Buổi 1: Cài đặt & làm quen với QT Creator

Vẽ các hình cơ bản bằng QT Creator

1. Cài đặt & làm quen với QT Creator

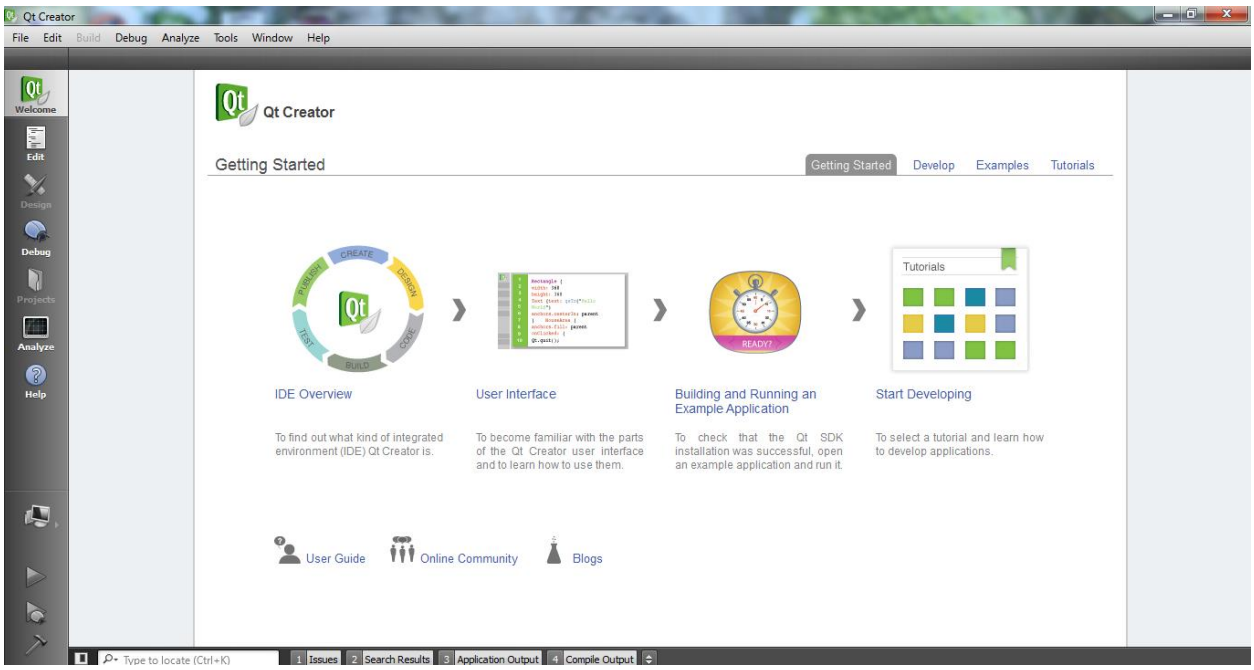
1.1. Download & Cài đặt

Qt là ứng dụng cross-platform và là framework giao diện người dùng (UI) cho các nhà phát triển sử dụng ngôn ngữ C++ hoặc QML (ngôn ngữ kiểu CSS & JavaScript). Thế mạnh của Qt là đồ họa và xử lý ảnh. OpenGL cũng được tích hợp vào trong Qt tạo nên một thư viện hoàn hảo về đồ họa máy tính. Đi kèm với Qt là Qt Creator, một môi trường phát triển tích hợp hỗ trợ cho phát triển các ứng dụng Qt. Qt được sử dụng trong các phần mềm KDE trên các hệ thống GNU/Linux. Plasma Workspaces

Download Qt từ trang http://qt-project.org/downloads. Phiên bản hiện tại của Qt là Qt 5.0. Trên Windows, nên sử dụng bản Qt 5.0.1 for Windows 32-bit (MinGW 4.7, 823 MB). Bản cài đặt này bao gồm thư viện Qt và môi trường phát triển tích hợp (IDE) Qt Creator.

1.2. Sử dụng QT Creator

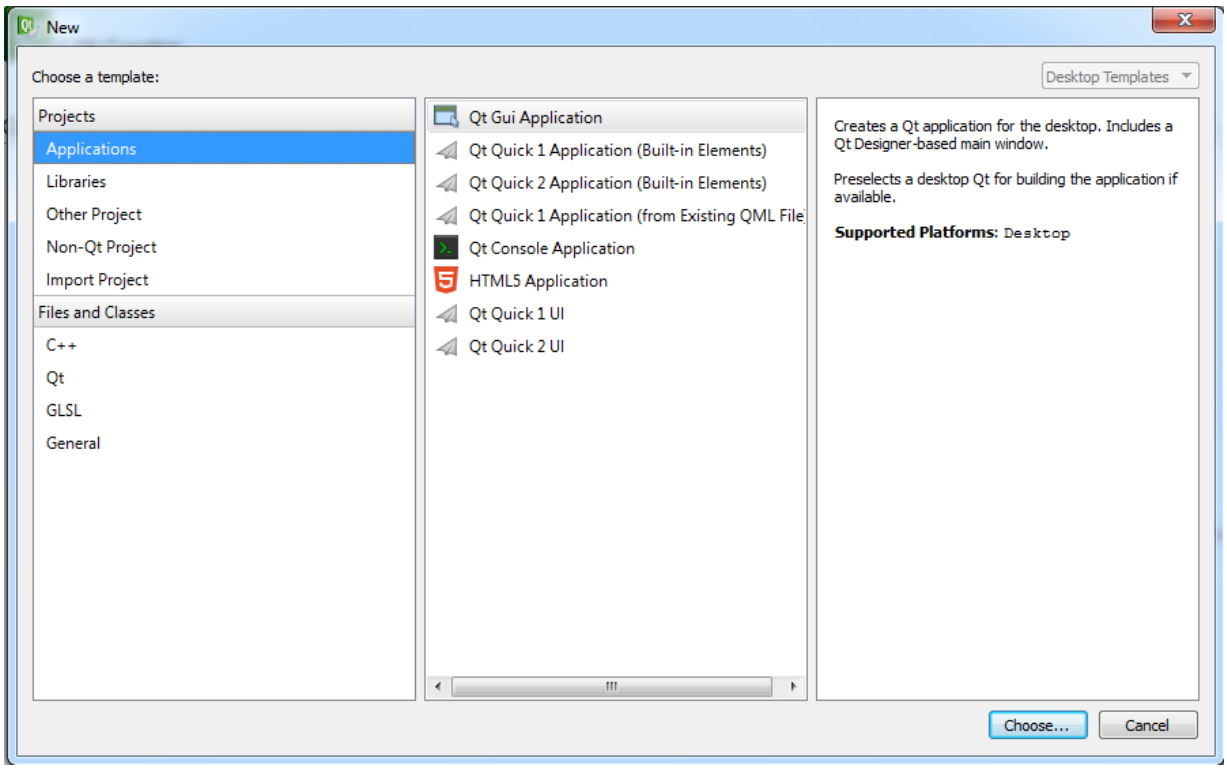
Giao diện chính của QT Creator:



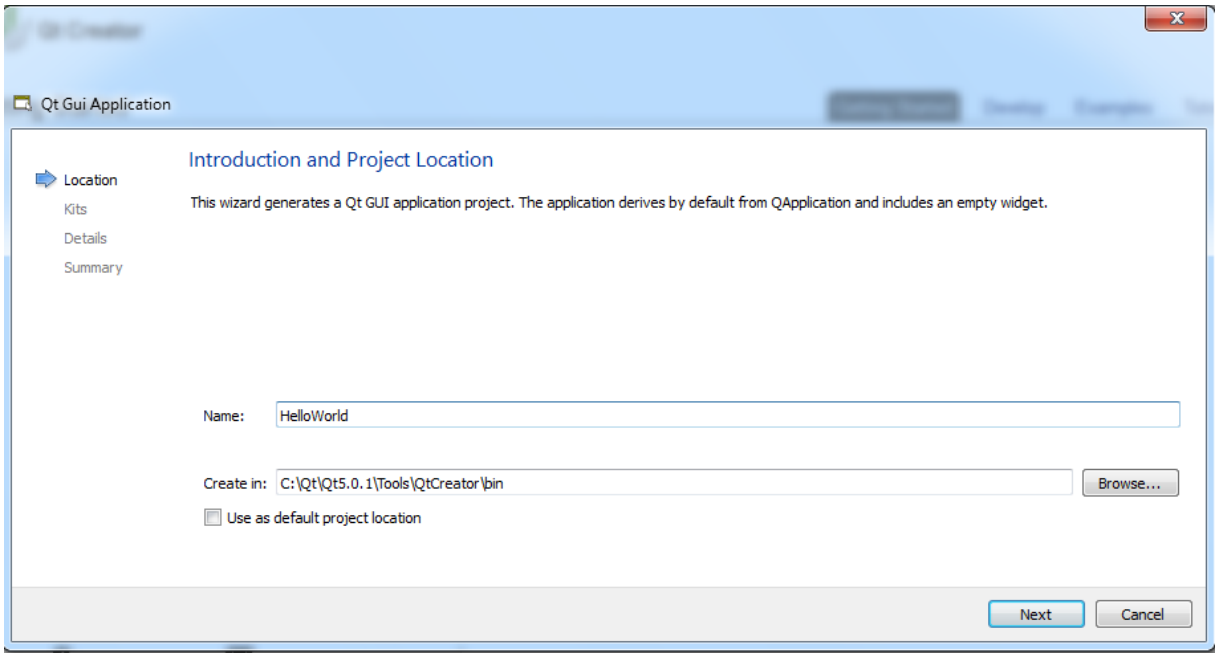
— Tạo dự án mới

Chọn Menu **File → New File Or Project** hoặc bấm tổ hợp phím **Ctrl + N.**

Chọn **Application →QT Gui Application → Choose**.



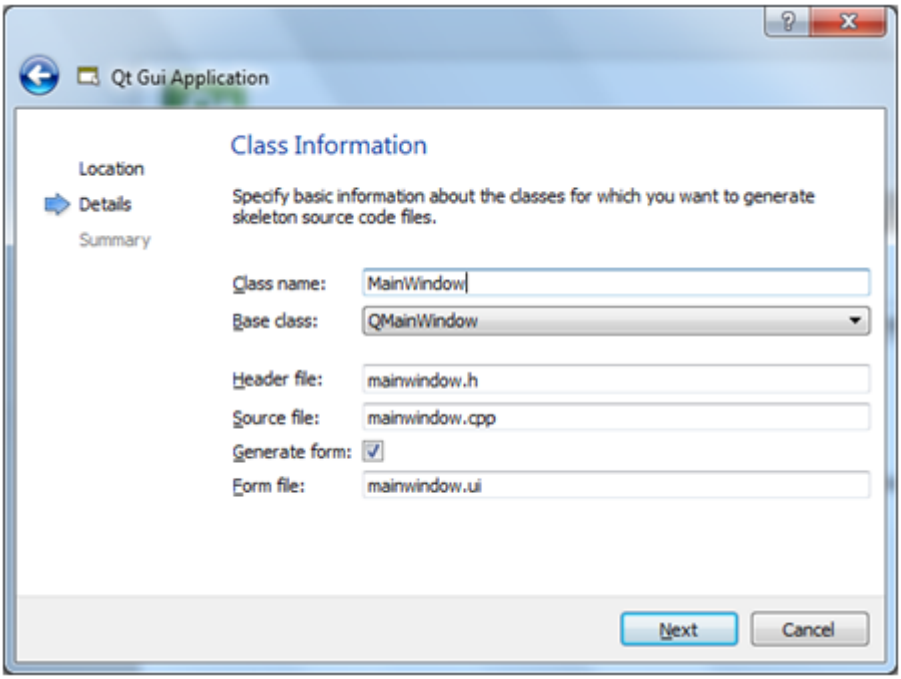
Nhập tên của dự án (Name) và thư mục chứa dự án. Sau đó click **Next**.



