Qt

Iuliana Bocicor
iuliana@cs.ubbcluj.ro

Babes-Bolyai University

2022

Overview

Qt toolkit I

- Qt is a cross-platform application and UI framework in C++.
- Using Qt, one can write GUI applications once and deploy them across desktop, mobile and embedded operating systems without rewriting the source code.
- Qt is supported on a variety of 32-bit and 64-bit platforms (Desktop, embedded, mobile):
  - Windows (MinGW, MSVS)
  - Linux (gcc)
  - Apple Mac OS
  - Mobile / Embedded (Symbian, Windows Embedded Compact, Windows 10 Mobile, Embedded Linux, Android, LG webOS)

Qt toolkit II

- Language bindings are available in C#, Java, Python (PyQt, Qt for Python), Ada, Pascal, Perl, PHP (PHP-Qt), Ruby (RubyQt).
- Qt is available under GPL v3, LGPL v2 and commercial license.
- Qt documentation: https://doc.qt.io/.

Applications/companies using Qt

- Spotify
- VLC Player
- Autodesk Maya 2011
- LibreCAD
- Google Earth desktop
- HP Envy printer (printer's touch screen)
- Roku Set-top Box (streaming players)
- German Air Traffic Control
- DreamWorks (movie production company)
- Wolfram Mathematica
- GNU Octave
- Samsung (e-readers and digital photo frames)
- VirtualBox
- Video gaming: Age of Wonders III, Blizzard Battle.net client, City of Heroes.

https://blogs.windows.com/devices/2011/03/15/10-qt-use-cases-you-didnt-know/

Download and install Qt

How to install Qt for VS 2022
http://www.cs.ubbcluj.ro/~iuliana/oop/ - Tutorials

If using CLion, refer to https://www.jetbrains.com/help/clion/qt-tutorial.html.

- Qt can be used with Microsoft Visual Studio, Eclipse (provided there is a C++ compiler installed) and CLion.
- There is also an IDE which is part of the SDK for the Qt GUI Application development framework - **QtCreator**. This also needs a C++ compiler.

Qt Hello World - Qt GUI Project Wizard I

- Create a new Qt application, as described in the tutorial mentioned on the previous slide.

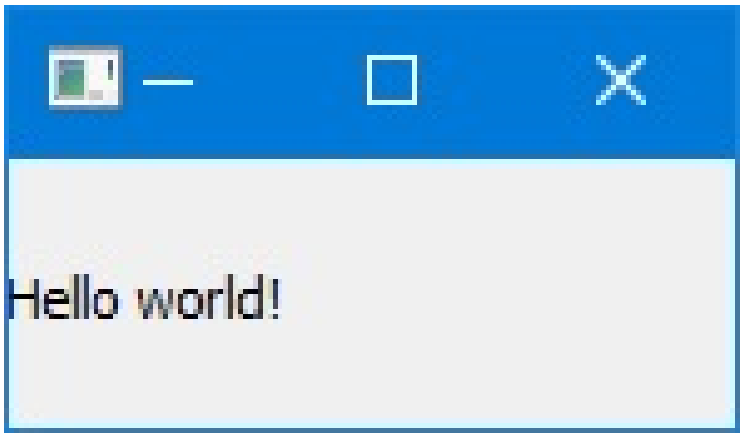
- Edit the file *main.cpp*: add a new QLabel and display it, as follows:

```
#include <QtWidgets/QApplication>
#include <QtWidgets/QLabel>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QLabel label("Hello world!");
    label.show();
    return a.exec();
}
```

Qt Hello World - Qt GUI Project Wizard II

- By executing the application, you should get the following result:



QApplication I

- The QApplication class manages the GUI application's control flow and main settings.
- QApplication contains the main event loop, where all events from the window system and other sources are processed and dispatched.
- For any GUI application using Qt, there is exactly one QApplication object (no matter how many windows the application has at any given time). This object is accessible using the function instance().

QApplication II

- Responsibility:
  - initializes the application with the user's desktop settings;
  - takes care of the *event loop*: performs event handling, it receives events from the underlying window system and dispatches them to the relevant widgets;
  - knows about the application's windows;
  - defines the application's look and feel.

