Selenium 4 Beginners

Rex Jones II

(Part 1) Building Blocks For Selenium

Table of Contents

Introduction

The Building Blocks For Selenium includes 3 parts: the programming part, the test framework part, and the Selenium part. Each part serves a different purpose. I will provide an introduction to each part starting with the programming language.

Programming Languages

The programming part is important when it comes to automating with Selenium. According to Selenium's HQ website, there are 7 preferred programming languages: Java, C Sharp - C#, Python, Ruby, PHP, Perl, and Javascript.

There is always a discussion regarding which programming language is the best to use. A lot of people want to know which language is the easiest. Truth be told, once you learning the concepts of one language, you can transition and learn another language. Therefore, when I decided to learn Selenium, I decided to learn the language that was most in demand from the market. In other words, I decided to learn the most popular language.

According to TIOBE, Java is the most popular programming language. TIOBE has an index for tracking the popularity of languages within the programming community. Java is #1, Python is #4, C# is #5, PHP is #7, JavaScript is #8, Ruby is #11, followed by Perl at #14. Based on my personal experience, plus talking to people who use Selenium for their automation efforts. Some of them use C#, Java, Python, and Ruby, I believe Java is the best language for Selenium.

Why do I believe Java is the best language for Selenium?

Test Frameworks

Because of TestNG. TestNG stands for Test Next Generation. It is a Java Test Framework designed for automation testers. The other Java Test Framework is JUnit. JUnit is an older test framework that influenced TestNG. JUnit is the most popular framework but TestNG is the most powerful framework. TestNG has the ability to perform Unit Testing like other frameworks but it added more features that allow testers to perform Data Driven Testing, Integration Testing, Cross Browser Testing, plus many more test features.

JUnit is part of a family called xUnit. xUnit includes JUnit for Java, PyUnit for Python, and nUnit for.Net programming languages such as C#. The xUnit family can perform Unit Testing but it cannot perform Data Driven Testing, Integration Testing, or Cross Browser Testing like TestNG. The additional features separate TestNG from all Test Frameworks and that is why I believe Java is the best language for Selenium.

Selenium Components

Selenium has 4 components: Selenium IDE, Selenium Remote Control, Selenium WebDriver, and Selenium Grid. Selenium IDE is the record and playback tool that is no longer supported by Firefox. Selenium Remote Control better known as Selenium RC was replaced by Selenium WebDriver. Selenium WebDriver allows us to automate all browser applications. Selenium Grid helps us with cross browser

testing so we can execute 1 Test Script at the same time on multiple browsers. Most of the times when a person refers to Selenium, they are talking about Selenium WebDriver which drives the web browsers.

That's a quick introduction of the programming languages, test frameworks, and Selenium.

In this tutorial, we are going to take a 3400-foot view of Java, Selenium WebDriver, and TestNG. Then we are going to finish with Questions, Answers, and Practice.

Now, let's get into Java, one of the programming languages for Selenium.

Java

There are 2 Java Data Types. The Data Types are Non-Object and Object. Non-Object is known as Primitive meaning it contains normal values such as numbers and numbers with decimals. Object Data Types are classified by classes. The purpose of Data Types is to make sure all operations are checked by the compiler for type compatibility. For example, we are not allowed to assign characters to a Data Type that allows numeric values.

These 6 Data Types int, double, boolean, String, WebDriver, and WebElement are some of the Java Data Types. Overall, there are 8 Non-Object Data Types but int, double, and boolean are the most used Non-Object Data Types. The other 5 Non-Object Data Types are byte, char, short, long, and float.

String, WebDriver, and WebElement are 3 of many Object Data Types. Do you see a difference between Non-Object and Object Data Types? Notice, how the Non-Object Data Types begins with a lowercase letter while the Object Data Types starts with an uppercase letter. The Object Data Types are uppercase because the Data Type is actually a class. All classes in Java starts with an uppercase letter.

Now, let's demo Java which begins by creating a new class.

We create a class by going to File -> New -> Class. The name will be Data Types. With Java, everything begins with a class. Classes are the groundwork for Java. It is the groundwork because everything is written inside of a class. Line 2 starts the class declaration. We have 2 Java reserved words also known as keywords written in purple: public and class.

public is an Access Modifier meaning this class can be accessed by other classes. The reserved word class declares that we are defining a class called Data Types.

