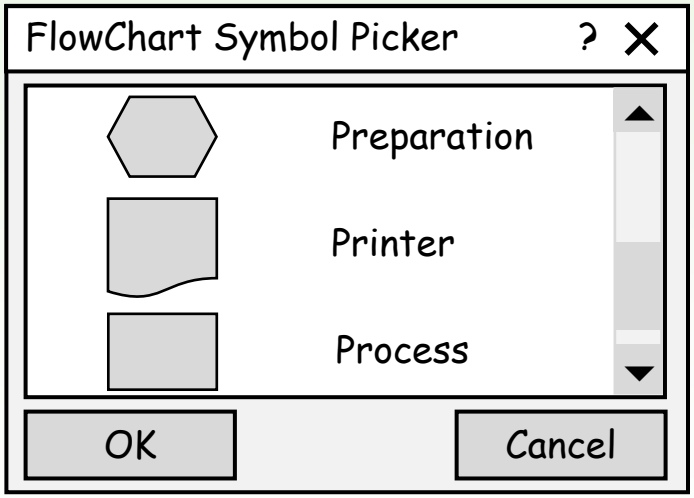
Adatmegjelenítő vezérlők

Adattárolás a grafikus vezérlőben

- Számos grafikus vezérlő tárol adatokat.
  - Ilyen a **`QLineEdit`**, **`QLCDNumber`**, de akár még a **`QLabel`** is.
  - Ugyanakkor vannak kifejezetten adathalmazok megjelenítésre és szerkesztésre specializálódott grafikus vezérlő objektumok (*view widget*), mint például a **`QListWidget`**, **`QTableWidget`**, **`QTreeWidget`**.
- A grafikus vezérlőkben tárolt adatok könnyen elérhetők, és ezek olvasása, módosítása könnyen beilleszthető signál-slot kapcsolatok rendszerébe is.
- A vezérlőben tárolt adat egyből megjelenik a vezérlőn; nem fordulhat elő, hogy mást lát a felhasználó és mást az alkalmazás.
- Mindez tálcán kínálja azt a programozási szokást, hogy a vezérlőinket adattárolásra használjuk.

1.Feladat

Készítsünk olyan dialógus alkalmazást, amely folyamatábra rajzolásához szükséges képelemeket listáz ki, és lehetővé teszi, hogy ezek közül egyet kiválasszunk, amelynek a sorszámát adja majd vissza az alkalmazás.



1.Feladat: tervezés

- Az alkalmazás számára származtatunk egy dialógus osztályt (**`FlowChartSymbolPicker`**), felhelyezünk rá egy lista vezérlőt (**`QListWidget`**) és két nyomógombot (**`QPushButton`**).

FlowChartSymbolPicker *QDialog*
- listWidget : QListWidget - okButton : QPushButton - cancelButton : QPushButton - id : int
+ FlowChartSymbolPicker(Qmap<int, QString>, QWidget) + selectedId() : int { query } + done(int) : void + iconForSymbol(const QString) : QIcon

1.Feladat: megvalósítás

```
#include <QApplication>
#include "flowchartsymbolpicker.h"
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QMap<int, QString> symbolMap;
    symbolMap.insert(132, QObject::tr("Data"));
    symbolMap.insert(135, QObject::tr("Decision"));
    symbolMap.insert(137, QObject::tr("Document"));
    symbolMap.insert(138, QObject::tr("Manual Input"));
    symbolMap.insert(139, QObject::tr("Manual Operation"));
    symbolMap.insert(141, QObject::tr("On Page Reference"));
    symbolMap.insert(142, QObject::tr("Predefined Process"));
    symbolMap.insert(145, QObject::tr("Preparation"));
    symbolMap.insert(150, QObject::tr("Printer"));
    symbolMap.insert(152, QObject::tr("Process"));

    FlowChartSymbolPicker picker(symbolMap);
    picker.show();

    return app.exec();
}
```

1.Feladat: megvalósítás

```
FlowChartSymbolPicker::FlowChartSymbolPicker(
  const QMap<int, QString> &symbolMap, QWidget *parent): QDialog(parent)
{
    setWindowTitle(tr("Flowchart Symbol Picker"));

    _listWidget = new QListWidget;
    _listWidget->setIconSize(QSize(60, 60));

    foreach(QString value, symbolMap){
        QListWidgetItem *item = new QListWidgetItem(value,_listWidget);
        item->setIcon(iconForSymbol(value));
        item->setData(Qt::UserRole, symbolMap.key(value));
    }

    _id = -1;  // nincs kiválasztott elem
    …
}
```

adathordozó listaelem (ikon és szöveg pár) hozzáadása a lista vezérlőhöz

1.Feladat: megvalósítás

```
FlowChartSymbolPicker::FlowChartSymbolPicker(
  const QMap<int, QString> &symbolMap, QWidget *parent): QDialog(parent)
{
      …
    _okButton = new QPushButton(tr("OK"));
    _okButton->setDefault(true);

    _cancelButton = new QPushButton(tr("Cancel"));

    connect(_okButton, SIGNAL(clicked()), this, SLOT(accept()));
    connect(_cancelButton, SIGNAL(clicked()), this, SLOT(reject()));

    QHBoxLayout *buttonLayout = new QHBoxLayout;
    buttonLayout->addStretch();
    buttonLayout->addWidget(_okButton);
    buttonLayout->addWidget(_cancelButton);

    QVBoxLayout *mainLayout = new QVBoxLayout;
    mainLayout->addWidget(_listWidget);
    mainLayout->addLayout(buttonLayout);
    setLayout(mainLayout);
}
```

mindkét esetben lefut a done() metódus is

1.Feladat: megvalósítás

Felüldefiniált metódus: akkor fut le, amikor az accept() vagy a reject().

