

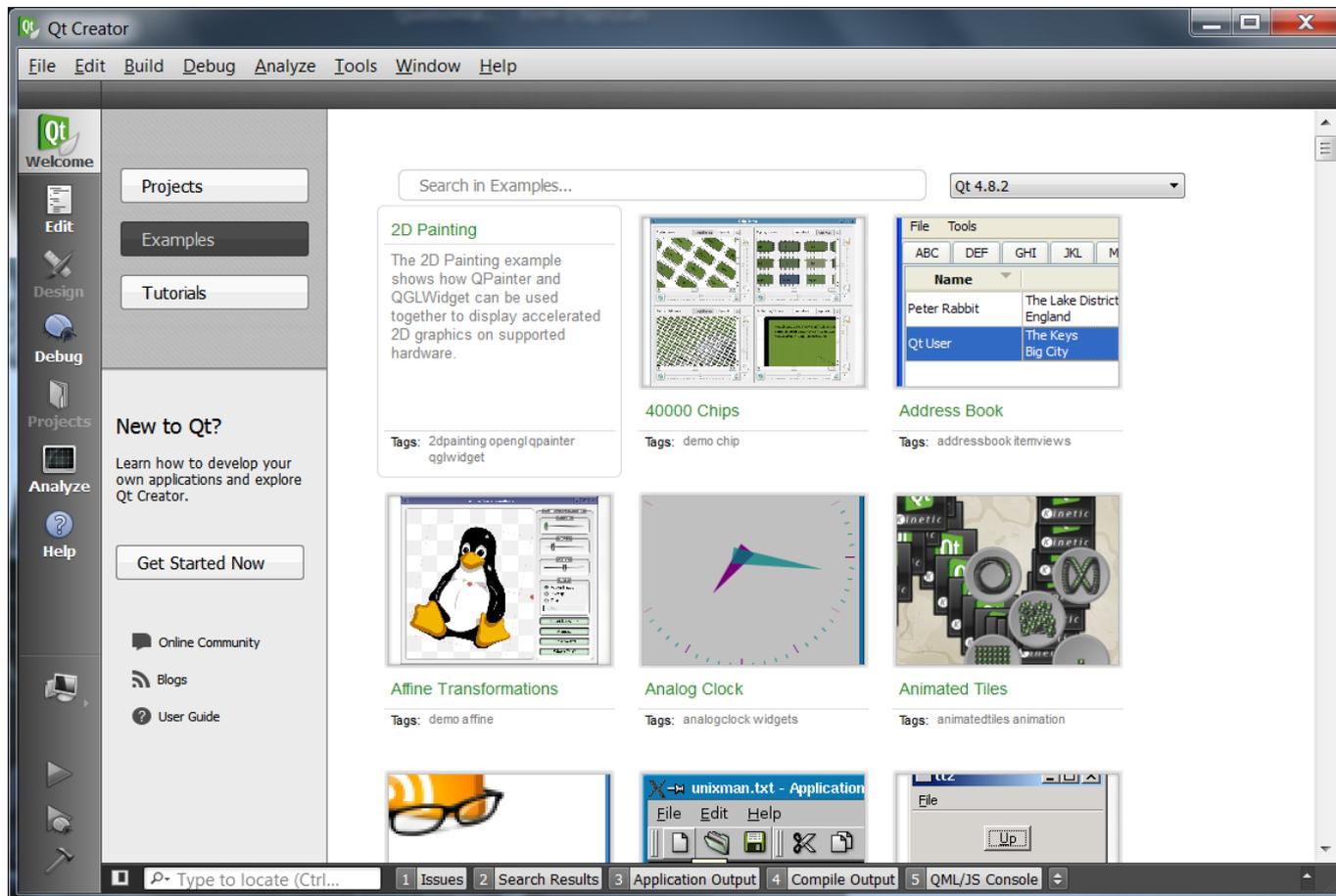
Aprendiendo a programar aplicaciones gráficas en C++ con Qt5

PRIMEROS PROGRAMAS SENCILLOS

Creando el primer programa

Realización de un programa sencillo: pedir un texto y pasarlo a mayúsculas

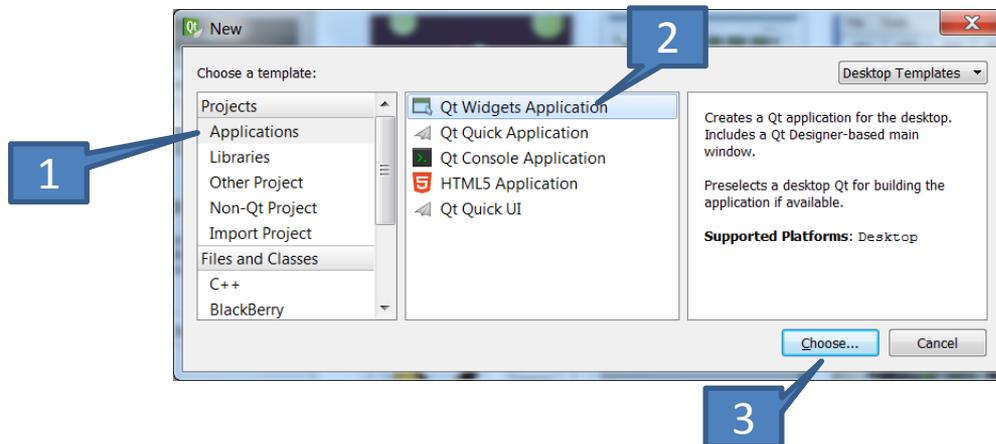
1) Arrancar QtCreator. Aparecerá una ventana como la siguiente



Creando el primer programa

Realización de un programa sencillo: pedir un texto y pasarlo a mayúsculas

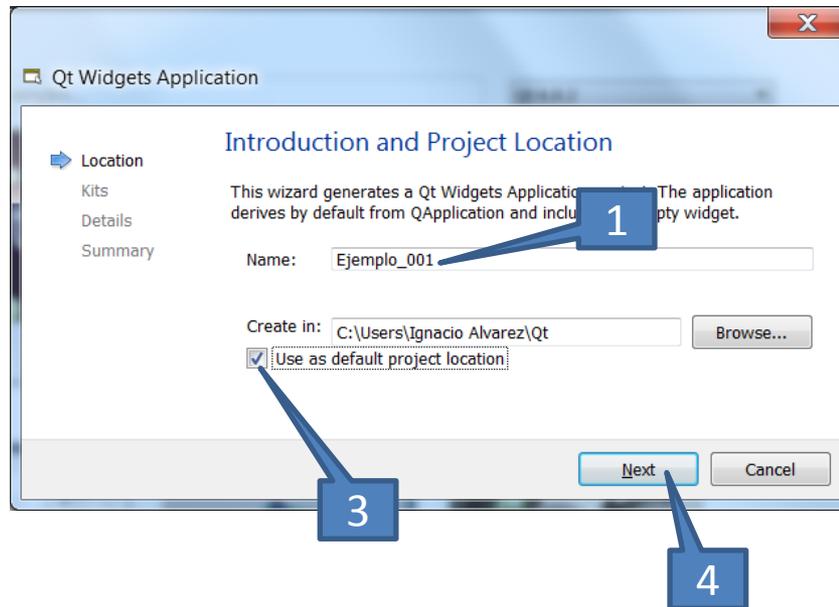
- 2) Seleccionar [File] ... [New File or Project].
- 3) En la ventana que aparece, seleccionar: [Applications] y [Qt Widgets Application]. A continuación [Choose...]
- 4) También es posible seleccionar [Qt Console Application] si se desea una aplicación modo consola.



Creando el primer programa

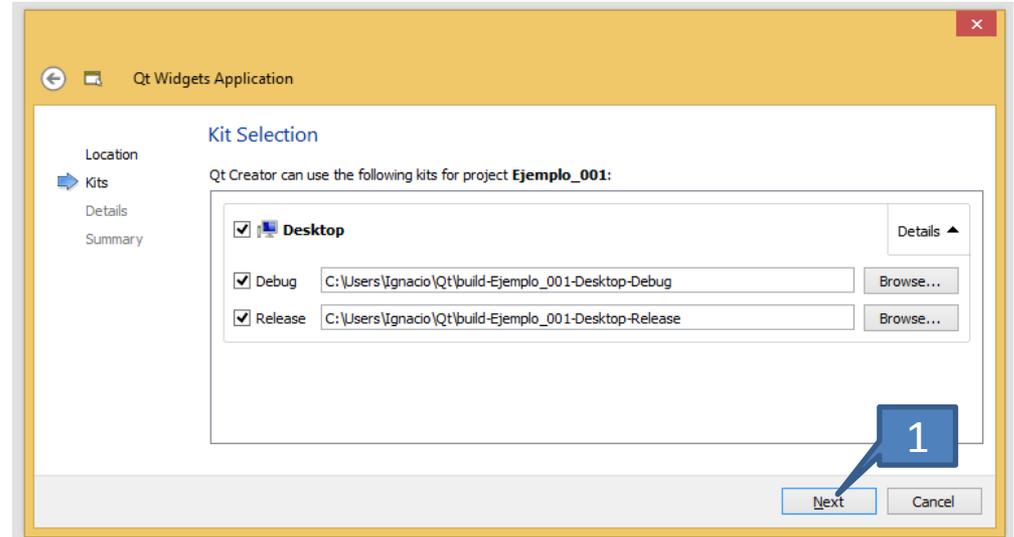
Realización de un programa sencillo: pedir un texto y pasarlo a mayúsculas

5) Elegir un nombre para el proyecto, y una carpeta donde guardarlo. Terminar pulsando [Next]

