Qt

1. Úvod
2. Historie
3. Qt Creator & qmake
4. Rychlý úvod do Qt
5. Základní koncepty Qt
6. Průvodce prvky GUI
7. Projekty

Úvod

- Ondřej Kolín
- Qt jsem učil 4 roky na SPŠMB
- Neočekávejte technické zázraky, nejsem zkušený programátor
- Něco jsem už napsal, tuším, co jsou to třídy a objekty

Qt

Proč by Vás mělo zajímat?

1. Opensource
2. Velká komunita
3. Aktivní vývoj
4. Dostupné nástroje
5. Multiplatformní, včetně mobilních platforem

Proč by vás nemělo zajímat?

1. Licence
2. Chcete velmi lehkou aplikaci (Qt minimal)
3. Vlastní makra
4. Nekonzistentnost pojmenovávání oproti standardním knihovnám
5. Nechcete psát v C++

Historie

Kdo stojí za Qt?

- 1991 - Trolltech, Haavard Nord and Eirik Chambe-Eng
- Mezitím: Změny licencování, K Desktop Enviroment, podpora FSF
- 2008 - Trolltech koupila Nokia
- 2011 - Nokia prodává komerční licencování společnosti Digia

Qt Creator

- Qt Creator je jedno z nejdotaženějších IDE na Linuxu
- Qt.io

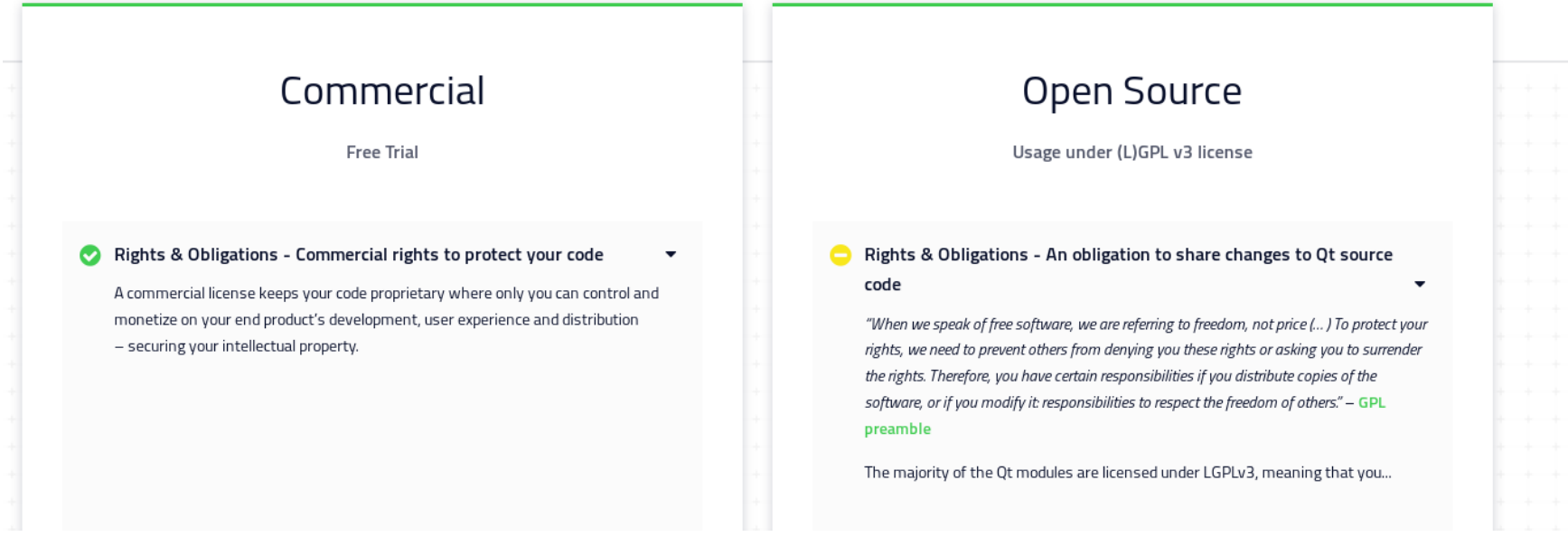
Výhody:

1. Není Java-based
2. Integrovaný Qt Designer
3. *Přehledné*

Nevýhody:

1. Licence
2. Zaměřené na Qt a C++

Licencování Qt Creatoru



Kup si Qt, za 459 $ měsíčně.

Kompilace Qt projektů

Vztah qmake a make

nástroj qmake připraví makefile, ten zkompiluje make, který volá už překladač (respektive sadu nástrojů překladače). Qt Creator qmake soubor bere jako projektový, tzn. soubory .pro jsou soubory Qt Creatoru a projekty jde bez instalace QtCreatoru přeložit.
qmake -> make -> ...

