



Eötvös Loránd Tudományegyetem
Informatikai Kar

Alkalmazott modul: Programozás

12. előadás

Grafikus felületű alkalmazások fejlesztése

Giachetta Roberto

`groberto@inf.elte.hu`

<http://people.inf.elte.hu/groberto>

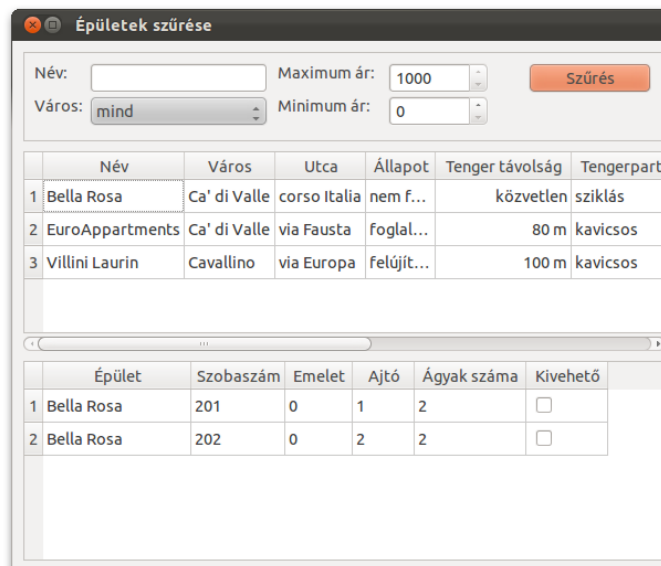
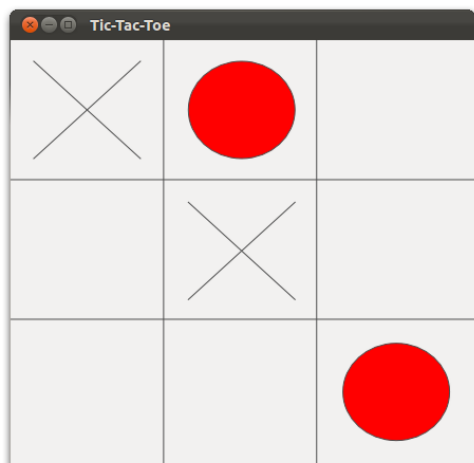
Grafikus felületű alkalmazások fejlesztése

A grafikus felületű alkalmazás

- *Grafikus felületű alkalmazásnak* nevezzük azt a programot, amely 2D-s interaktív felhasználó felületen (*GUI, Graphical User Interface*) keresztül kommunikál a felhasználóval
 - gazdagabb interakció a konzol felületnél, számos módon beleavatkozhatunk a programfutásba
 - a működés jórészt várakozás a felhasználói interakcióra
 - a felület egy, vagy több ablakból (*form/window*) áll, amelyek vezérlőket (*control/widget*) tartalmaznak (pl.: nyomógombok, listák, menük, ...)
 - mindig van egy aktív ablak, és egy aktív vezérlő (ezen van a *fókusz*)

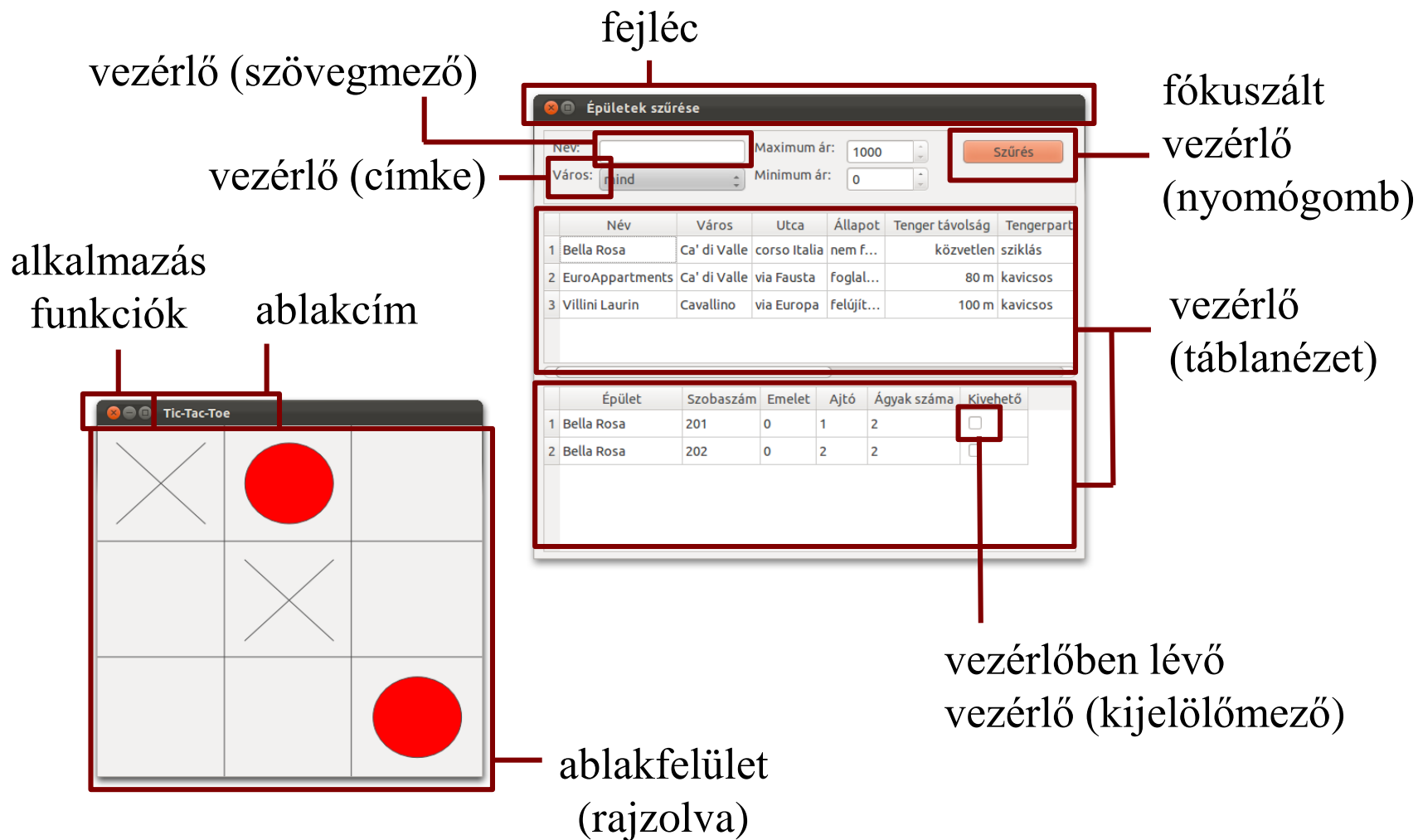
Grafikus felületű alkalmazások fejlesztése

A grafikus felületű alkalmazás



Grafikus felületű alkalmazások fejlesztése

A grafikus felületű alkalmazás



Grafikus felületű alkalmazások fejlesztése

A Qt keretrendszer

- A *Qt* egy alkalmazás-fejlesztési keretrendszer, amely számos platformot támogatja az asztali, mobil és beágyazott alkalmazások fejlesztését
 - elérhető a `qt.io` oldalról
 - támogatja a grafikus felületet, adatbázis-kezelést, multimédiát, 3D grafikát, hálózati és webes kommunikációt
 - rendelkezik nyílt forráskódú (LGPL) és kereskedelmi verzióval is
 - fejlesztésre elsősorban a C++-t támogatja, de más nyelvekre is elérhető, valamint rendelkezik saját leíró nyelvvel (Qt Quick)



