

**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.)
 - Vygenerujeme *Makefile* příkazem `qmake`
- Při **přidání/odebrání `#include`** v jakémkoli souboru:
 - Přegenerujeme *Makefile* příkazem `qmake`
- Při **přidání nového `.cpp/.h`** souboru:
 - a) ručně upravíme `.pro` soubor a přidáme *novysoubor.cpp* na řádek `SOURCES` či *novysoubor.h* na řádek `HEADERS`, 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 umísťujeme 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 členy tridy  
};  
  
#endif
```

Interaktivní prvky v knihovně Qt

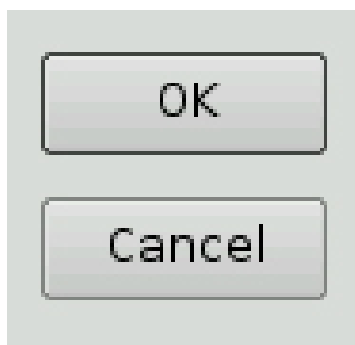
- Knihovna Qt obsahuje různé interaktivní prvky ("widgety"), které slouží pro ovládání programu uživatelem
- Pro každý interaktivní prvek existuje v Qt knihovně příslušná třída, např.:

QPushButton – tlačítko, po jehož stisknutí myší se vykoná specifikovaná operace (tlačítko obsahuje textový popis)

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

