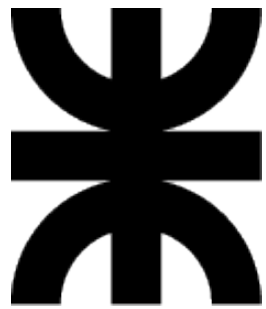
Informática II

Programación gráfica con Qt
Signals and slots

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2024 –

Signals and slots

- La mayoría de las bibliotecas de GUI tienen un mecanismo para detectar una acción del usuario y responder en consecuencia.

Signals and slots

- La mayoría de las bibliotecas de GUI tienen un mecanismo para detectar una acción del usuario y responder en consecuencia.
- Algunas usan *callbacks*, otras usan *listeners*, pero básicamente todas están inspiradas en el patrón *Observer*.

Signals and slots

- La mayoría de las bibliotecas de GUI tienen un mecanismo para detectar una acción del usuario y responder en consecuencia.
- Algunas usan *callbacks*, otras usan *listeners*, pero básicamente todas están inspiradas en el patrón *Observer*.

Observer (patrón de diseño)

Se utiliza cuando un objeto observable desea notificar a otros objetos observadores sobre un cambio de estado. Por ejemplo:

- Un usuario ha hecho click en un botón y debería aparecer un menú.
- Una página web termina de cargarse y un proceso debe extraer información de esta página cargada.
- Un usuario se desplaza (scroll) por una lista de elementos (p.e. en una tienda de aplicaciones), llegó al final y se deben cargar otros elementos.

Signals and slots

- La mayoría de las bibliotecas de GUI tienen un mecanismo para detectar una acción del usuario y responder en consecuencia.
- Algunas usan *callbacks*, otras usan *listeners*, pero básicamente todas están inspiradas en el patrón *Observer*.

Observer (patrón de diseño)

Se utiliza cuando un objeto observable desea notificar a otros objetos observadores sobre un cambio de estado. Por ejemplo:

- Un usuario ha hecho click en un botón y debería aparecer un menú.
- Una página web termina de cargarse y un proceso debe extraer información de esta página cargada.
- Un usuario se desplaza (scroll) por una lista de elementos (p.e. en una tienda de aplicaciones), llegó al final y se deben cargar otros elementos.

- Este patrón a menudo conduce a código repetitivo.

Signals and slots

- La mayoría de las bibliotecas de GUI tienen un mecanismo para detectar una acción del usuario y responder en consecuencia.
- Algunas usan *callbacks*, otras usan *listeners*, pero básicamente todas están inspiradas en el patrón *Observer*.

Observer (patrón de diseño)

Se utiliza cuando un objeto observable desea notificar a otros objetos observadores sobre un cambio de estado. Por ejemplo:

- Un usuario ha hecho click en un botón y debería aparecer un menú.
- Una página web termina de cargarse y un proceso debe extraer información de esta página cargada.
- Un usuario se desplaza (scroll) por una lista de elementos (p.e. en una tienda de aplicaciones), llegó al final y se deben cargar otros elementos.

- Este patrón a menudo conduce a código repetitivo.
- El mecanismo elegido por Qt elimina dicho código repetitivo y brinda una sintaxis agradable y limpia

Signals and slots

- La mayoría de las bibliotecas de GUI tienen un mecanismo para detectar una acción del usuario y responder en consecuencia.
- Algunas usan *callbacks*, otras usan *listeners*, pero básicamente todas están inspiradas en el patrón *Observer*.

Observer (patrón de diseño)

Se utiliza cuando un objeto observable desea notificar a otros objetos observadores sobre un cambio de estado. Por ejemplo:

- Un usuario ha hecho click en un botón y debería aparecer un menú.
- Una página web termina de cargarse y un proceso debe extraer información de esta página cargada.
- Un usuario se desplaza (scroll) por una lista de elementos (p.e. en una tienda de aplicaciones), llegó al final y se deben cargar otros elementos.

- Este patrón a menudo conduce a código repetitivo.
- El mecanismo elegido por Qt elimina dicho código repetitivo y brinda una sintaxis agradable y limpia → **signals and slots**.

