Qt basics

Tarmo Aidantausta, Digia Plc

Qt basics: About me

- I'm a Nokia Certified Qt Developer.
- I have around year of experience with it.
- I can read, copy, paste and make lists.

Qt basics: Overview

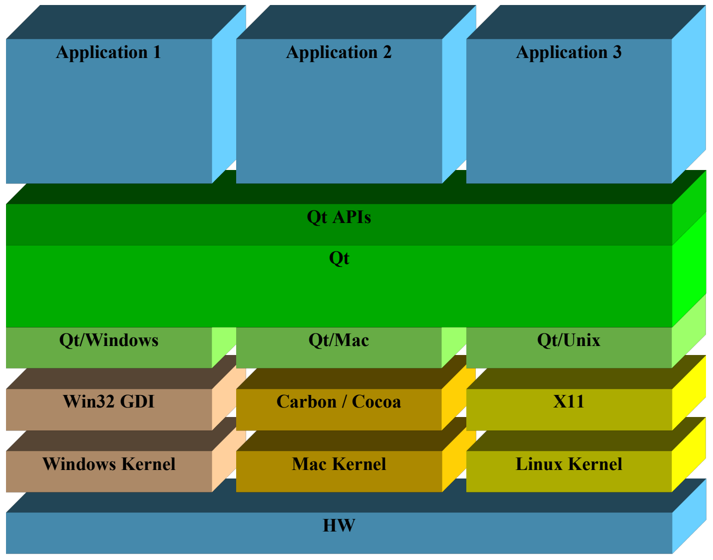
- Presentation covers only basics of Qt and simple component and graphics programming.
- What is Qt?
- What can you do with it?
  - Can I do console applications?
  - Can I do graphical user interfaces?
- Why would I want to use it?
- What do I need to know to use it?

Qt basics: Overview

- What are the fundamentals of Qt?
- How do I do GUI applications with it?
  - How do I do Hello World!?
  - How do I do events?
  - How do I do components?
- Most of the information has been gathered and some even just quoted as is from http://qt.nokia.com/doc/ .

What is Qt?

- Qt is a cross-platform application and UI framework.
  - Using Qt, you can write applications once and deploy them across desktop, mobile and embedded operating systems without rewriting the source code.
- Qt is partly C++ and partly native code depending on platform.
  - Qt UI APIs wrap native UI components.



What is Qt?

Qt is available for these platforms:
Embedded Linux
Mac OS X
Windows
Linux/X11
Windows CE/Mobile
Symbian
MeeGo

- Official support
  - Embedded Linux
  - Mac OS X
  - Windows
  - Linux/X11
  - Windows CE
  - S60
  - MeeGo

What is Qt?

- According to Wikipedia (http://en.wikipedia.org/wiki/Qt_(toolkit)) there's bindings to Qt from:
- Python
  - PySide (LGPL)
  - PyQt (GPL & commercial)
  - PythonQt (LGPL)
- Java
  - Jambi (LGPL)
- C#
  - Qyoto
- Ruby
  - QtRuby
- Ada
  - QtAda
- Pascal
  - FreePascal Qt4
- Perl
  - Perl Qt4
- PHP
  - PHP-Qt
- Haskell
  - Qt Haskell
- Lua
  - Lqt
- Dao
  - DaoQt
- TCL
  - Qtcl
- Common Lisp
  - CommonQt
- D
  - QtD

What can you do with it?

- You can do anything with it.
- There's plenty of stuff.

Can I do console applications?

- Yes, you can.
- QtCore http://qt.nokia.com/doc/qtcore.html
  - The QtCore module contains core non-GUI functionality.
  - QCoreApplication - Event loop for console Qt applications
    - http://doc.trolltech.com/qcoreapplication.html
- QtNetwork http://qt.nokia.com/doc/qtnetwork.html
  - The QtNetwork module offers classes that allow you to write TCP/IP clients and servers.

Can I do console applications?

- QtXml http://qt.nokia.com/doc/qtxml.html
  - The QtXml module provides a stream reader and writer for XML documents, and C++ implementations of SAX and DOM.
- QtXMLPatterns http://doc.qt.nokia.com/qtxmlpatterns.html
  - Contains classes for using XQuery and XPath in Qt programs.
- QtSql http://doc.qt.nokia.com/qtsql.html
  - The QtSql module helps you provide seamless database integration to your Qt applications.