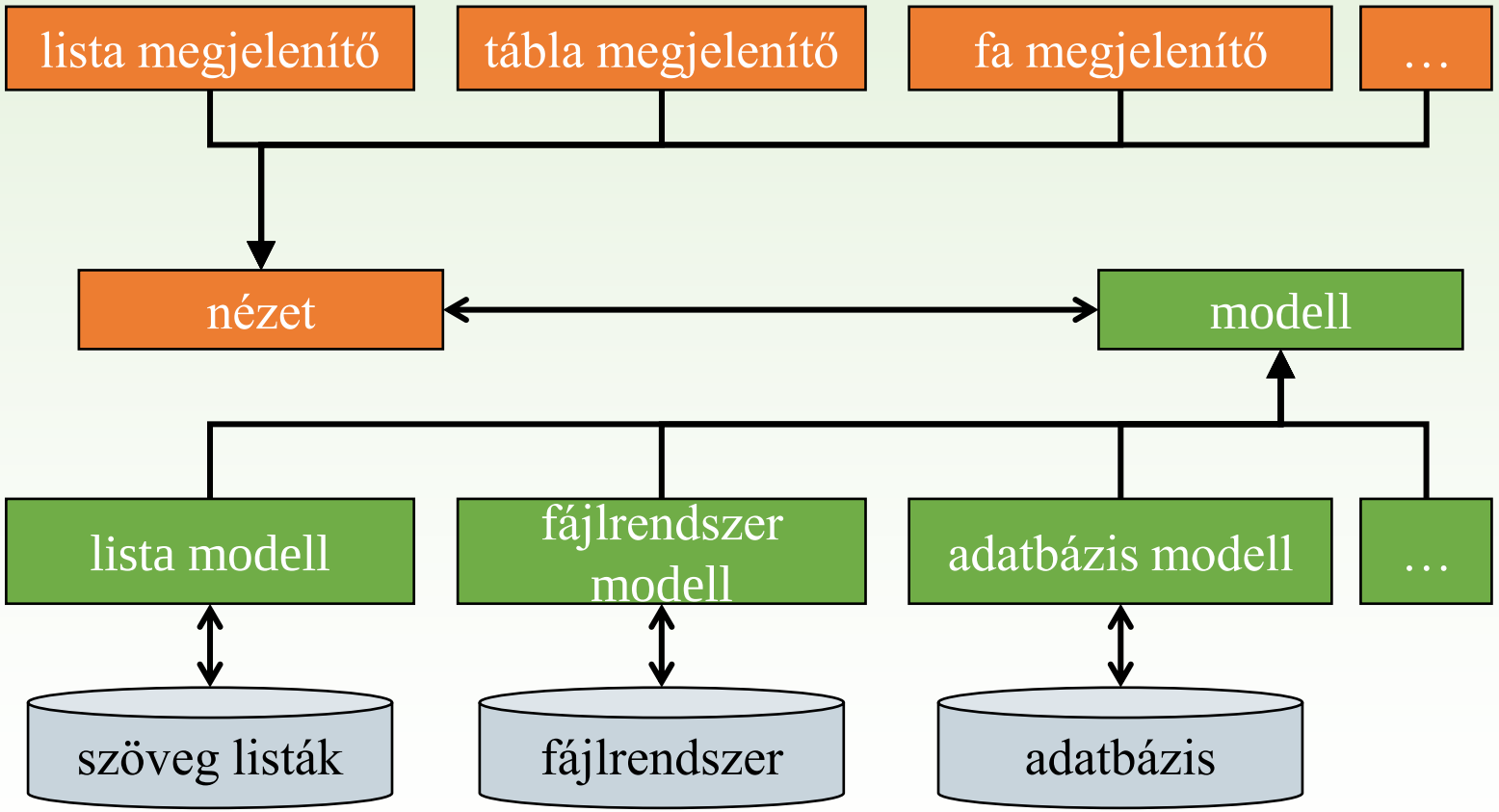
```
void FlowChartSymbolPicker::done(int result)
{
    _id = -1;
    if (result == QDialog::Accepted) {
        QListWidgetItem *item = _listWidget->currentItem();
        if (item) _id = item->data(Qt::UserRole).toInt();
    }
    QDialog::done(result);
}

QIcon FlowChartSymbolPicker::iconForSymbol(const QString &symbolName)
{
    QString fileName = ":/images/" + symbolName.toLower();
    fileName.replace(' ', '-');
    return QIcon(fileName);
}
```

accept() esetén a dialógus megjegyzi a kiválasztott listaelem sorszámát

Adat-modell és nézet elválasztása

- A grafikus vezérlőkben történő adattárolás akkor válik kényelmetlenné,
  - ha ugyanazon adatot több vezérlő is használja egyszerre, vagy
  - ha az adat egy nagyobb, háttérben tárolt adathalmaznak a része.
- Ugyanazon adat többszörös nyilvántartása egyrészt memória pazarlás, másrészt folyamatosan ügyelni kell a konzisztenciára, és szinkronizálni kell ugyanazon adatot tároló objektumokat: ha egyik helyen változik a tárolt adat, akkor a többi helyen is meg kell változtatni.
- Ilyenkor célszerű elválasztani az adatok tárolását (adat-modell) azok megjelenítésétől (nézet), azaz az M/V (*model/view*) architektúrát kell használni.

