnabledChanged(int, bool)),
            cardButtonGroup,SLOT(setCardButtonEnabled(int,bool)));
    connect(cardButtonGroup, SIGNAL(cardButtonClicked(int)),
            mGameManager, SLOT(handleCardButtonClick(int)));


    connect(exitButton, SIGNAL(clicked()), qApp, SLOT(quit()));
    connect(newButton, SIGNAL(clicked()), this, SLOT(newGame()));
    connect(configButton, SIGNAL(clicked()), this, SLOT(configure()));
```

Felsoroljuk, hogy az egyes objektumok különféle jelzéseire mely objektumok kapcsolódnak rá, és mely metódusokat kívánják végrehajtani az adott jelzés bekövetkezésekor.

```
    configure();
```

A játék konfigurálását akkor lehet csak elvégezni, amikor már „élnek a szereplők” (léteznek a példányok) és kiépítettük közöttük az együttműködésükhöz szükséges kapcsolatokat. Ezért a *configure()* metódust csak a *connect()*-ek után lehet meghívni.

```
}

MainForm::~MainForm()
{
    delete mCardPackSet;
    delete mGameManager;
}
```

A destruktor felszabadítja a **new** operátorral létrehozott objektumok számára lefoglalt tárhelyet.

```
void MainForm::newGame()
{
    cardButtonGroup->newCardButtonGroup(mNumRows,mNumCols);
    mGameManager->setCurrentPack(mCardPackSet->getByIndex(mPackIndex));
    mGameManager->newGame(mNumRows,mNumCols);
    newButton->setEnabled(false);
    configButton->setEnabled(false);
    exitButton->setEnabled(false);
}
```

A *newGame()* metódus akkor hajtódik végre, amikor a szereplők már léteznek, és egy új játékmenetet szeretnénk indítani, és a játékmezőt ennek megfelelően át kell alakítani. (Más méretű a játékmező, másik kártyacsomagot választottunk.)

```
void MainForm::configure()
{
    ConfigDialog dialog(mCardPackSet, mPackIndex, mNumRows, mNumCols, this);
    if(dialog.exec())
    {
        mNumRows = dialog.rowEdit->text().toInt();
        mNumCols = dialog.colEdit->text().toInt();
        mPackIndex = dialog.packComboBox->currentIndex();
    }
    newGame();
}
```

A *configure()* metódus megjeleníti a **ConfigDialog** dialógusablakot, és beállítja a játék paramétereit (játékmező mérete, kártyacsomag sorszáma)

```
void MainForm::initPlayers()
{
//      Get players' name via GUI

    players.clear();
    players.append(new Player ("Player1"));
    players.append(new Player ("Player2"));

    connect(players.at(0), SIGNAL(numHitsChanged(Player*)), this, SLOT(updateHits(Player*)));
    connect(players.at(0), SIGNAL(numFailuresChanged(Player*)), this, SLOT(updateFailures(Player*)));
    connect(players.at(1), SIGNAL(numHitsChanged(Player*)), this, SLOT(updateHits(Player*)));
    connect(players.at(1), SIGNAL(numFailuresChanged(Player*)), this, SLOT(updateFailures(Player*)));
```

```
        player1Name->setText(players.at(0)->getName());
        player2Name->setText(players.at(1)->getName());

        player1Info->setText("");
        player2Info->setText("");


    //      LoginDialog

    }
```

Terjedelmi okokból két, rögzített játékosra készítettük el a játék programot. Valójában az lenne a szép megoldás, ha a játék indításakor a játékosoknak be kellene jelentkezni. A szükséges kódot megtalálja a modul **login** alkönyvtárában.

```
void MainForm::handleGameOver(PlayerList players)
{
    player1Info->setText("");
    player2Info->setText("");
    player1Name->setText("<font color=black>" +  players.at(0)->getName() + "</font>");
    player2Name->setText("<font color=black>" +  players.at(1)->getName() + "</font>");
    QMessageBox::information(0, "Memory Game", "Game Over");

    newButton->setEnabled(true);
    configButton->setEnabled(true);
    exitButton->setEnabled(true);
}
```

Ez a metódus a játék végén kerül végrehajtásra. Ha el szeretnénk menteni a játékosok eredményeit, akkor azt ebben a metódusban kellene elvégezni.

```
void MainForm::updateCurrentPlayer(Player* currentPlayer)
{
    if(currentPlayer->getName() == players.at(0)->getName() ){
        player1Name->setText("<h2><font color=red>" + players.at(0)->getName() +
"</font></h2>");
        player2Name->setText("<font color=black>" +  players.at(1)->getName() + "</font>");
    }else{
        player2Name->setText("<h2><font color=blue>" + players.at(1)->getName() +
"</font></h2>");
        player1Name->setText("<font color=black>" + players.at(0)->getName() + "</font>");
    }
}

void MainForm::updatePlayerInfo(Player* currentPlayer,QString info)
{

    if(currentPlayer->getName() == players.at(0)->getName() ){
        player1Info->setText(info);
        player2Info->setText("");
    }else{
        player2Info->setText(info);
        player1Info->setText("");
    }
}
```

```
void MainForm::updateHits(Player* player)
{
    if(player->getName() == players.at(0)->getName())
        player1Hits->display(QString::number(player->getNumHits()));
    else
        player2Hits->display(QString::number(player->getNumHits()));

}

void MainForm::updateFailures(Player* player)
{
    if(player->getName() == players.at(0)->getName())
        player1Failures->display(QString::number(player->getNumFailures()));
    else
        player2Failures->display(QString::number(player->getNumFailures()));
}
```

Megjegyzés: A játékosokkal kapcsolatos tennivalókat esetleg célszerű lenne egy külön (**PlayerManager**) osztályban magvalósítani.

Az alkalmazás főprogramja

main.cpp

```
include <QApplication>
#include "mainform.h"

int main (int argc, char *argv[])
{
    QApplication app(argc,argv);

    MainForm *mainForm = new MainForm;
    mainForm->setWindowTitle("Memory Game");
    mainForm->show();
    return app.exec();
}
```

A játékprogram letölthető a people.inf.elte.hu/nacsa/qt4/eaf3/mod03/projects/game címről.

Ami kimaradt

- Játékosok bejelentkeztetése
- Kártyacsomagok frissítése: Ha futás közben másolunk képeket a **pack** alkönyvtárba, akkor arról már nem tud a program, csak ha újraindítjuk.
- Játék kettőnél több játékossal
- Játékállás archiválása
- A játékprogram a képeket „négyzetesíti”, így, ha a képfájl nem négyzetes, akkor „fura” kártyákat kapunk.
- Az időzítő sebességének beállítása futási időben
- Program állapot mentése (QSettings)
- „Üdvözlő oldal”, About űrlap
- A CardButtonGroup vezérlő „igazi” beillesztése a Qt Designer-be (nem promotion-nal)

- Kártyakép tárolás a binárisan, . . .

Tartalomjegyzék

A munkafüzet programjai letölthetők a people.inf.elte.hu/nacsa/qt4/eaf3/inv01/projects/ címről.

A munkafüzetben bemutatott programok készítésekor a Qt 4.2.2 verziót használtam.

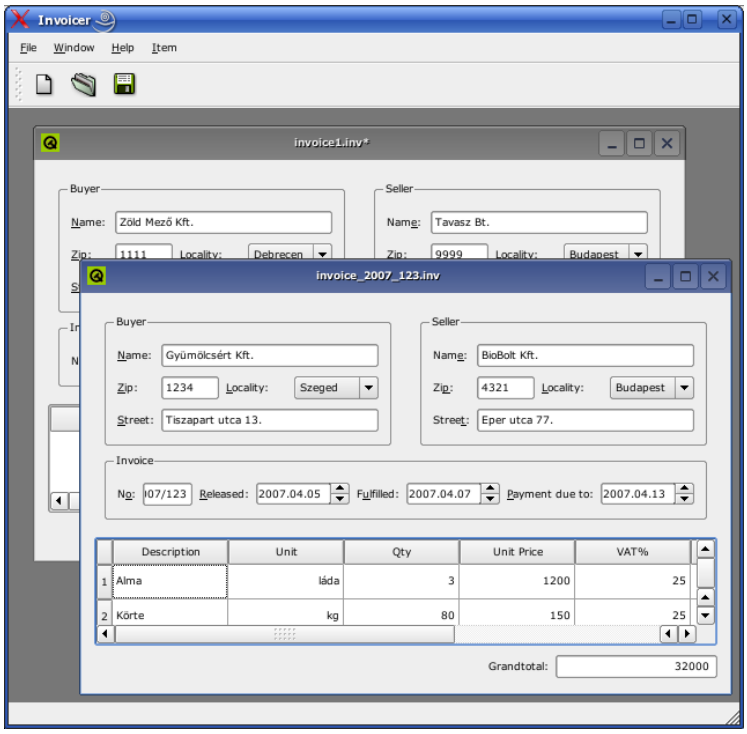
Készítette: Szabóné Nacsa Rozália

email: nacsa@inf.elte.hu

honlap: people.inf.elte.hu/nacsa

Feladat

Készítsünk programot számlák előállítására, karbantartására. A számlákat tároljuk külön-külön fájlokban. A program tegye lehetővé új számla létrehozását, meglévő számlák módosítását, a számlák (dokumentumok) mentését, betöltését.



MDI Számlakezelő program futás közben

A programot modularizáltan, **osztályok** segítségével készítjük el. A dokumentum adatkezelését (számla betöltése, mentése, számított adatok előállítása) önálló osztályra bízzuk. Adatkarbantartásnál (eladó, vevő adatainak megjelenítése, módosítása, számlatételek felvitele, módosítása, törlése) az *adatszolgáltató* és *adatmegjelenítő* funkciókat szétválasztjuk (*modell-nézet, model/view* tervezési minta). Projektünket MDI (*Multiple Document Interface*) interfésszel valósítjuk meg, így egy időben több dokumentumot (számlát) is kezelhetünk.

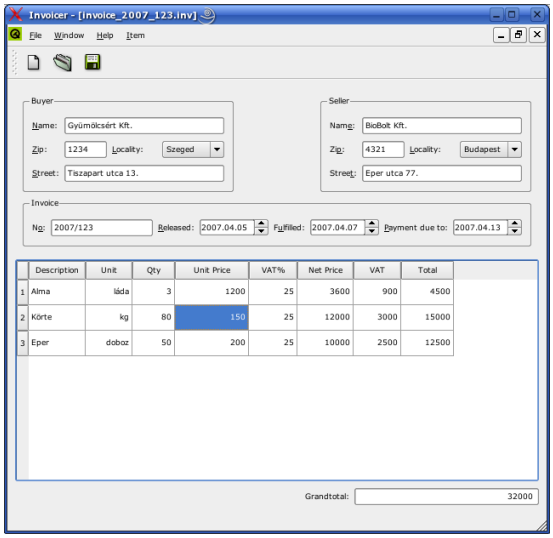
A számla elemei

fejléc (head): vevő adatai, eladó adatai, számla kelte, esedékesség, fizetési határidő

számla tétel (invoice item): a számlán szereplő egy-egy áru adatai (megnevezés, mennyiség, stb.)

számla tételek (invoice items): számla tételek listája

A dokumentumot tároló fájl felépítése



A fájl szerkezete

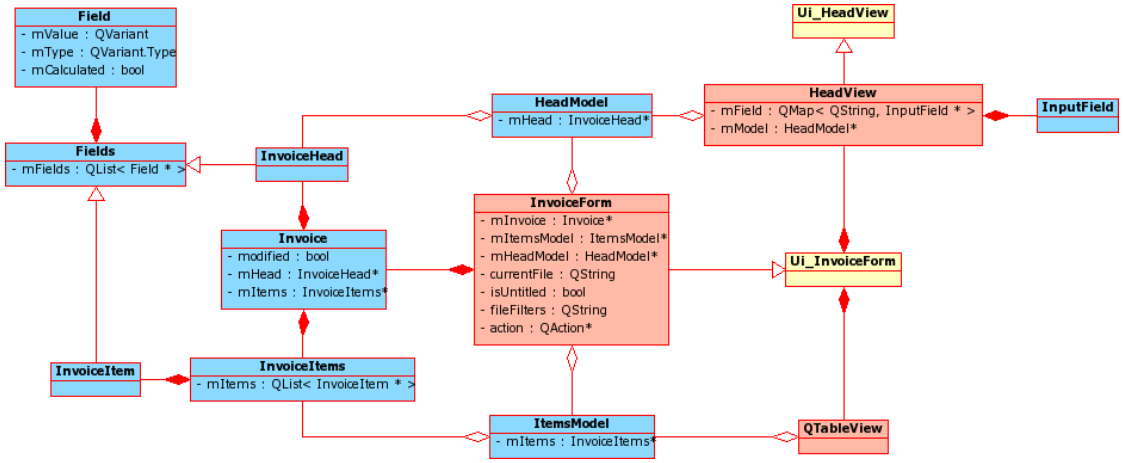
<fejlécben szereplő elemek száma (n)>
<fejléc adat 1>
. . .
<fejléc adat n>
<számlatételek száma (k)>
<1. áru megnevezése (description)>
<1. áru mértékegysége (unit)>
<1. áru mennyisége (quantity)>
<1. áru egységára (unitPrice)>
<1. áru ÁFA kulcsa százalékban (vatPercent)>
. . .
<k-dik áru megnevezése (description)>
<k-dik mértékegysége (unit)>
<k-dik áru mennyisége (quantity)>
<k-dik áru egységára (unitPrice)>
<k-dik áru ÁFA kulcsa százalékban (vatPercent)>

A példa számla fájlja

12
Gyümölcsért Kft.
1234
Szeged
Tiszapart utca 13.
BioBolt Kft.
4321
Budapest
Eper utca 77.
2007/123
2007-04-05
2007-04-07
2007-04-13
3
Alma
láda
3
1200
25
Körte
kg
80
150
25
Eper
doboz
50
200
25

Megjegyzés: A számla adatait szöveges fájlban tároljuk, mert így egyszerűbb a tesztelés. Természetesen „igazi” programnál a dokumentumok tárolására más fájlformátumot (pl. bináris fájlt) „illik” használni.

A számlakészítő program osztálydiagramja



Field: a számla egy tetszőleges adatát tároló és kezelő adat osztály

Fields: Field típusú mezők listáját tároló adat osztály

Invoice Head: a számla fejléc adatait kezelő adat osztály

InvoiceItem: a számlatétel adatait tároló adat osztály

InvoiceItems: a számlatételeket tároló adat osztály

Invoice: a számlát (dokumentumot) tároló adat osztály (InvoiceHead + InvoiceItems)

HeadModel/HeadView: a fejléc karbantartását végző *adatszolgáltató* és *adatmegjelenítő* osztályok

ItemsModel/QTableView: a számlatételeket karbantartó tábla *adatszolgáltató* és *adatmegjelenítő* osztályai

Input Field: absztrakt osztály a fejlécen használt vezérlők egységes kezelésére

Field osztály

Qt-ben bármely érték tárolható **QVariant** típusként. A **QVariant** típusú változóban tárolt értéket bármikor átalakíthatjuk tetszőleges más típusra. A **Field** osztály **QVariant** típusú adatok tárolására alkalmas *általános* osztály.

Az osztály attribútumai: a tárolt adat *típusa*; a tárolt adat *értéke*; egy *logikai változó*, amely jelzi, hogy „valódi” adatot, vagy számított adatot tárol.

field.h

```
#include <QObject>
#include <QVariant>

class Field : public QObject
{
    Q_OBJECT
public:

    Field(const QString& name, QVariant value, QVariant::Type type = QVariant::String,
          bool calculated = false);
    virtual ~Field() {}
```

```
    virtual QString name() {return objectName();}
    virtual QVariant value() {return mValue;}
    virtual QVariant::Type type() {return mType;}
    bool isCalculated() {return mCalculated;}

    QString toString() const;

public slots:
    virtual bool setValue(QVariant newValue);

signals:
    void valueChanged();

protected:
    void setType(QVariant::Type type) {mType = type;}
    void setCalculated(bool calculated = true) {mCalculated = calculated;}

private:
    QVariant mValue;
    QVariant::Type mType;
    bool mCalculated;
};
```

A **Field** osztályt a **QObject** osztályból származtatjuk. Az osztály rendelkezik saját jellel (*valueChanged()*) és jelkezelővel (*setValue()*), ezért megadjuk az **Q_OBJECT** makrót. Az osztály attribútumai a tárolt adat értéke *(mValue),* a tárolt adat típusa *(mType),* valamint egy logikai változó *(mCalculated),* amely jelzi, hogy „valódi" adatot, vagy számított adatot tárol az objektum.

field.cpp

```
#include "field.h"

Field::Field(const QString& name, QVariant value, QVariant::Type type, bool calculated)
        : mValue(value), mType(type), mCalculated(calculated)
{
    setObjectName(name);
}
```

Az objektum létrehozásakor megadjuk az objektum nevét, típusát és azt, hogy ezt az adatot számítással állítjuk-e elő. A paraméterként átadott értékekkel inicializáljuk az adattagokat. Az objektum nevét az ősosztály név attribútumaként tároljuk.

```
bool Field::setValue(QVariant newValue) {      //Mezőérték beállítása üzenetküldéssel
  if(mValue==newValue) return true;
    mValue = newValue;
  emit valueChanged();
  return true;
}
```

Ha az objektum érték adata megváltozik, akkor erről jelzést küldünk. *(valueChanged()).*

```
QString Field::toString() const {    //Objektum „szövegesítése"
  return QString("%1(%2) = %3").
    arg(objectName()).arg(mType).arg(mValue.toString());
}
```

Az argumentumokat a **QString** konstruktorában megadott, *%n* mintákra illesztjük.

Fields osztály

A **Fields** (többesszám!) osztály **Field** (egyesszám!) típusú elemek listáját kezelő osztály.

fields.h

```
#include <QVariant>
#include <QList>
#include <QObject>

class Field;

class Fields : public QObject
{
    Q_OBJECT

public:
    Fields(QObject* parent = 0);
    ~Fields();

    void addField( Field* value);
    Field* field(const QString& name);
    Field* field(int i);

    int count() {return mFields.count();}
    int columnCount(){return count();}

    QStringList names();
    QStringList fieldsToSave();

    void clear();
    void init();

    QString toString();

signals:
    void valueChanged();

private:
    QList <Field*> mFields;
};
```

fields.cpp

```
#include <QString>
#include <QStringList>
#include <QObject>
#include "fields.h"
#include "field.h"

Fields::Fields(QObject* parent)
{
    setParent(parent);
}

Fields::~Fields()
{
    clear();
}
```

```
void Fields::clear()  //Lista kiürítése
{
	while (!mFields.isEmpty())
		delete mFields.takeFirst();
}

void Fields::init()		//Listaelemek feltöltése kezdő értékkel
{
	for (int i = 0; i < count(); ++i)
		field(i)->setValue(QVariant());
}

Field* Fields::field(const QString& name)  //Visszaadja az adott nevű mezöre mutató pointert
{
	return mFields.at(names().indexOf(name));
}
```

A *field(const QString& name)* metódussal megszerezhetjük az *adott nevű mezőre* mutató pointert.