Instalace qmake v odpovídající verzi

```
sudo dnf install qt-devel
```

```
sudo apt install qt5-qmake
```

Dokumentace

```
sudo dnf install qt-doc
```

```
sudo apt install qt5-doc
```

A překladač C++ samozřejmě.

```
sudo dnf install g++
```

```
sudo rpm install gcc-c++
```

Nastavení v Qt Creatoru

Tools -> Options

1. Compilers
2. Qt versions
3. Kits (To je compiler + qmake)

Můžete experimentovat s různými verzemi, přidat platformy (Android, apod.)

Můj první projekt

New project -> Applications -> Qt Widgets Application

Projektový soubor je "qmake soubor"

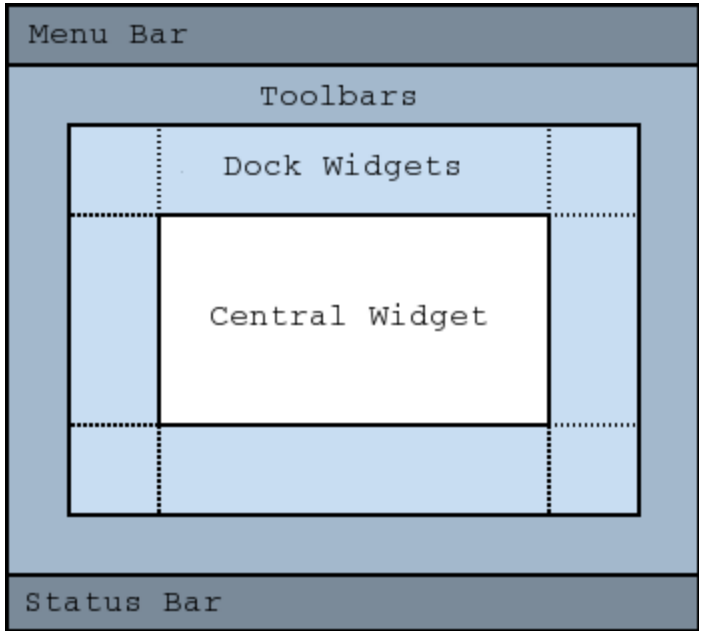
Soubory jsou podle 3 kategorií:

1. Hlavičkové soubory
2. Zdrojové soubory
3. Ui soubory

.ui soubory jsou ve formátu xml a editují se přes Qt Designer

Celá aplikace běží ve smyčce, která zpracovává události uživatelského prostředí.

Třída MainWindow



Jak z ui souboru dostanu svoje prvky?

`mainwindow.ui` -> `ui_mainwindow.h` -> `#include <ui_mainwindow.h>` -> `ui->setupUi()`

Hierarchie

`QMainWindow` -> `QWidget` -> `QObject` & `QPaintDevice`

Widgety

Všechno je widget. Tlačítko, seznam, atd.

Widgety jsou řazeny hierarchicky.

Hierarchie se tvoří pomocí `QWidget *parent`

Smazání rodičovského prvku vede ke smazání potomka, *automatická správa paměti*

`QWidget` definuje obrovskou spoustu věcí, událostí, metody, apod.

Layouty

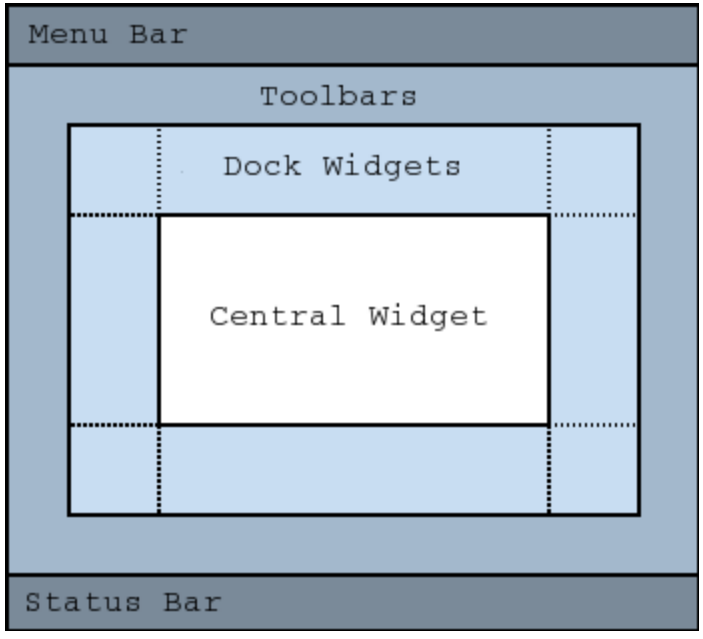
Pokud chci do jednoho widgetu (objekt třídy `QWidget`) vložit více prvků využiju `setLayout(QLayout *layout)`. Layouty umožňují různé typy rozložení.

V layoutu typicky chceme definovat chování velikosti (roztahuje se, neroztahuje), min. rozměry a další.

