FIEC

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

PROGRAMACIÓN APLICADA A LA AUTOMATIZACIÓN

M.Sc. Alexander Prieto León

Conferencia IX

Unidad III Programación visual y orientada a eventos para la creación de HMI.

3.1 RAD y la jerarquía de clases.

3.2 Componentes visuales y manejo de eventos.

Objetivos:

- Explicar los conceptos básicos y las principales definiciones relacionadas con la programación visual.
- Enunciar las características y principales elementos del Framework Qt.
- Identificar los principales elementos de la jerarquía de clases de Qt.
- Identificar los componentes básicos de un programa visual.
- Explicar los conceptos relacionados con eventos.
- Explicar cómo se manejan los eventos en Qt.

BIBLIOGRAFÍA

Bibliografía:

- Zhi Eng, Lee; Rischpater, Ray. Application Development with Qt Creator: Build cross-platform applications and GUIs using Qt 5 and C++, 3rd Edition. (2020). Packt Publishing. ISBN-10: 1789951755, ISBN-13: 978-1789951752.
- http://www.qtrac.eu/C++-GUI-Programming-with-Qt-4-1st-ed.zip
- Qt 5.14.2 Reference Documentation. Qt Creator Help.
- https://doc.qt.io/qtcreator/index.html
- https://doc.qt.io/qt-5/qtdesigner-manual.html

- **En la clase anterior:**
  - **¿Qué es el polimorfismo?**
  - **Solución de la práctica 8.**

- ¿Qué herramientas conocen para realizar programación visual con C++?

  - R/ C++ Builder, Visual C++, Eclipse,…, **Qt**. El lenguaje de alto nivel que se ha tratado durante la asignatura es el C++, que ya recibieron en clases precedentes, deben dominar y que es probablemente el lenguaje más utilizado en la ingeniería. El ambiente de programación será el Qt Creator, que es un ambiente de desarrollo creado por Nokia preparado para la programación guiada por eventos y visual.

- ¿Por qué se elige Qt?

  - R/ Es software libre y es multiplataforma lo que permite crear aplicaciones sin necesidad de pagar una licencia y se pueden crear aplicaciones para diversos sistemas operativos como Windows, Linux, Unix, Android, Symbian y otros.

Rapid Application Development (RAD)

- Los lenguajes de programación visual, también conocidos como lenguajes de 4ta generación son aquellos en los que cierto número de tareas, casi siempre comunes a muchos programas, se pueden realizar sin necesidad de escribir código, simplemente realizando determinadas operaciones con el ratón.

- Estas acciones que realiza el programador al diseñar la aplicación producen código fuente de forma automática. Este código que se genera de forma automática posteriormente debe ser completado por el programador para llevar a cabo otras tareas que son más específicas para la aplicación en particular.

Programas secuenciales y programas interactivos.

- Un programa **secuencial** es aquel en el cual es posible conocer el orden en que se van a ejecutar todas sus funciones. Estos programas son típicos del procesamiento por lotes (batch) y normalmente la intervención del usuario está muy restringida.

- Los programas **interactivos** son aquellos en que el orden de ejecución de algunas de sus funciones depende de la intervención del usuario y por lo tanto no es posible conocer de antemano el orden en que van a ser ejecutadas todas sus funciones; su orden depende de las decisiones del que ejecute la aplicación.

Programación Orientada a Eventos

- En Windows y otros sistemas operativos modernos se utiliza la programación guiada por eventos (manejando mensajes), la cual nos permite la realización de programas interactivos que nos facilitan, entre otras muchas cosas, manejar de forma mucho más sencilla la entrada de información desde cualquier dispositivo de entrada.

- Estos programas se caracterizan por estar constantemente chequeando los mensajes que le llegan (información de la ocurrencia de sucesos), y convirtiéndolos en eventos que es todo el mecanismo de manejo de los mensajes.

