# Linux Programozás

## Qt GUI

# Qt grafikus felület

- A QWidgetből származik minden grafikus elem
  - QObject-ből származik.
  - Kezeli a felhasználói eseményeket.
  - Kezeli az ablak jellemzőket.
  - Nyilvántartja a szülő-gyerek relációkat (QObject).
    - Grafikusan a szülőn belül láthatóak a gyerekek.
    - Ha nincs szülője, akkor önálló ablak.
    - A gyerek vezérlőelemeket gyakran layout managerek rendezik el.

# Ablakok

# Ablakok

- Dialógus
  - QDialog-ból származik
  - Egyszerűbb felület
  - Van visszatérési értéke
- Alkalmazás ablak
  - QMainWindow-ból származnak
  - Saját alapértelmezett elrendezése van: menüsor, eszközsor, központi terület, státusz sor

# Dialógus ablakok

- Típusai:
  - Modális
    - Blokkol minden hozzáférést a szülő ablakhoz.
  - Nem modális
    - Nem blokkolja a szülőablak használatát.
- A választás az ablak funkciójától függ:
  - Ha semmi hasznosat nem tudunk tenni a kérdés eldöntéséig, pl. állomány választás, akkor modális.
  - Ha közben mást is csinálunk, pl. keresés, akkor nem modális.

# Modális dialógus ablak

- A QDialog exec() függvényével jelenítjük meg.
- Az alkalmazás eseménykezelő ciklusa blokkolódik és a dialógus ablak indít sajátot.
  - Blokkolja a hozzáférést más ablakokhoz az alkalmazáson belül.
  - Engedi a képernyő frissítést.
- Ha bezárjuk, akkor az exec() visszatér és minden folytatódik.
  - A visszatérési érték megmondja, hogy melyik gombbal zártuk be.
- Tipikusan a felhasználó választásait a dialógus objektum tagváltozóiban tároljuk és a dialógus bezárása után a dialógus létrehozója kiolvashatja.

# Nem modális dialógus ablak

- A show() függvénnyel jelenítjük meg.
- Nincs saját eseménykezelő ciklusa, ami blokkolná a többit.
  - Egyben szükséges is, hogy a QApplication exec() függvénye már fusson.
- Párhuzamosan él az alkalmazás többi részével, ezért a kommunikáció signal-slot mechanizmussal történik.
- Nem bezárjuk, hanem elrejtjük (hide()).

# Dialógus ablak létrehozása

- Származtatunk egy osztályt a QDialog-ból.
- A konstruktorban
  - Létrehozunk vezérlőelem objektumokat (pl. QLabel). Szülőnek a „this”-t adjuk meg.
  - Beállítunk egy elrendezést.
  - A vezérlő elemek szignáljaihoz bekötünk szlot függvényeket.
- Modális dialógus ablak bezárásához az Ok, Cancel, stb. gombok clicked() szignáljára beköthetjük
  - a QDialog accept() és reject() szlotját
  - vagy saját szlotot, ami setResult()-al beállítja a visszatérési értéket és close()-al zárja az ablakot

# Egyszerűbb vezérlő elemek

| Vezérlő | Leírás |
|---|---|
| *QLineEdit, QTextEdit, QDateEdit, QDateTimeEdit, QTimeEdit* | Beviteli mezők szövegekhez, dátumokhoz és időpontokhoz |
| *QCheckBox, QRadioButton* | Jelölőnégyzet és választónégyzet osztályok. |
| *QComboBox* | Legördülő lista. |
| *QFontComboBox* | Betűtípus választó lista. |
| *QLabel* | Egyszerű szöveg, vagy kép megjelenítése. |
| *QMenu* | Menük megjelenítése. |

# Összetettebb vezérlők

| Vezérlő | Leírás |
|---|---|
| *QCalendarWidget* | Naptárat megjelenítő vezérlő |
| *QListWidget* | Elemeket listanézetben megjelenítő vezérlő. |
| *QTableWidget* | Elemeket táblázatnézetben megjelenítő vezérlő. |
| *QTreeWidget* | Elemeket fastruktúrában megjelenítő vezérlő. |
| *QWebView* | Webes dokumentumok megjelenítésére és szerkesztésére használható vezérlő. |

# Vezérlőelemek működése

- Minden vezérlőelem szignálokat publikál, amelyeket az értékük módosítása során adnak ki.
- A vezérlő elemek sokszor szlot függvényeket is nyújtanak az értékük módosításához.