QCheckBox – políčko se dvěma stavy (vybrán / nevybrán)

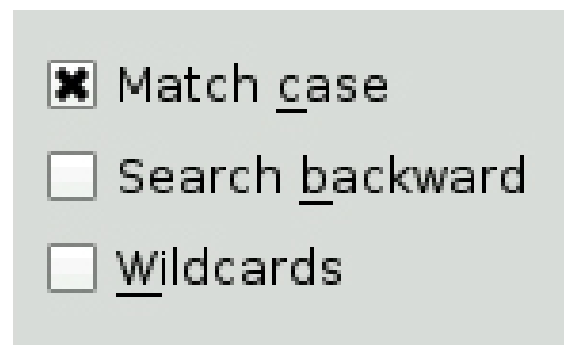
QRadioButton – jedna ze vzájemně se vylučujících možností tvořící přepínač



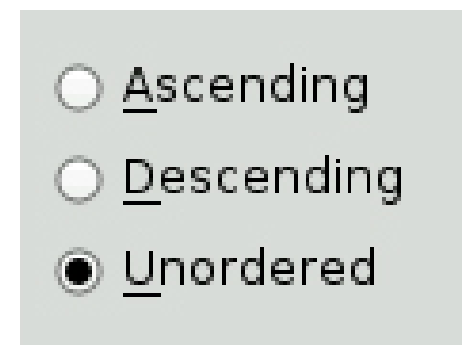
QPushButton



QToolButton



QCheckBox



QRadioButton

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:

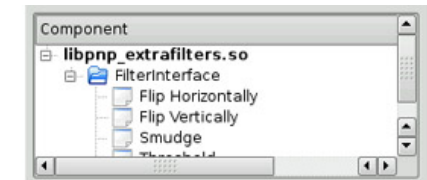
<https://doc.qt.io/qt-5/qtwidgets-index.html>

Warning: All unsaved information will be lost!

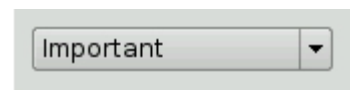
QLabel (text)



QListView (as list)



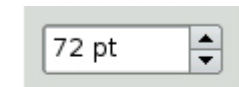
QTreeView



QComboBox



QLineEdit



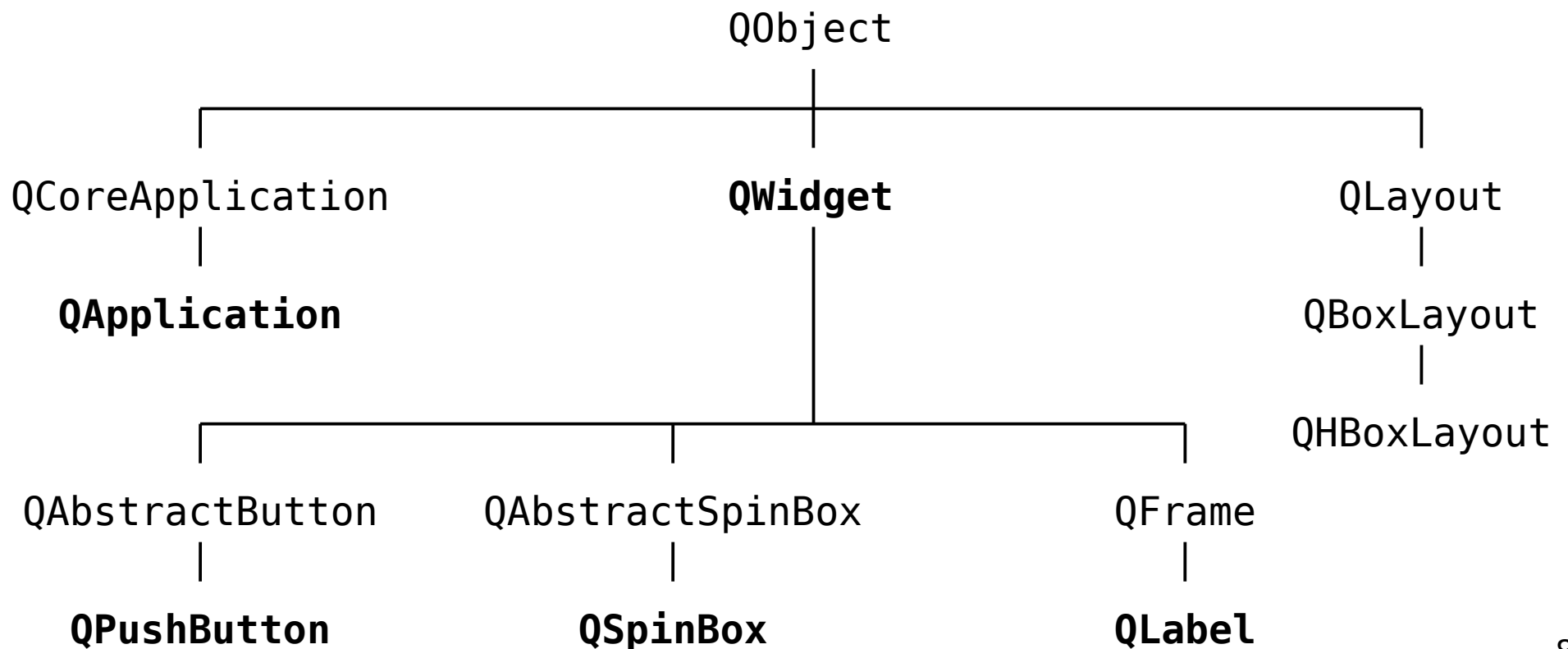
QSpinBox



QScrollBar

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 nadřazený 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; // Hlavní objekt spravující vše ostatní

    // Ostatní objekty budou podřízené
    QPushButton *okButton = new QPushButton("OK", &mainWindow);