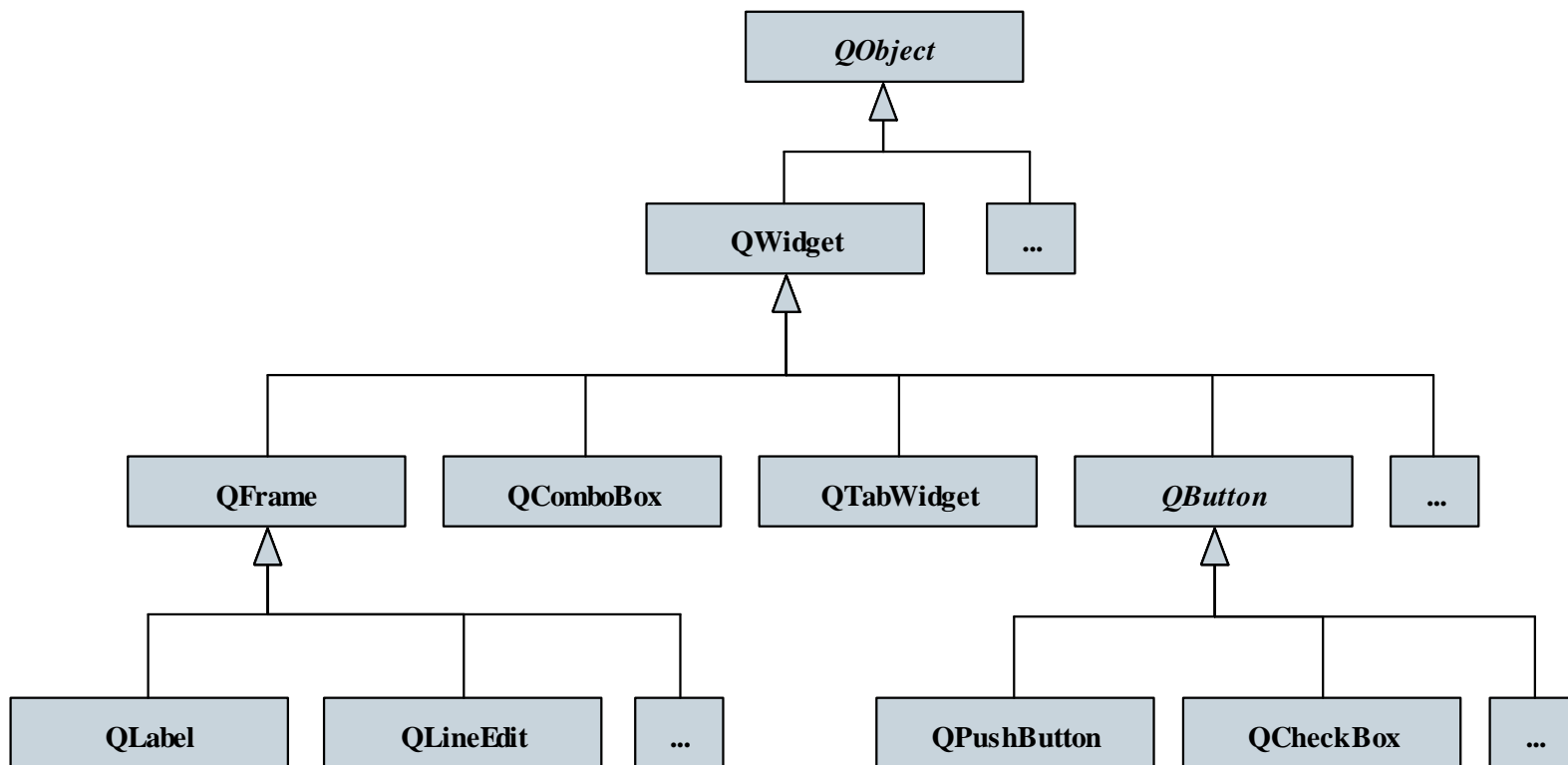
Grafikus felületű alkalmazások fejlesztése

A grafikus felület felépülése

- A grafikus felület *objektumorientáltan* épül fel
 - a vezérlőket osztályokként fogalmazzuk meg, megadjuk viselkedését (*metódusokkal*, pl. kattintás, megjelenés), illetve tulajdonságait (*mezőkkel*, pl. pozíció, méret, betűtípus)
 - a vezérlők sok hasonló tulajdonsággal bírnak, így könnyen öröklődési hierarchiába szervezhetőek
 - az öröklődési lánc legelején áll az általános vezérlő, új vezérlők származtatással definiálhatóak
 - a vezérlőket felhasználhatjuk más vezérlőkben, vagy használhatjuk önállóan, azaz *ablakként*

Grafikus felületű alkalmazások fejlesztése

A grafikus felület felépülése



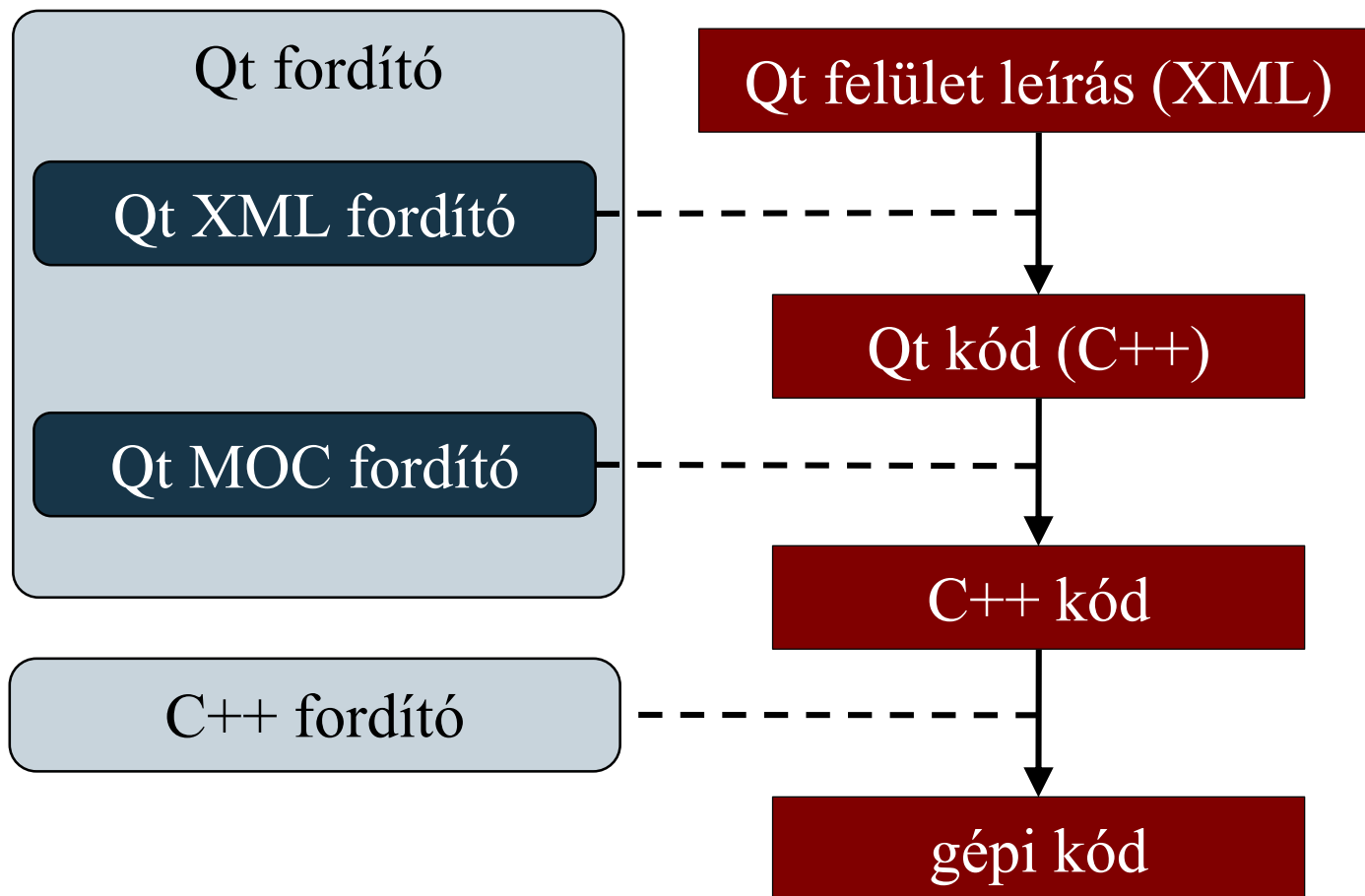
Grafikus felületű alkalmazások fejlesztése

