Qt 5 Application Framework
Gnuplot Plotting Library

Δρ. Γ. Χαμόδρακας

Qt / Gnuplot

- **Qt:** a cross-platform application framework
  - Main use: C++ application development with Graphical User Interfaces
  - Current version: Qt 5
- **Gnuplot:** a program generating 2 and 3-dimensional plots of functions or data
  - Can be built as a library providing QtGnuplotWidget class

Qt 5 installation

Ubuntu distribution:

```
$sudo apt-get install build-essential mesa-common-dev
$sudo apt-get install libglu1-mesa-dev -y
$sudo apt-get install libxt-dev qt5-default qtquick1-5-dev
qtdeclarative5-dev qtscript5-dev libqt5webkit5-dev qttools5-dev-tools
qtcreator
$sudo apt-get install ^libqt5svg5
```

If Ubuntu runs as a VirtualBox Appliance 3d acceleration must be disabled.

Gnuplot Installation (1)

Build Gnuplot as a library

```
$sudo apt-get install automake libtool cvs
$cvs -d:pserver:anonymous@gnuplot.cvs.sourceforge.net:/cvsroot/gnuplot login
$cvs -z3 -d:pserver:anonymous@gnuplot.cvs.sourceforge.net:/cvsroot/gnuplot co -P
gnuplot
$cd gnuplot
$wget http://sourceforge.net/p/gnuplot/patches/_discuss/thread/47917802/fbe5/attachment/qtlib6.patch
$patch -p0 < qtlib6.patch
$libtoolize
$./prepare
$./configure --with-qt=lib
$export QT_PATH=/usr
$export UIC=$QT_PATH/bin/uic
$export MOC=$QT_PATH/bin/moc
$export RCC=$QT_PATH/bin/rcc
$export CXX=g++
$export CC=gcc
```

Gnuplot Installation (2)

```
$export QT_LIBS="-Wl,--no-as-needed /usr/lib/x86_64-linux-gnu/libQt5Concurrent.so \
/usr/lib/x86_64-linux-gnu/libQt5Core.so /usr/lib/x86_64-linux-gnu/libQt5DBus.so \
/usr/lib/x86_64-linux-gnu/libQt5Declarative.so /usr/lib/x86_64-linux-gnu/libQt5Gui.so \
/usr/lib/x86_64-linux-gnu/libQt5Network.so /usr/lib/x86_64-linux-gnu/libQt5OpenGL.so \
/usr/lib/x86_64-linux-gnu/libQt5PrintSupport.so \
/usr/lib/x86_64-linux-gnu/libQt5Qml.so \
/usr/lib/x86_64-linux-gnu/libQt5QuickParticles.so \
/usr/lib/x86_64-linux-gnu/libQt5Quick.so /usr/lib/x86_64-linux-gnu/libQt5QuickTest.so \
/usr/lib/x86_64-linux-gnu/libQt5QuickWidgets.so \
/usr/lib/x86_64-linux-gnu/libQt5Script.so \
/usr/lib/x86_64-linux-gnu/libQt5ScriptTools.so /usr/lib/x86_64-linux-gnu/libQt5Sql.so \
/usr/lib/x86_64-linux-gnu/libQt5Svg.so /usr/lib/x86_64-linux-gnu/libQt5Test.so \
/usr/lib/x86_64-linux-gnu/libQt5WebKit.so \
/usr/lib/x86_64-linux-gnu/libQt5WebKitWidgets.so \
/usr/lib/x86_64-linux-gnu/libQt5Widgets.so /usr/lib/x86_64-linux-gnu/libQt5Xml.so"
```

Gnuplot Installation (3)

```
$export QT_CFLAGS="-F$QT_PATH -I/usr/include/x86_64-linux-gnu/qt5 \
-I/usr/include/x86_64-linux-gnu/qt5/QtConcurrent \
-I/usr/include/x86_64-linux-gnu/qt5/QtCore \ -I/usr/include/x86_64-linux-gnu/qt5/QtDBus \
-I/usr/include/x86_64-linux-gnu/qt5/QtDeclarative \
-I/usr/include/x86_64-linux-gnu/qt5/QtGui -I/usr/include/x86_64-linux-gnu/qt5/QtNetwork \
-I/usr/include/x86_64-linux-gnu/qt5/QtOpenGL \
-I/usr/include/x86_64-linux-gnu/qt5/QtOpenGLExtensions \
-I/usr/include/x86_64-linux-gnu/qt5/QtPlatformHeaders \
-I/usr/include/x86_64-linux-gnu/qt5/QtPrintSupport \
-I/usr/include/x86_64-linux-gnu/qt5/QtQml -I/usr/include/x86_64-linux-gnu/qt5/QtQuick \
-I/usr/include/x86_64-linux-gnu/qt5/QtQuickParticles \
-I/usr/include/x86_64-linux-gnu/qt5/QtQuickTest \
-I/usr/include/x86_64-linux-gnu/qt5/QtQuickWidgets \
-I/usr/include/x86_64-linux-gnu/qt5/QtScript \
-I/usr/include/x86_64-linux-gnu/qt5/QtScriptTools \
-I/usr/include/x86_64-linux-gnu/qt5/QtSql -I/usr/include/x86_64-linux-gnu/qt5/QtTest \
-I/usr/include/x86_64-linux-gnu/qt5/QtWebKit \
-I/usr/include/x86_64-linux-gnu/qt5/QtWebKitWidgets \
-I/usr/include/x86_64-linux-gnu/qt5/QtWidgets -I/usr/include/x86_64-linux-gnu/qt5/QtSvg \
-I/usr/include/x86_64-linux-gnu/qt5/QtXml"
```

Gnuplot Installation (4)

- Hack the qtterminal of Gnuplot:

Create a public function `QLabel * getStatusLabel() const` returning `m_statusLabel` private member in `QtGnuplotWidget` class (`src/qtterminal/QtGnuplotWidget.h & src/qtterminal/QtGnuplotWidget.cpp`)

add `#include <cmath>` in `src/qtterminal/qt_conversion.cpp`

delete demo subdir from SUBDIRS variable in Makefile with editor (in Gnuplot root path)

Run the following commands within Gnuplot root path:

```
$make
$sudo make install
```

