



# *Introducción a Qt*

Rafael Rivas Estrada  
rafael@ula.ve

Mayo - 2010

Departamento de Computación  
Universidad de Los Andes

# Índice

## 1 Introducción

## 2 Conceptos básicos

- Compilación
- Gestores de organización de componentes
- Gestión de eventos
- Algunos componentes 1era Parte
- Algunos componentes 2da Parte
- Qt Designer

## 3 Bibliografía

¿Qué es Qt?

¿Qué es Qt?

” es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario y también para el desarrollo de programas sin interfaz gráfica como herramientas de la consola y servidores. ”

Wikipedia

¿Qué es Qt?

” es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario y también para el desarrollo de programas sin interfaz gráfica como herramientas de la consola y servidores. ”

Wikipedia

En Internet: <http://qt.nokia.com/>

## Características:

- Biblioteca de clases para C++
- Portabilidad entre sistemas de escritorio y sistemas empujados.
- Sistema de desarrollo integrado

## Características:

- Biblioteca de clases para C++
- Portabilidad entre sistemas de escritorio y sistemas empuotrados.
- Sistema de desarrollo integrado

## Características:

- Biblioteca de clases para C++
- Portabilidad entre sistemas de escritorio y sistemas empujados.
- Sistema de desarrollo integrado



## Tipos de licencia:

- Qt Commercial Developer License
- Qt GNU LGPL v. 2.1
- Qt GNU GPL v. 3.0

## Tipos de licencia:

- Qt Commercial Developer License
- Qt GNU LGPL v. 2.1
- Qt GNU GPL v. 3.0

## Tipos de licencia:

- Qt Commercial Developer License
- Qt GNU LGPL v. 2.1
- Qt GNU GPL v. 3.0

## Tipos de licencia:

- Qt Commercial Developer License
- Qt GNU LGPL v. 2.1
- Qt GNU GPL v. 3.0

Tipos de licencia:

- Qt Commercial Developer License
- Qt GNU LGPL v. 2.1
- Qt GNU GPL v. 3.0

Más información: <http://qt.nokia.com/products/licensing>

## Ejemplo 0: Hola Mundo en QT

```
#include <QApplication>
#include <QLabel>

int main(int argc , char *argv []){
    QApplication programa(argc , argv);
    QLabel *etiqueta = new QLabel(" Hola _Qt!" );
    etiqueta->show();
    return programa.exec();
}
```

## Ejemplo 0: Hola Mundo en QT

```
#include <QApplication>
#include <QLabel>

int main(int argc, char *argv[]){
    QApplication programa(argc, argv);
    QLabel *etiqueta = new QLabel("Hola Qt!");
    etiqueta->show();
    return programa.exec();
}
```



# Creación del ejecutable en Linux

## Pasos:

- Creación del archivo de proyecto independiente de plataforma

- `qmake -project -o holaQT.pro`

Se obtiene el archivo `holaQT.pro`

- Creación de un Makefile dependiente de la plataforma usada

- `qmake holaQT.pro`

Se obtiene el Makefile, con las instrucciones necesarias para usar las bibliotecas instaladas

- `.-` Usar el Makefile para obtener el programa ejecutable

- `make`

Si la sintaxis y la versión de la biblioteca instalada es correcta se obtiene el ejecutable.



# Creación del ejecutable en Linux

## Pasos:

- Creación del archivo de proyecto independiente de plataforma
  - `qmake -project -o holaQT.pro`

Se obtiene el archivo `holaQT.pro`

- Creación de un Makefile dependiente de la plataforma usada
  - `qmake holaQT.pro`

Se obtiene el Makefile, con las instrucciones necesarias para usar las bibliotecas instaladas

- `.-` Usar el Makefile para obtener el programa ejecutable
  - `make`

Si la sintaxis y la versión de la biblioteca instalada es correcta se obtiene el ejecutable.