```
Field* Fields::field(int i)		//Visszaadja az adott indexű mezőre mutató poinert
{
	return mFields.at(i);
}
```

A *field(int i)* metódussal megszerezhetjük a lista *adott pozíciójában található* mezőre mutató pointert.

```
void Fields::addField(Field* field)  //Hozzáveszi a listához a field mezőt
{
	mFields.append(field);
}
```

A *addField()* metódussal egy új mezőt illeszthetünk a lista végére.

```
QStringList Fields::names()  //Visszaadja a mező neveket
{
	QStringList ret;
	for (int i = 0; i < count(); ++i) {
		ret << field(i)->name();
	}
	return ret;
}
```

A *names()* függvénnyel megkaphatjuk az adott számlatételben szereplő mező neveket.

```
QStringList Fields::fieldsToSave()
{
	QStringList ret;
	for (int i = 0; i < count(); ++i) {
		if (!field(i)->isCalculated())
			ret << field(i)->value().toString();
	}
	return ret;
}
```

A *fieldsToSave()* metódus szolgáltatja a számlatétel azon elemeit, melyeket a számla mentésekor ki kell írni a dokumentum fájlba.

```
QString Fields::toString()
{
	QString ret = "\n";
```

```
        for (int i = 0; i < count(); ++i) {
            ret += "\t" + field(i)->name() + "=" + field(i)->value().toString();
        }
        return ret;
}
```

InvoiceItem osztály

A számla tétel osztály egy olyan **Fields** osztály, melyet kiegészítettünk néhány, a számlára jellemző metódussal.

invoiceitem.h

```
#include <QVariant>
#include <QList>
#include <QObject>
#include "fields.h"

class InvoiceItem : public Fields
{
    Q_OBJECT
public:
    InvoiceItem(QString description=QString::null, QString unit=QString::null,
        int quantity=0, int unitPrice=0, int vatPercent=0, QObject* parent = 0);
    virtual ~InvoiceItem();

    static QStringList titles();

    QVariant totalValue();
    QVariant netValue();
    QVariant vatValue();

    void init();

public slots:
    void updateCalculateFields();
 };
```

invoiceitem.cpp

```
#include <QString>
#include <QStringList>
#include <QObject>
#include "invoiceitem.h"
#include "field.h"

InvoiceItem::InvoiceItem(QString description, QString unit,
        int quantity,int unitPrice, int vatPercent,QObject* parent) : Fields(parent)
{
    addField(new Field("description",description,QVariant::String));
    addField(new Field("unit",unit,QVariant::String));
    addField(new Field("quantity",quantity,QVariant::Int));
    addField(new Field("unitprice",unitPrice,QVariant::Int));
    addField(new Field("vatpercent",vatPercent,QVariant::Int));

    //Calculated fields
    addField(new Field("netvalue",netValue(),QVariant::Int,true));
    addField(new Field("vatvalue",vatValue(),QVariant::Int,true));
    addField(new Field("totalvalue",totalValue(),QVariant::Int,true));
```

```
        connect(field("quantity"),SIGNAL(valueChanged()),this,SLOT(updateCalculateFields()));
        connect(field("unitprice"),SIGNAL(valueChanged()),this,SLOT(updateCalculateFields()));
        connect(field("vatpercent"),SIGNAL(valueChanged()),this,SLOT(updateCalculateFields()));
}
```

A konstruktorban felépítjük a számlatételt reprezentáló listát. A számlatételt reprezentáló listába betesszük a számított adatokat is. Természetesen a számított adatokat nem mentjük el a számla fájljába. A *signal-slot* kapcsolatok miatt a számított adatok értéke bármely „számszerű” adat megváltozásakor újraszámolódik.

```
InvoiceItem::~InvoiceItem() {}

void InvoiceItem::init()
{
    field("description")->setValue("");
    field("unit")->setValue("db");
    field("quantity")->setValue(1);
    field("unitprice")->setValue(0);
    field("vatpercent")->setValue(20);
}

QStringList InvoiceItem::titles()
{
    QStringList titles;
    titles  << QObject::trUtf8("Description") << QObject::trUtf8("Unit")
            << QObject::trUtf8("Qty") << QObject::trUtf8("Unit Price")
            << QObject::trUtf8("VAT%") << QObject::trUtf8("Net Price")
            << QObject::trUtf8("VAT") << QObject::trUtf8("Total");
    return titles;
}
```

A *titles()* statikus (osztályszintű) metódus szolgáltatja a számlatételek oszlop feliratait.

```
void InvoiceItem::updateCalculateFields()
{
    field("netvalue")->setValue(netValue());
    field("vatvalue")->setValue(vatValue());
    field("totalvalue")->setValue(totalValue());
}

QVariant InvoiceItem::totalValue()  //A számlatétel bruttó értéke
{
    return field("quantity")->value().toInt() * field("unitprice")->value().toInt() *
                (100 +  field("vatpercent")->value().toInt())/100;
}

QVariant InvoiceItem::netValue()  //A számlatétel nettó értéke
{
    return field("quantity")->value().toInt() * field("unitprice")->value().toInt() ;
}

QVariant InvoiceItem::vatValue()  //A számlatétel ÁFA értéke
{
    return field("quantity")->value().toInt() * field("unitprice")->value().toInt() *
                field("vatpercent")->value().toInt()/100 ;
}
```

A fenti függvények megadják a számlatétel számított adatait.

InvoiceItems osztály

Az **InvoiceItems** osztály egyetlen adattagja a számlatételeket tartalmazó *lista*. Az osztályt a **QObject** osztályból származtatjuk, így a **Qt** „szülő-gyerek mechanizmusa” miatt kényelmes lesz az objektum felszámolása.

invoiceitems.h

```
#include <QObject>
#include <QList>

class InvoiceItem;

class InvoiceItems : public QObject
{

public:
    InvoiceItems(QObject* parent = 0);
    ~InvoiceItems();

    void init();

    InvoiceItem* item(int i) {return mItems.at(i);}

    void addItem(InvoiceItem *i);
    void insertItem(int pos, InvoiceItem *item);
    void deleteItem(InvoiceItem *i);
    void deleteItemAt(int pos);

    int columnCount();
    int count() {return mItems.count();}
    QString grandTotal();

    QString toString();

private:
    QList<InvoiceItem*> mItems;
};
```

invoiceitems.cpp

```
#include <QStringList>
#include "invoiceitems.h"
#include "invoiceitem.h"

InvoiceItems::InvoiceItems(QObject* parent)
{
    setParent(parent);
    setObjectName(QString::fromUtf8("InvoiceItems"));
}

InvoiceItems::~InvoiceItems()
{
    init();
}
```

A destruktor végzi az objektum által lefoglalt dinamikus tárhely felszabadítását. A lista „lebontására” máshol is szükség lehet, ezért ezt a feladatot egy önálló függvényben valósítjuk meg (*init()*).

```
void InvoiceItems::init()       // Lista inicializálása (kiürítése)
{
    while (!mItems.isEmpty())
        delete mItems.takeFirst();
}
```

Az *init()* metódussal inicializáljuk a számlatételeket. Az inicializálás valójában nem más, mint egy üres számlatétel lista, ezért szükség esetén az objektum által reprezentált listát lebontjuk.

```
void InvoiceItems::addItem(InvoiceItem *item)  //Lista kiegészítése egy új elemmel
{
    mItems.append(item);
}

void InvoiceItems::insertItem(int pos, InvoiceItem *item)  //Elem beszúrása egy adott pozícióra
{
    mItems.insert(pos,item);
}

void InvoiceItems::deleteItem(InvoiceItem *i)  //Egy adott elem törlése
{
    deleteItemAt(mItems.indexOf(i));
}

void InvoiceItems::deleteItemAt(int pos)  //Egy adott pozíción lévő elem törlése
{
    if (0 <= pos < mItems.size())
        delete mItems.takeAt(pos);
}
```

Az *addItem(), insertItem(), deleteItem(), deleteItemAt()* metódusokkal hozzávehetünk, és kitörölhetünk a listából egy-egy számlatételt. Vegyük észre, hogy a **QList** *takeAt()* függvénye „csak” kiveszi a listából az elemet. A függvény visszaadja az elemre mutató pointert, melyre mi adjuk ki a *delete* parancsot.

```
QString InvoiceItems::toString() //Az objektumot „szövegesíti
{
    QString ret = "\nInvoice: Item List";
    for (int i=0; i< count(); i++)
            ret += mItems.at(i)->toString();
    return ret;
}

int InvoiceItems::columnCount()  //oszlopok száma
{
    return mItems.count() > 0 ? mItems.at(0)->count(): 0;
}

QString InvoiceItems::grandTotal() //A számla végösszege
{
    int total=0;
    for(int i = 0; i < count(); ++i)
        total += mItems.at(i)->totalValue().toInt();
    return QString::number(total);
}
```

InvoiceHead osztály

invoicehead.h

```
#include <QObject>
#include <QVariant>
#include <QList>

#include "field.h"
#include "fields.h"

class InvoiceHead : public Fields
{
    Q_OBJECT

public:
    InvoiceHead(QObject* parent = 0);
    ~InvoiceHead();
    void init();
};
```

Az **InvoiceHead** osztályt a **Fields** osztályból állítjuk elő származtatással. Az osztály feladata a számla fejléc elkészítése (*konstruktor*) és a mezők inicializálása (*init()*).

invoicehead.cpp

```
#include <QString>
#include <QStringList>
#include <QDate>
#include <QVariant>

#include "field.h"
#include "invoicehead.h"

InvoiceHead::InvoiceHead(QObject* parent) : Fields(parent)
{
    addField(new Field("buyer_name",QVariant(QString()),QVariant::String));
    addField(new Field("buyer_zip",QVariant(QString()),QVariant::String));
    addField(new Field("buyer_city",QVariant(QString()),QVariant::String));
    addField(new Field("buyer_street",QVariant(QString()),QVariant::String));

    addField(new Field("seller_name",QVariant(QString()),QVariant::String));
    addField(new Field("seller_zip",QVariant(QString()),QVariant::String));
    addField(new Field("seller_city",QVariant(QString()),QVariant::String));
    addField(new Field("seller_street",QVariant(QString()),QVariant::String));

    addField(new Field("invNo",QVariant(QString()),QVariant::String));
    addField(new Field("released",QVariant(QDate()),QVariant::Date));
    addField(new Field("fulfilled",QVariant(QDate()),QVariant::Date));
    addField(new Field("dueTo",QVariant(QDate()),QVariant::Date));
}
```

A konstruktorban létrehozzuk a számla fejlécét alkotó adatmezőket, és elhelyezzük azokat a listában.

```
InvoiceHead::~InvoiceHead() {}
```

```
void InvoiceHead::init()
{
    field("buyer_name")->setValue("");
    field("buyer_zip")->setValue("");
    field("buyer_city")->setValue("Szeged");
    field("buyer_street")->setValue("");

    field("seller_name")->setValue("");
    field("seller_zip")->setValue("");
    field("seller_city")->setValue("Budapest");
    field("seller_street")->setValue("");

    field("invNo")->setValue("");
    field("released")->setValue(QDate::currentDate());
    field("fulfilled")->setValue(QDate::currentDate().addDays(2));
    field("dueTo")->setValue(QDate::currentDate().addDays(8));

}
```

Az *init()* metódussal beállítjuk a számla fejléc elemeinek kezdőértékeit. Alapértelmezésben a kibocsájtás dátumát az aktuális dátumra állítjuk, a teljesítés dátumát két nappal, az esedékesség dátumat nyolc nappal későbbre állítjuk. A vevő alapértelmezett székhelye Szeged, az eladóé Budapest.

Invoice osztály

A **Invoice** osztály az alkalmazás „dokumentuma”, amely a dokumentum osztályokra jellemző műveletekkel (*open, save, load, stb.*) rendelkezik.

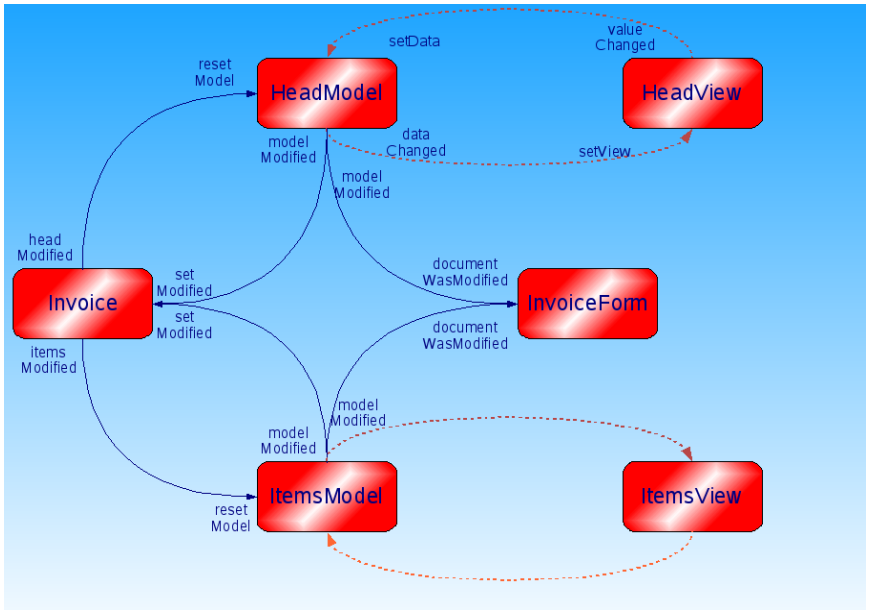
invoice.h

```
#include <QObject>
#include "invoiceitem.h"
#include "invoiceitems.h"

class InvoiceHead;

class Invoice : public QObject
{
    Q_OBJECT

public:
  Invoice(QObject*parent = 0);
  ~Invoice() {};

  bool saveAs(const QString &fileName);
  bool load(const QString &fileName);
  void newInvoice();
  bool isModified() const;

  InvoiceItems* items() const { return mItems; }
  InvoiceHead* head() const  {return mHead;}

  void addItem(InvoiceItem *i);
  void deleteItem(InvoiceItem *i);
```

```
    QString toString();

public slots:
    void setModified(bool val = true) { modified = val; }
    void loadHeadField(int i, QVariant value);

    void setHeadField(const QString &name,QVariant value);
    void setHeadField(int i, QVariant value);

signals:
    void itemsModified();
    void headModified();
    void headFieldChanged(const QString &, QVariant);

private:
    bool modified;
    InvoiceHead* mHead;
    InvoiceItems* mItems;
};
```

A számla két fő részből áll, a fejlécből (*mHead)* és a számla tételekből (*mItems*). A *modified* adattagban jelezzük módosult-e a számla, így kilépéskor figyelmeztetni tudjuk a felhasználót az adatok megváltozására, és eldöntheti, kívánja-e menteni a módosításokat.

A projektben létrehozott objektumok közötti együttműködést a **Qt** „*signál-slot*” mechanizmusával kívánjuk megoldani, ezért az **Invoice** osztályban definiáljuk az együttműködéshez szükséges jeleket (*signal*) és jelfeldolgozó függvényeket (*slot*). Az együttműködés megértéséhez - már előzetesen is - érdemes rápillantani az alkalmazás objektumai közötti kommunikációt leíró ábrára.



Az áttekinthetőség kedvéért ezen az ábrán csak az adatok tárolásával, megjelenítésével kapcsolatos jeladó és jelvevő (signal-slot) függvények szerepelnek.

invoice.cpp

```
#include <QDate>
#include <QFile>
#include <QTextStream>
#include <QString>
#include <QStringList>
#include <QObject>
#include <QMessageBox>
#include <QVariant>

#include "invoiceitems.h"
#include "invoicehead.h"
#include "invoiceitem.h"
#include "invoice.h"

Invoice::Invoice(QObject *parent)
{
    setParent(parent);
    setObjectName("Invoice");
    modified = false;
    mHead = new InvoiceHead(this);
    mItems = new InvoiceItems(this);
}
```

Az objektum ősosztályának *parent* attribútumát beállítjuk a paraméterként kapott értékre. A *modified* (módosítva) jelzőt hamisra állítjuk (új számlánk van, ami még nem változott meg), majd létrehozzuk a számla két összetevőjét, a *fejlécet* és a *számlatételek listáját*. A fejléc és a számlatétel szülő objektuma maga a számla, ezért ha számla objektum megszűnik, akkor ők is automatikusan törlődnek.

```
void Invoice::newInvoice()
{
    mHead->init();
    mItems->init();
    emit headModified();
    emit itemsModified();
    setModified(false);
}
```

A *newInvoive()* metódus alapértékre állítja a számla két összetevőjét. Az *emit* hatására a jeladó függvények (*signals*) az elemek megváltozásáról jelzést küldenek (*emit*), így a jelzésekre rákapcsolódó objektumok frissíthetik adataikat.

```
bool Invoice::saveAs(const QString & fileName)
{
    QFile invoiceFile(fileName);
    if(!invoiceFile.open(QIODevice::WriteOnly))
    {
        qWarning((QString("Cannot open file for writing:"
            + invoiceFile.errorString() + "\n")).toAscii());
        return false;
    } else {
        QTextStream out(&invoiceFile);
        out << QString::number(mHead->fieldsToSave().count()) << endl;
        for (int i=0; i < mHead->fieldsToSave().count(); ++i)
            out << mHead->fieldsToSave().at(i) << endl;
        out << QString::number(mItems->count()) << endl;
```

```
            for (int i=0; i< mItems->count(); ++i)
                for (int j=0; j < mItems->item(i)->fieldsToSave().count(); ++j)
                    out << mItems->item(i)->fieldsToSave().at(j) << endl;

            return true;
        }
    }
```

A *saveAs()* metódussal mentjük el a dokumentumot (számlát) a paraméterként megadott fájlba. A fájl kezelést a **QFile** osztállyal valósítjuk meg. A fájl nevével létrehozott **QFile** osztály *open()* metódusával írás céljából megnyitjuk a fájlt. Sikeres fájlnyitás után létrehozunk egy adatfolyam (**QTextStream**) típusú változót (*out*). Ezen a típuson léteznek a << beolvasó és >> kiíró műveletek, így a beolvasáshoz és a kiíráshoz használhatjuk őket.