Adatcsoportok kezelésének és megjelenítésének szétválasztása



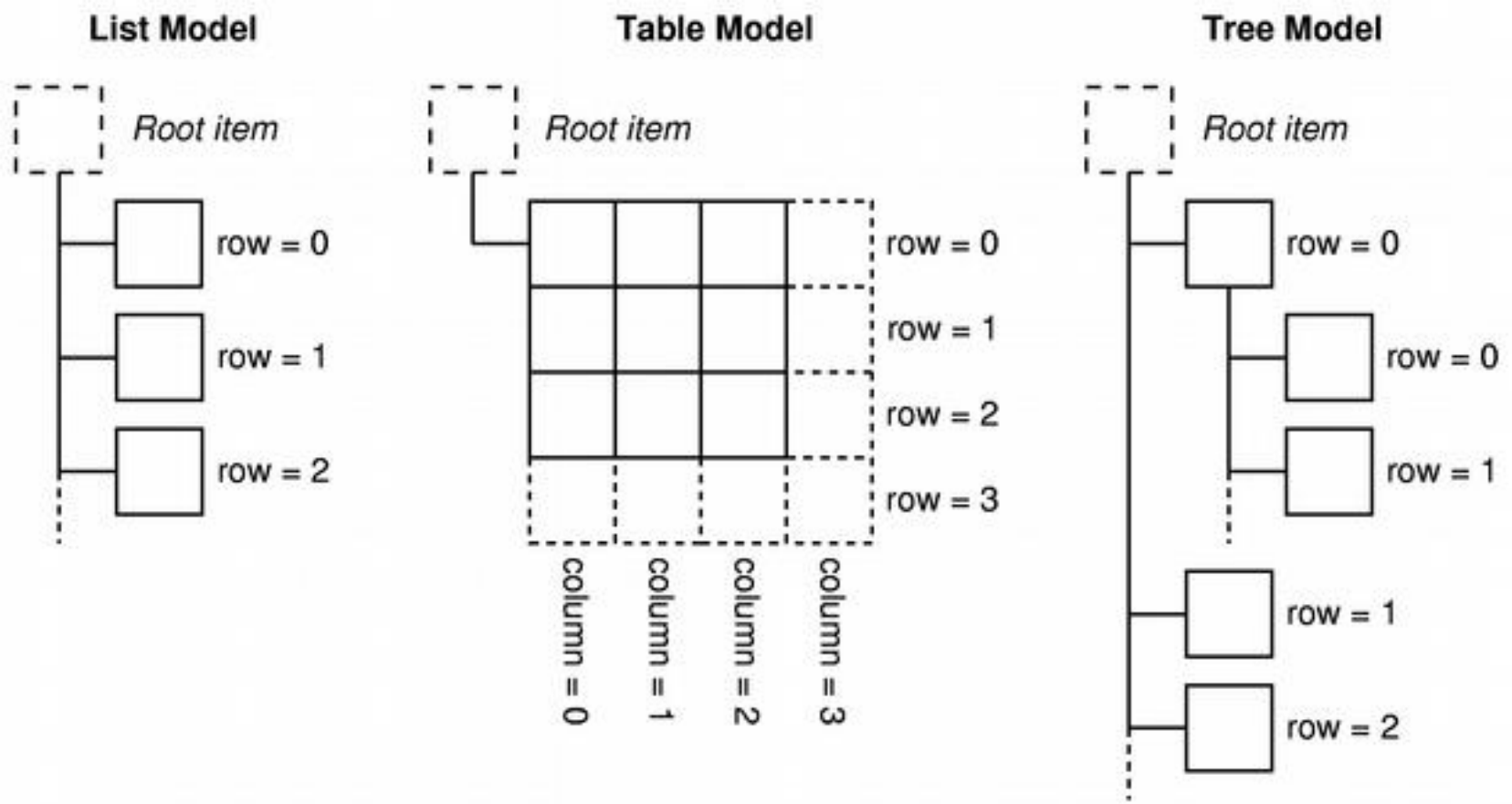
Csoportos adatkezelés és megjelenítés eszközei Qt-ben

- A nézethez a **`QAbstractItemView`** leszármazottjait használhatjuk:
  - **`QListView`**
  - **`QTableView`**
  - **`QTreeView`**
- A modellhez a **`QAbstractItemModel`** leszármazottjait használhatjuk:
  - **`QAbstractListModel`** leszármazottjai (pl. **`QListModel`**)
  - **`QAbstractTableModel`** leszármazottjai (pl. **`QTableModel`**)
    - beállítható az adatelemek kiválasztásának módja:
      - **`setSelectionBehavior()`** – állítja be, hogy egy klikkelés egyetlen elemet, egy teljes sort, vagy oszlopot válasszon-e ki
      - **`setSelectionMode()`** – beállításától függ, hogy egyszeres vagy csoportos kiválasztást engedünk-e meg
  - **`QDirModel`**

Modell elemeinek indexelése

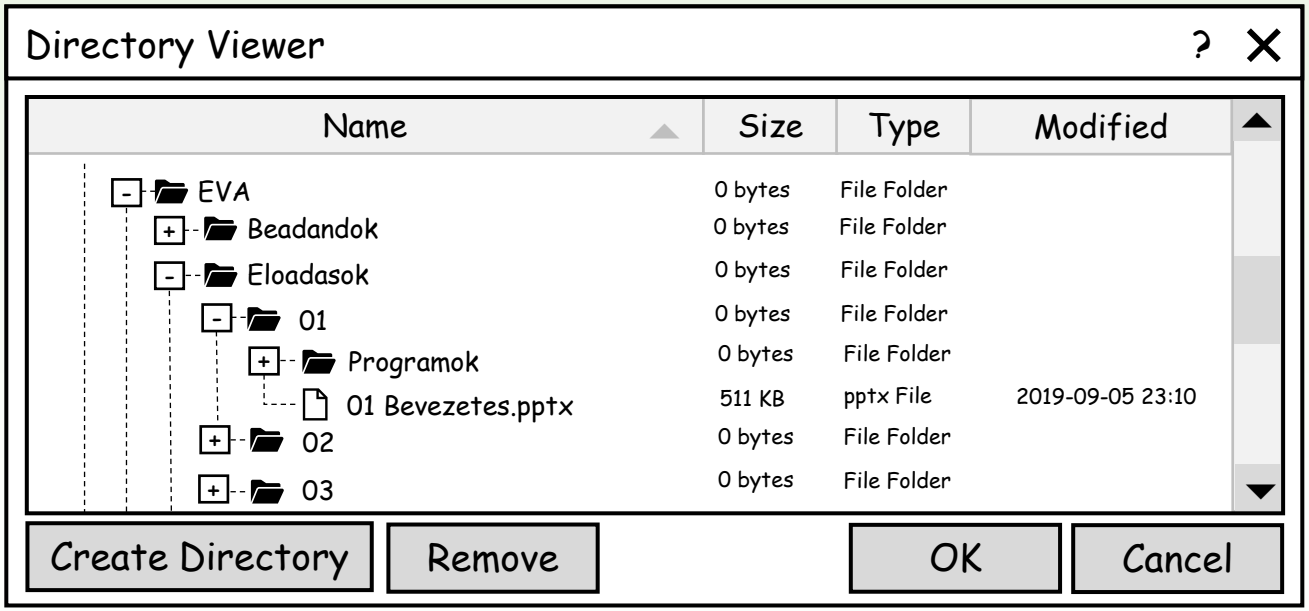
- A csoportos adatok (lista, táblázat, fa) modelljében az adatelemek lokalizálására a modell index objektum (**`QModelIndex`**) szolgál. Ennek
  - **`data()`** metódusa hivatkozik a lokalizált adatelemre;
  - **`row()`** metódusa adja meg az index által lokalizált adat sorszámát;
  - **`column()`** metódusa – táblázatos adattárolás esetén – mutatja az index által meghatározott adat oszlopszámát;
  - **`parent()`** metódusa – fa szerkezetben történő adattároláskor – adja vissza az index által kijelölt adat szülőjének indexét;
  - **`children()`** metódusától kapjuk meg – fa szerkezetben történő adattároláskor – az index által lokalizált adat gyerekeinek listáját (**`QObjectList`**).
- Az indexeket a nézetben is használhatjuk.
  - aktuális elem kijelölése: **`setCurrentIndex(<index>)`**
  - az **`edit(<index>)`** művelettel szerkeszthetővé tehetünk egy elemet, az **`update(<index>)`** frissíti az adott tartalmat.

