

# Eseményvezérelt alkalmazások fejlesztése

Előadók: Cserép Máté András

Gregorics Tibor

# Tantárgyról

- ❑ **Cél:** Objektum orientált módon felépített grafikus felületű, egy- , illetve többablakos, több rétegű, eseményvezérelt alkalmazások készítése
- ❑ **Eszközök:** Qt, C++, Linux,  
.NET, C#, Windows,  
UML
- ❑ **Előfeltétel:** Objektumelvű programozás tantárgy
- ❑ **Előismeretek:**
  - Objektumelvű tervezés (UML) és programozás,
  - C++
- ❑ **Előadás** (heti 2 óra)
- ❑ **Gyakorlat** (heti 2 óra) + **konzultáció** (heti 1 óra)

# Összevont számonkérés

- ❑ Kötelező **házi feladatok** egyenként 5 pontért:
  - 4 darab beadandó elkészítése:  
helyes (tesztelt) program + dokumentáció
  - határidő: legfeljebb 3 hét késés, hetenként 1 pont levonás,
  - személyesen kell bemutatni
  - plágium ellenőrzés
- ❑ **Géptermi zárthelyik** egyenként 5 pontért:
  - 2 darab zh (7. héten és a vizsgaidőszakban)
  - mindkettő legalább megfelelt szintű, az egyik javítható
  - használható az előadás honlapja, bármely írott/nyomtatott anyag, és a félév során leadott beadandók
- ❑ **Gyakorlati jegy:**
  - a beadandók, és a duplán számított zárthelyik eredményének átlaga

# Háttér anyagok

❑ <http://people.inf.elte.hu/gt/eva>

- előadások diái
- házi feladatok (követelmények leírása, feladatsor, minta)
- minta megoldások
- korábbi zárthelyik

❑ Giachetta Roberto anyagai:

[http://people.inf.elte.hu/groberto/elte\\_eva1/](http://people.inf.elte.hu/groberto/elte_eva1/)

[http://people.inf.elte.hu/groberto/elte\\_eva2/](http://people.inf.elte.hu/groberto/elte_eva2/)

# Qt keretrendszer

Néhány szó a grafikus felületről,  
a Qt programok fordításáról,  
és bevezetés az eseménykezelésbe



Code less.  
Create more.  
Deploy everywhere.

---

A Qt egy platformfüggetlen alkalmazás-fejlesztési keretrendszer, amellyel nemcsak asztali, de akár mobil vagy beágyazott alkalmazások is fejleszthetők.

- Támogatást ad a grafikus felület, az adatbázis-kezelés, a multimédia, a 3D grafika, a hálózati és webes kommunikáció készítéséhez.
- Rendelkezik nyílt forráskódú (LGPL) és kereskedelmi verzióval is.
- Fejlesztő nyelve elsősorban a C++, de más nyelvekre is elérhető, valamint rendelkezik saját leíró nyelvvel (Qt Quick).
- A Qt 5.x keretrendszer, QtCreator fejlesztőkörnyezet elérhető a <https://www.qt.io/> oldalról.
  - `apt-get install qt-sdk`

# Segédanyag

---

- ❑ Qt : <https://doc.qt.io>
- ❑ Jasmin Blanchette, Mark Summerfield: C++ GUI programming with Qt4, Prentice Hall, 2006. ISBN 0-13-187249-4
- Lee Zhi Eng: Hands-On GUI Programming with C++ and Qt5 – Build stunning cross-platform applications and widgets with the most powerful GUI framework, Packt Publishing, 2016. ISBN 978-1-78646-712-6
- Guillaume Lazar, Robin Penea: Mastering Qt 5 – Master application development by writing succinct, robust, and reusable code with Qt 5 Packt Publishing, 2018. ISBN 978-1-78839-782-7

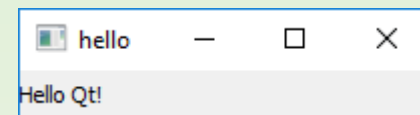
# Fejlesztés nyelve és környezete

---

- ❑ A fejlesztés C++/Qt nyelven, Linux környezetben történik.
  - Elérhető a teljes C++ utasításkészlet, nyelvi könyvtár.
  - Számos C++ nyelven megszokott osztálynak létezik Qt-s megfelelője (QString, QQueue, stb.)
  - A standard C++ nyelv kiterjesztését Qt-s makrók biztosítják, amelyeket a *Meta Object Compiler (MOC)* fordít le ISO C++ kódra.
- ❑ Az alapértelmezett fejlesztőeszköz a *QtCreator*, de más környezetekben is lehetőség van a Qt fejlesztésre (pl. *Code::Blocks*, *Visual Studio*).
- ❑ Külön tervezőprogram (*QtDesigner*) áll rendelkezésre a grafikus felület létrehozásához, amely XML nyelven írja le a felület felépítését.



# "Hello Qt" alkalmazás



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

int main(int argc, char *argv[]) {
    QApplication app(argc, argv); // alkalmazás példányosítása
    QLabel myLabel("Hello Qt!"); // címke a felirattal
    myLabel.show();               // címke megjelenítése