Chọn kiểu giao diện. Qt Creator cung cấp 3 kiểu ứng dụng giao diện đồ họa chính thừa kế từ các lớp cơ sở sau:

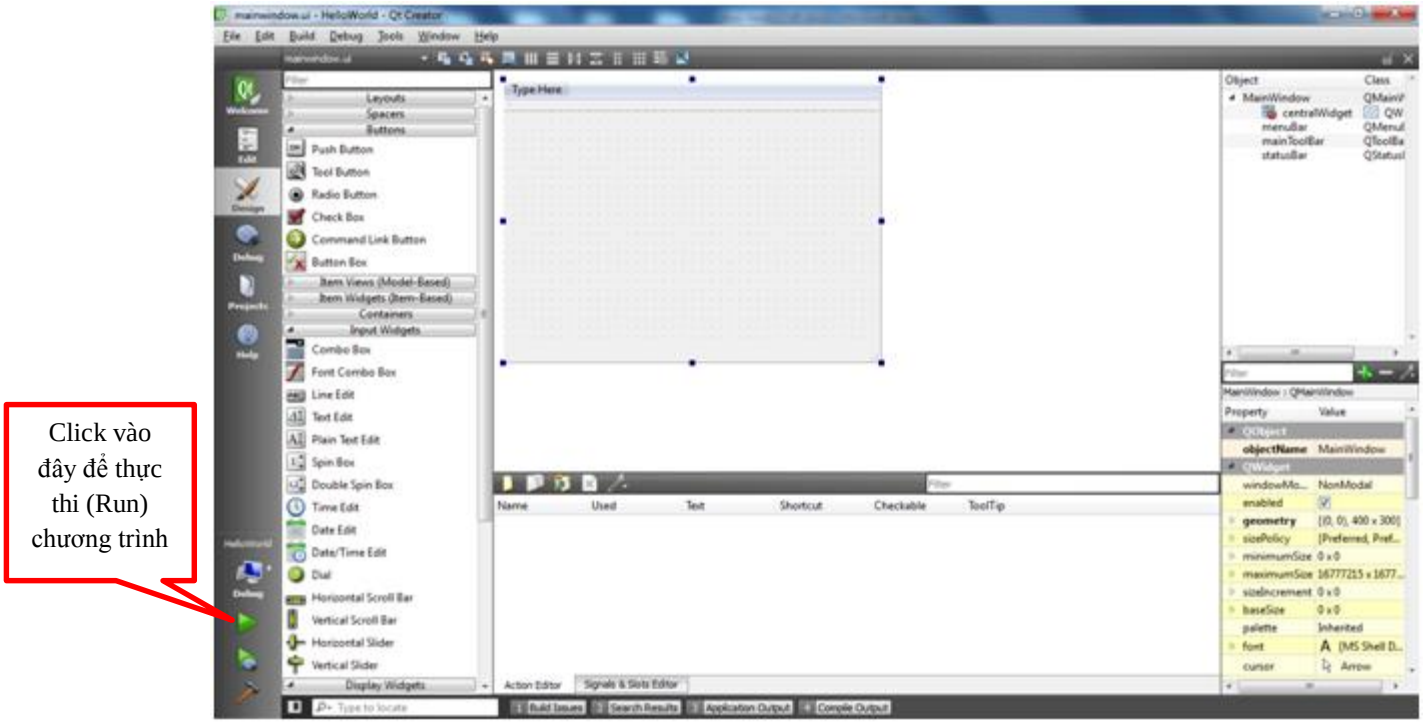
- **QMainWindow**: Kiểu giao diện này thường được sử dụng, giao diện gồm có Menu, Thanh công cụ (Toolbar), Cửa số trung tâm (Center Widget) và Thanh trạng thái (StatusBar)
- **QDialog**: Giao diện ứng dụng kiểu hộp hội thoại.
- **QWidget**: (Dành cho người dùng chuyên nghiệp), người dùng có thể tùy biến giao diện theo ý thích, nhưng phải lập trình khá nhiều.

— Nhập tên lớp cửa số chính (Class name): mặc định Qt Creator đặt tên lớp chính là MainWindow nếu lớp cơ sở là QMainWindow.



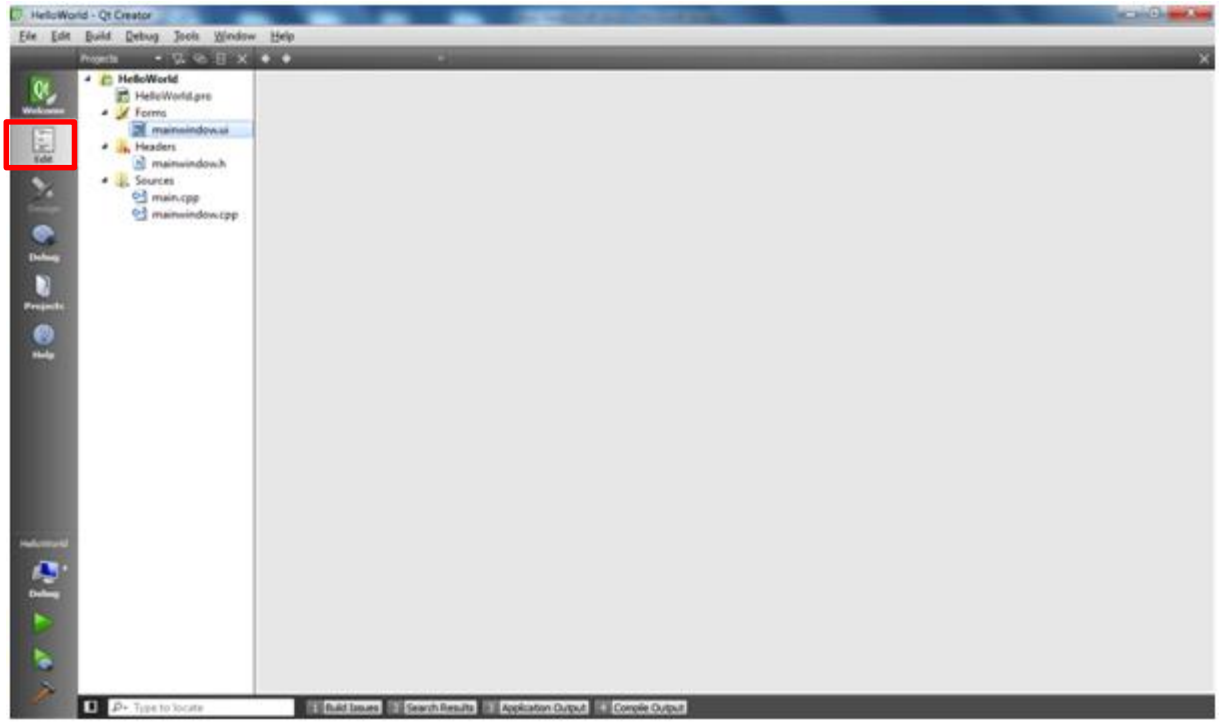
Click Next, chọn bộ quản lý phiên bản (version control). Có thể bỏ qua bước này.

Tiếp tục Click **Next**. Sau đó bấm **Finish** là hoàn tất việc tạo một dự án mới.



— Tổ chức của dự án

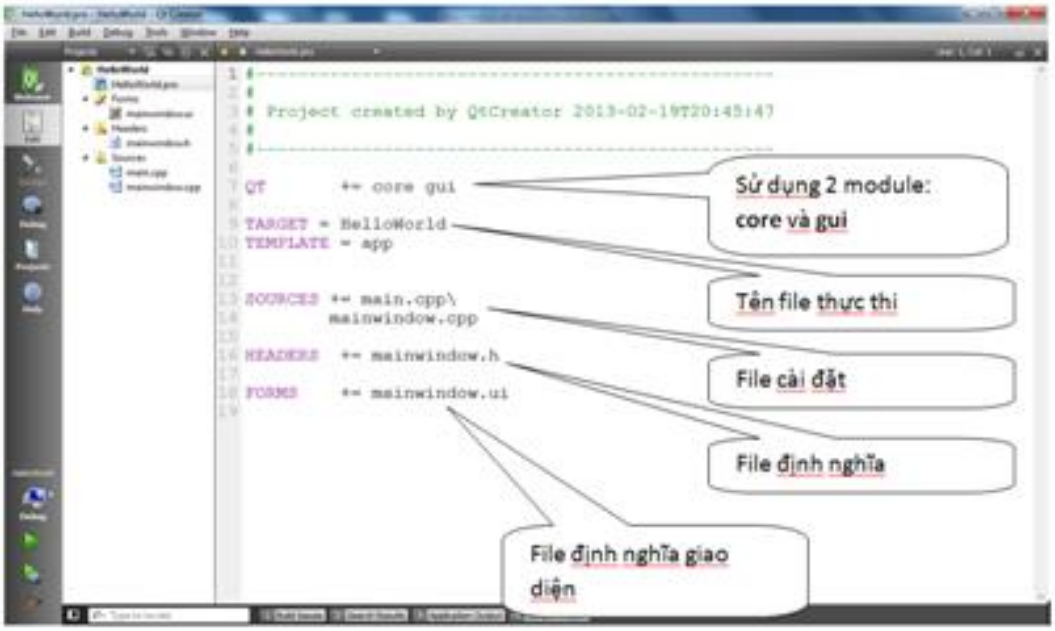
Đóng giao diện, Click Icon **Edit** bên trái để xem tổ chức của dự án



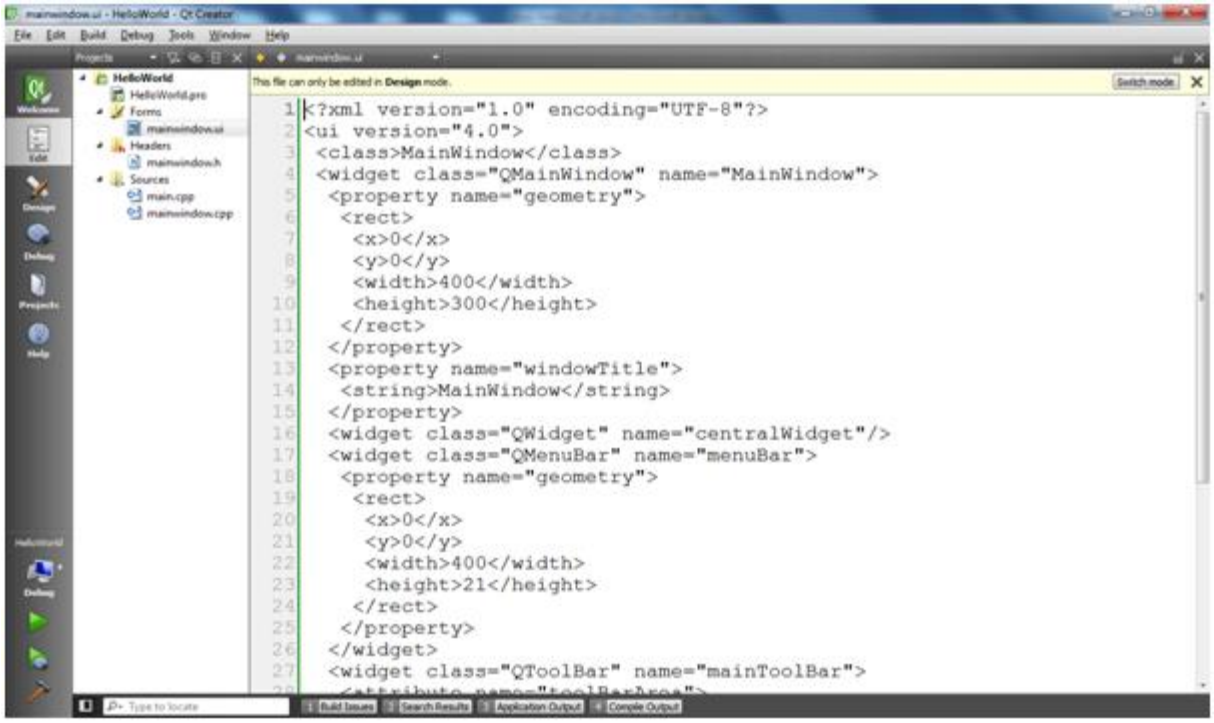
Dự án gồm các file sau:

— **HelloWorld.pro**: file chính của dự án
— **mainwindow.ui:** định nghĩa giao diện giao diện đồ họa (theo cấu trúc XML)
— **mainwindow.h**: định nghĩa lớp MainWindow
— **main.cpp**: chứa hàm main()
— **mainwindow.cpp**: cài đặt các hàm có trong lớp MainWindow

File **HelloWorld.pro**



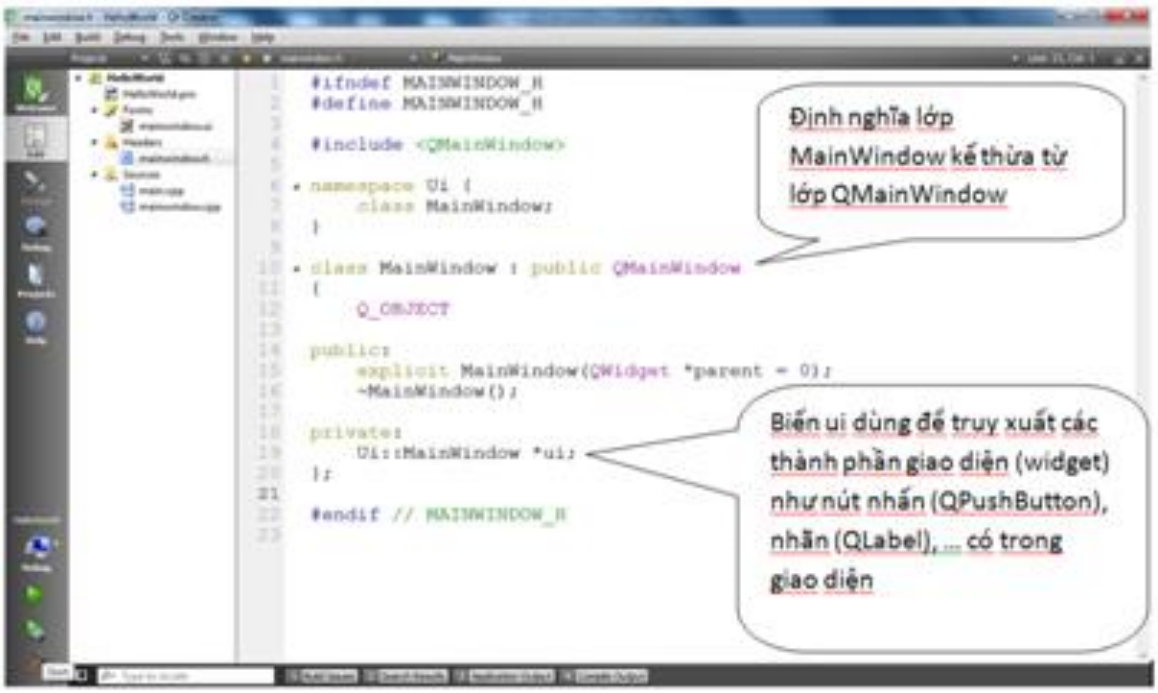
File **mainwindow.ui**



```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>400</width>
    <height>300</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralWidget"/>
  <widget class="QMenuBar" name="menuBar">
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>400</width>
     <height>21</height>
    </rect>
   </property>
  </widget>
  <widget class="QToolBar" name="mainToolBar">
```