Programación Orientada a Eventos (EDP en inglés)

- Ejemplos de Eventos:
  - Los más clásicos son del mouse (click, doble click, click derecho, presionar, soltar, alternar).
  - Presionar teclas del teclado.
  - Abrir o cerrar una nueva ventana.
  - Cambiar texto o valor en un widget o componente visual.
  - Eventos de tiempo o temporización…

- **El Framework o Ambiente de Desarrollo Integrado (IDE) Qt.**
  - Se publicó por vez primera en 1995, creado por Haavard Nord y Eirik Chambe-Eng (noruegos ambos, NOKIA), que luego también fundaron la compañía que hoy se llama Trolltech Inc.(en USA).
  - Qt surgió ante la necesidad de crear una aplicación de bases de datos en C++ para imágenes de ultrasonido con GUI, que pudiera ser utilizada en Unix, Macintosh y Windows.
  - En abril de 1997 Qt 1.2 se convirtió en el estándar para el desarrollo de aplicaciones visuales en C++ sobre Linux. Aun hoy, luego de su quinta versión, por ser código abierto y por tener grandes facilidades para aplicaciones en sistemas empotrados y móviles sigue ganando adeptos significativamente.

- **El Framework o Ambiente de Desarrollo Integrado (IDE) Qt.**
  - ¿Qué significa que es un **framework multiplataforma**?

    Esto significa que las aplicaciones que sean creadas con el pueden ser compiladas para diferentes plataformas, sistemas operativos por ejemplo, sin necesidad de realizar cambios en el código de la aplicación.

  - ¿Qt es un **superlenguaje** y qué es un superlenguaje?

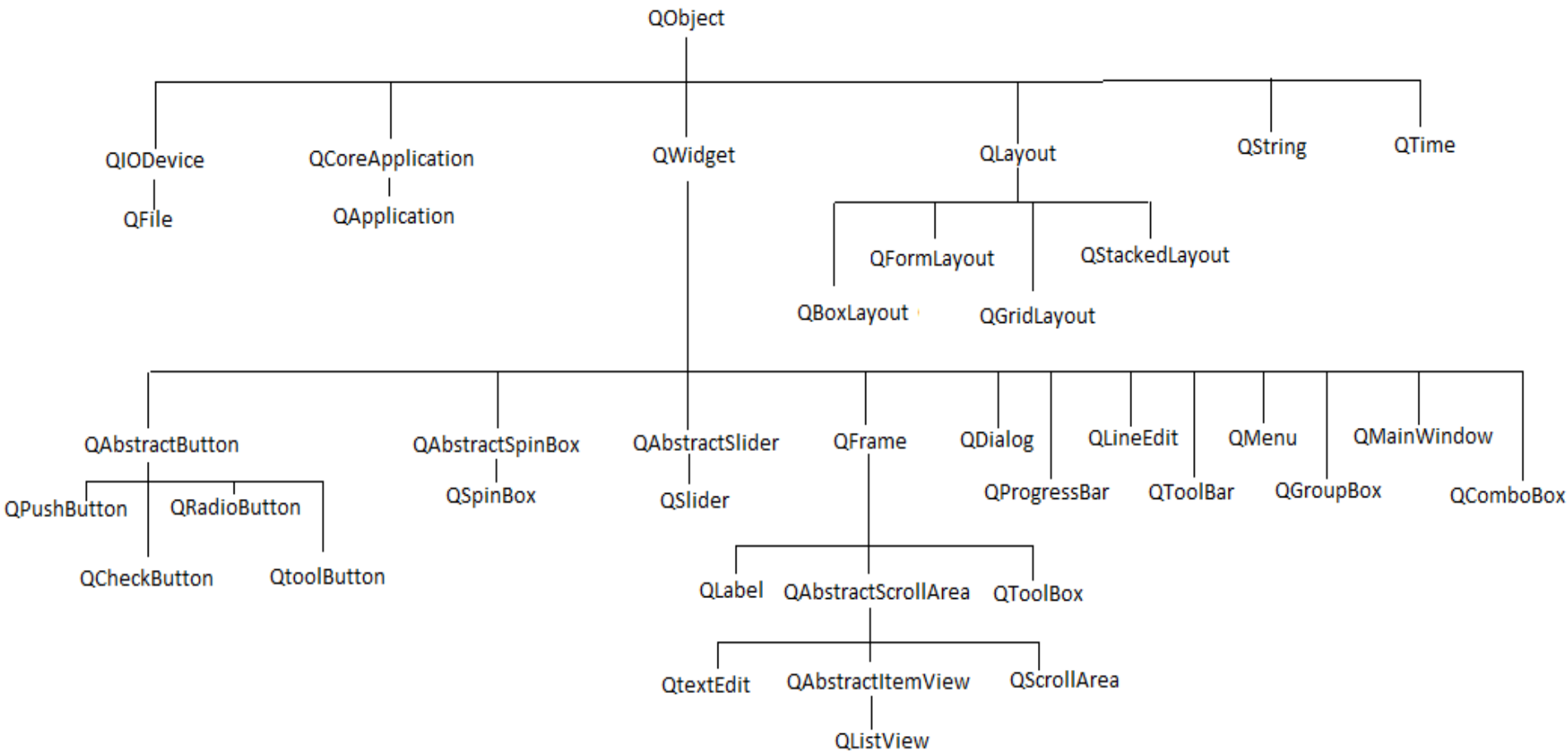
    En el ambiente de desarrollo Qt creator encontrarás todos los elementos de programación de C++, pero además, existen un conjunto de nuevos elementos propios del lenguaje Qt, por lo que se dice que es un **superlenguaje**.

