Pokročilé programování v jazyce C pro chemiky

(C3220)

Knihovna Qt – část 2

Program rozdělený do několika souborů

- Zdrojový kód programů v C++ obvykle rozdělujeme do několika souborů tak, aby každá větší třída byla umístěna v samostatném souboru **.cpp*
- Název souboru volíme tak, aby bylo zřejmé, jakou třídu obsahuje; obvykle se shoduje s názvem třídy (např. *application.cpp*, *graphicwidget.cpp*)
- Ke každému souboru **.cpp* vytvoříme hlavičkový soubor se stejným jménem, ale koncovkou *.h*
- Funkci `main()` je vhodné umístit do samostatného souboru *main.cpp*
- Nepoužíváme-li Qt, pravidla pro překlad jednotlivých souborů umístíme do souboru *Makefile*

Kompilace složitějších Qt programů

- Při práci s knihovnou Qt je *Makefile* generován automaticky, takže ho ručně nevytváříme ani neupravujeme.
- Jednou při založení projektu:
  - Vygenerujeme soubor projektu (`qmake -project`), ten upravíme dle potřeby (např. přidáním `QT += widgets` apod.)
- Při změně souboru .pro nebo při přidání/odebrání `#include` v jakémkoli `.cpp` souboru:
  - Přegenerujeme *`Makefile`* příkazem `qmake`
- Při přidání nového `.cpp` souboru:
  - a) ručně upravíme `.pro` soubor a přidáme *`novysoubor.cpp`* na řádek SOURCES, nebo
  - b) vytvoříme soubor `.pro` znovu (`qmake -project`), nesmíme ho ale zapomenout znovu upravit dle potřeby (`QT += widgets` atd.)
  - V obou případech pak aktualizujeme *`Makefile`* příkazem `qmake`
- Po jakékoli změně překompilujeme projekt příkazem `make`

Vložení hlavičkových souborů

- Hlavičkové soubory vkládáme na začátek souboru *.*cpp* pomocí direktivy preprocesoru #include
- Jako úplně první vkládáme vždy hlavičkový soubor příslušející k aktuální třídě
- Dále vkládáme hlavičkové soubory dalších součástí našeho programu, které uvádíme v uvozovkách "" (kompilátor je bude hledat v adresáři obsahujícím soubor *.cpp)
- Následují hlavičkové soubory knihoven (Qt, ...) a standardní knihovny C++, vše v lomených závorkách <>
- Možné je i opačné pořadí, od nejobecnějších souborů (systémových) k nejspecifičtějším (z aktuálního programu), i tehdy však hlavičkový soubor aktuální třídy patří na první místo (tím zajistíme jeho nezávislost na dalších hlavičkách v *.cpp souboru)
- V každé skupině řadíme hlavičkové soubory abecedně (pro snadnou orientaci)
- Vkládáme vždy pouze hlavičkové soubory těch tříd, které v daném souboru používáme

```
/******************  Soubor  application.cpp   ******************/
#include "application.h"

#include "graphicwidget.h"

#include <QApplication>
#include <QHBoxLayout>
#include <QPushButton>
#include <QVBoxLayout>
#include <QWidget>

#include <iostream>

using namespace std;
```

Struktura hlavičkového souboru

- Aby nemohlo dojít k nekonečné rekurzi, definujeme pomocí direktivy `#define` symbolickou konstantu odvozenou vhodným způsobem ze jména souboru, pomocí podmínky `#ifndef` zajistíme, že vícenásobné vložení téhož souboru bude ignorováno
- Zvolená symbolická konstanta nesmí začínat podtržítkem nebo obsahovat dvě po sobě jdoucí podtržítka (takové názvy jsou vyhrazeny pro kompilátor a standardní knihovnu)
- Na začátek hlavičkového souboru musíme vložit hlavičkové soubory tříd, které jsou v hlavičkovém souboru použity (pak už je znovu nedáváme do `.cpp` souborů používajících tento hlavičkový soubor)
- Do hlavičkového souboru umisťujeme zejména definice tříd (bloky `class`), nikoli samostatné definice metod (ty patří do souboru `*.cpp`)

```
/*********************  Soubor  application.h   *********************/
#ifndef APPLICATION_H
#define APPLICATION_H

#include "graphicwidget.h"

#include <QApplication>

class Application : public QApplication
{
  // Zde budou uvedeny cleny tridy
};

#endif
```