```
bool Invoice::load(const QString & fileName)
{
    QFile invoiceFile(fileName);
    if(!invoiceFile.open(QIODevice::ReadOnly | QIODevice::Text))
    {
        qWarning(QString("Warning: ** " + fileName
                + " file is missing or not a text file. ** ").toAscii());
        return false;
    }
    QTextStream in(&invoiceFile);
    mHead->init();
    int headCount = in.readLine().toInt();
    for (int i=0; i< headCount; i++)
        mHead->field(i)->setValue(in.readLine());
    emit headModified();

    int itemsCount = in.readLine().toInt();
    for (int i=0; i< itemsCount; i++)
    {
        QString name = in.readLine();
        QString unit = in.readLine();
        int quantity = in.readLine().toInt();
        int unitPrice = in.readLine().toInt();
        int vatPercent = in.readLine().toInt();
        mItems->addItem(new InvoiceItem(name,unit,quantity,unitPrice,vatPercent));
    }
    emit itemsModified();
    setModified(false);
    return true;
}
```

A *load()* metódussal beolvassuk a paraméterként kapott fájlt (*dokumentumot, számlát*), továbbá jelezzük mind a fejléc, mind pedig a számlatételek megváltozását.

```
bool Invoice::isModified() const
{
    return modified;
}

void Invoice::addItem(InvoiceItem *item)
{
    mItems->addItem(item);
    emit itemsModified();
```

```
        setModified(true);
}
```

Az *addItem()* metódussal új számlatételt fűzhetünk a számlatétel-lista végére. A változásról jelzést küldünk.

```
void Invoice::deleteItem(InvoiceItem *i)
{
        mItems->deleteItem(i);
        emit itemsModified();
        setModified(true);
}
```

A *deleteItem()* metódussal kitöröljük a listából a paraméterben megadott elemet. A változásról jelzést küldünk.

```
void Invoice::loadHeadField(int i,QVariant value)
{
        mHead->field(i)->setValue(value);
        setModified(true);
}
```

A *loadHeadField()* metódus a számla fejléc i-dik elemének értékét beállítja a paraméterül kapott adatra és jelzi, hogy a számla tartalma megváltozott.

```
void Invoice::setHeadField(const QString &name, QVariant value)
{
        if (mHead->field(name)->value()==value) return;
        mHead->field(name)->setValue(value);
        setModified(true);
        emit headFieldChanged(name,value);
}

void Invoice::setHeadField(int i,QVariant value)
{
        if (mHead->field(i)->value()==value) return;
        mHead->field(i)->setValue(value);
        setModified(true);
        emit headFieldChanged(mHead->field(i)->name(),value);
}
```

A *setHeadField()* metódusokkal *név* vagy *pozíció* szerint beállíthatjuk a mező értékét. Az objektum jelzi adata megváltozását.

```
QString Invoice::toString()
{

        QString ret = "Invoice:";
        ret += mHead->toString();
        ret += "\nInvoice Item List";
        for (int i=0; i< mItems->count(); i++)
    {
                ret += mItems->item(i)->toString();
        }
        return ret;
}
```

Tesztelés

Ez a munkafüzet a számlakészítő program első modulja. A modul programjai letölthetők a people.inf.elte.hu/nacsa/qt4/eaf3/inv01/projects/ címről.

A modulban szereplő osztályok működését, viselkedését tesztelheti a fenti címen található **invoice** projektben.

main.cpp

```
#include <QApplication>
#include "invoice.h"

int main (int argc, char *argv[])
{
    QApplication app(argc,argv);

    Invoice *invoice = new Invoice();
    invoice->load("./invoices/2007/invoice_2007_123.inv");
    qDebug(QString(invoice->toString()).toAscii());
    invoice->addItem(new InvoiceItem("Cseresznye","kg",10,150,20));
    invoice->saveAs("./invoices/2007/invoice_2007_456.inv");
    invoice->newInvoice();
    invoice->saveAs("./invoices/2007/invoice_new.inv");
    return true;
}
```

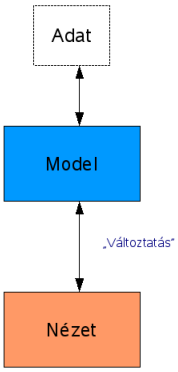
Az **invoice** projekt főprogramjában betöltjük az *invoice_2007_123.inv* fájlban tárolt számlát, majd annak tartalmát megjelenítjük a terminál ablakban (*qDebug()*). Ezután hozzáadunk a számlához egy új számlatételt („cseresznye”), végül a módosított számlát elmentjük az *invoice_2007_234.inv* fájlba. Ezután létrehozunk egy kezdeti értékekre beállított számlát, és elmentjük azt az *invoice_new.inv* fájlba. A program futtatása után nézze meg a fájlok tartalmát.

Modell-nézet (model-view) tervezési minta (bevezetés)

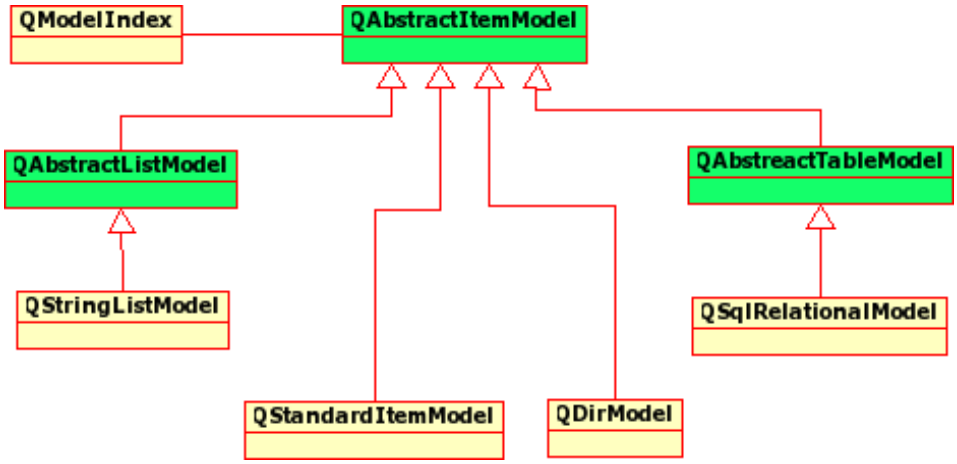
A **Qt 4** egyik újdonsága az **Interview** modul, amely a *modell-nézet* tervezési mintán alapuló programok készítésére alkalmas osztályokat tartalmazza.

A grafikus alkalmazásoknál használt vezérlők egy része az adatokat magában a vezérlőben tárolja. Ilyenkor a program a vezérlőben tárolt adatokon végzi el a szerkesztési, keresési műveleteket. Ez a megoldás nem megfelelő, amikor sok adattal dolgozunk, *web*-es alkalmazást készítünk, vagy ugyanazt az adatot többféleképpen is meg kell jeleníteni. Ilyen esetekben célszerű az *adat (modell)* és a *nézet* szétválasztása. Az *adat* és a *nézet* szétválasztása lehetővé teszi, hogy ugyanazt az adatot egyszerre több nézetben is megjelenítsük, továbbá a *modell-nézet* minta alkalmazásakor az adatstruktúra megváltoztatása nélkül is bevezethetünk új nézeteket.

A **modell-nézet (model-view)** minta a a *SmallTalk*-ból jól ismert **modell-nézet-vezérlő (model-view-controler)** mintán alapul. Az eredeti mintában az adattároló *modell* (*model*) és az adattartalmat megjelenítő *nézetek* (*view*) között a *vezérlő* (*controller*) biztosítja a kapcsolatot oly módon, hogy a *modell* minden változásáról értesítést küld a *nézeteknek*, így azok frissíteni tudják magukat. A *modell-nézet* mintában a *nézetet* és a *vezérlőt* összevonták. Az adatok tárolása és az adattartalom megjelenítése ebben a mintában is szétválik, de egyszerűbb, mint az eredeti *modell-nézet-vezérlő* minta.



A **Qt Interview** modul *modell* osztályai (részlet)



Projektünkben a *fejléc* és a *számlatétel* karbantartó programrészeket a **Qt 4 Interview** osztályait használva, a *modell-nézet* programtervezési minta szerint készítjük el.

A munkafüzet programjai letölthetők a people.inf.elte.hu/nacsa/qt4/eaf3/inv01/projects/ címről.

A számlakészítő program elkészítéséhez van még két munkafüzet!

Tartalomjegyzék

A munkafüzet programjai letölthetők a people.inf.elte.hu/nacsa/qt4/eaf3/inv03/projects/ címről.

A munkafüzetben bemutatott programok készítésekor a Qt 4.2.2 verziót használtam.

Készítette: Szabóné Nacsa Rozália

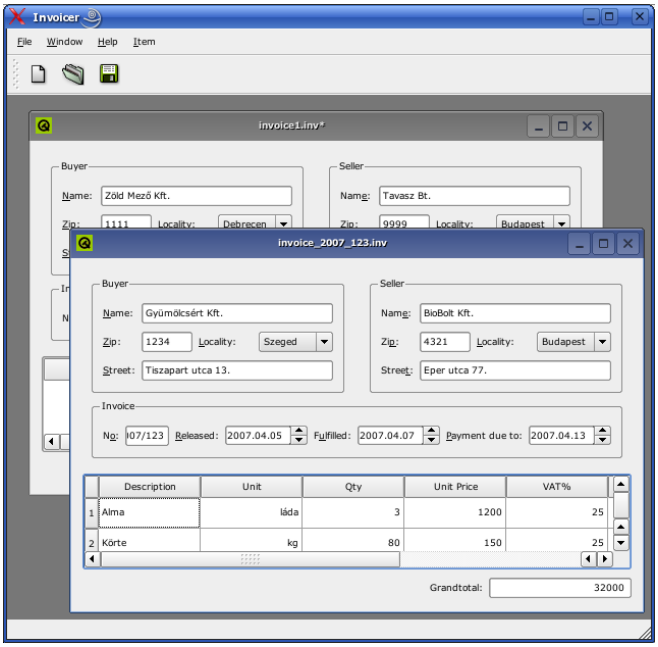
email: nacsa@inf.elte.hu

honlap: people.inf.elte.hu/nacsa

Feladat

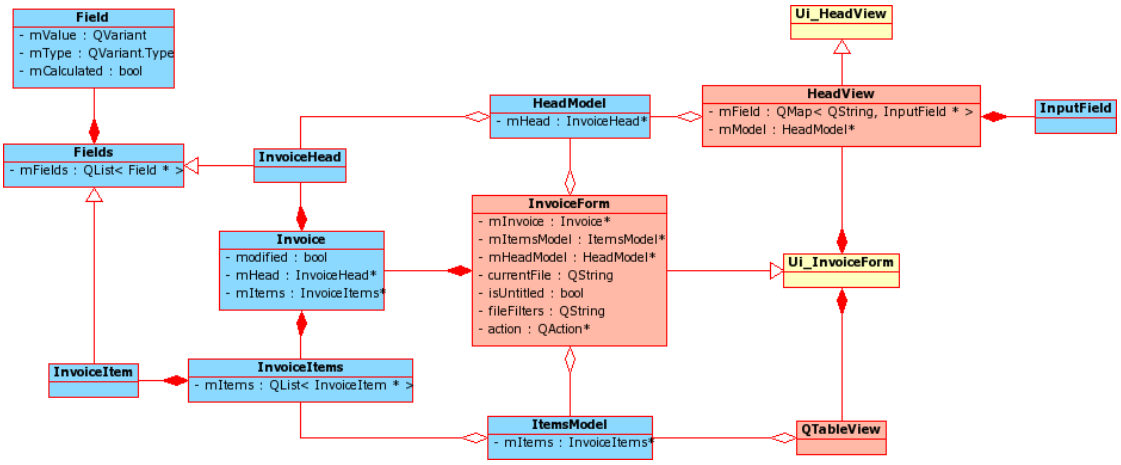
Készítsünk programot számlák előállítására, karbantartására. A számlákat tároljuk fájlokban. A program tegye lehetővé új számla létrehozását, meglévő számlák módosítását, a számlák (dokumentumok) mentését, betöltését.

Megjegyzés: Ez a munkafüzet a számlakészítő alkalmazás harmadik munkafüzete. Ebben a munkafüzetben feltételezzük, hogy Ön már feldolgozta a „Qt 4 /C++ alapú MDI alkalmazás: Számlakészítő program 1 és 2” munkafüzeteket.



MDI Számlakezelő program futás közben

Az alkalmazás osztálydiagramja

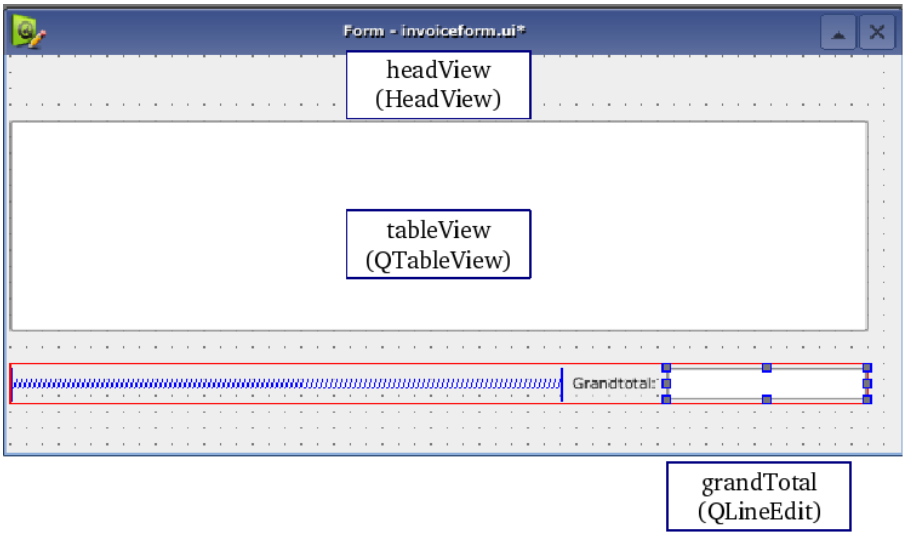


Ebben a modulban a korábban elkészített osztályok felhasználásával elkészítünk egy **több ablakos** számlakezelő programot. Több ablakon azt értjük, hogy az alkalmazás egy időben több számlát is tud kezelni. A számlák a karbantartó program főablakában helyezkednek el.

Űrlap elkészítése

Grafikus felület kialakítása

A főablak elkészítése előtt állítsuk össze a számlakarbantartó űrlapot (Form). Ez az űrlap jelenít meg egy-egy számlát. A több ablakos alkalmazásunkban ilyen űrlapból helyezhetünk el többet is az alkalmazás főablakában.



A felületen elhelyezett elemek

Típus (Class)	*Név (objectName)*	*Beállítások, megjegyzés*
QWidget	InvoiceForm	windowTitle: Invoicer
QTableView	tableView	
QHeadView	headView	Vezérlő beillesztése a Designerbe: QWidget + jobb klikk + Promote
QLineEdit	grandTotal	

Először **Qt Designerrel** megtervezzük a felületet, melyet **invoiceform.ui** néven mentünk el . Ezután a felülettervet használva, származtatással, „szövegszerkesztővel” elkészítjük az **InvoiceForm** osztályt.

invoiceform.h

```
#include <QWidget>
#include "ui_invoiceform.h"
#include "invoice.h"

class QModelIndex;
```

```
class ItemsModel;
class HeadModel;
class HeadView;

class QAction;

class InvoiceForm : public QWidget, public Ui_InvoiceForm
{
    Q_OBJECT

public:
     InvoiceForm(QWidget *parent=0);
    ~InvoiceForm();

protected:
    void closeEvent(QCloseEvent *event);

public:
    void newFile();
    bool open();
    bool openFile(const QString &fileName);
    bool save();
    bool saveAs();

    QAction* windowMenuAction() const {return action;}
    int itemCount();
    const QString& getCurrentFile(){return currentFile;}

private:
    bool okToContinue();
    bool saveFile(const QString &fileName);
    void setCurrentFile(const QString &fileName);
    QString strippedName(const QString &fullFileName);

private slots:
    void documentWasModified();

public slots:
    void insertItemBefore();
    void insertItemAfter();
    void deleteItem();
    void updateGrandTotal();

private:
    Invoice* mInvoice;
    ItemsModel* mItemsModel;
    HeadModel* mHeadModel;

    QString currentFile;
    bool isUntitled;
    QString fileFilters;
    QAction* action;
};
```

invoiceform.cpp

```
#include <QMessageBox>
#include <QModelIndex>
#include <QFileDialog>
#include <QCloseEvent>

#include "invoice.h"
#include "invoiceform.h"
#include "itemsmodel.h"
#include "headmodel.h"
#include "headview.h"
#include "utils.h"