Jerarquía de clases de Qt.

- La biblioteca de clases de Qt es bastante extensa y abarca una gran gama de funcionalidades. Esto se puede apreciar claramente en la documentación de Qt, al buscar la sección de **AllClass** o buscar **InheritanceHierarchy.**

- En la figura de la siguiente diapositiva se puede observar una parte de la jerarquía de clases del framework Qt. En esta se encuentran presentes algunas de las clases con las que más estaremos trabajando en el curso.

Jerarquía de clases de Qt.



➢ Jerarquía de clases de Qt.

▯ QObject:

QObject es la clase base para todos los objetos de Qt. En esta se implementan un conjunto de funcionalidades comunes a todos los objetos, tales como el mecanismo Signal & Slot, las Property, el manejador de eventos, de tiempo y de hilos.

➢ Jerarquía de clases de Qt.

▯ QApplication:

Es la clase encargada de llevar el flujo de control de las aplicaciones gráficas de los usuarios y sus principales configuraciones. Esta clase contiene un lazo de evento donde son procesados todos los eventos que llegan desde el sistema operativo.

Todas las aplicaciones gráficas que usen Qt contienen un objeto de la clase QApplication sin importar la cantidad de ventanas que tenga la aplicación.

➢ Jerarquía de clases de Qt.

▯ QApplication:

Es la clase encargada de llevar el flujo de control de las aplicaciones gráficas de los usuarios y sus principales configuraciones. Esta clase contiene un lazo de evento donde son procesados todos los eventos que llegan desde el sistema operativo.

Todas las aplicaciones gráficas que usen Qt contienen un objeto de la clase QApplication sin importar la cantidad de ventanas que tenga la aplicación.

Jerarquía de clases de Qt.

- **QWidget:** En la terminología de Qt y Unix**,** un **Widget** es un elemento visual en la interfaz de usuario**.** El término proviene de "window gadget" y es equivalente a lo que otros desarrolladores llaman **Componente** o **Control** en la terminología de Windows. Todos los objetos que se visualizan en una ventana de Windows u otro sistema incluida la misma ventana son Widgets, por ejemplo, un botón, una etiqueta, una caja de edición de texto, una ventana de diálogo, etc…

- **QWidget** es la clase base para todos los objetos de interfaz de usuario. Ella recibe los eventos del mouse, teclado y otros del S.O. También pinta una representación del mismo en la pantalla.

Jerarquía de clases de Qt.

- **QWidget:** Un widget que no se encuentra contenido dentro de otro widget es llamado ventana. En Qt, **QMainWindow** y varias subclases de **QDialog** son los tipos de ventanas más comunes.

