Qt Essentials - Objects Module

Qt Essentials - Training Course

Produced by Nokia, Qt Development Frameworks

Material based on Qt 4.7, created on December 15, 2010

http://qt.nokia.com

Module: Objects in Qt

- Common Features of Qt's Object Model
- Object Communication using Signals & Slots
- Signal/Slot Variations
- Handling Events in Qt

Module Learning Objectives

- Learn ...
  - ... about Qt's object model
  - ... about parent-child relationships in Qt
  - ... what a widget is
  - ... how to combine widgets
  - ... what signals & slots are
  - ... how to use signals & slots for object communication
  - ... which variations for signal/slot connections exist
  - ... how to create custom signals & slots
  - ... how Qt handles events

Module: Objects in Qt

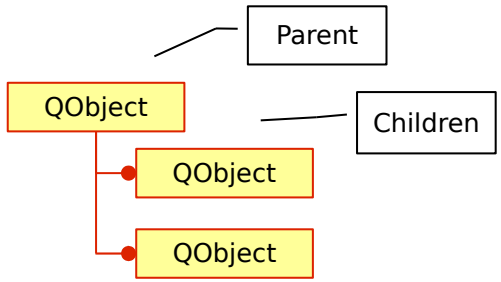
- Common Features of Qt's Object Model
- Object Communication using Signals & Slots
- Signal/Slot Variations
- Handling Events in Qt

Qt's C++ Object Model - `QObject`

- `QObject` is the heart of Qt's object model
- Adds features to C++, like ...
  - Signals and slots
  - Properties
  - Event handling
  - Memory management
  - ...
- Some features are standard C++
  - Some use Qt's meta-object system
- `QObject` has no visual representation

Object Tree

- `QObjects` organize themselves in object trees
  - Based on parent-child relationship
- `QObject(QObject *parent = 0)`
  - Parent adds object to list of children
  - Parent owns children
- Construction/Destruction
  - Tree can be constructured in any order
  - Tree can be destroyed in any order
    - if object has parent: object first removed from parent
    - if object has children: deletes each child first
    - No object is deleted twice



Note: Parent-child relationship is NOT inheritance

Creating Objects

- **On Heap** - `QObject` with parent:

```
QLabel *label = new QLabel("Some Text", parent);
```

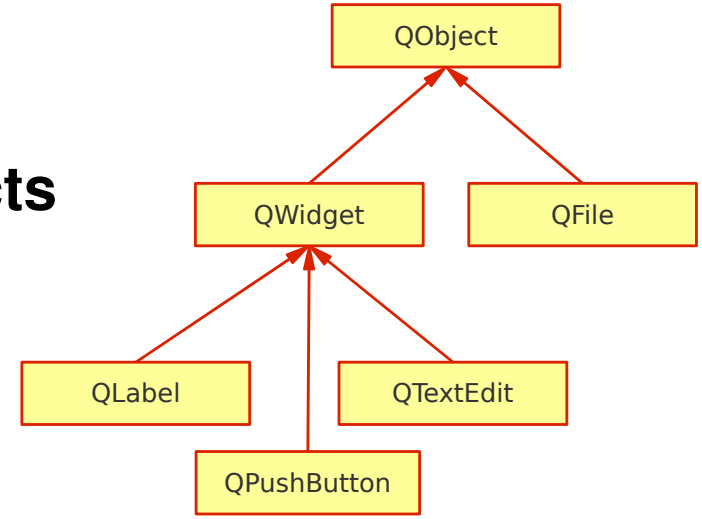
  - `QLayout::addWidget()` and `QWidget::setLayout()` reparent children automatically
- **On Stack** - `QObject` without parent:
  - `QFile`, usually local to a function
  - `QApplication` (local to `main()`)
  - Top level `QWidgets`: `QMainWindow`
- **On Stack** - "value" types See `QVariant::Type` Documentation

```
QString name;
QStringList list;
QColor color;
```