Modell elemeinek indexelése



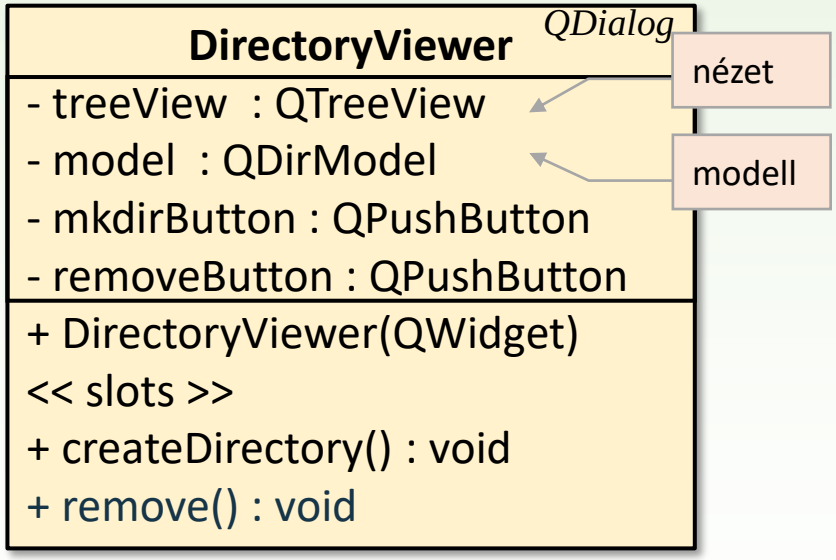
2.Feladat

Készítsünk könyvtárkezelő alkalmazást.



2.Feladat: tervezés

- A megoldáshoz egy dialógus osztályt készítünk (**`DirectoryViewer`**), felhelyezünk rá fanézetet (**`QTreeView`**) és egy könyvtármodellt (**`QDirModel`**), valamint két nyomógombot (**`QPushButton`**).



2.Feladat: megvalósítás

```
DirectoryViewer::DirectoryViewer(QWidget *parent) : QDialog(parent)
{
    _model = new QDirModel;
    _model->setReadOnly(false);
    _model->setSorting(QDir::DirsFirst | QDir::IgnoreCase | QDir::Name);

    _treeView = new QTreeView; treeView->setModel(_model);
    _treeView->header()->setStretchLastSection(true);
    _treeView->header()->setSortIndicator(0, Qt::AscendingOrder);
    _treeView->header()->setSortIndicatorShown(true);
    _treeView->header()->setSectionsClickable(true);

    QModelIndex index = _model->index(QDir::currentPath());
    _treeView->scrollTo(index);
    _treeView->resizeColumnToContents(0);
    …
}
```

modell

nézet

2.Feladat: megvalósítás

```
DirectoryViewer::DirectoryViewer(QWidget *parent) : QDialog(parent)
{
    …

    _mkdirButton  = new QPushButton(tr("&Create Directory..."));
    _removeButton = new QPushButton(tr("&Remove"));
    _quitButton   = new QPushButton(tr("&Quit"));

    connect(_mkdirButton,  SIGNAL(clicked()), this, SLOT(createDirectory()));
    connect(_removeButton, SIGNAL(clicked()), this, SLOT(remove()));
    connect(_quitButton,   SIGNAL(clicked()), this, SLOT(accept()));

    // elrendezők definiálása
    …

    setWindowTitle(tr("Directory Viewer"));
}
```

dialógus ablak beállításai

2.Feladat: megvalósítás

```
void DirectoryViewer::createDirectory()
{
     QModelIndex index = _treeView->currentIndex();
     if (!index.isValid()) return;
     QString dirName = QInputDialog::getText(this,
                              tr("Create Directory"), tr("Directory name"));
     if (!dirName.isEmpty()) {
          if (!model->mkdir(index, dirName).isValid())
               QMessageBox::information(this, tr("Create Directory"),
               tr("Failed to create the directory"));
     }
}
void DirectoryViewer::remove()
{
     QModelIndex index = _treeView->currentIndex();
     if (!index.isValid()) return;
     bool ok;
     if (_model->fileInfo(index).isDir()) { ok = model->rmdir(index); }
     else { ok = _model->remove(index); }
     if (!ok) QMessageBox::information(this, tr("Remove"),
               tr("Failed to remove %1").arg(_model->fileName(index)));
}
```

