MapMe

Writing a concurrent web application using J2EE

Omar Farooq

David Shrestha

Table of Contents

Problem definition 1

Program design 1

Test design and analysis 2

Program listings 9

References 10

**Problem definition**

JavaEE is used widely in the industry for web applications. It is used for enterprise application that needs to be multi-threaded or scaled. We wanted to make an application in Java EE and experiment with some of it’s features. Some of the technologies that our application(MapMe) uses are JSF, GlassFish server and JSP. We learned that our web application would be using a Model View Control (MVC) architecture when using Java EE.

**Program design**

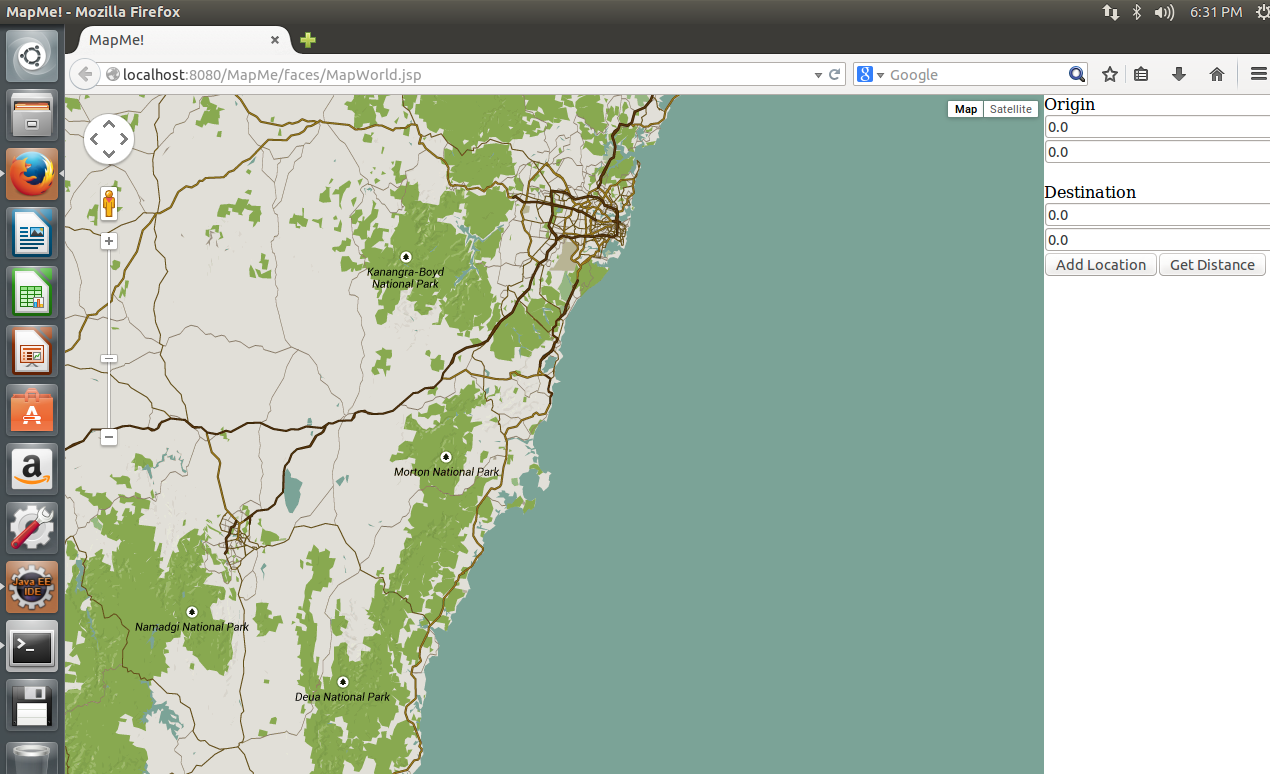
We used JSP as our front-end and Glassfish was used to host our application. The back-end was handled by Java EE. In terms of the MVC pattern, a Model is an object that represents data. In our application, the MapMe.java file has getters and setters that hold the instance of the application. The View is a form of visualization shown to the user. We are doing this through jsp and html in the file MapWorld.jsp. Our Controller was in MapMe.java that called the started our worker threads which called the computation functions. Our View which is the MapWorld.jsp then gets the output of the Controller. Our web application starts by the user typing in an origin and a destination location (latitude and longitude). The application then uses JSP to send the data to Java where it calculates the distance between the origin and the destination using the haversine formula. We are using threads as workers to do the haversine calculation concurrently. The Map API is provided by google maps.

