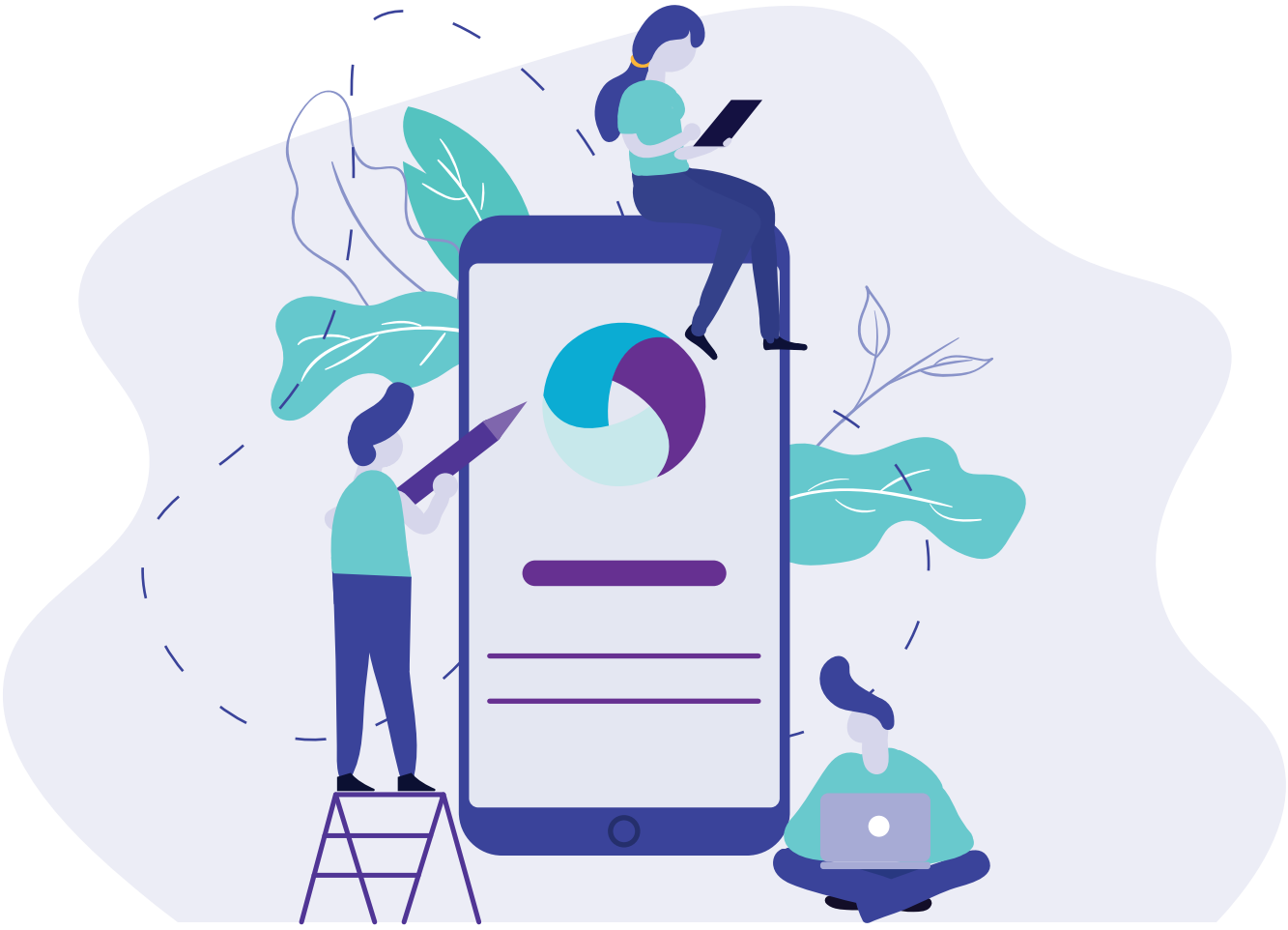
pCloudy.com

Start To End Guide
Mobile Test Automation
With Appium



CONTENTS

INTRODUCTION TO APPIUM

Appium is an open source mobile app UI testing framework. You can test all types of mobile apps and perform automation test on physical devices as well as on emulators and simulators. Appium does not have a dependency on mobile device OS and It supports cross-platform app testing as a single API works for both Android and iOS. Appium supports many popular languages like C, PHP, Python, C#, Java, Ruby, JavaScript, etc.

How Appium works?

When Appium is installed then a server is setup on your machine that exposes the REST API. It receives a command request from the client and executes that command on Android or iOS mobile devices. Then it responds back with an HTTP response. It uses mobile test automation frameworks like Apple instruments or UIAutomator2 to drive the UI of apps.

We will learn more about Appium Architecture in detail in the 3rd chapter of this ebook.

APPROACHES TO TEST AUTOMATION

There are two approaches for mobile test automation, Image-based and Object-based approach. Let's understand both in detail.

Image-Based Approach for Test Automation

This technique of object identification is based on the image processing attributes of the objects in the Application Under Test (AUT). Example: Automate user options like "click, type, drag-drop, mouse actions, etc."

Visual verification of the expected output

- *Not dependent on the platform underneath*
- *Can be used to automate emulators as well as a real device.*

Image - based Approach

Merits	Demerits
• Easy to automate • Can accurately test GUI and rendering of applications • Useful for automating multiple devices without getting into details of each platform technology • Imperative for end-user experience	• Highly dependent on the resolution • Cannot run in background • Slower than object level recognition as it requires scanning of complete screen

The object-based approach of test automation

This technique of test automation is based on recognizing the nativity of the objects in AUT. This nativity reorganization process for each individual object in the application is carried out using different attributes that are assigned to the object.

It is used to extract the application object identifier with its properties from the actual native operating system source code, just like the developer used. This is an accurate and fast method to recognize the buttons, lists and other objects used by the application.

One drawback of the object-based approach is that the recognition of the individual attributes

of the object involved restricts these techniques ability to function in test scenarios that require third-party application access. This reduces the automation coverage of utilizing this technique.

Object - based Approach

Merits	Demerits
• *100% accuracy in object finding* • *Faster than image-based approach* • *Supports multiple languages*	• *Does not capture GUI defects* • *Languages and implementation are different for different platforms* • *It is completely functional, does not defect UX defects*

Which approach should we choose?

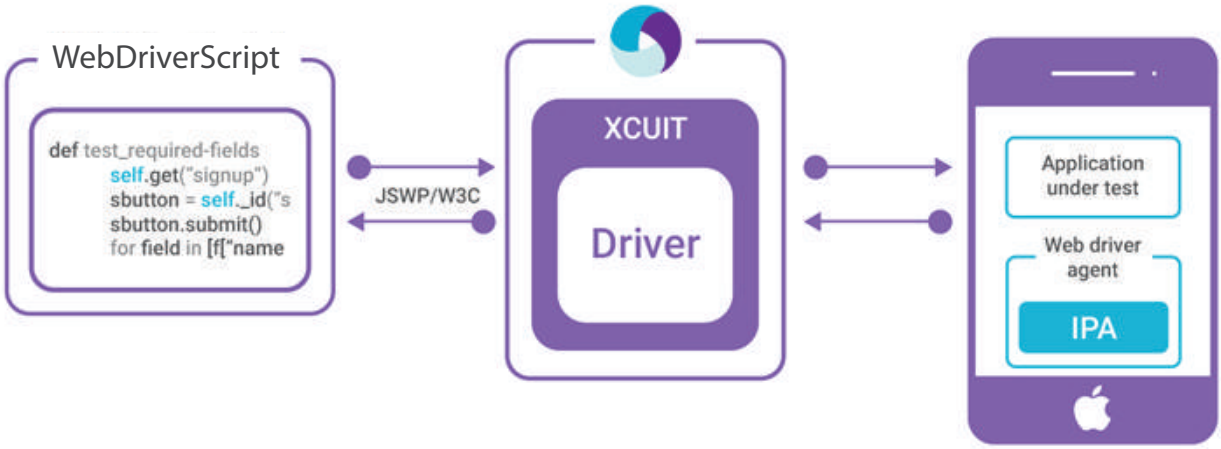
As we have seen, both approaches have their pros and cons. To get better results, you can merge both approaches and think about devising a hybrid test automation solution.

The combination of OCR (image based) and native (object-based) approach allows users to build a single script that will be portable across different devices. It will make your automation robust and efficient and allows the users to confidently detect the relevant native and GUI defect within the mobile application.

APPIUM OVERVIEW AND ARCHITECTURE

Appium is an HTTP server that manages Web Driver sessions. On iOS devices, Appium proxies command to a UI automation script running on Mac Instruments environment. Apple provides an application called instruments which are used to do a lot of activities like profiling, controlling and building iOS apps. It also provides an automation component where you can write some commands in JavaScript which uses UI Automation APIs which interact with the app user interface. Appium uses these same libraries to automate iOS apps.

Appium Architecture for IOS

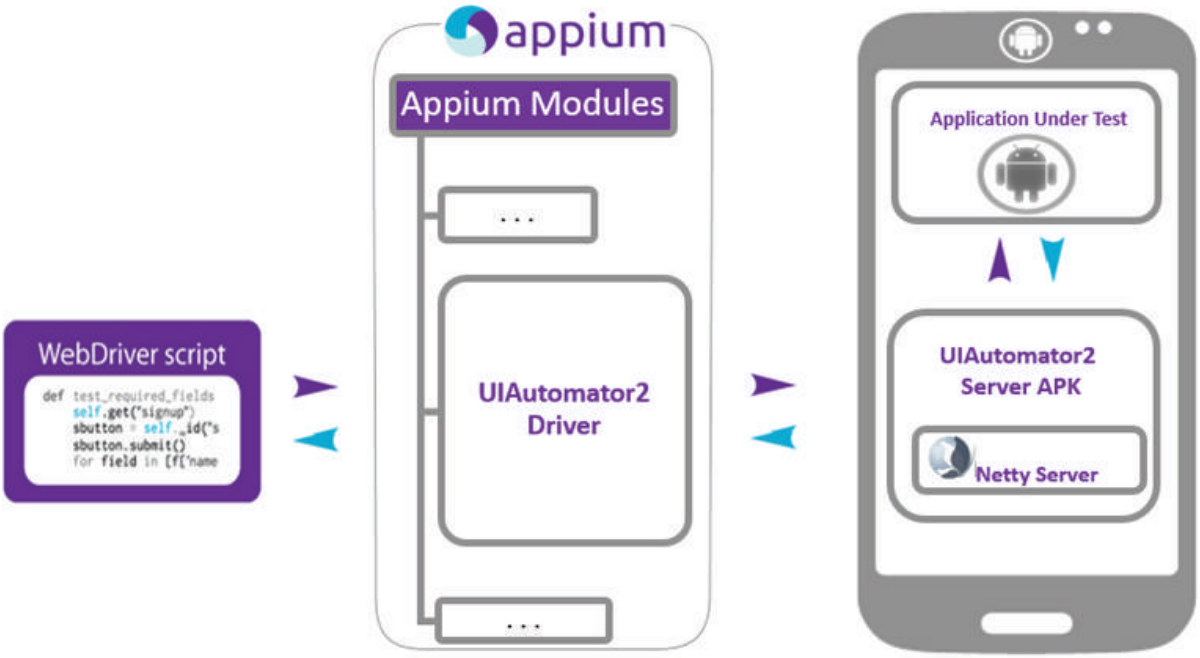


The Webdriver Script sends a command in the form of HTTP (JSWP) to the Appium Server. Then the Appium Server will then decide as per the request which driver should be initiated. So, in this case, the Appium Server will initiate the XCUITest driver and it will pass the request to the WebdriverAgent which is an IPA (WebdriverAgent.xcproj) developed by Facebook. WebdriverAgent is responsible to send the command to the Application Under Test (AUT) to carry out the actions in the app. Then the response will be sent to the Webdriver Script through the Appium server.

Only iOS 9.3 and above version are supported by the XCUITest Driver. You can find all the capabilities for XCUITest Driver in the link mentioned below.

https://github.com/appium/appium-xcuitest-driver.

Appium Architecture for IOS



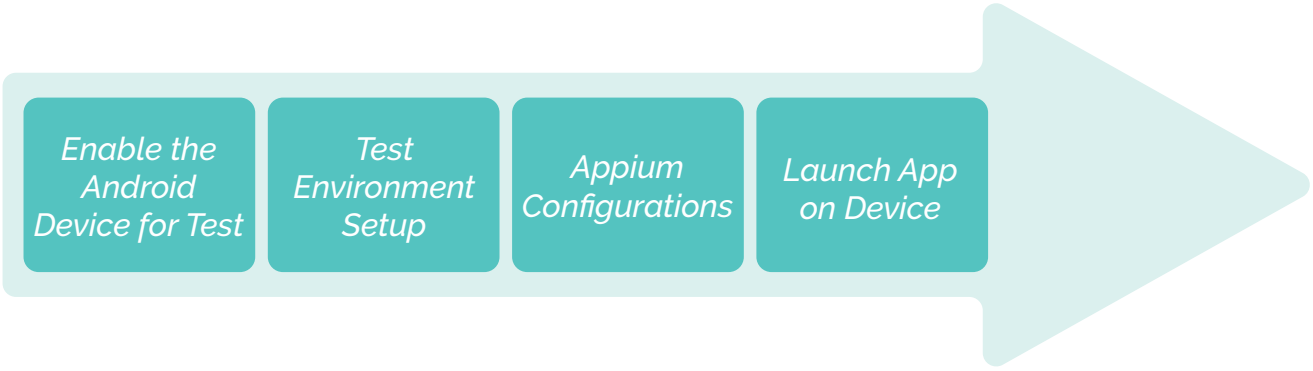
The situation is very similar in the case of Android where Appium proxies command to a UIAutomator2 test case running on the device. UIAutomator2 is Android's UI automation framework which supports running JUnit test cases directly into the device into the command line. It uses Java as the programing language but Appium will make it run from any of the web drivers supported languages.

Appium Philosophy

1. *Test the same app you submit to the marketplace.*
2. *You shouldn't be locked into a specific language or framework to write and run tests.*
3. *Use a standard automation specification and API.*
4. *Build a large and thriving open source community effort.*

APPIUM SETUP FOR WINDOWS

There are four steps required to setup Appium, they are enabling the Android device for test, Test environment setup, Appium Configurations, Launch the app on the device.



Pre-requisites for using Appium

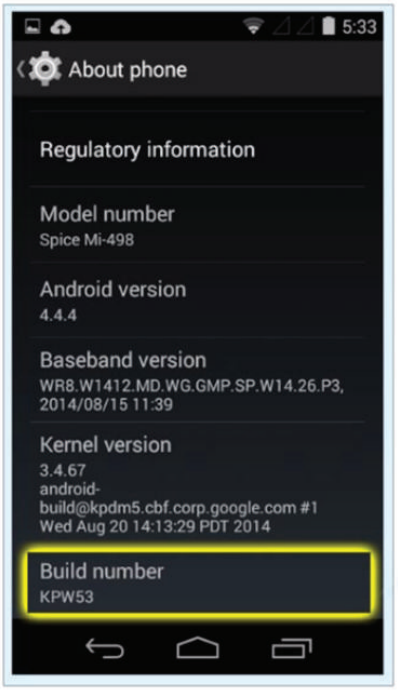
- *An Android device with OS 4.2+*
- *AUT(Application Under Test) file (.apk)*
- *Phone USB Drivers*
- *Java (JDK)*
- *Android Studio (SDK)*
- *Eclipse*
- *Selenium Standalone JAR*
- *Appium Java Client*
- *Appium for Windows*
- *.NET Framework 4.5*

Android Developer Options in device

Every Android smartphone contains a secret set of Android developer options which are used by app developers who need additional functions to test their apps they are making for Android devices. It allows you to enable debugging over USB, capture bug reports on your Android device and show CPU usage on the screen to measure the impact of your software.

To enable developer option in the phones to go to the settings, click on the about phone options and click on the build number 7 times to enable the developer options.

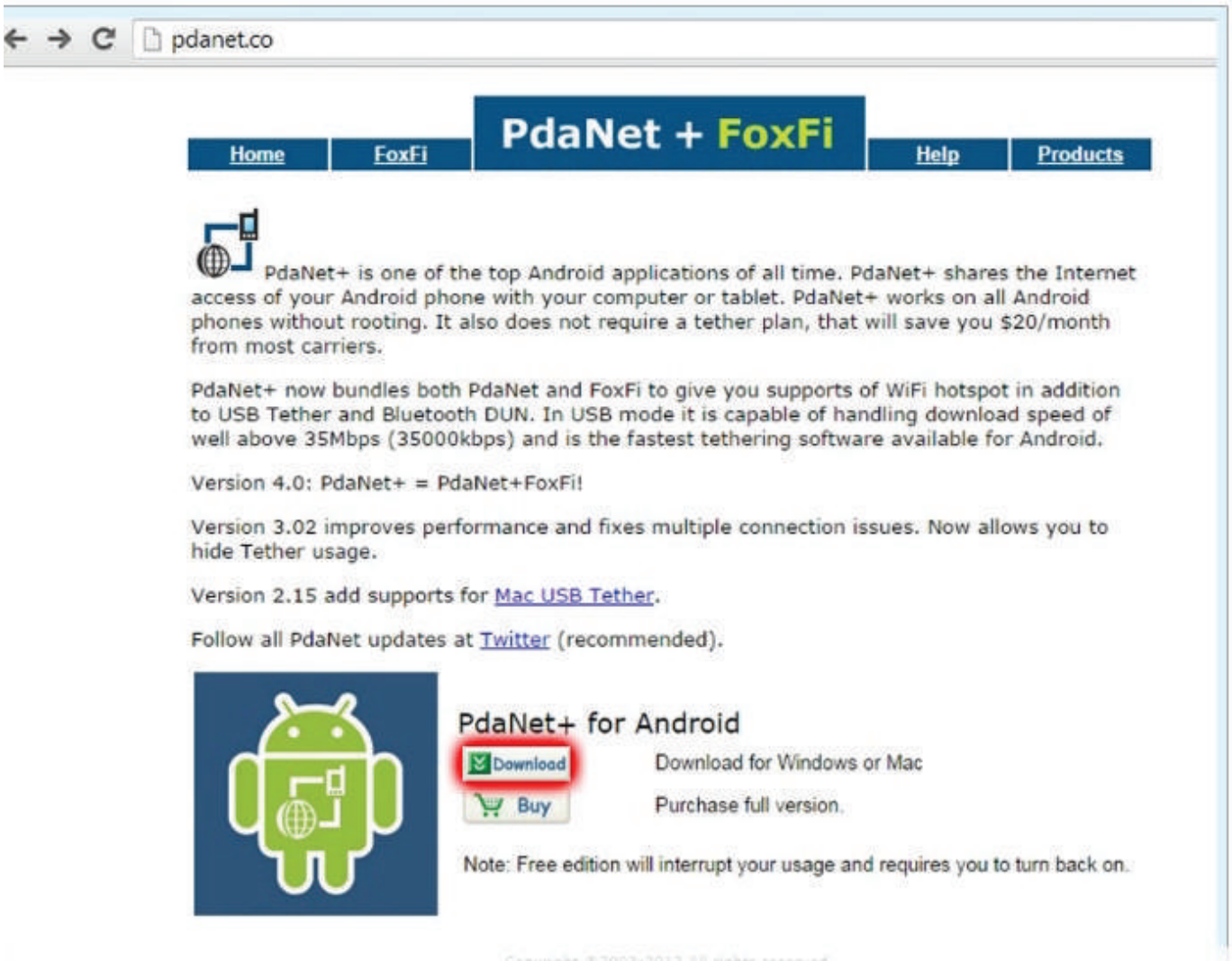
Doing so will display a toast message for enabling the developer option. the current message appears if the developer message is already enabled on the phone.



Phone Drivers

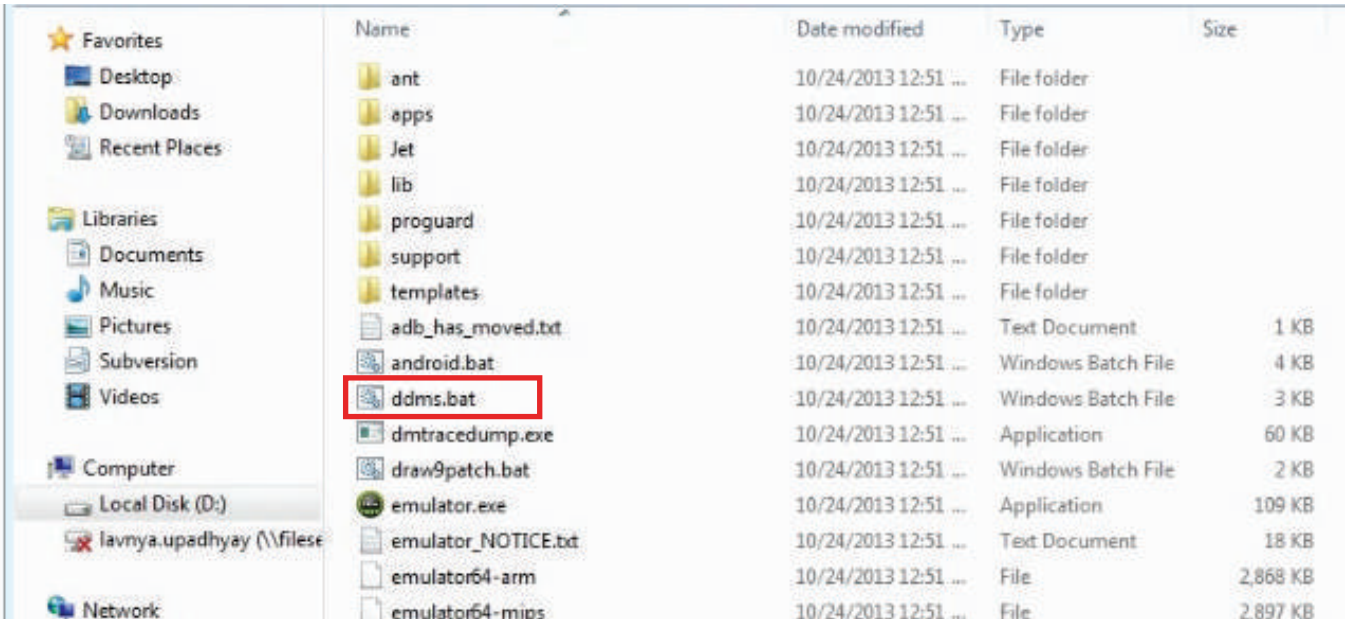
The phone can communicate with the PC only if there are proper drivers installed for the USB cable. Each phone manufacturers provides its own drivers for the phone. PDA net is a driver which works with all the Android devices. It makes sure that your phone is detected in DDMS.