QApplication III

- For non-GUI Qt applications, use QCoreApplication instead.
- The exec() method of the QApplication makes the application enter its *event loop*.
- When a Qt application is running, the event loop waits for user input, then events are generated and sent to the widgets of the application.
- The loop is terminated when any of the functions exit() or quit() is called.

Qt GUI Components (widgets) I

- Widgets are the basic building blocks for graphical user interface (GUI) applications built with Qt. E.g.: buttons, labels, textboxes, etc.
- A GUI component (widget) can be placed on the user interface window or can be displayed as an independent window.
- A widget that is not embedded in a parent widget is called a window.
- Windows provide the screen space upon which the user interface is built.

Qt GUI Components (widgets) II

- Windows visually separate applications from each other and usually provide a window decoration (show a title bar, allows the user to resize, position, etc).
- The Widgets module in Qt uses inheritance.
- All widgets inherit from QWidget, which is derived from QObject.

Qt GUI Components (widgets) III

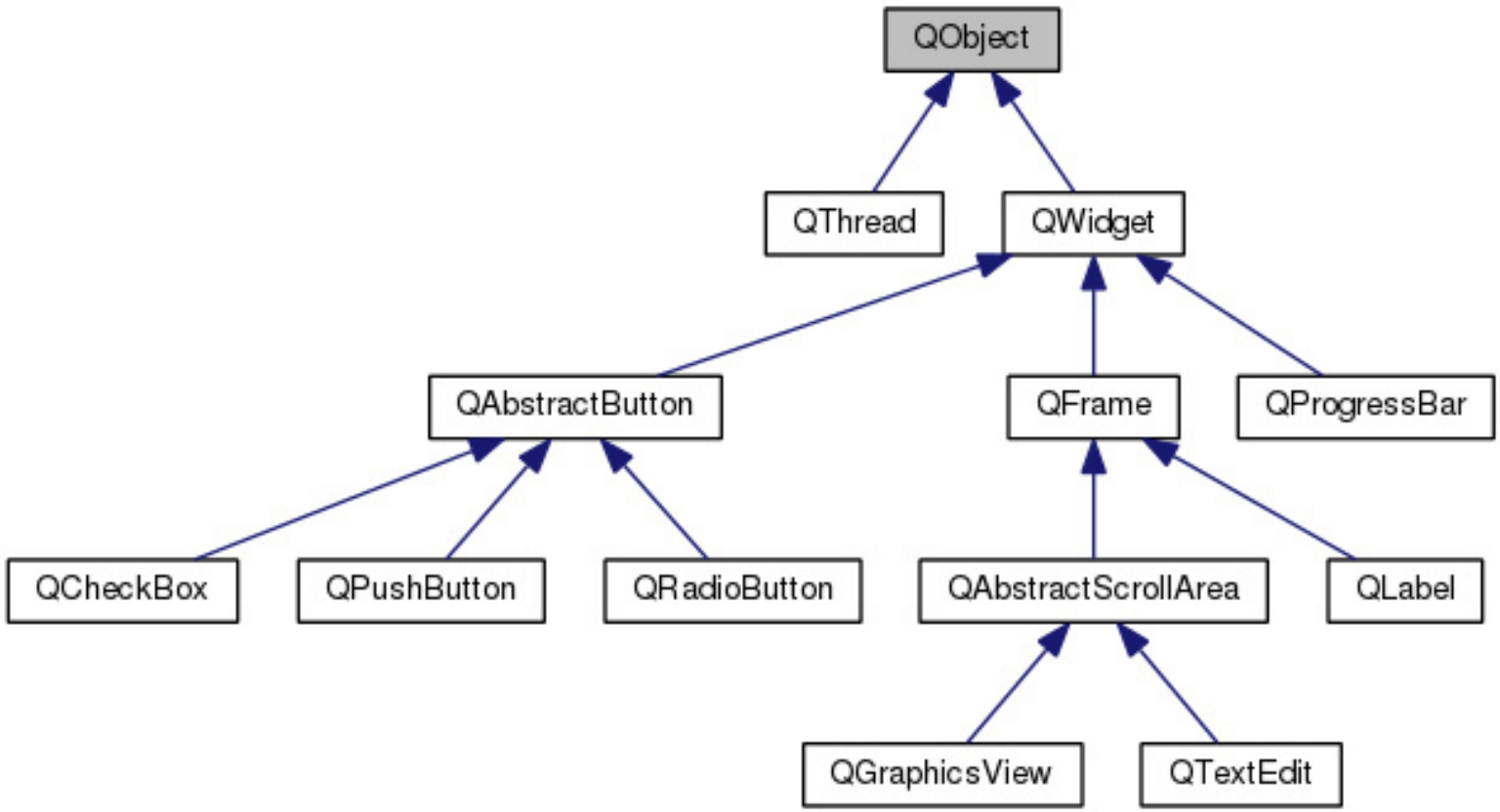


Figure source: https://wiki.qt.io/Qt_for_Beginners

Qt GUI Components (widgets) IV

- Widgets use the *parenting system*:
  - Any object that inherits from QObject can have a parent and children.
  - When an object is destroyed, all of its children are destroyed as well.
  - All QObjects have methods that allow searching the object's children.
  - Child widgets in a QWidget automatically appear inside the parent widget.