InvoiceForm::InvoiceForm(QWidget *parent)
    : QWidget(parent)
{
    setupUi(this); //Setup GUI elements created by designer

    action = new QAction(this);
    action->setCheckable(true);
    connect(action, SIGNAL(triggered()), this, SLOT(show()));
    connect(action, SIGNAL(triggered()), this, SLOT(setFocus()));

    setWindowIcon(QPixmap(":/images/document.png"));
    setAttribute(Qt::WA_DeleteOnClose);
    setWindowTitle(strippedName(currentFile) + "[*]");
    isUntitled = true;
    fileFilters =  tr("Invoice files (*.inv);; Any files (*)");

    mInvoice=new Invoice(this);

    mItemsModel = new ItemsModel(this);
    mItemsModel->setItems(mInvoice->items());
    tableView->setModel(mItemsModel);

    mHeadModel = new HeadModel(this);
    mHeadModel->setHead(mInvoice->head());
    headView->setModel(mHeadModel);

    connect(mInvoice, SIGNAL(itemsModified()),mItemsModel, SLOT(resetModel()));
    connect(mInvoice, SIGNAL(headModified()),mHeadModel, SLOT(resetModel()));

    connect(mHeadModel, SIGNAL(modelModified()),this, SLOT(documentWasModified()));
    connect(mItemsModel, SIGNAL(modelModified()),this, SLOT(documentWasModified()));

    connect(mHeadModel, SIGNAL(modelModified()), mInvoice, SLOT(setModified()));
    connect(mItemsModel, SIGNAL(modelModified()), mInvoice, SLOT(setModified()));

    connect(mItemsModel, SIGNAL(modelModified()),this, SLOT(updateGrandTotal()));

    mInvoice->newInvoice();
    setWindowModified(false);
}
```

A konstruktorban létrehozott *action* két állapotú esemény objektum kezeli az ablak (számla) kiválasztást

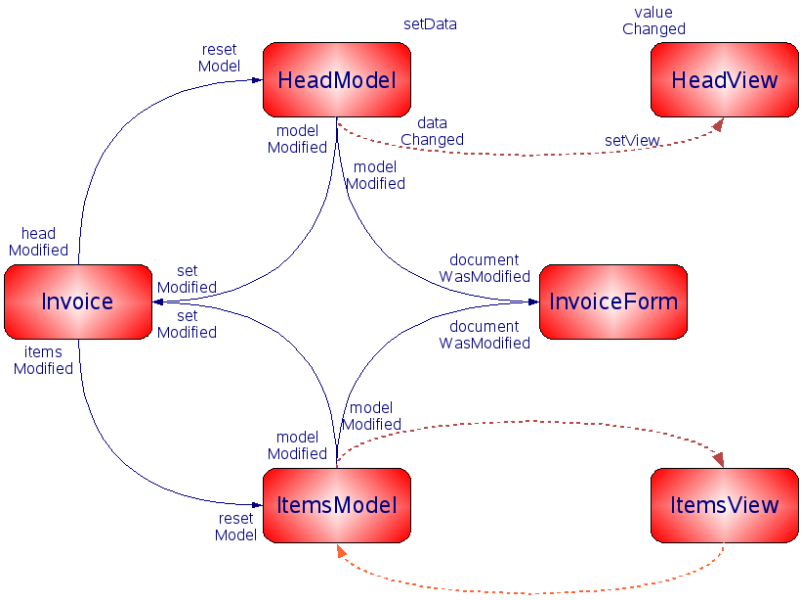
(*triggered()*). Ha a felhasználó erre az ablakra kattint, akkor ez lesz az aktív ablak, ezt az ablakot kell megjeleníteni (*show()*), és a kurzornak ebben az ablakban kell lennie (*setFocus()*). A *Qt::WA_DeleteOnClose* attribútum hatására az ablak bezárásakor automatikusan felszabadul az ablak által elfoglalt tárhely. (Alapértelmezésben az ablak bezárása csak az ablak elrejtését eredményezi.)

Minden **QWidget** rendelkezik egy *windowModified* tulajdonsággal, amelyet igazra állítunk, ha a vezérlő adata megváltozott. A *setWindowTitle()* függvény paraméterében a [*] azt jelzi, hol szeretnénk megjeleníteni a dokumentum megváltozását jelző csillagot.

A konstruktorban létrehozzuk az űrlap adatait tartalmazó dokumentumot (számlát). A *modell-nézet* programtervezési mintának megfelelően először létrehozzuk a számlatételek *adatmodelljét*, majd hozzárendeljük azt az őt megjelenítő *nézet*hez (*tableView*). Ezután létrehozzuk a fejléc adatmodelljét, és hozzárendeljük a fejlécet megjelenítő nézethez.

A komponensek közötti kommunikációt a *jeladó-jelvevő* (*signal-slot*) összekapcsolásával valósítjuk meg. (*connect()*)

A kapcsolatok kiépítése után az *Invoice::newInvoice()* metódusban kibocsájtott jelek hatására a nézetek frissülnek. Végül a **QWidget** *windowModified* tulajdonságát hamisra állítjuk.




```
InvoiceForm::~InvoiceForm() { }
```

Bár vannak **new** operátorral létrehozott objektumaink, a destruktornak nem kell azokat felszabadítani, mert ezek az objektumok **QObject**-ből származtatott, szülővel rendelkező objektumok, így a Qt *szülő-gyermek mechanizmusa* miatt ezek az objektumok a szülő megszűnésekor automatikusan megsemmisülnek.

```
void InvoiceForm::deleteItem()
{
    QModelIndex index = tableView->currentIndex();
    if(!index.isValid()) return;

    mItemsModel->removeInvoiceItem(index);

    int nextRow=(index.row()<mItemsModel->rowCount())?
```

```
                    index.row():mItemsModel->rowCount()-1;
        if(nextRow >= 0)
            tableView->setCurrentIndex(mItemsModel->index(nextRow,index.column()));
}
```

A *deleteItem()* függvény az *Item/Delete* esemény hatására kerül végrehajtásra. Az *ItemsModel::removeInvoiceItem()* függvénnyel a modellből töröljük a kérdéses sort. Vegye észre, hogy a *nézet* frissítésével nem kell foglalkoznunk. A függvény feladata az aktuális sor beállítása.

```
void InvoiceForm::insertItemBefore()
{
    QModelIndex index = tableView->currentIndex();
    mItemsModel->addInvoiceItem(index);
    if (mItemsModel->rowCount() == 1)
        index = mItemsModel->index(0,0);
    tableView->setCurrentIndex(index);
    tableView->edit(index);
}
```

Az *insertItemBefore()* függvény az *Item/Insert Before* esemény hatására kerül végrehajtásra. Az *ItemsModel::addInvoiceItem()* függvény beilleszt a modellbe, az *index* által meghatározott helyre, egy új sort (számlatételt). Vegye észre, hogy a *nézet* komponenssel „nem csináltunk semmit”. A *modell-nézet* programtervezési minta alkalmazása miatt a *nézet* frissítése automatikusan megtörténik.

```
void InvoiceForm::insertItemAfter()
{
    QModelIndex index = mItemsModel->index(tableView->currentIndex().row()+1,
                                           tableView->currentIndex().column() );
    mItemsModel->addInvoiceItem(index);
    if (mItemsModel->rowCount() == 1)
        index = mItemsModel->index(0,0);
    else
        index = mItemsModel->index(tableView->currentIndex().row()+1,
                                   tableView->currentIndex().column() );
    tableView->setCurrentIndex(index);
    tableView->edit(index);
}
```

Az *insertItemAfter()* függvény az *Item/Insert After* esemény hatására kerül végrehajtásra. Az *ItemsModel::addInvoiceItem()* függvény illeszti be az *index* által meghatározott helyre az új sort (számlatételt) a modellbe. Vegye észre, hogy a *nézet* komponenssel „nem csináltunk semmit”. A *modell-nézet* programtervezési minta alkalmazása miatt a *nézet* frissítése automatikusan megtörténik.

```
void InvoiceForm::updateGrandTotal()
{
    grandTotal->setText(mItemsModel->grandTotal());
}
```

Az *updateGrandTotal()* jelkezelő függvény (*slot*) frissíti a képernyőn a számla „*mindösszesen*” adatát. A megjelenítendő adatot a *modell*ből nyerjük ki.

```
bool InvoiceForm::save()
{
    if(isUntitled)          return saveAs();
    else                    return saveFile(currentFile);
}
```

A save*()* függvény a *Fájl/Save* eseményhez tartozó metódus. Ha a fájlnak van neve (a fájlt *open()* művelettel

töltöttük be), akkor a meglévő névvel a *saveFile()* függvényt, egyébként a *saveAs()* függvényt kell meghívni.

```
bool InvoiceForm::saveAs()
{
    QString fileName = QFileDialog::getSaveFileName(this, tr("Save Invoice"),
                                Utils::getApplicationDirPrefix() + "invoices/" +
                                QDate::currentDate().toString("yyyy"),fileFilters);
    if (fileName.isEmpty()) return false;
    if (!fileName.contains("."))    return saveFile(fileName + ".inv");
    else                            return saveFile(fileName);

}
```

A *saveAs()* függvényben a *QFileDialog::getSaveFileName()* metódusával bekérjük a felhasználótól a fájl nevét. Ha a felhasználó a *Cancel* gombra kattint, akkor hamis értékkel térünk vissza, amit visszatérési értékként továbbítunk a hívónak. (*save()*). Ha az adott néven már létezik fájl, akkor a *getSaveFileName()* függvény megkérdezi, felülírható-e ez a fájl.

```
bool InvoiceForm::saveFile(const QString &fileName)
{
    if(!mInvoice->saveAs(fileName)) return false;
    setCurrentFile(fileName);
    return true;
}

void InvoiceForm::newFile()
{
    static int documentNumber = 1;
    currentFile = tr("invoice%1.inv").arg(documentNumber);
    setWindowTitle(currentFile + "[*]");
    action->setText(currentFile);
    isUntitled=true;
    ++documentNumber;
}
```

A *newFile()* függvény a *Fájl/New* esemény hatására kerül végrehajtásra A *documentNumber* statikus változóban tároljuk, hogy az alkalmazásban eddig összesen hány dokumentumot (számlát) nyitottunk meg. A *newFile()* függvény generál egy nevet az új dokumentum számára (pl. *invoice1.inv*).

Minden **QWidget** rendelkezik egy *windowModified* tulajdonsággal, amelyet igazra állítunk, ha a vezérlő adata megváltozott. A dokumentum megváltozását szokták a címke sorban megjelenő csillaggal jelezni. A *setWindowTitle()* függvény paraméterében a [*] marker azt jelzi, hol szeretnénk megjeleníteni a dokumentum megváltozását jelző *-ot.

```
bool InvoiceForm::open()
{
    QString fileName =
        QFileDialog::getOpenFileName(this, tr("Open Invoice"),
                                Utils::getApplicationDirPrefix() + "invoices/" +
                                QDate::currentDate().toString("yyyy"), fileFilters);
        if (fileName.isEmpty())
            return false;
        return openFile(fileName);
}
```

Az *open()* függvényben a *QFileDialog::getOpenFilename()* metódusával lekérdezzük a megnyitandó fájl nevét.

A függvény első paramétere a dialógusablak *szülője*. Dialógus ablakoknál a szülő szerepe eltér a Qt-ben megszokott szülő-gyermek mechanizmustól. A dialógusablak mindig *önálló* ablak. Ha megadjuk a szülőt, akkor a dialógus ablak a szülő ablak közepén jelenik meg. A negyedik paraméter a fájllok szűrésére szolgál. A *fileFilters* változó a konstruktorban kapott értéket. Az értékül adott tr("Invoice files (*.inv);; Any files (*)") string azt jelzi, hogy *.inv* kiterjesztésű fájlokat szeretnénk megnyitni.

```
bool InvoiceForm::openFile(const QString &fileName)
{
    if(!mInvoice->load(fileName))
        return false;
    setCurrentFile(fileName);
    tableView->update();
    return true;
}
```

Az *openFile()* metódusban a számla Invoice::load() függvénnyel töltjük be a számla tartalmát. Beállítjuk az aktuális fájl nevet, és frissítjük a nézetet.

```
void InvoiceForm::setCurrentFile(const QString &fileName)
{
    currentFile = fileName;
    isUntitled = false;
    action->setText(strippedName(currentFile));
    mInvoice->setModified(false);
    setWindowTitle(strippedName(currentFile) + "[*]");
    setWindowModified(false);
}
```

A *setCurrentFile()* függvényben kitöltjük a *currentFile* privát változót az éppen szerkesztett fájl nevével. Mielőtt megjelenítenénk a főablak címke sorában a fájl nevét, a *strippedName()* függvénnyel leszedjük a fájl elérési útvonalát. Minden **QWidget** rendelkezik a *windowModified* tulajdonsággal, melyet akkor kell igazra állítani, amikor a dokumentum (számla) tartalma megváltozott. Ha egy dokumentum megváltozott, és még nem mentettük el, akkor ezt a tényt az ablak címke sorábana név mellett található * karakter jelzi. A **Qt** erről maga gondoskodik. A mi feladatunk mindössze a *windowModified* tulajdonság karbantartása.

```
void InvoiceForm::closeEvent(QCloseEvent *event)
{
    if(okToContinue())
        event->accept();
    else
        event->ignore();
}
```

A *closeEvent()* függvény akkor kapja meg a vezérlést, amikor a felhasználó a *Fájl/Exit, Fájl/Close, Fájl/Close All* menüpontok valamelyikét választja ki, vagy rákattint az ablak bezáró gombra (x). Ez egy „*close*” eseményt küld a widget-nek. A *QWidget::closeEvent()* felül definiálásával félbeszakíthatjuk a bezárás folyamatát, és eldönthetjük, valóban bezárhatjuk-e az ablakot.

```
bool InvoiceForm::okToContinue()
{
    if(isWindowModified()) {
        int ans = QMessageBox::warning(this, tr("Invoice"),
                    tr("Invoice \"") + strippedName(currentFile) +
                    tr("\" has been modified.\n Do you want to save your changes?"),
                    QMessageBox::Yes | QMessageBox::Default,
```

```
                    QMessageBox::No,
                    QMessageBox::Cancel|QMessageBox::Escape);
            if( ans == QMessageBox::Yes)
                return save();
            else if (ans == QMessageBox::Cancel)
                return false;
        }
        return true;
    }
```

Az *okToContinue()* függvényben megvizsgáljuk az *isWindowModified* tulajdonságot. Ha a dokumentumot (számlát) módosítottuk, akkor az *okToContinue()* privát függvény megkérdezi a felhasználót, valóban be akarja-e zárni az ablakot. A függvény hamis értéket ad vissza, ha a felhasználó a *Cancel* gombra kattint.

```
QString InvoiceForm::strippedName(const QString &fullFileName)
{
    return QFileInfo(fullFileName).fileName();
}
```

A *strippedName()* metódus visszaadja a fájl elérési útvonal nélküli nevét.

```
void InvoiceForm::documentWasModified()
{
    setWindowModified(true);
}
```

A *documentWasModified()* jelkezelő (slot) igazra állítja a widget *windowModified* tulajdonságát.

```
int InvoiceForm::itemCount()
{
    if(mInvoice)
        return mInvoice->items()->count();
    else
        return 0;
}
```

Az *itemCount()* függvény megadja hány számlatétele van a számlának.

Főablak elkészítése

A főablak a felhasználói felületnek biztosít keretet, menüpontokkal, státusz sorral, eszközgombokkal.

Eseménykezelés QAction osztállyal

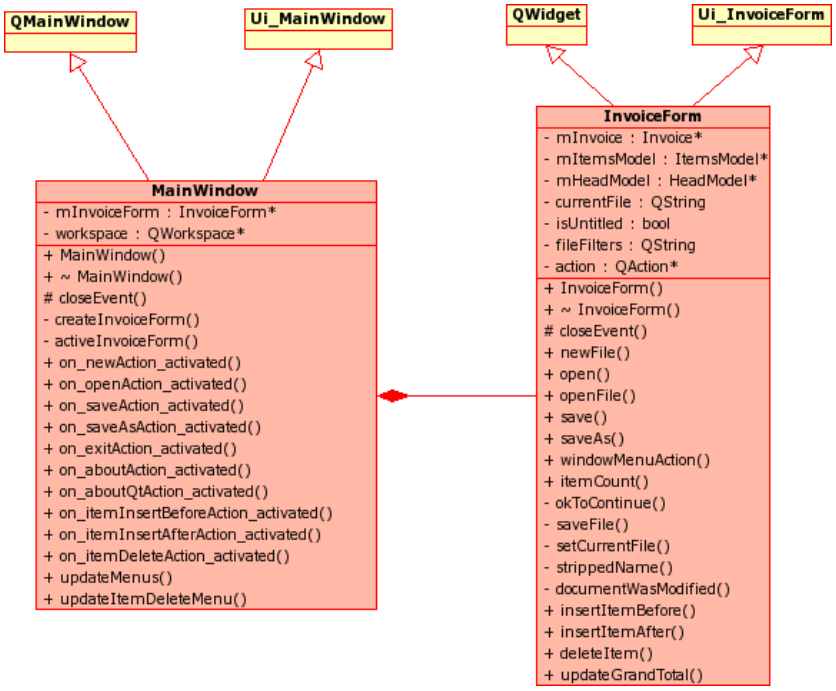
A grafikus alkalmazásoknál többféleképpen is kezdeményezhetünk egy-egy tevékenységet. Egy fájlt elmenthetünk **menüből**, **eszköztárral** (*toolbar*) vagy **gyorsbillentyű** (*accelerator*) segítségével is.

A **QAction** osztály a felhasználói felületről kezdeményezhető tevékenységek kezelésére szolgáló osztály. Egy tevékenység kezelésére egyetlen tevékenység objektumot (*action*) kell létrehozni, melyet az alkalmazásunkban több helyről is aktivizálhatunk. A tevékenységhez rendelt összes vezérlő mindenkor a tevékenység pillanatnyi állapotát tükrözi.

A tevékenység megjelenését meghatározó jellemzőket is a tevékenység objektumban kell megadni: **ikon**, **menü szövege**, **státusz szövege**, "**what is this**" **szöveg**, **súgó szöveget** (*tooltip*). Ezek a beállítások futási időben módosíthatók.

A *setCheckable()* függvénnyel a tevékenység objektumot két állapotúra (*toggle action*) állíthatjuk be (*pl. Bold toolbar eszközgomb)*. Ekkor az állapotváltást az objektum *toggled()* üzenete jelzi. Parancs típusú (*command action*) tevékenységeknél (*pl. File open menüpont)* a tevékenység aktivizálását a *triggered()* üzenet jelzi.

Osztálydiagram



Grafikus felület kialakítása

A főablakot **Qt Designer** segítségével tervezzük meg, majd származtatással készítjük el. Qt Designerben hozzon létre egy főablakot (Main Window), és helyezze el az alkalmazás menüpontjait az alábbiak szerint:

Menüpont	*Név (objectName)*	*Típus (Class)*	*Beállítások*
File	fileMenu	QMenu	
New	newAction	QAction	text: &New icon: new.png toolTip: New File statusTip: Creates a new invoice whatsThis: New File\n\nCreates a new document shortcut: Ctrl+N
Open	openAction	QAction	text: &Open icon: open.png toolTip: Open File statusTip: Opens an existing invoice whatsThis: Opens an existing document shortcut: Ctrl+O
Save	saveAction	QAction	text: &Save icon: save.png toolTip: Save File statusTip: Saves the current invoice whatsThis: Saves the current invoice shortcut: Ctrl+S
SaveAs	saveAsAction	QAction	text: Save &As icon: - toolTip: Save File As ... statusTip: Saves the current invoice whatsThis: Saves the current invoice shortcut: -
Close	closeAction	QAction	text: Cl&ose icon: toolTip: Close Window statusTip: Close Window whatsThis: Close Window shortcut: Ctrl+F4
CloseAll	closeAllAction	QAction	text: Close &All
Exit	exitAction	QAction	text: E&xit statusTip: Exit the application shortcut: Ctrl+Q
Window	windowMenu	QMenu	
Tile	tileAction	QAction	text: &Tile
Cascade	cascadeAction	QAction	text: &Cascade
Next	nextAction	QAction	text: Ne&xt shortcut: Ctrl+F6
Previous	previousAction	QAction	text: Pre&vious shortcut: Ctrl+Shift+F6
Help	helpMenu	QMenu	
About	aboutAction	QAction	text: &About

Menüpont	*Név (objectName)*	*Típus (Class)*	*Beállítások*
About Qt	aboutQtAction	QAction	text: About &Qt
Item	itemMenu	QMenu	
Insert Before	itemInsertBeforeAction	QAction	text: Insert &Before
Insert After	itemInsertAfterAction	QAction	text: Insert &After
Delete	itemDeleteAction	QAction	text: &Delete

Megjegyzés: Itt csak a tervezővel elhelyezhető elemeket soroltuk fel. Az alkalmazás tartalmaz dinamikusan keletkező menüpontokat is, melyet programból helyezzük rá a főablakra. (pl.utoljára használt fájlok,)

A felület tervet mentse el **mainwindow.ui** néven. A felület tervhez tartozó osztályból származtatással készítjük el a **MainWindow** osztályt.

mainwindow.h

