Practice

Appium Mobile Automation Testing

Group | 30332

Student | Fildan Claudiu

Institution | Technical University of Cluj-Napoca

Period | July 30 – September 3, 2020

Supervisor | As. Dr. Ing. Goța Dan

Claudiu Fildan
TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

Table of Contents

Contents

Chapter 1: Introduction in Appium

Introduction to Appium

Appium is an open source test automation for native mobile application and mobile browsers in Android and IOS. Because of the frequent changes in nowadays apps, it is necessary to use a tool for the increasing number and complexity of these apps. Appium has the advantage of being used for mobile browser testing as well. In this way with only one tool, it is possible to test the entire mobile platform. For this thing, it is using function inherited from Selenium, a test automation tool used only for web testing.

Appium Features

The main advantage of Appium is the fact that it is one of a few cross-platform test-supporting tool. Thereby it is compatible with both Android and IOS, without the necessity of rewriting code in two different tools.

Mobile applications are classified in three main category: native, hybrid and web. A native app is an exclusive app written for one of the two main mobile platform, Android or IOS. A native app run directly on the operating system. A mobile web app is a website that behaves like a native app, for example, some custom-made software programs use by some companies for their own operation. Hybrid apps are a combination of the previous two types. It involves both the native and the web part. For example, some task will redirect you to a website, for example in the case of registration. Another feature is the fact that it supports all of the web drivers APIs that are supported by the Selenium, making it very versatile. The user can write code in any programming language supported by the web driver. Some of them are Java, C#, JavaScript, Python and Ruby.

Appium Internal Architecture

Appium Client Code is the Appium code that can be written in multiple languages. The JavaScript Object Notation (JSON) is used for exchanging data between and an Integrated Development Environment like Eclipse and a server. Because the server only accepts the data as a text, the code need to be transform in a JSON format. The Appium Server have the ability to understand and perform the code on the mobile device. In addition, redirect the request to the corresponding platform based on the information provided in the code. In the case of Android, the server uses the IAutomator2 to connect with an Android device. In the case of the IOS, it uses XCUITest.

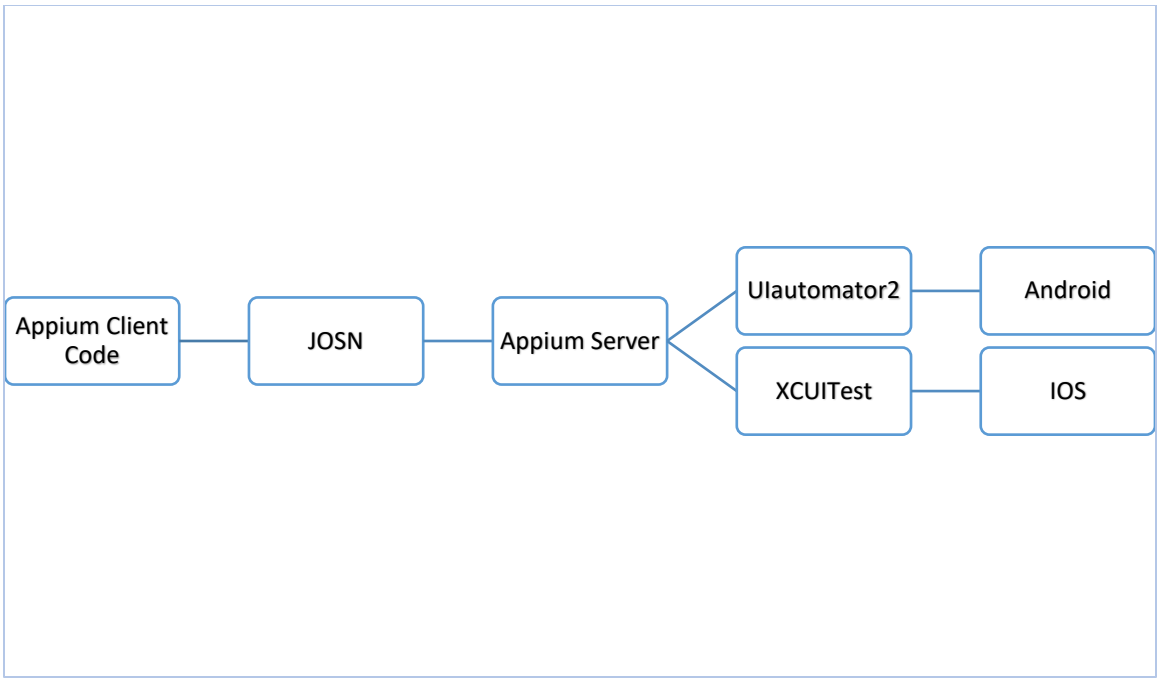

Appium Client Code
JOSN
Appium Server
Ulautomator2
XCUITest
Android
IOS

Figure 1.1

Chapter 2: Appium Installation on Windows for Android Automation

Installation of the Appium Infrastructure

In order to run the automation test cases, it is required to install all the necessary tools and programs with the proper configuration. This thing is done in the following steps.

Step 1. Download Java and set the Java_Home environmental variables.

Open the search engine on your browser and search for "jdk download". Enter on the "Java SE Downloads" result. Find the button "JDK Download" and download the "JAVA SE Development Kit" compatible with your device, in our case it is Windows. Run the downloaded ".exe" file and all the required files are automatically installed. After the download ends, open File Explorer and enter in the Program Files. Enter in the Java folder and find the so called "jdk" directory. Copy the path to the folder as in Figure 2.1.

Figure 2.1

Enter in the Control Panel and select System. Find the Advance system settings. In the pop-up window, select the Environment Variables. As in Figure 2.2, press the New or the Edit button, from the bottom of the window. Put as the name "JAVA_HOME" and past the path of the JDK folder, like in Figure 2.4.

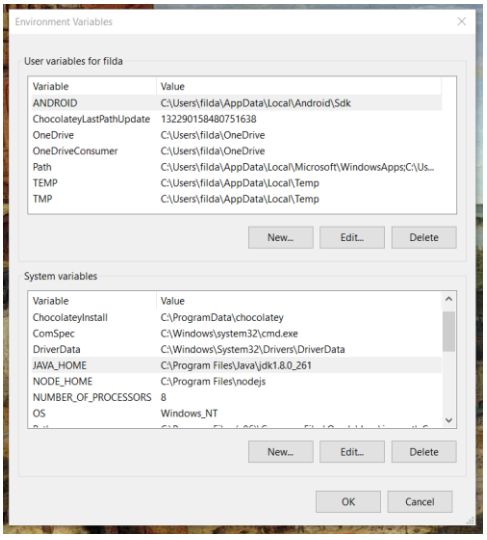


Figure 2.2

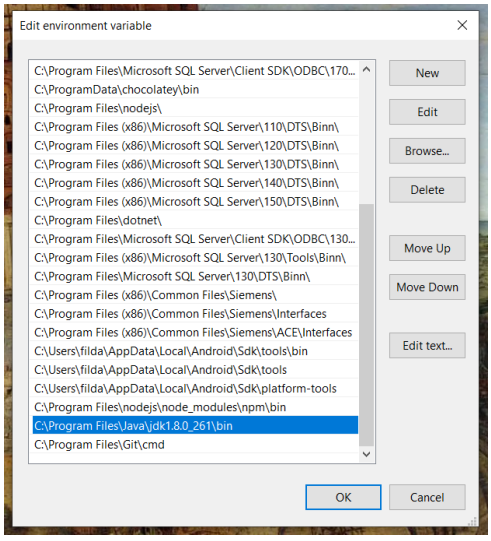


Figure 2.3

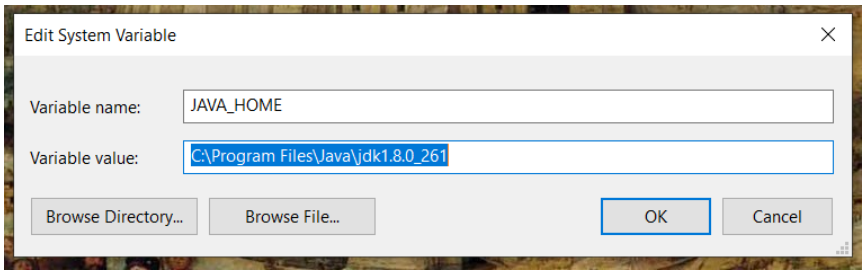


Figure 2.4

In the window open in Figure 2.3 check for the PATH variable and press on the Edit button. Paste the JDK folder address and add ”/bin” to the end of the address, as in Figure 2.3. For older Windows version, put a semicolon after the last variable value and paste the address with the “/bin” termination included also.

Step 2. Download Android Studio

For the installation of Android Studio open the browser and search for “Android Studio download” or input the following address “https://developer.android.com/studio”. Press the download button and wait until it is finish. After that, open the “.exe” file and press Run to install the program.

Step 3. Find the Android installation path

Open File Explorer, find the Users folder and select your username used in Windows. In the File Explorer window, in the upper part, find the View Section and the Show/Hide option. Select the Hidden items checkbox. Afterwards select the AppData folder and look after the Local folder. In the Android folder, find the directory called “Sdk”. Copy the path to the folder, as in Figure 2.5.

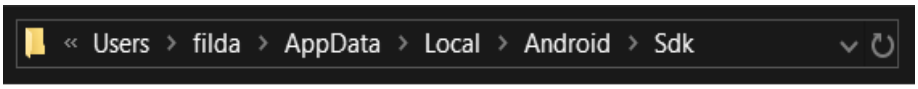


Figure 2.5

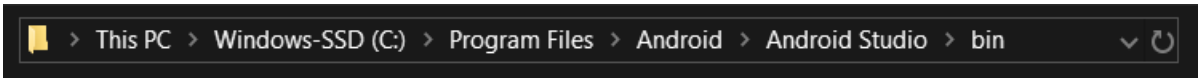


Figure 2.6

Step 4. Set Android Home environmental variables path to the SDK location.

Open the Environment Variable window, as in Figure 2.2, and add a new System Variable called “ANDROID_HOME” and paste the path to the Android SDK folder, like in Figure 2.6.

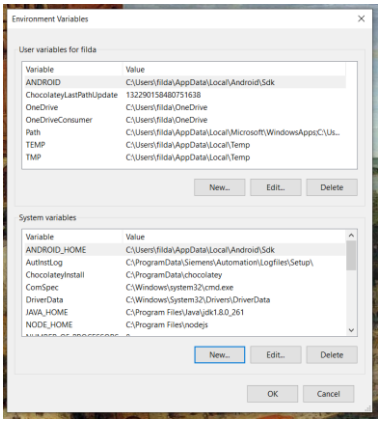


Figure 2.7

Enter in the path provide in Figure 2.7 and find the “studio64” folder and double click on it. It opens the Android Studio in order to download the necessary tools. When the program is open for the first time is will appear the option to select a Project Template. Choose the Basic Activity or Empty Activity option. In the Create New Project window, you can choose any option or name. Wait until the loading is done and the message “Gradle: Build…” from the bottom of the page disappears. Select the Tool settings and click on SDK Manager, as in Figure 2.8. In the Android SDK menu select the SDK Tools column and uncheck the Hide Obsolete Packages option. Check Android SDK Tools (Obsolete), like in Figure 2.9.

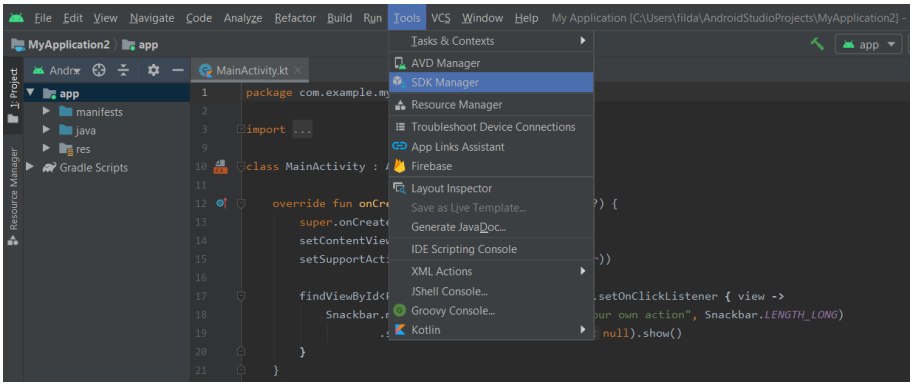


Figure 2.8

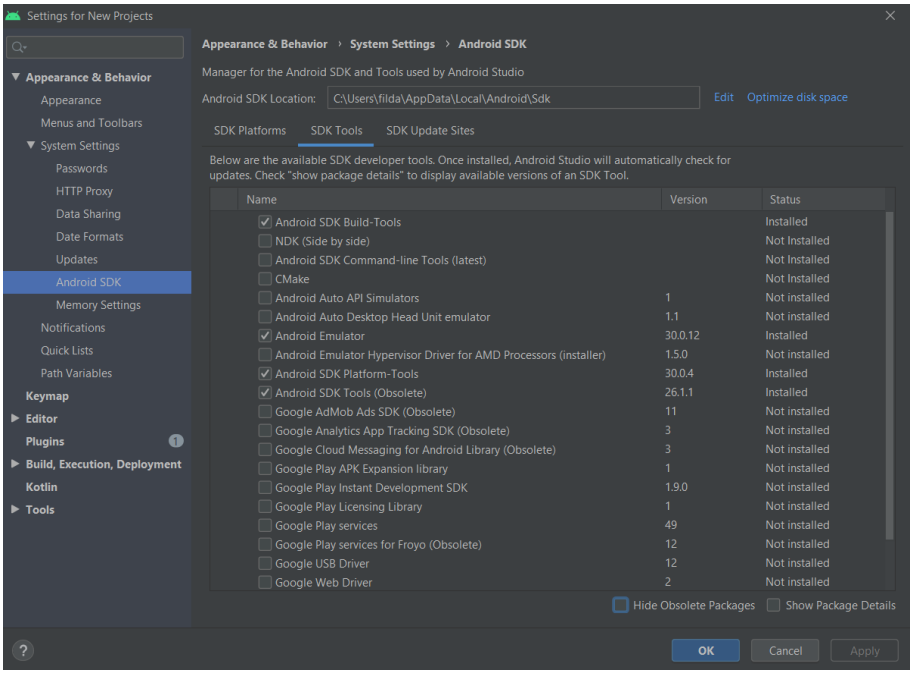


Figure 2.9

Press the OK button and afterwards confirm the changes. After the download is done, go the path in the Figure 2.10.

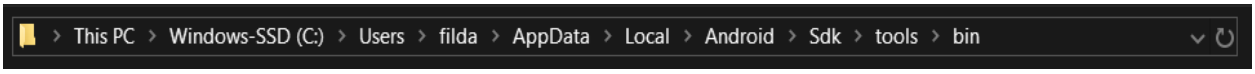


Figure 2.10

Go the Environment Variables and, inside of the System Variables section, search for the Path variables as before. Click on Edit button and then paste the path, as in Figure 2.11. In the File Explorer go back the tools folder and copy the path. Then paste the path in the Edit Environment Variable as before, like in Figure 2.12. Do the same with the “platform-tools” folder from the “Sdk” folder and copy the path in the same location as the previous two.

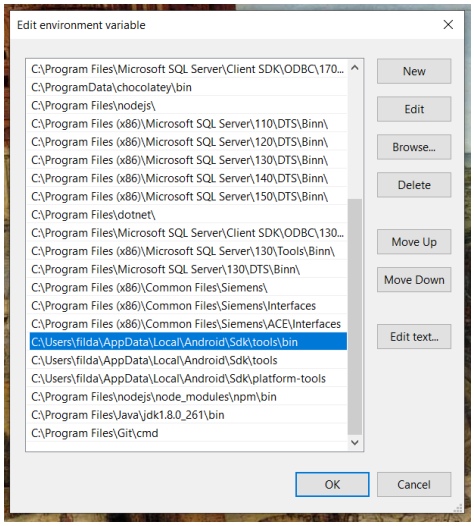


Figure 2.11

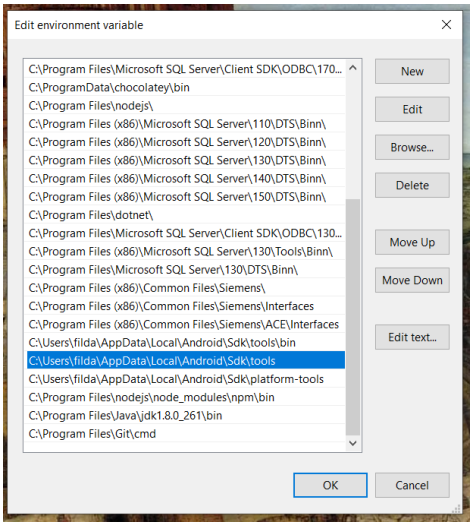


Figure 2.12

Step 5. Open Android Studio and configure emulator (Virtual Device)

Open Android Studio as previously and wait until none of processes are still running in the bottom left of the page. Look for the Tools option from the upper part and click on the AVD Manager. AVD stands for Android Virtual Device. Click on Create Virtual Device and select any option. Click the Next button and then select a system image. This system image is the operating system, on which the virtual smartphone is running. If the desired operating system cannot be selected, click on the Download button and then click the Next button. Wait until the download is finished. Afterwards click of the Next button. In the Verify Configuration section, put the AVD Name. The AVD Name will be use in the Appium code to connect to the virtual device. Click on the Play button (The green triangle), like in Figure 2.13, and the virtual device opens as in Figure 2.14.

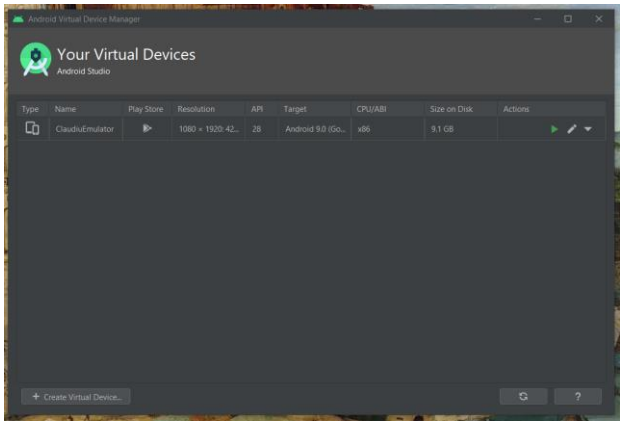


Figure 2.13

Figure 2.14

Step 6. Open emulator from Command Prompt

It is not require to have all of the Android Studio opened when the test are perform. It is possible to open only the emulator. In order to do this, close the virtual device and open the Command Prompt. In File Explorer go to the Android SDK folder and select the emulator folder. Copy the path as in Figure 2.15

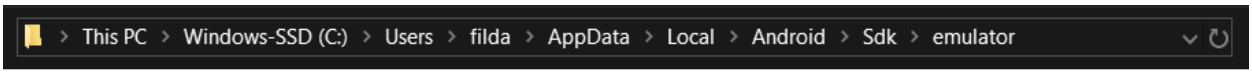


Figure 2.15

In Command Prompt write the command “cd” and paste the path from the Android SDK folder. After that, write “emulator -avd”, where “-avd” stands for Android Virtual Device. At the end put the AVD Name provided in previous step, as in Figure 2.16.

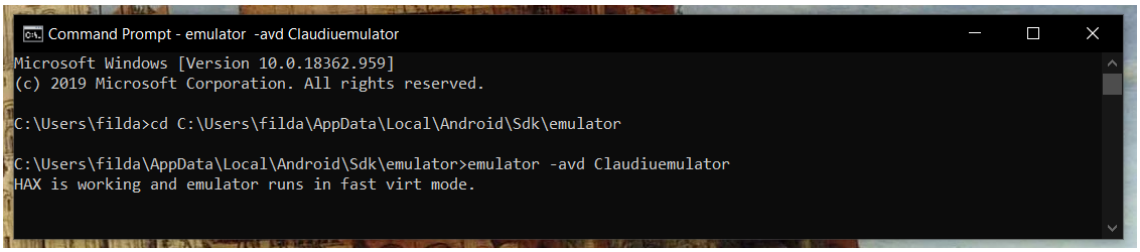


Figure 2.16

Step 7. Download Node.js

Open the web browser and search “download node.js” or open the following link “https://nodejs.org/en/download/”. Choose the compatible installer and download it. Afterwards run it.

Step 8. Set Node_Home environmental variables path

Open File Explore and in the Program Files look after the “node.js” folder. Click on it and copy the path to the folder. In the Environment Variables window, press the New button. Write the name of the variable as “NODE_HOME” and paste the path to the Node folder, as in Figure 2.17.

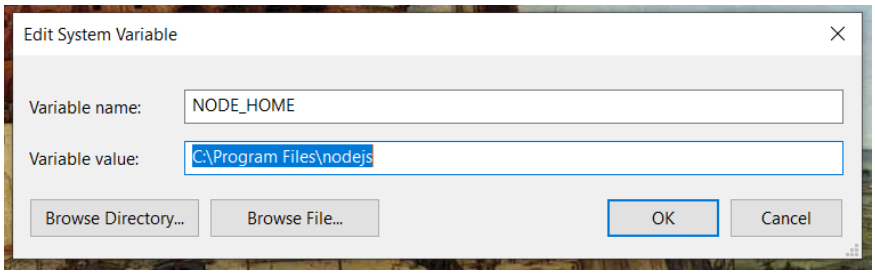


Figure 2.17

Step 9. Set the npm environmental variables path

In File Explorer, select the “node_modules” folder and then the “npm” folder. Enter in the “bin” folder and copy the path, as in Figure 2.18. Afterwards, go to the Environment Variables window and search for the Path variable. Add the last path as previously.

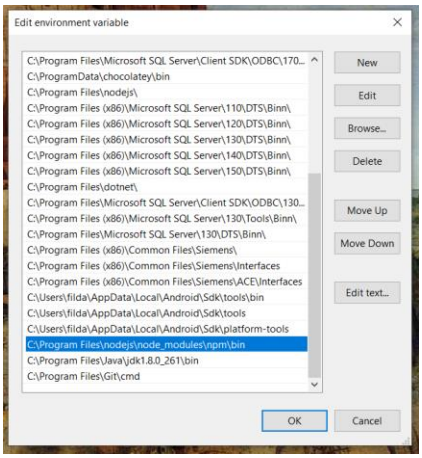


Figure 2.18

Step 10. Download Appium Server from Node

Node.js is a program in which the “npm” is a command line installer. It can install any Node module, in our case for example Appium module. Appium comes in two different component. One is the server side, which is downloaded by the “npm” command. The second one is the client side that comes in the jar format. Java Archive is a package file format typically used to aggregate many Java class files and resources.

Open Command Prompt and write the following command “npm install –g appium”, as in Figure 2.19. The “npm install” will activate the installer and the “-g” stands for installing the module globally in the system and the “appium” is the name of the module. Wait until all of the components are downloaded.

In order to start the Appium Server write in Command Prompt “appium” and the server will start automatically. To stop the server press Control + C. Then input “y”, as in figure 2.19.

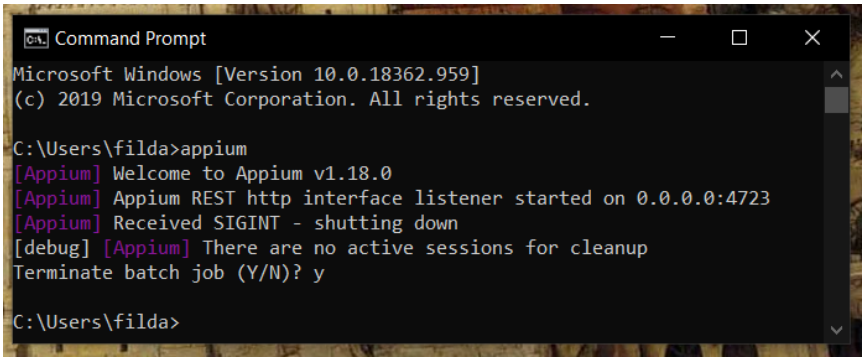


Figure 2.19

Step 11. Download Appium Java client library

Open the web browser and search for “download appium” or access the following link “http://appium.io/downloads.html”. In the “Appium language binding” section select the Java one. Click on download and select the jar option.

Step 12. Download Eclipse

Download Eclipse and make all the settings properly regarding the java virtual machine. Search on internet for additional information regarding the installation of Eclipse.

Step 13. Open Eclipse

Open Eclipse and make a new project. It can have any name. Make a package and a class called “based”. In order to make the project aware of Appium, click on the project and select Properties. Select Java Built Path menu and choose the Libraries column. Find the Add External JARs button and select the Java jar form the previous step.

Step 14. Download Selenium

In the web browser search for “download selenium” or enter on the following link “https://www.selenium.dev/downloads/”. Find the Java section and download the files. In Eclipse, click on the project and open the Properties like before. Go to the Libraries and click on Add External JARs. Find the downloaded selenium folder and open it. Import the first two jars. Open the “libs” directory and import all of the jars inside it. Press Apply and afterwards click on Apply and Close, as in Figure 2.20.

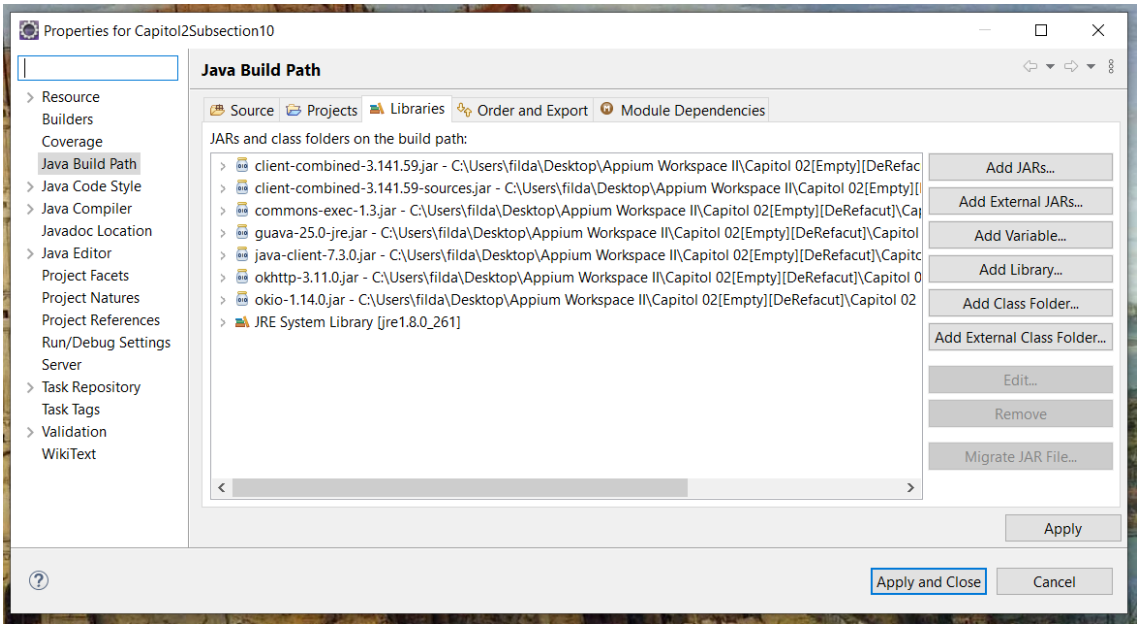

Properties for Capitol2Subsection10
Java Build Path
Resource
Builders
Coverage
Java Build Path
Java Code Style
Java Compiler
Javadoc Location
Java Editor
Project Facets
Project Natures
Project References
Run/Debug Settings
Server
Task Repository
Task Tags
Validation
WikiText
Source
Projects
Libraries
Order and Export
Module Dependencies
JARs and class folders on the build path:
client-combined-3.141.59.jar - C:\Users\filda\Desktop\Appium Workspace II\Capitol 02[Empty][DeRefac
client-combined-3.141.59-sources.jar - C:\Users\filda\Desktop\Appium Workspace II\Capitol 02[Empty][
commons-exec-1.3.jar - C:\Users\filda\Desktop\Appium Workspace II\Capitol 02[Empty][DeRefacut]\Cap
guava-25.0-jre.jar - C:\Users\filda\Desktop\Appium Workspace II\Capitol 02[Empty][DeRefacut]\Capitol
java-client-7.3.0.jar - C:\Users\filda\Desktop\Appium Workspace II\Capitol 02[Empty][DeRefacut]\Capitc
okhttp-3.11.0.jar - C:\Users\filda\Desktop\Appium Workspace II\Capitol 02[Empty][DeRefacut]\Capitol 0
okio-1.14.0.jar - C:\Users\filda\Desktop\Appium Workspace II\Capitol 02[Empty][DeRefacut]\Capitol 02
JRE System Library [jre1.8.0_261]
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...
Apply
Apply and Close
Cancel

Figure 2.20

Chapter 3: First Appium Program

Setting Up the Appium Workspace

In order to explain the following concept, it is necessary to use a demonstration app. It has the name “ApiDemo-debug.apk”. Download it from the course materials.

To write a code for test automation, it is require doing setting up the necessary infrastructure. Let‘s propose to connect the code with the virtual device set in the installation step. In addition, we need a demonstration app to test also. The Appium server makes the connection between these two components.

Open Command Prompt and write “appium”, as in Figure 2.19, to start the Appium server. To start the emulator, open another Command Prompt and use the “cd” command to go to the emulator location and write “emulator –avd emulatorName”, where “emulatorName” is the name of the AVD Emulator set in the Android Studio. This step is described in Figure 2.16.

Open Eclipse and create a new project. To this project, it is require adding the Appium Client Java file and the Selenium files as shown previously in Figure 2.20. Copy the demonstration app APK file, downloaded at the previous step, and paste in the project in Eclipse in the package. To start programming, create a class called “base” in the package that contains the ApiDemos file. In the main method, the test code is written.

Select the Android Virtual Device

In Selenium, there is a class called DesiredCapabilities, which help us make the necessary setting connection between the Appium code and the app. This class has a method called setCapabilities in which the user can set the connection parameters in the next format.

setCapability(capabilityName, value)

The first parameter is the name of a certain property we want to set and the second is the value we want to set to it.

In the main method of the class, create an instance of the class DesiredCapabilities named “cap”. Afterwards using the setCapabilities method, set the parameter as “MobileCapabilityType.DEVICE_NAME”. This parameter it is useful to set the name of the Android emulator. Set the value with the name of the Android emulator set in the installation step. In this way, the code will know which virtual device to access. If Eclipse show an error, click on the error icon situated next to the line number and select import the DesiredCapabilities class from Selenium. For the MobileCapabiliety method, follow the same procedure in order to import the method from Appium library.

Select the Android Application

In order to connect with the wanted app, access the setCapabilities method again and write “MobileCapabilityType.APP” as a parameter. The termination “APP” is referring to the application we want to test. Create an instance called “f” of the File class and set it with the text “src”. After that, create another instance called “fs” of the File class and set it with the name of the APK file. In the value area, access the method getAbsolutePath of the “fs” object. It is possible to insert the value as the path to the APK file as a string, but this thing is avoided because it would hardcode to much the code.

Afterwards, create an instance named “driver” of the class AndroidDriver and type AndroidElement. In the constructor, introduce the address of the Appium server from the Command Prompt and the variable “cap” as in Figure 3.1. In addition, if an error appears please import the URL class from “java.net” library and click on the throws exception suggested by Eclipse. Import the AndoidDriver and AndroidElement as well.

Select the Android UI Automator

Android updated its internal structure to “UIautomator2” and now it is require mentioning the wanted version. To overcome this problem, use the setCapabilities method to set the “**MobileCapabilityType.AUTOMATION_NAME”** parameter with the value “**uiautomator2".**

```
package package22;

import java.io.File;
import java.net.MalformedURLException;
import java.net.URL;
import org.openqa.selenium.remote.DesiredCapabilities;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;
import io.appium.java_client.remote.MobileCapabilityType;

public class base {
    public static void main(String[] args) throws MalformedURLException {

        // selenium class use to connect the code
        // with the server and the app
        DesiredCapabilities cap = new DesiredCapabilities();
        File f = new File("src");
        File fs = new File(f,"ApiDemos-debug.apk");

        // method used to set the parameters and their values
        // cap.setCapability(capabilityName, Value);

        // set the emulator name which we want to access
        cap.setCapability(MobileCapabilityType.DEVICE_NAME, "ClaudiuEmulator");

        // Update
        cap.setCapability(MobileCapabilityType.AUTOMATION_NAME,"uiautomator2");

        // set the app name which we want to test
        cap.setCapability(MobileCapabilityType.APP, fs.getAbsolutePath());

        AndroidDriver<AndroidElement> driver= new AndroidDriver<>(new URL("http://127.0.0.1:4723/wd/hub"),cap);
    }
}
```

Figure 3.1

Note: Please wait 60 seconds between running the test code because the Appium server will otherwise generate an error.

UI Automator Tool

To test an application it is required to identify different graphic element of the Android interface. To do this thing, the developers created a tool named “UIAutomator”. With it, the developers are able to identify and to see the properties of different design element present on the emulator screen.

Start the Android emulator as before in Figure 2.16. In File Explorer, go the Android SDK folder and select “tool”, then “bin”. Here is a file name “uiautomatorviewer”, like in Figure 3.2. Click on it and the Command Prompt will appear. Wait some couples of second until a window appears, as in Figure 3.4. Press on the second button on the upper part of the window. The automator will scan the screen of the emulator and will display the screenshot taken together with properties of the screen elements, as in Figure 3.5.

