Selenium WebDriver

From Foundations To Framework

Yujun Liang & Alex Collins

Table of Contents

Introduction

This book is a hands-on guide to dozens of specific ways you can use to get the most of WebDriver in your test automation development. This practical handbook gives you instantly-useful solutions for important areas like interacting with and testing web applications and using the WebDriver APIs. As you read, you'll graduate from WebDriver fundamentals to must-have practices ranging from how to interact with, control and verify web pages and exception handling, to more complex interactions like page objects, alerts, and JavaScript, as well as, mobile testing, and much more. Finally, you'll learn how to build your own framework. By the end of the book, you'll be confident and skilled at testing your web applications with WebDriver.

About the technology

Web applications are difficult to test because so much depends on the way a user interacts with individual pages. The Selenium WebDriver web testing framework helps you build reliable and maintainable test automation for your web applications across multiple browsers, operating systems and programming languages. Much like a human, it can click on links, fill out forms, and read the web pages, and unlike a human, it does not get bored. WebDriver can do nearly anything you ask it to—the trick is to come up with a unified approach to testing. Fortunately, that's where this book really shines.

What's inside

- Specific, practical WebDriver techniques
- Interacting with, controlling, and testing web applications
- Using the WebDriver APIs
- Making maintainable tests
- Automated testing techniques

About the reader

This book assumes you're comfortable reading code in Java or a similar language and that you know the basics of building and testing applications. No WebDriver experience is required.

Errata and Discussion

If you find any errors or problems with this book, or if you want to talk about the content:

https://github.com/selenium-webdriver-book/manuscript/issues

About the authors

Yujun Liang is a Technical Agile Coach who teaches agile software development technologies including test automation using Selenium WebDriver. He used to work for ThoughtWorks and helped clients build automation testing for web applications with rich user interaction and complex business logic.

Alex Collins is a Technical Architect in the UK, a technology blogger, public speaker, and OSS contributor. Alex has been working with Selenium WebDriver since 2011.

Part 1: Fundamentals

In this section we will introduce you to Selenium WebDriver. We’ll teach common techniques that are useful in writing tests, such as locating, interacting and verifying elements. We’ll also show you how to make your code more maintainable using Page Objects and how to deal with errors* You’ll be able to write code for many common web pages by the end of it.

Chapter 1: First Steps

This chapter covers

- What is WebDriver?
- Why choose WebDriver?
- "Hello WebDriver!"

Nowadays, more and more business transactions are carried out on the Internet through web pages built by people. Some websites are simple enough that they can be set up by one or two people, but some websites are so complex that they are built by hundreds or even thousands of developers. Before each release, the site must be tested to make sure it is free of critical bugs. It is time-consuming to test the whole site manually, and as the site grows, so does the cost of testing. More than that, as time passes, a new feature that was well-tested when it first became available may be forgotten about later—we risk of a loss of consistency and quality, and as a result bugs in what we thought were solid pieces of functionality creep in.

In the textile industry, manual labor dominated the process of making clothes for a long time. When weaving machines were invented, productivity improved dramatically.

The same thing is happening in software testing. Just as weaving machines changed the textile industry, we are now building "automatic testing machines" to replace manual testing, to improve the productivity, quality, and consistency of the software.

Since its inception in 2008, **Selenium WebDriver** (also known as Selenium 2) has established itself as the de facto web automation library.

Before Selenium WebDriver, there was Selenium 1.0, which enabled automation by injecting JavaScript into web pages. WebDriver is a re-invention of that idea, but is more reliable, more powerful, and more scalable.

Selenium has evolved, and so has the World Wide Web. **HTML5** and **CSS3** are now standard; AJAX rich web applications are no longer even cutting edge. This means that web automation is now a more complex and interesting topic.

This chapter will rapidly cover the basics, making sure that by the end of it you understand the basic architecture can write basic code.

In this chapter we'll introduce WebDriver, what it is, how it works, and reasons for choosing it. We'll also briefly talk about some of the tools we used in this book, the ones we'd recommend to all developers.

What is WebDriver?

Selenium WebDriver automates web browsers. It sits in the place of the person using a web browser. Like a user, it can open a website, click links, fill in forms, and navigate around. It can also examine the page, looking at elements on it and making choices based on what is sees.