```
#include <QMainWindow>
#include "ui_mainwindow.h"

class InvoiceForm;
class QWorkspace;
class QActionGroup;

class MainWindow : public QMainWindow, public Ui_MainWindow
{
    Q_OBJECT
 public:
    MainWindow(QMainWindow *parent = 0);
    ~MainWindow();

    void openFile(const QString &fileName);

protected:
    void closeEvent(QCloseEvent *event);

private:
    InvoiceForm* createInvoiceForm();
    InvoiceForm* activeInvoiceForm();
    void writeSettings();
    void readSettings();

private slots:
    void openRecentFile();
    void createRecentFileMenus();
    void updateRecentFileActions(const QString& fileName);
    void updateRecentFileActions();

public slots:
    void on_newAction_activated();
    void on_openAction_activated();
    void on_saveAction_activated();
    void on_saveAsAction_activated();
```

```
    void on_exitAction_activated();
    void on_aboutAction_activated();
    void on_aboutQtAction_activated();

    void on_itemInsertBeforeAction_activated();
    void on_itemInsertAfterAction_activated();
    void on_itemDeleteAction_activated();

    void updateMenus();
    void updateItemDeleteMenu();

private:
    InvoiceForm* mInvoiceForm;
    QWorkspace* workspace;
    QActionGroup* windowActionGroup;

    enum {MaxRecentFiles = 5};
    QAction* recentFileActions[MaxRecentFiles];
    QAction* separatorAction;
    QStringList recentFiles;
};
```

mainwindow.cpp

```
#include <QMessageBox>
#include <QWorkspace>
#include <QCloseEvent>
#include "invoiceform.h"
#include "mainwindow.h"

MainWindow::MainWindow(QMainWindow *parent)   : QMainWindow(parent)
{
    setupUi(this); //Setup GUI elements created by designer
    workspace = new QWorkspace;
    setCentralWidget(workspace);

    windowActionGroup = new QActionGroup(this);
    createRecentFileMenus();

    connect(workspace,SIGNAL(windowActivated(QWidget*)),this,SLOT(updateMenus()));
    connect(closeAction, SIGNAL(triggered()),workspace, SLOT(closeActiveWindow()));
    connect(closeAllAction, SIGNAL(triggered()),workspace, SLOT(closeAllWindows()));
    connect(tileAction, SIGNAL(triggered()), workspace, SLOT(tile()));
    connect(cascadeAction, SIGNAL(triggered()), workspace, SLOT(cascade()));
    connect(nextAction, SIGNAL(triggered()),workspace, SLOT(activateNextWindow()));
    connect(previousAction, SIGNAL(triggered()),workspace, SLOT(activatePreviousWindow()));

    setWindowTitle(tr("MDI Invoicer"));
    readSettings();
    updateMenus();
}
```

Az alkalmazás központi ablakaként létrehozunk egy *QWorkspace* objektumot, amely a feldolgozás alatt álló űrlapokat (számlákat) tartalmazza. A Window menüpontban megjelenítjük a megnyitott számlákat. A myitott számlák menüpontjait a windowActionGroup-ban elhelyezett QAction-ok reprezentálják.

A *workspace windowActivated()* jelzésére aktualizáljuk a kiválasztott ablak menüpontjait (*updateMenus()*). A Close, Close All, Tile, Cascade, Next, Previous menüpontokhoz a QWorkspace osztály által szolgáltatott eseménykezelőket rendeljük. A *readSettings()* metódussal beolvassuk az alkalmazásunk legutolsó állapotát. Ezzel lehetővé válik, hogy a legutobb használt számlákat feltüntessük a fájl menüpontban.

```
MainWindow::~MainWindow()
{ }

InvoiceForm* MainWindow::createInvoiceForm()
{
    InvoiceForm* invoiceForm = new InvoiceForm;
    workspace->addWindow(invoiceForm);
    windowMenu->addAction(invoiceForm->windowMenuAction());
    windowActionGroup->addAction(invoiceForm->windowMenuAction());
    return invoiceForm;
}
```

A *createInvoiceForm()* függvényben létrehozunk egy új számla űrlapot, hozzávesszük a *workspace* által kezelt dokumentumokhoz, és visszaadjuk az új számlára mutató poinert. A *Window* menühöz hozzáveszünk egy, a számlánkat reprezentáló *QAction* típusú eseményt. Az eseményt az InvoiceForm szolgáltatja (az InvoiceForm *action* adattagja). Ezt az eseményt felvesszük egy *QActionGroup*-ba is, mellyel biztosítjuk, hogy a *Window* menüpontban megjelenő számlákhoz tartozó események közül mindig csak egy legyen „kiválasztott” (*checked*). Amikor egy számlát bezárunk, akkor megszűnik a számla űrlapja. Az űrlappal együtt törlődnek az ő gyerekei, többek között az action adattagja is. Az *action* adattag megszűnésével a neki megfelelő menüpont törlődik a Window menüből is.

```
void MainWindow::on_newAction_activated()
{
    statusBar()->showMessage(tr("Creating new file..."),2000);
    InvoiceForm *invoiceForm = createInvoiceForm();
    invoiceForm->newFile();
    invoiceForm->show();
    statusBar()->showMessage(tr("Ready."),2000);
}
```

Az *on_newAction_activated()* eseménykezelő a *newAction* hatására (New menüpont, New eszközgomb, Ctrl+N billentyűk) kerül végrehajtásra. Először létrehozunk egy új számla űrlapot, majd meghívjuk az aktuális űrlap *InvoiceForm::newFile()* metódusát. Az újonnan létrehozott űrlapot megjelenítjük a képernyőn (show()).

Vegye észre, hogy az eseménykezelő nevét a Qt4 névkonvenciójának megfelelően választottuk, így nem kellett a konstruktorban a *connect()* függvénnyel a *newAction* eseményt és annak eseménykezelőjét összekapcsolni.

```
void MainWindow::on_openAction_activated()
{
    statusBar()->showMessage(tr("Opening file..."),2000);
    InvoiceForm *invoiceForm = createInvoiceForm();
    if(invoiceForm->open()) {
        invoiceForm->show();
        updateRecentFileActions(invoiceForm->getCurrentFile());
        statusBar()->showMessage(tr("File opened"),2000);
    } else {
        invoiceForm->close();
        statusBar()->showMessage(tr("File open failed"),2000);
    }
}
```

Az *on_openAction_activated()* eseménykezelő a *openAction* hatására (Open menüpont, Open eszközgomb,

Ctrl+O billentyűk) kerül végrehajtásra. Először létrehozunk egy új számla űrlapot, majd végrehajtjuk az aktuális űrlap *InvoiceForm::open()* metódusát. Ha sikerült a kiválasztott számla megnyitása (betöltése), akkor megjelenítjük a képernyőn, és ezt a számlát felvesszük az utoljára nyitott fájlok listájába, és megjelenítjük a File menűben is (*updateRecentFileActions()*).

Ha nem sikerült a számlát megnyitni, akkor egyszerűen csak bezárjuk a számlát, hiszen a nyitás sikertelenségéről az *InvoiceForm::close()* metódusában már tájékoztattuk a felhasználót. Az űrlapot nem kell törölni, mert az *InvoiceForm* osztályban beállított *Qt::WA_DeleteOnClose* attribútum miatt a számla űrlap automatikusan törlődik.

```
void MainWindow::openFile(const QString &fileName)
{
    statusBar()->showMessage(tr("Opening file..."),2000);
    InvoiceForm *invoiceForm = createInvoiceForm();
    if(invoiceForm->openFile(fileName)) {
        updateRecentFileActions(invoiceForm->getCurrentFile());
        invoiceForm->show();
        statusBar()->showMessage(tr("File opened"),2000);
    } else {
        invoiceForm->close();
        statusBar()->showMessage(tr("File open failed"),2000);
    }
}
```

Az *openFile()* publikus függvény feladata egy adott nevű dokumentum betöltése. Ezt a függvényt akkor használjuk, amikor a betöltendő számlák neveit a program indításakor, parancssorban adjuk meg.

```
void MainWindow::on_saveAction_activated()
{
    statusBar()->showMessage(tr("Saving file..."),2000);
    if(activeInvoiceForm()){
        activeInvoiceForm()->save();
        updateRecentFileActions(activeInvoiceForm()->getCurrentFile());
    }
    statusBar()->showMessage(tr("File saved"),2000);
}
```

Az *on_saveAction_activated()* eseménykezelő az *InvoiceForm::save()* metódusát hívja meg, ha van számla. A mentéssel kapcsolatos tényleges munkát az *InvoiceForm* osztály végzi. Az elmentett számla (dokumentum) neve bekerül az utoljára használt fájlok listájába és a File menüpontba.

```
void MainWindow::on_saveAsAction_activated()
{
    statusBar()->showMessage(tr("Saving file..."),2000);
    if(activeInvoiceForm()) {
        activeInvoiceForm()->saveAs();
        updateRecentFileActions(activeInvoiceForm()->getCurrentFile());
   }
    statusBar()->showMessage(tr("File saved"),2000);
}
```

Az *on_saveAsAction_activated()* eseménykezelő az *InvoiceForm::saveAs()* metódusát hívja meg az aktív számlára. A mentéssel kapcsolatos tényleges munkát az *InvoiceForm* osztály végzi. Az elmentett számla nevét elhelyezzük az utoljára használt fájlok listájába, valamint a File menüpontba.

```
void MainWindow::on_exitAction_activated()
```

```
{
    close();
}
```

Az *exitAction* esemény kiváltásakor végrehajtjuk a *MainWindow::close()* függvényét. A függvényben megnézzük, van-e módosított számla és rákérdezünk azok elmentésére.

```
void MainWindow::on_aboutAction_activated()
{
}
```

Nincs implementálva. A függvény megvalósítását az olvasóra bízzuk. ( )

```
void MainWindow::on_aboutQtAction_activated()
{ }
```

Nincs implementálva. A függvény megvalósítását az olvasóra bízzuk. ( )

```
void MainWindow::on_itemInsertBeforeAction_activated()
{
    activeInvoiceForm()->insertItemBefore();
    updateItemDeleteMenu();
}
```

Ezt az eseménykezelőt az *itemInsertBeforeAction* esemény hatására hajtjuk végre. Ezt a menüpontot csak akkor kezdeményezhetjük, ha valóban van aktív űrlapunk (lásd: *updateMenus()* függvény). A számlatétel beszúrása után frissítjük az *Item/Delete* menüpont elérhetőségét. (Az *Item/Delete* menüpont csak akkor legyen aktív, ha van legalább egy számlatételünk.).

```
void MainWindow::on_itemInsertAfterAction_activated()
{
    activeInvoiceForm()->insertItemAfter();
    updateItemDeleteMenu();
}
```

Ezt az eseménykezelőt az *itemInsertAfterAction* esemény hatására hajtjuk végre. Ezt a menüpontot csak akkor kezdeményezhetjük, ha valóban van aktív űrlapunk (lásd: *updateMenus()* függvény). A számlatétel beszúrása után frissítjük az *Item/Delete* menüpont elérhetőségét. (Az *Item/Delete* menüpont csak akkor legyen aktív, ha van legalább egy számlatételünk.).

```
void MainWindow::on_itemDeleteAction_activated()
{
    activeInvoiceForm()->deleteItem();
    updateItemDeleteMenu();
}
```

Ezt az eseménykezelőt az *itemDeleteAction* esemény hatására hajtjuk végre. Ezt a menüpontot csak akkor kezdeményezhetjük, ha valóban van aktív űrlapunk (lásd: *updateMenus()* függvény), és valóban van legalább egy számlatételünk. A számlatétel törlése után frissítjük az *Item/Delete* menüpont elérhetőségét. (Az *Item/Delete* menüpont csak akkor legyen aktív, ha van legalább egy számlatételünk.).

```
void MainWindow::updateItemDeleteMenu()
{
    itemDeleteAction->setEnabled(activeInvoiceForm() && (activeInvoiceForm()->itemCount() != 0));

}
```

Az *Item/Delete* menüpont csak akkor legyen aktív, ha van legalább egy űrlapunk és legalább egy számlatételünk

```
void MainWindow::updateMenus()
{
    bool hasInvoiceForm = (activeInvoiceForm() != 0);
    saveAction->setEnabled(hasInvoiceForm);
    saveAsAction->setEnabled(hasInvoiceForm);
```

```
    closeAction->setEnabled(hasInvoiceForm);
    closeAllAction->setEnabled(hasInvoiceForm);
    tileAction->setEnabled(hasInvoiceForm);
    cascadeAction->setEnabled(hasInvoiceForm);
    nextAction->setEnabled(hasInvoiceForm);
    previousAction->setEnabled(hasInvoiceForm);

    itemInsertBeforeAction->setEnabled(hasInvoiceForm);
    itemInsertAfterAction->setEnabled(hasInvoiceForm);
    updateItemDeleteMenu();

    if(activeInvoiceForm())
        activeInvoiceForm()->windowMenuAction()->setChecked(true);
}
```

Az *updateMenus()* függvényt akkor hívjuk meg, amikor egy ablak aktívvá válik, vagy amikor az utolsó ablakot zárjuk be. A legtöbb menüpontnak csak akkor van értelme, ha az ablak aktív, így ennek megfelelően állítjuk be az egyes menüpontok elérhetőségét. Végül az *InvoiceForm* action adattagjának checked attribútumát igazra állítva jelezzük, hogy ez a számla lesz az aktív űrlapunk. A korábbi ablak aktív állapotát nem kell „kikapcsolni” (uncheck), mert a QActionGroup tagság miatt ez automatikusan megtörténik.

```
InvoiceForm* MainWindow::activeInvoiceForm()
{
    return qobject_cast<InvoiceForm*>(workspace->activeWindow());
}
```

Az *activateInvoiceForm()* privát függvény visszaadja az aktív gyerek ablakra mutató pointert, ha van ilyen, egyébként pedig null értékkel tér vissza.

```
void MainWindow::closeEvent(QCloseEvent *event)
{
    workspace->closeAllWindows();
    if(activeInvoiceForm())
        event->ignore();
    else {
        writeSettings();
        event->accept();
    }
}
```

A *closeEvent()* eseménykezelő felüldefiniálásával minden egyes gyerek ablakra kiváltjuk a *close()* eseményt (*closeAllWindow()*). Ha valamelyik gyerek ablak érvényteleníti a *close* eseményt (a felhasználó a „nem mentett változások” üzenet ablakon a *Cancel* gombra kattintott), akkor a főablakon is érvénytelenítjük (*event->ignore()*), egyébként pedig elfogadva továbbítjuk (*event->accept()*) ezt az eseményt.

```
void MainWindow::createRecentFileMenus()
{
    //create actions
    for (int i=0; i < MaxRecentFiles; ++i) {
        recentFileActions[i] = new QAction(this);
        recentFileActions[i]->setVisible(false);
        connect(recentFileActions[i],SIGNAL(triggered()),this, SLOT(openRecentFile()) );
    }

    //update and create menus
    fileMenu->removeAction(exitAction);
    for (int i=0; i < MaxRecentFiles; ++i) {
```

```
            fileMenu->addAction(recentFileActions[i]);
        }
        separatorAction = fileMenu->addSeparator();
        fileMenu->addAction(exitAction);
    }
```

A *createRecentFileMenus()* függvénnyel illesztjük be a File menübe az utoljára használt számlák (dokumentumok) menü pontjait. Először létrehozzuk a menüpontok eseményeit (QAction), majd beillesztjük a megfelelő helyre. Egy pillanatra kivesszük a menüpontok közül az Exit menüpontot, elhelyezzük az utoljára használt számlák menüpontjait reprezentáló QAction típusú változókat, majd egy vonallal elválasztva visszatesszük az Exit menüpontot. Ha a fájl menüben valamelyik, korábban használt számla nevére kattintunk, akkor az openRecentFile() eseménykezelőt kell végrehajtani. Ezt adtuk meg a connect()-ben.

```
void MainWindow::openRecentFile()
{
    QAction* action = qobject_cast<QAction*>(sender());
    if(action)
        openFile(action->data().toString());
}
```

Az *openRecentFile()* eseménykezelő (slot) feladata az utoljára megnyitott fájlok listájából kiválasztott számla megnyitása. Az eseménykezelőben (slot) meghívott *sender()* függvény visszatérési értéke a hívó objektumra mutató pointer, mellyel megtudhatjuk az üzenetet küldő adatait, így kinyerhetjük az eseményt kiváltó fájl nevét.

```
void MainWindow::updateRecentFileActions(const QString& fileName)
{
    recentFiles.removeAll(fileName);
    recentFiles.prepend(fileName);
    updateRecentFileActions();
}
```

Először kiszedjük az utoljára használt fájlok listájából a paraméterben kapott fájlt, azért, hogy minden fájl név csak egyszer szerepeljen ebben a listában, majd a lista elejére elhelyezzük azt. Ezután frissítjük a File menü utoljára használt fájljainak menüpontjait.

```
void MainWindow::updateRecentFileActions()
{
    QMutableStringListIterator i(recentFiles);
    while(i.hasNext()) {
        if(!QFile::exists(i.next()))
            i.remove();
    }

    for(int j=0; j < MaxRecentFiles; ++j) {
        if (j < recentFiles.count()) {
            QString text = tr("%1 %2").arg(j+1).arg(QFileInfo(recentFiles[j]).fileName());
            recentFileActions[j]->setText(text);
            recentFileActions[j]->setData(recentFiles[j]);
            recentFileActions[j]->setVisible(true);
        } else {
            recentFileActions[j]->setVisible(false);
        }
        separatorAction->setVisible(!recentFiles.isEmpty());
    }
}
```

Az utoljára használt fájlok menüpontjainak frissítését azzal kezdjük, hogy kivesszük a listából a nem létező fájlok menüpontjait. A megmaradó fájlokra beállítjuk és láthatóvá tesszük a rájuk vonatkozó menüpontokat. A maradék menüpontot elrejtjük (*setVisibla(false)*).