Fejlesztés és fordítás

- A fejlesztés C++/Qt nyelven történik
 - elérhető a teljes C++ utasításkészlet, nyelvi könyvtár
 - a C++ nyelven felül további makrókat, kiegészítéseket tartalmaz, amelyeket a *Meta Object Compiler (MOC)* fordítja le ISO C++ kódra
- Az alapértelmezett fejlesztőeszköz a Qt Creator, de más környezetekben is megjelent a Qt fejlesztés lehetősége (pl. *Code::Blocks, Visual Studio*)
- Külön tervezőprogram (*Qt Designer*) adott a grafikus felület létrehozására, amely XML nyelven írja le a felület felépítését, ez automatikusan C++/Qt kódra fordul

Grafikus felületű alkalmazások fejlesztése

Fejlesztés és fordítás



Grafikus felületű alkalmazások fejlesztése

Fejlesztés és fordítás

- A fordítás projektszinten történik, a szükséges információk *projektfájlokban* (.pro) tárolódnak, amely tartalmazza
 - a felhasznált modulokat, kapcsolókat
 - forrásfájlok, erőforrások (pl. kép, szöveg,) listáját
 - eredmény paramétereket
- A fordítás közvetlenül is elvégezhető a fordítóval:

```
qmake -project
```

```
# projektfájl automatikus létrehozása
```

```
qmake # fordítófájl (makefile) előállítás
```

```
make # projekt fájlnak megfelelő fordítás és  
# szerkesztés végrehajtása
```

Grafikus felületű alkalmazások fejlesztése

Modulok

- A keretrendszer felépítése modularizált, a legfontosabb modulok:
 - központi modul (`QtCore`)
 - grafikus felület (`QtGui`), grafikus vezérlők (`QtWidgets`)
 - adatbázis-kezelés (`QtSQL`)
- A projektben használandó modulokat a projektfájlban kell megadnunk, pl.:
`QT += core gui widgets`
- Egy modul tartalmát osztályonként, illetve egyszerre is betölthetjük az aktuális fájlba (pl. `#include <QtGui>`)

Grafikus felületű alkalmazások fejlesztése

Osztályhierarchia

- A nyelvi könyvtár osztályainak jelentős része teljes származtatási hierarchiában helyezkedik el
 - minden egy őosztály (`QObject`) leszármazottja
 - az őosztály biztosítja az eseménykezelést (`connect`), a tulajdonságkezelést, az időzítést (`timer`), stb.
- Számos segédosztállyal rendelkezik, pl.:
 - adatszerkezetek (`QVector`, `QStack`, `QLinkedList`, ...)
 - fájl és fájlrendszer kezelés (`QFile`, `QTextStream`, `QDir`, ...)
 - párhuzamosság és aszinkron végrehajtás (`QThread`, `QSemaphore`, `QFuture`, ...)

Grafikus felületű alkalmazások fejlesztése

Grafikus felületű alkalmazások vezérlése

- A konzol felületű alkalmazások csak billentyűzettől fogadnak bemenetet a programfutas meghatározott pontjain, a vezérlés módját mi szabályozzuk (pl. főprogram, menü)
- A grafikus felületű alkalmazások billentyűzettől és egértől (érintőképernyőtől, stb.) fogadnak bemenetet a programfutas szinte bármely pillanatában, a vezérlés módja előre definiált
- A grafikus felületű alkalmazás vezérlését az *alkalmazás osztály* (*application class*) látja el
 - kezeli a felhasználói bevitelt, a felület elemeit, beállítja az alkalmazástulajdonságokat (megjelenés, elérési útvonal, ...)
 - a Qt-ben az alkalmazást a **QApplication** típus biztosítja

Grafikus felületű alkalmazások fejlesztése

Eseménykezelés

- A grafikus felületen tehát számos módon és ponton kezdeményezhetünk interakciót a programmal
- A program által kezelhető, lereagálható interakciókat nevezzük *eseményeknek (event/signal)*, az interakció kezdeményezését nevezzük az esemény *kiváltásának*
 - pl.: gombra kattintás, egér húzás, listaelem kiválasztás