The most common use case for WebDriver is automated testing. Until recently, to run a regression test on your website, you'd need to have a set of scripts that would have to be manually executed by developers or QAs. Any reports would need to be manually collated too. This can be both time-consuming and costly. Instead, WebDriver can be used to execute those scripts, and automatically gather reports on how successful they were, at the push of a button. Each subsequent execution will be no more expensive than the first.

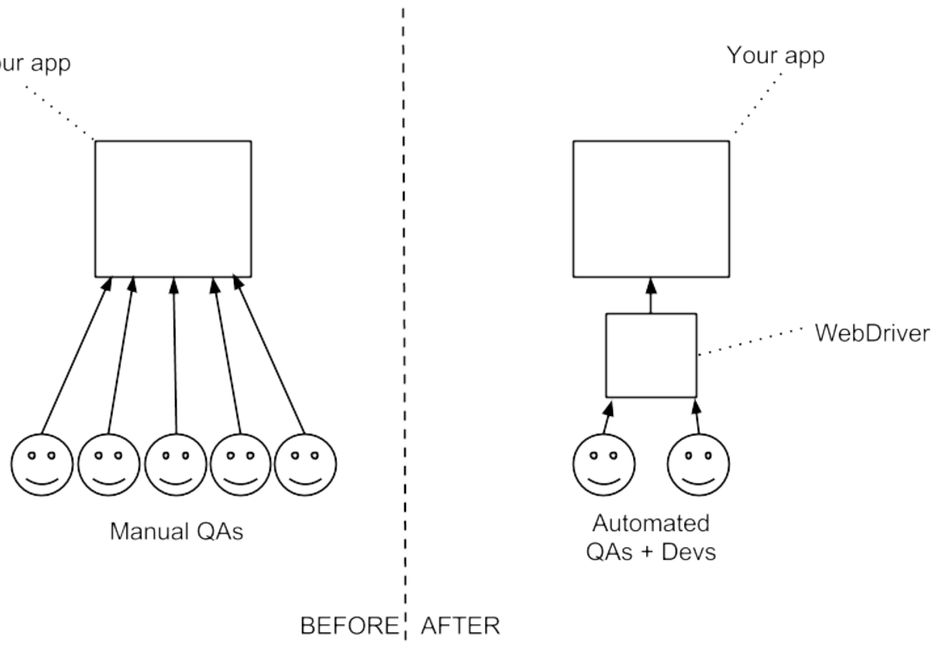


Figure 1. Before WebDriver

Long gone are the days when you needed to create one version on your website for the pervasive and notoriously standards non-compliant Internet Explorer 6, and another for other browsers. While most modern browsers are much more consistent in their behavior, the way a web page looks or acts can still greatly vary as the number of different browsers, operating system, and platforms in common use has increased. You can still have a high-value customer complain that they can’t access your site. Historically, the only way to mitigate this was to have army of QAs manually test on a variety of different configurations, a time-consuming and costly process. WebDriver can run tests on different operating systems and different browser configurations, and in a fraction of the time of a human being. Not only that, you can use it to run them much more consistently and reliably than a manual tester.

Applications and websites provide useful services, but sometimes these are only accessible by web pages. Another use case for WebDriver is to make those pages accessible to applications via WebDriver. You might have an administration application written several years ago and a client or Product Owner has asked for some actions on it to be automated. But maybe no one knows where the source code is. It might be much easier to use WebDriver to automate this task.

How WebDriver works

WebDriver works in all major browsers and with all major programming languages. How is this possible? Well, WebDriver has several interacting components:

1. A **web browser**.
2. A **plugin or extension** to the browser that lives inside the browser, which itself contains a server that implements the WebDriver JSON API.
3. A **language binding** (in our case Java) that makes HTTP requests to that API.