Most classes have methods. A method is a block of code that should perform only 1 task. I say should because it is not wrong for a method to perform more than 1 task. However, a good programming practice is to make sure your method only performs 1 task.

Our method will be called main. I will type main then Control Space (CTRL + SPACE). Control Space (CTRL + SPACE) brings up intellisense that allows us to select the main method. We can also create a main method when we first create our new class by going to File -> New -> Class then clicking the checkbox public static void main.

Now, let's focus on the Non-Object and Object Data Types. I will write a comment for Non-Object Data Types. The 3 Non-Object Data Types are int, double, and boolean. Each data type must have a name

which is called a variable. The variable for int will be age, double will contain price, and boolean will contain result.

Okay, here's one of the advantages of a strongly typed language like Java. The Data Type determines what type of data you can assign to the variable. int allows whole numbers without decimals, double allows numbers with decimals, and boolean only allow the values true or false.

An error will appear if we assign the wrong data to a data type. For example, let's assign 12.34 to age. The red x shows up and returns a message stating "Type mismatch: cannot convert from double to int". Change the value and only keep 34 then we no longer see the red x. However, the data type double has no problem with 12.34 because it allows decimals. We will write true for boolean.

Object Data Types. Let's start with String. In Java, String is considered a class and classes have methods. The same way this class is called DataTypes and it have a method called main. With Object Data Types, the names are actually object references. However, most people say objects so we will roll with object. In this case, we will add customer as the object and assign John Doe as the value. Notice how John Doe is surrounded by double quotes. All String values have double quotes.

An object is a template of a class which means customer will imitate or serve as a copy of String. Therefore, the same methods available for String, will also be available for customer. Let's take a look at the methods for String by going to Javadoc. Here's a short cut for navigating to Javadoc. Hover over String, click inside the Description box, then select Open Attached Javadoc in a Browser. One of the first things we see is Class String which indicates String is a class. Javadoc is a documentation for Java. It provides information on syntax, terminology, just about anything you need regarding Java.

Let's scroll down to the methods for String. The first method we see is charAt with a Description stating Returns the char value at the specified index. We also see compareTo and concat. We will discuss Strings and their methods in a subsequent video but for now I wanted to show you Javadoc so you can see some of the methods for String. Now, let's go back to our Data Types class.

I will show you how the same methods available for a class is also available for its object. On the next line, let's type customer dot and we see a list of methods in the intellisense including charAt. Look the description is the same "Returns the char value at the specified index". I will replace index with 0.

That's what separates Non-Object Data Types from Object Data Types. First of all, Non-Object Data Types begin with a lowercase alphabet while Object Data Types begin with an uppercase alphabet. Secondly, Object Data Types allow its object to contain methods which are the same as its class. An error will appear if we add a dot after a variable. Let's type result dot – Error. How about price dot? The error still shows up.

Next on the Tutorial Plan is Selenium WebDriver.

Selenium WebDriver

The WebDriver class controls all of the major web browsers: Google Chrome, Firefox, and Internet Explorer. It controls the browsers by implementing other classes such as ChromeDriver, FirefoxDriver, and InternetExplorerDriver. You can look at the names of each driver and get an idea which driver controls which browser. ChromeDriver controls Google Chrome, FirefoxDriver controls Firefox browser, and InternetExplorerDriver controls Internet Explorer.

WebElement represents an element on a web browser. An element is anything you see on a web browser such as buttons, boxes, images, it can even be text. WebDriver and WebElement are classes that contain methods. Some of the methods for WebDriver are get and findElement while WebElement contain click, sendKeys, and getText. Did you notice from one of the previous slides that WebDriver and WebElement are Object Data Types?

Let me go back to the slide right quick. We see both classes WebDriver and WebElement are located under Object. However, there is a different between these Object Data Types. String is associated with Java while WebDriver and WebElement are associated with Selenium WebDriver.

Let's go to Eclipse and take a look at WebDriver and WebElement.

I will start by writing WebDriver as the class and driver as the object. Notice the red x in front of Line 17 and the red line underneath WebDriver. Both of them indicate, there is an error with WebDriver. Let's hover over WebDriver and we see 12 quick fixes are available. The quick fix we need is to import WebDriver class from package org.openqa.selenium. I will select the first option.

Do you see the import statement at Line 1? This statement shows WebDriver was imported plus gives us a path to see the WebDriver class. Let's breakdown the path then view the WebDriver methods.