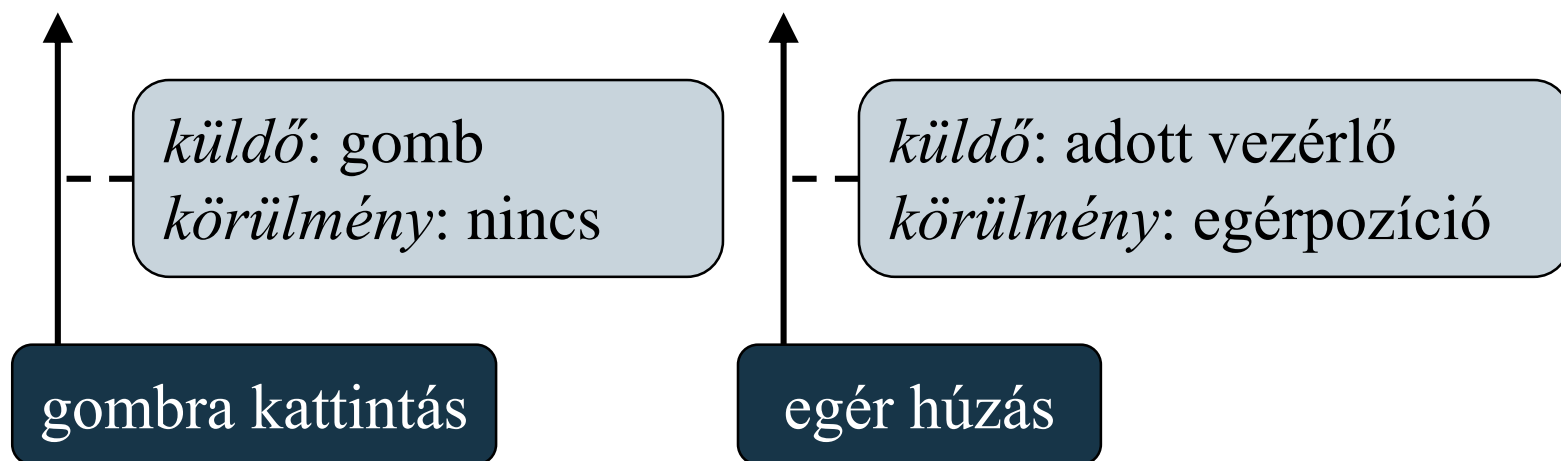
gombra kattintás

egér húzás

Grafikus felületű alkalmazások fejlesztése

Eseménykezelés

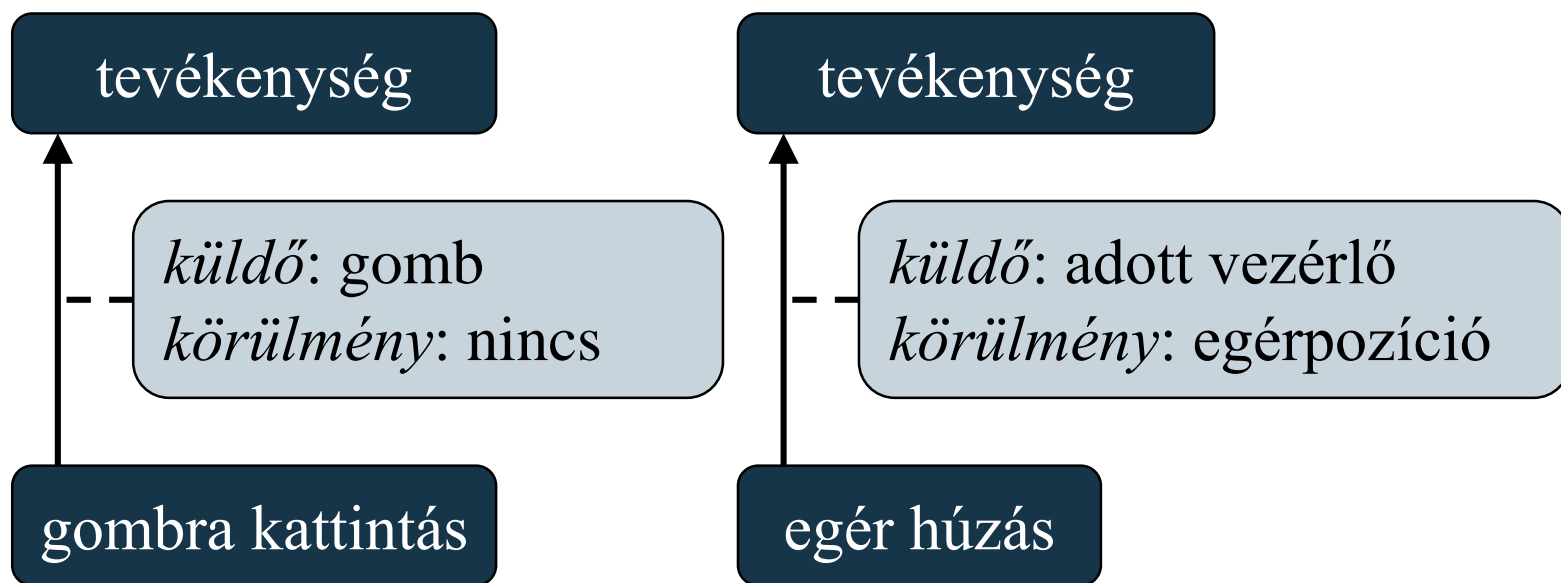
- Az eseménynek van:
 - *küldője (sender)*: kiváltja az eseményt, pl. gomb, lista
 - *körülményei (arguments)*: meghatározza az esemény paramétereit, pl. egér pozíciója a húzáskor, kiválasztott listaelem indexe



Grafikus felületű alkalmazások fejlesztése

Eseménykezelés

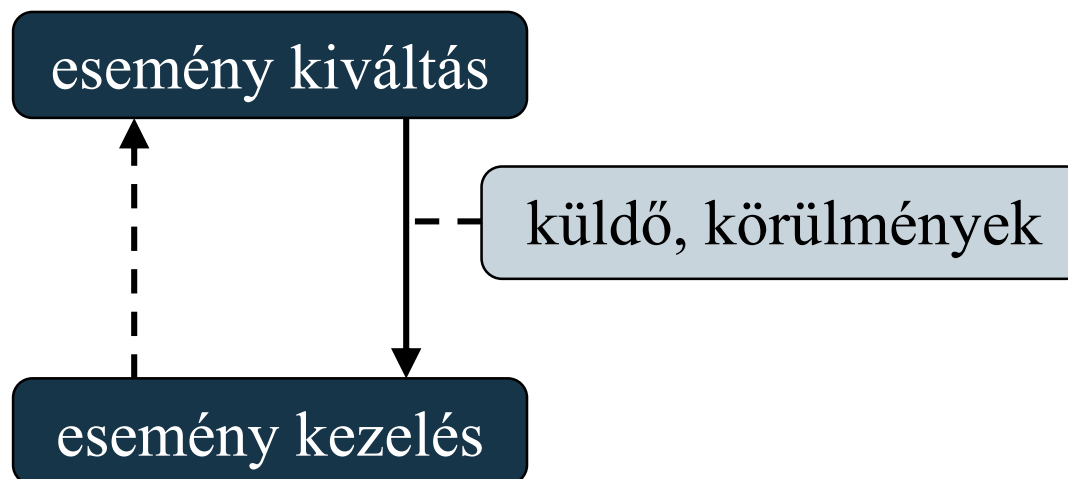
- Az eseményekre reagálva a program futtathat egy alprogramot, ezt nevezzük *eseménykezelőnek (event handler/slot)*
 - ha nem biztosítunk eseménykezelőt az eseményhez, akkor az *lekezeletlen* marad



Grafikus felületű alkalmazások fejlesztése

Eseménykezelés

- Az *összetett események* úgy valósulnak meg, hogy a program egy egyszerű eseményre kivált egy másikat
 - pl.: az egérrel kattintunk, és az egér a gombon van, akkor kiváltódik a gomb kattintása esemény
 - tehát az eseménykezelés egy több lépcsős, *ciklikus folyamat*



Grafikus felületű alkalmazások fejlesztése

Eseménykezelő társítás

- Az eseménykezeléshez összekapcsoljuk az eseményt az eseménykezelővel, ezt *társításnak* nevezzük
 - Qt-ban ehhez a `connect` metódust használjuk, pl.:
`connect (&button, SIGNAL(clicked()),
 &app, SLOT(quit())) ;`
 - megadjuk, mely küldő objektum (*sender*) mely eseményére (**SIGNAL**) mely fogadó objektum (*receiver*) mely eseménykezelője (**SLOT**) kell, hogy fusson
 - mindig mutatókat adunk meg, bármely két alkalmas objektum összeköthető
 - a kötés elvégezhető `QObject` leszármazott típusban, illetve statikus metódus hivatkozással

Grafikus felületű alkalmazások fejlesztése

Eseménykezelő társítás

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

int main(int argc, char *argv[]){
    QApplication app(argc, argv); // alkalmazás
    QPushButton quit("Quit"); // gomb
    quit.resize(75, 30); // méret
    quit.setFont(QFont("Times", 18, QFont::Bold));
        // betűtípus
    QObject::connect(&quit, SIGNAL(clicked()),
                    &app, SLOT(quit()));
    quit.show(); // gomb megjelenítése
    return app.exec();
}
```

Grafikus felületű alkalmazások fejlesztése

A grafikus felület

- A grafikus felhasználói felület ablakokból tevődik össze, amelyeken vezérlőket helyezünk el
 - a vezérlők objektumorientáltan valósulnak meg, öröklődés segítségével szerveződnek hierarchiába
 - minden vezérlő őssosztálya a **QWidget**, amelynek van egy további őssosztálya, a **QObject**
- A **QObject** azon típusok őse, amely kihasználja a Qt speciális vonásait, úgymint *események* és *eseménykezelők*, *tulajdonságok*, *időzítés*
 - a **QObject** példányok nem másolhatóak, ezért jórész mutatók és referenciák segítségével kezeljük őket

Grafikus felületű alkalmazások fejlesztése

Vezérlők

- A vezérlők számos *tulajdonsággal* rendelkeznek
 - tulajdonságnak nevezzük a vezérlők azon külsőleg elérhető értékeit (mezőit), amelyeket *lekérdező* (*getter*), illetve *beállító* (*setter*) műveletek segítségével szabályozhatunk
 - a lekérdező művelet neve a tulajdonság neve, a beállító művelet tartalmaz egy **set** előtagot
 - pl.:

```
QLabel myLabel; // címke létrehozása
myLabel.setText("Hello World!");
    // beállítjuk a címke szövegét (text)
QString text = myLabel.text();
    // lekérdezzük a címke szövegét
```

Grafikus felületű alkalmazások fejlesztése

Vezérlők

- A vezérlők fontosabb tulajdonságai:
 - méret (**size**), vagy geometria (elhelyezkedés és méret, **geometry**)
 - szöveg (**text**), betűtípus (**font**), stílus (**styleSheet**), színpaletta (**palette**), előugró szöveg (**toolTip**)
fókuszáltság (**focus**), láthatóság (**visible**)
 - engedélyezés (használható-e a vezérlő, **enabled**)
- A vezérlőkön (pl. **QLabel**, **QLineEdit**) elhelyezett szöveg formázható több módon
 - pl. formátummal (**textFormat**), illetve használhatóak HTML formázó utasítások is

Grafikus felületű alkalmazások fejlesztése

Vezérlők

```
#include <QPushButton>
```

```
...
```

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

```
...
```

```
    QPushButton myButton; // gomb
```

```
    myButton.resize(75, 30); // méret
```

```
    myButton.setFont(QFont("Times", 20)); // betűtípus
```

```
    myButton.setText("<h1>My Button<h1><br>This is  
        my button!"); // formázott szöveg
```

```
    myButton.setToolTip("You can try klicking on  
        it..."); // előugró szöveg
```

```
    myButton.show(); // gomb megjelenítése ablakként
```

```
...
```


Grafikus felületű alkalmazások fejlesztése

Vezérlők

- A leggyakrabban használt vezérlők:
 - címke (`QLabel`)
 - LCD kijelző (`QLCDNumber`), folyamatjelző (`QProgressBar`)
 - nyomógomb (`QPushButton`), kijelölő gomb (`QCheckBox`), rádiógomb (`QRadioButton`)
 - szövegmező (`QLineEdit`), szövegszerkesztő (`QTextEdit`)
 - legördülő mező (`QComboBox`)
 - dátumszerkesztő (`QDateEdit`), időszerkesztő (`QTimeEdit`)
 - csoportosító (`QGroupBox`), elrendező (`QLayout`)
 - menü (`QMenu`), eszköztár (`QToolBox`)

Grafikus felületű alkalmazások fejlesztése

Vezérlők hierarchiája

- A grafikus vezérlők között *hierarchiát* állíthatunk fel, amely egy fának megfelelő struktúrával reprezentálható
 - a vezérlőnek lehet *szülője* (**parent**), amelyen belül található
 - a vezérlőnek lehetnek *gyerekei* (**children**), azon vezérlők, amelyek rajta helyezkednek el
 - amennyiben egy vezérlőt megjelenítünk (**show ()**), az összes gyerek vezérlője is megjelenik
 - ha egy szülő vezérlőt elrejtünk/megjelenítünk, kikapcsolunk/bekapcsolunk, vagy megsemmisítünk, akkor a gyerekein is megtörténik a tevékenység

Grafikus felületű alkalmazások fejlesztése

Ablakok

- A grafikus felületű alkalmazásokban a vezérlőket *ablakokra* helyezzük, amely a vezérlő szülője lesz
 - ablaknak minősül bármely vezérlő, amely egy **QWidget**, vagy bármely leszármazottjának példánya, és nincs szülője
 - vezérlő szülőjét konstruktor paraméterben, vagy a **parent** tulajdonságon keresztül adhatjuk meg
 - pl.:

```
QWidget parentWidget; // ablak
QPushButton childButton(&parentWidget); // gomb
...
parentWidget.show();
    // a gomb is megjelenik az ablakkal
```

Grafikus felületű alkalmazások fejlesztése

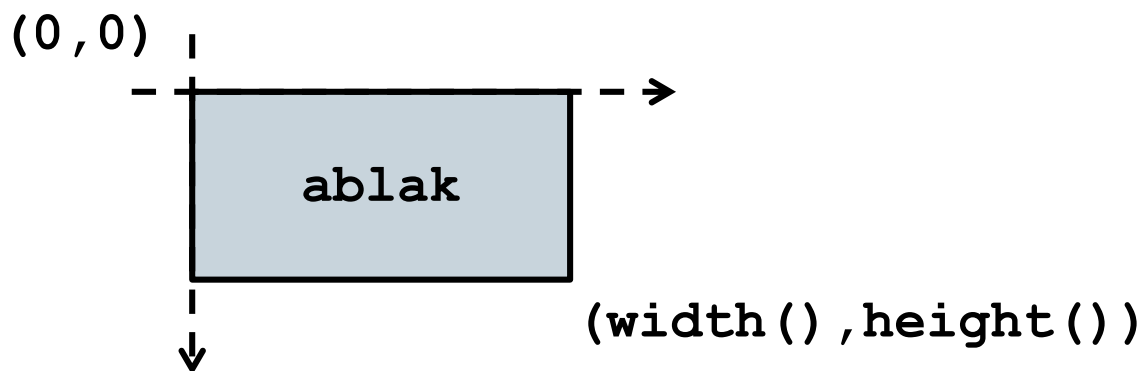
Ablakok

- Az ablakként használt vezérlő további beállításai:
 - cím (`windowTitle`), ikon (`windowIcon`)
 - állítható teljes/normál képernyőre, vagy a tálcára (`showMaximized`, `showNormal`, `showMinimized`)
 - egyszerre mindig csak egy aktív ablak lehet (`isActiveWindow`)
- A vezérlők mérete többféleképpen befolyásolható
 - alapból változtatható méretűek, ekkor külön állítható minimum (`minimumSize`), maximum (`maximumSize`), valamint az alapértelmezett (`baseSize`) méret
 - a méret rögzíthető (`setFixedSize`)

Grafikus felületű alkalmazások fejlesztése

Ablakok

- Amennyiben egy vezérlőt az ablakon helyezünk el, meg kell adnunk a pozícióját és méretét (`setGeometry(int, int, int, int)`)
 - az ablak koordinátarendszere a bal felső sarokban indul a (0,0) koordinátával, és balra, illetve lefelé növekszik



- az ablak területébe nem számoljuk bele az ablak fejlécének területét, amit külön lekérdezhetünk (`frameGeometry`)

Grafikus felületű alkalmazások fejlesztése

Ablakok

...

```
QWidget myWidget; // ablak létrehozása
myWidget.setBaseSize(200, 120); // méretezés
myWidget.setWindowTitle("Demo Window");
    // ablakcímke megadása
```

```
QPushButton quitButton("Quit", &myWidget);
    // gomb az ablakra
quitButton.setGeometry(10, 40, 180, 40);
    // elhelyezés az ablakon
QObject::connect(&quitButton, SIGNAL(clicked()),
                &app, SLOT(quit()));
window.show(); // ablak megjelenítése
...
```

Grafikus felületű alkalmazások fejlesztése

Egyedi ablakok

- Célszerű a saját ablakainknak saját osztályt létrehozni
 - magában az osztályban szerkeszthetjük a tulajdonságait, eseménykezelését, nincs szükségünk a főprogramra

pl.:

```
class MyWindow : public QWidget
{
public:
    MyWindow(QWidget* parent = 0);
    // a konstruktor megkaphatja a szülőt
private:
    QPushButton* quitButton; // gomb az ablakon
};
```

Grafikus felületű alkalmazások fejlesztése

Egyedi ablakok