Can I do graphical user interfaces?

- Of course you can.
  - ...multiple ways
- You can use all the previously mentioned non-GUI modules.
- These are some of the widgets from the QtGui module:
  - http://qt.nokia.com/doc/gallery.html
- Here are all the classes under QtGui module:
  - http://qt.nokia.com/doc/qtgui.html
- You can even do 3D graphics with it:
  - QtOpenGL http://qt.nokia.com/doc/qtopengl.html
  - The QtOpenGL module offers classes that make it easy to use OpenGL in Qt applications.
  - Examples can be found from: http://qt.nokia.com/doc/examples.html#opengl

Can I do graphical user interfaces

- There's even a web browser you can embed into your application:
  - QtWebkit http://qt.nokia.com/doc/qtwebkit.html
  - QtWebKit provides a web browser engine that makes it easy to embed content from the World Wide Web into your Qt application.
  - http://qt.nokia.com/doc/examples.html#webkit
- QtSvg http://qt.nokia.com/doc/qtsvg.html
  - The QtSvg module provides classes for displaying the contents of SVG files.

Why would I want to use it?

- You are a **NERD** and want to show off your graphical programming skills?
- Qt has a quite good penetration on different platforms.
- It's fairly straight-forward to port your application to platforms where Qt is supported.
- Qt has been around for a long time, so it has matured as a framework.
- It adds somewhat automatic memory management on top of C++ through different means.
- There's plenty of components.
- Because Nokia is using it!
  - Jobs?

What do I need to know to use it?

- As mentioned before Qt has bindings for different languages, but best support is for C++.
- I think that level for learning Qt is just basics of programming and C++.
- I had only done some C++ before started, but plenty of programming (mostly with Java).
- Knowing about C++ helps.
  - Pointers
  - References
  - Macros
  - Templates
  - All that ”weird” stuff

What are the fundamentals of Qt?

- Source: http://qt.nokia.com/doc/object.html
- Mostly automatic memory management through different means
  - hierarchical and queryable object trees that organize object ownership
    - http://qt.nokia.com/doc/objecttrees.html
    - Implemented with QObjects
  - implicitly shared classes
    - http://doc.trolltech.com/shared.html
  - variety of different smart pointers

What are the fundamentals of Qt

- object communication method called signals and slots
  - http://qt.nokia.com/doc/signalsandslots.html
  - Implemented through MOC http://qt.nokia.com/doc/moc.html#moc
- events and event filters
  - http://qt.nokia.com/doc/eventsandfilters.html
- queryable and designable object properties
  - http://qt.nokia.com/doc/properties.html
  - Implemented through MOC http://qt.nokia.com/doc/moc.html#moc

What are the fundamentals of Qt

- contextual string translation for internationalization
  - http://qt.nokia.com/doc/i18n.html
  - Simply use QObject::tr(”Text”);
- interval driven timers
  - http://qt.nokia.com/doc/timers.html
  - We’ll leave these out, but it’s good to mention them.
- QObjects accompanied with moc are mostly the source of magic in Qt
  - They enable some of the cool functionality based on inheritance from QObject and code generation.

What are fundamentals of Qt?

- The Meta-Object Compiler, moc, is the program that handles Qt's C++ extensions.
  - http://qt.nokia.com/doc/moc.html#moc
  - Provides us the Qts Meta-Object System
    - http://qt.nokia.com/doc/metaobjects.html

