FIRST EDITION – CHAPTER 1 REV 1

Kevin Thomas

Forward

It was Christmas morning, 1988, I opened the largest box under the tree and low and behold I saw a Commodore 64 personal computer!

I frantically ripped the box apart to get at the C-64 console and hijacked the main living room television to hook it up. I turned it on and saw for the first time a light blue screen and a blinking prompt which showed a READY and a blinking cursor.

I was able to acquire a small handful of games but I was obsessed with connecting with other people in the world that I can communicate with outside of the, “solo game experience”.

I got my hands on a 300 baud modem and was able to get it hooked up and connected to Quantum Link which was a U.S. and Canadian online service for the Commodore 64 and 128 personal computers that operated starting November 5, 1985. In October 1989 the service was renamed to "America Online" and the service was made available to users of PC systems in addition to Commodore users.

I thought, “what if I can have my own online system”. This was well before the days of the Internet. I found a piece of software called DMBBS 4.8 which was a Commodore 64 BBS or bulletin board system that allowed me to create my own online experience that others could log into. I found a BBS or bulletin board system called, “The Boozers Place” which checked all the boxes of what I was looking for.

I reached out to the owner or SYSOP of, “The Boozer’s Place”, and he showed me how I could create my own BBS which I called, “THE ALLNIGHTER”. I got it set up and launched it within a day.

Many friends would scoff and say, “only geeks use these computers and this is a fad, this will die out like every other fad in the past”. Time proved them starkly incorrect.

As we fast forward to today we find ourselves not only with one or more household computers but multiple cell phones, tablets, work and personal computers along with a ton of microcontrollers which have shaped our entire existence.

Today’s software is not like the software of yesterday. Unlike the days of old where you had one application that had a very limited feature scope, we ever increasingly divorce ourselves from the monolithic architecture and gravitating toward the microservices architecture.

In modern software, our interface is quite different to where it once was with either a command-line or even native OS GUI user experience to that of a web-centric user experience.

Despite all of the careful unit testing that occurs for each of these back-end components no matter how big or small the product, we find an increasing need for a comprehensive integration testing framework to which we have a solution with Selenium.

If we think about any modern app such as YouTube we take for granted all of the complex back-end services that exist which connect up to the web interface. No matter how comprehensive the individual micro service components are designed and unit tested they alone exist within only their narrow focus of their respective scope.

In other words you could have multiple databases tied up to a single web interface. Let’s examine one single instance of a database to which there will be code that communicates with the that single database instance and then connects to yet another database and/or other segments of code. Each would represent a micro service which could live in a Docker container or Kubernetes pod which then communicates with each other possibly using some message broker like Kafka or ZMQ to which you find yourself in an extremely complex integration.

The need for Automation Engineering and a comprehensive Automation Testing Framework has never been more real as we continue to find ourselves with larger and larger integrated micro service products.

Think of it this way, if a single component which was designed well and was properly unit tested which stands alone provides no real way of working properly when integrated into a larger system.

Think of a single unit test as a brand new house where it is just you and your one of three dogs. You walk into the house, with dog 1 and you walk in and out of each room. You test the dogs reaction and see how she functions in this new environment.

After your traversal throughout the house you find that the dog responds positively and all is well.

You repeat this step with dog 2 and three with the same result and you feel confident that when you integrate all three dogs all will be well!

Imagine if those dogs never met or interacted with each other before? This is much like individual software components.

You now are confident that everything is perfect and you bring in the entire family and the three dogs and BOOM everything blows up and you scratch your head thinking what could have went wrong?

This simple analogy demonstrates the increasing demand of good integration testing with Automation and the most robust solution exists within a Java Selenium environment.

This text will be a journey where we start by setting up our development environment on Windows, as MAC will be an almost identical setup, and creating our first automation where we automate a Google web search for coffee.

We then spend time learning about how information is captured from the web interface such that we could automate it by uniquely identifying individual locators that can be used in an Automation Framework to properly test integration and/or scrape data on a custom web site designed solely for this course working with XPATH.