    return app.exec();            // alkalmazás futtatása
}
```

# Grafikus felületű alkalmazás

---

- ❑ **Grafikus felületű alkalmazás**nak nevezzük azt a programot, amely 2D-s interaktív felhasználói felületen (*GUI, Graphical User Interface*) keresztül kommunikál a felhasználóval
  - A konzol felületnél lényegesen változatosabb interakciós lehetőséget biztosít a programfutásba történő beavatkozáshoz.
  - A felület grafikus megjelenítéssel is ellátott **vezérlő**kből (*control/widget*) áll (mint például nyomógombok, listák, menük, stb.).
  - Egy grafikus vezérlő megjelenhet önállóan – **ablak**ként (*form/window*) –, vagy egy másik vezérlő részeként. Több ablak esetén mindig van egy aktív ablak, és egy aktív vezérlő (ezen van a **fókusz**).
  - A működését a felhasználói interakcióra történő várakozás jellemzi.

# Példák GUI alkalmazásokra

Diagram illustrating GUI components for two applications: "Épületek szűrése" (Building Search) and "Tic-Tac-Toe".

**Épületek szűrése (Building Search):**

- vezérlő (szövegmező):** Text input field for "Név:" (Name).
- vezérlő (címke):** Label for "Város:" (City).
- vezérlő (számmező):** Spin box for "Maximum ár:" (Maximum price).
- vezérlő (számmező):** Spin box for "Minimum ár:" (Minimum price).
- fókuszált vezérlő (nyomógomb):** "Szűrés" (Filter) button.
- vezérlő (táblanézet):** Table displaying search results.

	Név	Város	Utca	Állapot	Tenger távolság	Tengerpart
1	Bella Rosa	Ca' di Valle	corso Italia	nem f...	közvetlen	sziklás
2	EuroApartments	Ca' di Valle	via Fausta	foglal...	80 m	kavicsos
3	Villini Laurin	Cavallino	via Europa	felújit...	100 m	kavicsos

**Tic-Tac-Toe:**

- ablakcím:** Window title "Tic-Tac-Toe".
- ablakfelület (rajzolva):** The game board area.

Épület	Szobaszám	Emelet	Ajtó	Ágyak száma	Kivehető
1 Bella Rosa	201	0	1	2	<input type="checkbox"/>
2 Bella Rosa	202	0	2	2	<input type="checkbox"/>

**vezérlőben lévő vezérlő (kijelölőmező):** Checkboxes in the table.

# Grafikus felület felépítése

---

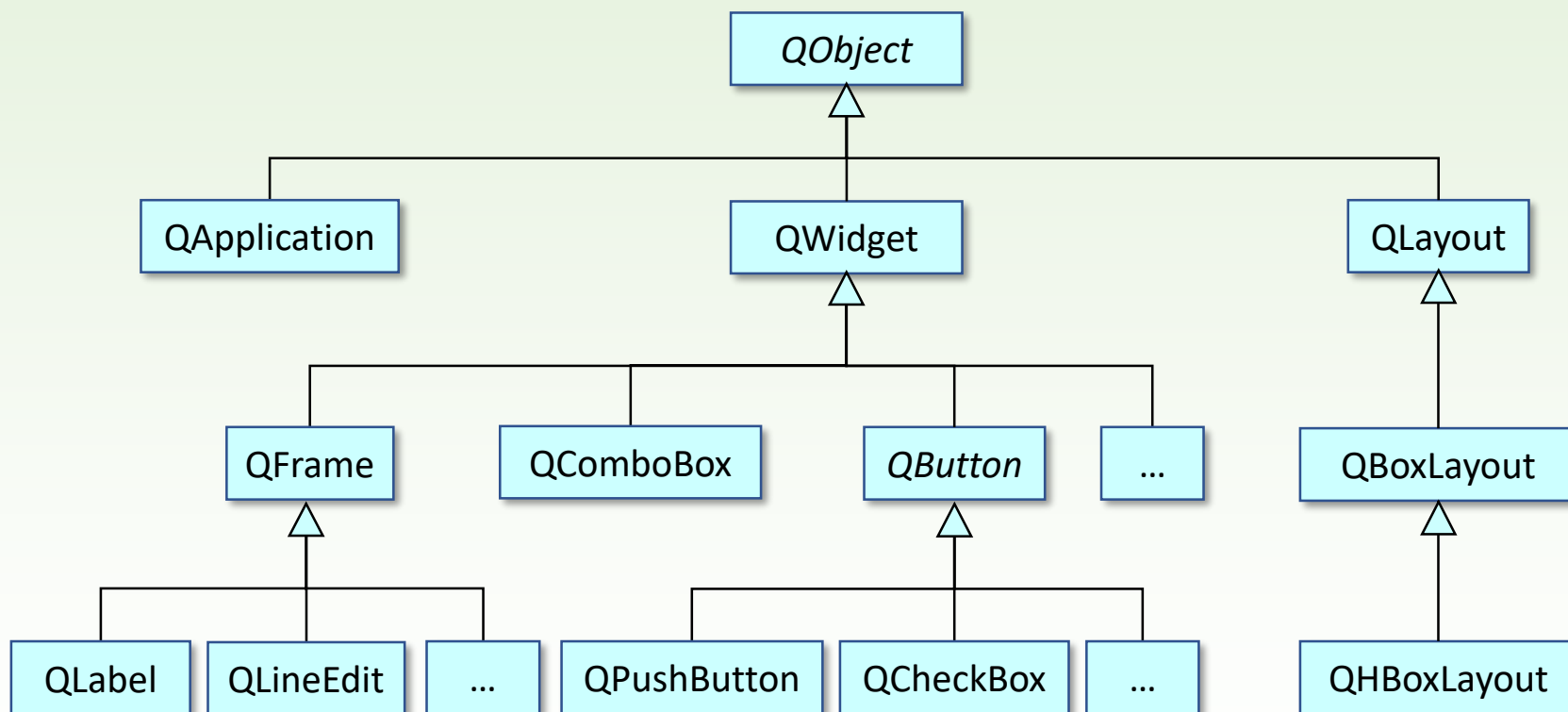
- ❑ A grafikus felület *objektumorientáltan* épül fel.
  - minden vezérlő egy **objektum**, amelyet az osztálya jellemez.
  - Mivel a különféle vezérlők sok hasonló tulajdonsággal bírnak, így osztályaik **öröklődési hierarchiába** szervezhetők.
  - A Qt keretrendszer nyelvi könyvtára egy általános vezérlőt és az abból származtatott nevezetes vezérlőket tartalmazza, de ha egyedi vezérlőre van szükségünk, akkor annak osztályát nekünk kell származtatással definiálni.
  - A vezérlők egy adott alkalmazásban **elhelyezkedési hierarchiába** rendeződnek aszerint, hogy egy vezérlő önállóan, azaz *ablakként*, vagy egy másik vezérlő részeként jelenik meg.

# Qt osztálykönyvtár

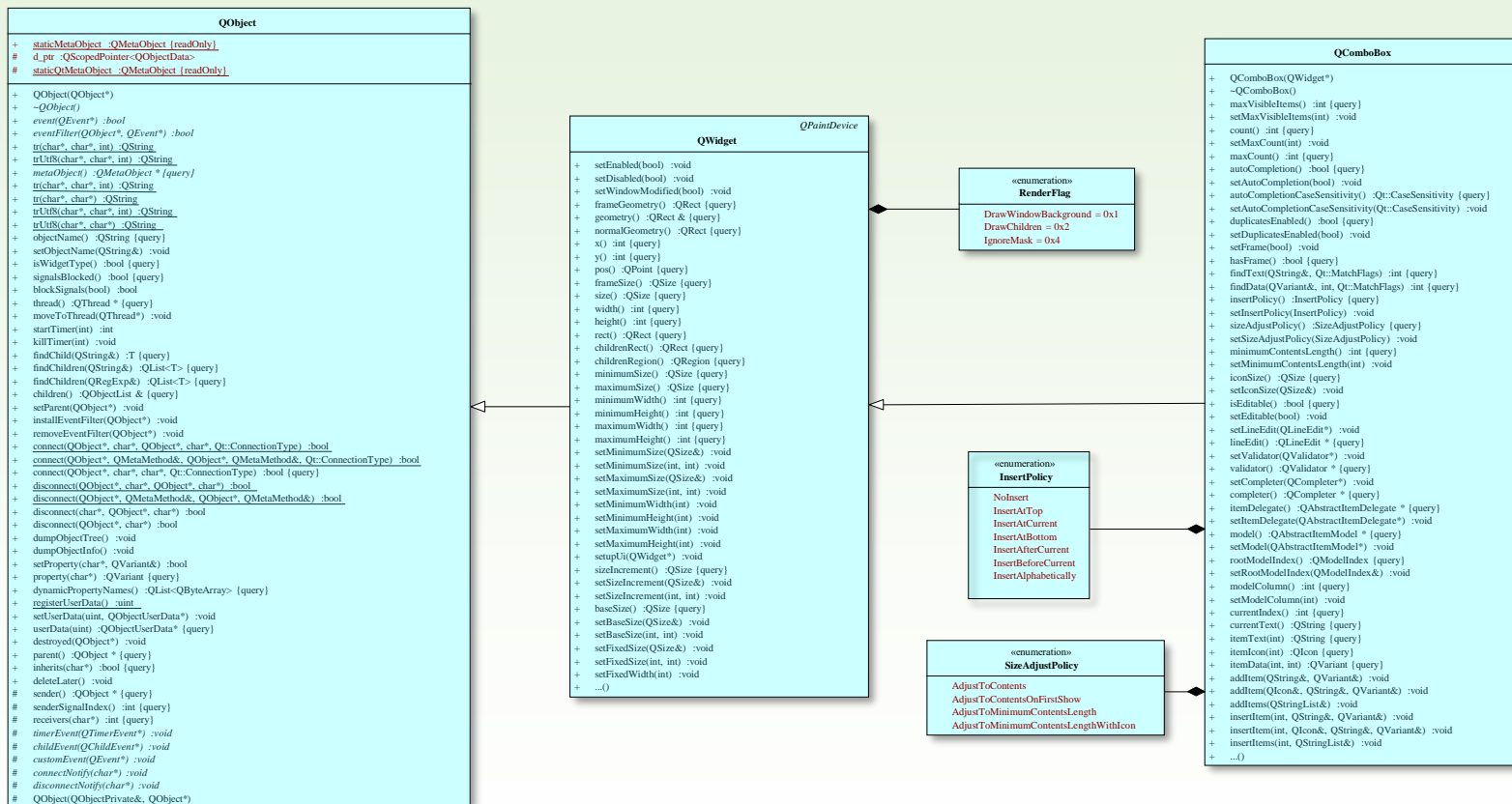
---

- ❑ A Qt nyelvi könyvtár osztályai mind a `QObject` őssztály leszármazottjai.
- ❑ Számos segédosztállyal rendelkezik, pl.:
  - adatszerkezetek (`QVector`, `QStack`, `QLinkedList`, ...)
  - fájl és fájlrendszer kezelés (`QFile`, `QTextStream`, `QDir`, ...)
  - párhuzamosság és aszinkron végrehajtás (`QThread`, `QSemaphore`, `QFuture`, ...)
- ❑ Az osztályok jelentős részét teszik ki a vezérlők osztályai, amelyek mind a `QObject` osztályból származnak.
- ❑ A vezérlők egy csoportja grafikus megjelenéssel is rendelkezik. Ezek közös őse a `QWidget` osztály.

# Vezérlők osztályhierarchiája



# Vezérlők osztályai



# Fordítás

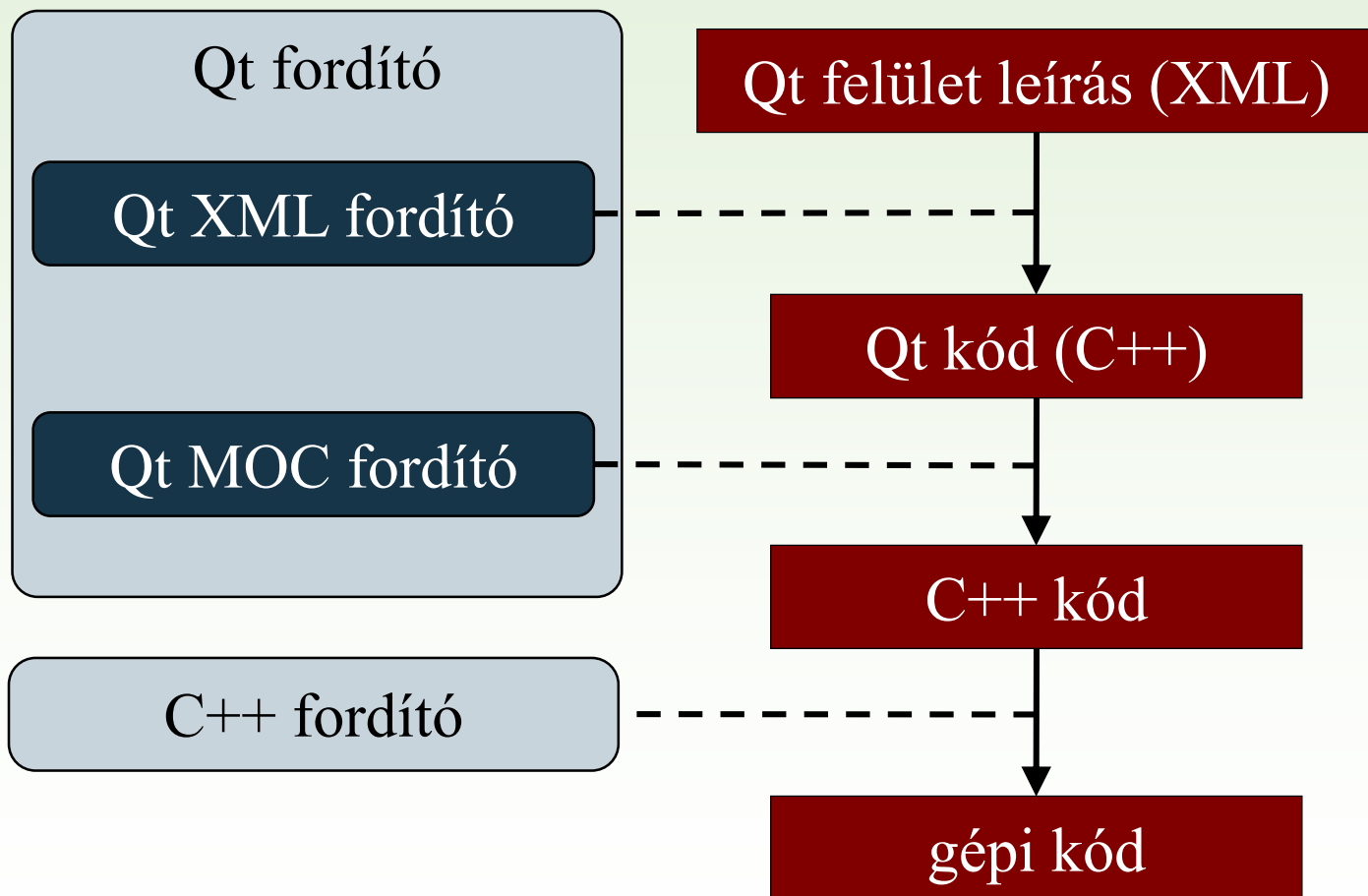
---

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