Signals and slots

En lugar de tener objetos observadores y observables y tener que registrarlos, Qt proporciona dos conceptos de alto nivel: signals and slots.

Signals and slots

En lugar de tener objetos observadores y observables y tener que registrarlos, Qt proporciona dos conceptos de alto nivel: signals and slots.

- Una señal (signal) es un mensaje que un objeto puede enviar, la mayoría de las veces para informar de un cambio de estado.

Signals and slots

En lugar de tener objetos observadores y observables y tener que registrarlos, Qt proporciona dos conceptos de alto nivel: signals and slots.

- Una señal (signal) es un mensaje que un objeto puede enviar, la mayoría de las veces para informar de un cambio de estado.
- Una ranura (slot) es una función que se utiliza para aceptar y responder a una señal.

Signals and slots

En lugar de tener objetos observadores y observables y tener que registrarlos, Qt proporciona dos conceptos de alto nivel: signals and slots.

- Una señal (signal) es un mensaje que un objeto puede enviar, la mayoría de las veces para informar de un cambio de estado.
- Una ranura (slot) es una función que se utiliza para aceptar y responder a una señal.

Algunos ejemplos de señales de la clase `QPushButton`:

- `clicked`
- `pressed`
- `released`

Signals and slots

En lugar de tener objetos observadores y observables y tener que registrarlos, Qt proporciona dos conceptos de alto nivel: signals and slots.

- Una señal (signal) es un mensaje que un objeto puede enviar, la mayoría de las veces para informar de un cambio de estado.
- Una ranura (slot) es una función que se utiliza para aceptar y responder a una señal.

Algunos ejemplos de señales de la clase `QPushButton`:

- `clicked`
- `pressed`
- `released`

Algunas slots (de diferentes clases):

- `QApplication::quit`
- `QWidget::setEnabled`
- `QPushButton::setText`

Signals and slots

En lugar de tener objetos observadores y observables y tener que registrarlos, Qt proporciona dos conceptos de alto nivel: signals and slots.

- Una señal (signal) es un mensaje que un objeto puede enviar, la mayoría de las veces para informar de un cambio de estado.
- Una ranura (slot) es una función que se utiliza para aceptar y responder a una señal.

Algunos ejemplos de señales de la clase `QPushButton`:

- `clicked`
- `pressed`
- `released`

Algunas slots (de diferentes clases):

- `QApplication::quit`
- `QWidget::setEnabled`
- `QPushButton::setText`

Por ejemplo, se podría *conectar* la señal `QPushButton::clicked` con la ranura `QApplication::quit`

Signals and slots

- Para poder responder a una señal es necesario conectar un ranura a dicha señal.

Signals and slots

- Para poder responder a una señal es necesario conectar un ranura a dicha señal.
- Qt proporciona el método `QObject::connect` (método estático de la clase `QObject`).

(Documentación de `QObject::connect`)

Signals and slots

- Para poder responder a una señal es necesario conectar un ranura a dicha señal.
- Qt proporciona el método `QObject::connect` (método estático de la clase `QObject`).

(Documentación de `QObject::connect`)

Observación: Básicamente, las señales y las ranuras son métodos que pueden tener argumentos o no, pero que nunca devuelven nada. Si bien la noción de una señal como método es inusual, una ranura es en realidad un método real y se puede llamar como otros métodos o mientras se responde a una señal.

Signals and slots – Ejemplo `QuitButton`

1. Abrir Qt Creator y crear un aplicación (`QuitButton`) que herede heredada de `QWidget` (sin archivo de formulario).
2. Eliminar el archivo cabecera de la clase de ventana.
3. Modificar el archivo `main.cpp` y copiar el ejemplo `ButtonInWidget`.