Widget example - label

QLabel

- QLabel is used for displaying text or an image.
- No user interaction functionality is provided.
- A QLabel is often used as a label for an interactive widget.
- For this use QLabel provides a useful mechanism for adding an mnemonic that will set the keyboard focus to the other widget (called the QLabel's "buddy").
- Is defined in the header <QLabel>.

```
QLabel label("Hello :)");
label.show();
```

Widget example - textbox

QLineEdit

- QLineEdit widget is a one-line text editor.
- A line edit allows the user to enter and edit a single line of plain text with a useful collection of editing functions, including undo and redo, cut and paste, and drag and drop.
- A related class is QTextEdit which allows multi-line, rich text editing.
- Is defined in the header <QLineEdit>.

```
QLineEdit lineEdit;
QLabel label("&Hello :)");
label.setBuddy(&lineEdit);
```

Widget example - button

QPushButton

- The QPushButton widget provides a command button.
- Push (click) a button to command the computer to perform some action.
- Push buttons display a textual label, and optionally a small icon. A shortcut key can be specified by preceding the preferred character with an ampersand.
- Is defined in the header <QPushButton>.

DEMO

Push button (*Lecture9_demo_widgets* - function *buttonExample*).

Widget example - list

QListWidget

- The QListWidget widget provides an item-based list widget.
- The widget presents a list of items to the user.
- QListWidget uses an internal model to manage each item in the list (QListWidgetItem).
- Is defined in the header <QListWidget>.