```
qmake -project # könyvtár nevével azonos projektfájl létrehozása
qmake          # fordítófájl (makefile) előállítás a projektfájlból
make          # projekt fájlnak megfelelő fordítás és
               # szerkesztés végrehajtása
```



# Fordítás lépései



# Modulok

---

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

# Vezérlés

---

- ❑ Egy eseményvezérelt alkalmazás vezérlését egy **alkalmazás objektum** (a Qt-ben a **QApplication** osztály példánya) látja el.
  - Beállítja az alkalmazás szintű tulajdonságokat (megjelenés, elérési útvonal).
  - Felügyeli az alkalmazás vezérlő objektumait, azokon belül a grafikus felület elemeit.
  - A programfutás során bekövetkező különféle történéseket, akciókat (pl. billentyűzet-, vagy egérhasználat, egy objektum üzenete, stb.) eseményként értelmezi.
  - Gondoskodik arról, hogy az események eljussanak azokhoz a vezérlőkhöz, amelyek megfelelően reagálhatnak az eseményre.

# Akció – Esemény – Tevékenység

---

- ❑ Az **akció** az adott alkalmazás aktuális tevékenységétől függetlenül bekövetkező történés, amelyet kezdeményezhet
  - a felhasználó (billentyűzet, egér, érintőképernyő, stb.),
  - az alkalmazás egyik objektuma (pl. időzítő tikkélése),
  - egy másik alkalmazás
- ❑ Az **esemény** (*event*) az alkalmazás által érzékelt akció, amely objektumként jelenik meg az alkalmazásban.
  - a billentyű leütés akciójából a lenyomás és a felengedés eseménye lesz
  - az egér jobb fülével történő kattintás akcióból egy „egérgomb-lenyomás” és egy „egérgomb-felengedés” eseményt generál
- ❑ Az eseményre reagálhat az alkalmazás: az esemény akár több olyan vezérlő objektumhoz is eljuthat, amelynek egy **tevékenysége** az esemény hatására hajtódik végre.
  - az enter billentyű lenyomása a fókuszbán levő nyomógombnál ugyanazt a tevékenységet váltja ki, mint a gombon történő egér-kattintás

# Az eseménykezelés módjai

---

- ❑ **Származtatás:** egy létező vezérlő osztályból származtatunk egy új osztályt, és abban felüldefiniáljuk az eseménykezelő metódusokat.
  - Qt-ben például a **keyPressEvent()** metódust lehet (ha kell) ilyen módon felülimplementálni.
- ❑ **Függőség befecskendezés:** átadunk a vezérlőnek egy olyan megfigyelő objektumot, amelyiknek adott metódusa végzi az esemény kezelését.
  - Qt-ben például egy vezérlőhöz az **installEventFilter()** metódussal rendelhetünk futási időben egy ún. *monitoring* objektumot, amelynek **eventFilter()** metódusa kezeli a vezérlő eseményeit.
- ❑ **Signal-Slot:** az eseményvizsgáló metódusok az esemény hatására egy szignált (*signal*) váltanak ki (*emit*), amelyre futási időben iratkozhatnak fel az eseményt kezelő metódusok, az úgynevezett **slot**-ok.
  - Qt-ben ezzel a technikával fogunk legtöbbször találkozni (futási idejű, de nem objektum-orientált technológia). Ennek során egy speciális **connect()** függvény rendel egy szignálhoz egy eseménykezelő metódust, de később ez a kapcsolat meg is szüntethető **disconnect()**.

# Akció – Esemény – Szignál



# Események és szignálok

---

- ❑ Az események a **QEvent** osztályból származtatott objektumok.
  - A több, mint száz különféle **esemény típust** *enum* értékek azonosítják, amelyeket a **QEvent::type()** ad meg.
    - Pl: **QEvent::KeyPress**, **QEvent::MouseButtonPress**
  - Egy eseménynek lehetnek **paraméterei** (*arguments*).
    - Pl. egérfül lenyomásakor: melyik fület nyomták le, mi az egérmutató pozíciója.
- ❑ Egy szignálra úgy gondoljunk, mint egy objektum **törzs nélküli void-os függvényére**, amelyet ezért nem „meghívunk”, hanem „kiváltunk”.
- ❑ Ne keverjük össze az esemény és a szignál fogalmait.
  - Amikor felhasználunk egy már definiált vezérlőt, akkor annak a szignáljaival kell foglalkoznunk: megmondjuk, hogy melyik szignálhoz milyen eseménykezelőt akarunk társítani.
  - Amikor új vezérlőt tervezünk és implementálunk, akkor az ahhoz érkező eseményekre koncentrálunk, és döntünk arról, hogy melyeket kell (közvetlenül vagy szignál kiváltásával) lekezelni.

# Eseménykezelő (slot)

---

- ❑ Egy szignálhoz eseménykezelő tevékenységet köthetünk. Ez egy olyan metódus, amelynek a társítás pillanatában nem kell létezni.
  - Ha van a szignálhoz társítva eseménykezelő tevékenység, akkor majd ez hajtódik végre a szignál kiváltásakor.
  - Ha nincs, akkor a szignál kiváltása üres programként viselkedik: nem okoz hibát, a szignált kiváltó esemény **kezeletlen** marad.
- ❑ Egy szignálnak lehetnek **aktuális paraméterei**, amelyet a szignálhoz társított eseménykezelő formális paraméterváltozói vesznek át. Az eseménykezelő formális paraméterlistájának illeszkedni kell az aktuális paraméterekhez (sorrendben, típusban, de darabszámban nem).
- ❑ Egy szignálhoz több tevékenység is társítható, több különböző szignálhoz ugyanaz a tevékenység is, sőt egy szignálhoz egy mási szignál is.



# Eseménykezelő társítása szignálhoz

---

- ❑ A társítást végző **connect** bármelyik **QObject** leszármazott típus metódusaként jelen van, de statikus metódus hivatkozással is elérhető.

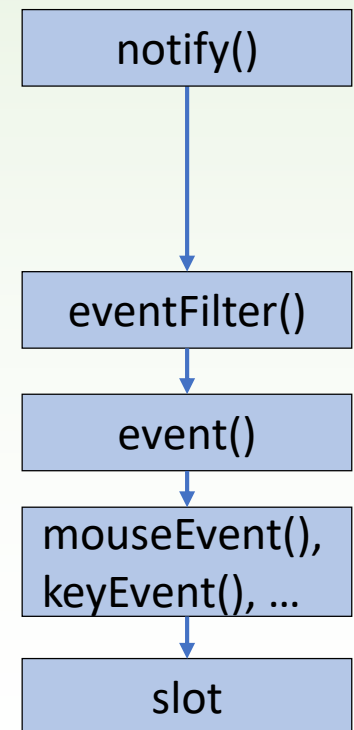
- Ennek paraméterei:
  - a szignált kiváltó küldő objektum (*sender*)
  - a szignált (**SIGNAL**)
  - a fogadó objektum (*receiver*)
  - a fogadó metódusaként definiált eseménykezelő (**SLOT**).
- Pl.:

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

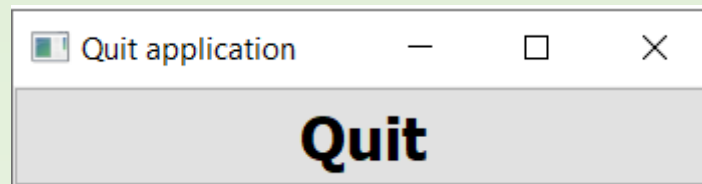
- ❑ Egy szignálhoz rendelt eseménykezelő metódusban mindig lekérdezhető a szignált kiváltó vezérlő objektum: a szignál **küldője** (*sender*).

# Eseménykezelés szintjei

- ❑ Egy esemény feldolgozását az alkalmazás objektum irányítása mellett elosztott módon végzik a megfelelő sorrendben hívott **eseménykezelő metódusok**.
  - Az alkalmazás objektum először a **notify()** metódusát (amely felüldefiniálható) futtatja. Ez gondoskodik arról, hogy az esemény a megfelelő (fókuszban levő vagy egér mutató alatti) vezérlőhöz eljusson.
  - Egy vezérlőhöz opcionálisan hozzárendelt figyelő (*monitoring*) objektum **eventFilter()** metódusa végezhet előzetes eseményfeldolgozást. Több vezérlőnek lehet közös figyelő objektuma is, sőt maga az alkalmazás objektum is ilyen.
  - Egy vezérlő a hozzá eljutó eseményt az **event()** metódusával (amely felüldefiniálható) dolgozza fel, amelyre az esemény fajtájától függően lefut még a **mousePressEvent()**, vagy **keyPressEvent()**, vagy **paintEvent()** eseménykezelő is.
  - Ha ezek valamelyike szignált is kivált, akkor a szignálhoz rendelt tevékenységek (slot) végrehajtására kerül sor.



# "Quit" alkalmazás