- El constructor de la clase **QWidget** acepta uno o dos parámetros estándar:
  - **QWidget** *parent = 0. Si es 0 (por defecto) el nuevo widget será una ventana. De lo contrario, este será hijo de un widget padre y estará limitado por la geometría del widget padre.
  - Qt::WindowFlags f = 0. Mediante este argumento se pueden cambiar algunas características de los widget que se representaran como ventanas. El valor por defecto (0) incluye características comunes a todos los widget.

Jerarquía de clases de Qt.

- **QWidget:**
- Esta clase encapsula funciones y propiedades para diferentes usos como pueden ser:
  - Funciones de ventanas, ej: show(), hide(), raise(), lower(), close().
  - Top-Level Windows, ej: windowTitle, etc.
  - Contenido de la ventana, ej: update(), repaint(), scroll(),etc.
  - Geometría, ej: width(), height(), move(), resize(),etc.
  - Modo, ej: isWindow(),visibleRegion(), etc.
  - Estilo, ej: style(), fontInfo(), etc.
  - Foco del teclado, ej: setFocus(), clearFocus(), etc.
  - Manejo de eventos, ej: event(), mousePressEvent(),changeEvent(), etc.
  - Sistema, ej: window(), setParent(),etc.

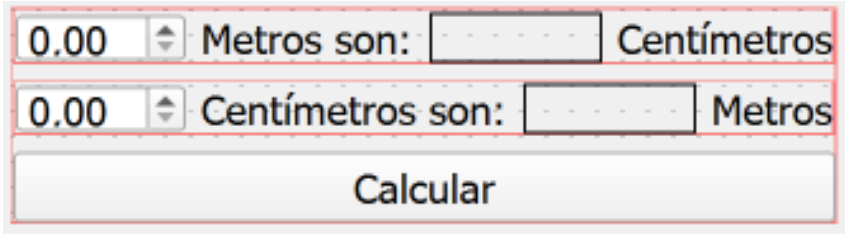
Jerarquía de clases de Qt.

- **QMainWindow:**

Esta es la clase que por lo general define la ventana principal de una aplicación. Esta clase provee su propio layout al cual se le pueden añadir **QToolBars**, **QDockWidgets**, un **QMenuBar**, y un **QStatusBar**. El área central del layout podrá contener todos los widget que se quieran visualizar en la aplicación.

Jerarquía de clases de Qt.

- **QLayout:**



Esta es la clase base de las clases manejadoras de geometría. Existen varias subclases que se derivan de esta como **QBoxLayout** y **QGridLayout**. El objetivo de todas es obtener una mejor alineación de los widget que se encuentran dentro del layout así como un redimensionamiento automático de dichos widgets.

Jerarquía de clases de Qt.

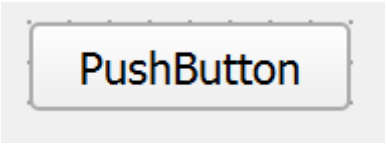
QString:

Esta clase provee las funcionalidades para trabajar con cadenas de caracteres Unicode.

Función	Uso
QString::number(float) QStringObject.setNum(float)	Convierte número a QString
QStringObject. toInt() QStringObject. toDouble()	Convierte QString a número
QStringObject.append(QString) QStringObject.Operator+=(QString)	Le adiciona al final un QString
QStringObject.length()	Retorna el tamaño de la cadena

Jerarquía de clases de Qt.

QPushButton:

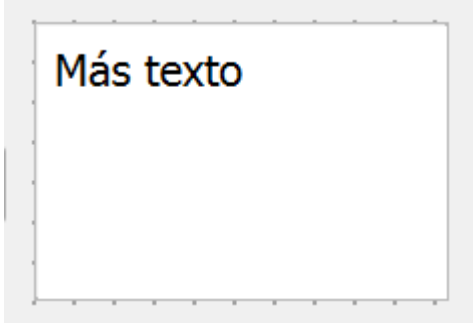


QPushButton es la clase que provee las principales funcionalidades de los botones. Este es posiblemente el widget más comúnmente usado en las aplicaciones de interfaz gráfica.