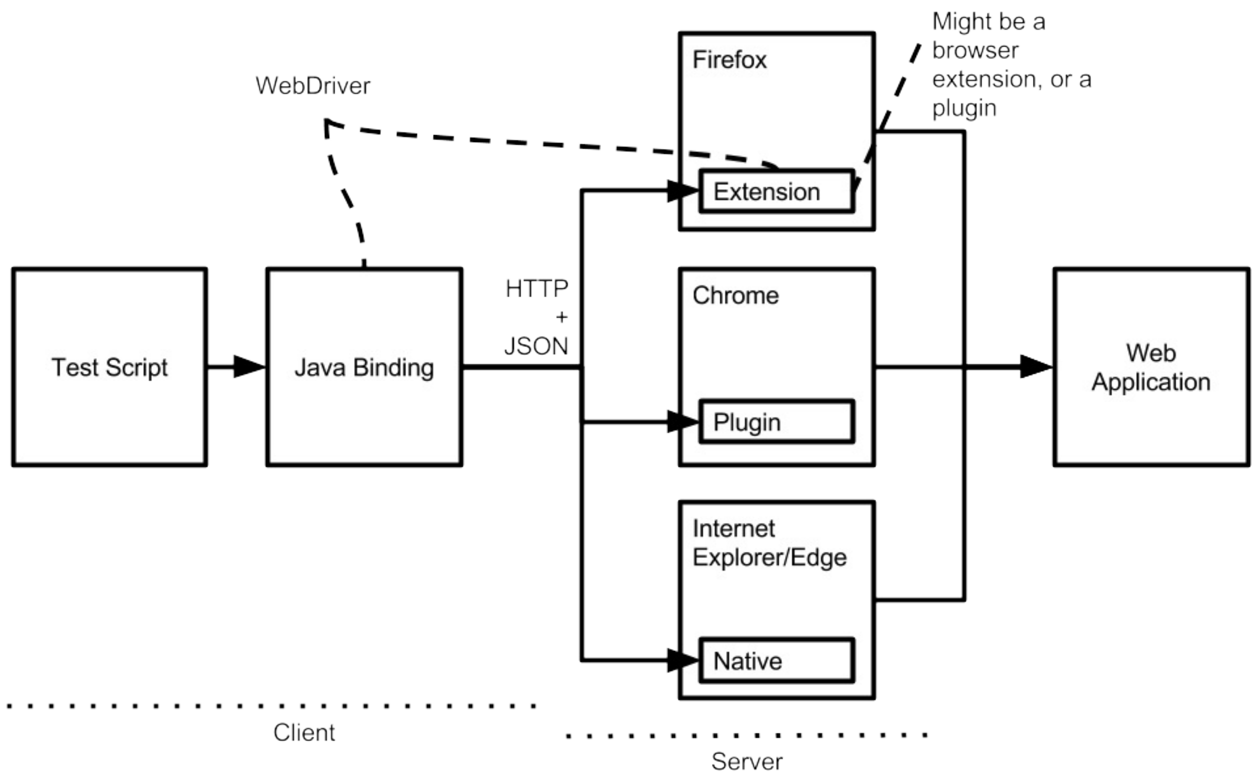
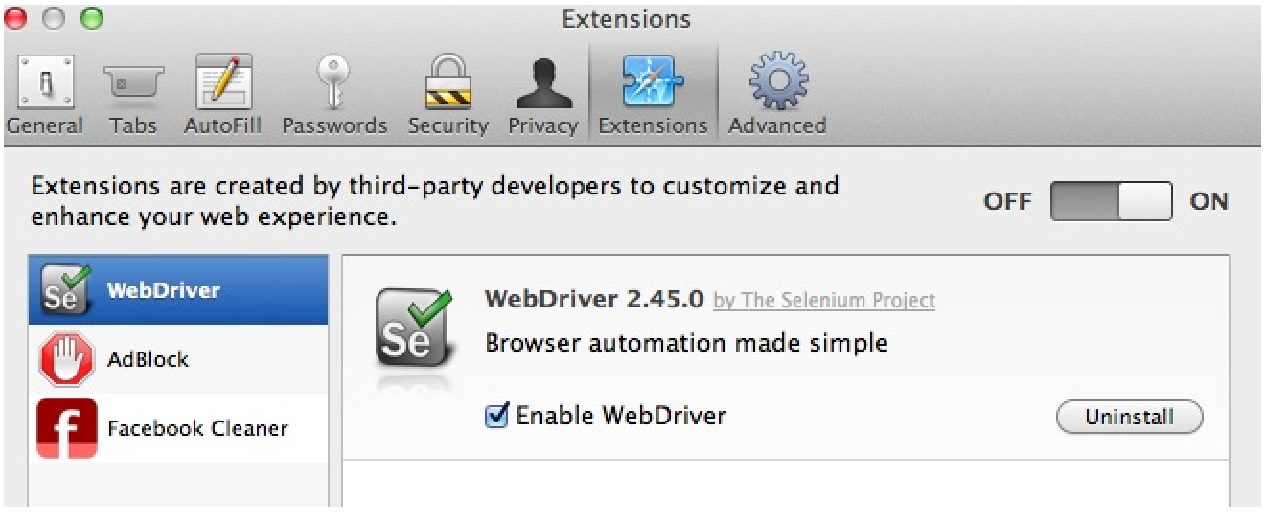


Figure 2. Web driver diagram

When you start code that uses WebDriver, it will open up the browser, which in turn starts the plugin. You can then send requests to perform the actions you want, such as clicking on links or typing text. As a plugin only needs to implement the JSON API, people have written plugins for all major browsers. To use a browser that has a plugin, you just need to implement a client to the JSON protocol.