download QtExample.zip from e-class and unzip in a proper dir

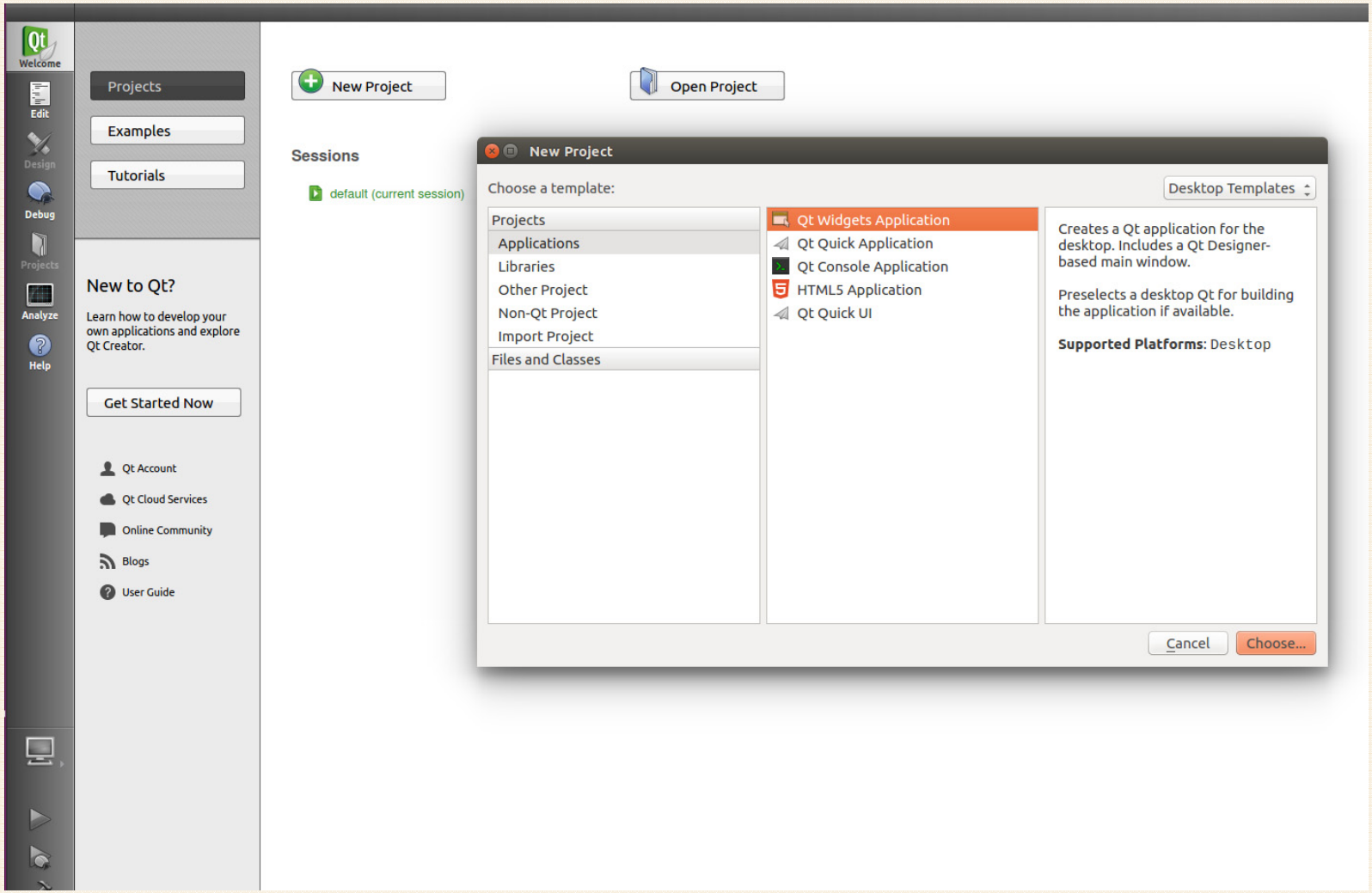
```
$qtcreator
```

open QtExample project

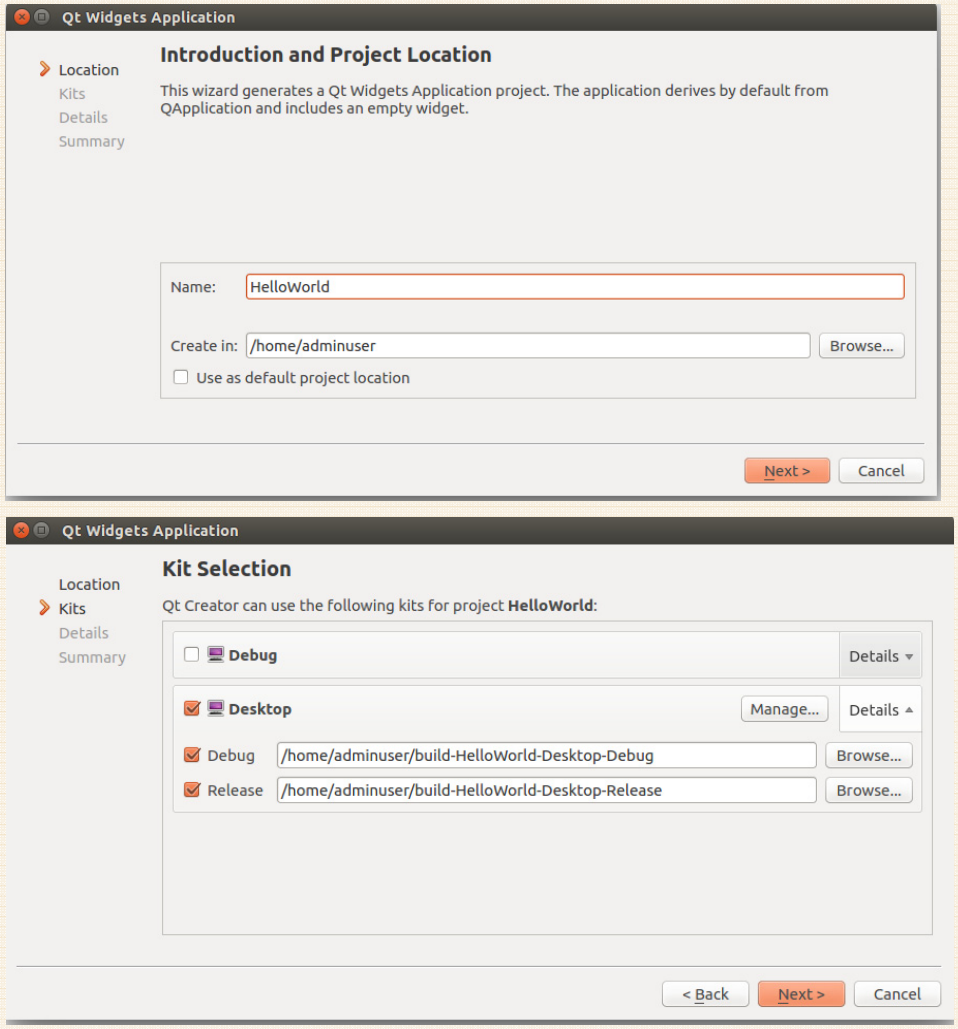
Qt 5 Tutorial

- **Qt modules**:
  - **Qt Library**: C++ class library providing application building blocks and GUI widget classes
  - **Qt Creator:** An IDE for Qt Applications
  - **Qt Designer:** A tool of Qt Creator for designing GUIs, generating .ui xml files
  - **qmake:** tool for generating Makefile
  - There is a plug-in for Eclipse IDE