# Creación del ejecutable en Linux

## Pasos:

- Creación del archivo de proyecto independiente de plataforma
  - `qmake -project -o holaQT.pro`

Se obtiene el archivo `holaQT.pro`

- Creación de un Makefile dependiente de la plataforma usada
  - `qmake holaQT.pro`

Se obtiene el Makefile, con las instrucciones necesarias para usar las bibliotecas instaladas

- `.-` Usar el Makefile para obtener el programa ejecutable
  - `make`

Si la sintaxis y la versión de la biblioteca instalada es correcta se obtiene el ejecutable.

# Creación del ejecutable en Linux

## Pasos:

- Creación del archivo de proyecto independiente de plataforma
  - `qmake -project -o holaQT.pro`

Se obtiene el archivo `holaQT.pro`

- Creación de un Makefile dependiente de la plataforma usada
  - `qmake holaQT.pro`

Se obtiene el Makefile, con las instrucciones necesarias para usar las bibliotecas instaladas

- `.-` Usar el Makefile para obtener el programa ejecutable
  - `make`

Si la sintaxis y la versión de la biblioteca instalada es correcta se obtiene el ejecutable.

# Creación del ejecutable en Linux

## Pasos:

- Creación del archivo de proyecto independiente de plataforma
  - `qmake -project -o holaQT.pro`

Se obtiene el archivo `holaQT.pro`

- Creación de un Makefile dependiente de la plataforma usada
  - `qmake holaQT.pro`

Se obtiene el Makefile, con las instrucciones necesarias para usar las bibliotecas instaladas

- `.-` Usar el Makefile para obtener el programa ejecutable
  - `make`

Si la sintaxis y la versión de la biblioteca instalada es correcta se obtiene el ejecutable.

# Creación del ejecutable en Linux

## Pasos:

- Creación del archivo de proyecto independiente de plataforma
  - `qmake -project -o holaQT.pro`

Se obtiene el archivo `holaQT.pro`

- Creación de un Makefile dependiente de la plataforma usada
  - `qmake holaQT.pro`

Se obtiene el Makefile, con las instrucciones necesarias para usar las bibliotecas instaladas

- `.-` Usar el Makefile para obtener el programa ejecutable
  - `make`

Si la sintaxis y la versión de la biblioteca instalada es correcta se obtiene el ejecutable.

# Creación del ejecutable en Linux

## Pasos:

- Creación del archivo de proyecto independiente de plataforma
  - `qmake -project -o holaQT.pro`

Se obtiene el archivo `holaQT.pro`

- Creación de un Makefile dependiente de la plataforma usada
  - `qmake holaQT.pro`

Se obtiene el Makefile, con las instrucciones necesarias para usar las bibliotecas instaladas

- **.- Usar el Makefile para obtener el programa ejecutable**
  - `make`

Si la sintaxis y la versión de la biblioteca instalada es correcta se obtiene el ejecutable.

# Creación del ejecutable en Linux

## Pasos:

- Creación del archivo de proyecto independiente de plataforma
  - `qmake -project -o holaQT.pro`

Se obtiene el archivo `holaQT.pro`

- Creación de un Makefile dependiente de la plataforma usada
  - `qmake holaQT.pro`

Se obtiene el Makefile, con las instrucciones necesarias para usar las bibliotecas instaladas

- `.-` Usar el Makefile para obtener el programa ejecutable
  - `make`

Si la sintaxis y la versión de la biblioteca instalada es correcta se obtiene el ejecutable.

# Creación del ejecutable en Linux

## Pasos:

- Creación del archivo de proyecto independiente de plataforma
  - `qmake -project -o holaQT.pro`

Se obtiene el archivo `holaQT.pro`

- Creación de un Makefile dependiente de la plataforma usada
  - `qmake holaQT.pro`

Se obtiene el Makefile, con las instrucciones necesarias para usar las bibliotecas instaladas

- `.-` Usar el Makefile para obtener el programa ejecutable
  - `make`

Si la sintaxis y la versión de la biblioteca instalada es correcta se obtiene el ejecutable.



