Selenium WebDriver Recipes in Java

The Problem Solving Guide to Selenium WebDriver

Zhimin Zhan

Selenium WebDriver Recipes in Java

The problem solving guide to Selenium WebDriver in Java

Zhimin Zhan

This book is available at https://leanpub.com/selenium-recipes-in-java

This version was published on 2025-07-18

Also By Zhimin Zhan

Web Test Automation in Action: Volume 1

Practical Performance and Load Testing

Practical Continuous Testing

Practical Desktop App Test Automation with Appium

Selenium WebDriver Recipes in Node.js

API Testing Recipes in Ruby

Selenium WebDriver Recipes in Python

Learn Swift Programming by Examples

Learn Ruby Programming by Examples

Selenium WebDriver Recipes in Ruby

Watir Recipes

Practical Web Test Automation with Selenium WebDriver

To Dominic and Courtney!

Contents

Preface

After observing many failed test automation attempts by using expensive commercial test automation tools, I am delighted to see that the value of open-source testing frameworks has finally been recognized. I still remember the day (a rainy day at a Gold Coast hotel in 2011) when I found out that the Selenium WebDriver was the most wanted testing skill in terms of the number of job ads on the Australia's top job-seeking site.

Now Selenium WebDriver is big in the testing world. We all know software giants such as Facebook and LinkedIn use it, immensely-comprehensive automated UI testing enables them pushing out releases several times a day[1]. However, from my observation, many software projects, while using Selenium WebDriver, are not getting much value from test automation, and certainly nowhere near its potential. A clear sign of this is that the regression testing is not conducted on a daily basis (if test automation is done well, it will happen naturally).

Among the factors contributing to test automation failures, a key one is that automation testers lack sufficient knowledge in the test framework. It is quite common to see some testers or developers get excited when they first create a few simple test cases and see them run in a browser. However, it doesn't take long for them to encounter some obstacles: such as being unable to automate certain operations. If one step cannot be automated, the whole test case does not work, which is the nature of test automation. Searching solutions online is not always successful, and posting questions on forums and waiting can be frustrating (usually, very few people seek professional help from test automation coaches). Not surprisingly, many projects eventually gave up test automation or just used it for testing a handful of scenarios.

The motivation of this book is to help motivated testers work better with Selenium. The book contains over 190 recipes for web application tests with Selenium WebDriver. If you have read one of my other books: *Practical Web Test Automation*[2], you probably

[1]http://www.wired.com/business/2013/04/linkedin-software-revolution/
[2]https://leanpub.com/practical-web-test-automation

know my style: practical. I will let the test scripts do most of the talking. These recipe test scripts are 'live', as I have created the target test site and included offline test web pages. With both, you can:

1. **Identify** your issue
2. **Find** the recipe
3. **Run** the test case
4. **See** test execution in your browser

Who should read this book

This book is for testers or programmers who are writing (or want to learn) automated tests with Selenium WebDriver. In order to get the most of this book, basic (very basic) Java coding skills is required.

How to read this book

Usually, a 'recipe' book is a reference book. Readers can go directly to the part that interests them. For example, if you are testing a multiple select list and don't know how, you can look up in the Table of Contents, then go to the chapter. This book supports this style of reading. Since the recipes are arranged according to their levels of complexity, readers will also be able to work through the book from the front to back if they are looking to learn test automation with Selenium.

Recipe test scripts

To help readers to learn more effectively, this book has a dedicated site[3] that contains the recipe test scripts and related resources.

[3]http://zhimin.com/books/selenium-recipes-java

As an old saying goes, “There’s more than one way to skin a cat.” You can achieve the same testing outcome with test scripts implemented in different ways. The recipe test scripts in this book are written for simplicity, there is always room for improvement. But for many, to understand the solution quickly and get the job done are probably more important.

If you have a better and simpler way, please let me know.

All recipe test scripts are Selenium 2 (aka Selenium WebDriver) compliant, and can be run on Firefox, Chrome and Internet Explorer on multiple platforms. I plan to keep the test scripts updated with the latest stable Selenium version.

Send me feedback

I would appreciate your comments, suggestions, reports on errors in the book and the recipe test scripts. You may submit your feedback on the book site.

Zhimin Zhan

Brisbane, Australia

1. Introduction

Selenium is a free and open source library for automated testing web applications. I assume that you have had some knowledge of Selenium, based on the fact that you picked up this book (or opened it in your eBook reader).

1.1 Selenium

Selenium was originally created in 2004 by Jason Huggins, who was later joined by his other ThoughtWorks colleagues. Selenium supports all major browsers and tests can be written in many programming languages and run on Windows, Linux and Macintosh platforms.

Selenium 2 marked the merger with another testing framework, WebDriver, originally led by Simon Stewart at Google (who later joined Facebook). This integration is why the project is now known as "Selenium WebDriver". Selenium 2.0 was released in July 2011, followed by Selenium 3.0 in October 2016 and Selenium 4 in October 2021.

1.2 Selenium language bindings

Selenium tests can be written in multiple programming languages such as Java, C#, JavaScript, Python and Ruby (the core ones). All examples in this book are written in Selenium with Java binding. As you will see the examples below, the use of Selenium in different bindings are very similar. Once you master one, you can apply it to others quite easily. Take a look at a simple Selenium test script in four different language bindings: Java, C#, JavaScript, Python and Ruby.