```
void MainWindow::writeSettings()
{
    QSettings settings("ELTE IK EAF3","Invoicer" );
    settings.setValue("geometry",geometry());
    settings.setValue("recentFiles",recentFiles);
}

void MainWindow::readSettings()
{
    QSettings settings("ELTE IK EAF3","Invoicer" );

    QRect rect = settings.value("geometry", QRect(200,200,400,400)).toRect();
    move(rect.topLeft());
    resize(rect.size());

    recentFiles = settings.value("recentFiles").toStringList();
    updateRecentFileActions();
}
```

Qt-ben lehetőség van az alkalmazás beállításainak elmentésére. Az elmentett adatok kulcs-adat párban kerülnek elmentésre és az alkalmazás elindításakor ugyanilyen formában visszatölthetők.

Programunkban elmentjük az alkalmazás elhelyezkedésére és méretére vonatkozó adatokat, valamint az utoljára használt fájlok elérési útvonalát és nevét. Az alkalmazás elindításakor betöltjük az utoljára használt fájl neveket és a File menüben létrehozzuk a nekik megfelelő menüpontokat.

Főprogram

```
#include <QApplication>
#include <QSplashScreen>
#include <QPixmap>
#include <QThread>
#include "mainwindow.h"

int main (int argc, char *argv[])
{
    QApplication app(argc,argv);

    QSplashScreen *splash = new QSplashScreen;
    splash->setPixmap(QPixmap("./images/splash.jpg"));
    splash->show();

    Qt::Alignment topRight = Qt::AlignRight | Qt::AlignTop;

    splash->showMessage(QObject::tr("Welcome ..."));
    MainWindow *mainWindow = new MainWindow;

    QStringList args = app.arguments();
    if (args.count() > 1) {
        for (int i=1; i < args.count(); i++)
            mainWindow->openFile(args[i]);
    }

    mainWindow->show();
    sleep(3);
    splash->finish(mainWindow);
```

```
    return app.exec();

}
```

A számlakezelő alkalmazásunk főprogramjában az eddigiekhez képest két újdonságot találunk.

Ha a felhasználó fájl neveket ad meg a *parancssorban*, akkor betöltjük az adott nevű fájlokat. A Qt specifikus opciókat a Qt automatikusan leszedi a parancssorból, így pédául az

```
invoicer -style motif invoice01.inv
```

parancs esetén a *QApplication::arguments()* lista két elemet tartalmaz:"*invoicer*" és „*invoice01.inv*" elemeket és a programunk az invoice01.inv számlával indul.

Az alkalmazás indításakor a képernyőn egy *„üdvözlő" képet* jelenítünk meg (*QSplashScreen*). Ezt a megoldást általában akkor szokták alkalmazni, amikor a program betöltése időigényes, mert ily módon tájékoztatják a felhasználót a betöltés egyes fázisairól. Esetünkben programunk gyorsan betöltődik, ezért egy kis várakoztatást is beletettünk a kezdőkép megjelenítéséhez.

A munkafüzet programjai letölthetők a people.inf.elte.hu/nacsa/qt4/eaf3/inv03/projects/ címről.

Tartalomjegyzék

A munkafüzet programjai letölthetők a people.inf.elte.hu/nacsa/qt4/eaf3/inv02/projects/ címről.

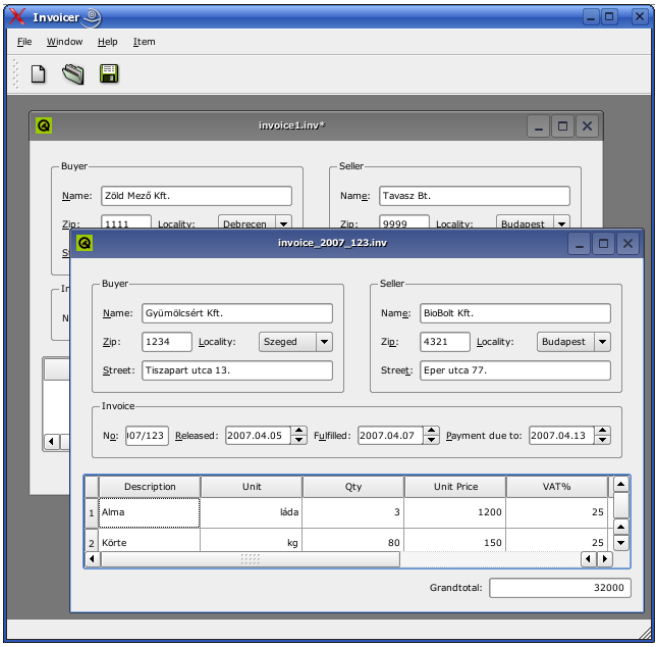
A munkafüzetben bemutatott programok készítésekor a Qt 4.2.2 verziót használtam.

Készítette: Szabóné Nacsa Rozália
email: nacsa@inf.elte.hu
honlap: people.inf.elte.hu/nacsa

Feladat

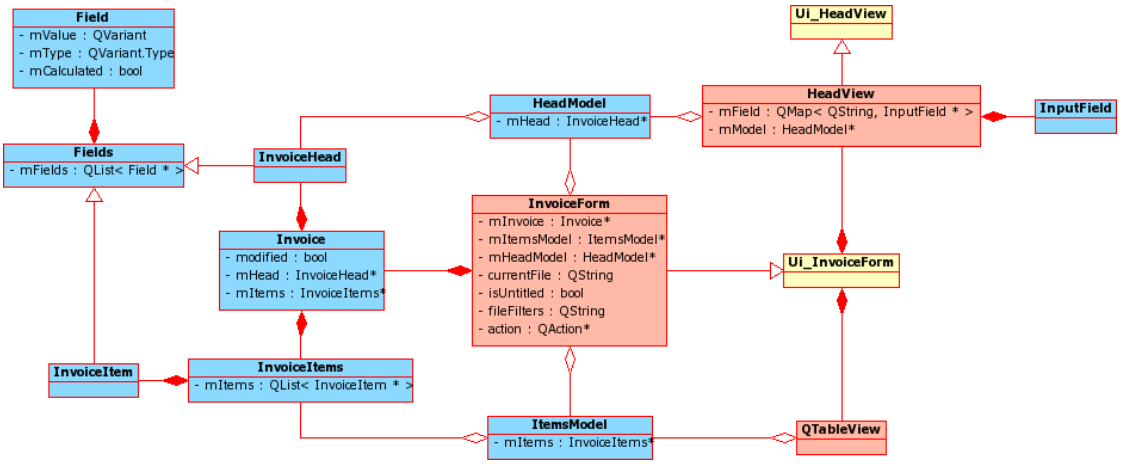
Készítsünk programot számlák előállítására, karbantartására. A számlákat tároljuk fájlokban. A program tegye lehetővé új számla létrehozását, meglévő számlák módosítását, a számlák (dokumentumok) mentését, betöltését.

Megjegyzés: Ez a munkafüzet a számlakészítő alkalmazás második munkafüzete. Ebben a munkafüzetben feltételezzük, hogy Ön már feldolgozta a „Qt 4 /C++ alapú MDI alkalmazás: Számlakészítő program 1” munkafüzetet.



MDI Számlakezelő program futás közben

Az alkalmazás osztálydiagramja

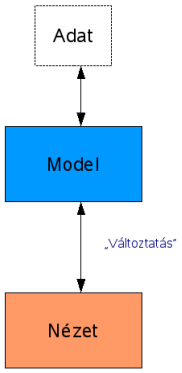


Az előző modulban már elkészítettük a **Field**, **Fields**, **InvoiceItem**, **InvoiceHead**, **Invoice** osztályokat. Ebben a modulban elkészítjük a számlakezelő alkalmazás fejléc-karbantartó és a számlatételeket karbantartó részprogramját. A fejléc karbantartó és a számlatételeket karbantartó osztályok működését, használatát megnézheti a **modelview_head**, **modelview_items** (teszt) projektekben.

Modell-nézet (model-view) tervezési minta (bevezetés)

A grafikus alkalmazásoknál használt vezérlők egy része az adatokat magában a vezérlőben tárolja. Ilyenkor a program a vezérlőben tárolt adatokon végzi el a szerkesztési, keresési műveleteket. Ez a megoldás nem megfelelő, amikor sok adattal dolgozunk, web-es alkalmazást készítünk, vagy ugyanazt az adatot többféleképpen is meg kell jeleníteni. Ilyen esetekben célszerű az adat és a nézet szétválasztása. Az adat és a nézet szétválasztása lehetővé teszi, hogy ugyanazt az adatot egyszerre több nézetben is megjelenítsük, továbbá a *modell-nézet* minta alkalmazásakor az adatstruktúra megváltoztatása nélkül is bevezethetünk új nézeteket.

A **modell-nézet (model-view)** minta a a SmallTalk-ból jól ismert **modell-nézet-vezérlő (model-view-controler)** mintán alapul. Az eredeti mintában az adattároló *modell* (*model*) és az adattartalmat megjelenítő *nézetek* (*view*) között a *vezérlő* (*controller*) biztosítja a kapcsolatot oly módon, hogy a *modell* minden változásáról értesítést küld a *nézeteknek*, így azok frissíteni tudják magukat. A *modell-nézet* mintában a *nézetet* és a *vezérlőt* összevonták. Az adatok tárolása és az adattartalom megjelenítése ebben a mintában is szétválik, de egyszerűbb, mint az eredeti *modell-nézet-vezérlő* minta.



A **Qt 4** egyik újdonsága az **Interview** modul, amely a **(model-view)** tervezési mintán alapuló programok készítésére alkalmas osztályokat tartalmazza. Projektünkben a fejléc és a számlatétel karbantartó programrészeket a **Qt 4 Interview** osztályait felhasználva, a *modell-nézet* programtervezési minta szerint készítjük el.

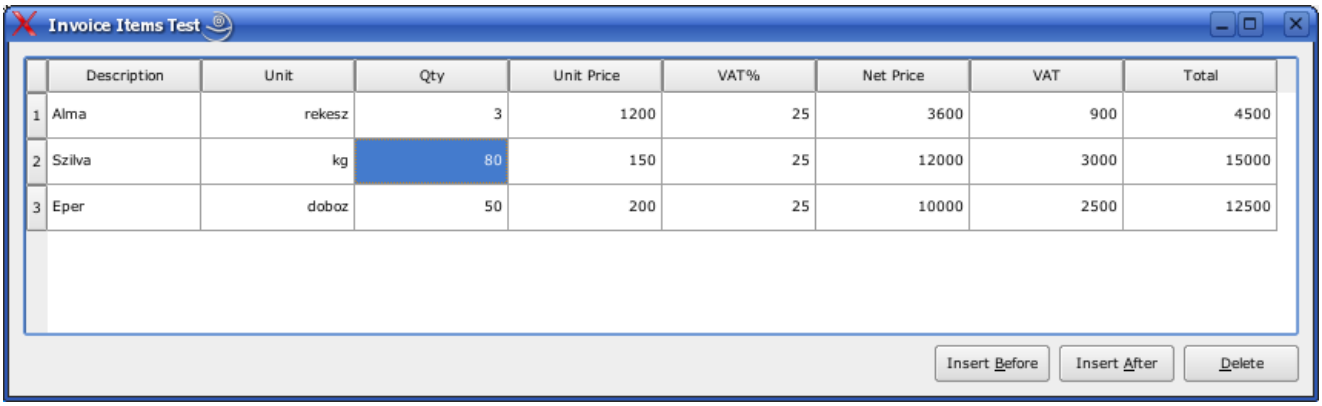
A **Qt4 QListView**, **QTreeView** és **QTableView** vezérlői az **Interview** modul *nézet osztályai*. Az ilyen vezérlőket úgy kell használni, hogy először elkészítjük a hozzá illő *modell* osztályt, majd a modellt hozzárendeljük a *nézethez*. Ezután a két komponens összehangoltan működik, és a karbantartási funkciókat automatikusan elvégzi. A Qt osztályai között vannak olyan absztrakt osztályok, amelyekből származtatással könnyen előállíthatunk adatszolgáltató *modell* osztályokat.

Modell-nézet mintát alkalmazó programrész:

```
FooModel* model = new FooModel;
FooView *  view = new FooView()
view->setModel(model);
```

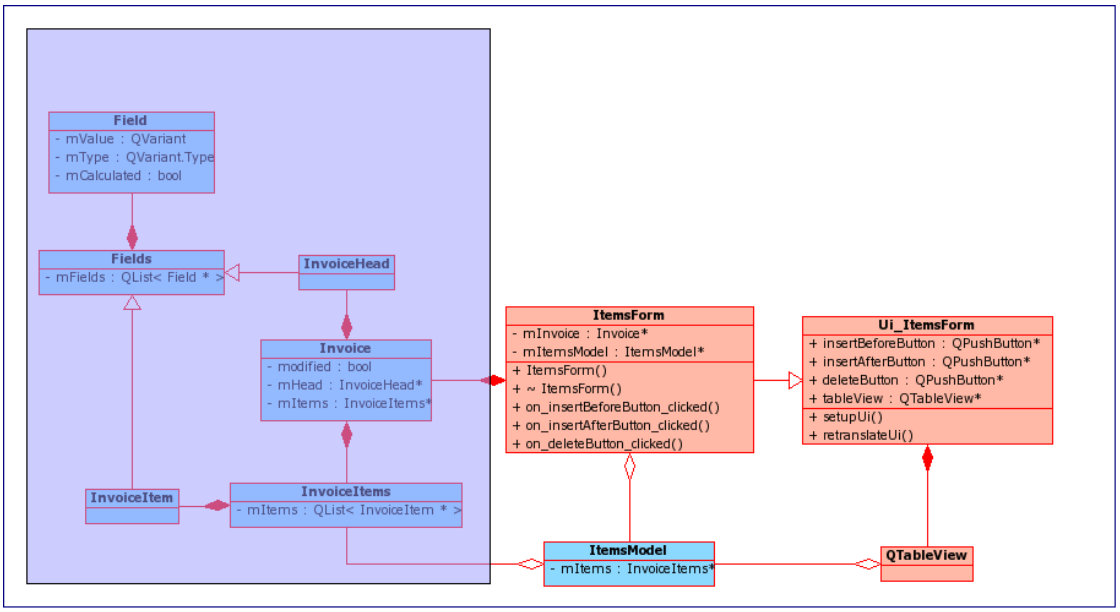
Számlatételek kezelése, karbantartása („táblázat kezelés”)

Mielőtt elkészítenénk a végleges számla karbantartó programot - az egyszerűség kedvéért - készítsünk el egy olyan kisebb (teszt) alkalmazást, amely csak a számlatételekre koncentrál. A projekt neve: **modelview_items**.



Számlatételkezelő alkalmazás működés közben – modelview_items projekt

A „táblázatkezelő” projekt osztálydiagramja:



A számlatételek bevitelét, karbantartását **QTableView** típusú adatvezérlővel oldjuk meg. Ehhez a vezérlőhöz készítünk egy adatszolgáltató *modell* osztályt (**ItemsModel**), melyet a **QAbstractTableModel** osztályból származtatva állítunk elő. A származtatott osztályban megvalósítjuk a a **QTableView** nézet osztály használatához szükséges metódusokat. A grafikus felületet tervezővel alakítjuk ki. A Designerrel megtervezett fájlt **itemsform.ui** néven mentjük el. Ezután származtatással elkészítjük az **ItemsForm** osztályt.

ItemsModel osztály

Az **ItemsModel** osztály számlatétel adatokat szolgáltató *modell* osztály, melyet a **QAbstractTableModel** osztályból készítünk el származtatással. Az elkészített *modell* osztály a **QTableView** *nézet* osztálynak szolgáltat adatokat, ezért az alábbi függvényeket mindenképpen definiálni kell:

rowCount(),columnCount(), data(), headerData()

Ha a megjelenített táblázat adatai szerkeszteni is lehet, akkor további négy függvény implementációját kell megadni:

itemFlags(), setData(),insertRows(), removeRows()

A függvények implementálása után összerakhatjuk a *modell* és *nézet* komponenseket. Az „összerendelés” után a karbantartás a táblakezelésnél elvárt módon működik. A *nézet* osztályon keresztül szerkeszthetjük a táblázatot, a *modell* osztályból pedig kinyerhejük az adatokat.

itemsmodel.h

```
#include <QAbstractTableModel>
#include <QAbstractItemModel>
#include "invoiceitems.h"

class ItemsModel : public QAbstractTableModel
{
    Q_OBJECT
public:
    ItemsModel(QObject* parent=0);
    ~ItemsModel() {}

    /* A QAbstractItemsModel miatt definiált metódusok*/
    int rowCount(const QModelIndex & parent = QModelIndex()) const;
    int columnCount(const QModelIndex & parent = QModelIndex()) const;
    QVariant data(const QModelIndex & index, int role = Qt::DisplayRole ) const;
    QVariant headerData (int section, Qt::Orientation orientation,
            int role = Qt::DisplayRole) const;

    Qt::ItemFlags flags ( const QModelIndex & index ) const;
    bool setData ( const QModelIndex & index, const QVariant & value, int role = Qt::EditRole );
    bool insertRows(int position, int rows,const QModelIndex &index = QModelIndex());
    bool removeRows(int position, int rows,const QModelIndex &index = QModelIndex());

    // Egyéb metódusok
    void addInvoiceItem(QModelIndex &index);
    void removeInvoiceItem(QModelIndex &index);
    void setItems(InvoiceItems* items);
    QString grandTotal();

    QString toString();

public slots:
    void resetModel();

signals:
    void dataChanged(QModelIndex,QModelIndex);
    void modelModified();

private:
```

```
    InvoiceItems* mItems;
};
```

itemsmodel.cpp

```
include <QStringList>
#include "itemsmodel.h"
#include "invoice.h"
#include "fields.h"
#include "field.h"

ItemsModel::ItemsModel(QObject* parent) : QAbstractTableModel(parent), mItems(NULL)
{
        setObjectName(QString::fromUtf8("ItemsModel"));
}
int ItemsModel::rowCount(const QModelIndex &/* parent*/) const
{
    return mItems->count();
}
int ItemsModel::columnCount(const QModelIndex &/*parent*/) const
{
    return InvoiceItem::titles().count();
}
```

A *modell* a *rowCount*() és *columnCount*() függvényekkel „árulja el” magáról, hány sora és oszlopa van. A *nézet* osztály csak a nézethez szükséges információkat igényli, a megjelenítés nem függ attól, hogy a *modell* osztály hogyan ábrázolja, illetve honnan szerzi meg az adatait.