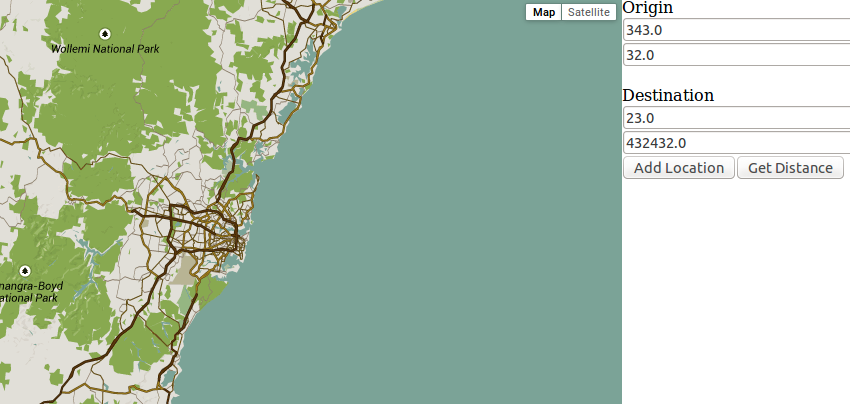
We tried to experiment with our concurrent web application by using synchronized vs. unsynchronized threads in the hopes of making our web-application faster. We tried to do this because we found that there are different ways to do concurrent programming that can result to the same output. With some programs that use Threads, they need a lot of synchronization. What that means is that the threads need to access shared resources (e.g. memory, or variable) a lot. The thread that needs the resource has to block all the other threads while it is running it’s critical section. On the other hand, there are programs that can be implemented where threads don’t have to be synchronized. This is where multithreading can shine because threads don’t have to wait for each other. With real programs however, it will be a mix of both styles where you need to synchronize threads for some parts as well as make them run a computation without any critical section. We’re trying to compare the effect of using mutex on a concurrent program where the mutex is not necessary.

The way we implemented the above was by timing the program execution with and without using the synchronization method to our haversine function. However, we were not able to find a consistent result. So we could not test our hypothesis that unsynchronized calculation of our program would be faster than the synchronized version because of the mutual exclusion locks it uses.

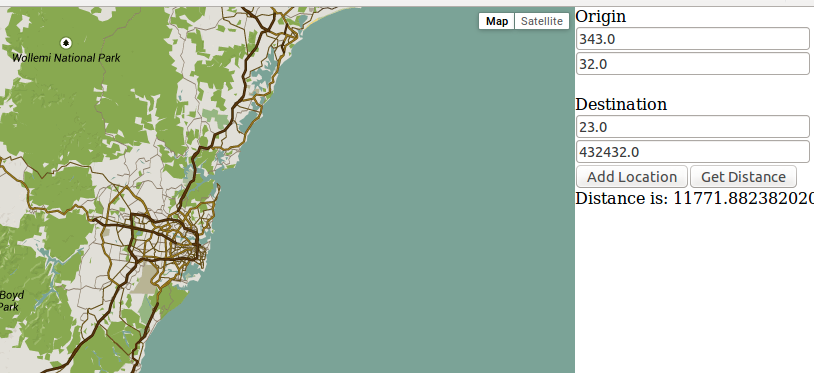
**Test design and Analysis**

When we were verifying the distance between two locations, we used google’s official distance calculator and compared it with our own implementation which used the haversine formula. To display the distances stored in an ArrayList, we had to learn a way to use for loops inside jsp. JSTL provided a way to use for loop to display our ArrayList structure as well.