We start by going to Referenced Libraries then expand the client-combined-3.9.jar. There are many packages but the first package matches the import statement org.openqa.selenium. Expand the package then navigate to the WebDriver class. The WebDriver class has many methods such as close, getTitle, and quit. Now, let's verify the driver object have the same methods. We type driver and the dot operator. Yes, the object has the same methods as WebDriver class: close, getTitle, and quit. Do you have a description on the right side of your methods? If not, then you must attach the Javadoc or Source Code for your description to show up. I created another video called "How To Attach Javadoc and Source Code For Selenium." You see a description when I hover over WebDriver since I added Javadoc and Source Code. However, a note stating "This element neither has attached source nor attached Javadoc" will show up if you do not have one of them attached to your project.

A good question someone may ask is "I see Java allow a class and its object to contain the same methods but why. Why does a class and an object contain the same methods?" They contain the same methods because the object represents the class. In this example, driver represents WebDriver. The only way for a class to operate in a program is through an object.

If we add WebElement and an object, we will see the same concept. A red x and red line shows up because we have not imported the class. Another question someone may ask is "Why do we import

WebDriver and WebElement but did not import String?" There's at least 2 reasons for importing WebDriver and WebElement: The first reason is WebDriver and WebElement are classes outside of this class called DataTypes. The second reason is WebDriver and WebElement are Selenium WebDriver classes. We do not have to import String because String is a built-in Java class.

If we view the second import statement, we see org.openqa.selenium.WebElement. Some of the methods are clear, click, and submit. The button object contains the same methods: clear, click, and submit. Recall, a WebElement is anything we see on web browser. Therefore, we can clear text from a web browser and we can click a button on a web browser.

Next, let's talk about how we find WebElements on a web browser.

The syntax for finding a WebElement is driver.findElement(By.locator("value")). In some cases, we may instantiate an object by writing WebElement and the object name then find the element. However, before we do anything, we instantiate the driver by writing WebDriver driver. WebDriver is the class controlling the web browser and driver is the object. In this syntax, Chrome is the web browser. WebElement is the class representing an element on the web page while element is the object's name.

Let's say our element is a button. findElement will return the button on that web page. By is a parameter for findElement that finds the button by a locator type. There are 8 types of locators which locates the element. The 8 locators in alphabetical order are: className, cssSelector, id, linkText, name, partialLinkText, tagName, and xpath. Value is located while inspecting the web page.

I will show you how to find a WebElement when automating our Test Script. As a side note, there is a difference between a Test Case and a Test Script. I understand both terms are used interchangeably in our industry. However, a Test Case refers to the manual steps while a Test Script refers to the automation steps.

Let's dive into our Test Case.

The name of our Test Case is Search For Orange T-SHIRTS. First, we load the web page for Automation Practice by going to http://automationpractice.com/index.php then click the Sign in hyperlink. Next, we enter the login credentials: Selenium@Automation.com for the Email address and SeleniumAutomation1234 for the Password then click the Sign in button. After clicking the Sign in button, we click T-SHIRTS, followed by entering a color. In our example, we will enter Orange then click the Search button. Finally, we Sign out by clicking the Sign out hyperlink.

Before we automate, let's demonstrate the Test Case by walking through the application.

This is the application we are going to use. Click the Sign in hyperlink, enter email address - Selenium@Automation.com and Password - SeleniumAutomation1234. You can also create a new account if you want to.

Next, we click the Sign in button, select T-SHIRTS, enter Orange in the Search box, then click the Search button. We end our walk through by clicking the Sign out hyperlink. Most web-based applications consist of text fields, buttons, and hyperlinks. Therefore, those steps of entering data into text fields,

clicking buttons and clicking hyperlinks will provide a solid foundation for automation. You can take this foundation and automate many Test Scenarios using Selenium WebDriver.

Now, let's get started with our automation demonstration by going back to Eclipse.

The first step is to create a class which we will call Search_T_SHIRTS. I want to ask you a question. Did you notice the warning message "The use of the default package is discouraged." when we created our DataTypes class? It's not wrong to leave the Package field blank but it is best to provide a package name while creating a class.

There are times when I will do things unconventional but I plan to let you know the best practice. For example, the package name should start with a lowercase letter. I will write an uppercase B and watch the warning message "This package name is discouraged. By convention, package name usually start with a lowercase letter".