A destacar:

- Incluir `QApplication` y `QLabel`
- Crear un objeto de la clase `QApplication`
- Método `show()`
- `return programa.exec();`

A destacar:

- Incluir QApplication y QLabel
- Crear un objeto de la clase QApplication
- Método show()
- `return programa.exec();`

A destacar:

- Incluir QApplication y QLabel
- Crear un objeto de la clase QApplication
- Método show()
- `return programa.exec();`

## A destacar:

- Incluir QApplication y QLabel
- Crear un objeto de la clase QApplication
- Método show()
- `return programa.exec();`

# Ejemplo 0: Hola Mundo en QT v0.2

```
#include <QApplication>
#include <QLabel>

int main(int argc, char *argv[]){
    QApplication programa(argc, argv);
    QLabel *etiqueta =
        new QLabel("<h2><i>Hola</i> <font color=red> Qt!</font></h2>");
    etiqueta->show();
    return programa.exec();
}
```

# Ejemplo 0: Hola Mundo en QT v0.2

```
#include <QApplication>
#include <QLabel>

int main(int argc, char *argv[]){
    QApplication programa(argc, argv);
    QLabel *etiqueta =
        new QLabel("<h2><i>Hola</i> <font color=red> Qt!</font></h2>");
    etiqueta->show();
    return programa.exec();
}
```



# Ejemplo 1: Layout

## Organización de componentes por medio de Layout

- Los contenedores(Layout) principales son objetos de las clases `QHBoxLayout`, `QVBoxLayout`, `QGridLayout` y `QFormLayout`
- Para agregar un `QWidget` a un layout se usa el método `addWidget`
- Es posible agregar un Layout a otro Layout por medio del método `addLayout`
- Si se invoca el método `show()` de un Layout, se invocará el método `show` a todos los los objetos contenidos en este.

ver ejemplo `organizarQT.cpp`

# Ejemplo 1: Layout

## Organización de componentes por medio de Layout

- Los contenedores(Layout) principales son objetos de las clases QHBoxLayout, QVBoxLayout, QGridLayout y QFormLayout
- Para agregar un QWidget a un layout se usa el método `addWidget`
- Es posible agregar un Layout a otro Layout por medio del método `addLayout`
- Si se invoca el método `show()` de un Layout, se invocará el método `show` a todos los los objetos contenidos en este.

ver ejemplo `organizarQT.cpp`



# Ejemplo 1: Layout

## Organización de componentes por medio de Layout

- Los contenedores(Layout) principales son objetos de las clases QHBoxLayout, QVBoxLayout, QGridLayout y QFormLayout
- Para agregar un QWidget a un layout se usa el método addWidget
- Es posible agregar un Layout a otro Layout por medio del método addLayout
- Si se invoca el método show() de un Layout, se invocará el método show a todos los los objetos contenidos en este.

ver ejemplo organizarQT.cpp

# Ejemplo 1: Layout

## Organización de componentes por medio de Layout

- Los contenedores(Layout) principales son objetos de las clases QHBoxLayout, QVBoxLayout, QGridLayout y QFormLayout
- Para agregar un QWidget a un layout se usa el método addWidget
- Es posible agregar un Layout a otro Layout por medio del método addLayout
- Si se invoca el método show() de un Layout, se invocará el método show a todos los los objetos contenidos en este.

ver ejemplo organizarQT.cpp

# Ejemplo 1: Layout

## Organización de componentes por medio de Layout

- Los contenedores(Layout) principales son objetos de las clases QHBoxLayout, QVBoxLayout, QGridLayout y QFormLayout
- Para agregar un QWidget a un layout se usa el método addWidget
- Es posible agregar un Layout a otro Layout por medio del método addLayout
- Si se invoca el método show() de un Layout, se invocará el método show a todos los los objetos contenidos en este.