Los botones son usados para indicarle a la computadora la ejecución de alguna acción. Estos botones generalmente tienen forma rectangular y muestran un texto describiendo la acción a ejecutar.

Jerarquía de clases de Qt.

- **QTextEdit:**



Este widget provee de las funcionalidades para editar y mostrar texto plano y enriquecido. El widget **QTextEdit** es un avanzado visualizador/editor WYSIWYG que soporta formato de texto enriquecido usando etiquetas HTML. Este está optimizado para manejar largos documentos de textos y responder rápidamente a las entradas de los usuarios.

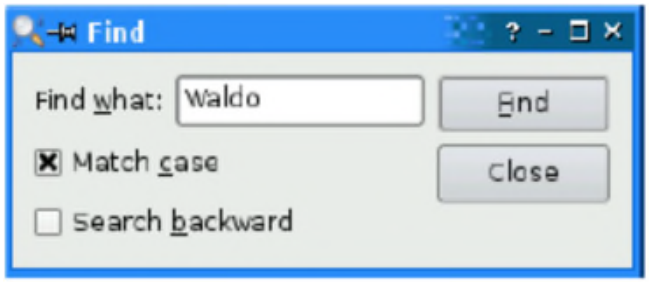
Jerarquía de clases de Qt.

- **QLabel:**
- Este widget es utilizado para mostrar texto o una imagen. Un QLabel puede contener uno de los siguientes tipos de contenido:

Tipo de contenido	Entrada
Texto plano	Pasando un **QString** mediante setText()
Texto enriquecido	Pasando un **QString** mediante setText()
Pixmap	Pasando un **QPixmap** mediante setPixmap()
Una película	Pasando un **QMovie** mediante setMovie()
Un número	Pasando un **int** o un **double** mediante setNum(), el cual convierte el número en texto plano.
Nada	Este es el valor por defecto.

Jerarquía de clases de Qt.

- **QDialog:**



Esta es la clase base de todos las ventanas de diálogos. Una ventana de diálogo es un top-level window mayormente usadas para tareas pequeñas y de rápida comunicación con el usuario.

Estas pueden ser modales o no. Además pueden proveer un valor de retorno y tener botones por defectos. La localización por defecto de las ventanas de diálogos será en el centro del área de su padre.

Jerarquía de clases de Qt.

- **QLineEdit:**

Este widget permite al usuario entrar y editar una simple línea de texto plano con una muy útil colección de funciones de edición, incluyendo, hacer y deshacer, cortar y pegar y drag & drop.

Jerarquía de clases de Qt.

