ECE 3574: Applied Software Design

Signals and Slots

Today we will learn about a variation of the Observer design pattern that is used prominently within Qt, called signals and slots.

- Observer and Publish/Subscribe Pattern
- Observers as callback functions
- Observers using signals
- Qt signals
- Examples
- Exercise

The *Observer* or *Publish / Subscribe* design pattern is a way to communicate among objects without them knowing much about one another.

Recall the notion of an event handler.

- To call the event handler we need a pointer or reference to the object handling the event
- This is an example of a callback function

A callback is simply a pointer to a function. This should be familiar from project 1.

Example 1: a simple callback function

See `callbacks.cpp`

Example 2: using a member function as a callback

See `callbacks_methods.cpp`

There are drawbacks to callbacks as illustrated in Example 1 and 2.

- They represent a one-to-one communication
- The communication is always-on

Fixing this requires a good deal of effort to manage the callback connections.

- make the callback a list of callbacks
- call each callback in the list

Factoring this code out into a library results in managed callbacks, or *signals* and *slots*.

Signals and Slots

- *Signals* (publishers) are callbacks with multiple targets or *slots* (receivers or subscribers).
- Signals are *connected* to slots
- Signals are *emitted*
- Slots connected to a signal are called when the signal is emitted

This raises an important issue, how are return values from slots used?

- Some systems do not use them (Qt)
- Other systems provide a way to aggregate them (boost::signals)

C++ libraries that provide a signal/slot mechanism

- Boost is a very popular collection of C++ library that provides boost::signal.
- POCO is another popular collection that provides an event system that works like signals/slots.
- Qt has a signals and slots mechanism implemented as an extension of C++.

Qt signals and slots extend the syntax of C++.

- Every class that wants to communicate via signals and slots must derive from QObject directly or indirectly (derive from a subclass of QObject)
- The class should have the macro `Q_OBJECT` in its private section.
- slots are defined in a private, protected, or public section called slots and implemented
- signals are defined in a section called signals, but **not** implemented
- signals are emitted using the keyword `emit`
- connections are made using the `QObject::connect` function.

The connections between signals and slots can be synchronous or queued.

An Example: a settings widget

See `qtmain.cpp`. `receiver_object.*`, `settings_widget.*`, and `settings.h`.

Exercise

See website

Next Actions and Reminders

- Read about Models and Views in Qt
- Project 1 final is due tomorrow, Friday March 3rd at 5pm.

Be **sure** to commit the changes you want to have graded, tag and push before 5pm.

```
git push origin final
```