# Layout manager

- A vezérlő elemeket el lehet statikusan helyezni, de nem érdemes.
  - Ablak átméretezés probléma.
  - Más nyelveken más lehet a feliratok hossza.
- A dinamikus elrendezést rábízhatjuk a layout managerekre.
  - QGridBoxLayout: négyzetrácsba rendez
  - QHBoxLayout: vízszintes rendezés
  - QVBoxLayout: függőleges rendezés
- A layout managereket egymásba ágyazhatjuk.

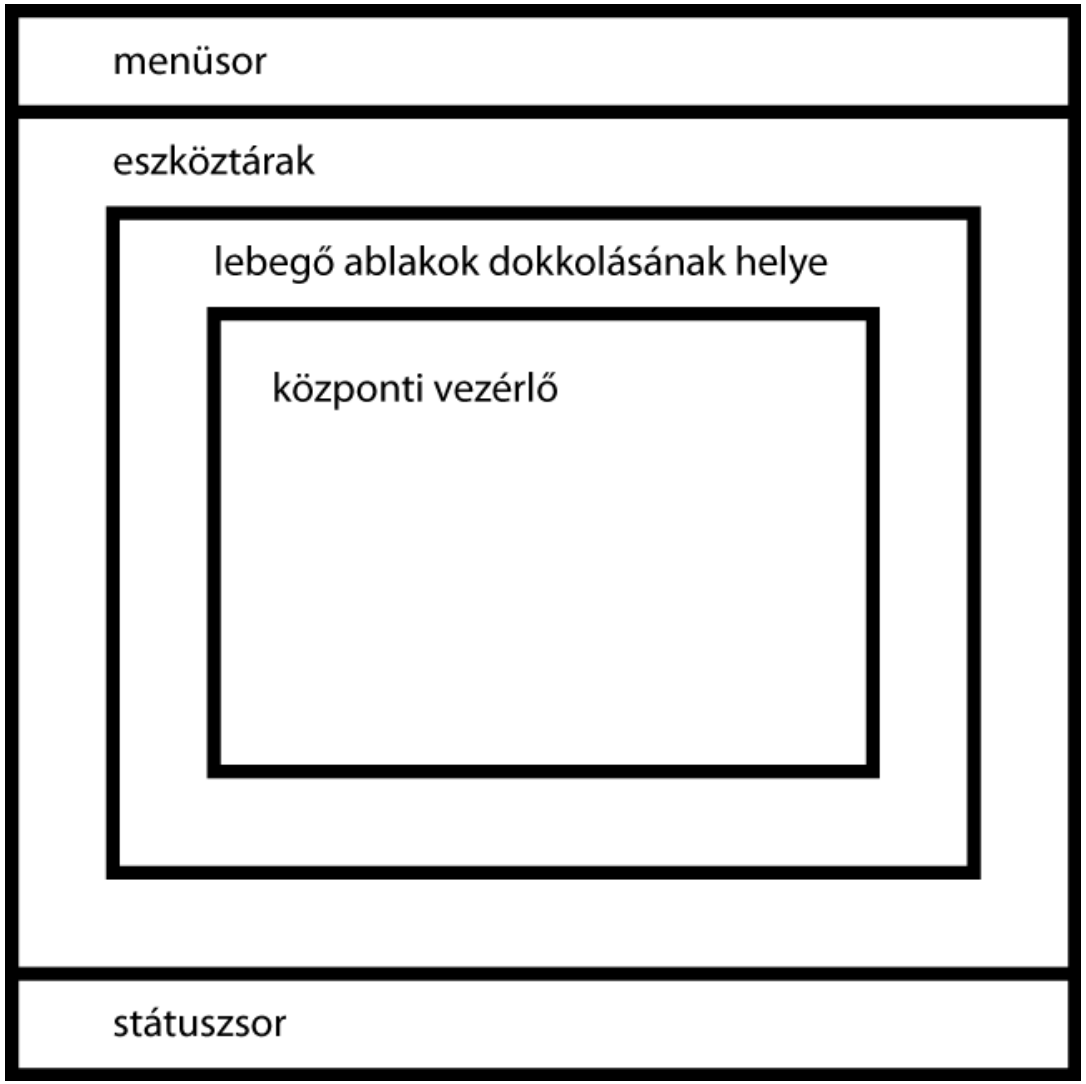
# Dialógus ablakok

| Vezérlő | Leírás |
|---|---|
| *QColorDialog* | Színválasztó dialógusablak. |
| *QFileDialog* | Állományok és könyvtárak kiválasztására használható dialógusablak. |
| *QFontDialog* | Dialógus ablak betűtípus beállításainak szerkesztésére. |
| *QInputDialog* | Egy darab egyszerű típusú adat beolvasása a felhasználótól (a beolvasandó típus lehet szöveg, szám, vagy egy előre megadott lista egy eleme). |
| *QMessageBox* | Üzenet megjelenítése a felhasználó számára. |