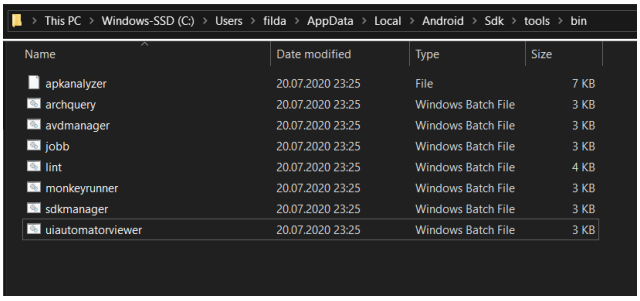


Figure 3.2

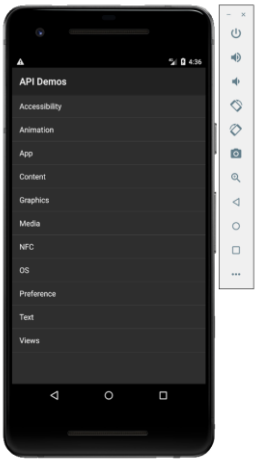


Figure 3.3

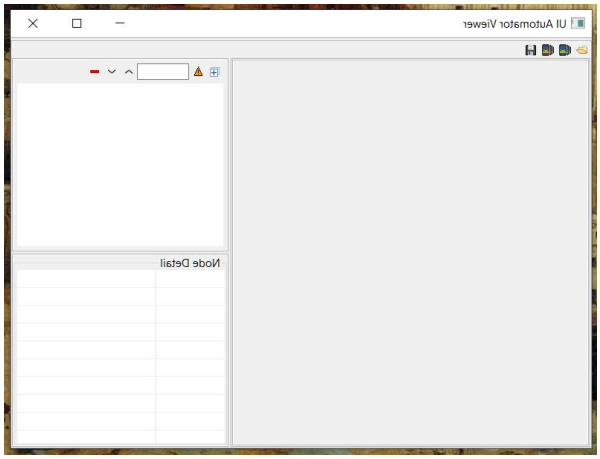


Figure 3.4

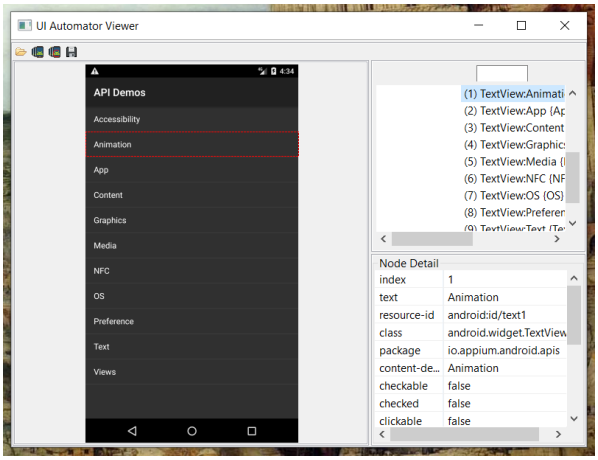


Figure 3.5

Because it is require using the already written code for every next test, it is a good practice to put it in a separate class. Modify the “public static void main” method in a “public static AndroidDriver<AndroidElement> Capabilities throws URLException” method. This modifies the method to transmit the object of type “AndroidDriver”. At the end of the method write “return driver”, which is an instance of the class “AndroidDriver”, as in Figure 3.6.

After that, create a new class called ”basics” which inherits the methods of the base class. In the main methods initiate an object “driver” of class “AndroidDriver” and transmit to it the output of the Capabilities method. See the Figure 3.7.

```
1  package package26;
2
3  import java.io.File;
4  import java.net.MalformedURLException;
5  import java.net.URL;
6  import org.openqa.selenium.remote.DesiredCapabilities;
7  import io.appium.java_client.android.AndroidDriver;
8  import io.appium.java_client.android.AndroidElement;
9  import io.appium.java_client.remote.MobileCapabilityType;
10
11 public class base {
12         public static  AndroidDriver<AndroidElement> Capabilities() throws MalformedURLException{
13
14             AndroidDriver<AndroidElement>  driver;
15
16             File f = new File("src");
17             File fs= new File(f, "ApiDemos-debug.apk");
18
19             DesiredCapabilities capabilities = new DesiredCapabilities();
20
21             capabilities.setCapability(MobileCapabilityType.DEVICE_NAME, "ClaudiuEmulator2");
22             capabilities.setCapability(MobileCapabilityType.APP, fs.getAbsolutePath());
23
24             driver = new AndroidDriver<>(new URL("http://127.0.0.1:4723/wd/hub"), capabilities);
25             return driver;
26
27             }
28 }
29
```

Figure 3.6

```
1  package package26;
2
3  import java.net.MalformedURLException;
4
5  import io.appium.java_client.android.AndroidDriver;
6  import io.appium.java_client.android.AndroidElement;
7
8  public class basics extends base{
9
10     public static void main(String[] args) throws MalformedURLException {
11         AndroidDriver<AndroidElement> driver=Capabilities();
12     }
13 }
```

Figure 3.7

Chapter 4: Appium Methods for Native Application

Find by Xpaths and Text Attributes

Let us start our first test automation. Start the Appium server, the Android emulator and the UIAutomator Viewer, if there are not yet open. In the emulator enter in the demonstration app and take a screenshot of the emulator screen with UIAutomator Viewer. The further code will use the previous line of code provided in the previous section. One thing need to be taken into consideration, is the maximum interval of time the system waits if it does not find the specified element. As a convention, it will be set at 10 second with the help of the next code line from the Figure 4.1.

```
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
```

Figure 4.1

In the UIAutomator Viewer select on the "Preference" option and look in the right down part of the program windows. The name of the element and their value can be seen in the Node Detail. In the code, the class "android.widget.TextView" need to select in order to enter in the next menu, the Preference menu. If you select any element from the list, the parameters "resource-ID", "class" and "package" has the same values. This mean that if the code is accessing the class "android.widget.TextView" it will access the first element, as in a vector in which without an index specified the first element is access. The "index" and "text" are the parameters that differentiate each element of the list. The Android driver class has a number of methods for finding a certain element. One of these is the "findElementByXPath" method, in which a path will be introduce.

The Xpath is made of following parameters. The first one is the tagname, which is the name of the class in our case. The second one is the attribute, the text parameter in our case. The last is the value, which is the name of the element from the list we want to access, as in Figure 4.3. In our case, the value will be "Preference", as in Figure 4.2. Once the element is identified, the method "click" simulates the stimulus provided by the real user and enters in the next menu. Write the next line of code as in Figure 4.4. In the end, the code in the "basics" class should look like in Figure 4.5.

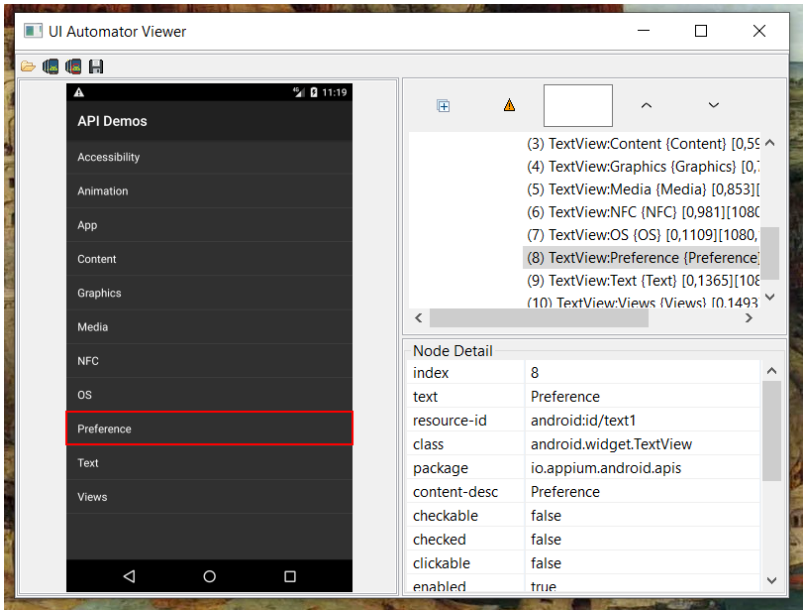


Figure 4.2

```
//xpath id className, androidUIautomator
/*
 * xpath Syntax
 * //tagName[@attribute='value']
 */
```

Figure 4.3

```
driver.findElementByXPath("//android.widget.TextView[@text='Preference']").click();
```

Figure 4.4

```java
package package27;

import java.net.MalformedURLException;
import java.util.concurrent.TimeUnit;

import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;

public class basics extends base{

    public static void main(String[] args) throws MalformedURLException {

        AndroidDriver<AndroidElement> driver=Capabilities();

        // 27.
        // Setting the amount of time before the "No Such Element Exception"
        // is throwed

        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

        //xpath id className, androidUIautomator
        /*
         * xpath Syntax
         * //tagName[@attribute='value']
         */

        // select the "Preference" option from the list

        driver.findElementByXPath("//android.widget.TextView[@text='Preference']").click();

    }
}

```

Figure 4.5

Note: The UI Automator does not work when the code is tested.

Find by Id and Class Name

In the emulator, enter the Preference menu and take a new screenshot of the emulator screen with UIAutomator, as in Figure 4.7. Let us propose to select the Preference dependencies. As it can be observed, the “resource-id”, “package” and “class” are the same with the previous menu. To select the element we will use the same line of code as before but with “3. Preference dependencies” as value, like in Figure 4.6.

```java
driver.findElementByXPath("//android.widget.TextView[@text='3. Preference dependencies']").click();
```

Figure 4.6

In the Preference dependencies there, are a checkbox and an unavailable menu. As it can be seen in Figure 4.8, the “text” parameter does not have any value attributed to it. Therefore, the identification of the element after the “text” parameter is not possible anymore. Nevertheless, the method findElementById can identify an element after the “resource-id” which is present in our case. In order to select the checkbox, write the following code from Figure 4.9.

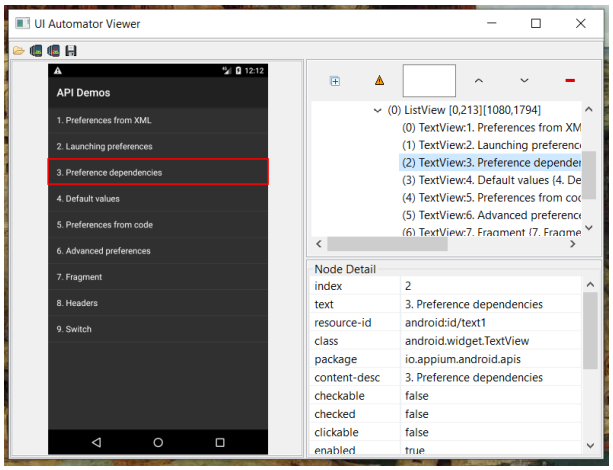


Figure 4.7

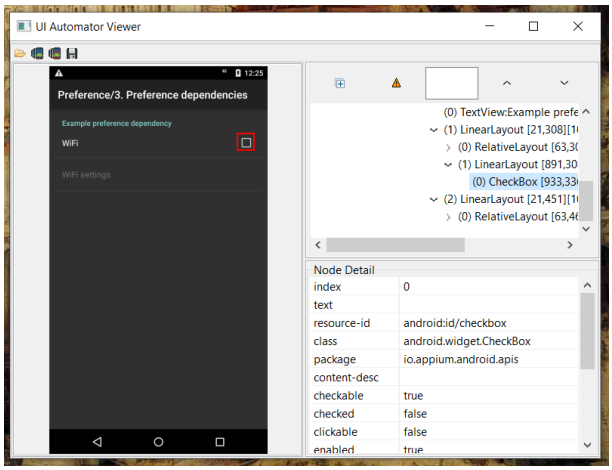


Figure 4.8

```
driver.findElementById("android:id/checkbox").click();
```

Figure 4.9

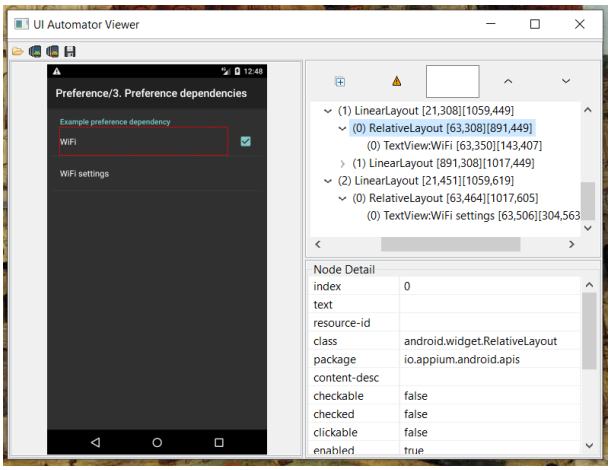


Figure 4.10

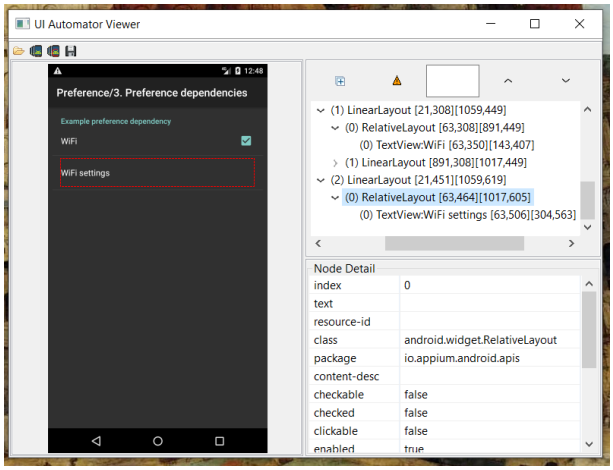


Figure 4.11

```
driver.findElementByXPath("(//android.widget.RelativeLayout)[2]").click();
```

Figure 4.12

Once the checkbox for Wi-Fi is selected, the Wi-Fi settings will be available. In Figure 4.10 and Figure 4.11, both the Wi-Fi and the Wi-Fi settings have the same parameter. In the right upper part the difference can be observe when clicking on the two Relative Layout. For each of the element there are different Relative Layouts. Using the “findElementByXPath” method the

Relative Layout parameter can be access. The server will automatically access the first element, but we want to access the only the one with the id equal with two. This one is corresponding with Wi-Fi settings. So the index of the Relative Layout is put as well, please see the Figure 4.12.

After the Wi-Fi element is access a pop-up windows appears. In this window, there is a textbox where the user can introduce a password, a cancel button and an OK button. Let us introduce the text “hello” and then press the OK button.

As in Figure 4.15, the textbox is identified in the right upper part of the viewer as “EditText”. As there is only one textbox, so no confusion can be made. Using the findElementByClassName method, the element can by access from the name provided in the right upper window of the viewer. In order to input a text the sendKeys method is used with the wanted text as a parameter just like in Figure 4.13.

```
driver.findElementByClassName("android.widget.EditText").sendKeys("hello");
```

Figure 4.13

```
driver.findElementsByClassName("android.widget.Button").get(1).click();
```

Figure 4.14

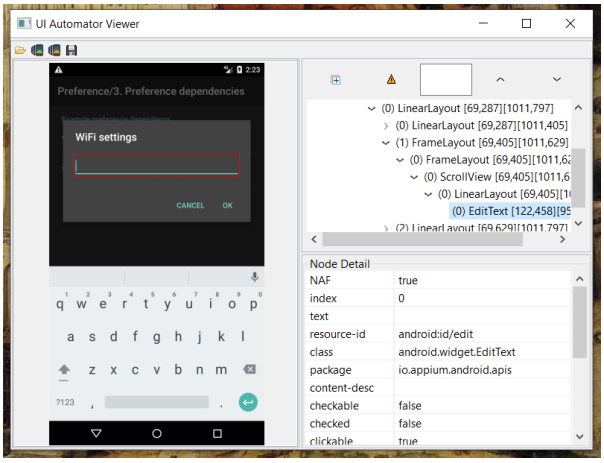


Figure 4.15

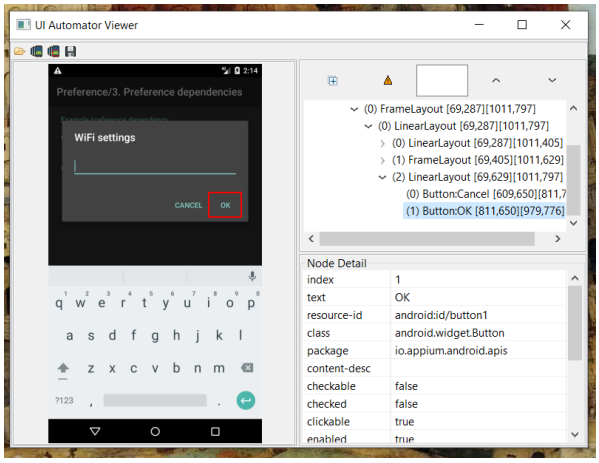


Figure 4.16

Handing Objects Indexes

To press the OK button, in Figure 4.16, there are two buttons. One is for OK and one is for Cancel. It is possible to use the previous methods of identity the proper element by text or id but for variety, other facilities can be used. The findElementByClassName is different from the

previous method by the fact that it searches for all of the objects with the same value of parameter not only for the first found. To identify the wanted object from the results the method “get” is used with the index as a parameter. The first result will have the index 0 so in order to access the second result the index search need to be equal with one. Once identified the element it remain only to click on it with the method “click”. Follow the code in the Figure 4.14.

Find by UI Automator

The developers introduced some special methods. One of this is the findElementByAndroidUIautomator. This method has the following syntax: “attribute (“value”)”. Because Java does not allow quotation mark inside of another quotation mark, it is necessary to put the backslash character.

If the developer want for example to enter in the View menu, he will need to put the following syntax “Text (\”Views\”)”. Moreover, it is possible for example to display how many element has the property “clickable” with a certain Boolean values. All of these can be seen in the following code in Figure 4.17.

```
1  package package30;
2
3  import java.net.MalformedURLException;
4  import java.util.concurrent.TimeUnit;
5
6  import io.appium.java_client.android.AndroidDriver;
7  import io.appium.java_client.android.AndroidElement;
8
9  public class Uiautomaortest extends base {
10     //30.
11     public static void main(String[] args) throws MalformedURLException {
12
13         AndroidDriver<AndroidElement> driver=Capabilities();
14
15         // Syntax
16         // driver.findElementByAndroidUIAutomator("attribute("value")");
17
18         driver.findElementByAndroidUIAutomator("text(\"Views\")").click();
19         driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
20
21         // Syntax
22         // driver.findElementsByAndroidUIAutomator("new UiSelector().property(value)");
23
24         // Display of number of elements which have the clickable feature with value true from the list
25         System.out.println(driver.findElementsByAndroidUIAutomator("new UiSelector().clickable(value)").size());
26
27     }
28
29 }
30
```

Figure 4.17

Chapter 5: Appium Gestures

Tap Gesture

Until now, the way to transmit a text to the emulator was presented, but what if the user wish to select a text. For this it is recommended to create a new class named “gesture” which inherits the base class just like the ”basics” class before. In addition, the driver object need to initialize like in “basics” class. The required text is place in the “Views” element so in order to access it follow the example for the Preference element. Initialize an instance called “t” of class TouchAction that has passed to its constructor the “driver” object. Also, if an error is display by Eclipse, click on the errors and the select the import of the necessary Appium library. Furthermore, it is necessary to import several libraries that unfortunately are not imported automatically by Eclipse. The “import static” provide the possibility to access directly the member of its class without class or any object. The next step is to navigate to the “Views” menu.

With a variable “expandList” of class WebElement, the element of Expandable List option is provided. The TouchAction class has a method called “tap” which converts the provided element to the tapOptions type. It uses a little complicated way of accessing the data from the variable “expandList”, so it is not explain in details. Look at Figure 5.2 for the code in the “gestures” class.

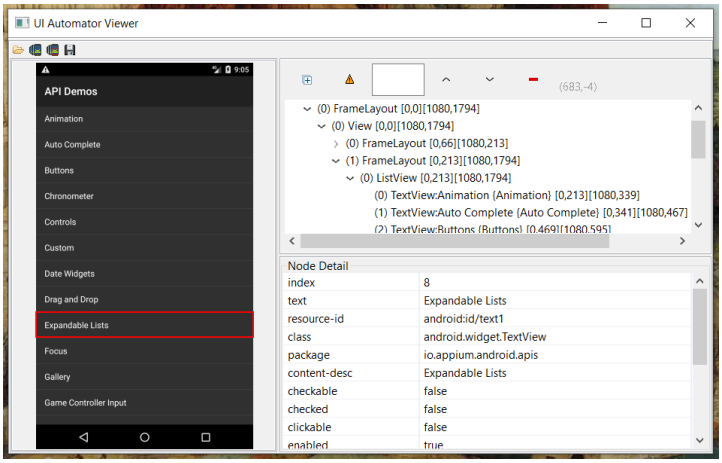


Figure 5.1

```java
package package32;

import java.net.MalformedURLException;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.WebElement;
import io.appium.java_client.TouchAction;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;

import static io.appium.java_client.touch.TapOptions.tapOptions;
import static io.appium.java_client.touch.offset.ElementOption.element;

public class gestures extends base {
             // 31.
        public static void main(String[] args) throws MalformedURLException {

            AndroidDriver<AndroidElement> driver=Capabilities();
            driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
            driver.findElementByXPath("//android.widget.TextView[@text='Views']").click();

            // 32.
            TouchAction t = new TouchAction(driver);

            // Tap
            WebElement expandList = driver.findElementByXPath("//android.widget.TextView(@text='Expandable Lists']");
            t.tap(tapOptions().withElement(element(expandList))).perform();

        }
}

```

Figure 5.2

Long Press Gesture

From the View Further, let us navigate to the “1. Custom Adapter” menu using the findElementByXPath method.

In mobile development, there is a gesture of keeping pressed on a button or link in order to see some option related with that element. For this there is the method named longPress that has as argument the longPressOption object. With the help of WebElement class, all of the elements related with the “People Names” are extracted. The method withDuration sets the amount of time the tap is hold. The release method will the stop the tap, as a user who takes their finger from the screen. The perform method is an equivalent with the click method and it triggers the action set up to the TouchAction object. The LongPressOptions library and the Java time library need to be imported as well like in Figure 5.4.

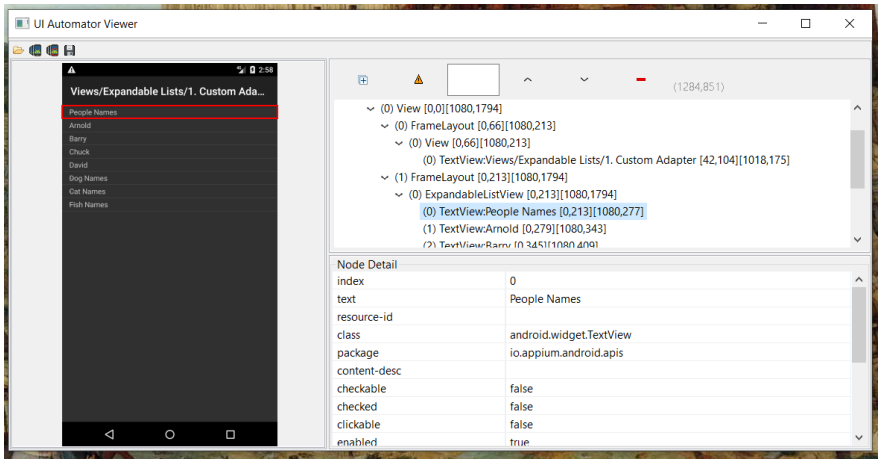


Figure 5.3

```java
package package33;

import java.net.MalformedURLException;

import org.openqa.selenium.WebElement;
import static io.appium.java_client.touch.TapOptions.tapOptions;
import static io.appium.java_client.touch.LongPressOptions.longPressOptions;

import io.appium.java_client.TouchAction;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;

import static java.time.Duration.ofSeconds;
import static io.appium.java_client.touch.offset.ElementOption.element;

public class gestures extends base {
            // 31.
        public static void main(String[] args) throws MalformedURLException {

            AndroidDriver<AndroidElement> driver=Capabilities();
            driver.findElementByXPath("//android.widget.TextView[@text='Views']").click();
            // 32.
            TouchAction t = new TouchAction(driver);
            // Tap
            WebElement expandList = driver.findElementByXPath("//android.widget.TextView(@text='Expandable Lists']");
            t.tap(tapOptions().withElement(element(expandList))).perform();
            // 33.
            driver.findElementByXPath("//android.widget.TextView[@text='1. Custom Adapter']").click();
            WebElement pn=driver.findElementByXPath("//android.widget.textView[@TextView[@text='People Names");

            t.longPress(longPressOptions().withElement(element(pn)).withDuration(ofSeconds(2))).release().perform();

        }
}
```

Figure 5.4

Search through all the Classes

Make a new class named “swipedemo” which inherits the base class and initialize the “driver” object as previously. Navigate through the Views menu to the Date Widgets with the help of the findElementByPath method. Afterwards use the findByElementByAndroidUIAutomator method to select the Inline option. The interface of a clock should appear on the emulator screen. Take a screenshot of the screen with UI Automator Viewer as in Figure 5.5. With the knowledge

gain until now, there would be the temptation of finding the element by class. However, at a second look there is a special character known the dollar sign “$” in the name of the class.

The Appium server does not recognize special character, so this solution cannot be used. A solution to this problem is to use the “*” character. In programing it is used often, import the entire library. In this automation testing the “*” character will search through all of the class in order to find the ask parameter with the require values. Please look at the code in Figure 5.6.

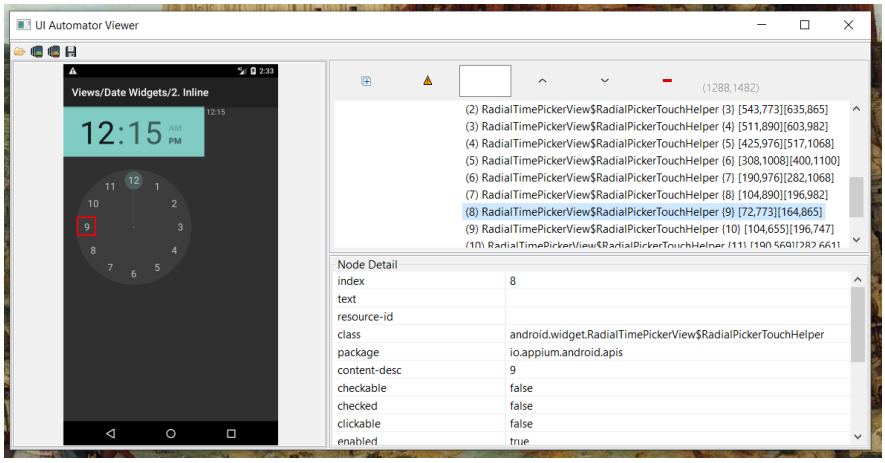


Figure 5.5

```java
package package34;

import java.net.MalformedURLException;
import java.util.concurrent.TimeUnit;

import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;

public class swipedemo extends base{

    public static void main(String[] args) throws MalformedURLException {

        // 34.

        AndroidDriver<AndroidElement> driver = Capabilities();

        driver.manage().timeouts().implicitlyWait(10,TimeUnit.SECONDS);

        driver.findElementByXPath("//android.widget.TextView[@text='Views']").click();

        driver.findElementByXPath("//android.widget.TextView[@text='Date Widgets']").click();

        driver.findElementByAndroidUIAutomator("text(\"2. Inline\")").click();

        driver.findElementByXPath("//*[@content-desc='9']").click();


    }

}
```

Figure 5.6

Swipe Gesture

In a real life scenario, a user would use the clock app to set an alarm. Some clock application use as interface an analog clock to set the hour and the minute. The user puts their finger on highlighted hour and he moves the finger to the desired hour. This gesture is known as swipe. This is very similar with the movement of the finger between two taps. In the code, the syntax of the tapping gesture is used between the two elements. To identify the element

corresponding with the 45 minutes and the element corresponding with the 15 minutes, use the same method as in the case of the hour. In a WebElement, keep the start position of the swap and in the second WebElement the destination point. The method moveTo is used to connect the point having as a parameter the destination position. Do not forget to import the TouchAction library.

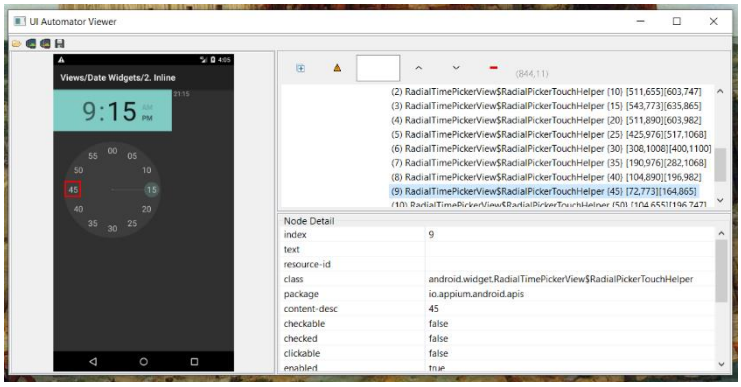


Figure 5.7

```java
package package35;

import java.net.MalformedURLException;
import java.util.concurrent.TimeUnit;

import org.openqa.selenium.WebElement;
import io.appium.java_client.TouchAction;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;

import static java.time.Duration.ofSeconds;
import static io.appium.java_client.touch.offset.ElementOption.element;
import static io.appium.java_client.touch.LongPressOptions.longPressOptions;

public class swipedemo extends base{
    public static void main(String[] args) throws MalformedURLException {

        // 34.
        AndroidDriver<AndroidElement> driver = Capabilities();
        driver.manage().timeouts().implicitlyWait(10,TimeUnit.SECONDS);

        driver.findElementByXPath("//android.widget.TextView[@text='Views']").click();
        driver.findElementByXPath("//android.widget.TextView[@text='Date Widgets']").click();

        driver.findElementByAndroidUIAutomator("text(\"2. Inline\")").click();
        driver.findElementByXPath("//*[@content-desc='9']").click();

        // 35.
        TouchAction t=new TouchAction(driver);

        WebElement first=driver.findElementByXPath("//*[@content-desc='15']");
        WebElement second=driver.findElementByXPath("//*[@cotent-desc='45'");

        t.longPress(longPressOptions().withElement(element(first)).withDuration(ofSeconds(2))).moveTo(element(second)).release();
    }
}
```

Figure 5.8

Scrolling Gesture

In the real life in a menu can be more items than those that fit on the screen. Therefore, it is necessary to scroll in order to reach an item at the bottom of the list. Create a new class named “scrollingdemo” which inherits the base class and initialize the “driver” object as previously. Let us propose to scroll in the Views menu until the last item named “Radio Group”. To understand the syntax of the UIScrollable class, it is require a better understanding of the Android developer environment. For this reason, we will not go into more details. Broadly, this class scrolls until it

selects an element. This element can be identified only when it appears on the screen on the device. Thereby it scrolls until the wanted item appears on the screen. Into the scrollIntoView, the name of the parameter and the value is introduced. The item is identified by the text parameter that has the value “Radio Group” in this case.

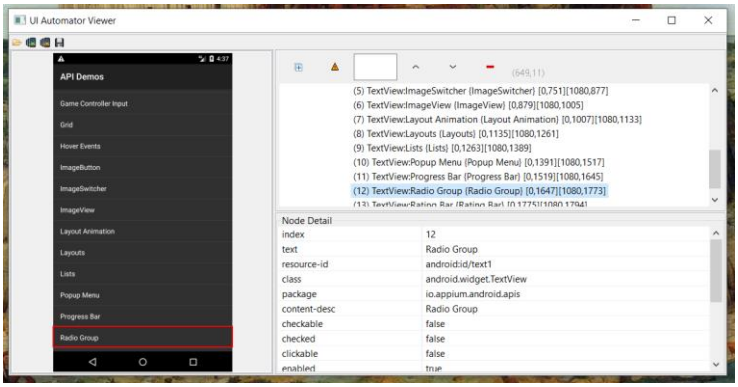


Figure 5.9

```java
package package36;

import java.net.MalformedURLException;
import java.util.concurrent.TimeUnit;

import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;

public class scrollingdemo extends base{
    public static void main(String[] args) throws MalformedURLException {

        // 36.
        AndroidDriver<AndroidElement> driver = Capabilities();
        driver.manage().timeouts().implicitlyWait(10,TimeUnit.SECONDS);

        driver.findElementByXPath("//android.widget.TextView[@text='Views']").click();
        driver.findElementByAndroidUIAutomator("new Uiscrollable(new UiSelector()).scrollIntoView(text(\"Radio Group\"));");
    }
}

```

Figure 5.10

Drag and Drop Gesture

One gesture very common in nowadays operating system is the drag and drop. Let us propose to move one red circle over another one. The syntax is very similar with the swipe gesture. There are two implementation of this gesture. The first one is similar with the swipe gesture but without the withDuration method. The second one is a shorter version of the first one. It does not need the longPressOption class, but it manipulates the graphical element directly. Second variant is good for a simple drag and drop. For composed action, it is better to use the first variant.

To move the left circle over the right one, create a new class named “dragdropdemo” which inherits the base class and initialize the “driver” object as previously. Use the UIAutomator Viewer

to navigate in the Views and afterward in the Drag and Drop item using the findElementByXPath method. Use two WebElement variables to keep and identify the dragged red circle and the red circle over it is drop. Look at Figure 5.11 and Figure 5.12. Instantiate an object of class TouchAction and import the TouchAction library. Look in the Figure 5.13 for the code of the class.