Java:

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

public class GoogleSearch  {
  public static void main(String[] args) {
    // Create a new instance of the html unit driver
    // Notice that the remainder of the code relies on the interface,
    // not the implementation.
    WebDriver driver = new FirefoxDriver();

    // And now use this to visit Google
    driver.get("http://www.google.com");

    // Find the text input element by its name
    WebElement element = driver.findElement(By.name("q"));

    // Enter something to search for
    element.sendKeys("Hello Selenium WebDriver!");

    // Submit the form based on an element in the form
    element.submit();

    // Check the title of the page
    System.out.println("Page title is: " + driver.getTitle());
  }
}
```

C#:

```
using System;
using OpenQA.Selenium;
using OpenQA.Selenium.Firefox;
using OpenQA.Selenium.Support.UI;

class GoogleSearch
{
  static void Main()
  {
    IWebDriver driver = new FirefoxDriver();
    driver.Navigate().GoToUrl("http://www.google.com");
    IWebElement query = driver.FindElement(By.Name("q"));
    query.SendKeys("Hello Selenium WebDriver!");
```

```
    query.Submit();
    Console.WriteLine(driver.Title);
  }
}
```

JavaScript:

```
var webdriver = require('selenium-webdriver');
var driver = new webdriver.Builder()
    .forBrowser('chrome')
    .build();

driver.get('http://www.google.com/ncr');
driver.findElement(webdriver.By.name('q')).sendKeys('webdriver');
driver.findElement(webdriver.By.name('btnG')).click();
driver.wait(webdriver.until.titleIs('webdriver - Google Search'), 1000);
console.log(driver.title);
```

Python:

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.get("http://www.google.com")

elem = driver.find_element_by_name("q")
elem.send_keys("Hello WebDriver!")
elem.submit()

print(driver.title)
```

Ruby:

```
require "selenium-webdriver"

driver = Selenium::WebDriver.for :firefox
driver.navigate.to "http://www.google.com"

element = driver.find_element(:name, 'q')
element.send_keys "Hello Selenium WebDriver!"
element.submit

puts driver.title
```

1.3 Cross browser testing

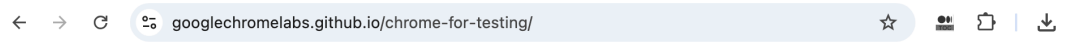
The biggest advantage of Selenium over other web test frameworks, in my opinion, is that it supports all major web browsers: Firefox, Chrome and Edge. The browser market nowadays is more diversified (based on the StatsCounter[1], the usage share in June 2025 for Chrome, Firefox, Safari, Edge and are 66.5%, 5.9%, 7.4% and 13.1% respectively). It is logical that all external facing websites require serious cross-browser testing. Selenium is a natural choice for this purpose, as it far exceeds other commercial tools and free test frameworks.

1.3.1 Chrome

To run Selenium tests in Google Chrome, you'll need both the Chrome browser and **ChromeDriver** installed. As of Selenium v4.12 and Chrome v155, the new Selenium Manager handles ChromeDriver installation automatically. However, it's still helpful to know how to install ChromeDriver manually, just in case.

Installing ChromeDriver is easy: go to ChromeDriver site[2].

[1]https://gs.statcounter.com/browser-market-share/desktop/worldwide
[2]https://googlechromelabs.github.io/chrome-for-testing/



chromedriver	**linux64**	https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.69/linux64/chromedriver-linux64.zip
chromedriver	**mac-arm64**	https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.69/mac-arm64/chromedriver-mac-arm64.zip
chromedriver	**mac-x64**	https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.69/mac-x64/chromedriver-mac-x64.zip
chromedriver	**win32**	https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.69/win32/chromedriver-win32.zip
chromedriver	**win64**	https://storage.googleapis.com/chrome-for-testing-public/130.0.6723.69/win64/chromedriver-win64.zip

Download the one matching your browser and target platform, unzip it and put **chromedriver** executable in your PATH. To verify the installation, open a command window (terminal for Unix/Mac), execute command *chromedriver*, You should see output similar to the following:

```
Starting ChromeDriver 130.0.6723.69 (...) on port 0
Only local connections are allowed.
ChromeDriver was started successfully on port 64654.
```

The test script below opens a website in a new Chrome browser window and closes it one second later.

```
import org.openqa.selenium.chrome.ChromeDriver;
//...
WebDriver driver = new ChromeDriver();
```

1.3.2 Edge

Mircosoft Edge Chromium is like Google Chrome in the context of test automation, except installing Microsoft Edge WebDriver[3] instead of ChromeDriver.

[3]https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/

```
import org.openqa.selenium.edge.EdgeDriver;
import org.openqa.selenium.edge.EdgeOptions;
//...
String edgeDriverPath = "C:\\agileway\\testing\\msedgedriver.exe";
System.setProperty("webdriver.edge.driver", edgeDriverPath);
EdgeOptions options = new EdgeOptions();
WebDriver driver = new EdgeDriver(options);
```

1.3.3 Firefox

Selenium tests require Gecko Driver[4] to drive Firefox. The test script below will open a website in a new Firefox window.

```
import org.openqa.selenium.firefox.FirefoxDriver;
// ...
WebDriver driver = new FirefoxDriver();
```

1.4 JUnit

The examples above drive browsers, strictly speaking, they are not tests. To make the effective use of Selenium scripts for testing, we need to put them in a test framework that defines test structures and provides assertions (performing checks in test scripts). The de facto test framework for Java is JUnit, and here is an example using JUnit 4.

