APPENDIX B: SELENIUM FRAMEWORK TUTORIAL

This appendix is a tutorial about implementing the Selenium framework for black-box testing at user level. It also contains code examples on how to use Selenium.

The goal with this tutorial is to show how to implement and use the Selenium testing framework. Selenium is a black-box testing framework that focuses on testing the web-based user interface of a system without the need of learning a scripting language. It accomplishes this in different ways and some of these are brought up in this tutorial.

Before using this tutorial, it is assumed that NetBeans 7.0.1 or above has been installed together with Mozilla Firefox web browser 26.0 or above. The user should have access to the Java EE Web project, The Recruitment System.

This tutorial has only been tested on a PC running Windows 7. It has not been verified to work on other operating systems but there should not be any major differences since this tutorial focuses on NetBeans and Mozilla Firefox.

1.1 Downloading the necessary files.

Visit the Selenium official website at: http://docs.seleniumhq.org/download/
Download and install the latest Selenium IDE release, called selenium-ide-x.y.z.xpi. A link to a direct download and install is available in the main page. Firefox will prompt to restart. Do so. When downloading the plug-in, Firefox may ask to allow installing of third party software as shown in Figure 1. Click *accept*.

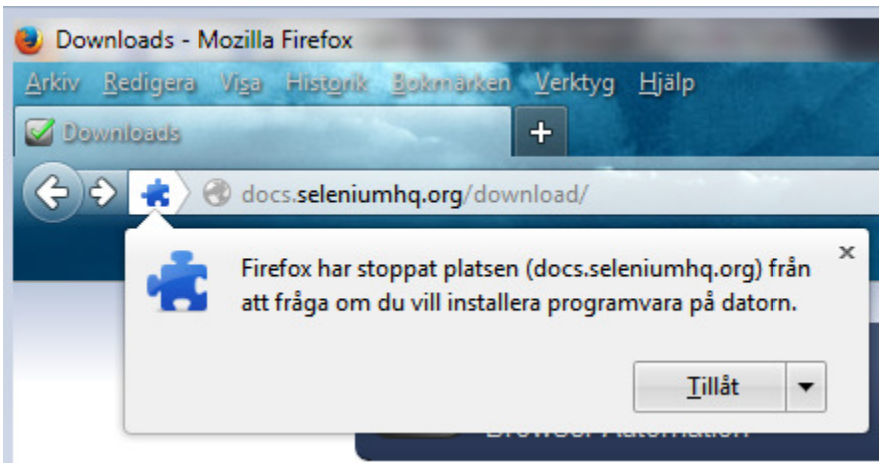

Figure 1 – The pop up that needs to be confirmed for the Selenium plug-in to be installed.

Also, download the zip file for the Java language bindings found under *Selenium Client & WebDriver Language Bindings* further down on the same download page (Figure 2). The file is called selenium-java-w.xy.z.zip. Save the zip-file to a location of your choice, preferably on the desktop.

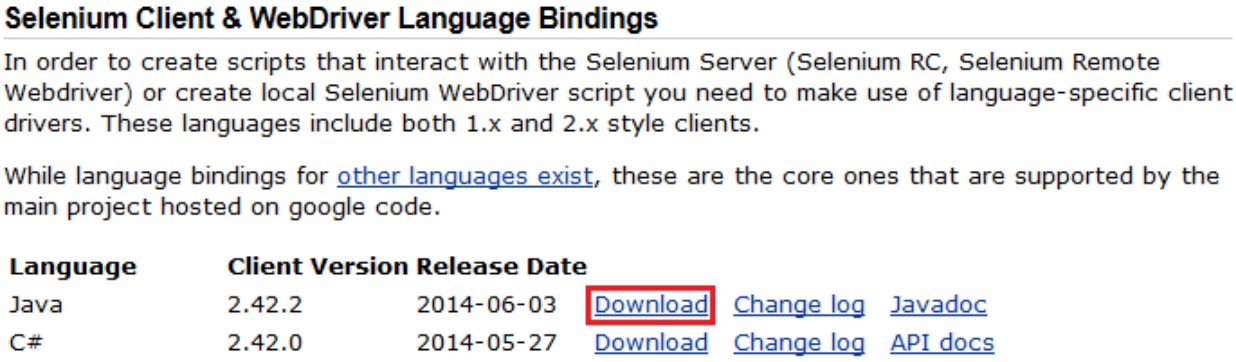

Figure 2 – Download ink to Java language bindings zip file.

Extract the contents of the zip file.

The Selenium IDE plug-in is for Firefox and it will be installed and used within the Firefox container. The zip file contains several files. Among them two jar folders which will be imported into the web project in NetBeans IDE later on. These will give you access to the Selenium testing framework in NetBeans.

1.2 Implementing Selenium to Firefox and NetBeans.

There are different ways to test your Java EE web project interface with Selenium. We have chosen to focus on three of them. Only one of these methods described below should be followed to avoid confusion.

The first method is to use Selenium IDE through the Firefox plug-in to record or script your own interactions with the website. The recording or script can be run in the plug-in directly and this will constitute your test. To implement this plug-in, simply download the Selenium IDE and allow Selenium to install the plug-in to the Firefox browser. Once the Firefox plug-in is downloaded, it is installed and you are prompted to restart the browser. Once restarted, it has access to Selenium. These steps are done in activity 1.1. We will come back to the recording tool in activity 1.3.

The second method is to use the Selenium framework through NetBeans IDE by exporting a recording or written script as Java code and run the test in NetBeans instead of the Firefox plug-in. This method is explained from activity 1.2.2.

The third method is to implement a Selenium test solely in NetBeans. With this alternative, you have the ability to code your own test case and not depend on the recording tool provided through Selenium IDE. This is explained in activity 1.2.1.

- To clarify, if you want to record and play back a test solely using the Selenium IDE plug-in and not worry about manually coding the test, please jump to activity 1.3 Recording with Selenium IDE plug-in

- If you want to export a recording to Java code in order to execute the test in NetBeans IDE, please jump to activity 1.2.2 Exporting recording to Netbeans IDE.

- If you want to manually code your own Selenium test solely in Java, please jump to activity 1.2.1 Creating test through Netbeans IDE.

Figure 3 illustrates the different paths that you can take when implementing Selenium with this tutorial. The boxes describe the methods and the circles are the different chapters, called activities, to follow for each of the three methods. Each path is also color-coded so that they are easier follow. For example, at the end of activity 1.3 Recording with Selenium IDE plug-in you can either run the recording in the Selenium IDE plug-in or export it to Java code and execute it in NetBeans IDE. This depends on which method you choose and it is illustrated by the two differently colored lines coming out of the activity circle 1.3. It is emphasized once again that only one method should be followed to avoid confusion.

