Earthquake Data – Phase 2

Members – Group 6

Jamieson Allare – Documentation, Repository, and Google Map API

Jeff Tran – Testing and Generating Reports

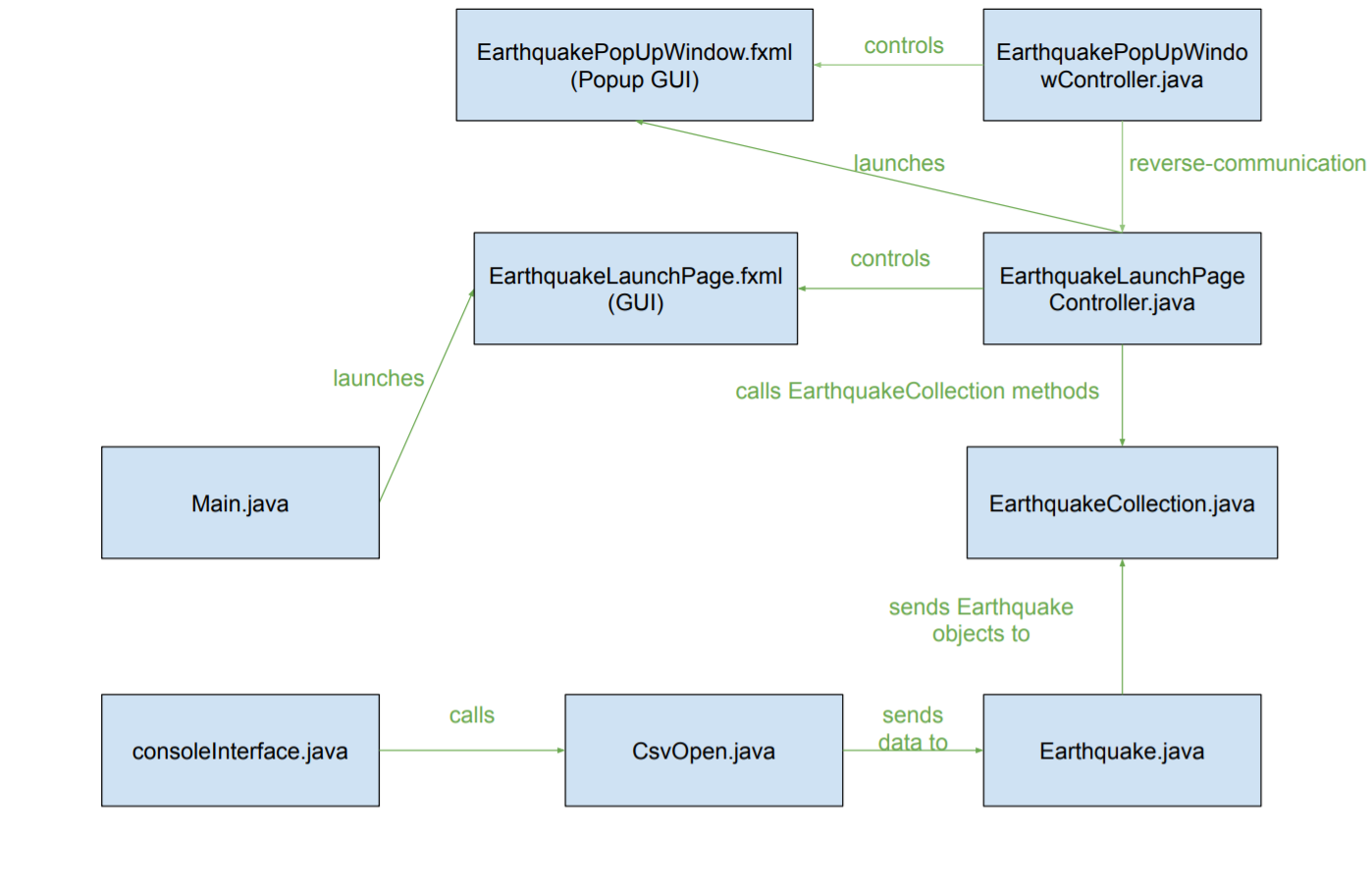
Laura Buckles – Architecture and GUI Layout

Our Goal

Our goal was to add in a functional GUI to the earthquake data explorer. This was accomplished in three phases:

* Phase 1: Prepare Team 2’s code for the project.
* Phase 2: Create the GUI layout.
* Phase 3: Add functionality to GUI and Map. Then conduct testing.

