NHERI SIMCENTER PROGRAMMING BOOTCAMP

JULY 30 THROUGH AUGUST 3, 2018, AT UC BERKELEY’S RICHMOND FIELD STATION

GUI Development
GUI Design Fundamentals

The Qt Framework
- Common Data Types/Classes
- Building the UI
- Layout Management
- Signals and Slots
- Model – View – Controller Concept
- Helper Widgets

Quite a few Exercise Sessions
GUI FUNDAMENTALS

What is a WINDOW?
What is a WINDOW?

- “A rectangular area on your screen”
- “Any rectangular area on your screen”
GUI FUNDAMENTALS

Characteristics of an Application with a GUI

- Arbitrary sequence of execution
- May change shape/size
- May be (partially) covered
- Can be active or inactive
DESIGNING AN APPLICATION

1. CLOSE YOUR LAPTOP/WALK AWAY FROM YOUR COMPUTER!

2. Define target requirements – write them down!
   - Basic functionality
   - Available/required input
   - Desired outcome/output

3. Develop User Interface (UI)
   1. Sketch on paper/whiteboard/napkin/BART ticket/etc.
   2. Redo a few times till you like it; Draw a large sketch of the final version
   3. Identify all objects by type and functionality
   4. Play use-scenarios on paper
   5. Update your design as needed
DESIGNING AN APPLICATION

- On the way to Version 0.1 of the PileGroupTool
  - First idea
  - Rough sketch of elements and layout
DESIGNING AN APPLICATION

- Version 0.1 of PileGroupTool
- Layout of groups / Widgets
- Initial definitions

<table>
<thead>
<tr>
<th>Element ID</th>
<th>Element</th>
<th>Description</th>
<th>Category</th>
<th>Action &amp; Events</th>
<th>Else</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Problem definition area</td>
<td>container</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Parameter definition area</td>
<td>notebook</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>result visualization area</td>
<td>notebook</td>
<td></td>
<td></td>
<td>use instances of QCP</td>
</tr>
</tbody>
</table>
DESIGNING AN APPLICATION

- Version 0.1 of PileGroupTool
- Identifying individual elements
- Define type
- Define functionality / actions if clicked/changed/...

1a
1b
1c
1d
2a
2b
2c
2d
2e
2f
3a
3b
3c
3d
3e
## DESIGNING AN APPLICATION

<table>
<thead>
<tr>
<th>Element ID</th>
<th>Element</th>
<th>Description</th>
<th>Category</th>
<th>Action &amp; Events</th>
<th>Else</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Problem definition area</td>
<td>container</td>
<td></td>
<td>store info and adjust plot in section 1</td>
<td></td>
</tr>
<tr>
<td>1a</td>
<td>applied horizontal force</td>
<td>textinput</td>
<td></td>
<td>store info and adjust plot in section 1</td>
<td></td>
</tr>
<tr>
<td>1b</td>
<td>layer #1 thickness</td>
<td>textinput</td>
<td></td>
<td>store info and adjust plot in section 1</td>
<td></td>
</tr>
<tr>
<td>1c</td>
<td>layer #2 thickness</td>
<td>textinput</td>
<td></td>
<td>store info and adjust plot in section 1</td>
<td></td>
</tr>
<tr>
<td>1d</td>
<td>visualization/pile</td>
<td>graphic</td>
<td></td>
<td>double-click activates property section 2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Parameter definition area</td>
<td>notebook</td>
<td></td>
<td>update property variable upon change</td>
<td></td>
</tr>
<tr>
<td>2a</td>
<td>specific weight</td>
<td>textinput</td>
<td></td>
<td>update property variable upon change</td>
<td></td>
</tr>
<tr>
<td>2b</td>
<td>friction angle</td>
<td>textinput</td>
<td></td>
<td>update property variable upon change</td>
<td></td>
</tr>
<tr>
<td>2c</td>
<td>shear modulus</td>
<td>textinput</td>
<td></td>
<td>update property variable upon change</td>
<td></td>
</tr>
<tr>
<td>2d</td>
<td>pu (ultimate pressure)</td>
<td>textinput</td>
<td></td>
<td>update property variable upon change</td>
<td></td>
</tr>
<tr>
<td>2e</td>
<td>k-parameter</td>
<td>textinput</td>
<td></td>
<td>update property variable upon change</td>
<td></td>
</tr>
<tr>
<td>2f</td>
<td>ground water table</td>
<td>combo box: above</td>
<td>below</td>
<td>update property variable upon change</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>result visualization area</td>
<td>notebook</td>
<td></td>
<td>use instances of QCP</td>
<td></td>
</tr>
<tr>
<td>3a</td>
<td>displacement graph selector</td>
<td>visualize computed displacements</td>
<td>tab</td>
<td>change page in notebook to show respective result</td>
<td></td>
</tr>
<tr>
<td>3b</td>
<td>moment graph selector</td>
<td>tab</td>
<td></td>
<td>change page in notebook to show respective result</td>
<td></td>
</tr>
<tr>
<td>3c</td>
<td>shear graph selector</td>
<td>tab</td>
<td></td>
<td>change page in notebook to show respective result</td>
<td></td>
</tr>
<tr>
<td>3d</td>
<td>pile position axis</td>
<td>QCP</td>
<td></td>
<td>measured from top down</td>
<td></td>
</tr>
<tr>
<td>3e</td>
<td>result value axis</td>
<td>adjust to max value</td>
<td>QCP</td>
<td>allow to zoom in/out</td>
<td></td>
</tr>
</tbody>
</table>
EXERCISE #1: GUI DESIGN