  - Do not inherit QObject
  - Passed by value everywhere
  - Exception: QString is implicitly shared (COW strategy)
- **Stack or Heap** - `QDialog` - depending on lifetime

Qt's Widget Model - `QWidget`

- **Derived from `QObject`**
  - Adds visual representation
- **Base of user interface objects**
- **Receives events**
  - e.g. mouse, keyboard events
- **Paints itself on screen**
  - Using styles

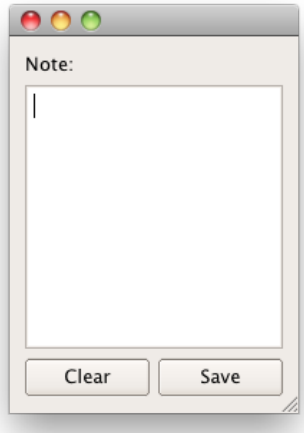


Object Tree and QWidget

- `new QWidget(0)`
  - Widget with no parent = "window"
- **QWidget's children**
  - Positioned in parent's coordinate system
  - Clipped by parent's boundaries
- **QWidget parent**
  - Propagates state changes
    - hides/shows children when it is hidden/shown itself
    - enables/disables children when it is enabled/disabled itself
- ***Tristate* mechanism**
  - For hide/show and enable/disable, ensures that e.g. an explicitly hidden child is not shown when the parent is shown.
  - Demo objects/ex-showhide

Widgets that contain other widgets

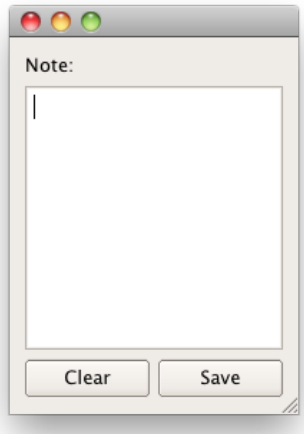
- Container Widget
  - Aggregates other child-widgets
- Use layouts for aggregation
  - In this example: `QHBoxLayout` and `QVBoxLayout`
  - Note: Layouts are *not* widgets
- Layout Process
  - Add widgets to layout
  - Layouts may be nested
  - Set layout on container widget



Example Container Widget

```
// container (window) widget creation
QWidget container;          // top-level widget on stack
QLabel* label = new QLabel("Note:", container);
QTextEdit* edit = new QTextEdit(container);
QPushButton* clear = new QPushButton("Clear", container);
QPushButton* save = new QPushButton("Save", container);
```

```
// widget layout
QVBoxLayout* outer = new QVBoxLayout();
outer->addWidget(label);
outer->addWidget(edit);
QHBoxLayout* inner = new QHBoxLayout();
inner->addWidget(clear);
inner->addWidget(save);
```




```
container.setLayout(outer);
outer->addLayout(inner); // nesting layouts
```

Demo objects/ex-simplelayout

Questions And Answers

- What is an object tree?
- Which pointers to QObjects do you need to keep around?
- What does it mean when a QWidget has no parent?
- Allocate on heap or stack?
  `QWidget; QStringList; QApplication; QString; QFile`
- Name some layout managers and when to use them
- What does it mean to nest layouts?

Questions And Answers

- What is an object tree?
- Which pointers to QObjects do you need to keep around?
- What does it mean when a QWidget has no parent?
- Allocate on heap or stack?
  `QWidget; QStringList; QApplication; QString; QFile`
- Name some layout managers and when to use them
- What does it mean to nest layouts?

Questions And Answers

- What is an object tree?
- Which pointers to QObjects do you need to keep around?
- What does it mean when a QWidget has no parent?
- Allocate on heap or stack?
  `QWidget; QStringList; QApplication; QString; QFile`
- Name some layout managers and when to use them
- What does it mean to nest layouts?