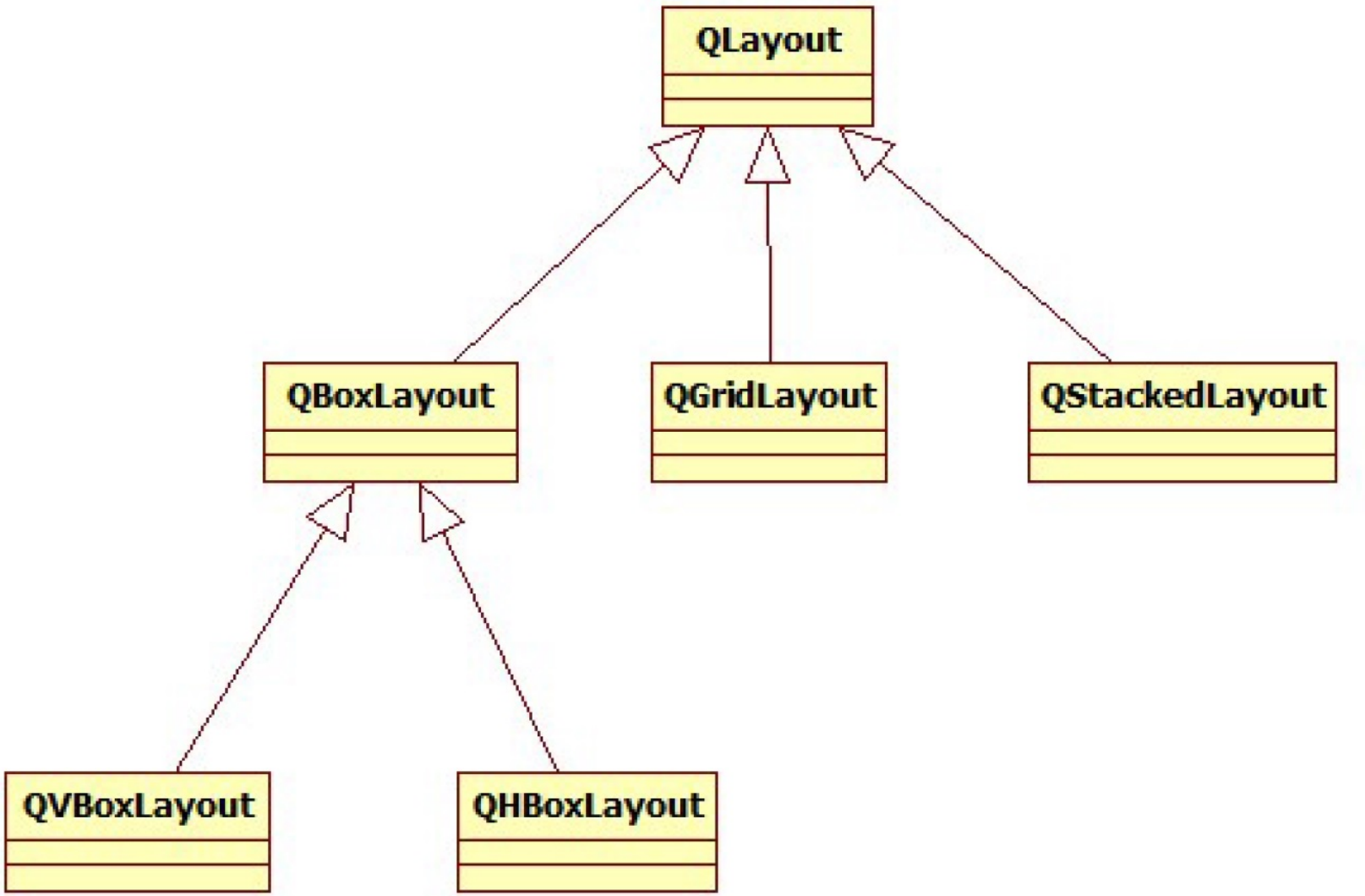
DEMO

List (*Lecture9_demo_widgets* - function *listExample*).

Layout management I

- The Qt layout system provides a way to automatically arrange child widgets within a widget to ensure that they make good use of the available space.
- Qt includes a set of layout management classes that are used to describe how widgets are laid out in an application's user interface.
- These layouts automatically position and resize widgets when the amount of space available for them changes, ensuring that they are consistently arranged and that the user interface as a whole remains usable.

Layout management II



QHBoxLayout

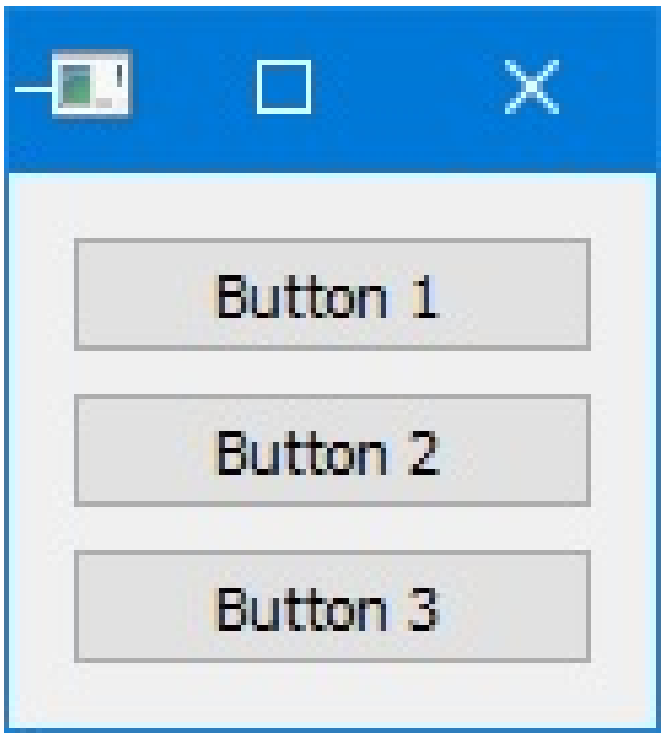
- Widgets are aligned horizontally.