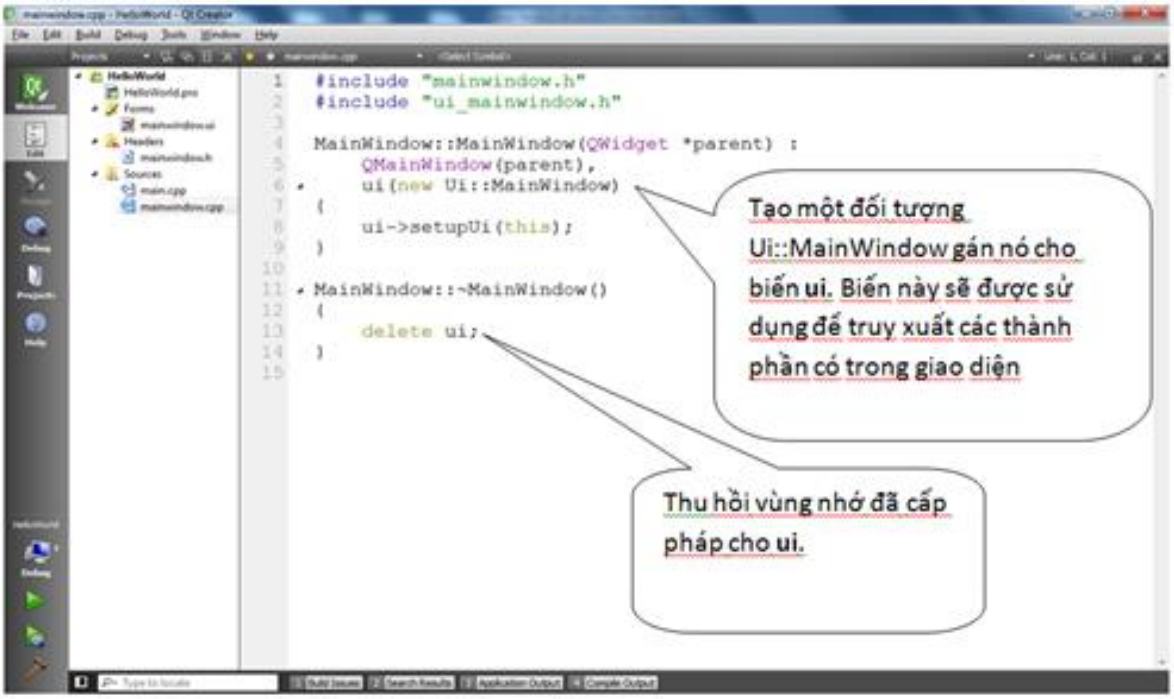
Khi biên dịch Qt sẽ sinh ra lớp Ui::MainWindow được dùng trong lớp MainWindow.