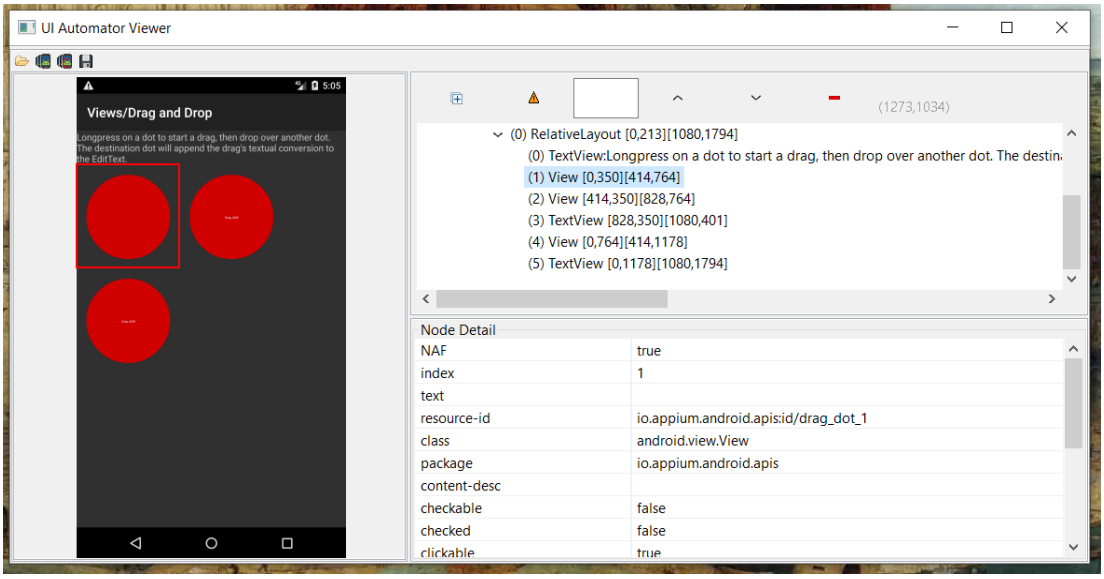


Figure 5.11

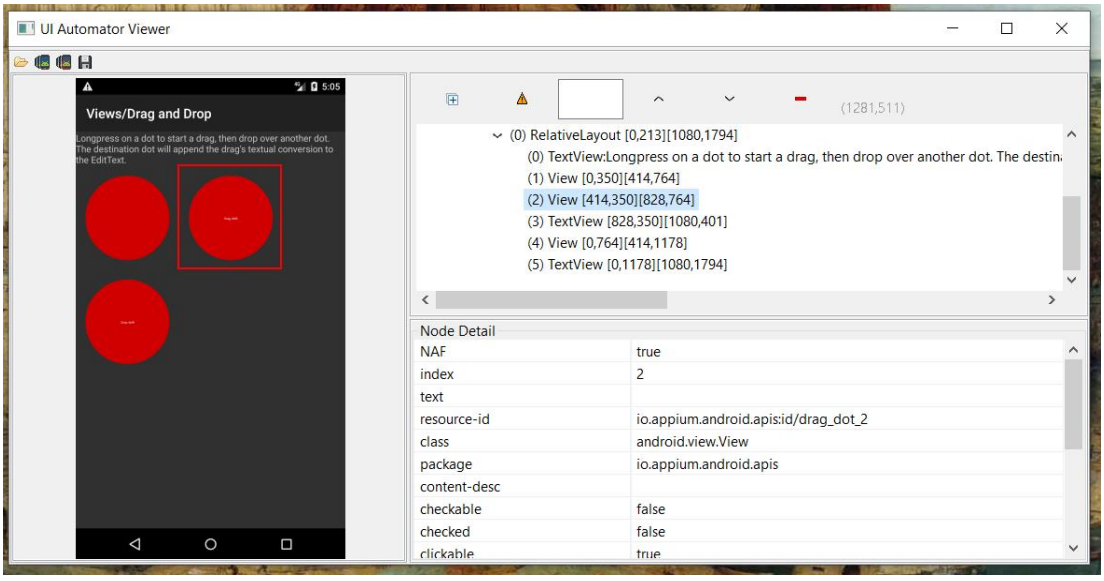


Figure 5.12

```java
package package37;

import java.net.MalformedURLException;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.WebElement;
import io.appium.java_client.TouchAction;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;

import static io.appium.java_client.touch.LongPressOptions.longPressOptions;
import static io.appium.java_client.touch.offset.ElementOption.element;

public class dragdropdemo extends base{
    public static void main(String[] args) throws MalformedURLException {

        // 37.
        AndroidDriver<AndroidElement> driver = Capabilities();
        driver.manage().timeouts().implicitlyWait(10,TimeUnit.SECONDS);

        driver.findElementByXPath("//android.widget.TextView[@text='Views']").click();
        driver.findElementByXPath("//android.widget.TextView[@text='Drag and Drop']").click();

        WebElement source = driver.findElementsByClassName("android.view.View").get(0);
        WebElement destination = driver.findElementsByClassName("android.view.View").get(1);

        TouchAction t = new TouchAction(driver);

        //t.longPress(longPressOptions().withElement(element(source))).moveTo(element(destination)).release().perform();

        // Second Variant
        t.longPress(element(source)).moveTo(element(destination)).release().perform();

    }
}
```

Figure 5.13

Chapter 6: Appium on Android Device

Connect Appium to an Android Device

Until now, our test have been done using the emulator provide by the Android Studio. Appium can work not only on an emulator but also on a real device. Enter on the following link: "https://developers.google.com/web/tools/chrome-devtools/remote-debugging".

On this web site, there are information about the requirements necessary to connect Appium with an Android device. For example, it is necessary to have a Chrome version 32 to newer, have the USB driver installed and have a USB cable. The Android Version need to be 4 or later and the smartphone need to have the Chrome browser installed.

On the smartphone enter in the Settings and look for the "Developer Options". If it cannot be found, enter in the About phone option from the Settings and tap seven time on the Built Number. After that, enter in the Developer Options and check on the USB debugging. Now the smartphone is recognizable by the computer. To check this thing open the Command Prompt and write the code from Figure 6.1. It should provide a list of the connected device to the computer.

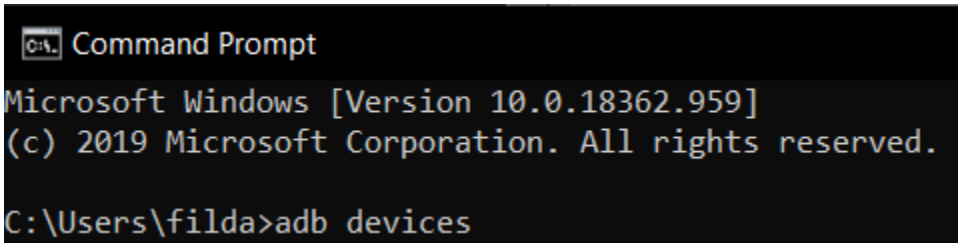


Figure 6.1

Note: If the computer does not recognize the connected device when typing the command "adb devices" in Command Prompt or displays the message "Unauthorized" use the following commands: "adb kill-server", "start –server", "adb devices". With these three command, the service restarts.

Capabilities Configuration

To test the code on a real device, go the base class from Chapter 4. Change the value from the name of the Android Virtual Device to "Android Device". That is all the change require to be made. However, sometimes the developer want to write the code on the emulator and then to test the code on the real device. In order not to change or hardcode the file too much let us made some changes in order to make the switch between the two very easy.

In Eclipse click on the base class and copy it. When you paste it, Eclipse will ask to put a new name. Put “hybridbase” as the name. To the Capabilities method, introduce a new argument of type string. Name it “device”. This variable offers to the method, the device on which the test are done. In the situation in which the variable is “emulator”, the code executes on the Android Virtual Device. If the variable is “real”, the code executes on the real Android device. For code, please see the Figure 6.2 and Figure 6.3.

```
package package42;

import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;
import io.appium.java_client.remote.MobileCapabilityType;

import java.io.File;
import java.net.MalformedURLException;
import java.net.URL;
import org.openqa.selenium.remote.DesiredCapabilities;

public class hybridbase {


    public static AndroidDriver<AndroidElement> Capabilities(String device) throws MalformedURLException{

        File f = new File("src");
        File fs = new File(f,"ApiDemos-debug.apk");

        DesiredCapabilities cap = new DesiredCapabilities();

        if(device.equals("emulator")) {
            cap.setCapability(MobileCapabilityType.DEVICE_NAME, "ClaudiuEmulator2");
        }else if(device.equals("real")){
            cap.setCapability(MobileCapabilityType.DEVICE_NAME, "Android Device");
        }

        cap.setCapability(MobileCapabilityType.AUTOMATION_NAME, "uiautomator2");
        cap.setCapability(MobileCapabilityType.APP, fs.getAbsolutePath());

        AndroidDriver<AndroidElement> driver = new AndroidDriver<>(new URL("http://127.0.0.1:4723/wd/hub"), cap);
        return driver;

    }
}

```

Figure 6.2

```
package package42;

import java.net.MalformedURLException;
import java.util.concurrent.TimeUnit;

import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;

public class basics extends hybridbase{

    public static void main(String[] args) throws MalformedURLException {

        // 42.
        AndroidDriver<AndroidElement> driver=Capabilities("real");

        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

        driver.findElementByXPath("//android.widget.TextView[@text='Preference']").click();
        driver.findElementByXPath("//android.widget.TextView[@text='3. Preference dependencies']").click();

        driver.findElementById("android:id/checkbox").click();
        driver.findElementByXPath("(//android.widget.RelativeLayout)[2]").click();

        driver.findElementByClassName("android.widget.EditText").sendKeys("hello");
        driver.findElementsByClassName("android.widget.Button").get(2).click();


    }
}

```

Figure 6.3

Appium on Chrome Mobile Browser

Until now, the way of testing the native application of the Android Platform was covered. Nowadays multiple web sites want their application to run correctly on a mobile browser.

Create a new class called “basechrome”. It will have a method “Capabilities” as the previous “base” class. Initialize a variable named “cap” of type DesiredCapabilies. Set the “DEVICE_NAME” capability with “Android Device” for testing on the real device. Set the “BROWER_NAME” capability with the name of the mobile browser in which the web application is tested. In this case, it is Chrome. Initialize a variable driver as in the previous “base” class.

Make a new class named “browser” in which an AndroidDriver object that is taken from the “Capabilities” method, just as in the previous “basics” class. The actual web testing has an easy syntax. Only access the get method of the AndroidDriver and insert the web address of the desired web site. In this case, it is “https://www.google.com/”. See the Figure 6.4 and Figure 6.5.

```
1  package package44;
2
3  import io.appium.java_client.android.AndroidDriver;
4  import io.appium.java_client.android.AndroidElement;
5  import io.appium.java_client.remote.MobileCapabilityType;
6
7  import java.net.MalformedURLException;
8  import java.net.URL;
9  import org.openqa.selenium.remote.DesiredCapabilities;
10
11 public class basechrome {
12
13
14     public static AndroidDriver<AndroidElement> Capabilities() throws MalformedURLException{
15
16         // 44.
17
18         DesiredCapabilities cap = new DesiredCapabilities();
19
20         cap.setCapability(MobileCapabilityType.DEVICE_NAME, "Android Device");
21
22         cap.setCapability(MobileCapabilityType.BROWSER_NAME, "Chrome");
23
24         AndroidDriver<AndroidElement> driver = new AndroidDriver<>(new URL("http://127.0.0.1:4723/wd/hub"), cap);
25         return driver;
26
27     }
28 }
29
```

Figure 6.4

```
1  package package44;
2
3  import java.net.MalformedURLException;
4  import io.appium.java_client.android.AndroidDriver;
5  import io.appium.java_client.android.AndroidElement;
6
7  public class browser extends basechrome{
8
9      public static void main(String[] args) throws MalformedURLException {
10
11         // 44.
12         AndroidDriver<AndroidElement> driver=Capabilities();
13
14         driver.get("http://google.com");
15
16     }
17 }
18
```

Figure 6.5

Connect your real device to the computer using an USB cable. As in Figure 6.1, follow again the code in the Command Prompt. Open the Chrome browser from your computer, enter in the Google search page and press “Settings” in the Chrome menu. Afterwards select “More tools”, and find the “Developer tools”. Finally look for “Remote devices”. For some version of chrome write in the URL section the next address “chrome://inspect/#devices”. Here you will find the connected mobile device. There is a textbox next to the device name. You can insert a web address and it opens in the mobile device, as in Figure 6.6.

Let us go to the Facebook page and insert some user credentials. Enter on the “https://m.facebook.com/” on the computer Chrome browser and click right on the page. Select the Inspect option. Alternatively, you can use the key shortcut “Ctrl+Shift+I”. A window opens on the right site of the browser that displays the HTML code of the website, as I Figure 6.8.

Move your mouse over the email textbox and click right. Select the Inspect option as previously. The section of the code that is responsible to that graphic element is highlighted on right side of the browser. Right click on it and select “Copy” and then “Copy XPath”. In the “browser” class, modify the get method with the “http://facebook.com” address instead of the Google address. In the findElementByXPath, paste what it is copied from the Chrome Inspector. Once the selected element, transmit the name of the user with the sendKeys method.

Right click on the Password textbox and select Inspect. The code, which corresponds to this element, it is highlighted as well. In Figure 6.9, it can be observed that the property “name” with value “pass” can be used to identify the element. Use the method findElementByName. We also want to input a username as before. The last thing is to press the Log In button. We can use the parameter “value” named “Log In” to identify the corresponding button using Xpath form the Figure 6.10. The code is in Figure 6.7.

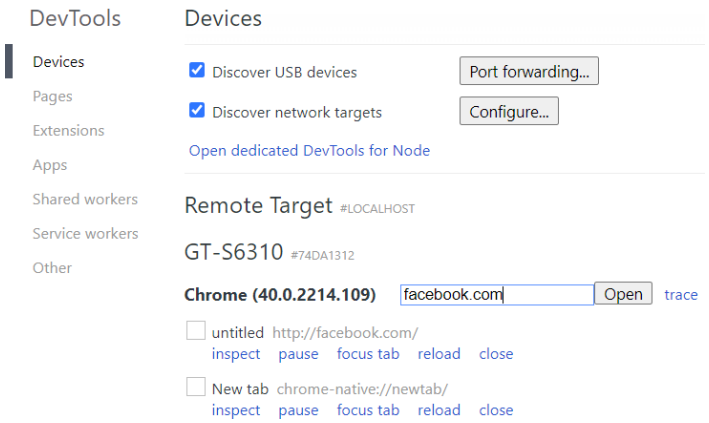


Figure 6.6

```java
package package46;

import java.net.MalformedURLException;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;

public class browser extends basechrome{

    public static void main(String[] args) throws MalformedURLException {

        // 46.
        AndroidDriver<AndroidElement> driver=Capabilities();

        driver.get("http://facebook.com");

        driver.findElementByXPath("//*[@id=\'u_0_1']/div[1]/div/input").sendKeys("qwerty");

        driver.findElementByName("pass").sendKeys("12345");

        driver.findElementByXPath("//button[@value='Log In']").click();

    }
}

```

Figure 6.7

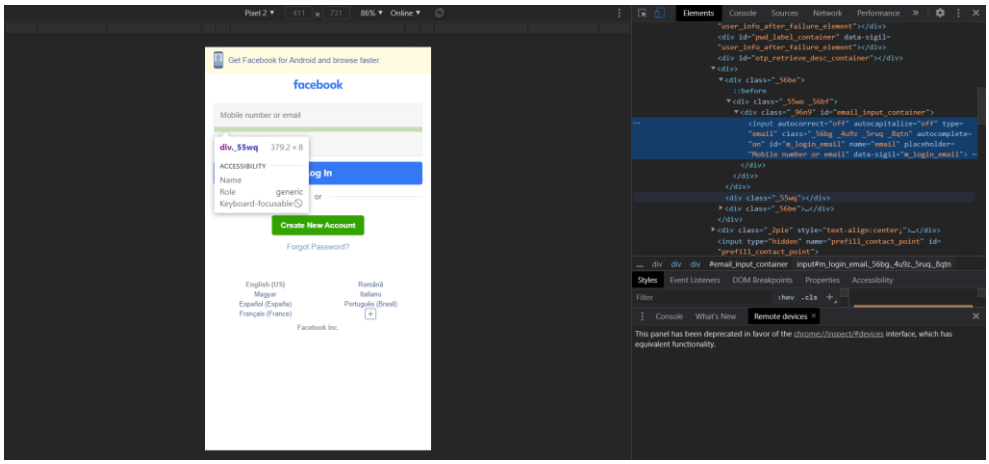


Figure 6.8

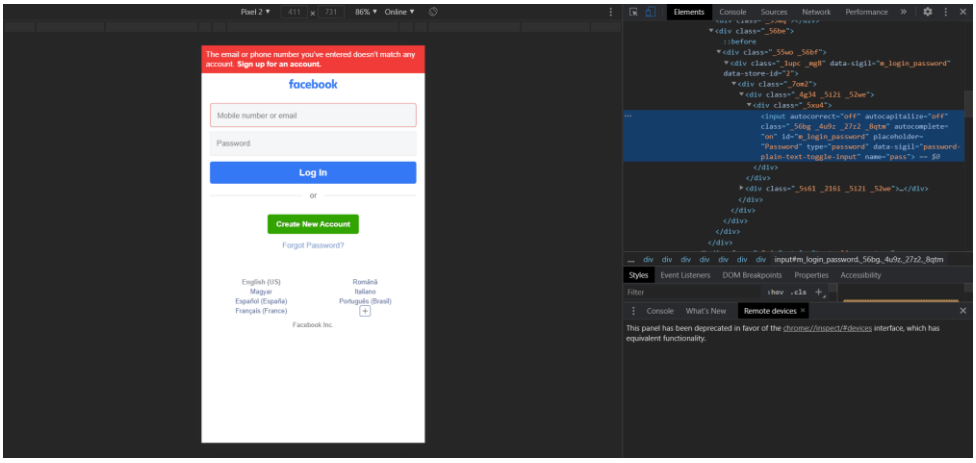


Figure 6.9

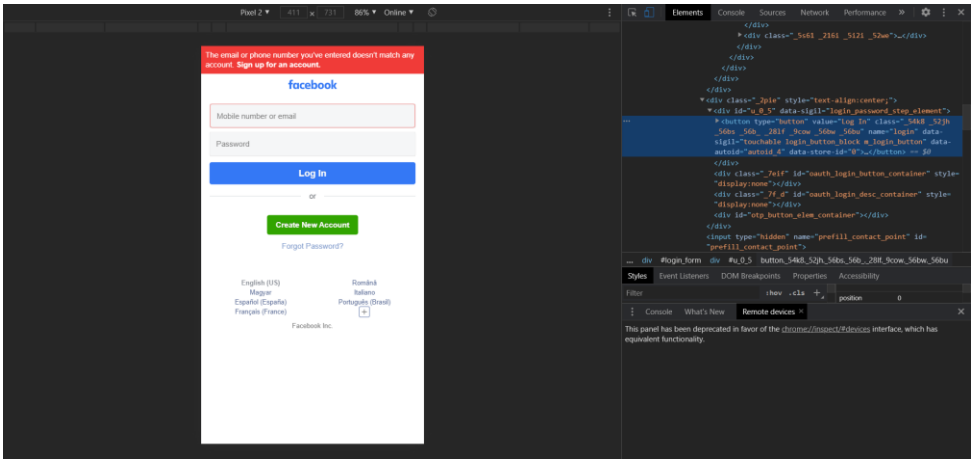


Figure 6.10

Find By Cascading Style Sheets

It is worth mentioning that there are two types of site. There are web responsive sites and the ones that are not. A web responsive site changes its format according to the type of device it is display, for example for a computer or a smartphone. The non-responsive web sites have different versions for each platform. In this case, the address "m.facebook.com" is the mobile version of the Facebook page. The extension "m." from the beginning of the address redirects the mobile browser from the desktop version. The problem with having separate versions for each platform in the case of automation testing is the fact that the class, names or packages cannot correspond between the versions. If in the case of the Google site, it is possible to write test cases regrades of the platform, in the case of Facebook, this thing is not possible.

As an exercise, enter on the "http://m.cricbuzz.com". Let us press on the Menu button. Right click on it and select Inspect. In the "basechrome" class from before, delete everything after the get method. It is wanted to enter on the "http://cricbuzz.com" so modify the "get" method accordantly. In addition, let us press the Menu button in order to come back to the main page. Using the Xpath, recognize the Main button after the "a ref" parameter. The value attributed to this parameter is "#menu", like in Figure 6.11. In Chrome, press on the Menu button and then right click on the Home. Select Inspect, and let us take the title of the element as the identification, as highlighted in Figure 6.12. In addition, a new method called findElementbyCssSelector takes of the CSS component of the site. This method accept the same format like the previous methods as finding by the class name. The code is provided in Figure 6.13.

This site use in this example has ads, which in development and testing are disable. To see how they affect the everyday experience let us print the current URL address. As it can be seen that after the code is run, the address is different from the one it is coded to the Appium server. This is because of the personalize ads that users receive. As it can be seen, the testing it is not disturbed or affected in any way by the presence of the ads.

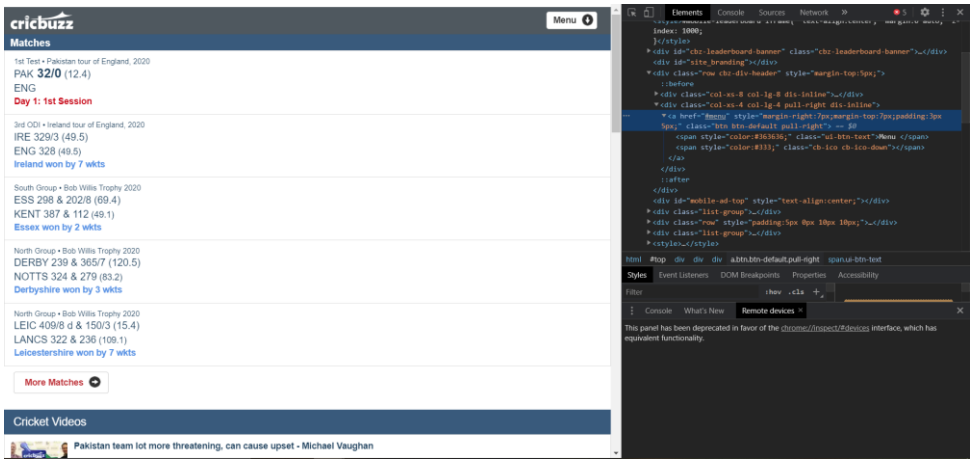


Figure 6.11

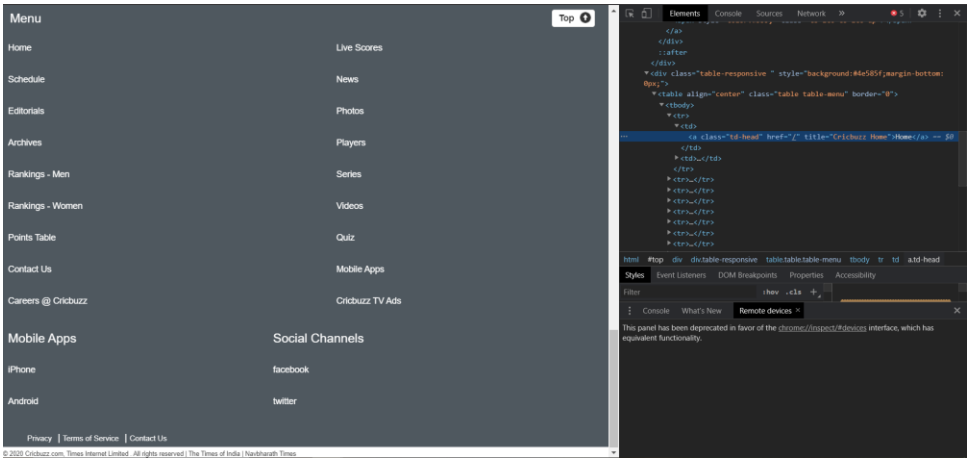


Figure 6.12

```java
package package47;

import java.net.MalformedURLException;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;

public class browser extends basechrome{

    public static void main(String[] args) throws MalformedURLException {

        // 47.
        AndroidDriver<AndroidElement> driver=Capabilities();

        driver.get("http://cricbuzz.com");
        driver.findElementByXPath("//a[@href='menu'").click();
        driver.findElementByCssSelector("a[title='Cricbuzz Home").click();

        System.out.println(driver.getCurrentUrl());

    }
}
```

Figure 6.13

Note: If the computer does not recognize the connected device when typing the command “adb devices” in Command Prompt or displays the message Unauthorized” use the following commands: “adb kill-server”, “start –server”, “adb devices”. With these three command, the service will restart.

Scrolling on Mobile Websites

To scroll in the mobile Chrome browser, the JavaScript elements are used. The class JavascriptExecutor takes the role of the Android driver on the web page. This class comes from Selenium. Create an instance of the JavascriptExecutor named “jse” which it is converted form the Android driver. The method executeScript has two parameters, the first one is the script and the second one is the argument. The last parameter is optional. The script “windows.scrollBy(0,480)” scrolls the webpage with 480 pixels on the vertical axis. Scrolling in the horizontal axis is set to zero.

```
1  package package48;
2
3  import java.net.MalformedURLException;
4  import org.openqa.selenium.JavascriptExecutor;
5
6  import io.appium.java_client.android.AndroidDriver;
7  import io.appium.java_client.android.AndroidElement;
8
9  public class browser extends basechrome{
10
11     public static void main(String[] args) throws MalformedURLException {
12
13         // 47.
14         AndroidDriver<AndroidElement> driver=Capabilities();
15
16         driver.get("http://cricbuzz.com");
17         driver.findElementByXPath("//a[@href='menu'").click();
18         driver.findElementByCssSelector("a[title='Cricbuzz Home").click();
19
20         System.out.println(driver.getCurrentUrl());
21
22         // 48.
23         JavascriptExecutor jse=(JavascriptExecutor) driver;
24         jse.executeScript("window.scrollBy(0,480)","");
25
26     }
27 }
28
```

Figure 6.14

Chapter 7: E-Commerce Application

Introduction to the App

For exercising the presented knowledge until now, let us do some Appium testing using a simple online commerce app. Let us propose a few things to be tested:

1. Fill the form details and verify Toast error messages displayed appropriately for wrong inputs.
2. Shop the items in the app by scrolling to specific product and add to cart.
3. Validate if the items selected in the page 2 are matching with items displayed in the checkout page.
4. Validate the total amount displayed in the checkout page matches with the sum of product amounts selected for shopping.
5. Validate mobile gestures working for links (long press) and navigate to web view.
6. Verify if the user can do operations on web view and can navigate back to native app if needed.

In order to install the e-commerce application on the emulator, download the "General-Store.apk" file. Open Command Prompt and go to the location of the Android SDK using the "cd" command. Then write "adb install" together with the path to the APK file. Alternatively, start the emulator and drag and drop the APK file over the emulator screen.

Test 1: Filling the Form Details

First thing it is needed it to create the "base" class for the test cases. Use previous "base" class as a model. The difference is that instead of the "ApiDemos-debug.akp" it is "General-Store.apk". In this way, the tests perform on the e-commerce app instead of the demonstration app used until now. See the Figure 7.1.

Make a class named "ecommenrce_tc_1" which inherits the base class and have a variable "driver" of the AndroidDriver class. Appium have different method and syntaxes that can perform the same task. For diversity, a new way of writing the Appium code is presented. The method findElement can identify different types of design elements with the help of the By class. This class has the way in which an element can be identify, for example by Id or by Xpath.

On the main page of the app, there are four main facilities. One is to input a country, the second one is to input the name of the user and the last one is third is a check box for gender of the user and the last one a button which open the next page. Let us fill all of these steps in order to enter in the next section of the app.

First, let us fill the name text box with the message "hello". From Figure 7.3, it is possible to find the element by id and to send the wanted message. There is a problem because when

Appium insert the text, the keyboard is displayed. Because of this, the server does not have access to the graphical elements hidden by the keyboard. The Android driver have method named hideKeyboard that closes it. Form Figure 7.4 let us identify the element using the Xpath. For selecting the country, a list appears on the screen and the user need to scroll thorough and to select the wanted country, in this case let us choose Argentina. The list can be access using its id as in Figure 7.5. Use the Scrollable class to go to Argentina. With the Xpath, search from the list the element, which has the name of the country as text parameter like in Figure 7.6. Click on this option. Identify the “Let’s shop” button using its id and click on it.

```
package package51;

import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;
import io.appium.java_client.remote.MobileCapabilityType;

import java.io.File;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.concurrent.TimeUnit;

import org.openqa.selenium.remote.DesiredCapabilities;

public class base {

    public static AndroidDriver<AndroidElement> Capabilities() throws MalformedURLException{

        // 51.
        File appDir=new File("src");
        File app=new File(appDir,"General-Store.apk");
        DesiredCapabilities capabilities = new DesiredCapabilities();

        capabilities.setCapability(MobileCapabilityType.DEVICE_NAME, "Claudiuemulator2");
        capabilities.setCapability(MobileCapabilityType.AUTOMATION_NAME, "uiautomator2");

        capabilities.setCapability(MobileCapabilityType.NEW_COMMAND_TIMEOUT, 15);
        capabilities.setCapability(MobileCapabilityType.APP, app.getAbsolutePath());

        AndroidDriver<AndroidElement> driver = new AndroidDriver<>(new URL("http://127.0.0.1:4723/wd/hub"), capabilities);
        driver.manage().timeouts().implicitlyWait(10,TimeUnit.SECONDS);

        return driver;

    }
}
```

Figure 7.1

```
package package51;

import java.net.MalformedURLException;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;

public class ecommerce_tc_1 extends base {

    public static void main(String[] args) throws MalformedURLException {

        // 51.
        AndroidDriver<AndroidElement> driver=Capabilities();
        driver.manage().timeouts().implicitlyWait(10,  TimeUnit.SECONDS);

        driver.findElement(By.id("com.androidsample.generalstore:id/nameField")).sendKeys("Hello");
        driver.hideKeyboard();
        driver.findElement(By.xpath("*//[@text='Female']")).click();
        driver.findElement(By.id("android:id/text1")).click();

        driver.findElementByAndroidUIAutomator("new UiScrollable(new UiSelector()).scrollIntoView(\"Argentina\"));");

        // Second variant of the previous line
        // driver.findElement(MobileBy.AndroidUIAutomator("new UiScrollable(new UiSelector().scallable(true).instance(0)).scrollIntoView("
        //      + "new UiSelector().textMatches(\""+containedText+"\").instance(0)"));

        driver.findElement(By.xpath("//*text='Argentina']")).click();
        driver.findElement(By.id("com.androidsample.generalstore:id/btnLetsShop")).click();

    }

}
```