```
MyWindow::MyWindow(QWidget* parent)
    : QWidget(parent) // ős konstruktor meghívása
{
    setBaseSize(200, 120);
    setWindowTitle("Demo Window");
    quitButton = new QPushButton("Quit", this);
    // gomb az ablakra
    quitButton->setGeometry(10, 40, 180, 40);

    connect(quitButton, SIGNAL(clicked()),
            QApplication::instance(), SLOT(quit()));
    // az eseménykezeléshez lekérdezzük az
    // alkalmazás példányt
}
```


Grafikus felületű alkalmazások fejlesztése

Példa

Feladat: Készítsünk egy egyszerű alkalmazást, amelyben egy csúszkával állíthatunk egy digitális kijelzőn megjelenő számot.

- az alkalmazás számára létrehozunk egy új ablak osztályt (**NumberWidget**), amelyre felhelyezünk egy csúszkát (**QSlider**), valamint egy számkijelzőt (**QLCDNumber**)
- összekötjük a csúszka változást jelző eseményét (**valueChanged(int)**) a kijelző számbeállító eseménykezelőjével (**display(int)**), így egyben paraméterben át is adódik az aktuális érték
- az összekötéseket a konstruktorban megfogalmazhatjuk, így már csak a destruktort kell megvalósítanunk, amely törli a vezérlőket

Grafikus felületű alkalmazások fejlesztése

Példa

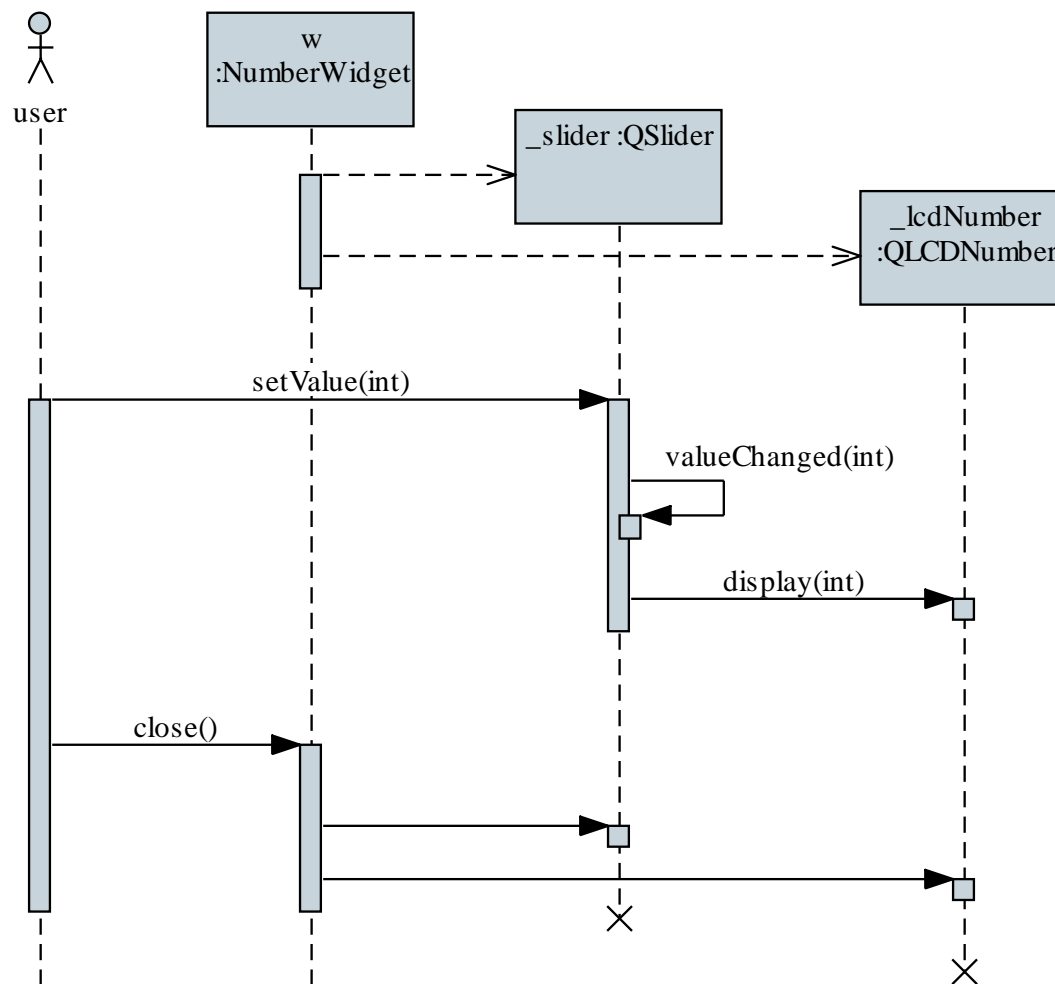
Tervezés:

	<i>QWidget</i>
NumberWidget	
-	<i>_slider :QSlider*</i>
-	<i>_lcdNumber :QLCDNumber*</i>
+	<i>NumberWidget(QWidget*)</i>
+	<i>~NumberWidget()</i>

Grafikus felületű alkalmazások fejlesztése

Példa

Tervezés:



Grafikus felületű alkalmazások fejlesztése

Példa

Megvalósítás (main.cpp):

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

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    NumberWidget w;
    w.show();
    // a főprogram csak példányosítja és
    // megjeleníti az ablakot

    return a.exec();
}
```

Grafikus felületű alkalmazások fejlesztése

Példa

Megvalósítás (numberwidget.cpp):

```
NumberWidget::NumberWidget(QWidget *parent)
    : QWidget(parent) {
    // meghívjuk az ős konstruktorát
    setWindowTitle("Number Display"); // ablakcím
    setFixedSize(300, 175);
    // rögzített méret beállítása
    _slider = new QSlider(this);
    // a vezérlő megkapja szülőnek az ablakot
    ...
    connect(_slider, SIGNAL(valueChanged(int)),
           _lcdNumber, SLOT(display(int)));
    // esemény és eseménykezelő összekötése
    ...
```

Grafikus felületű alkalmazások fejlesztése

Speciális ablakok

- Amellett, hogy ablak bármilyen vezérlő lehet, adottak speciális ablaktípusok, pl.:
 - *üzenőablak* (`QMessageBox`), elsősorban üzenetek közlésére, vagy kérdések feltételére, pl.:

```
QMessageBox::warning(this, "Warning",  
    "This is annoying.\nDo something!");  
// figyelmeztető üzenet
```
 - *dialógusablak* (`QDialog`), amelynek eredménye van, elfogadható (`accept`), vagy elutasítható (`reject`)
 - *főablak* (`QMainWindow`), amely számos kiegészítést biztosít összetett ablakok megvalósítására (menü, állapotsor, beágyazott ablakok kezelése)

Grafikus felületű alkalmazások fejlesztése

Egyedi események és eseménykezelők

- A saját osztályainkban lehetőségünk van egyedi események és eseménykezelők létrehozására, továbbá tetszőleges eseményt kiválthatunk
 - az osztályt el kell látni a `Q_OBJECT` makróval, és a `QObject` osztály leszármazottjának kell lennie
 - eseményeket az `<eseménynév>(<paraméterek>)` utasítással válthatunk ki, pl.: `clicked(false)` ;
 - új eseményeket az osztálydefiníció `signals` részében helyezhetünk el
 - új eseménykezelőket az osztálydefiníció `slots` részében helyezhetünk el, és az eseménykezelőknek adhatunk láthatóságot is

Grafikus felületű alkalmazások fejlesztése

Egyedi események és eseménykezelők

- az események, illetve eseménykezelők eljárások (`void` típussal), tetszőleges paraméterezéssel
- eseményeket csak deklarálnunk kell, az eseménykezelőket definiálni is kell
- Pl.:

```
class MyObject : public QObject {
    Q_OBJECT // az osztályban definiálhatunk
             // eseményt és eseménykezelőt
signals: // saját események
    void mySignal(int param);
public slots: // publikus eseménykezelők
    void mySlot(int param) { ... }
};
```


Grafikus felületű alkalmazások fejlesztése

Események paraméterezése és kiváltása

- Az események paraméterezhetők
 - az esemény paraméterátadását az eseménykezelőnek a társításnál adhatjuk meg, pl.:
`connect(this, SIGNAL(mySignal(int)),
 this, SLOT(mySlot(int)));`
 - a paraméterek átadása sorrendben történik, ezért csak a típust jelezzük
 - az eseménynek legalább annyi paraméterrel kell rendelkeznie, mint az eseménykezelőnek
 - lehetnek alapértelmezett paraméterek is, pl.:
`signals:
 void mySignal(int param = 0);`

Grafikus felületű alkalmazások fejlesztése

Példa

Feladat: Készítsünk egy egyszerű alkalmazást, amelyben egy szavakból álló listát jelenítünk meg, és egy szövegdoboz segítségével szűrhetjük a tartalmat. A szavakat szöveges fájlból töltjük be.

