C++

14. HÉT – QT GUI

A JEGYZETET KÉSZÍTETTE:

CSAPÓ ÁDÁM BALÁZS ÉS ŐSZ OLIVÉR

Qt ökoszisztéma

https://www.qt.io

Qt application framework

- Platformfüggetlen GUI framework C++ nyelven (de más nyelvekből is használható)
- Tetszőleges compiler használható hozzá

Qt Creator

- IDE, sok hasznos eszközzel (verziókezelők, debugger, emulátor, profiler, stb.)

qmake

- Build automatizáló eszköz (makefile-okat generál, mint a CMake)

Könyvtárak

- Pl.: hálózati kommunikációhoz, multimédiához, adatbázis-kapcsolathoz

A 2 Qt GUI keretrendszer

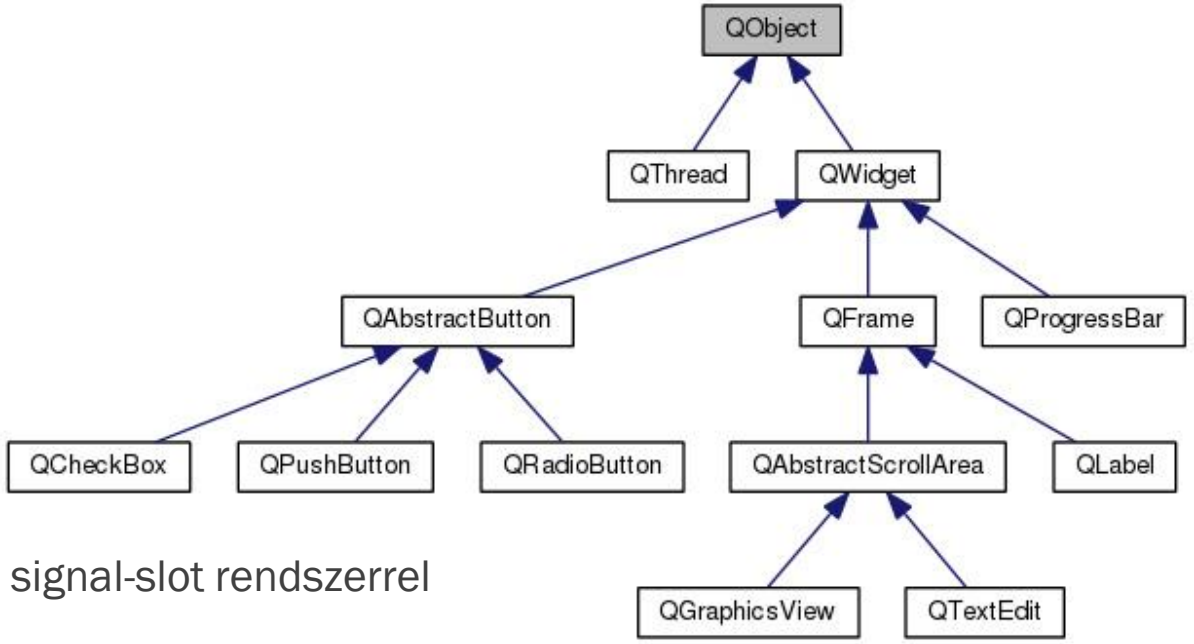
Qt Widgets

- C++ osztályok

Qt Quick

- QML alapú (CSS-re és JavaScriptre épül)
- Integrálható C++ osztályokkal

Qt Widgets



A Qt könyvtárakban szinte minden osztálynak a QObject az őse

- Rendelkeznek szálbiztos eseménykezelésre alkalmas signal-slot rendszerrel
- Objektumhierarchiával
  - A szülő számon tartja és automatikusan destruktálja a gyerekeit
  - Név alapján megkereshető egy gyerekobjektum a findChild() metódussal
- Dinamikus, gyengén típusos property kezeléssel

A Widgets könyvtár osztályai pedig a QWidget-ből származó GUI elemek

- Számos jól ismert beépített elem van, és saját widget osztályokat is származtathatunk belőlük

QApplication

Egy Qt Widgets-et használó GUI program indításához létre kell hozni egy QApplication objektumot

- Ezt kell legelőször létrehozni

Majd a widgetek inicializálása után meghívni az exec() metódusát

- Ez egy külön szálon elindítja az event-loopot, ami majd az események hatására meghívja az objektumok megfelelő függvényeit

```
#include <QApplication>
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    return app.exec();
}
```

Widgetek megjelenítése

Bármely widget önmagában is lehet egy ablak:

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

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QPushButton b("Hello Qt!");
    b.show();
    return app.exec();
}
```

Widgetek csoportosítása

Minden widget a szülőjén belül kerül elhelyezésre

- A szülőhöz viszonyított relatív koordináta-rendszert használ
- A QWidget egy üres widget, de használható csoportosításra, ha gyerek widgeteket adunk hozzá
- A widgeteknek a konstruktorban adható meg a szülőjük
- Amelyik widgetnek nincs szülője, az egy különálló ablakban jelenik meg, ha meghívjuk a show() metódusát