Figure 7.2

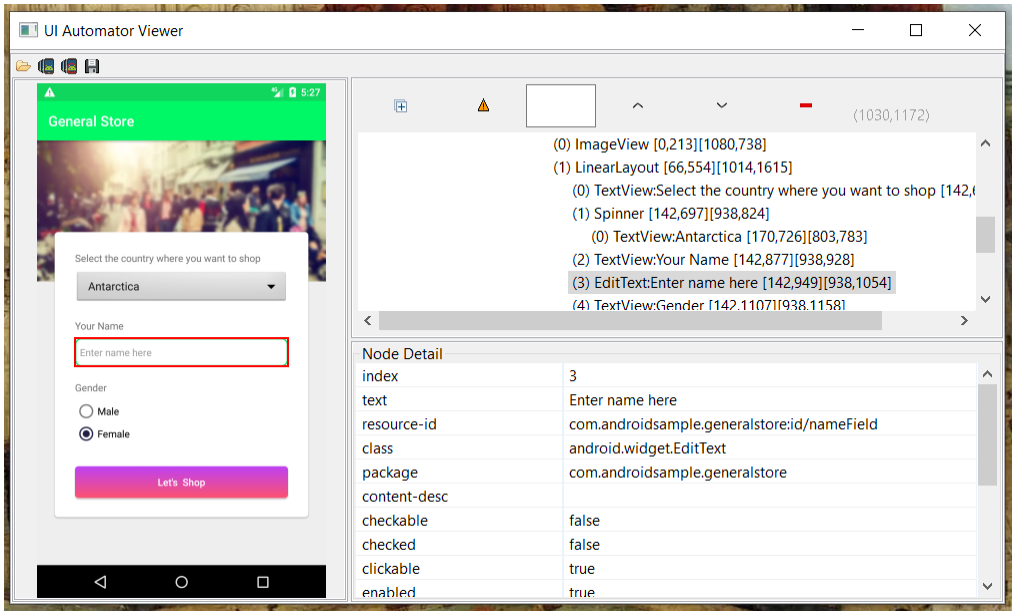


Figure 7.3

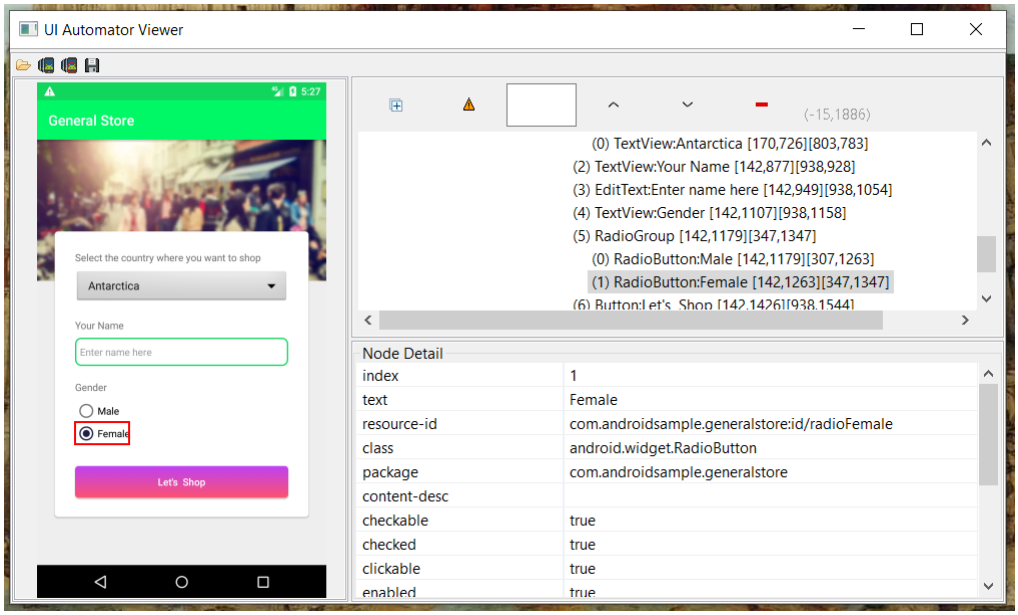


Figure 7.4

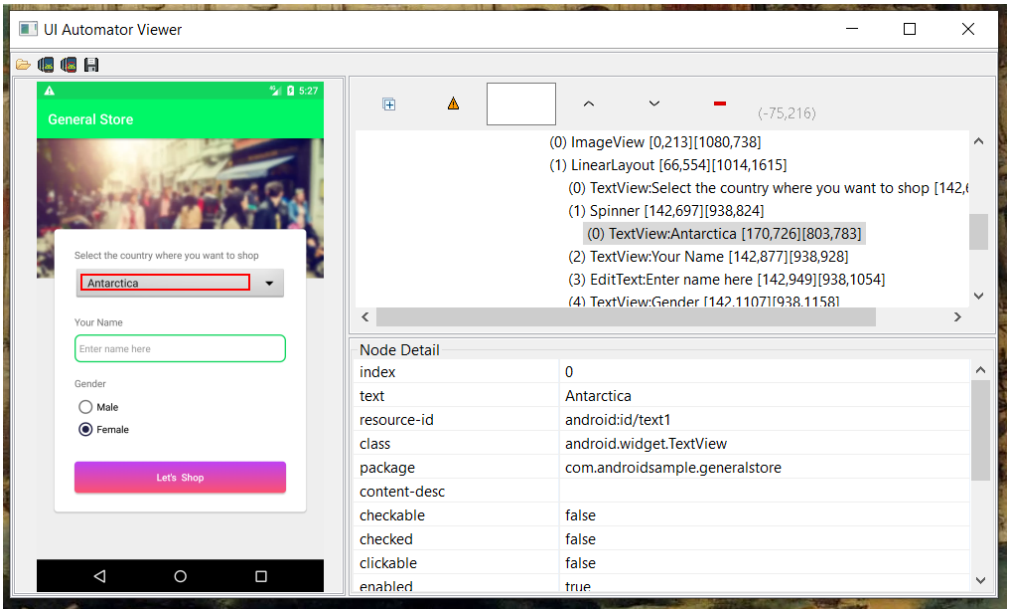


Figure 7.5

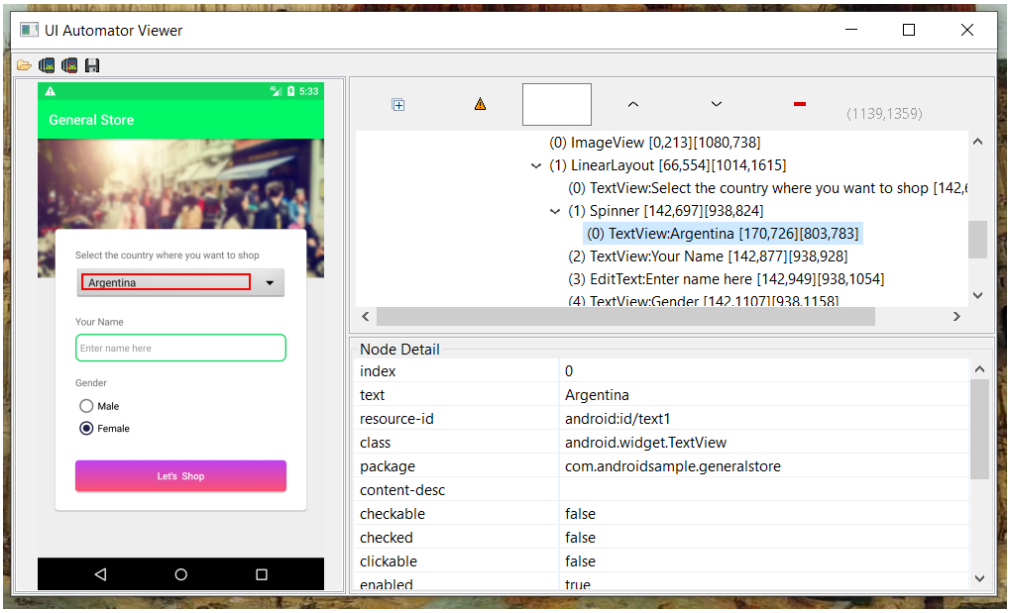


Figure 7.6

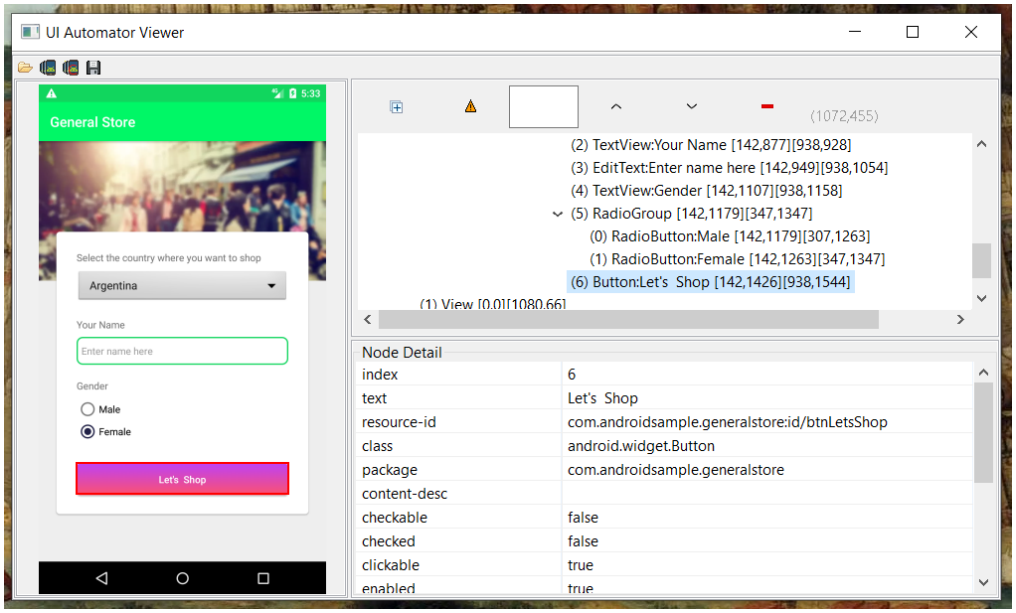


Figure 7.7

Test 1: Toast Messages

A toast provides simple feedback about an operation in a small popup. It only fills the amount of space required for the message and the current activity remains visible and interactive. Toasts automatically disappear after a timeout.

The e-commerce app has toast messages when one of the option is not fill. The UI Automator does not have direct access to this toast message. It is impossible to find the parameters of a toast message from the UI automator viewer. To overcome this problem a convention was made and all the toast messages are in the Toast class. In a page, there can be a multitude of toast messages. To access certain on put after the class, inside the brackets, the index of the wanted toast. All the toast have the form “android.widges.Toast”. For the first message, write “android.widges.Toast[1]” and so on for the next one.

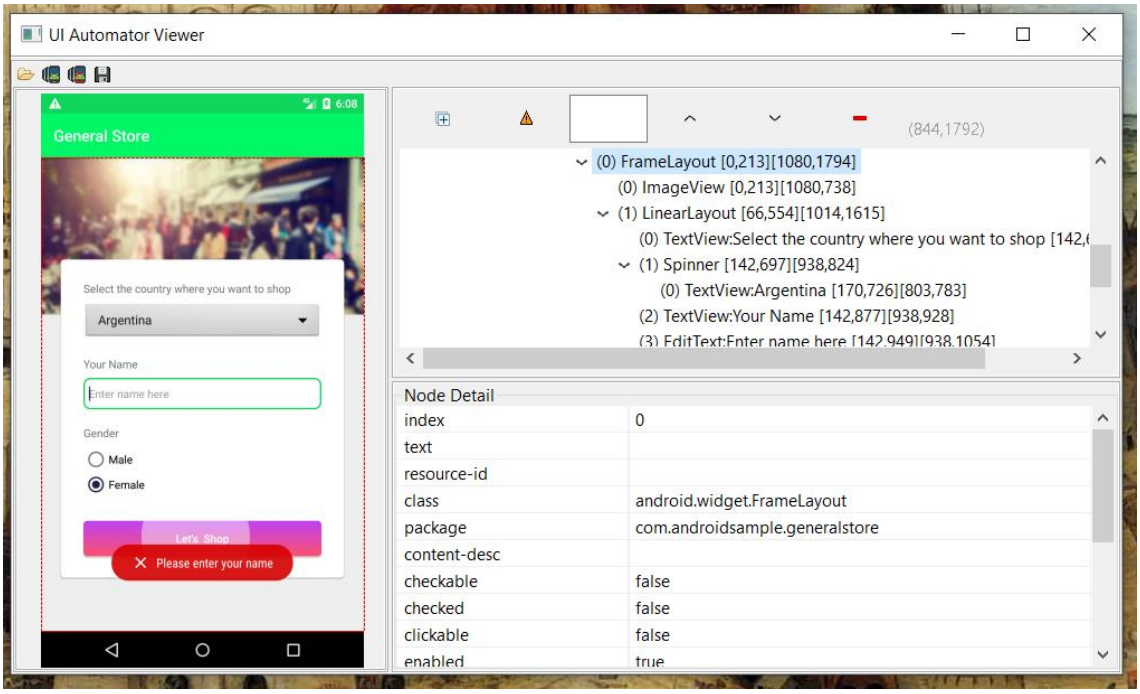


Figure 7.8

```java
package package53;

import java.net.MalformedURLException;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;

public class ecommerce_tc_1 extends base {
    public static void main(String[] args) throws MalformedURLException {

        // 51.
        AndroidDriver<AndroidElement> driver=Capabilities();
        driver.manage().timeouts().implicitlyWait(10,  TimeUnit.SECONDS);

        // The code is commented for testing the Error message
        // driver.findElement(By.id("com.androidsample.generalstore:id/nameField")).sendKeys("Hello");
        // driver.hideKeyboard();
        driver.findElement(By.xpath("*//[@text='Female']")).click();
        driver.findElement(By.id("android:id/text1")).click();

        driver.findElementByAndroidUIAutomator("new UiScrollable(new UiSelector()).scrollIntoView(\"Argentina\"));");

        // Second variant of the previous line
        // driver.findElement(MobileBy.AndroidUIAutomator("new UiScrollable(new UiSelector().scallable(true).instance(0)).scrollIntoView("
        //      + "new UiSelector().textMatches(\""+containedText+"\").instance(0)"));

        driver.findElement(By.xpath("//*text='Argentina']")).click();
        driver.findElement(By.id("com.androidsample.generalstore:id/btnLetsShop")).click();

        // 52.
        String toastMessage=driver.findElement(By.xpath("//android.widget.Toast[1]")).getAttribute("name");
        System.out.println(toastMessage);

    }

}
```

Figure 7.9

Test 2: Scroll through products

In this part, we are interested in scrolling thought the products and add an item to the shopping cart. After it is added, it is necessary to check if it is available to buy. In the UI Automaton Viewer, it can be observe that all of the product have the same id. Therefore, it is possible to get all of the items from the list. Afterwards, search through the all of the ids. If it finds the right one, then add the product to the cart.

However, here is a problem. If we test how many items the test can see with the id "com.androidsample.generalstore:id/productName", something strange happens. It displays the value of two, although the number of items available in the shop is bigger than two. The reason is the fact that the test can only see the elements that appears on the screen, not all the ones. If there are two products on the screen, the result is two. If the screen is larger and there are three product on the screen, the result is three.

A solution to this problem is to use the scrollable function to reach the desired product, but a new problem appears. Using the scrollable function, the test search for the text provided and it will stop scrolling ones the text appears on the screen, no matter if the Add button is visible on the screen. In the case of this app, the Add button in below the name of the product and it is possible to not to be visible on the screen. This means that the Add button cannot be press and the test fails.

The solution to this entire problem can be found in the Appium official GitHub depository available at the following address: "https://github.com/appium/java-client/blob/master/src/test/java/io/appium/java_client/android/AndroidSearchingTest.java".

```
60      @Test public void findScrollable() {
61          driver.findElementByAccessibilityId("Views").click();
62          MobileElement radioGroup = driver
63                  .findElementByAndroidUIAutomator("new UiScrollable(new UiSelector()"
64                  + ".resourceId(\"android:id/list\")).scrollIntoView("
65                  + "new UiSelector().text(\"Radio Group\"));");
66          assertNotNull(radioGroup.getLocation());
67      }
68  }
```

Figure 7.10

In this solution, the entire class, which has the text "Radio Group", is places in a list. In that list, it scrolls until the entire class is on the screen. In this way, all of the graphical elements of the class are displayed on the screen. Let us scroll in the through the product until the "Jordan 6 Rings" is found. The solution is a quite a long line of code from the Figure 7.11.

```
driver.findElement(MobileBy.AndroidUIAutomator("new UiScrollable(new UiSelector().resourceId()"
        + ".resourceId(\"com.androidsample.generalstore:id/rvProductList\")).scrollableIntoView("
        + "new UiSelector().textMatches(\"Jordan 6 Rings\").instance(0))"));
```

Figure 7.11

```
package package55;

import java.net.MalformedURLException;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import io.appium.java_client.MobileBy;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;

public class ecommerce_tc_2 extends base {
    public static void main(String[] args) throws MalformedURLException {

        // 55.
        AndroidDriver<AndroidElement> driver=Capabilities();
        driver.manage().timeouts().implicitlyWait(10,  TimeUnit.SECONDS);

        driver.findElement(By.id("com.androidsample.generalstore:id/nameField")).sendKeys("Hello");
        driver.hideKeyboard();
        driver.findElement(By.xpath("*//[@text='Female']")).click();
        driver.findElement(By.id("android:id/text1")).click();
        driver.findElementByAndroidUIAutomator("new UiScrollable(new UiSelector()).scrollIntoView(\"Argentina\"));");

        // Second variant of the previous line
        // driver.findElement(MobileBy.AndroidUIAutomator("new UiScrollable(new UiSelector().scallable(true).instance(0)).scrollIntoView("
        //      + "new UiSelector().textMatches(\""+containedText+"\").instance(0)"));

        driver.findElement(By.xpath("//*text='Argentina']")).click();
        driver.findElement(By.id("com.androidsample.generalstore:id/btnLetsShop")).click();

        // 55.
        // The demonstration that the first option is not a good one
        System.out.println(driver.findElements(By.id("com.androidsample.generalstore:id/productName")).size());

        driver.findElement(MobileBy.AndroidUIAutomator("new UiScrollable(new UiSelector().resourceId()"
                + ".resourceId(\"com.androidsample.generalstore:id/rvProductList\")).scrollableIntoView("
                + "new UiSelector().textMatches(\"Jordan 6 Rings\").instance(0))"));
    }
}
```

Figure 7.12

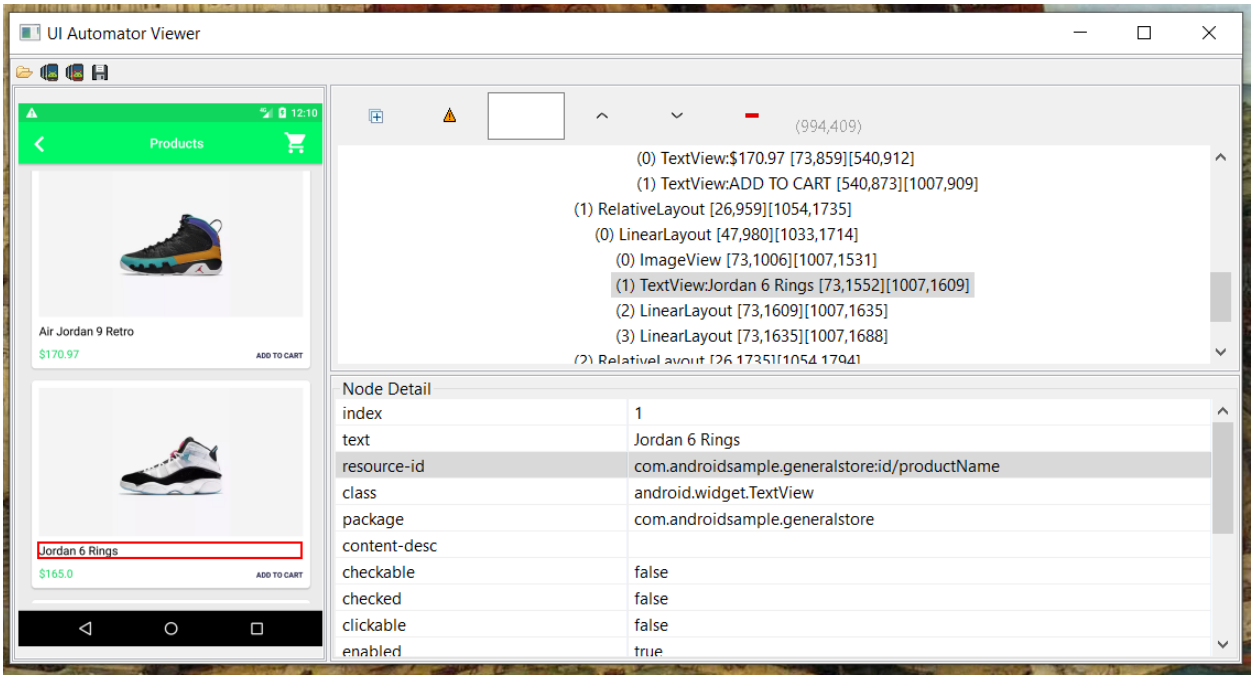


Figure 7.13

Test 2: Add Products

After achieving scrolling to the desired product, it remains to add it to the cart. In order to press the Add button, it is require identifying the corresponding Add button of the desired product. All the product have this button with the same text so it is very hard to identify the button corresponding with the desired product. A solution is to scroll through the entire product present on the screen and keep this value in a variable “count”. Then take each product one by one and get the text of each product in an auxiliary string. When the text of the product matches with the name of the desired product, the code click on the product with the corresponding index. The “break” syntax stops the algorithm from passing to all of the products. After the wanted produces are added, the next step is to press on the shopping cart icon on the right-up corner.

```
package package56;

import java.net.MalformedURLException;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import io.appium.java_client.MobileBy;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;

public class ecommerce_tc_2 extends base {
    public static void main(String[] args) throws MalformedURLException {

        // 56.
        AndroidDriver<AndroidElement> driver=Capabilities();
        driver.manage().timeouts().implicitlyWait(10,  TimeUnit.SECONDS);

        driver.findElement(By.id("com.androidsample.generalstore:id/nameField")).sendKeys("Hello");
        driver.hideKeyboard();
        driver.findElement(By.xpath("*//[@text='Female']")).click();
        driver.findElement(By.id("android:id/text1")).click();
        driver.findElementByAndroidUIAutomator("new UiScrollable(new UiSelector()).scrollIntoView(\"Argentina\"));");
        driver.findElement(By.xpath("//*text='Argentina']")).click();
        driver.findElement(By.id("com.androidsample.generalstore:id/btnLetsShop")).click();

        driver.findElement(MobileBy.AndroidUIAutomator("new UiScrollable(new UiSelector().resourceId()"
                + ".resourceId(\"com.androidsample.generalstore:id/rvProductList\")).scrollableIntoView("
                + "new UiSelector().textMatches(\"Jordan 6 Rings\").instance(0))"));

        // Scrolling through the total number of products present on the screen and keep it in the count variable
        int count= driver.findElements(By.id("com.androidsample.generalstore:id/productName")).size();

        for(int i=0;i<count;i++) {

            String text=driver.findElements(By.id("com.androidsample.generalstore:id/productName")).get(i).getText();
            if(text.equalsIgnoreCase("Jordan 6 Rings")) {
                driver.findElements(By.id("com.androidsample.generalstore:id/productAddCart")).get(i).click();
                break;
            }
        }
        driver.findElement(By.id("com.androidsample.generalstore:id/appbar_btn_cart")).click();
    }
}

```

Figure 7.14

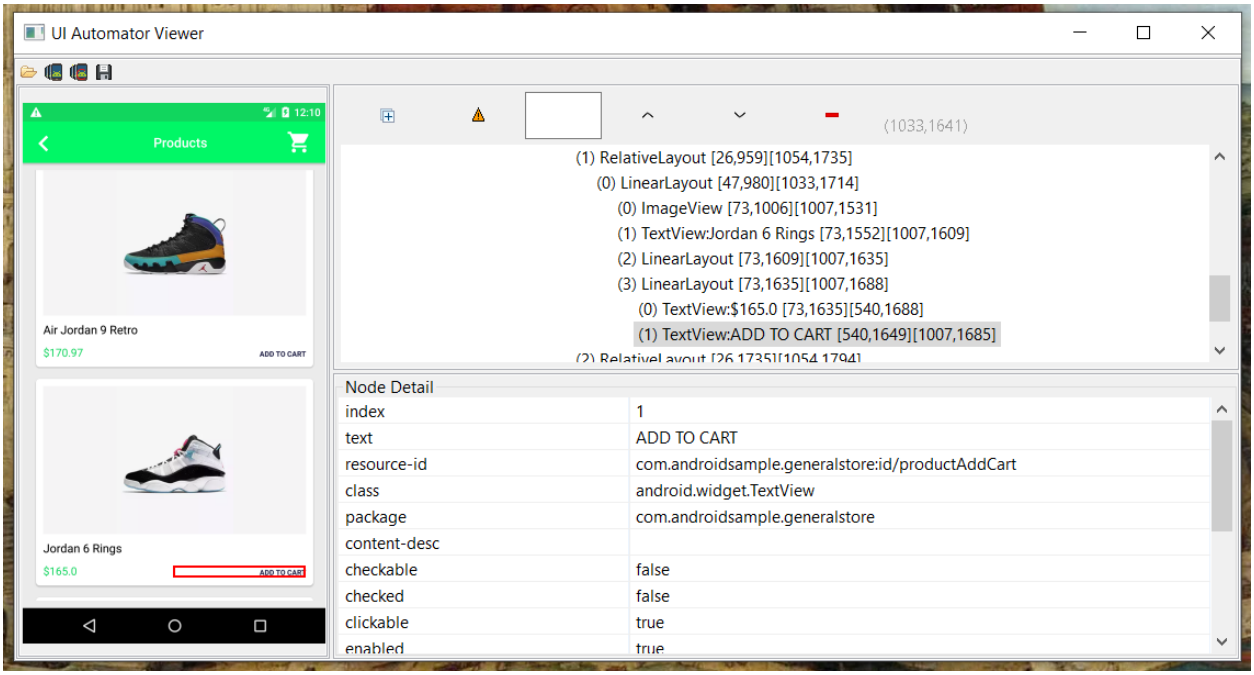


Figure 7.15

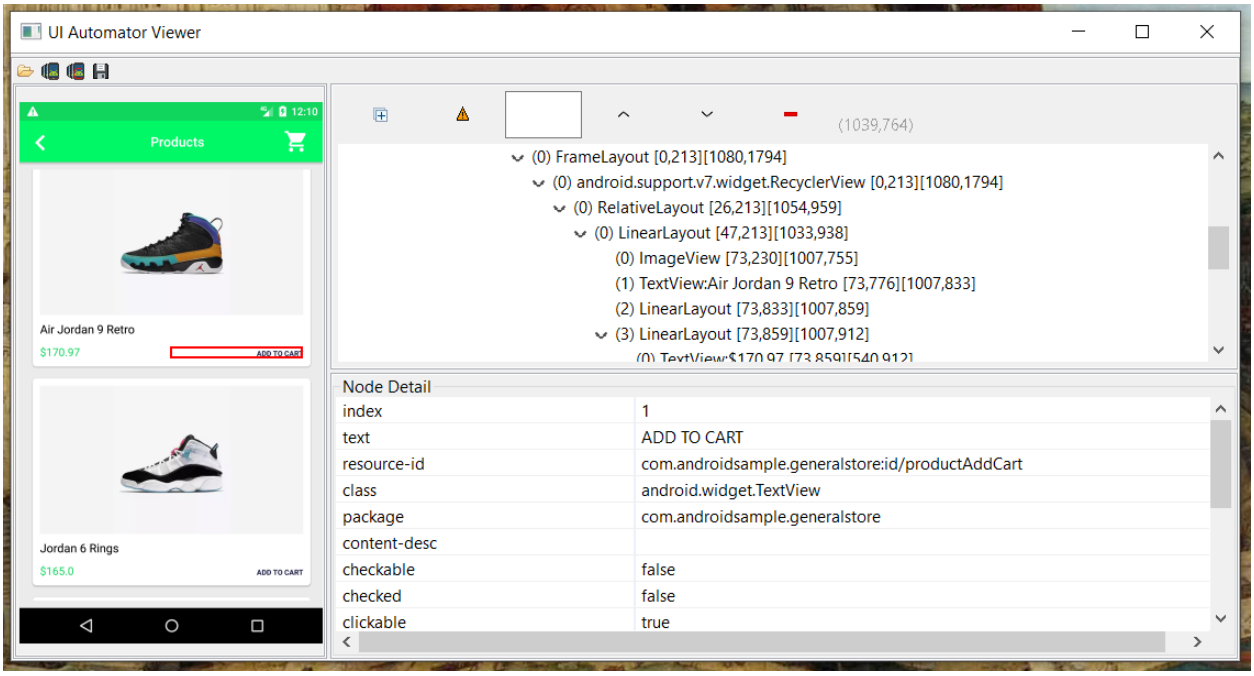


Figure 7.16

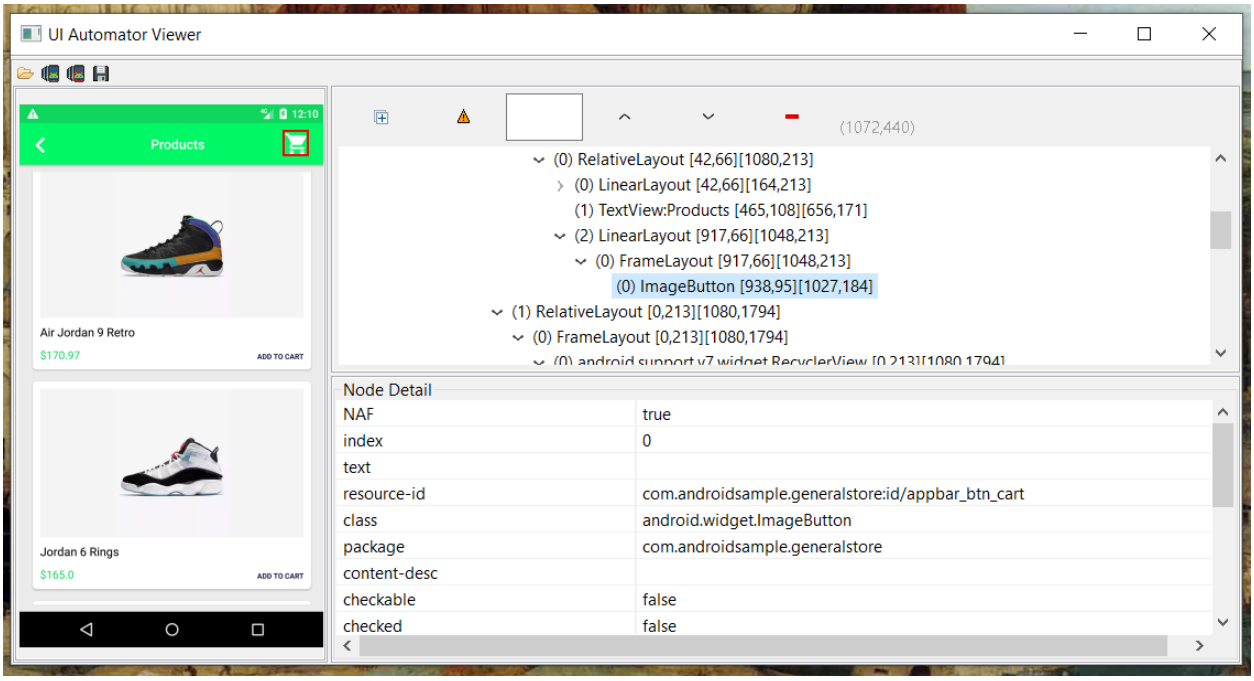


Figure 7.17

Test 3: Validate Product Name

In this part, let us propose checking if the price and name from the Product List is the same with the price and name at the Checkout page. Because it is desired to check if the text is the same, it is not possible to identify the elements by the text parameter. In order to do this, let us use a string variable in which the text from added product is kept.

An assertion allows testing the correctness of any assumptions that have been made in the program. Assertion is achieved using the **Assert class** in Java. While executing assertion, it is believed to be true. If it fails, the Java Virtual Machine throws an error named **AssertionError.** It is mainly used for testing purposes during development.

To compare the name of the product from the Checkout page with the name of the selected product, let us use the assertion. If the strings are the same, the assertion is true. If the strings are not matching, the assertion is false. Download the “junit-4.13.jar” and add it to the project. Import it in order to have access to the Assert class.

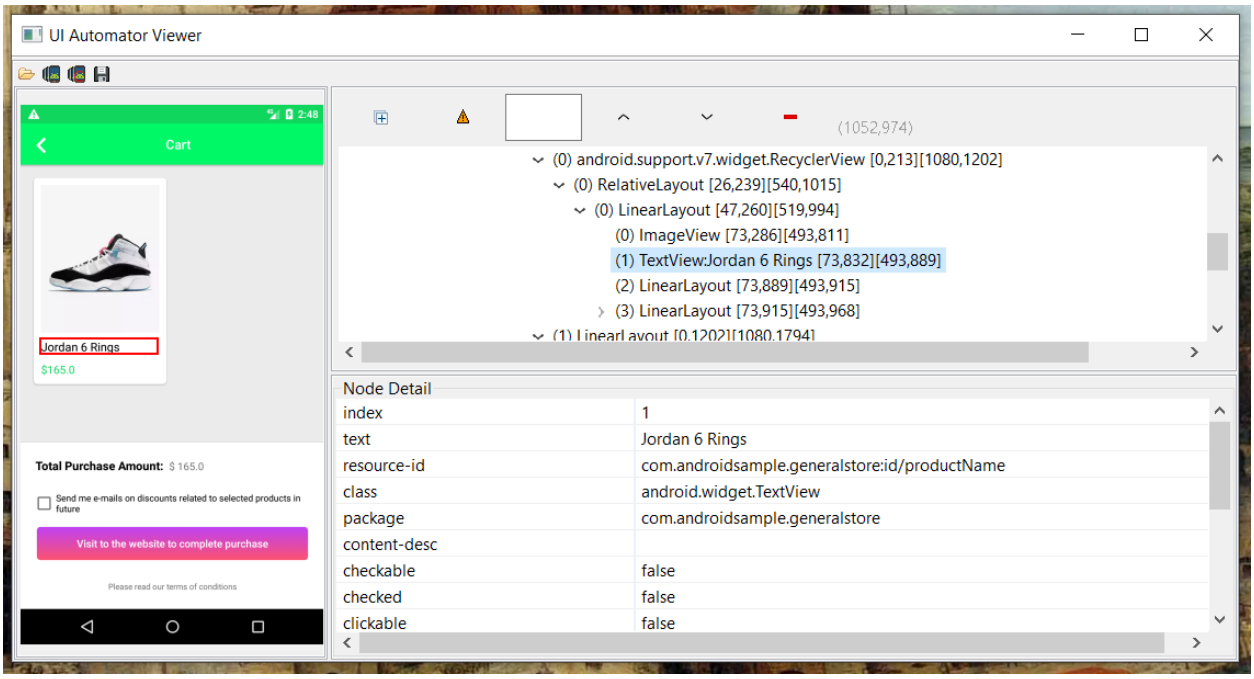


Figure 7.18

```java
package package59;

import java.net.MalformedURLException;
import java.util.concurrent.TimeUnit;
import org.junit.Assert;
import org.openqa.selenium.By;
import io.appium.java_client.MobileBy;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;

public class ecommerce_tc_3 extends base {

    public static void main(String[] args) throws MalformedURLException {

        AndroidDriver<AndroidElement> driver=Capabilities();
        driver.manage().timeouts().implicitlyWait(10,  TimeUnit.SECONDS);

        driver.findElement(By.id("com.androidsample.generalstore:id/nameField")).sendKeys("Hello");
        driver.hideKeyboard();
        driver.findElement(By.xpath("*//[@text='Female']")).click();
        driver.findElement(By.id("android:id/text1")).click();
        driver.findElementByAndroidUIAutomator("new UiScrollable(new UiSelector()).scrollIntoView(\"Argentina\"));");
        driver.findElement(By.xpath("//*text='Argentina']")).click();
        driver.findElement(By.id("com.androidsample.generalstore:id/btnLetsShop")).click();

        driver.findElement(MobileBy.AndroidUIAutomator("new UiScrollable(new UiSelector().resourceId()"
                + ".resourceId(\"com.androidsample.generalstore:id/rvProductList\")).scrollableIntoView("
                + "new UiSelector().textMatches(\"Jordan 6 Rings\").instance(0))"));

        // Scrolling through the total number of products present on the screen and keep it in the count variable
        int count= driver.findElements(By.id("com.androidsample.generalstore:id/productName")).size();

        for(int i=0;i<count;i++) {
            String text=driver.findElements(By.id("com.androidsample.generalstore:id/productName")).get(i).getText();

            if(text.equalsIgnoreCase("Jordan 6 Rings")) {

                driver.findElements(By.id("com.androidsample.generalstore:id/productAddCart")).get(i).click();
                break;
            }
        }

        driver.findElement(By.id("com.androidsample.generalstore:id/appbar_btn_cart")).click();
```

Figure 7.19

```
44
45          // 54.
46          String lastpageText=driver.findElement(By.id("com.androidsample.generalstore:id/productName")).getText();
47
48          Assert.assertEquals("Jordan 6 Rings",lastpageText);
49      }
50  }
51
```

Figure 7.20

Test 4: Sequential Adding of the Products

In order to keep the code clean and short enough, let us pass over the scrolling through the products and instead choose directly the first two products from the list. The two button can be identified using the text parameter with the value “ADD TO CART”. As on the screen of the emulator, there are two product, using the get method it is possible to have access to their indexes. Afterwards it is possible to click on them. As common sense, it is logical to put the index zero and then the index one, but this approach will fail. The reason is that the code work in sequential way. Thereby selecting the first product and clicking on it, generate the changes of the element from “ADD TO CART” in “ADDED TO CART”. Afterwards the code tries to access the second element with the text “ADD TO CART”. The problem is that there are not any second element on the screen with this text. On the screen is an element with “ADDED TO CART” and one element with “ADD TO CART”. The second element with ‘ADD TO CART” is now the first and only one. To pass over this problem, it is necessary to click twice on the first element with “ADD TO CART”.