[4]https://github.com/mozilla/geckodriver/releases/

```
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.support.pagefactory.*;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.ie.InternetExplorerDriver;

/**
 * Start 4 difference browsers by Selenium
 */
public class GoogleSearchDifferentBrowsersTest {

    @Test
    public void testInIE() throws Exception {
        WebDriver driver = new InternetExplorerDriver();
        driver.get("https://agileway.com.au/demo");
        Thread.sleep(1000);
        driver.quit();
    }

    @Test
    public void testInFirefox() throws Exception {
        WebDriver driver = new FirefoxDriver();
        driver.get("https://agileway.com.au/demo");
        Thread.sleep(1000);
        driver.quit();
    }


    @Test
    public void testInChrome() throws Exception {
        WebDriver driver = new ChromeDriver();
        driver.get("https://agileway.com.au/demo");
        Thread.sleep(1000);
        driver.quit();
    }

    @Test
    public void testInEdge() throws Exception {
        WebDriver driver = new EdgeDriver();
```

```
        driver.get("https://agileway.com.au/demo");
        Thread.sleep(1000);
        driver.quit();
    }
}
```

`@Test` annotates a test case below, in a format of `testCamelCase()`. You will find more about JUnit from its home page[5]. However, I honestly don't think it is necessary. The part used for test scripts is not much and quite intuitive. After studying and trying out some examples, you will be quite comfortable with JUnit.

1.4.1 JUnit fixtures

If you worked with xUnit before, you must know `setUp()` and `tearDown()` fixtures, used run before or after every test. In JUnit 4, by using annotations (`@BeforeClass`, `@Before`, `@After`, `@AfterClass`), you can choose the name for fixtures. Here are mine:

```
@BeforeClass
public static void beforeAll() throws Exception {
  // run before all test cases
}

@Before
public void before() throws Exception {
  // run before each test case
}


@Test
public void testCase1() throws Exception {
  // one test case
}

@Test
public void testCase2() throws Exception {
  // another test case
}
```

[5]http://junit.org/

```
@After
public void after() throws Exception {
  // run after each test case
}

@AfterClass
public static void afterAll() throws Exception {
  // run after all test cases
}
```

1.5 Set up Development Environment

Most Java developers write code using an IDE (Integrated Development Environment) such as Eclipse, IntelliJ IDEA, or NetBeans. For this book, I'll be using IntelliJ IDEA (Community Edition), which is free and well-suited for test automation. All IDE-related instructions are kept generic, so readers can easily adapt them to their preferred IDE.

1.5.1 Prerequisite:

- Download and install JDK (jdk21 is the version used in recipes).
- Download and install Intellij IDEA IDE.
- Download the latest Selenium Java binding libraries, eg. selenium-java-4.34.0.zip[6], about 35MB in size (including dependent jars)
- Download and install Apache Ant, for running tests or test suites from command line.
- Your target browser is installed, such as Chrome or Firefox.

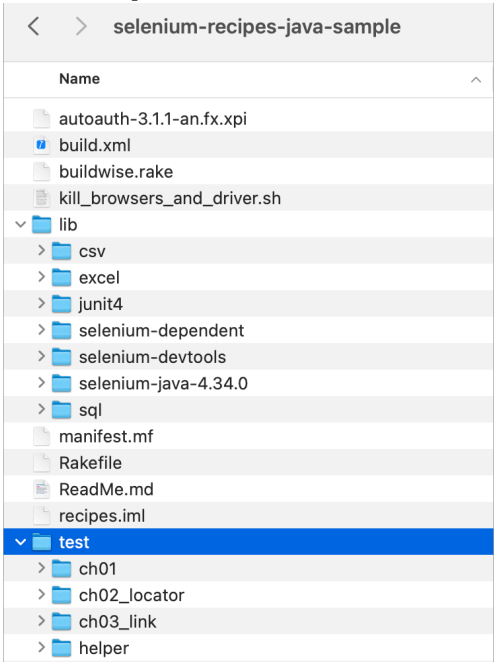
1.6 Set up Intellij IDEA project

I suggest start the same project (downloadable from the book site):

[6]https://www.selenium.dev/downloads/

- selenium-recipes-java.zip, test script in an Intellij IDEA project.
- lib.zip, requried libraries (jar files)

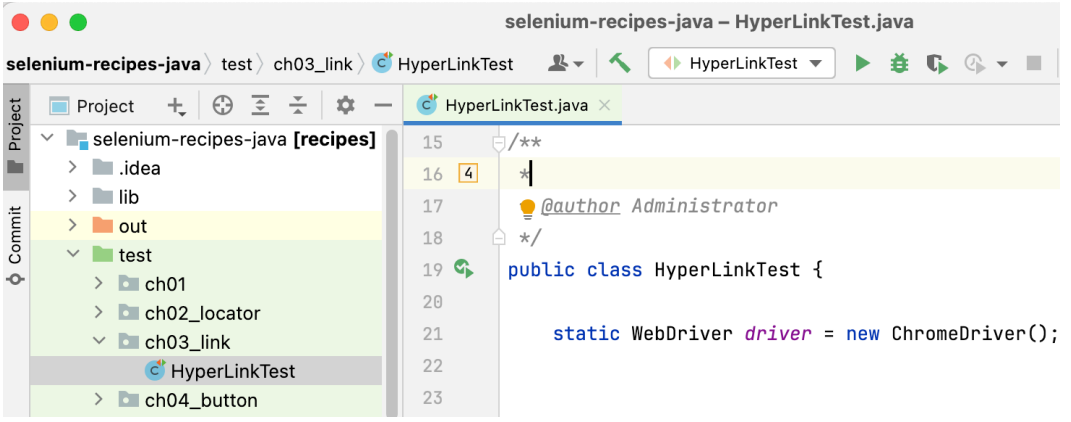
Unzip the project first. Then unzip the `lib.zip` under it.