```
#include <QApplication>
#include <QPushButton>
int main(int argc, char *argv[]) {
    QApplication app(argc, argv); // alkalmazás

    QPushButton quit;           // nyomógomb létrehozása
    quit.setText("Quit");        // ez konstruktorban is beállítható
    quit.resize(350, 50);       // méret beállítása
    quit.setFont(QFont("Times", 18, QFont::Bold)); // betűtípus
    quit.setWindowTitle("Quit application"); // ablak címe
    quit.setToolTip("You can exit"); // előugró szöveg

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

    quit.show();                // nyomógomb megjelenítése

    return app.exec();          // alkalmazás indítása
}
```

# Felület tervező

---


A *felülettervező* (*Qt Designer*) lehetőséget ad a felület gyors elkészítésére

- ❑ Az elkészített felületterv XML-ben mentődik (`<ablaknév>.ui`), majd abból egy Qt osztály készül (`ui_<ablaknév>.h`)
- ❑ Az így generált osztály funkcionalitását egy saját, `QWidget`-ből származó osztály részeként használjuk:
  - vagy őse lesz a saját osztálynak a generált osztály is,
  - vagy komponensként tartalmazza a saját osztály a generált osztály egy példányát, pontosabban egy adattagja hivatkozik arra.
- ❑ A generált osztály a tervezőben adott név (**name**) tulajdonságot kapja névként, valamint az `ui_` előtagot (ehelyett használhatjuk az `ui` névteret).
  - a generált osztály vezérlőire a nevükkel hivatkozhatunk,
  - a kialakításukat a generált osztály `setupUi (QWidget* parent)` metódusának hívásával végezhetjük el.

# Példa

---