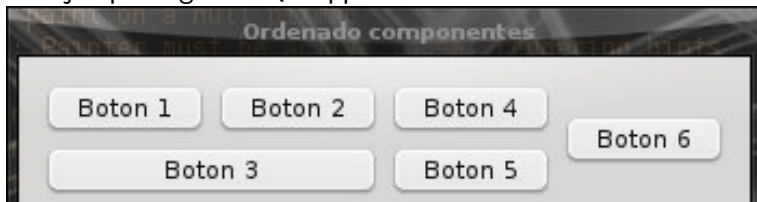
ver ejemplo organizarQT.cpp

## Ejemplo 1: Layout

### Organización de componentes por medio de Layout

- Los contenedores(**Layout**) principales son objetos de las clases QHBoxLayout, QVBoxLayout, QGridLayout y QFormLayout
- Para agregar un QWidget a un layout se usa el método addWidget
- Es posible agregar un Layout a otro Layout por medio del método addLayout
- Si se invoca el método show() de un Layout, se invocará el método show a todos los los objetos contenidos en este.

ver ejemplo organizarQT.cpp



# Conexión entre eventos y métodos

- Los **Qt Signals** y los **Qt slots** se utilizan para la comunicación entre objetos. Los signals y el mecanismo de slots es un elemento central de Qt y probablemente la parte que más se diferencia de otros entornos de desarrollo.
- Un signal se emite cuando se produce un evento en particular. Los Widgets de Qt tienen signals predefinidas, pero siempre se pueden crear subclases a partir de QWidgetets para agregar nuevos signals propios.
- Un slot es un método que es invocado en respuesta a un signal particular. Los Widgets de Qt tienen slots predefinidas, pero es una práctica común derivar subclases a partir de QWidgetets y añadir nuevos slots que pueda gestionar los signals que se desean capturar.

# Conexión entre eventos y métodos

- Los **Qt Signals** y los **Qt slots** se utilizan para la comunicación entre objetos. Los signals y el mecanismo de slots es un elemento central de Qt y probablemente la parte que más se diferencia de otros entornos de desarrollo.
- Un signal se emite cuando se produce un evento en particular. Los Widgets de Qt tienen signals predefinidas, pero siempre se pueden crear subclases a partir de QWidgetets para agregar nuevos signals propios.
- Un slot es un método que es invocado en respuesta a un signal particular. Los Widgets de Qt tienen slots predefinidas, pero es una práctica común derivar subclases a partir de QWidgetets y añadir nuevos slots que pueda gestionar los signals que se desean capturar.

# Conexión entre eventos y métodos

- Los **Qt Signals** y los **Qt slots** se utilizan para la comunicación entre objetos. Los signals y el mecanismo de slots es un elemento central de Qt y probablemente la parte que más se diferencia de otros entornos de desarrollo.
- Un signal se emite cuando se produce un evento en particular. Los Widgets de Qt tienen signals predefinidas, pero siempre se pueden crear subclases a partir de QWidgets para agregar nuevos signals propios.
- Un slot es un método que es invocado en respuesta a un signal particular. Los Widgets de Qt tienen slots predefinidas, pero es una práctica común derivar subclases a partir de QWidgets y añadir nuevos slots que pueda gestionar los signals que se desean capturar.