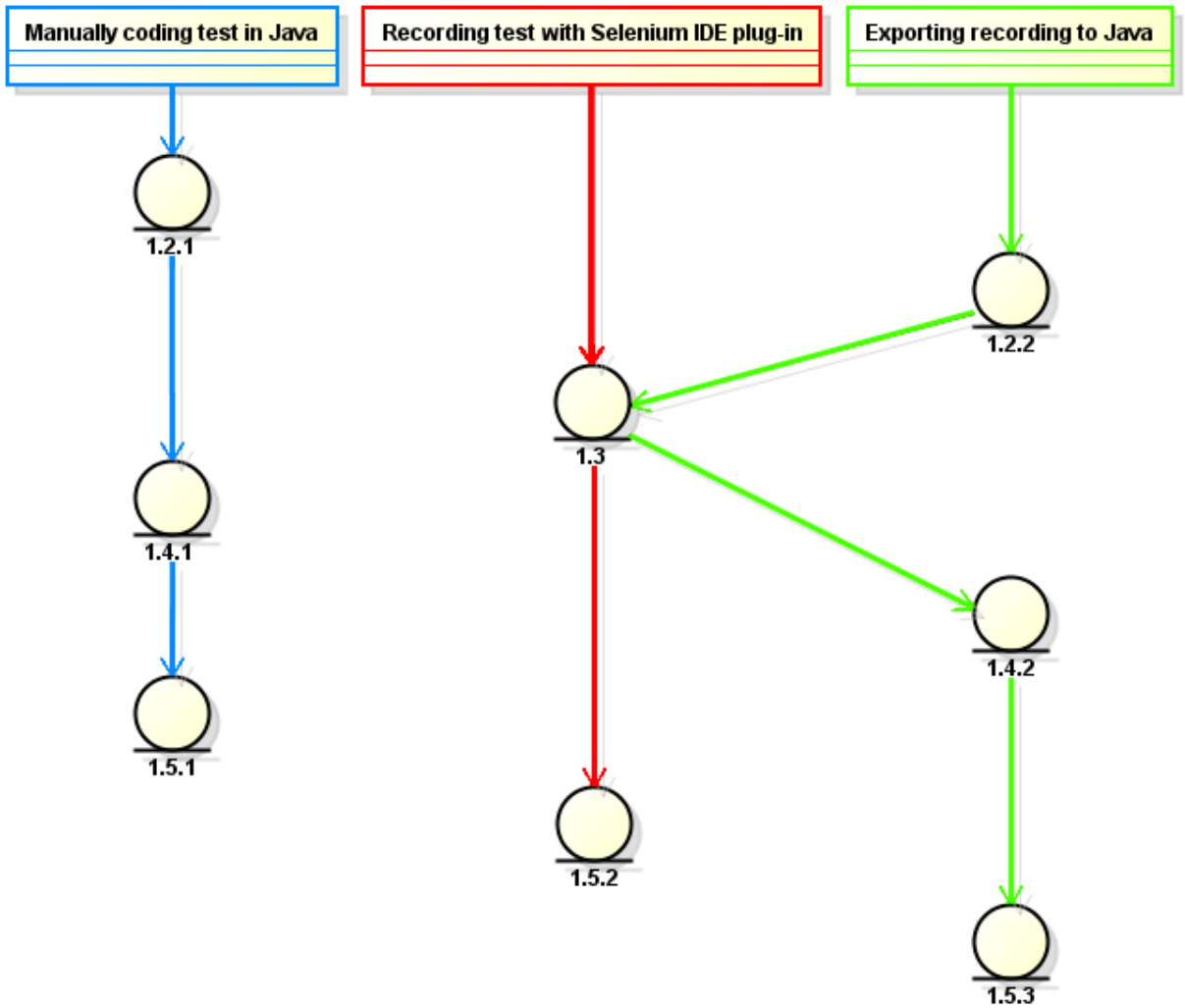

Figure 3 – Possible implementation paths in this tutorial.

1.2.1 Creating test through Netbeans IDE

This activity explains how to import the Selenium framework into the NetBeans IDE in order to be able to run manually written tests. It also shows how to create a new Java project for the test case. The main issue here is that a java class with a main method must be coded manually but more on this later. Here, it is only explained how to enable Selenium as a testing tool in a new Java project. Follow the directions as described below.

Start NetBeans.

Click on *File* and then click on *New Project…*
Choose the category *Java* on the left hand side under *Categories* and choose project *Java Application* (Figure 4).

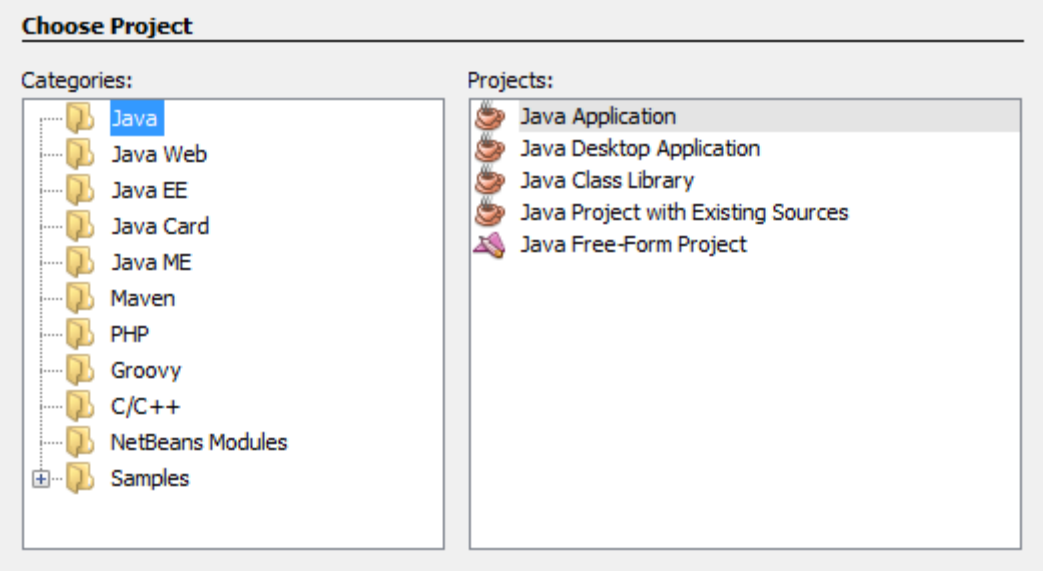

Figure 4 – Creating a new Java Application.

Click *Next* >.

Call your new Java project *"seleniumFirstMethod"* and click *Finish*.

Netbeans IDE will now show your new Java project with a Java class, containing a main method. To access your newly created Java project go to your *Projects*-tree usually located on the left hand side of NetBeans IDE.

Once the java project has been located, right-click on the folder *Libraries* found in the newly created Java Application (Figure 5) and choose "*Add Jar/Folder*" (also shown in Figure 5).

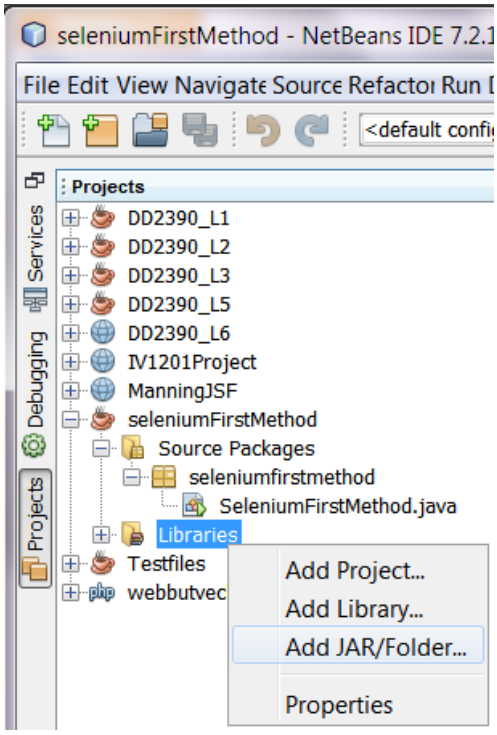

Figure 5 – The new Java Application.

Navigate on your PC to the folder which contains the extracted files from the Selenium library zip file. Choose a file by left-clicking it and then click *Open*. Repeat this process for each file to be added. The names of the files that should be imported are listed below:

- *selenium-java-w.xy.z-srcs.jar*
- *selenium-java-w.xy.z.jar*
- All jar files in the subfolder called *libs*. Do this quickly by left-clicking on the first jar file and then press *Ctrl +A* to select all of them.

If imported correctly, the *libraries* folder should look something like this (Figure 6):

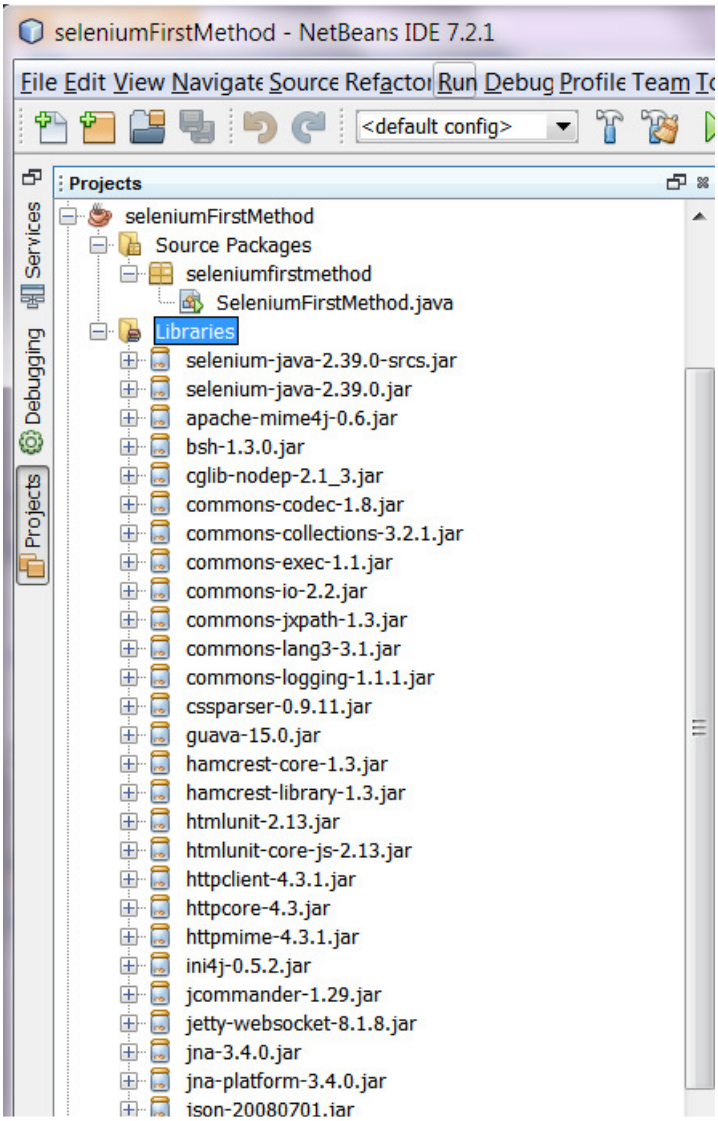

Figure 6 – The jar files for Selenium added under libraries

You are now able to use Selenium as a testing tool through the Netbeans IDE. You can now jump to activity 1.4.1 Guidelines for a manually coded test.