Signals and slots – Ejemplo `QuitButton`

1. Abrir Qt Creator y crear un aplicación (`QuitButton`) que herede heredada de `QWidget` (sin archivo de formulario).
2. Eliminar el archivo cabecera de la clase de ventana.
3. Modificar el archivo `main.cpp` y copiar el ejemplo `ButtonInWidget`.
4. Agregar la configuración de **signals and slots**:

```
QObject::connect(quitBtn, &QPushButton::clicked, qApp, &QApplication::quit)
```

Signals and slots – Ejemplo `QuitButton`

1. Abrir Qt Creator y crear un aplicación (`QuitButton`) que herede heredada de `QWidget` (sin archivo de formulario).
2. Eliminar el archivo cabecera de la clase de ventana.
3. Modificar el archivo `main.cpp` y copiar el ejemplo `ButtonInWidget`.
4. Agregar la configuración de **signals and slots**:

```
QObject::connect(quitBtn, &QPushButton::clicked, qApp, &QApplication::quit)
```

   donde:

   - `qApp` es un puntero global al objeto de la aplicación, está definido en el header `QApplication`
   - Otra manera es llamando a `QApplication::instance()`

Signals and slots – Ejemplo `QuitButton`

1. Abrir Qt Creator y crear un aplicación (`QuitButton`) que herede heredada de `QWidget` (sin archivo de formulario).
2. Eliminar el archivo cabecera de la clase de ventana.
3. Modificar el archivo `main.cpp` y copiar el ejemplo `ButtonInWidget`.
4. Agregar la configuración de **signals and slots**:

   ```
   QObject::connect(quitBtn, &QPushButton::clicked, qApp, &QApplication::quit)
   ```

   donde:
   - `qApp` es un puntero global al objeto de la aplicación, está definido en el header `QApplication`
   - Otra manera es llamando a `QApplication::instance()`
5. Luego convertir la aplicación a clase.
6. Separar la clase de la ventana en un archivo `.h` y `.cpp` separados

Signals and slots – Ejemplo `QuitButton`

Aplicación sin clase de ventana (archivo `main.cpp`)

```
1  #include <QApplication>
2  #include <QWidget>
3  #include <QPushButton>
4
5  int main(int argc, char *argv[])
6  {
7      QApplication app(argc, argv);
8
9      QWidget window;
10     window.setWindowTitle("Quit Button");
11     QPushButton *quitBtn = new QPushButton("Salir", &window);
12     quitBtn->setGeometry(50, 40, 75, 30);
13
14     // Conectar señal con ranura
15     QObject::connect(
16         quitBtn, &QPushButton::clicked, qApp, &QApplication::quit);
17
18     window.show();
19     return app.exec();
20 }
```

Signals and slots – Ejemplo `QuitButton`

Aplicación con clase de ventana (archivo `main.cpp`)

```
1  #include <QApplication>
2  #include <QWidget>
3  #include <QPushButton>
4
5  class QuitButton : public QWidget {
6    public:
7      QuitButton(QWidget *parent = nullptr);
8  };
9
10 QuitButton::QuitButton(QWidget *parent) : QWidget(parent) {
11   QPushButton *quitBtn = new QPushButton("Salir", this);
12   quitBtn->setGeometry(50, 40, 75, 30);
13 }
14
15 int main(int argc, char *argv[]) {
16   QApplication app(argc, argv);
17   QuitButton window;
18   window.setWindowTitle("Quit Button");
19   window.show();
20   return app.exec();
21 }
```

Signals and slots – Ejemplo `PlusMinus`

1. Crear una aplicación de nombre `PlusMinus` que herede de `QWidget`.
2. Fijar `PlusMinus` como nombre de la clase e indicar que no incluya formulario.
3. La definición de la clase en el archivo `plusminus.h` es

```
class PlusMinus : public QWidget {
  Q_OBJECT

public:
  PlusMinus(QWidget *parent = nullptr);
  ~PlusMinus();

private:
  QLabel *lbl;

private slots:
  void OnPlus();
  void OnMinus();
};
```

Signals and slots – Ejemplo `PlusMinus`

4. El constructor de la clase es:

```
PlusMinus::PlusMinus(QWidget *parent) : QWidget(parent)
{
  QPushButton *plsBtn = new QPushButton("+", this);
  QPushButton *minBtn = new QPushButton("-", this);
  lbl = new QLabel("0", this);

  QGridLayout *grid = new QGridLayout(this);
  grid->addWidget(plsBtn, 0, 0);
  grid->addWidget(minBtn, 0, 1);
  grid->addWidget(lbl, 1, 1);
  setLayout(grid);

  connect(plsBtn, &QPushButton::clicked, this, &PlusMinus::OnPlus);
  connect(minBtn, &QPushButton::clicked, this, &PlusMinus::OnMinus);
}
```