- a saját ablakban (`FilteredListWidget`) felveszünk egy listamegjelenítőt (`QListWidget`) és egy szövegdobozt (`QLineEdit`)
- szükségünk van egy egyedi eseménykezelőre (`filterList`), amely a szűrést elvégzi
- a betöltés az `input.txt` fájlból történik, először egy szöveglistába (`QStringList`), ehhez Qt-s fájlkezelést alkalmazunk (`QFile`, `QStringList`)

Grafikus felületű alkalmazások fejlesztése

Példa

Tervezés:

<i>QWidget</i>
FilteredListWidget
<ul style="list-style-type: none">- <code>_itemStringList :QStringList</code>- <code>_queryLabel :QLabel*</code>- <code>_queryLineEdit :QLineEdit*</code>- <code>_resultListWidget :QListWidget*</code>
<ul style="list-style-type: none">+ <code>FilteredListWidget(QWidget*)</code>+ <code>~FilteredListWidget()</code>- <code>loadItems(QString) :void</code>
«slot»
<ul style="list-style-type: none">- <code>filterList() :void</code>

Grafikus felületű alkalmazások fejlesztése

Példa

Megvalósítás (filteredlistwidget.cpp):

```
class FilteredListWidget : public QWidget {
    Q_OBJECT
    ...
private slots: // eseménykezelők
    void filterList(); // lista szűrése
private:
    ...
    QStringList _itemStringList; // szavak listája
    QLabel *_queryLabel; // címke
    QLineEdit *_queryLineEdit; // sorszerkesztő
    QListWidget *_resultListWidget;
        // listamegjelenítő
};
```

Grafikus felületű alkalmazások fejlesztése

Példa

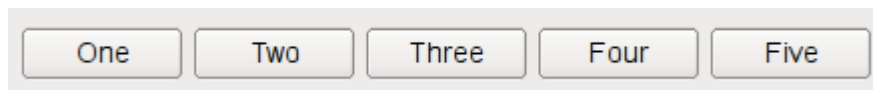
Megvalósítás (`filteredlistwidget.cpp`):

```
void FilteredListWidget::filterList()
{
    ...
    for (int i = 0; i < _itemStringList.size();
        i++)
        if (_itemStringList[i].contains(
            _queryLineEdit->text()))
            // ha tartalmazza a megadott szöveget
            _resultListWidget->addItem(
                itemStringList[i]);
            // akkor felvesszük a listára
    ...
}
```

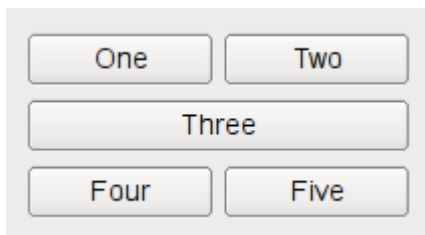
Grafikus felületű alkalmazások fejlesztése

Vezérlők elrendezése

- Mivel az ablak átméretezésével a vezérlők elhelyezkedését módosítani kell, célszerű az átméretezhető ablakoknál *elhelyezéseket (layout)* használni
- Az elhelyezések gyerekvezérlőiket megfelelő sorrendben jelenítik meg, automatikusan áthelyezik és átméretezik, pl.:



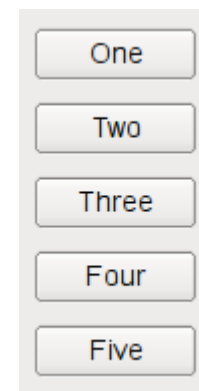
QHBoxLayout



QGridLayout



QFormLayout



QVBoxLayout

Grafikus felületű alkalmazások fejlesztése

Vezérlők elrendezése

- Az elhelyezéseket ráállíthatjuk a vezérlőre (elsősorban az ablakra) a `setLayout(QLayout*)` utasítással
- Számos formának megfelelően helyezhetjük a vezérlőket
 - vízszintes (`QHBoxLayout`), függőleges (`QVBoxLayout`), rács (`QGridLayout`)
 - űrlap (`QFormLayout`), amelyen címkézhetjük a vezérlőket
 - keret (`QBorderLayout`), amely az oldalához, vagy középre tudja igazítani az elemeket
 - dinamikus (`QStackedLayout`), ahol változhat a megjelenő elem
 - az elemek távolsága szabályozható (`spacing`)

Grafikus felületű alkalmazások fejlesztése

Vezérlők elrendezése

- Pl.:

```
QGridLayout* myLayout = new QGridLayout();
myLayout->addWidget(someButton, 0, 0);
    // gomb behelyezése az 1. sor 1. oszlopába
myLayout->addWidget(otherButton, 0, 1, 1, 2);
    // gomb behelyezése a 2. sor 1. oszlopában úgy,
    // hogy két oszlopon is átnyúljon
QFlowLayout* innerLayout = new QFlowLayout();
    // belső, folyamatos elhelyezés
...
myLayout->addLayout(innerLayout);
    // elhelyezés beágyazása
setLayout(myLayout);
    // elhelyezés beágyazása az ablakba
```


Grafikus felületű alkalmazások fejlesztése

Fájldialógus

- Egy speciális dialógusablak a fájldialógus (`QFileDialog`), amely lehetőséget fájlok/könyvtárak kiválasztására
 - statikus műveletekkel közvetlenül használható fájlok megnyitásához (`getOpenFileName`, `getOpenFileNames`), fájlok mentéséhez (`getSaveFileName`) és könyvtárak megnyitásához (`getExistingDirectory`)
- pl.:

```
QString fileName =  
    QFileDialog::getOpenFileName(this,  
    "Open file", "/home", "Text files (*.txt)");  
    // szövegfájl megnyitása a home könyvtárból
```
- ha a felhasználó visszalép, a fájlnev üres lesz (`isNull`)

Grafikus felületű alkalmazások fejlesztése

Példa

Feladat: Módosítsuk az előző alkalmazást úgy, hogy lehessen átméretezni az ablakot, és a tartalom alkalmazkodjon az új mérethez, továbbá lehessen tetszőleges szöveges fájl tartalmát betölteni

- felveszünk egy új gombot, amely a betöltésre szolgál, és hozzá egy új eseménykezelőt (`loadFile`)
- a felhasználó egy fájl kiválasztó dialógusablakban (`QFileDialog`) adhatja meg a fájl nevét
- a felületen felveszünk két elrendezést, egy vízszinteset (`QHBoxLayout`) a felső sornak, és egy függőlegeset a teljes tartalomnak (`QVBoxLayout`)

Grafikus felületű alkalmazások fejlesztése

Példa

Tervezés:

<i>QWidget</i>
FilteredListWidget
<ul style="list-style-type: none">- <code>_itemStringList :QStringList</code>- <code>_queryLabel :QLabel*</code>- <code>_queryLineEdit :QLineEdit*</code>- <code>_resultListWidget :QListWidget*</code>- <code>_loadButton :QPushButton*</code>- <code>_upperLayout :QHBoxLayout*</code>- <code>_mainLayout :QVBoxLayout*</code>
<ul style="list-style-type: none">+ <code>FilteredListWidget(QWidget*)</code>+ <code>~FilteredListWidget()</code>- <code>loadItems(QString) :void</code>
«slot»
<ul style="list-style-type: none">- <code>filterList() :void</code>- <code>loadFile() :void</code>