Hierarchical and queryable object trees

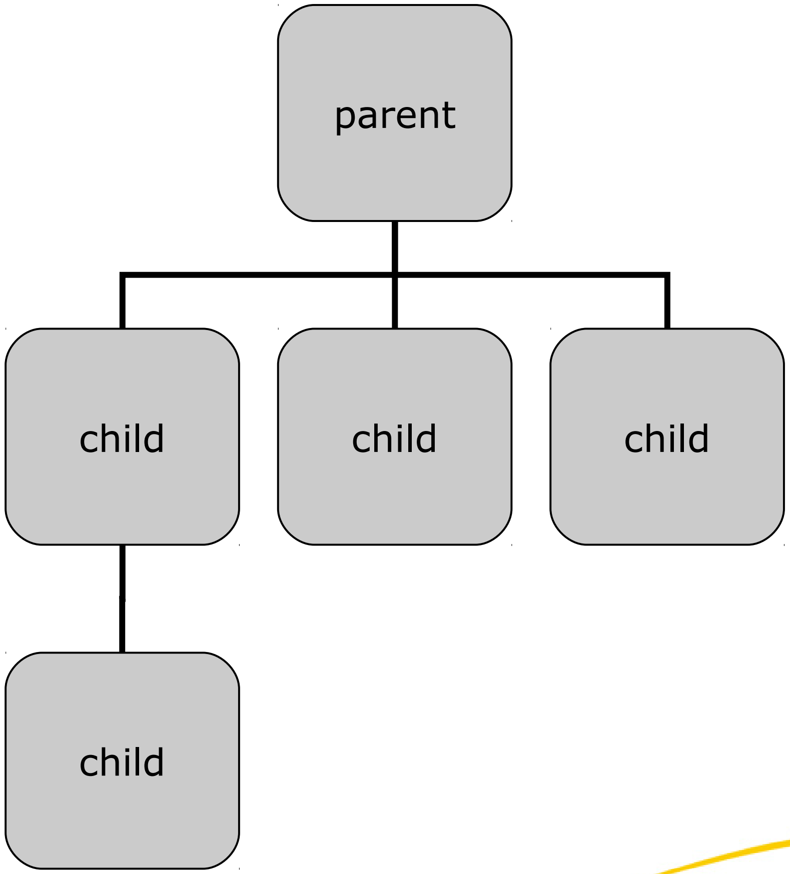
- Source: http://qt.nokia.com/doc/objecttrees.html
- QObjects organize themselves in object trees.
- When you create a QObject with another object as parent.
  - It's added to the parent's children() list.
  - It is deleted when the parent is.
- A QShortcut (keyboard shortcut) is a child of the relevant window.
  - So when the user closes that window, the shorcut is deleted too.

Construction/Destruction Order of QObjects

- You can construct a tree of QObjects in any order you want.
  - Of course you have to create the QObject before you can use it as a parent.
- After the object tree has been created, it can be destroyed in any order.
- When any QObject in the tree is deleted
  - if the object has a parent
    - the destructor automatically removes the object from its parent
  - if the object has children
    - the destructor automatically deletes each child
- No QObject is deleted twice, regardless of the order of destruction.

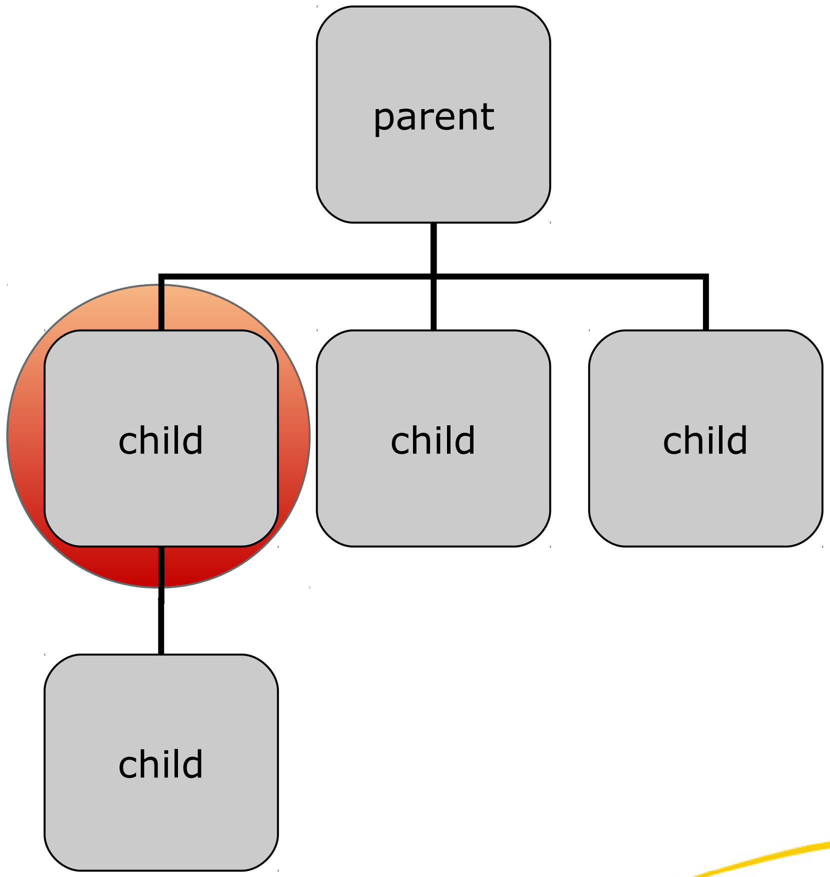
Construction/Destruction order of QObjects

