Praktische Aspekte
der Informatik

Moritz Mühlhausen
Prof. Marcus Magnor
GUI Programming

Qt, Signals and Slots, moc, ui, and Qt Designer
Warning!
The following slides are meant to give you a very superficial introduction.

If you want to learn more, have a look at:
http://doc.qt.io/qt-5
About Qt

Signals and Slots (.moc)

User Interfaces (.ui)

Resources (.qrc)

QObject

Assignment
What is Qt?

• Qt is a cross-platform application and UI framework

• Qt comes with tons of extras
  ▪ It’s own build system (qmake)
  ▪ It’s own specialized IDE (Qt Creator)
  ▪ A great documentation (Qt Assistant)
What is Qt?

• Benefits:
  ▪ Big and Powerful
  ▪ Filled with useful extras
  ▪ Has excellent documentation (use it!)
  ▪ Comes with its own tool chain

• Drawbacks:
  ▪ May require extra compile steps
  ▪ You might not need everything
#include <QApplication>
#include <QMainWindow>

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

    QWidget * window = new QWidget;
    window->resize(320, 240);
    window->setWindowTitle("Simple example");

    window->show();
    return app.exec();
}
```cpp
#include <QApplication>
#include <QMainWindow>
#include <QBoxLayout>
#include <QPushButton>

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

    QWidget * window = new QWidget;
    window->resize(320, 240);
    window->setWindowTitle("Simple example");

    window->setLayout(new QBoxLayout(QBoxLayout::TopToBottom));

    QPushButton * button = new QPushButton("Magic!");
    window->layout()->addWidget(button);

    window.show();
    return app.exec();
}
```
What’s different in Qt?

• Signals and Slots
  ▪ Classes can communicate with SIGNALS and SLOTS.
  ▪ Can be queried/connected at runtime.
  ▪ Can communicate across threads.

• Code Generators
  ▪ Qt-specific stuff is translated to standard C++
  ▪ MOC: Qt-enhanced headers to C++
  ▪ UIC: ui-xml to C++ headers
  ▪ RCC: Resources to binaries

https://graphics.tu-bs.de/teaching/ss19/padi/
About Qt

**Signals and Slots (.moc)**

User Interfaces (.ui)

Resources (.qrc)

QObject

Assignment
class Counter : public QObject {
    Q_OBJECT // Mandatory!!

    public:
        Counter(QObject * parent = 0);
        virtual ~Counter();
        int value() const;

    public slots:
        void setValue(int value);

    signals:
        void valueChanged(int newValue); // needs no implementation!

    private:
        int value_;
};

...
Counter * sender = new Counter;
Counter * receiver = new Counter;

// Old Syntax (still valid)
connect(sender, SIGNAL(valueChanged(int)),
        receiver, SLOT(setValue(int)));

// NEW: Alternative syntax
connect(sender, &Counter::valueChanged,
        receiver, &Counter::setValue);

// NEW: Connecting to simple function
connect(sender, &Counter::valueChanged, someFunction);

https://wiki.qt.io/New_Signal_Slot_Syntax
Meta Object Compiler

- Sources (*.cpp)
- Compiled
- Objects (*.o)
- Linked
- Executable

- Headers (*.h)
  - included
  - moc’ed

- User Interfaces (*.ui)
  - uic’ed

- Resources (*.qrc)
  - moc’ed

- Sources (moc_*.cpp)

- Sources (qrc_*.cpp)


https://graphics.tu-bs.de/teaching/ss19/padi/
Makefile

$(EXE): main.o moc_counter.o counter.o
    g++ $(LIBPATH) $(OPTIONS) $^ -o $@

main.o: main.cpp
    g++ $(INCPATH) $(CCFLAGS) -c $<

moc_counter.o: moc_counter.cpp
    g++ $(INCPATH) $(CCFLAGS) -c moc_counter.cpp

moc_counter.cpp: counter.h
    moc $(INCPATH) $< -o $@

counter.o: counter.cpp
    g++ $(INCPATH) $(CCFLAGS) -c counter.cpp

clean:
    rm -f *.o *~ $(EXE) moc_*_.cpp
Outline

About Qt

Signals and Slots (.moc)

User Interfaces (.ui)

Resources (.qrc)

QObject

Assignment

https://graphics.tu-bs.de/teaching/ss19/padi/
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
<class>CommunicateClassName</class>
<widget class="QWidget" name="CommunicateClassName">
  <property name="geometry">
    <rect>
      <x>0</x>
      <y>0</y>
      <width>350</width>
      <height>190</height>
    </rect>
  </property>
  <property name="windowTitle">
    <string>Communicate</string>
  </property>
</widget>
<widget class="QPushButton" name="plus">
  <property name="geometry">
    <rect>
      <x>50</x>
      <y>40</y>
      <width>75</width>
      <height>30</height>
    </rect>
  </property>
</widget>
[...]

https://graphics.tu-bs.de/teaching/ss19/padi/
User Interfaces

https://graphics.tu-bs.de/teaching/ss19/padi/
User Interfaces

You can...