# Alkalmazás ablak

- A QMainWindow osztályon alapul
  - Menü
  - Eszközsor
  - Státusz információk
  - Lebegő és dokkolható ablakok

# Alkalmazás ablak



# SDI / MDI

- A QMainWindow használatakor választhatunk SDI és MDI felületek közül.
- SDI: single document interface
  - A központi területen egy dokumentum nyílik meg.
  - A központi vezérlő elem bármi lehet.
- MDI: multiple document interface
  - Egyszerre több dokumentumablakon dolgozhatunk.
  - A központi vezérlő elem QMdiArea típusú. Ehhez adhatunk bármilyen vezérlőelemet, amelyek külön ablakok lesznek.

# Akciók

# Akció

- Egyes funkciók több helyről elérhetőek:
  - Menü
  - Eszköztár
  - Gyorsbillentyű
- Ezeket egységesen akció objektumként (QAction) implementáljuk.
- Az akció objektum
  - Tartalmazza a funkció nevét, ikonját, leírását, stb.
  - Publikál egy szignált, amelyet aktiváláskor ad ki.
- Az akciót hozzáadhatjuk (addAction()) a menühöz, eszköztárhoz, vagy csak az ablakhoz.

# Menü

- A QMainWindow menuBar() függvénye
  - Visszatérési értéke a menüsor (QMenuBar) mutatója.
  - Első meghívásakor jön létre a menüsor.
- A menüsorhoz az addMenu() függvénnyel adhatunk QMenu elemeket.
- A QMenu addAction() függvényével adhatunk akció objektumot a menühöz.

```
QMenu * fileMenu = this->menuBar()->addMenu("Fájl");
exitAction = new QAction("Kilép", this);
fileMenu->addAction(exitAction);
```

# Eszköztár

- Hasonlít a menüsorhoz, azonban több is lehet belőle.
- A QMainWindow addToolBar() függvényével hozhatjuk létre.
- QToolBar a típusa.
- A QToolBar addAction() függvényével adhatunk hozzá akció objektumot.

# Státusz sor

- Egy lehet belőle.
- A QMainWindow statusBar() függvényével érhetjük el.
- Információk típusai:
  - Ideiglenes (temporary) szöveg
  - Általános (normal) információ
  - Állandó (permanent) információ
- Az általános és állandó információ nem csak szöveg lehet.
  - Az addWidget() és removeWidget() függvényekkel adhatunk hozzá, vagy vehetünk el widgetet.
- Ideiglenes szöveget a showMessage() függvénnyel jeleníthetünk meg.
- Ideiglenes szöveg az általános információt eltakarhatja, de az állandót nem.

# Lokalizáció

# Többnyelvűség

- A Qt keretrendszer támogatja a többnyelvűséget.
- A megjelenített szövegkonstansok erőforrásállományokban tárolódnak.
- Az erőforrásállományokat lefordíthatjuk különböző nyelvekre. (Laikusok is megtehetik.)
- A fordításokat betöltjük a programunkban.

# Többnyelvűség

- A szövegkonstanst a tr() függvénnyel jelezzük:

```
QString QObject::tr(const char *
sourceText, const char *
disambiguation = 0, int n = -1 )
```

- sourceText: Az alapértelmezett szöveg
- disambiguation: Jelezhetjük a kontextust
- n: Többesszám kezeléséhez a szám értéke

# Fordítás

- A lupdate parancs generál egy „.ts” kiterjesztésű xml állományt, amiben el kell készítenünk a fordítást.
- Projekt állományban jeleznünk kell:

```
TRANSLATIONS += ContactsApplication_en_US.ts
```

- Majd le kell generálni:

```
lupdate ContactsApplication.pro
```

- A lupdate megkeresi a forráskódban a tr() függvényeket.
- Ha létezik az állomány, akkor az lupdate csak kiegészíti.

# Fordítás

- A „.ts” állományt tömörítenünk kell „.qm” állományra:

```
lrelease ContactsApplication.pro
```

- Az lupdate és az lrelease automatikus futtatására, illetve a szerkesztésre grafikus segédprogramok is rendelkezésre állnak.

# Lokalizáció

- A QTranslator osztályból létrehozunk egy objektumot.
- A load() függvényével betöltjük a „.qm” állományt.
- A QApplication objektum installTranslator() függvényével aktiváljuk.
- Több állomány esetén megismételjük a műveleteket.

