Introduction

This document is meant to provide you with suggestions when working on your project. Some of these suggestions are project specific, but many have been taken from sources like Effective Java, and are widely applicable and will hopefully serve you well beyond this course.

1 Know How to Use the Debugger

Debuggers are powerful tools that greatly facilitate understanding and troubleshooting code. If you aren't already comfortable using the Eclipse debugger, you should look at this tutorial (http://www.vogella.com/articles/EclipseDebugging/article.html). For those of you who are already comfortable you may still want to glance at it anyway for some of the more advanced features such as:

1. Conditional Breakpoints – Breakpoints that are only activated when some other condition is true, for instance if you need to debug a loop, you can have the breakpoint trigger only on the 140th iteration of the loop.
2. Exceptional Breakpoints – You can define a breakpoint to activate whenever a particular type of Exception is thrown, or only when it is uncaught.
3. Watchpoints – You can define a breakpoint to activate when a particular field is accessed and/or modified.
4. Drop to Frame – You can partially restart your applications execution at some point in the call stack. For instance you could restart the existing method to see the buggy behavior happen again.

While many problems can be solved with strategic use of System.out.println(), few people who know how to use a debugger opt out of using it and learning how to use it will pay dividends not just over time, but even in this project.

2 Exceptions & Stack Traces

As you are writing code that runs within the GWT Framework the stack traces may look very different than what you are used to and may span many pages. When diagnosing exceptions, the first thing to do is to find out how the exception relates to your code.