```
QVariant ItemsModel::headerData (int section, Qt::Orientation orientation, int role) const
{

    if(role != Qt::DisplayRole) return QVariant();
    if (orientation == Qt::Horizontal)
        return (InvoiceItem::titles())[section];
    else
        return section + 1;
}
```

A *modell* minden eleméhez több, különböző szerepet játszó (különböző funkciójú) adat tartozik. Ezekkel a „szerepekkel” jelezzük a nézet osztály számára, hogy a szolgáltatott adatnak mi a célja. Így maghatározhatjuk egy elem küllemét, értékét, háttér színét, súgó szövegét, stb. (Qt::DisplayRole, Qt::EditRole, Qt::TooltipRole, Qt:BackgroundRole, Qt:TextColorRole, Qt::UserRole)

A *headerData*() metódus a táblázat oszlopainak és sorainak címke-feliratát szolgáltatja. Az *orientation* paraméterből megtudhatjuk, hogy a megjelenítő *nézet* osztály sor vagy oszlop címkét kér. A *section* paraméter megadja, hanyadik sorról, illetve oszlopról van szó. A *role* paraméter azt jelzi, „miért” (milyen tevékenység okán) hívták meg ezt a függvényt (adat megjelenítése, igazítás, háttérszín beállítás, stb,).

Az oszlop feliratokat az **InvoiceItem** osztály *titles()* statikus függvényéből nyerhetjük ki. A *section* paraméter a kérdéses oszlop sorszáma. A sorok címkéje az adott sor sorszáma.

```
QVariant ItemsModel::data(const QModelIndex & index, int role) const
{
    if(!index.isValid()) return QVariant();

    if (role == Qt::TextAlignmentRole) {
        if (index.column() == 0)
            return int(Qt::AlignLeft | Qt::AlignVCenter);
        else
```

```
                return int(Qt::AlignRight | Qt::AlignVCenter);
        } else if (role == Qt::DisplayRole) {
                return mItems->item(index.row())->field(index.column())->value();
        }
        return QVariant();
}
```

A *data()* függvény az *index*-szel azonosított elem (*cella*) valamelyik adatát (a *role* paramétertől függően értéket, színt, igazítás módját, stb.) szolgáltatja. A táblázat egy-egy elemére a *modell* osztályhoz tartozó, **QModelIndex** típusú *index* változóval hivatkozhatunk. Az így definiált *index* csak rövid ideig „él", ezért mindig rá kell kérdezni, érvényes-e.

```
bool ItemsModel::
setData ( const QModelIndex & index, const QVariant & value, int role )
{
    if(index.isValid() && role == Qt::EditRole){
        mItems->item(index.row())->field(index.column())->setValue(value);
        emit dataChanged(index,index);
        emit modelModified();
        return true;
    } else
        return false;
}
```

A *setData()* metódussal definiáljuk, hogyan tárolja el a modell a paraméterben megkapott adatot. Az index paraméterből nyerhetjük ki, hogy a tábla mely eleméről van szó. Az index változó két fontos attribútuma (táblás modell esetén), a sor és az oszlopértékek, melyet az index.row(), index.column() ad meg. Esetünkben a paraméterül kapott értéket be kell írni a számlatételek megfelelő sorának a megfelelő mezejébe. Az adatelem megváltozásáról jelzést küldünk.

```
void ItemsModel::resetModel()
{
    reset();
    emit modelModified();
}
```

A *resetModel()* függvény jelzéseket küld, melyek hatására a hozzárendelt nézet osztályok frissíthetik megjelenített adataikat.

```
Qt::ItemFlags ItemsModel::
flags(const QModelIndex &index) const {
  if (!index.isValid())
    return Qt::ItemIsEnabled;
    if(index.isValid() && index.column() > 4) //computef fields are read only
      return QAbstractItemModel::flags(index) | Qt::ItemIsEnabled;
    else
      return QAbstractItemModel::flags(index) | Qt::ItemIsEditable;

}
```

A *flags()* metódusban dönthetünk adataink szerkeszthetőségéről. A számlatételekben az utolsó oszlopok számított adatokat tartalmaznak, ezért ezek módosítását nem engedélyezzük (Qt::ItemIsEnabled).

```
bool ItemsModel::
insertRows(int position, int rows,const QModelIndex &parent)
{
    if (position==-1)
```

```
            position=rowCount()-1;
        beginInsertRows(parent, position, position + rows -1);
        for (int row = 0; row < rows; ++row)
            mItems->insertItem(position,new InvoiceItem());
        QModelIndex topLeft(index(position,0,parent));
        QModelIndex bottomRight(index(position+1,columnCount(),parent));

        emit dataChanged(topLeft,bottomRight);
        emit modelModified();
        endInsertRows();
        return true;
}
```

Az *insertRows()* metódussal a táblázat *position* sorától kezdődően *rows* darab sort szúrunk be. A *dataChanged()* szignállal jelezzük, mely cellák adatai változtak meg. Az adat megváltoztatásával modellünk is módosult, ezért erről is küldünk jelzést. A sorok beszúrása előtt mindig meg kell hívni a *beginInsertRows()*, majd befejezésnél az *endInsertRows()* függvényt.

```
bool ItemsModel:: removeRows(int position, int rows, const QModelIndex &parent)
{
    if(rowCount() == 0) return false;
    beginRemoveRows(parent, position, position + rows -1);
    for (int row = 0; row < rows; ++row)
        mItems->deleteItemAt(position);
    QModelIndex topLeft(index(position,0,parent));
    QModelIndex bottomRight(index(position+1,columnCount(),parent));

    emit dataChanged(topLeft,bottomRight);
    emit modelModified();
    endRemoveRows();
    return true;
}
```

A *removeRows()* függvény *rows* darab sort töröl az adatokból, és erről jelzést is küld. A sorok beszúrása előtt mindig meg kell hívni a *beginInsertRows()*, majd befejezésnél az *endInsertRows()* függvényt.

```
void ItemsModel::addInvoiceItem(QModelIndex &index)
{
    if(!index.isValid()){
        insertRows(rowCount(),1);
    }else{
        insertRows(index.row(),1);
    }
}

void ItemsModel::removeInvoiceItem(QModelIndex &index)
{
    if(!index.isValid() || rowCount() == 0)
        return;
    else
        removeRows(index.row(),1);
}
```

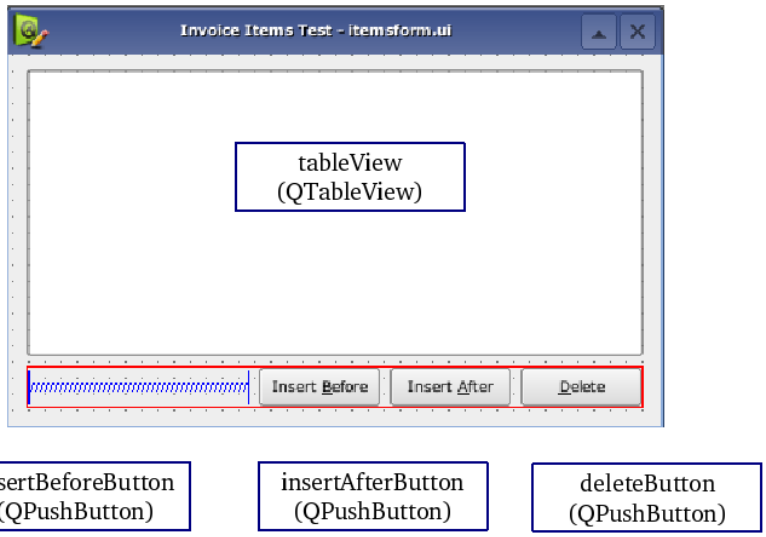
Az *addInvoiceItem()* és a *removeInvoiceItem()* függvénnyekkel egyetlen számlatételt tudunk beszúrni és törölni.

```
void ItemsModel::setItems(InvoiceItems* items)
{
```

```
        mItems = items;
        resetModel();
}

QString ItemsModel::grandTotal()
{
        return mItems->grandTotal();
}

QString ItemsModel::toString()
{
        QString ret = "\n Items Model ";
        for (int i=0; i< mItems->count(); ++i)
                ret += mItems->item(i)->toString();
        return ret;
}
```

Grafikus felület kialakítása

Készítsük el a számlatételeket karbantartó űrlapot.



A felületen elhelyezett elemek

Típus	*Név(objectName)*	*Beállítások, megjegyzés*
QWidget	ItemsForm	windowTitle: Invoice Items Test
QTableView	tableView	
QPushButton	insertBeforeButton	text = Insert &After
QPushButton	insertAfterButton	text = Insert &before
QPushButton	deleteButton	text = &Delete

Először **Qt Designerrel** megtervezzük a felületet, melyet **itemsform.ui** néven mentünk el. Ezután a felülettervet használva, származtatással, „szövegszerkesztővel” elkészítjük az **ItemsForm** osztályt.

itemsform.h

```
#include <QWidget>
#include "ui_itemsform.h"
#include "invoice.h"

class ItemsModel;
class ItemsForm : public QWidget, public Ui_ItemsForm {
    Q_OBJECT

public:
     ItemsForm(Invoice* invoice, QWidget *parent=0);
    ~ItemsForm(){}

public slots:
    void on_insertBeforeButton_clicked();
    void on_insertAfterButton_clicked();
    void on_deleteButton_clicked();

private:
    Invoice* mInvoice;
    ItemsModel* mItemsModel;
};
```

itemsform.cpp

```
#include <QMessageBox>
#include <QModelIndex>
#include "invoice.h"
#include "itemsform.h"
#include "itemsmodel.h"
#include "utils.h"

ItemsForm::ItemsForm(Invoice* invoice, QWidget *parent) : QWidget(parent), mInvoice(invoice)
{
    setupUi(this);

    mItemsModel = new ItemsModel(parent);
    mItemsModel->setItems(mInvoice->items());
    tableView->setModel(mItemsModel);

    if (mItemsModel->rowCount() > 0)
        tableView->setCurrentIndex(mItemsModel->index(0,0));
}
```

A konstruktorban létrehozzuk az adatszolgáltató modell egy példányát. A modellhez hozzárendeljük a számla megfelelő részét. A létrehozott modellt hozzárendeljük a kívánt megjelenítőhöz, és máris működik a táblakarbantartó program.

```
void ItemsForm::on_deleteButton_clicked() {
    QModelIndex index = tableView->currentIndex();
    if(!index.isValid()) return;

    mItemsModel->removeInvoiceItem(index);      //Aktuális számlatétel törlése

    int nextRow=(index.row()<mItemsModel->rowCount())?
                index.row():mItemsModel->rowCount()-1;
    if(nextRow >= 0){
        tableView->setCurrentIndex(mItemsModel->index(nextRow,index.column()));
```

```
    }
}
```

Az *on_deleteButton_clicked()* metódussal egy számlatételt törlünk a szálatételek listájából. Törlés után az aktuális sor fogalma nem definiált, ezért gondoskodunk arról, hogy a kurzor (ha mód van rá) valamilyen létező soron (számlatételen) álljon.

```
void ItemsForm::on_insertBeforeButton_clicked()
{
    QModelIndex index = tableView->currentIndex();

    mItemsModel->addInvoiceItem(index);    //Új számlatétel beillesztése

    if (mItemsModel->rowCount() == 1)
        index = mItemsModel->index(0,0);
    tableView->setCurrentIndex(index);
    if(index.isValid()) tableView->edit(index);
}
```

Az *on_insertBeforeButton_clicked()* metódussal egy új számlatételt illesztünk be az aktuális számlatétel elé. A beszúrandó sor indexének meghatározását a *modell* osztály *index()* függvényével adjuk meg.

```
void ItemsForm::on_insertAfterButton_clicked()
{
    QModelIndex index = mItemsModel->index(tableView->currentIndex().row()+1,
                        tableView->currentIndex().column());

    mItemsModel->addInvoiceItem(index);    //Új számlatétel beillesztése

    if (mItemsModel->rowCount() == 1)
        index = mItemsModel->index(0,0);
    else
        index = mItemsModel->index(tableView->currentIndex().row()+1,
                tableView->currentIndex().column());
    tableView->setCurrentIndex(index);
    if(index.isValid()) tableView->edit(index);
}
```

Az *on_insertAfterButton_clicked()* metódussal egy új számlatételt illesztünk be az aktuális számlatétel mögé. A beszúrandó sor indexének meghatározását a *modell* osztály *index()* függvényével adjuk meg.

A főprogram

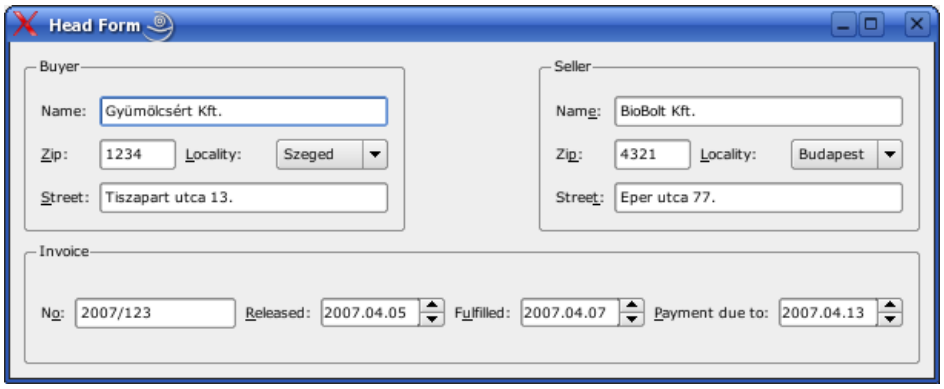
```
#include <QApplication>
#include "invoice.h"
#include "itemsform.h"

int main (int argc, char *argv[]) {
    QApplication app(argc,argv);
    Invoice* invoice = new Invoice();
    invoice->load("./invoices/2007/invoice_2007_123.inv");

    ItemsForm *itemsForm = new ItemsForm(invoice);
    itemsForm->show();

    bool ret = app.exec();
    invoice->saveAs("./invoices/2007/invoice_2007_567.inv");
    return ret;
}
```

A program letölthető a people.inf.elte.hu/nacsa/qt4/eaf3/inv02/projects/modelview_items címről.

Fejléc kezelése, karbantartása („Új űrlapkezelés”)

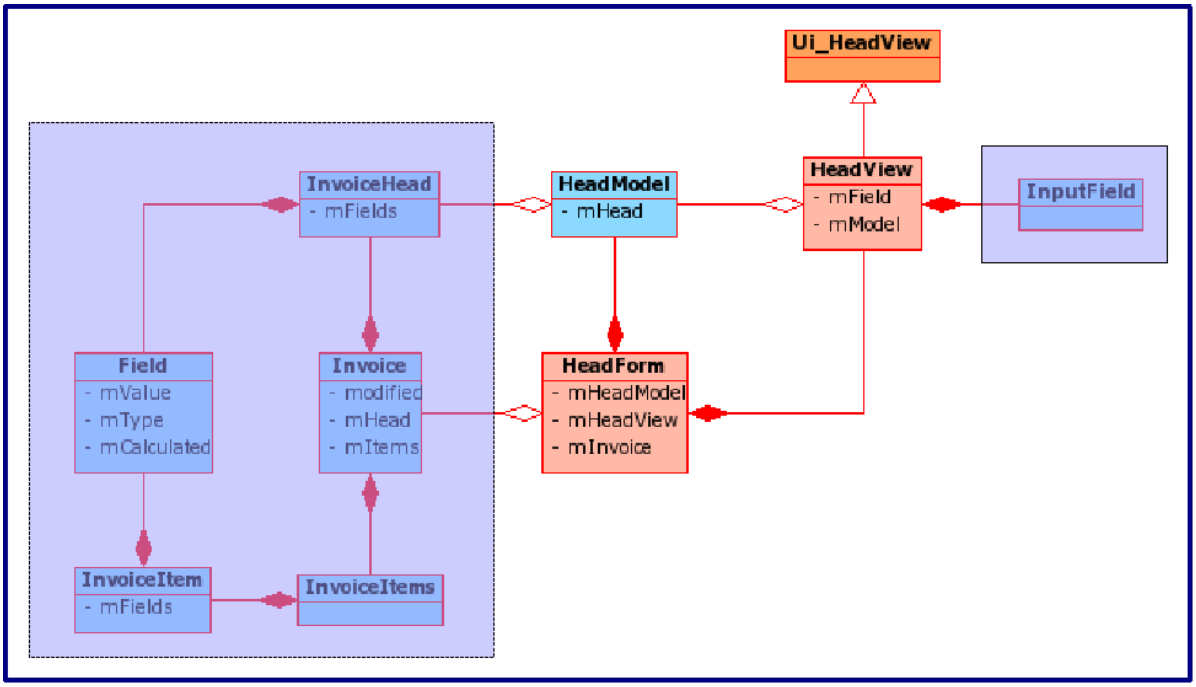
A modul második részében elkészítjük a számla-fejléc karbantartó programját, melyben a fejléc (űrlap) kezelését *modell-nézet* programozási minta szerint valósítjuk meg. A projekt neve: **modelview_head.**

A fejléc karbantartó programot úgy szeretnénk elkészíteni, hogy az könnyen újra felhasználható legyen más űrlapok karbantartására is („FormModel-FormView”). Az űrlapon található adatbeviteli mezők egységes kezelésére először készítünk egy absztrakt osztályt (**InputField**). A *nézet* osztály az űrlapon található adatbeviteli vezérlőket ilyen **InputField** típusú objektumokba „csomagolva” kezeli.

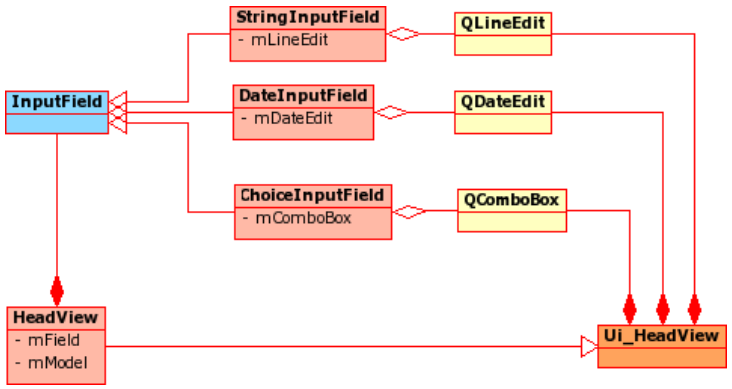


Számlafejléc-kezelő programrész működés közben – modelview_head projekt

A fejléc karbantartó program osztálydiagramja:



InputField absztrakt osztály



A *nézet* osztály adatbeviteli mezői **InputField** típusú objektumok. Egy-egy konkrét típusú adatbeviteli mezőt az **InputField** absztrakt osztályból állítjuk elő származtatással. A **InputField**-ből származtatott **StringInputField** példány az űrlapon szereplő **QLineEdit** típusú adatbeviteli mezőt kezelő objektum., a **DateInputField** példány a **QDateEdit** vezérlőkhöz készített **InputField**, a **ChoiceInputField** a **QComboBox**-okhoz készített InputField.

Az osztályok bevezetése után az űrlapon található adatbeviteli mezőket egységesen kezelhetjük. A *nézet* osztály **InputField** mezőjét és a *modell* osztály **Field** típusú adatát a bennük szereplő *name* attribútum köti össze.

inputfield.h