We then move into some Java basics and develop a simple Test Automation to automate this web site and then move on to developing a more comprehensive object-oriented Automation Framework which can scale in a production environment.

This course will be a comprehensive solution to learn and develop a sophisticated Automation Framework which could be used in a Test Automation production environment. I am excited to start this journey with you! Let’s get started!

Table Of Contents

Chapter 1: Getting Started

How often do you use the web? How often do your friends, family, coworkers and really anyone you come into contact with?

Quite simply everything we do interfaces with the web in some way and will continue to do so going forward. Each website is a unique product to which the importance of testing the integration of all of the moving parts of each site is critical to ensure a cohesive user experience.

When you hear the words, "Front-End Automation", we typically refer to Automated Testing or web automation. This is a topic we will be covering in this course however web automation is much more than testing the integration of a site as it also provides an ability to literally automate any possible interaction between a human being and a website.

Understanding web automation can help you also to scrape data from literally any source and then perform basic data science on that data which could be fed into a machine-learning model.

The, "why", has so many answers in this case and taking the time to learn web automation will give you an in-demand set of skills which could help you obtain a career as a SEiT (Software Engineer in Test).

In this course we are going to use the Chrome and Firefox browser and focus on Selenium which is a free (open-source) automated testing framework which will provide us the ability to achieve everything we have discussed thus far.

Let’s first download Eclipse which will be our integrated development environment.

https://www.eclipse.org/downloads/

Once installed, let’s create a new Maven project. What is Maven you say? Maven means accumulator of knowledge in Yiddish and developed by Apache has become one of the most powerful software project management tools available today. It is built on the concept of a POM or project object model. Maven is also open-source.

Let’s open up Eclipse and get started by following the below steps.

```
File
New
Maven Project
CHECK Create a simple project (skip archetype selection)
CHECK Use default Workspace location
Next
Group Id: Selenium Framework
Artifact Id: Selenium Framework
Finish

OPEN pom.xml
Dependencies

(VISIT https://mvnrepository.com)
(SEARCH Selenium Java)
(CLICK latest version i.e. 4.0.0-rc-1)
Add
Group Id: org.seleniumhq.selenium
Artifact Id: selenium-java
Version: 4.0.0-rc-1
OK

(VISIT https://mvnrepository.com)
(SEARCH Junit Jupiter API)
(CLICK latest version i.e. 5.8.0)
Add
Group Id: org.junit.jupiter
Artifact Id: junit-jupiter-api
Version: 5.8.0
OK

RT CLICK /src/test/java
New
File
BrowserTest.java
Finish

RT CLICK SeleniumFramework
New
Folder
drivers
Finish

RT CLICK drivers
New
Folder
chromedriver
Finish

RT CLICK drivers
New
Folder
geckodriver
Finish
```

(VISIT https://sites.google.com/chromium.org/driver/downloads)
(Current Releases – click on link to your current Chrome version)
(Extract chromedriver.exe)
DRAG AND DROP /drivers/chromedriver
CLICK Copy files
OK

(VISIT https://github.com/mozilla/geckodriver/releases)
(Assets – click latest version i.e. geckodriver-v0.30.0-win64.zip)
(Extract geckodriver.exe)
DRAG AND DROP /drivers/geckodriver
CLICK Copy files
OK

Now let's populate our **BrowserTest.java** file which will open a Chrome browser, navigate to *google.com* and wait 3 seconds and close.