Pár poznámek k Widgetům

`centralWidget` bývá typicky obyčejný objekt třídy QWidget s nastaveným layoutem.

Pokud si tam dáte něco jiného, tak se Vám stane, že změny v Qt Designeru se neprojeví, protože se logicky nemají kam vložit (`QPushButton` nemůže mít děti)



QObject

... je defacto všechno co interaguje. V kombinaci s makrem `Q_OBJECT` slouží k implementaci interakce pomocí signálů a slotů.
(česky signálů a zdířek)

Signály a sloty

Každý Widget a obecněji objekt mají definované události. Při vyvolání události dojde ke spuštění všech propojených metod. Propojování se používá funkce `QObject::connect`

```
void sayHello() {
  label->setText("Hello. Is that me you are looking for?");
}

//konstruktor
MainWindow::MainWindow (QWidget *parent = nullptr) {
  QPushButton *tlacitko = new QPushButton("Push me hard!");
  connect(tlacitko, &QPushButton::clicked, this, &MainWindo
}
```

Můžeme samozřejmě spojovat i složitější signály, třeba ty co něco vrací, musí odpovídat parametry. Respektive, nemůže slot chtít nějaké parametry, které signál nedává.

```
  connect(posuvnik, &QSpinBox::valueChanged, popisek, &Label::s
```

Qt Designer

Polografický přístup

Qt Designer umí automaticky vytvořit sloty pro události. Stačí klepnout pravým na prvek `Go to slot` a vybrat událost.

Pozor, že při mazání musíte odebrat deklaraci i z hlavičkového souboru.

Plně grafický přístup

Defacto simuluje to, co jsme dělali v příkladu s posuvníkem. Přepnete si na režim editace signálů a slotů (malinké tlačítko v Qt Designeru, nebo klávesa `F4`) a spojováním ve stylu *drag'n'drop* spojujete prvky. V tabulce vybíráte reakce na události.

Jak zjistím, kdo vyvolal událost?

`QObject::sender()`

Použití:

```
QPushButton *zdrojove_tlac = static_cast<QPushButton *>(QObj
```

Poznámky

Qt implementuje spoustu vlastních tříd. Třeba QString pro práci s řetězci. QStringy se dají spojovat pomocí operátoru +

Převod číslo <-> řetězec

```
QString retezec = QString::number(42);
int cislo = retezec.toInt();

QString banner = "LinuxDays" + QString::number(letosni_rok);
```

Pro výstup používáme ladění z `qdebug.h`

Příklady výstupů:

```
qdebug() << "Streaming out";
qwarning() << "Screaming out";
```

Kvízová otázka

Jaký bude výsledek v tmp? A proč právě `AhojA`?

```
int cislo = 65;
QString tmp = "Ahoj";
tmp = tmp+cislo;
```

Základní widgety

QPushButton

Prostě tlačítko.
(K prvkům vytvořeným přes Qt Designer přistupujeme typicky přes `this->ui->tlaciko->` ...)

Zajímavé signály:

```
clicked(bool checked = False);
```

Zajímavé vlastnosti

Vlastnosti (Attributes) se nastavují typicky přes `setAttribute(hodnota)` a získávají přes `attribute()` není tam get.

```
QString text;  // Zobrazeny popisek
```

QLabel

Prostě jednořádkový text

Zajímavé signály:

Skoro žádné, kromě těch zděděných od `QObject` a pár navíc

Zajímavé vlastnosti

```
QString text; // Zobrazeny text, setText(QString text), QStri
```

Zajímavé metody

```
void QLabel::setNum ( double num ); //Nastavi cislo jako text
```

QLineEdit

Jednořádkový vstup pro text

Zajímavé vlastnosti

```
QString text; // Zobrazeny text, setText(QString text), QStri
```

Zajímavé signály:

```
textChanged();
returnPressed();
editingFinished();
...
```

QSpinBox a QDoubleSpinBox

Abychom nezapomněli na čísla:

```
setValue(int/double value) //Nastavuje hodnotu ve spinboxu
int/double value() // bere hodnotu
```

A spoustu zajímavých metod pro nastavování minima, maxima, kroku, apod.