# Saját vezérlők (custom widget)

# Lehetőségeink

- Meglévő vezérlő osztályból származtatunk és módosítjuk.
- QWidget osztályból származtatunk és a felületet meglévő vezérlőkből állítjuk össze.
- QWidget osztályból származtatunk és mi írjuk meg teljesen:
  - Felület kirajzolása
  - Felhasználói események lekezelése
  - Kommunikáció a program többi részével

# Események kezelése

- Az események szignálok, amelyekre szlotokat kell bekötni.
- A QWidget tartalmazza a kész szlotokat virtuális függvényként: paintEvent(), resizeEvent(), stb.
- Ha valamelyik eseményre az alapértelmezettől eltérő implementációt szeretnénk felül kell definiálni az esemény virtuális függvényét.
- Mindegyik metódus rendelkezik egy esemény leíró paraméterrel.

# Eseménykezelő függvények

- paintEvent:
  - Felület / felület egy részének újrarajzolása
- resizeEvent:
  - Átméretezték a felületet.
- keyPressEvent, keyReleaseEvent:
  - Billentyű lenyomása és felengedése
- mousePressEvent, mouseReleaseEvent, mouseDoubleClickEvent:
  - Egér gombjának lenyomása, felengedése, dupla klikk
- mouseMoveEvent:
  - Egér mozgatása

# Kommunikáció a programmal

- A vezérlő elemek tipikusan a szülő ablakkal kommunikálnak. Esetleg a szülő ablakon belül egy másik vezérlővel.
- A vezérlő elem szignálokkal jelzi a felhasználó beavatkozását, állapotának megváltozását.
- Szlotokkal adhat lehetőséget arra, hogy más vezérlőelemek vezérelhessék.

# Rajzolás

- A rajzolást a QPainter osztály segítségével végezzük.
  - Rajz funkciókat szolgáltat. (Pont, vonal, szöveg, stb.)
- A rajzolást általában a vezérlő (QWidget származék) felületére végezzük.
- A rajzolást a paintEvent() függvény implementációjában végezzük.
- A felület frissítését, a paintEvent() meghívását a programból is kiválthatjuk:
  - update – üzenet soron keresztül
  - repaint – közvetlen hívással

# Rajzolás példa

```
void CustomWidget::paintEvent(QPaintEvent *event)
{
  QPainter painter(this);
  QPen pen;
  pen.setColor(Qt::blue);
  pen.setWidth(4);
  painter.setPen(pen);

  QBrush brush;
  brush.setColor(Qt::red);
  brush.setStyle(Qt::SolidPattern);
  painter.setBrush(brush);

  painter.drawRect(0, 0, this->size().width(),
    this->size().height());
}
```

# QPainter rajzolási célok

- QPaintDevice típusú objektumok:
  - QImage
    - HW független kép reprezentáció
  - QPicture
    - Felveszi és visszajátsza a rajzolási parancsokat
  - QPixmap
    - Bittérkép, aminek a tartalmát az ablakba másolhatjuk
  - QPrinter
    - Nyomtató kezelő
  - QWidget

# Átméretezés konfigurálása

- Beállíthatjuk hogyan lehet a vezérlőt átméretezni.

```
void setSizePolicy(QSizePolicy::Policy
horizontal, QSizePolicy::Policy vertical);
```

- horizontal: vízszintes átméretezés szabálya
- vertical: függőleges átméretezés szabálya

# Átméretezési szabályok

| A szabály kódja | Leírás |
|---|---|
| *QSizePolicy::Fixed* | A vezérlő méretét nem lehet megváltoztatni, az mindig az alapértelmezett érték lesz. |
| *QSizePolicy::Minimum* | A vezérlő alapértelmezett mérete a minimumérték, tehát annál kisebbre nem lehet átméretezni. |
| *QSizePolicy::Maximum* | A vezérlő alapértelmezett mérete a maximumérték, tehát annál nagyobbra nem lehet átméretezni. |
| *QSizePolicy::Preferred* | A vezérlő alapértelmezett mérete az optimális méret, de ha szükséges, kisebbre és nagyobbra is lehet állítani. |
| *QSizePolicy::Expanding* | Ha van elég hely, akkor kitölti, de szükség esetén a méretet csökkenteni is lehet. |
| *QSizePolicy::MinimumExpanding* | A vezérlőt kisebbre nem lehet venni, mint az alapértelmezett érték, de egyébként kitölti a teret. |
| *QSizePolicy::Ignored* | A vezérlő alapértelmezett értékét figyelmen kívül kell hagyni, akkora helyet töltsön ki, amennyit csak lehet. |