Open pdanet.co and go to download screen and download the latest version to install it on your PC.

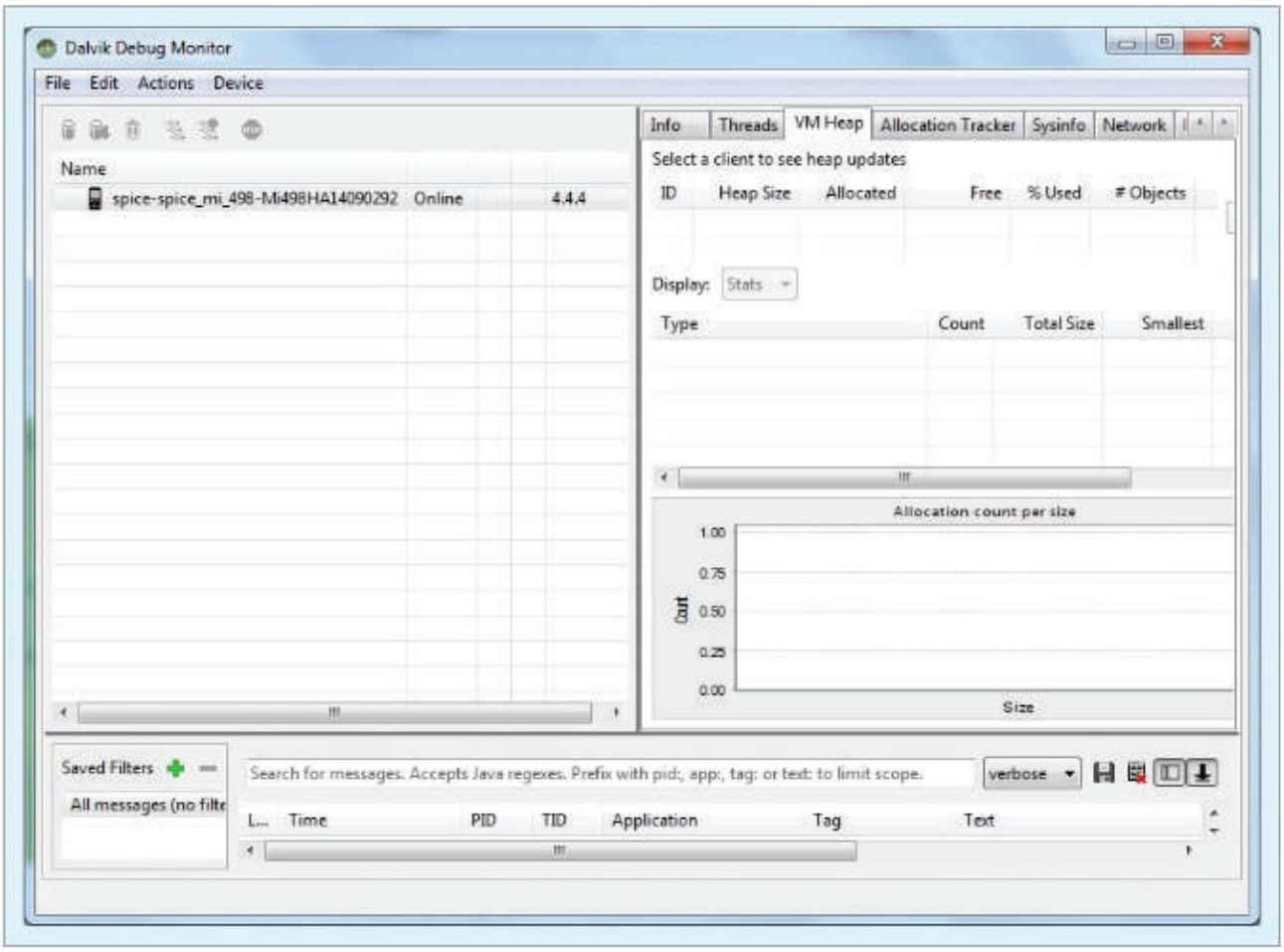


Let's verify if the phone is prepared.

The ddms.bat file is present in the Android SDK tools folder.



*Double click on the file and you will see the **Dalvic debug monitor** window.*

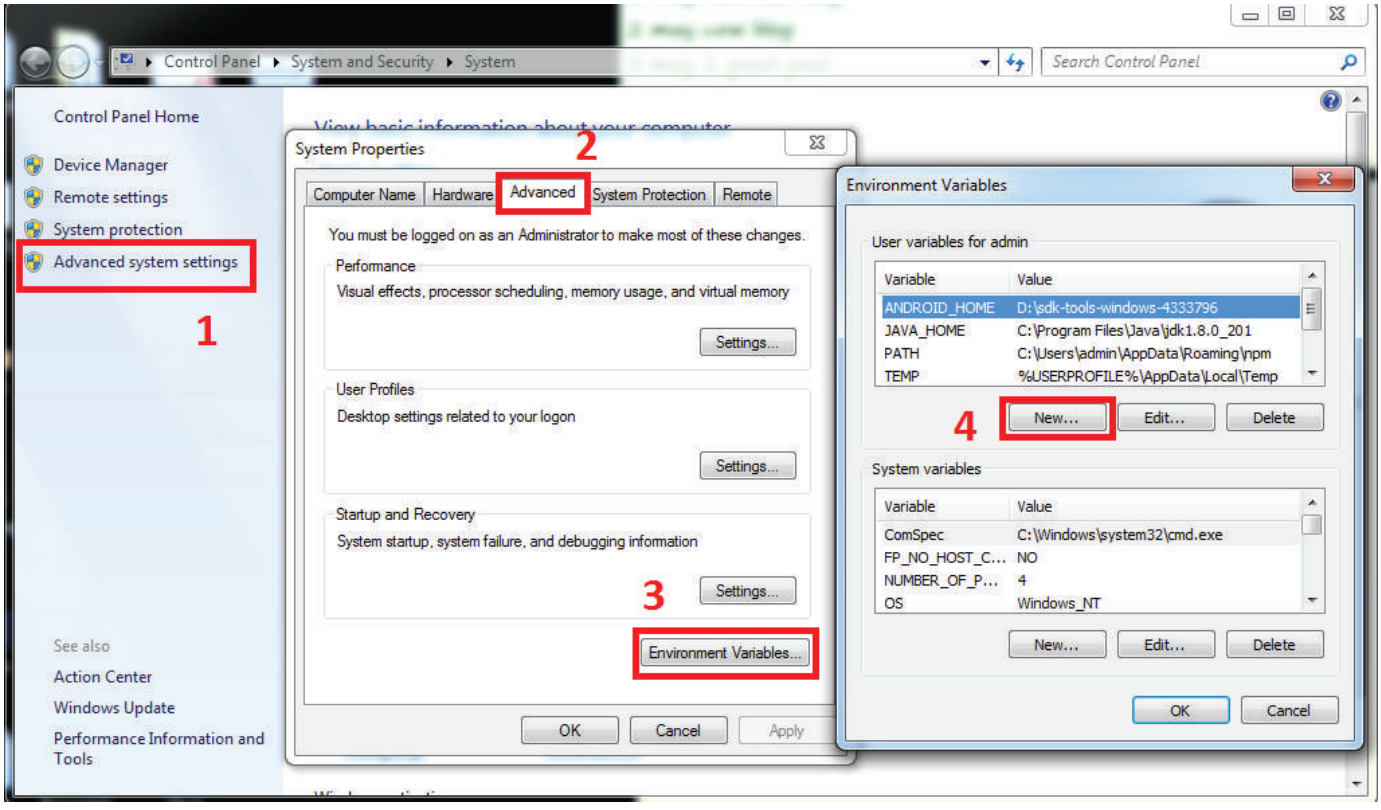


If your DDMS option is enabled i.e. if you have installed the PDA net software and connected the device to the machine, that device should be detected in this monitor. Now let's move on to the second step.

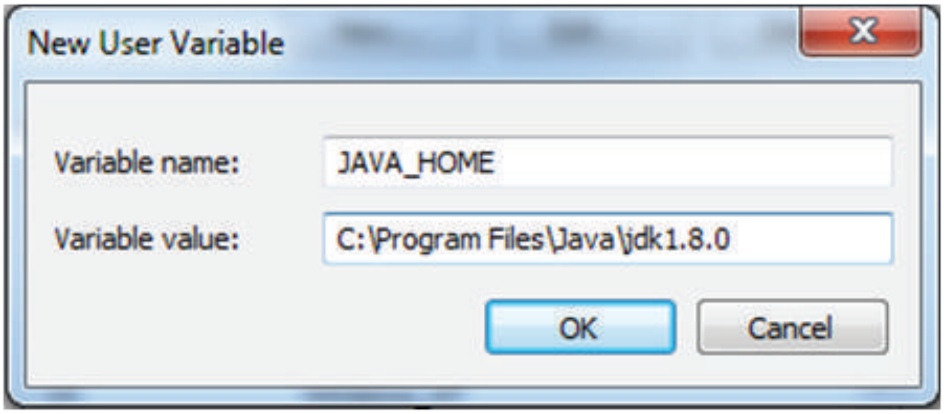
Test Environment Setup

First, you need to download JDK from the Oracle website.

Once downloaded, install it to your machine. Now you need to set the Java installation path in your ***Environment variable****. Right click on* ***computer*** *option in the* ***start*** *menu and select the* ***properties*** *option. Select* ***advanced system settings*** *and then select the* ***Environment variables*** *option in the* ***Advanced*** *tab.*

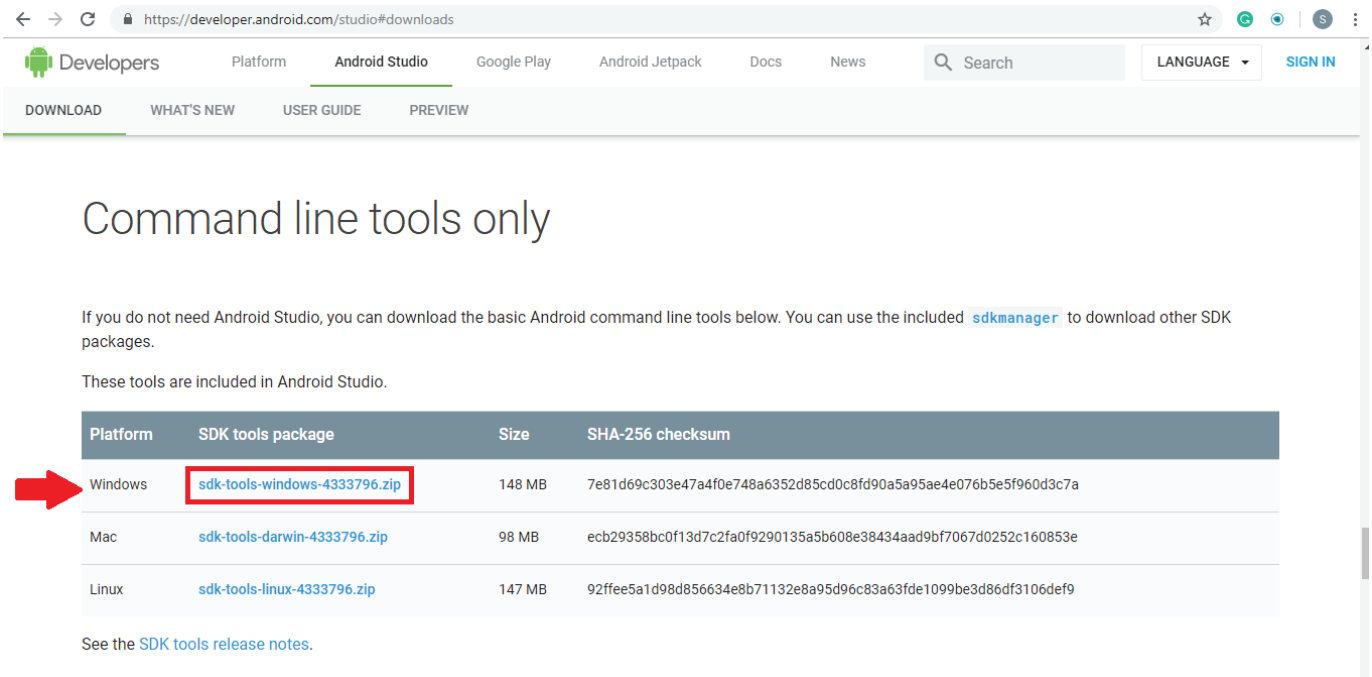


Then select the new option and enter the new variable name as JAVA_HOME.



Set the path of JDK to variable value and then click OK.

To get an Android emulator you need to go to www.developer.android.com\studio#downloads and Scroll down to ***Command line tools only*** *section to download the zip file of SDK tools package for Windows.*



Don't download the EXE file. Once the SDK file is downloaded, copy it to the C drive, make a new folder and extract the zip file in the new folder.

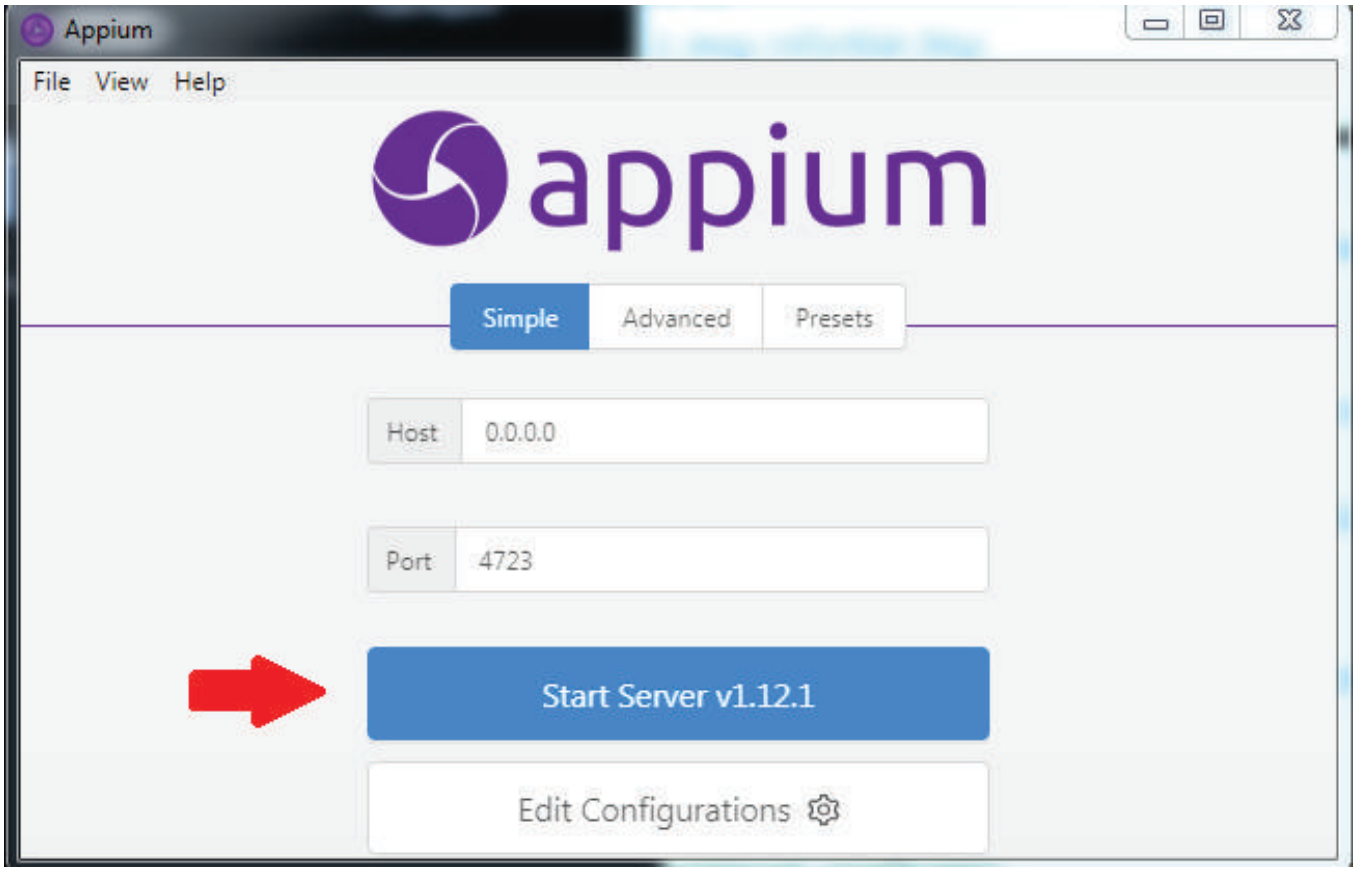
Now you need to search for SDK manager in the bins folder and open SDK manager. Once the SDK manager window is open, go to tools and then get into the bin folder and in the address bar type cmd and hit enter. Now in the command prompt type ***SDK manager "platform-tools" "platforms; Android-28*** *and then hit enter. So now you will be able to see Platform tools and Platform*

folders in now folder you created for the extracted files.

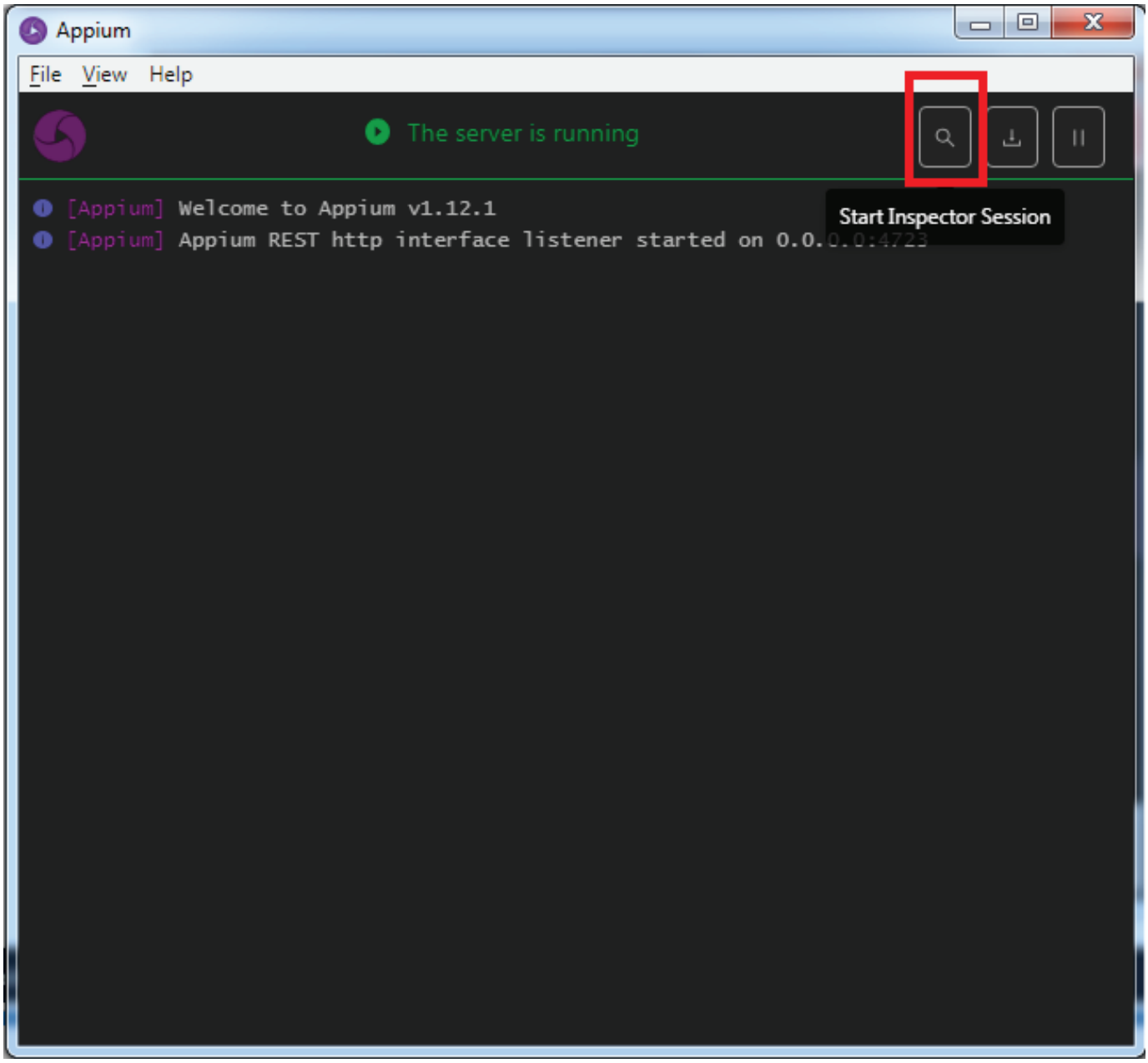
Click on platform tools and copy the address bar text and then again go to an environment variable. Then go to the path, a new window will pop up, make a new path and enter the copied text there and then click on OK.

