Step 1 Baseline 1

Step 1.1 – pull latest code and instructions 1

Step 1.2 – run project 1

## Reference

If at any point during this lab, you want to explore another feature, see the

<http://junit.org/junit5/docs/current/user-guide>.

## Step 1 Baseline

In this lab, we are going to be new developers on the san-francisco-tourism project. The project currently uses JUnit 4. The goal of step 1 is to confirm we have a known good state to start with.

### Step 1.1 – pull latest code and instructions

Clearly, the first thing we have to do is pull the latest code for the project! Your teammate (me) helped by pulling an older version of the project to test connectivity and make it as easy as possible.

TODO – create screenshots for eclipse and intellij once set up VM

There were likely changes to this document after it was created so please copy the latest from https://github.com/boyarsky/JavaOne2017-HOL-JUnit5

### Step 1.2 – run project

This project currently uses JUnit 4. We want to make sure that we can run builds now before changing anything. After all, we want a known good state before writing code.

TODO – create screenshots for eclipse and intellij (maven and built in UI) once set up VM (include setting up eclipse launch configuration)

## Step 2 – Convert to JUnit 5

JUnit 5 is available and we want to use it. The goal of step 2 is run all the JUnit 4 style tests using a JUnit 5 runner. This is the first step in migration on a real project. Once all the plumbing is ready, it will be time to actually use JUnit 5 syntax.

### Step 2.1 – Update pom.xml

The following walk you through updating the pom.xml to use JUnit 5 with legacy JUnit 4 support. If you aren’t familiar with Maven (or aren’t sure what the updated pom.xml should look like based on these steps), feel free to look at the [solution guide pom.xml](https://github.com/boyarsky/JavaOne2017-HOL-JUnit5/blob/master/san-francisco-tourism-solution/pom.xml).

1. Add property to pom.xml so can run unit tests.

<surefire.version>2.19.1</surefire.version>

1. Add properties to pom.xml to pull JUnit. The first two are always required. The third is so you can run JUnit 5 tests.

<junit.jupiter.version>5.0.0-RC3</junit.jupiter.version>

<junit.platform.version>1.0.0-RC3</junit.platform.version>

<junit.vintage.version>4.12.0-RC3</junit.vintage.version>

1. Add the Surefire plugin to the <build> section so that a version of Surefire that works with JUnit 5 is used. (2.20 does not work as of this time so using 2.19).

*Note that this lab doesn’t have any integration tests. If your project does, you’ll need to do the same with the maven-failsafe-plugin.*

<plugin>

<artifactId>maven-surefire-plugin</artifactId>

<version>${surefire.version}</version>

<dependencies>

<dependency>

<groupId>org.junit.platform</groupId>

<artifactId>junit-platform-surefire-provider</artifactId>

<version>${junit.platform.version}</version>

</dependency>

</dependencies>

</plugin>

1. Add the following to the <dependencies> section to pull all required JUnit jars:

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<version>${junit.jupiter.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-params</artifactId>

<version>${junit.jupiter.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.platform</groupId>

<artifactId>junit-platformlauncher</artifactId>

<version>${junit.platform.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-engine</artifactId>

<version>${junit.jupiter.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

<version>${junit.vintage.version}</version>

<scope>test</scope>

</dependency>

### Step 2.2 – Run Maven build

TODO – add screenshot and count of expected tests

Search in the output console for “surefire” and look at how many tests were run. If you got a non-zero answer, you updated the POM properly.

### Step 2.3 – Run in IDE

TODO add eclipse and intellij screenshots/instructions

Note how many tests were run in the console. It should also match.

## Step 3 – Converting basic tests to JUnit 5

Now that we’ve seen JUnit 5 can run JUnit 4 tests, it is time to convert these tests to JUnit 5.

### Step 3.1 – Updating the FishermansWharfTest

This test uses the most basic features of JUnit; literally just a setup method and a few assertions.

1. Change the static import to:

**import** **static** org.junit.jupiter.api.Assertions.\*;

Notice that there is a new package naming convention for JUnit 5 containing “jupiter.” Having a new package name allows running existing tests without wholesale changes. Also, note that the **Assert** class has been renamed to **Assertions**.

1. Change the regular import to follow the new naming convention as well so JUnit 5 can find the annotations:

**import** org.junit.jupiter.api.\*;

1. Speaking of annotations, the @Before annotation has been renamed to @BeforeEach so change that too.
2. One more compiler error to fix. In JUnit 5, the String message parameter is the last parameter instead of the first one. Just fix the assertion in oldestSeaLionForLastMessage() to make the code compile:

*assertEquals*(15, actual.getAge(), "oldest");

1. Now that the test compile, run the tests again. You should get one failing test. Notice that the AssertionFailedError comes from opentest4j; which got pulled in as a transitive dependency. Also note that there is a bunch of Java 8 stack trace below the code that actually caused the error.



1. Now you can fix the failing test by changing the parameter order:

*assertEquals*("http://www.fishermanswharf.org", wharf.getUrl(), "url");

1. Run the test again and you get a green bar.

### Step 3.1 – Updating the xxx