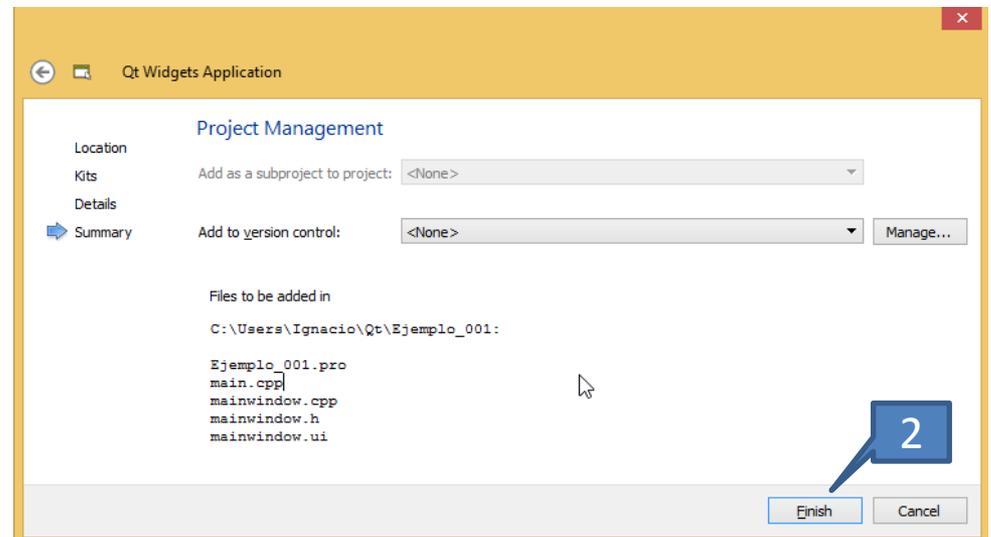


Creando el primer programa

6) A continuación, dejar las opciones que presenta tal y como están, y pulsar [Next].



7) Seguir haciendo lo mismo hasta llegar a la pantalla con el botón [Finish]. Púlsarlo para crear definitivamente.



Creando el primer programa

- 8) La ventana de programa toma un aspecto como el siguiente. Se han creado diversos archivos de forma automática y se han añadido al proyecto.



- 9) En la ventana de proyectos, abrir la pestaña [Forms] y dentro de ella hacer doble-click sobre mainwindow.ui (diseño del formato de la ventana del programa)

Creando el primer programa

10) Aparecerá la ventana de diseño del formulario, y se modifica el resto del interfaz, pasando a tener el aspecto siguiente:

DISEÑO DEL FORMULARIO
Aquí podemos arrastrar y después seleccionar los diferentes componentes que queremos que tenga nuestra aplicación, y colocarlos donde nos apetezca. Los componentes gráficos se llaman “Widgets”

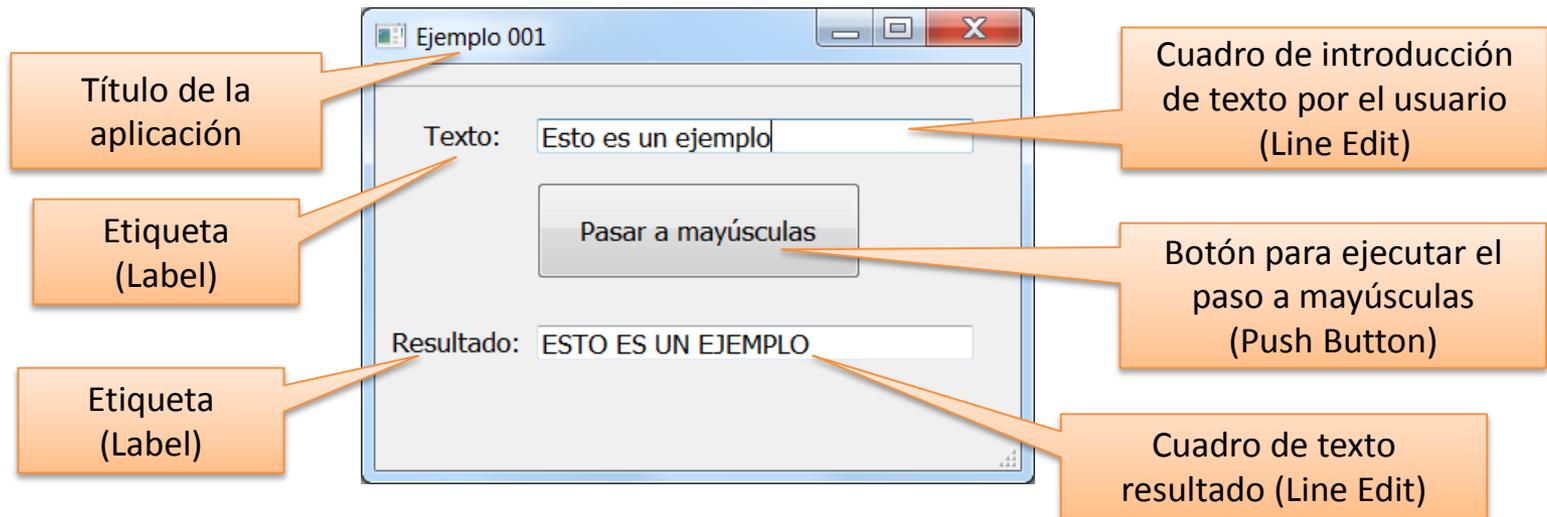
WIDGETS
Aquí podemos seleccionar los elementos que queremos incorporar a nuestro formulario, simplemente arrastrándolos a la posición deseada en la ventana DISEÑO DEL FORMULARIO

PROPIEDADES
Aquí podemos modificar las propiedades del Widget que hayamos seleccionado en la ventana de DISEÑO DEL FORMULARIO

objectName	MainWindow
windowModality	NonModal
enabled	<input checked="" type="checkbox"/>
geometry	[(0, 0), 400 x. ...]
X	0
Y	0
Width	
Height	

Creando el primer programa

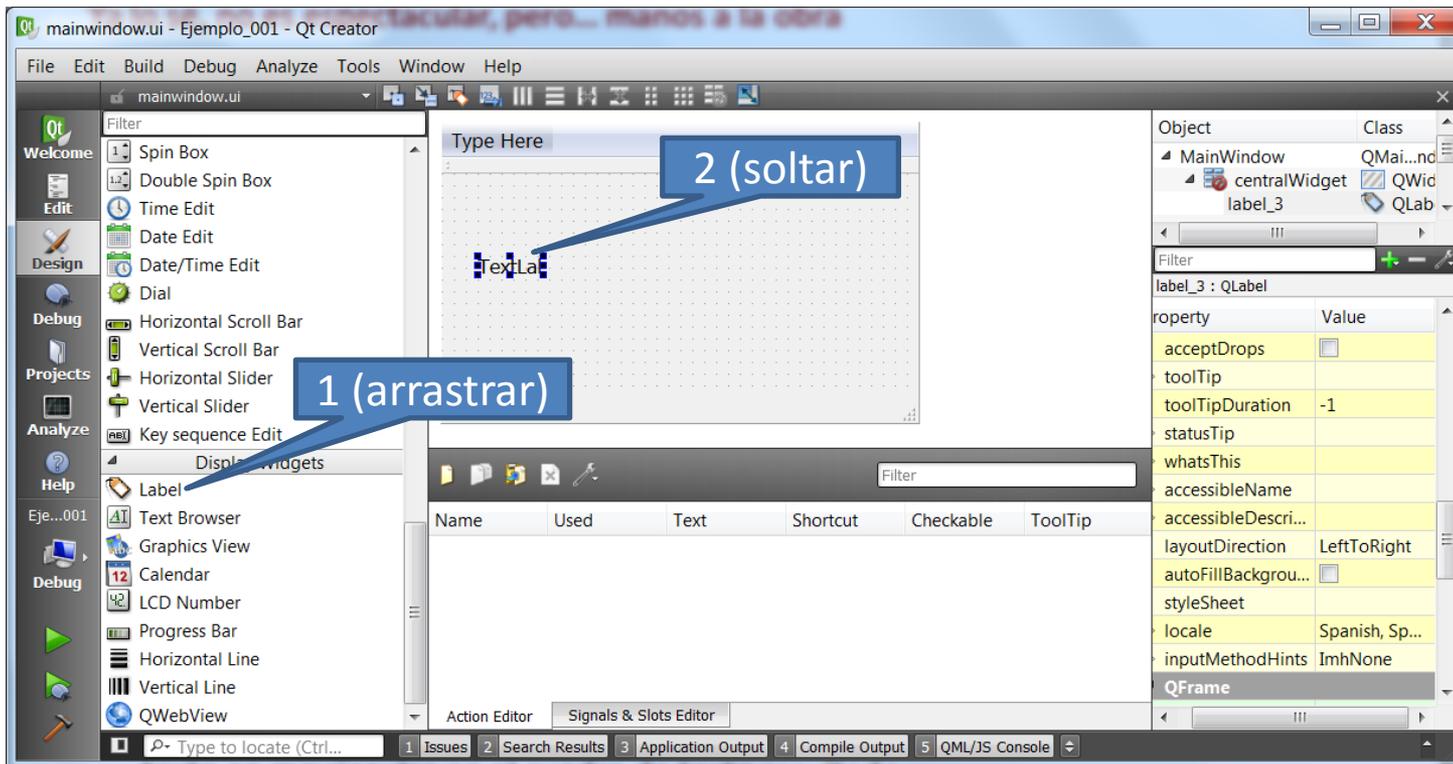
11) En primer lugar, hay que tener claro el aspecto visual que se desea para la aplicación, y qué interacción tendrá con el usuario. Para el ejemplo propuesto (pedir un texto y pasarlo a mayúsculas) nos interesa un aspecto como el siguiente (por ejemplo):



12) Nuestro objetivo: que el usuario pueda teclear el texto deseado en el cuadro de introducción, y que cuando pulse el botón [Pasar a mayúsculas] aparezca el mismo texto en mayúsculas en el cuadro de texto resultado.