1.2.2 Exporting recording to Netbeans IDE

In this activity, a new Java project is created. Here, the Selenium framework is imported as a testing framework into NetBeans IDE. Also, a java class with a main method is not needed when creating a new Java project. This way of importing the Selenium Framework as a testing framework is required in order to later be able to export a recording and run it in NetBeans IDE.

To import the Selenium library into NetBeans IDE, follow the directions as described below.

Start NetBeans.

Click on *File* and then click on *New Project…*

Choose the category *Java* on the left hand side under *Categories* and choose project *Java Application* (Figure 7).

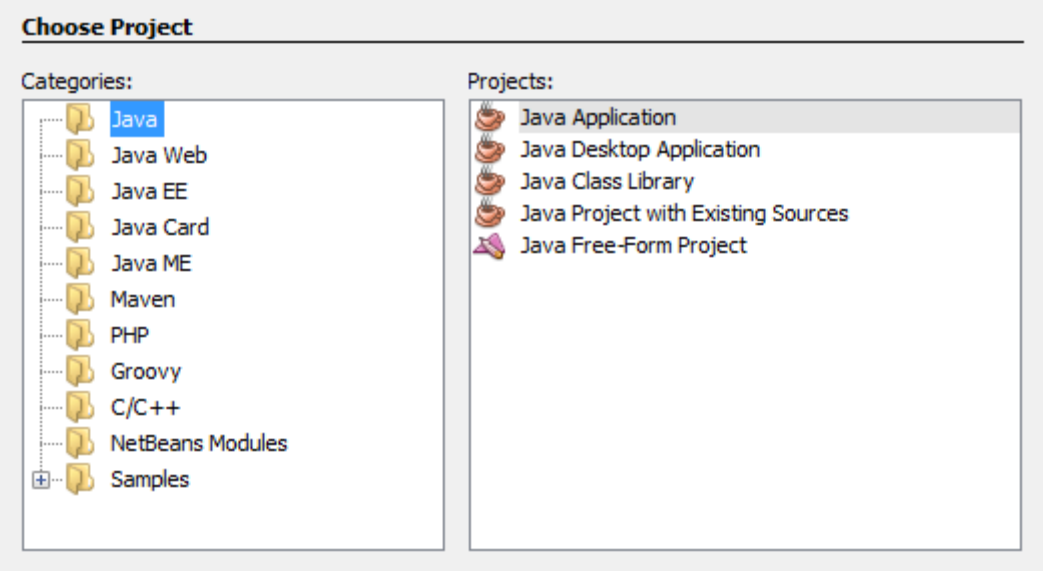

Figure 7 - Creating a new Java Application.

Click *Next* >.

Name your project *"seleniumSecondMethod"*

Since a main class is not needed, please uncheck the option for it as shown in Figure 8 below.

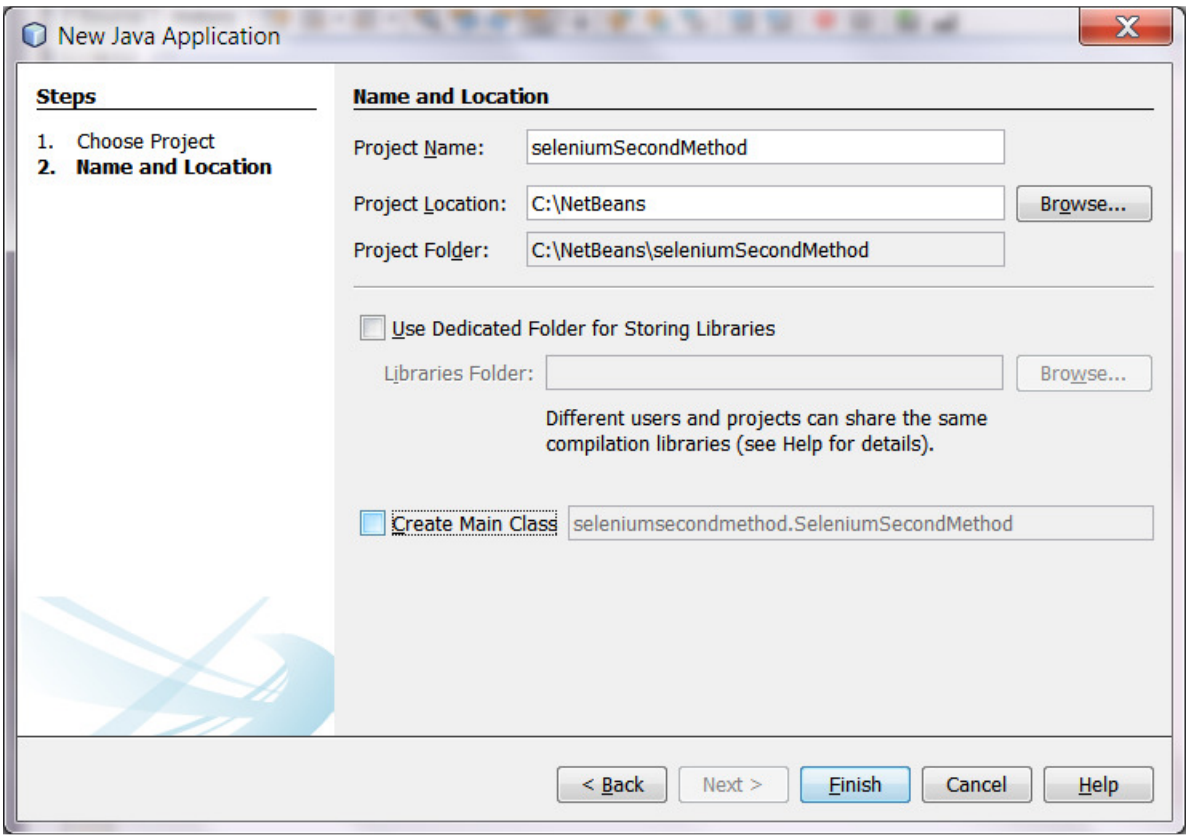

Figure 8 – uncheck the checkbox called *Create Main Class*.

Click *Finish*.

Netbeans IDE will now show your new Java project tree. Since there is no main class, the source packages is empty.

Now, it is time to create a unit test class in order to import the Selenium framework as a testing framework. Right-click on your Java project, at the root of the project tree and navigate to *New* and then click *Other…*

Choose the category *Unit Tests* on the left hand side under *Categories* and choose the file type *JUnit Test* and click *Next* >. On earlier Netbeans builds the category name is called *JUnit.*

Call your new unit test class *"SeleniumUnitTest"* and click *Finish*. Notice the warning displayed that you should not add your java classes in the default package. It is good practice to follow this advice. However, not doing so will not affect the examples in this tutorial (Figure 9).