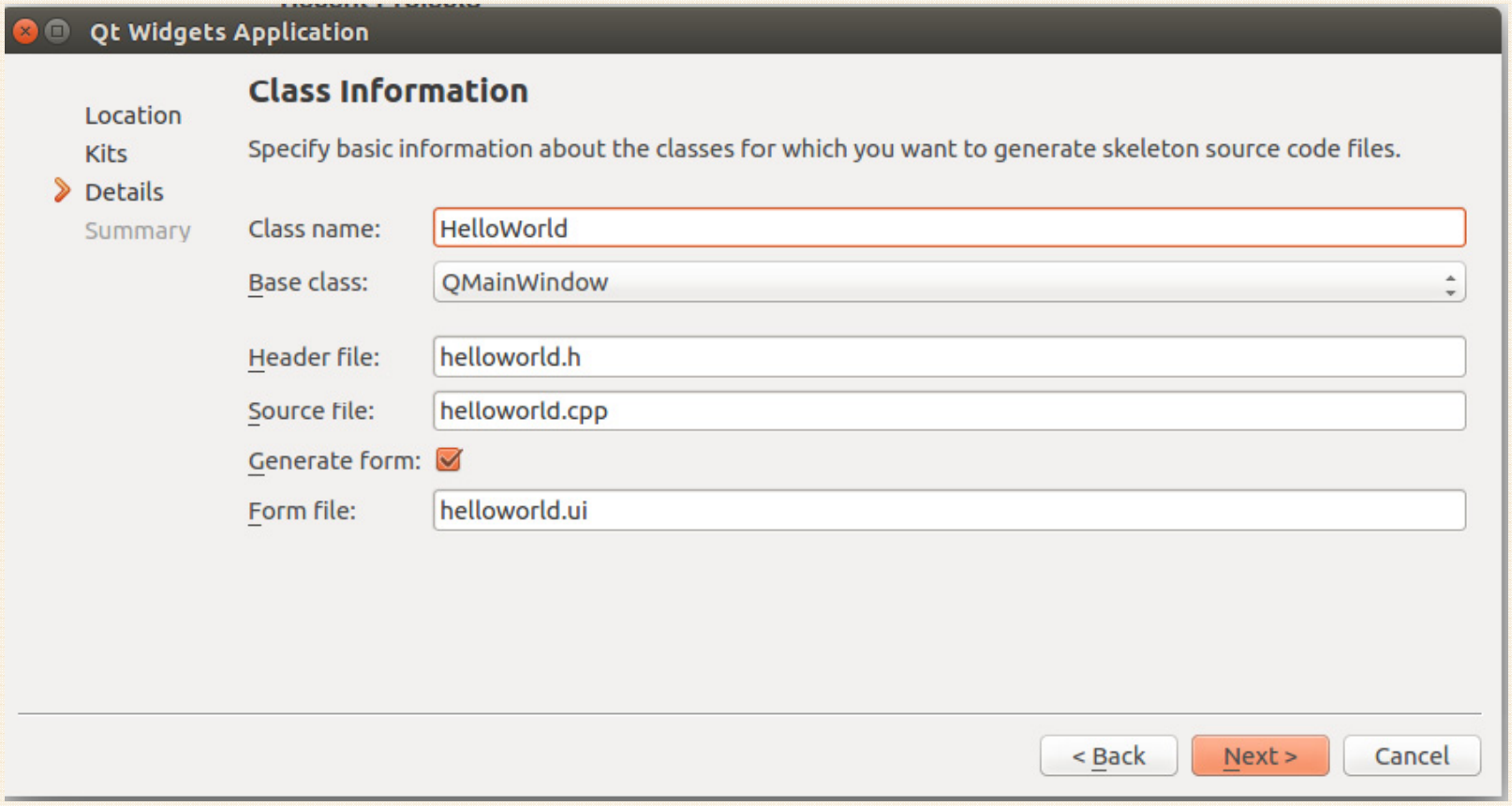
Qt Creator: Hello world project



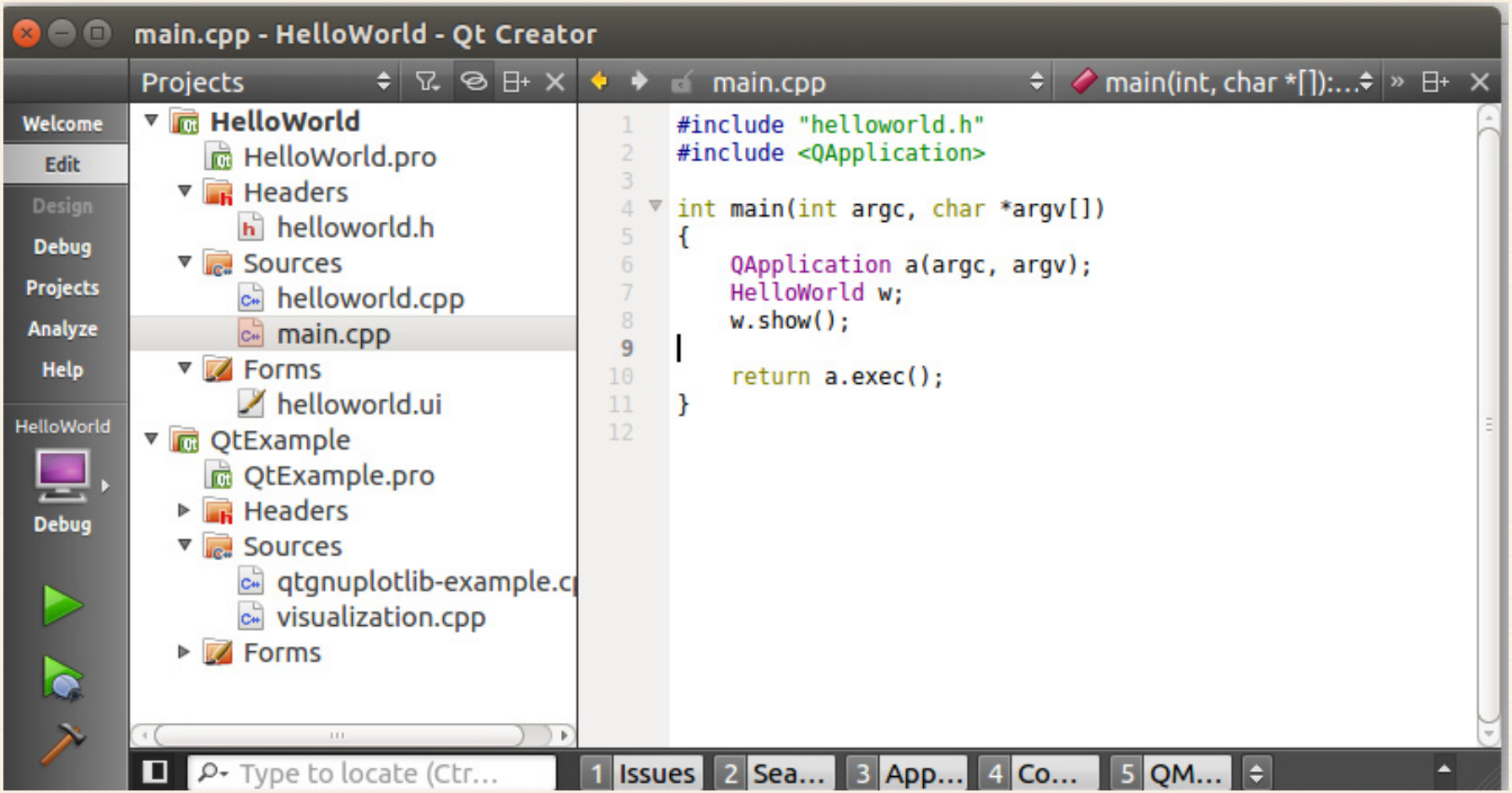
Hello World project (cont.d)



Hello World project (cont.d)



Hello World: main function



Hello World: main function (cont.d)

- The QApplication object contains the main event loop, where all events from the window system, the keyboard, the mouse, etc. are processed and dispatched.
- There is only one QApplication object in a Qt GUI application.
- app.exec(); makes the QApplication enter is event loop.
- The application is compiled and executed with the first green “Play” button.
- The “Play” button with the bug initiates a debugging session.
- The “Hammer” button builds the project
- In order to build the project from command line run:

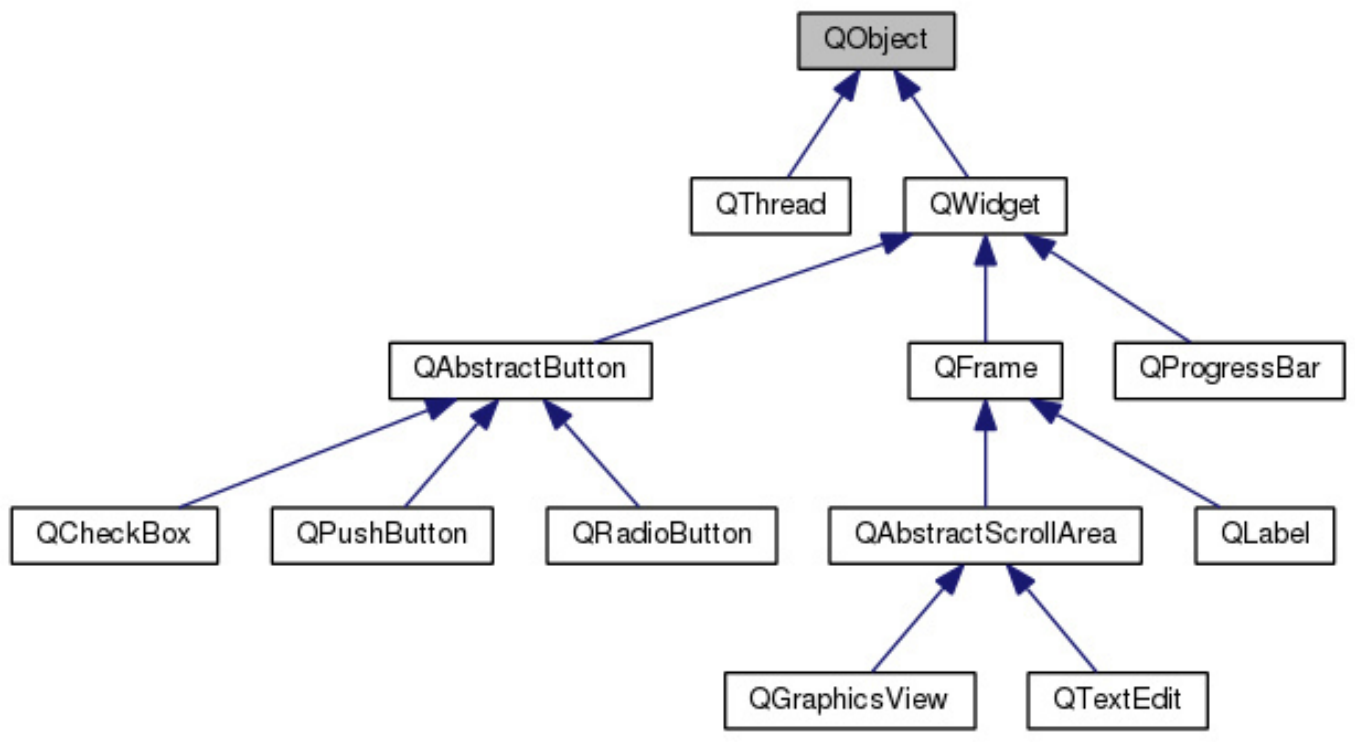
  `$qmake` (generates the Makefile)

  `$make`

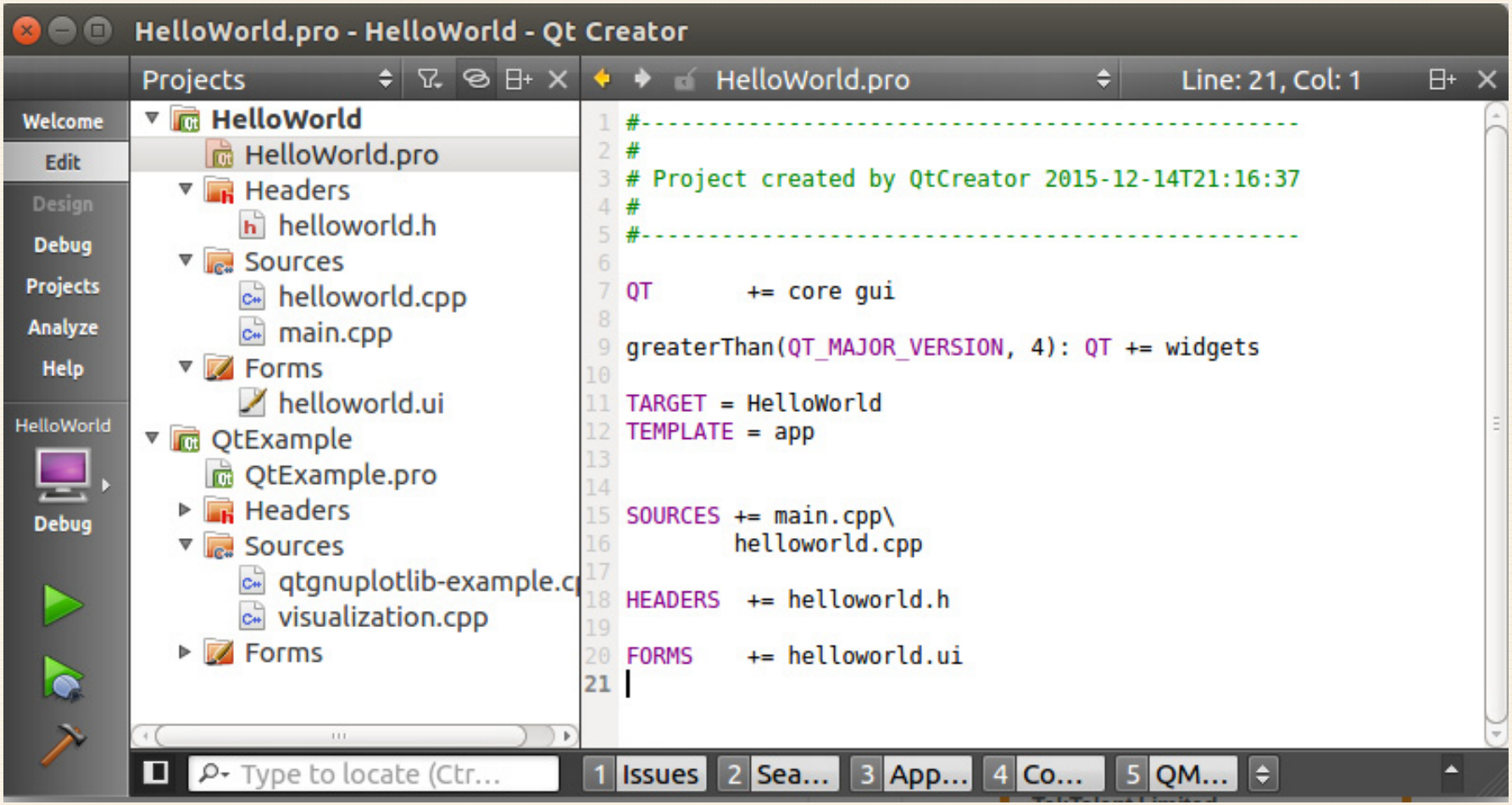
Hello World: main function (cont.d)

- A Qt GUI component is defined in the main class after the QApplication
- Usually the main window of the app is defined here
- HelloWorld class inherits QMainWindow class
- The main window is shown with the call: w.show();
- QMainWindow inherits QWidget class

Qt Class Hierarchy (fragment)