Creando el primer programa

- 13) Vamos seleccionando los diferentes elementos en la ventana de Widgets, y arrastrándolos a la posición deseada en la ventana de Formulario. Una vez en la ventana formulario, se pueden mover, cambiar de tamaño, Copiar y Pegar, etc.
- 14) Arrastra el primero (ej. **Label**) y mira la siguiente página antes de hacerlo con los demás.



Creando el primer programa

- 15) Cada vez que añadimos un nuevo elemento, debemos seleccionar sus propiedades en la VENTANA DE PROPIEDADES (recuerda, abajo a la derecha).
- 16) Cada Widget tiene propiedades diferentes. En un Label, **normalmente** sólo nos interesa la propiedad **Text** (el texto que se ve). Selecciónala en la lista, y cambia su contenido (también puedes cambiar el tamaño del Widget para que se vea bien, el tipo de letra, etc.)

The screenshot shows the Qt Creator interface with a design view of a Qt widget. The widget contains a label with the text "Texto:". Three callouts are present:

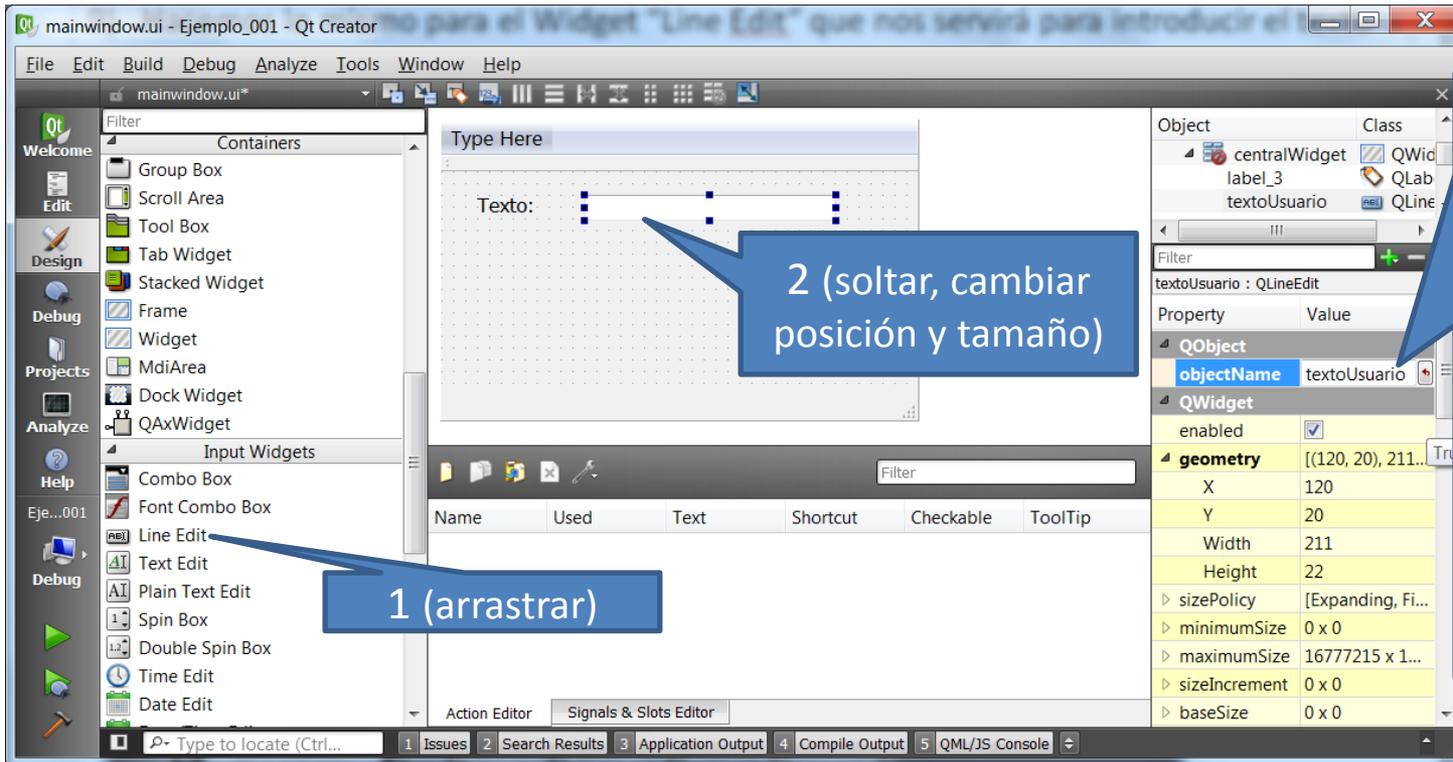
- Callout 1: "1 (seleccionar)" points to the label widget in the design view.
- Callout 2: "2 (cambiar propiedad Text)" points to the "text" property in the Properties window.
- Callout 3: "3 (cambiar posición y tamaño)" points to the label widget in the design view.

The Properties window on the right shows the following properties for the selected label:

property	Value
frameShape	NoFrame
frameShadow	Plain
lineWidth	1
midLineWidth	0
text	Texto
textFormat	AutoText
pixmap	
scaledContents	<input type="checkbox"/>
alignment	AlignLeft, Al...
wordWrap	<input type="checkbox"/>
margin	0
indent	-1

Creando el primer programa

17) Hacemos lo mismo para el Widget “**Line Edit**” que nos servirá para introducir el texto por teclado. En este caso, las propiedades a modificar son **Text** (dejamos vacío) y, sobre todo, **objectName** (siempre es la 1ª propiedad) donde ponemos *textoUsuario* **objectName** es el nombre con el que nos referiremos más adelante a este Widget, así que conviene poner un nombre que nos recuerde “quién es” con facilidad.



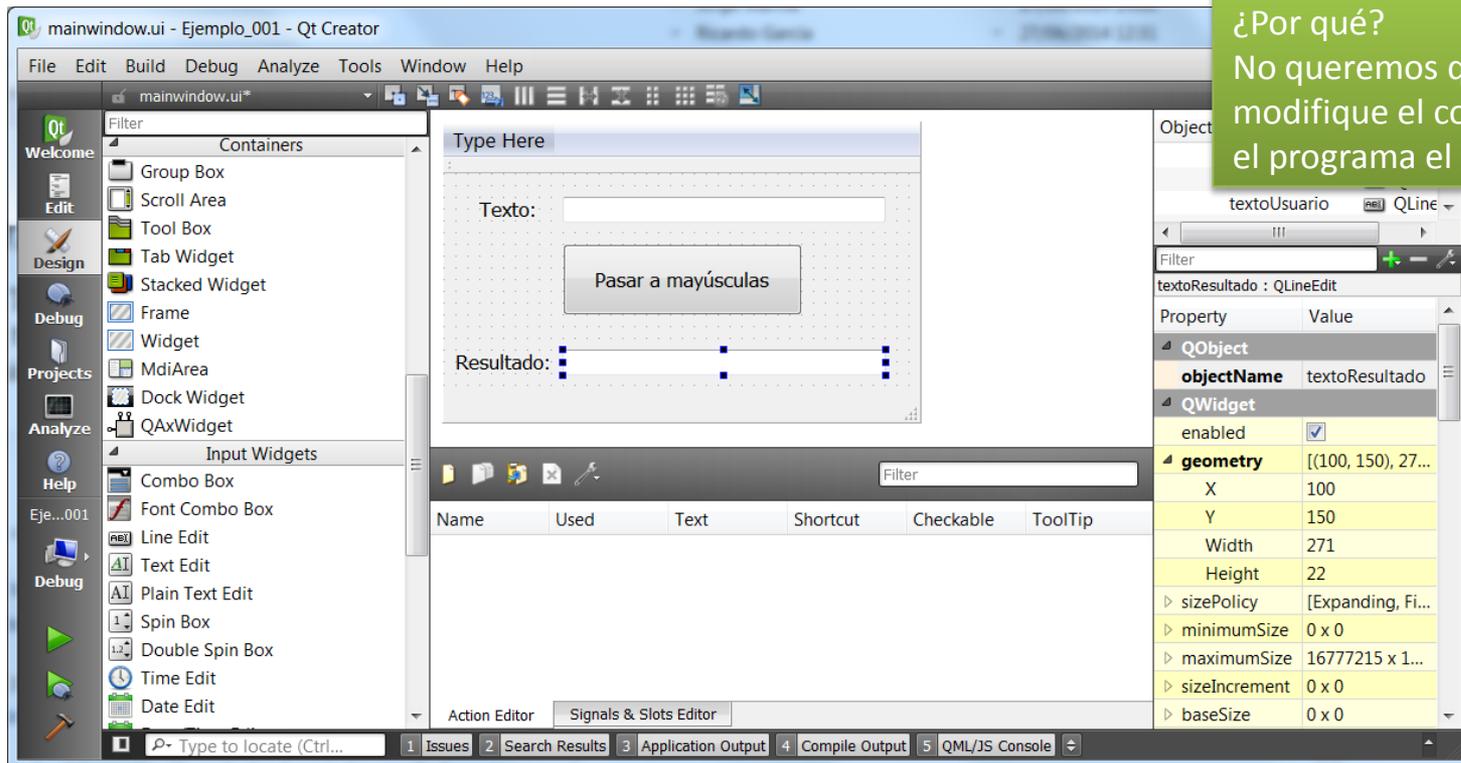
Siempre necesario para Widgets que conlleven acciones, como Line Edit o Push Button
Sólo usar letras, números y _

Creando el primer programa

18) Sigue haciendo lo mismo para añadir:

- ❑ Un **Push Button** (**Text**: Cambio a Mayúsculas, **objectName**: *hacerCambio*)
- ❑ Otro **Label** (**Text**: Resultado, no hace falta **objectName**)
- ❑ Otro **Line Edit** (**Text**: vacío, **objectName**: *textoResultado*, **readOnly**: true)

de forma que el aspecto final sea el siguiente.