Interaktivní prvky v knihovně Qt

- Knihovna Qt obsahuje různé interaktivní prvky ("widgety"), které slouží pro ovládání programu uživatelem
- Pro každý interaktivní prvek existuje v Qt knihovně příslušná třída, např.:

  `QPushButton` – tlačítko, po jehož stisnutí myší se vykoná specifikovaná operace (tlačítko obsahuje textový popisek)

  `QToolButton` – také tlačítko, ale místo textu obsahuje ikonu (používá se hlavně v nástrojových lištách)

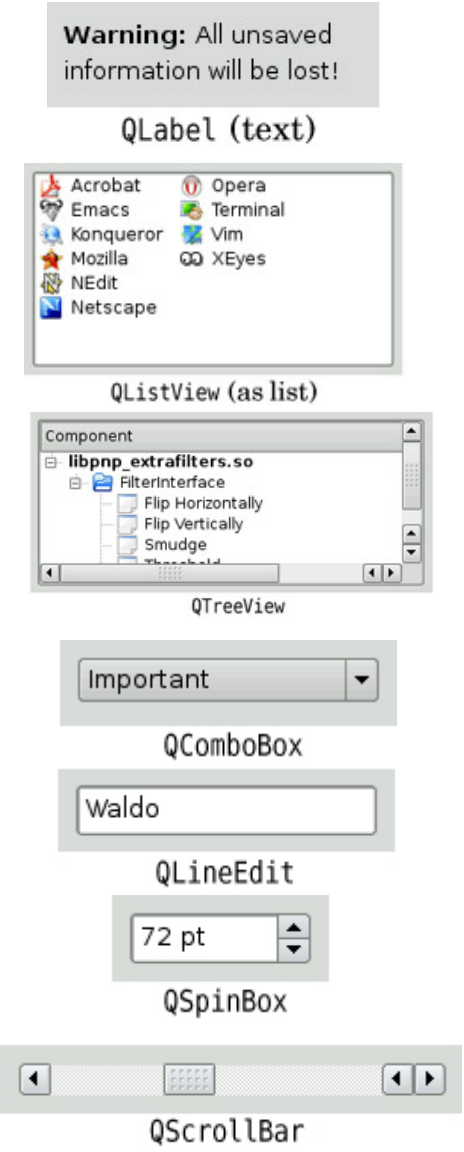
  `QCheckBox` – políčko se dvěma stavy (vybrán / nevybrán)