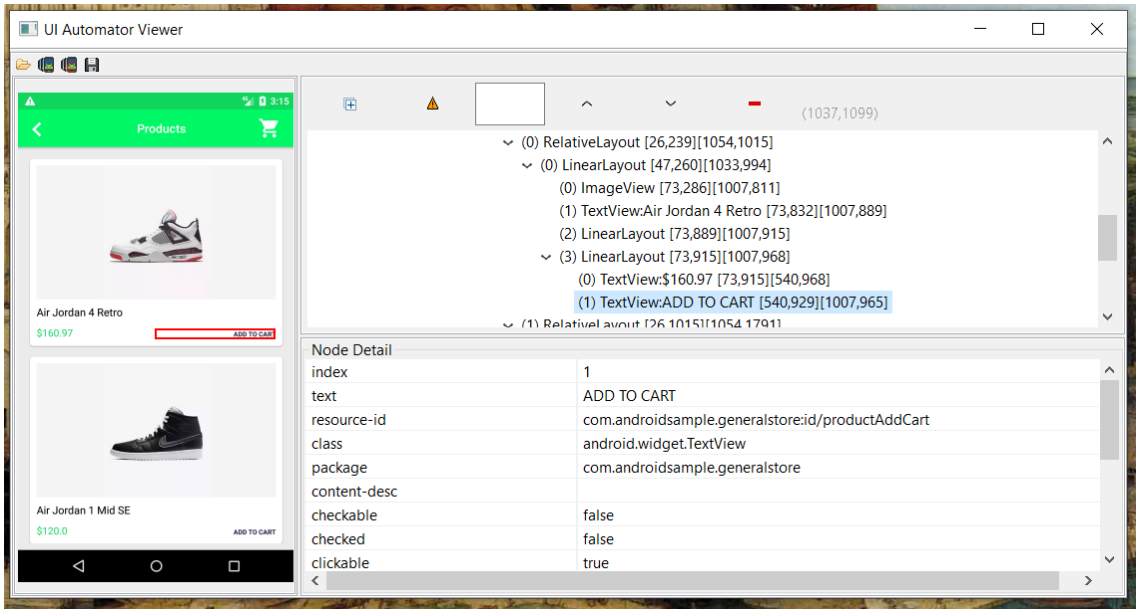


Figure 7.21

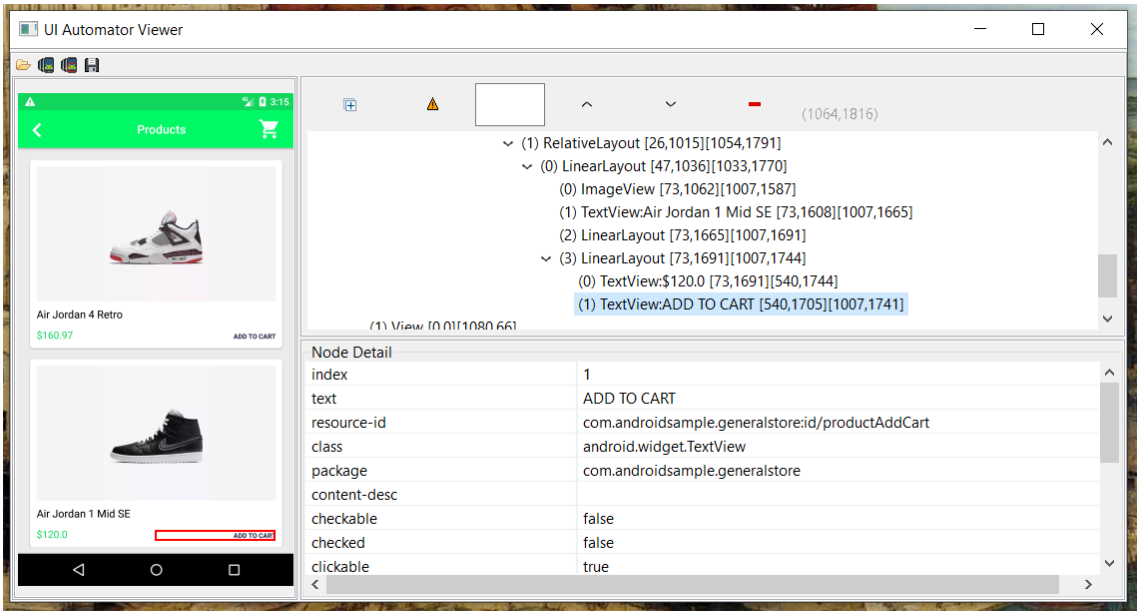


Figure 7.22

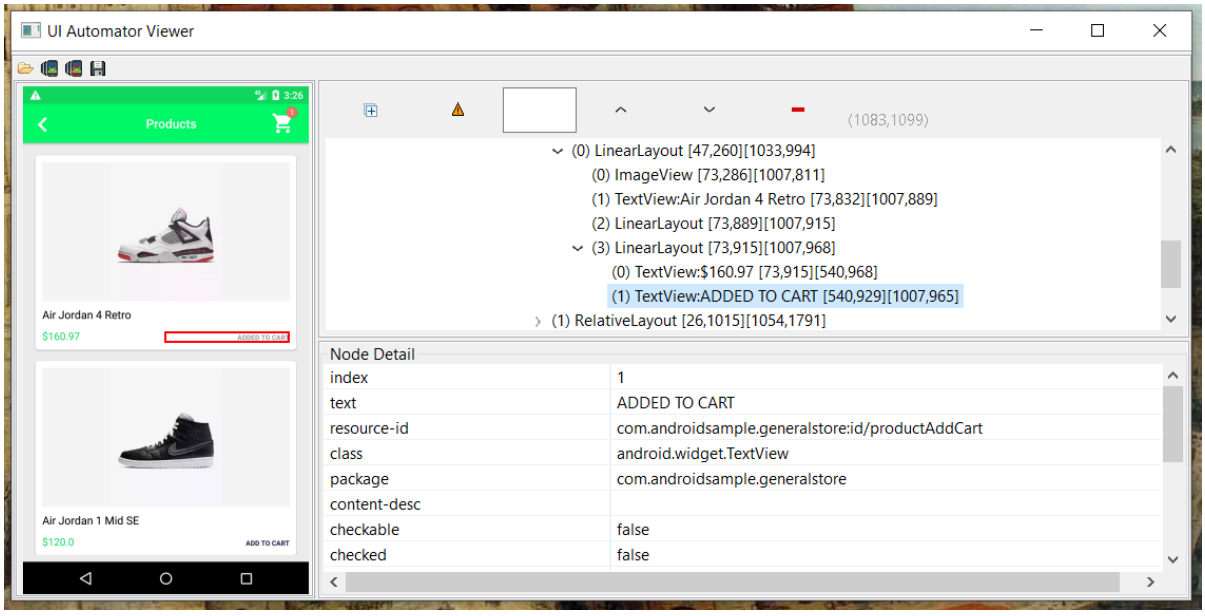


Figure 7.23

```java
package package59;

import java.net.MalformedURLException;
import java.util.concurrent.TimeUnit;

import org.junit.Assert;
import org.openqa.selenium.By;
import org.openqa.selenium.remote.RemoteWebElement;
import io.appium.java_client.MobileBy;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;

public class ecommerce_tc_4 extends base {

    public static void main(String[] args) throws MalformedURLException {

        AndroidDriver<AndroidElement> driver=Capabilities();
        driver.manage().timeouts().implicitlyWait(10,  TimeUnit.SECONDS);

        driver.findElement(By.id("com.androidsample.generalstore:id/nameField")).sendKeys("Hello");
        driver.hideKeyboard();
        driver.findElement(By.xpath("*//[@text='Female']")).click();
        driver.findElement(By.id("android:id/text1")).click();
        driver.findElementByAndroidUIAutomator("new UiScrollable(new UiSelector()).scrollIntoView(\"Argentina\"));");
        driver.findElement(By.xpath("//*text='Argentina']")).click();
        driver.findElement(By.id("com.androidsample.generalstore:id/btnLetsShop")).click();

        // 59.
        // Add the first two products from the list
        driver.findElements(By.xpath("//*[@text='ADD TO CART']")).get(0).click();

        // This line will show an errors because the second element do not exist
        //driver.findElements(By.xpath("//*[@text='ADD TO CART']")).get(1).click();

        // This line will not show an error because only one element exist
        driver.findElements(By.xpath("//*[@text='ADD TO CART']")).get(0).click();

        driver.findElement(By.id("com.androidsample.generalstore:id/appbar_btn_cart")).click();

    }
}

```

Figure 7.24

Test 4: Validate Final Product Price

Now, let us propose to check if the added prices for the selected produced, is equal with the value shown by the application. For doing this, it requires to take the two values and compare them with the value given by the app. As observed in the UI Automator Viewer, the two prices has the same id, so they ca be access very easily. The get method provides the access to each element separately. The price is in a text format so the method getText is used for extracting the price in a string format. The price contains the dollar sign that need to be remove before it is converted to the double data type. The substring method creates a separate string starting from the index provided. The dollar sign is on the index 0, so it is needed to start from index 1 in order to have only the digits. The next step is to convert them to the double data type, so the numbers can be added. The resulted sum provided by the app need to pass though the same process as well. In the end the two result are displayed on the console and the Assert class tests if the amount are equal. The test is stopped for four seconds because it take a little time to find and access the elements for the Appium server. In addition, the loading of a new page requires time. Using this break the server cannot mistake one page from the next one.

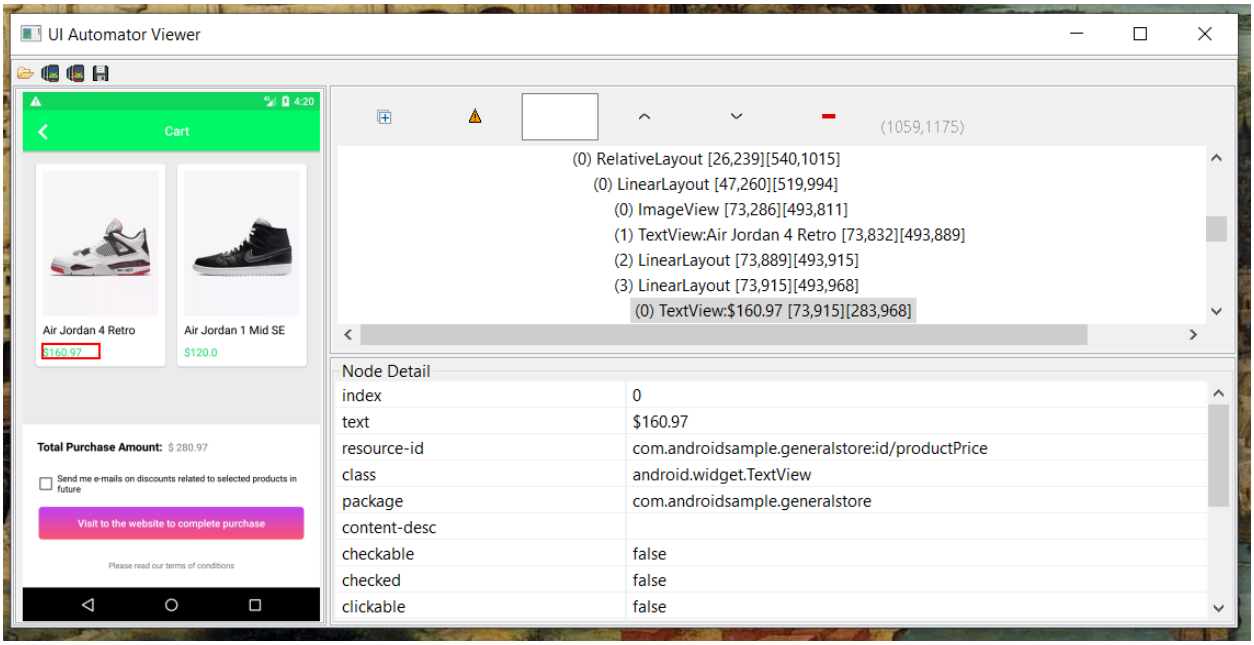


Figure 7.25

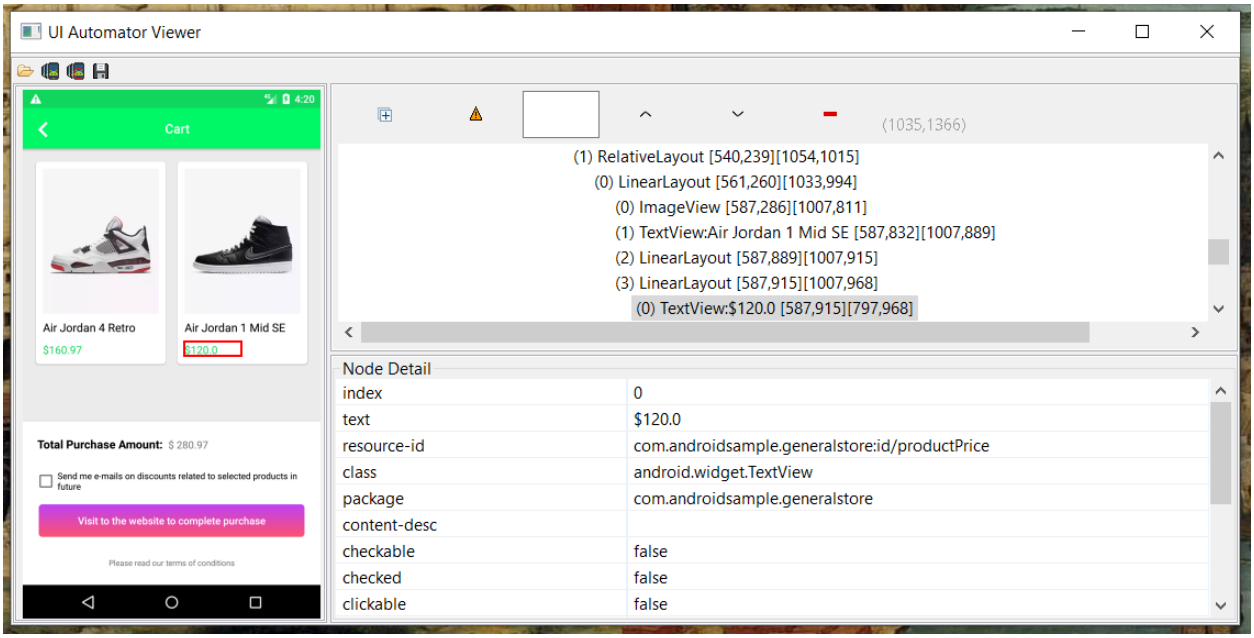


Figure 7.26

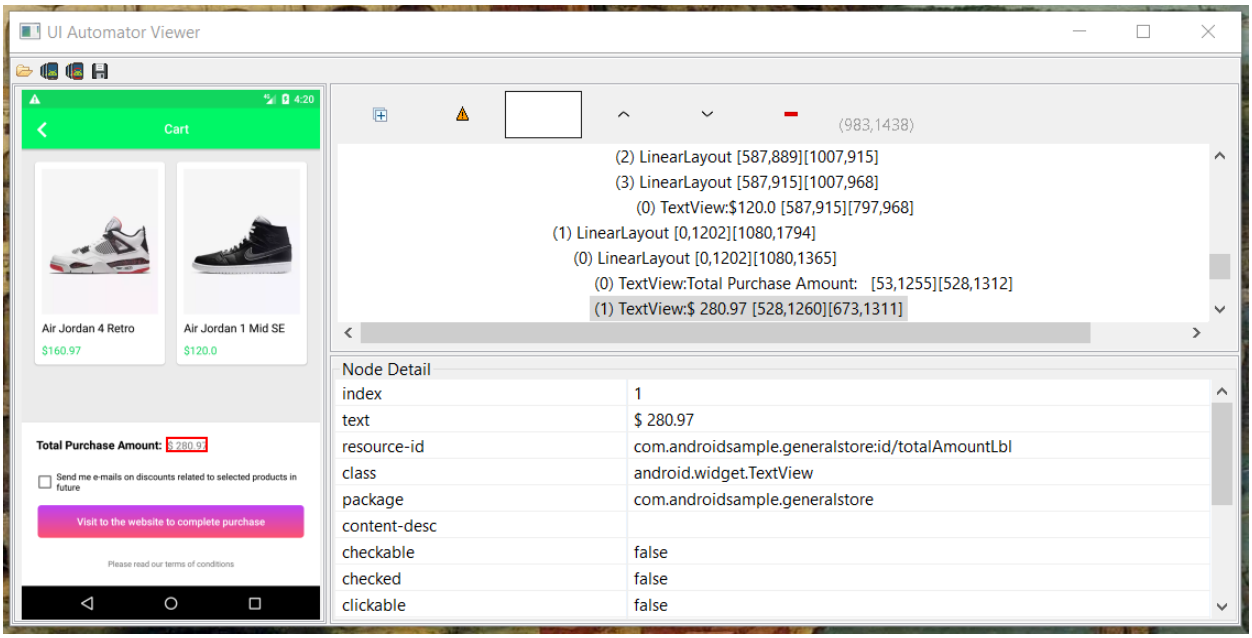


Figure 7.27

```java
package package60;

import java.net.MalformedURLException;
import java.util.concurrent.TimeUnit;

import org.junit.Assert;
import org.openqa.selenium.By;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;

public class ecommerce_tc_4 extends base {

    public static void main(String[] args) throws MalformedURLException, InterruptedException {

        AndroidDriver<AndroidElement> driver=Capabilities();
        driver.manage().timeouts().implicitlyWait(10,  TimeUnit.SECONDS);

        driver.findElement(By.id("com.androidsample.generalstore:id/nameField")).sendKeys("Hello");
        driver.hideKeyboard();
        driver.findElement(By.xpath("*//[@text='Female']")).click();
        driver.findElement(By.id("android:id/text1")).click();
        driver.findElementByAndroidUIAutomator("new UiScrollable(new UiSelector()).scrollIntoView(\"Argentina\"));");
        driver.findElement(By.xpath("//*text='Argentina']")).click();
        driver.findElement(By.id("com.androidsample.generalstore:id/btnLetsShop")).click();

        // 59.
        driver.findElements(By.xpath("//*[@text='ADD TO CART']")).get(0).click();
        driver.findElements(By.xpath("//*[@text='ADD TO CART']")).get(0).click();
        driver.findElement(By.id("com.androidsample.generalstore:id/appbar_btn_cart")).click();

        // 60.
        // Wait 4 seconds so that the test does not confuse the pages
        Thread.sleep(4000);

        String amount1=driver.findElements(By.id("com.android.sample.generalstore:id/productPrice")).get(0).getText();
        // In the variable amount1 is the text: $120.0
        amount1=amount1.substring(1);
        double amount1value=Double.parseDouble(amount1);

        String amount2=driver.findElements(By.id("com.android.sample.generalstore:id/productPrice")).get(1).getText();
        // In the variable amount1 is the text: $160.97
        amount2=amount2.substring(1);
        double amount2value=Double.parseDouble(amount2);
```

Figure 7.28

```
44
45          String total=driver.findElement(By.id("com.android.sample.generalstore:id/totalAmountLbl")).getText();
46          // In the variable amount2 is the text: $280.97
47          total=total.substring(1);
48          double totalValue=Double.parseDouble(total);
49
50          double sumOfProducts=amount1value+amount2value;
51
52          Assert.assertEquals(sumOfProducts, totalValue);
53          System.out.println("Sum of the product:"+sumOfProducts);
54          System.out.println("Total value of the products:"+totalValue);
55
56      }
57  }
58
```

Figure 7.29

Test 4: Validate Final Product Price Optimized

As observed in the previous part, some part of the code is repeating. As the Appium code in written in Java. One of the advantages of its Object-oriented programming it is the possibility of putting reuse code inside of classes or methods that can be called. Let us make a function that deals with extracting the value from the string of the price.

```
1  package package61;
2
3  import java.net.MalformedURLException;
4  import java.util.concurrent.TimeUnit;
5
6  import org.junit.Assert;
7  import org.openqa.selenium.By;
8  import io.appium.java_client.android.AndroidDriver;
9  import io.appium.java_client.android.AndroidElement;
10
11 public class ecommerce_tc_4 extends base {
12
13     public static void main(String[] args) throws MalformedURLException, InterruptedException {
14
15         AndroidDriver<AndroidElement> driver=Capabilities();
16         driver.manage().timeouts().implicitlyWait(10,  TimeUnit.SECONDS);
17
18         driver.findElement(By.id("com.androidsample.generalstore:id/nameField")).sendKeys("Hello");
19         driver.hideKeyboard();
20         driver.findElement(By.xpath("*//[@text='Female']")).click();
21         driver.findElement(By.id("android:id/text1")).click();
22         driver.findElementByAndroidUIAutomator("new UiScrollable(new UiSelector()).scrollIntoView(\"Argentina\"));");
23         driver.findElement(By.xpath("//*text='Argentina']")).click();
24         driver.findElement(By.id("com.androidsample.generalstore:id/btnLetsShop")).click();
25
26         // 59.
27         driver.findElements(By.xpath("//*[@text='ADD TO CART']")).get(0).click();
28         driver.findElements(By.xpath("//*[@text='ADD TO CART']")).get(0).click();
29         driver.findElement(By.id("com.androidsample.generalstore:id/appbar_btn_cart")).click();
30
31         // 61.
32         // Wait 4 seconds so that the test does not confuse the pages
33         Thread.sleep(4000);
34
35         int count=driver.findElements(By.id("com.android.sample.generalstore:id/productPrice")).size();
36         double sum=0;
37         for(int i=0;i<count;i++) {
38
39             String amount1=driver.findElements(By.id("com.android.sample.generalstore:id/productPrice")).get(i).getText();
40             double amount=getAmount(amount1);
41             sum=sum+amount;
42         }
43
```

Figure 7.30

```
43
44          String total=driver.findElement(By.id("com.android.sample.generalstore:id/totalAmountLbl")).getText();
45          total=total.substring(1);
46          double totalValue=Double.parseDouble(total);
47
48          Assert.assertEquals(sum, totalValue);
49          System.out.println("Sum of the product:"+sum);
50          System.out.println("Total value of the products:"+totalValue);
51
52      }
53      public static double getAmount(String value) {
54          value=value.substring(1);
55          double amountvalue=Double.parseDouble(value);
56          return amountvalue;
57      }
58  }
59
```

Figure 7.31

Test 5: Validate Mobile Gesture

In the Checkout page, there is a check box for email advertisement. Let us check it. There is a text “Please read our terms of conditions”. After it is pressed for a longer period, it opens a pop-up window with a message and a Close button. Click on the Close button. In the end, click on the button with the text “Visit to the website to complete purchase”. Identify the check box using the class name and the terms of condition link with the Xpath. Use the tap gesture to select the check box and the long press gesture to the terms of condition. To identify the buttons use the id.