Creando el primer programa

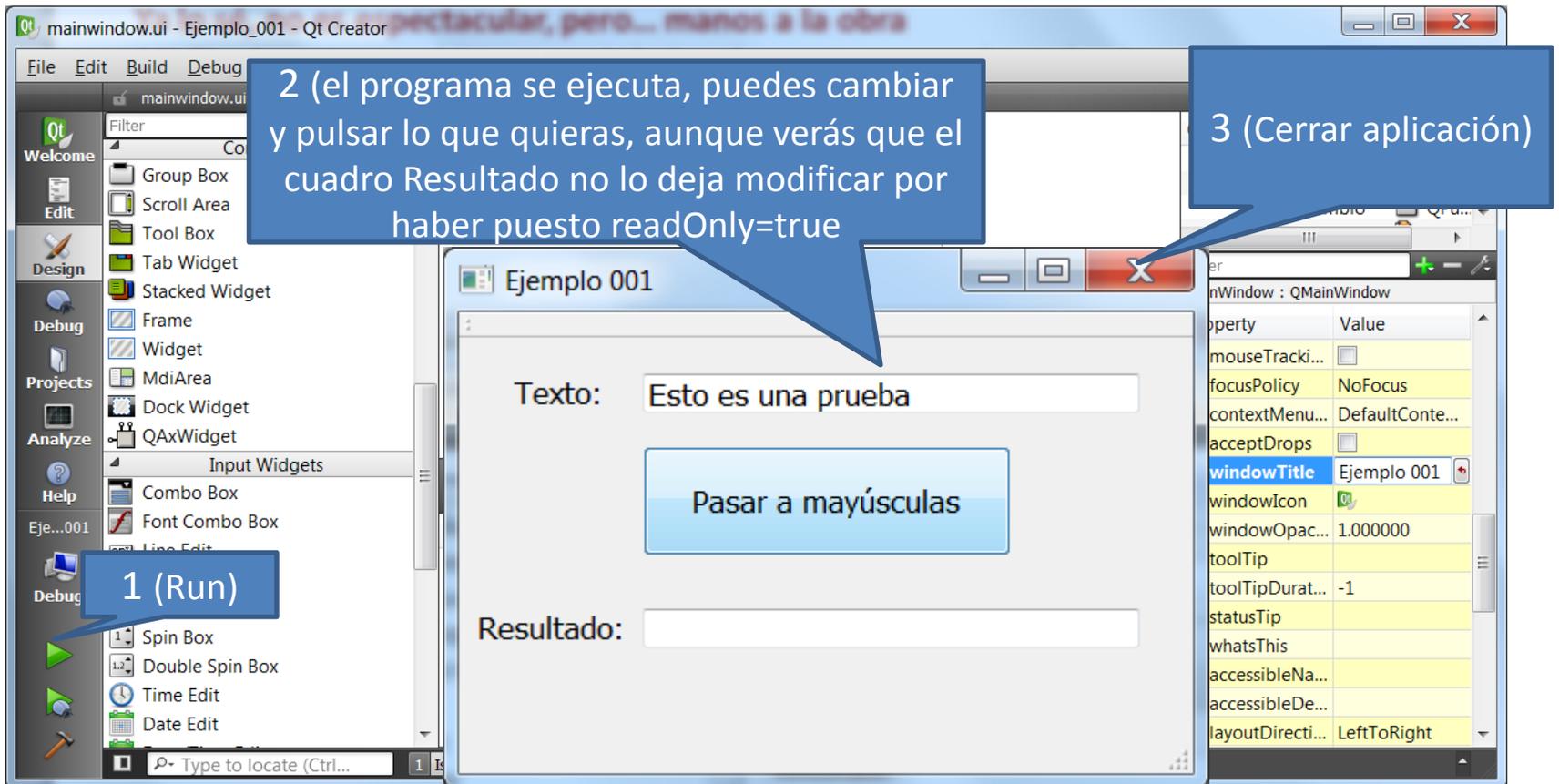
- 19) Finalmente, cambiaremos el título a la ventana, para lo cual seleccionamos el formulario completo (pulsando fuera de todos los Widgets) y cambiamos su propiedad **windowTitle**: Ejemplo 001

The screenshot shows the Qt Creator IDE with a Qt widget form titled "mainwindow.ui - Ejemplo_001". The form contains a text input field labeled "Texto:", a button labeled "Pasar a mayúsculas", and a text output field labeled "Resultado:". A blue callout box labeled "1 (seleccionar formulario)" points to the entire form area. Another blue callout box labeled "2 (cambiar windowTitle)" points to the "windowTitle" property in the Properties panel, which is set to "Ejemplo 001".

Property	Value
mouseTracki...	<input type="checkbox"/>
focusPolicy	NoFocus
contextMenu...	DefaultConte...
acceptDrops	<input type="checkbox"/>
▶ windowTitle	Ejemplo 001
▶ windowIcon	
windowOpac...	1.000000
toolTip	
toolTipDurat...	-1
statusTip	
whatsThis	
▶ accessibleNa...	
▶ accessibleDe...	
layoutDirecti...	LeftToRight

Creando el primer programa

20) Ya está diseñado el aspecto visual del formulario. Podemos probar pulsando de nuevo el botón [Run]  abajo a la izquierda.



1 (Run)

2 (el programa se ejecuta, puedes cambiar y pulsar lo que quieras, aunque verás que el cuadro Resultado no lo deja modificar por haber puesto readOnly=true)

3 (Cerrar aplicación)

Property	Value
mouseTracki...	<input type="checkbox"/>
focusPolicy	NoFocus
contextMenu...	DefaultConte...
acceptDrops	<input type="checkbox"/>
windowTitle	Ejemplo 001
windowIcon	
windowOpac...	1.000000
toolTip	
toolTipDurat...	-1
statusTip	
whatsThis	
accessibleNa...	
accessibleDe...	
layoutDirecti...	LeftToRight

Creando el primer programa

Ya tenemos el aspecto visual que queremos ... pero no hace lo que queremos

21) Sólo hemos creado el interfaz de usuario, ahora tenemos que describir las acciones que queremos que sucedan cuando se accionen los diferentes elementos del interfaz. El entorno de programación ofrece una librería de clases para interactuar con los elementos del interfaz.

22) Lo primero, es enlazar una acción del usuario con una parte de código que se ejecuta cuando la realiza . Esa parte de código se denomina slot.

En nuestro programa, la única acción que debe provocar un cambio es cuando el usuario pulsa el botón *hacerCambio* (siempre nos referiremos a los elementos de interfaz por su **objectName**) en cuyo caso queremos, por este orden:

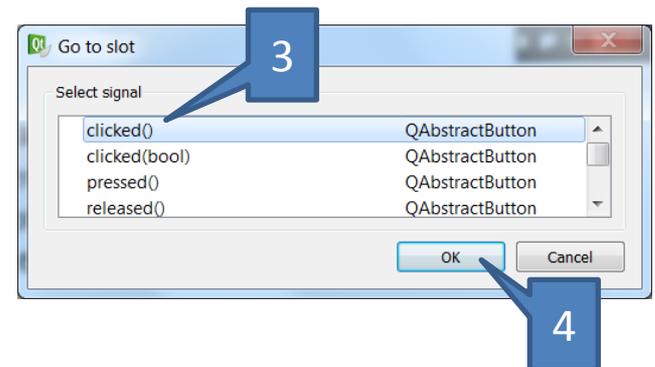
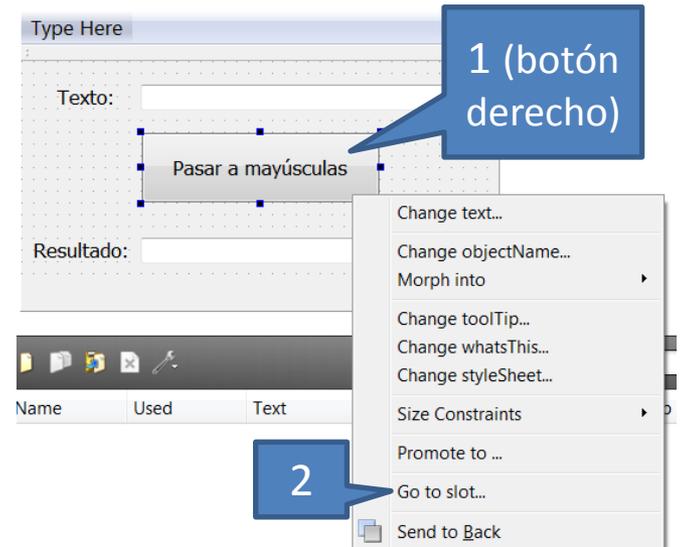
- Obtener el texto que haya en ese momento en el cuadro *textoUsuario*.
- Convertir el texto a mayúsculas.
- Poner el texto en mayúsculas en el cuadro *textoResultado*.

Creando el primer programa

Escribiendo el código

23) Para enlazar la acción (pulsar botón *hacerCambio*) con su código (tomar *textoUsuario*, pasar a mayúsculas y dejar en *textoResultado*), seleccionamos el botón, pulsamos botón derecho, y en el menú que aparece seleccionamos la opción [Go to slot...].

24) Aparece una nueva ventana, en la que nos enumera las diferentes acciones (signals) que puede hacer un botón. La que nos interesa es `clicked()`, o sea, botón pulsado. Escogemos esa señal, y pulsamos [Ok].



Creando el primer programa

25) Se ha producido un importante cambio en la aplicación: desaparecen las ventanas de edición del formulario, y en su lugar volvemos a la ventana de edición de código, pero con una modificación.

```
1 #include "mainwindow.h"
2 #include "ui_mainwindow.h"
3
4 QMainWindow::MainWindow(QWidget *parent) :
5     QMainWindow(parent),
6     ui(new Ui::MainWindow)
7 {
8     ui->setupUi(this);
9 }
10
11 QMainWindow::~MainWindow()
12 {
13     delete ui;
14 }
15
16 void QMainWindow::on_hacerCambio_clicked()
17 {
18 }
19
20
```

Esta función ha sido creada por el IDE. Es un slot (sirve una señal) y es una función miembro de la clase MainWindow, y aquí tendremos que escribir el código que haga lo que pretendemos.