DEMO

QHBoxLayout (*Lecture9_demo_widgets* - function *hBoxLayout*).

QVBoxLayout

- Widgets are aligned vertically.

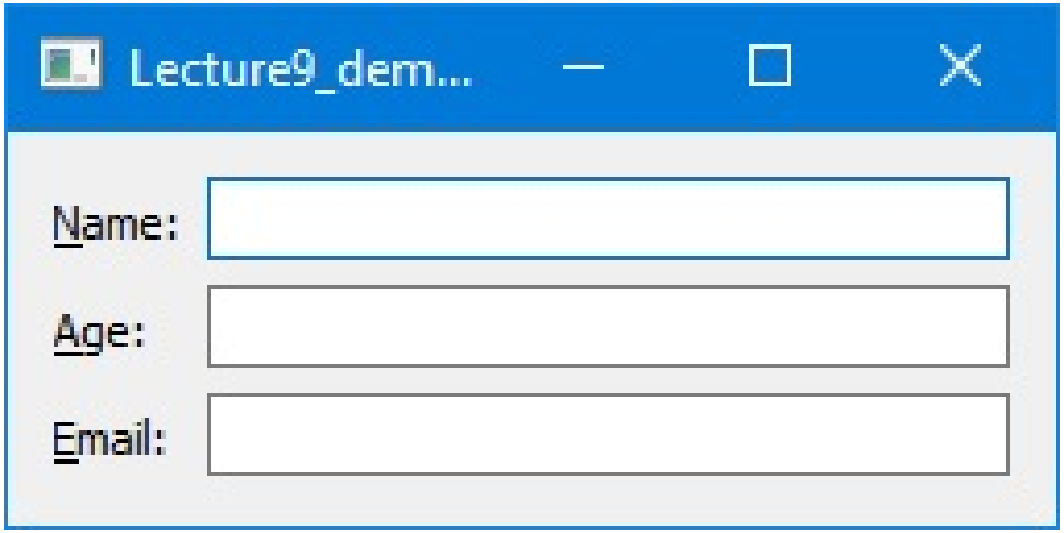


DEMO

QVBoxLayout (*Lecture9_demo_widgets* - function *vBoxLayout*).

QFormLayout

- Widgets are layed out in a two column form.
- The left column contains labels and the right column contains widgets.

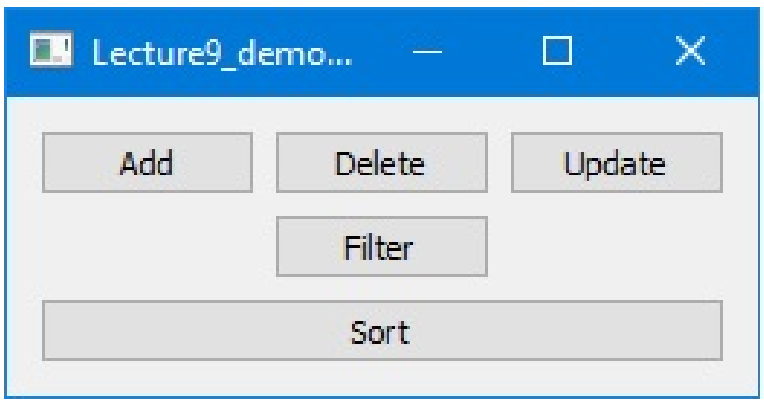


DEMO

QFormLayout (*Lecture9_demo_widgets* - function *formLayout*).

QGridLayout

- Widgets are layed out in a grid.
- The space is divided into rows and columns and each widget is put in the specified cell.
- It is also possible for a widget to occupy multiple cells by spanning the row/column.