```
QApplication app (argc, argv);
QWidget window;

QPushButton *a = new QPushButton("A", &window);
QPushButton *b = new QPushButton("B", &window);
QPushButton *c = new QPushButton("C", b); // inside b

a->setGeometry( 50, 100, 100, 200); // x, y, w, h
b->setGeometry(200, 100, 350, 200);
c->setGeometry( 50,  50,  50,  50);

window.show();
return app.exec();
```

Layout használata

A widgetek helyzetét pixelekben megadni kényelmetlen és nem reszponzív

- Az ablak átméretezése letiltható, de ez nem túl felhasználóbarát megoldás:

```
window.setFixedSize(600, 400);
```

A widgetek elrendezésére különböző stratégiát használó Layout osztályok állnak rendelkezésre

- QHBoxLayout: vízszintesen egymás mellé
- QVBoxLayout: függőlegesen egymás alá
- QGridLayout: táblázatszerű elrendezés, sorokkal és oszlopokkal
- QFormLayout: két oszlopos elrendezés formokhoz (címke, mező)

Layout használata

A Layout nem egy widget, az általa kezelt widgetek nem a gyerekei

A Layout szülőjeként kell beállítani a widgetet, aminek az elrendezését kezeli
- A Layout-hoz adott widgeteknek automatikusan az általa kezelt widgetet állítja be szülőként

Az addWidget() metódussal rendelhetünk hozzá widgetet

```
QWidget window;

QPushButton *a = new QPushButton("A"); // no parent
QPushButton *b = new QPushButton("B");
QPushButton *c = new QPushButton("C");

QHBoxLayout *layout = new QHBoxLayout(&window);
layout->addWidget(a); // window becomes parent of A
layout->addWidget(b);
layout->addWidget(c);

window.show();
```

Layout használata

A Layoutban lévő widgetek mérete és helyzete automatikusan változik, ha a szülő widget átméreteződik

A növekedés arányait az addWidget()-nek megadott stretch factorral lehet szabályozni

Az igazítást pedig az AlignmentFlag megadásával

```
QHBoxLayout *layout = new QHBoxLayout(&window);
layout->addWidget(a, 1, Qt::AlignTop);
layout->addWidget(b); // stretch factor = 0
b->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding); // hor, vert
layout->addWidget(c, 2, Qt::AlignBottom);
```

Eseménykezelés

Az események kezelése az Observer design pattern-re épül

- Sok más GUI framework-höz hasonlóan

Az Observer objektumok feliratkoznak a számukra fontos eseményekre, és meghívásra kerülnek, amikor bekövetkezik az esemény

- A feliratkozás során megjegyzésre kerülnek az Observerek, akiket értesíteni kell

Qt-ban az események egy queue-ba kerülnek, és a bekövetkezési sorrendben kerülnek feldolgozásra

- Ha egy esemény egy újabb eseményt vált ki, az a sor végére kerül
- Ha egy esemény egy időigényesebb választ vált ki, azt érdemes egy külön szálon elvégezni

Signalok és slotok

Qt-ban az eseményeket signaloknak is hívják, melyeket kibocsáthatnak QObjectek

Más QObjectek feliratkozhatnak egy objektum signaljaira

- Meg kell adni, melyik metódusuk (slotjuk) hívódjon meg a signal hatására
- Feliratkozni a QObject::connect(sender, signal, receiver, slot) statikus függvénnyel lehet
- Leiratkozás: QObject::disconnect (sender, signal, receiver, slot)

A feliratkozók nem a küldőnél kerülnek tárolásra, hanem a Meta Object rendszer tárolja őket

- A Meta Object Compiler (moc) makrók segítségével kezeli a feliratkozásokat
- A connect-ben a signal és slot megadásánál a SIGNAL() és SLOT() makrófüggvényeket kell használni, melyeknek a signal/slot függvények szignatúráját kell megadni paraméterként
  - Qt5 óta átadható függvény pointer is, így az std::bind segítségével az eltérő paraméterlista is kezelhető
- Saját osztályainkban a Q_OBJECT makrót kell elhelyezni, hogy működjenek a signalok, slotok

Signal-slot példa

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

int main(int argc, char *argv[]) {
  QApplication app (argc, argv);
  QWidget window;
  QPushButton *btn = new QPushButton("Show message", &window);

  QMessageBox *msg = new QMessageBox();
  msg->setText("Button pressed");

  QObject::connect(btn, SIGNAL(clicked()), msg, SLOT(exec())); // Qt4 syntax
  QObject::connect(btn, &QPushButton::clicked, msg, &QDialog::exec); // new syntax

  window.show();
  return app.exec();
}
```