Content of the recipe project

All test scripts (classes in Java) are located under `test` folder.

Launch Intellij IDEA and open the project folder, e.g. `selenium-recipes-java`.



1.7 Run recipe scripts

Test scripts for all recipes can be downloaded from the book site. They are all in ready-to-run state. I include the target web pages/sites as well as Selenium test scripts. There are two kinds of target web pages: local HTML files and web pages on a live site. To run tests written for a live site requires Internet connection.

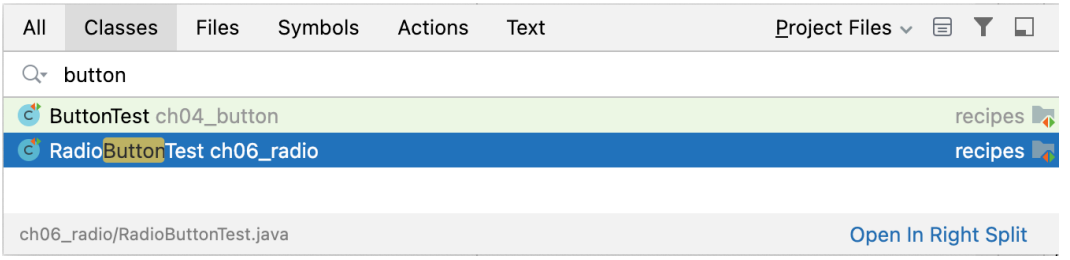
1.7.1 Run tests in a Java IDE

A test class may contain one or more test cases. The most convenient way to run one test case or a test class is to do it in an IDE.

1.7.1.1 Navigate to the test case

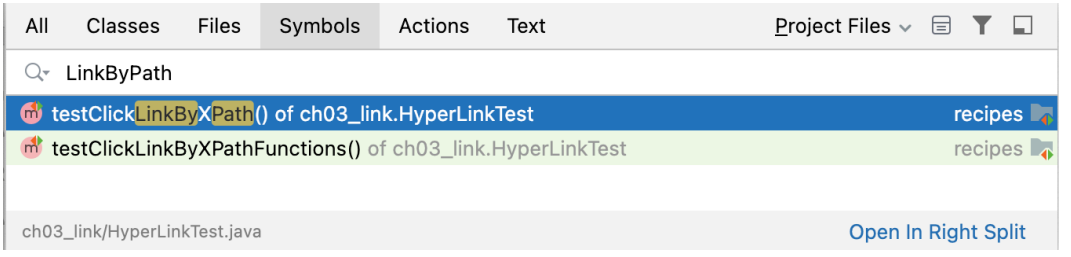
You can locate the recipe either by following the chapter or searching by name. There are about 200 test cases in one test project. Here is the quickest way to find the one you want in Intellij IDEA.

Select the menu 'Navigate' → 'Class ...'. A pop-up window will appear. As you begin typing, the search starts immediately.



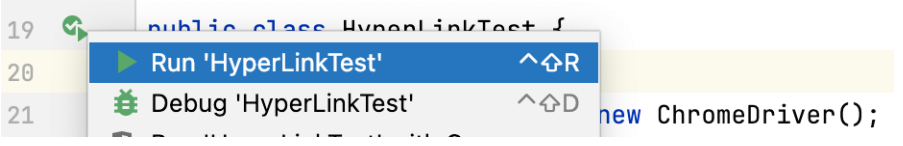
Pressing the Enter key opens the selected test class.

You can also quickly navigate to a specific test case. Select the menu 'Navigate' → 'Symbol ...'.

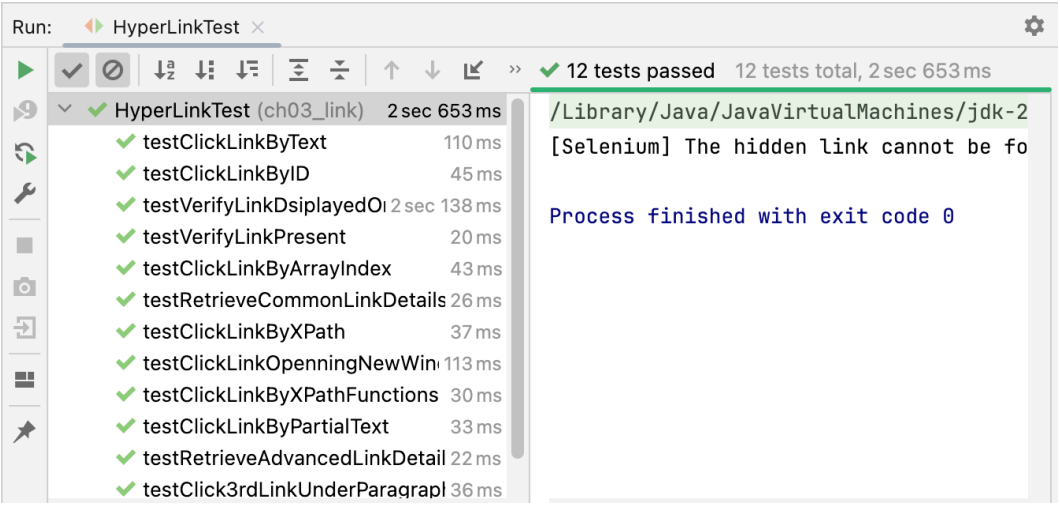


1.7.1.2 Run all test cases in a test class

In Intellij IDEA, A double green triangle icon appears next to the test class declaration (e.g., `public class XxxTest {`).