File **mainwindow.h**

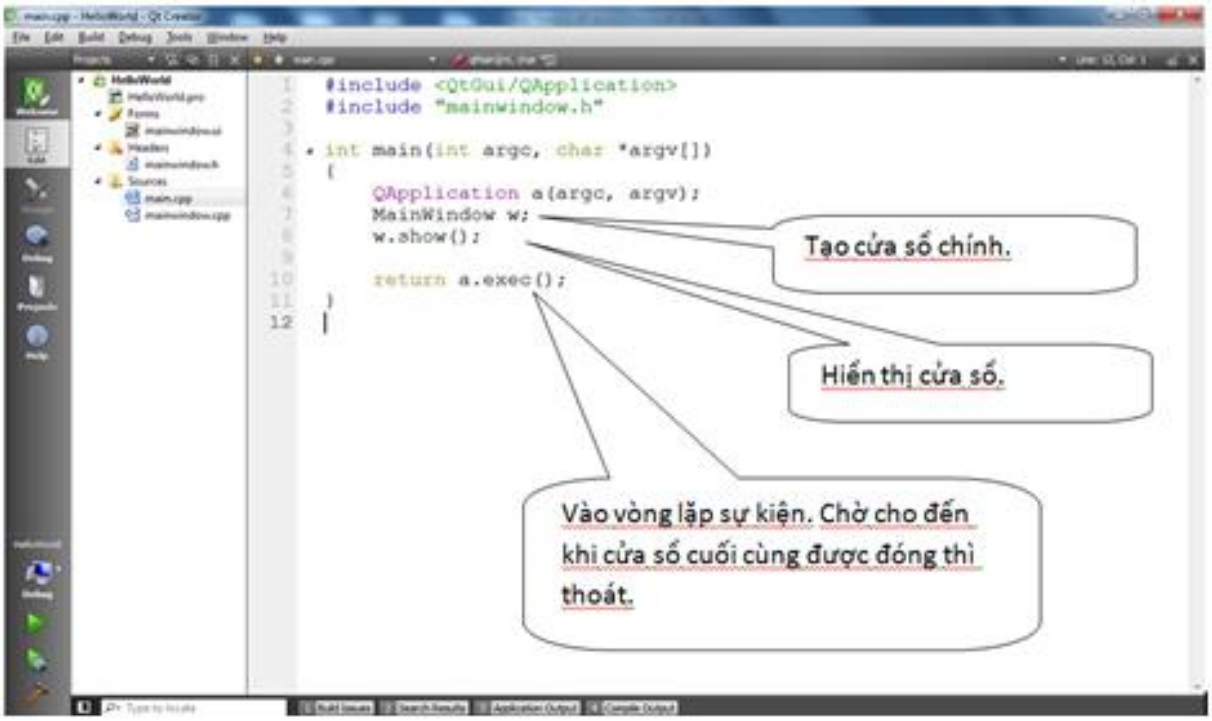



```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

namespace Ui {
    class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

File **mainwindow.cpp**



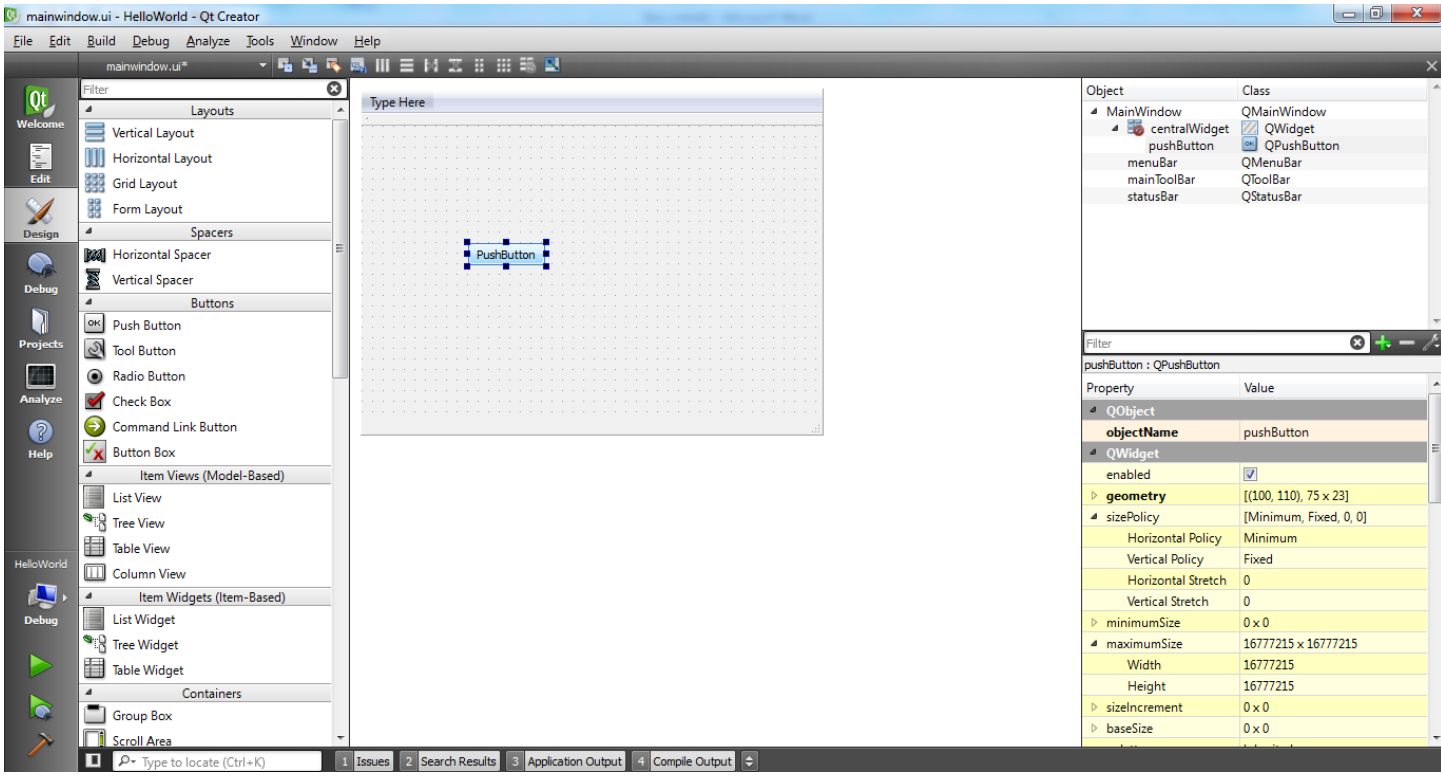
File **main.cpp**



— Thiết kế giao diện và xử lý sự kiện

Trong phần này ta sẽ thiết kế một giao diện gồm một nút nhấn (PushButton) có nhãn: "Click me!", có tên (name): btnClick, khi click chuột vào sẽ hiện hộp thông báo "Hello world!"

Thiết kế giao diện: (Click) mở file **mainwindow.ui** ở chế độ Design, thêm một nút nhấn (Push Button) vào khung cửa sổ trung tâm. Kéo nút nhấn (Push Button) thả (drag and drop) vào trong cửa sổ trung tâm.



Đặt tên và nhãn cho button: Cần phân biệt rõ ràng Tên và nhãn (Object name & text)

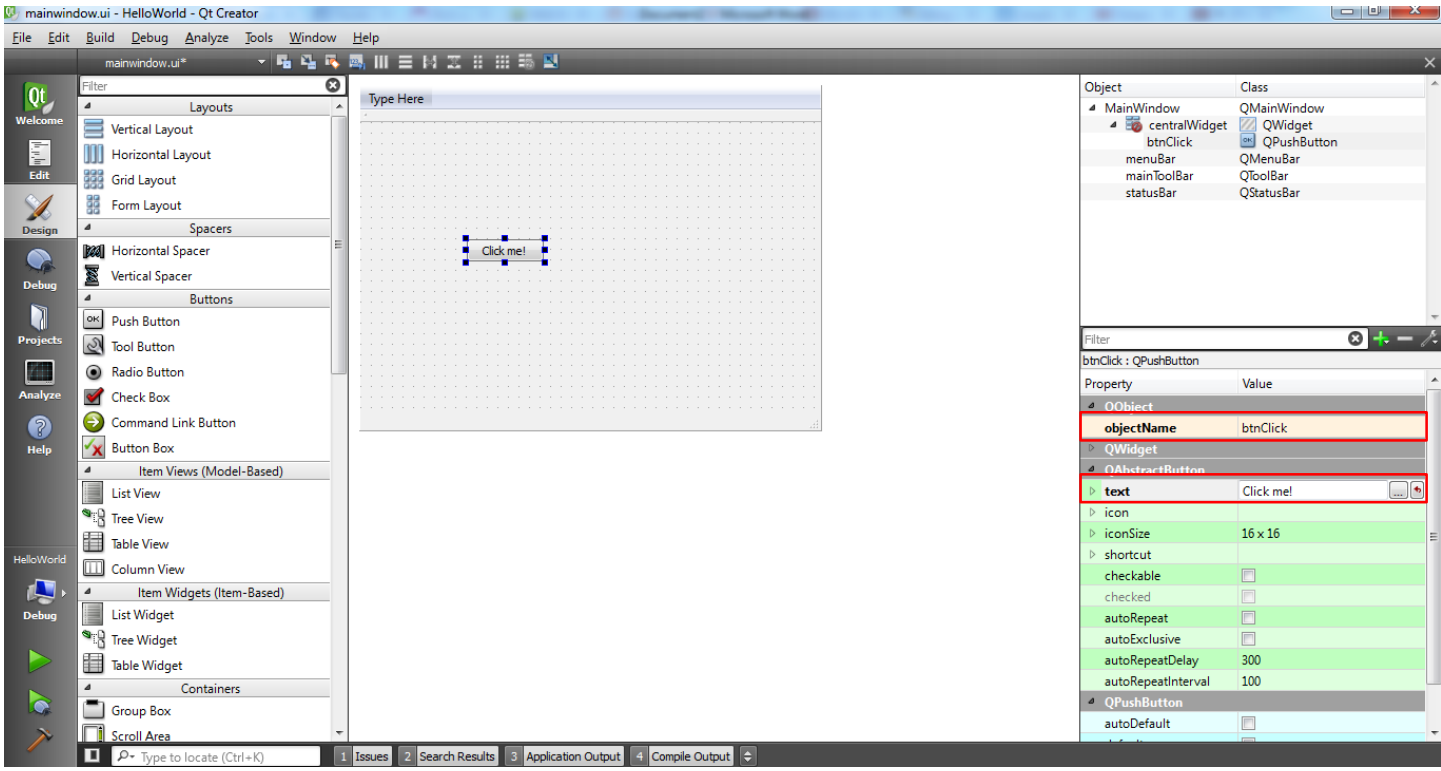
- Tên: Object Name: dùng để phân biệt giữa Object này và object khác, Trong một chương trình thì một Object Name là duy nhất (Các object khác không được đặt trùng tên). Dùng khi ta cần truy xuất đến các thuộc tính của đối tượng.
- Nhãn: là một thuộc tính của đối tượng. chỉ mang tính chất hiển thị. (Dòng chữ hiển thị trên button).

Thay đổi tên của button:

- Ở cửa sổ Properties, Tìm thuộc tính Object Name, Đổi tên từ "pushButton" thành "btnClick"

Thay đổi nhãn của button:

- Ở cửa sổ Properties, Tìm thuộc tính text, Đổi tên từ "pushButton" thành "Click me!"



<u>Xử lý sự kiện</u> :

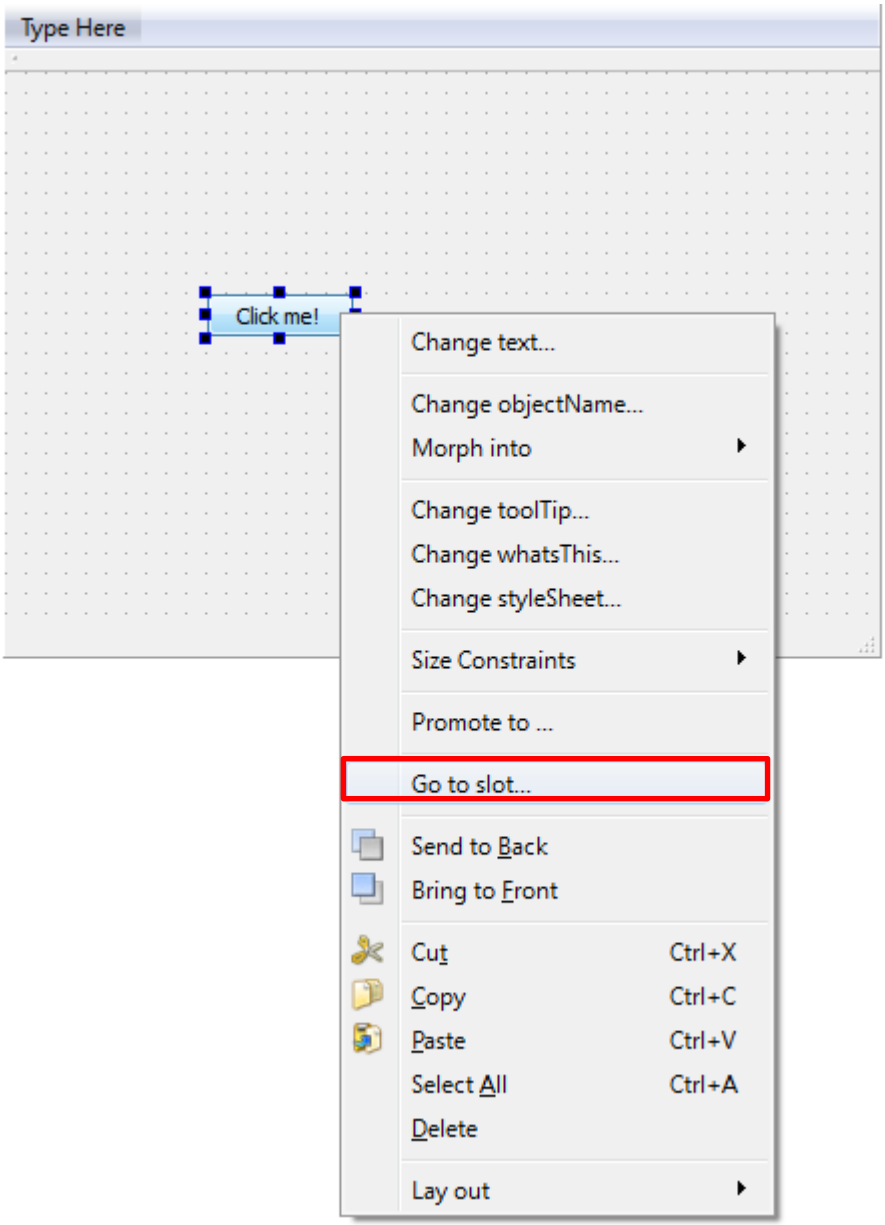
Qt cung cấp cơ chế lập trình hướng sự kiện thông qua: **SIGNAL** và **SLOT**.

- SIGNAL: Sự kiện. Sự kiện có thể là sự kiện của hệ thống như click chuột, di chuyển chuột, bấm phím, … Sự kiện cũng có thể do người lập trình tự định nghĩa và kích hoạt trong chương trình.
- SLOT: Hàm xử lý sự kiện. Về bản chất SLOT cũng là một phương thức (method). Để khai báo một phương thức là một slot ta thêm phát biểu **public slots:** hoặc **private slots:** trước phương thức tương ứng.