    GraphicsWidget *drawing = new GraphicsWidget(&mainWindow);

    // Další kód aplikace: mainWindow.neco(); drawing->neco();
    return QApplication::exec();

    // Zde budou při destrukci mainWindow automaticky smazány
    // i ostatní 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

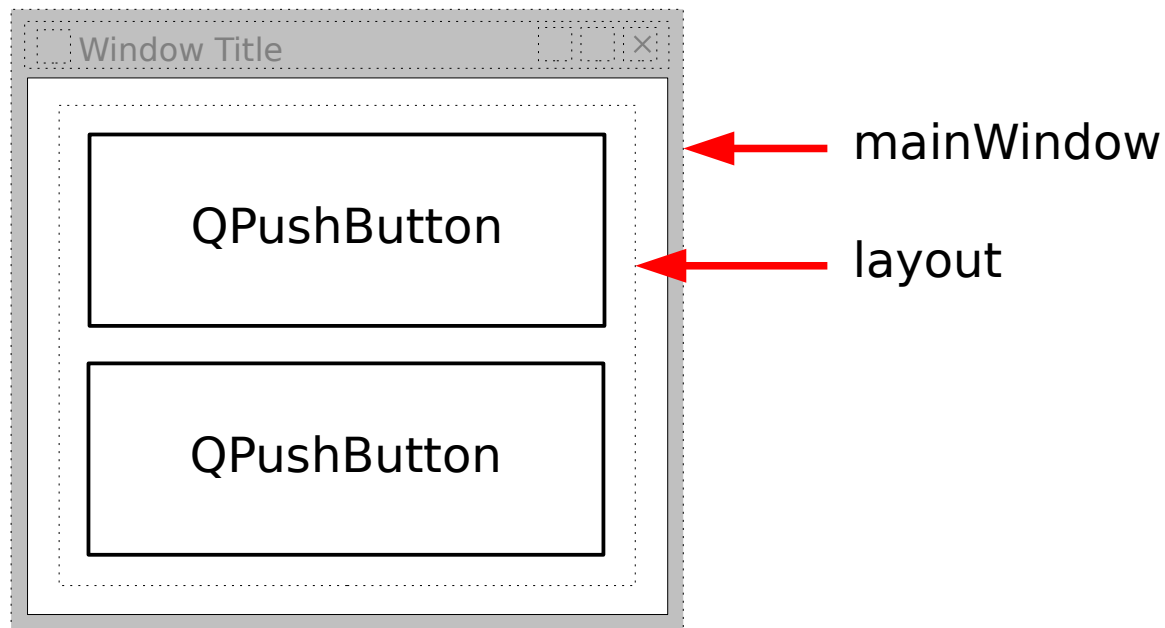
class Application : public QApplication
{
    Q_OBJECT

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

    private:
        QWidget mainWindow;
        QWidget* 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 `QVBoxLayout` potom přiřadíme do okna metodou `setLayout()` třídy `QWidget`
- Pro přidávání widgetů do objekt typu `QHBoxLayout` nebo `QVBoxLayout` používáme metodu `addWidget()`
- Objekt typu `QHBoxLayout` nebo `QVBoxLayout` zajistí nastavení pozice a velikost prvků a také mezery mezi nimi

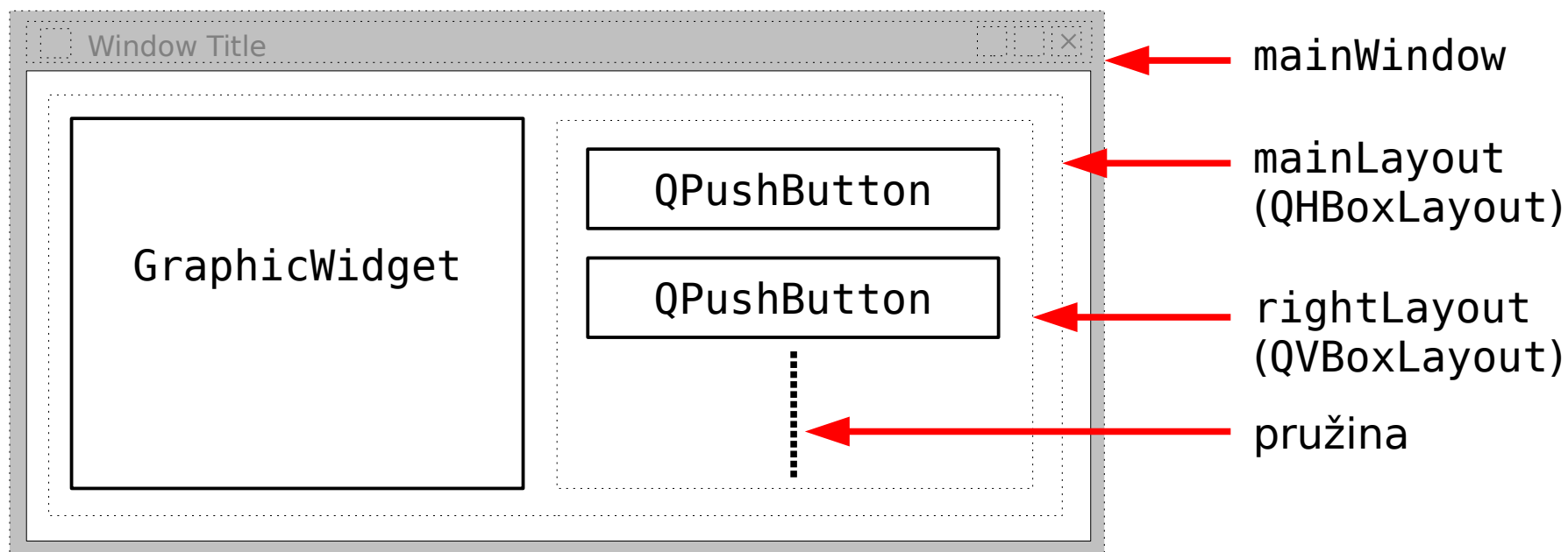


Rozvržení prvků v okně - příklad 1