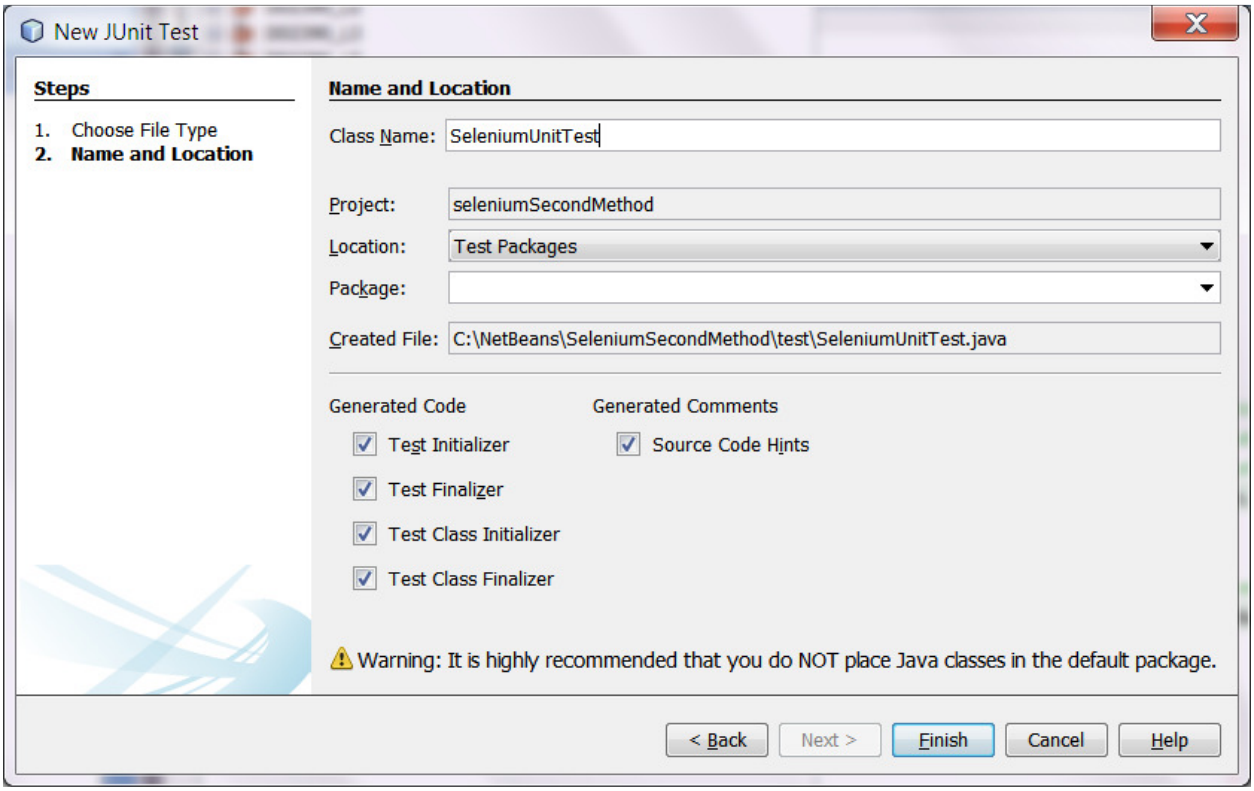

Figure 9 – Creating a JUnit test class. Ignoring the warning will not affect the tutorial.

If more than one version of the JUnit testing framework is available, NetBeans will ask which one to use when creating the unit test class (Figure 10). For this tutorial JUnit 4.x was used. It has not been tested with earlier versions of JUnit.

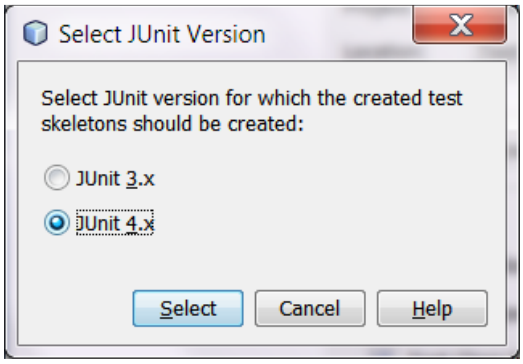

Figure 10 – Selecting JUnit version. Window will only show if more than one version is available.

Once the java project has been located under *Projects*, right-click on the folder *Test Libraries* located in the newly created Java Project (Figure 11). Notice that the *Source packages*-folder is empty and a new Java folder called *Test packages* has appeared. This folder was created when a new Unit test class for the JUnit testing framework was created (Figure 9).

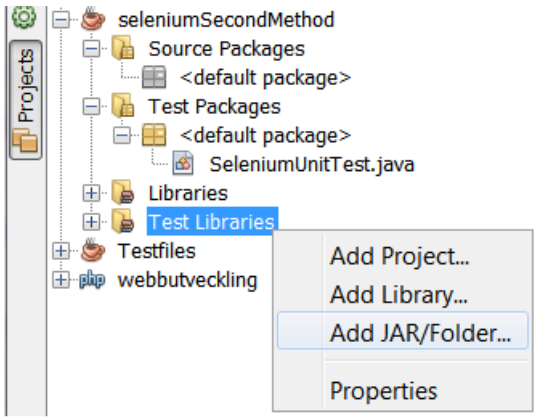

Figure 11 - The new Java Application.

Click *Add Jar/Folder* (also shown in Figure 11) and then navigate on your PC to the folder which contains the extracted files from the Selenium library zip file. Choose a file and click Open. Repeat this process for each file to be added. The names of the files that should be imported are listed below:

- *selenium-java-w.xy.z-srcs.jar*
- *selenium-java-w.xy.z.jar*
- All jar files in the subfolder called *libs*. Do this quickly by left-clicking on the first jar file and then press *Ctrl +A* to select all of them.

If imported correctly, the *Test Libraries* folder should look something like this (Figure 12):

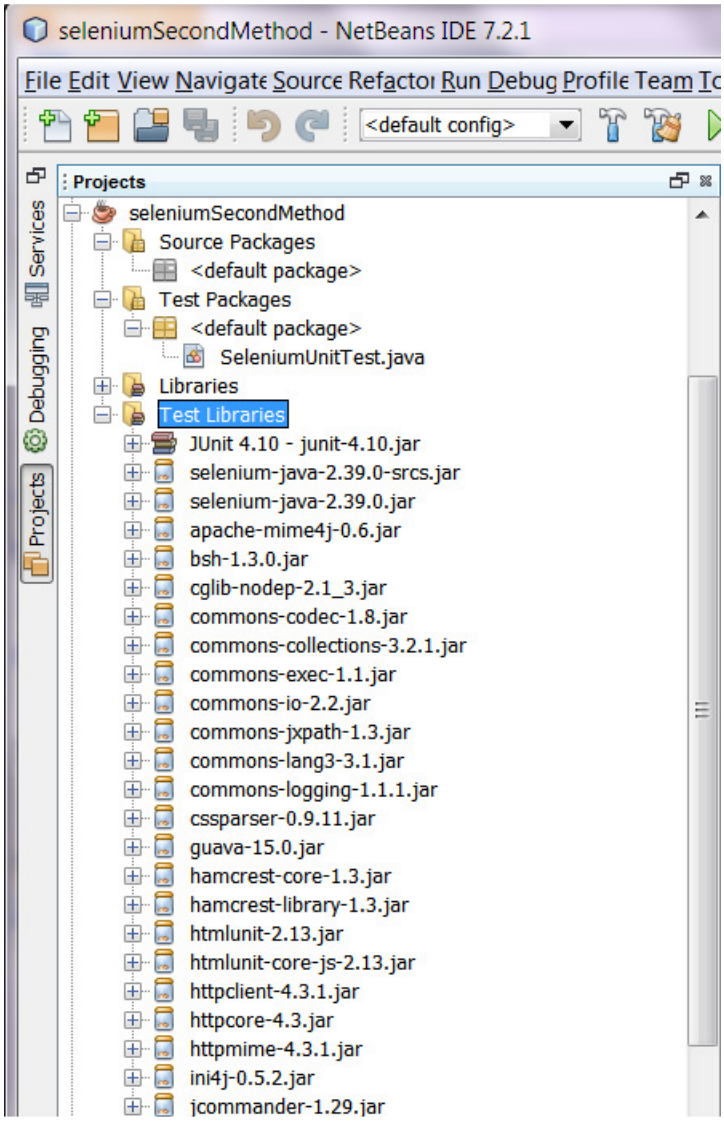

Figure 12 – The jar files for Selenium added under libraries

You are now able to use Selenium as a testing tool. Please continue to activity 1.3 Recording with Selenium IDE plug-in.

1.3 Recording with Selenium IDE plug-in

In this activity, it is shown with examples how to do a black-box test with Selenium. Make sure to deploy your Java EE Web project through NetBeans IDE to be able to access the website of the project.

Open up the Mozilla Firefox browser and go to the website of your Java EE Web project, usually found at *http://localhost:8080/Projectname/* where "*Projectname*" is the name of your Java EE Web project.

Click on the Selenium IDE plug-in icon on the top right corner the Firefox browser window (Figure 13).

