Experience Report
Java Candidate Assignment

Rohan Honwade

March 8, 2015

1 Strategy Evolution

1.1 Rationale

After reading through the set of tasks for the assignment, I realized that the assignment was not only about testing a candidates coding skills, but also about checking his comfort level with tools used during development process - source versioning, package building and continuous integration. I first decided to have a

1. Sample code around which I would set up the tools
2. When this setup was completed, I would take up the coding task (calculator code) and complete

This strategy made sense to me because tools such as Maven would help me in automated searching of required jar files and also packaging them together. Jenkins would help me monitor every commit I pushed to Github did not break my builds. Developing in smaller chunks is useful for maintainability, reviewing and scrapping and Github's periodic commits would help me in this. It is for these reasons, I set out to have the development tools setup ready first.

1.2 How it evolved

Given that my Ubuntu system already had Maven and Git installed and my love for vim, I created a Maven sample app.

I then moved on to setting up the good old logging framework log4j. However, I realized that vim was not going to help me much with the auto-complete and debugging features that an IDE such as Eclipse would provide me. So I switched to Eclipse and had a basic log4j + Maven + Eclipse + Git integration complete. I did not have much issues in this part.

The next task was to setup Jenkins on an AWS instance, not just a local machine. Fortunately having a free tier with AWS, I created an Ubuntu instance and apt-get worked

like a charm as usual to help me install Jenkins with ease. I ensured the required ports on my AWS instance were open to accepting connections. I spent quite some frustrating time fiddling with getting Jenkins configurations, plugins, credentials and having it listen to Github commits, working. In the end, fixing the following issues helped me get the web hook working -

1. Although the public-private key pair had been created on my Jenkins machine, I realized the file owner was root and not jenkins user. So when checked for Github authentication, it failed. I deleted the dsa keys and redid the process. Also, provided the public key to Github (for password-less interaction) - a step I had missed out in my previous steps.

2. The webhook URL does not work unless we add a "/" in the end

3. Github should not push its notifications in the form of JSON. The notifications were accepted by Jenkins only when sent as a content type of x-www-form-urlencoded

A trial and error with the above helped me have a working communication between Jenkins and Github.

The last part was writing the code for the calculator; I manually solved the problem on paper and coded up the part to make the operations work. In the interest of time, I did not take up coding the 'let' command of the calculator. However, I added an exhaustive enough set of tests to test the code I had written. A small issue I faced while writing the tests was verifying console output made by my code using JUnit, but was able to solve this. Also, I shifted from JUnit version 3 to 4 during the process because JUnit 4 provides a simpler interface to test Exceptions and my testing was Exception heavy.

When this was ready and I exported my jar file and ran it, I realized that my dependency jars were not exported along with my jar. Also, Maven was using JDK 1.3 for compiling my Java code and as a result did not recognize Java generics code. I fixed these issues as well by making suitable changes in my pom.xml.

In the end, I also checked if my setup worked cross-platform, both on Windows machine as well as a Linux based machine. I fixed an issue related to the way my code expected its inputs and now the applications works fine on both the platforms. Maven was really helpful in this regard as we really don't have to worry about downloading the jars and adding them to the classpath. It is all handled behind the scenes by Maven.

I enjoyed the whole process.

2 Timeline

Serial No	Task	Time taken (approx)
1	Maven + Git setup on Ubuntu machine	45 min
2	Eclipse + Github + Maven + log4j + Jenkins setup + Github-Jenkins communication	7 hours
3	Calculator code	2.5 hours
4	Maven pom.xml changes + code comments + Exception handling completeness	1.5 hour
5	Tests + readme file	1.5 hours
6	Report writing	2 hours

3 Setup and Running

The source code can be downloaded from Github and built by Maven to get the jar (packaged along with its dependency jars). The steps involved in the process are -

3.1 Setup

```
git clone https://github.com/rhcode/simple-calculator.git
cd simple-calculator/
```

3.2 Build

```
mvn package
```

3.3 Running

```
cd target/
java -jar simple-calculator-1.0-SNAPSHOT-jar-with-dependencies.jar "add(2,2)"
```

4 Areas of Improvement

I lost out time on getting the Jenkins setup ready. As a result, I did not have enough time to implement the let command in the calculator. So I think I should have timeboxed that step to ensure I did not lose out much time in the setup. Timeboxing not just the entire task, but also timebox the individual subtasks as part of it. I would also like to see if I could make use of any design patterns in my code so as to ensure that future additions to my code, say, to add new commands should be as easy as possible, without breaking much of the code. I should have made a separate detailed document to write about the issues I faced in the process and how I fixed them, so that the next time I would not have to spend much time if they would crop up.