The best way to do this is to look for your root package name, in this case it's **ca.ubc.cpsc310.exceptionTest** and it only occurs on one line of the approximately 60 lines of output:

```
com.google.gwt.user.server.rpc.UnexpectedException: Service method 'public abstract java.lang.String
ca.ubc.cpsc310.exceptionTest.client.GreetingService.greetServer(java.lang.String) throws
java.lang.IllegalArgumentException' threw an unexpected exception: java.lang.NullPointerException
	at com.google.gwt.user.server.rpc.RPC.encodeResponseForFailure(RPC.java:389)
	at com.google.gwt.user.server.rpc.RPC.invokeAndEncodeResponse(RPC.java:579)
	at
com.google.gwt.user.server.rpc.RemoteServiceServlet.processCall(RemoteServiceServlet.java:208)
	at
com.google.gwt.user.server.rpc.RemoteServiceServlet.processPost(RemoteServiceServlet.java:248)
	at
com.google.gwt.user.server.rpc.AbstractRemoteServiceServlet.doPost(AbstractRemoteServiceServlet.java:62
)
	at javax.servlet.http.HttpServlet.service(HttpServlet.java:637)
	at javax.servlet.http.HttpServlet.service(HttpServlet.java:717)
```

```
        at org.mortbay.jetty.servlet.ServletHolder.handle(ServletHolder.java:511)
        at org.mortbay.jetty.servlet.ServletHandler$CachedChain.doFilter(ServletHandler.java:1166)
        at com.google.appengine.api.socket.dev.DevSocketFilter.doFilter(DevSocketFilter.java:74)
        at org.mortbay.jetty.servlet.ServletHandler$CachedChain.doFilter(ServletHandler.java:1157)
        at 
com.google.appengine.tools.development.ResponseRewriterFilter.doFilter(ResponseRewriterFilter.java:123)
        at org.mortbay.jetty.servlet.ServletHandler$CachedChain.doFilter(ServletHandler.java:1157)
        at 
com.google.appengine.tools.development.HeaderVerificationFilter.doFilter(HeaderVerificationFilter.java:
34)
        at org.mortbay.jetty.servlet.ServletHandler$CachedChain.doFilter(ServletHandler.java:1157)
        at com.google.appengine.api.blobstore.dev.ServeBlobFilter.doFilter(ServeBlobFilter.java:61)
        at org.mortbay.jetty.servlet.ServletHandler$CachedChain.doFilter(ServletHandler.java:1157)
        at 
com.google.apphosting.utils.servlet.TransactionCleanupFilter.doFilter(TransactionCleanupFilter.java:43)
        at org.mortbay.jetty.servlet.ServletHandler$CachedChain.doFilter(ServletHandler.java:1157)
        at com.google.appengine.tools.development.StaticFileFilter.doFilter(StaticFileFilter.java:125)
        at org.mortbay.jetty.servlet.ServletHandler$CachedChain.doFilter(ServletHandler.java:1157)
        at 
com.google.appengine.tools.development.BackendServersFilter.doFilter(BackendServersFilter.java:97)
        at org.mortbay.jetty.servlet.ServletHandler$CachedChain.doFilter(ServletHandler.java:1157)
        at org.mortbay.jetty.servlet.ServletHandler.handle(ServletHandler.java:388)
        at org.mortbay.jetty.security.SecurityHandler.handle(SecurityHandler.java:216)
        at org.mortbay.jetty.servlet.SessionHandler.handle(SessionHandler.java:182)
        at org.mortbay.jetty.handler.ContextHandler.handle(ContextHandler.java:765)
        at org.mortbay.jetty.webapp.WebAppContext.handle(WebAppContext.java:418)
        at 
com.google.appengine.tools.development.DevAppEngineWebAppContext.handle(DevAppEngineWebAppContext.java:
94)
        at org.mortbay.jetty.handler.HandlerWrapper.handle(HandlerWrapper.java:152)
        at 
com.google.appengine.tools.development.JettyContainerService$ApiProxyHandler.handle(JettyContainerServi
ce.java:383)
        at org.mortbay.jetty.handler.HandlerWrapper.handle(HandlerWrapper.java:152)
        at org.mortbay.jetty.Server.handle(Server.java:326)
        at org.mortbay.jetty.HttpConnection.handleRequest(HttpConnection.java:542)
        at org.mortbay.jetty.HttpConnection$RequestHandler.content(HttpConnection.java:938)
        at org.mortbay.jetty.HttpParser.parseNext(HttpParser.java:755)
        at org.mortbay.jetty.HttpParser.parseAvailable(HttpParser.java:218)
        at org.mortbay.jetty.HttpConnection.handle(HttpConnection.java:404)
        at org.mortbay.io.nio.SelectChannelEndPoint.run(SelectChannelEndPoint.java:409)
        at org.mortbay.thread.QueuedThreadPool$PoolThread.run(QueuedThreadPool.java:582)
Caused by: java.lang.NullPointerException
        at 
ca.ubc.cpsc310.exceptionTest.server.GreetingServiceImpl.greetServer(GreetingServiceImpl
.java:33)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:601)
        at com.google.appengine.tools.development.agent.runtime.Runtime.invoke(Runtime.java:115)
        at com.google.gwt.user.server.rpc.RPC.invokeAndEncodeResponse(RPC.java:561)
        ... 38 more
```

Exceptions such as the above, are nested. Nested exceptions occur when an exception thrown by one method is caught, and then wrapped by another exception and then thrown again. The reason for doing this is generally to make it easier on the caller. In the above example, the client probably does not care what specific problem occurred and only cares that one did, and so it receives an UnexpectedException. The wrapping of exceptions occurs with code blocks such as these:

```java
} catch(RuntimeException e) {
      throw new UnexpectedException("Unexpected exception received", e);
}
```

3 Use an Uncaught Exception Handler

On the client consider setting an Uncaught Exception handler like the following:

```
/**
 * This is the entry point method.
 */
public void onModuleLoad() {

      if(GWT.isProdMode())
      {
      GWT.setUncaughtExceptionHandler(new UncaughtExceptionHandler() {

            @Override
            public void onUncaughtException(Throwable e) {
                  Window.alert("An Uncaught Exception has occured : " +
                                          e.toString());
            }

            });
      }
```

If an exception that you do not catch occurs (such as a RuntimeException) in deployed mode it will silently notify the browser's internal JavaScript debugger. Most users will not notice this, and it will look as though your application has just broken randomly. By using the above code snippet it will be very obvious that an exception has occurred. You can also do most fancy exception processing, the above just lets you know that it has occurred.