Saját widget származtatása

Ahhoz, hogy a beépített funkciókon felül saját viselkedést tudjunk adni a widgeteknek, saját osztályt kell származtatnuk belőlük

A QWidget-ből vagy leszármazottjából származtatunk

Q_OBJECT makró

A private/protected/public részek mellett megadható signals és private/protected/public slots

- A signalok implementáció nélküli void visszatérésű függvények
- Kibocsátásuk: `emit mySignalName(params);`
- A slotok ugyanolyanok, mint a hagyományos tagfüggvények, csak egy jelzés, hogy ez eseményekre reagál
  - Normál módon is hívható függvények

```
#ifndef CLICKCOUNTER_HH
#define CLICKCOUNTER_HH

#include <QWidget>
#include <QLabel>
#include <QPushButton>

class ClickCounter : public QWidget
{
    Q_OBJECT
public:
    explicit ClickCounter(QWidget *parent = nullptr);

signals:
    void countChanged(int);

public slots:
    void increase();
    void reset();

private:
    int count = 0;
    QLabel* label;
    QPushButton* addBtn;
    QPushButton* resetBtn;
};

#endif // CLICKCOUNTER_HH
```

```cpp
#include "clickcounter.hh"
#include <QVBoxLayout>

ClickCounter::ClickCounter(QWidget *parent)
    : QWidget{parent}
    , label(new QLabel("0"))
    , addBtn(new QPushButton("Add"))
    , resetBtn(new QPushButton("Reset"))
{
    QVBoxLayout *layout = new QVBoxLayout(this);
    layout->addWidget(label);
    layout->addWidget(addBtn);
    layout->addWidget(resetBtn);

    QObject::connect(addBtn, SIGNAL(clicked()), this, SLOT(increase()));
    QObject::connect(resetBtn, &QPushButton::clicked, this, &ClickCounter::reset);
}

void ClickCounter::increase() {
    ++count;
    label->setText(QString::number(count));
    emit countChanged(count);
}

void ClickCounter::reset() {
    count = 0;
    label->setText(QString::number(count));
    emit countChanged(count);
}
```

Qt Designer

Qt Creator-ben van egy vizuális szerkesztő is, amivel kényelmesebben állítható az elemek elrendezése

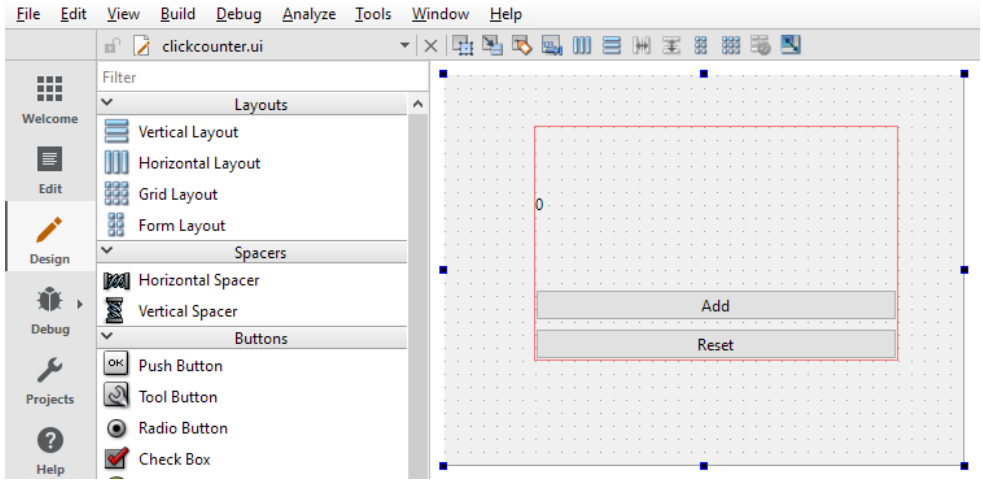
Az elrendezést egy XML-alapú .ui fájl tárolja

- Ebből generál egy C++ osztályt
- A logikát tartalmazó saját osztályunkban tárolhatjuk egy példányát, így az kezeli helyettünk a GUI-elemeket

Létrehozás:

- Project > Add New… > Qt Designer Form

Példa




```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>ClickCounter</class>
 <widget class="QWidget" name="ClickCounter">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>400</width>
    <height>300</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Form</string>
  </property>
  <widget class="QWidget" name="verticalLayoutWidget">
   <property name="geometry">
    <rect>
     <x>69</x>
     <y>39</y>
     <width>281</width>
     <height>181</height>
    </rect>
   </property>
   <layout class="QVBoxLayout" name="verticalLayout">
    <item>
     <widget class="QLabel" name="label">
      <property name="text">
       <string>0</string>
      </property>
     </widget>
    </item>
    <item>
     <widget class="QPushButton" name="addBtn">
      <property name="text">
       <string>Add</string>
      </property>
     </widget>
    </item>
```