Questions And Answers

- What is an object tree?
- Which pointers to QObjects do you need to keep around?
- What does it mean when a QWidget has no parent?
- Allocate on heap or stack?
  `QWidget; QStringList; QApplication; QString; QFile`
- Name some layout managers and when to use them
- What does it mean to nest layouts?

Questions And Answers

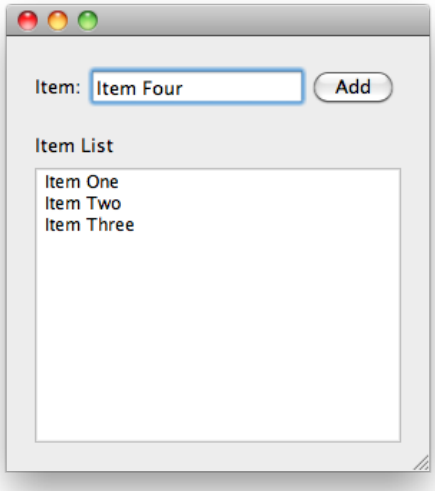
- What is an object tree?
- Which pointers to QObjects do you need to keep around?
- What does it mean when a QWidget has no parent?
- Allocate on heap or stack?
  `QWidget; QStringList; QApplication; QString; QFile`
- Name some layout managers and when to use them
- What does it mean to nest layouts?

Questions And Answers

- What is an object tree?
- Which pointers to QObjects do you need to keep around?
- What does it mean when a QWidget has no parent?
- Allocate on heap or stack?
  `QWidget; QStringList; QApplication; QString; QFile`
- Name some layout managers and when to use them
- What does it mean to nest layouts?

Lab: Your first Qt Application

- **Implement the application shown here**
  - Search the widgets
    - See Qt Widget Gallery Documentation
    - ... and choose your os style
  - Layouts: `QHBoxLayout, QVBoxLayout`
    - See previous slides how to use them
- **Optionally**
  - Provide a window title
  - Add Edit, Remove buttons
    - On the right of list
  - Use group box to provide list caption



Lab objects/lab-firstapp

Module: Objects in Qt

- Common Features of Qt's Object Model
- Object Communication using Signals & Slots
- Signal/Slot Variations
- Handling Events in Qt

Callbacks

General Problem

How do you get from "the user clicks a button" to your business logic?

- Possible solutions
  - Callbacks
    - Based on function pointers
    - Not type-safe
  - Observer Pattern (Listener)
    - Based on interface classes
    - Needs listener registration
    - Many interface classes
- Qt uses
  - Signals and slots for high-level (semantic) callbacks
  - Virtual methods for low-level (syntactic) events.

Signals & Slots

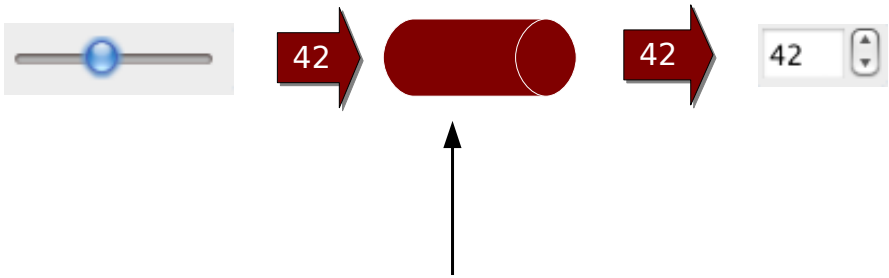
- Object Communication
  - **Signal** - *emitted to notify other objects*
  - **Slot** - *method called in response to signal*
- Provides type-safe callbacks
- After getting used to it, they are
  - easier to use than message maps,
  - more secure than callbacks,
  - more flexible than virtual methods
- Fosters component-based programming