- Design a UI for an application that collects a person’s information
  - First and last name
  - Address, city, state, ZIP
  - Date of birth

- Create a table listing each element

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Action</th>
<th>Widget</th>
<th>notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Text input</td>
<td>none</td>
<td>???</td>
<td>Check for valid name?</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Share with neighbor, discuss options, revise your design as appears useful
What is Qt?

- A framework to
  - Create platform-independent applications
    - Desktop: Windows, Mac, Linux
    - Mobile devices: iOS, Android
    - Cars, Medical devices, …
  - Provide a large number of very useful data representation classes

- IT IS NOT FREE !!!!
  - Free for OpenSource
  - Free for personal use
#include <QString.h>
QString mString;

**QString**

- A smart string object
- No worries about ‘\0’ (which is a pain even for experienced C-programmers, honestly)
- Has formatting tools
  ```
  mString = “this is process {} of {}”;  
  mString.arg(proc).arg(numProcs);
  ```
- Has Unicode support (Asian fonts, European fonts)
COMMON DATA CLASSES

- `QVector< TYPE >`
  - `QVector<double> array1;`
  - `QVector<double> *array2 = new QVector<double>();`
  - `QVector<QVector<double> *> array3;`

- `array1.append(42.0);`
- `int n = array2->size();`
- `array3[2] = new QVector<double>();`
COMMON DATA CLASSES

- **QList<TYPE>**
  - `QList<QString> stringList1;`
  - `QStringList stringList2;`

- Looping made simple:
  ```cpp
  #include <iostream.h>
  #include <QString.h>
  #include <QStringList.h>

  foreach (QString s, stringList1) {
    // do something with string s
    std::cout << s << std::endl;
  }
  ```
BUILDING THE GUI

■ **Option #1:**
  ■ Directly in code
  ■ Check out [http://zetcode.com/gui/qt5/](http://zetcode.com/gui/qt5/) *(THESE GUYS ROCK !)*

■ **Option #2:**
  ■ Using Qt Designer (built into Qt Creator)
  ■ Let’s switch and build your app together *(Live Demo)*
DEVELOPER TOOL FOR QT

Qt Creator
STARTING A NEW PROJECT

Choose a template:

- Projects
  - Application
    - Library
    - Other Project
    - Non-Qt Project
    - Import Project
- Files and Classes
  - C++
  - Modeling
  - Qt
  - GLSL
  - General
  - Java
  - Python

Qt Widgets Application

- Qt Console Application
- Qt Quick Application - Empty
- Qt Quick Application - Scroll
- Qt Quick Application - Stack
- Qt Quick Application - Swipe
- Qt Quick Application - Canvas 3D

Supports Platforms: Desktop
STARTING A NEW PROJECT

This wizard generates a Qt Widgets Application project. The application derives by default from QApplication and includes an empty widget.

