THE UNIVERSITY OF WESTERN ONTARIO

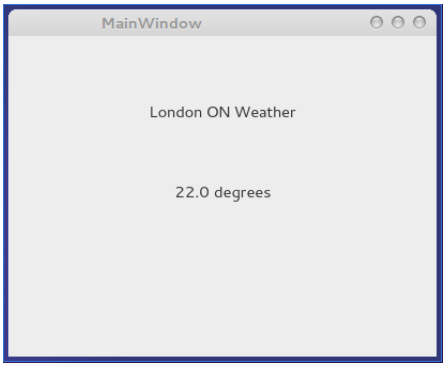
Computer Science 3307a - Fall 2013
Object-Oriented Design and Analysis

INDIVIDUAL ASSIGNMENT

Due October 4, 2013 at 11:59 PM

1 Assignment Overview

You will refactor and improve a basic weather program written in C++ using Qt. Currently, when the program opens, it queries a web service and retrieves the current temperature in London, Ontario and displays it.



In completing this assignment, you will become more familiar with:

- Creating and using classes in C++
- Using QtCreator to design software using Qt
- Using the designer to create GUI elements in Qt
- Interacting with a web service
- Applying design principles
- Applying the Model View Controller design pattern

2 The Web Service: Open Weather Map

The website `http://openweathermap.org/` provides free weather maps and forecast data for web and software applications. To query the data they host, you create a structured HTTP request, which is sent to the server.

`http://openweathermap.org/data/2.5/weather?APPID=11111111&q=London,can&units=metric`

NOTE: The APPID is an account ID for using the web service. This one is not real, and the above request will not go through. The APPID you will use is included in the provided code, or you can register for your own.

The request will return information in JSON, XML or HTML formats. We will be working with the JSON response, which is the default.

JSON is an open standard for describing data. For example, consider the following basic JSON object:

```
{
  "Name" :  "Susan MacMillan",
  "Age"  :   21
}
```

The object is denoted by the {} braces, and each element has a key (Name, Age) as well as a value (Susan MacMillan, 21 respectively), and they are separated by a comma. These are called key-value pairs.

Objects can be nested, as well as collected into arrays (using [ ] braces, separating each element with a comma). An example of a response we might get from the above query to the Open Weather Map service:

```
{
  "coord"   :{"lon":-81.246208190918, "lat":42.986888885498},
  "sys"     :{"country":"Canada", "sunrise":1379502533, "sunset":1379546922},
  "weather" :[{"id":801, "main":"Clouds", "description":"few clouds", "icon":"02d"}],
  "base"    :"gdps stations",
  "main"    :{"temp":294.15, "pressure":1020, "humidity":49,
              "temp_min":294.15, "temp_max":294.15},
  "wind"    :{"speed":4.1, "deg":190},
  "clouds"  :{"all":20},
  "dt"      :1379534400,
  "id"      :6058560,
  "name"    :"London",
  "cod"     :200
}
```

The program then parses this response and displays the `"temp"` element from the `"main"` object.

3 Provided Code

The provided code is an entire QtCreator project. Including the source code and the UI description file. It implements a very simple interaction with the Open Weather Map web service.

To develop for this assignment, it is assumed that you are working with the machines in mc10, or a copy of the VMware virtual machine configured in the same way. No support will be provided for setting up your own machine with QtCreator and C++.

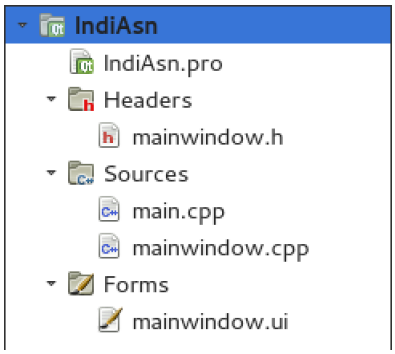
3.1 Opening the project and exploring the code

To open the project, either

- Open QtCreator from the Applications> Programming> QtCreator,
  then select File > Open File or Project> Select the .pro file.
- Right click on the .pro file, and select “Open with QtCreator"

The first time you open the project you will asked to select build directories. These can be changed later.

Under the Edit tab you will be able to see the file listing for the project.



- `main.cpp` Contains the main method. This does little more that initiate the main QApplication, which is common in Qt projects.
- `mainwindow.h` Contains the declaration for the main window. Note the public slot function `replyFinished` that is declared.
- `mainwindow.cpp` Contains the implementation of the functions declared in mainwindow.h. The constructor performs the request made to the web service, and the slot function processes the response.
- `mainwindow.ui` This file contains the UI declaration. To modify how the GUI looks, you can use the designer included with QtCreator. To open this, double-click on the mainwindow.ui file.

3.2 Run the project

To run the project, simply press the green play button in the bottom left of the QtCreator window. This will bring up the main window and display the current temperature.