- It looks something like this:



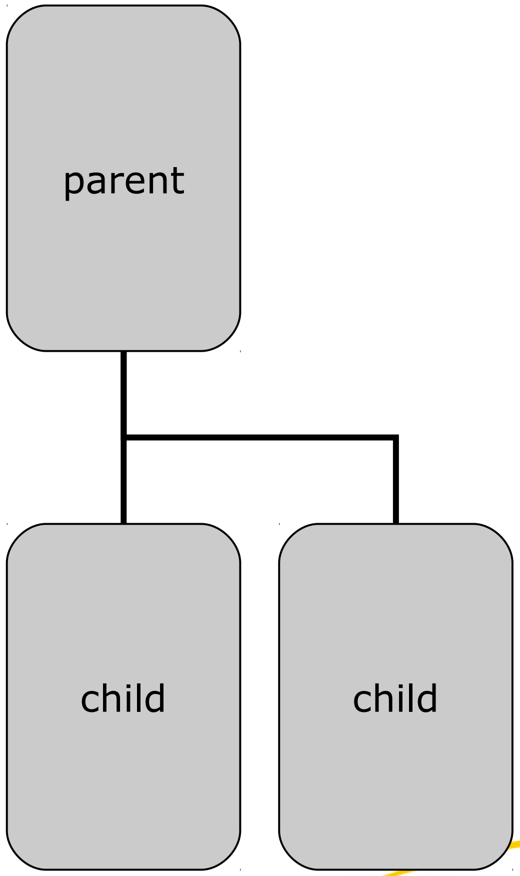
Hierarchical and queryable object trees

- We do a delete on one child:



Hierarchical and queryable object trees

- We get:



What is the stack?

- Stack is something like:
- Image source: http://www.apartmenttherapy.com/uimages/ny/4-17-stack-drawers-1.jpg

What is the stack?

- Source: http://en.wikipedia.org/wiki/Call_stack
- A call stack is a stack data structure
  - stores information about the active subroutines
- This kind of stack is also known as
  - an execution stack
  - control stack
  - function stack
  - run-time stack
  - the stack
- When memory allocation happens to the stack
  - It will be automatically released after it goes out of scope.