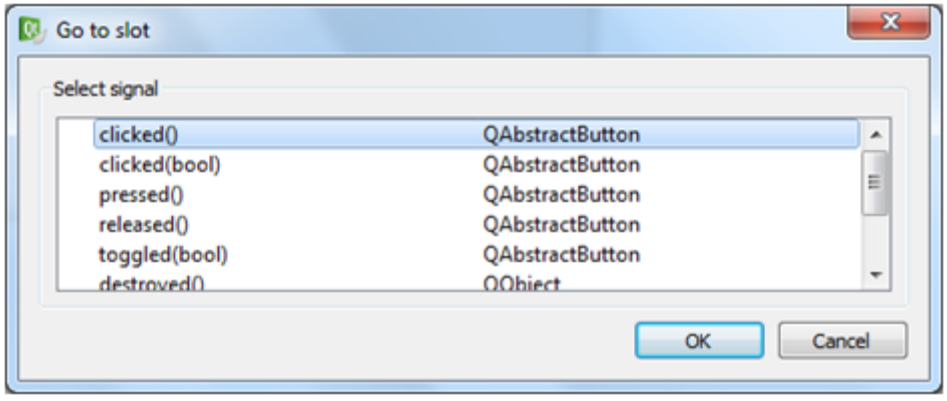
SIGNAL và SLOT được gắn kết với nhau qua hàm **connect**.

Để gắn kết sự kiện do một thành phần có trong giao diện phát ra, click phải trên thành phần đó, chọn **Go to slot…**, trong ví dụ này là click phải trên nút nhấn, chọn **Go to slot…**

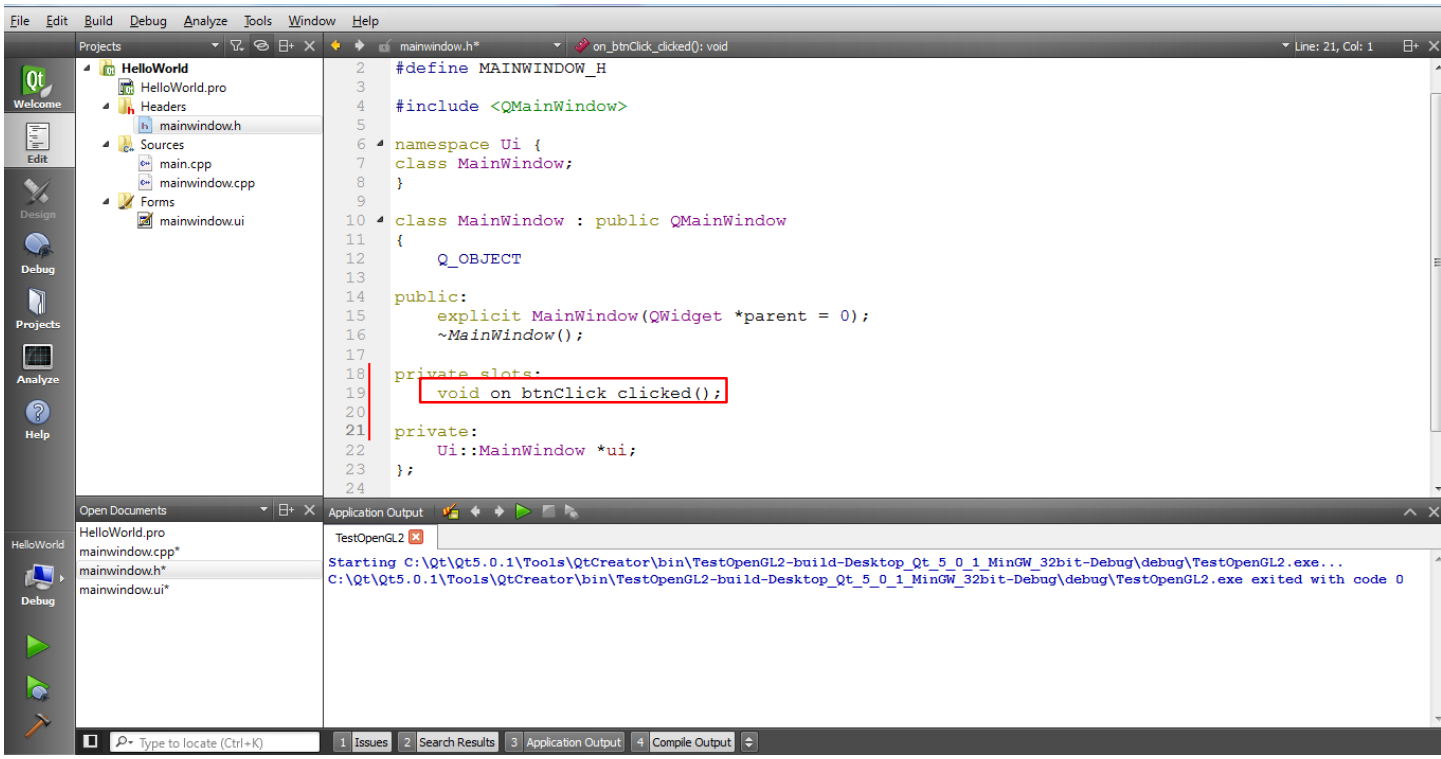
Chú Ý: Cần <u>Đổi tên</u> trước khi <u>xử lý sự kiện</u> để tránh xảy ra lỗi không tìm thấy tên hàm (Đổi tên rồi mới Go To Slot)



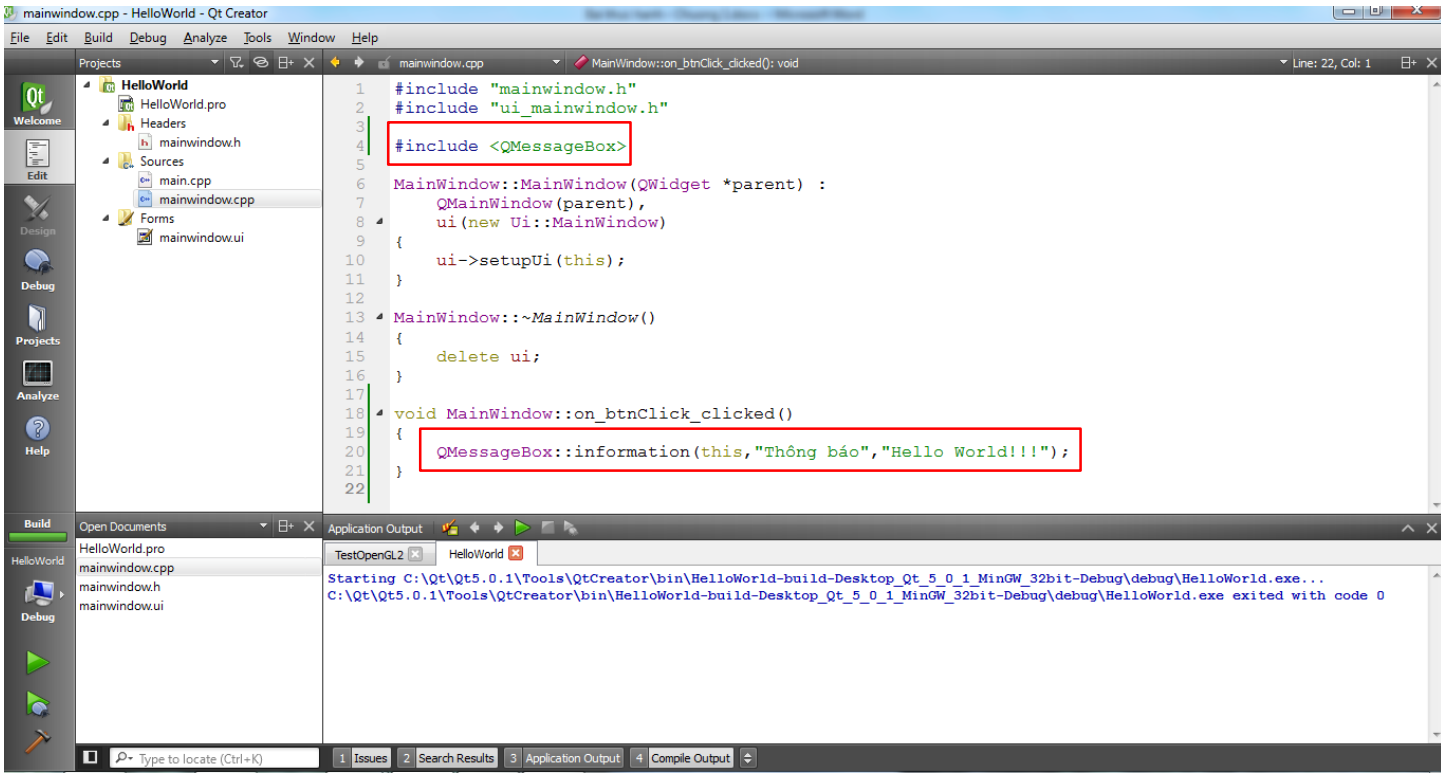
Chọn tín hiệu/sự kiện **clicked()**. Sự kiện này xảy ra người dùng click chuột lên nút nhấn hoặc nhấn ENTER khi nút nhấn đang được kích hoạt.



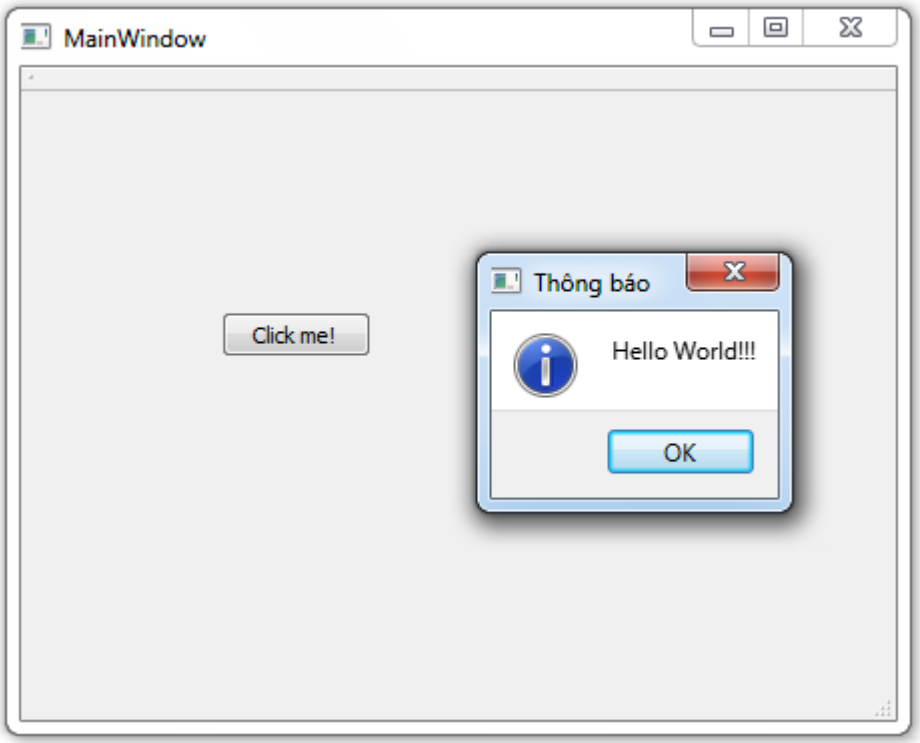
Xem lại file **mainwindow.h** ta sẽ thấy Qt đã tự động thêm vào một slot tên on_btnClick_clicked(). Việc kết nối SIGNAL clicked() và SLOT on_btnClick_clicked() được thực hiện tự động khi ta gọi hàm **ui->setupUi(this)**.



Giờ ta quay lại file **mainwindow.cpp** để định nghĩa code xử lý cho SLOT **on_btnClick_clicked().**
Thêm các dòng code sau vào file **mainwindow.cpp**



Thực thi chương trình ta thu được kết quả như sau:



**** Sinh viên dành thời gian để làm quen với QT và tìm hiểu về các Control khác trước khi chuyển sang phần tiếp theo****

2. Xây dựng một chương trình đồ họa với QT Creator

Để có thể vẽ bằng Qt, chúng ta cần một ***thiết bị vẽ*** (QPaintDevice) và một ***bộ vẽ*** (QPainter). Thiết bị vẽ thông dụng nhất là QWidget. Các lớp thiết bị vẽ khác là: QImage, QPixmap,…Vẽ trực tiếp lên QWidget sẽ cho kết quả lên màn hình, trong khi vẽ lên QImage, QPixmap kết quả sẽ được lưu vào trong bộ nhớ để sau đó chúng ta có thể lưu xuống đĩa hoặc vẽ lại lên màn hình.

Cách đơn giản nhất để tạo một ứng dụng đồ hoạ gồm 3 bước:

1. **Tạo một lớp mới** kế thừa lớp QWidget (lớp graphics kế thừa lớp QWidget) (thiết bị vẽ)

