Linux Programozás

Dokumentum / Nézet architektúra

A probléma

- A GUI-s alkalmazások esetén gyakran egy adathalmazt kívánunk megjeleníteni többféle módon.
- Ha módosítunk az egyik nézetben a többinek is frissülnie kell.
- A keresztbe kommunikációk helyett egységes tárolásra van szükség.

Dokumentum / Nézet architektúra

- Dokumentum
  - Az adatokat tárolja.
  - Az adatok módosulása esetén az összes nézetet értesíti.
  - Tartalmazza a nézetek mutatóinak listáját.
- Nézet
  - Megjeleníti a dokumentum adatait.
  - Lehetőséget ad a módosításra, amelyet a dokumentumnak küld el.
- Alkalmazás
  - Inicializálja a programot.
  - Létrehozza a dokumentumot, a nézeteket és összekapcsolja őket.

Egy / több dokumentum

- Egy dokumentum
  - A program csak egy dokumentumot képes kezelni.
  - A dokumentum mutatóját tartalmazhatja az alkalmazás és a nézet onnan lekérdezheti.
- Több dokumentum
  - Egyszerre több dokumentumot kezel a program.
  - Minden nézethez nyilván tartjuk a hozzá tartozó dokumentum referenciáját.

Qt SDI / MDI

- SDI – Single Document Interface
  - Egy dokumentum-ablakos alkalmazás
- MDI – Multiple Document Interface
  - Több dokumentum-ablakos alkalmazás
- Az SDI / MDI kifejezések csak az alkalmazás felületén megnyitott ablakokra utalnak.
- Egy dokumentumos – több nézetes alkalmazás is lehet MDI a Qt terminológia szerint, ha külön ablakokban vannak a nézetek.

Alkalmazás komponens

- Általában az alkalmazás főablakát tartalmazza.
  - `QMainWindow` leszármazott
  - Menü, eszközsor, státusz kezelés
- Tárolja a dokumentumok listáját.
- Kezeli az állomány betöltés, mentés, bezárás műveleteket.
- Az ablak közepe `QMdiArea` típusú.

Dokumentum komponens

- Feladatai:
  - Adatok tárolása
  - Interfész az adatok lekérdezésére és módosítására
  - Nézetek listájának tárolása, kezelése
  - Ha módosul az adat, akkor az összes nézet értesítése
  - Az adatok kimentése és visszatöltése (segítség: `QTextStream`, `QDataStream`)
- A komponens implementálása a mi feladatunk, nem tartozik Qt osztály hozzá. (`QObject` leszármazott.)

Nézet komponens(ek)

- A nézet osztályokat a közös interfész érdekében egy közös ősből származtatjuk, amelynek virtuális függvényeit a leszármazottakban felülírjuk.
- A közös ős tipikusan `QWidget`-ből származik.
- Feladatai:
  - Referencia tárolása a dokumentumra
  - Adatok megjelenítése
  - Felhasználói események kezelése

Többszálú alkalmazás

Figyelem!

- Figyelni kell, hogy az objektumokat melyik szálban hozzuk létre.
- `QObject` objektumot csak abban a szálban hozhatunk létre, ahol a szülőjét.
- A szál törlése előtt gondoskodnunk kell a szál objektumainak törléséről.
- A GUI elemei mindig a főszálban futnak.

Szál indítása

- A szál létrehozásához szükség van egy `QThread` osztályból származó objektumra.
- A `QThread::run()` függvényt kell implementálnunk.
- Indítás: `QThread::start()`
- Állapot:

```
bool QThread::isFinished() const
bool QThread::isRunning() const
void QThread::started()          // signal
void QThread::finished()         // signal
void QThread::terminated()       // signal
```

Eseménykezelés szálban

- A szignál-szlot mechanizmus szálak között is működik, ahogy korábban láttuk.
  - Ehhez szükséges egy eseménykezelő ciklus.
- Eseménykezeléshez:
  - A `QThread::run()` implementációban meg kell hívnunk a `QThread::exec()` függvényt. (Lásd. `main()`)
- Leállítás:

```
void QThread::quit()
void QThread::exit(int returnCode = 0)
```

(Mindkettő szlot)

Szinkronizációs eszközök

Szinkronizációs osztály	Leírás
QMutex	Kölcsönös kizárást (mutex) megvalósító zár.
QReadWriteLock	Hasonló a mutexhez, de az objektumokhoz való hozzáférés során megkülönbözteti az írási és az olvasási szándékot.
QSemaphore	Szemafort megvalósító osztály.
QWaitCondition	Lehetővé teszi, egy feltétel létrehozását, ennek a bekövetkeztéig egy szál várakozik. A feltételt teljesülését más szálakból tudjuk engedélyezni.