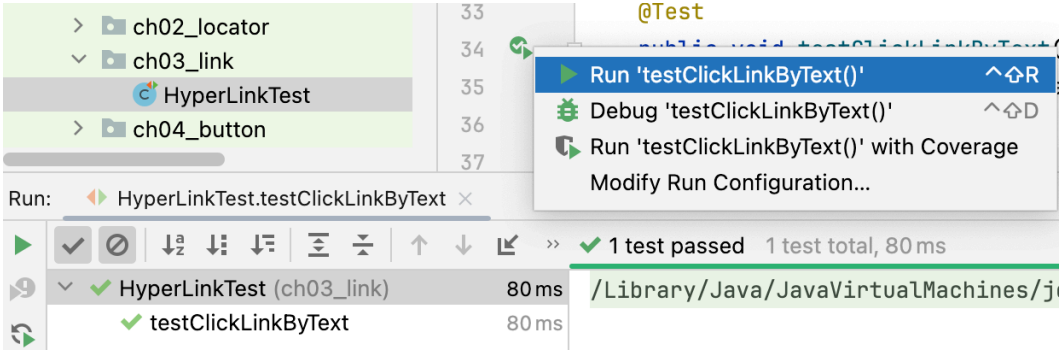
Click it to run the test class.



1.7.1.3 Run individual test case

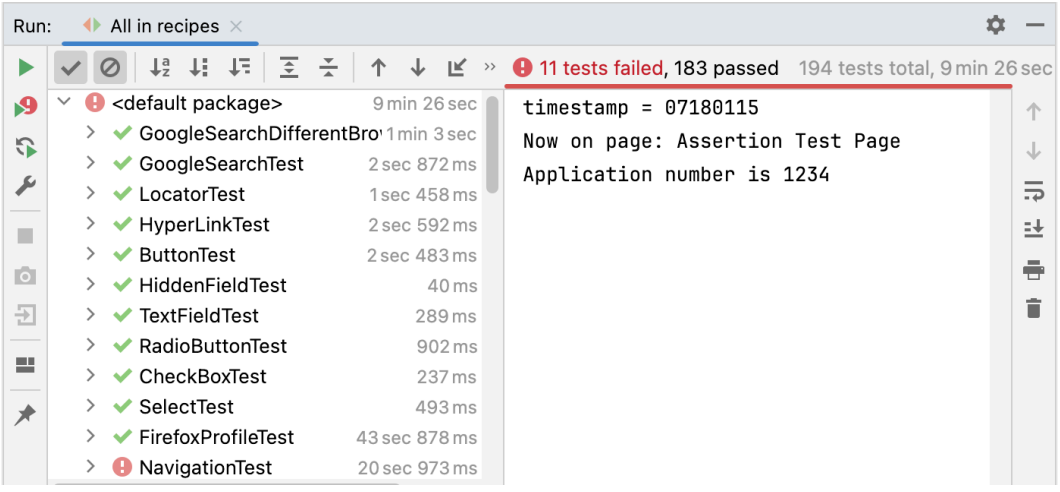
When developing a new test script or debugging an existing one, running invidual test case is the most common execution method.

Click the green triangle in the front of test class delcration, e.g. `public void testClick-LinkByID() throws Exception {`, to run this test case.



1.7.1.4 Run all tests

"You can also run all test cases in the project using IntelliJ IDEA. Right-click the `test` folder in the Project view and select **Run 'All Tests'**.



Running all 190+ tests in the recipe project using IntelliJ IDEA

> **It is not recommended to run large suites of automated end-to-end (E2E) tests in an IDE**
>
> Unlike unit tests, automated E2E tests take much longer to run and have more dependencies. The best practice is to execute the entire test suite on a Continuous Testing server, such as BuildWise.

1.7.2 Run tests from command line

One key advantage of open-source test frameworks, such as Selenium, is FREEDOM. You can edit the test scripts in any text editors and run them from a command line.

To run a Java class, you needs to compile it first (Within IDE, IDEs do it for you automatically). Running code in compiled language (such as Java) with many libraries dependency from command line is not easy as dynamic ones (such as Ruby). Build tools such as Ant can help on this.

I included an Ant build.xml (with recipe source) to simplify the test execution from com-

mand line. To run test cases in a test script file (named ch10_assertion.AssertionTest.java), enter command

```
> ant runTest -DTestName=ch10_assertion.AssertionTest
```

Example Output

```
compile:
    [mkdir] Created dir: /Users/zhimin/books/SeleniumRecipes-Java/recipes/build/c\
lasses
    [javac] Compiling 22 source files to /Users/zhimin/books/SeleniumRecipes-Java\
/recipes/build/classes

runTest:
    [junit] Running ch10_assertion.AssertionTest
    [junit] Tests run: 11, Failures: 0, Errors: 0, Time elapsed: 0.659 sec

BUILD SUCCESSFUL
Total time: 9 seconds
```

Also, to run all recipe tests (within the test folder)

```
> ant runAll
```

which generate JUnit style test report like this

Summary

Tests	Failures	Errors	Success rate	Time
108	0	0	100.00%	159.075

Note: *failures* are anticipated and checked for with assertions while *errors* are unanticipated.