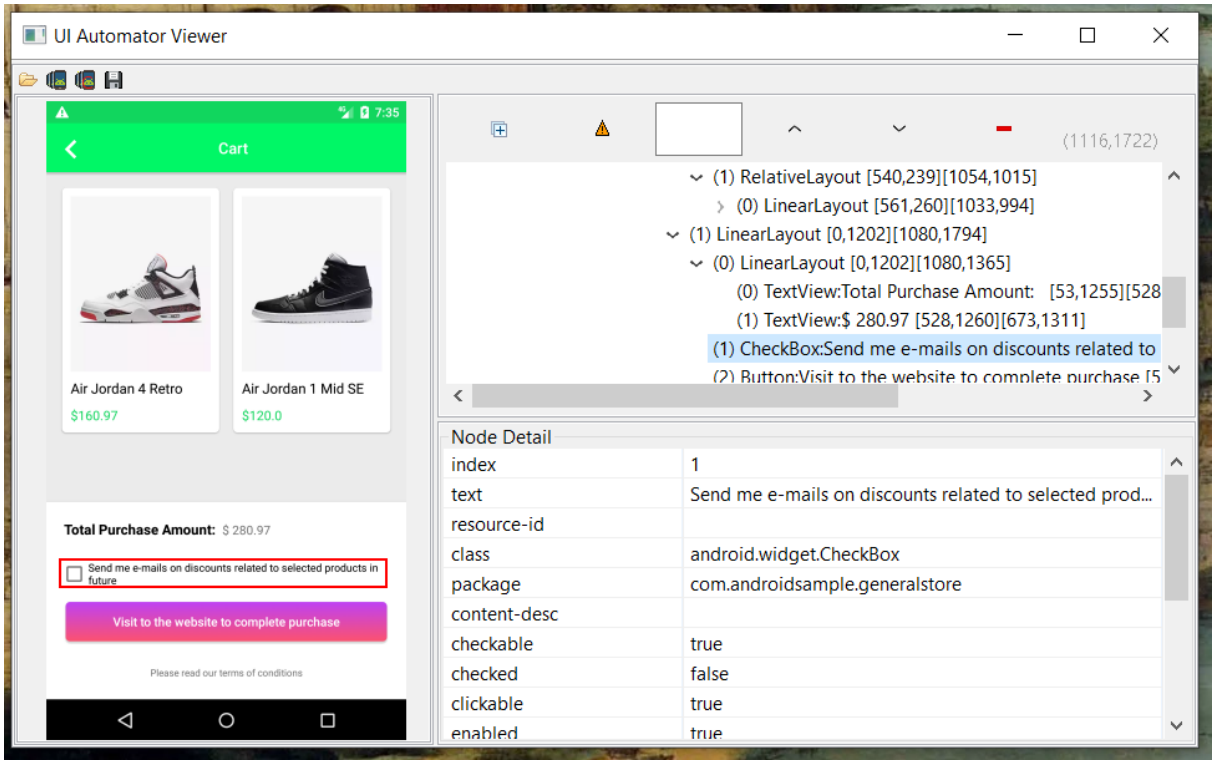


Figure 7.32

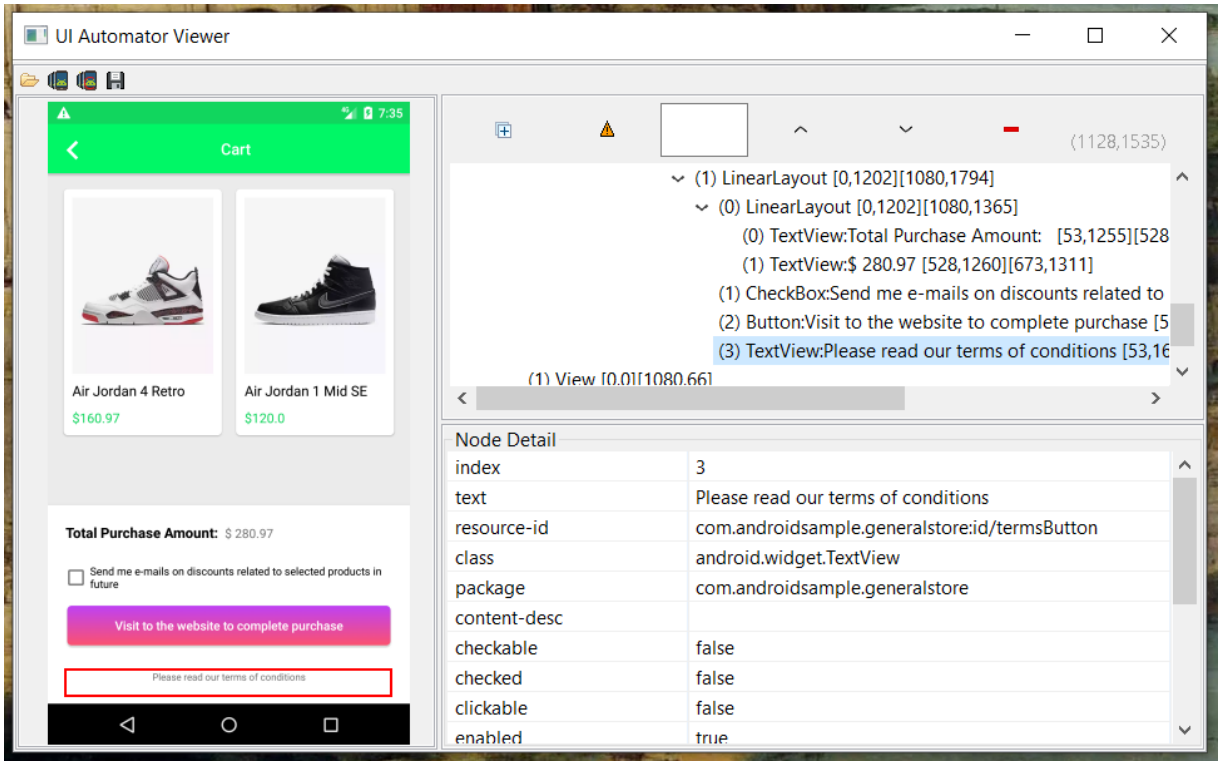


Figure 7.33

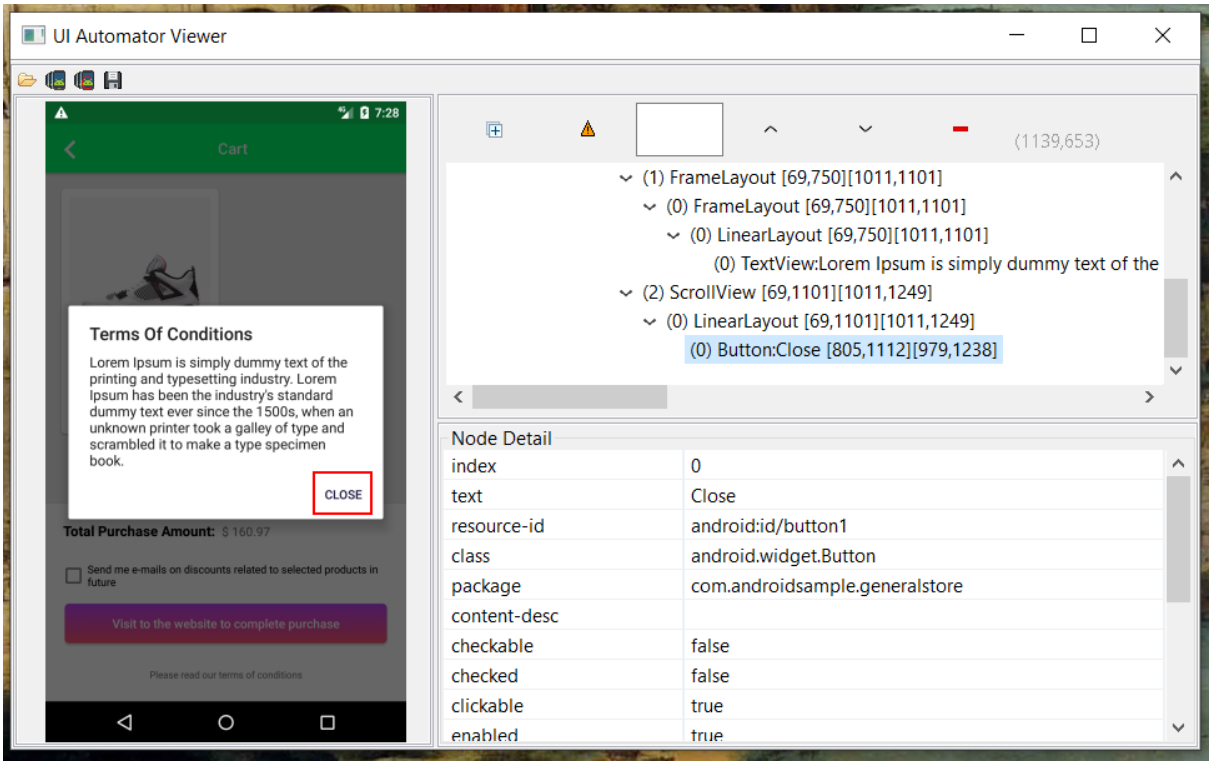


Figure 7.34

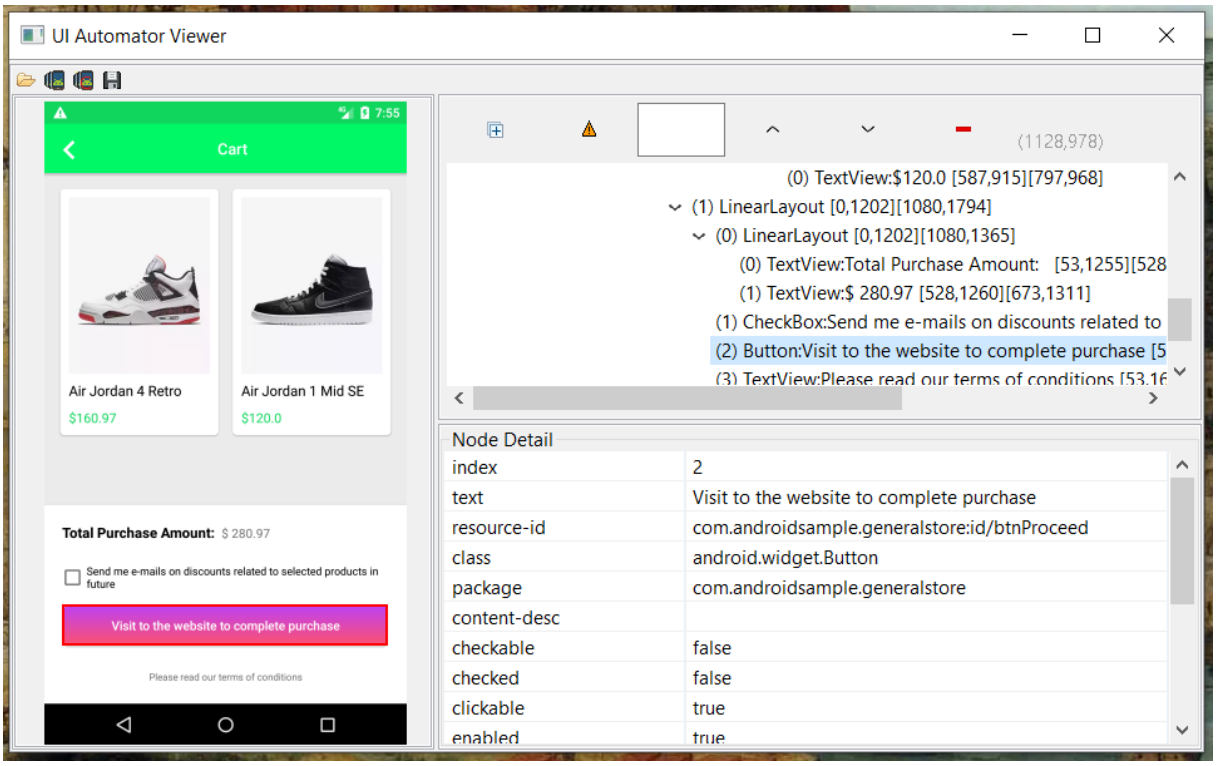


Figure 7.35

```java
package package63;

import java.net.MalformedURLException;
import java.util.concurrent.TimeUnit;

import org.junit.Assert;
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import io.appium.java_client.TouchAction;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;
import static io.appium.java_client.touch.TapOptions.tapOptions;
import static io.appium.java_client.touch.LongPressOptions.longPressOptions;
import static io.appium.java_client.touch.offset.ElementOption.element;
import static java.time.Duration.ofSeconds;

public class ecommerce_tc_5 extends base {

    public static void main(String[] args) throws MalformedURLException, InterruptedException {

        AndroidDriver<AndroidElement> driver=Capabilities();
        driver.manage().timeouts().implicitlyWait(10,  TimeUnit.SECONDS);

        driver.findElement(By.id("com.androidsample.generalstore:id/nameField")).sendKeys("Hello");
        driver.hideKeyboard();
        driver.findElement(By.xpath("*//[@text='Female']")).click();
        driver.findElement(By.id("android:id/text1")).click();
        driver.findElementByAndroidUIAutomator("new UiScrollable(new UiSelector()).scrollIntoView(\"Argentina\"));");
        driver.findElement(By.xpath("//*text='Argentina']")).click();
        driver.findElement(By.id("com.androidsample.generalstore:id/btnLetsShop")).click();

        // 59.
        driver.findElements(By.xpath("//*[@text='ADD TO CART']")).get(0).click();
        driver.findElements(By.xpath("//*[@text='ADD TO CART']")).get(0).click();
        driver.findElement(By.id("com.androidsample.generalstore:id/appbar_btn_cart")).click();

        // 61.
        // Wait 4 seconds so that the test does not confuse the pages
        Thread.sleep(4000);

        int count=driver.findElements(By.id("com.android.sample.generalstore:id/productPrice")).size();
        double sum=0;
        for(int i=0;i<count;i++) {
```

Figure 7.36

```
44
45              String amount1=driver.findElements(By.id("com.android.sample.generalstore:id/productPrice")).get(i).getText();
46              double amount=getAmount(amount1);
47              sum=sum+amount;
48          }
49
50          String total=driver.findElement(By.id("com.android.sample.generalstore:id/totalAmountLbl")).getText();
51          total=total.substring(1);
52          double totalValue=Double.parseDouble(total);
53
54          Assert.assertEquals(sum, totalValue);
55          System.out.println("Sum of the product:"+sum);
56          System.out.println("Total value of the products:"+totalValue);
57
58          // 63.
59          TouchAction t=new TouchAction(driver);
60          WebElement checkbox=driver.findElement(By.className("android.widget.CheckBox"));
61          t.tap(tapOptions().withElement(element(checkbox))).perform();
62
63          WebElement tc=driver.findElement(By.xpath("//*[@text='Please read our terms of conditions']"));
64          t.longPress(longPressOptions().withElement(element(tc)).withDuration(ofSeconds(2))).release().perform();
65
66          driver.findElement(By.id("android:id/button1")).click();
67          driver.findElement(By.id("com.androidsample.generalstore:id/btnProceed")).click();
68      }
69      public static double getAmount(String value) {
70          value=value.substring(1);
71          double amountvalue=Double.parseDouble(value);
72          return amountvalue;
73      }
74  }
75
```

Figure 7.37

Chapter 8: Hybrid Automation

Test 5: Navigate to Web View

On the e-commerce app, the button “Visit to the website to complete purchase” opens the Chrome bowser. Application which uses both native and web parts are called hybrid apps. One problem that occurs with these types of apps is the necessity to switch the testing automation from native to web testing syntaxes. For more information about this, please search on Google “appium hybrid app” and enter on their official website. Alternatively, access this link: “http://appium.io/docs/en/writing-running-appium/web/hybrid/”. On this website, this example is presented.

For understanding it, let us explain it in our code. Once the “Visit to the website to complete purchase” is pressed, the Chrome browser is opened. In Java language, the Set interface is an unordered collection of objects in which duplicate values cannot be store. This collection has in our case two possible components, one is the message “NATIVE_APP” and the second possible on is the “WEBVIEW”. In this collection, the Appium test server record if it is in the native or web environment. For example if the code runs, the result is “NATIVE_APP”. The reason is the fact that the app is moving much slower than the test code. Thereby the code press on the button in order to enter the web browser and right after this it registers the type of environment in which it is. However, the web browser has just started and it is not fully loaded. Because of this, it is necessary to stop the code from running for some amount of time until the web browser has fully started. Removing the commented line of code, the result will be “NATIVE_APP” and on the next row “WEBVIEW_com.androidsample.generalstore”. In this way, the code succeeds to record both environments. Please look in Figure 8.2.

It is require making the change from the native to web environment using the context method with the name of the desired environment as a parameter. The remaining thing to do is to insert a text in the search textbox. In most search engines nowadays, this textbox is named simply “q”. The entire code is provided in Figure 8.5 and 8.6.

Examples

Java | Python | Javascript | Ruby | PHP

```
// assuming we have a set of capabilities
driver = new AppiumDriver(new URL("http://127.0.0.1:4723/wd/hub"), capabilities);

Set<String> contextNames = driver.getContextHandles();
for (String contextName : contextNames) {
    System.out.println(contextName); //prints out something like NATIVE_APP \n WEBV
IEW_1
}
driver.context(contextNames.toArray()[1]); // set context to WEBVIEW_1

//do some web testing
String myText = driver.findElement(By.cssSelector(".green_button")).click();

driver.context("NATIVE_APP");

// do more native testing if we want

driver.quit();
```

Figure 8.1

```
driver.findElement(By.id("com.androidsample.generalstore:id/btnProceed")).click();

// 64.
//Thread.sleep(7000);
Set<String> contexts=driver.getContextHandles();
for(String contextName :contexts) {
    System.out.println(contextName);
}
```

Figure 8.2

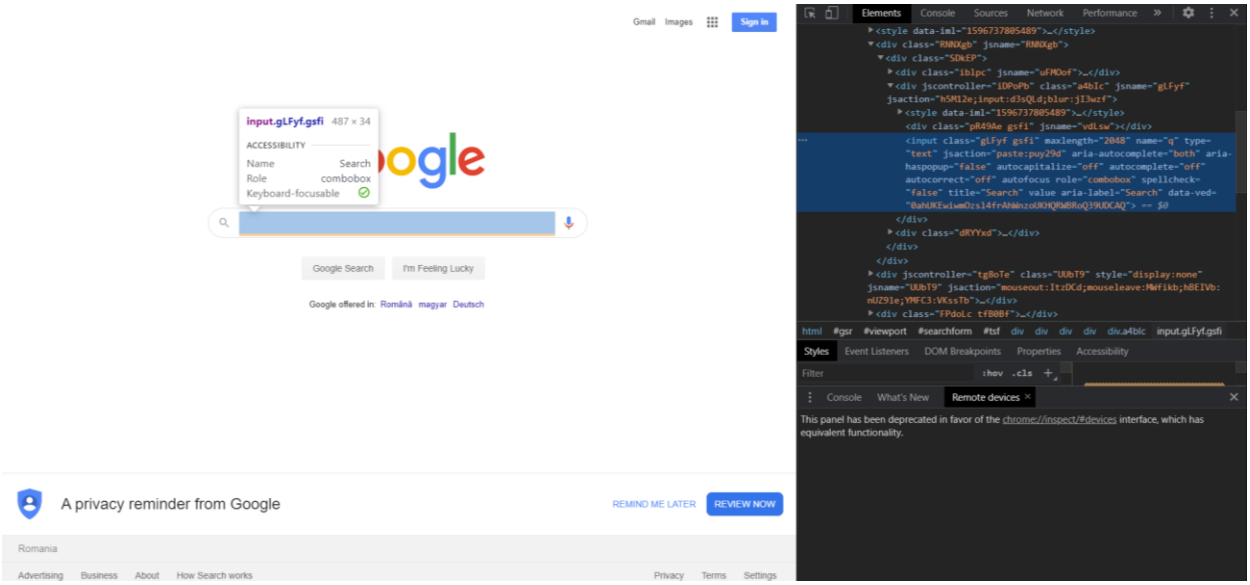


Figure 8.3

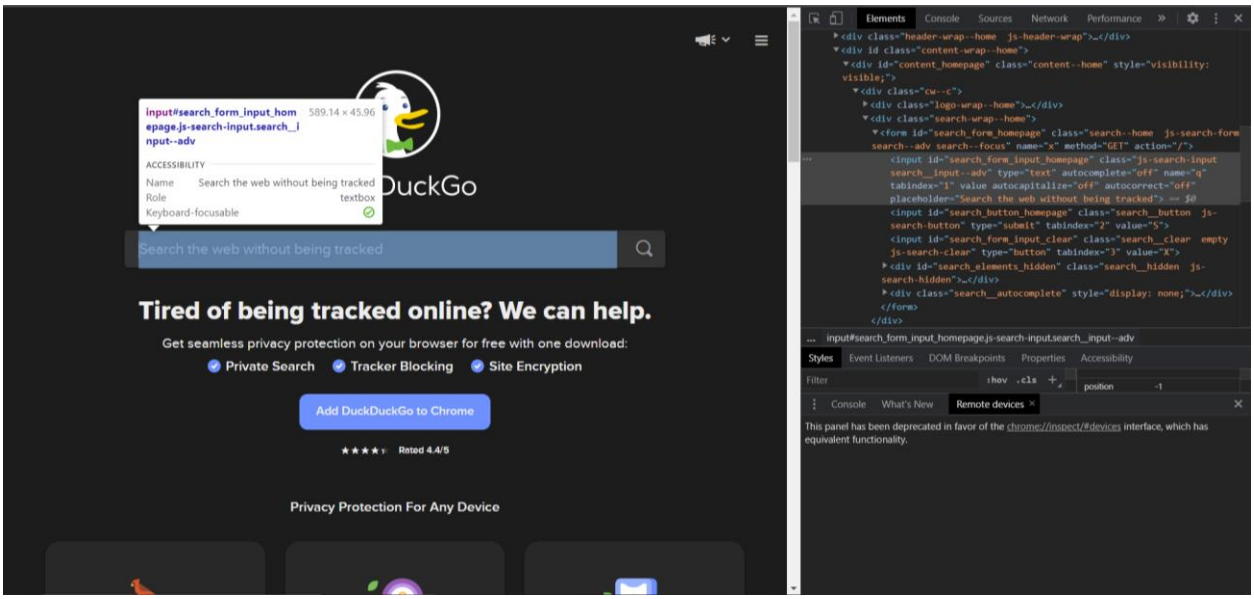


Figure 8.4

```java
package package64;

import java.net.MalformedURLException;
import java.util.Set;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import io.appium.java_client.TouchAction;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;
import static io.appium.java_client.touch.TapOptions.tapOptions;
import static io.appium.java_client.touch.offset.ElementOption.element;

public class ecommerce_tc_5 extends base {

    public static void main(String[] args) throws MalformedURLException, InterruptedException {

        AndroidDriver<AndroidElement> driver=Capabilities();
        driver.manage().timeouts().implicitlyWait(10,  TimeUnit.SECONDS);

        driver.findElement(By.id("com.androidsample.generalstore:id/nameField")).sendKeys("Hello");
        driver.hideKeyboard();
        driver.findElement(By.xpath("*//[@text='Female']")).click();
        driver.findElement(By.id("android:id/text1")).click();
        driver.findElementByAndroidUIAutomator("new UiScrollable(new UiSelector()).scrollIntoView(\"Argentina\"));");
        driver.findElement(By.xpath("//*text='Argentina']")).click();
        driver.findElement(By.id("com.androidsample.generalstore:id/btnLetsShop")).click();

        // 59.
        driver.findElements(By.xpath("//*[@text='ADD TO CART']")).get(0).click();
        driver.findElements(By.xpath("//*[@text='ADD TO CART']")).get(0).click();
        driver.findElement(By.id("com.androidsample.generalstore:id/appbar_btn_cart")).click();

        // 61.
        // Wait 4 seconds so that the test does not confuse the pages
        Thread.sleep(4000);

        WebElement checkbox=driver.findElement(By.className("android.widget.CheckBox"));
        TouchAction t=new TouchAction(driver);
        t.tap(tapOptions().withElement(element(checkbox))).perform();

        driver.findElement(By.id("com.androidsample.generalstore:id/btnProceed")).click();

```

Figure 8.5

```
        // 64.
        //Thread.sleep(7000);
        Set<String> contexts=driver.getContextHandles();
        for(String contextName :contexts) {
            System.out.println(contextName);
        }
        driver.context("WEBVIEW_com.androidsample.generalstore");

        //65.
        driver.findElement(By.name("q")).sendKeys("hello");

    }
}
```

Figure 8.6

Test 6: Operation in Web View

In the last test, it is require to test if the user can introduce a text in the search engine and then if he succeeds to come back to the native app. From the previous part, the process of writing in the search text box in already know. In addition, in modern search engine when the user introduce a text, multiple other results are displayed. Therefore, it is necessary to press enter on the search engine page in order to see the results page. This can be obtain using the “Keys.ENTER” syntax. To return to the native app, the back button from the bottom of the emulator needs to be pressed. A new class called keyEvent can perform this action. The object “Android.BACK” represents the back button in Android environment. In the end, using the “context” method with the text ‘NATIVE_APP” as a parameter, the Appium server changes from web to native mode.

```
package package65;

import java.net.MalformedURLException;
import java.util.Set;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebElement;
import io.appium.java_client.TouchAction;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.AndroidElement;
import io.appium.java_client.android.nativekey.AndroidKey;
import io.appium.java_client.android.nativekey.KeyEvent;

import static io.appium.java_client.touch.TapOptions.tapOptions;
import static io.appium.java_client.touch.offset.ElementOption.element;

public class ecommerce_tc_6 extends base {

    public static void main(String[] args) throws MalformedURLException, InterruptedException {

        AndroidDriver<AndroidElement> driver=Capabilities();
        driver.manage().timeouts().implicitlyWait(10,  TimeUnit.SECONDS);

        driver.findElement(By.id("com.androidsample.generalstore:id/nameField")).sendKeys("Hello");
        driver.hideKeyboard();
        driver.findElement(By.xpath("*//[@text='Female']")).click();
        driver.findElement(By.id("android:id/text1")).click();
        driver.findElementByAndroidUIAutomator("new UiScrollable(new UiSelector()).scrollIntoView(\"Argentina\"));");
        driver.findElement(By.xpath("//*text='Argentina']")).click();
        driver.findElement(By.id("com.androidsample.generalstore:id/btnLetsShop")).click();

        // 59.
        driver.findElements(By.xpath("//*[@text='ADD TO CART']")).get(0).click();
        driver.findElements(By.xpath("//*[@text='ADD TO CART']")).get(0).click();
        driver.findElement(By.id("com.androidsample.generalstore:id/appbar_btn_cart")).click();

        // 61.
        // Wait 4 seconds so that the test does not confuse the pages
        Thread.sleep(4000);

        WebElement checkbox=driver.findElement(By.className("android.widget.CheckBox"));
        TouchAction t=new TouchAction(driver);
```

Figure 8.7

```
        t.tap(tapOptions().withElement(element(checkbox))).perform();

        driver.findElement(By.id("com.androidsample.generalstore:id/btnProceed")).click();

        // 64.
        //Thread.sleep(7000);
        Set<String> contexts=driver.getContextHandles();
        for(String contextName :contexts) {
            System.out.println(contextName);
        }
        driver.context("WEBVIEW_com.androidsample.generalstore");

        // 65.
        driver.findElement(By.name("q")).sendKeys("hello");
        driver.findElement(By.name("q")).sendKeys(Keys.ENTER);
        driver.pressKey(new KeyEvent(AndroidKey.BACK));
        driver.context("NATIVE_APP");

    }

}

```

Figure 8.8

Note: Something the version of Android Chrome and the version, which the Appium server can manage, are compatible with the version of the emulator. For example, the emulator has the version 69, but the lowest version supported by Appium is 71. This thing happens because, at the installation, the Appium server comes with the latest version on Android Chrome. To overcome this situation please verify the compatible Chrome version supported by the corresponding version of Chrome driver on this website: “https://chrome.chromium.org/downloads”. Download it according to your computer operating system from this site: ”https://chromedriver.storage.googleapis.com/index.html”. Unzip the folder and replace the current Chrome driver with the downloaded one on the next path:

”C:\Users\filda\AppData\Roaming\npm\node_modules\appium\node_modules\appium-chromedriver\chromedriver\win”.

Chapter 9: TestNG Framework

Install TestNG

First step in order to install TestNG is to search on Google “TestNG” and access their official website. Alternatively, use the following link: “https://testng.org/doc/”. Click on the “Eclipse” from the upper part of the page. Then click on the text “install the plug-in”. Scroll down until The “Eclipse plug-in” section. Follow the steps after the “Install from update site”. Open Eclipse and select Help and then Install New Software. From the website, copy the link after “Update site for release” and copy in the Eclipse window. The “Pending” message will appear for few moments. If this thing does not happen, press on the Add button. Check the textbox of the TestNG and press Next. Accept the terms and click Finish. It is possible that the Eclipse shows a warning because of the downloading from unknown source, but click on Install anyway. In the end, restart the Eclipse. Add TestNG library from the Java Built Path in the Properties of the project. Select Add Library then TestNG. In the end, press Next and Finish.

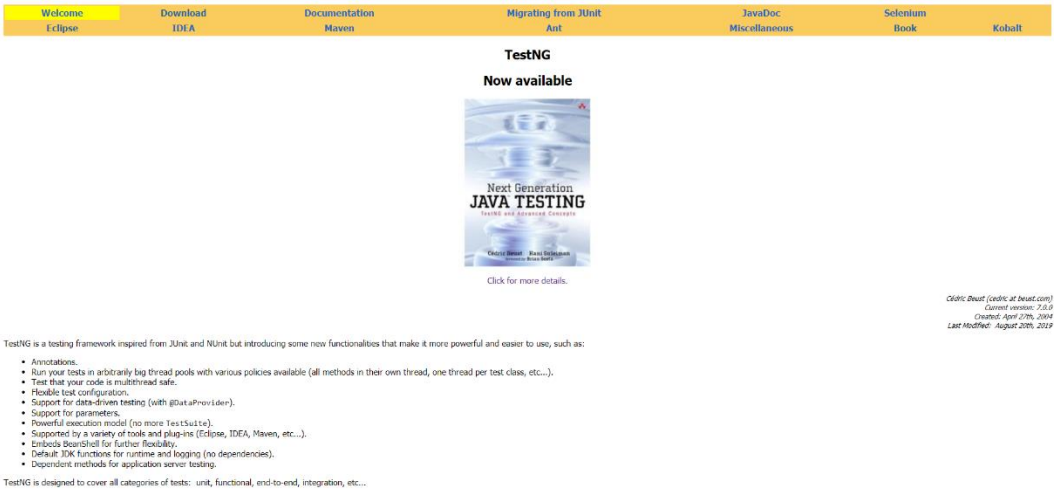
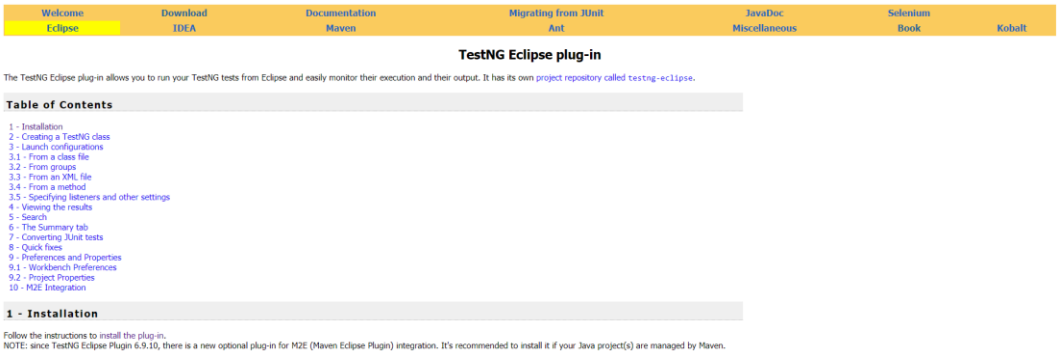


Figure 9.1



Welcome | Download | Documentation | Migrating from JUnit | JavaDoc | Selenium
Eclipse | IDEA | Maven | Ant | Miscellaneous | Book | Kobalt

TestNG Eclipse plug-in

The TestNG Eclipse plug-in allows you to run your TestNG tests from Eclipse and easily monitor their execution and their output. It has its own project repository called `testng-eclipse`.

Table of Contents

1 - Installation

Follow the instructions to install the plug-in.
NOTE: since TestNG Eclipse Plugin 6.9.10, there is a new optional plug-in for M2E (Maven Eclipse Plugin) integration. It's recommended to install it if your Java project(s) are managed by Maven.

Figure 9.2

Eclipse plug-in

Java 1.7+ is required for running the TestNG for Eclipse plugin.

Eclipse 4.2 and above is required. Eclipse 3.x is NOT supported any more, please update your Eclipse to 4.2 or above.

You can use either the Eclipse Marketplace or the update site:

Install via Eclipse Marketplace

Go to the TestNG page on the Eclipse Market Place and drag the icon called "Install" onto your workspace.

Install from update site

- Select *Help / Install New Software...*
- Enter the update site URL in "Work with:" field:
  - Update site for release: https://dl.bintray.com/testng-team/testng-eclipse-release/.
- Make sure the check box next to URL is checked and click *Next*.
- Eclipse will then guide you through the process.

You can also install older versions of the plug-ins here. Note that the URL's on this page are update sites as well, **not** direct download links.

Figure 9.3

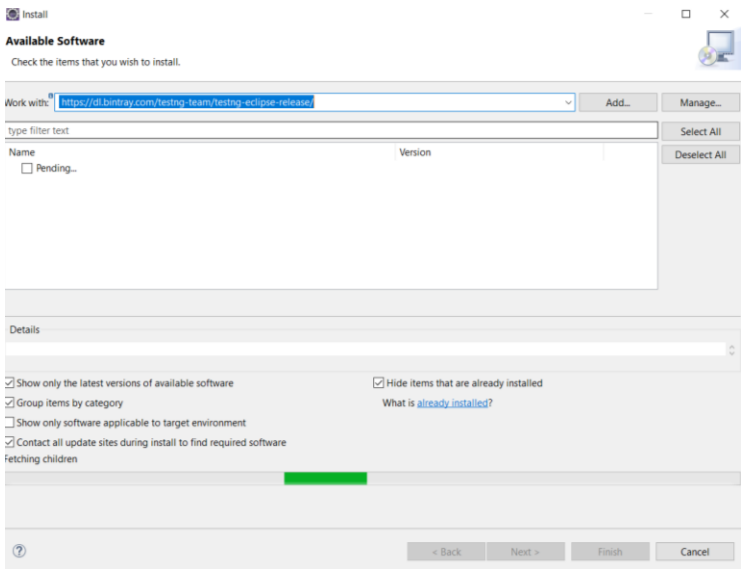


Figure 9.4

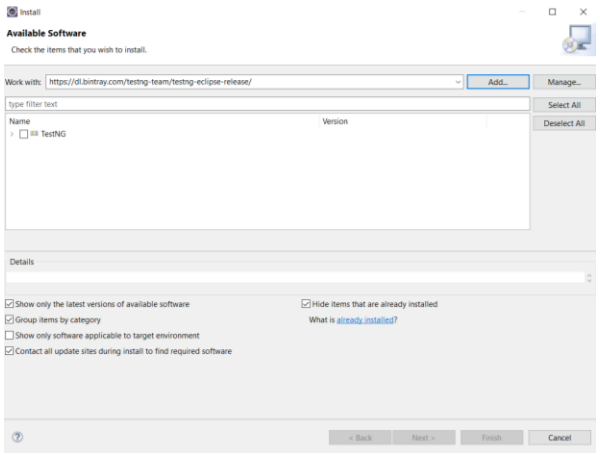


Figure 9.5

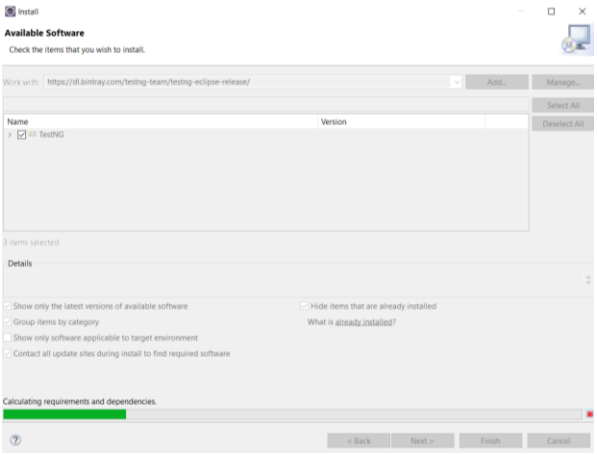


Figure 9.6

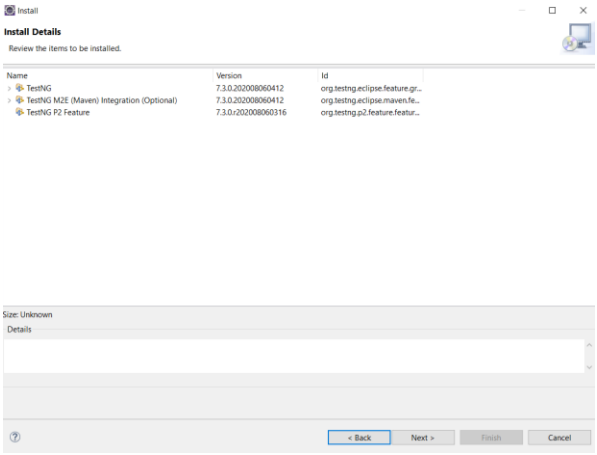


Figure 9.7

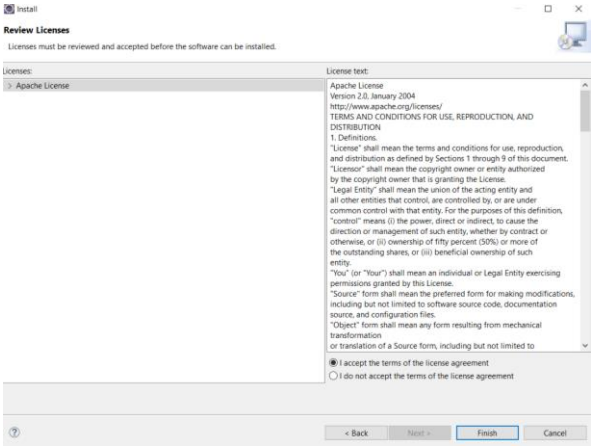


Figure 9.8

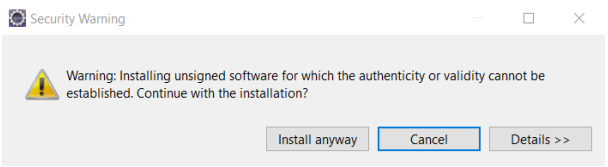


Figure 9.9

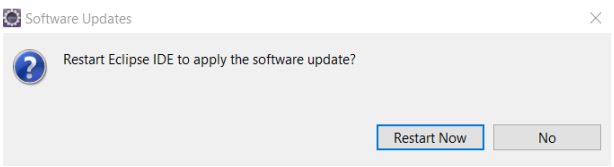


Figure 9.10

TestNG Annotation

Until now, the knowledge presented was bases on the Java programming language and its way of working. TestNG (NG stands for Next Generation) is a testing framework which uses the annotations (@). It overcomes the disadvantages founded by the developers. It is designed to make end-to-end testing easier. With this framework, it is possible to generate a proper report with all the passed, failed and skipped tests.

It uses the TestNG engine library instead of the java compiler. However, that does not mean that the Java Virtual Machine is not necessary to run the test. In the case of the normal Java execution, the test cases must be place in the main method or called from it. In the case of this framework, the test code is placed inside of a simple method with the annotation symbol followed by the text “Test”. With this annotation, the Eclipse recognizes it as a test case. Of course, it is necessary to import the TestNG library inside the code. For the first time the Eclipse does not recognize the annotation so the library it is needed to be imported manually. The framework has implemented an internal mechanism which display a report each time is run making the work of the tester more organized. In the case of pure Java, testing the only message displayed in the next code is “hello”. In the case of the TestNG framework more information are available. For example, the status of the test, the number of test that ran the number of the failed test and the number of skipped tests. In addition, the failed test cases can be executed separately. To run the test, right click on the code, go to “Run as” and select “TestNG test”.

```
1  package package87;
2
3  public class javabasics {
4
5      public static void main(String[] args) {
6          System.out.println("hello");
7      }
8
9  }
10
```

Figure 9.11

```
1  package package87;
2
3  import org.testng.annotations.Test;
4
5  public class basics {
6
7      @Test
8      public void Demo() {
9          System.out.println("hello");
10     }
11
12 }
```

Figure 9.12

```
[RemoteTestNG] detected TestNG version 7.3.0
hello
PASSED: Demo

===============================================
    Default test
    Tests run: 1, Failures: 0, Skips: 0
===============================================


===============================================
Default suite
Total tests run: 1, Passes: 1, Failures: 0, Skips: 0
===============================================
```

Figure 9.13

XML File

In a project, there could be a great number of test cases. The greater the number the harder is to manage them. A big number of test cases in a single file can make the file very hard to follow. In TestNG, there is the possibilities to write test cases in multiple file. In our example, there are three test cases in two different files. Right click on the package or project and select “TestNG” and then “Convert to TestNG”. Press “Finish” and a file with the extension XML is generated. Open it and select the Design window instead of the Source. Inside this file, the desired test cases can be called and all of them can run together. The structure of this file is compose of the Test Suite that are compose of Test Folder (Module) that are made themselves of test cases.

```java
package package88;

import org.testng.annotations.Test;

public class basics {

    @Test
    public void Demo() {
        System.out.println("hello");
    }

    @Test
    public void SecondTest() {
        System.out.println("bonjour");
    }

}

```

Figure 9.14

```java
package package88;

import org.testng.annotations.Test;

public class secondbasics {

    @Test
    public void ThirdTest() {
        System.out.println("ciao");
    }
}

```

Figure 9.15

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <test thread-count="5" name="Test">
    <classes>
      <class name="package88.basics"/>
      <class name="package88.secondbasics"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

Figure 9.16

```
[RemoteTestNG] detected TestNG version 7.3.0
hello
bonjour
ciao

===============================================
Suite
Total tests run: 3, Passes: 3, Failures: 0, Skips: 0
===============================================
```

Figure 9.17

Include and Exclude

There are situations in which it is wanted to include a certain test case or to run all of the test cases with the exception of some. For this thing with the “exclude” and “include” commands, it is possible to include or exclude a test case from being run. The syntax is quite similar with HTML. In a class, it is possible to call all of its methods in which the test cases are place. After the “exclude” or “include” command, the name of the method test case is written.