Connecting Signals to Slots

Connecting Signals to Slots

Connecting Signals to Slots

Connecting Signals to Slots

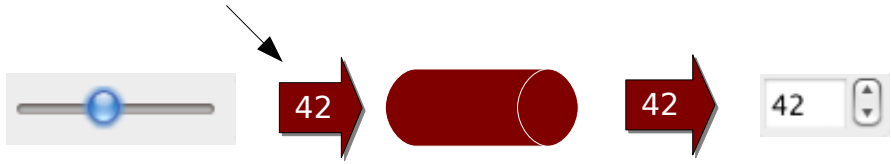


Connecting Signals to Slots


```
QObject::connect( slider, SIGNAL( valueChanged( int ) ),
                  spinbox, SLOT( setValue( int ) ) );
```

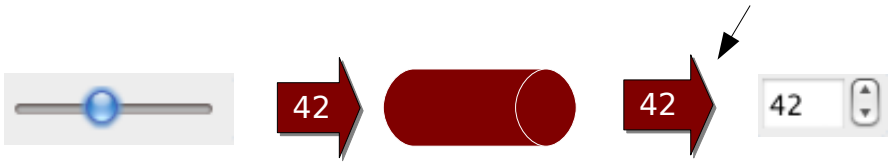
Connecting Signals to Slots

```
void QSlider::mousePressEvent(...)
{
  ...
  emit valueChanged( newValue );
  ...
}
```



Connecting Signals to Slots

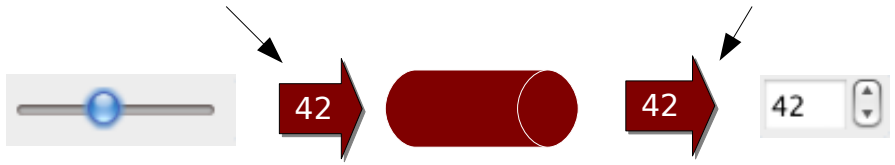
```
void QSlider::setValue( int value )
{
    ...
    m_value = value;
    ...
}
```



Connecting Signals to Slots

```
void QSlider::mousePressEvent(...)
{
  ...
  emit valueChanged( newValue );
  ...
}
```

```
void QSlider::setValue( int value )
{
  ...
  m_value = value;
  ...
}
```



Signal emitted

Signal/Slot connection

Slot implemented

```
QObject::connect( slider, SIGNAL( valueChanged( int ) ),
                  spinbox, SLOT( setValue( int ) ) );
```

Demo objects/ex-connect

Custom Slots

- File: **myclass.h**

```
class MyClass : public QObject
{
  Q_OBJECT // marker for moc
  // ...
public slots:
  void setValue(int value); // a custom slot
};
```

- File: **myclass.cpp**

```
void MyClass::setValue(int value) {
  // slot implementation
}
```

Custom Signals

- File: **myclass.h**

```
class MyClass : public QObject
{
  Q_OBJECT // marker for moc
  // ...
signals:
  void valueChanged(int value); // a custom signal
};
```

- File: **myclass.cpp**

```
// No implementation for a signal
```

- Sending a signal

```
emit valueChanged(value);
```

Q_OBJECT - flag for MOC

- **Q_OBJECT**
  - Enhances QObject with meta-object information
  - Required for Signals & Slots
  - **moc** creates meta-object information

```
moc -o moc_myclass.cpp myclass.h
c++ -c myclass.cpp; c++ -c moc_myclass.cpp
c++ -o myapp moc_myclass.o myclass.o
```

  - qmake takes care of moc files for you
- **Analyze definition of**
  - Q_OBJECT
  - signals and slots
  - emit
  - At *$QTDIR/src/corelib/kernel/qobjectdefs.h*
- **Look at moc generated files**
  - Demo objects/ex-signalslots

Back to the Original Problem

We asked some slides ago...

How to react to a button being clicked?