```
// Program vytvoří okno a do něj budou vložena dvě tlačítka,  
// uspořádaná vertikálně nad sebou pomocí objektu QVBoxLayout  
  
// Aktuální třída obsahuje položku QWidget mainWindow;  
mainWindow.setWindowTitle("Program vytvořený v Qt!");  
  
// Vytvoríme dvě tlačítka  
QPushButton* button1 = new QPushButton("Button 1", &mainWindow);  
QPushButton* button2 = new QPushButton("Button 2", &mainWindow);  
// Vytvoríme objekt QVBoxLayout připojený k mainWindow,  
// který bude rozmisťovat tlačítka vertikálně nad sebou,  
// pozice a velikost se nastaví automaticky  
QVBoxLayout *layout = new QVBoxLayout(&mainWindow);  
// Tlačítka přidáme pomocí 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 centrovány 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 vytvoří okno a do něj bude vložen widget
// typu QWidget a vpravo budou dvě tlačítka nad sebou
QWidget mainWindow;
mainWindow.setWindowTitle("Program vytvořený v Qt!");

QWidget graphicWidget = new QWidget;
// Pro objekt graphicWidget nastavíme minimální 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 tlačítka přidáme prázdnou vycpávku
rightLayout->addStretch();

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

// Do hlavního okna nastavíme příslušný layout
// Alternativa: předat &mainWindow konstruktoru mainLayout
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 zasláný 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 zasláným 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
    public 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
// tlačítka buttonHide

void GraphicWidget::hideRectangle()
{
    // Vypíšeme informaci o stisknutí tlačítka na terminal
    cout << "Bylo stisknuto tlačítko Hide" << endl;

    // Do proměnné displayRectangle přiřadíme hodnotu false
    // indikující ze obdelník nemá být vykreslovan
    displayRectangle = false;

    // Vyvoláme požadavek na překreslení 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