To download Appium go to www.appium.io and click on the ***download*** *button.*

Under the latest version, click on the Appium-windows-1.12.1.exe file. Once the file is downloaded, open the appium.exe file. Click on the ***Start Server v1.12.1 button.***



*Then in the server window click on the **Start inspection session** icon at the top right corner.*

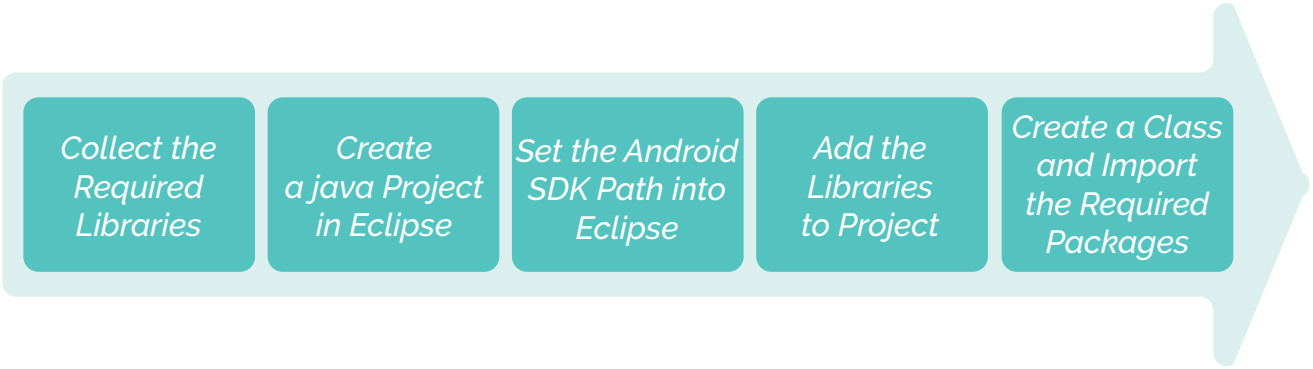


FIRST APPIUM SCRIPT - ANDROID

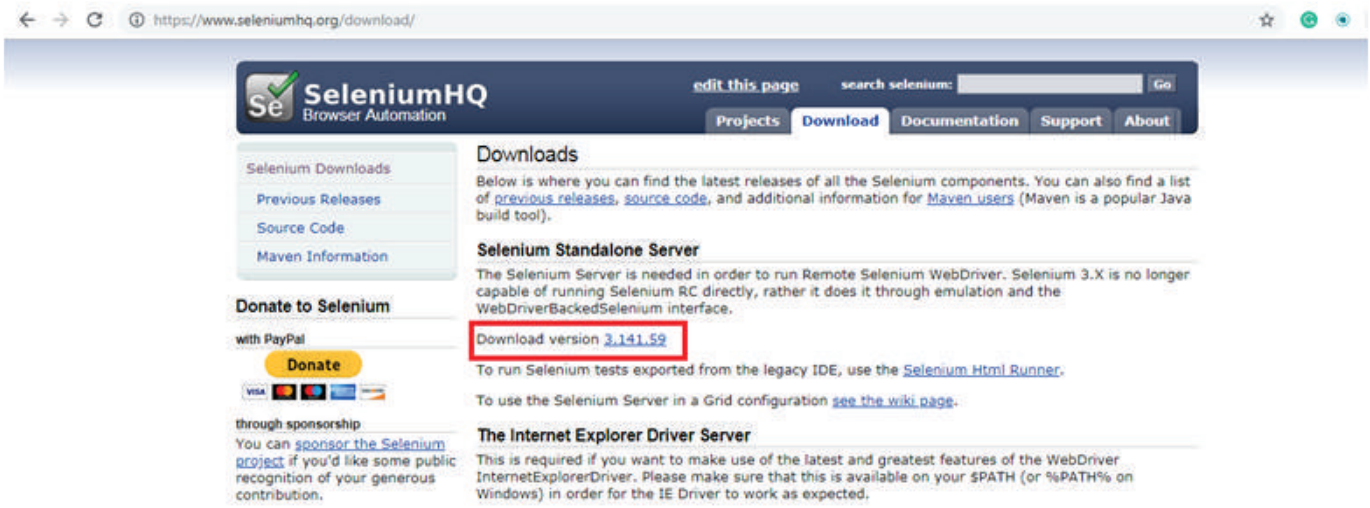
From this section onwards, we will start handling the application by writing the Java code. We will require the following software:

- *Eclipse*
- *Selenium standalone Jar*
- *Appium Java Client*

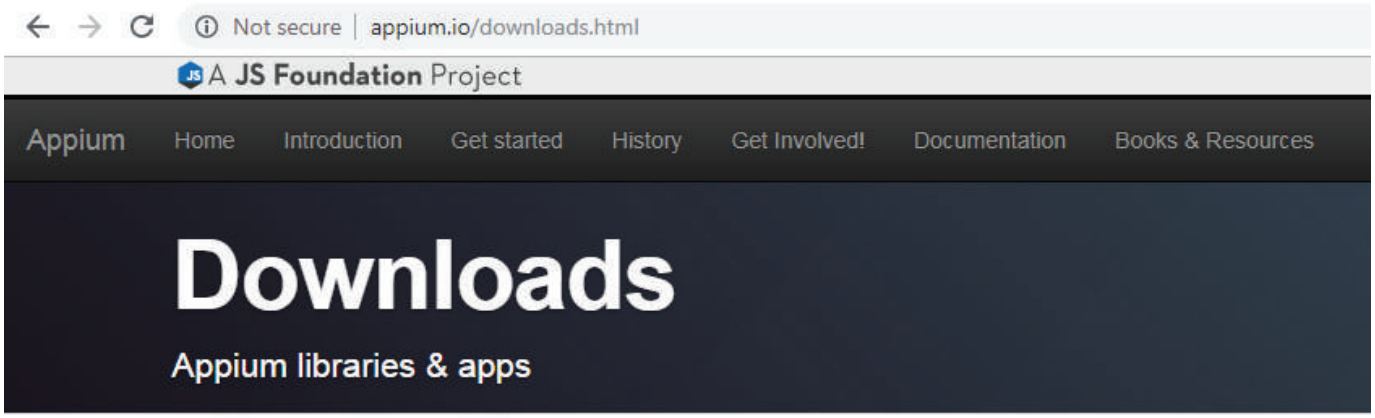
This process consists of five steps as depicted in the picture below.



First, we need to collect the Selenium Standalone JAR and Appium Java Client Libraries. To download the Selenium standalone JAR file, go to seleniumhq.org/download then click on the Download version.



For Appium Java client you need to go to appium.io/downloads and download the libraries for the selected language.

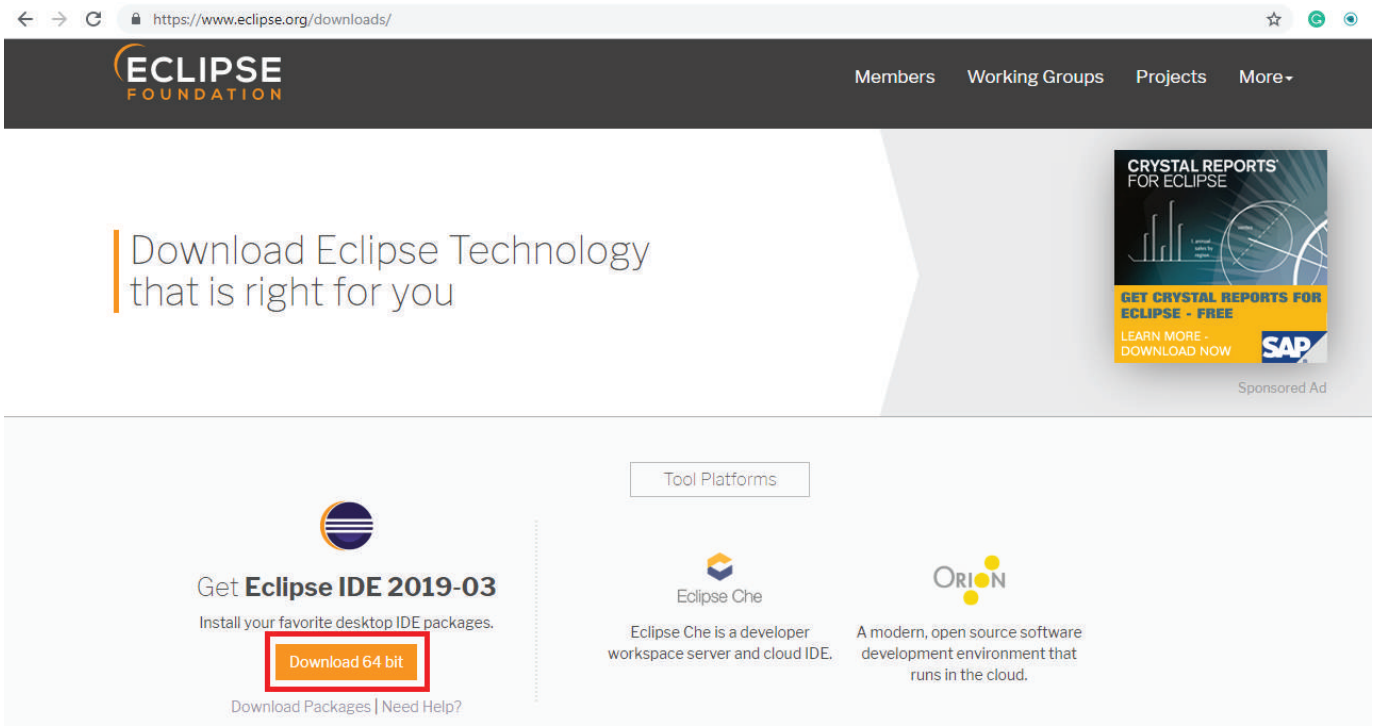


Appium Client Libraries

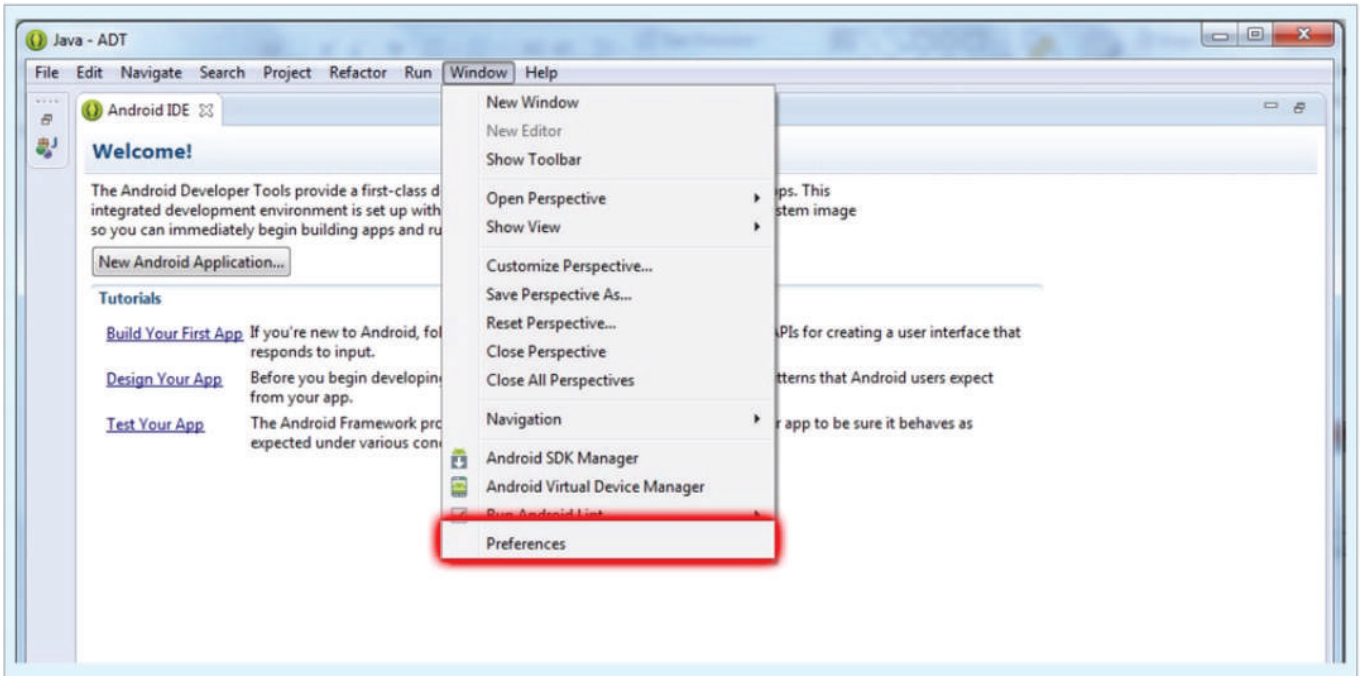
Appium has language bindings for:

- Ruby
- Python
- Java
- JavaScript
- PHP
- C#
- RobotFramework

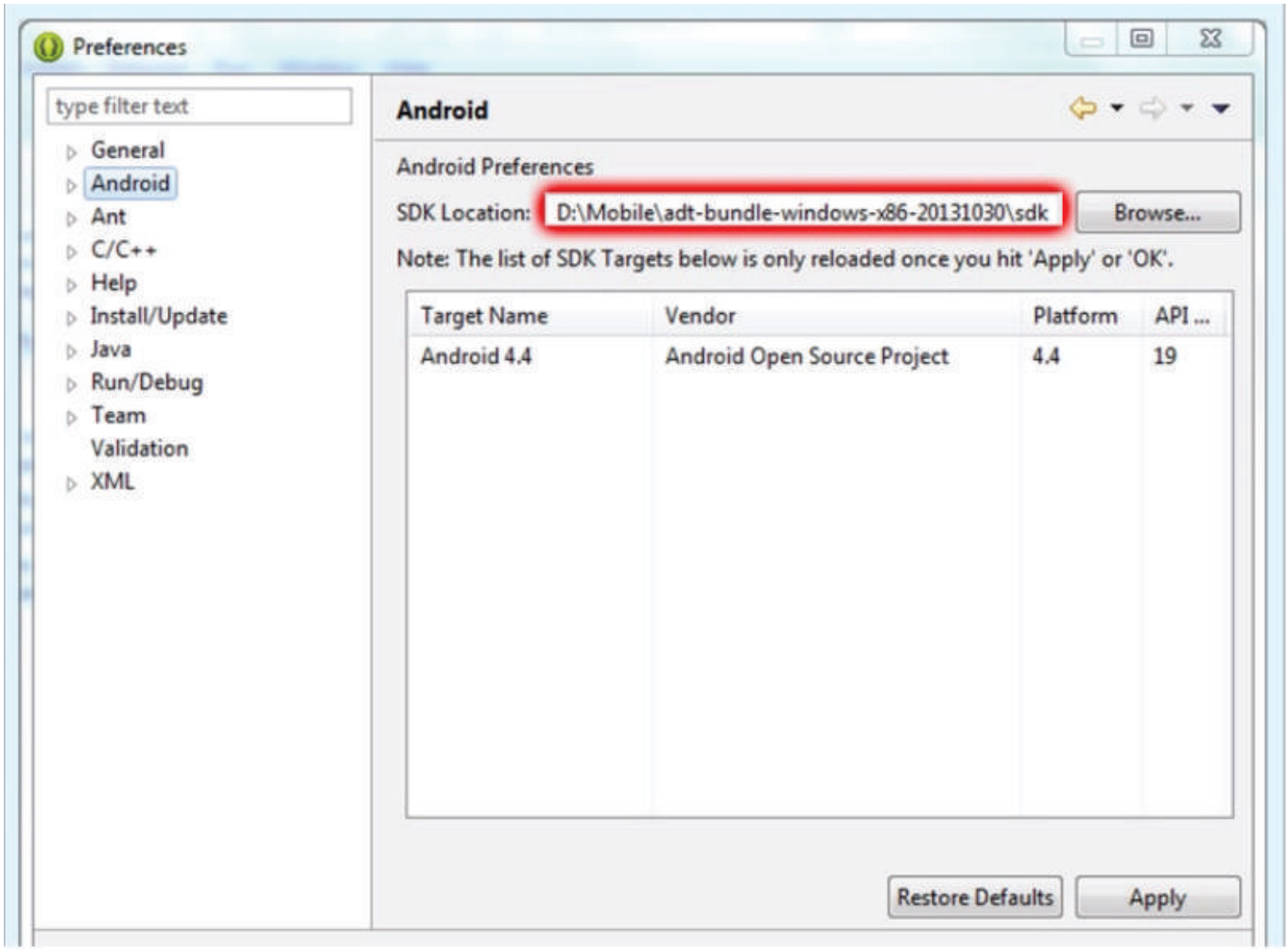
Now to create a java project download Eclipse from eclipse.org/downloads. Launch Eclipse and select the workspace location.



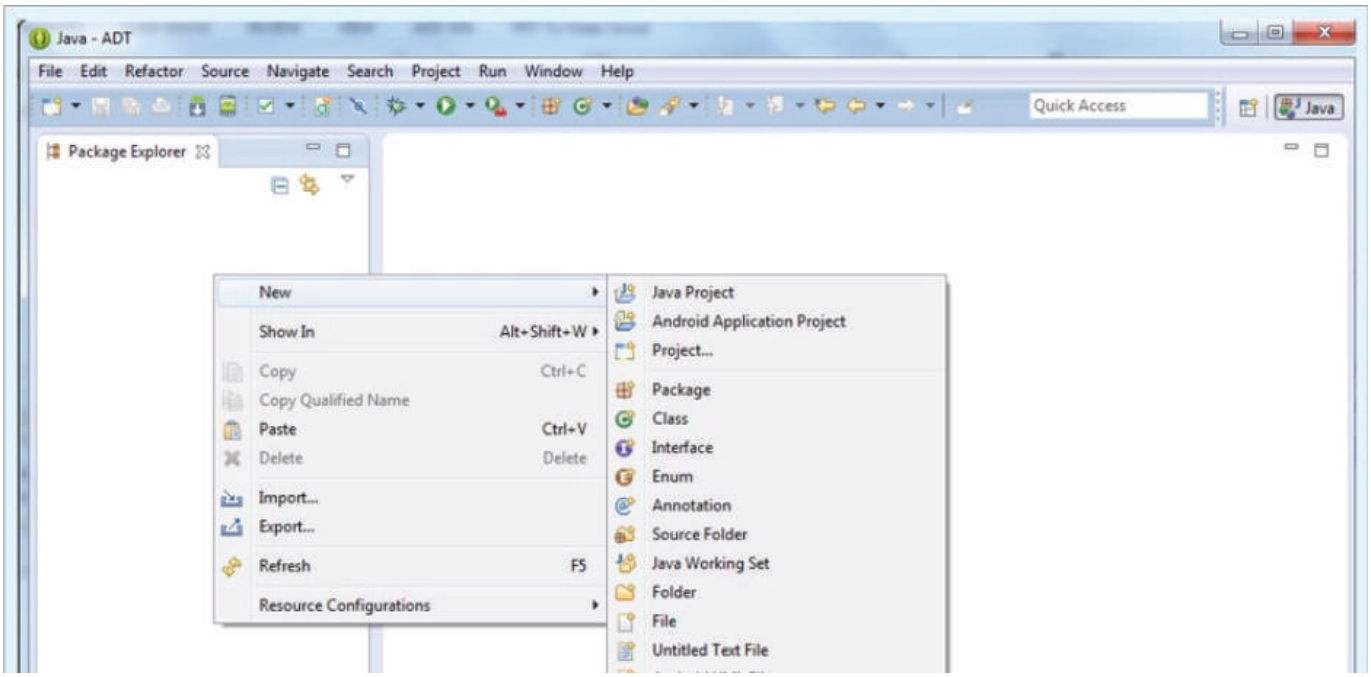
*To set the Android SDK path into Eclipse, click on the **Windows** tab in the menu bar and select **Preferences** in the drop-down list.*



*Then select the **Android** option and browse your Android **SDK location** and click on **Apply.***

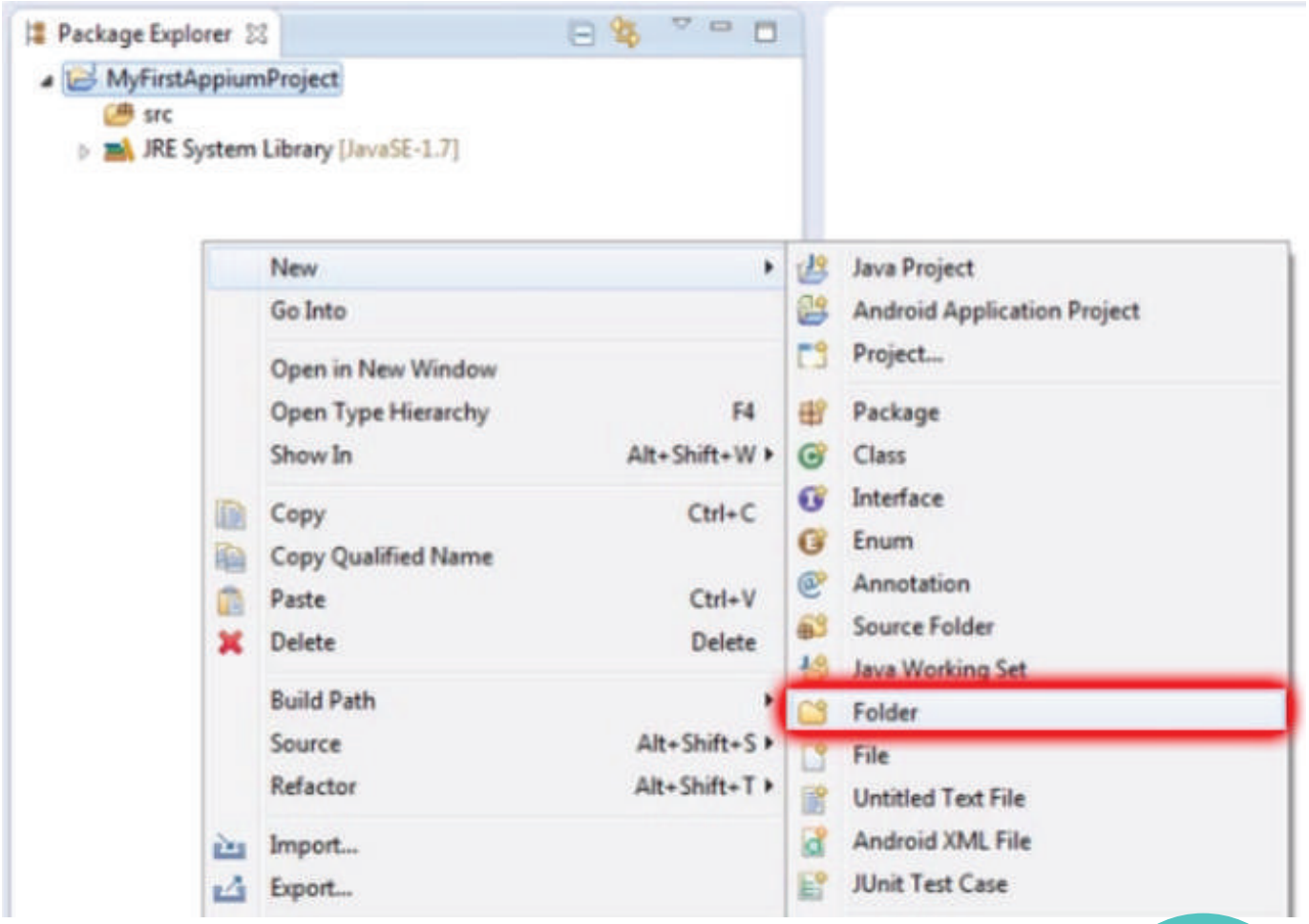


*Launch Eclipse and right click on **Package Explorer**. Then select a **new Java Project**.*