During demos, I usually write most names as CamelCase where each word begins with a capital letter. I believe it's easier to read "BuildingBlocks" with 2 capital "B's" rather than "buildingblocks" with all lower case letters. However, I would follow conventional standards on an automation project.

As you can see, one of my goals for Training or Teaching is to pause, explain the concepts rather than hurrying up and rushing through the concepts. Is it okay if I take my time?

For now, we will use the main method.

For starters, I will write System.setProperty then add a key and a value. The key will be webdriver.ie.driver while value is the location of IEDriverSever. Go to my Drivers folder, select SHIFT + Right mouse click, then Copy as Path for IEDriverServer. Paste the path as the value.

We have the option of adding 2 backward slashes or 1 forward slash.

Let's use our Test Case to assist us with our Test Script. The Test Case will serve as a guideline for our Test Script.

We need help from Selenium WebDriver to help us Search T-SHIRTS. Therefore, we will add WebDriver driver so this class and its object can drive the Web Browser. The object can be any name. However, driver is the conventional name. What browser do we want to drive?

Let's drive Internet Explorer by typing = new InternetExplorerDriver. InternetExplorerDriver is also a class. WebDriver and InternetExplorerDriver must be imported to help us automate on IE's browser. If we execute Search_T_SHIRTS, a blank IE browser will show up.

The same happens for Firefox and Chrome. I am going to change InternetExplorerDriver to FirefoxDriver. Use the shortcut keys CTRL + SHIFT + O to import FirefoxDriver. Modify the System.setProperty statement. Change ie to gecko, copy and paste the geckodriver then execute again. Firefox is blank.

Last but not least, change FirefoxDriver to ChromeDriver and import ChromeDriver. Change gecko to chrome, copy and paste the path for chromedriver. Run our program. We see a blank Chrome browser page just like IE and Firefox.

You can tell it's a Selenium script because a note states "Chrome is being controlled by automated test software".

Objects are created to represent classes. Therefore, driver will represent WebDriver throughout this entire Search_T_SHIRTS class. After loading a blank Google Chrome Browser, the next step is to access the web application. Get is a method for loading a new web page. Next, we will copy and paste the URL so we can access the web page. Now running our Search_T_SHIRTS class will load the web application.

Everything you see on this web page is considered a web element: Text, Textboxes, buttons, images, links. Before performing an action on the web elements such as typing data or clicking a button, we must first locate the web element. Elements are located using an inspector. All major browsers have built-in inspectors to help find web elements.

For Chrome, we right click on the web page then select Inspect. All of this (HTML) HyperText Markup Language information, in the Elements tab helps us locate a web element. For Internet Explorer, we right click then select Inspect Element. For Firefox, we right click and also select Inspect Element.

A question that people sometimes ask is "Can we inspect elements and execute our Test Scripts on different browsers?" Yes, we can. For example, we can inspect an element on Chrome and execute our Test Script on Firefox.

Let's inspect the Sign in hyperlink using Internet Explorer. Click the Select Element icon then locate Sign in. Do you see the blue highlighted section within HTML? Let's focus on the highlighted section because we need this information to inspect and find our element. It shows 1 Tag Name, 4 Attribute Names and 4 Attribute Values. The Tag Name is <a while the Attribute Names are title, class, href, and rel. Each Attribute Name is labeled red with an Attribute Value labeled blue surrounded by double quotes:

We are going to copy the text of Sign in.

Now we are going to use Selenium WebDriver to locate the Sign in hyperlink. First, we write driver dot Find Element By and we see all 8 locator types that will help us find an element. In alphabetical order, the locators are className, cssSelector, id, linkText, name, partialLinkText, tagName, and xpath. In our demo Building Blocks For Selenium, we will use 7 of the 8 locators. Tag Name will not be used.

We will use the linkText locator for Sign in. Read the description for Parameters: The exact text to match against. In better words, the value should be the same text as what's displayed on the web application. Paste the Attribute Value Sign in.

After finding the Web Element, the next step is to perform an action on the Web Element. We click hyperlinks and the method for clicking is called click.

Next, is the Email address field. Inspect the element which is a text field. This time, there is 1 Tag Name, 6 Attribute Names, and 5 Attribute Values. One of the Attribute Values is blank. We are going to copy the Attribute Value of email for id Attribute Name.