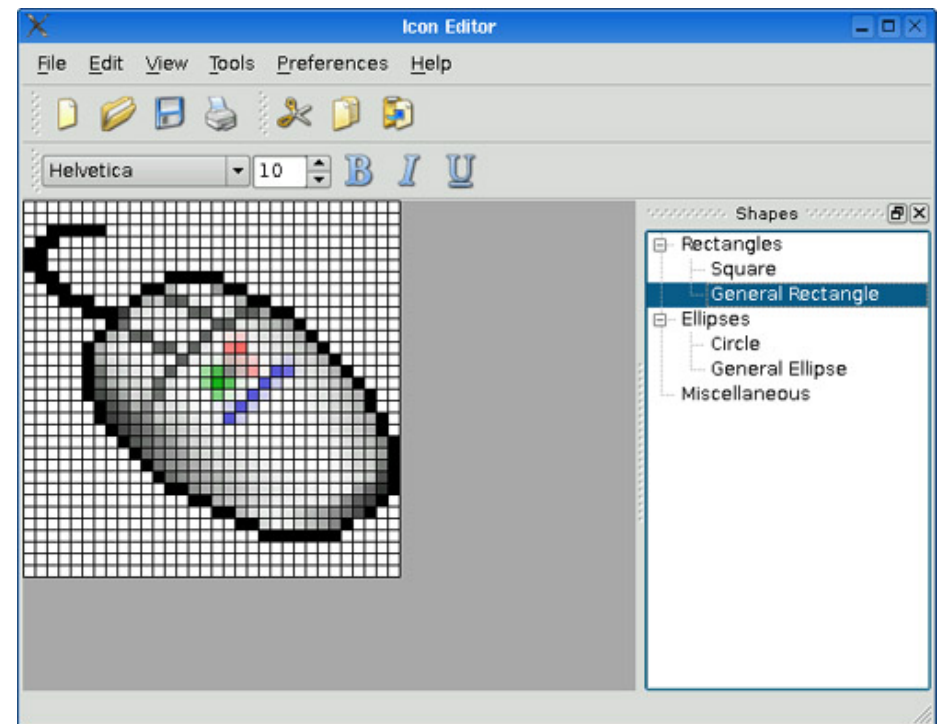
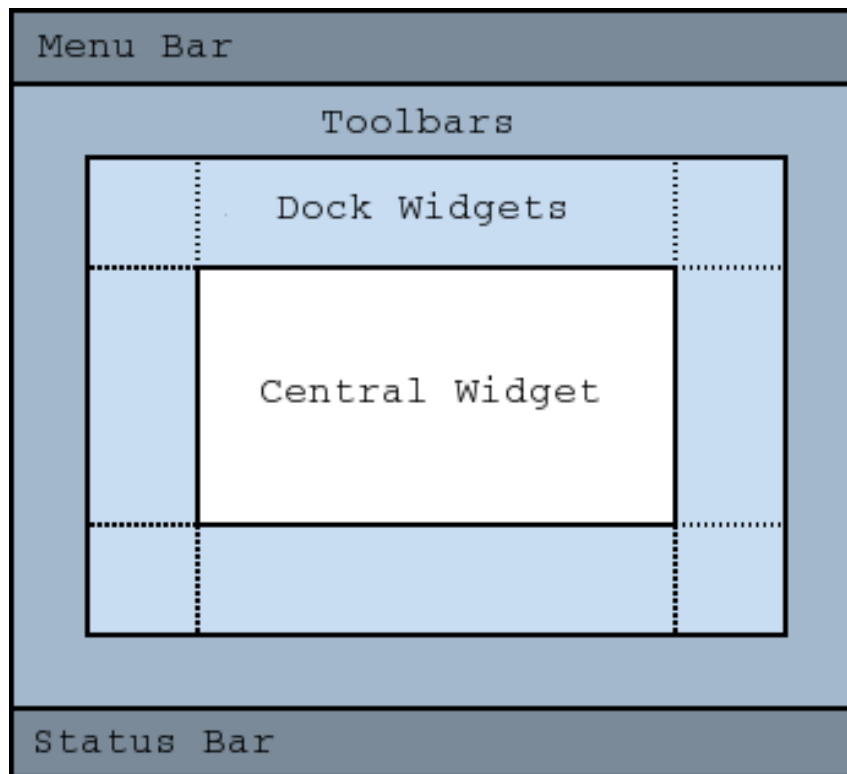
- Pro jednoduchou práci s dialogovými okny jsou ve knihovně Qt předdefinovány statické metody, které automaticky vytvoří příslušný objekt dialogového okna a okno zobrazí
- Dialogové okno pro vybrání souboru lze otevřít metodou `QFileDialog::getOpenFileName()`, která vrátí jméno souboru

```
// Nasledující metoda je zavolána po stisknutí tlačítka pro
// otevření souboru
void GraphicWidget::openFile()
{
    // Knihovna Qt používá pro řetězce třídu QString místo string
    QString fileName;
    // Dialogové okno pro výběr souboru otevřeme následující metodou,
    // která vrátí jméno souboru jako řetězec typu QString
    fileName = QFileDialog::getOpenFileName(this, "Vyber soubor", ".");
    // Pokud nebylo vybráno jméno souboru, je řetězec prázdný
    if (fileName.isEmpty()) return;

    // Standardní výstupní proudy umí pracovat jen s proměnnými typu
    // string, na které musíme konvertovat proměnnou fileName, která
    // je typu QString
    cout << "Jmeno souboru: " << fileName.toString() << endl;
}
```

Třída QMainWindow

- Hlavní okno aplikace se ve knihovně Qt obvykle odvozuje ze třídy QMainWindow, která poskytuje podporu pro vytváření hlavního menu, nástrojových lišt, 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 neumísťujte 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**