Packages

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
ch01	4	0	0	28.604	2013-12-25T22:08:30	imac
ch02_link	11	0	0	3.476	2013-12-25T22:09:07	imac
ch03_button	8	0	0	3.304	2013-12-25T22:09:19	imac
ch04_textfield	7	0	0	0.735	2013-12-25T22:09:30	imac

The command syntax is identical for Windows, Mac OS X and Linux platforms.

2. Locating web elements

As you might have already figured out, to drive an element in a page, we need to find it first. Selenium uses what is called locators to find and match the elements on web page. There are 8 locators in Selenium:

Locator	Example
ID	`findElement(By.id("user"))`
Name	`findElement(By.name("username"))`
Link Text	`findElement(By.linkText("Login"))`
Partial Link Text	`findElement(By.partialLinkText("Next"))`
XPath	`findElement(By.xpath("//div[@id='login']/input"))`
Tag Name	`findElement(By.tagName("body"))`
Class Name	`findElement(By.className("table"))`
CSS	`findElement(By.cssSelector, "#login > input[type='text']"))`

You may use any one of them to narrow down the element you are looking for.

2.1 Start browser

Testing web sites starts with a browser. The test script below launches a Firefox browser window and navigate to a site.

```
static WebDriver driver = new FirefoxDriver();
driver.get("https://agileway.com.au/demo")
```

Use `ChromeDriver` and `IEDriver` for testing in Chrome and IE respectively.

Test Pages

I prepared the test pages for the recipes, you can download them (in a zip file) at the book's site[a]. Unzip to a local directory and refer to test pages like this:

```
// in TestHelper
public static String siteUrl() {
  if (isWindows()) {
    return "file:///C:/agileway/books/SeleniumRecipes-Java/site/";
  } else if (isMac()) {
      return "file:///Users/zhimin/work/books/SeleniumRecipes-Java/site/";
  } else {
     throw new RuntimeException("Your OS is not support!!");
  }
}

// in test script

driver.get(TestHelper.siteUrl() + "locators.html");
```

[a]http://zhimin.com/books/selenium-recipes-java

I recommend, for beginners, closing the browser window at the end of a test case.

```
driver.quit();
```

2.2 Find element by ID

Using IDs is the easiest and the safest way to locate an element in HTML. If the page is W3C HTML conformed[1], the IDs should be unique and identified in web controls. In comparison to texts, test scripts that use IDs are less prone to application changes (e.g. developers may decide to change the label, but are less likely to change the ID).

[1]http://www.w3.org/TR/WCAG20-TECHS/H93.html

```
driver.findElement(By.id("submit_btn")).click();
driver.findElement(By.id("cancel_link")).click();  // Link
driver.findElement(By.id("username")).sendKeys("agileway");  // Textfield
driver.findElement(By.id("alert_div")).getText();  // HTML Div element
```

2.3 Find element by Name

The name attributes are used in form controls such as text fields and radio buttons. The values of the name attributes are passed to the server when a form is submitted. In terms of least likelihood of a change, the name attribute is probably only second to ID.

```
driver.findElement(By.name("comment")).sendKeys("Selenium Cool");
```

2.4 Find element by Link Text

For Hyperlinks only. Using a link's text is probably the most direct way to click a link, as it is what we see on the page.

```
driver.findElement(By.linkText("Cancel")).click();
```

2.5 Find element by Partial Link Text

Selenium allows you to identify a hyperlink control with a partial text. This can be quite useful when the text is dynamically generated. In other words, the text on one web page might be different on your next visit. We might be able to use the common text shared by these dynamically generated link texts to identify them.

```
// will click the "Cancel" link
driver.findElement(By.partialLinkText("ance")).click();
```

2.6 Find element by XPath

XPath, the XML Path Language, is a query language for selecting nodes from an XML document. When a browser renders a web page, it parses it into a DOM tree or similar. XPath can be used to refer a certain node in the DOM tree. If this sounds a little too much technical for you, don't worry, just remember XPath is the most powerful way to find a specific web control.

```
// clicking the checkbox under 'div2' container
driver.findElement(By.xpath("//*[@id='div2']/input[@type='checkbox']")).click();
```

Some testers feel intimidated by the complexity of XPath. However, in practice, there is only limited scope of XPath to master for testers.

Avoid using copied XPath from Browser's Developer Tool

Browser's Developer Tool (right click to select 'Inspect element' to show) is very useful for identifying a web element in web page. You may get the XPath of a web element there, as shown below (in Chrome):

The copied XPath for the second "Click here" link in the example:

```
//*[@id="container"]/div[3]/div[2]/a
```

It works. However, I do not recommend this approach as the test script is fragile. If developer adds another `div` under `<div id='container'>`, the copied XPath is no longer correct for the element while `//div[contains(text(), "Second")]/a[text()="Click here"]` still works.

In summary, XPath is a very powerful way to locating web elements when `id`, `name` or `linkText` are not applicable. Try to use a XPath expression that is less vulnerable to structure changes around the web element.