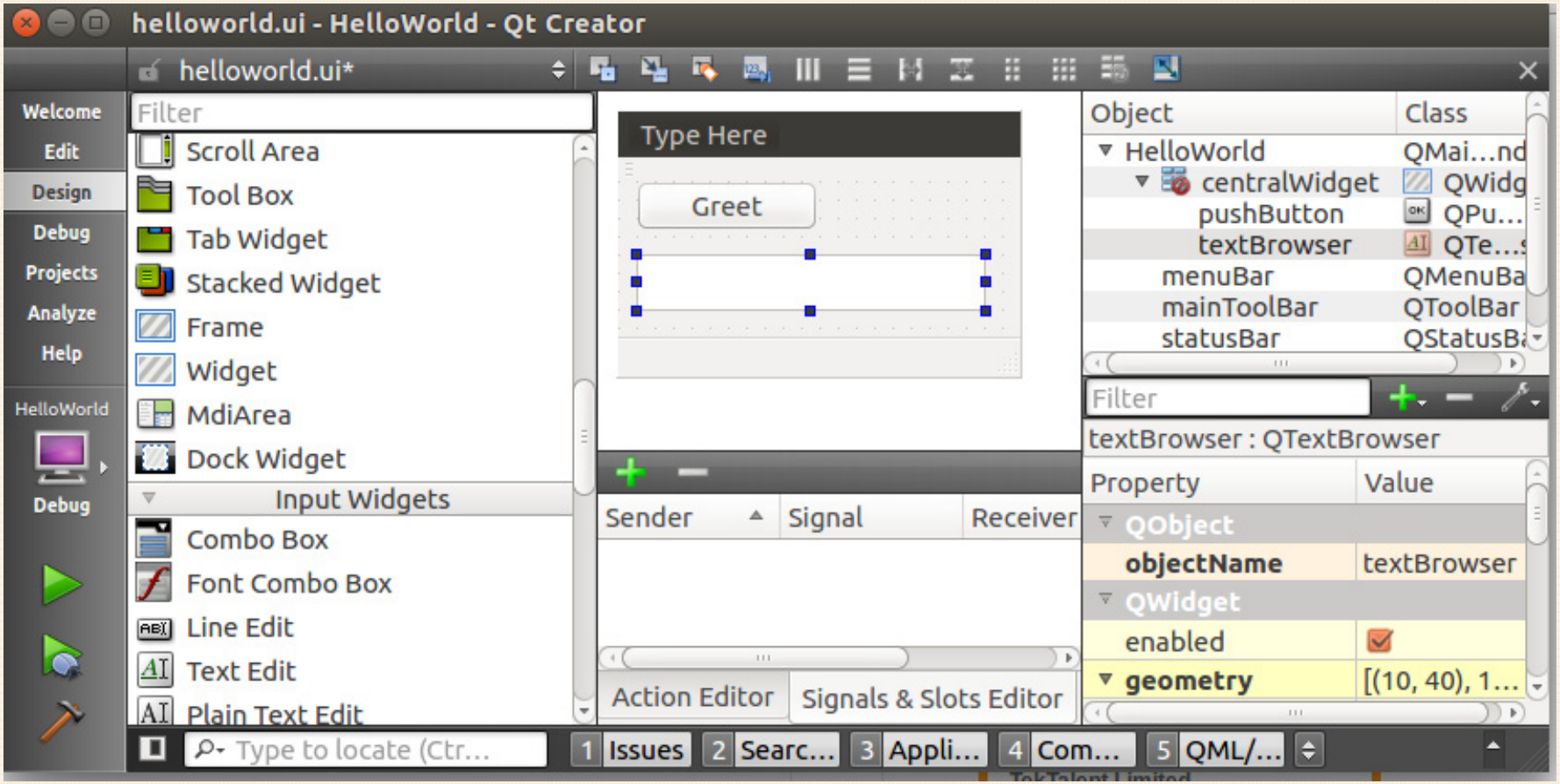
Project configuration file



Project configuration file (cont.d)

- QT: indicates what libraries are used in the project
- TARGET: indicates the name of the application
- TEMPLATE: indicates the build type (app or library)
- SOURCES / HEADERS: source and header files of the app (apart from generated ones)
- FORMS: xml files representing the GUI structure of a widget

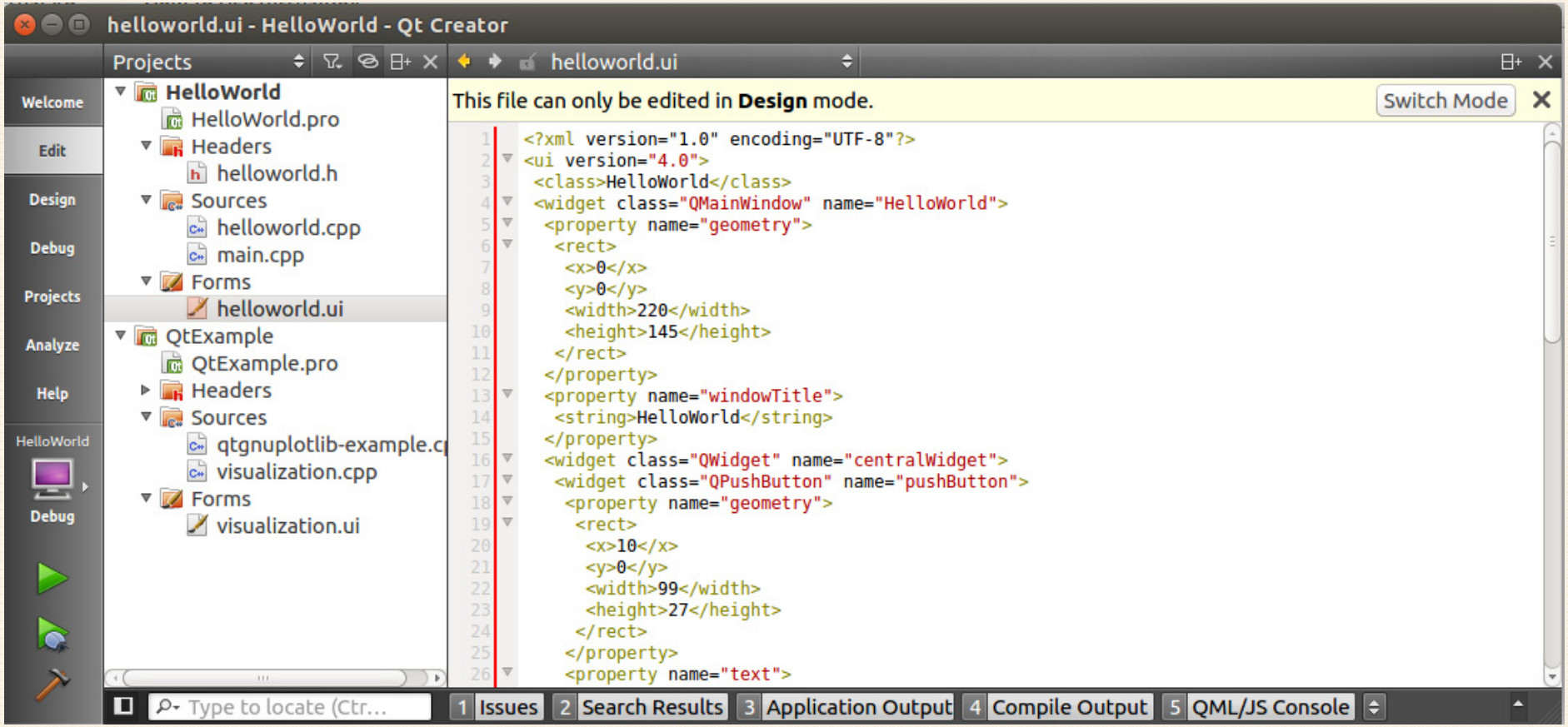
Qt Designer


helloworld.ui - HelloWorld - Qt Creator
helloworld.ui*
Welcome
Edit
Design
Debug
Projects
Analyze
Help
HelloWorld
Debug
Filter
Scroll Area
Tool Box
Tab Widget
Stacked Widget
Frame
Widget
MdiArea
Dock Widget
Input Widgets
Combo Box
Font Combo Box
Line Edit
Text Edit
Plain Text Edit
Type Here
Greet
Object
Class
HelloWorld
centralWidget
pushButton
textBrowser
menuBar
mainToolBar
statusBar
Filter
textBrowser : QTextBrowser
Property
Value
QObject
objectName
textBrowser
QWidget
enabled
geometry
Sender
Signal
Receiver
Action Editor
Signals & Slots Editor
Type to locate (Ctr...
1 Issues
2 Searc...
3 Appli...
4 Com...
5 QML/...