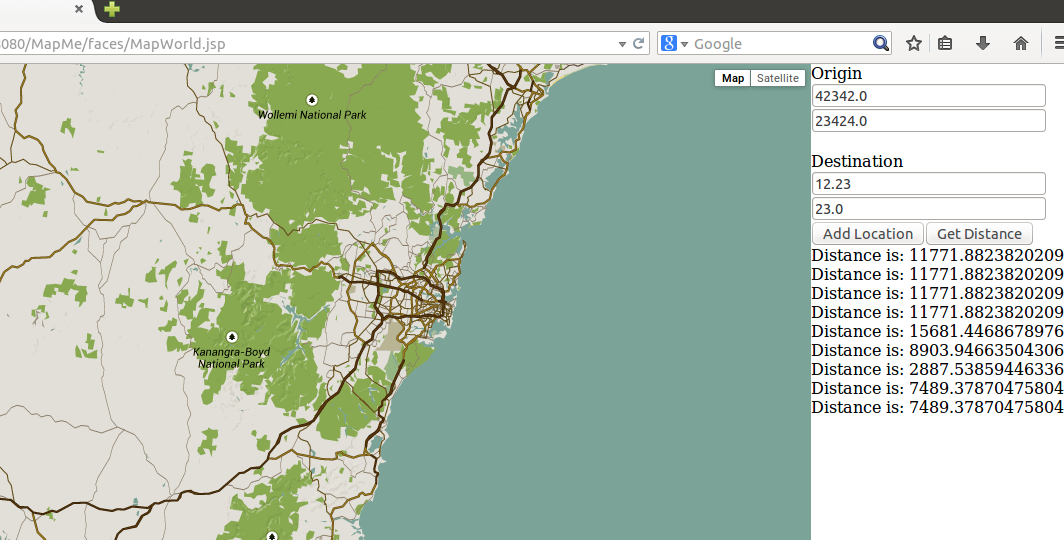




Click, ‘Add location’ to add this location onto our application. Then ‘Get Distance’ to concurrently get all of the results of all of the locations which have been added.



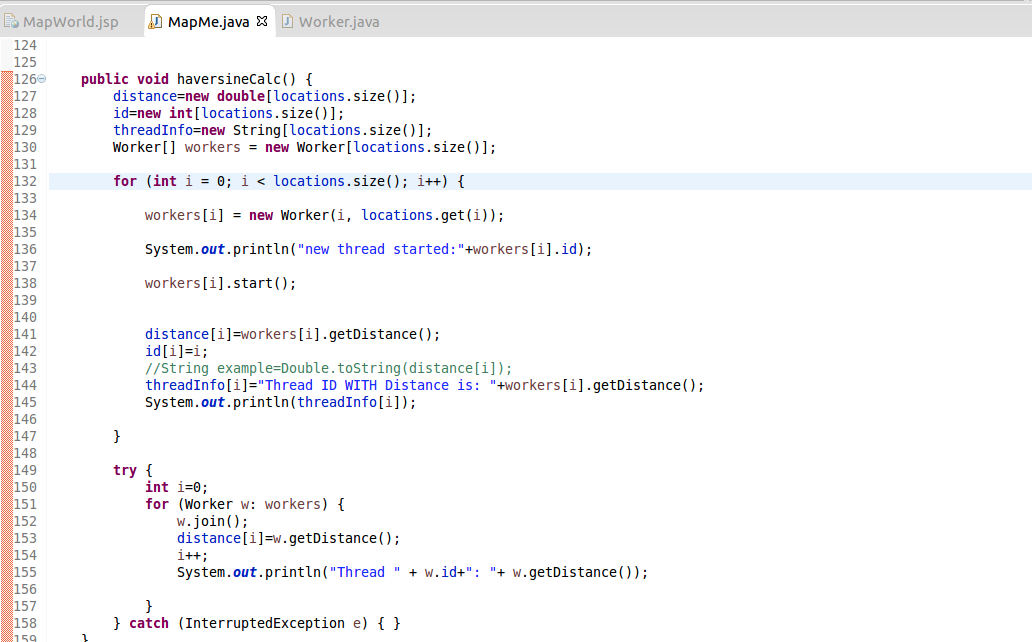
Above is a photo of just one location added