Zajímavé metody

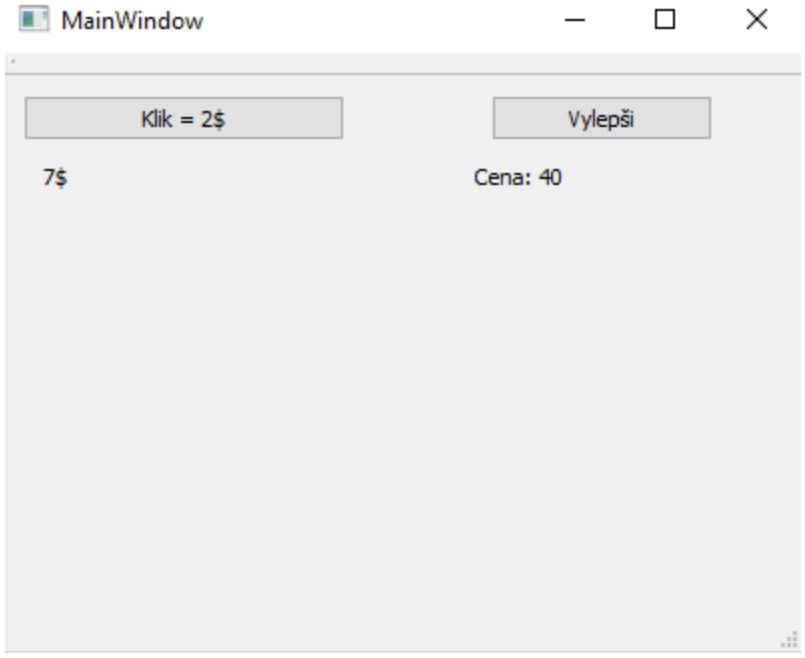
Spoustu, umí nastavit vzhled, skrývání znaku jako u hesla, maximální délku, předvyplněný text ...

Čas na základní příklady

Cookie clicker

Tipy zadarmo:

1. Proměnné dělejte jako členské proměnné třídy hlavního okna, možná to není nejčistší přístup, ale je to nejjednodušší
2. Udělejte tlačítko pro zlepšení klikače
3. Pokud chcete přidat něco, co vydělává automaticky, tak se mrkněte níže na `QTimer`. Nebo spíš do dokumentace, jako správný vývojář.


MainWindow
Klik = 2$
Vylepši
7$
Cena: 40

Kalkulačka

Tipy zadarmo

1. Dokud uživatel mačká čísla, tak to jako řetězec spojujte v nějakém displeji.
2. Jak zmáčkněte něco jiného, zkontrolujte, jestli nemáte už něco uložené, pokud ano, tak to spočítejte a vyplivněte na displej.
3. Chcete, aby na displej šli psát složitější výrazy i se závorkami? Někdo to bude muset parsovat. To už není tak snadné, bude to založené na hledání operátoru, který není v závorkách a rekurzi. Zkuste nad tím pobádat, středoškoláci na to po pár nápovědách a roční přípravě přišli (teda pár z nich)


MainWindow
133
1
2
3
+
4
5
6
-
7
8
9
*
Clear
0
=
/

Víceřádkové widgety

QTextEdit, QTextBrowser

liší se tím, který jde upravovat. Oba umí zobrazovat HTML tagy.

Jak něco přidat?

```
setText(QString text);  // čistě technicky nic nepřidává, ale prostě nastavuj
setHtml(QString text);  // jako setText, ale pokusí se interpretovat HTML
appendText(QString text);  // Připojí text s novým řádkem
appendHtml(QString text);  // lautr to samé, ale interpretuje HTML
```

Jak něco získat? Teda jak všechno získat?

```
QString toPlainText();  // dej text
QString toHtml();  // dej kompletní html kód
```

A co trochu komplexnější widgety?

QComboBox

Znáte z HTML jako `<select>`. Vybíráte prvek z předem dané množiny. Jak do něj nasypat data?

```
QComboBox *combik = new QComboBox();
combik->addItem("Prvek");
QStringList polozky; // Tohle je QList<QString>
polozky << "Prvek taky" << "A za nim dalsi";
combik->addItems(polozky);
```

Kdybych chtěl, můžu vložit prvek na konkrétní místo.

```
insertItem ( int index, const QString & text, const QVariant &
```

a nebo nastavit konkrétnímu text

```
setItemText ( int index, const QString & text )
```

trochu zajímavější je třeba to, že prvkům můžete nastavit nějaká "skrytá data". Třeba, že zobrazujete text, ale máte u nich přiřazenou hodnotu (v html atribute `value`). Typicky třeba ID v databázi

```
setItemData ( int index, const QVariant & value, int role = Qt
```

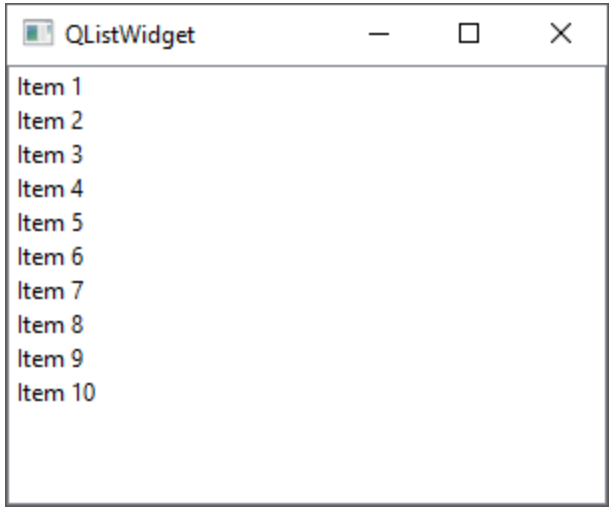
Chceme něco číst z QComboBox?

```
int currentIndex () const
QString currentText () const
```

QListWidget (item-based)

Tohle už je zajímavější, nějaký seznam reprezentovaný jako položky, které je možné mazat, přesouvat, apod. Také znáte, typicky s ikonkami.

