

SPEED WALKTHROUGHS

JAVA

Select a destination machine. This is setup as a default for the workshop, but if you want to know what to do for another environment if you want to set it up from scratch:

Prerequisite: Download a Virtual Machine disk image or set up a cloud instance. You can also install this on your own desktop/laptop machine as well.

* = These are older versions of software chosen for maximum compatibility. As you become more proficient and knowledgeable regarding the components, you can update them as you see fit.

- Download and install Chrome browser
- Download and install Firefox browser

- Download and install Java 8 JDK

(<https://java.com/en/download/>)*

- Set your \$JAVA_HOME and \$PATH variables so that your system can find Java
(run `echo \$JAVA_HOME` and `javac -version` to determine if you have successfully completed these steps.

- Download and unpack Eclipse Oxygen 3*

(<https://www.eclipse.org/downloads/packages/release/oxygen/3/eclipse-ide-java-ee-developers>)

(Eclipse doesn't actually install, meaning once you unpack it, you can move the folder anywhere and it's all self contained)

- Download Selenium 3.12 Client & WebDriver Language Bindings*

<https://selenium-release.storage.googleapis.com/index.html?path=3.12/>)

(contains multiple jar files to integrate with Eclipse projects)

-- Unzip and place in a folder

--- client-combined-3.12.0.jar

--- Client-combined-3.12.0-sources.jar

--- libs folder (numerous other jars for selenium)

apache-mime4j-0.6.jar

bsh-2.0b4.jar

cglib-nodep-2.1_3.jar

commons-codec-1.10.jar

commons-exec-1.3.jar

commons-io-2.4.jar

commons-logging-1.2.jar

gson-2.3.1.jar

guava-19.0.jar
hamcrest-core-1.3.jar
hamcrest-library-1.3.jar
httpclient-4.5.1.jar
httpcore-4.4.3.jar
httpmime-4.5.jar
jcommander-1.48.jar
jna-4.1.0.jar
jna-platform-4.1.0.jar
junit-4.12.jar
netty-3.5.7.Final.jar
phantomjsdriver-1.2.1.jar
testng-6.9.9.jar

- Download Apache commons-io 2.6 jar
(<http://mirror.reverse.net/pub/apache/commons/io/binaries/commons-io-2.6-bin.tar.gz>)
(useful for File interactions)

Install Maven (you will be glad you did, just trust me)
<http://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>

Create a project skeleton via command line:

- cd to where you want to make your project and enter this command:
- `$ mvn archetype:generate -DgroupId=Cucumber -DartifactId=AutomationFramework -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false`
- Once this is made, if you later want to import this maven project into eclipse, run the command:
`Mvn eclipse:eclipse`
This will create a couple of files so that Eclipse recognizes it s a maven project.

Create a Project Skeleton in Eclipse:

- Create a project skeleton via Maven
- In Package explorer, "Select New:Other:Maven:Maven Project"
- Use Default Workspace Location, click "Next"
- Select "maven-archetype-quickstart", Click "Next"
- Give a group ID of "Cucumber"
- Give an Artifact ID of "AutomationFramework"

Set up the pom.xml file and add maven repository dependencies

- Go to <https://mvnrepository.com/>
- Search for the following:
- cucumber-java (info.cukes, 1.2.5)

- cucumber-unit (info.cukes, 1.2.5)
- junit (junit, 4.12)
- Copy the Maven <dependency> blocks for each and put them in the <dependencies> section of pom.xml
- Examine the "Maven Dependencies" section of the Package Explorer for the project
- See the jar files associated with Maven dependencies

In the pom.xml file, reference the maven sureFire plugin
<https://maven.apache.org/surefire/maven-surefire-plugin/usage.html>

```
<build>
  <pluginManagement>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>2.22.0</version>
      </plugin>
    </plugins>
  </pluginManagement>
</build>
```

- Select the Natural 0.7.6 Editor from the Eclipse Marketplace
- Click Install. The Cucumber Editor is selected by default. Click "Confirm".
- Let the Natural Editor configure. Select "accept license agreement" and click "Finish"
- Click "File: Restart Eclipse". If you see a warning about installing unsigned content, click "Install anyway".

Feature Files - Gherkin statements that define our test cases

- In the src/test/java folder, create a package named "features"
- in the features package, create a file called "GoogleHomepage.feature"
- create a scenario that will examine some elements on the Google Home page (Google was chosen because it's a ubiquitous example and unlikely to change dramatically)
- Save the file.

Step Definition Files - Code mapping associated with Feature files

- in the /src/test/java folder, create a package called "stepDefinitions"
- take the contents of the feature file and copy/paste into Tidy Gherkin (this will generate step definitions in Java)
- create a class in "stepDefinitions" called "StepDefinitions".
- paste the contents from tidy gherkin into this step definitions file (each line/block corresponding with one and only one line from the feature file).
- as a placeholder, create print statements inside the "throws Throwable block for each statement"

- System.out.println("Enter your text here");
- Repeat for each statement. (Note: these are temporary and only for confirming everything is communicating correctly.
- Click "Save"

Test Runner Files - Statements that allow for the test run to execute

Create a Java package called testRunners

Create a class inside that package called TestRunner

Set up the test runner with the following calls:

```
package testRunners;
```

```
import org.junit.runner.RunWith;
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;
```

```
@RunWith(Cucumber.class)
@CucumberOptions(
    features="src/test/java/features",
    glue="stepDefinitions")
```

```
public class TestRunner {
}
//End Code
```

Create selenium commands that will become basis for step definitions.

Create a package in src/test/java called "common".

This is a good place to put classes that will extend objects.

To start the selenium commands, make a class called "Page_BasePage" that will allow for a global call for the driver object.

```
package common;
```

```
import org.openqa.selenium.WebDriver;
```

```
public class Page_BasePage {
    Public static WebDriver driver;
}
```

Set up a package called "selenium" where the standard selenium commands you want to call can be separated into individual methods.

Create a class called "Page_GoogleHomePage" that extends from Page_BasePage so that the driver object can be used for all functions.

For AutomationFramework code files, you can go to <https://github.com/mklttesthead/flamechamp>

References:

AutomationTestingHub (<http://www.automationtestinghub.com/>)

- Seriously, this is a fantastic resource and covers Selenium, Appium, Java and Cucumber in a very down to earth way and in very plain language. Focuses on doing the steps as standalone as possible, though leverages Eclipse as the driver.

Shetty, Rahul, Cucumber with Java: Build Automation Framework in Less Code, Packt Publishing

- Lots of great details about setting up projects with Maven and streamlining the setup of Eclipse with plugins

Shetty, Rahul, Selenium Web Driver with Java, Basics to advanced, Packt Publishing

- A lot of information but a great follow-on if you want to get into the nitty gritty of Selenium

Sundberg, Thomas, Al Wasim, Ripon, Mastering Selenium Testing Tools, Packt Publishing

- A little old now, but still a great resource for how to leverage the variety of Selenium setups available

Rose, Wynne, Hellesoy, The Cucumber Book for Java, Pragmatic Publishing

- Detailed and beautifully explained. Takes time to get through, but take that time, it will be worth it, I promise :).

Garg, Shankar, Cucumber Cookbook, Packt Publishing

- Lots of small projects to help you get your head around interesting details of Cucumber.