La función pertenece a MainWindow (el formulario inicial)

El nombre que tiene la función indica cuando se ejecuta:
on = cuando
hacerCambio = el nombre del objeto que hemos seleccionado (el botón)
clicked = ha sido pulsado

Creando el primer programa

26) Modificamos el contenido de la función, para que haga lo que pretendemos (el código). Como es la primera vez, vamos paso por paso. Recordamos que nuestro objetivo es:

- Obtener el texto que haya en ese momento en el cuadro *textoUsuario*.
- Convertir el texto a mayúsculas.
- Ponerlo el texto en mayúsculas en el cuadro *textoResultado*.

27) Para obtener el texto del cuadro *textoUsuario* necesitamos una variable que almacene ese contenido. Como el contenido es texto, la variable será de **tipo** QString. Una variable se declara con el formato:

tipo nombre ;

En nuestro caso:

```
void MainWindow::on_hacerCambio_clicked()
{
    QString texto ;
}
```

Se pueden usar los tipos básicos de C (int, float, char, etc.) o clases (propias del usuario o proporcionadas por la librería de desarrollo). La librería proporciona un conjunto de clases muy útil.

Creando el primer programa

28) Lo siguiente es que la nueva variable *texto* tome el contenido del cuadro *textoUsuario*. Pero no podemos hacerlo directamente, porque *textoUsuario* es un objeto que contiene más que un texto (es el cuadro completo, con todas sus características como posición, tipo y tamaño de letra, etc.). Necesitamos usar una función de *textoUsuario* que nos dé sólo el texto (en este caso `text()`).

```
void MainWindow::on_hacerCambio_clicked()
{
    QString texto ;
    texto = ui->textoUsuario->text() ;
}
```

Todos los elementos de interfaz parten de `ui` (user interface).

La flecha `->` indica miembro de la clase (accedido mediante puntero).

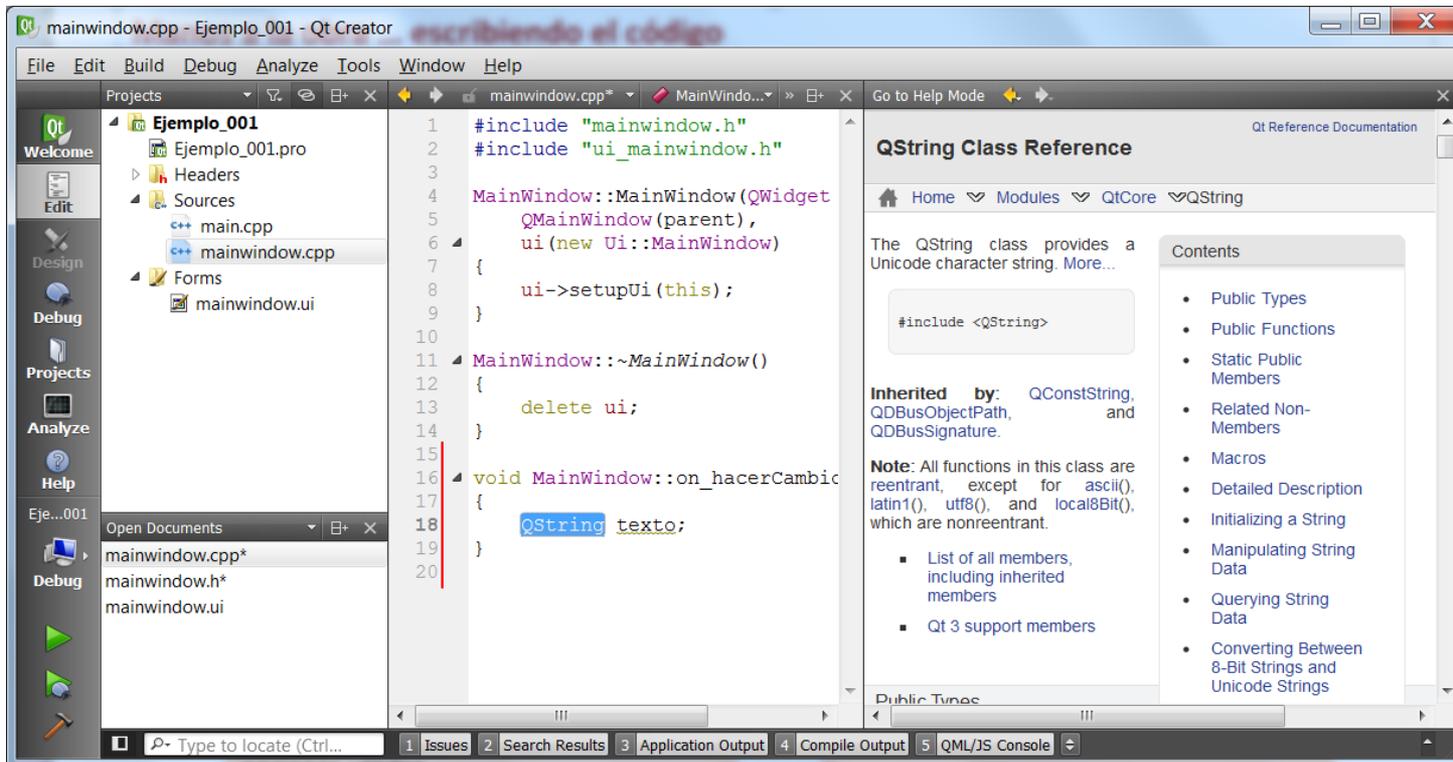
Verás cuando vas tecleando que el programa te ayuda a seleccionar las opciones válidas. Si está bien escrito, cambiará de color.

El objeto `textoUsuario` (que pertenece a `ui`) tiene una función `text()`. Nuevamente se accede a ella con la `->` (si está bien escrito, cambiará de color).

La función `text()` devuelve el contenido en forma de texto (`QString`).

Creando el primer programa

29) A continuación, debemos pasar el contenido de la nueva variable *texto* a mayúsculas. El tipo QString dispone de muchas funcionalidades para hacer cosas habituales con texto, entre ellas pasar a mayúsculas. Puedes ver todo lo que se puede hacer con un QString seleccionando la palabra y pulsando F1 (ayuda):



Creando el primer programa

30) Si navegas por la ayuda, verás que hay una función `QString::toupper()` , junto a un ejemplo de cómo usarla.

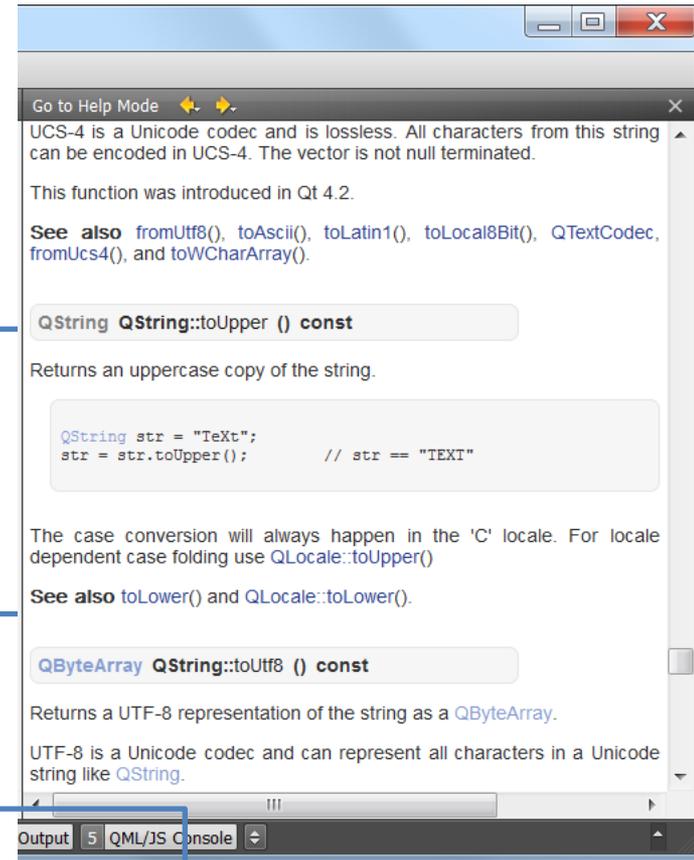
A mayúsculas

Lo añadimos a nuestro código:

```
void MainWindow::on_hacerCambio_clicked()
{
    QString texto ;
    texto = ui->textoUsuario->text() ;
    texto = texto.toUpperCase() ;
}
```

31) Finalmente, sólo nos queda copiar el texto en mayúsculas al objeto *textoResultado*. Esto se hace con la función `setText()` de dicho objeto:

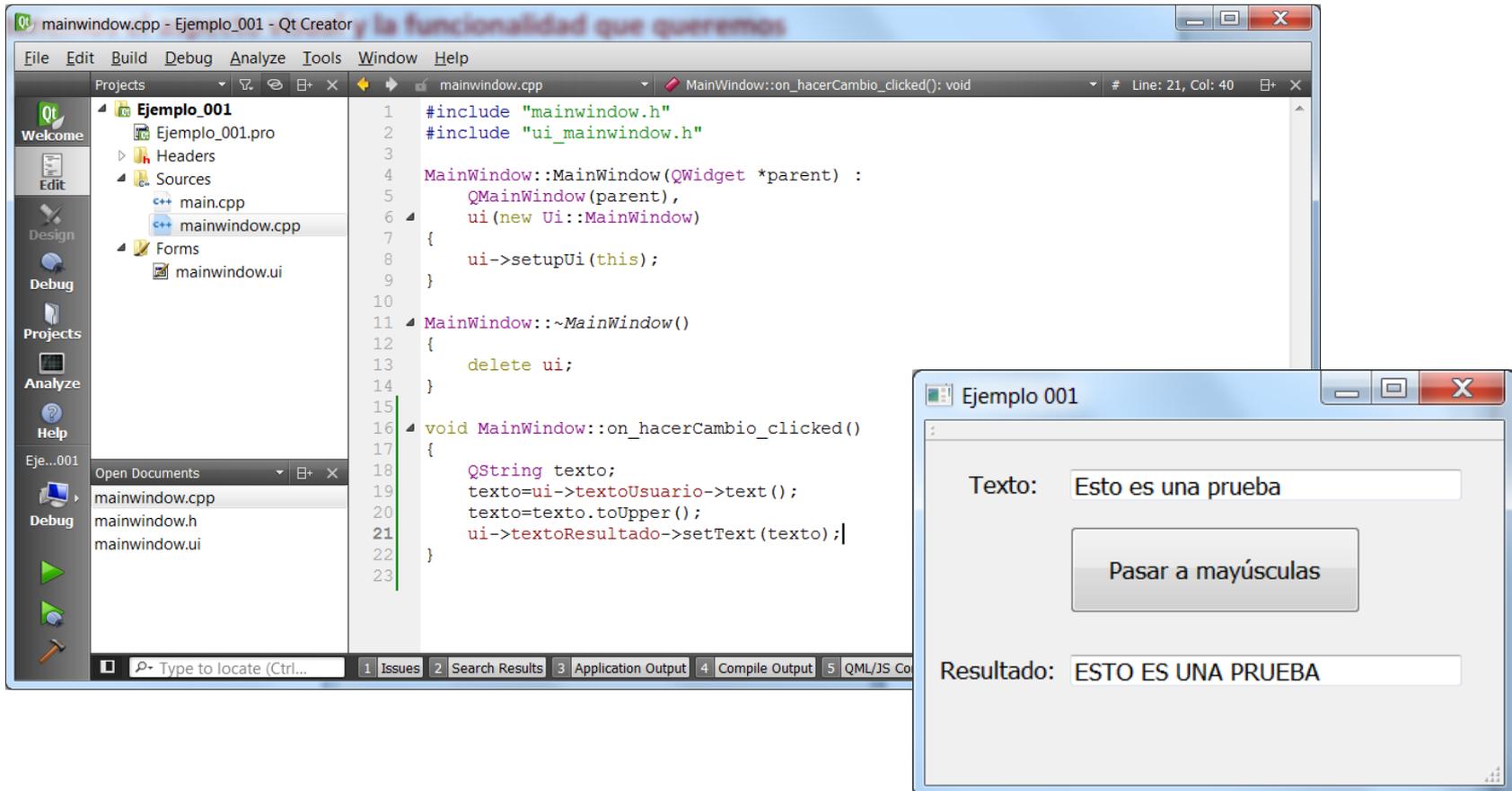
```
void MainWindow::on_hacerCambio_clicked()
{
    QString texto ;
    texto = ui->textoUsuario->text() ;
    texto = texto.toUpperCase() ;
    ui->textoResultado->setText(texto) ;
}
```



Creando el primer programa

Ya tenemos el aspecto visual y la funcionalidad que queremos

32) Podemos probar pulsando de nuevo el botón [Run]  abajo a la izquierda, ahora debería suceder lo que hemos programado.



The image shows the Qt Creator IDE interface. The main editor displays the following C++ code for `mainwindow.cpp`:

```
1 #include "mainwindow.h"
2 #include "ui_mainwindow.h"
3
4 MainWindow::MainWindow(QWidget *parent) :
5     QMainWindow(parent),
6     ui(new Ui::MainWindow)
7 {
8     ui->setupUi(this);
9 }
10
11 MainWindow::~MainWindow()
12 {
13     delete ui;
14 }
15
16 void MainWindow::on_hacerCambio_clicked()
17 {
18     QString texto;
19     texto=ui->textoUsuario->text();
20     texto=texto.toUpper();
21     ui->textoResultado->setText(texto);
22 }
23
```

On the right, a window titled "Ejemplo 001" is shown running the application. It features a text input field with the text "Esto es una prueba", a button labeled "Pasar a mayúsculas", and a text output field displaying "ESTO ES UNA PRUEBA".

Creando el primer programa

Ejercicio propuesto:

Añadir los siguientes elementos de interfaz al formulario:

- Un GroupBox que contenga los siguientes elementos (recordar dar nombre a los objetos y modificar las propiedades que se deseen):
 - Una lista (widget QListWidget)
 - Un checkbox [Añadir a lista]
 - Un nuevo botón [Borrar Lista].

La funcionalidad deseada es:

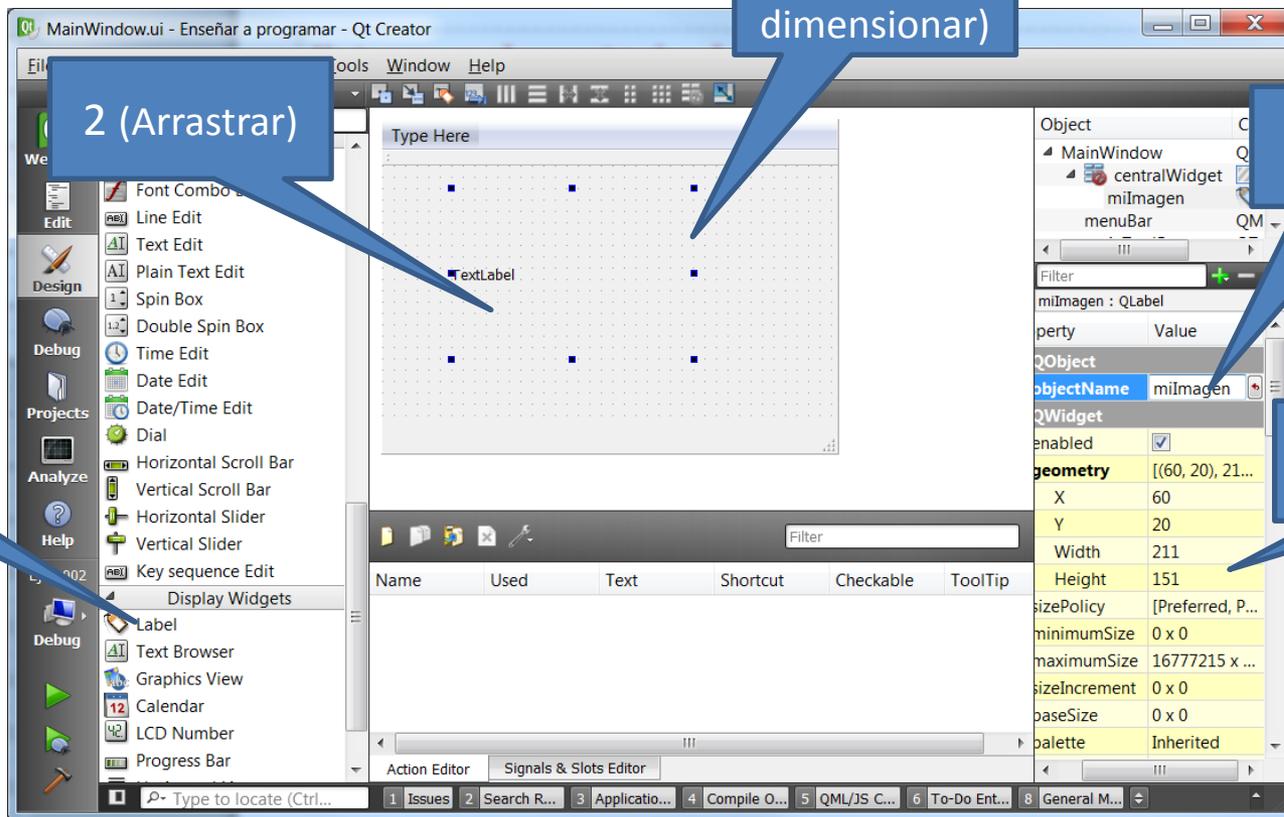
- Cada vez que se pulsa [Pasar a mayúsculas], se comprueba si [Añadir a lista] está activo (función isChecked() de la clase QCheckBox). En caso afirmativo, se añade el texto a la lista (función addItem() del QListWidget).
- Si se pulsa el botón [Borrar Lista], se vacía la misma (función clear() del QListWidget).

Añadir imágenes y gráficos

Los siguientes pasos permiten añadir imágenes y gráficos a nuestra aplicación

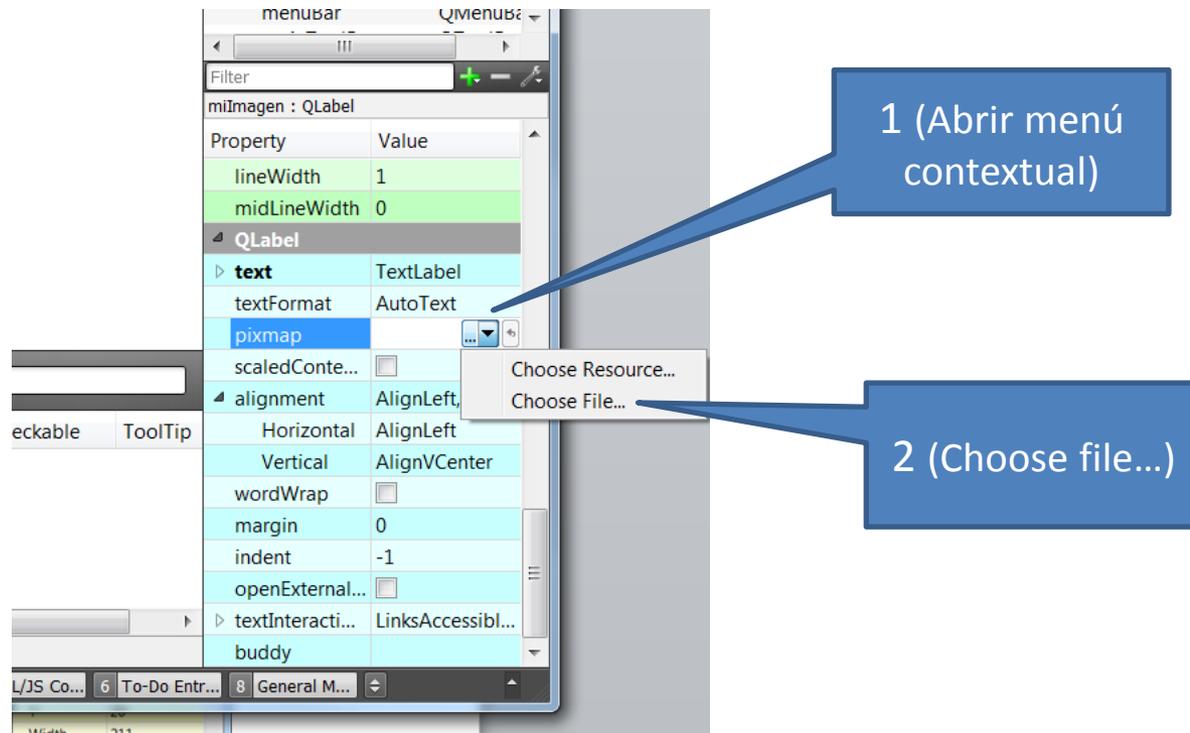
33) Para añadir imágenes y gráficos en nuestra aplicación, lo más sencillo es usar el control QLabel y las clases QPixmap y QImage.