Chci si třeba udělat seznam lidí, co můžou na oslavu.



http://www.poketcode.com/img/pyqt4/qlistwidget.png

Tak jdeme na to.

Přidávání, nastavování prvků

```
void    insertItem ( int row, QListWidgetItem * item )
void    insertItem ( int row, const QString & label )
void    addItem ( const QString & label )
void    addItem ( QListWidgetItem * item )
void    addItems ( const QStringList & labels )
void    insertItem ( int row, QListWidgetItem * item )
void    insertItem ( int row, const QString & label )
void    insertItems ( int row, const QStringList & labels )
```

Koho máme vybraného, co je na řádku x?

```
QListWidgetItem * currentItem () const
int  currentRow () const
```

Chci někoho odebrat?

```
int radek = ui->seznam->currentRow();
if(radek < 0) return; //-1 dostáváme při nevybraném řádku
QListWidgetItem *polozka = ui->seznam->takeItem(radek);
delete polozka;
// Tady musím paměť uvolnit ručně, nebo bychom třeba mohli přesunout prvek
```

Tady to je trochu zajímavé, máme dokonce vlastní třídu pro prvky, objektům této třídy můžeme nastavovat některé vlastnosti (nebo je číst samozřejmě), jako:

```
setText(QString text);
setTooltip(QString text);
setData ( int role, const QVariant & value )
```

Tabulky

Tabulkový widget je QTableWidget. Pracuje podobně jako QListWidget, ale má o rozměr více. :-). Mrkněte se do dokumentace.

Kreslení

Začneme tvořit grafiku skoro jako pastelkami!

Qt má dvě třídy, které se používají při vykreslování grafiky.

QGrahpicsView && QGRaphicsScene

`QGrahpicsView` - Pohled. Něco, skrze co se koukáme. Něco jako internetový prohlížeč. Definuje, co chceme zobrazit a jak se na to budeme koukat.

`QGRaphicsScene` - Samotná scéna. Vlastně stejně jako v divadle. Na scéně se něco děje a váš pohled (`QGraphicsView`) definuje, co vidíte.

Je vhodné si typicky uchovávat ukazatel na scénu. Takže já si v `MainWindow.h` často udělám

```
QGraphics *scene;
```

No a v konstruktoru už si nějakou tu scénu vytvořím a přiřadím do `view` vytvořené v návrháři

```
scene = new QGRaphicsScene();
ui->view->setScene(scene);
```

Je libo čtverec, čáru, kružnici nebo snad polygon?

```
scene->addLine(100, 200, 300, 400); //Objekty jde přidávat přímo p
//Nebo jako objekty
QGraphicsEllipseItem *tmp = QGraphicsEllipseItem(100, 100,
scene->addItem(tmp); //Nic se nebojte, ze ukazatel ztratite, elipsu po vas
Pokud se vam ukazatel na pridany objekt hodi, funkce vam h
QGraphicsItem *rect = scene->addRect(0, 0, 50, 50);
rect->setX(100); //Presun ctverec na X souradnici 100
//QPoint je bod
QPoint a(20, 50), b(200, 100);
QPointF c(60.5, 60); //QPointF taky, ale můžu mu dát desetinná čísla be
QPolygonF trojuhelnik;
trojuhelnik.append(a); //Pripojime body
trojuhelnik.append(b);
trojuhelnik.append(c);
//Namichame vypln a obrys
QPen obrys(Qt::blue); //Qt ma nejake konstanty pro barvy pripravene
//Vypln bude strafkovana
QBrush vypln(Qt::red, Qt::FDiagPattern);
//Tohle cele pouzijeme pri vkladani polygonu
scene->addPolygon(trojuhelnik, obrys, vypln);
```

Časovače

Pokud chcete nějakou činnost vyvolat za nějakou dobu, nebo v nějakém intervalu. Ukazatel na časovač se hodí mít někde snadno přístupný, třeba jako členskou proměnnou třídy `QMainWindow`.