The GWT.isProdMode() is not necessary, this check means it will only install the UncaughtExceptionHandler if the application is deployed. Depending on what your handler does you may also want it to occur in development mode.

For more information about this see: http://www.summa-tech.com/blog/2012/06/11/7-tips-for-exception-handling-in-gwt/.

Java also supports setting Uncaught Exception Handlers, but you can not use this functionality when running a Servlet for security reasons. In your other applications it may be handy to know, and so the the following javadoc tells you how to set them in general: http://docs.oracle.com/javase/1.5.0/docs/api/java/lang/Thread.html#setDefaultUncaughtExceptionHandler(java.lang.Thread.UncaughtExceptionHandler)

4 Do Not Return Null Values

Returning a null value to the caller of a method requires that caller to have to deal with a special case every time, this can cause the code to be cluttered with unnecessary details. Depending on what your method returns, there are two general strategies for this:

1. If your method returns a collection, (e.g. A Set, List or Map), return an empty collection, (for instance by calling Collection.emptyMap()).

2. If your method returns a single object, consider using a *Special Case Object*[1], that logically represents the null value but can supply intelligent defaults

Consider for instance an e-commerce website that allows users to browse products both when they are logged in and not and supports a wish list for users:

```
User u = getLoggedInUser();
if(u == null) {
      displayLanguage(ENGLISH);
      displayGreetingWithName("Guest");
      displayWishListSize(0);
      displayShoppingCartSize(0);
} else      {
      displayLanguage(u.getLanguage());
      displayGreetingWithName(u.getName());
      if(u.getWishList() != null)   {
            displayWishListSize(u.getWishList().size());
      } else {
            displayWishListSize(0);
      }

      if(u.getShoppingCart() != null)     {
            displayShoppingCartSize(u.getShoppingCart().size());
      } else {
            displayShoppingCartSize(0);
      }
}
```

If we make sure that getWishList and getShoppingCart return empty lists instead of null values, we can simplify the case when a user is logged in to:

```
User u = getLoggedInUser();

displayLanguage(u.getLanguage());
displayGreetingWithName(u.getName());
displayWishListSize(u.getWishList().size());
displayShoppingCartSize(u.getShoppingCart().size());
```

Furthermore if we modify getLoggedInUser() to simply return the following user instead of null when no one is logged in, we get the exact same functionality without any complicated control flow.

```
public class GuestUser extends User {


      public String getName() {
            return "Guest";
      }

      public List<Object> getWishList() {
            return Collections.emptyList();
      }
```

1 http://en.wikipedia.org/wiki/Special_case_pattern

```
	public String getLanguage() {
		return ENGLISH;
	}

	public List<Object> getShoppingCart() {
		return Collections.emptyList();
	}

}
```

5 Override toString()[2]

The toString() method is invoked anytime an object is to be converted to a string. This occurs most often during debugging, but can also happen at other times. Consider the User class in the previous example, without a toString() method, when one inspects a List of them you'll see something like the following:

```
[foo.User@45e6612c, foo.User@17f5b38e, foo.GuestUser@3df3bec, foo.User@34d704f0]
```

An appropriate toString() method can make the output much more useful and greatly simplify debugging, the following is an appropriate toString() method:

```
public String toString() {
	return getName() + " (" + getLanguage() +")";
}
```

Using the above toString() method, examining the objects in a debugger or on the console, is now much easier:

```
[Mariano (ES), François (FR), Guest (EN), Angela (DE)]
```

6 Be Mindful of Concurrency

Concurrency occurs when two parts of your program are doing something at the same time. Every AJAX request is a concurrent process and the portion of your application running in the browser executes these concurrently with the rest of your code. Additionally the Servlet it communicates with on the other end is processing many requests concurrently. If you are not careful your code may be susceptible to race conditions[3]:

"A *race condition* occurs when the correctness of a computation depends on the relative timing or interleaving of multiple threads by the runtime[4]".