2.7 Find element by Tag Name

There are a limited set of tag names in HTML. In other words, many elements share the same tag names on a web page. We normally don't use the `tag_name` locator by itself to locate an element. We often use it with others in a chained locators (see the section below). However, there is an exception.

```
driver.findElement(By.tagName("body")).getText();
```

The above test statement returns the text view of a web page, this is a very useful one as Selenium WebDriver does not have built-in method return the text of a web page.

2.8 Find element by Class

The `class` attribute of a HTML element is used for styling. It can also be used for identifying elements. Commonly, a HTML element's class attribute has multiple values, like below.

```
<a href="back.html" class="btn btn-default">Cancel</a>
<input type="submit" class="btn btn-deault btn-primary">Submit</input>
```

You may use any one of them.

```
driver.findElement(By.className("btn-primary")).click(); // Submit button
driver.findElement(By.className("btn")).click();   // Cancel link

// the below will return error "Compound class names not permitted"
// driver.findElement((By.className("btn btn-deault btn-primary")).click();
```

The `className` locator is convenient for testing JavaScript/CSS libraries (such as TinyMCE) which typically use a set of defined class names.

```
// inline editing
driver.findElement(By.id("client_notes")).click();
Thread.sleep(500);
driver.findElement(By.className("editable-textarea")).sendKeys("inline notes");
Thread.sleep(500);
driver.findElement(By.className("editable-submit")).click();
```

2.9 Find element by CSS Selector

You may also use CSS Path to locate a web element.

```
driver.findElement(By.cssSelector("#div2 > input[type='checkbox']")).click();
```

However, the use of CSS selector is generally more prone to structure changes of a web page.

2.10 Chain findElement to find child elements

For a page containing more than one elements with the same attributes, like the one below, we could use `XPath` locator.

```
<div id="div1">
  <input type="checkbox" name="same" value="on"> Same checkbox in Div 1
</div>
<div id="div2">
  <input type="checkbox" name="same" value="on"> Same checkbox in Div 2
</div>
```

There is another way: chain `findElement` to find a child element.

```
driver.findElement(By.id("div2")).findElement(By.name("same")).click();
```

2.11 Find multiple elements

As its name suggests, `findElements` return a list of matched elements back. Its syntax is exactly the same as `findElement`, i.e. can use any of 8 locators.

The test statements will find two checkboxes under `div#container` and click the second one.

```
List<WebElement> checkbox_elems = driver.findElements(By.xpath("//div[@id='contai\
ner']//input[@type='checkbox']"));
System.out.println(checkbox_elems); // => 2
checkbox_elems.get(1).click();
```

Sometimes `findElement` fails due to multiple matching elements on a page, which you were not aware of. `findElements` will come in handy to find them out.

3. Hyperlink

Hyperlinks (or links) are fundamental elements of web pages. As a matter of fact, it is hyperlinks that makes the World Wide Web possible. A sample link is provided below, along with the HTML source.

Recommend Selenium

HTML Source

```
<a href="index.html" id="recommend_selenium_link" class="nav" data-id="123"
   style="font-size: 14px;">Recommend Selenium</a>
```

3.1 Click a link by text

Using text is probably the most direct way to click a link in Selenium, as it is what we see on the page.

```
driver.get(TestHelper.siteUrl()  + "link.html");

driver.findElement(By.linkText("Recommend Selenium")).click();
```

3.2 Click a link by ID

```
driver.findElement(By.id("recommend_selenium_link")).click();
```

Furthermore, if you are testing a web site with multiple languages, using IDs is probably the only feasible option. You do not want to write test scripts like below:

```
if (is_italian()) {
  driver.findElement(By.linkText("Accedi")).click();
} else if (is_chinese()) { // a helper function determines the locale
  driver.findElement(By.linkText, "□□").click();
} else {
  driver.findElement(By.linkText("Sign in")).click();
}
```

3.3 Click a link by partial text

```
driver.findElement(By.partialLinkText("Recommend Seleni")).click();
```

3.4 Click a link by XPath

The example below is finding a link with text 'Recommend Selenium' under a `<p>` tag.

```
driver.findElement(By.xpath( "//p/a[text()='Recommend Selenium']")).click();
```

Your might say the example before (find by `linkText`) is simpler and more intuitive, that's correct. but let's examine another example:

First div Click here
Second div Click here

On this page, there are two 'Click here' links.

HTML Source

```
<div>
  First div
  <a href="link-url.html">Click here</a>
</div>
<div>
  Second div
  <a href="link-partial.html">Click here</a>
</div>
```

If test case requires you to click the second 'Click here' link, the simple `findElement(By.linkText("Click here"))` won't work (as it clicks the first one). Here is a way to accomplish using XPath:

```
driver.findElement(By.xpath("//div[contains(text(), \"Second\")]/a[text()=\"Click\
 here\"]")).click();
```

3.5 Click Nth link with exact same label

It is not uncommon that there are more than one link with exactly the same text. By default, Selenium will choose the first one. What if you want to click the second or Nth one?

The web page below contains three 'Show Answer" links,

1. Do you think automated testing is important and valuable? Show Answer
2. Why didn't you do automated testing in your projects previously? Show Answer
3. Your project now has so comprehensive automated test suite, What changed? Show Answer

To click the second one,

```
assert driver.findElements(By.linkText("Show Answer")).size() == 2;
driver.findElements(By.linkText("Show Answer")).get(1).click(); // 2nd link
```

`findElements` return a list (also called array) of web controls matching the criteria in appearing order. Selenium (in fact Java) uses 0-based indexing, i.e., the first one is 0.