34) Agregar un control tipo Label al formulario, y darle el tamaño, nombre y propiedades deseados.



Añadir imágenes y gráficos

35) Si se desea añadir una imagen disponible en un archivo, seleccionar la propiedad pixmap del objeto QLabel, y elegir el archivo de imagen deseado.



Añadir imágenes y gráficos

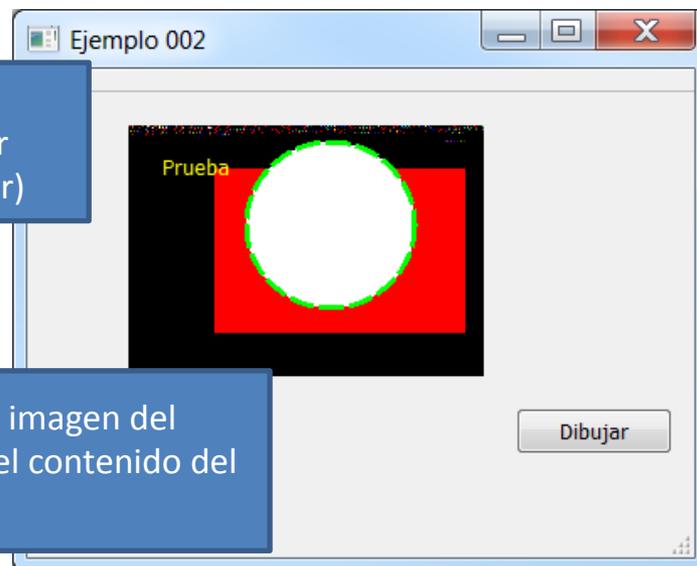
- 36) Si se desea dibujar contenidos gráficos a partir de datos del programa, hay que realizar más pasos y son por medio de programación.
- 37) En 1^{er} lugar, hay que definir en qué momento se desea realizar o modificar el dibujo. Normalmente, será cuando suceda algún evento, que se debe capturar mediante el slot correspondiente.
- 38) En el código de dicho slot, se realizan las operaciones según se indica en el ejemplo siguiente:

```
17
18 void MainWindow::on_drawPushButton_clicked()
19 {
20
21     QPixmap pixmap(ui->miImagen->size());
22     QPainter painter(&pixmap);
23
24     painter.setBrush( Qt::red );
25     painter.drawRect(50, 25, 150, 100);
26
27     QPen pen(Qt::DashLine);
28     pen.setColor(Qt::green);
29     pen.setWidth(3);
30     painter.setPen(pen);
31     painter.setBrush(Qt::white);
32     painter.drawEllipse(QPoint(120,60),50,50);
33
34     pen.setColor(Qt::yellow);
35     painter.setPen(pen);
36     painter.drawText(20,30,"Prueba");
37
38     ui->miImagen->setPixmap(pixmap);
39
40 }
```

- 1) Obtener objeto QPixmap de las dimensiones del destino
- 2) Obtener objeto QPainter para dibujar sobre el pixmap

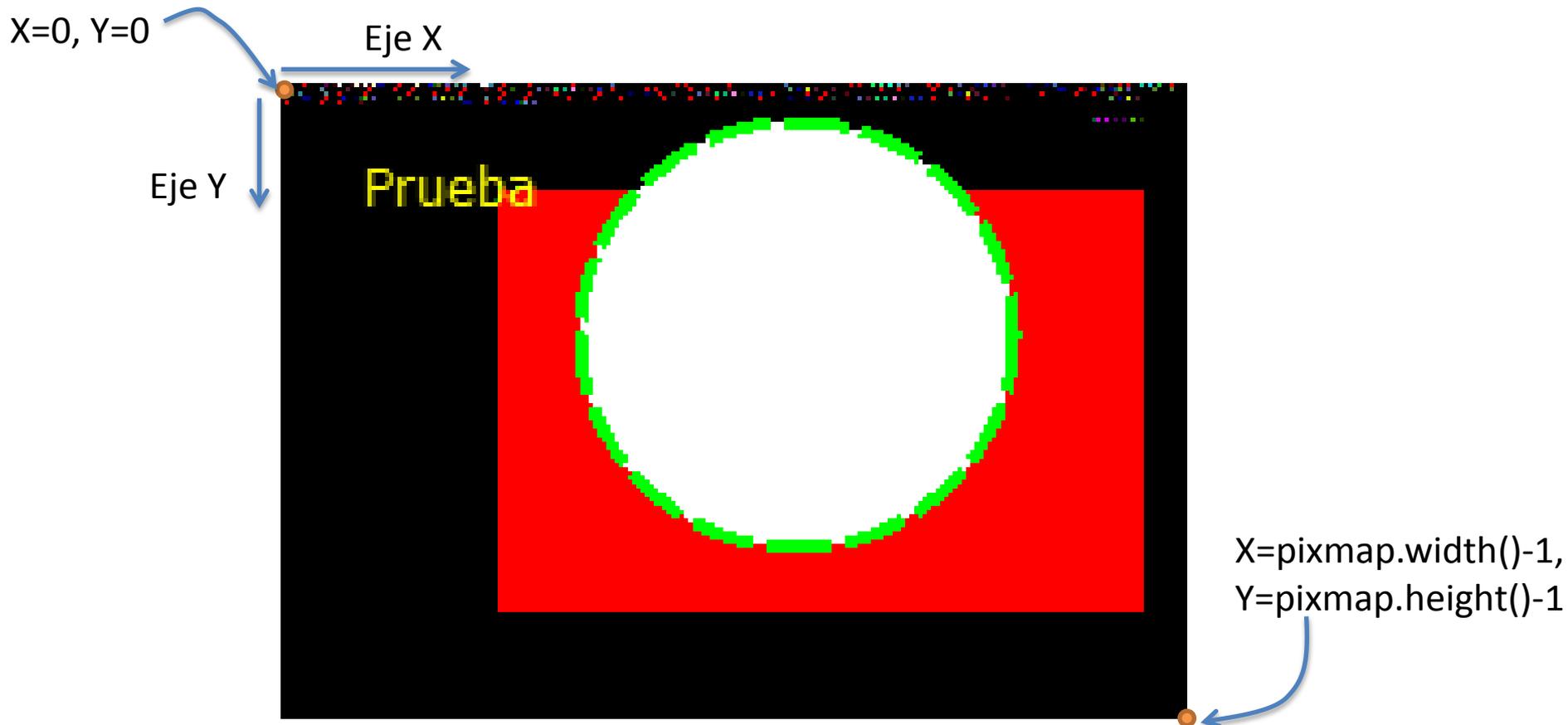
3) Realizar acciones de dibujo sobre el painter (ver ayuda de QPainter)

- 4) Actualizar la imagen del QLabel con el contenido del pixmap



Añadir imágenes y gráficos

39) Hay que tener en cuenta las dimensiones del widget destino (QLabel) y la dirección de los ejes X e Y para dibujar de forma correcta



Añadir otros elementos

40) Existe un buen número de clases que encapsulan elementos de interfaz y auxiliares, simplemente hay que investigar si existe una clase para realizar el trabajo deseado. En todos los casos (y los anteriores) habrá que realizar el `#include <nombre_de_clase>` correspondiente (nombre_de_clase sin extensión .h). Todo ello se indica en la documentación accesible con [F1]. Por ejemplo:

- QTimer: para realizar temporizaciones periódicas mediante slots.
- QDateTime: manejo de fecha y hora.
- QDir, QFile: acceso a directorios y archivos.
- QPoint, QPointF, QLine, QLineF, QRect, QRectF: puntos, vectores y rectángulos.
- QSerialPort: comunicaciones serie.
- QTcpSocket, QTcpServer: comunicaciones por red.

41) En todos los casos, hay que comprobar en el archivo .pro del proyecto que se carga el módulo necesario (se indica en la ayuda).

Si no estaba presente:

- Añadir en el .pro
- Ejecutar Build -> Run qmake

QTcpSocket Class

The QTcpSocket class provides a TCP socket. [More..](#)

Header: `#include <QTcpSocket>`

qmake: `QT += network`

Inherits: [QAbstractSocket](#).

Inherited By: [QSslSocket](#).