3.3 A Note on Signals and Slots

You may already be familiar with this concept from other GUI API's. The idea is that functions can emit signals, which can then be received by slot functions. To connect a signal to a slot, we use the `QObject::connect` function. An example of this can be found in the constructor of MainWindow. Notice that both the signal and the slot are associated with specific objects ("manager" and "this"). Once connected, any time that signal is emmited from that object, the slot function will be called on the other object.

Many of the GUI elements will emit signals when the user interacts with them. To respond to these actions, you can define more slot functions that will handle those user signals. Signals and slots can be managed in the code with the `QObject::connect` function, or through the Design>Signals and Slots Editor.

4 Assignment Tasks

This section describes the tasks associated with grades on the assignment. There are marks associated with each task, so a good strategy would be to complete them as units, rather than trying to implement them all at once, ensuring you have a working program at each step. That way, if you are unable to complete the assignment for any reason you will have a gradable product. If your program will not run, the maximum grade you can receive is 50%, depending on how much is there.

4.1 Required Tasks

These tasks are the basic tasks you must complete. By completing these you are eligible for a grade of 60% on the assignment.

4.1.1 The Model-View-Controller Design Pattern

The design of this project is quite bad. It is an example of the "God Class" anti-design pattern, where one class is responsible for almost all aspects of the software. To improve the design you will apply the Model-View-Controller (MVC) design pattern.

The MVC pattern describes the separation of the following elements of a software system:

- Model: The data
- View: The interface (in this case, our GUI elements)
- Controller: The application logic/functionality

The pattern describes that these elements be decoupled. In a typical implementation, the View will initiate signals based on the user's actions, and the controller will dispatch these as appropriate to update the data, or access the data to update the view.

By virtue of the way QtCreator implements our GUI elements, these are already separately described in the mainwindow.ui file, and handled by qmake. However, the data and application logic are intertwined. In our case, the data is actually from a remote source. The communication with this data source is independent of the application logic, and should be encapsulated.

4.1.2 Add a "Get Weather" button

As the program is implemented now, the weather is only retrieved when the program starts. In this task, you will add a button to the GUI, which when pressed, will get the current weather by querying the web service again.

4.2 Optional Tasks

There are a number of ways that the program can be improved. The following is a list of tasks that you can combine in any way that you wish. Each tasks is worth an additional 10% of the assignment grade, to a maximum of 110%.

You must not violate the Model-View-Controller pattern with your additions. You are encouraged to add additional classes if they become necessary, and should encapsulate elements according to basic object oriented design principles.

4.2.1 Add more detail to the weather report

Add additional information to the users display, including at least 5 additional elements from the JSON object returned from the web service that are subject to change (sunrise, sunset, current condition description, pressure, humidity, minimum temperature, maximum temperature, wind speed, wind direction).

4.2.2 Improve the appearance

Give the program a unique, more professional look by choosing custom colours, fonts and font sizes to improve the appearance. You should take care in laying out elements so the information is easy to read, and the program is intuitive to use. You must also include at least one image, as well as include a credit statement to OpenWeatherMap.org.

4.2.3 Add a Celsius, Fahrenheit and Kelvin display option

Instead of only having the temperature come up as celsius, add a radio button or drop down menu that will allow the user to change the units of the temperature displayed. The current units displayed should be obvious to the user. Note that this should update all temperatures displayed. A change to the GUI element should immediately update the weather report, without requiring the user to press "Get Weather".

4.2.4 Make additional cities available

Add at least five cities that the user can choose to view the weather for. You can implement this with whichever GUI elements you wish, as long as the functionality is correct. You should take a look at the std::vector or QVector classes when implementing this. The available cities should NOT exist only in the GUI.

4.2.5 Allow the user to Add/Remove cities from the cities available

Note that this is related to the task above. Allow the user to add additional cities to the list of cities available. When adding a city, a test should be performed to ensure the city can be accessed, and only added if this test succeeds (for simplicity, you do NOT need to provide an error message). The available cities do NOT need to persist between runs of the program.

4.2.6 Multi-Day Forecast

Add a daily forecast for at least the next three days (not including today). You will need to use a different base URL, where "weather?" is replaced by "forecast/daily?". See API for details.

`http://bugs.openweathermap.org/projects/api/wiki/Api_2_5_forecast`

Note: the “dt" field in the JSON format specifies the day/time in the Unix time format. There should be a clear indication of the date for each forecast, and it must include at minimum the "day" temperature for each date. The forecast must update whenever the main weather updates.

4.2.7 Data persistance

Have all optional elements saved between runs of the program. You can implement this with a configuration text file, or try your hand at serialization, with QDataStream. The data should be automatically loaded when your program starts, and automatically saved when the program exits.

5 Coding Guidelines

Your code should be written in adherence to the Coding Guidelines available on the course website. Deviations will result in deductions from your assignment grade up to 10%.

6 Submission Instructions

You will be submitting your code to your private repository on GitHub. Detailed submission instructions can be found on the course website. Deviations from correct submission will result in deductions from your assignment grade up to 10%.