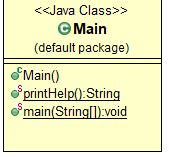
Project Architecture



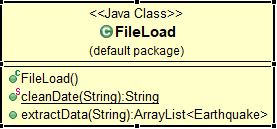
When Main is run, the actions (colored green) are executed. First Main launches the EarthquakeLaunchPage.fxml. This is the main GUI interface, which is controlled by the EarthquakeLaunchPageController class. The EarthquakeLaunchPageController will launch the EarthquakePopUpWindow.fxml when the user wants save data to a file. This GUI is controlled by the EarthquakePopUpWindowController file. When the user wants to filter data through the GUI choice box, the EarthquakeLaunchPageController will call methods in the EarthquakeCollection class to send appropriate data back to a GUI text field. The EarthquakeCollection stores all of he Earthquake objects that are constructed in the Earthquake class. Earthquake.java is sent raw data from the CsvOpen class when the consoleInterface.java calls it.

Project UML Class Diagrams

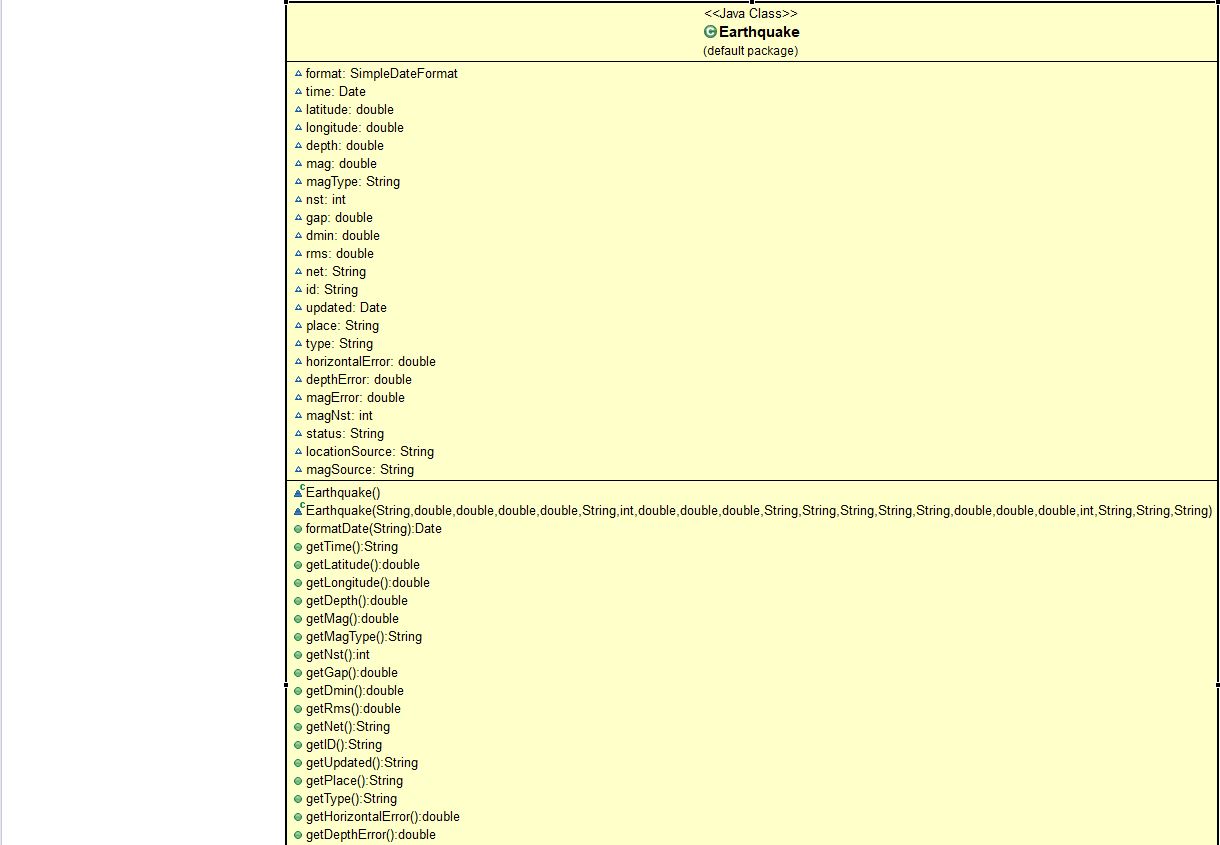
**Main.java:**

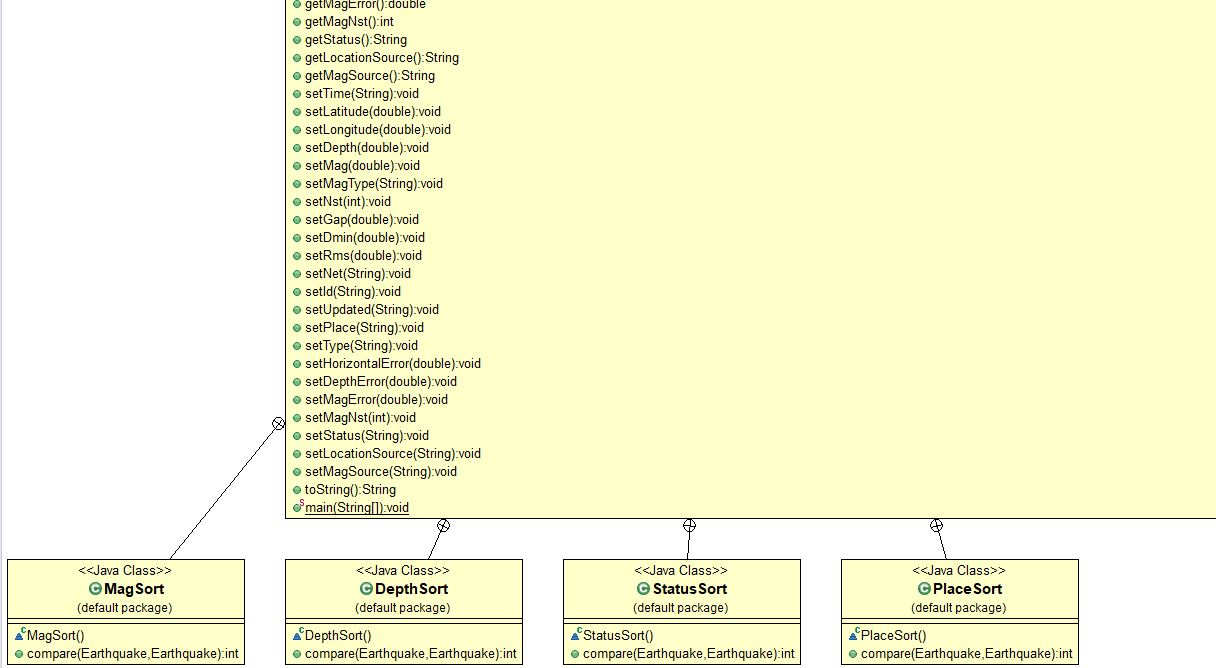


**FileLoad.java:**

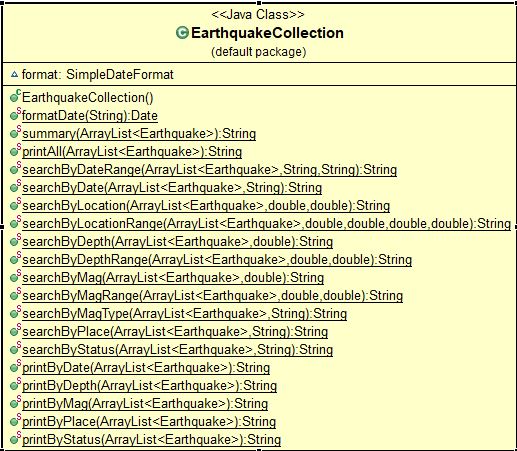


**Earthquake.java:**





**EarthquakeCollection.java:**



Testing

Console Interface:

In Console Interface, there was only one function to test with JUnit. There was a function that took a word and added dashes to the front and back of it to make it look like an item on a menu. For the user inputs, we manually testing everything to see what users such might as misspell a word. To get around this issue, we checked if the user input was part of our string array of the different functions available. If it wasn’t part of the list, the user would see an invalid message and the program would ask for another