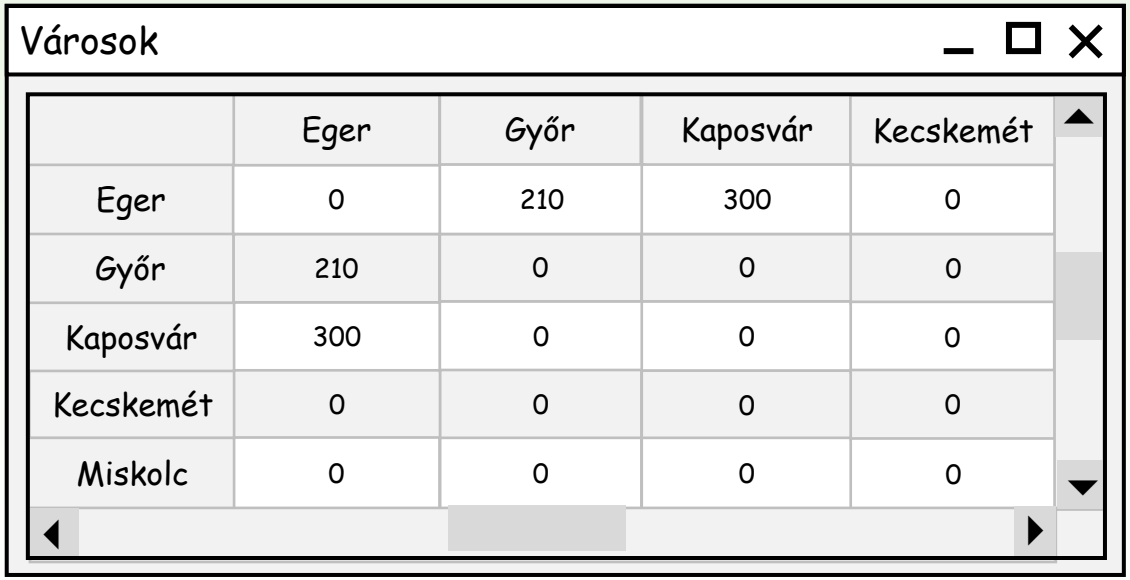
modell-index szerepe

létrehozza az új könyvtárat, ha lehet, különben hibát jelez

törli a kiválasztott bejegyzést: könyvtárat vagy fájlt

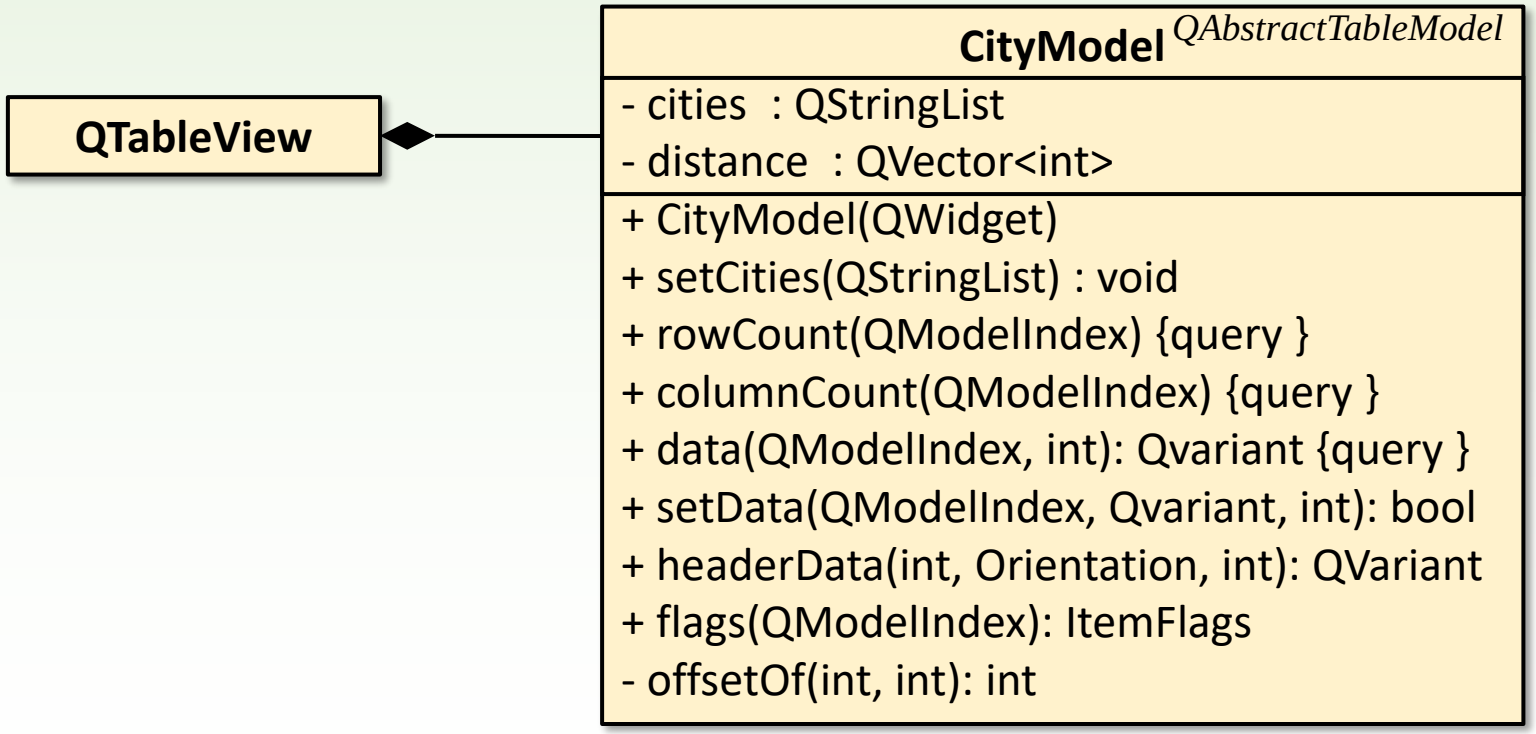
3.Feladat

Készítsünk olyan alkalmazást, amelyikben adott városok távolságait lehet megadni (nézegetni és szerkeszteni).



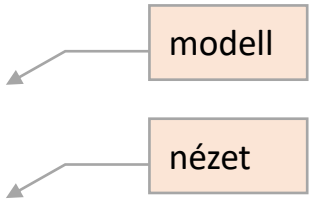
3.Feladat: tervezés

- A megoldáshoz egy speciális táblamodell osztályt származtatunk a **`QAbstractTableModel`** osztályból, amely majd egy **`QTableView`** segítségével jeleníti meg az adatokat.