Adatbázis kezelés

Adatbázis kezelés támogatása

- Az SQL adatbázisok elérését a QtSql modul biztosítja.
  - Driver réteg: A különböző backendek kezelését végzi.
  - SQL API: Platform független programozói interfész.
- Használat:
  - `#include <QtSql>`
  - Projekt állományba: `QT += sql`

A hozzáférés lépései

- Felépítjük a kapcsolatot: ehhez szükség van az adatbázis címére és a belépési adatokra.
- Elküldünk az adatbázisnak egy SQL nyelvű utasítást.
- Feldolgozzuk az utasítás eredményét. Ha az utasítás egy lekérdezés, akkor annak visszatérési eredményét soronként tudjuk bejárni és feldolgozni. Egyéb esetben a visszatérési érték egy skalár.
- Lezárjuk az adatbáziskapcsolatot.

Kapcsolat felépítése

```
QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
db.setHostName("localhost");
db.setDatabaseName("qtDatabase");
db.setUserName("dbUser");
db.setPassword("passw");
if (!db.open())
{
   qDebug() << "Cannot connect to database...";
   qDebug() << db.lastError();
   return 1;
}
```

Kapcsolat felépítése

```
QSqlDatabase QSqlDatabase::addDatabase(
    const QString & type,
    const QString & connectionName =
    QLatin1String(defaultConnection))
```

Vezérlőnév	Adatbáziskezelő rendszer
QDB2	IBM DB2
QMYSQL	MySQL
QSQLITE	SQLite (3-as verzió)
QSQLITE2	SQLite (2-es verzió)
QOCI	Oracle Call Interface Driver
QPSQL	PostgreSQL
QODBC	ODBC- (Open Database Connectivity) kompatibilis adatbázis-kezelők, például Microsoft SQL Server

SQL kérés kiadása

```
QSqlQuery selectQuery;
if(!selectQuery.exec("select * from Book"))
{
    qDebug() << "Error with executing query";
}
```

SQL kérés kiadása

- `QSqlQuery` objektummal
- Az SQL kérést megadhatjuk a konstruktorban, vagy az `exec()` függvény paramétereként.

```
bool QSqlQuery::exec(const QString& query)
bool QSqlQuery::exec()
```

Lekérdezés eredménye

```
while(selectQuery.next())
{
    QString title = selectQuery.value(1).toString();
    QString author = selectQuery.value(2).toString();
    qDebug() << author << ": " << title;
}
```

- A legelső és minden további elem lekérdezéséhez a `next()` függvényt használjuk.
- Ha a `next()` visszatérési értéke false, akkor az a lista vége.
- Az érték lekérdezése:

```
QVariant QSqlQuery::value(int index) const
```

QVariant

- Általános típus a cella értékek tárolására.
- A tárolt típus lekérdezése:

```
Type QVariant::type() const
```

- A tagfüggvényeivel megfelelő Qt típusokra konvertálható az érték.
- Ellenőrzés a konverzió előtt:

```
bool QVariant::canConvert(Type t) const
```

QVariant-ban tárolható típusok

Konstans	Adattípus	Leírás
QVariant::Invalid		hibás adat – nincs típus
QVariant::Bitmap	*QBitmap*	kép
QVariant::Bool	*bool*	logikai igaz/hamis érték
QVariant::Date	*QDate*	dátum
QVariant::DateTime	*QDateTime*	dátum és időpont
QVariant::Time	*QTime*	időpont
QVariant::String	*QString*	szöveg
QVariant::Int	*int*	egész szám

Kapcsolat lezárása

- Az adatbázis kapcsolat lezárásához meghívjuk a kapcsolat `QSqlDatabase::close()` függvényét.
- Ha el is akarjuk távolítani a listából, akkor meghívjuk a `QSqlDatabase::removeDatabase()` statikus függvényt.