... compose a new widget from existing ones
... organize child widgets using layouts
... style widgets using stylesheets (qss)

```cpp
#include <QWidget>
#include "ui_communicate.h"

class Communicate : public QWidget {
    Q_OBJECT

public:
    Communicate(QWidget * parent = NULL)
        : QWidget(parent), ui(new Ui::Communicate) {
        ui->setupUi(this);
        connect(ui->plus, SIGNAL(clicked()), this, SLOT(onPlus()));
        connect(ui->minus, SIGNAL(clicked()), this, SLOT(onMinus()));
    }

    virtual ~Communicate() {
        delete ui;
    }

public slots:
    void onPlus() { ... }
    void onMinus() { ... }

private:
    Ui::Communicate * ui;
};
```
### Makefile

$(EXE): main.o moc_communicate.o communicate.o
    g++ $(LIBPATH) $(OPTIONS) $^ -o @$

main.o: main.cpp ui_communicate.h
    g++ $(INCPATH) $(CCFLAGS) -c $<

moc_communicate.o: moc_communicate.cpp
    g++ $(INCPATH) $(CCFLAGS) -c moc_communicate.cpp

moc_communicate.cpp: communicate.h ui_communicate.h
    moc $(INCPATH) $< -o @$

communicate.o: communicate.cpp ui_communicate.h
    g++ $(INCPATH) $(CCFLAGS) -c communicate.cpp

ui_communicate.h: communicate.ui
    uic $< -o @$
About Qt
Signals and Slots (.moc)
User Interfaces (.ui)

Resources (.qrc)
QObject
Assignment
icons.qrc
<!DOCTYPE RCC><RCC version="1.0">
<qresource>
     <file>images/copy.png</file>
     <file>images/cut.png</file>
     <file>images/new.png</file>
     <file>images/open.png</file>
     <file>images/paste.png</file>
     <file>images/save.png</file>
     ...
</qresource>
</RCC>

In your code:
QImage icon(":/images/copy.png");

http://doc.qt.io/qt-5/resources.html
Resource Compiler

Sources (*.cpp) -> Compiled

Objects (*.o) -> Linked

Executable

Headers (*.h) -> moc'ed

Sources (moc_*.cpp)

User Interfaces (*.ui) -> uic'ed

Resources (*.qrc) -> qrc'ed

Sources (qrc_*.cpp)

Outline

About Qt
Signals and Slots (.moc)
User Interfaces (.ui)
Resources (.qrc)

QObject
Assignment

https://graphics.tu-bs.de/teaching/ss19/padi/
What’s different in Qt?

• Code Generators
  ▪ Qt-specific stuff is translated to standard C++
  ▪ MOC: Qt-enhanced headers to C++
  ▪ UIC: ui-xml to C++ headers
  ▪ RCC: Resources to binaries
  ▪ ... or just use qmake

• Signals and Slots
  ▪ Classes communicate using SIGNALS and SLOTS.
    ▪ (That means objects do not have to know each other)
  ▪ Can be queried/connected at runtime.
  ▪ Can communicate across threads.

https://graphics.tu-bs.de/teaching/ss19/padi/
A few cool things about QObjects

- Have a look at the documentation of QObject!
  

**SIGNAL / SLOT**

**Events** (especially for QWidget)

**findChildren**

**Q_PROPERTY** macro

**QMetaObject**
If you want to create your very own Widget...

... you can add interaction by overloading these protected functions:

```cpp
virtual void keyPressEvent(QKeyEvent * event);
virtual void keyReleaseEvent(QKeyEvent * event);
virtual void mousePressEvent(QMouseEvent * event);
virtual void mouseReleaseEvent(QMouseEvent * event);
virtual void mouseMoveEvent(QMouseEvent * event);
```

... you can also customize what is drawn:

```cpp
virtual void paintEvent(QPaintEvent * event) {
    QPainter p(this); // check out the documentation!
    p.setPen(Qt::blue);
    p.setFont(QFont("Arial", 32));
    p.drawText(this->rect(), Qt::AlignCenter, "My Widget!");
    ... 
}
```
• By design QObjects know **nothing** about each other

• QObject are organized in a tree structure.
  - Delete one object (node) and the entire subtree gets cleaned up.
  - Objects should only communicate using **events** or **signals / slots**.

• `QObject::findChildren` can be useful:

  // Temporarily find all children that are QPushButtons
  QList<QPushButton*> allButtons = parent.findChildren<QPushButton*>(());

  // Temporarily find all children that are QWidgets named “widgetname”
  QList<QWidget*> widgets = parent.findChildren<QWidget*>("widgetname");
#include <QObject>

class MyObject : public QObject {

    Q_OBJECT
    Q_PROPERTY(int foo READ getFoo WRITE setFoo)

public:
    MyObject(QObject *parent = 0);
    virtual ~MyObject();

    int getFoo() const;
    void setFoo(int foo);

private:
    int foo;
}

... 

// Data-driven design
MyObject a;
a.setProperty("foo", 42); // calls: a.setFoo(42);
QMetaObject

• Every QObject has a **QMetaObject**.

```cpp
myObject->metaObject()
```

• QMetaObjects are magic
  ▪ What is the name of this object’s class?
  ▪ How many function does my class have?
  ▪ Please, invoke function 5 with these arguments!
  ▪ … and many, many more.

• It's great for data-driven design!

A few cool things about QObjects

• Have a look at the documentation of QObject!

SIGNAL / SLOT

Events  (especially for QWidget)

findChildren

Q_PROPERTY macro

QMetaObject
Outline

About Qt
Signals and Slots (.moc)
User Interfaces (.ui)
Resources (.qrc)
QObject

Assignment
A. **Simple Data Visualization**

Implement a small Qt widget that reads a .csv file, and draws a simple graph (Bar Graph, Line Graph, etc.)

Look at `assignment_stub` for more information.

B. **Have Fun!**

How about more graph types? How about multiple graphs? How about animation?

...