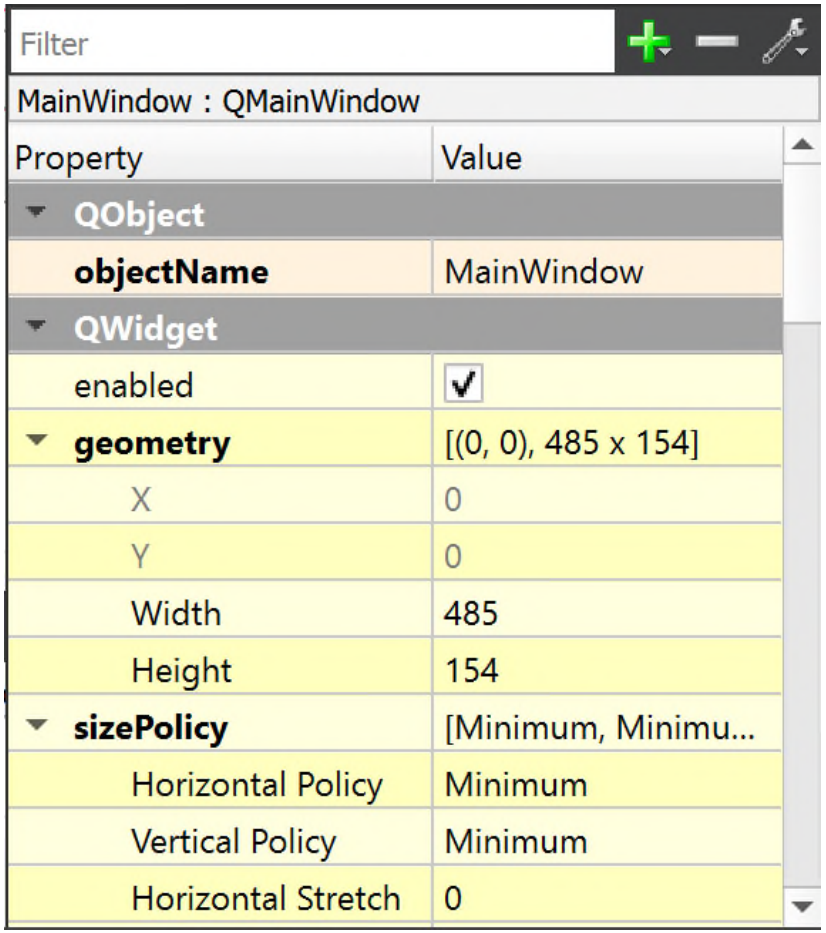
- **QSpinBox:** 

Es un widget para la entrada de números enteros que permite aumentar o disminuir el número utilizando dos botones que incluye en su parte derecha. A este se le puede configurar límites máximos y mínimos, el paso con el que cambia el número y con la función **value()** se puede obtener su valor.

Las Propiedades.

- Bajo la filosofía RAD el usar métodos de acceso a miembros privado o protegidos (get y set) es inconveniente por el trabajo y el tiempo que requiere.
- Las propiedades facilitan este desarrollo rápido de manera visual pues se tendrá fácil acceso mediante una tabla en el IDE.

Property Editor



Las Propiedades.

- Para entender en qué consiste veamos la siguiente tabla que muestra diferentes elementos de una clase **TCuadrado**:

Propiedad	Función de Lectura	Función de Escritura	Dato privado asociado
int Lado;	int getL() { return L; }	void setL(int lad) { if (Lad>=0)L=Lad; }	int L;
float Area;	float getArea() { return L*L; }	No tiene	No tiene

- **Lado=5;** //llama automáticamente a la función **setL(5)**;

- También se puede crear una propiedad que no tenga directamente un dato privado asociado como es el caso de la propiedad **Area** en la tabla.

El primer programa visual.

- Se desea implementar un programa que muestre en una ventana de Windows el texto: "Mi primer programa visual".
  - La mayoría de las aplicaciones gráficas en Qt utilizan como formulario principal de la aplicación (ventana principal) a un widget del tipo **QMainWindow** o **QDialog**, pero hay que destacar que Qt es tan flexible que cualquier widget puede ser un formulario.
  - En el ejemplo que se desarrolla a continuación se utilizará a **QMainWindow** como formulario de la aplicación, aunque en próximas conferencias se abordará más detalladamente este widget.

El primer programa visual.

- Para ello se crea un nuevo proyecto del tipo Qt Widget Application y se especifica que se utilizará como clase base de la ventana principal a **QMainWindow**.

```
01    #include <QApplication>
02    #include "mainwindow.h"
03    int main(int argc, char *argv[])
04    {
05            QApplication a(argc, argv);
06            MainWindow w;
07            w.show();
08            return a.exec();
09    }
```

El primer programa visual.

- Las líneas 1 y 2 incluyen las definiciones de las clases **QApplication** y MainWindow. Esta última una clase hereda directamente de **QMainWindow** y encapsula las funcionalidades de la ventana principal de la aplicación. Para todas las clases de Qt existe un archivo de encabezamiento (.h) con el mismo nombre.
- Línea 5 crea un objeto **QApplication** para gestionar los recursos de toda la aplicación. El constructor tiene dos argumentos: argc y argv, pues Qt soporta argumentos de línea de comandos.
- La línea 6 crea un objeto "w" de un widget de tipo **QMainWindow** a través de la clase MainWindow que hereda de esta.
- La línea 7 hace visible el objeto "w" ya que estos siempre se crean ocultos.
- La línea 8 pasa el control a Qt. En este punto, el programa entra en un bucle de eventos. Esta es una especie de modo stand-by, el programa espera acciones del usuario, como los clics de ratón o pulsaciones de teclas. Las acciones del usuario generar eventos (también llamados "mensajes") que el programa puede responder llamando a una o más funciones.