Go back to Eclipse, write our statement driver dot Find Element By, we will use id as the locator type. Paste email as the value. So far, we have found the element which is Email address and now we perform

an action of entering text. The method for entering text is sendKeys which has a description that states “simulate typing into an element”. Selenium@Automation.com is the text we are going to send as the email address.

Let’s run. We loaded the application, clicked the Sign in hyperlink and entered an email address.

The same concept applies for Password. For Password, we will use name as our locator type. Find the element driver dot Find Element By Name Dot then perform an action of Send Keys. SeleniumAutomation1234 is the value we are going to send for our Password

Inspect the element and we see name has a value of passwd. Sometimes multiple Attribute Names have the same Attribute Value. In this case, name and id have passwd as their value. Email address also had the same value for name and id.

Copy and paste the value for name.

Let’s sign into our application. Driver dot Find Element By CSS Selector and click is the action for buttons. We are going to use Chrome for inspecting the Sign in button and get the CSS Selector’s value.

I forgot to add 4 to the Password. Locate the button using the inspector. To get the CSS Selector’s value, we right click the highlighted line, select Copy, then select Copy selector.

Go back to Eclipse and add the value. Let’s run Search_T_SHIRTS. We have successfully logged into the application.

Next, we search for T-SHIRTS by clicking the T-SHIRTS hyperlink.

This time, we are going to use partialLinkText for our locator. Driver dot Find Element By Partial Link Text Dot Click

Inspect the element. Notice how text in the HTML and application are different. HTML has some letters written in lowercase while the application has all letters written in uppercase. Sometimes, it’s beneficial to copy the text from HTML but not this time. It can be beneficial because we might misspell the text if we type the text in Selenium. Also, there can be an extra space before the words or between the words that is difficult to see while looking at the application. The Test Script will fail when there is a mismatch.

In this Test Script, I will show you how linkText and partialLinkText both fail when using text from HTML. Starting with linkText, I am going to copy the value and paste it into Selenium then Run. It failed because we do not see the background highlighted black for T-SHIRTS.

Plus, the Console shows a NoSuchElementException – Unable to locate element: {“method”:”link text”,”selector”:”T-Shirts”}.

Now, Partial Link Text, I will remove the T and hyphen (-) then change linkText to partialLinkText and Execute. It failed again. T-SHIRTS is not highlighted black. Let’s looks at the Console. NoSuchElementException – unable to locate element: {“method”:”partial link text”,”selector”:”Shirts”}.

This time, I will make SHIRTS all capital letters like the application then Run.

It's all good.

Someone may wonder, why have partialLinkText and linkText as a locator? The purpose of Link Text is when you want to enter the complete hyperlink name but what if the hyperlink name is very long. What if part of the hyperlink is static and a separate part is dynamic?

For example, like a greeting that has static text such as Hello or Welcome but the person's name is dynamic which changes according to who signed into the application. You have the option of using static text for partialLinkText locator.

Let me show you how to find the xpath value for Search text box with Chrome's default inspector. There are 2 ways. The first way is to select Control F (CTRL + F) then type our xpath value in this area that says "Find by string, selector, or XPath". We can also use this area for CSS Selector. The second way is to right click the highlighted portion, select Copy, then copy XPath.

Before going to Selenium, I'm going to paste the value in this text area so we can investigate it. Notice how the xpath value has an id Attribute Name and Value that matches HTML. By default, when we copy and paste an xpath value, it will include id. However, we can use any of these Attribute Names and Values.

Also, notice how the id value "search_query_top" is surrounded by double quotes. Selenium WebDriver will not recognize this value unless we change the double quotes to single quotes or add a backward slash in front of the double quotes.

Watch what happens when we paste the value into Selenium WebDriver. Go back to Eclipse and type Driver dot Find Element By XPath then paste the XPath Dot Send Keys. See how there's an escape character of a backward slash in front of the double quotes. A syntax error will show up if we remove the backward slashes. Let's view the error "Syntax error on token "search_query_top", invalid AssignmentOperator".

We are going to send Orange as the keys to type.

The next step is to click the Search button. We are going to use xpath again. Driver dot Find Element By xpath Dot Click. Inspect the Search button. Check out the blue highlighted section. Do you see an id Attribute Name and Value? The format of this xpath value will be different. It's going to be different because there is no id Attribute Name and Value

Right click the highlighted portion, select Copy, then select Copy XPath. Check out what happens when we paste the value. Id is still included although we did not see id. It's using id from the form tag then using the button tag. That's how we get id = searchbox.