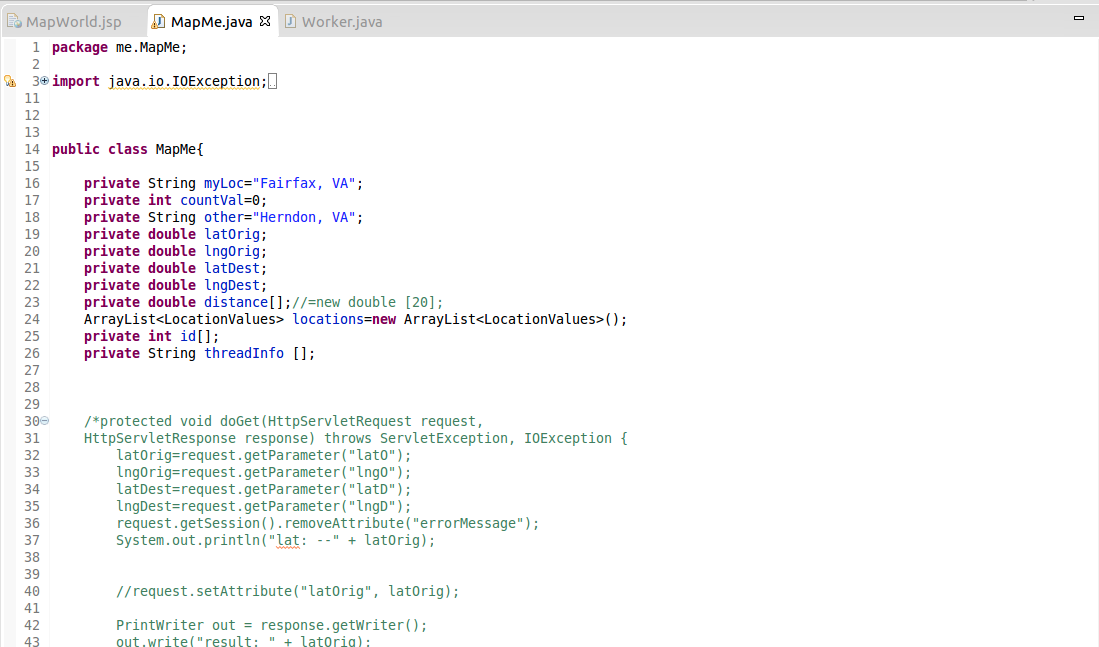
Above is a photo of multiple locations which have been added, and then calculated concurrently



Above is a control function which is the thread, that calculates the haversine distance



Once ‘Get Distance is clicked’, function haversineCalc() will run





Simple getters and setters (Model)

**Program listings**

Required external libraries: GlassFish package for eclipse.

Java version: 7

Our project MapMe consists of these files:

MapMe.java - This is the backend that handles the creation of the threads and calls the haversine calculation in Worker

Worker.java - This is the worker thread file that has actually does the calculation.

MapWorld.jsp - This is the front-end file that lets user interact with it and also communicates with the back-end file(MapMe.java).

References

Java EE at Glance, Oracle, Retrieved 4th May, 2015.

http://www.oracle.com/technetwork/java/javaee/overview/index.html

Java Servlet Faces, oracle, Retrieved 4th May, 2015.

http://docs.oracle.com/javaee/6/tutorial/doc/bnaph.html