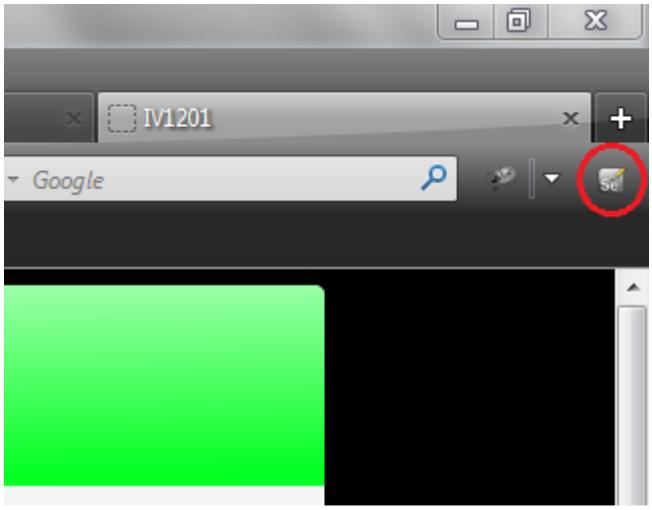

Figure 13 – The Selenium IDE icon.

The code examples are based off of the Java EE Web project interface that is included in the source files. The web interface is built with JSF pages and can be found under the source folder *Web Pages*. As mentioned earlier, this tutorial will focus on two ways to test with Selenium, through Mozilla Firefox and NetBeans IDE.

When clicking on the Selenium IDE plug-in icon the program is executed and new window is presented in a new Firefox browser instance (Figure 14). Here, it is possible to manually enter a variety of commands or record an interaction with the website.

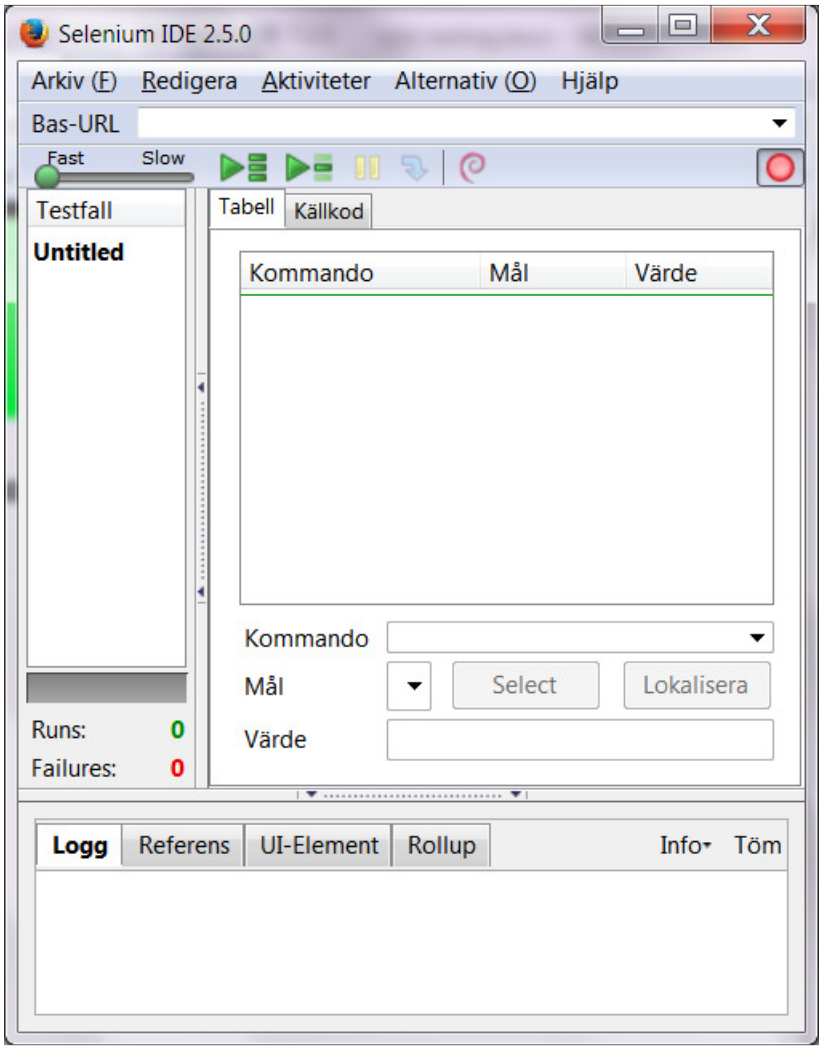

Figure 14 – Selenium IDE window through Firefox. Notice that recording is on (red circle, top right corner).

For this tutorial, a number of interactions with the project website are recorded. Firstly, the language of the website is changed from English to Swedish. Secondly, a login to the admin window is recorded by clicking on the admin link in the menu to the left. Once at the admin window, a username and password is entered. The recording ends once *Login* has been clicked.

A recording is started by clicking on the red circle on the top right corner on the Selenium IDE window. This activates a new recording (Figure 15).

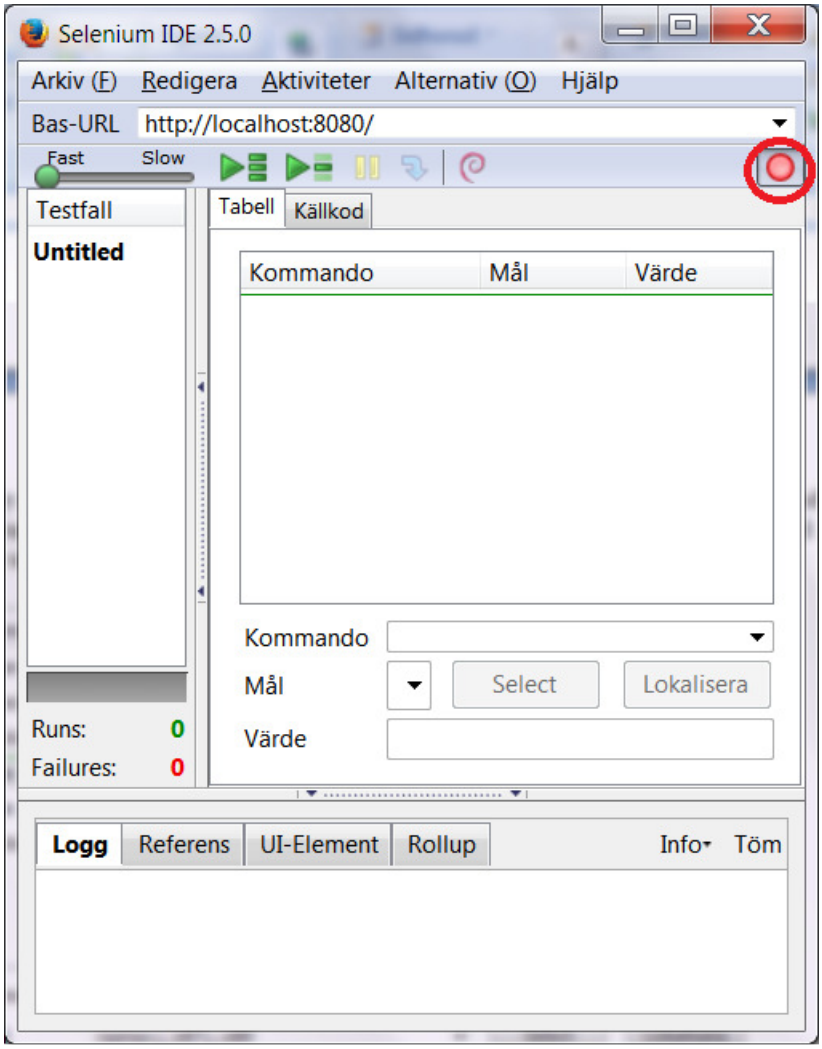

Figure 15 – Start recording

Anything that is clicked on the website from now on is recorded and entered to the script. After the interactions described above are made, the recording is stopped by clicking on the red circle.

Figure 16 shows the commands after recording the interactions.