  `QRadioButon` – jedna ze vzájemně se vylučujících možností tvořící přepínač

Interaktivní prvky v knihovně Qt

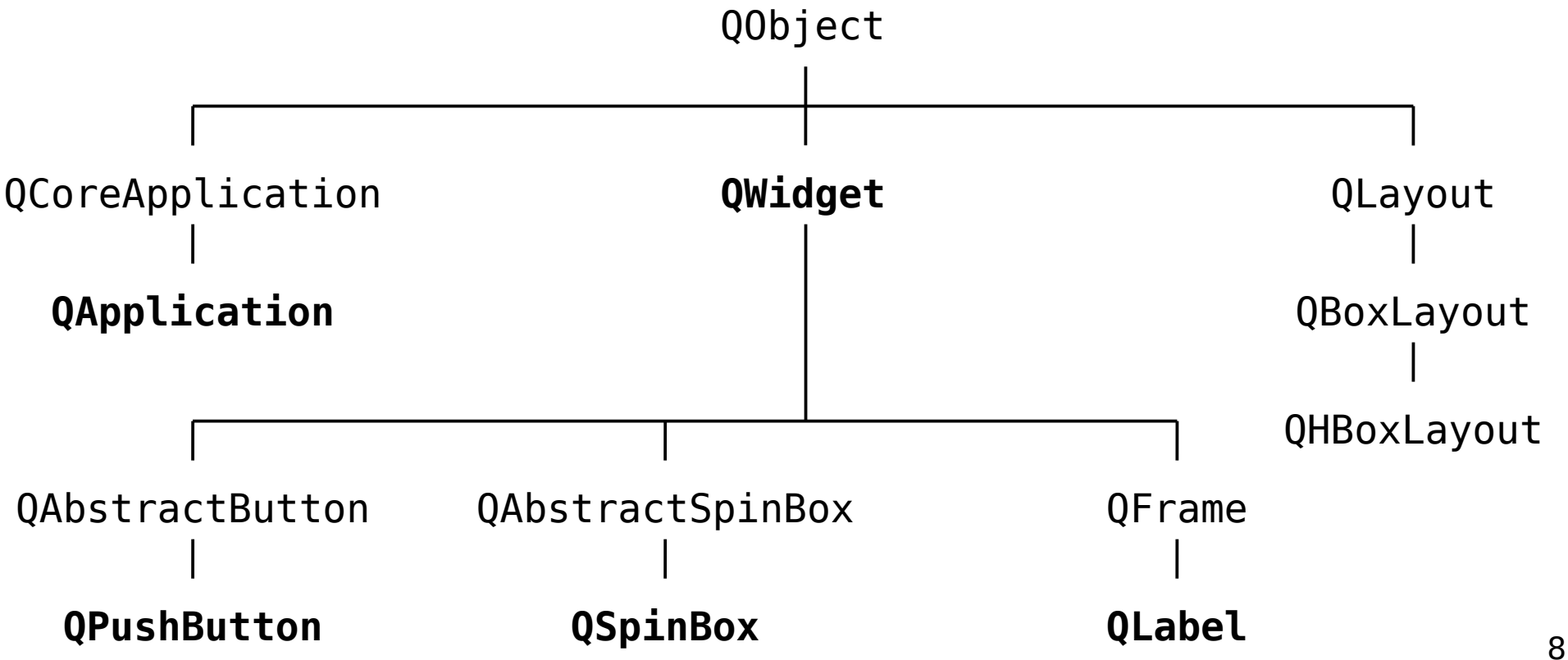
- Další interaktivní prvky knihovny Qt:
  - `QLabel` – textový popisek
  - `QListView` – seznam textových položek
  - `QTreeView` – hierarchický seznam (strom)
  - `QComboBox` – rozklikávací seznam (po kliknutí zobrazí seznam položek k výběru)
  - `QLineEdit` – políčko s jednořádkovým editovatelným textem
  - `QTextEdit` – políčko s víceřádkovým editovatelným textem
  - `QSpinBox` – políčko pro specifikaci číselné hodnoty
  - `QScrollBar` – posuvník (vodorovný nebo svislý)

- Úplný seznam lze nalézt na: https://doc.qt.io/qt-5/qtwidgets-index.html



Hierarchie tříd v knihovně Qt

- Většina tříd knihovny Qt je odvozena od základního typu `QObject`, který zajišťuje společné funkce pro všechny komponenty grafických programů (zejména správu objektů a jejich komunikaci)
- Mimo tuto hierarchii stojí jen pomocné třídy (`QString`, `QColor`, ...)
- Podrobnější informace na https://doc.qt.io/qt-5/objecttrees.html a https://doc.qt.io/qt-5/hierarchy.html



Životní cyklus objektů QObject

- Objekty tříd odvozených od QObject běžně tvoří jeden strom vlastnictví, který zajistí správné smazání nepotřebných objektů.
- Program tedy "ručně" spravuje jen jeden objekt (typicky hlavní okno), ten pak zajistí správu všech dalších widgetů, tlačítek, ...
- Pro zapojení objektů do stromu předáváme konstruktorům všech podřízených objektů ukazatel parent odkazující na nadřízený objekt
- Nejjednodušší je alokovat všechny podřízené objekty pomocí **new**. Nepoužíváme pro ně chytré ukazatele ani nevoláme **delete**, smazání zajistí sama knihovna Qt při smazání hlavního objektu.
- Pro hlavní objekt použijeme běžnou proměnnou či chytrý ukazatel.

```
void Application::run()
{
    QWidget mainWindow; // Hlavni objekt spravujici vse ostatni

    // Ostatni objekty budou podrizene
    QPushButton *okButton = new QPushButton("OK", &mainWindow);
    GraphicWidget *drawing = new GraphicWidget(&mainWindow);

    // Dalsi kod aplikace: mainWindow.neco(); drawing->neco();
    return QApplication::exec();

    // Zde budou pri destrukci mainWindow automaticky smazany
    // i ostatni objekty
}
```