```
#include "ui_quitwidget.h" // tervező által generált
...
class MyWidget : public QWidget {
    Q_OBJECT
public:
    MyWidget(QWidget *parent) : QWidget(parent), ui(new Ui::MyWindow)
    {
        ui->setupUi(this);
    }
private:
    Ui::MyWindow* ui;
};
```



innentől használhatók a tervezővel definiált vezérlők: `ui->...`

# Egyszerű, egyablakos alkalmazások

vezérlők, ablakok, elrendezők

# Vezérlők

---

- ❑ Egy eseményvezérelt alkalmazás speciális objektumokból, úgynevezett vezérlőkből áll.
- ❑ A vezérlők olyan elemek, amelyek **események kezelésére** és **szignálok kiváltására** alkalmasak. Számos esetben egy rájuk jellemző **grafikus megjelenés** is tartozik hozzájuk.
- ❑ A vezérlők osztályai (akár előre definiáltak, akár sajátok) objektum-orientáltan valósulnak meg: származtatás segítségével szerveződnek hierarchiába.
  - Minden **QObject**-ból származó osztály példánya egy vezérlő, ennél fogva kihasználja a Qt speciális vonásait, azaz eseményeket definiál és vált ki (aszinkron üzenet), eseménykezelőkkel, tulajdonságokkal rendelkezik, időzítés kezelésre alkalmas.
  - a **QObject** példányok nem másolhatók, ezért jórész mutatók és referenciák segítségével kezeljük őket

# Tulajdonságok szabályozása

- A vezérlők tulajdonságai az adattagjainak **lekérdező** (*getter*), illetve **beállító** (*setter*) műveleteinek segítségével szabályozhatók
  - a lekérdező művelet neve a tulajdonság neve,
  - a beállító művelet tartalmaz egy **set** előtagot

a szöveg más nyelven történő megjelenítését szolgálja

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



# Grafikus vezérlők

---

- ❑ A leggyakrabban használt (előre definiált) grafikus 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`)

# Vezérlők grafikus tulajdonságai

---

- ❑ A grafikus megjelenéssel is rendelkező vezérlők (grafikus vezérlők) a **QWidget** osztályból származnak (amely a **QObject** alosztálya).
- ❑ Fontosabb tulajdonságaik:
  - **méret** (**size**), vagy **geometria** (elhelyezkedés és méret, **geometry**)
    - A vezérlők mérete többféleképpen befolyásolható: változtatható méretűek esetén külön állítható minimum (**minimumSize**), maximum (**maximumSize**), valamint az alapértelmezett (**baseSize**) méret. A méret rögzíthető (**setFixedSize**).
  - **szöveg** (**text**), **betűtípus** (**font**), **stílus** (**styleSheet**), **színpaletta** (**palette**), **előugró szöveg** (**toolTip**)
    - A grafikus vezérlőkön (pl. **QLabel**, **QLineEdit**) elhelyezett szöveg formázható több módon pl. formátummal (**textFormat**), vagy HTML formázó utasításokkal.
  - **fókuszáltság** (**focus**),
  - **láthatóság** (**visible**)
  - **engedélyezés** (használható-e a vezérlő, **enabled**)

# Szövegkezelés

---

- ❑ Qt-ben a karakterek 16 bites Unicode (UTF8) kódolásúak.
  - Ehhez már a `QObject` típus biztosít konverziót egy osztályszintű művelettel (`QObject::trUtf8`).
- ❑ A karakterek kezelését a `QChar` típus biztosítja, míg szövegre a `QString` típus alkalmazható.
  - kompatibilis a C++ standard könyvtár `string` típusával, pl.:  
`QString::fromStdString(stdstr)`
  - megkülönbözteti az üres és a nem létező szöveget (`isNull`, `isEmpty`)
  - alkalmas típuskonverziókra, pl.:  
`QString::number(4), str.toInt()`

# Vezérlők elhelyezkedési hierarchiája

---

- ❑ Egy alkalmazás grafikus vezérlői között **elhelyezkedési hierarchiát** állíthatunk fel, amely egy fának megfelelő struktúrával reprezentálható.
  - A vezérlőnek lehet **szülője** (**parent**), amelyen belül található. A vezérlő szülőjét konstruktor paraméterben, vagy a **parent** tulajdonságon keresztül adhatjuk meg.
  - A vezérlőnek lehetnek **gyerekei** (**children**), azon vezérlők, amelyek rajta helyezkednek el.
  - Ha egy szülő vezérlőt elrejtünk/megjelenítünk, ki-/bekapcsolunk, vagy megsemmisítünk, akkor az összes gyerekein is megtörténik ugyanez a tevékenység.

# Ablakok

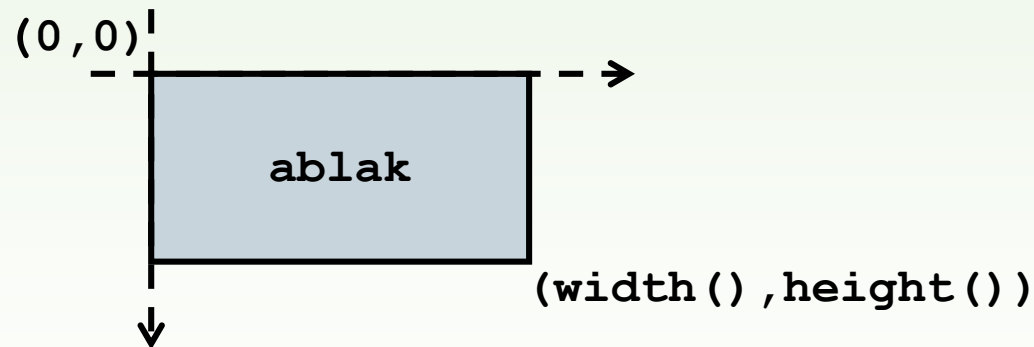
---

- ❑ Egy grafikus felhasználói felület ablakokból tevődik össze, amelyeken vezérlőket helyezünk el.
- ❑ A Qt-ben ablaknak minősül bármely grafikus vezérlő (azaz egy `QWidget`-nek, vagy annak leszármazottjának példánya), amelynek nincs szülője.
- ❑ Az ablak speciális tulajdonságai:
  - **cím** (`windowTitle`), **ikon** (`windowIcon`) vagy akár ezek nélkül: `setWindowState(Qt::WindowFullScreen)`
  - **mérete** állítható teljes/normál képernyőre, vagy a tálcára (`showMaximized`, `showNormal`, `showMinimized`)
  - egyszerre csak egy **aktív** ablak lehet (`isActiveWindow`), amelyet az `activeWindow()` metódus tesz fókuszba.
- ❑ Egy ablak lehet
  - **modális** (*modal*), ha megnyitása után csak a bezárásával lehet az alkalmazás másik ablakát fókuszba tenni, illetve
  - **nem modális** (*modeless*), ha bezárása nélkül át tudunk váltani az alkalmazás másik ablakára.

# Vezérlő ablakban

---

- Amennyiben egy grafikus vezérlőt egy ablakban helyezünk el, meg kell adnunk az **elhelyezkedését** (geometriáját), azaz a pozícióját és méretét (`setGeometry(int, int, int, int)`).
  - Az ablak koordinátarendszere a bal felső sarokból indul a (0,0) koordinátával, és balra, illetve lefelé növekszik.



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

# Példa

---

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

QPushButton quitButton(tr("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
...
```

# Egyedi ablakok

- ❑ A saját testreszabott ablakainkat érdemes saját osztályból létrehozni.

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

a konstruktor  
megkaphatja a szülőt

```
MyWindow::MyWindow(QWidget* parent) : QWidget(parent)
{
    setBaseSize(200, 120);
    setWindowTitle(tr("Demo Window"));
    quitButton = new QPushButton("Quit", this);
    quitButton->setGeometry(10, 40, 180, 40);

    connect(quitButton, SIGNAL(clicked()),
            QApplication::instance(), SLOT(quit()));
}
```

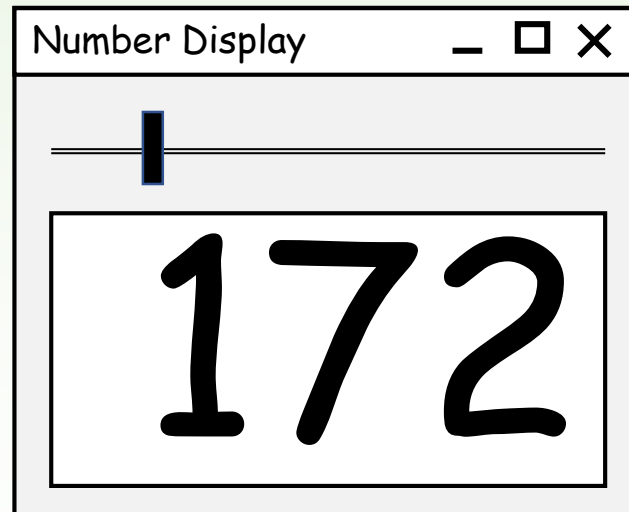
az eseménykezeléshez  
lekérdezzük az alkalmazás-  
példányt