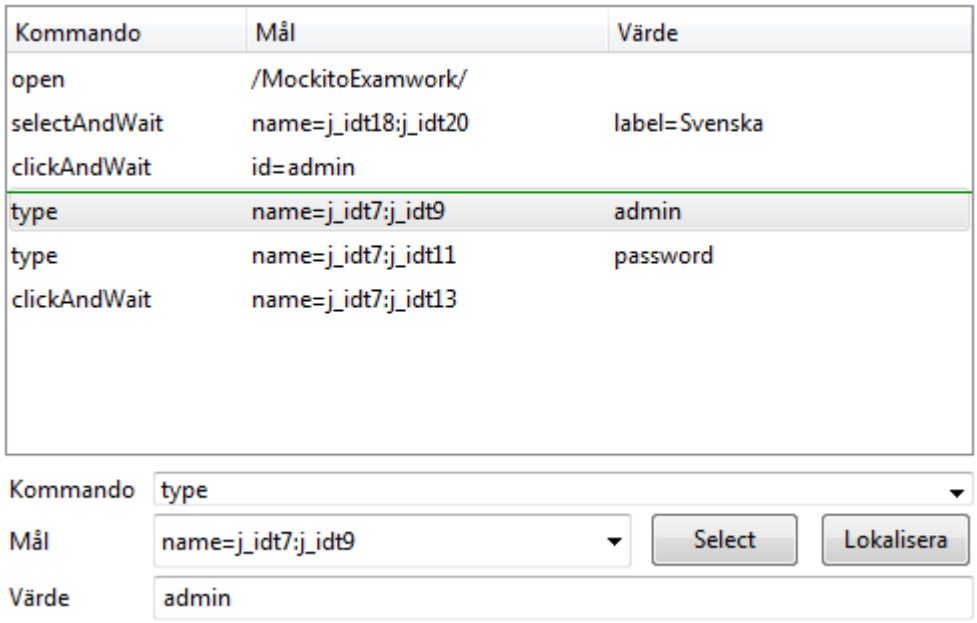

Figure 16 – The result from recording the interactions with the website.

The recording can be played back through the plug-in directly or exported as Java code to be executed in NetBeans IDE. To run the recording in the Selenium IDE plug-in jump to activity 1.5.2 Executing recording in Selenium IDE *(red colored path in figure 3).* If you wish to export the recording to Java code in order to execute in NetBeans instead, continue to activity 1.4.2 Implementing an exported recording *(green-colored path in figure 3).*

1.4 Implementing a test through Netbeans IDE

In this activity, it is explained how to create a Selenium test solely with Netbeans. It is also explained how to export a recording done in the Selenium IDE plug-in for Firefox in order to run the test in NetBeans IDE.

There is more than one way to execute a Selenium black-box test with NetBeans IDE. This example will focus on two methods. Firstly, by manually coding a test case and secondly, by exporting the code we recorded using the Selenium IDE Plug-in through Firefox (shown in activity 1.3).

1.4.1 Guidelines for a manually coded test

This activity shows how to manually code a test case with the Selenium framework, keeping in mind that any illustrations of code are just pseudo code. The reason for not sharing the full code base is because is it very specific to the website and will not be applicable to other Java EE web project interfaces. Instead, see these illustrations as guidelines on how to go about when creating a test. The code does the same procedures as during the recording. It is divided into parts in order to be explained in detail.

In the following examples, Selenium works by referring to an element in the HTML code through either the element name or ID. It then simulates an action on this element. In order to code a test, the element names or IDs must be known. To find an element name or ID, Firefox has an option called "*Inspect element*". This option will show the corresponding HTML code for an element on a given website. Once the HTML code is shown, the ID or name of the element can be extracted.

To use this option, it is as simple as right-clicking on a specific object on the website and left-clicking on *"Inspect element"* (Figure 17).

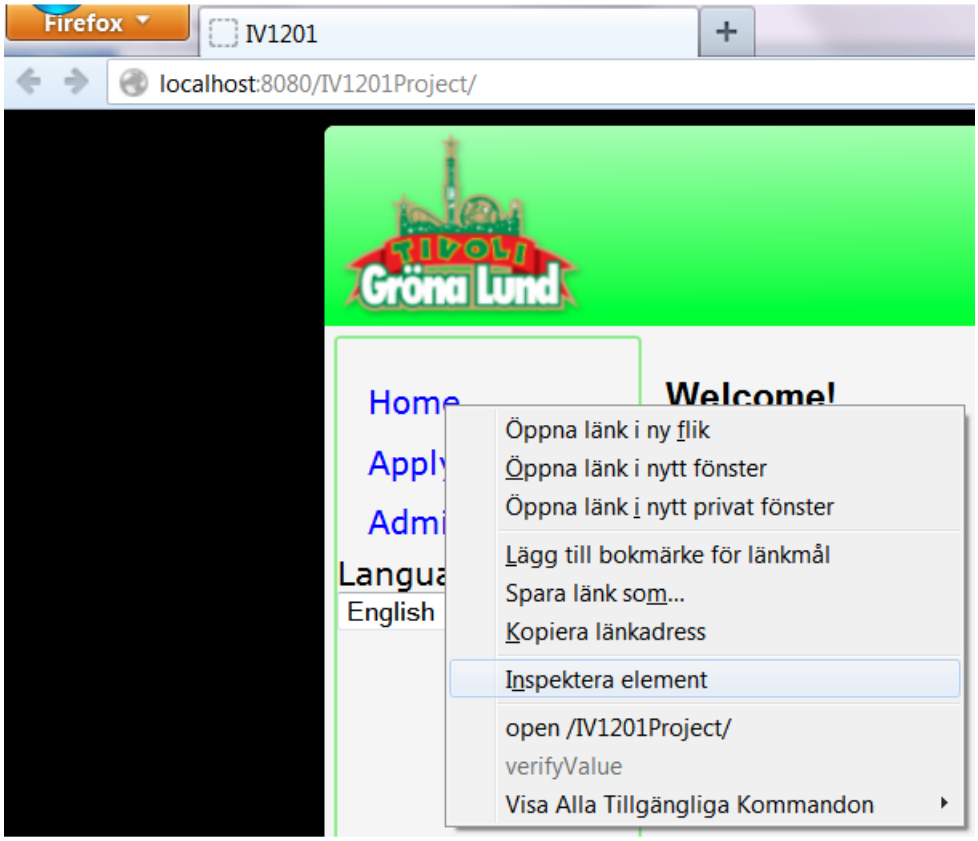

Figure 17 – right-clicking on *Home* to find its element name or ID by inspecting it.

Upon inspecting an element, a new window will appear at the bottom showing the corresponding HTML code for the chosen element. It is here where the element name or ID is found (Figure 18).

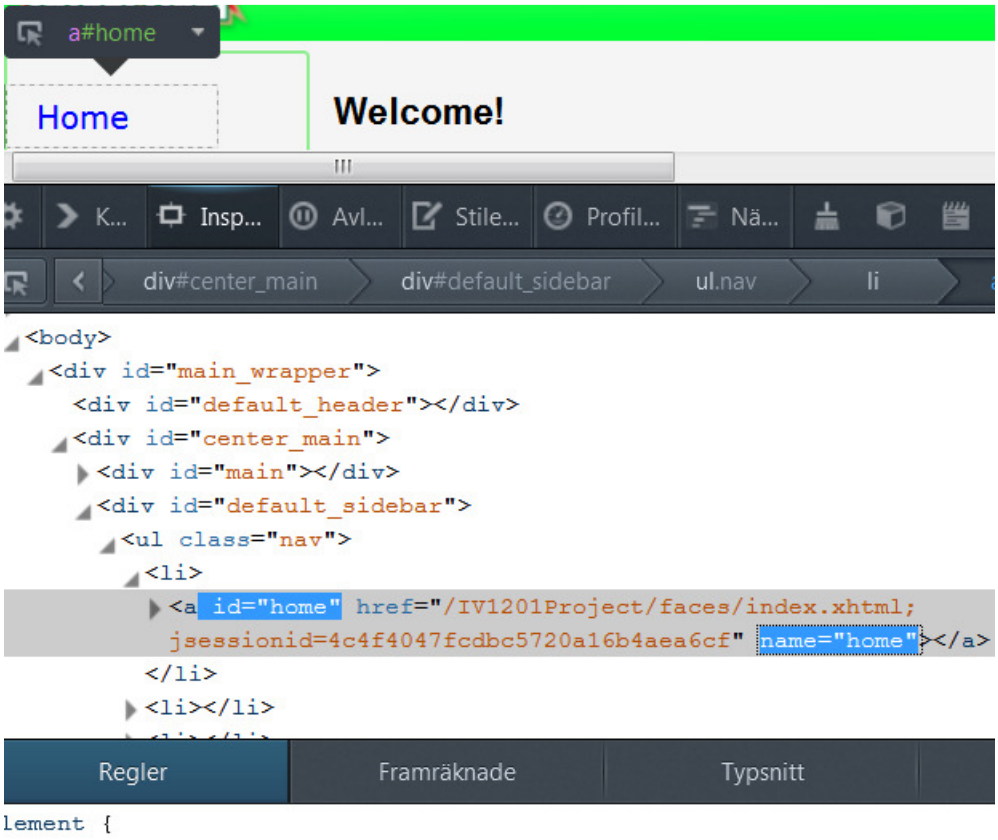

Figure 18 – Inspecting element *Home*. Its ID is *"home"* and its name is *"home"*.