Példa

```h
#ifndef CLICKCOUNTER_HH
#define CLICKCOUNTER_HH
#include <QWidget>

QT_BEGIN_NAMESPACE
namespace Ui { class ClickCounter; }
QT_END_NAMESPACE

class ClickCounter : public QWidget {
    Q_OBJECT
public:
    explicit ClickCounter(QWidget *parent = nullptr);

signals:
    void countChanged(int);

public slots:
    void increase();
    void reset();

private:
    int count = 0;
    Ui::ClickCounter* ui;
};
#endif // CLICKCOUNTER_HH
```

```cpp
#include "clickcounter.hh"
#include "ui_clickcounter.h"

ClickCounter::ClickCounter(QWidget *parent)
  : QWidget{parent}
  , ui(new Ui::ClickCounter)
{
  ui->setupUi(this);

  QObject::connect(ui->addBtn, SIGNAL(clicked()), this, SLOT(increase()));
  QObject::connect(ui->resetBtn, SIGNAL(clicked()), this, SLOT(reset()));
}

void ClickCounter::increase() {
  ++count;
  ui->label->setText(QString::number(count));
}

void ClickCounter::reset() {
  count = 0;
  ui->label->setText(QString::number(count));
}
```

Qt Quick

QML nyelven leírható GUI megjelenés és JavaScript logika

- Designerek számára könnyebb a UI megtervezése
- Hasonlóan a widgeteknél látott Designerhez, van hozzá WYSIWYG szerkesztő, de különálló programként: Qt Design Studio

C++ nyelven írhatunk új saját típusokat is, amik elérhetőek lesznek QML-ben

- És a QML-ben megírt elemek is felhasználhatóak más elemek részeiként

QML

```
import QtQuick 2.15
import QtQuick.Window 2.15

Window {
  width: 640
  height: 480
  visible: true
  title: qsTr("Hello World")

  Rectangle {
    width: 300
    height: 200
    border.width: 1
    Text {
      anchors.horizontalCenter: parent.horizontalCenter
      anchors.verticalCenter: parent.verticalCenter
      text: "Rectangle"
    }
  }
}
```

JavaScript a QML-ben

```
Rectangle {
  width: 300
  height: 200
  border.width: 1
  Text {
    anchors.horizontalCenter: parent.horizontalCenter
    anchors.verticalCenter: parent.verticalCenter
    text: "Rectangle"
  }
  MouseArea {
    anchors.fill: parent
    onClicked: {
      if (parent.border.width == 1) {
        parent.border.width = 3
      } else {
        parent.border.width = 1
      }
    }
  }
}
```

Saját C++ osztály QML-ben

Ahhoz, hogy az osztályunkat tudjuk használni QML-ben:

- QObject-ből kell származtatni
- Q_OBJECT és QML_ELEMENT makrók
- #include <qqml.h>
- Project fájlban beállítások:
  - ```
    CONFIG += qmltypes
    QML_IMPORT_NAME = module.name
    QML_IMPORT_MAJOR_VERSION = 1
    ```

QML-ből is elérhető property megadása az osztályhoz:

- Q_PROPERTY makrófüggvény
  - ```
    Q_PROPERTY(type name
      (READ getFunction [WRITE setFunction] |
       MEMBER memberName [(READ getFunction | WRITE setFunction)])
      [RESET resetFunction] [NOTIFY notifySignal] [REVISION int]
      [DESIGNABLE bool] [SCRIPTABLE bool] [STORED bool] [USER bool] [CONSTANT] [FINAL] [REQUIRED])
    ```

Példa C++ osztály

```
#ifndef USERDATA_HH
#define USERDATA_HH

#include <QObject>
#include <QString>
#include <qqml.h>

class UserData : public QObject {
    Q_OBJECT
    Q_PROPERTY(QString userName READ userName WRITE setUserName NOTIFY userNameChanged)
    QML_ELEMENT
public:
    explicit UserData(QObject *parent = nullptr);

QString userName() { return m_userName; }

void setUserName(const QString &userName) {
    m_userName = userName;
    emit userNameChanged();
}

signals:
    void userNameChanged

private:
    QString m_userName = "User";

};
#endif // USERDATA_HH
```

Példa QML oldal

```
import custom.userdata 1.0

Rectangle {
  width: 300
  height: 200
  border.width: 1

  UserData {
    id: user
  }

  Text {
    anchors.horizontalCenter: parent.horizontalCenter
    anchors.verticalCenter: parent.verticalCenter
    text: user.userName
  }

  MouseArea {
    anchors.fill: parent
    onClicked: {
      user.userName = "Admin"
    }
  }
}
```