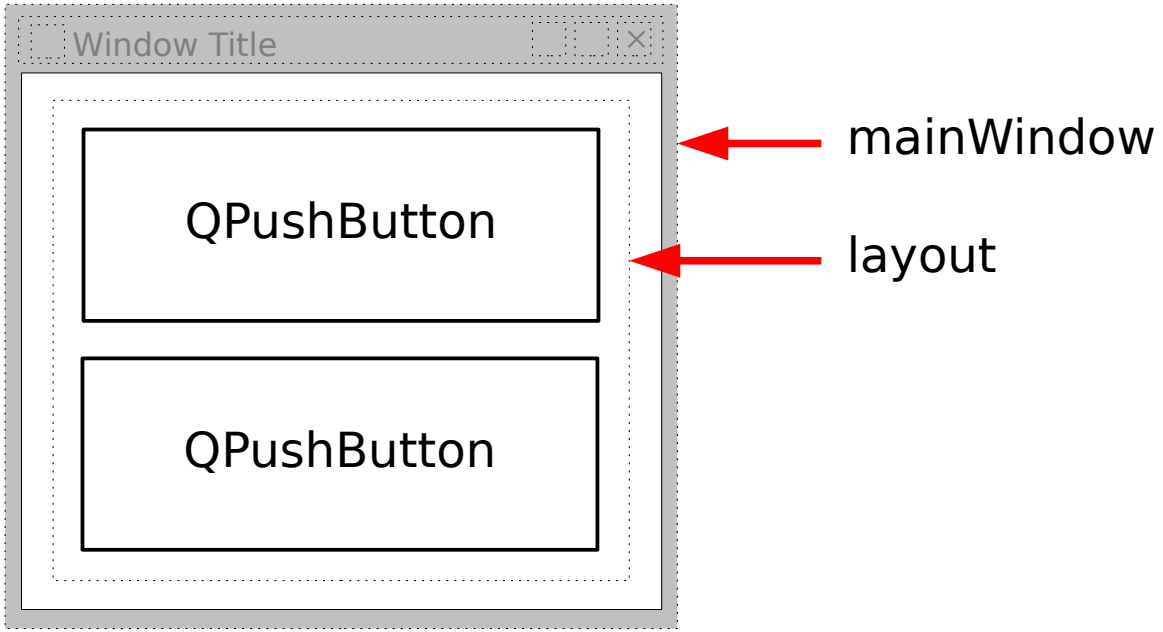
Makro Q_OBJECT

- U tříd odvozených od třídy `QObject` nebo jejích potomků (tj. např. `QApplication`, `QWidget`) musíme na úplný začátek definice třídy vložit makro `Q_OBJECT`, které je nezbytné pro zajištění základních funkcí knihovny Qt

```
// Ukazka definice tridy Application v souboru application.h

class Application : public QApplication
{
  Q_OBJECT

  public:
    Application(int &argc, char *argv[]);
    virtual ~Application();
    int run();

  private:
    QWidget mainWindow;
    GraphicWidget* graphicWidget;
};
```

Rozvržení prvků v okně

- Pro automatické rozmístění interaktivních prvků v okně používáme objekty tříd odvozených od `QLayout`, hlavně `QHBoxLayout` a `QVBoxLayout`
- `QHBoxLayout` rozmisťuje objekty horizontálně, `QVBoxLayout` je rozmisťuje vertikálně
- Objekt typu `QHBoxLayout` nebo `QHBoxLayout` potom přiřadíme do okna metodou `setLayout()` třídy `QWidget`
- Pro přidávání widgetů do objekt typu `QHBoxLayout` nebo `QHBoxLayout` používáme metodu `addWidget()`
- Objekt typu `QHBoxLayout` nebo `QVBoxLayout` zajistí nastavení pozice a velikost prvků a také mezery mezi nimi