Name: Example
Create in: /Users/pmackenz/Development/SimCenter/SimCenterBootcamp/Code/Qt
Use as default project location

Continue
STARTING A NEW PROJECT
STARTING A NEW PROJECT

Class Information

Specify basic information about the classes for which you want to generate skeleton source code files.

- **Class name:** MainWindow
- **Base class:** QMainWindow
- **Header file:**mainwindow.h
- **Source file:**mainwindow.cpp
- **Generate form:**
- **Form file:**mainwindow.ui
STARTING A NEW PROJECT
STARTING A NEW PROJECT
STARTING A NEW PROJECT

Step #1: only needed if UI changed

Step #2:
STARTING A NEW PROJECT

RUN!
EXERCISE #2: CREATING YOUR GUI

- Let’s return to your GUI design from Exercise #1

1. Create a new Qt Widget Application project using Qt Creator
2. Open Forms => MainWindow.ui
3. Create your GUI as close to your design as possible
4. Go through all the objects and assign them a more descriptive name like:
   - TB_firstName
   - CB_theState
   - Etc.
5. Run qmake, build the app, and run it

This one should be surprisingly easy 😊
QFrame *frame = new QFrame();
QFrame *frame2 = new QFrame(frame);

pos = (x, y), size = (dx, dy)
A SIMPLE APPLICATION
```cpp
QFrame *frame = new QFrame();
Qframe *frame2 = new QFrame(frame);
pos = (x, y), size = (dx, dy)
```

QFrame definition and positioning with size.
A SIMPLE APPLICATION USING LAYOUTS
LAYOUT MANAGEMENT

QFrame *frame = new QFrame();
Qwidget(frame)
QFrame *frame2 = new QFrame(frame);
pos = (x, y), size = (dx, dy)

1
2
...
A SIMPLE APPLICATION USING MULTIPLE LAYOUT OBJECTS
EXERCISE #3B: IN CASE THE ORIGINAL PROBLEM WAS TOO SIMPLE
EXERCISE #4: CREATING YOUR GUI

Let’s return to your GUI design from Exercise #2

1. BEFORE doing anything, think about layout for your app.
   - How do you want each field to line up?
   - How shall each field grow relative to each other?
   - How can you achieve that with the least of layouts?

2. Move on and implement your layout
   1. Select container object
   2. Right-click and select layout
   3. Choose the desired layout

This one is usually harder but VERY IMPORTANT
**SIGNALS AND SLOTS**

How does a GUI work?

- Create the graphics
  - Instance of QMainWindow
  - Add child widgets
    - QFrame
    - QPushButton
    - etc.
- Emit **signals** for events
- Connect **signals** to **slots**
- Run the Event loop
SIGNALS AND SLOTS
OPTION 1: OVERLOADING DEFAULT SLOTS

- Each Widget emits signals on specific events

**Signals**

<table>
<thead>
<tr>
<th>void</th>
<th>clicked(bool checked = false)</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>pressed()</td>
</tr>
<tr>
<td>void</td>
<td>released()</td>
</tr>
<tr>
<td>void</td>
<td>toggled(bool checked)</td>
</tr>
</tbody>
</table>

- 3 signals inherited from QWidget
- 2 signals inherited from QObject

- Each widget has a unique name

- Example:
  - Widget name: run_button

- Event clicked connects to default slot:
  - on_run_button_clicked()
  - You can overload that slot in your application

- Implementation made easy:
  - Qt Creator
    - Right click => go to slot => clicked
OPTION 2: CREATING YOUR OWN SLOTS

// app.h: definition
...
class MyClass {
...
private slots:
    void react_to_button_clicked();
}