3.6 Click Nth link by CSS Selector

You may also use CSS selector to locate a web element.

```
driver.findElement(By.cssSelector("p > a:nth-child(3)")).click(); // 3rd link
```

However, generally speaking, the stylesheet are more prone to changes.

3.7 Verify a link present or not?

```
assert driver.findElement(By.linkText("Recommend Selenium")).isDisplayed();
assert driver.findElement(By.id("recommend_selenium_link")).isDisplayed();
```

3.8 Getting link data attributes

Once a web control is identified, we can get its other attributes of the element. This is generally applicable to most of the controls.

```
WebElement seleniumLink = driver.findElement(By.linkText("Recommend Selenium"));
assert seleniumLink.getAttribute("href").equals(TestHelper.siteUrl() + "index.htm\
l");
assert "recommend_selenium_link".equals(seleniumLink.getAttribute("id"));
assert "Recommend Selenium".equals(seleniumLink.getText());
assert "a".equals(seleniumLink.getTagName());
```

Also you can get the value of custom attributes of this element and its inline CSS style.

```
assert "font-size: 14px;".equals(driver.findElement(By.id("recommend_selenium_lin\
k")).getAttribute("style"));
//  Please note using attribute_value("style") won't work
assert "123".equals(driver.findElement(By.id("recommend_selenium_link")).getAttri\
bute("data-id"));
```

3.9 Test links open a new browser window

Clicking the link below will open the linked URL in a new browser window or tab.

```
<a href="https://agileway.com.au/demo" target="_blank">Open new window</a>
```

While we could use `switchTo()` method (see chapter 10) to find the new browser window, it will be easier to perform all testing within one browser window. Here is how:

```
String currentUrl = driver.getCurrentUrl();
String newWindowUrl = driver.findElement(By.linkText("Open new window")).getAttri\
bute("href");
driver.navigate().to(newWindowUrl);
driver.findElement(By.name("name")).sendKeys("sometext");
driver.navigate().to(currentUrl); // back
```

In this test script, we use a local variable 'currentUrl' to store the current URL.

4. Resources

Recipe test scripts
http://zhimin.com/books/bought-selenium-recipes-java[1]

Username: `agileway`
Password: `SITEWISE12`

Log in with the above, or scan QR Code to access directly.

4.1 Books

- **Practical Web Test Automation**[2] by Zhimin Zhan

  Solving individual selenium challenges (what this book is for) is far from achieving test automation success. *Practical Web Test Automation* is the book to guide you to the test automation success, topics include:
  - Page object model
  - Functional Testing Refactorings
  - Cross-browser testing against IE, Firefox and Chrome
  - Strategies on team collaboration and test automation adoption in projects and organizations

- **Practical Continuous Testing**[3] by Zhimin Zhan

- **Selenium WebDriver Recipes in C#, 2nd Edition**[4] by Zhimin Zhan

  Selenium WebDriver recipe tests in C#, another popular language that is quite similar to Java.

[1] http://zhimin.com/books/bought-selenium-recipes-java
[2] https://leanpub.com/practical-web-test-automation
[3] https://leanpub.com/practical-continuous-testing
[4] http://www.apress.com/9781484217412

- **Selenium WebDriver Recipes in Ruby**[5] by Zhimin Zhan

  Selenium WebDriver tests can also be written in Ruby, a beautiful dynamic language very suitable for scripting tests. Master Selenium WebDriver in Ruby quickly by leveraging this book.

- **Selenium WebDriver Recipes in Python**[6] by Zhimin Zhan

  Selenium WebDriver recipes in Python, a popular script language that is similar to Ruby.

- **Selenium WebDriver Recipes in Node.js**[7] by Zhimin Zhan

  Selenium WebDriver recipe tests in Node.js, a very fast implementation of WebDriver in JavaScript.

- **API Testing Recipes in Ruby**[8] by Zhimin Zhan

  The problem solving guide to testing APIs such as SOAP and REST web services in Ruby language.

4.2 Web Sites

- **Selenium Java API** https://www.selenium.dev/selenium/docs/api/java/org/openqa/seleni summary.html[9]

- **Selenium Home** (https://www.selenium.dev[10])

4.3 Blog

- **Agile Way Blog** (https://agileway.substack.com[11])

[5] https://leanpub.com/selenium-recipes-in-ruby
[6] https://leanpub.com/selenium-recipes-in-python
[7] https://leanpub.com/selenium-webdriver-recipes-in-nodejs
[8] https://leanpub.com/api-testing-recipes-in-ruby
[9] https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/package-summary.html
[10] https://www.selenium.dev
[11] https://agileway.substack.com

I share my experience and views on Test Automation and Continuous Testing there.

4.4 Tools

- **Intellij IDEA IDE** (https://www.jetbrains.com/idea/[12])

  The most popular Java IDE, the community edition is free.

- **NetBeans IDE** (https://netbeans.org/downloads[13])

  Free Java IDE from Sun (now Oracle).

- **BuildWise** (https://agileway.com.au/buildwise[14])

  AgileWay's free and open-source continuous testing server, purposely designed for running automated UI tests with quick feedback.

[12] https://www.jetbrains.com/idea
[13] https://netbeans.org/downloads
[14] https://agileway.com.au/buildwise