Interacción signal/slot

- 42) Cualquier clase derivada (directa o indirectamente) de `QObject` puede manejar el mecanismo signal/slot para interactuar con las clases de usuario:
- ❑ Una señal (signal) es un evento que un objeto genera para ser atendido por otros.
 - ❑ Un receptor (slot) es una función de que se ejecuta cuando sucede el evento.
 - ❑ Es necesario conectar cada señal de interés con su receptor correspondiente.
 - ❑ La clase `QMainWindow` está ejecutando el bucle de espera por evento, para sí misma o para cualquier otro `QObject`. Cuando se produce ese evento, llama al receptor que se ha conectado.
- 43) Ejemplo: `QTimer` → permite generar una señal llamada `timeout()` de forma periódica, útil para temporizaciones. Para usarlo, añadiremos en nuestro código:
- ❑ Un objeto de tipo `QTimer`: `QTimer temporizador;`
 - ❑ Una función receptora en nuestra clase: `void OnTemporizador();`
 - ❑ Una conexión entre la señal de temporizador y nuestra función:
`connect(&temporizador, SIGNAL(timeout()), this, SLOT(OnTemporizador()));`
 - ❑ Una indicación al objeto temporizador para que genere el evento:
`temporizador.start(1000); // Genera el evento timeout() cada 1000 ms`
 - ❑ A partir de ese momento, cada vez que se produzca el evento se ejecutará nuestra función.

Interacción signal/slot

44) Ejemplo: incrementar un contador en 1 cada 2 segundos:

- ❑ Creamos en nuestro interfaz un label para el texto, y un SpinBox para el valor

The screenshot shows the Qt Designer interface for a Qt widget. The central canvas displays a design with a label 'Contador:' and a SpinBox widget. The SpinBox contains the value '0'. The interface is annotated with five numbered callouts:

- 1 (Seleccionar)**: Points to the SpinBox widget in the widget palette on the left.
- 2 (Arrastrar)**: Points to the SpinBox widget being dragged onto the canvas.
- 3 (Mover y dimensionar)**: Points to the SpinBox widget on the canvas, which has a blue selection box around it.
- 4 (Poner nombre)**: Points to the 'contadorSpinBox : QSpinBox' entry in the Object Inspector on the right.
- 5 (Ajustar propiedades)**: Points to the 'readOnly=true' property in the Property Inspector on the right.

The Object Inspector on the right shows the following structure:

Object	Class
MainWindow	QMain...ndow
centralWidget	QWidget
contad...pinBox	QSpinBox
label	QLabel
menuBar	QMenuBar
mainToolBar	QToolBar
statusBar	QStatusBar

The Property Inspector on the right shows the following properties for 'contadorSpinBox : QSpinBox':

Property	Value
QObject	
objectName	contadorSpinBox
QWidget	
enabled	<input checked="" type="checkbox"/>
geometry	[(200, 40), 101 x 31]
X	200
Y	40
Width	101

Interacción signal/slot

44) Ejemplo: incrementar un contador en 1 cada 2 segundos:

- ❑ Añadimos un QTimer y un slot en la declaración de la clase MainWindow (.h), y realizamos las operaciones deseadas en el código de la clase (.cpp)

MainWindow.h

```
...
#include <QTimer> // Hay que incluir la declaración de las clases que usamos
...
class MainWindow : public QMainWindow
{
    Q_OBJECT          // Macro necesaria para clases derivadas de QObject
    ...
private:
    Ui::MainWindow *ui;
    QTimer temporizador; // Las variables que antes se declaraban en main()
                        // ahora se añaden a la clase MainWindow

public slots:
    void OnTemporizador();
};
```

MainWindow.cpp

```
...
MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent), ui(new Ui::MainWindow)
{
    ...
    connect(&temporizador, SIGNAL(timeout()), this, SLOT(OnTemporizador()));
    temporizador.start(2000);
}
void MainWindow::OnTemporizador()
{
    ui->contadorSpinBox->setValue(ui->contadorSpinBox->value()+1);
}
```

this: puntero a este objeto
(MainWindow)

Interacción signal/slot

45) Las señales y slots pueden llevar parámetros, pero no devolver resultado (siempre void). Cuando se indican parámetros en connect, sólo se ponen los tipos, no sus nombres.

46) Ejemplo, clase QFileDialog para seleccionar un archivo en el árbol de directorios:

The QFileDialog class provides a dialog that allow users to select files or directories. [More...](#)

Header: `#include <QFileDialog>`
qmake: `QT += widgets`
Inherits: `QDialog.`

Signals

void `currentChanged`(const QString & path)
void `currentUrlChanged`(const QUrl & url)
void `directoryEntered`(const QString & directory)

Reimplemented Protected Functions

virtual void `accept`()
virtual void `changeEvent`(QEvent * e)
virtual void `done`(int result)

MainWindow.h

```
...  
class MainWindow : public QMainWindow  
{  
    ...  
public slots:  
    void OnFileChanged(QString file);  
};
```

MainWindow.cpp

```
...  
void MainWindow::OnSelectFile ()  
{  
    QFileDialog dlg(this, "Seleccione archivo");  
    connect (&dlg, SIGNAL(currentChanged(QString)),  
            this, SLOT(OnFileChanged(QString)));  
}  
void MainWindow::OnCurrentFileChanged(QString file)  
{  
    if (file.endsWith(".txt"))  
        QMessageBox::information(this, "SELECCION CORRECTA",  
                                  "Ha elegido archivo txt");  
}
```

connect: sólo tipos de los parámetros

Eventos mediante función virtual

47) Algunos eventos son servidos por la clase base. En estos casos, la forma de servirlos en la clase derivada es mediante funciones virtuales.

48) Ejemplo, eventos del ratón en una clase derivada de QWidget (todas las ventanas derivan de QWidget):

QWidget Class

The QWidget class is the base class of all user interface objects.

```
Header: #include <QWidget>
qmake:  QT += widgets
Inherits: QObject and QPaintDevice.
```

...

Protected Functions

...

```
virtual void mouseDoubleClickEvent(QMouseEvent * event)
virtual void mouseMoveEvent(QMouseEvent * event)
virtual void mousePressEvent(QMouseEvent * event)
virtual void mouseReleaseEvent(QMouseEvent * event)
```

MainWindow.h

```
...
class MainWindow : public QMainWindow
{
    ...
protected:
    virtual void mousePressEvent(QMouseEvent *event);
};
```

MainWindow.cpp

```
...
void MainWindow::mousePressEvent(QMouseEvent *event)
{
    ... Usar datos de event ...

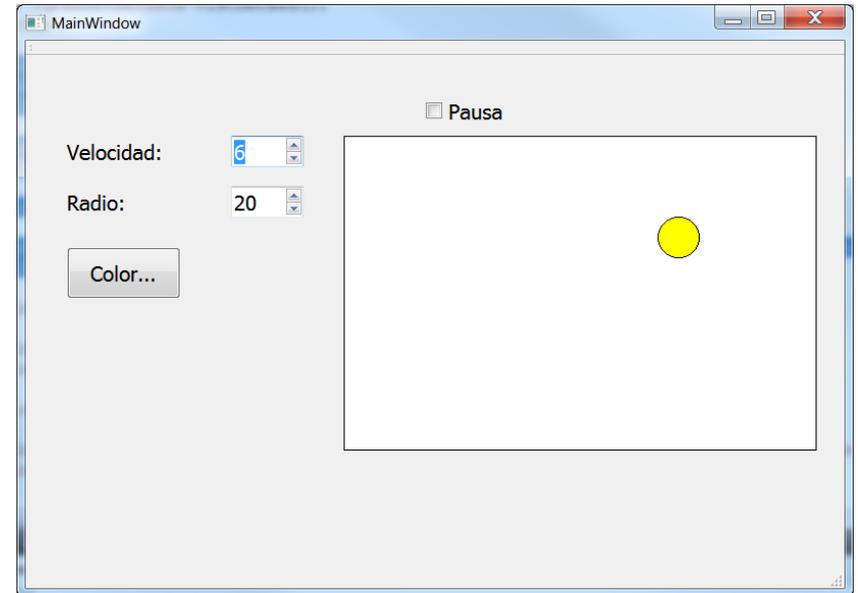
    // Opcional : pasarle el evento a la clase base
    QMainWindow::mousePressEvent(event);
}
```

Ejercicio final

Ejercicio propuesto:

Realizar un programa basado en Qt que dibuje un círculo moviéndose por un espacio rectangular, con las siguientes condiciones:

- Se realizará un desplazamiento cada 100 ms.
- El círculo rebota contra las paredes al llegar a los límites del espacio.
- Un SpinBox permite seleccionar el radio del círculo (en pixels)
- Un SpinBox permite seleccionar la velocidad del círculo (nº de pixels de desplazamiento por cada temporización).
- Un botón permite seleccionar el color de fondo del círculo (QColorDialog).
- Un checkbox permite pausar/reanudar el movimiento.
- Si se pulsa el botón izquierdo del ratón en el espacio de dibujo, la dirección del movimiento del círculo se dirige hacia el punto pulsado.



Ejercicio final

Ejercicio propuesto:

A tener en cuenta:

- ❑ Situar variables necesarias de forma permanente en la clase MainWindow.
- ❑ Recordar #include de todas las clases que se van añadiendo.
- ❑ Clase círculo: posición XY, vector desplazamiento XY, radio, color fondo.
- ❑ Rebote contra un borde vertical de la ventana ui->qDrawLabel (similar para bordes horizontales):

```
circulo.x += circulo.vx;
if (circulo.x - circulo.radio <= 0)
{
    circulo.x = -circulo.x + 2*circulo.radio;
    circulo.vx = -circulo.vx;
}
if (circulo.x+circulo.radio>=ui->qDrawLabel->width())
{
    circulo.x = ui->qDrawLabel->width() - (circulo.x - ui->qDrawLabel->width())-2*circulo.radio;
    circulo.vx=-circulo.vx;
}
```

- ❑ Posición del ratón relativa a la ventana ui->qDrawLabel :

```
void MainWindow::mousePressEvent(QMouseEvent *event)
{
    QPoint relPos=event->pos() - ui->centralWidget->pos() - ui->qDrawLabel->pos();
    ...
}
```