Tady je příklad (teda bude)

`mainwindow.h`

```
QTimer *casovac;

public slots:
  void allHailToTheHypnotoad();
```

`mainwindow.cpp`

```
MainWindow::MainWindow(QObject * parent = nullptr)
{
  casovac = new QTimer(this); //QTimer je potomek QObject, takze
  connect(casovac, &QTimer::timeout, this, &MainWindow::al
  casovac->setInterval(100);
  casovac->start();
}

MainWindow::allHailToTheHypnotoad()
{
  qDebug() << "Všichni máme rádi hypnožábu a to Každých 100 ms";
}
```

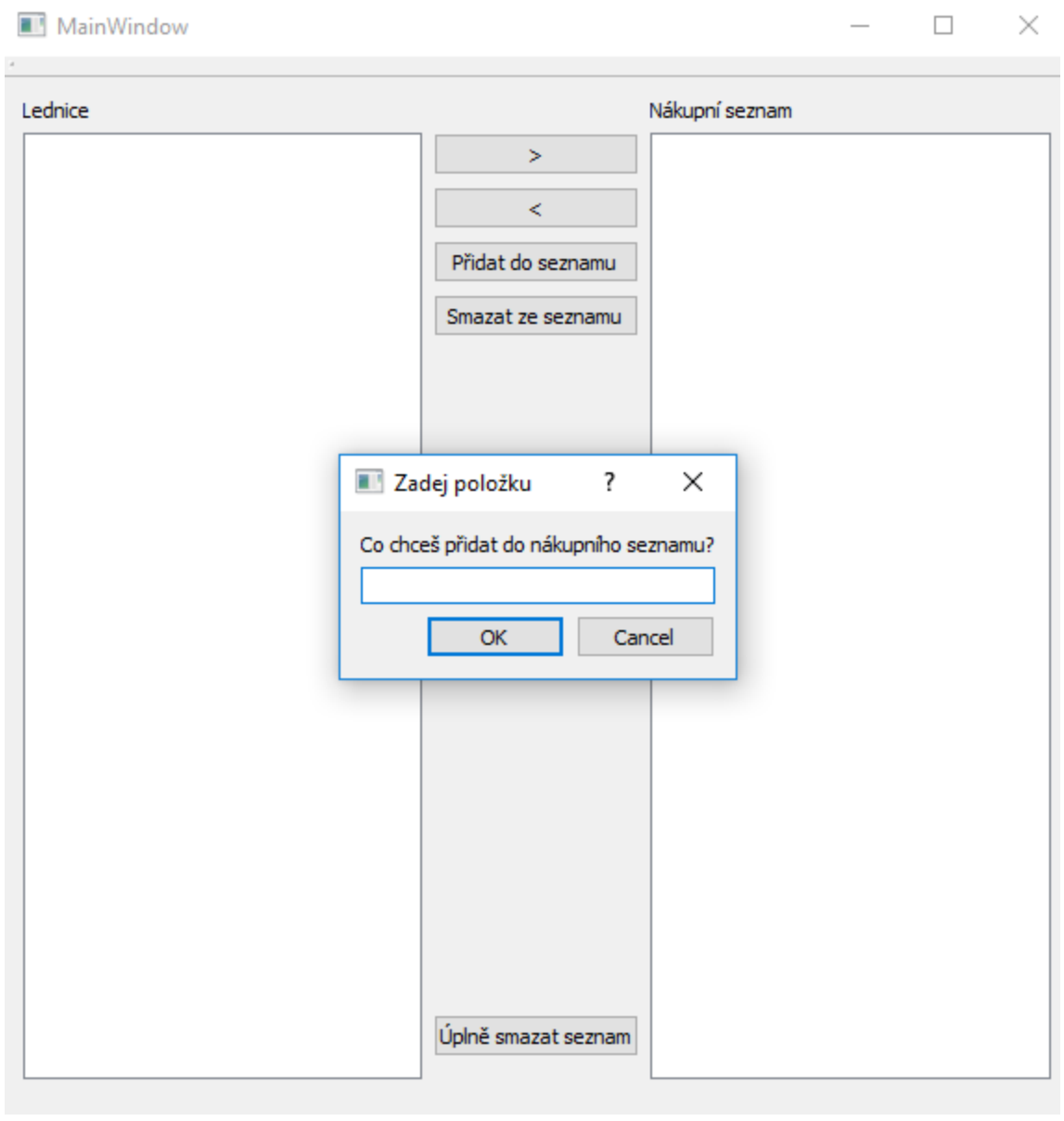
Oboje je redakčně kráceno, šetříme místem