This means that all the major browsers and all the major programming languages support WebDriver.

The plugin can usually be seen in the browser’s preferences, such as in figure 1.3.



Extensions are created by third-party developers to customize and enhance your web experience.

OFF ON

WebDriver
AdBlock
Facebook Cleaner

WebDriver 2.45.0 by The Selenium Project
Browser automation made simple
☑ Enable WebDriver
Uninstall

Figure 3. Safari Extensions panel

Why choose WebDriver?

There a number of great reasons to choose WebDriver:

- WebDriver can run browsers **locally and remotely** with minimal configuration changes.
- WebDriver is **supported by major browser vendors**: both Firefox and Chrome are active participants in WebDriver's development.
- WebDriver more closely mimics a user. Where possible it uses **native events** to operate, to make it accurate and stable.
- WebDriver is **Open Source Software (OSS)**. This means that it is both free and is supported by an excellent community.
- WebDriver supports all major operating systems such as **OS X**, **Windows**, and **Linux**. It also has support for **Android** and **iOS**.
- WebDriver is going to become a **W3C** standard, so you can expect that it will be supported for a long time.
- WebDriver doesn't suffer from some of the problems that Selenium 1.0 suffered from, such as with uploading files, or handling pop-ups.
- WebDriver has a more **concise syntax** than Selenium 1.0, making it faster to write code.

What WebDriver cannot do

WebDriver provides a way to control a web browser, but that is all. When you buy a new car, you get a manual that will tell you how to operate the radio and how change the oil. But that manual won't tell you the best place to get your car serviced, or teach you how to drive. Like driving a car, there are things you must do for yourself. Here are some things WebDriver does not do:

- WebDriver doesn't have the control of the timing of the elements appearing on web page. Some might appear later and you'll need to handle this yourself.
- WebDriver does not know when things have changed on the page, so you can't ask it to tell you when things have changed.
- WebDriver doesn't provide many utilities for writing your code. You need to write these yourself.
- WebDriver doesn't provide built-in support for page elements that are composed of multiple elements, such as JavaScript calendars.
- WebDriver does not provide a framework to write your code in. JUnit is a natural choice.
- WebDriver doesn't manage the browser. For example, you need to clean up after you have used it.
- WebDriver won't install or maintain your browsers. You need to do this yourself.

We'll cover all these important tasks in this book.

The history of Selenium

Selenium is a suite of web testing tools, including **Selenium IDE**, **Selenium RC**, Selenium WebDriver, and **Selenium Grid**. The earliest Selenium is called Selenium Core, which came out of ThoughtWorks's Chicago office developed by **Jason Huggins**. It was written to simulate a human user's action with Internet Explorer. It was different from Record/Replay type of tools from other vendors, since it didn't require an additional GUI tool to support its operation. It just needed Java, which most developers already installed on their machines.

Later, **Shinya Kasatani** developed a Firefox plugin called **Selenium IDE** on top of Selenium Core. Selenium IDE is a graphic interface allowing user to record a browser navigation session, which can be replayed afterward. Selenium IDE integrated with Firefox and provided the same Record/Replay function as the other proprietary tools.

Since Selenium IDE is a free tool, it soon captured a big market share among QAs and business analysts who didn't have the necessary programming skills to use Selenium Core. Later Selenium Core evolved into Selenium RC ("RC" meaning "Remote Control"), along with Selenium Grid, which allowed tests can be run on many machines at the same time.

Selenium WebDriver was originally created by Simon Stewart at Thoughtworks. It was originally presented at Google Test Automation Conference, and this can still be seen online https://www.youtube.com/watch?v=tGu1ud7hk5I.