Remember I said, by default, when we copy and paste an xpath value, it will include id. We have the option of using other Attribute Names and Values. To show you, I am going to use type as the Attribute Name, submit as the value, and remove forward slash button. I will also use single quotes rather than double quotes so you can see how Selenium WebDriver accepts single quotes.

In an upcoming video, I will explain the hierarchy of HTML and show you how to customize values for xpath and cssSelector.

Let's paste the value and run Search_T_SHIRTS.

Finally, we sign out of the application by clicking the Sign out hyperlink. This time, we are going to use className. Driver dot Find Element By Class Name Dot Click

Let's use Firefox's inspector since we have used Internet Explorer and Chrome. I will sign into the application so we can see the Sign out hyperlink.

Inspect the Sign out hyperlink. We see Attribute class has logout as a value. Copy the value logout and paste it into Selenium.

Did you notice how I would click X in the top right corner after each run? That's because we have not written a statement to close the browser. driver.quit() closes the browser. This time when I run, I will not have to manually close the browser.

Let's end this tutorial, (Part 1) Building Blocks For Selenium, showing you how similar the programming languages are for Selenium WebDriver.

This is a screenshot of a Firefox plug-in called WebDriver Element Locator. It was used on Google's Search page and generated all 4 of these results. The plug-in identifies any web element using xpath locator for C#, Java, Python, and Ruby. However, this screenshot only shows C# and Java.

I need to let you know that this plug-in WebDriver Element Locator is no longer supported by Firefox. In addition, Firebug and FirePath are no longer supported by Firefox. I miss Firebug and FirePath. They were good for helping you locate elements especially FirePath which is similar to ChroPath.

Nevertheless, I wanted to show you this screenshot so we can focus on C# and Java. Do you see the resemblance between each script? They almost look the same. The only difference is capitalization. C# has an uppercase F in FindElement while Java has a lowercase f in findElement. C# has an uppercase XP in XPath while Java has a lowercase xp in xpath. Even the programming component is alike.

Remember at the beginning of this tutorial, I said "learn the concepts of one language, you can transition and learn another language." In other words, the key to learning a language is the syntax. The syntax is a combination of words, rules, and symbols.

Popularity does not mean superiority. Yet, I chose Java because it is the most popular language in the market. More jobs are looking for Java than any other language. Also, I chose Java because of TestNG. We will take a look at TestNG in our next tutorial session.

Questions, Answers, and Practice.

Questions

#1. How many Data Types comes with Java?

#2. What is a template of a Class?

#3. Name the 8 Selenium WebDriver locators:

Answers

#1. How many Data Types comes with Java? There are 2 Java Data Types called Non-Object and Object. Sometimes Non-Object is called Primitive while Object is called Class.

#2. What is a template of a Class? An object is a template of a Class.

#3. Name the 8 Selenium WebDriver locators: className, cssSelector, id, linkText, name, partialLinkText, tagName, xpath

Practice

If you have time, watch this video (Part 1) Building Blocks For Selenium at least one more time. You may notice something that you did not see the first time.

Execute the same Test Script from video which searches for a T-SHIRT. Perhaps you search for a different color.

Create A Test Script To Save Your Registration For Demo Web Shop Using Different Locators. Try to use the same 7 locators we covered in searching for a T-SHIRT.

Here's the Demo Web Shop sample application by Tricentis. You will click the Register hyperlink, select Gender, enter Personal Details, enter Passwords, click the Register button, click the Account information which is the email address, click the Save button, after clicking Save, you will see waiting for message at the bottom, click the Log out hyperlink, then close the browser. Let's take a look at your Test Case and Test Steps.

I plan to add more Test Cases as I create more videos. Go to your Test Case "Register For Demo Web Shop". Here's your 13 steps.

To download your Test Case and Test Steps, go to https://tinyurl.com/Part1-Selenium-Building-Blocks. Your zip file will include your Test Cases, Test Steps, presentation, automation code from this video and automation code for saving a registration for Demo Web Shop.

Thank You for watching (Part 1) Building Blocks For Selenium. Next week, I will cover (Part 2) Building Blocks For Selenium.