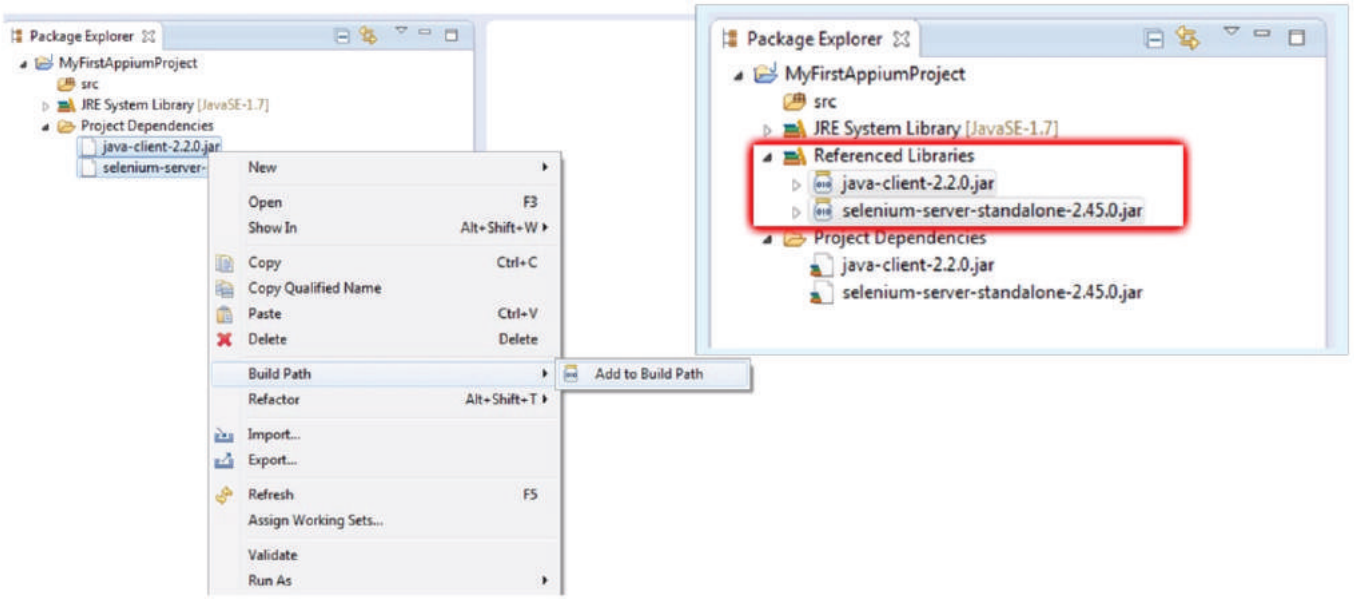


Enter a project name and click finish.

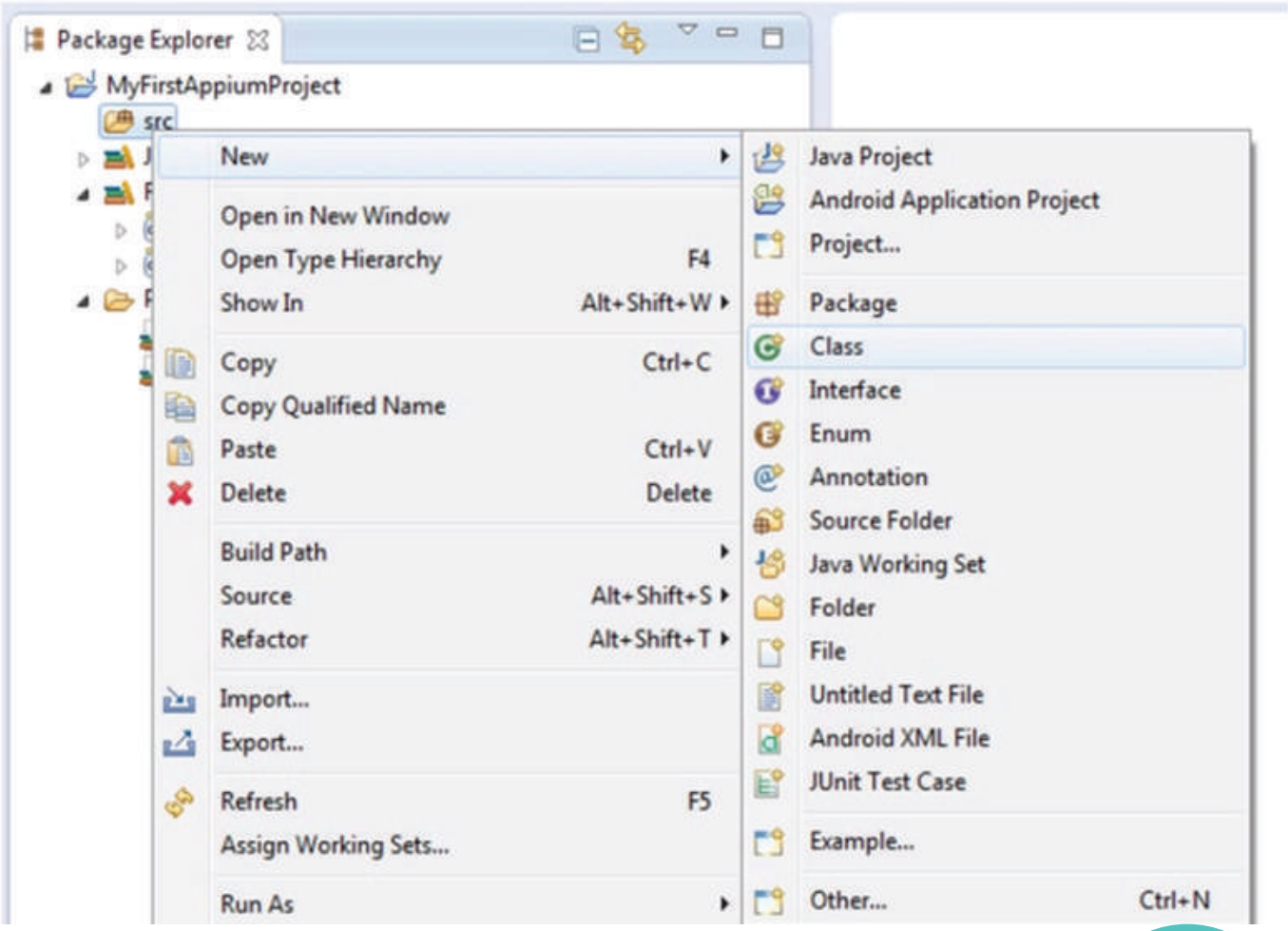
Once the project is created, add a folder to the project, for the project dependency files [Selenium Standalone] and [Appium Client Library] which you have downloaded.



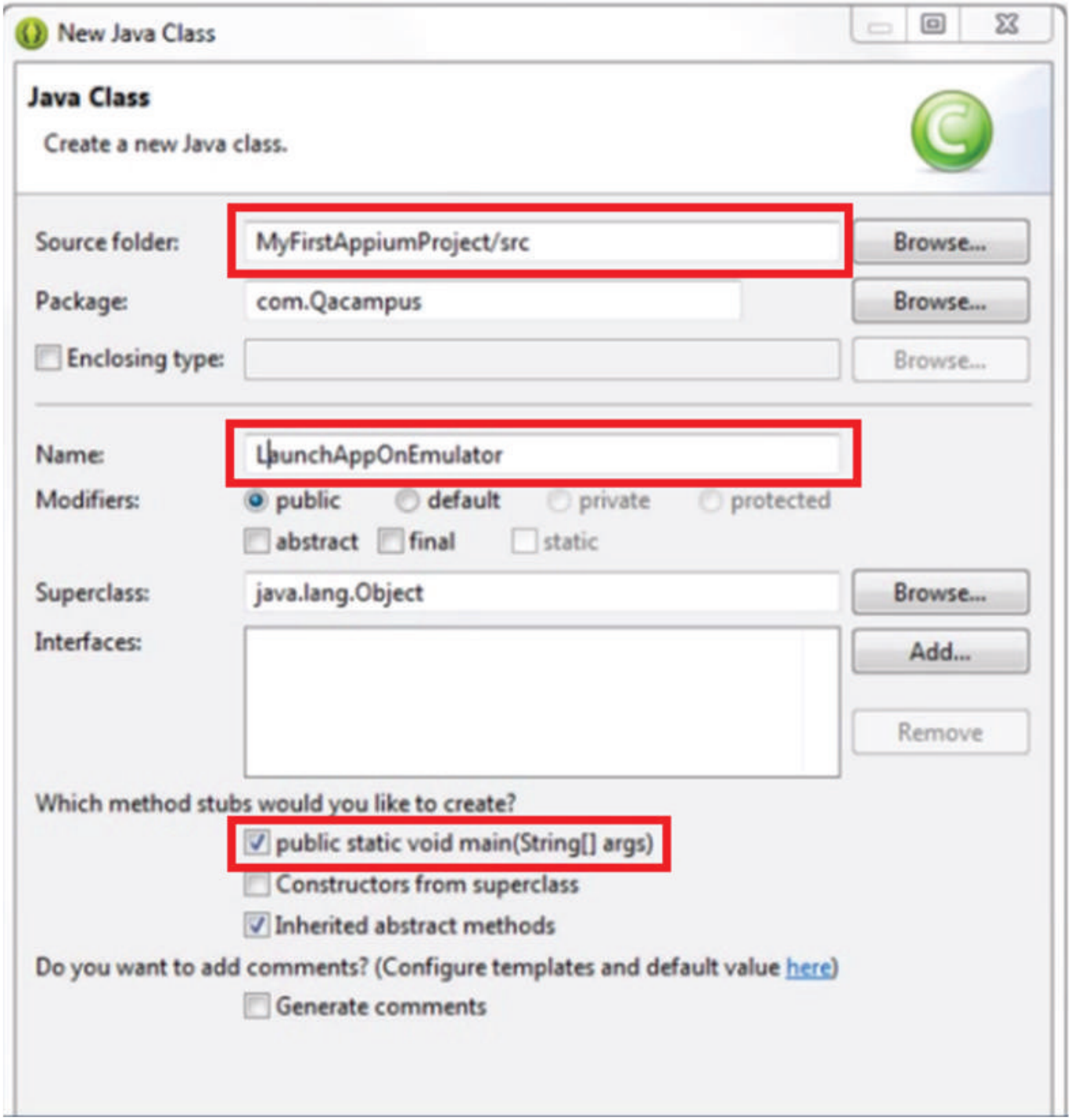
Copy the downloaded file into the newly created project dependencies folder. Select both files and right click. Then select the add to ***Build path*** *option and then* ***Add to Build Path****. Thus both classes have been added to your project reference libraries.*



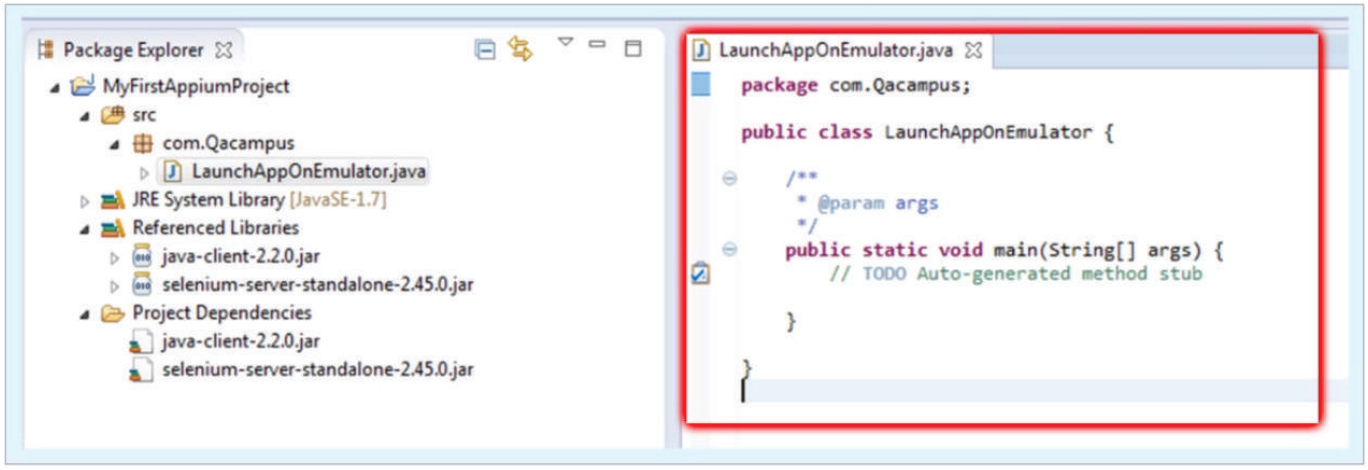
Create a class and import the required packages. Now right click on the ***src*** *folder, hover the mouse over the new option and select the* ***Class*** *option.*



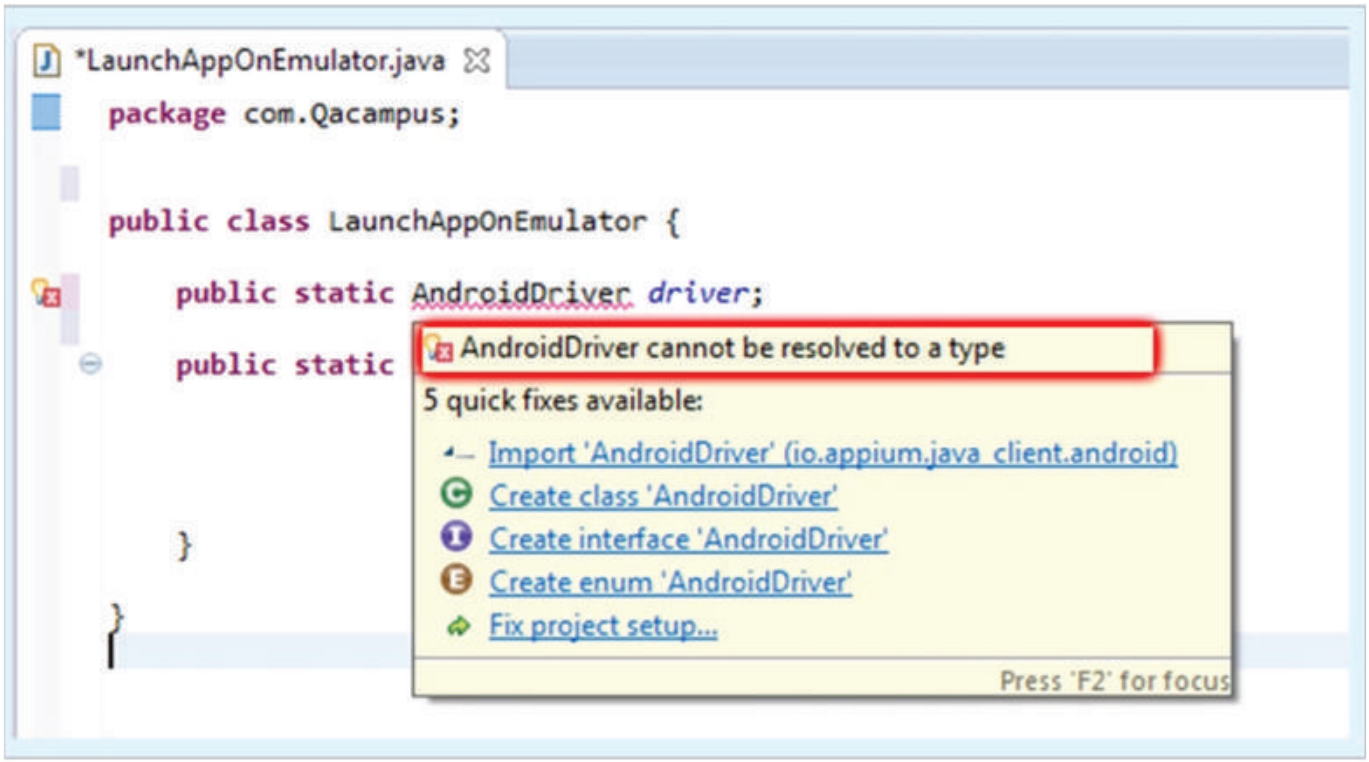
Provide a package name, the name of class then select the main method check box.



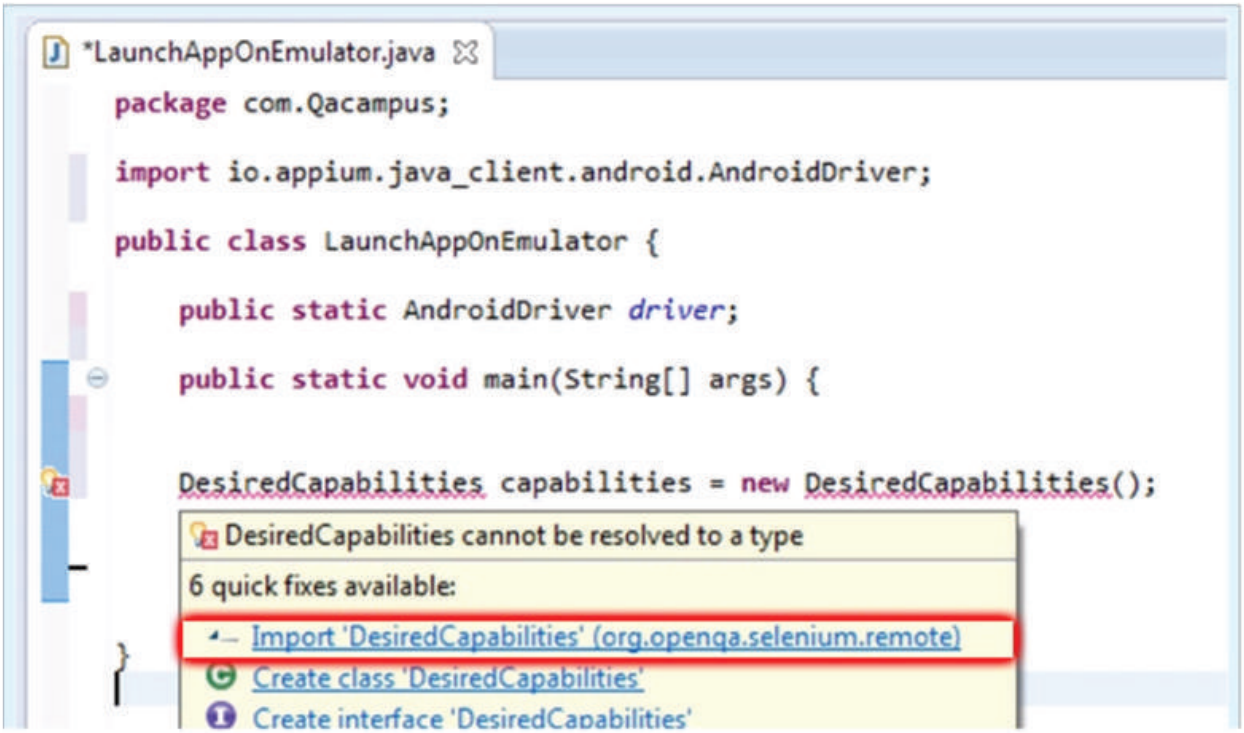
A java file within the package is created in Package Explorer panel. An auto-generated the main method is generated on the right panel.



*You need to define a public class variable of **AndroidDriver** as your first line of code. You will now see an error for an android driver. Now hover your mouse over the error, you will get a list of quick fixes. Select **Import 'AndroidDriver'.** Once you select the import package option, the android driver package will be imported and the error will be removed.*



Create an object of DesiredCapabilities. Again you will get an error for DesiredCapabilities. Hover the mouse over ***DesiredCapabilities*** *and then select* ***Import 'DesiredCapabilities'*** *from the quick fix list.*



Once you select the import package option, the DesiredCapabilities package will be imported and the error will be removed.

Now set the DesiredCapabilities and also provide the package of an application and the name of application launcher activity.

Now you need to instantiate the Android driver.