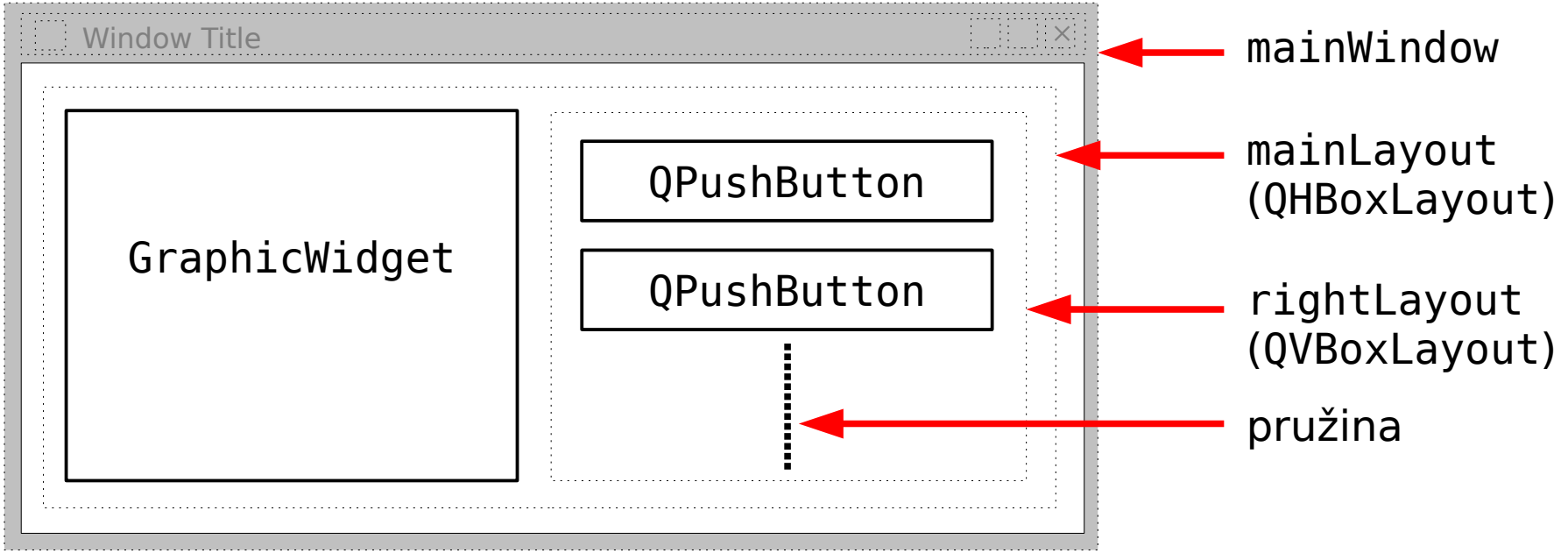


Rozvržení prvků v okně – příklad 1

```
// Program vytvori okno a do nej budou vlozeny dve tlacitka
// usporadana vertikalne nad sebou pomoci objektu QVBoxLayout

// Aktualni trida obsahuje polozku QWidget mainWindow;
mainWindow.setWindowTitle("Program vytvoreny v Qt!");

// Vytvorime dve tlacitka
QPushButton* button1 = new QPushButton("Button 1", &mainWindow);
QPushButton* button2 = new QPushButton("Button 2", &mainWindow);
// Vytvorime objekt QVBoxLayout pripojeny k mainWindow,
// ktery bude rozmistovat tlacitka vertikalne nad sebou,
// pozice a velikost se nastavi automaticky
QVBoxLayout *layout = new QVBoxLayout(&mainWindow);
// Tlacitka pridame pomoci metody addWidget()
layout->addWidget(button1);
layout->addWidget(button2);
```

Rozvržení prvků v okně

- Objekty typu QHBoxLayout nebo QVBoxLayout lze vnořovat do sebe pomocí metody addLayout()
- Kombinací objektů typu QHBoxLayout nebo QVBoxLayout můžeme vytvořit i složitější rozmístění objektů
- Při roztažení okna jsou objekty rozmisťovány tak, aby byly centrované a mezery mezi nimi proporcionální, toto chování lze ovlivnit vložením "pružné výplně" metodou addStretch()