Čas úkolů podruhé

Hlídač obsahu lednice a vytvářeč nákupního seznamu

Může se hodit

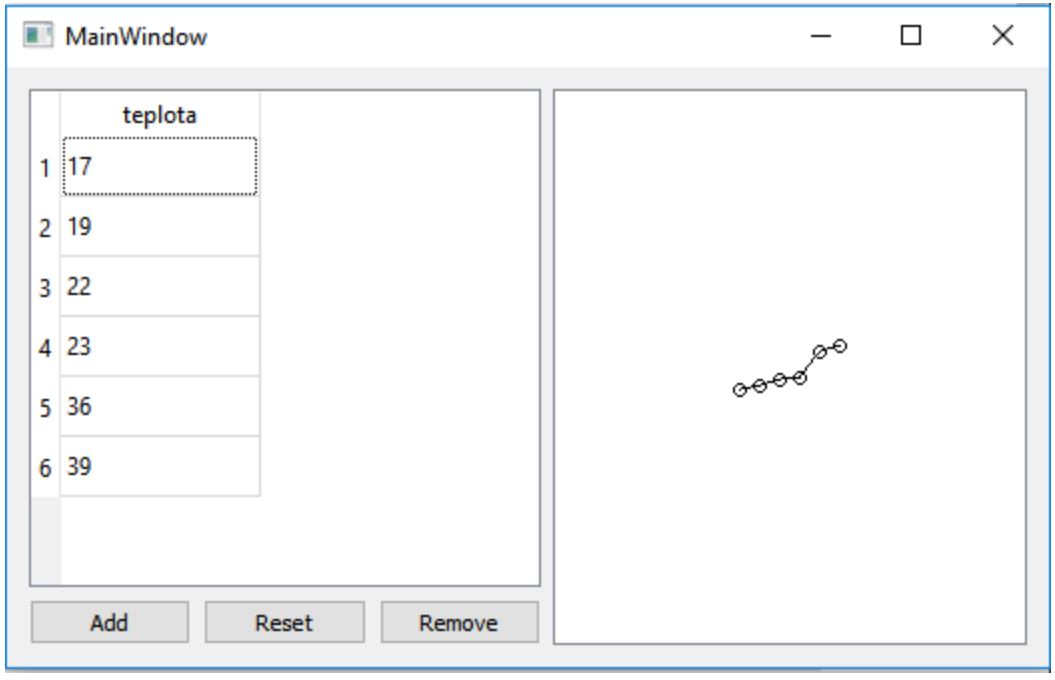
1. `QListWidgetItem * takeItem(int row)` sice odebere prvek na jedné straně, ale nesmaže ho, takže ho můžete přehodit jinam.
2. Když z QListWidget uděláte QTableWidget, tak můžete hlídat i množství.


MainWindow
Lednice
Nákupní seznam
>
<
Přidat do seznamu
Smazat ze seznamu
Zadej položku
?
Co chceš přidat do nákupního seznamu?
OK
Cancel
Úplně smazat seznam

Meteostaniční graf

Může se hodit

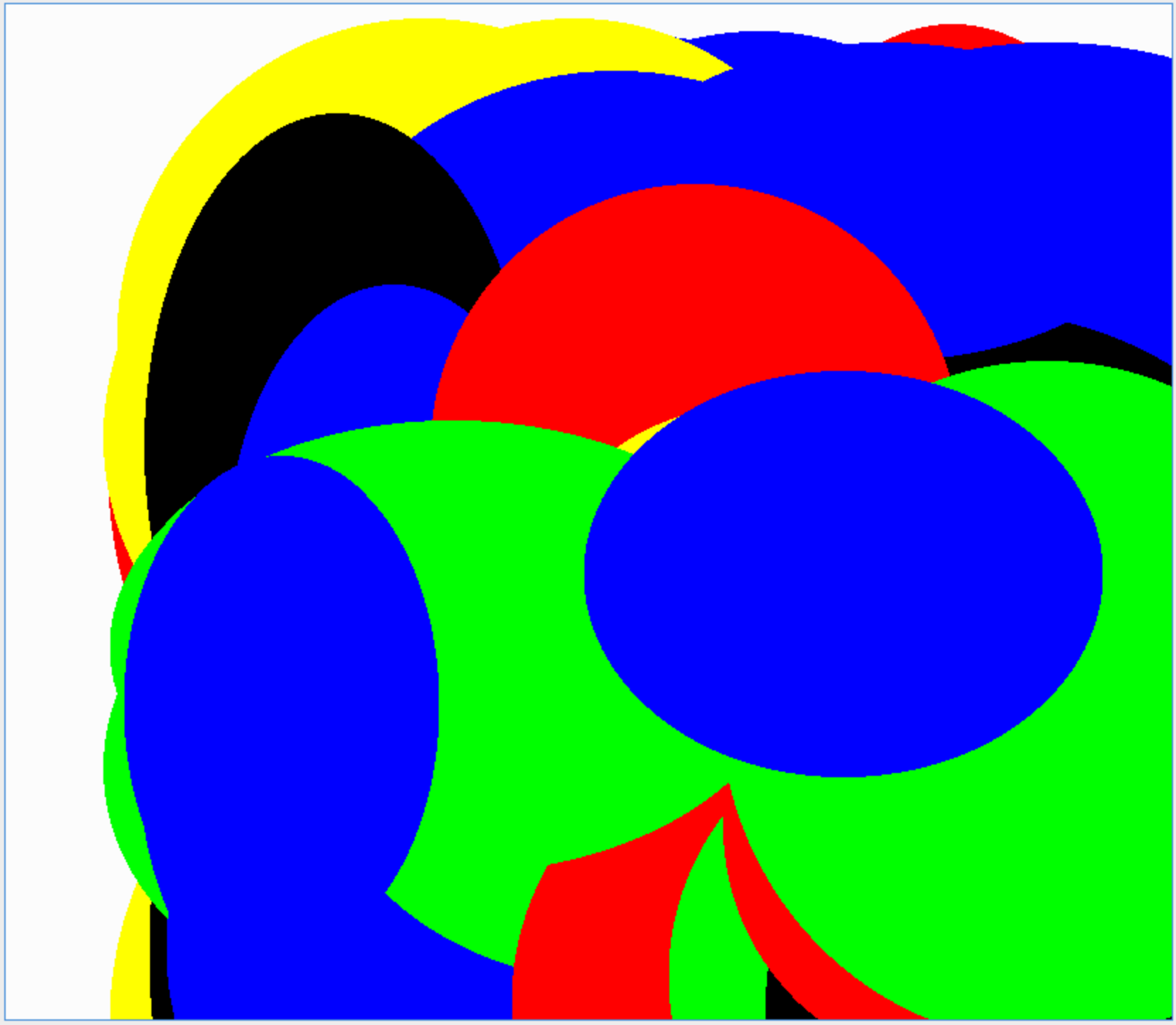
1. `setSceneRect(QRectF)` určí jaké rozměry má `QGraphicsView` přednostně zobrazovat a nebude se snažit scénu centrovat
2. Při jakékoliv změně v datech překreslete graf (udělejte si na to funkci). Můžete zkusit prvky přesouvat po změně, ale doporučuji je spíše rovnou mazat.
3. Udělejte to přes QTableWidget a můžete mít víc grafů vedle sebe, teda nad sebou a pod sebou.
4. A nezapomeňte na osy (obrázek je vytvořený lenochem)