- Solution:
  - *Implement a slot in your widget*
  - *Connect the button's clicked signal to the slot*
- Connect statement

```
connect(sender, signal, receiver, slot);
```

- Example

```
connect(button, SIGNAL(clicked()), this, SLOT(onClicked()));
```

Lab: Connect to Click

- **Create an application as shown here**
  - Clicking on “Select Color” updates label with color’s name.
- **Hints**
  - `QColorDialog::getColor()` to fetch a color
  - `QColor::name()` to get the color name
- **Optional**
  - In `QColorDialog`, honor the user clicking “cancel”, and provide it with the current color to start from.
  - Set the selected color as the label’s background (Hint: see `QPalette`)

Lab objects/lab-selectcolor

Module: Objects in Qt

- Common Features of Qt's Object Model
- Object Communication using Signals & Slots
- Signal/Slot Variations
- Handling Events in Qt

Variations of Signal/Slot Connections

Signal(s)	Connect to	Slot(s)
one	√	many
many	√	one
one	√	another signal

- Signal to Signal connection

```
connect(bt, SIGNAL(clicked()), this, SIGNAL(okSignal()));
```

- **Not** allowed to name parameters

```
connect( m_slider, SIGNAL( valueChanged( int value ) )
         this,     SLOT( setValue( int newValue ) ) )
```

Making the Connection

Rule for Signal/Slot Connection

Can ignore arguments, but not create values from nothing

Signal		Slot
rangeChanged(int,int)	√	setRange(int,int)
	√	setValue(int)
	√	updateUi()
valueChanged(int)	√	setValue(int)
	√	updateUi()
	✗	setRange(int,int)
	✗	setValue(float)
textChanged(QString)	✗	setValue(int)
clicked()	√	updateUi()
	✗	setValue(int)

Making the Connection

Rule for Signal/Slot Connection

Can ignore arguments, but not create values from nothing

Signal		Slot
rangeChanged(int,int)	√	setRange(int,int)
	√	setValue(int)
	√	updateUi()
valueChanged(int)	√	setValue(int)
	√	updateUi()
	✗	setRange(int,int)
	✗	setValue(float)
textChanged(QString)	✗	setValue(int)
clicked()	√	updateUi()
	✗	setValue(int)

Making the Connection

Rule for Signal/Slot Connection

Can ignore arguments, but not create values from nothing

Signal		Slot
rangeChanged(int,int)	√	setRange(int,int)
	√	setValue(int)
	√	updateUi()
valueChanged(int)	√	setValue(int)
	√	updateUi()
	✗	setRange(int,int)
	✗	setValue(float)
textChanged(QString)	✗	setValue(int)
clicked()	√	updateUi()
	✗	setValue(int)

Making the Connection

Rule for Signal/Slot Connection

Can ignore arguments, but not create values from nothing

Signal		Slot
rangeChanged(int,int)	√	setRange(int,int)
	√	setValue(int)
	√	updateUi()
valueChanged(int)	√	setValue(int)
	√	updateUi()
	✗	setRange(int,int)
	✗	setValue(float)
textChanged(QString)	✗	setValue(int)
clicked()	√	updateUi()
	✗	setValue(int)

Making the Connection

Rule for Signal/Slot Connection

Can ignore arguments, but not create values from nothing

Signal		Slot
rangeChanged(int,int)	√	setRange(int,int)
	√	setValue(int)
	√	updateUi()
valueChanged(int)	√	setValue(int)
	√	updateUi()
	✗	setRange(int,int)
	✗	setValue(float)
textChanged(QString)	✗	setValue(int)
clicked()	√	updateUi()
	✗	setValue(int)

Making the Connection

Rule for Signal/Slot Connection

Can ignore arguments, but not create values from nothing

Signal		Slot
rangeChanged(int,int)	√	setRange(int,int)
	√	setValue(int)
	√	updateUi()
valueChanged(int)	√	setValue(int)
	√	updateUi()
	✗	setRange(int,int)
	✗	setValue(float)
textChanged(QString)	✗	setValue(int)
clicked()	√	updateUi()
	✗	setValue(int)