// app.cpp: implementation

void MyApp::react_to_button_clicked() {
    // your response to button clicked
}

// app.cpp: constructor

void MyApp::MyApp(QObject *parent = 0) {
    ....
    // connect signal to slot
    connect(ui->myButton, SIGNAL(on_myButton_clicked()),
            this, SLOT(react_to_button_clicked()));
}
OPTION 2: CREATING YOUR OWN SLOTS

private slots:
   //
   // program controls
   //

   // menu actions
void on_actionExit_triggered();
void on_actionNew_triggered();
void on_actionSave_triggered();
void on_actionOpen_triggered();
void on_actionExport_to_OpenSees_triggered();
void on_actionReset_triggered();

/* ***** menu actions ***** */
void MainWindow::bn_actionExit_triggered()
{
   this->close();
}

/* ***** check box status changes ***** */
void MainWindow::on_chkBox_assume_rigid_cap_clicked(bool checked)
{
   assumeRigidFileHeadConnection = checked;
}

void CWE_file_manager::linkMainWindow(CWE_MainWindow *theMainWin)
{
   CWE_Super::linkMainWindow(theMainWin);
   if (!cwe_globals::get_CWE_Driver()->inOfflineMode())
   {
      ui->remoteTreeView->setModellink(theMainWin->getFileModel());
      QObject::connect(ui->remoteTreeView, SIGNAL(customContextMenuRequested(QPoint)), this, SLOT(customFileMenu(QPoint)));
      QObject::connect(cwe_globals::get_file_handle(), SIGNAL(fileOpDone(RequestState,QString)), this, SLOT(remoteOpDone(RequestState,QString)));
      QObject::connect(cwe_globals::get_file_handle(), SIGNAL(fileOpStarted()), this, SLOT(remoteOpStarted()));
      setControlsEnabled(true);
   }
DEBUGGING WITH SIGNALS AND SLOTS

- **Clean way**
  1. Set breakpoints at entries to slot implementation(s)
  2. Start (“run”) application
  3. Don’t stop at first occurrence but continue till app accepts new user input.

- **Brute-force method:**
  - Write debug output at start of slot implementation(s)

```cpp
#include <QDebug.h>

void MyClass::MySlot(int arg1) {
    qDebug() << “Entering MySlot”; // your code here
}
```
EXERCISE #5: ADDING CALLBACK FUNCTIONS

- Let’s add some functionality to your GUI
  1. Create a class method (function) that collects the information from the UI and stores it in a private structure like this one:

```
typedef struct {
    QString firstName;
    QString lastName;
    ...
} DATA;
```

  2. Create a slot that writes out a formatted address label
  3. Create a button (if you don’t have one yet) labeled “Print Address Label”
  4. Connect this button’s clicked signal to your slot
  5. Qmake => build => run
DESIGN CONSIDERATIONS

- **VIEW – CONTROLLER – DATA model**
  - **VIEW**
    - Visual parts, display classes
  - **CONTROLLER**
    - Registers user requests
    - Manages actions in analysis models
    - Controls flow of data
  - **DATA**
    - All kinds: text, floats, arrays, class objects, …
MODEL – VIEW CONCEPT

- QAbstractItemView
  - QTreeView
  - QTableView
  - QListView

- QAbstractItemModel
  - QAbstractItem

Connecting data and view:

The Display Widget

```
QTreeView mView;
QAbstractItemModel *model = new QAbstractItemModel();
mView.setModel(model);
```

Note: this is just a pointer to the model, NOT a copy.
USEFUL HELPER WIDGETS

- QDialog
  - QFileDialog
  - QMessageDialog
  - QColorDialog
  - QFontDialog
  - ...

- QDir … all the help you need dealing with paths across different platforms

- QDateTime … dealing with time formats, date formats, calculating number of days, elapsed time, time zones
EXERCISE #6: CREATE A NICE ADDRESS LABEL

1. Update your slot for `create_label_button_clicked()` (or add another one) such that is
   - Pops open a dialog showing a nicely formatted address label in a QTextBrowser widget