In 2008, Selenium incorporated WebDriver API and formed Selenium 2.0. Selenium WebDriver became the most popular choice among the Selenium tool suite, since it offers standardized operation to various Browsers through a common interface, WebDriver. In favor of this new simplified WebDriver API, Selenium RC has been deprecated and its usage is no longer encouraged. The developers who maintain Selenium also provided a migration guide helping Selenium RC users migrating from Selenium RC to WebDriver.

Today, when people talk about "Selenium," they're usually talking about Selenium WebDriver.

Why it is called Selenium?
Jason Huggins joked about a competitor named Mercury in an email, saying that you can cure mercury poisoning by taking selenium supplements. That's where the name Selenium came from.

The tools you need to get started

On top of a computer, access to the Internet, and a development environment, you will need some additional pieces of software to get started.

Why we use Java in this book
While we're aware that many developers won't be using Java as their main language. We chose it as the language for this book because we wanted to write for the most people possible, and Java is the most popular programming language.

The API to WebDriver is similar in all languages. The languages of the web — JavaScript, CSS, and HTML — are the same regardless of the language you're writing your tests in. If you're using one of the many languages that WebDriver supports, such as C#, Ruby or Python, you should be able to replicate many of the techniques in this book.

Java Development Kit (JDK)

As Java is among the most popular and widely used development languages, we will be using it throughout this book. Java 8 introduces a number of features, such as streams and lambda expressions, that make it faster and more efficient to work with WebDriver.

You can check to see if (and which version of) the JDK is already installed from a terminal using the `javac` command:

```
$ javac -version
javac 1.8.0_20
```

Note that it is typical to refer to a Java version by the middle digit of the full version number, so "Java 1.8" is usually known as "Java 8."

Linux users can install Java using the `yum` or `apt` package managers. Windows and OS X users can download it from Oracle at http://www.oracle.com/technetwork/java/javase/downloads/index.html.

Apache Maven

Throughout this book we will use the Apache Maven build tool for managing our code. The main reason for this is because Maven can manage the many dependencies that we need to create an application. You can check to see if you have Maven installed from the terminal:

```
$ mvn -version
Apache Maven 3.3.1
```

If you do not have it installed, Linux users can install it using their package manager (e.g. Apt or Yum), OS X users can install it using the Homebrew package manage (http://brew.sh)

For example (on OS-X using Brew):

```
brew install maven
```

Or (on Ubuntu Linux):

```
sudo apt-get install maven
```

Windows users can download it from the Apache Maven website at https://maven.apache.org.

Google Chrome

Çhrome browser is the best supported browser. Unlike other browsers, it is available on every platform, it's standards compliant, and has the simple out-of-the-box integration with WebDriver.

As usual, Linux users can install Chrome using their package manager; otherwise you can download Chrome from Chrome.

Later on the book we will look at other browsers such as Firefox, but having Chrome installed now will get you through the first few chapters.

Git

You'll need to install Git if you want to check out the source code for this book. On OS-X using Brew:

```
brew install git
```

Or on Ubuntu Linux using Apt:

```
sudo apt-get install git
```

If you use Windows, you can download it from https://git-scm.com/ .

The test project

As part of this project, we have put all the source code into the Git version control system. This contains all the sample code, as well as a small website the code runs against.

You can get this by running these commands:

```
git clone https://github.com/selenium-webdriver-book/source.git
cd source
```

The project has a built-in web server that can be started by entering the following:

```
mvn jetty:run
```

You can view the website it creates at http://localhost:8080/hello-webdriver.html. You should see a page similar to figure Hello WebDriver. This forms the basis of many of the tests in the project, so you'll probably want to keep it running all the time.

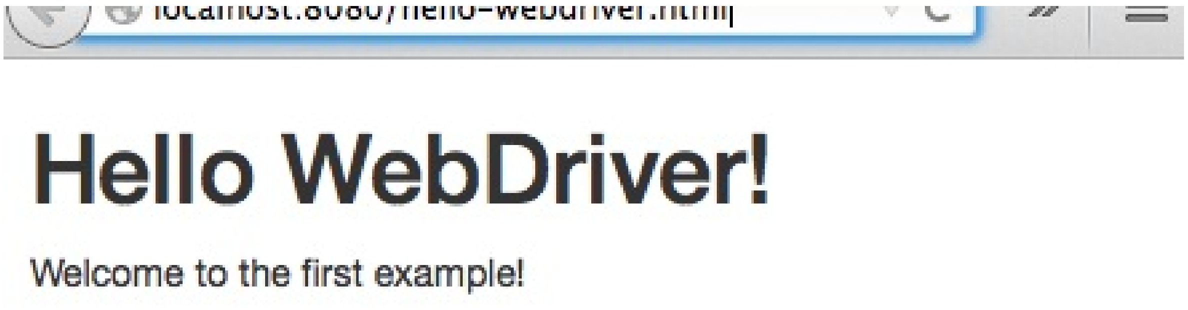

localhost:8080/hello-webdriver.html
Hello WebDriver!
Welcome to the first example!