Making the Connection

Rule for Signal/Slot Connection

Can ignore arguments, but not create values from nothing

Signal		Slot
rangeChanged(int,int)	√	setRange(int,int)
	√	setValue(int)
	√	updateUi()
valueChanged(int)	√	setValue(int)
	√	updateUi()
	✗	setRange(int,int)
	✗	setValue(float)
textChanged(QString)	✗	setValue(int)
clicked()	√	updateUi()
	✗	setValue(int)

Making the Connection

Rule for Signal/Slot Connection

Can ignore arguments, but not create values from nothing

Signal		Slot
rangeChanged(int,int)	√	setRange(int,int)
	√	setValue(int)
	√	updateUi()
valueChanged(int)	√	setValue(int)
	√	updateUi()
	✗	setRange(int,int)
	✗	setValue(float)
textChanged(QString)	✗	setValue(int)
clicked()	√	updateUi()
	✗	setValue(int)

Making the Connection

Rule for Signal/Slot Connection

Can ignore arguments, but not create values from nothing

Signal		Slot
rangeChanged(int,int)	√	setRange(int,int)
	√	setValue(int)
	√	updateUi()
valueChanged(int)	√	setValue(int)
	√	updateUi()
	X	setRange(int,int)
	X	setValue(float)
textChanged(QString)	X	setValue(int)
clicked()	√	updateUi()
	X	setValue(int)

Making the Connection

Rule for Signal/Slot Connection

Can ignore arguments, but not create values from nothing

Signal		Slot
rangeChanged(int,int)	√	setRange(int,int)
	√	setValue(int)
	√	updateUi()
valueChanged(int)	√	setValue(int)
	√	updateUi()
	X	setRange(int,int)
	X	setValue(float)
textChanged(QString)	X	setValue(int)
clicked()	√	updateUi()
	X	setValue(int)

Making the Connection

Rule for Signal/Slot Connection

Can ignore arguments, but not create values from nothing

Signal		Slot
rangeChanged(int,int)	√	setRange(int,int)
	√	setValue(int)
	√	updateUi()
valueChanged(int)	√	setValue(int)
	√	updateUi()
	✗	setRange(int,int)
	✗	setValue(float)
textChanged(QString)	✗	setValue(int)
clicked()	√	updateUi()
	✗	setValue(int)

Questions And Answers

- How do you connect a signal to a slot?
- How would you implement a slot?
- Name some possible signal/slot connection combinations?
- How would you emit a signal?
- Do you need a class to implement a slot?
- Can you return a value from a slot?
- When do you need to run `qmake`?
- Where do you place the `Q_OBJECT` macro and when do you need it?

Questions And Answers

- How do you connect a signal to a slot?
- How would you implement a slot?
- Name some possible signal/slot connection combinations?
- How would you emit a signal?
- Do you need a class to implement a slot?
- Can you return a value from a slot?
- When do you need to run `qmake`?
- Where do you place the `Q_OBJECT` macro and when do you need it?

Questions And Answers

- How do you connect a signal to a slot?
- How would you implement a slot?
- Name some possible signal/slot connection combinations?
- How would you emit a signal?
- Do you need a class to implement a slot?
- Can you return a value from a slot?
- When do you need to run `qmake`?
- Where do you place the `Q_OBJECT` macro and when do you need it?

Questions And Answers

- How do you connect a signal to a slot?
- How would you implement a slot?
- Name some possible signal/slot connection combinations?
- How would you emit a signal?
- Do you need a class to implement a slot?
- Can you return a value from a slot?
- When do you need to run `qmake`?
- Where do you place the `Q_OBJECT` macro and when do you need it?

Questions And Answers

- How do you connect a signal to a slot?
- How would you implement a slot?
- Name some possible signal/slot connection combinations?
- How would you emit a signal?
- Do you need a class to implement a slot?
- Can you return a value from a slot?
- When do you need to run `qmake`?
- Where do you place the `Q_OBJECT` macro and when do you need it?