Grafikus felületű alkalmazások fejlesztése

Példa

Megvalósítás (`filteredlistwidget.cpp`):

```
FilteredListWidget::FilteredListWidget(QWidget
    *parent) : QWidget(parent) {
    ...
    _mainLayout = new QVBoxLayout;
    _mainLayout->addLayout(_upperLayout);
    // másik elrendezés felvétele
    _mainLayout->addWidget(_resultListWidget);
    _mainLayout->addWidget(_loadButton);

    setLayout(_mainLayout);
    // elrendezés beállítása
    ...
}
```

Grafikus felületű alkalmazások fejlesztése

Példa

Megvalósítás (`filteredlistwidget.cpp`):

```
void FilteredListWidget::loadFile() {
    QString fileName =
        QFileDialog::getOpenFileName(this,
            trUtf8("Fájl megnyitása"), "",
            trUtf8("Szöveg fájlok (*.txt)"));
    // fájl megnyitó dialógus használata,
    // megadjuk a címét és a szűrési
    // feltételt
    if (!fileName.isNull())
        // ha megadtunk valamilyen fájlnevet és
        // OK-val zártuk le az ablakot
        loadItems(fileName);
}
```

Grafikus felületű alkalmazások fejlesztése

A felülettervező

- A *felülettervező* (*Qt Designer*) lehetőséget ad a felület gyors elkészítésére
 - az elkészített terv XML-ben mentődik (`<ablaknév>.ui`), majd abból Qt osztály készül (`moc_<ablaknév>.h`)
 - a generált osztály az tervezőben adott név (`name`) tulajdonságot kapja névként, valamint az `Ui_` előtagot (ehelyett használhatjuk az `Ui` névteret)
 - a vezérlőkre a megfelelő névvel hivatkozhatunk, a kialakítás a generált osztály `setupUi (QWidget* parent)` metódusába kerül
 - az így generált osztályt a saját osztályokban attribútumként használjuk fel, és hivatkozunk rajta keresztül a vezérlőkre

Grafikus felületű alkalmazások fejlesztése

A felülettervező

```
#include "ui_demowindow.h" // tervező által generált
...
class MyWindow : public QWidget {
    Q_OBJECT
public:
    MyWindow(...) : ..., ui(new Ui::MyWindow)
    {
        ui->setupUi(this);
        // innentől használhatóak a vezérlők
        // pl. ui->quitButton->...
    }
    ...
private:
    Ui::MyWindow* ui;
};
```

Grafikus felületű alkalmazások fejlesztése

Példa

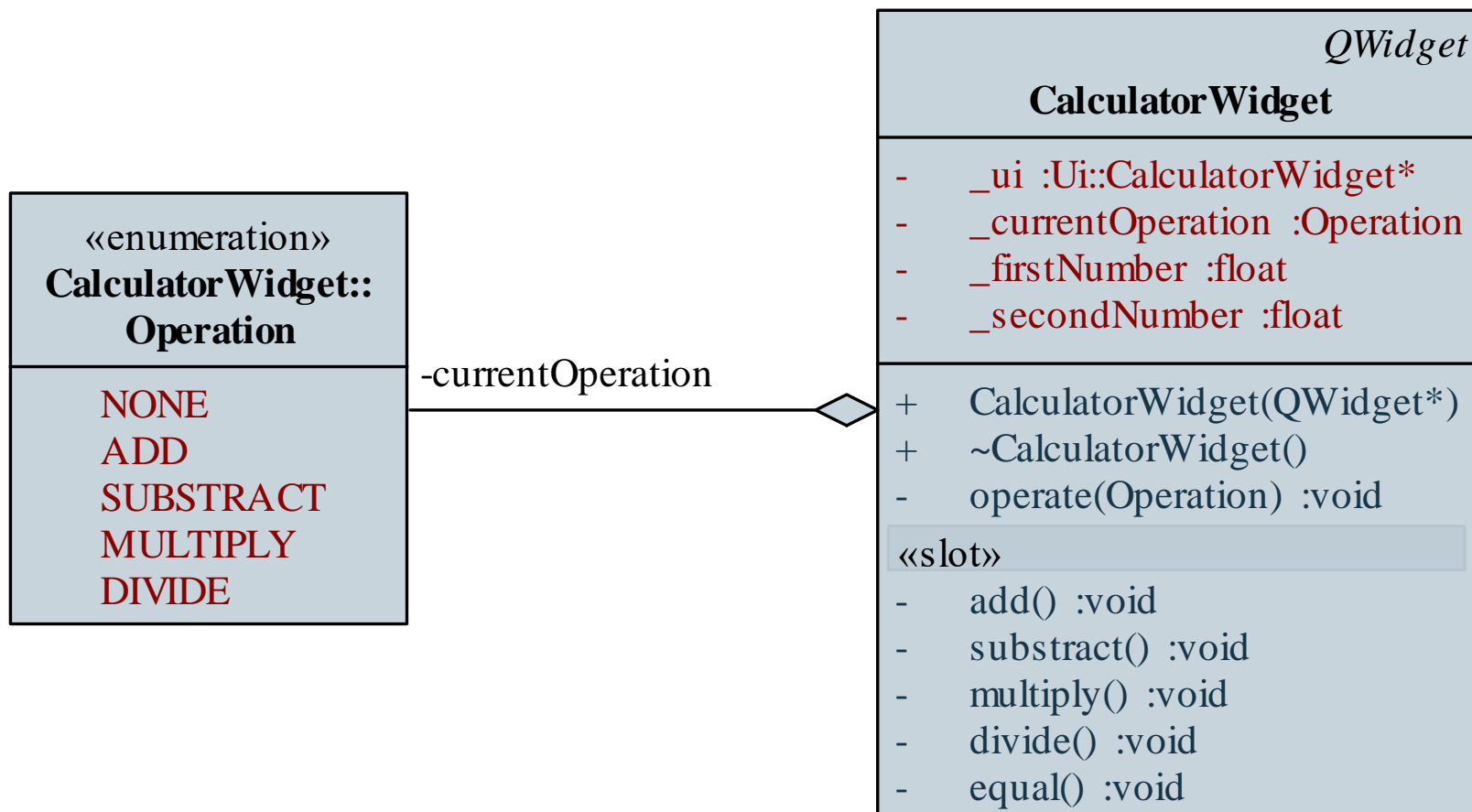
Feladat: Készítsünk egy egyszerű számológépet, amellyel a négy alapműveletet végezhethetjük el, egy beviteli mezővel, amely az előző művelet eredményét jeleníti meg.

- az alkalmazás felületét a felülettervezővel készítjük el, elhelyezünk 5 gombot, valamint egy szövegbeviteli mezőt használunk
- az ablak osztályban (`CalculatorWidget`) létrehozunk öt eseménykezelőt a gombokra, amelyek a megfelelő műveleteket végzik el
- ügyelnünk kell arra, hogy mindig az előző műveletet végezzük el, ne az aktuálisan megadottat, ezért az előző műveletet, illetve az értéket mindig eltároljuk
- a szövegmezőbe csak számok bevitelét tesszük lehetővé

Grafikus felületű alkalmazások fejlesztése

Példa

Tervezés:



Grafikus felületű alkalmazások fejlesztése

Példa

Megvalósítás (calculatorwidget.cpp):

```
CalculatorWidget::CalculatorWidget(QWidget
    *parent) : QWidget(parent) ,
    _ui(new Ui::CalculatorWidget)
{
    // grafikus felület létrehozása
    _ui->setupUi(this);
    // grafikus felület összeállítása
    setFixedSize(172,250); // méret rögzítése
    ...
    _ui->numberLineEdit->setFocus();
    // a szövegmezőre állítjuk a fókuszt
    _ui->numberLineEdit->selectAll();
    // az összes szöveg kijelölése
}
```