To do this we need to have two parameters, the first is the ***Appium server address*** *with the* ***port number*** *which it is running and the* ***Capabilities.***

```
package com.Qacampus;

import org.openqa.selenium.remote.DesiredCapabilities;

import io.appium.java_client.android.AndroidDriver;

public class LaunchAppOnEmulator {

public static AndroidDriver driver;

public static void main(String[] args) {

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setCapability("deviceName", "Mi498HA14090292");
capabilities.setCapability("platformVersion", "4.4.2");
capabilities.setCapability("platformName", "Android");
capabilities.setCapability("appPackage", "com.bsb.hike");
capabilities.setCapability("appActivity", "com.bsb.hike.ui.HomeActivity");

driver = new AndroidDriver(new URL("http://127.0.0.1:4723/wd/hub"), capabilities);

}
}
```

Hover mouse on URL and import URL from Java.net. You will get an error for complete new URL section, hover mouse on the error and select ***Add throws declaration.*** *By doing so exception has been added into your main method.*

Now start the Appium server and connect a device to the system. Now return to Eclipse and execute the code. This will launch the app in the device.

In the code, we have declared the class as public so that we can access it anywhere inside out test.

Diving Deep into the Code

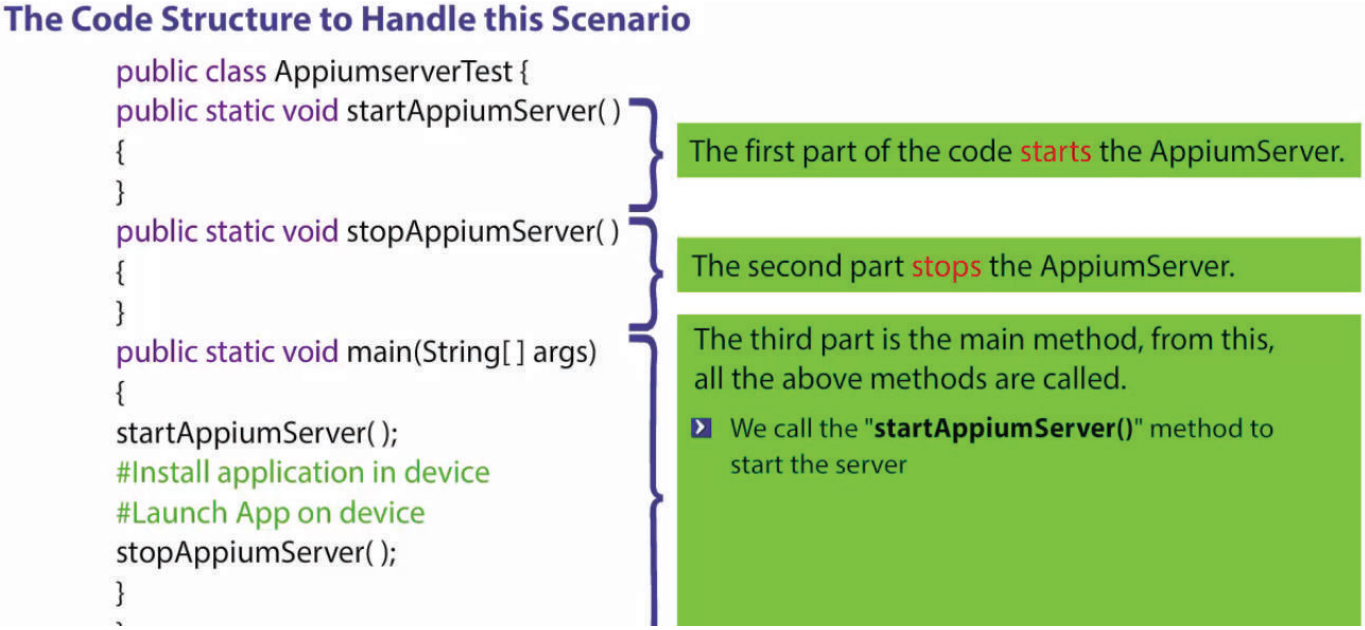
```
public class LaunchAppOnEmulator {

public static AndroidDriver driver;

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

DesiredCapabilities capabilities = new DesiredCapabilities();

capabilities.setCapability("deviceName", "Mi498HA14090292");
capabilities.setCapability("platformVersion", "4.4.2");
capabilities.setCapability("platformName", "Android");
capabilities.setCapability("appPackage", "com.bsb.hike");
capabilities.setCapability("appActivity", "com.bsb.hike.ui.HomeActivity");

driver = new AndroidDriver(new URL("http://127.0.0.1:4723/wd/hub"), capabilities);

driver.findElement(By.id("com.bsb.hike:id/btn_continue")).click();
driver.findElement(By.name("Phone Number")).sendKeys("8130221546");
driver.quit();
```

The java main method is highlighted in yellow. We created an object of desired capabilities class which you can see in the green box. In the method ***setCapability****, there are two parameters. First is the capability name and second is the capability value as highlighted in blue. Then we created an object of AndroidDriver class highlighted in the black box. The code highlighted in gray is the findElement(By) method which can locate an element on the screen.*

STARTING APPIUM SERVER AND LAUNCHING THE APP FROM CODE - ANDROID

The code structure is segregated into three parts, the first part of the code starts the Appium server, the second part stops the Appium server.

The Code Structure to Handle this Scenario



The third part is the main method which is necessary to execute the class and from this main method, all the above method are called.

Starting Appium from code requires a path of two files which are kept inside the Appium folder.

- ***node.exe***
- ***main.js***

Starting Appium Server

In the code highlighted in red, we have called process class which is a Java class and declared it static.

```
public class AppiumserverTest {

private static Process process;

//Calling the node.exe and appium.js

private static String STARTSERVER = "D:\\Mobile\\AppiumForWindows-1.3.4.1\\Appium\\node.exe D:\\Mobile\\AppiumForWindows-
1.3.4.1\\Appium\\node_modules\\appium\\bin\\ main .js";

//Starting the Appium Server
public static void startAppiumServer() throws IOException, InterruptedException {

//Runtime class this is again a java internal class
Runtime runtime = Runtime.getRuntime();

process = runtime.exec(STARTSERVER);

Thread.sleep(5000);

if (process != null) {

System.out.println("Appium server started");
}}
```

We then created a start server variable and pass the path to node.exe and main.js, highlighted in yellow. In the code highlighted in green, we added both paths into the same variable with spaces and created a method called ***startAppiumServer()*** *which takes care of the Appium server startup process. Next step is to create an object of* ***Runtime*** *class which is again a java class and call the method* ***getRuntime()****, highlighted in blue. We will pass the variable "STARTSERVER" into* ***runtime.exec()*** *method. It will start the Appium Server.*

Once the ***process*** *is started, we have to store the current state of the process into this variable. We will give a sleep time of 5 seconds as it takes time to start the process. what we are trying to validate here is, if the process is not* ***null****; it means that the process is started as highlighted in the gray box.*

Stop Appium Server

Here we need to create another method named ***stopAppiumServer()****. We again verify the state of the process and if it is not null, call a method of process class* ***destroy().***

```
public class AppiumserverTest {


private static Process process;

//Calling the node.exe and appium.js

private static String STARTSERVER = "D:\\Mobile\\AppiumForWindows-
1.3.4.1\\Appium\\node.exe D:\\Mobile\\AppiumForWindows-
1.3.4.1\\Appium\\node_modules\\appium\\bin\\appium.js";

//Stopping the Appium Server
public static void stopAppiumServer() throws IOException {

if (process != null) {
process.destroy();
}
System.out.println("Appium server stopped");
}
```

So now we know how to start and stop the Appium server from code. Let's try to install and launch an app on a mobile device. Let's see how we can install a .apk file on an Android device and launch it automatically without providing the path in Appium Server.

Installing and launching the application

*To begin with, we call the **stopAppiumServer()** method to verify if any instance of Appium Server is already running and if so, then it closes that instance, as highlighted in red. The method highlighted in yellow starts the Appium Server.*

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

stopAppiumServer();

startAppiumServer();

File appDir = new File("D:\\");

File app = new File(appDir, "com.bsb.hike.apk");

DesiredCapabilities capabilities=new DesiredCapabilities();

capabilities.setCapability("platformName", "Mi498HA14090292");

capabilities.setCapability("platformVersion","4.4.2");

capabilities.setCapability("deviceName","android");

capabilities.setCapability("app", app.getAbsolutePath());
```

*Then store the apk path file into a variable of File class, which is a Java class as highlighted in blue. Create another variable of File class **app**. It takes two parameters:*

- *The absolute path of the apk file.*
- *Name of the apk file.*

*Now the absolute path is stored into **appDir** variable and the name of apk file in the second part as highlighted in the green box. The rest of the things are the same as discussed in the earlier sessions.*

The complete code is shown here.

```
package com.Qacampus;
import io.appium.java_client.android.AndroidDriver;
import java.io.File;
import java.io.IOException;
import java.net.URL;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.remote.DesiredCapabilities;
public class AppiumserverTest
{
//Appium Launch on Windows
private static Process process;
//Calling the node.exe and appium.js
private static String STARTSERVER = "D:\\Mobile\\AppiumForWindows-1.3.4.1\\Appium\\node.exe
D:\\Mobile\\AppiumForWindows-1.3.4.1\\Appium\\node_modules\\appium\\bin\\appium.js";
//Starting the Appium Server
public static void startAppiumServer() throws IOException, InterruptedException
{
//Runtime class.this is again a java internal class
Runtime runtime = Runtime.getRuntime();
process = runtime.exec(STARTSERVER);
Thread.sleep(7000);
if (process != null)
{