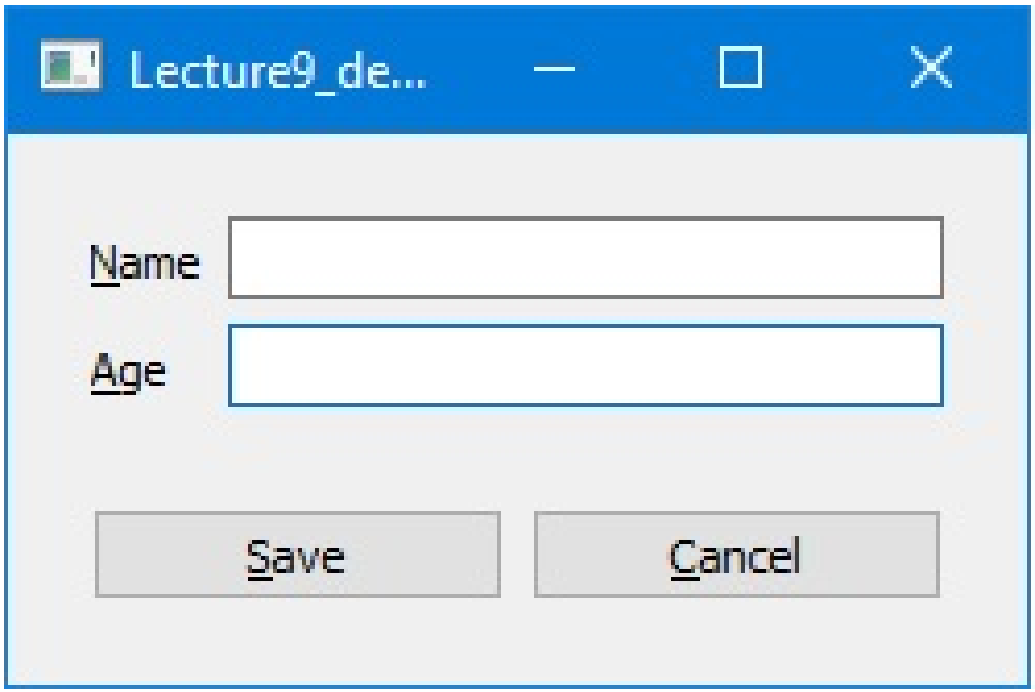


DEMO

QGridLayout (*Lecture9_demo_widgets* - function *gridLayout*).

Layout and widgets' combinations

- Multiple widgets can be nested and different layouts can be used to create a GUI.



DEMO

Multiple layouts (*Lecture9_demo_widgets* - function *multipleLayouts*).

Key benefits of using layout managers I

- They provide a consistent behavior across different screen sizes and styles.
- Layout managers handle resize operations.
- They automatically adapt to different fonts and platforms. If the user changes the systems font settings, the applications forms will respond immediately, resizing themselves if necessary.

Key benefits of using layout managers II

- They automatically adapt to different languages. If the applications user interface is translated to other languages, the layout classes take into consideration the widgets translated contents to avoid text truncation.
- If a widget is added to or removed from a layout, the layout will automatically adapt to the new situation (the same thing happens when applying the *show()* or *hide()* functions for a widget).

Absolute positioning I

- An absolute position can be specified for a widget using the function setGeometry(), which builds a rectangle using the given parameters (x and y positions, width and height).

Absolute positioning disadvantages

- If the window is resized, the widgets with absolute positions remain unchanged.
- Some text may be truncated (large font or change in the labels).
- The positions and sizes must be calculated manually (error-prone, hard to maintain).

Absolute positioning II

DEMO

Absolute positioning (*Lecture_8_widgets* - functions *createAbsolute* and *createWithLayout*).

Qt Designer I

- Qt Designer is the Qt tool for designing and building GUIs.
- Windows and dialogs can be designed in a what-you-see-is-what-you-get manner.
- Objects can be dragged from the widget box and dropped on the form.
- Object properties can be modified interactively.

Qt Designer II

- Using the Qt Designer can be faster than hand-coding the interface.
- One can experiment with different designs quickly.
- A **.ui** file is created, representing the widget tree of the form in XML format. The *User Interface Compiler (uic)* can then be used to create a corresponding C++ header file

DEMO

Qt Designer (*Lecture_8_qt_designer*).

When should we write code programatically?

- When the elements in the dialog must change dynamically.
- When we want to use custom widgets.

How?

1. Create a new class, by inheriting from QWidget.
2. Implement the GUI.
3. Show the newly created widget.

DEMO

Dynamically changing elements (*Lecture_8_qt_designer* - *ProgramaticallyDesignedWidget* class).

Common pattern to build a GUI

1. Instantiate the required Qt widgets.
2. Set properties for these, if necessary.
3. Add the widgets to a layout (the layout manager will take care of the position and size).
4. Connect the widgets using the signal and slot mechanism (will be presented next week).