SPEED WALKTHROUGH - JAVA

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

Prerequisite:

Download a Virtual Machine disk image or set up a cloud instance.
For a quick VM setup like the one in the workshop, you can get the following:
https://www.osboxes.org/ubuntu/
Download “Ubuntu 18.04.1 Bionic Beaver (Final)”
Uncompress the file and you will have a .vdi (virtualbox disk image).
Install VirtualBox, create a new machine, attach an existing disk and point to the .vdi file downloaded.

Start up the machine.
Username: osboxes.org
Password: osboxes.org
Change the name of the machine if you wish.

Create a sudo user if you want (osboxes.org is a sudo user)

```
$ adduser [username]
$ usermod -aG sudo [username]
$ su - username
```

After getting the VM to the state you want, let’s start installing the necessary software:

* = 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
https://askubuntu.com/questions/510056/how-to-install-google-chrome

Download and install Firefox browser
(installed by default on Ubuntu)

Download and install ChromeDriver
- http://chromedriver.chromium.org/downloads

Download and install GeckoDriver
- https://github.com/mozilla/geckodriver/

These can be placed anywhere as they are standalone files. I typically put them in a “drivers” folder in my home directory.

Download and install Java 8 JDK

```
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt-get update
$ sudo apt-get install oracle-java8-installer
$ sudo apt install oracle-java8-set-default
```

Set your $JAVA_HOME and $PATH variables so that your system can find Java

```
$ sudo update-alternatives --config java
$ sudo vi /etc/environment

JAVA_HOME="/usr/lib/jvm/java-8-oracle" # or wherever you put it.

$ source /etc/environment
$ echo $JAVA_HOME
$ javac -version
```

Install Maven

```
 $ sudo apt install maven
 $ mvn -version
```

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 is a maven project.

Alternately, you can download the flamechamp project from:

https://github.com/mkltesthead/flamechamp

You can set up git and clone the repo or download as a zip and extract the files. Either way it is set up as a maven project and matches the format specified in the project generation command above.

From here, in the root directory of the project, you can run:

```
$ mvn clean
$ mvn compile
$ mvn test
```

Set up A Local Version of Jenkins

```
$ wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo
apt-key add -
$ sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ >
/etc/apt/sources.list.d/jenkins.list'

$ sudo apt-get update
$ sudo apt-get install jenkins
$ sudo systemctl start jenkins
$ sudo systemctl status jenkins
```

Edit /etc/default/jenkins to replace the line

```
HTTP_PORT=8080
```

with

```
HTTP_PORT=9191
```

Or some other port if desired.

A local instance of Jenkins has already been set up on the VM and can be accessed at http://potchbox:9191/

Username: flamechamp
Password: LaMer

Click on “Manage Jenkins: Global Tool Configuration”

Add a JDK:
Enter the JDK value:

Look for $JAVA_HOME and paste it along with the Java version
Enter the Maven value
Look for $MAVEN_HOME and paste it along with the Maven version
Click Save.

If the values are accurate, there will be no errors on the “Configure Jenkins” page.

(You may see “Reverse Proxy” appears to be broken, it is not set up in this configuration).

Eclipse Specific Steps

If you want to use Eclipse as an IDE and use its maven capabilities, read on. Otherwise, what’s listed above is enough for you to write feature files, create step definitions, put together underlying code statements to map the step definitions to and create a test runner to run the tests and create reports.

- 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".

Create a Maven Project Skeleton (or import flamechamp) in Eclipse:
-- In Package explorer, "Select File: Import:Maven: Existing Maven Project"
-- Select the AutomationFramework project from where it was unpacked
-- If you see pom.xml selected, click“Finish”
-- Note; it may take some time to get everything pulled down

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

For ease of setup, you can copy these lines into your pom.xml, but understand the process to get/update these entries above:

```
<dependencies>
<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.6</version>
</dependency>

<!--
https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-j
ava -->
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>3.14.0</version>
</dependency>

<!-- https://mvnrepository.com/artifact/info.cukes/cucumber-java -->
<dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>1.2.5</version>
</dependency>

<!-- https://mvnrepository.com/artifact/info.cukes/cucumber-junit -->
<dependency>
    <groupId>info.cukes</groupId>
    <artifactId>cucumber-junit</artifactId>
    <version>1.2.5</version>
    <scope>test</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/junit/junit -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>

</dependencies>
```

-- 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 (put this block before dependencies)
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>
```

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 feture file and cop/y 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 comunicating 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 {
}
```

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.

From there, you are good to go :).

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. Leverages Eclipse IDE.

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 IDE 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, Matering 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.