System.out.println("Appium server started");
}
}
//Stopping the Appium Server
public static void stopAppiumServer() throws IOException
{
if (process != null)
{
process.destroy();
}
System.out.println("Appium server stopped");
}
public static void main(String[] args) throws Exception
{
stopAppiumServer();
startAppiumServer();
File appDir = new File("D:\\");
File app = new File(appDir, "com.bsb.hike.apk");
DesiredCapabilities capabilities=new DesiredCapabilities();
capabilities.setCapability("platformName", "Android");
capabilities.setCapability("platformVersion","4.4.2");
capabilities.setCapability("deviceName","android");
capabilities.setCapability("app", app.getAbsolutePath());
```

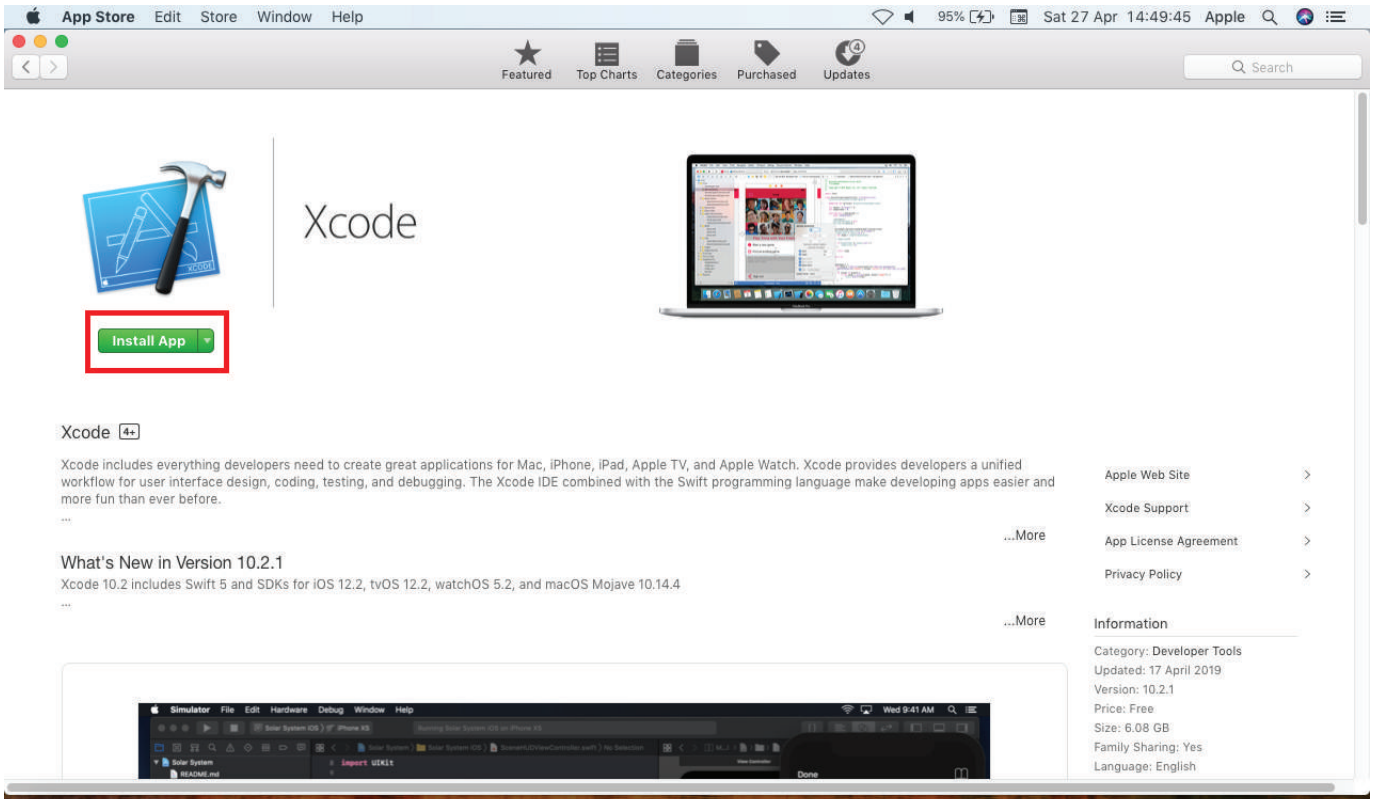
```
AndroidDriver driver=new AndroidDriver(new URL("http://127.0.0.1:4723/wd/hub"),capabilities);
driver.manage().timeouts().implicitlyWait(30,TimeUnit.SECONDS);
driver.findElement(By.id("com.bsb.hike:id/btn_continue")).click();
Thread.sleep(3000);
driver.quit();
stopAppiumServer();
}
}
```

APPIUM FOR IOS

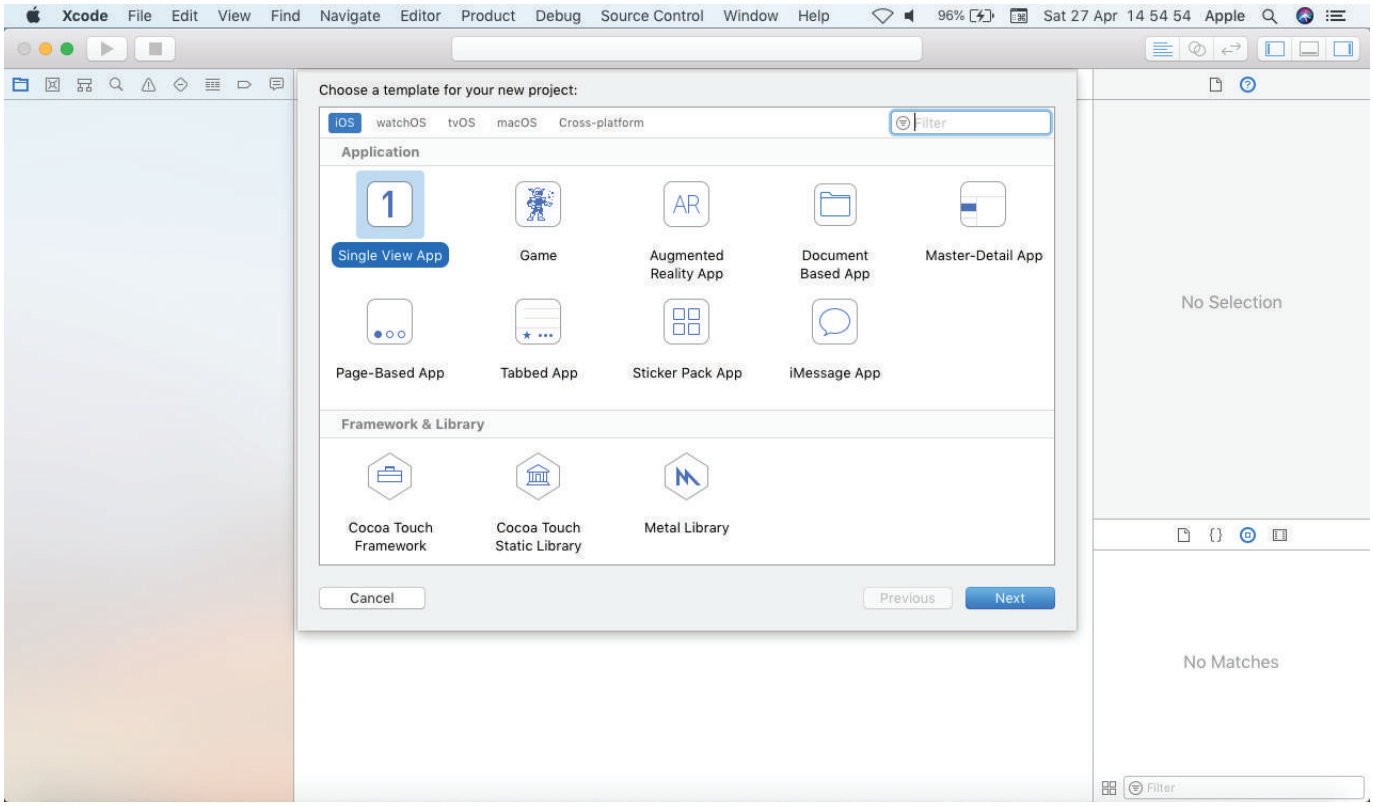
Pre-requisite

- *Mac Book*
- *Xcode*
- *Simulator or real device*
- *Appium*
- *Eclipse*
- *Java*

*The first step is to install Xcode in the Mac Book. To do that, open app store and search for Xcode and click on the **Get** button to download Xcode.*

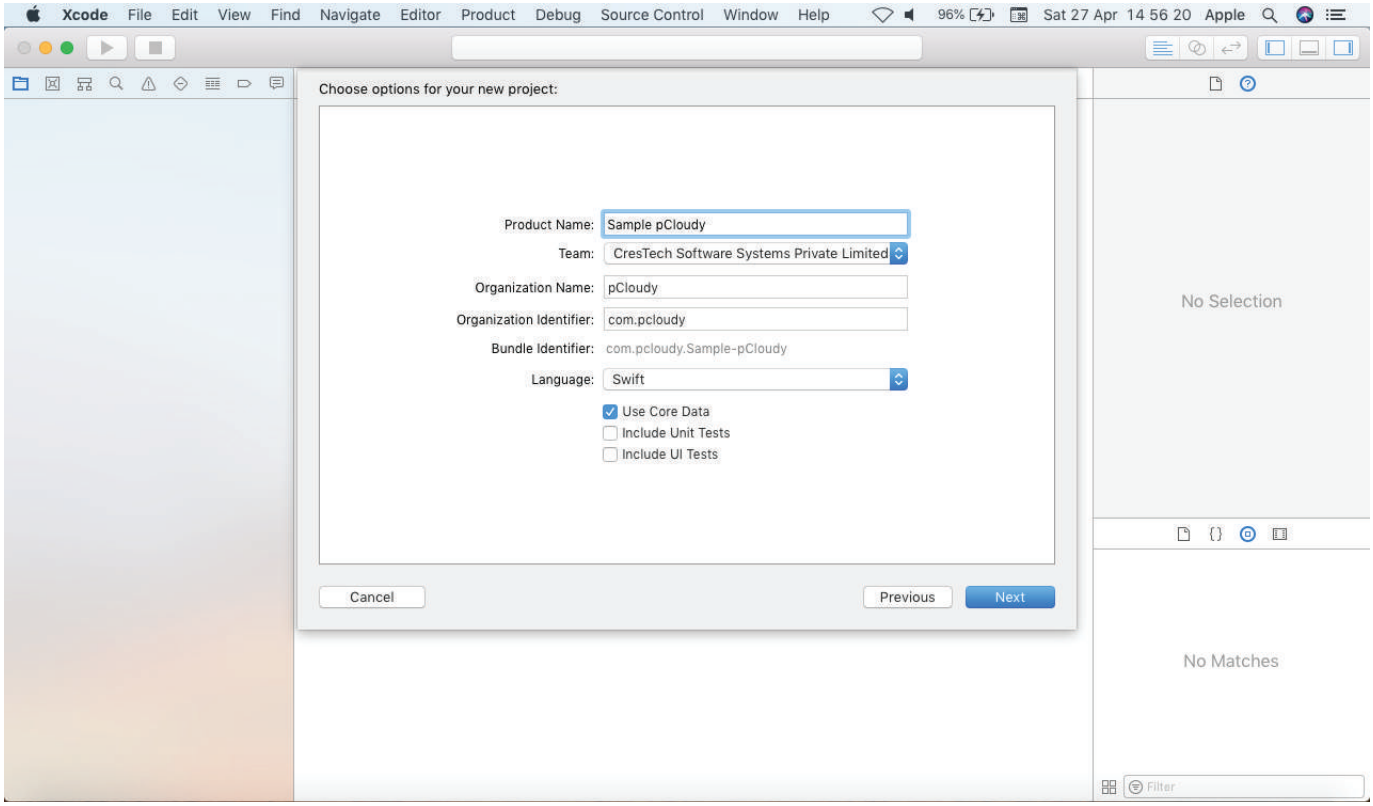


*Then click on the **Install App** option. This will install Xcode on your Mac machine. Now click on the **Agree** button if you agree with the terms and conditions and it will start installing components. Once the Xcode is installed, you need to select "Create a new Xcode project" and you will be redirected to the template selection option.*



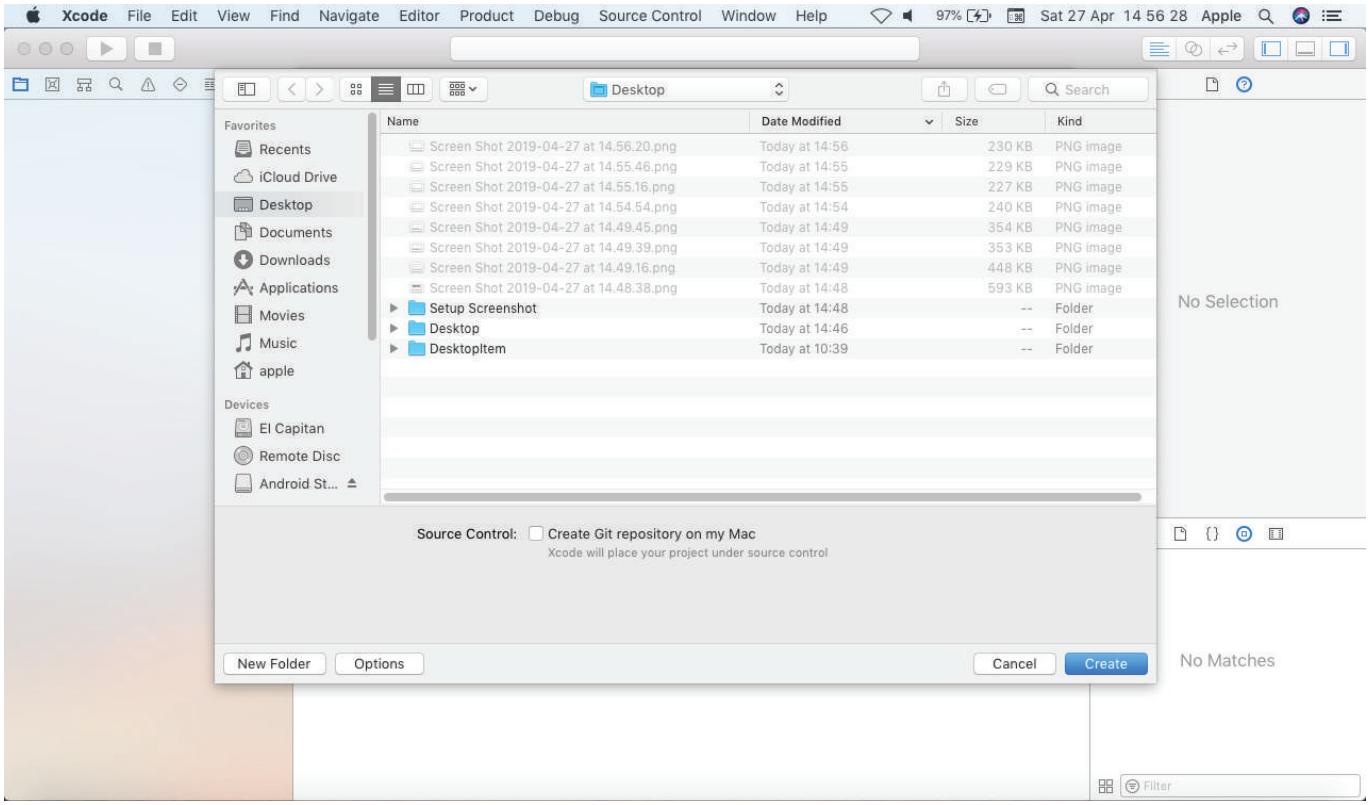
You may select any of the templates or leave default settings as they are. Click on the next button to navigate to the next screen.

Now you will be asked to provide a product name and other product-related settings.