Questions And Answers

- How do you connect a signal to a slot?
- How would you implement a slot?
- Name some possible signal/slot connection combinations?
- How would you emit a signal?
- Do you need a class to implement a slot?
- Can you return a value from a slot?
- When do you need to run `qmake`?
- Where do you place the `Q_OBJECT` macro and when do you need it?

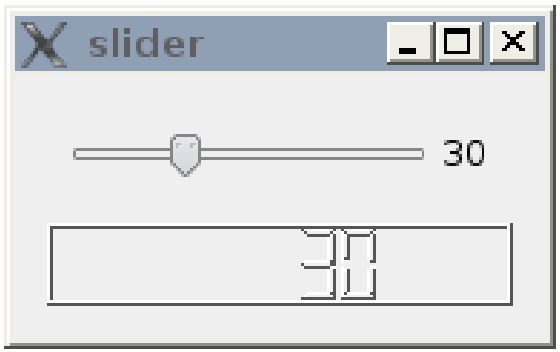
Questions And Answers

- How do you connect a signal to a slot?
- How would you implement a slot?
- Name some possible signal/slot connection combinations?
- How would you emit a signal?
- Do you need a class to implement a slot?
- Can you return a value from a slot?
- When do you need to run `qmake`?
- Where do you place the `Q_OBJECT` macro and when do you need it?

Questions And Answers

- How do you connect a signal to a slot?
- How would you implement a slot?
- Name some possible signal/slot connection combinations?
- How would you emit a signal?
- Do you need a class to implement a slot?
- Can you return a value from a slot?
- When do you need to run `qmake`?
- Where do you place the `Q_OBJECT` macro and when do you need it?

Lab: Source Compatibility



- **Implement custom slider**
  - API compatible with `QSlider`
  - Shows current value of slider
- **To create custom slider**
  - use `QSlider` and `QLabel`
- **To test slider**
  - `main.cpp` provides test code
  - `QLCDNumber` is part of test code
- **Optional:**
  - Discuss pros and cons of inheriting from QSlider instead of using an instance in a layout.

Lab objects/lab-slider

Module: Objects in Qt

- Common Features of Qt's Object Model
- Object Communication using Signals & Slots
- Signal/Slot Variations
- Handling Events in Qt

Event Processing

Qt is an event-driven UI toolkit

`QApplication::exec()` runs the *event loop*

1. **Generate Events**
   - by input devices: keyboard, mouse, etc.
   - by Qt itself (e.g. timers)
2. **Queue Events**
   - by event loop
3. **Dispatch Events**
   - by `QApplication` to receiver: `QObject`
     - *Key events sent to widget with focus*
     - *Mouse events sent to widget under cursor*
4. **Handle Events**
   - by `QObject` event handler methods

Event Handling

- QObject::event(QEvent *event)
  - Handles all events for this object
- Specialized event handlers
  - `QWidget::mousePressEvent()` for mouse clicks
  - `QWidget::keyPressEvent()` for key presses
- Accepting an Event
  - `event->accept()` / `event->ignore()`
    - Accepts or ignores the event
    - Accepted is the default.
- Event propagation
  - Happens if event is ignored
  - Might be propagated to parent widget

Demo objects/ex-allevents

Example of Event Handling

- `QCloseEvent` delivered to top level widgets (windows)
- Accepting event allows window to close
- Ignoring event keeps window open

```
void MyWidget::closeEvent(QCloseEvent *event) {
  if (maybeSave()) {
    writeSettings();
    event->accept(); // close window
  } else {
    event->ignore(); // keep window
  }
}
```

Demo objects/ex-closeevent

Events and Signals

Signals and slots are used instead of events:

- To communicate between components.
- In cases where there is a well-defined sender and receiver.
  - For example: a button and a slot to handle clicks.
- For some events, there is no sender in Qt.
  - For example: redraw, keyboard and mouse events.
- To describe high level logic and control flow.

Developers can create custom events if they need to.