Figure 4. Hello WebDriver

When you are done, press `Ctrl+C` to quit the server.

If you want to find examples from the book in the code, look for the package named after the chapter. For example, if you're looking for chapter one's examples, then they can be found in `src/test/java/swb/ch01intro`.

To run all the tests with the book, run the following:

```
mvn verify
```

Instructions on how to run with different browsers can be found in the `README.md` file.

"Hello WebDriver!"

Let's look at an example of using WebDriver to automate a basic task, and end up with a working example. A WebDriver automation script usually consists of several operations:

1. Create a new WebDriver, backed by either a local or remote browser.

2. Open a web page.
3. Interact with that page, for example clicking links or entering text.
4. Check whether the page changes as expected.
5. Instruct the WebDriver to quit.

Create a directory with this `pom.xml`:

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://maven.apache.org/POM/4.0.0"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>hello-webdriver</groupId>
    <artifactId>hello-webdriver</artifactId>
    <version>1.0.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-chrome-driver</artifactId> (1)
            <version>LATEST</version> (2)
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.3</version>
                <configuration>
                    <source>1.8</source> (3)
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
```

```
                <artifactId>maven-failsafe-plugin</artifactId>
                <version>2.18.1</version>
                <executions>
                    <execution>
                        <goals>
                            <goal>integration-test</goal> (4)
                            <goal>verify</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

1. You can choose a different browser, e.g. `selenium-firefox-driver` is for the Chrome browser.
2. Always use the latest version that is available.
3. Compile using the latest version of Java – Java 1.8.
4. Make sure that tests are run using Maven's failsafe plugin.

To start the driver, you'll need a special binary program to start it up. For Chrome, this is called `chromedriver` and can be found at https://sites.google.com/a/chromium.org/chromedriver/downloads. Download it and then save it into the root of the project.

Create `src/test/java/swb/intro/HelloWebDriverIT.java`:

HelloWebDriverIT.java

```
package swb.ch01intro;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import org.openqa.selenium.By; (1)
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

import static org.junit.Assert.assertEquals;

public class HelloWebDriverIT { (2)

    private WebDriver driver;

    @Before
    public void setUp() throws Exception {
        System.setProperty("webdriver.chrome.driver", "chromedriver"); (3)
        driver = new ChromeDriver(); (4)
    }

    @After
    public void tearDown() throws Exception {
        driver.quit(); (5)
    }

    @Test
    public void helloWebDriver() throws Exception {

        driver.get("http://localhost:8080/hello-webdriver.html"); (6)

        WebElement pageHeading
            = driver.findElement(By.tagName("h1")); (7)

        assertEquals("Hello WebDriver!",
            pageHeading.getText()); (8)
    }
}
```

1. Standard Java imports for WebDriver.
2. We use the IT suffix for test in this book; this is the Maven convention for integration tests that run using the Failsafe plugin [1].
3. Tell web driver via this system property the location of the driver binary.
4. Create a new driver which connected to an instance of the Chrome browser.

5. Make sure that the browser quits when the test finishes.
6. Open a web page in the browser.
7. Locate an element on the current page, in this case the page's heading.
8. Verify that the heading is the value you expect.

You'll need to start up the test project as shown in the previous section before you run the test. Then, when you run the test, you should see the browser open a page similar to figure Hello WebDriver.

Summary

- You can use WebDriver to save time and money by automating browser tasks.
- It is especially suited to automated browser testing.
- WebDriver is built around a standard JSON protocol, and that means all major browsers and languages support it.
- There are some great reasons to use WebDriver over manual testing. For example, you can save costs and improve quality at the same time.
- You need some tools to get started. We'll be using Maven and Java in this book.

In the next chapter we will start out on our journey by looking at the first part of any automation script—locating elements on pages.

1. https://maven.apache.org/surefire/maven-failsafe-plugin/