3.Feladat: megvalósítás

```
#include "citymodel.h"
int main(int argc, char *argv[]) {
    QApplication app(argc, argv);

    QStringList cities;
    cities << "Békéscsaba" << "Budapest" << "Debrecen" << "Eger"
           << "Győr" << "Kaposvár" << "Kecskemét" << "Miskolc"
           << "Nyíregyháza" << ...  << "Zalaegerszeg";

    CityModel cityModel;
    cityModel.setCities(cities);

    QTableView tableView;
    tableView.setModel(&cityModel);
    tableView.setAlternatingRowColors(true);
    tableView.setWindowTitle(QObject::tr("Városok"));
    tableView.show();

    return app.exec();
}
```



3.Feladat: megvalósítás

```
CityModel::CityModel(QObject *parent) : QAbstractTableModel(parent) { }

void CityModel::setCities(const QStringList &cityNames)
{
    _cities = cityNames;
    _distances.resize(_cities.count() * (_cities.count() - 1) / 2);
    _distances.fill(0);
    resetInternalData();
}


int CityModel::offsetOf(int row, int column) const
{
    if (row < column) qSwap(row, column);
    return (row * (row - 1) / 2) + column;
}
```

n város esetén n(n-1)/2 távolságot kell tárolni (alsó háromszög mátrix)

3.Feladat: megvalósítás

```
int CityModel::rowCount(const QModelIndex & ) const
{
    return _cities.count();
}
int CityModel::columnCount(const QModelIndex & ) const
{
    return _cities.count();
}
QVariant CityModel::data(const QModelIndex &index, int role) const
{
    if (!index.isValid()) return QVariant();
    if (role == Qt::TextAlignmentRole) {
        return int(Qt::AlignRight | Qt::AlignVCenter);
    } else if (role == Qt::DisplayRole) {
        if (index.row() == index.column()) return 0;
        int offset = offsetOf(index.row(), index.column());
        return _distances[offset];
    }
    return QVariant();
}
```

felüldefiniálandó metódusok

modell-index szerepe

megjelenítésnél a megfelelő távolságot kell megmutatni

3.Feladat: megvalósítás

A bevitt adatot tárolni kell, és a tábla szimmetrikus elemében megjeleníteni

```
bool CityModel::setData(const QModelIndex &index,
                        const QVariant &value, int role)
{
    if (index.isValid() &&
            index.row()!=index.column() && role==Qt::EditRole) {
        int offset = offsetOf(index.row(), index.column());
        _distances[offset] = value.toInt();
        QModelIndex transposedIndex =
                createIndex(index.column(), index.row());
        emit dataChanged(index, index);
        emit dataChanged(transposedIndex, transposedIndex);
        return true;
    }
    return false;
}
```

modellindex szerepe

tartományt jelöl ki

3.Feladat: megvalósítás

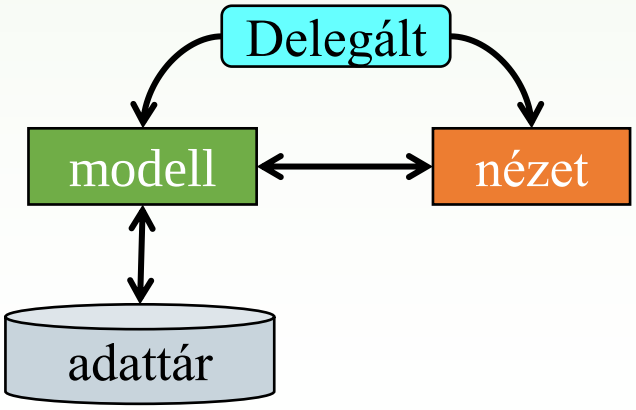
a fejléc megfelelő pozícióján megjelenő adat

```
QVariant CityModel::headerData(int section, Qt::Orientation, int role)
const
{
    if (role == Qt::DisplayRole) return _cities[section];
    return QVariant();
}

Qt::ItemFlags CityModel::flags(const QModelIndex &index) const
{
    Qt::ItemFlags flags = QAbstractItemModel::flags(index);
    if (index.row() != index.column()) flags |= Qt::ItemIsEditable;
    return flags;
}
```

csak a táblázat átlóján kívüli elemek szerkeszthetők

Adatkezelése grafikus vezérlőkben

- A modellben tárolt adatoknak egy grafikus vezérlőben történő megjelenítése, illetve – ha a vezérlő engedi – a szerkesztése az alapértelmezett mód helyett egyedi módon is történhet, feltéve, hogy ennek leírását megadjuk.
- Ezt általában az ún. MVC (*model-view-controller*) architektúra biztosítja. Qt-ben ennek egy olyan változata került bevezetésre, amelynél a modellben tárolt adatok megjelenítésének, illetve szerkesztésének módját ún. delegált (*delegate*) osztályokkal adhatjuk meg.



Egyedi megjelenítés és szerkesztés

- A nézet adatmezőiben a modell adatelemeinek megjelenési és szerkesztési módját a **`QAbstractItemDelegate`**-ből leszármazott *delegált* osztályok biztosítják. Az előre definiált delegált osztályok közül az alap megjelenítést és szerkesztést a **`QItemDelegate`** szolgáltatja.
- Lehetőségünk van saját delegált osztályokat definiálni. (Nyilván az a kényelmes, ha ezt egy megfelelő ősosztályból származtatjuk, amelynek metódusait közvetlenül is használhatjuk, de felül is írhatjuk). A leggyakrabban az alábbi metódusokat kell felülírni:
  - **`paint()`**, **`drawDisplay()`**, **`drawFocus()`**
  - **`createEditor(…)`**, **`setEditorData(…)`**, **`setModelData(…)`**