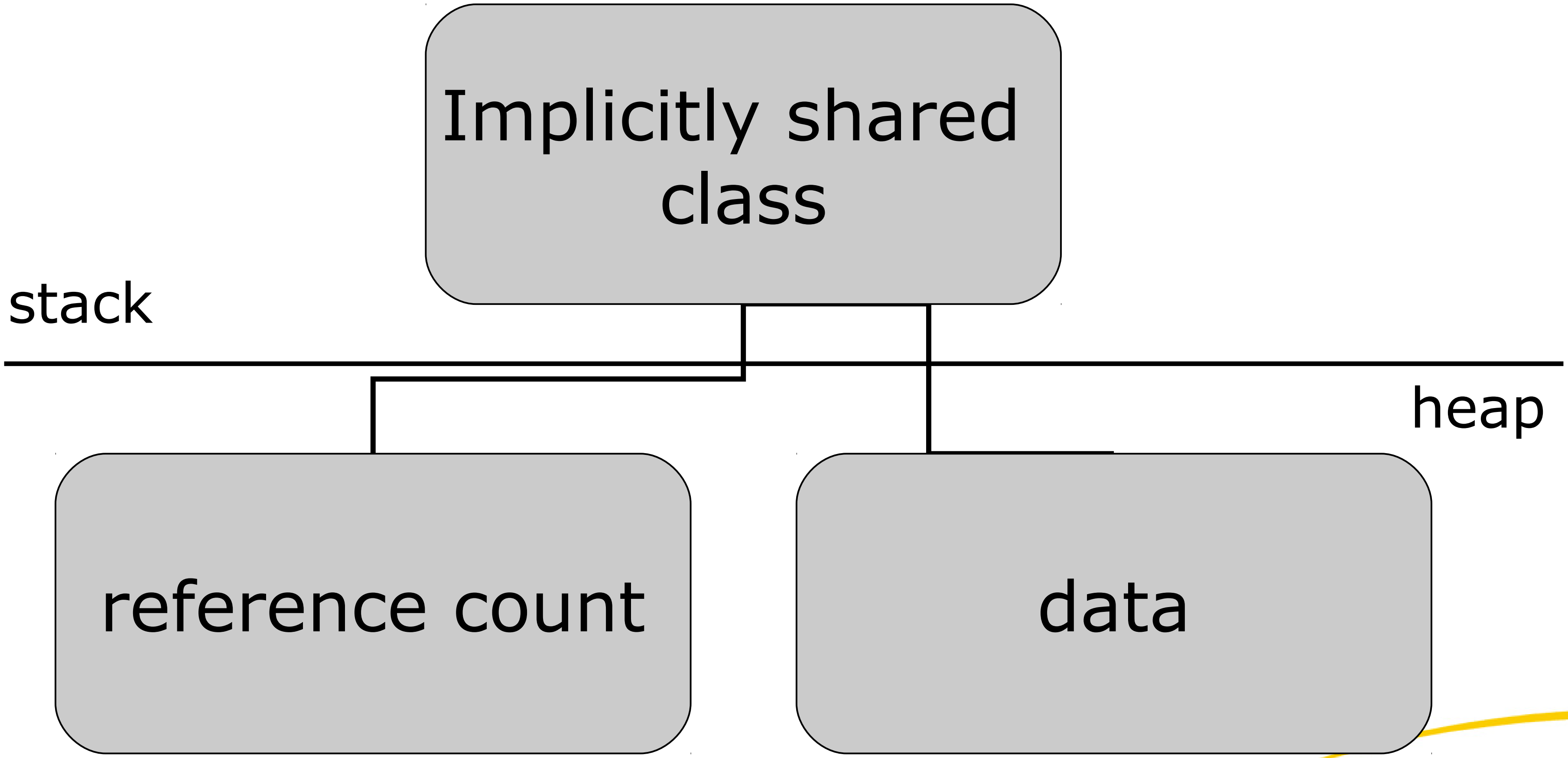
What is the heap?

- Source: http://en.wikipedia.org/wiki/Dynamic_memory_allocation
- dynamic memory allocation
  - also known as heap-based memory allocation
  - is the allocation of memory storage for use during the runtime
- Dynamically allocated memory exists until it is released
  - explicitly by the programmer
    - like in C++
  - by the garbage collector
    - like in Java and C#
- This is in contrast to static memory allocation, which has a fixed duration. It is said that an object so allocated has a dynamic lifetime (as in stack).

Implicitly shared classes

- Source: http://doc.trolltech.com/shared.html
- When you see stack allocated object
  - they might not always be storing all the data in the stack
- But data might be in the heap
  - the stack is only used for automatic dereferencing
- Stack objects point to a heap allocated shared data.
- A shared class consists of:
  - a pointer to a shared data block that contains:
    - a reference count
    - the data.

Implicitly shared classes



Implicitly shared classes

- When a shared object is created
  - it sets the reference count to 1
  - the reference count is incremented whenever a new object references the shared data
  - and decremented when the object dereferences the shared data
  - The shared data is deleted when the reference count becomes zero

Implicitly shared classes

- So when you see something like:
  - QString foo = "foo";
- QString foo is in the stack
  - BUT it has a reference to a data block which has
    - a reference count
      - 1
    - and the data
      - "foo"

Implicitly shared classes

- So when you do QString bar = foo;
  - QString class now does a shallow copy (to the stack)
    - the foo and bar now actually point to the SAME data block
    - only the reference count was increased
      - 2
- If you now do bar.append("bar");
  - Now the QString class does a deep copy for bar.
    - The data from foo was copied.
      - "bar" was appended to it.
      - reference count is now 1
    - foos reference count is now 1.
- Simple, isn’t it!