Rozvržení prvků v okně – příklad 2

- Objektům zapojeným do nějakého QLayout nemusíme předávat ukazatel parent (bude nastaven automaticky pomocí addWidget() a setLayout())

```
// Program vytvori okno a do nej bude vlozen widget
// typu GraphicWidget a vpravo budou dve tlacitka nad sebou
QWidget mainWindow;
mainWindow.setWindowTitle("Program vytvoreny v Qt!");

graphicWidget = new GraphicWidget;
// Pro objekt graphicWidget nastavime minimalni velikost
graphicWidget->setMinimumSize(300, 350);
QPushButton *buttonHide = new QPushButton("Hide");
QPushButton *buttonShow = new QPushButton("Show");

QVBoxLayout *rightLayout = new QVBoxLayout;
rightLayout->addWidget(buttonHide);
rightLayout->addWidget(buttonShow);
// Pod tlacitka pridame pruznou vycpavku
rightLayout->addStretch();

QHBoxLayout *mainLayout = new QHBoxLayout;
mainLayout->addWidget(graphicWidget);
mainLayout->addLayout(rightLayout);

// Do hlavniho okna nastavime prislusny layout
mainWindow.setLayout(mainLayout);

mainWindow.show();
```

Komunikace mezi objekty v Qt

- Ke komunikaci mezi libovolnými objekty odvozenými od `QObject` slouží systém tzv. signálů a slotů (*signals and slots*)
- Signály a sloty se používají nejčastěji pro zaslání informace od interaktivního objektu (např. informace o stisknutí tlačítka) do jiného objektu (hlavního okna nebo jiného widgetu)
- Signál je metoda deklarovaná v objektu, od něhož signál pochází
- Slot je metoda, kterou vytvoříme ve třídě, která bude zpracovávat zaslaný signál
- Signály a sloty jsou ve třídách deklarovány ve speciálních sekcích označených `signals` a `slots`
- Ve třídě `QPushButton` je definován signál `clicked()`, který je generován po stisknutí tlačítka
- Propojení mezi zaslaným signálem a slotem provedeme pomocí funkce `connect()`:

```
QObject::connect(object1, signal, object2, slot);
```

- Podrobnější popis najdete v https://doc.qt.io/qt-5/signalsandslots.html

Komunikace mezi objekty v Qt – příklad

```
// Deklarace slotu ve tride GraphicWidget v souboru graphicwidget.h
class GraphicWidget : public QWidget
{
  Q_OBJECT
  private slots:
    // Nasledujici metody slotu budou volany po zmacknuti
    // prislusnych tlacitek Hide a Show
    void hideRectangle();
    void showRectangle();
  // Deklarace dalsich clenu tridy
};
```

```
  // Program ukazuje propojeni mezi signalem clicked() od dvou
  // tlacitek se sloty v objektu tridy GraphicWidget

  graphicWidget = new GraphicWidget;
  QPushButton* buttonHide = new QPushButton("Hide");
  QPushButton* buttonShow = new QPushButton("Show");

  // Signal clicked() z tlacitka buttonHide zpusobi volani metody
  // hideRectangle() definovane ve tride GraphicWidget
  QObject::connect(buttonHide, &QPushButton::clicked,
                   graphicWidget, &GraphicWidget::hideRectangle);
  // Podobne pro tlacitko buttonShow bude volana metoda
  // showRectangle() definovana ve tride GraphicWidget
  QObject::connect(buttonShow, &QPushButton::clicked,
                   graphicWidget, &GraphicWidget::showRectangle);
```

Definice slotu

- Vhodné signály jsou zpravidla již předdefinované ve třídách Qt knihovny, většinou potřebujeme definovat pouze sloty
- Metody slotu obsahují kód reagující na signál

```
// Definice metody slotu, která je zavolána po stisknutí
// tlacitka buttonHide

void GraphicWidget::hideRectangle()
{
  // Vypiseme informaci o stisknuti tlacitka na terminal
  cout << "Bylo stisknuto tlacitko Hide" << endl;

  // Do promenne displayRectangle priradime hodnotu false
  // indikujici ze obdelnik nema byt vykreslovan
  displayRectangle = false;

  // Vyvolame pozadavek na prekresleni okna metodou update()
  update();
}
```