Megjelenítés egyedivé tétele

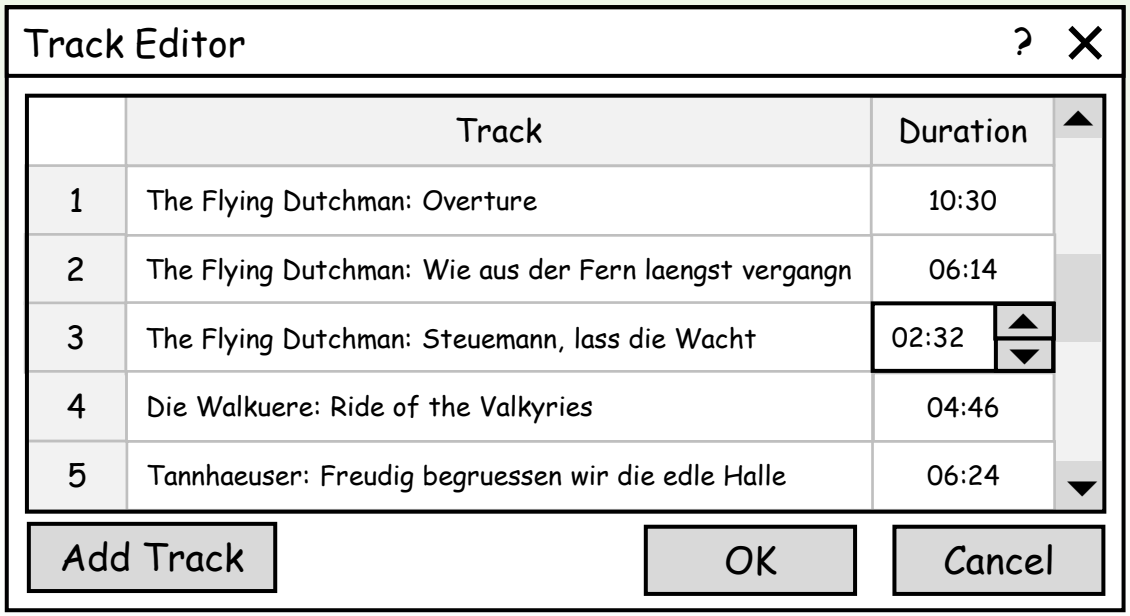
- A **`paint()`** metódus felel az adatelemek értékének kirajzolásáért.
  - Paraméterben kapja meg a kirajzoló objektumot (**`QPainter`**), a kirajzolási stílust (**`QStyleOptionViewItem`**), valamint a kirajzolandó adatot (**`QModelIndex`**).
  - A stílusban adhatjuk meg a méretet, a tagolás és igazítás módját.
- A **`drawDisplay()`** művelettel rajzolhatjuk meg az adatelemet megjelenítő adatmező felületét.
- A **`drawFocus()`** művelet rajzolja meg a fókuszban levő adatmezőt.
- Táblázatok megjelenítését oszloponként is szabályozhatjuk , de hívhatjuk közvetlenül az ősosztályból örökölt műveletet, így az eredeti viselkedést is visszakaphatjuk.

Szerkesztés egyedivé tétele

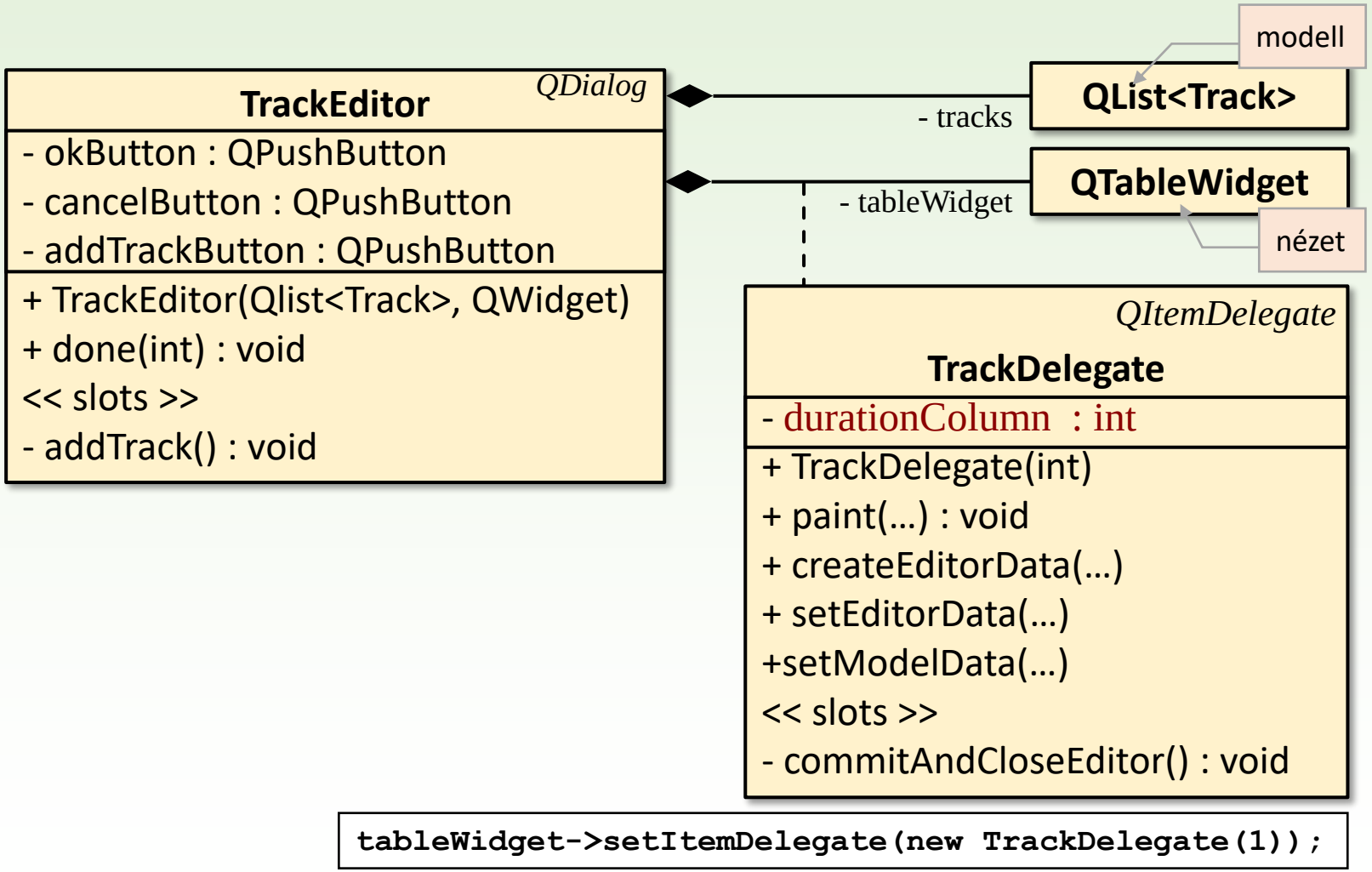
- A **`createEditor(…)`** művelet felelős a szerkesztőmező tetszőleges **`QWidget`**-ként való létrehozásáért, amely akkor jelenik meg az adatelem mezőjében, amikor azt szerkeszteni akarjuk.
- A **`setEditorData(…)`** felelős azért, hogy a szerkesztőmező widget-jében a megfelelő modellbeli adat értéke jelenjen meg.
- A **`setModelData(…)`** felelős a szerkesztőmezőben történt módosítás visszaírásáért a modellbe.
- A szerkesztést táblázat esetén oszloponként is szabályozhatjuk, de hívhatjuk közvetlenül az ősosztályból örökölt műveletet, így az eredeti viselkedést is visszakaphatjuk.