Different smart pointers

- Source: http://labs.trolltech.com/blogs/2009/08/25/count-with-me-how-many-smart-pointer-classes-does-qt-have/

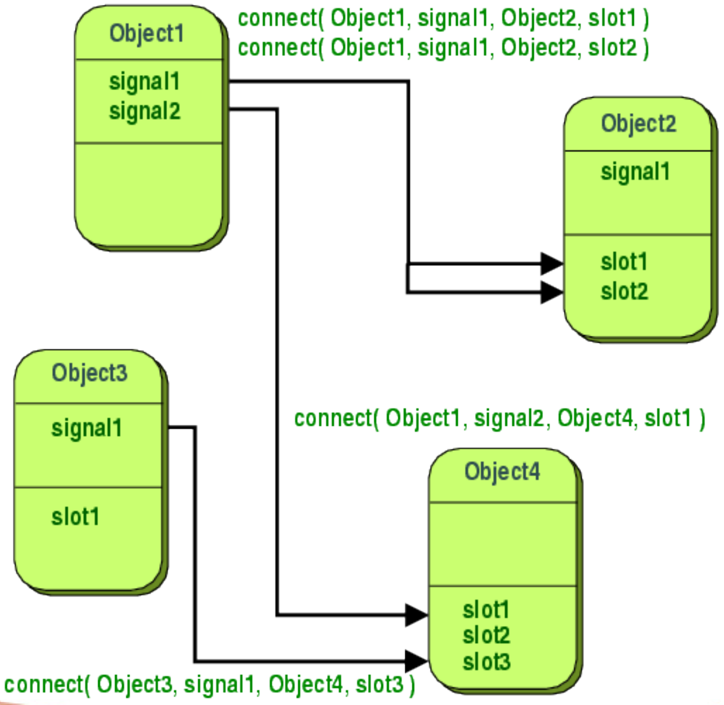
QSharedDataPointer / QExplicitlySharedDataPointer	Implements sharing of data (not of pointers), implicitly and explicitly, respectively. The QSharedDataPointer class represents a pointer to an implicitly shared object.
QSharedPointer	Implements reference-counted strong sharing of pointers. QSharedPointer will delete the pointer it is holding when it goes out of scope, provided no other QSharedPointer objects are referencing it.

Different smart pointers

QWeakPointer	Implements reference-counted weak sharing of pointers. The QWeakPointer is an automatic weak reference to a pointer in C++. It cannot be used to dereference the pointer directly, but it can be used to verify if the pointer has been deleted or not in another context. QWeakPointer objects can only be created by assignment from a QSharedPointer.
QScopedPointer / QScopedArrayPointer	Implements non-reference-counted strong pointer wrapper (QSharedPointer's little brother). The QScopedPointer class stores a pointer to a dynamically allocated object, and deletes it upon destruction.

Qt object model: Signals and slots

- Source: http://qt.nokia.com/doc/signalsandslots.html
- It's a way for QObjects to communicate without knowing each other.



Object1
signal1
signal2
connect(Object1, signal1, Object2, slot1)
connect(Object1, signal1, Object2, slot2)
Object2
signal1
slot1
slot2
Object3
signal1
slot1
connect(Object1, signal2, Object4, slot1)
Object4
slot1
slot2
slot3
connect(Object3, signal1, Object4, slot3)

Signals and slots

- Signals are emited.
- Slots receive signals.
- Signals and slots are connected through QObject::connect
- Many slots can be connected to a signal.
- Signals can be connected to signals.

Signals and slots

```
QObject* sender = new QObject;
QObject* receiver = new QObject;
/* This will connect the signal for senders destruction
 * to receivers deleteLater. It will result to a situation
 * where if sender is deleted, the signal destroyed()
 * will be sent just before destruction and it will trigger
 * the receivers deleteLater which will result the receiver
 * being deleted as well. */
QObject::connect(sender, SIGNAL(destroyed()),
                 receiver, SLOT(deleteLater()));
sender->deleteLater();
```

Events and event filters

- Source: http://doc.trolltech.com/eventsandfilters.html
- In Qt, events are objects, derived from the abstract QEvent class.

How events are delivered?

- Source: http://qt.nokia.com/doc/eventsandfilters.html#how-events-are-delivered
- As event occurs
  - Qt creates an event object to represent it
    - by constructing an instance of the appropriate QEvent subclass
      - http://doc.trolltech.com/qevent.html
      - ie. QMouseEvent, QKeyEvent, QPaintEvent
  - delivers it to a particular instance of QObject (or one of its subclasses) by calling its event() function

How events are delivered?