```
class graphics : public QWidget
{

};
```

2. **Định nghĩa lại** (override) phương thức **void paintEvent(QPaintEvent *)** trong lớp mới

```
class graphics : public QWidget
{
    ...
    void paintEvent(QPaintEvent *);  //bộ vẽ
    ...

};
```

3. Trong phần thân của phương thức **paintEvent**, chúng ta tạo một **painter** có kiểu **QPainter**

```
void graphics::paintEvent(QPaintEvent *)
{
    QPainter painter(this); //cọ vẽ
    …
};
```

4. Gọi các phương thức vẽ của lớp QPainter

```
void graphics::paintEvent(QPaintEvent *)
{
    QPainter painter(this);
    painter.drawRect(10,5,100,50);

}
```

"Làm thế nào để hiển thị các nội dung được vẽ trong hàm *paintEvent* ?"

1. Tạo một đối tượng **cửa sổ vẽ MyWidget** và gọi hàm **show()** để hiển thị nó.
2. Nhúng **cửa sổ MyWidget** vào một cửa sổ khác. Ví dụ nhúng **cửa sổ MyWidget vào cửa sổ trung tâm QMainWindow**)

Trong phần tiếp theo chúng ta sẽ thiết kế và hiển thị các hình vẽ theo cách 2.

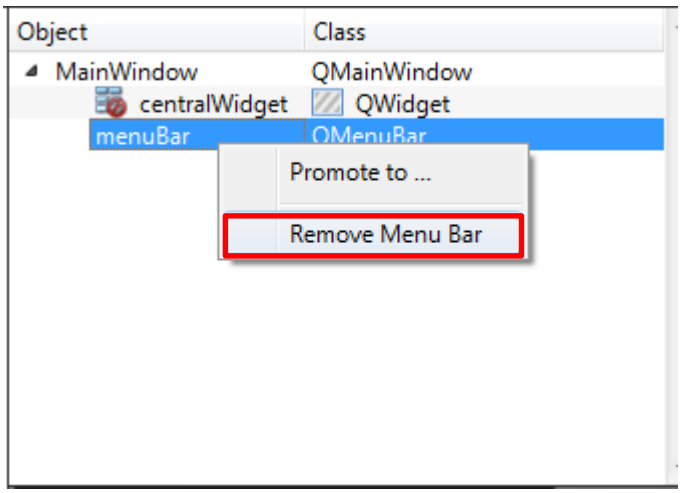
3. Tạo một lớp mới kế thừa lớp QWidget

Xây dựng khung một chương trình đồ họa

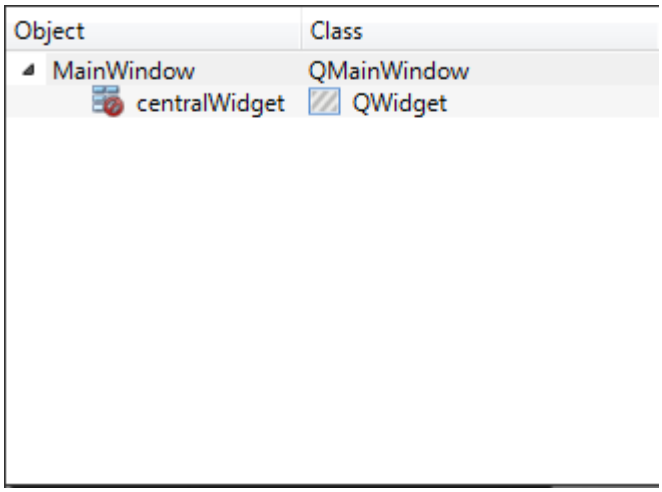
Tạo một dự án mới, đặt tên dự án là **ThucHanhBuoi1.**

Mở file **mainwindow.ui** để chỉnh sửa giao diện.

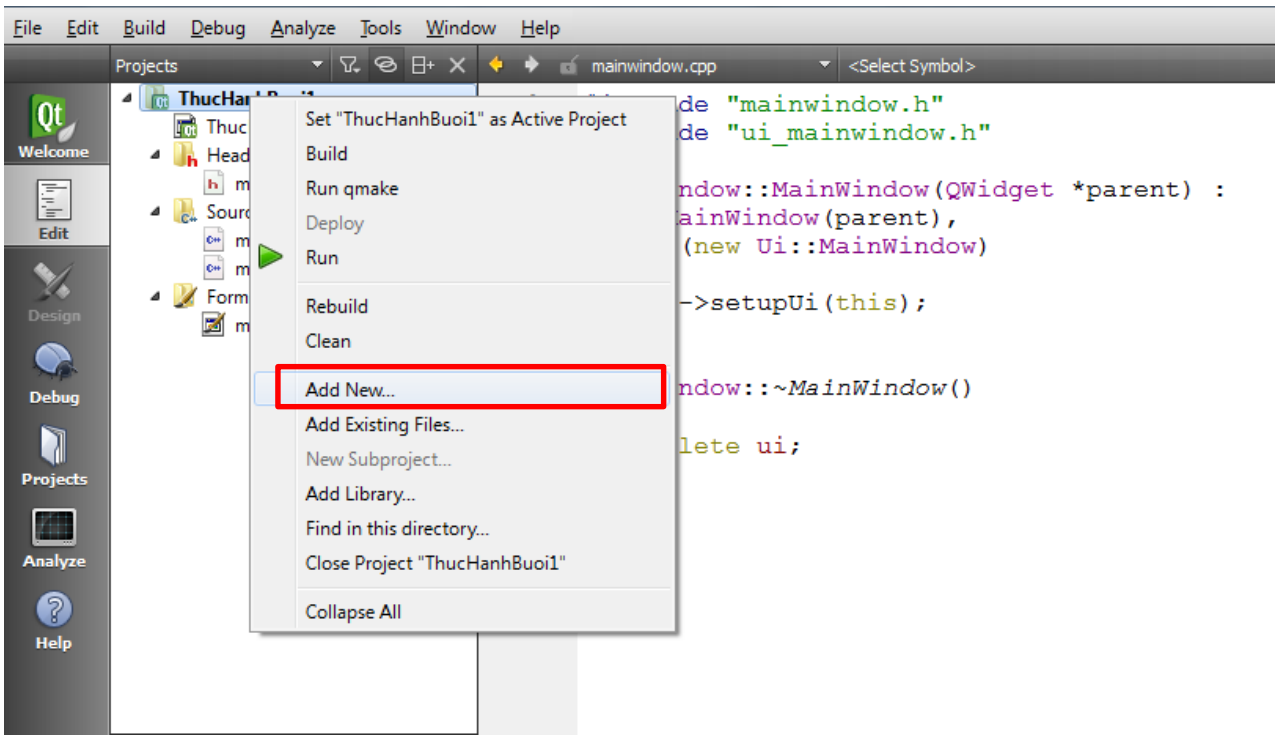
Ở cửa sổ quản lý các object của dự án. Remove các object sau: menuBar, mainToolBar, statusBar bằng cách Click chuột phải→Remove (vì không dùng tới)



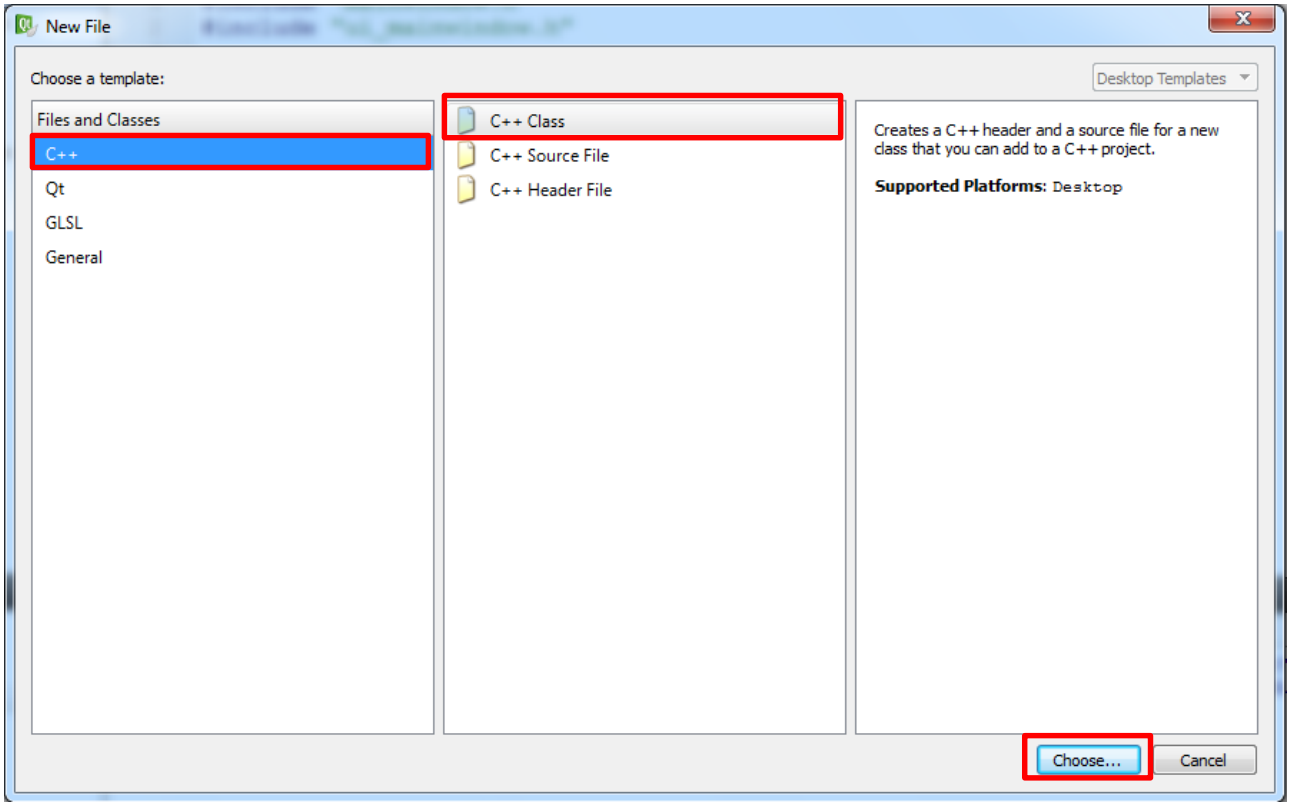
Sau khi remove hết các đối tượng đó, ta còn lại một đối tượng duy nhất là central Widget như trong hình.



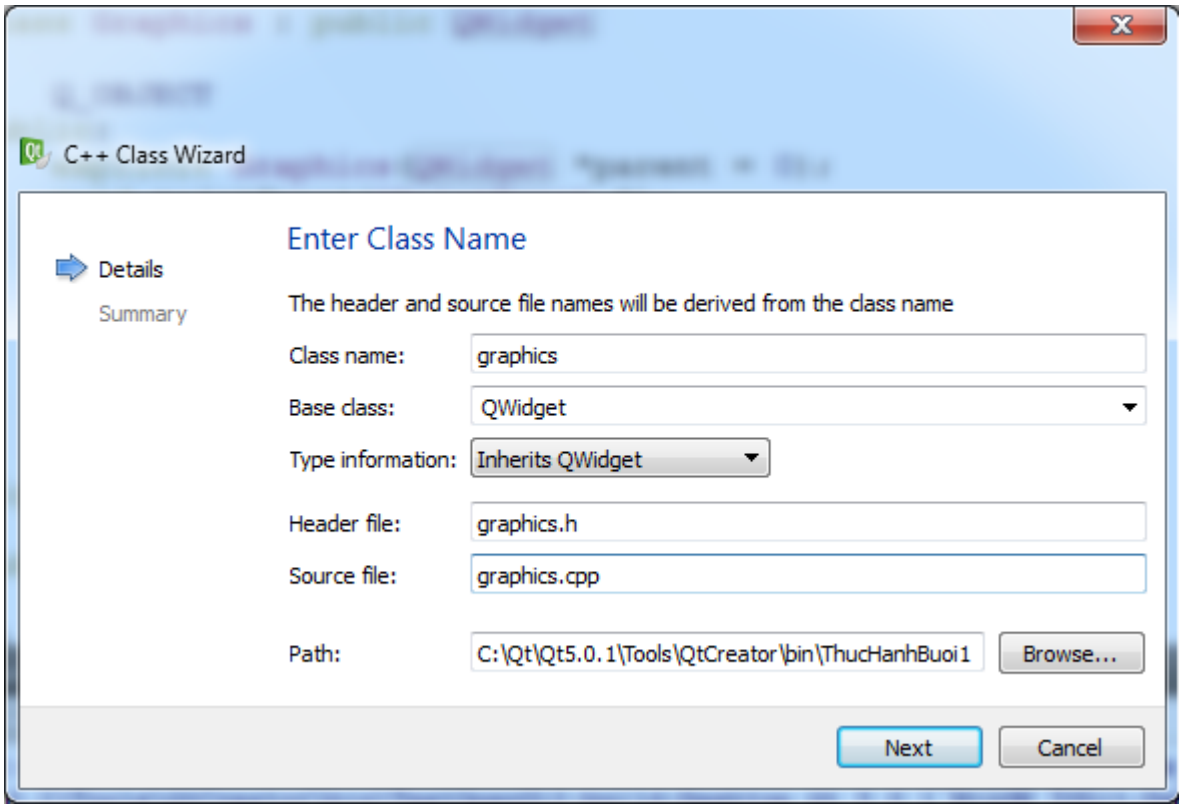
Quay về cửa sổ **Edit**. Click phải vào tên của dự án, chọn **Add New**



Chọn **C++ → C++ Class → Choose**



Đặt tên lớp và chỉnh các tùy chọn như sau:



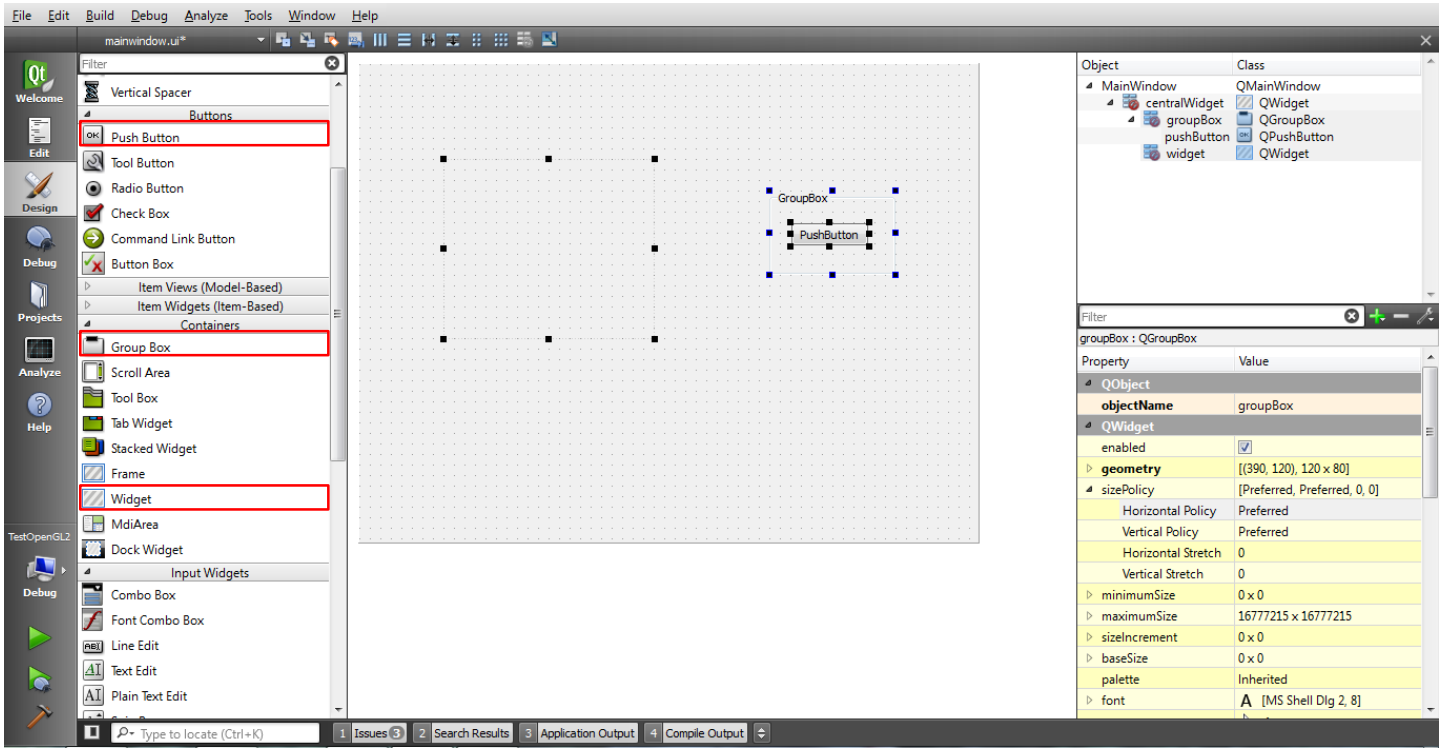
Ở bước sau thì chọn **Finish** là hoàn tất việc tạo một lớp mới.

Class Graphics cùng để thực hiện những phép biểu diễn đồ họa. Lúc này chương trình sẽ tạo nên 2 file mới: **graphics.h** và **graphics.cpp**

Quay trở lại phần thiết kế giao diện

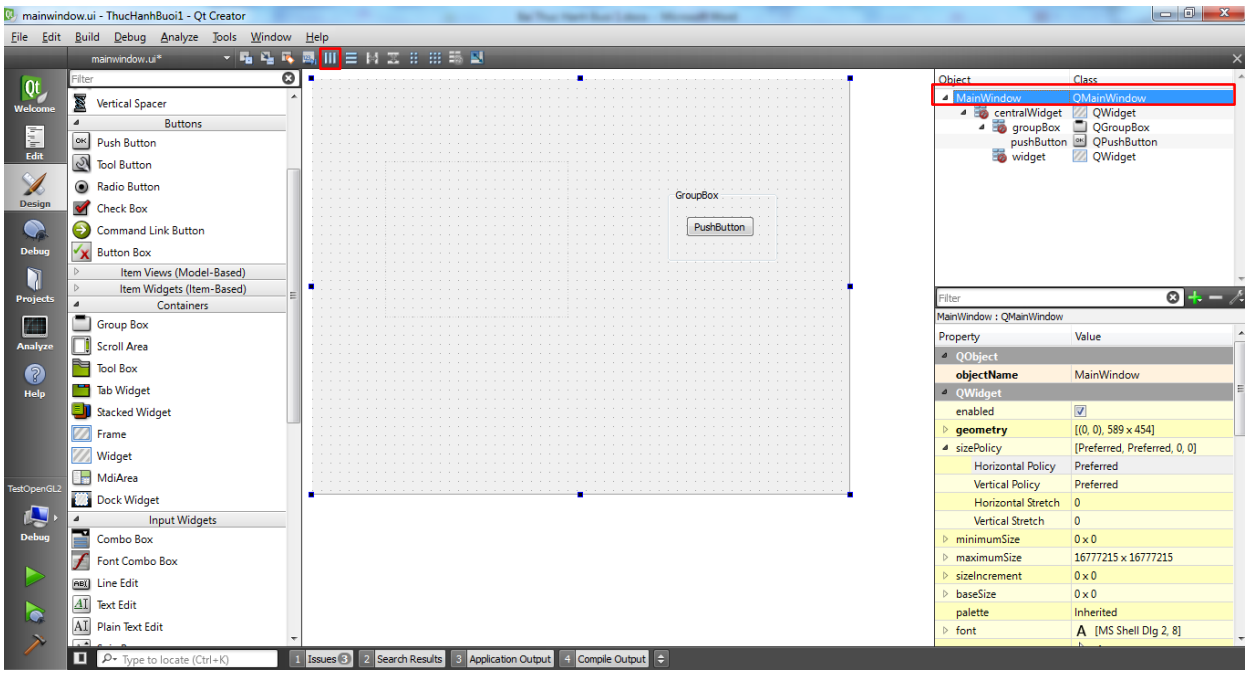
Thêm các Object sau vào chương trình:

- Thêm 1 **Widget**: đây là phần sẽ hiển thị các đối tượng đồ họa mà chúng ta vẽ nên.