MainWindow
teplota
1
17
2
19
3
22
4
23
5
36
6
39
Add
Reset
Remove

Náhodný kreslič

Může se hodit

1. `qsrand()` vám připraví generátor náhodných čísel
2. `qrand()` vygeneruje náhodné číslo
3. `setSceneRect(QRectF)` určí jaké rozměry má `QGraphicsView` přednostně zobrazovat a nebude se snažit scénu centrovat



Kam dál?

Tady jsme u mě v souboru na řádku 533, což se mi zdá jako dost, je na čase, abyste začali pracovat sami. Tady je pár tipů, nad čím přemýšlet dál.

Databáze

Gui je sice fajn, ale je super k prezentaci nějakých dat. Databáze data už podle jména jde k sobě.

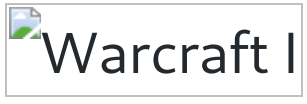
Připojit databázi (`QSqlDatabase`) a posílat na ní dotazy (`QSqlQuery`) je prkotina, s SQLITE si jde moc hezky pohrát.

Qt umí pracovat s modely. To je pojem známý asi hlavní webařům. Mrkněte se na to, teoreticky jde tabulku s GUI propojit na pár řádcích přes `QSqlTableModel`

Kreslení lvl 2

Kreslit změť koleček a čárek je sranda, ale co takhle komplexnější obrazce? Vytvořte si třídu odvozenou od `QGraphicsItem` (musíte implementovat), zachytávejte události klávesnice, využijte časované události `QTimer` a může vzniknout třeba i hra ;).

Ačkoliv dělat hru v Qt může jenom sebevrah. (Zdravíme do Mladé Boleslavi pana Kamila K. autora klonu Wacraft I v Qt).


Warcraft I
;

Multimédia

Je libo vlastní přehrávač? VLC nemůže přeci mít každý. Zkuste nový KarelNovakPlayer©

Multimediální soubory je ve `QGraphicsView` přes `QGraphicsScene` a `QGraphicsVideoItem` možné renderovat.

JSON Rest API

Stáhnout data z internetu, zpracovat je a vykreslit třeba předpověď na 3 dny dopředu. Na trhu s aplikacemi je přeplněno, ale to nevadí, můžete přidat svojí

`QNetworkManager` se bude starat o spojení se světem a když odpověď proženete přes `QJsonDocument` a získáte potřebná data, můžete vykreslovat sluníčka, mráčky, nebo blesky podle toho, jaké bude počasí

Android a jiné platformy

Qt jde přeložit i pro Android, vyžaduje to nainstalovat ARM překladač, telefon, či emulátor pár úprav kódu, zapřemýšlení nad ergonomií mobilního telefonu, ale jde to!

Web assembly

Qt běží i na webu přes projekt WebAssembly. Wow. Ještě jsem to teda nezkoušel, pouze demíčka (a fungují)

Díky za pozornost

https://facebook.com/ondrej.kolin.5
https://twitter.com/o_kolin
ondrej.kolin@gmail.com

Mějte se skvěle!