# 1.Feladat

---

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



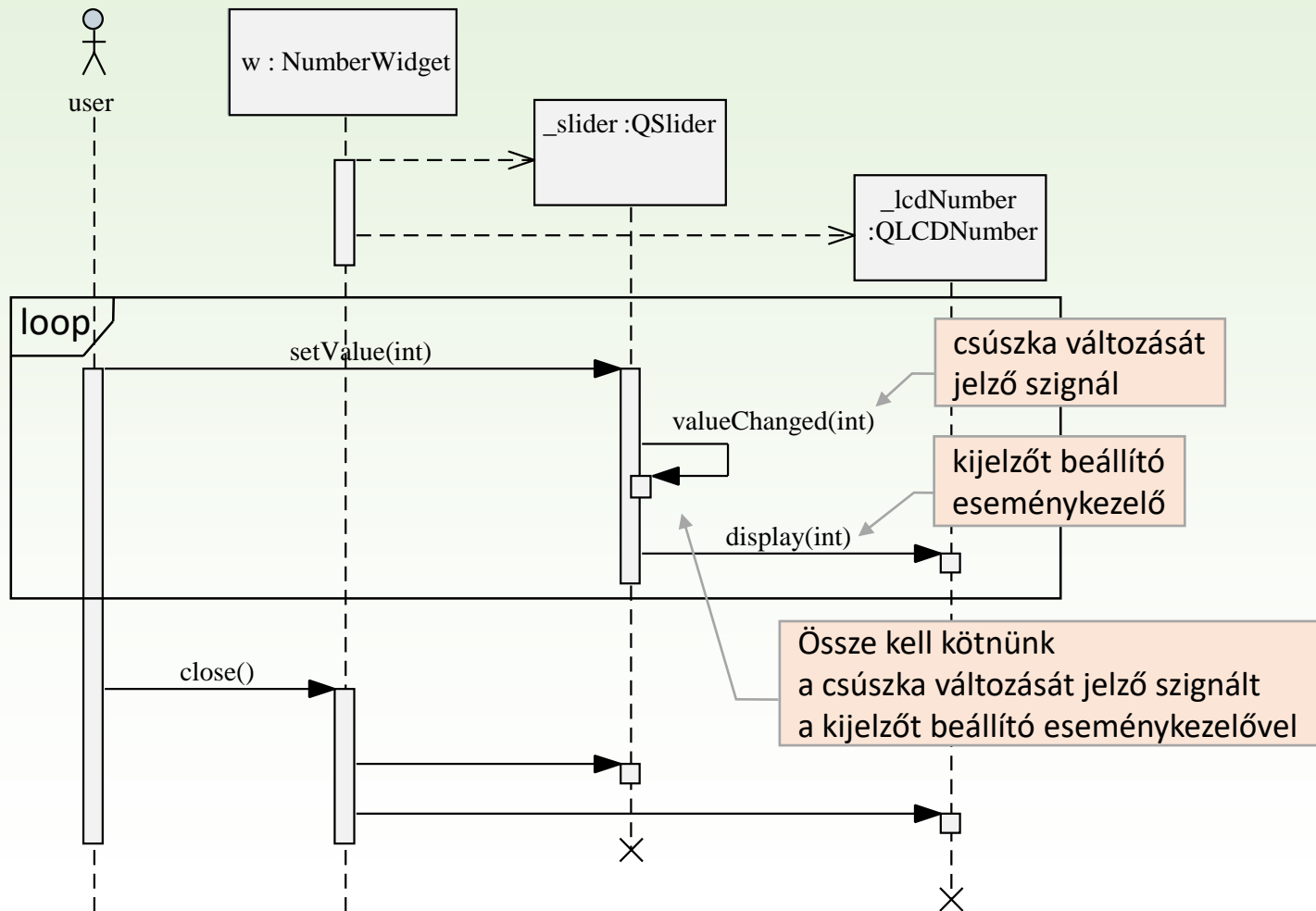
# 1.Feladat: tervezés

---

- Az alkalmazás számára létrehozunk egy új ablak osztályt (**NumberWidget**), felhelyezünk rá egy csúszkát (**QSlider**), és egy számkijelzőt (**QLCDNumber**).
  - A signal-slot társításokat a konstruktorban is megadhatjuk, így már csak a destruktort kell megvalósítanunk, amely törli a vezérlőket.

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

# 1.Feladat: tervezés

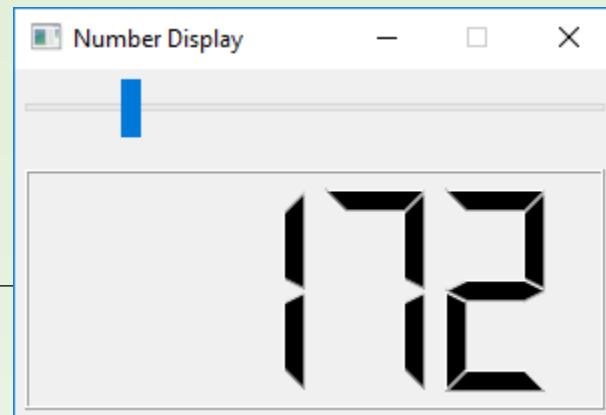


# 1.Feladat: megvalósítás

```
#include <QApplication>
#include "numberwidget.h"
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    NumberWidget w;
    w.show();
    return a.exec();
}
```

```
#include <QWidget>
#include <QSlider>
#include <QLCDNumber>
class NumberWidget : public QWidget
{
public:
    NumberWidget(QWidget *parent = 0);
    ~NumberWidget();
private:
    QSlider* _slider;
    QLCDNumber* _lcdNumber;
};
```

# 1.Feladat: megvalósítás



```
NumberWidget::NumberWidget(QWidget *parent) : QWidget(parent)
{
    setWindowTitle(tr("Number Display")); // ablakcím
    setFixedSize(300, 175);               // rögzített méret beállítása
    _slider = new QSlider(this);           // a vezérlő szülője az ablak
    _slider->setMinimum(0);                 // számhatárok beállítása
    _slider->setMaximum(1000);
    _slider->setValue(0);                   // aktuális érték beállítása
    _slider->setOrientation(Qt::Horizontal); // csúszkairány
    _slider->setGeometry(5, 5, 290, 30);    // elhelyezkedés
    _lcdNumber = new QLCDNumber(4, this);   // a számjegyek száma
    _lcdNumber->display(0);                  // érték megjelenítése
    _lcdNumber->setGeometry(5, 50, 290, 120);

    connect(_slider, SIGNAL(valueChanged(int)),
            _lcdNumber, SLOT(display(int)));
}
```

# Egyedi szignálok és eseménykezelők

- ❑ Egyedi szignálok és kezelőiket csak azon `QObject` osztályból származtatott osztályokban használhatunk, amelyek tartalmazzák a `Q_OBJECT` makrórt.

- ❑ Ezeket az osztály **signals** és **slots** részében kell deklarálni a láthatóságuk megadásával együtt.

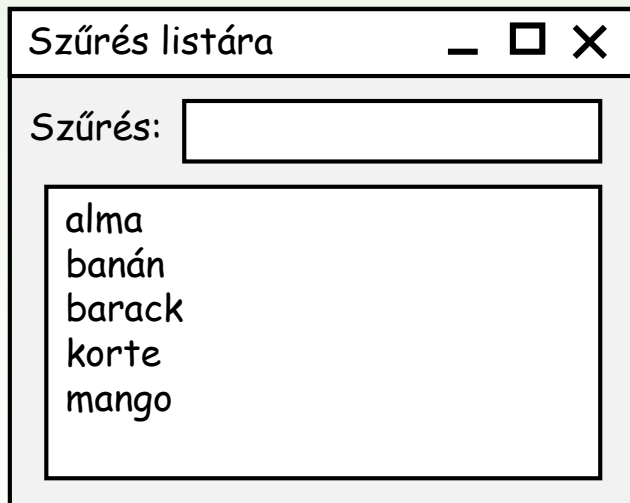
- ❑ Az eseménykezelőket definiálni is (**void** típussal), tetszőleges paraméterezéssel.