- Thêm 1 **GroupBox**: dùng để chứa các Control Object như Push button, check box,…
- Thêm 1 **Push Button**: dùng để phát các lệnh vẽ lên đối tượng widget. Đối tượng Push Button được đặt **nằm chồng lên** đối tượng GroupBox. (**Chú ý: rất quan trọng**).

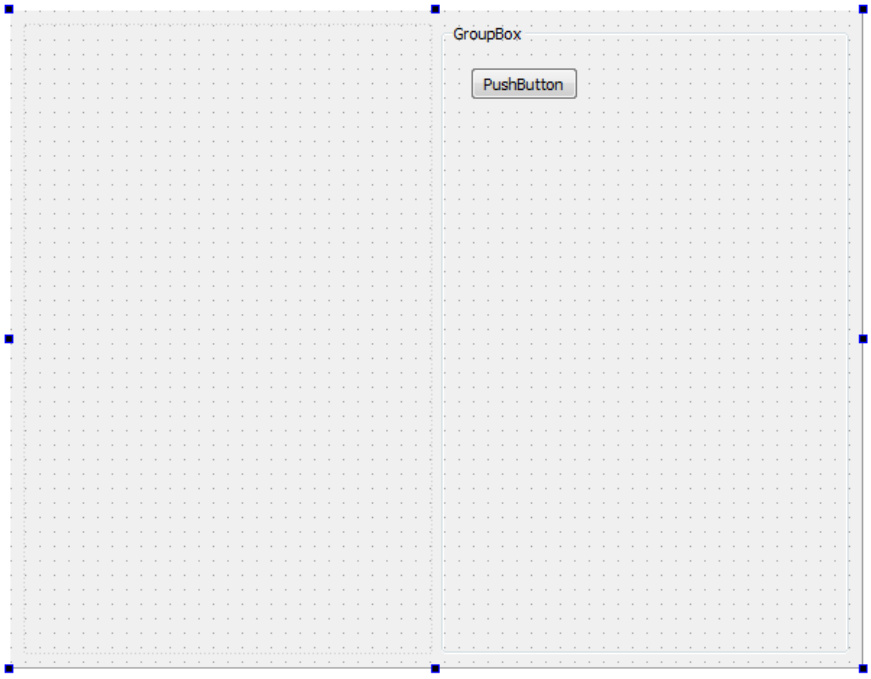


Tiếp tục thực hiện việc thiết kế giao diện :

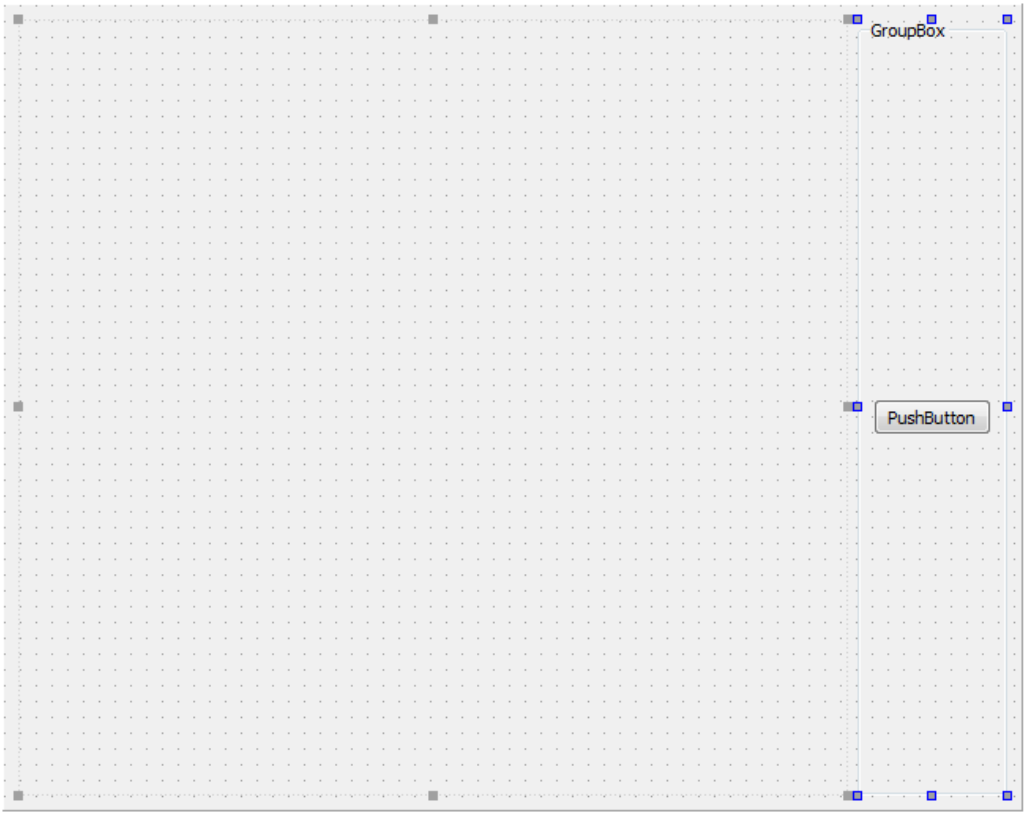
- Trên cửa sổ quản lý các Object, chọn vào **MainWindow**.
- Trên thanh công cụ nằm bên dưới menu, chọn vào icon **Lay out Horizontally** hoặc bấm tổ hợp phím **Ctrl + H**.



Lúc này, giao diện sẽ tự động được chuyển đổi như sau:



- Click chọn vào phần Widget, Bên phần cửa sổ properties, thay đổi thuộc tính **sizePolicy→Horizontal Policy→ Expanding**.
- Click chọn vào GroupBox, trên thanh công cụ nằm bên dưới menu, chọn vào icon **Lay out Vertically** hoặc bấm tổ hợp phím **Ctrl + L**. Lúc này giao diện sẽ có hình dạng như sau:



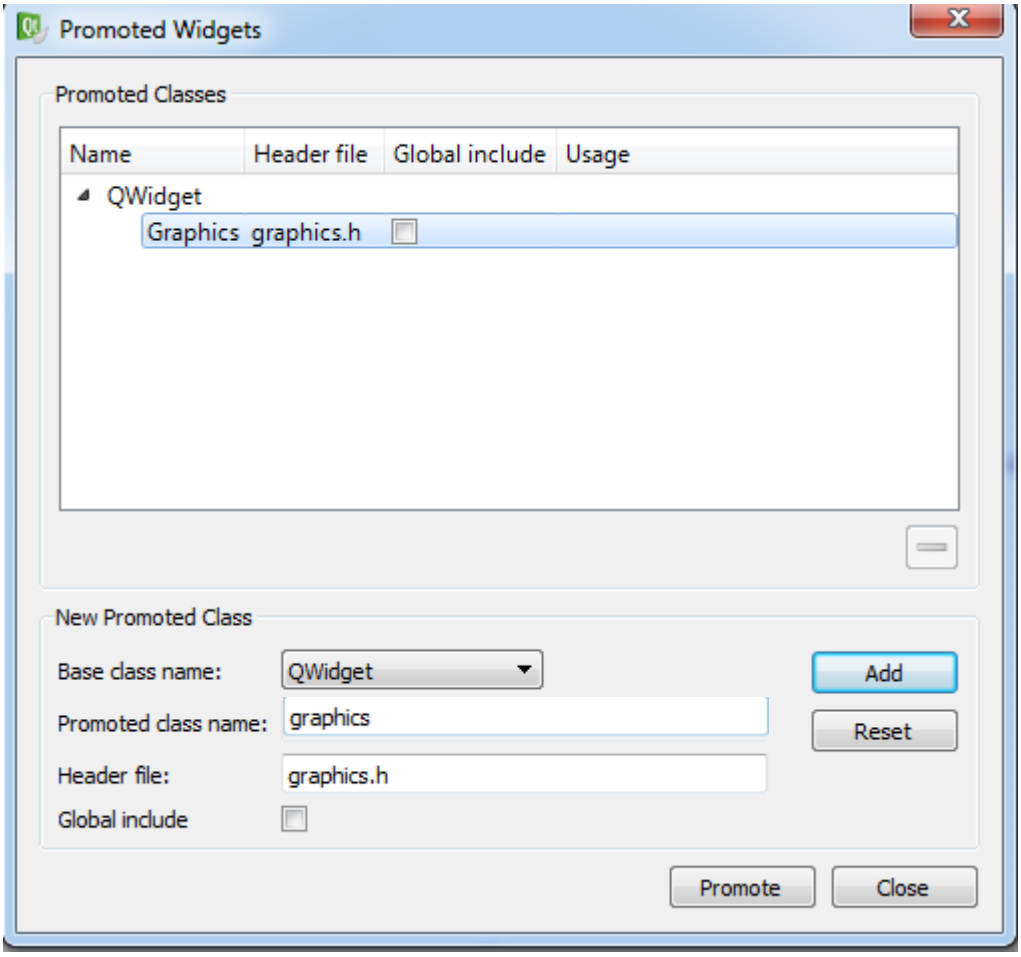
- Thay đổi tên của **GroupBox**: trong cửa sổ **Property→Object Name→GraphicsControl**
- Thay đổi nhãn của **GroupBox**: trong cửa sổ **Property→Title→GraphicsMode.**
- Thay đổi tên của **Widget**: trong cửa sổ **Property→Object Name→GraphicsPresenter**

— Thay đổi tên của **PushButton**: trong cửa sổ **Property→Object Name→ BtnDrawSimple.**
— Thay đổi nhãn của **PushButton**: trong cửa sổ **Property→Text→DrawSimple.**

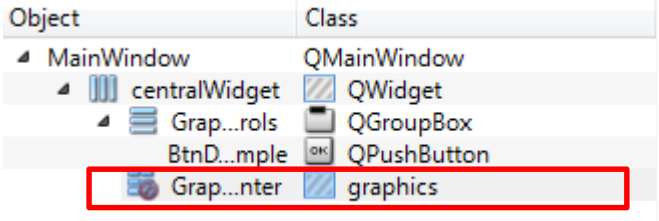
Đến đây, về cơ bản, ta đã hoàn thành phần thiết kế giao diện cho một chương trình đồ họa đơn giản. Ở bước tiếp theo, ta sẽ thực hiện quá trình gán quyền điều khiển đối tượng widget (lúc này đối tượng widget này mang tên là GraphicsPresenter).

Các bước của phép gán quyền:

— Click phải vào đối tượng **widget** (GraphicsPresenter). Chọn **Promote to**.
— Ở khung Promoted Class name: nhập vào **graphics** (Gán quyền điều khiển đối tượng này cho lớp graphics mà ta vừa mới tạo)



— Click **Add → Promote**
— Lúc này, xem lại cửa sổ quản lý Object, ta sẽ thấy Đối tượng **Widget(GraphicsPresenter)** đã có **Class** là **graphics** (Thay vì QWidget như cũ).



3.1. Edit code cho chương trình

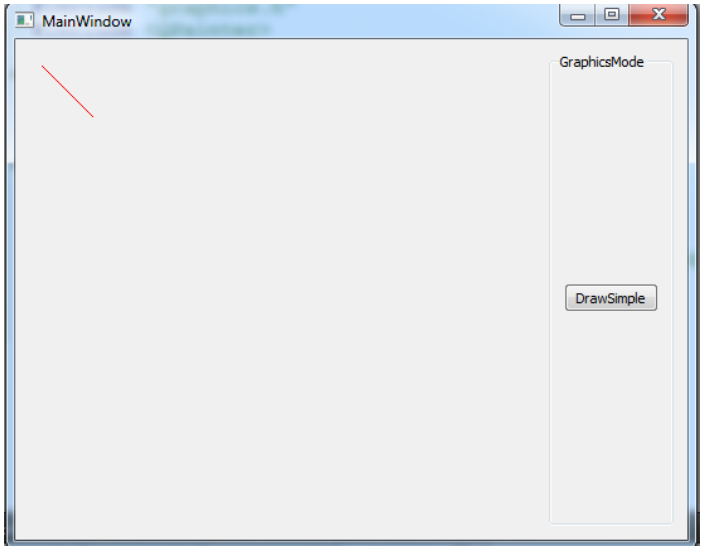
Ở file **graphics.h**, thêm vào dòng code sau:

```
#ifndef GRAPHICS_H
#define GRAPHICS_H

#include <QWidget>

class graphics : public QWidget
{
    Q_OBJECT
public:
    explicit graphics(QWidget *parent = 0);
    void paintEvent(QPaintEvent *);

signals:

public slots:

};

#endif // GRAPHICS_H
```

Ở file **graphics.cpp**, ta tiến hành edit phần thân hàm vừa mới khai báo.

```
#include "graphics.h"
#include <QPainter>
graphics::graphics(QWidget *parent) :
    QWidget(parent)
{
}

void graphics::paintEvent(QPaintEvent *) {

    QPainter painter(this);
    painter.setPen(Qt::red);//Set màu cho cọ vẽ
    painter.drawLine(10,10,50,50);// Vẽ 1 đường thẳng từ điểm (10,10) đến điểm (50,50)

}
```

Run thử chương trình, xem kết quả.



Đến thời điểm này, ta đã cơ bản chuẩn bị xong một chương trình đồ họa đơn giản.

Ở bước tiếp theo, ta sẽ gán lệnh vẽ vào sự kiện click của Button.

- Ở file graphics.h, thêm vào một biến mode để xác định các chế độ vẽ
- Khai báo một hàm để thực hiện phép vẽ

```
2   #define GRAPHICS_H
3
4   #include <QWidget>
5
6   class graphics : public QWidget
7   {
8       Q_OBJECT
9   public:
10      explicit graphics(QWidget *parent = 0);
11      void paintEvent(QPaintEvent *);
12      void DrawLines(QPainter& painter);
13
14
15      int mode = 0;
16
17  signals:
18
19  public slots:
20
21  };
22
23  #endif // GRAPHICS_H
24
```

- Ở file graphics.cpp, chuyển phần vẽ trong hàm paintevent về hàm DrawLines
- Sau đó trong hàm Paintevent, gọi hàm DrawLines để thực hiện phép vẽ.

```
1   #include "graphics.h"
2   #include <QPainter>
3   graphics::graphics(QWidget *parent) :
4       QWidget(parent)
5   {
6   }
7
8   void graphics::paintEvent(QPaintEvent *) {
9
10      QPainter painter(this);
11      if(mode == 1)DrawLines(painter);//gọi hàm DrawLines nếu mode vẽ = 1;
12
13  }
14
15  void graphics::DrawLines(QPainter& painter)
16  {
17      painter.setPen(Qt::red);//Set màu cho cọ vẽ
18      painter.drawLine(10,10,50,50);// Vẽ 1 đường thẳng từ điểm (10,10) đến điểm (50,50)
19  }
20
```

— Lúc này, ta sử dụng sự kiện Click của button Drawsimple, Click phải vào button→go to slot, thêm vào các dòng code sau:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

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

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

void MainWindow::on_BtnDrawSimple_clicked()
{
    ui->GraphicsPresenter->mode = 1;//set mode vẽ thành 1
    ui->GraphicsPresenter->repaint();//hàm repaint sẽ gọi đến paintevent
    ui->GraphicsPresenter->mode = 0;//set mode vẽ về lại 0
}
```

— Thực thi chương trình và xem kết quả. Chương trình chỉ thực hiện phép vẽ khi ta click vào button DrawSimple.

Đến giai đoạn này, ta đã cơ bản hoàn thành khung một chương trình đồ họa đơn giản cho phép ta thực hiện những phép vẽ cơ bản. Để tiếp tục những chức năng vẽ khác:

— Sinh viên khai báo một hàm vẽ mới trong file graphics.h và graphics.cpp
— Thêm 1 button mới vào groupbox
— Gán quyền vẽ cho groupbox với một mode khác

QT còn hỗ trợ nhiều hàm vẽ khác. Sinh viên tự tìm hiểu và ứng dụng để vẽ những hình từ cơ bản tới nâng cao

Các hàm vẽ cơ bản:
drawPoint () : hàm vẽ điểm
drawPoints() : hàm vẽ nhiều điểm
drawLine() : hàm vẽ đường thẳng
drawLines() : hàm vẽ nhiều đường thẳng
drawRect() : hàm vẽ hình chữ nhật
drawEllipse() : hàm vẽ ellipse
…..

Các hàm còn lại sinh viên tự tìm hiểu

4. Ví dụ & Bài tập

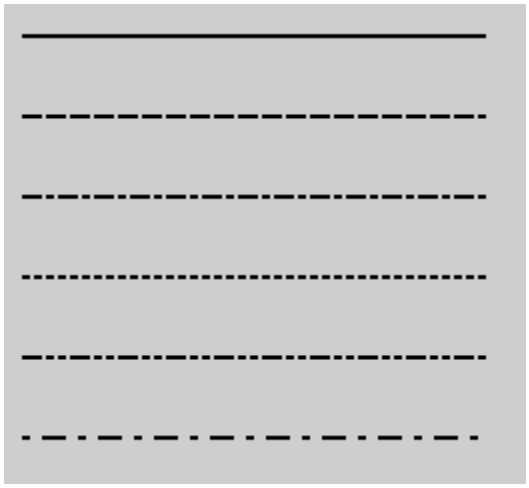
4.1. Ví dụ:

Các dạng đường thẳng: Tạo một hàm vẽ mới đặt tên hàm DrawMultiLines, hàm này sẽ vẽ các dạng đường thẳng khác nhau mà QT có hỗ trợ.

Trong nội hàm DrawMultiLines, Thêm các dòng code sau:

```
void graphics::DrawMultiLines(QPainter& painter)
{

    QPen pen(Qt::black, 2, Qt::SolidLine);
    painter.setPen(pen);
    painter.drawLine(20, 40, 250, 40);

    pen.setStyle(Qt::DashLine);
    painter.setPen(pen);
    painter.drawLine(20, 80, 250, 80);

    pen.setStyle(Qt::DashDotLine);
    painter.setPen(pen);
    painter.drawLine(20, 120, 250, 120);

    pen.setStyle(Qt::DotLine);
    painter.setPen(pen);
    painter.drawLine(20, 160, 250, 160);

    pen.setStyle(Qt::DashDotDotLine);
    painter.setPen(pen);
    painter.drawLine(20, 200, 250, 200);

}
```

Thực thi chương trình và xem kết quả



Tô màu : Tạo một hàm mới tên là DrawColorExample, hàm này sẽ vẽ các hình chữ nhật và tô màu với các màu sắc khác nhau

Trong nội hàm DrawColorExample, thêm các dòng code sau:

```
void graphics::DrawColorExample(QPainter& painter)
{


    painter.setPen(QColor("#d4d4d4"));

      painter.setBrush(QBrush("#c56c00"));
      painter.drawRect(10, 15, 90, 60);

      painter.setBrush(QBrush("#1ac500"));
      painter.drawRect(130, 15, 90, 60);

      painter.setBrush(QBrush("#539e47"));
      painter.drawRect(250, 15, 90, 60);

      painter.setBrush(QBrush("#004fc5"));
      painter.drawRect(10, 105, 90, 60);

      painter.setBrush(QBrush("#c50024"));
      painter.drawRect(130, 105, 90, 60);

      painter.setBrush(QBrush("#9e4757"));
      painter.drawRect(250, 105, 90, 60);

      painter.setBrush(QBrush("#5f3b00"));
      painter.drawRect(10, 195, 90, 60);

      painter.setBrush(QBrush("#4c4c4c"));
      painter.drawRect(130, 195, 90, 60);

      painter.setBrush(QBrush("#785f36"));
      painter.drawRect(250, 195, 90, 60);
}
```

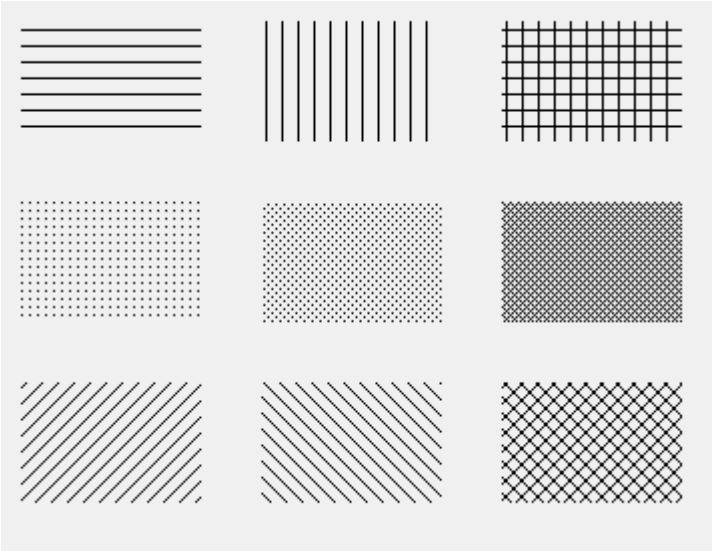
Kết quả:

Vẽ các mẫu: Tạo một hàm mới tên là DrawPatternExample, hàm này sẽ vẽ các hình chữ nhật bằng những pattern khác nhau.

Trong nội hàm DrawPatternExample, Thêm các dòng code sau:

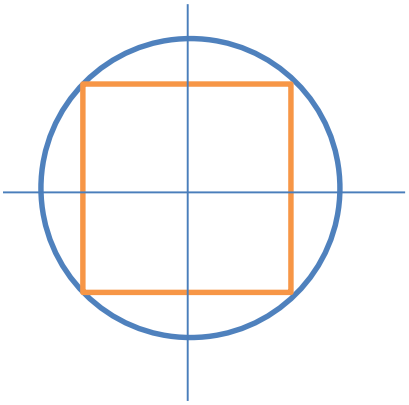
```
void graphics::DrawPatternExample(QPainter& painter)
{
	painter.setPen(Qt::NoPen);
	painter.setBrush(Qt::HorPattern);
	painter.drawRect(10, 15, 90, 60);

	painter.setBrush(Qt::VerPattern);
	painter.drawRect(130, 15, 90, 60);

	painter.setBrush(Qt::CrossPattern);
	painter.drawRect(250, 15, 90, 60);

	painter.setBrush(Qt::Dense7Pattern);
	painter.drawRect(10, 105, 90, 60);

	painter.setBrush(Qt::Dense6Pattern);
	painter.drawRect(130, 105, 90, 60);

	painter.setBrush(Qt::Dense5Pattern);
	painter.drawRect(250, 105, 90, 60);

	painter.setBrush(Qt::BDiagPattern);
	painter.drawRect(10, 195, 90, 60);

	painter.setBrush(Qt::FDiagPattern);
	painter.drawRect(130, 195, 90, 60);

	painter.setBrush(Qt::DiagCrossPattern);
	painter.drawRect(250, 195, 90, 60);
}
```

Run và xem kết quả

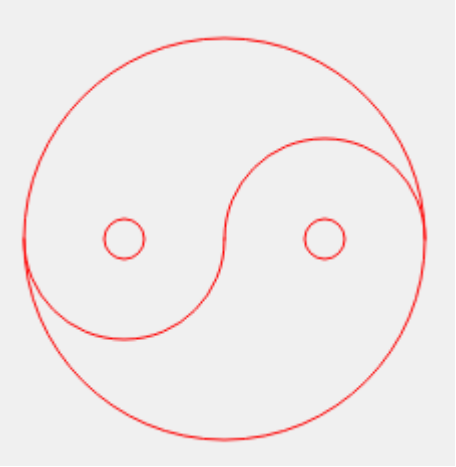


4.2. Bài tập:

1) Viết thủ tục để vẽ một hình vuông nội tiếp một hình tròn. Tâm của hình tròn nằm ở giữa GraphicsPresenter. Mỗi hình vẽ bằng một màu khác nhau (gợi ý: có thể vẽ hình tròn trước, hình vuông sau hoặc ngược lại)

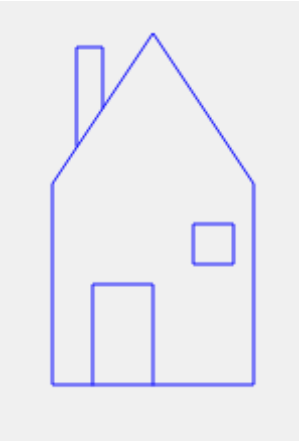


2) Viết thủ tục để vẽ hình thái cực.



Gợi ý: chia hình thái cực ra thành 1 hình tròn lớn, 2 hình tròn nhỏ và 2 nữa cung tròn.
Dùng hàm drawEllipse() để vẽ các hình tròn, dùng hàm drawArc() để vẽ các cung tròn. (SV tự tìm hiểu cách sử dụng hàm drawEllipse() và drawArc())

3) Viết thủ tục để vẽ ngôi nhà



Gợi ý: Dùng 2 hàm drawPolyline() và drawPolygon() để truyền vào tọa độ các đỉnh vẽ.

Hàm drawPolyline() dùng để vẽ các đường gấp khúc hở. (Vẽ cửa chính và ống khói)

Hàm drawPolygon() dùng để vẽ các đường gấp khúc kín (Vẽ viền của ngôi nhà)

Dùng hàm drawRect() để vẽ cửa sổ.

4) Viết thủ tục để vẽ bàn cờ vua.
Gợi ý: bàn cờ vua gồm có 8x8 hình vuông với màu sắc xen kẽ, cần xác định vị trí và kích thước của các khối vuông để vẽ.