Signals and slots – Ejemplo `PlusMinus`

5. Definir los dos **slots**:

```
void PlusMinus::OnPlus()
{
  int val = lbl->text().toInt();
  val++;
  lbl->setText(QString::number(val));
}

void PlusMinus::OnMinus()
{
  int val = lbl->text().toInt();
  val--;
  lbl->setText(QString::number(val));
}
```

Signals and slots – Ejemplo `PlusMinus`

6. Y la función `main`:

```
int main(int argc, char *argv[])
{
  QApplication app(argc, argv);

  PlusMinus window;
  window.resize(300, 100);
  window.setWindowTitle("Plus minus");
  window.show();

  return app.exec();
}
```

Signals and slots desde Qt Designer

1. Crear un nuevo proyecto como subclase heredada de `QWidget`

Signals and slots desde Qt Designer

1. Crear un nuevo proyecto como subclase heredada de `QWidget`
2. Editar el Form desde Qt Designer haciendo doble-click al archivo `mainwindow.ui`

Signals and slots desde Qt Designer

1. Crear un nuevo proyecto como subclase heredada de `QWidget`
2. Editar el Form desde Qt Designer haciendo doble-click al archivo `mainwindow.ui`
3. Agregar los widgets **Horizontal Slider** y un **Progress Bar** en la ventana

Signals and slots desde Qt Designer

1. Crear un nuevo proyecto como subclase heredada de `QWidget`
2. Editar el Form desde Qt Designer haciendo doble-click al archivo `mainwindow.ui`
3. Agregar los widgets **Horizontal Slider** y un **Progress Bar** en la ventana
4. Abrir el editor de signals&slots (botón a la derecha de la lista de selección de formularios)

Signals and slots desde Qt Designer

1. Crear un nuevo proyecto como subclase heredada de `QWidget`
2. Editar el Form desde Qt Designer haciendo doble-click al archivo `mainwindow.ui`
3. Agregar los widgets **Horizontal Slider** y un **Progress Bar** en la ventana
4. Abrir el editor de signals&slots (botón a la derecha de la lista de selección de formularios)
5. Realizar la conexión (arrastrar y soltar) desde el **Horizontal Slider** al **Progress Bar**

Signals and slots desde Qt Designer

1. Crear un nuevo proyecto como subclase heredada de `QWidget`
2. Editar el Form desde Qt Designer haciendo doble-click al archivo `mainwindow.ui`
3. Agregar los widgets **Horizontal Slider** y un **Progress Bar** en la ventana
4. Abrir el editor de signals&slots (botón a la derecha de la lista de selección de formularios)
5. Realizar la conexión (arrastrar y soltar) desde el **Horizontal Slider** al **Progress Bar**
6. Seleccionar la señal `valueChanged(int)` del **Horizontal Slider** y conectarlo al slot `setValue(int)` del **Progress Bar**

Signals and slots desde Qt Designer

1. Crear un nuevo proyecto como subclase heredada de `QWidget`
2. Editar el Form desde Qt Designer haciendo doble-click al archivo `mainwindow.ui`
3. Agregar los widgets **Horizontal Slider** y un **Progress Bar** en la ventana
4. Abrir el editor de signals&slots (botón a la derecha de la lista de selección de formularios)
5. Realizar la conexión (arrastrar y soltar) desde el **Horizontal Slider** al **Progress Bar**
6. Seleccionar la señal `valueChanged(int)` del **Horizontal Slider** y conectarlo al slot `setValue(int)` del **Progress Bar**
7. La misma conexión se puede realizar desde el código fuente.
   - borrar la conexión desde el IDE de Qt Designer y
   - agregar el siguiente código en el constructor de la clase `MainWindow`

```
QObject::connect(ui->horizontalSlider, &QSlider::valueChanged,
                 ui->progressBar, &QProgressBar::setValue);
```