Race conditions are annoying to deal with because they may only appear in some circumstances and may be incredibly hard to track down. For instance in our experience examples with race conditions such as the following almost never manifest themselves when running locally, only when the application is deployed does it silently break.

2 Adapted from Item 10, of Effective Java.
3 Race conditions are only one possible failure scenario in concurrent applications, but for the purposes of this discussion it is the only one we will talk about.
4 Java Concurrency in Practice, Brian Goetz, p 20.

```
public List<ChatUser> chatUserList = new ArrayList<ChatUser>();;

public void joinChatRoom(String chatroom) {

      // Assume chatService has been created successfully
      chatService.getUsers(chatroom, new AsyncCallback<ChatUser[]>() {

            public void onFailure(Throwable error) {
                  Window.alert("Error occured: " + error);
            }

            public void onSuccess(ChatUser[] users) {

                  for (ChatUser u : users) {
                        chatUserList.add(u);
                  }
            }
      });

      // Do some other stuff
      updateChatUserCount(chatUserList.size());
      updateChatUserList(chatUserList);
}
```

The race condition here occurs because the updateChatUserCount() and updateChatUserList() calls expect the chatUserList to be populated by the onSuccess() method. But the onSuccess() method is called concurrently with joinChatRoom(), and so may execute at any time.

In some cases it will work correctly, such as in Figure 1. Figure 2 shows one of many unlucky[5] cases where the code does not work correctly. In Figure 2, by the time the size() is checked, the chatService has not added elements to the list yet.

5 The case where all the items are added after was shown for simplicity. Other things that can happen is that it could happen between additions to the list, or it could happen while the list is in the middle of being modified and whatever loop invariants are no longer true.