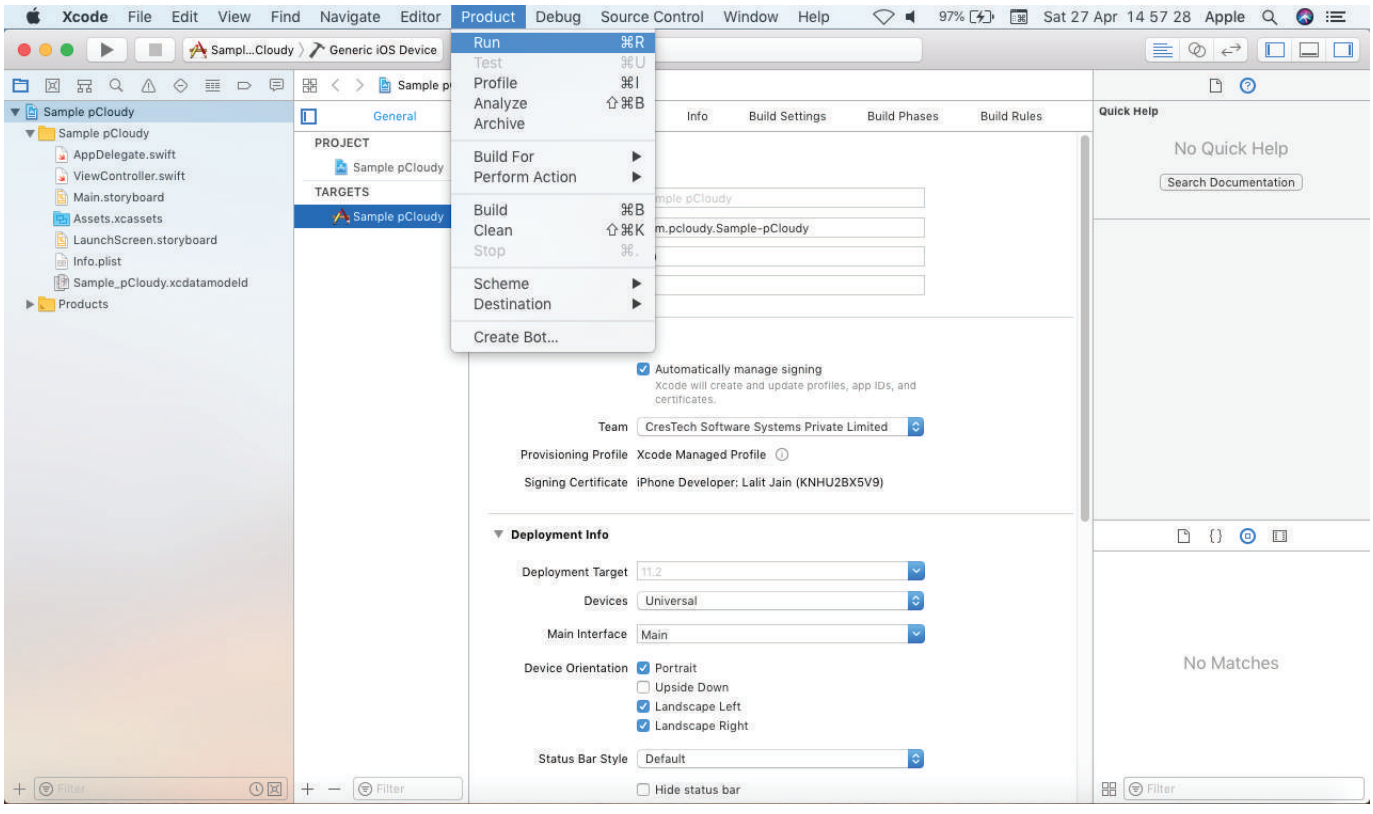


Enter the required details and click on the next button. In the last seen of product settings, you will be asked to give the location in which to save your project.

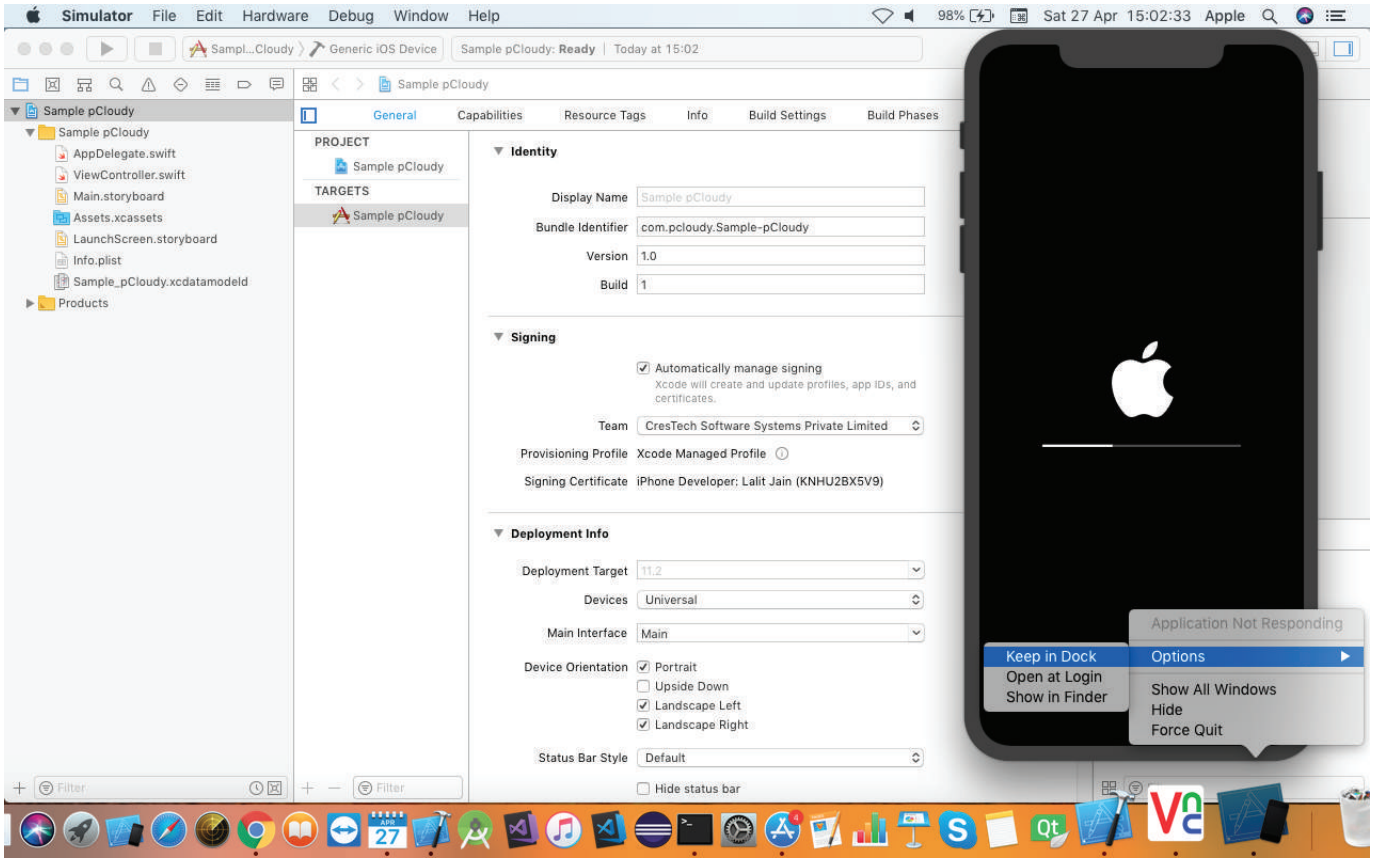
Select the location anywhere as per your requirements and click on the create button.



*Once the project is loaded go to the **Product** menu option and click on the **Run** option.*



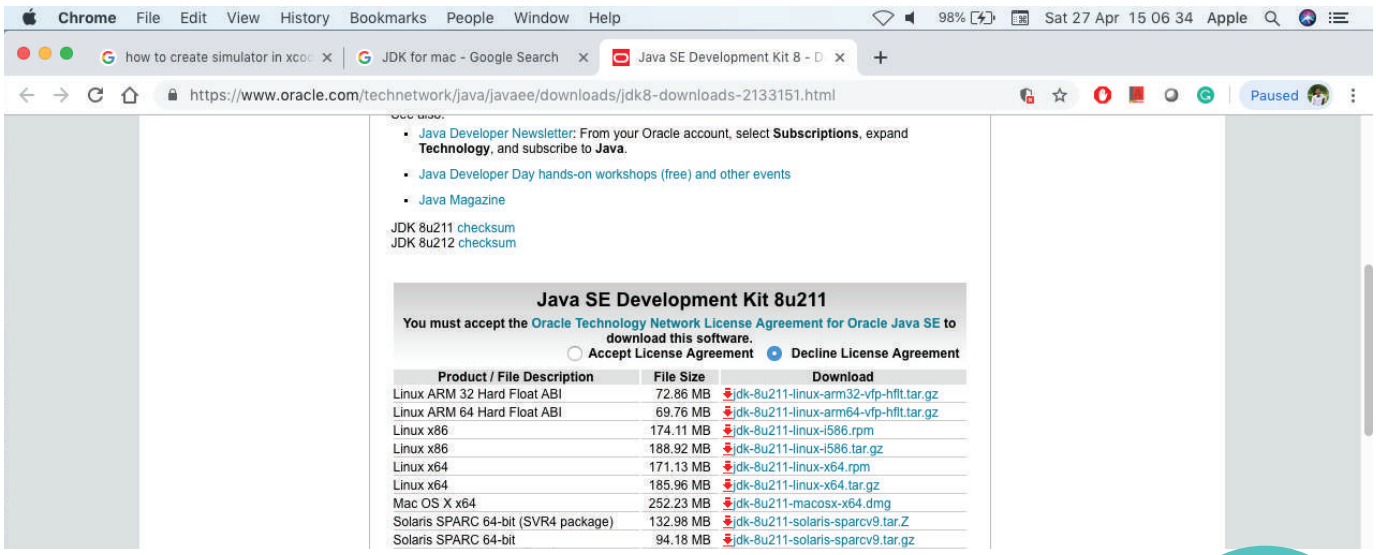
Once you click on the run option, it will launch iOS simulator.



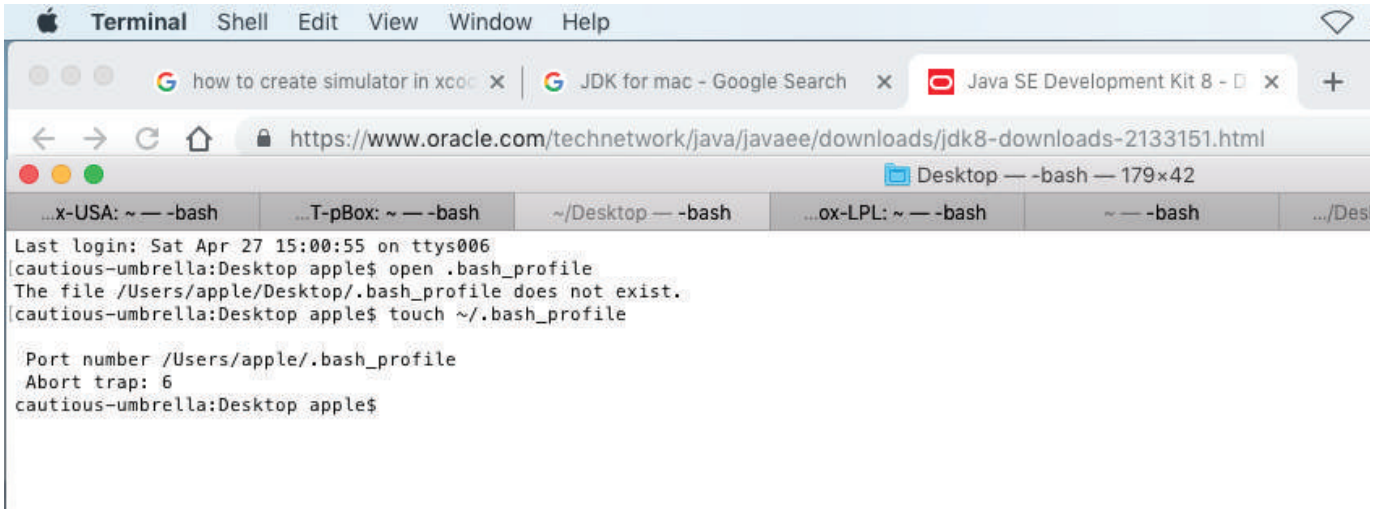
Now right click on the simulator icon on the doc panel and select the keep in doc panel. In this way, you don't need to start Xcode to launch the simulator. Thus you don't need to open the Xcode project each time to launch iPhone simulator. If you want to see the list of simulators, got to the ***Window*** *menu and select the* ***Devices*** *option. You will get a list of all the simulators available with this version of Xcode.*

Installing JDK and setting the path

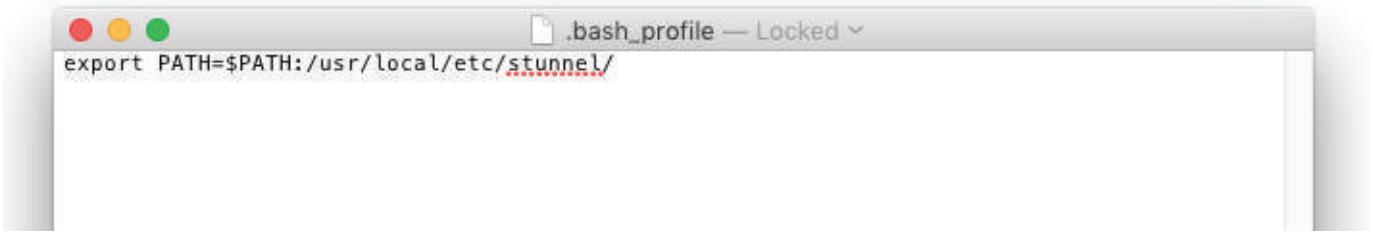
Download JDK for Mac OS and once the file is downloaded, double click on the file and install



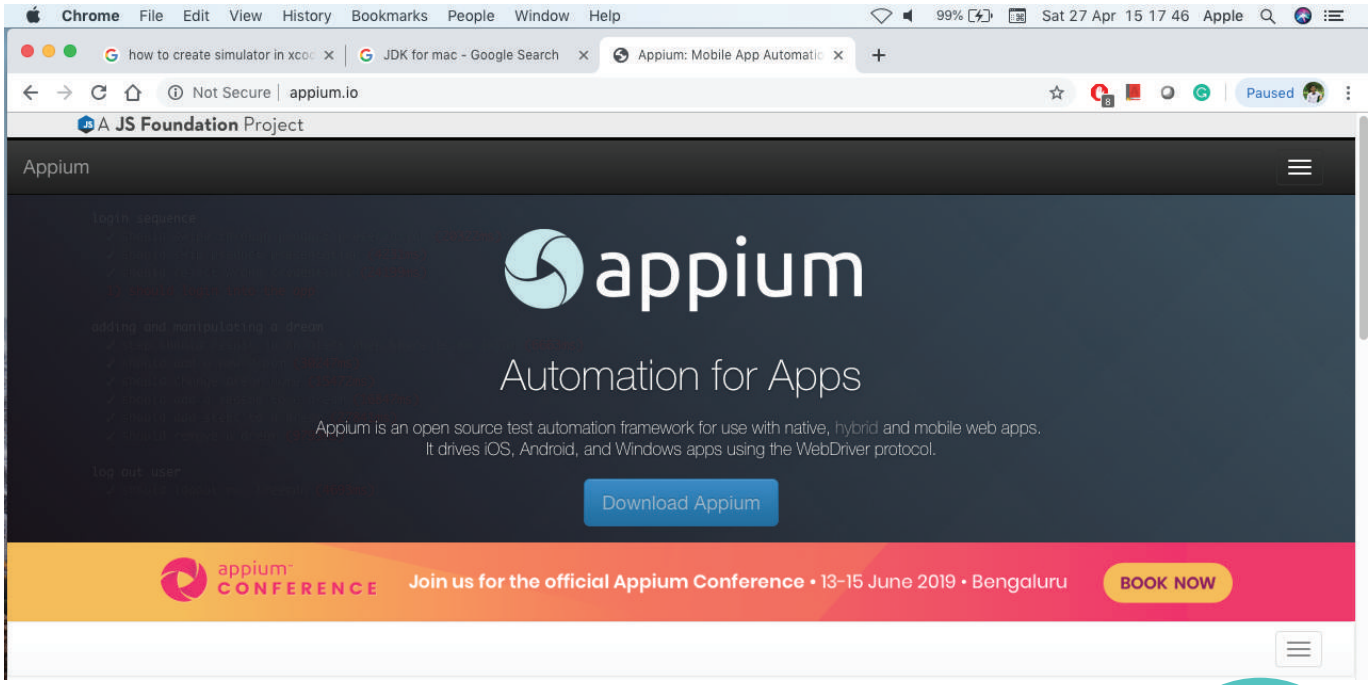
Now you need to set the Java installation path in your environment variable. Open a terminal and write "open.bash_profile".



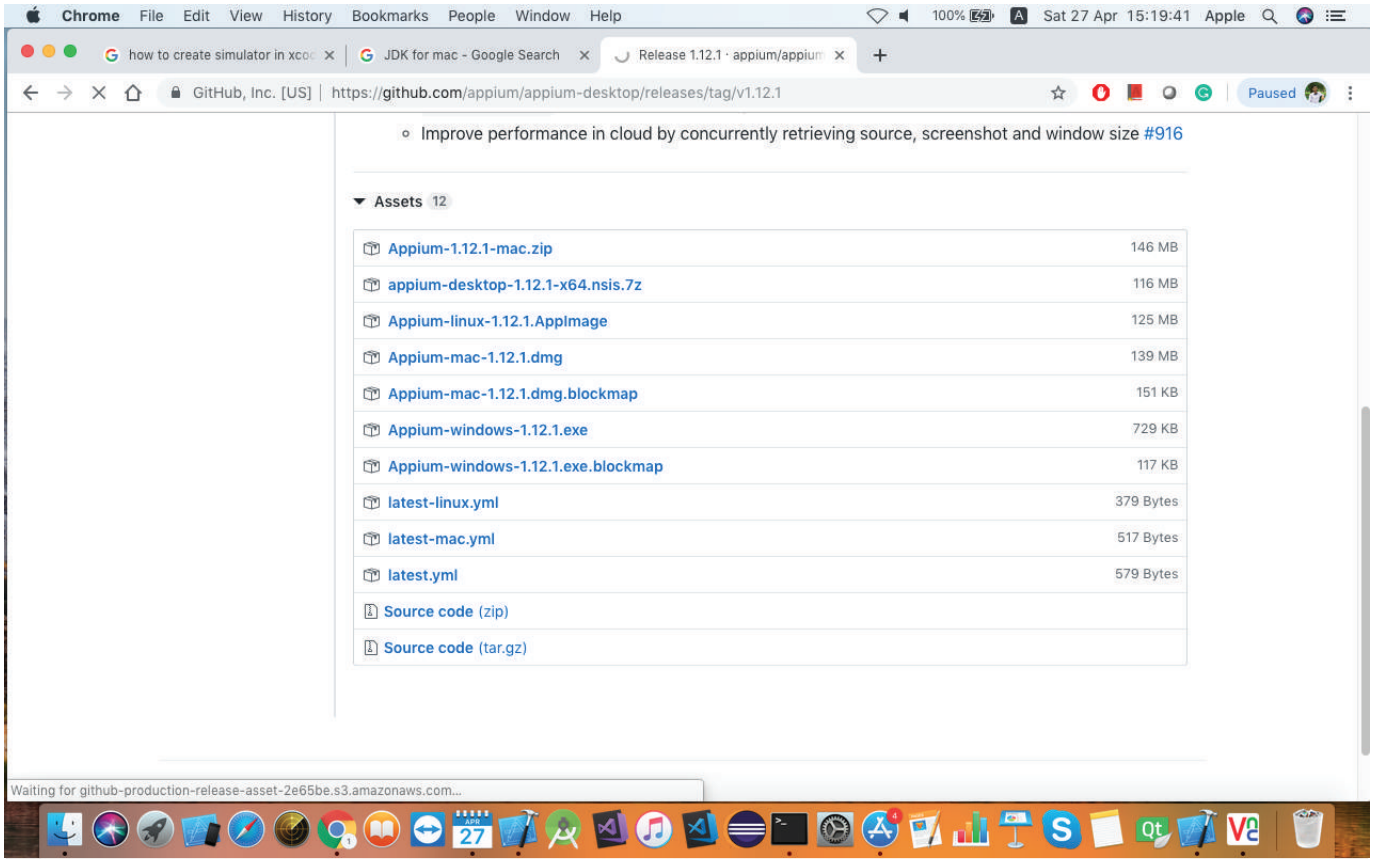
If ".bash_profile" does not exist then execute the command "touch~/.bash_profile". This will create the file. So when you execute the command, it opens the ***".bash_profile"*** *file.*



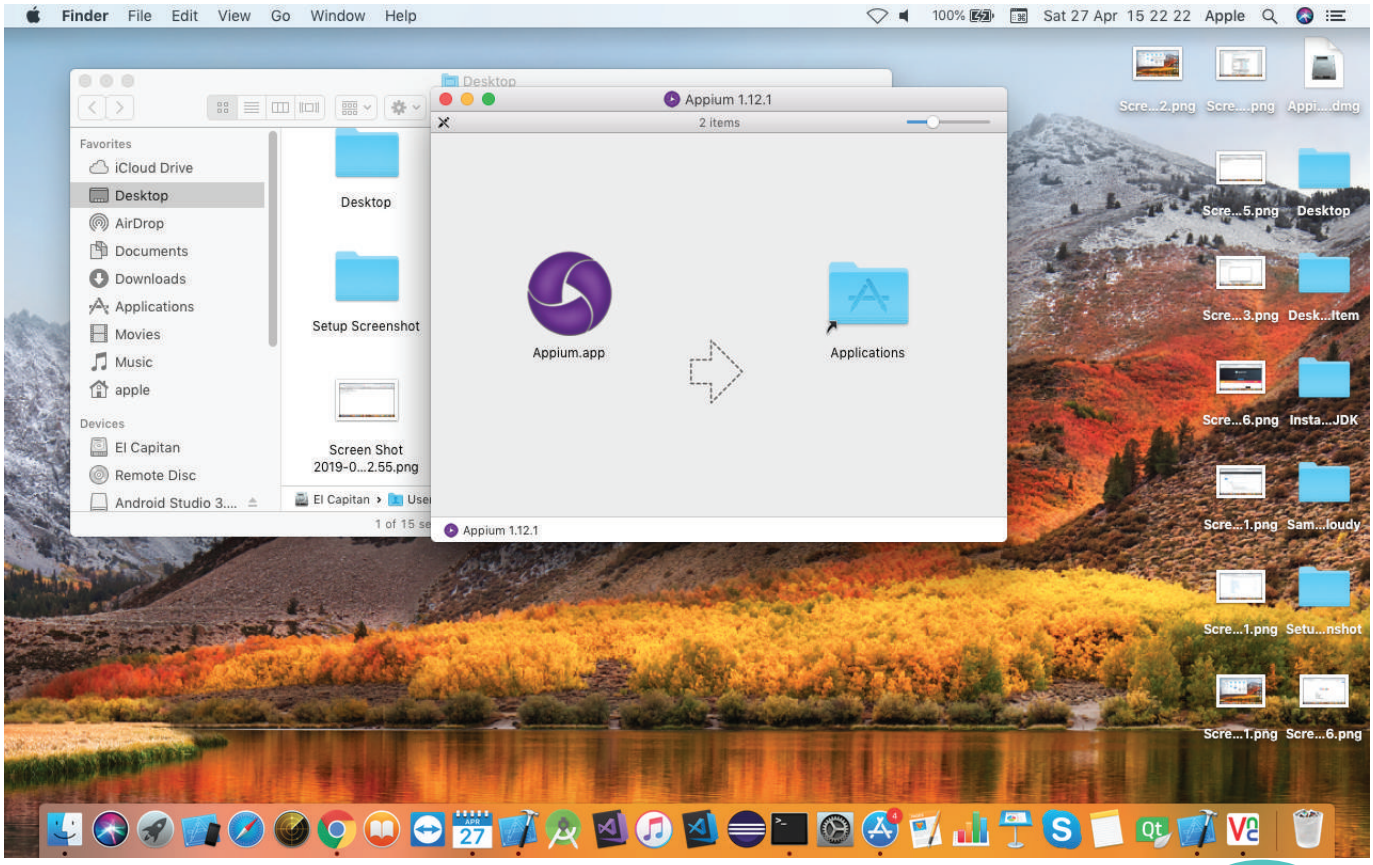
Copy the path to JDK home, write ***"export JAVA_Home=<Path to JDK Home> "*** *and then save this file. Open command prompt and execute the java - version command. Open Appium.io and click on* ***Download Appium*** *button.*



Now download the latest .dmg file and once downloaded, double click on the .dmg file.



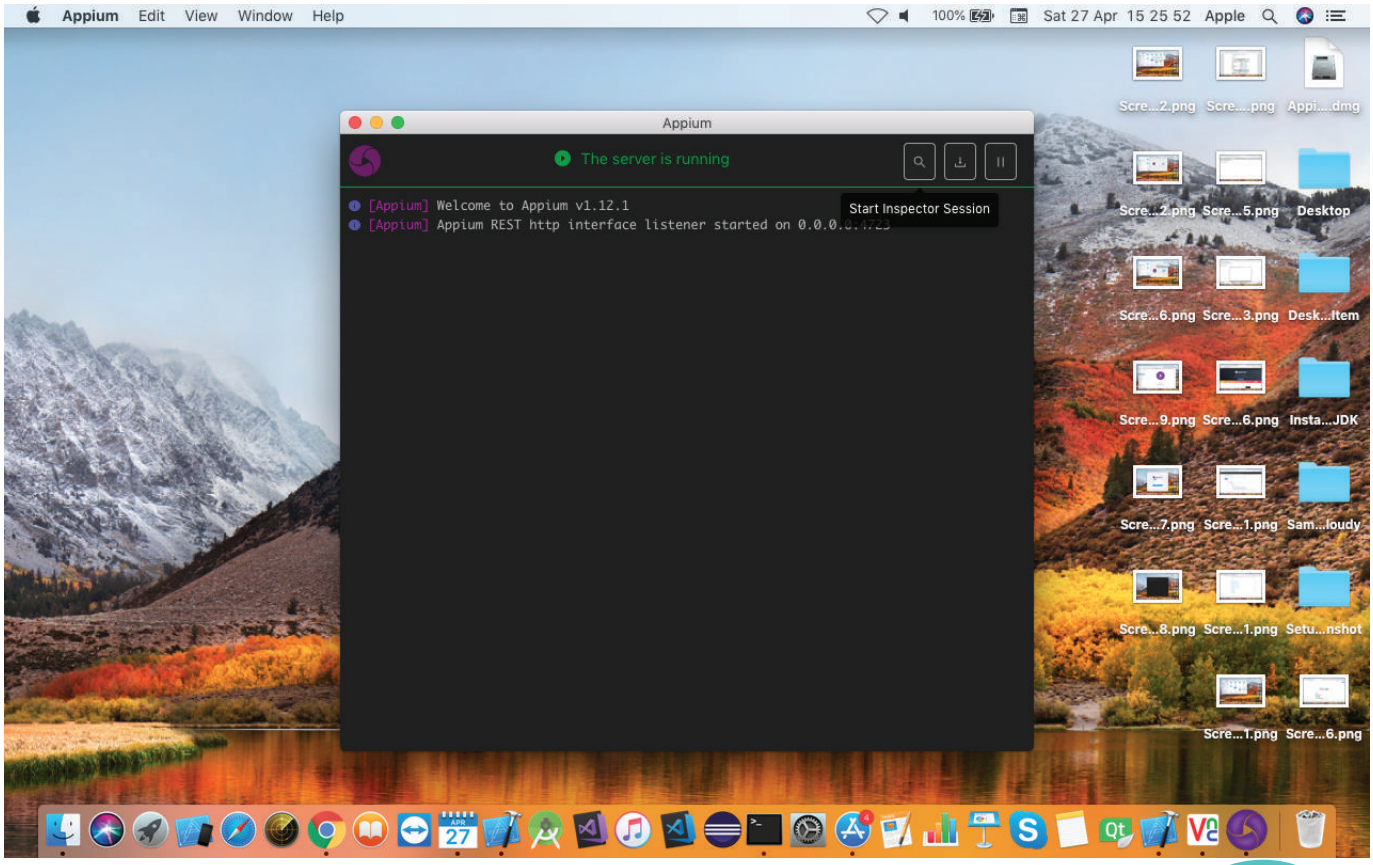