Please observe the possibility to give a name to the suite. In addition, the suite can be compose of multiple test as in the example below. Another observation is the fact that to end a tag, the slash character it is required to be inserted at the end of the command. Alternatively, write again the tag, but with the slash at the beginning. In this example, it can be observed that from the first class only the “SecondTest” method is display with the message “bonjour”. From the second class, all of the methods are called, but from the third class, there are only the “APIloginCarLoan” and the “WebloginCarLoan” present. To run the xml file, click right select “Run as”. Then select “TestNG Suite”.

```java
package package90;

import org.testng.annotations.Test;

public class basics {

    @Test
    public void Demo() {
        System.out.println("hello");
    }

    @Test
    public void SecondTest() {
        System.out.println("bonjour");
    }

    @Test
    public void ThirdTest() {
        System.out.println("ciao");
    }

}
```

Figure 9.18

```java
package package90;

import org.testng.annotations.Test;

public class secondbasics {

    @Test
    public void ForthTest() {
        System.out.println("Monday");
    }
    @Test
    public void FifthTest() {
        System.out.println("Friday");
    }
    @Test
    public void SixthTest() {
        System.out.println("Sunday");
    }
}
```

Figure 9.19

```java
package package90;

import org.testng.annotations.Test;

public class thirdbasics {

    @Test
    public void WebloginCarLoan() {
        System.out.println("WebloginCarLoan");
    }
    @Test
    public void MobileloginCarLoan() {
        System.out.println("MobileloginCarLoan");
    }
    @Test
    public void APIloginCarLoan() {
        System.out.println("APIloginCarLoan");
    }
}
```

Figure 13.20

```java
package package90;

import org.testng.annotations.Test;

public class forthbasics {
    @Test
    public void WebloginHomeLoan() {
        System.out.println("WebloginHomeLoan");
    }
    @Test
    public void MobileloginHomeLoan() {
        System.out.println("MobileloginHomeLoan");
    }
    @Test
    public void APIloginHomeLoan() {
        System.out.println("APIloginHomeLoan");
    }
}
```

Figure 13.21

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Loan Departament">

  <test thread-count="5" name="Personal Loan">
    <classes>

      <class name="package90.basics">
            <methods>
                <include name="SecondTest"></include>
            </methods>
      </class>
      <class name="package90.secondbasics"/>

    </classes>
  </test>


  <test thread-count="5" name="Car Loan">
    <classes>

       <class name="package90.thirdbasics">
             <methods>
                   <exclude name="MobileloginCarLoan"/>
             </methods>
       </class>
    </classes>
  </test>
</suite>

```

Figure 13.22

```
[RemoteTestNG] detected TestNG version 7.3.0
bonjour
Friday
Monday
Sunday
APIloginCarLoan
WebloginCarLoan

===============================================
Loan Departament
Total tests run: 6, Passes: 6, Failures: 0, Skips: 0
===============================================
```

Figure 13.23

Selecting Methods with Similar Name

In this example, all of the classes are the same with the ones in the previous part with the exception of the “thirdbasics”. In this class, two new methods are added for the demonstration purpose. The developer can use a convention in which all of the test cases involving the mobile part are starting with the letters “Mobile”. This framework has the capacity to select all of the methods, which start with these letters.

```
[RemoteTestNG] detected TestNG version 7.3.0
bonjour
Friday
Monday
Sunday
APIloginCarLoan
WebloginCarLoan

===============================================
Loan Departament
Total tests run: 6, Passes: 6, Failures: 0, Skips: 0
===============================================
```

Figure 9.24

```java
package package91;

import org.testng.annotations.Test;

public class thirdbasics {

    @Test
    public void WebloginCarLoan() {
        System.out.println("WebloginCarLoan");
    }
    @Test
    public void MobileloginCarLoan() {
        System.out.println("MobileloginCarLoan");
    }
    @Test
    public void MobileSignIn() {
        System.out.println("Mobile Sign In");
    }
    @Test
    public void MobileSignOut() {
        System.out.println("Mobile Sign Out");
    }
    @Test
    public void APIloginCarLoan() {
        System.out.println("APIloginCarLoan");
    }
}
```

Figure 9.25

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Loan Departament">

  <test thread-count="5" name="Personal Loan">
    <classes>

      <class name="package90.basics">
            <methods>
                <include name="SecondTest"></include>
            </methods>
      </class>
      <class name="package90.secondbasics"/>

    </classes>
  </test>


  <test thread-count="5" name="Car Loan">
    <classes>

       <class name="package90.thirdbasics">
             <methods>
                   <exclude name="Mobile.*"/>
             </methods>
       </class>
    </classes>
  </test>
</suite>

```

Figure 9.26

Select a package

The framework offers the possibility to run of the all desired methods if there are inside of a package. In this example, all of the classes are the same with the ones in the previous part.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Loan Departament">

  <test thread-count="5" name="Personal Loan">

    <packages>
        <package name="package91"/>
    </packages>

  </test>
</suite>

```

Figure 9.27

```
Friday
Monday
Sunday
APIloginCarLoan
Mobile Sign In
Mobile Sign Out
MobileloginCarLoan
WebloginCarLoan

===============================================
Loan Departament
Total tests run: 14, Passes: 14, Failures: 0, Skips: 0
===============================================
```

Figure 9.28

Test Annotation

In some situation there is necessary for example to clear the data from the previous test. Doing this manually is time consuming, so a method can be used for this task. The problem is that it is needed to be sure that this method is executed first. With the help of the annotation, BeforeTest a method can be forced to be executed first in its test. The opposite thing can be made with the annotation AfterTest. The method is executed last. In the example below the method which should be executed last, is not on the last raw. This thing happen because the methods after it are part of the second test inside the same suite. When the file is run, the whole suite is executed with all of its tests. In this example, all of the classes are the same with the ones in the previous part with the exception of the “basics” and “secondbasics”.

```java
package package92;

import org.testng.annotations.AfterTest;
import org.testng.annotations.Test;

public class basics {

    @AfterTest
    public void prerequiste() {
        System.out.println("I will execute last");
    }
    @Test
    public void Demo() {
        System.out.println("hello");
    }

    @Test
    public void SecondTest() {
        System.out.println("bonjour");
    }

    @Test
    public void ThirdTest() {
        System.out.println("ciao");
    }

}
```

Figure 9.29

```java
package package92;

import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class secondbasics {

    @Test
    public void ThirdTest() {
        System.out.println("ciao");
    }
    @Test
    public void ForthTest() {
        System.out.println("Monday");
    }
    @Test
    public void FifthTest() {
        System.out.println("Friday");
    }
    @Test
    public void SixthTest() {
        System.out.println("Sunday");
    }
    @BeforeTest
    public void prerequiste() {
        System.out.println("I will execute first");
    }
}
```

Figure 9.30

```
[RemoteTestNG] detected TestNG version 7.3.0
I will execute first
bonjour
Friday
Monday
Sunday
ciao
I will execute last
APIloginCarLoan
WebloginCarLoan

===============================================
Loan Departament
Total tests run: 7, Passes: 7, Failures: 0, Skips: 0
===============================================
```

Figure 9.31

Suite and Method Annotation

In addition, with this annotation there are some for suites as well. With the BeforeSuite and AfterSuite, the developer can run the method before or after every suite. Each with multiple tests inside of them. The annotation BeforeMethod and AfterMethod runs a method before or after each other, methods situated in the same test. In this example, all of the classes are the same with the ones in the previous part with the exception of the “basics” and “thirdbasics”.

```java
package package93;

import org.testng.annotations.AfterSuite;

public class basics {

    @AfterSuite
    public void AfterTheSuite() {
        System.out.println("I am the last from the suite");
    }
    @AfterTest
    public void prerequiste() {
        System.out.println("I will execute last");
    }
    @Test
    public void Demo() {
        System.out.println("hello");
    }

    @Test
    public void SecondTest() {
        System.out.println("bonjour");
    }

    @Test
    public void ThirdTest() {
        System.out.println("ciao");
    }

}
```

Figure 9.32

```java
package package93;

import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.Test;

public class thirdbasics {

    @BeforeSuite
    public void BeforeTheSuite(){
        System.out.println("I am the first from the suite");
    }
    @BeforeMethod
    public void BeforeTheMethod(){
        System.out.println("I am before each the method");
    }
    @AfterMethod
    public void AfterTheMethod(){
        System.out.println("I am after each the method");
    }
    @Test
    public void WebloginCarLoan() {
        System.out.println("WebloginCarLoan");
    }
    @Test
    public void MobileloginCarLoan() {
        System.out.println("MobileloginCarLoan");
    }
    @Test
    public void MobileSignIn() {
        System.out.println("Mobile Sign In");
    }
    @Test
    public void MobileSignOut() {
        System.out.println("Mobile Sign Out");
    }
    @Test
    public void APIloginCarLoan() {
        System.out.println("APIloginCarLoan");
    }
}
```

Figure 9.33

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Loan Departament">

  <test thread-count="5" name="Personal Loan">
    <classes>

      <class name="package93.basics">
            <methods>
                <include name="SecondTest"></include>
            </methods>
      </class>
      <class name="package93.secondbasics"/>
      <class name="package93.forthbasics"/>
    </classes>
  </test>


  <test thread-count="5" name="Car Loan">
    <classes>

       <class name="package93.thirdbasics">
             <methods>
                  <exclude name="Mobile.*"/>
             </methods>
       </class>
    </classes>
  </test>
</suite>

```

Figure 9.34

```
[RemoteTestNG] detected TestNG version 7.3.0
I am the first from the suite
I will execute first
bonjour
Friday
Monday
Sunday
APIloginHomeLoan
MobileloginHomeLoan
WebloginHomeLoan
I will execute last
I am before each the method
APIloginCarLoan
I am after each the method
I am before each the method
WebloginCarLoan
I am after each the method
I am the last from the suite

===============================================
Loan Departament
Total tests run: 9, Passes: 9, Failures: 0, Skips: 0
===============================================
```

Figure 9.35

Class Annotation

With the BeforeClass and AfterClassSuite, the developer can run the method before or after every class. In this example, all of the classes are the same with the ones in the previous part with the exception of the “thirdbasics”.

```java
package package94;

import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.Test;

public class thirdbasics {

    @BeforeClass
    public void BeforeTheClass(){
        System.out.println("I am before all the methods form from class thirdbasics");
    }
    @AfterClass
    public void AfterTheClass(){
        System.out.println("I am after all the methods form from class thirdbasics");
    }
    @BeforeSuite
    public void BeforeTheSuite(){
        System.out.println("I am the first from the suite");
    }
    @BeforeMethod
    public void BeforeTheMethod(){
        System.out.println("I am before each method from class thirdbasics");
    }
    @AfterMethod
    public void AfterTheMethod(){
        System.out.println("I am after each method from class thirdbasics");
    }
    @Test //(groups={"Smoke"})
    public void WebloginCarLoan() {
        System.out.println("WebloginCarLoan");
    }
    @Test
    public void MobileloginCarLoan() {
        System.out.println("MobileloginCarLoan");
    }
```

Figure 9.36

```java
    @Test
    public void MobileSignIn() {
        System.out.println("Mobile Sign In");
    }
    @Test
    public void MobileSignOut() {
        System.out.println("Mobile Sign Out");
    }
    @Test //(groups={"Smoke"})
    public void APIloginCarLoan() {
        System.out.println("APIloginCarLoan");
    }
}

```

Figure 9.37

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Loan Departament">

  <test thread-count="5" name="Personal Loan">
    <classes>

      <class name="package94.basics">
           <methods>
               <include name="SecondTest"></include>
           </methods>
      </class>
      <class name="package94.secondbasics"/>
      <class name="package94.forthbasics"/>
    </classes>
  </test>


  <test thread-count="5" name="Car Loan">
    <classes>

       <class name="package94.thirdbasics">
            <methods>
                 <exclude name="Mobile.*"/>
            </methods>
       </class>
    </classes>
  </test>
</suite>

```

Figure 9.38

```
[RemoteTestNG] detected TestNG version 7.3.0
I am the first from the suite
I will execute first
bonjour
Friday
Monday
Sunday
APIloginHomeLoan
MobileloginHomeLoan
WebloginHomeLoan
I will execute last
I am before all the methods form from class thirdbasics
I am before each method from class thirdbasics
APIloginCarLoan
I am after each method from class thirdbasics
I am before each method from class thirdbasics
WebloginCarLoan
I am after each method from class thirdbasics
I am after all the methods form from class thirdbasics
I am the last from the suite

===============================================
Loan Departament
Total tests run: 9, Passes: 9, Failures: 0, Skips: 0
===============================================
```

Figure 9.39

Group Annotation

In the case in which the developer want to run some methods, which do not share the beginning of the name, it is possible to use the group annotation. In the following example, the developer can run only the methods that are part of the “Smoke” group. In this example, all of the classes are the same with the ones in the previous part with the exception of the “secondbasics” and “thirdbasics”.

```java
package package94;

import org.testng.annotations.BeforeTest;

public class secondbasics {

    @Test
    public void ForthTest() {
        System.out.println("Monday");
    }
    @Test
    public void FifthTest() {
        System.out.println("Friday");
    }
    @Test (groups=("Smoke"))
    public void SixthTest() {
        System.out.println("Sunday");
    }
    @BeforeTest
    public void prerequiste() {
        System.out.println("I will execute first");
    }
}

```

Figure 9.40

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Loan Departament">

  <test thread-count="5" name="Personal Loan">
    <groups>
        <run>
            <include name="Smoke"/>
        </run>
    </groups>
    <classes>

      <class name="package94.basics"/>
      <class name="package94.secondbasics"/>
      <class name="package94.thirdbasics"/>
      <class name="package94.forthbasics"/>

    </classes>
  </test>

</suite>

```

Figure 9.41

```java
package package94;

import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.Test;

public class thirdbasics {

    @BeforeClass
    public void BeforeTheClass(){
        System.out.println("I am before all the methods form from class thirdbasics");
    }
    @AfterClass
    public void AfterTheClass(){
        System.out.println("I am after all the methods form from class thirdbasics");
    }
    @BeforeSuite
    public void BeforeTheSuite(){
        System.out.println("I am the first from the suite");
    }
    @BeforeMethod
    public void BeforeTheMethod(){
        System.out.println("I am before each method from class thirdbasics");
    }
    @AfterMethod
    public void AfterTheMethod(){
        System.out.println("I am after each method from class thirdbasics");
    }
    @Test (groups=("Smoke"))
    public void WebloginCarLoan() {
        System.out.println("WebloginCarLoan");
    }
    @Test
    public void MobileloginCarLoan() {
        System.out.println("MobileloginCarLoan");
    }
```

Figure 9.42

```java
    @Test
    public void MobileSignIn() {
        System.out.println("Mobile Sign In");
    }
    @Test
    public void MobileSignOut() {
        System.out.println("Mobile Sign Out");
    }
    @Test (groups=("Smoke"))
    public void APIloginCarLoan() {
        System.out.println("APIloginCarLoan");
    }
}
```

Figure 9.43

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Loan Departament">

  <test thread-count="5" name="Personal Loan">
    <groups>
        <run>
            <include name="Smoke"/>
        </run>
    </groups>
    <classes>

      <class name="package94.basics"/>
      <class name="package94.secondbasics"/>
      <class name="package94.thirdbasics"/>
      <class name="package94.forthbasics"/>

    </classes>
  </test>

</suite>
```

Figure 9.43

```
[RemoteTestNG] detected TestNG version 7.3.0
Sunday
APIloginCarLoan
WebloginCarLoan

===============================================
Loan Departament
Total tests run: 3, Passes: 3, Failures: 0, Skips: 0
===============================================
```

Figure 9.44

Method Execution Dependency

In the case in which the developer want to run a method only if a previous method or methods are already executed, it is possible to use the dependsOnMethods annotation. In the following example, the developer can run the “APIloginCarLoan” only after the methods “MobileloginCarLoan” and “WebloginCarLoan” are already executed. The methods inside of a class are executed in the alphabetical order. In this example, all of the classes are the same with the ones in the previous part with the exception of the “thirdbasics”.

```java
package package95;

import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.Test;

public class thirdbasics {

    @BeforeClass
    public void BeforeTheClass(){
        System.out.println("I am before all the methods form from class thirdbasics");
    }
    @AfterClass
    public void AfterTheClass(){
        System.out.println("I am after all the methods form from class thirdbasics");
    }
    @BeforeSuite
    public void BeforeTheSuite(){
        System.out.println("I am the first from the suite");
    }
    @BeforeMethod
    public void BeforeTheMethod(){
        System.out.println("I am before each method from class thirdbasics");
    }
    @AfterMethod
    public void AfterTheMethod(){
        System.out.println("I am after each method from class thirdbasics");
    }
    @Test
    public void WebloginCarLoan() {
        System.out.println("WebloginCarLoan");
    }
    @Test
    public void MobileloginCarLoan() {
        System.out.println("MobileloginCarLoan");
    }
```

Figure 9.45

```java
    @Test
    public void MobileSignIn() {
        System.out.println("Mobile Sign In");
    }
    @Test
    public void MobileSignOut() {
        System.out.println("Mobile Sign Out");
    }
    @Test (dependsOnMethods={"MobileloginCarLoan","WebloginCarLoan"})
    public void APIloginCarLoan() {
        System.out.println("APIloginCarLoan");
    }
}

```

Figure 9.46

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Loan Departament">

  <test thread-count="5" name="Personal Loan">
    <classes>

      <class name="package95.thirdbasics"/>

    </classes>
  </test>

</suite>

```

Figure 9.47

```
[RemoteTestNG] detected TestNG version 7.3.0
I am the first from the suite
I am before all the methods form from class thirdbasics
I am before each method from class thirdbasics
Mobile Sign Out
I am after each method from class thirdbasics
I am before each method from class thirdbasics
MobileloginCarLoan
I am after each method from class thirdbasics
I am before each method from class thirdbasics
WebloginCarLoan
I am after each method from class thirdbasics
I am before each method from class thirdbasics
Mobile Sign In
I am after each method from class thirdbasics
I am before each method from class thirdbasics
APIloginCarLoan
I am after each method from class thirdbasics
I am after all the methods form from class thirdbasics

===============================================
Loan Departament
Total tests run: 5, Passes: 5, Failures: 0, Skips: 0
===============================================
```

Figure 9.48

Parameters from XML File

It is possible to send parameter from the XML file to the methods. In the following example the URL address “generalloan.com” is transmitted to the “APIloginCarLoan” . In this example, all of the classes are the same with the ones in the previous part with the exception of the “secondbasics” and “thirdbasics”.

```java
package package96;

import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;

public class thirdbasics {

    @BeforeClass
    public void BeforeTheClass(){
        System.out.println("I am before all the methods form from class thirdbasics");
    }
    @AfterClass
    public void AfterTheClass(){
        System.out.println("I am after all the methods form from class thirdbasics");
    }
    @BeforeSuite
    public void BeforeTheSuite(){
        System.out.println("I am the first from the suite");
    }
    @BeforeMethod
    public void BeforeTheMethod(){
        System.out.println("I am before each method from class thirdbasics");
    }
    @AfterMethod
    public void AfterTheMethod(){
        System.out.println("I am after each method from class thirdbasics");
    }

    @Parameters({"URL"})
    @Test (groups={"Smoke"})
    public void WebloginCarLoan(String urlname) {
        System.out.println("WebloginCarLoan:"+urlname);
    }
```

Figure 9.49

```java
    @Test
    public void MobileloginCarLoan() {
        System.out.println("MobileloginCarLoan");
    }
    @Test
    public void MobileSignIn() {
        System.out.println("Mobile Sign In");
    }
    @Test
    public void MobileSignOut() {
        System.out.println("Mobile Sign Out");
    }
    @Test (groups={"Smoke"},dependsOnMethods={"MobileloginCarLoan","WebloginCarLoan"})
    public void APIloginCarLoan() {
        System.out.println("APIloginCarLoan");
    }
}
```

Figure 9.50

```java
package package96;

import org.testng.annotations.BeforeTest;

public class secondbasics {

    @Test
    public void ForthTest() {
        System.out.println("Monday");
    }
    @Test
    public void FifthTest() {
        System.out.println("Friday");
    }
    @Parameters({"URL"})
    @Test (groups={"Smoke"})
    public void SixthTest(String urlname) {
        System.out.println("Sunday:"+urlname);
    }
    @BeforeTest
    public void prerequiste() {
        System.out.println("I will execute first");
    }
}
```

Figure 9.51

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Loan Departament">
    <parameter name="URL" value="generalloan.com"/>
  <test thread-count="5" name="Personal Loan">
    <parameter name="URL" value="personalloan.com"/>
    <classes>

      <class name="package96.basics"/>
      <class name="package96.secondbasics"/>
      <class name="package96.forthbasics"/>

    </classes>
  </test>


  <test thread-count="5" name="Car Loan">
    <parameter name="URL" value="carloan.com"/>
    <classes>

       <class name="package96.thirdbasics">

       </class>
    </classes>
  </test>
</suite>
```

Figure 9.52

```
[RemoteTestNG] detected TestNG version 7.3.0
I am the first from the suite
I will execute first
hello
bonjour
ciao
Friday
Monday
Sunday:personalloan.com
APIloginHomeLoan
MobileloginHomeLoan
WebloginHomeLoan
I will execute last
I am before all the methods form from class thirdbasics
I am before each method from class thirdbasics
Mobile Sign Out
I am after each method from class thirdbasics
I am before each method from class thirdbasics
MobileloginCarLoan
I am after each method from class thirdbasics
I am before each method from class thirdbasics
WebloginCarLoan:carloan.com
I am after each method from class thirdbasics
I am before each method from class thirdbasics
Mobile Sign In
I am after each method from class thirdbasics
I am before each method from class thirdbasics
APIloginCarLoan
I am after each method from class thirdbasics
I am after all the methods form from class thirdbasics
I am the last from the suite

===============================================
Loan Departament
Total tests run: 14, Passes: 14, Failures: 0, Skips: 0
===============================================
```

Figure 9.53

Enabled, Time Out and Multiple Data Sets

There are situations in which during the test cases it is required to introduce multiple values of the same parameter. In the following example, the class “getData” keeps the values in a matrix. The advantage of this, it is the fact that the Appium server test the same method for each of it parameters. The developer does not need to call the method for each data set. In some cases, a method can take a lot of time to be perform. For example for loading a web page. For this reason, a test is stopped when it takes more than a certain amount of time to run. The annotation “timeout” perform this task. If the developer does not want to run a test, it can use the “enabled” annotation.

```java
package package98;

import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;

public class thirdbasics {

    @BeforeClass
    public void BeforeTheClass(){
        System.out.println("I am before all the methods form from class thirdbasics");
    }
    @AfterClass
    public void AfterTheClass(){
        System.out.println("I am after all the methods form from class thirdbasics");
    }
    @BeforeSuite
    public void BeforeTheSuite(){
        System.out.println("I am the first from the suite");
    }
    @BeforeMethod
    public void BeforeTheMethod(){
        System.out.println("I am before each method from class thirdbasics");
    }
    @AfterMethod
    public void AfterTheMethod(){
        System.out.println("I am after each method from class thirdbasics");
    }

    @Parameters({"URL","APIKey/username"})
    @Test (groups={"Smoke"})
    public void WebloginCarLoan(String urlname,String key) {
        System.out.println("WebloginCarLoan:"+urlname+" and "+key);
    }
```

Figure 9.53

```java
    @DataProvider
    public Object[][] getData() {
        //1st-username password- good credit history
        //2nd-username password- no credit history
        //3rd-username password- bad credit history

        Object[][] data=new Object[3][2];

        data[0][0]="firstUsername";
        data[0][1]="firstPassword";

        data[1][0]="secondUsername";
        data[1][1]="secondPassword";

        data[2][0]="thirdUsername";
        data[2][1]="thirdPassword";

        return data;
    }
    @Test(dataProvider="getData")
    public void MobileloginCarLoan(String username,String password) {
        System.out.println("MobileloginCarLoan");
        System.out.println(username+":"+password);
    }
    @Test(enabled=false)
    public void MobileSignIn() {
        System.out.println("Mobile Sign In");
    }
    @Test(timeOut=4000)
    public void MobileSignOut() {
        System.out.println("Mobile Sign Out");
    }
    @Test (groups={"Smoke"},dependsOnMethods={"MobileloginCarLoan","WebloginCarLoan"})
    public void APIloginCarLoan() {
        System.out.println("APIloginCarLoan");
    }
}
```

Figure 9.54

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Loan Departament">
    <parameter name="URL" value="generalloan.com"/>
  <test thread-count="5" name="Personal Loan">
    <classes>

      <class name="package98.basics"/>
      <class name="package98.secondbasics"/>
      <class name="package98.forthbasics"/>

    </classes>
  </test>


  <test thread-count="5" name="Car Loan">
    <parameter name="APIKey/username" value="123456"/>
    <classes>

       <class name="package98.thirdbasics">

       </class>
    </classes>
  </test>
</suite>

```

Figure 9.55

```
I will execute first
hello
bonjour
ciao
Friday
Monday
Sunday:generalloan.com
APIloginHomeLoan
MobileloginHomeLoan
WebloginHomeLoan
I will execute last
I am before all the methods form from class thirdbasics
I am before each method from class thirdbasics
Mobile Sign Out
I am after each method from class thirdbasics
I am before each method from class thirdbasics
MobileloginCarLoan
firstUsername:firstPassword
I am after each method from class thirdbasics
I am before each method from class thirdbasics
MobileloginCarLoan
secondUsername:secondPassword
I am after each method from class thirdbasics
I am before each method from class thirdbasics
MobileloginCarLoan
thirdUsername:thirdPassword
I am after each method from class thirdbasics
I am before each method from class thirdbasics
WebloginCarLoan:generalloan.com and 123456
I am after each method from class thirdbasics
I am before each method from class thirdbasics
APIloginCarLoan
I am after each method from class thirdbasics
I am after all the methods form from class thirdbasics
I am the last from the suite

===============================================
Loan Departament
Total tests run: 15, Passes: 15, Failures: 0, Skips: 0
===============================================
```

Figure 9.56

Chapter 10: Maven Framework

Introduction to Maven

Maven is a project management and comprehension tool that provides developers a complete build lifecycle framework. Development team can automate the project's build infrastructure in almost no time as Maven uses a standard directory layout and a default build lifecycle. In case of multiple development teams environment, Maven can set-up the way to work as per standards in a very short time. As most of the project setups are simple and reusable, Maven makes life of developer easy while creating reports, checks, build and testing automation setups.

Maven takes care of the JAR files required in the project automatically. All of the components are placed in a central repository from where they are downloaded. The components are called dependencies.

Installation of Maven

The requirement is to have the Java Virtual Machine on your computer the Java Environment path set. Search on the browser “maven download”. Enter on the official site. It is necessary to have the Java Virtual Machine and the Java Environment path set on the computer. Download the “apache-maven-3.6.3-bin.zip” from the first column and second row.

Enter in the Control Panel and select System. Look after “Advanced system setting” and press on the “Environment Variables” button. In addition, create a new path named “MAVEN_HOME” and paste the path to Maven unzipped folder. Press on the OK button and search for the Path variable. Select to edit it and paste the path to the maven folder until the bin folder. To check if the installation was properly made, open Command Prompt and write “mvn –version”. If the installation is good, the prompter should look like in the next figures. Otherwise, is the prompter display that the line of code is not recognized as an internal or external command, check again the system paths.

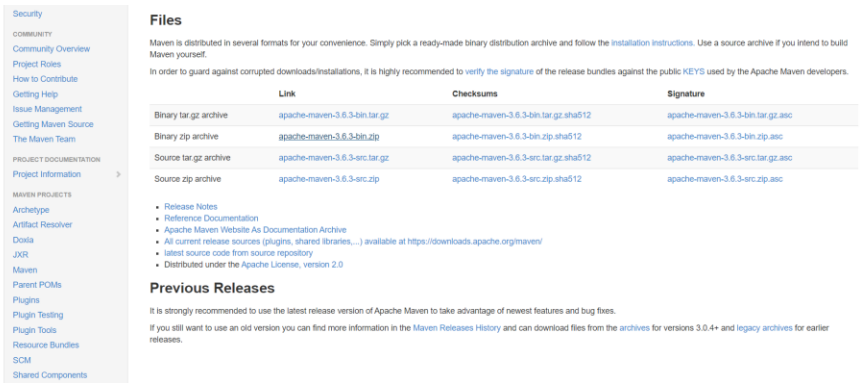


Figure 10.1

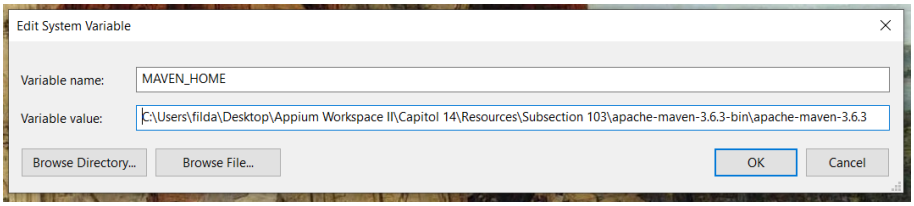


Figure 10.2

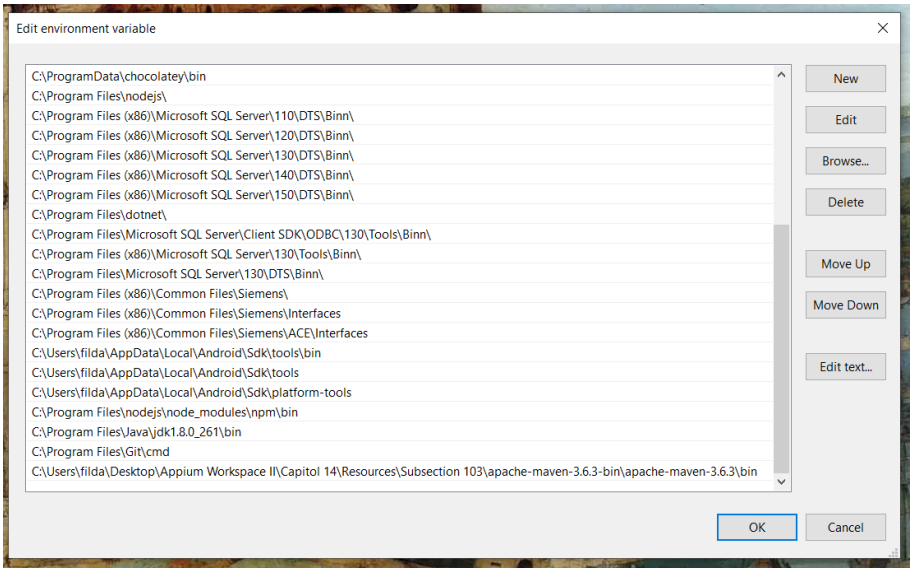


Figure 10.3

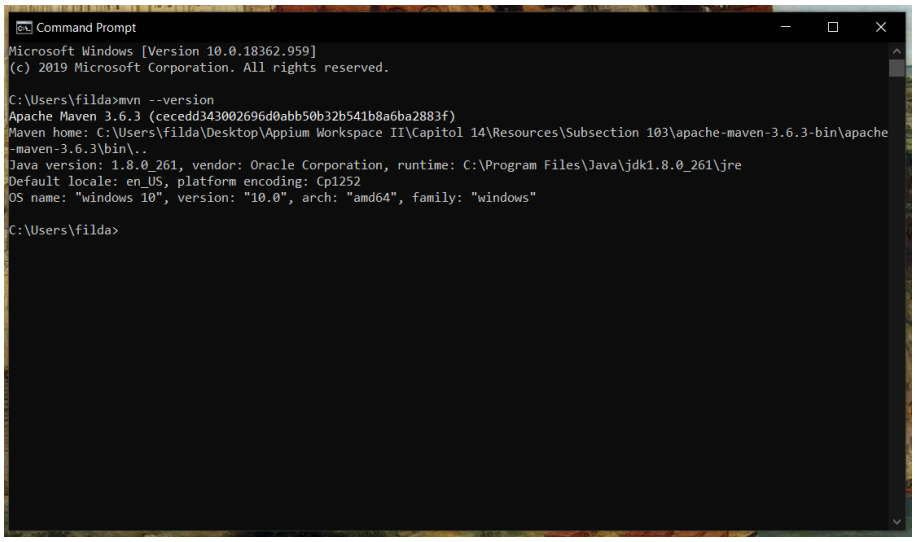


Figure 10.4

Create a Maven Project

In the Project Explorer of the Eclipse, right click on it. On the pop-up option window, select New and then Project. Select Maven Project and provide the name of the new project. In the filter text box, search for the “quickstart” archetype version 1.4 from Apache in order have the general Maven setting. Follow the next figures in order to create the first Maven Project.