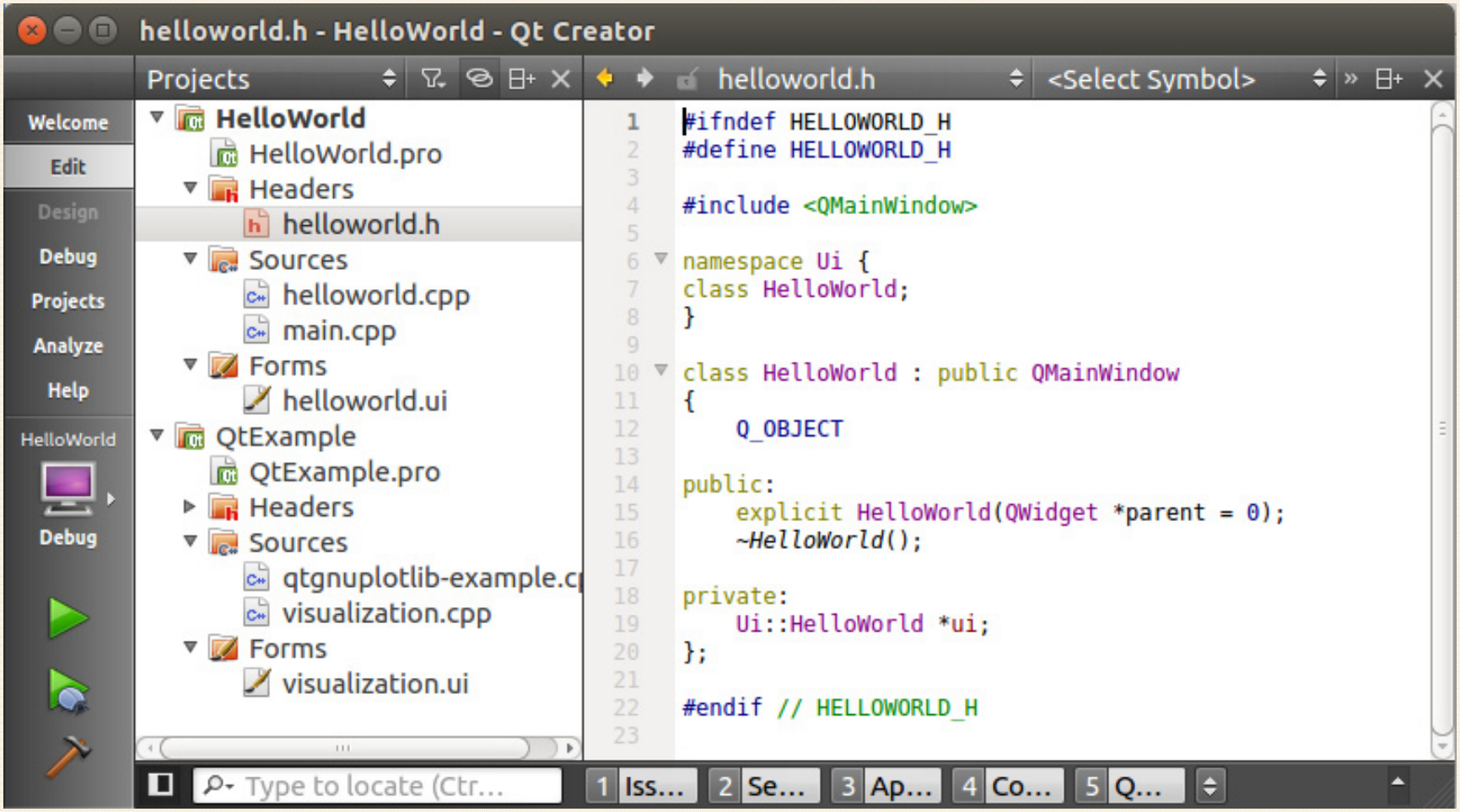
Qt Designer

- GUI components (widgets) can be added to a window with Qt Designer.
- Widgets are added with drag and drop.
- Widget design properties can be set by the property editor shown in the lower left side of Qt Designer view.

Generated helloworld.ui file



HelloWorld class definition



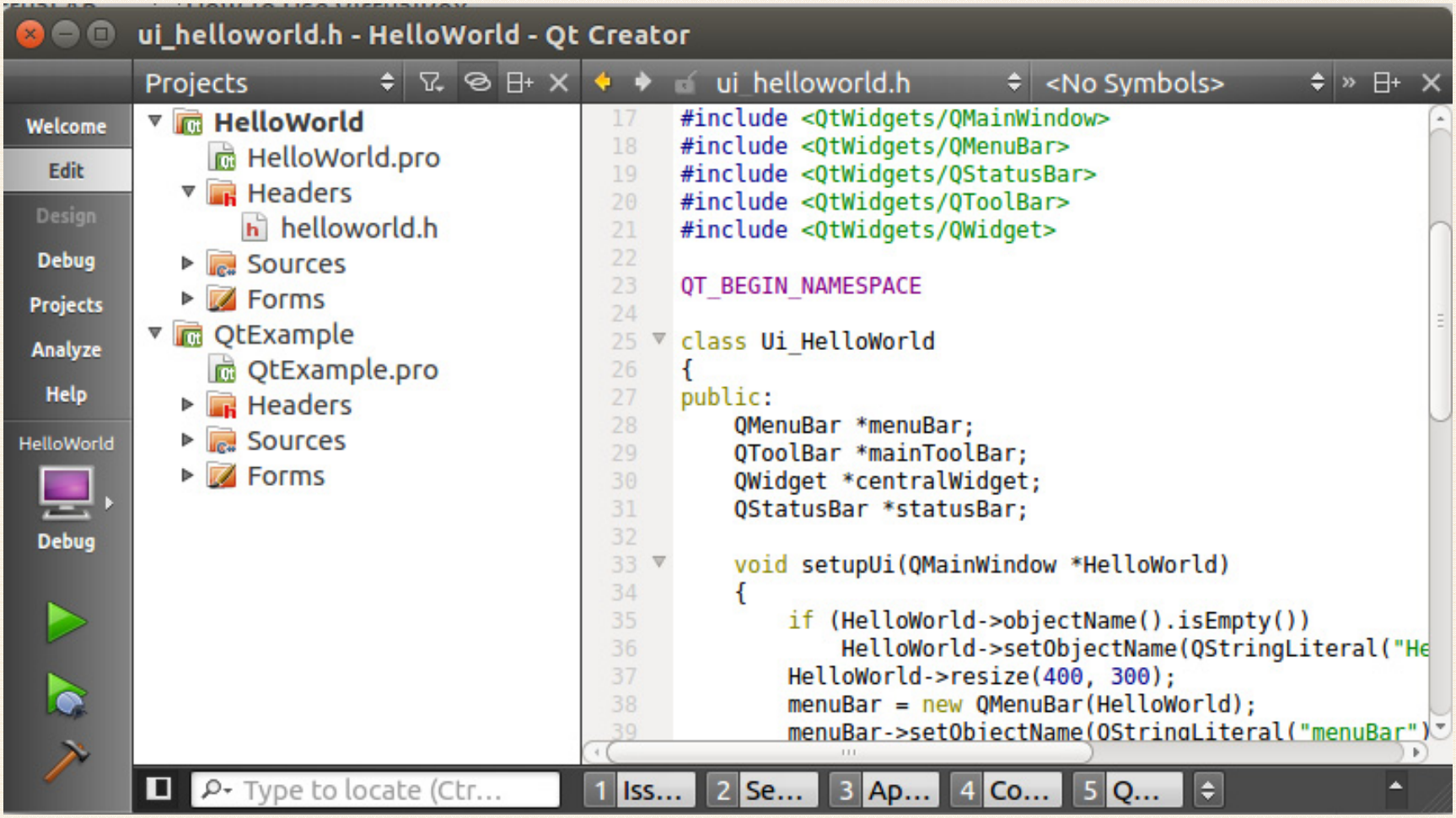
HelloWorld class definition (cont.d)

- Qt extends C++ with a meta-object system allowing run-time introspection which enables the signal-slot object communication mechanism, dynamic properties, etc.
- GUI components must contain the Q_OBJECT macro to support the extensions.
- The moc meta-object compiler reads the C++ header file. If it finds one or more class declarations that contain the Q_OBJECT macro, it produces a C++ source file containing the meta-object code for those classes.