```
#include <QVariant>
#include <QString>
#include <QObject>

class InputField: public QObject {
   Q_OBJECT

public:
   InputField(QString name, QObject* parent=0);
   virtual QVariant value() const =0;
   virtual QWidget* widget() const = 0;

public slots:
   virtual void setView(QVariant newValue) = 0;
   virtual void clearView() = 0;
   virtual void setReadOnly(bool) = 0;

signals:
   void valueChanged(QVariant val);
};
```

inputfield.cpp

```
#include "inputfield.h"

InputField::InputField(QString name, QObject* parent) {
   setObjectName(name);
   setParent(parent);
}
```

StringInputField osztály

stringinputfield.h

```
#include "inputfield.h"

class QLineEdit;

class StringInputField : public InputField {
   Q_OBJECT
public:
   StringInputField(QLineEdit* lineEdit, QString name, QWidget* parent = 0);
   QVariant value() const ;
   QWidget* widget() const ;
public slots:
   void setReadOnly(bool v);
   void setView(QVariant qv);
   void clearView();
      void slotTextChanged(const QString&);
private:
   QLineEdit *mLineEdit;
};
```

stringinputfield.cpp

```
#include <QLineEdit>

#include "stringinputfield.h"

StringInputField::StringInputField(QLineEdit* lineEdit, QString name, QWidget* parent)
      : InputField(name, parent), mLineEdit(lineEdit) {
      connect(mLineEdit,SIGNAL(textChanged(const QString&)),this,SLOT(slotTextChanged(const
QString&)))
}

void StringInputField::slotTextChanged(const QString& text) {
      emit valueChanged(QVariant(text));
}

void StringInputField::setReadOnly(bool v) {
   mLineEdit->setReadOnly(v);
}

QVariant StringInputField::value()  const {
   return QVariant(mLineEdit->text());
}

void StringInputField::setView(QVariant qv) {
   mLineEdit->setText(qv.toString());
}
void StringInputField::clearView() {
   mLineEdit->setText(QString());
}
QWidget* StringInputField::widget() const {
   return mLineEdit;
}
```

DateInputField osztály

dateinputfield.h

```
#include "inputfield.h"
class QDateEdit;

class DateInputField : public InputField {
   Q_OBJECT
public:
   DateInputField(QDateEdit* dateEdit,QString name, QWidget* parent=0);
   QVariant value() const ;

   QWidget* widget() const ;
public slots:
   void setReadOnly(bool v);
   void setView(QVariant qv);
   void clearView();
      void slotDateChanged(const QDate & date);

private:
   QDateEdit* mDateEdit;
};
```

dateinputfield.cpp

```
#include <QDateEdit>
#include "dateinputfield.h"

DateInputField::DateInputField(QDateEdit* dateEdit, QString name ,QWidget* parent)
      : InputField(name, parent), mDateEdit(dateEdit) {
   mDateEdit->setDisplayFormat("yyyy.MM.dd");
      connect(mDateEdit,SIGNAL(dateChanged(const QDate &)),
            this,SLOT(slotDateChanged(const QDate &)));
}

void DateInputField::slotDateChanged(const QDate & date)
{
      emit valueChanged(QVariant(date));
}

void DateInputField::setReadOnly(bool v)
{
   mDateEdit->setReadOnly(v);
}

QVariant DateInputField::value() const
{
   return QVariant(mDateEdit->date());
}

void DateInputField::setView(QVariant qv)
{
   mDateEdit->setDate(qv.toDate());
}

void DateInputField::clearView()
{
```

```
    mDateEdit->setDate(QDate::currentDate());
}

QWidget* DateInputField::widget() const
{
    return mDateEdit;
}
```

ChoiceInputField osztály

choiceinputfield.h

```
#include "inputfield.h"

class QComboBox;

class ChoiceInputField : public InputField {
    Q_OBJECT

public:
    ChoiceInputField(QComboBox* comboBox, QString name, QWidget* parent = 0);
    QVariant value() const ;
    QWidget* widget() const ;

public slots:
    void setReadOnly(bool v);
    void setView(QVariant qv);
    void clearView();
    void slotTextChanged(const QString&);
private:
    QComboBox* mComboBox;
};
```

choiceinputfield.cpp

```
include <QComboBox>
#include "choiceinputfield.h"

ChoiceInputField::ChoiceInputField(QComboBox* comboBox, QString name, QWidget* parent)
:InputField(name, parent), mComboBox(comboBox)
{
    connect(mComboBox,SIGNAL(currentIndexChanged (const QString&)),
            this,SLOT(slotTextChanged(const QString&)));
}

void ChoiceInputField::slotTextChanged(const QString& text)
{
    emit valueChanged(QVariant(text));
}

QVariant ChoiceInputField::value() const
{
    return QVariant(mComboBox->currentText());
}

void ChoiceInputField::setReadOnly(bool /*v*/)
{
```

```
    mComboBox->setEditable(false);
}

void ChoiceInputField::setView(QVariant qv)
{
    QString vvalue = qv.toString();
    for (int i=mComboBox->count()-1; i>=0; --i) {
        QString text = mComboBox->itemText(i);
        if (text.startsWith(vvalue, Qt::CaseInsensitive)) {
            mComboBox->setCurrentIndex(i);
            return;
        }
    }
    mComboBox->setCurrentIndex(-1);
}

void ChoiceInputField::clearView()
{
    mComboBox->setCurrentIndex(-1);
    mComboBox->setEditable (true);
    mComboBox->clearEditText();
    mComboBox->setEditable(false);
}

QWidget* ChoiceInputField::widget() const
{
    return mComboBox;
}
```

HeadView osztály

headview.h

```
#include <QWidget>
#include <QList>
#include "inputfield.h"
#include "ui_headview.h"

class QLineEdit;
class HeadModel;

class HeadView : public QWidget, private Ui_HeadView {
    Q_OBJECT

public:
    HeadView(QWidget *parent = 0);
    ~HeadView();

    InputField* field(const QString& name) {return mField[name];}
    void setField(const QString& name, InputField* field){mField[name] = field;}
    QVariant value(const QString& fieldName);
    QStringList names();
    HeadModel* model() {return mModel;}

public slots:
    void setView(const QString& fieldName, QVariant newValue);
    void setModel(HeadModel* model);
```

```
        void initFields();
        void slotValueChanged(QVariant val);

    signals:
        void valueChanged(const QString& fieldName,QVariant val);

    private:
        QMap<QString, InputField*> mField;
        HeadModel* mModel;
};
```

headview.cpp

```
#include <QLineEdit>
#include <QDateTimeEdit>
#include <QList>
#include <QString>
#include <QStringList>
#include <QMap>

#include "stringinputfield.h"
#include "dateinputfield.h"
#include "choiceinputfield.h"
#include "headmodel.h"
#include "headview.h"

HeadView::HeadView(QWidget *parent) : QWidget(parent), mModel(NULL)
{
    setupUi(this);
    initFields();
}

HeadView::~HeadView()
{
    foreach(QString key, mField.keys())
        delete mField[key];
    mField.clear();
}

void HeadView::initFields()
{
    //init QLineEdit fields
    QList<QLineEdit *> lineEditList = findChildren<QLineEdit *>();
    foreach(QLineEdit* inputField, lineEditList)
    {
        mField[inputField->objectName()] =
                new StringInputField(inputField,inputField->objectName(),this);
          connect(mField[inputField->objectName()],SIGNAL(valueChanged(QVariant)),
                this,SLOT(slotValueChanged(QVariant)));
    }

    //init QDateEdit fields
    QList<QDateEdit *> dateEditList = findChildren<QDateEdit *>();
    foreach(QDateEdit * inputField, dateEditList)
    {
        mField[inputField->objectName()] =
                new DateInputField(inputField,inputField->objectName(),this);
```

```
            connect(mField[inputField->objectName()],SIGNAL(valueChanged(QVariant)),
                    this,SLOT(slotValueChanged(QVariant)));
        }

        //init QComboBox fields
        QList<QComboBox *> comboBoxList = findChildren<QComboBox *>();
        foreach(QComboBox * inputField, comboBoxList)
        {
            mField[inputField->objectName()] =
                    new ChoiceInputField(inputField,inputField->objectName(),this);
            connect(mField[inputField->objectName()],SIGNAL(valueChanged(QVariant)),
                    this,SLOT(slotValueChanged(QVariant)));
        }
}

void HeadView::slotValueChanged(QVariant value)
{
    emit valueChanged(QObject::sender()->objectName(),value);
}

QStringList HeadView::names()
{
    QStringList ret;
    foreach(QString key, mField.keys())
        ret << mField[key]->objectName();
    return ret;
}

void HeadView::setView(const QString& fieldName, QVariant value)
{
    if(field(fieldName)->value() == value) return;
    field(fieldName)->setView(value);
    emit valueChanged(fieldName, field(fieldName)->value());
}

QVariant HeadView::value(const QString& fieldName)
{
    return field(fieldName)->value();
}

void HeadView::setModel(HeadModel* model)
{
    if(!model) return;
    mModel = model;
    connect(mModel,SIGNAL(dataChanged(const QString&,QVariant)),
        this,SLOT(setView(const QString&,QVariant)));
    connect(this,SIGNAL(valueChanged(const QString&,QVariant)),
        mModel,SLOT(setData(const QString&,QVariant)));
}
```

HeadModel osztály

headmodel.h

```
#include <QObject>
#include <QMap>
```

```
#include "invoicehead.h"

class HeadModel : public QObject
{
    Q_OBJECT

public:
    HeadModel(QObject *parent=0);
    ~HeadModel() {}

    QVariant data(const QString& name);
    void setHead(InvoiceHead* head);
    QString toString();

public slots:
    void setData(const QString& name, QVariant data);
    void resetData(const QString& name, QVariant data);
    void resetModel();

signals:
    void dataChanged(const QString& name, QVariant data);
    void modelModified();

private:
    InvoiceHead* mHead;
};
```

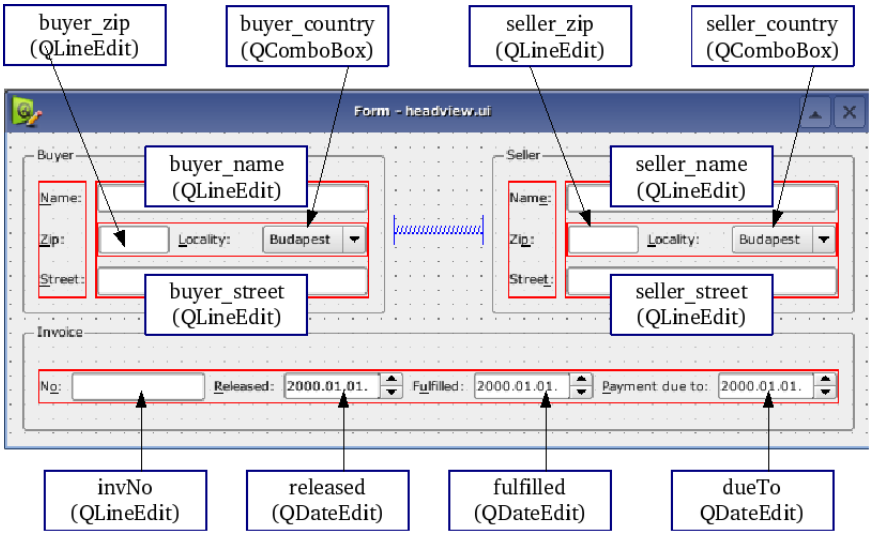
headmodel.cpp

```
#include <QStringList>
#include <QString>
#include <QMessageBox>
#include "headmodel.h"

HeadModel::HeadModel(QObject *parent)
    : QObject(parent), mHead(NULL)
{
        setObjectName(QString::fromUtf8("HeadModel"));
}

QVariant HeadModel::data(const QString& name)
{
    return mHead->field(name)->value();
}

void HeadModel::setData(const QString& name, QVariant data)
{
    if(mHead->field(name)->value() == data) return;
    mHead->field(name)->setValue(data);
    emit dataChanged(name,data);
    emit modelModified();
}

void HeadModel::setHead(InvoiceHead* head)
{
    mHead = head;
```

```
        resetModel();
}

void HeadModel::resetData(const QString& name, QVariant data)
{
        emit dataChanged(name,data);
        emit modelModified();

}

void HeadModel::resetModel()
{
        for (int i = 0; i < mHead->count(); ++i)
                resetData(mHead->field(i)->name(), mHead->field(i)->value());
}

QString HeadModel::toString()
{
        return mHead->toString();
}
```

Grafikus felület kialakítása

Készítünk egy űrlapot (QWidget), mellyel tesztelni tudjuk a táblázatunkat.



A felületen elhelyezett elemek

Típus	*Név(objectName)*	*Beállítások, megjegyzés*
QWidget	HeadForm	windowTitle: Head Form
QLineEdit	buyer_name	
QLineEdit	buyer_zip	
QComboBox	buyer_country	
QLineEdit	buyer_street	

Típus	*Név(objectName)*	*Beállítások, megjegyzés*
QLineEdit	seller_name	
QLineEdit	seller_zip	
QComboBox	seller_country	
QLineEdit	seller_street	
QLineEdit	invNo	
QDateEdit	released	
QdateEdit	fulfilled	
QDateEdit	dueTo	

Először **Qt Designerrel** megtervezzük a felületet, melyet **headform.ui** néven mentünk el. Ezután a felülettervet használva, származtatással, „szövegszerkesztővel” elkészítjük a **HeadForm** osztályt.

headform.h

```
#include <QWidget>

class HeadModel;
class HeadView;
class Invoice;

class HeadForm : public QWidget {
    Q_OBJECT

public:
    HeadForm(Invoice* invoice, QWidget *parent = 0 );
    ~HeadForm(){};

private:
    HeadModel* mHeadModel;
    HeadView* mHeadView;
    Invoice* mInvoice;
};
```

headform.cpp

```
#include "headform.h"
#include "headmodel.h"
#include "headview.h"
#include "invoice.h"

HeadForm::HeadForm(Invoice* invoice, QWidget *parent )
    : QWidget(parent), mInvoice(invoice)
{
    setWindowTitle("Head Form");

    mHeadModel = new HeadModel(this);
    mHeadModel->setHead(mInvoice->head());
    mHeadView = new HeadView(this);
    mHeadView->setModel(mHeadModel);
    mHeadModel->resetModel();
```

```
}
```

A konstruktorban először létrehozzuk a *modell* osztály egy példányát, megadjuk a *modell* adatforrását (*setHead()*), majd modellünket hozzárendeljük a megjelenítőhöz.

A főprogram

```
#include <QApplication>

#include "invoice.h"
#include "headform.h"

int main (int argc, char *argv[])
{
    QApplication app(argc,argv);
    Invoice* invoice = new Invoice();
    invoice->load("./invoices/2007/invoice_2007_123.inv");

    HeadForm *headForm = new HeadForm(invoice);
    headForm->show();

    bool ret = app.exec();
    invoice->saveAs("./invoices/2007/invoice_2007_567.inv");
    return ret;
}
```

A főprogramban létrehoztunk egy számlapéldányt, majd betöltöttük az *invoice_2007_123.inv* számlát. A számlát a **HeadForm** osztállyal szerkesztgethetjük. Szerkesztés után a *saveAs()* elmenti a módosításokat.

A munkafüzet programjai letölthetők a people.inf.elte.hu/nacsa/qt4/eaf3/inv02/projects/ címről.

A számlakészítő program elkészítéséhez lesz még egy munkafüzet!

Többnyelvűség biztosítása (Qt Linguist)

Egészítsük ki a decimális-hexadecimális konvertáló programunkat oly módon, hogy a felületen magyar szöveg jelenjen meg. Ehhez a **Qt Linguist** programot fogjuk használni.

Ahhoz, hogy az alkalmazásunk többféle nyelven is futtatható legyen Önnek az alábbiakat kell tennie:

1. Másolja át a **convert** projekt alábbi fájljait a **convert_tr** alkönyvtárba: (convertdialog.ui, convertdialog.h, convertdialog.cpp, main.cpp)
2. Készítse el a projekt leíró fájlt: qmake -project
3. Módosítsa a projekt leíró fájlt (convert.pro módosítása)

```
TEMPLATE = app
TARGET =
DEPENDPATH += .
INCLUDEPATH += .

# Input
HEADERS += convertdialog.h
FORMS += convertdialog.ui
SOURCES += convertdialog.cpp main.cpp
TRANSLATIONS += convert_hu.ts
```

1. Gyűjtse ki a grafikus szöveges részeket (string) a projekt programjaiból:

   lupdate -verbose convert_tr.pro

   Ez a program azokat a szövegrészeket gyűjti ki, melyeket a QString::fromUtf8("szöveg") alakban adtunk meg, azaz amelyeket a QString::fromUtf8 függvénnyel „becsomagoltunk”.
4. Készítse el a szótárat a Qt Linguis program segítségével.

   Indítsa el a programot, töltse be a convert_hu.ts fájlt és adja meg az angol szavak magyar megfelelőjét.
5. Készítse el az adott nyelv bináris fájlját:

   lrelease -verbose convert_tr.pro
6. Futási időben (main() függvényben) beállíthatja a kívánt nyelvet.

   Módosítsa a main() függvényt az alábbiak szerint:

```
#include <QApplication>
#include <QTranslator>
#include "convertdialog.h"

int main (int argc, char *argv[])
 {
      QApplication app(argc,argv);

      QTranslator translator;
      translator.load("convert_hu");
      app.installTranslator(&translator);

      ConvertDialog *dialog = new ConvertDialog;
      dialog->show();

      return app.exec();
}
```

A program letölthető a people.inf.elte.hu/nacsa/qt4/eaf3/mod01/projects/convert_tr címről.