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class BrowserTest {
	public static void main(String[] args) throws InterruptedException {
		String projectPath = System.getProperty("user.dir");
		System.setProperty("webdriver.chrome.driver",
projectPath+"//drivers/chromedriver/chromedriver.exe");
		// System.setProperty("webdriver.gecko.driver",
projectPath+"//drivers/geckodriver/geckodriver.exe");

		WebDriver driver = new ChromeDriver();
		// WebDriver driver = new FirefoxDriver();

		driver.get("https://google.com");
		Thread.sleep(3000);
		driver.close();
	}
}
```

CTRL+F11

Congratulations! You just created your first Automation code in Java! Time for cake!

Let's take a moment and review the official Selenium docs.

(VISIT https://www.selenium.dev/documentation)

We will use these docs to create our entire framework step-by-step.

Chapter 2: Basic Automation

Now that we have our setup complete, let’s dive into some basic automation with our own custom site. Let’s begin by copying the Site directory within our GitHub repo https://github.com/mytechnotalent/Selenium-For-Everyone to our Desktop.

Assuming you have Python3 installed, navigate to your Site folder on your Desktop within a terminal and type the following.

```
Python3 -m http.server
```

Open your Chrome browser and navigate to the following.

```
http://localhost:8000
```

Now we have a simple web server and site available for local testing.

Let’s right-click on our search box and click inspect.

```
<input autocomplete="on" class="form-control search" name="q" placeholder="Search in google.com" required="required" type="text">
```

Here we see a number of identifiers. The one we want to target is the *name* as it has a unique value of *q*.

Let’s open our **BrowserTest.java** file and add the following code and run.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class BrowserTest {
	public static void main(String[] args) throws InterruptedException {
		String projectPath = System.getProperty("user.dir");
		System.setProperty("webdriver.chrome.driver",
projectPath+"//drivers/chromedriver/chromedriver.exe");
		// System.setProperty("webdriver.gecko.driver",
projectPath+"//drivers/geckodriver/geckodriver.exe");

		WebDriver driver = new ChromeDriver();
		// WebDriver driver = new FirefoxDriver();

		driver.get("http://localhost:8000");

		WebElement textBox = driver.findElement(By.name("q"));

		textBox.sendKeys("coffee");

		Thread.sleep(3000);

		driver.close();
	}
}
```

CTRL+F11

Here we see the word *coffee* being typed into our text box and then it waits 3 seconds and closes.

Within Java we have the *WebElement* class which represents an HTML element. Generally, all interesting operations to do with interacting with a page will be performed through this interface.

As a professional SEiT, the MOST IMPORTANT skill you will develop will be your ability to read and parse the official Selenium documentation.

(VISIT https://www.selenium.dev/documentation/webdriver/locating_elements)

Turn your attention to the Locating one element area at the top of the page.

We learn here how to find elements on a page. We have a number of built-in selector types as illustrated in the docs.

We simply instantiate the *WebElement* class with the *textBox* object by calling the *findElement* method and passing the param *By.name("q")*.

We can call methods on the textBox object. Let's refer to the docs.

(VISIT https://www.selenium.dev/documentation/webdriver/keyboard)

We see the *sendKeys* method which types a key sequence in the respective DOM element. We simply type `textBox.sendKeys("coffee");` this will handle our text input.

Our next step is to programmatically press the *Search* button.

```
<button class="button" type="submit">Search</button>
```

Here we create a searchBtn object and call the By.className method as we have a class called button `WebElement searchBtn = driver.findElement(By.className("button"));` this will setup our object to be called.

We then call the click method on the searchBtn object `searchBtn.click();` this will click our button.

The entire code block is as follows.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class BrowserTest {
	public static void main(String[] args) throws InterruptedException {
		String projectPath = System.getProperty("user.dir");
		System.setProperty("webdriver.chrome.driver",
projectPath+"//drivers/chromedriver/chromedriver.exe");
		// System.setProperty("webdriver.gecko.driver",
projectPath+"//drivers/geckodriver/geckodriver.exe");

		WebDriver driver = new ChromeDriver();
		// WebDriver driver = new FirefoxDriver();

		driver.get("http://localhost:8000");

		WebElement textBox = driver.findElement(By.name("q"));

		textBox.sendKeys("coffee");

		WebElement searchBtn = driver.findElement(By.className("button"));

		searchBtn.click();

		Thread.sleep(3000);

		driver.quit();
	}
}
```

Congrats! We successfully took our first steps into some basic interaction through automation!