- ❑ A szignálnak legalább annyi paraméterrel kell rendelkeznie, mint a neki megfeleltetett eseménykezelőnek. A paraméterek átadása sorrendben történik, ezért a társításnál (`connect`) csak a típusokat jelezzük. A szignál paramétereinek lehet alapértelmezett értéke is.

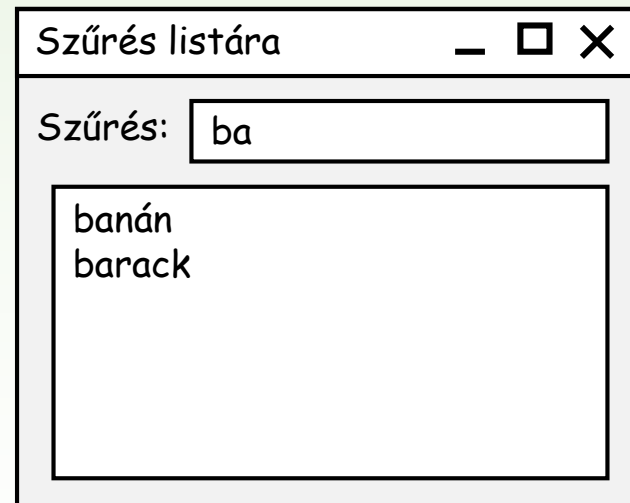
```
class MyObject : public QObject {
    Q_OBJECT
signals:
    void mySignal(int param = 0);
public slots:
    void mySlot(int param){ ... }
};
...
connect(this, SIGNAL(mySignal(int)),
        this, SLOT(mySlot(int)));
```

## 2.Feladat

Készítsünk egy egyszerű alkalmazást, amelyben egy szavakból álló listát jelenítünk meg, amely tartalmát egy szövegdoboz segítségével szűrhetjük. A szavakat szöveges állományból töltjük be.



The screenshot shows a window titled "Szűrés listára" with standard Windows window controls (minimize, maximize, close). Inside the window, there is a label "Szűrés:" followed by an empty text input box. Below the input box is a list box containing five items: "alma", "banán", "barack", "korte", and "mango".



The screenshot shows the same window titled "Szűrés listára". The "Szűrés:" text input box now contains the text "ba". The list box below it displays only two items: "banán" and "barack", which are the words from the original list that start with "ba".

## 2.Feladat: tervezés

- ❑ Az új ablak (**FilteredListWidget**) grafikus felülete tartalmaz egy listamegjelenítőt (**QListWidget**) és egy szövegdobozt (**QLineEdit**) egy címkével (**QLabel**).
- ❑ A háttérben a szűretlen szavak listáját egy szöveglistában tároljuk (**QStringList**). Ezt az **input.txt** fájlból töltjük fel (**loadItems()**) Qt-s fájlkezelést használva (**QFile**).
- ❑ Szükségünk van továbbá egy egyedi eseménykezelőre (**filterList()**), amely a szűrést elvégzi.

<i>QWidget</i> <b>FilteredListWidget</b>	
-	<b>_itemStringList :QStringList</b>
-	<b>_queryLabel :QLabel*</b>
-	<b>_queryLineEdit :QLineEdit*</b>
-	<b>_resultListWidget :QListWidget*</b>
+	<b>FilteredListWidget(QWidget*)</b>
+	<b>~FilteredListWidget()</b>
-	<b>loadItems(QString) :void</b>
«slot»	
-	<b>filterList() :void</b>



## 2.Feladat: megvalósítás

```
#include <QWidget>
#include <QLabel>
#include <QLineEdit>
#include <QListWidget>

class FilteredListWidget : public QWidget {
    Q_OBJECT
public:
    FilteredListWidget(QWidget *parent = nullptr);
    ~FilteredListWidget();
private slots:
    void filterList();    // lista szűrése
private:
    void loadItems(QString fileName); // adatok betöltése fájlból

    QStringList _itemStringList;    // szavak listája
    QLabel *_queryLabel;            // címke
    QLineEdit *_queryLineEdit;      // sorszerkesztő QLineEdit
    *_resultListWidget;             // listamegjelenítő
};
```

## 2.Feladat: megvalósítás

```
FilteredListWidget::FilteredListWidget(QWidget *parent)
: QWidget(parent)
{
    setFixedSize(256, 232);
    setWindowTitle(tr("Szűrés listára"));

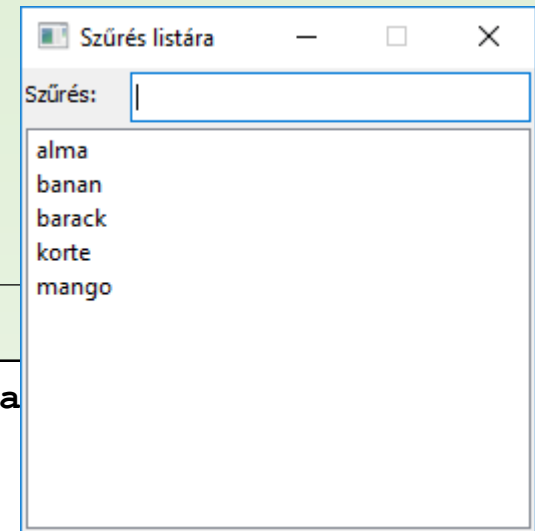
    _queryLabel = new QLabel(tr("Szűrés:"), this);
    _queryLabel->setGeometry(2, 2, 50, 20);

    _queryLineEdit = new QLineEdit(this);
    _queryLineEdit->setGeometry(54, 2, 200, 25);

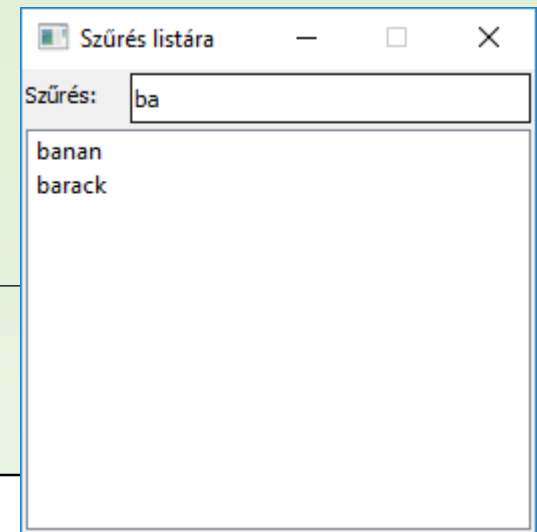
    _resultListWidget = new QListWidget(this);
    _resultListWidget->setGeometry(2, 30, 252, 200);

    connect(_queryLineEdit, SIGNAL(textChanged(QString)),
            this, SLOT(filterList()));

    loadItems("input.txt");
}
```



## 2.Feladat: megvalósítás



```
void FilteredListWidget::filterList()
{
    _resultListWidget->clear(); // kitöröljük a korábbi tartalmat

    if (_queryLineEdit->text().isNull()) { // ha nincs szűrés
        _resultListWidget->addItem(_itemStringList);
        // mindent felvesszünk a listára
    } else { // ha van szűrés
        for (int i = 0; i < _itemStringList.size(); i++)
            if (_itemStringList[i].contains(_queryLineEdit->text()))
                _resultListWidget->addItem(_itemStringList[i]);
        // felvesszük a listára, ha tartalmazza a
        // megadott szöveget
    }
}
```

## 2.Feladat: megvalósítás

---

```
void FilteredListWidget::loadItems(QString fileName)
{
    QFile file(fileName);           // logikai fájl létrehozása
    if (file.open(QFile::ReadOnly)) { // megnyitás csak olvasásra
        _itemStringList.clear();    // régi elemek törlése
        QTextStream stream(&file);  // szöveggént olvassuk be a fájlt
        QString line = stream.readLine(); // soronként olvasunk
        while (!stream.atEnd()) {    // !line.isNull() is lehetne
            _itemStringList << line; // _itemStringList.append(line);
            line = stream.readLine();
        }
        _queryLineEdit->clear();    // töröljük tartalmát
        _resultListWidget->clear(); // töröljük tartalmát
        _resultListWidget->addItem(_itemStringList); // új elemek
    } else
        QMessageBox::warning(this, tr("Hiba!"), tr("A ")
                               + fileName + tr(" fájl nem található!"));
        // ha nem sikerült megnyitni, előugró ablakot mutatunk
}
```

# Speciális ablakok

---

- ❑ **Főablak** (`QMainWindow`), amely egy alkalmazás grafikus vezérlőinek tárolására szolgál. Számos kiegészítést biztosít összetett ablakok megvalósítására:
  - menübár,
  - állapotsor,
  - ikonbár,
  - beágyazott ablakok kezelése
- ❑ **Dialógus ablakok** a `QDialog` osztályból származtatott osztályok példányai, amelyek lezárása után lekérdezhetjük az annak használata során megadott felhasználói választásokat, vagy éppen a kilépésre használt gombot.