function. When a user selects a function, he or she can also give the wrong type of data to it. To get around this, we have implemented a system where it would keep asking for a valid data type until it is correct. If a user gives wrong information, it would repeat the process and ask for another value.

Csv Open:

In Csv Open, there was only one function to test as well. This class’s job was to parse the data out of the CSV file of earthquakes correctly and make objects of each instance. To check if this worked correctly, we conducted a JUnit test seeing if the array list had more than 0 items in it. If it had more, we know we completed the task successfully. There was also another function in this class that the other team had implemented from an online source. The function’s job was to see if there were any special characters between strings and to ignore it. We did not conduct testing on this because it had been done online when we checked out the source.

Earthquake:

In the Earthquake Class, the previous team did not have getter and setter methods, so we had to implement this. To test out how they worked, we created a random earthquake and returned the data to see if it matched what we had wanted. Another thing we had to test out is how to sort the data when users ask for it. We looked for an outside source to learn how to do so. In our Junit testing we created sample Earthquake objects to use these sorting functions. If the comparison turned out to be equal, 0 is returned. If something was less than the other, a -1 was returned. Anything greater than the other returned 1.

Earthquake Collection:

In the Earthquake Collection Class, many functions returned large strings. To test out if each function worked properly, we compare the outputs with text files. Test text files had all the data that should be returned when the function ran. First we read the text file and created a string out of it. Afterwards, we compared the outputs of the function to the string we created and found it to be successful. We got the idea of using text files from doing research online because we did not want to type out the string variable on Java. That would have taken too much time.

Project Timeline

* Day one (4/9/19)
  + Receive project assignments
  + Fork repo
  + Analyze and test code
* Day two (4/11/19)
  + Fix file load
  + Create GUI layout
  + Research map API
* Day three (4/12/19)
  + Reach out for jar file for map
  + Add comments and clean up CSVloader class
  + Add functionality to GUI
* Day four (4/16/19)
  + Get map into GUI
  + Finish fixing console interface
* Day five (4/18/19)
  + Continue fixing console interface
  + Add functionality to GUI
  + Fix map API size
* Day six (4/19/19)
  + Continue working on GUI functionality
    - Text field in GUI for functions with choice box filter
  + Fix "This page cannot load Google Maps correctly"
  + To do:
    - Move map code from main to controller
    - Fix console functions
    - File output (printwriter)
* Day seven (4/23/19)
  + Move Map API to controller
  + Conduct last unit test
  + Refine code with (try and except)
* Day eight (4/24/19)
  + Map markers
  + Clean up code
  + Finish testing and documentation
* Day nine (4/26)
  + Present
  + Finish documentation

Unresolved Issues/Bugs

None.

Link to Presentation on Kaltura:

https://iu.mediaspace.kaltura.com/media/EarthquakeDataDemo/1\_hap9k414