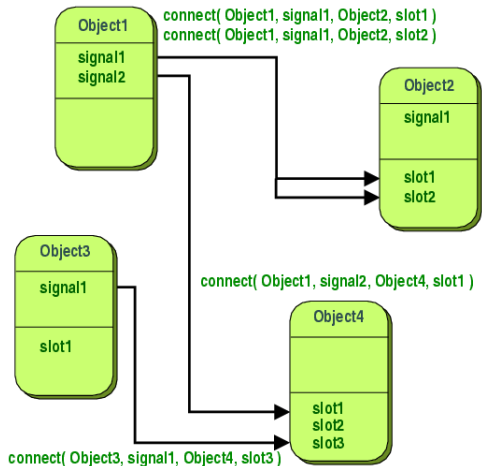
# Conexión entre eventos y métodos

- Los **Qt Signals** y los **Qt slots** se utilizan para la comunicación entre objetos. Los signals y el mecanismo de slots es un elemento central de Qt y probablemente la parte que más se diferencia de otros entornos de desarrollo.
- Un signal se emite cuando se produce un evento en particular. Los Widgets de Qt tienen signals predefinidas, pero siempre se pueden crear subclases a partir de QWidgets para agregar nuevos signals propios.
- Un slot es un método que es invocado en respuesta a un signal particular. Los Widgets de Qt tienen slots predefinidas, pero es una práctica común derivar subclases a partir de QWidgets y añadir nuevos slots que pueda gestionar los signals que se desean capturar.

No existe relación entre los Signals de UNIX y los Signals de Qt.



# Mecanismos de Signals y Slots



## Ejemplo 2: Detección de números primos usando QtSignals y QtSlots

```
#include <QObject>
#include <stdlib.h>
#include <time.h>

class CNumeros: public QObject{
    Q_OBJECT

    int esPrimo(long);

public:
    CNumeros();

    long crearNumeros();
signals:
    void nuevoPrimo(long);
};
```

## Ejemplo 2: Detección de números primos usando QtSignals y QtSlots

```
long CNumeros::crearNumeros(){  
    long x;  
  
    x = rand()%100 +1;  
  
    if (this->esPrimo(x))  
        emit nuevoPrimo(x);  
  
    return(x);  
}
```

## Ejemplo 2: Detección de números primos usando QtSignals y QtSlots

```
#include <QObject>
#include <iostream>

class CDetectaPrimo: public QObject{
    Q_OBJECT

    public slots:
        void primoCapturado(long);
};
```

## Ejemplo 2: Detección de números primos usando QtSignals y QtSlots

```
int main() {  
    CNumeros nuevo;  
    CDetectaPrimo detector;  
    CDetectaMenores detectorM(50);  
    long x;  
  
    QObject::connect(&nuevo, SIGNAL(nuevoPrimo(long)), &detector, SLOT(primoCapturado(long)));  
    QObject::connect(&nuevo, SIGNAL(nuevoPrimo(long)), &detectorM, SLOT(esMenor(long)));  
  
    for (;;) {  
        x = nuevo.crearNumeros();  
  
        std::cout << "Numero: " << x << '\n';  
        sleep(1);  
    }  
  
    return(0);  
}
```

# Resumen de Signals y Slots

- Un Qt signal puede conectarse a muchos Qt slots.
- Muchas Qt signals pueden estar conectadas a un mismo Qt slot.
- Un Qt signal puede estar conectado a otro Qt signal.
- Las conexiones pueden pueden eliminarse.

# Resumen de Signals y Slots

- Un Qt signal puede conectarse a muchos Qt slots.
- Muchas Qt signals pueden estar conectadas a un mismo Qt slot.
- Un Qt signal puede estar conectado a otro Qt signal.
- Las conexiones pueden pueden eliminarse.

# Resumen de Signals y Slots

- Un Qt signal puede conectarse a muchos Qt slots.
- Muchas Qt signals pueden estar conectadas a un mismo Qt slot.
- Un Qt signal puede estar conectado a otro Qt signal.
- Las conexiones pueden pueden eliminarse.



# Resumen de Signals y Slots

- Un Qt signal puede conectarse a muchos Qt slots.
- Muchas Qt signals pueden estar conectadas a un mismo Qt slot.
- Un Qt signal puede estar conectado a otro Qt signal.
- Las conexiones pueden pueden eliminarse.