- event() function does not handle the event itself
  - but based on the type of event delivered
    - it calls an event handler for that specific type of event
    - sends a response based on whether the event was accepted or ignored

Event types

- Source: http://qt.nokia.com/doc/eventsandfilters.html#event-types
- Most event types have special classes
- As said earlier, they are descents of QEvent
    - http://qt.nokia.com/doc/qevent.html

Event types

- common ones are:
  - QResizeEvent http://qt.nokia.com/doc/qresizeevent.html
    - Resize events!
  - QPaintEvent http://qt.nokia.com/doc/qpaintevent.html
    - Paint events!
  - QMouseEvent http://qt.nokia.com/doc/qmouseevent.html
    - Mouse events!
  - QKeyEvent http://qt.nokia.com/doc/qkeyevent.html
    - Key events!
  - QCloseEvent http://qt.nokia.com/doc/qcloseevent.html
    - Close events (like window, dialog, application)

Event handlers

- Source: http://qt.nokia.com/doc/eventsandfilters.html#event-handlers
- Usually there's a virtual function in a base class.
  - like QWidget
- Handling an event happens so that:
  - You inherit the class and implement the virtual function.

Event filters

- Source: http://qt.nokia.com/doc/eventsandfilters.html#event-filters
- It is possible to write event filters
  - Event filters can look or even intercept the events
  - Inherit QObject
  - Implement bool QObject::eventFilter(QObject *object, QEvent *event)
  - They work per QObject basis
    - You set an event filter to a QObject with
      - QObject::installEventFilter() http://qt.nokia.com/doc/qobject.html#installEventFilter

Queryable and designable object properties

- Source: http://qt.nokia.com/doc/properties.html
- A property behaves like a class data member.
- It has additional features accessible through the Meta-Object System.
- It is implemented through moc .
  - Needs Q_OBJECT in the class header declaration.

Queryable and designable object properties

- You can do something like:
  - `Q_PROPERTY(bool enabled READ isEnabled WRITE setEnabled)`
    - bool refers to the type of the property
    - enabled is the property name
    - READ obviously refers to the reading
    - isEnabled is the function called for reading
      - it has to have a public implementation
    - WRITE, again obviously, refers to writing
    - setEnabled is the function to be called when writing
      - it has to have a public implementation

Moc and meta-object model

- Source: http://qt.nokia.com/doc/moc.html#moc
- The Meta-Object Compiler, moc, is the program that handles Qt's C++ extensions.
- Provides us the Qts Meta-Object System
- Magic happens through macros like:
  - Q_OBJECT
  - Q_PROPERTY
  - Q_ENUMS
  - signals:
  - public slots:
  - private slots:

Moc and meta-object model

- This moc tool reads your header file
- Tries to find the Q_OBJECT macro
- Generates meta-object code for those classes
  - code for signals and slots
    - where signals, public slots or private slots are used
  - code for properties
    - where Q_PROPERTY is used
  - code for enums
    - where Q_ENUMS is used
    - Registering an enumeration type makes the enumerator names available for use in calls to QObject::setProperty().

How do I do GUI applications with it?

- You can do with a designer like in Qt Creator
- Or like real NERDS, by hand.
  - Yeah, just kidding...
- The Qt Creator is mostly in the way of you want to do anything more complicated.
- It’s also fairly cumbersome to add your own components to the designers component library.

How do I do GUI applications with it?

- There’s (at the moment) two ways of doing UI.
  - QWidget based approach
    - You do inherit Qt’s components and add functionality
    - Inherit just QWidget
      - Implement QWidget::paintEvent(QPaintEvent*)
      - do custom painting with QPainter
        - http://doc.trolltech.com/qpainter.html
    - Use Qt’s style sheets to style the widgets
      - http://doc.trolltech.com/stylesheet.html
  - Graphics View framework
    - Totally custom drawing.
    - Can use QWidgets in Graphics View through proxys.
    - http://doc.trolltech.com/graphicsview.html
  - Qt Quick
    - Custom UI
    - Declarative description of the UI with EcmaScript

How do I do Hello World!?

```
#include <QtGui/QApplication>
#include <QtGui/QLabel>
int main(int argc, char **argv)
{
  QApplication a(argc, argv);
  /** Create QLabel to stack
   * so we don't have to delete it. */
  QLabel label("Hello World!");
  /** Show it! */
  label.show();
  /** Kick off the event loop. */
  return a.exec();
}
```

How do I do Hello World?!