El primer programa visual.

- Veamos ahora los archivos de la clase MainWindow, comencemos por el archivo cabecera.

```
01  #ifndef MAINWINDOW_H
02  #define MAINWINDOW_H
03  #include <QMainWindow>
04  namespace Ui {
05      class MainWindow;
06  }
07  class MainWindow : public QMainWindow
08  {
09      Q_OBJECT
10  public:
11      explicit MainWindow(QWidget *parent = 0);
12      ~MainWindow();
13  private:
14      Ui::MainWindow *ui;
15  };
16  #endif // MAINWINDOW_H
```

El primer programa visual.

- Las líneas 1, 2 y 16 se utilizan para evitar múltiples inclusiones del archivo cabecera en el proyecto. En la línea 3 se incluye la definición de la clase **QMainWindow**.

- En las líneas 4, 5 y 6 se incluye el espacio de trabajo Ui, el cual Qt define en el fichero **ui_mainwindow.h**. En este espacio de trabajo también se define una clase llamada MainWindow (siempre se crea con el mismo nombre que el dado por el programador a la ventana principal). Esta nueva clase se utiliza para trabajar con el editor de la interfaz visual de Qt (Qt Designer) y en ella se declaran todos los widgets que se incluyan en la aplicación, y se inicializan entre otras cosas. De esta manera se logra separar el proceso de diseño de la interfaz visual de la implementación de sus funcionalidades. No importa que ambas clases se llamen iguales pues una de ellas está definida dentro el namespace Ui por lo que no existirá ambigüedad.

El primer programa visual.

- En la línea 7 se comienza la definición de la clase MainWindow (donde se implementarán las funcionalidades del formulario principal) la cual como se puede observar hereda públicamente de QMainWindow.
- La línea 9 incluye la macro **Q_OBJECT** para poder utilizar el mecanismo de señales y slots en esta clase. Cualquier clase que derive directa o indirectamente de **QObject** y quiere poder usar dicho mecanismo debe incluir dicha macro. Las señales y los slots serán explicados con mayor claridad en la próxima conferencia.
- En las líneas 11 y 12 se definen el constructor y destructor respectivamente de la clase.
- Finalmente en la línea 14 se declara un puntero llamado ui de la clase MainWindow perteneciente al espacio de trabajo Ui. Recuerde que esta es la clase que contiene el diseño de la interfaz visual de la aplicación, por lo que es lógico que esté contenida dentro de la clase donde se implementan las funcionalidades.

El primer programa visual.

- Sigamos ahora con el archivo de código correspondiente.

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

03      MainWindow::MainWindow(QWidget *parent):
04                              QMainWindow(parent),
05                              ui(new Ui::MainWindow)
06      {
07          ui->setupUi(this);
08      }

09      MainWindow::~MainWindow()
10      {
11          delete ui;
12      }
```

El primer programa visual.

- Las líneas 1 y 2 incluyen las definiciones de las clases MainWindow y Ui::MainWindow.

- En la línea 3 se comienza la implementación del constructor de la clase MainWindow, en la línea 4 se le pasan los parámetros al constructor de la clase base QMainWindow y en la 5 se inicializa el puntero ui con un objeto dinámico de la clase Ui::MainWindow.

- La línea 7 invoca a la función setupUi de la clase Ui::MainWindow la cual se encarga de inicializar todos los widgets que se encuentran en la interfaz visual. El puntero this es un puntero a sí mismo (class Mainwindow).

- En las líneas del 9 al 12 se implementa el destructor de la clase MainWindow cuya única función en este momento es eliminar el objeto dinámico apuntado por ui, lo cual se realiza en la línea 11 mediante el operador delete.

El primer programa visual.

- Al compilar esta aplicación aparecerá una ventana en blanco como resultado del programa que hemos creado.

- Ahora añadámosle un botón y un label al programa de tal manera que al dar un clic sobre el botón aparezca en el label "Mi primer programa visual".

- Para ello edite la interfaz visual del programa utilizando el IDE Qt Creator y añada un widget de tipo **QPushButton** y otro de tipo **QLabel**. Si usted es curioso y se fija en el fichero ui_mainwindow.h notará que en la clase Ui::MainWindow se han añadido dos punteros, cada uno de ellos a los widgets nuevos que aparecen en la interfaz de la aplicación.

El primer programa visual.

- Al dar clic derecho sobre el botón sale un menú emergente donde se puede seleccionar la opción **Go to slot...** mediante la cual se puede especificar la señal (evento) que se quiere procesar, en este ejemplo seleccionamos la señal **clicked()** que es la que se activa al dar un clic sobre el botón.
- Al hacer esto se adiciona en el fichero mainwindow.cpp el siguiente método para manejar el evento click izquierdo sobre el **pushButton.** Le adicionamos la línea de código para poner texto en el **Label**:

```
void MainWindow::on_pushButton_clicked()
{
   ui->label->setText("Mi primer programa visual");
}
```

- Fíjese que para acceder al label hay que hacerlo mediante el puntero ui, que es quien apunta al objeto de la clase que contiene todos los widget de la interfaz visual.

Los Eventos en Qt.

En Qt, los eventos son objetos, obtenidos de la clase abstracta **QEvent**, que representan las cosas que han ocurrido dentro de una aplicación o una actividad exterior de la que la aplicación tiene que estar al tanto. Los eventos pueden ser recibidos y manejados por cualquier instancia de una subclase de **QObject**, y estos son especialmente relevantes para los widget.

Función o Propiedad	Uso
objectName : QString	Para indicar el nombre del objeto.
bool **connect** (constQObject * sender, const char * signal, const char * method, Qt::ConnectionType type = Qt::AutoConnection) const	Para conectar una señal y un slot al objeto.
bool **disconnect** (const char * signal = 0, constQObject * receiver = 0, const char * method = 0)	Para desconectar una señal y un slot del objeto.
virtual void **timerEvent** (QTimerEvent * event)	Para manejar los eventos de tiempo.
Int **startTimer** (int interval)	Comienza el contador de tiempo de tiempo (Timer) para que se ejecute el evento del timer cada *interval* milisegundos.
virtual bool **event** (QEvent * e)	Para recibir los eventos de un objeto, devuelve *true* si el evento es reconocido y procesado.

Los Eventos en Qt.

- Cuando un evento ocurre, Qt crea un objeto de evento para representarlo mediante la construcción de una instancia de la apropiada subclase de **QEvent** y se lo entrega a una instancia particular de **QObject** llamando a su función **event()**.

- Esta función no maneja al evento en sí mismo; sino que sobre la base del tipo de evento entregado llama a un manejador de eventos para obtener ese tipo de evento específico y envía una respuesta en base a si el evento fue aceptado o rechazado.

- Algunos eventos, como **QMouseEvent** y **QKeyEvent**, vienen del sistema operativo; algunos, como **QTimerEvent**, son de otros orígenes; y otros vienen de la aplicación misma.

Los Eventos en Qt. Tipos de eventos

- La mayoría de los tipos de eventos en Qt tienen clases especiales, algunos de las más notables son **QResizeEvent**, **QPaintEvent**, **QMouseEvent**, **QKeyEvent**, y **QCloseEvent**. Cada una de ellas son subclases de **QEvent** y añaden funcionalidades específicas. Por ejemplo, **QResizeEvent** añade **size()** y **oldSize()** para habilitar a los widgets de darse cuenta de cómo sus dimensiones han cambiado.

- Algunas clases soportan más de un tipo de evento. **QMouseEvent** soporta los eventos de presionar los botones del mouse, los eventos de doble click, los de movimientos del mouse y otras operaciones relacionadas.

- Cada evento tiene un tipo asociado, definido en **QEvent::Type**, y estos pueden ser usados conveniente como una fuente de información en tiempo de ejecución para determinar cual subclase del objeto de evento dado fue instanciada.

CONCLUSIONES