## Ejemplo 3: Qt Signal, Qt Slot y Qt Widgets

```
#include <QApplication>
#include <QPushButton>

int main(int argc, char *argv[])
{
    QApplication programa(argc, argv);
    QPushButton *boton = new QPushButton("Salir");
    //
    // Conexión del evento click sobre el boton con el método quit de programa
    QObject::connect(boton, SIGNAL(clicked()), &programa, SLOT(quit()));
    //
    boton->show();
    return programa.exec();
}
```

## Ejemplo 4: Más Qt Widgets



## Ejemplo 4: Lista de componentes requeridos en el Dialogo del

- 3 Objetos de la clase QLabel
- 2 Objetos de la clase QLineEdit
- 2 Objetos de la clase QPushButton
- 1 Qt Slot para gestionar la orden de sumar
- Los contenedores necesarios para organizar los componentes.

## Ejemplo 4: Lista de componentes requeridos en el Dialogo del

- 3 Objetos de la clase QLabel
- 2 Objetos de la clase QLineEdit
- 2 Objetos de la clase QPushButton
- 1 Qt Slot para gestionar la orden de sumar
- Los contenedores necesarios para organizar los componentes.

## Ejemplo 4: Lista de componentes requeridos en el Dialogo del

- 3 Objetos de la clase QLabel
- 2 Objetos de la clase QLineEdit
- 2 Objetos de la clase QPushButton
- 1 Qt Slot para gestionar la orden de sumar
- Los contenedores necesarios para organizar los componentes.

## Ejemplo 4: Lista de componentes requeridos en el Dialogo del

- 3 Objetos de la clase QLabel
- 2 Objetos de la clase QLineEdit
- 2 Objetos de la clase QPushButton
- 1 Qt Slot para gestionar la orden de sumar
- Los contenedores necesarios para organizar los componentes.

## Ejemplo 4: Lista de componentes requeridos en el Dialogo del

- 3 Objetos de la clase QLabel
- 2 Objetos de la clase QLineEdit
- 2 Objetos de la clase QPushButton
- 1 Qt Slot para gestionar la orden de sumar
- Los contenedores necesarios para organizar los componentes.



## Ejemplo 4: Declaración de la clase CDialogoSuma

```
class QLabel;  
class QLineEdit;  
class QPushButton;  
  
class CDialogoSuma : public QDialog  
{  
    Q_OBJECT  
  
public:  
    CDialogoSuma(QWidget *parent = 0);  
  
private slots:  
    void sumarClicked();  
  
private:  
    QLabel *etiqueta1;  
    QLabel *etiqueta2;  
    QLabel *etiqueta3;  
    QLabel *resultado;  
    QLineEdit *sumando1;  
    QLineEdit *sumando2;  
    QPushButton *botonSumar;  
    QPushButton *botonFin;  
};
```

## Ejemplo 4: Conexiones

```
//Botón Sumar
connect(this->botonSumar, SIGNAL(clicked()),
        this, SLOT(sumarClicked()));
//Botón Salir
connect(botonFin, SIGNAL(clicked()),
        this, SLOT(close()));
```

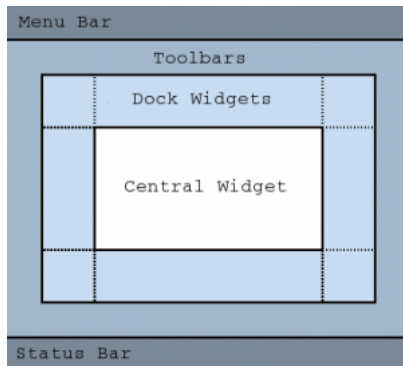
## Ejemplo 4: Slot sumarClicked()

```
void CDialogoSuma::sumarClicked()
{
    QString dato1 = this->sumando1->text();
    QString dato2 = this->sumando2->text();
    float dato3;

    dato3 = dato1.toFloat() + dato2.toFloat();

    this->resultado->setNum((double)dato3);
}
```