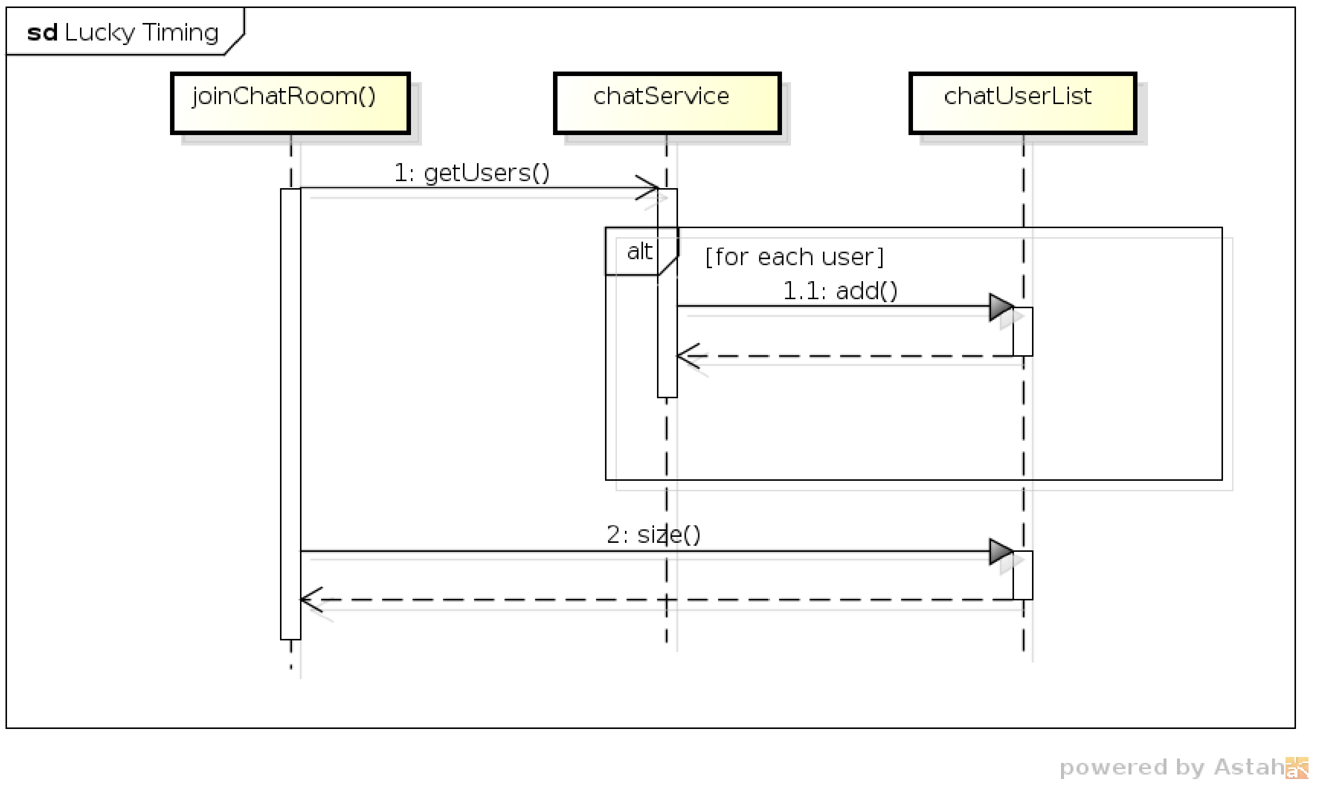

Figure 1: Lucky Timing Scenario

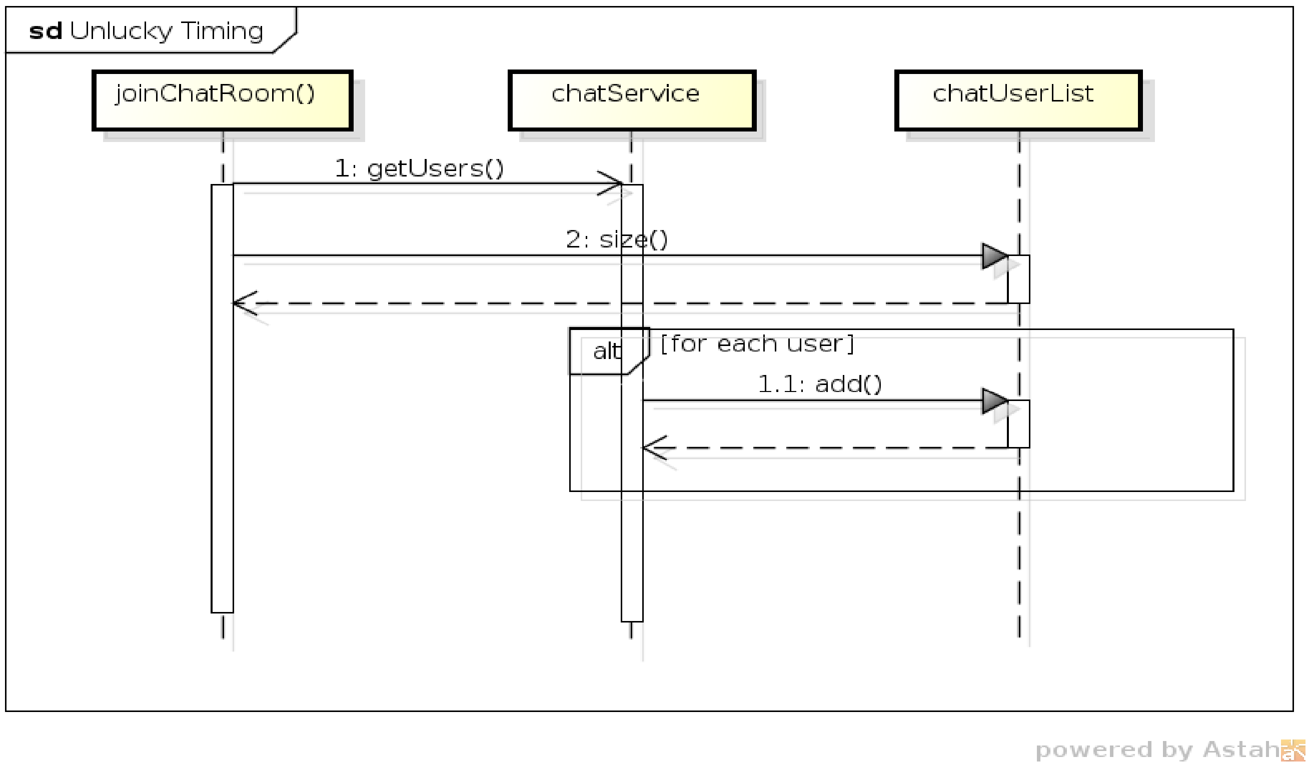

Figure 2: Unlucky Timing Scenario

Things can get really complicated really fast when dealing with concurrency, but thankfully there are two really easy ways to structure your program so that you actually don't have to worry about concurrency at all[6].

Rule 1

Any code on the client that depends upon the result of an asynchronous request to the server, should be executed as a result of the callback. In the above code this means putting the calls to updateChatCount() and updateChatUserList() in the onSuccess() method.

Rule 2

On the server, use the persistence store to store all data. This is related to the next comment, but essentially do not keep any state around in member variables between requests to your Servlet.

7 Avoid Member Variables In Your Servlets

A common mistake people make is putting user specific state information as a member variable to a Servlet. This will work properly in development mode but if you deploy and more than one user is using your application it will not work properly as there is only one object handling all the requests and the information will be overwritten. User specific information should either be put in the Persistence storage, or you can use this.getThreadLocalRequest().getSession(), this will return an object you can use like a Map<String,String>.

It is not only user specific state that can cause problems. Global variables (set statically or not) introduce concurrency related issues as they are accessed by multiple threads, and when deployed on Google App Engine there is no guarantee that the values won't disappear.

8 Do not add External Libraries to your Project

When you need to add a library to your project, say Google Maps, the easiest way to do this is to create a **lib/** folder in your project and then add this jar to the build path as a library. When you add an *External Library* you are linking to something that will not be placed in git. Consequently when your team mates update their projects, they will now have a broken project and will be stuck until they can find a copy of that jar. When they find it, they will probably remove your *External Library*, and add their copy as an *External Library.