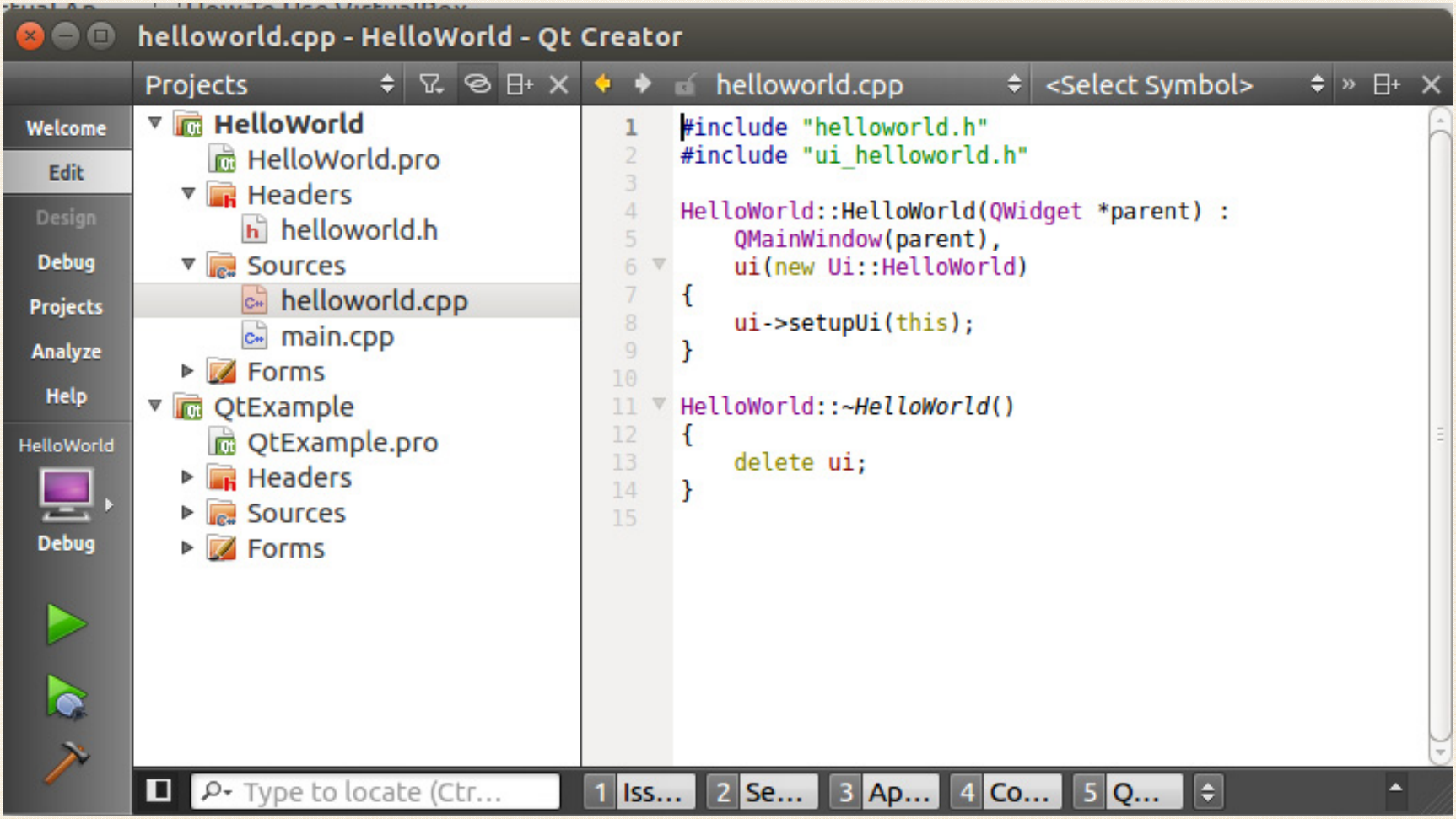
HelloWorld class definition (cont.d)

- The explicit keyword in the constructor prevents implicit conversions to instantiate an object
- The constructor argument defines the parent of a QObject. Thus, object hierarchies are created.
- Object hierarchies are part of Qt memory management. When a parent QObject is deleted, all its children in the hierarchy are also deleted.
- Child GUI components are rendered in the same window.
- The code for the Ui::HelloWorld class is generated by uic (User Interface Compiler) according to the ui xml file.

Generated UI header file



HelloWorld class implementation



HelloWorld implementation (cont.d)

- The constructor of HelloWorld invokes the constructors of QMainWindow and Ui::HelloWorld generated class.
- The User Interface components are initialized with setupUi(this) invocation.

Signals and Slots

- The communication between objects is performed through the signal-slot mechanism.
- Signals are emitted from objects when specific events happen.
- Qt widgets have predefined signals. Widgets are subclassed to add user-defined signals.
- Signals are public access functions in a QObject with no return types (return void).
- It is recommended to emit them only from the class itself and its subclasses .
- Signals are automatically generated by the meta-object compiler and must not be implemented in the .cpp file

Signals and Slots

- Slots are normal member C++ functions declared as slots.
- A slot is called when a signal connected to it is emitted.
- Slots are called when a connected signal is emitted regardless of their access level (even if private).
- The signature of a signal must match the signature of the receiving slot.
- Signals are connected with slots through the call of:

  QObject::connect(const QObject * *sender*, const char * *signal*, const QObject * *receiver*, const char * *method*, Qt::ConnectionType *type* = Qt::AutoConnection)
- There are slots automatically connected with signals through the user interface compiler (according to standard naming conventions)