## Ejemplo 5: QMainWindow



Ver ejemplo `ejemplo5/ventanaPrincipal`

# Qt Designer

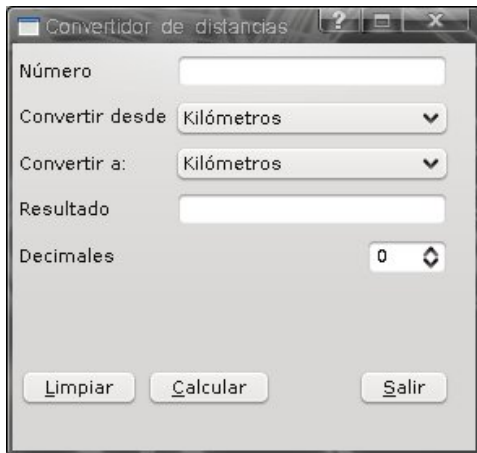
Qt designer es una utilidad de para diseñar y construir interfaces gráficas de usuario (GUIs) con componentes Qt.

# Qt Designer

Qt designer es una utilidad de para diseñar y construir interfaces gráficas de usuario (GUIs) con componentes Qt.

Gracias a QT podemos colocar un objetos de clases de la biblioteca de Qt de forma gráfica, permitiendo establecer la posición y el tamaño, entre otros atributos.

En el ejemplo 6 se presenta el archivo convertidor.ui realizado con Qt Designer que describe en XML la siguiente interfaz:



# Convertir una interfaz en un código C++

Para obtener el código equivalente en C++ que produzca la interfaz de la lámina anterior use el compilador de interfaces de Qt



# Convertir una interfaz en un código C++

Para obtener el código equivalente en C++ que produzca la interfaz de la lámina anterior use el compilador de interfaces de Qt. La instrucción a usar es:

```
uic-qt4 convertidor.ui -o ui_convertidor.h
```

# Método ::setupUi

Observe el archivo generado destacando:

- Archivos incluidos
- Atributos públicos
- Método setupUi
- Espacio de nombres Ui

# Método ::setupUi

Observe el archivo generado destacando:

- Archivos incluidos
- Atributos públicos
- Método setupUi
- Espacio de nombres Ui

# Método ::setupUi

Observe el archivo generado destacando:

- Archivos incluidos
- Atributos públicos
- Método setupUi
- Espacio de nombres Ui

# Método ::setupUi

Observe el archivo generado destacando:

- Archivos incluidos
- Atributos públicos
- Método setupUi
- Espacio de nombres Ui

# Método ::setupUi

Observe el archivo generado destacando:

- Archivos incluidos
- Atributos públicos
- Método setupUi
- Espacio de nombres Ui

¿Qué hacer con el código generado ?

# Una solución:

```
#ifndef CONVERTIDOR_H
#define CONVERTIDOR_H

#include<QDialog>
#include<ui_convertidor.h>

class CConvertidor: public QDialog, public Ui::convertidor{
    Q_OBJECT

public:
    CConvertidor(QWidget *wPadre = 0);

private slots:
    void convertir();
};
#endif
```

Nota: En el constructor se debe invocar el método `setupUi` ¿Por qué?

# El programa principal

Una vez completado el código, declarado y definidos los Qt Slots, el programa principal puede ser parecido a:

```
#include <QApplication>
#include <QDialog>
#include "convertidor.h"

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    CConvertidor *dialogo = new CConvertidor;
    dialogo->show();
    return app.exec();
}
```



# Bibliografía

- <http://doc.trolltech.com/4.4/tutorials.html>
- <http://www.programacion-linux.com/interfaz-grafica-de-usuario/introduccion-a-qt>
- Jasmin Blanchette & Mark Summerfield, "C++ GUI Programming with Qt 4", Second Edition

Presentación hecha en L<sup>A</sup>T<sub>E</sub>X