4.Feladat

Készítsünk zenei számokat nyilvántartó alkalmazást úgy, hogy az éppen szerkesztés alatt álló zeneszám idejét egy speciális (QTimeEdit) vezérlővel lehessen megadni.



4.Feladat: tervezés




```
tableWidget->setItemDelegate(new TrackDelegate(1));
```

4.Feladat: megvalósítás

```
#include "trackeditor.h"
int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    QList<Track> tracks;
    tracks << Track("The Flying Dutchman: Overture", 630)
           << Track("The Flying Dutchman: Wie aus der Fern laengst"
                    "vergangner Zeiten", 374)
           << …
           << Track("Tristan und Isolde: Mild und leise, wie er "
                    "laechelt", 375);

    TrackEditor editor(&tracks);
    editor.resize(600, 300);
    editor.show();

    return app.exec();
}
```

inicializálás

```
class Track {
public:
    Track(const QString &title = "",
    int duration = 0)
        : _title(title), _duration(duration) {}
    QString _title;
    int _duration;
};
```

4.Feladat: megvalósítás

```
TrackEditor::TrackEditor(QList<Track> *tracks, QWidget *parent) :
    QDialog(parent)
{
    _tracks = tracks;
    tableWidget = new QTableWidget(tracks->count(), 2);

    tableWidget->setHorizontalHeaderLabels(
        QStringList() << tr("Track") << tr("Duration"));
    for (int row = 0; row < _tracks->count(); ++row) {
        Track track = _tracks->at(row);
        QTableWidgetItem *item0 = new QTableWidgetItem(track.title);
        tableWidget->setItem(row, 0, item0);
        QTableWidgetItem *item1 =
            new QTableWidgetItem(QString::number(track.duration));
        item1->setTextAlignment(Qt::AlignRight);
        tableWidget->setItem(row, 1, item1);
    }
    tableWidget->resizeColumnToContents(0);
    …
}
```

modell

nézet

4.Feladat: megvalósítás

```
TrackEditor::TrackEditor(QList<Track> *tracks, QWidget *parent) :
    QDialog(parent)
{
    …
    _addTrackButton = new QPushButton(tr("&Add Track"));
    _okButton = new QPushButton(tr("OK"));
    _okButton->setDefault(true);
    _cancelButton = new QPushButton(tr("Cancel"));

    connect(_addTrackButton, SIGNAL(clicked()), this, SLOT(addTrack()));
    connect(_okButton,       SIGNAL(clicked()), this, SLOT(accept()));
    connect(_cancelButton,   SIGNAL(clicked()), this, SLOT(reject()));

    // elrendezők definiálása
    …

    setWindowTitle(tr("Track Editor"));
}
```

dialógus ablak beállításai

4.Feladat: megvalósítás

```
TrackDelegate::TrackDelegate(int durationColumn, QObject *parent) :
QItemDelegate(parent)
{
    this->_durationColumn = durationColumn;
}

void TrackDelegate::paint(QPainter *painter, const QStyleOptionViewItem
&option, const QModelIndex &index) const
{
    if (index.column() == _durationColumn) {
        int secs = index.model()->data(index, Qt::DisplayRole).toInt();
        QString text = QString("%1:%2").
           arg(secs/60, 2, 10, QChar('0')).arg(secs%60, 2, 10, QChar('0'));

        QStyleOptionViewItem myOption = option;
        myOption.displayAlignment = Qt::AlignRight | Qt::AlignVCenter;
        drawDisplay(painter, myOption, myOption.rect, text);
        drawFocus(painter, myOption, myOption.rect);
    } else { QItemDelegate::paint(painter, option, index); }
}
```

egy adatelem kirajzolásakor fut le

ha modell-index az időtartamot mutató oszlopra hivatkozik, akkor egyedi megjelenítést alkalmazunk

a többi oszlopra az alapértelmezés áll fenn

4.Feladat: megvalósítás

egy adatelem szerkesztésének kezdetekor :

1. létrejön a szerkesztés vezérlője (createEditor)
2. amelybe aztán megfelelő tartalom töltődik be (setEditorData)

```
QWidget *TrackDelegate::createEditor(QWidget *parent,
    const QStyleOptionViewItem &option, const QModelIndex &index) const
{
    if (index.column() == _durationColumn) {
        QTimeEdit *timeEdit = new QTimeEdit(parent);
        timeEdit->setDisplayFormat("mm:ss");
        connect(timeEdit, SIGNAL(editingFinished()), this,
        SLOT(commitAndCloseEditor()));
        return timeEdit;
    } else { return QItemDelegate::createEditor(parent, option, index); }
}
void TrackDelegate::setEditorData(QWidget *editor,
                                  const QModelIndex &index) const
{
    if (index.column() == _durationColumn) {
        int secs = index.model()->data(index, Qt::DisplayRole).toInt();
        QTimeEdit *timeEdit = qobject_cast<QTimeEdit *>(editor);
        timeEdit->setTime(QTime(0, secs / 60, secs % 60));
    } else { QItemDelegate::setEditorData(editor, index); }
}
```

az időtartam oszlopban egyedi

többi oszlopban alapértelmezett

4.Feladat: megvalósítás

egy adatelem szerkesztésének befejezésekor (enter, fókusz, …) :

1. értesíteni kell a szerkesztést végző vezérlőt (commitAndCloseEditor)
2. frissíteni kell a modellt (setModelData)

```
void TrackDelegate::commitAndCloseEditor()
{
    QTimeEdit *editor = qobject_cast<QTimeEdit *>(sender());
    emit commitData(editor);
    emit closeEditor(editor);
}
void TrackDelegate::setModelData(QWidget *editor,
        QAbstractItemModel *model, const QModelIndex &index) const
{
    if (index.column() == _durationColumn) {
        QTimeEdit *timeEdit = qobject_cast<QTimeEdit *>(editor);
        QTime time = timeEdit->time();
        int secs = (time.minute() * 60) + time.second();
        model->setData(index, secs);
    } else { QItemDelegate::setModelData(editor, model, index); }
}
```

az időtartam oszlopban egyedi

többi oszlopban alapértelmezett