Knowing how to find the names and IDs we can proceed with examples. The following pseudo code in Figure 19 is the initialization of the *HtmlUnitDriver*. This driver helps with automated testing of the website. The following method, *get*, loads the website that is submitted for testing, in this case, the web project website.

```
public class X {
    public static void main(String[] args) {
        // Create a new instance of the html unit driver.
        WebDriver driver = new HtmlUnitDriver();

        // Open up the website of the project.
        driver.get("https://localhost:8181/MockitoExamwork");
```

Figure 19 – Initialize the driver & load the website.

The pseudo code in Figure 20 is an example on how to interact with the website. In the first method, the focus is set to the menu bar with the different language options. The method will simulate a click on the menu bar and then select the option labelled "*Svenska*". This will change the language from English to Swedish.

```
// Find the element of the dropdown list and change the language to
// swedish, by choosing the text "Svenska".
new Select(driver.findElement(By.name("j_idt14:j_idt16")))
        .selectByVisibleText("Svenska");

// Find the element to the link and click on it.
WebElement adminLink = driver.findElement(By.id("admin"));
adminLink.click();

// Click on the username field and enter "admin" as username
WebElement usrname = driver.findElement(By.name("j_idt7:j_idt9"));
usrname.sendKeys("admin");

// Click on the password field and enter "password" as password
WebElement pword = driver.findElement(By.name("j_idt7:j_idt11"));
pword.sendKeys("password");

// Find the element for the submit button and click it.
WebElement submitButton = driver.findElement(By.name("j_idt7:j_idt13"));
submitButton.submit();
```

Figure 20 – Simulated interaction of the interface but in NetBeans.

The second method in Figure 20 will find the element in the HTML code for which ID is set to "*admin*". This ID is found in the JSF-page called *masterLayout.xhtml*. The JSF-page is found in the source code folder *Web pages* for the Web project. The method will simulate a click on said element once found. In other words, this method simulates the interaction of entering the admin window by clicking on *Admin* in the menu to the left of the website.

The third method shown in Figure 20 has the same procedure as the method above it but describes a different interaction. It will find an element, but in this case it will be found through the name of this element and not the ID. This is only to show that there is more than one way to refer to an element using the Selenium framework.

When the element has been found, a string of characters will be send to it, "*admin*". This represents the interaction of clicking on the text field in the admin window and entering a username. The interaction of entering the password is similar to this, which is the fourth method.

The fifth and last method in Figure 20 simulates the interaction of clicking on the *Login* button. It follows the same structure as the second method.

The pseudo code in Figure 21 is to confirm if the test succeeded, in other words, if the login was successful. There are various ways to check if the test passes. One way is to check if there is an element with ID "*logout*" that is visible at the website. The reasoning behind this is that, upon a successful login, a link called *Logout* will appear. If not, then this link will not be visible for the user, meaning that the login failed. Depending on if this element is visible or not, a message will be printed to the NetBeans IDE console.

```
// A confirmation.
WebElement x = driver.findElement(By.id("j_idt25:logout"));

// The "Logga ut" link visible means that it logged in to the adminpage.
if (x.isDisplayed())
    System.out.println("Login success!");

else
    System.out.println("No success!");
```

Figure 21 – One of many ways to confirm if the test has passed.

This test case was manually coded in the NetBeans IDE and not recorded through the plug-in. This test case works behind the scenes meaning, it is not possible to follow the execution of the test like it is with the Selenium IDE plug-in in Firefox. The whole test is executed without a Firefox window. It is now time to execute the test. Please jump to activity 1.5 Executing a test.

1.4.2 Implementing an exported recording

Now, it is time explain how to export the code from the Selenium IDE plug-in recording. When done recording a test, it must be exported as Java code. How to do this is explained below.

Click on *File*, and then hover on *Export Test Case As…* until a new window appears.

Click on *Java / JUnit4 / WebDriver*.

Save the file with the same name as the java class created in activity 1.2.2 (Figure 9). Figure 22 shows how to export a recording.

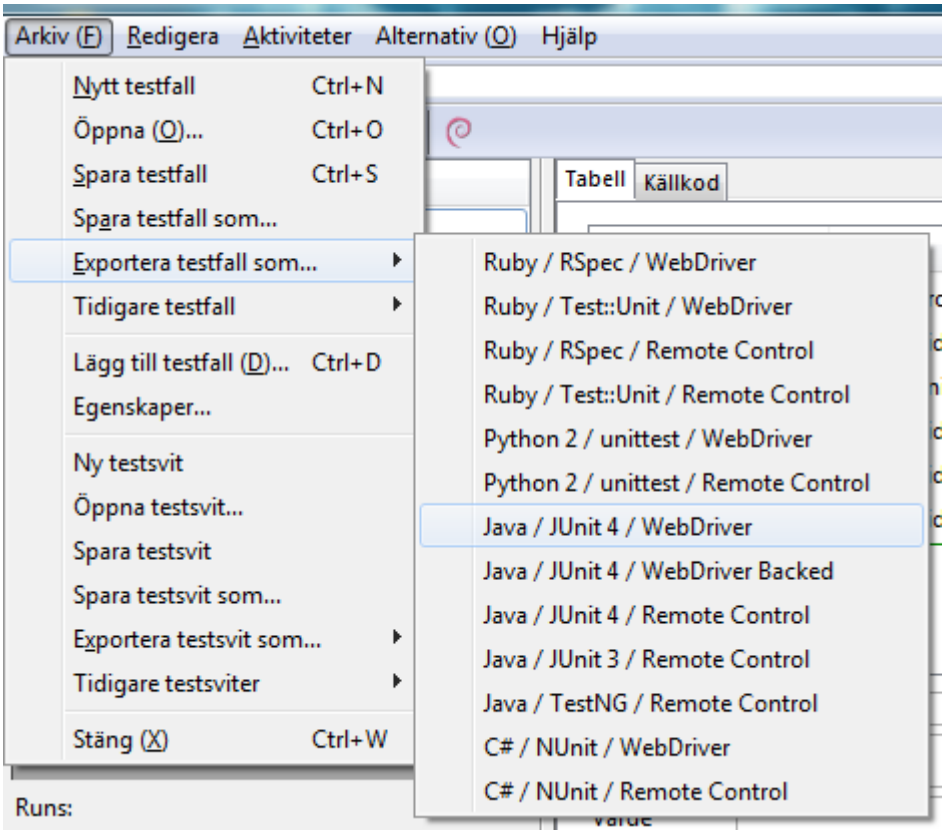

Figure 22 – Exporting the recorded interaction to java code.

This file contains the recorded interaction in java code. Next step is to open the test class that was created in activity 1.2.2 (Figure 9). You will now be presented with a new test class that contains around 150 lines of automatically generated test code. For this example, please delete this code.

This class will be located under a new folder called *Test Packages* in your new project called *"seleniumSecondMethod"*.

Copy the java code from the exported recording and paste it in the JUnit test class created in activity 1.2.2 (Figure 9). Remove package name declaration found in line 1 and replace it with the package name for where the java class resides.

There are some unused additional methods in this example and can be removed if desired. These are all the methods from *isAlertPresent* and after. Also, unused variables and imported libraries can be removed if desired. Let us now go over the code.

Once the test class is set to run, a couple of things need to happen before the actual test code is executed. Before the test can execute, the driver needs to be initiated and the website loaded. That is done by using the JUnit annotation *@Before* as shown in Figure 23.

```
@Before
public void setUp() throws Exception {
  driver = new FirefoxDriver();
  baseUrl = "http://localhost:8080/";
  driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
}
```

Figure 23 – Initialization of driver.

The Firefox driver is initialized, similar to the initialization in Figure 19. The difference here is that the Firefox driver will be initialized, which means that the test will be executed on a new Firefox browser instance. This also means that you will be able to follow the progress of the execution while the test is running.

The *@Test* annotation specifies that the following method is a test case.
The pseudo code in Figure 24 shows the object references to the different elements in the HTML web pages

```
@Test
public void testAdminLogin() throws Exception {
  driver.get(baseUrl + "/MockitoExamwork/");
  new Select(driver.findElement(By.name("j_idt14:j_idt16"))).
          selectByVisibleText("Svenska");
  driver.findElement(By.id("admin")).click();
  driver.findElement(By.name("j_idt7:j_idt9")).clear();
  driver.findElement(By.name("j_idt7:j_idt9")).sendKeys("admin");
  driver.findElement(By.name("j_idt7:j_idt11")).clear();
  driver.findElement(By.name("j_idt7:j_idt11")).sendKeys("password");
  driver.findElement(By.name("j_idt7:j_idt13")).click();
}
```

Figure 24 – The test case.