Signal / Slot example

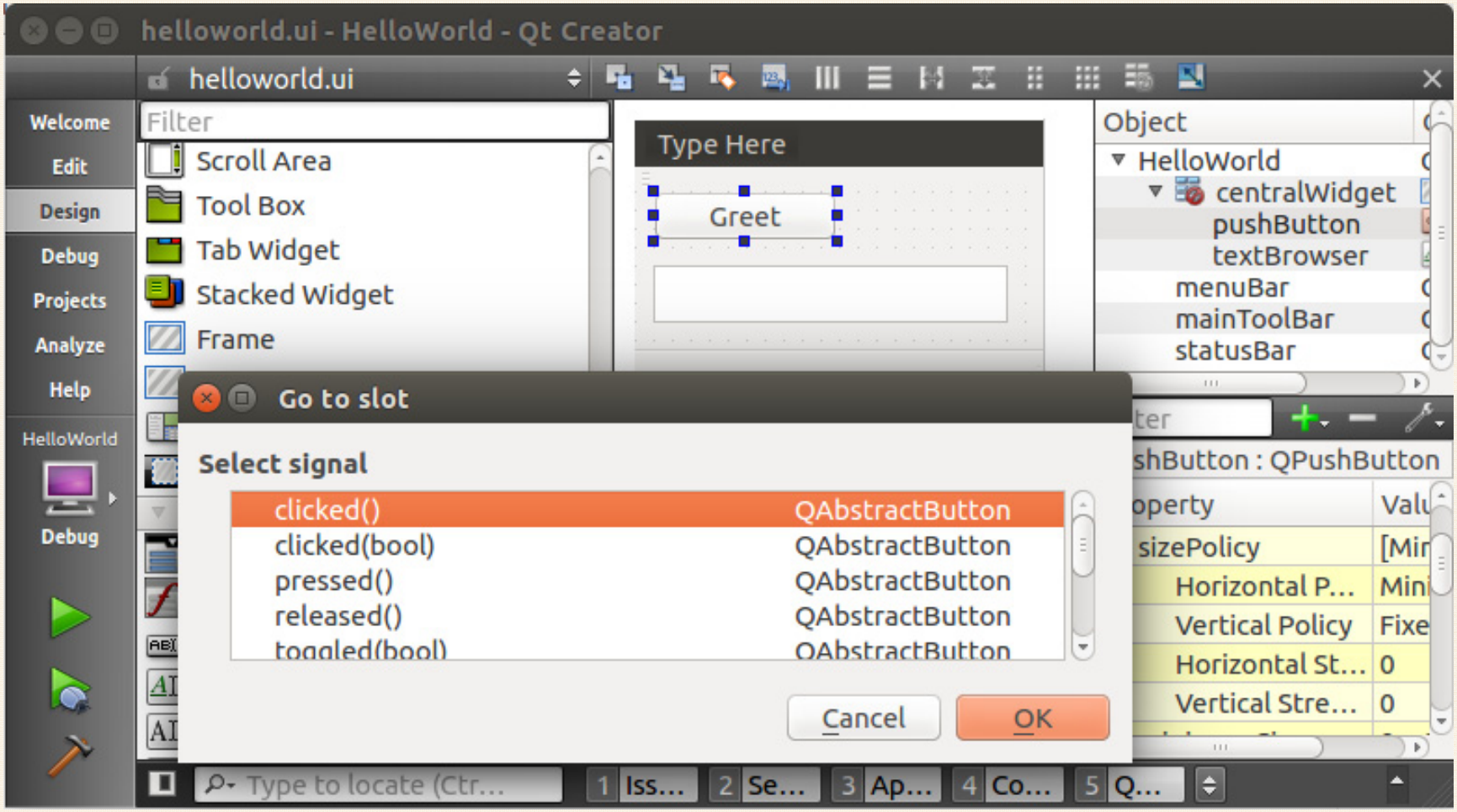
```
#include <QObject>

class Counter : public QObject {
    Q_OBJECT
public: Counter() {
    m_value = 0;
}
int value() const {
   return m_value;
}
public slots:
   void setValue(int value);
signals:
   void valueChanged(int newValue);
private:
   int m_value;
};
```

```
void Counter::setValue(int value)
{
    if (value != m_value) {
        m_value = value;
        emit valueChanged(value);
    }
}
...
Counter a, b;
QObject::connect(&a,&Counter::valueChanged,
                 &b, &Counter::setValue);
/*a.value() == 12, b.value() == 12*/
a.setValue(12);
/* a.value() == 12, b.value() == 48*/
b.setValue(48);

...
```

Signal / Slot automatic connection



Signal / Slot automatic connection (2)

```
#ifndef HELLOWORLD_H
#define HELLOWORLD_H

#include <QMainWindow>

namespace Ui {
class HelloWorld;
}

class HelloWorld : public QMainWindow
{
    Q_OBJECT

public:
    explicit HelloWorld(QWidget *parent = 0);
    ~HelloWorld();

private slots:
    void on_pushButton_clicked();

private:
    Ui::HelloWorld *ui;
};

#endif // HELLOWORLD_H
```

```
#include "helloworld.h"
#include "ui_helloworld.h"

HelloWorld::HelloWorld(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::HelloWorld)
{
    ui->setupUi(this);
}

HelloWorld::~HelloWorld()
{
    delete ui;
}

void HelloWorld::on_pushButton_clicked()
{
    ui->textBrowser->setText(QString("Hello World!"));
}
```

Event system

- In Qt, events are objects, derived from the abstract QEvent class, that represent things that have happened either within an application or as a result of outside activity that the application needs to know about.
- Standard event classes: `QResizeEvent, QPaintEvent, QMouseEvent, QKeyEvent, and QCloseEvent`
- The normal way for an event to be delivered is by calling a virtual function. If you do not perform all the necessary work in your implementation of the virtual function, you may need to call the base class's implementation.
- Events are delivered to a specific QObject according to their context.

Event handler example

```
void MyCheckBox::mousePressEvent(QMouseEvent *event)
{
    if (event->button() == Qt::LeftButton) {
        // handle left mouse button here
    } else {
        // pass on other buttons to base class
        QCheckBox::mousePressEvent(event);
    }
}
```

Event filters

- Sometimes an object needs to look at, and possibly intercept, the events that are delivered to another object.

```
Visualization::Visualization(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::Visualization)
{
    ui->setupUi(this);
    widget = new QtGnuplotWidget();
    widget->installEventFilter(this);
    widget->setStatusLabelActive(true);
    instance = new QtGnuplotInstance();
    instance->setWidget(widget);
}
```

Event filters (2)

```
bool Visualization::eventFilter(QObject *obj, QEvent *event)
{
    if (event->type() == QEvent::MouseButtonPress)
    {
        if (obj == this->widget) {
            QMouseEvent *mouseEvent = static_cast<QMouseEvent *>(event);
            if (mouseEvent->button() == Qt::LeftButton) {
                ui->outputTxt->setText( this->widget->getStatusLabel()->text());
            }
        }
    }
    return QObject::eventFilter(obj, event);
}
```

QtGnuplot

- QtGnuplot library provides 2 basic classes:
  - `QtGnuplotWidget` (subclass of `QWidget`)
  - `QtGnuplotInstance`
- The first class is a QWidget where Gnuplot plots are painted.
- The second class is an interface to the Gnuplot program

QtGnuplot example

```h
#ifndef VISUALIZATION_H
#define VISUALIZATION_H

#include <QMainWindow>
#include <QtGnuplot/QtGnuplotWidget.h>
#include <QtGnuplot/QtGnuplotInstance.h>

namespace Ui {
class Visualization;
}

class Visualization : public QMainWindow
{
    Q_OBJECT

public:
    explicit Visualization(QWidget *parent = 0);
    ~Visualization();

private:
    QtGnuplotWidget *widget;
    QtGnuplotInstance *instance;

protected:
    bool eventFilter(QObject *obj, QEvent *event);


private slots:
    void on_actionExit_triggered();

    void on_actionOpen_triggered();

    void on_pushButton_clicked();

private:
    Ui::Visualization *ui;
};

#endif // VISUALIZATION_H
```

```cpp
Visualization::Visualization(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::Visualization)
{
    ui->setupUi(this);
    widget = new QtGnuplotWidget();
    widget->installEventFilter(this);
    widget->setStatusLabelActive(true);
    instance = new QtGnuplotInstance();
    instance->setWidget(widget);
}
```

```cpp
bool Visualization::eventFilter(QObject *obj, QEvent *event)
{
    if (event->type() == QEvent::MouseButtonPress)
    {
        if (obj == this->widget) {
            QMouseEvent *mouseEvent = static_cast<QMouseEvent *>(event);
            if (mouseEvent->button() == Qt::LeftButton) {
                ui->outputTxt->setText( this->widget->getStatusLabel()->text());
            }
        }
    }
    return QObject::eventFilter(obj, event);
}
```

QtGnuplot example (2)

```
void Visualization::on_pushButton_clicked()
{
    widget->show();
    widget->resize(QSize(800,600));
    *instance <<\
    "set yrange [-1.5:1.5]\nset xrange [-1.5:1.5]\nset isosamples 500,500\nf(x,y)= x**2+y**2-1\nset contour\nset cntrparam levels discrete 0\nset view 0,0\nunset ztics\nunset surface\nsplot f(x,y)\n";
    //*instance << "set tics scale 0.75\nset xtics 1\nset ytics 1\nset yrange [-10:10]\nset xlabel 'x'\nset ylabel 'y'\nset zeroaxis\nplot \"<echo '1 2'\" notitle\n";
}
```

- The uncommented line plots a circle. In order to do that we take the contour line of the function z = f(x,y) at z = 0
- The commented line plots a point in a 2-dimensional diagram.
- Documentation concerning Gnuplot commands can be found in: http://www.gnuplot.info/docs_4.0/gpcard.pdf
- Helpful material can be found in: http://www.gnuplotting.org/plotting-single-points/ and http://www.gnuplotting.org/plotting-data/