Dialogová okna v knihovně Qt

- V knihovně Qt můžeme vytvářet dialogová okna, která odvozujeme ze třídy `QDialog`
- V knihovně je předdefinováno několik nejčastěji používaných dialogových oken:
  - `QFileDialog` – dialogové okno pro výběr souboru nebo adresáře
  - `QColorDialog` – dialogové okno pro výběr barvy
  - `QFontDialog` – dialogové okno pro výběr fontu
  - `QMessageBox` – dialogové okno pro zobrazení textové zprávy
  - `QInputDialog` – dialogové okno pro získání jedné textové nebo číselné hodnoty od uživatele
- Úplný seznam se nachází na https://doc.qt.io/qt-5/dialogs.html

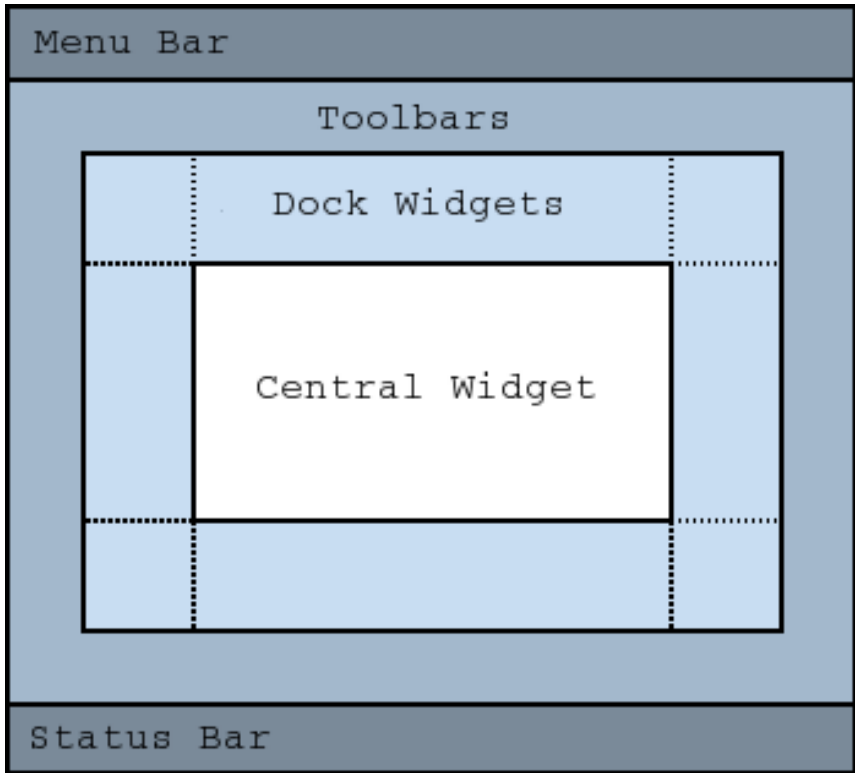
Dialogové okno pro výběr souboru

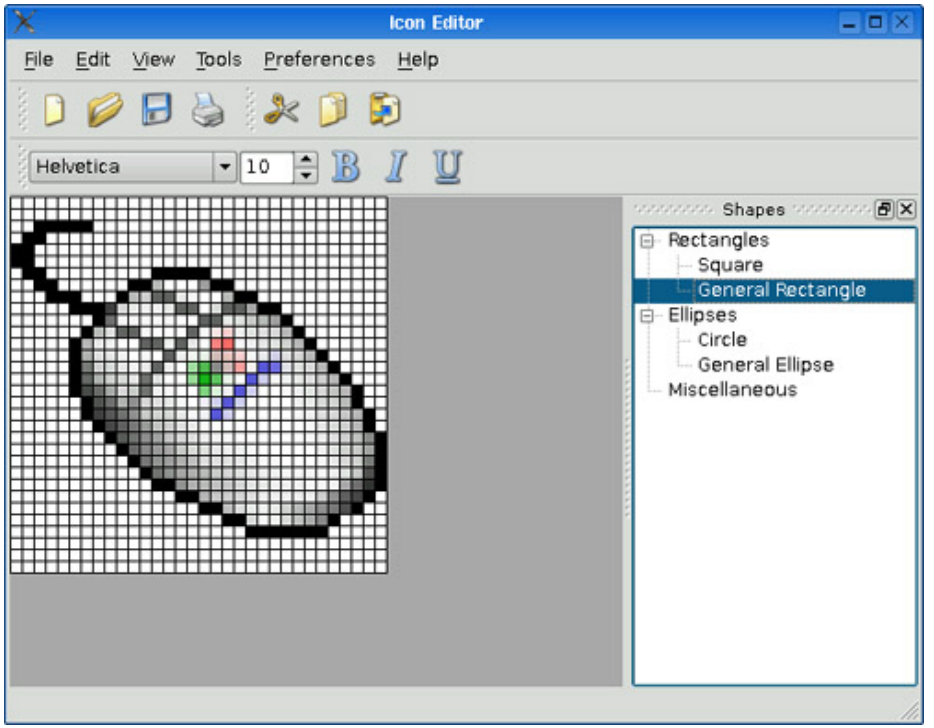
- Pro jednoduchou práci s dialogovými okny jsou ve knihovně Qt předdefinovány statické metody, které automaticky vytvoří příslušný objekt dialogového okna a okno zobrazí
- Dialogové okno pro vybrání souboru lze otevřít metodou `QFileDialog::getOpenFileName()`, která vrátí jméno souboru

```
// Nasledujici metoda je zavolana po stisknuti tlacitka pro
// otevreni souboru
void GraphicWidget::openFile()
{
  // Knihovna Qt pouziva pro retezce tridu QString misto string
  QString fileName;
  // Dialogove okno pro vyber souboru otevreme nasledujici metodou,
  // ktera vrati jmeno souboru jako retezec typu QString
  fileName = QFileDialog::getOpenFileName(this, "Vyber soubor", ".");
  // Pokud nebylo vybrano jmeno souboru, je retezec prazdny
  if (fileName.isEmpty()) return;

  // Standardni vystupni proudy umi pracovat jen s promennymi typu
  // string, na ktere musime konvertovat promennou fileName, ktera
  // je typu QString
  cout << "Jmeno souboru: " << fileName.toStdString() << endl;
}
```

Třída QMainWindow