The *@After* annotation in Figure 25 specifies what happens after the test has been executed. This can be removed if desired. This method will close the Firefox browser once the test case has finished executing.

```
@After
public void tearDown() throws Exception {
  driver.quit();
  String verificationErrorString = verificationErrors.toString();
  if (!"".equals(verificationErrorString)) {
    fail(verificationErrorString);
  }
}
```

Figure 25 – After the test has been executed.

1.5 Executing a test

In this activity, it is explained how to execute a test through the Selenium IDE plug-in or NetBeans IDE.

1.5.1 Executing a manually coded test

The manually coded test case is executed just like any other java class that contains a main method. Right-click on the class found under *Source Packages* and simply click on the option *Run File*. This is shown in Figure 26.

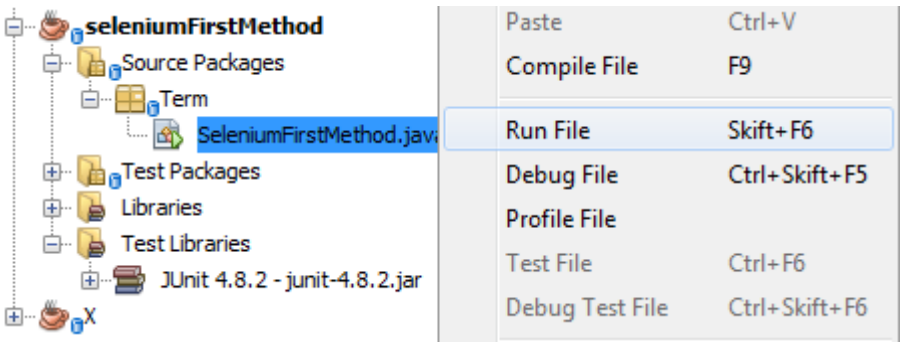

Figure 26 – Executing a test with a main method.

There is no method of confirmation if the test passes or fails. Because of this, a simple if-case was created to check if the logout element is visible or not. Depending on if it is visible or not, a message will be printed to the NetBeans IDE console. See Figure 21 in activity 1.4.1 for the code.

If the logout element is visible, the print will be "*Login success!*" as shown in Figure 27.

Figure 27 – The "confirmation" that the test passed.

1.5.2 Executing recording in Selenium IDE

Figure 28 shows the Selenium IDE action bar. You can control the recorded test using this bar by pressing play, pause, stop, choose the execution speed etc.

When done recording your interactions with the Selenium IDE plug-in, it is possible to choose how fast the recording should run. It is recommended to lower the speed to lowest since it may not be possible to see the interactions at higher speeds.

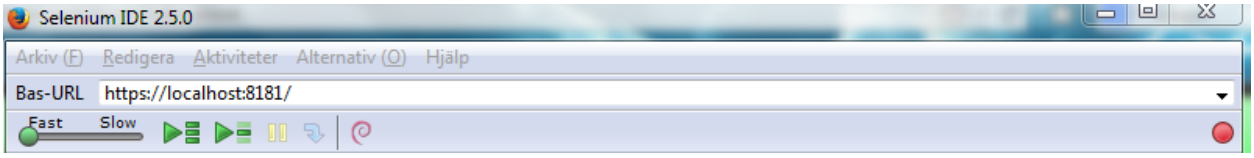

Figure 28 – Action bar used to control the test.

To play the recording from the beginning, press the play symbol. In this example, the website will open up, the language will be changed to Swedish and the script will click on the admin link. Finally, entering username, password and clicking on the login button. You will be able to follow the steps during the execution either by watching the Firefox browser or by following the output to the log window in the Selenium IDE plug-in (Figure 29).

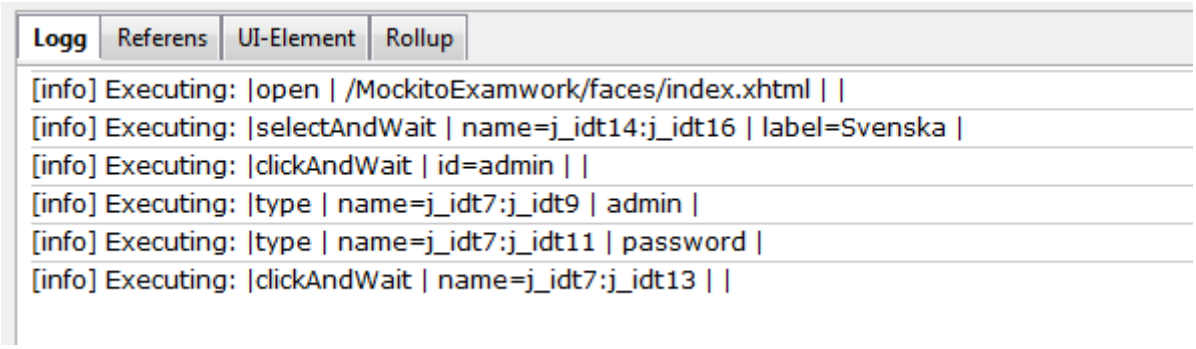

Figure 29 – Output when playing a recorded interaction in the plug-in window of Selenium IDE.

1.5.3 Executing exported recording

To execute the exported Selenium IDE test, right-click on the test class at the project tree and choose *Run File* or *Test File* (Figure 30). A less common error that might occur when running the test is that Selenium cannot find the binary path to Firefox. To solve this issue, make sure that the Mozilla Firefox folder is located at your PC's *Program Files*-folder.

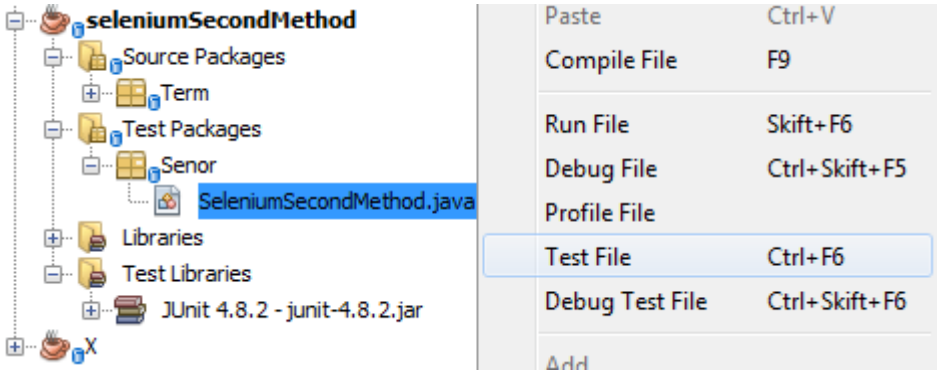

Figure 30 – Executing the exported recording from the Selenium IDE plug-in.

A new window will appear showing the test results at the lower left in NetBeans IDE (Figure 31).

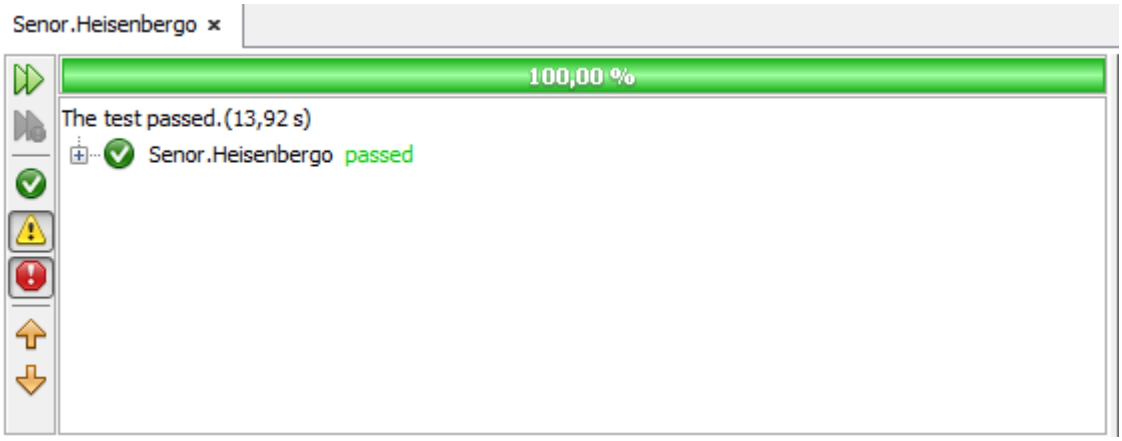

Figure 31 – Test result showing the test passed.

End of tutorial.