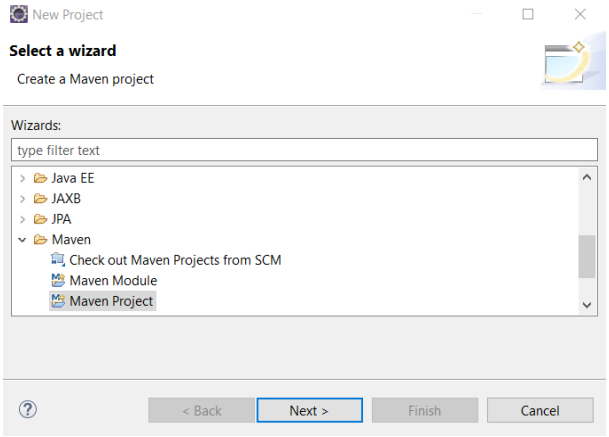


Figure 10.5

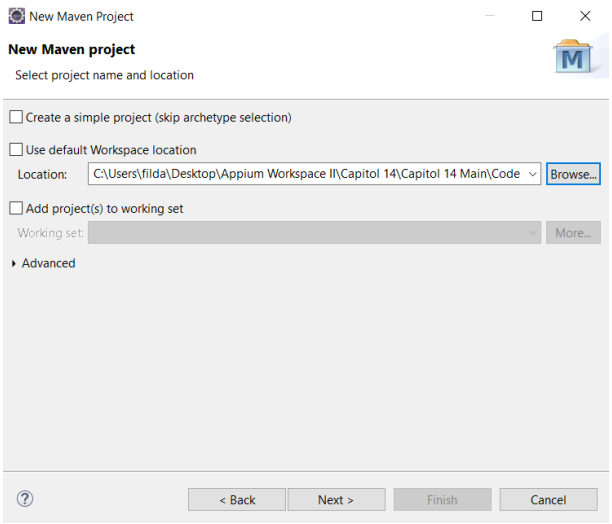


Figure 10.6

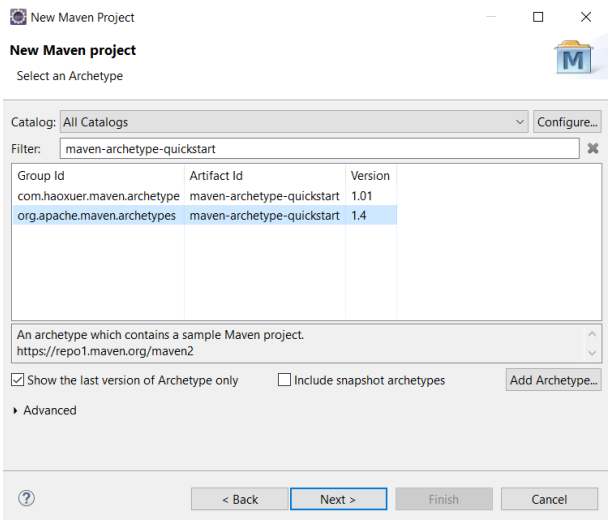


Figure 10.7

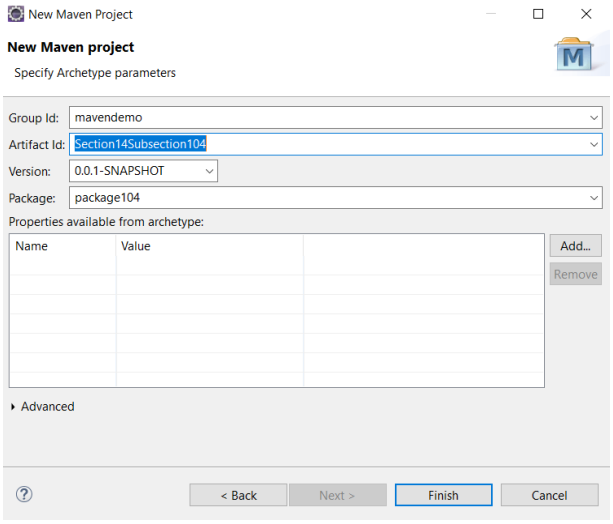


Figure 10.8

Project Object Model

Maven takes care of the JAR files, plugins and libraries. In order to do it, it is necessary to transmit which dependencies the developer want. In this framework, there are a file name “pom.xml”. In this file all the maven configuration can be done. For example the required dependencies for the current project.

In the following example, let us import three depositories in the project. The dependencies are for the Appium, Selenium and Rest. There are also a dependency for TestNG. Search on the browser “maven repository” and enter on their official website. Search each of the dependencies on the website and copy the lines of code provided in the Maven section. Let us create three class for each of the testing platforms. They can have two methods for the demonstration purposes.

In addition, a plugin from Apache permits the execution of the test form the Command Prompt. It can be found on the Maven official website on the Usage section. To run the test cases from Maven, enter in Command Prompt and go the directory of the project. With the command “mvn clean”, the data about previous tests is deleted. The command “mvn compile” verifies if there are any errors. In the end, use the command “mvn test” to run the test cases from Maven.

In this file, there is the possibility to create multiple command for running the test. In the following example, there are the “Smoke” and “Regression” test that can be run as a command from Command Prompt. They work like the group annotation in the TestNG.

Figure 10.9

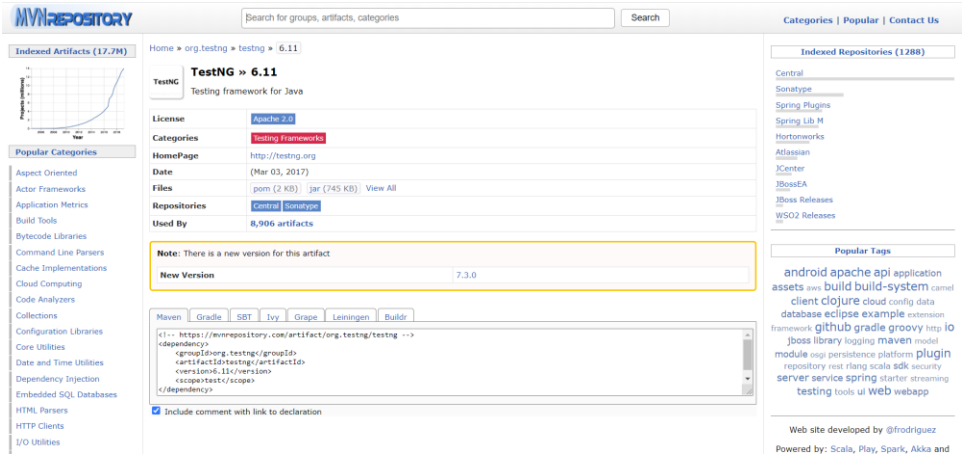


Figure 10.10

Figure 10.11

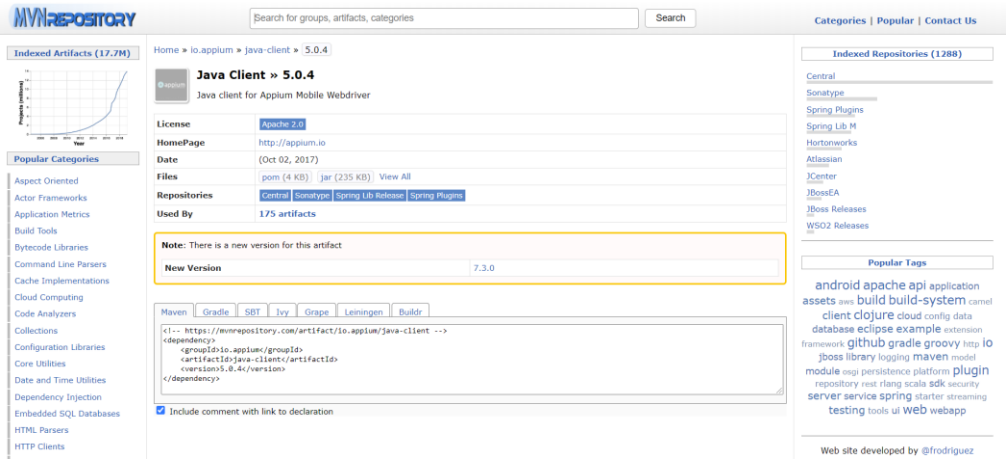


Figure 10.12

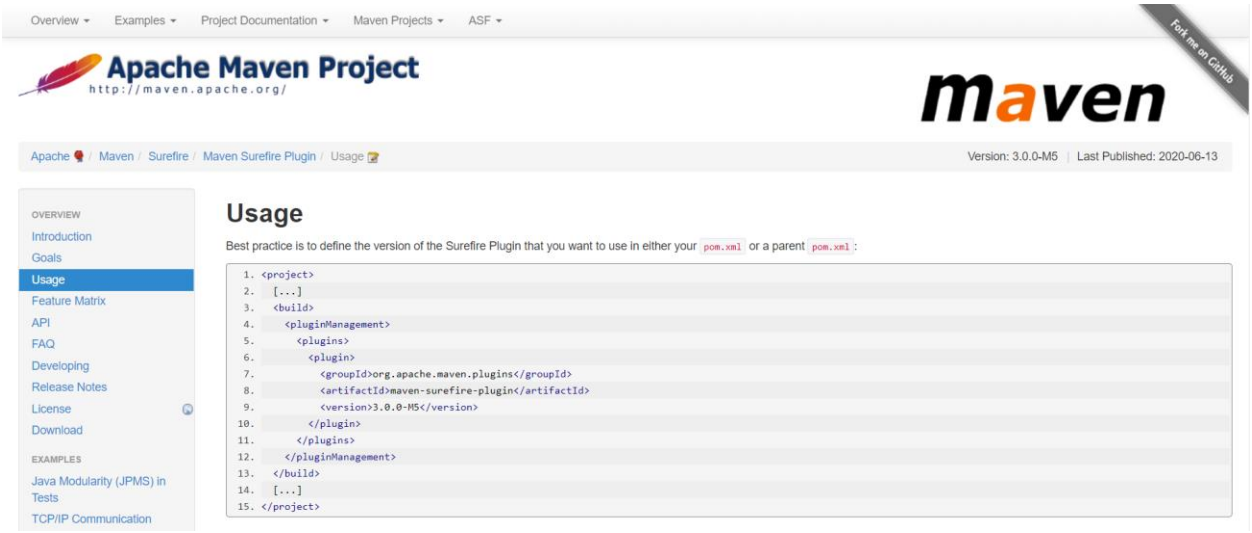


Figure 10.13

```java
package package104;

import org.testng.annotations.Test;

public class AppiumTest {

    @Test
    public void NativeAppAndroid() {
        System.out.println( "NativeAppAndroid" );
    }
    @Test
    public void IOSApps() {
        System.out.println( "IOSApps" );
    }
}

```

Figure 10.13

```java
package package104;

import org.testng.annotations.Test;

public class RestAPItest {
    @Test
    public void PostJira() {
        System.out.println( "PostJira" );
    }
    @Test
    public void DeleteTwitter() {
        System.out.println( "DeleteTwitter" );
    }
}

```

Figure 10.14

```java
package package104;

import org.testng.annotations.Test;

public class SeleniumTest {

    @Test
    public void BrowserAutomation() {
        System.out.println("BrowserAutomation");
    }
    @Test
    public void ElementsUI() {
        System.out.println("ElementsUI");
    }
}

```

Figure 10.16

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <test name="Test">
    <classes>
      <class name="package102.SeleniumTest"/>
      <class name="package102.RESTAPITest"/>
      <class name="package102.AppiumTest"/>

    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

Figure 10.17

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>qaclickacademy</groupId>
  <artifactId>Mavenjava</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Mavenjava</name>
  <url>http://maven.apache.org</url>

  <profiles>
  <profile>
  <id>Regression</id>

  <build>
    <pluginManagement>
      <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>2.20.1</version>
           <configuration>
          <suiteXmlFiles>
            <suiteXmlFile>testng2.xml</suiteXmlFile>
          </suiteXmlFiles>
        </configuration>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>
  </profile>
  <profile>

```

Figure 10.18

```xml
   <id>Smoke</id>

  <build>
    <pluginManagement>
      <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>2.20.1</version>
           <configuration>
          <suiteXmlFiles>
            <suiteXmlFile>testng.xml</suiteXmlFile>
          </suiteXmlFiles>
        </configuration>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>

  </profile>
  </profiles>


  <dependencies>

  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>3.6.0</version>
</dependency>

```

Figure 10.19

```
    <!-- https://mvnrepository.com/artifact/org.testng/testng -->
<dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>6.11</version>
    <scope>test</scope>
</dependency>

    <!-- https://mvnrepository.com/artifact/io.rest-assured/rest-assured -->
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <version>3.0.5</version>
    <scope>test</scope>
</dependency>
    <!-- https://mvnrepository.com/artifact/io.appium/java-client -->
<dependency>
    <groupId>io.appium</groupId>
    <artifactId>java-client</artifactId>
    <version>5.0.4</version>
</dependency>

  </dependencies>
</project>

```

Figure 10.20

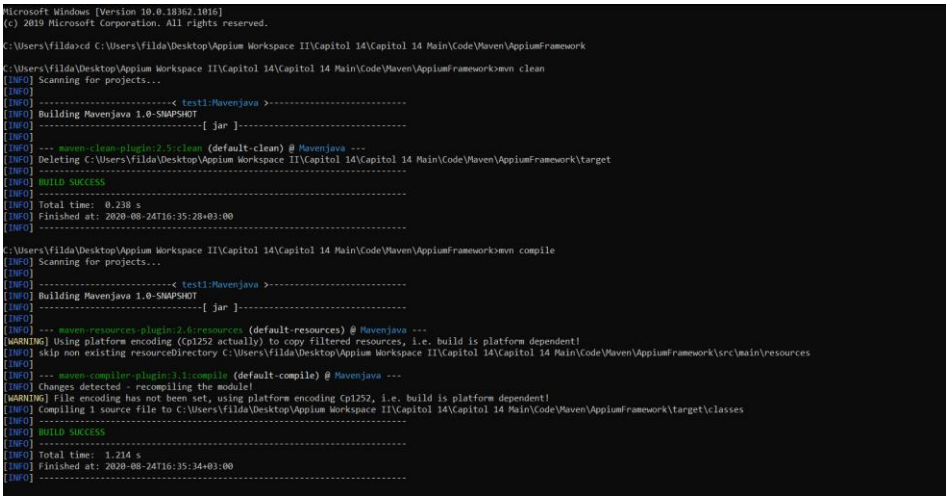


Figure 10.21

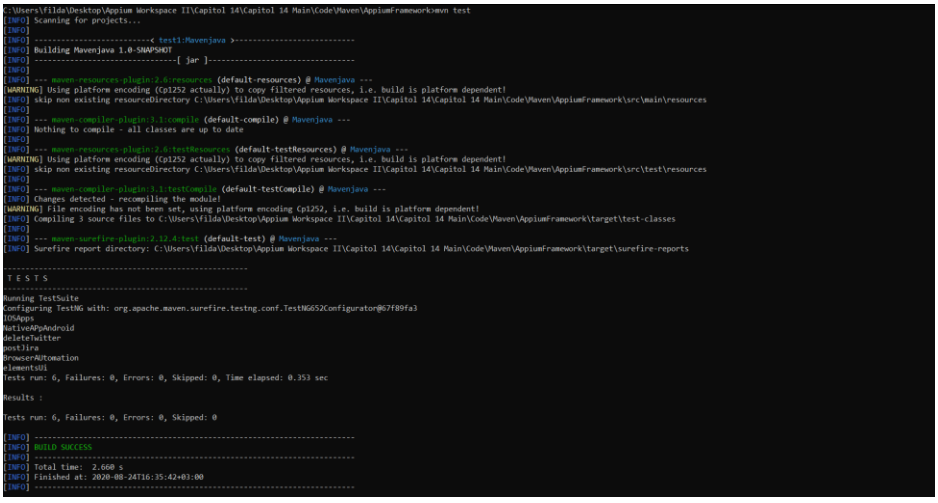


Figure 10.22

Chapter 11: Jenkins Framework

Introduction to Jenkins

To run the test from the Command Prompt it is more convenient then having Eclipse open, but there are still a great number of inconveniences. It is require to work in command prompt and for there are still a few command to write. With Jenkins, there is the possibility pass over the interaction with the Command Prompt. Jenkins is an automation server that offers a graphical interface. It success to resolve many of the problems regarding testing. It is best known in the field of Continuous Integration. Jenkins interface can be access from any web browser.

Continuous Integration is a development practice in which the developers are required to commit changes to the source code in a shared repository several times a day or more frequently. Every commit made in the repository is then built. This allows the teams to detect the problems early.

Installation of Jenkins

Open the web browser and search "Jenkins download". Alternatively, open the following link: "https://www.jenkins.io/download/". Download the "Generic Java Package (.war)". To start the Jenkins, open the Command Prompt and go the directory of the downloaded file using the "cd" command. Then, write "java –jar Jenkins.war –htttpPort-9090". Enter in the browser on the following address "http://localhost:8080/". In the installation process, it is require providing a username and a password. Also the number of the port on which the server will run.

After the installation is done, on the main page of the Jenkins, press on Manage Jenkins. Then select Global Tool Configurator. Here provide in the JDK section the JAVA_HOME path. In the Maven section, write the MAVEN_HOME path.

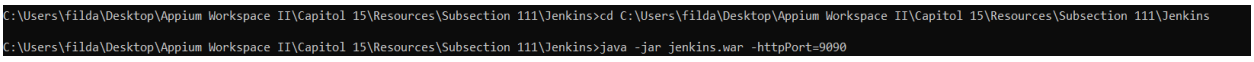


Figure 11.1

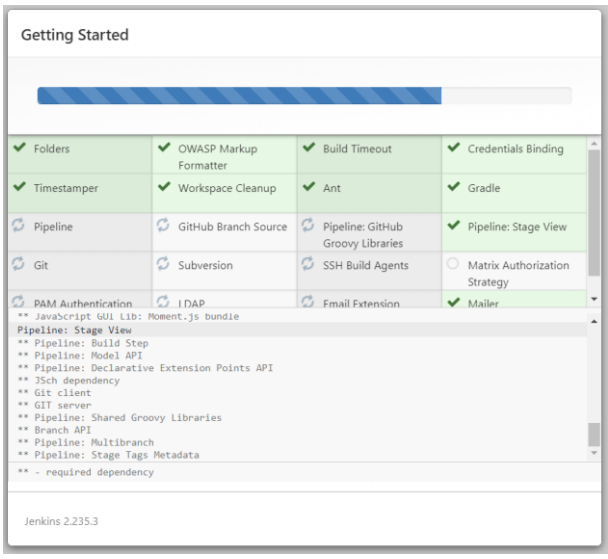


Figure 11.2

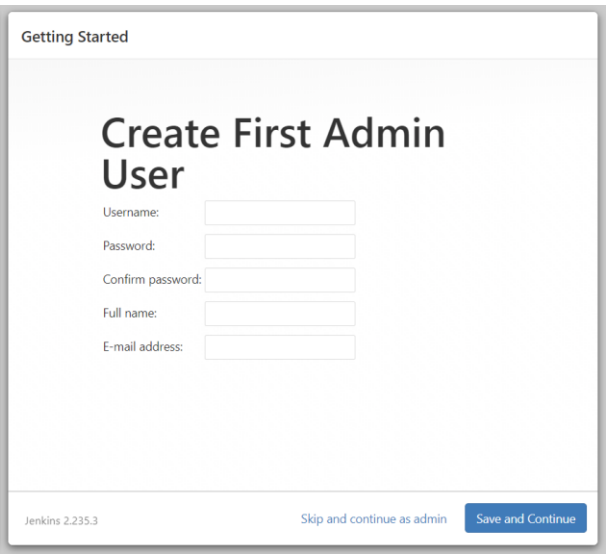


Figure 11.3

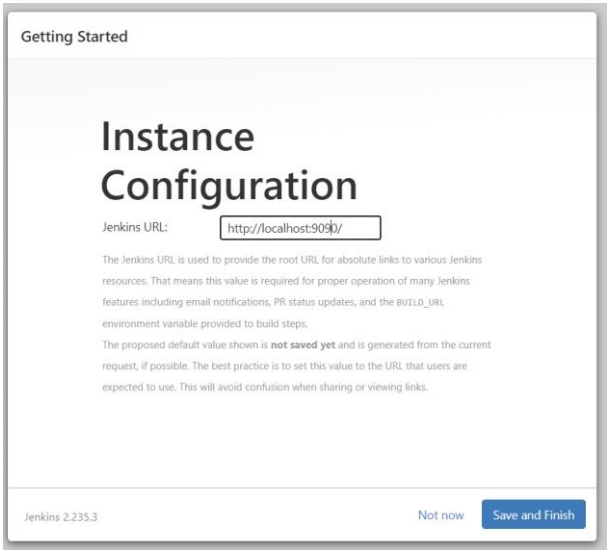


Figure 11.4

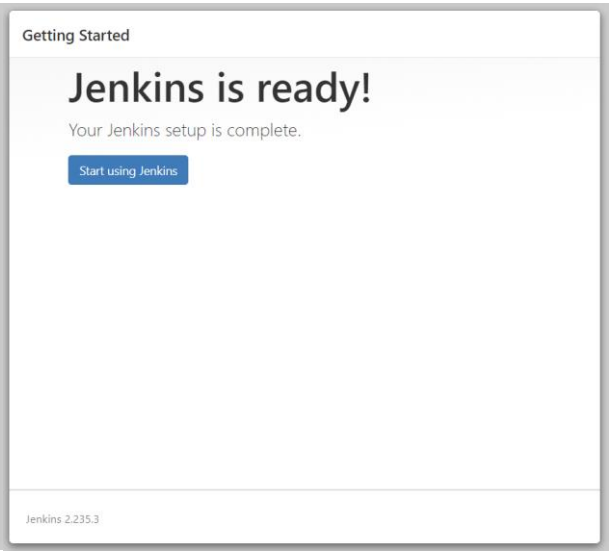


Figure 11.5

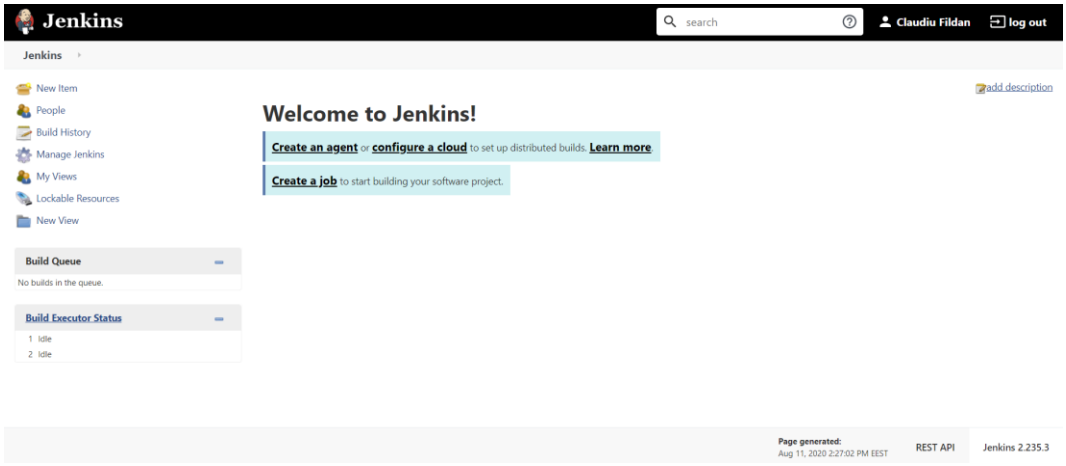


Figure 11.6

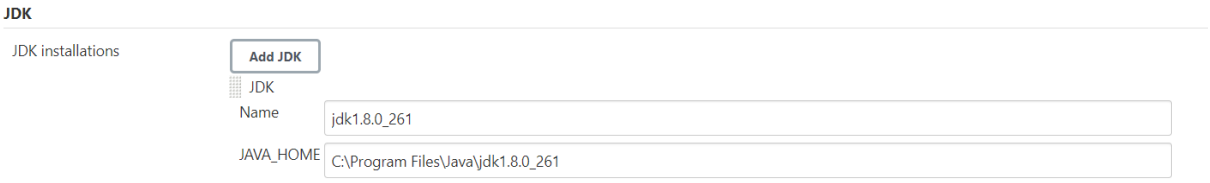


Figure 11.7

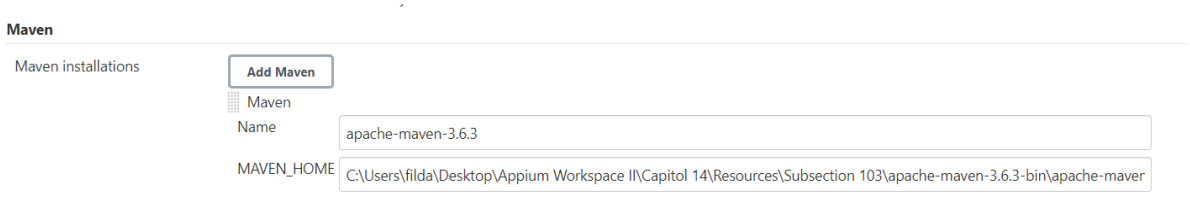


Figure 11.8

Workspace in Jenkins

To connect a project to Jenkins, select New Item. Introduce the name of the Project and choose type of the project to be “Freestyle project”. Copy the project folder and put it in the following path: “C:\Users\filda\.jenkins”. The program detects it automatically. In the general section, press on the Advanced button. Check the “use custom workspace” and write the following line :”$(JENKINS_HOME)/Mavenjava”. Here “Mavenjava” is the name of the project. The program detects the project in its directory. In the Built Environment, section go to the Build segment. Choose the option “Invoke top-level maven targets”. In this window, it is require introducing the version of the maven server and the command to the Maven. In this example, the test cases from the “Regression” are run with the command “test –P”. Press on the save button. Return on the main page and press on the Build Now. In the build History a new item appear. If the circle has the color red, the test failed. If the color is blue, the test run. Click on the circle then select on Console Output. Here the output of the test are displayed.

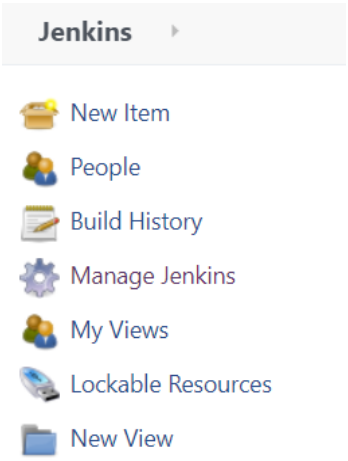


Figure 11.9

Figure 11.10

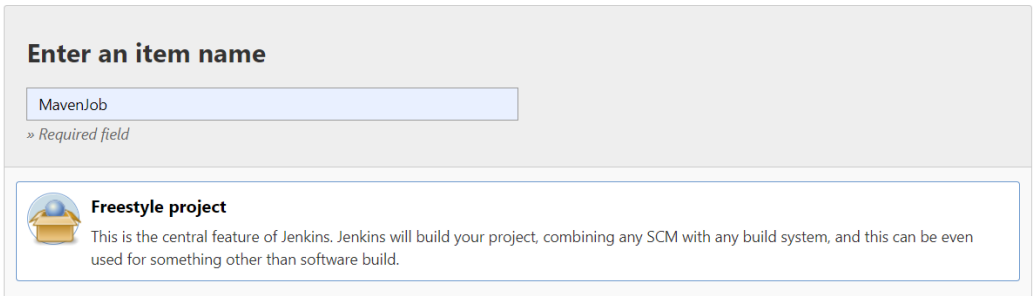


Figure 11.11

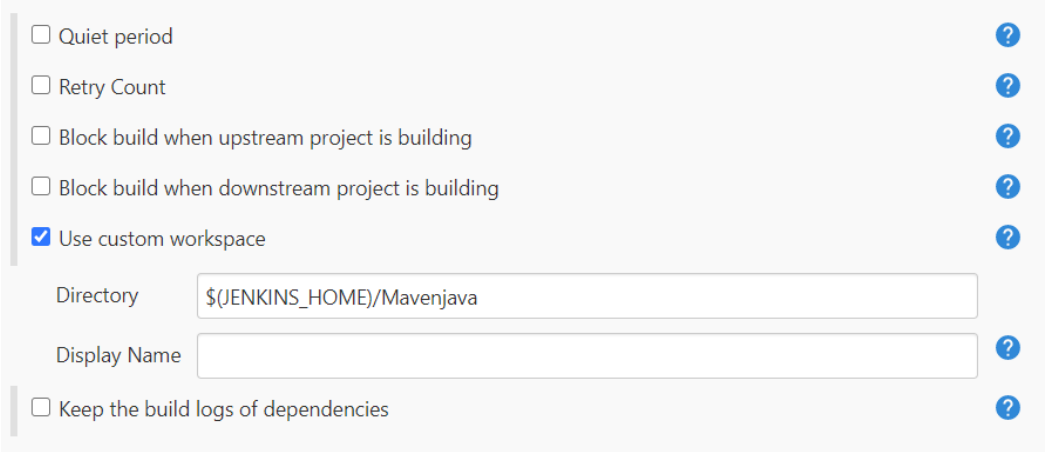


Figure 11.12

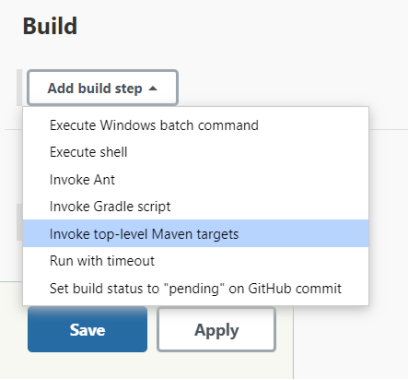


Figure 11.13

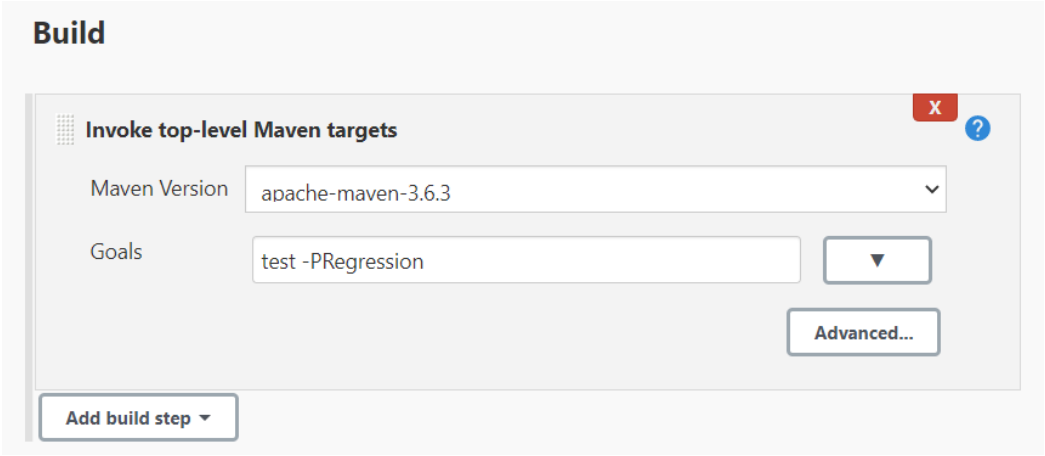


Figure 11.14

```
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/common-java5/2.20.1/common-java5-2.20.1.jar (50 kB at 147 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/surefire/surefire-testng/2.20.1/surefire-testng-2.20.1.jar (44 kB at 114
kB/s)
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running TestSuite
BrowserAUtomation
elementsUi
deleteTwitter
postJira
IOSApps
NativeAPpAndroid
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.387 s - in TestSuite
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 6, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  11.812 s
[INFO] Finished at: 2020-08-11T15:38:34+03:00
[INFO] ------------------------------------------------------------------------
Finished: SUCCESS
```

Figure 11.15

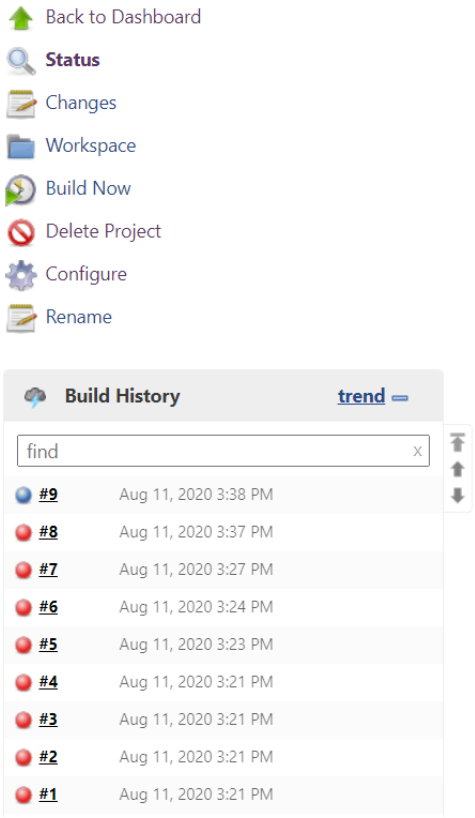


Figure 11.16