* When you pull their updated code, your project will now be broken in the same way, ad infinitum. Keeping all the dependencies in your project and therefore in git will make it much easier for your team to interact.

9 Do not commit broken code

You should not push broken code to the source code repository, especially if you broke code that was previously working. If you need to use source control for things that aren't read to be shared, commit them locally (ideally on a separate branch), and only integrate them with the project when it is working.

6 Some of you will no doubt be tempted to think about using a flag variable or a lock to fix the above example. In short almost none of these will work, even ignoring the fact that this code is compiled into JavaScript. Interested readers, you are strongly encouraged to read Java Concurrency in Practice, by Brian Goetz.

10 Eliminate Warnings

When you start your project you will have a project with 0 warnings, and you should strive to keep it that way. Most warnings are inert (such as the unused import exception), but occasionally warnings can alert you to a serious bug in your code that you would otherwise not see. If you let the warnings get out of hand, then you are likely to miss the serious problems in your code. Occasionally (like when dealing with Generics) you cannot get rid of a warning, and in these cases you should suppress them using the the @SuppressWarnings annotation, once you are confident the code is in fact correct.

11 Use Find Bugs

Find bugs (http://findbugs.sourceforge.net/) is a tool that analyzes your code and tries to detect bugs. It is not perfect but does a pretty good job at detecting many types of bugs. The installation is straight forward:

Installation Instructions: http://findbugs.cs.umd.edu/eclipse/

Once installed you can right click on your project and click Find Bugs → Find Bugs and it will add them to your Problems View.