- A bit more complicated Hello World with QMainWindow, QFrame, QVBoxLayout and QLabel.

How do I do events?

- Sending events
  - `bool QCoreApplication::sendEvent(QObject* receiver, QEvent* event)`
- Receiving events
  - Use the specialized event handlers
    - paintEvent
    - resizeEvent
    - etc…

How do I do signals and slots?

- I'll show the example code...

How do I do components?

- The basic idea is to have very atomic use cases.
- Try to keep the dependencies as low as possible.
- Use signals and slots for communication.
- Try to use basic types as parameters for the signals and slots

How do I do components?

- A simple component without graphical interface.
  - QCounter : public QObject
    - public
      - int value()
    - public slots
      - increment()
      - increment(int amount)
      - decrement()
      - setValue(int newValue)
    - signals
      - valueChanged(int newValue)
      - valueChanged(const QString& value)
      - valueChangedDelta(int delta)
    - private
      - int m_value

How do I do components

- We wrap that QCounter to a graphical representation
  - QGraphicalCounter : public QLabel
    - We inherit now from QLabel so we get the graphical representation easily
    - We can’t really use multiple inheritance because QCounter inherits from QObject as does QLabel.
      - I tried virtual inheritance, but couldn’t get it working.
    - Everything else is the same, we expose the same interface but we internally use QCounter.
    - We wire signal valueChanged(const QString&) to setText(const QString&) slot.
    - And we’ve got a graphical interface for the counter.

How do I do components?

- Let's do a CarCalculator
  - We need a addition button
  - So we'll create a QAddButton

How do I do components?

- Then we need a reset button.
  - For that we'll create a iterator which iterates over QGraphicalCounters.
  - The iterator will send a reference to a found QGraphicalCounter every time it finds one.
    - We'll use the QObjects object tree feature to implement this feature
      - QObjects are usually created by assigning a parent.
      - That created object will be the child of the parent object
      - The tree can be easily iterated.

How do I do components?

- …reset button continues:
  - We'll wire just an ordinary QPushButton to a slot which will give the QCounterIterator a starting point for iteration
  - We'll wire the found signal to one of the slots which will reset the counter.

How do I do components?

- Ok so we've glued all this together with code...

Qt Quick

- http://doc.qt.nokia.com/qtquick.html
- Qt User Interface Construction Kit
- First bundle release of Qt Quick on 4.7
- Technology to build intuitive modern-looking fluid user interfaces
- Qt Quick is concept of
  - Tools:
    - QtCreator
    - QML Designer
    - QMLViewer
  - QML language
  - Declarative Runtime (interprets QML code)
- Fast design to market time
- Encourages to implement tailored UIs for each platform / device

QML

- Script language
- Uses JavaScript as glue code for controlling the behaviour of the user interface
- Declarative way to define UI behaviour
- Contains set of very basic user interface controls called as QML elements for example
  - http://doc.qt.nokia.com/qdeclarativeelements.html
  - Rectangle
  - Text
  - Image
  - Flickable
- Elements have properties which can have bindings to other properties

QML

- Complete applications can be made with QML + JavaScript and executed by QMLViewer
- More sophisticated and performance critical applications require Qt / C++ code.
- Behind scenes
  - QML elements are interpreted as QGraphicsObjects and placed into QGraphicsScene
  - QGraphicsView is used to visualize the objects / QML elements
- However, deployment of Qt Quick application is recommended to be constructed as Qt application that invokes declarative runtime
  - This can be accomplished easily by using QDeclarativeView class and handing the main QML file to it

Qt Quick links

- Qt 4.7 and Qt Creator
  - http://qt.nokia.com/downloads/
- Qt Quick documentation
  - http://doc.qt.nokia.com/qtquick.html
- Qt Quick Playground
  - https://projects.forum.nokia.com/qtquickplayground
- Qt Bubble Level
  - https://projects.forum.nokia.com/qtbubblelevel
- Forum Nokia Wiki on Qt Quick
  - http://wiki.forum.nokia.com/index.php/Category:Qt_Quick

Questions?

- If we have time for any…

Thank You!

tarmo.aidantausta@digia.com