# Dialógus ablakok

- ❑ Egy dialógus ablak megjelenítésének módja
  - modális, ha ehhez az `exec()` metódust használjuk,
  - nem modális, ha ehhez a `show()` metódust használjuk, feltéve, hogy nem állítottuk a `setModal()` metódussal eleve modálisra.
- ❑ A dialógus ablakokat az `accept()` vagy a `reject()` eseménykezelővel zárhatjuk be: modális hívás esetén az `exec()` metódus igaz értékkel tér vissza, ha az `accept()`-et használjuk, hamissal, ha a `reject()`-et.
- ❑ Számos előre definiált dialógus ablak létezik:
  - Rögzített dialógusok: `QFileDialog`, `QColorDialog`, `QFontDialog`, `QPrintDialog`, `QInputDialog`, `QProgressDialog`, `QErrorMessage`.
  - Konfigurálható üzenőablak (`QMessageBox`), amely alkalmas üzenet (`information`), hiba (`critical`), figyelmeztetés (`warning`) közlésére, vagy kérdés (`question`) feltételére.

```
QMessageBox::question(this, tr("Confirm"), tr("Do you want to exit?" ),  
                      QMessageBox::Yes | QMessageBox::Default, QMessageBox::No);
```

# Fájl dialógus

---

- ❑ Egy speciális dialógusablak a fájl dialógus (`QFileDialog`), amely lehetőséget fájlok/könyvtárak kiválasztására
  - statikus műveletekkel közvetlenül használható fájlok megnyitásához (`getOpenFileName`, `getOpenFileNames`), fájlok mentéséhez (`getSaveFileName`) és könyvtárak megnyitásához (`getExistingDirectory`)

```
QString fileName = QFileDialog::getOpenFileName(this,  
    tr("Open file"), "/home", tr("Text files (*.txt)"));  
    // szövegfájl megnyitása a home könyvtárból
```

# Grafikus vezérlők elrendezése

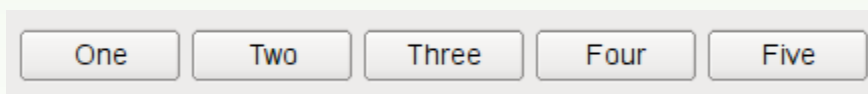
---

- ❑ Mivel az ablak átméretezésével a vezérlők elrendezését módosítani kell, célszerű az átméretezhető ablakoknál **elrendezéseket** (*layout*) használni.
- ❑ Az elrendezések a gyerekvezérlőiket megfelelő sorrendben jelenítik meg, automatikusan áthelyezik és átméretezik.
- ❑ Az elrendezések hierarchikusan egymásba ágyazhatók (**`addLayout()`**), és a hierarchia tetején levő elrendezést a **`setLayout(QLayout*)`** utasítással állíthatunk rá az azt tartalmazó vezérlőre (elsősorban az ablakra).
- ❑ Az elemek távolsága egy elrendezésen belül szabályozható (**`spacing`**).

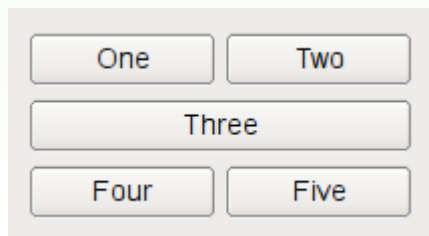


# Grafikus vezérlők elrendezői

- ❑ Számos formának megfelelően rendezhetjük 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



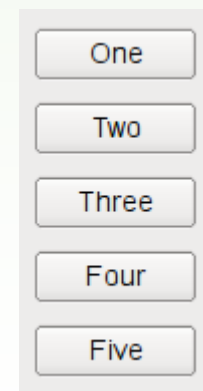
**QHBoxLayout**



**QGridLayout**



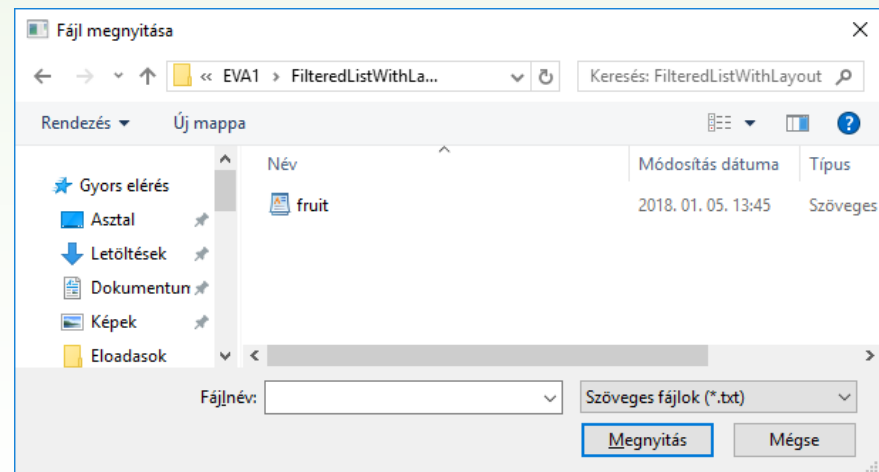
**QFormLayout**



**QVBoxLayout**

# 3.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.



# 3.Feladat: tervezés

Az eddigieken túl

- ❑ Felveszünk egy új nyomógombot, amely a szöveges állományból történő beolvasás indítja el egy erre a célra készített eseménykezelővel (`loadFile()`). Ez az állomány nevét egy fájl kiválasztó dialógusablak (`QFileDialog`) segítségével olvassa be, majd meghívja a már meglevő `loadItems()` metódust.
- ❑ Alkalmazunk **elrendezőket** a felületen, a felső sornak egy vízszinteset (`QHBoxLayout`), a teljes tartalomnak egy függőlegeset (`QVBoxLayout`).

<i>QWidget</i> <b>FilteredListWidget</b>	
-	<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>
+	<code>FilteredListWidget(QWidget*)</code>
+	<code>~FilteredListWidget()</code>
-	<code>loadItems(QString) :void</code>
«slot»	
-	<code>filterList() :void</code>
-	<code>loadFile() :void</code>

# 3.Feladat: megvalósítás

```
FilteredListWidget::FilteredListWidget(QWidget *parent) :
QWidget(parent) {
    ...
    _upperLayout = new QHBoxLayout;
    _upperLayout->addWidget(_queryLabel);
    _upperLayout->addWidget(_queryLineEdit);

    _mainLayout = new QVBoxLayout;
    _mainLayout->addLayout(_upperLayout);
    _mainLayout->addWidget(_resultListWidget);
    _mainLayout->addWidget(_loadButton);

    setLayout(_mainLayout);
    ...
}
```

```
void FilteredListWidget::loadFile() {
    QString fileName =
        QFileDialog::getOpenFileName(this,
        tr("Fájl megnyitása"), "", tr("Szöveg fájlok (*.txt)"));
    if (!fileName.isNull()) loadItems(fileName);
}
```

fájl megnyitó dialógus

ha OK-val zártuk le a fájl dialógust