- Hlavní okno aplikace se ve knihovně Qt obvykle odvozuje ze třídy `QMainWindow`, která poskytuje podporu pro vytváření hlavního menu, nástrojových lišť, stavového řádku atd.
- Více informací: https://doc.qt.io/qt-5/qmainwindow.html





Dodržujte následující pravidla

- Pro každou třídu vytvořte samostatný soubor `.h` a případně `.cpp`.
- V hlavičkovém souboru vždy použijte direktivy uvedené v sekci „Struktura hlavičkového souboru“.
- Na začátek souborů *`*.cpp`* vložte vždy jen hlavičkové soubory s těmi třídami, které v daném souboru opravdu používáte. Jako první uveďte vždy hlavičkový soubor příslušející danému *`*.cpp`* souboru.
- Do adresáře s projektem ani jeho podadresářů neumisťujte žádné jiné soubory *`*.cpp`* a *`*.h`* než ty, které jsou pro projekt potřeba. Tyto soubory by totiž byly automaticky zahrnuty do souboru projektu (při jeho generování příkazem *`qmake -project`*) a byly by tedy i kompilovány.
- Všechny soubory *`*.cpp`*, *`*.h`* a *`*.pro`* patřící k jedné úloze odevzdejte do příslušné pododevzdávárny („Úloha 1“ a „Úloha 2“). V těchto odevzdávárnách již nevytvářejte žádné další podsložky.

Cvičení

1. Vytvořte program vycházející z programu z předchozího cvičení, který bude v hlavním okně obsahovat widget `GraphicWidget` a napravo dvě tlačítka s popisem například *Hide* a *Show*. Po stisknutí prvního tlačítka dojde ke skrytí obdélníku (tj. okno se překreslí a vykreslí se jen elipsa a čára). Po stisknutí druhého tlačítka se obdélník opět zobrazí. **2 body**

2. Do programu přidejte třetí tlačítko, po jehož stisknutí se zobrazí dialogové okno pro výběr libovolného souboru. Po vybrání souboru se jeho jméno vypíše na terminál a také se zobrazí v okně s grafikou. **2 body**