It will open a new window where you need to drag and drop Appium into the application folder.



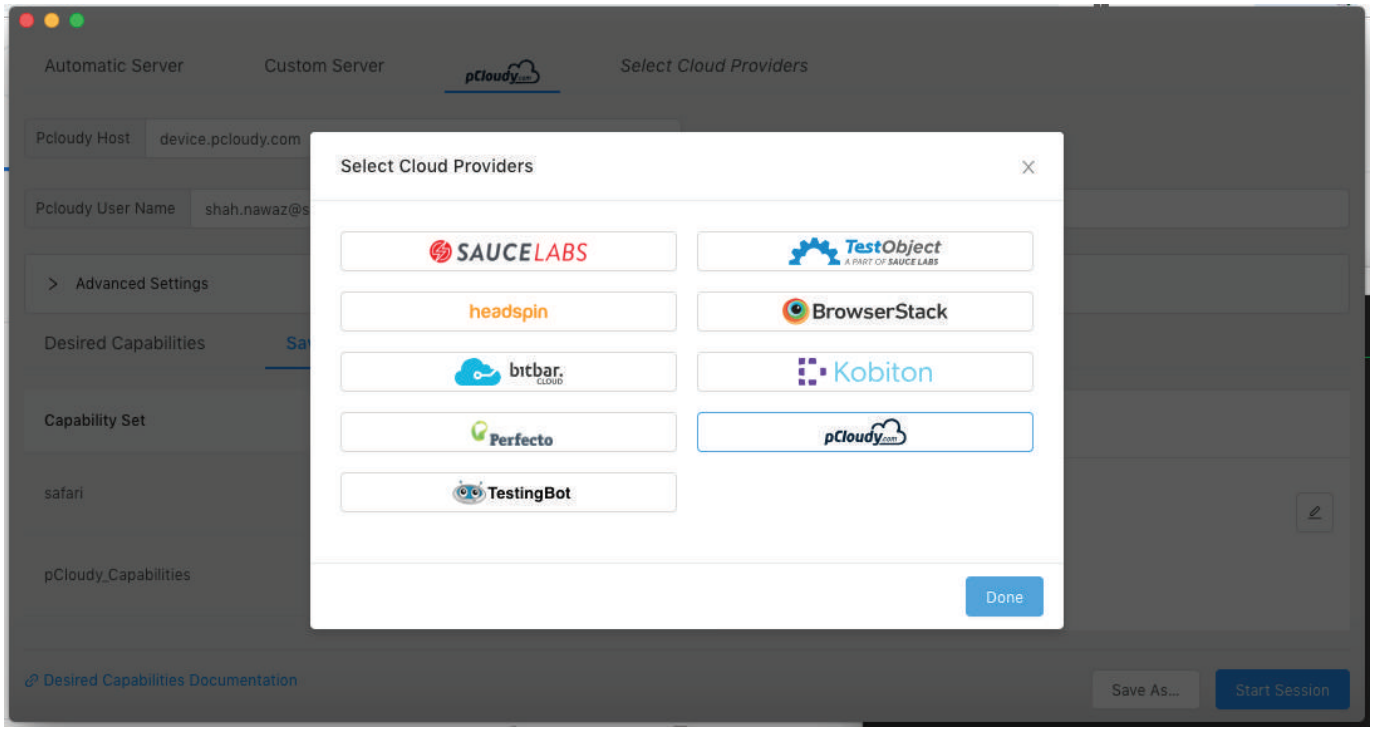
Search for Appium and double click on the Appium icon. This will launch the Appium server on your Mac machine.

*To Launch Appium on a simulator click on the **Start Server** button.*

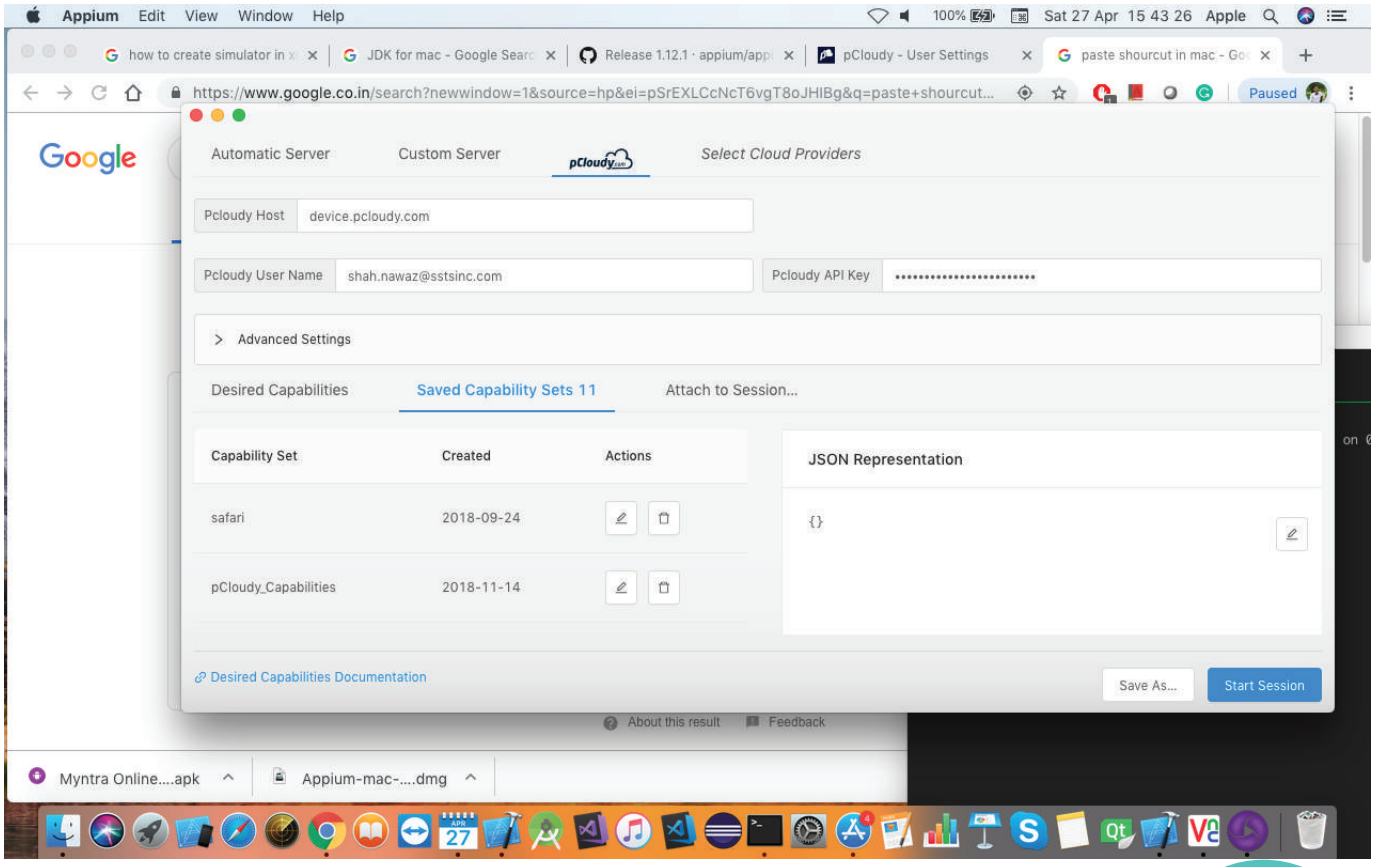
*Now click on the **Start Inspector Session** button in the Appium server.*



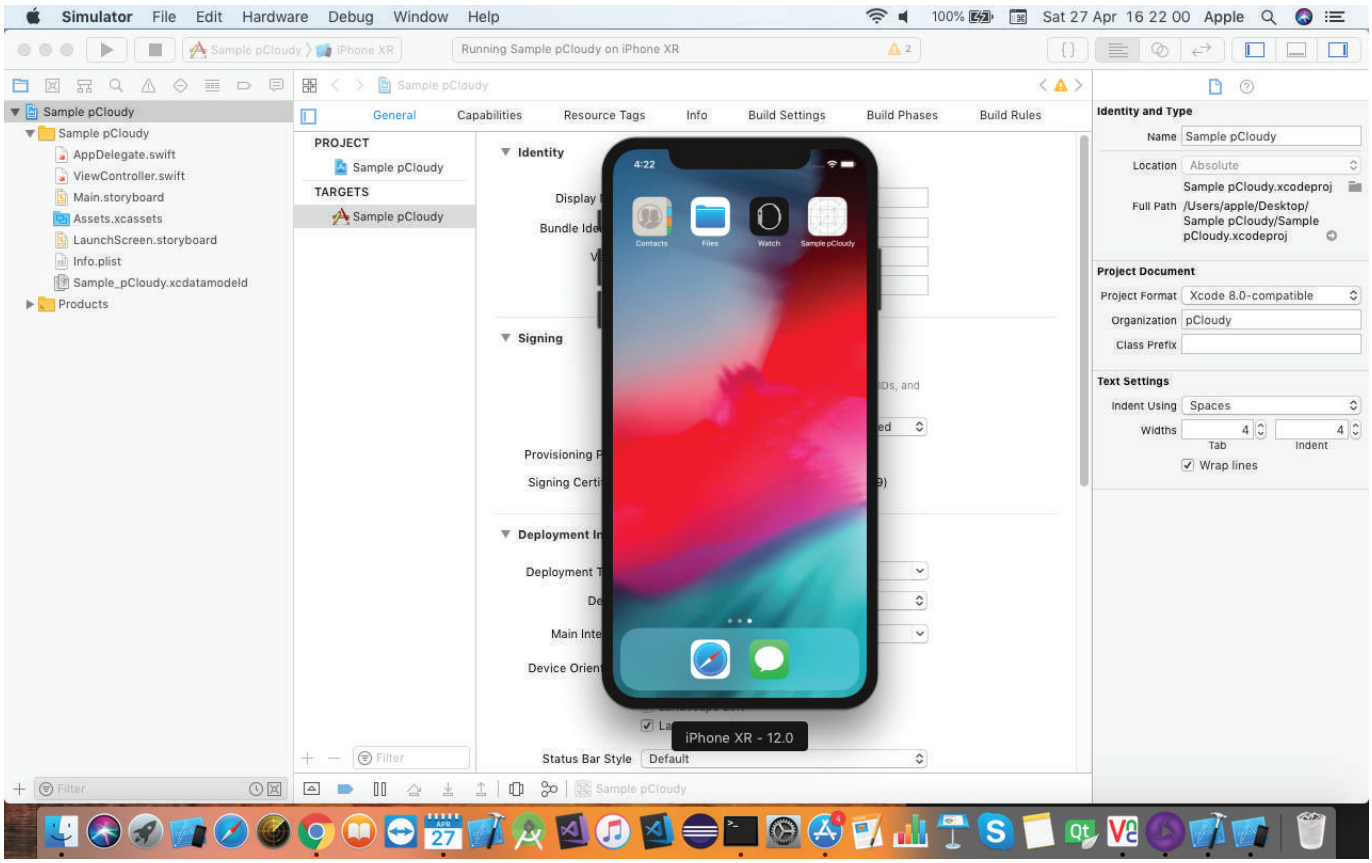
Once you select the cloud provider from the list you can click on ***Done.***



Then you need to enter the ***Host name, User name, API Key, Desired Capabilities*** *and then click on* ***Start Session.***



Your simulator is now connected.

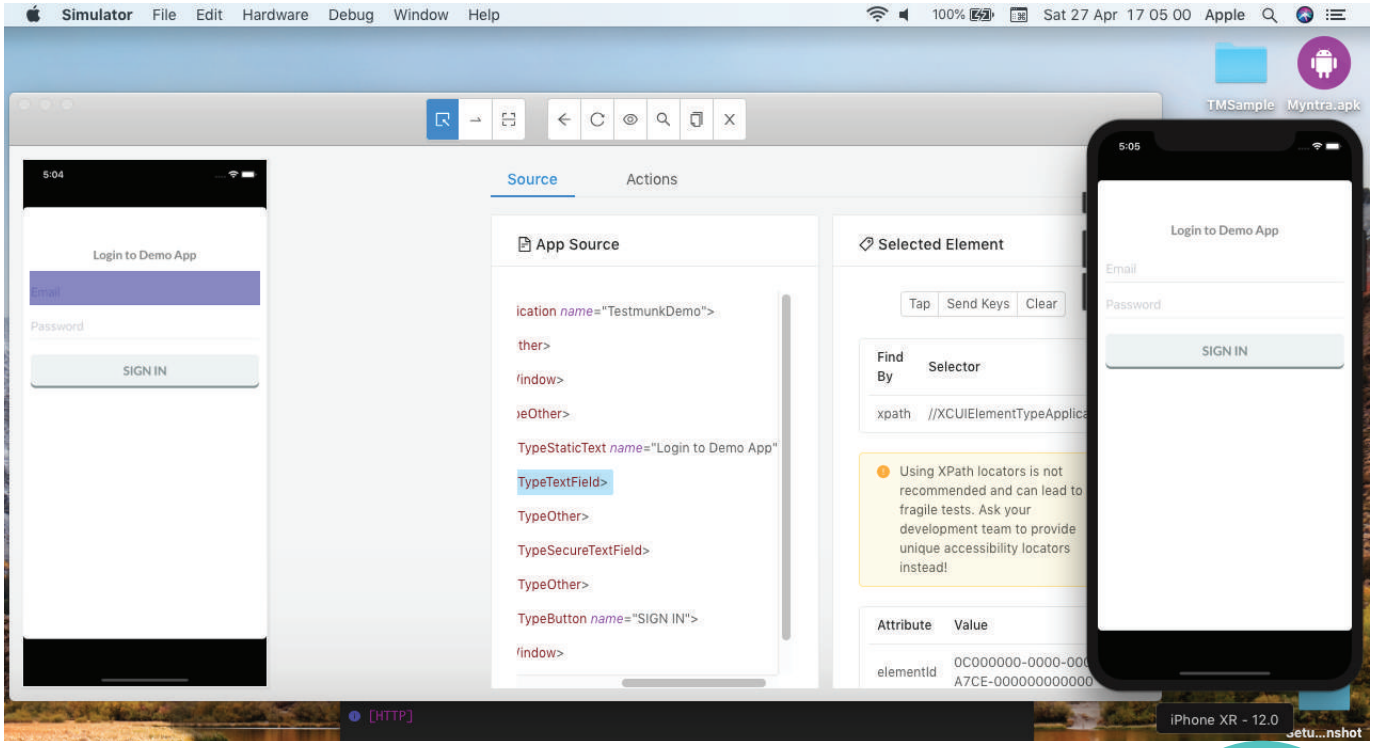


APPIUM INSPECTOR

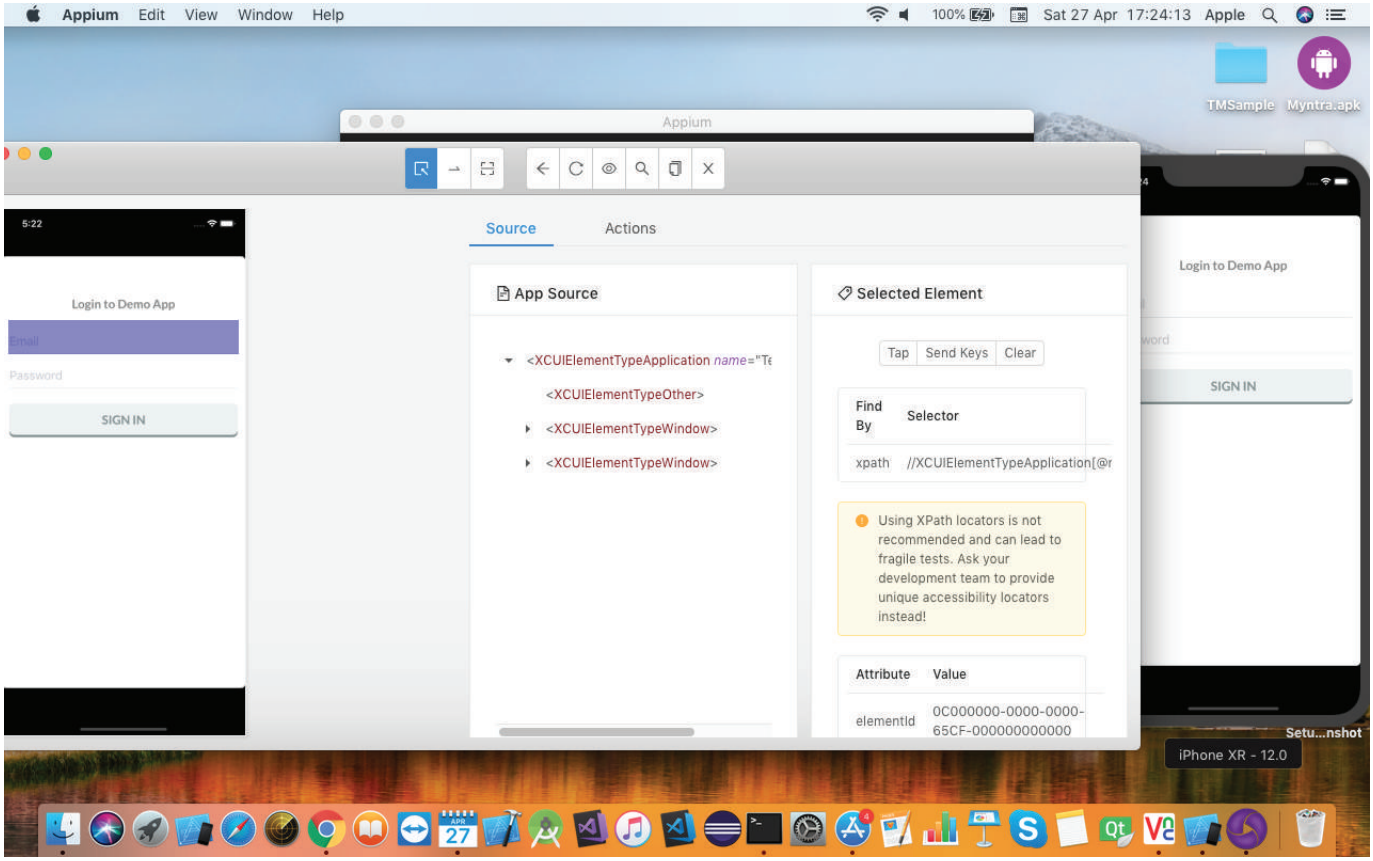
In our previous chapter on Android, we learned about UI Automator Viewer, Which is available on Android SDK, to get the properties of the application object. In the case of iOS, Appium itself provides an Inspector which helps users to locate those elements in the application.

First, open the simulator by clicking on the dock option.

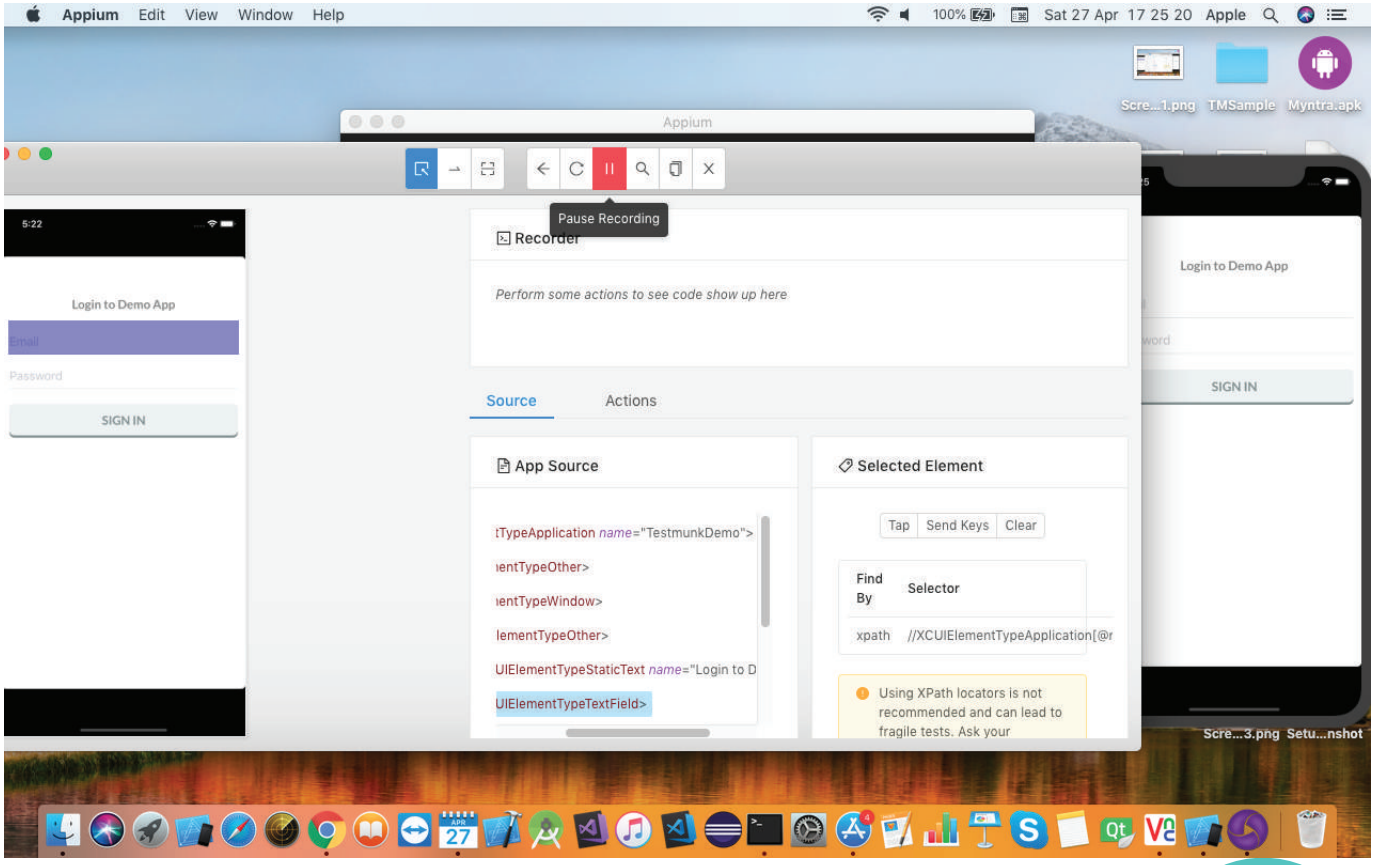
*Now in the **Device/Simulators** window, select the simulator. Open the Appium Desktop and keep the simulator side by side.*



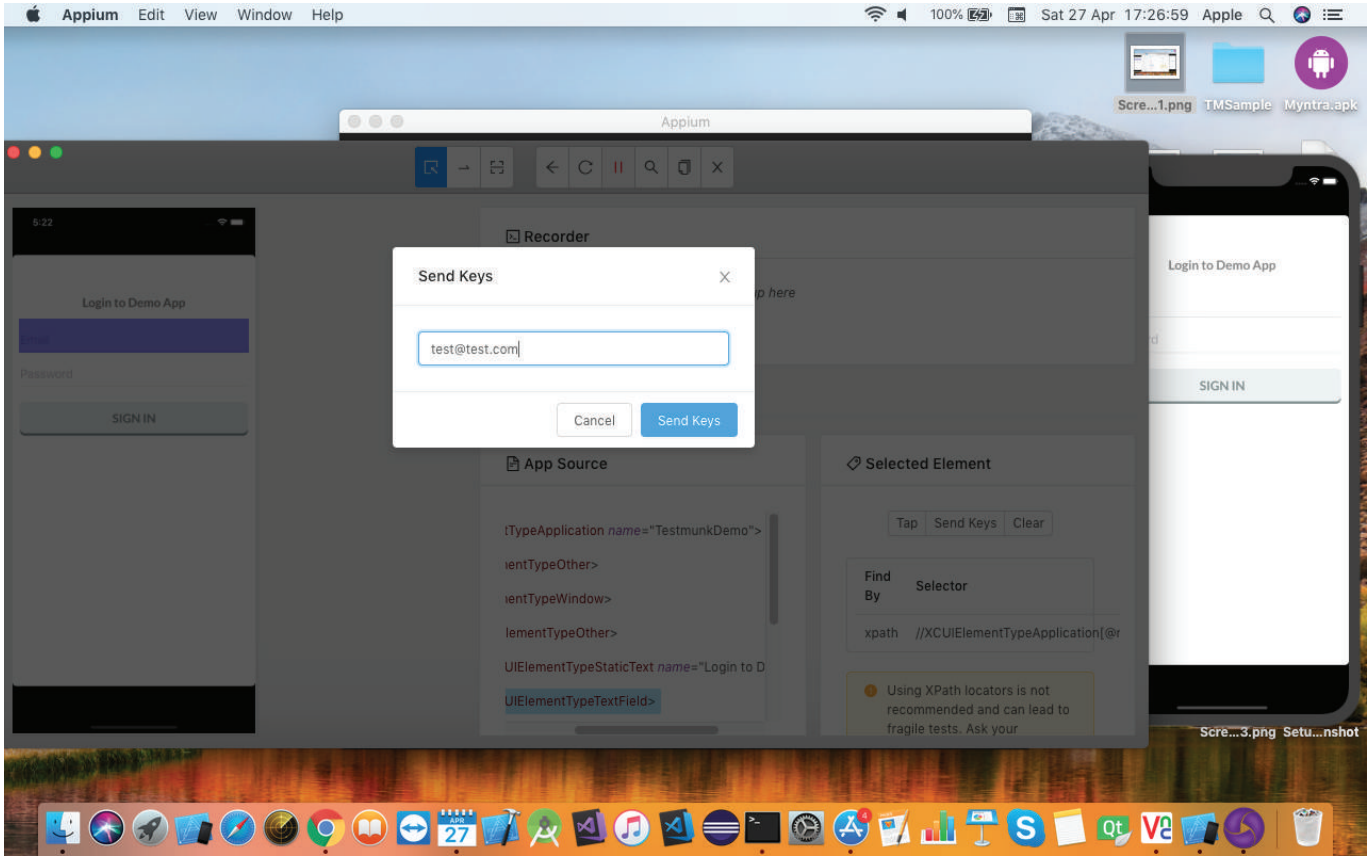
Once the inspector is started, select any of the objects on the screen. It will show you the complete hierarchy and properties of that object.



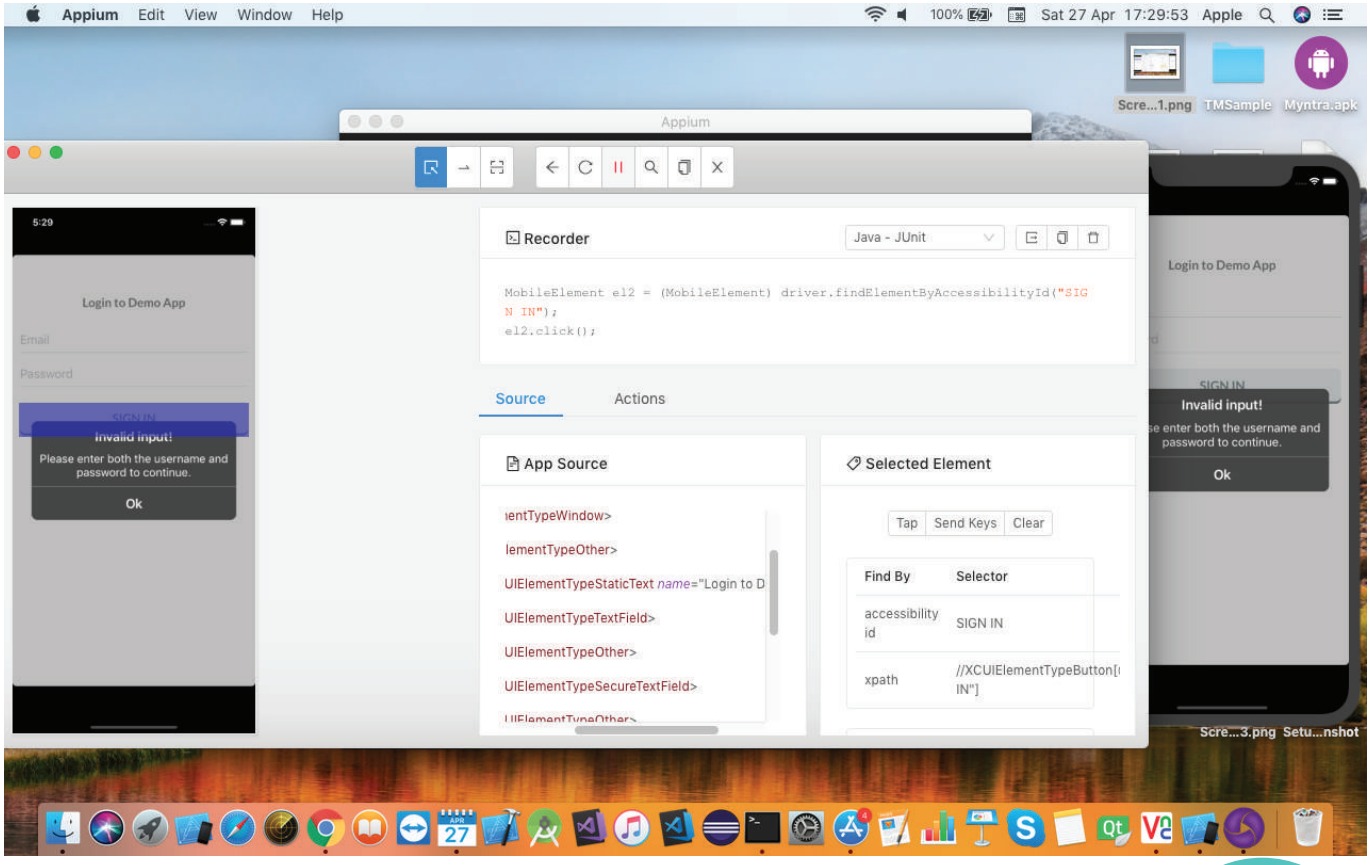
*At the top of the window, you can see the **Record** button which is used to record all the actions taken and record the script.*



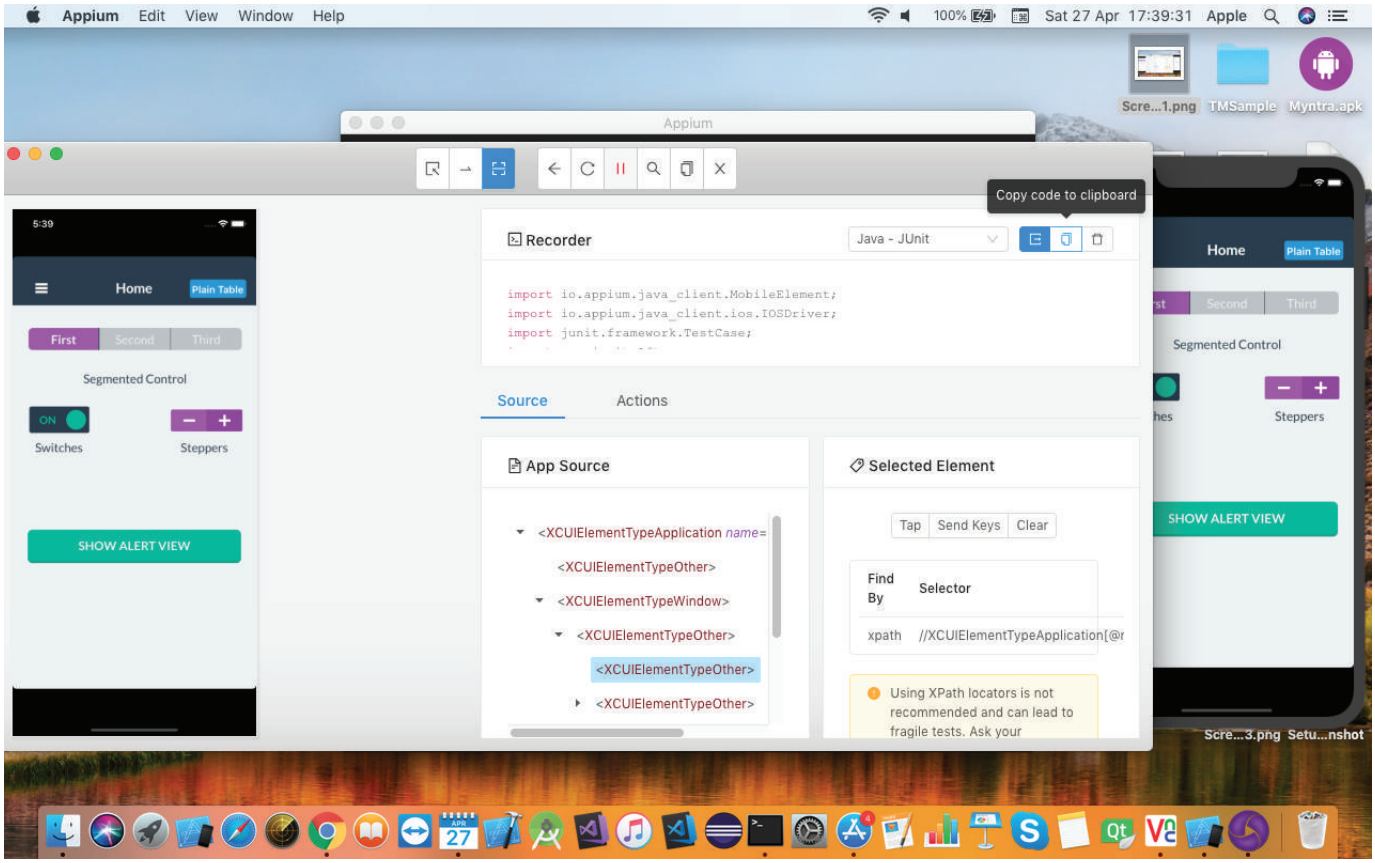
*To select any object, click on the **Select Element** button and then you can use **Tap** button to click on an object, **Send Keys** to enter text and clear to undo the action.*



As soon as you perform an action on an object, it is recorded in the form of a script.



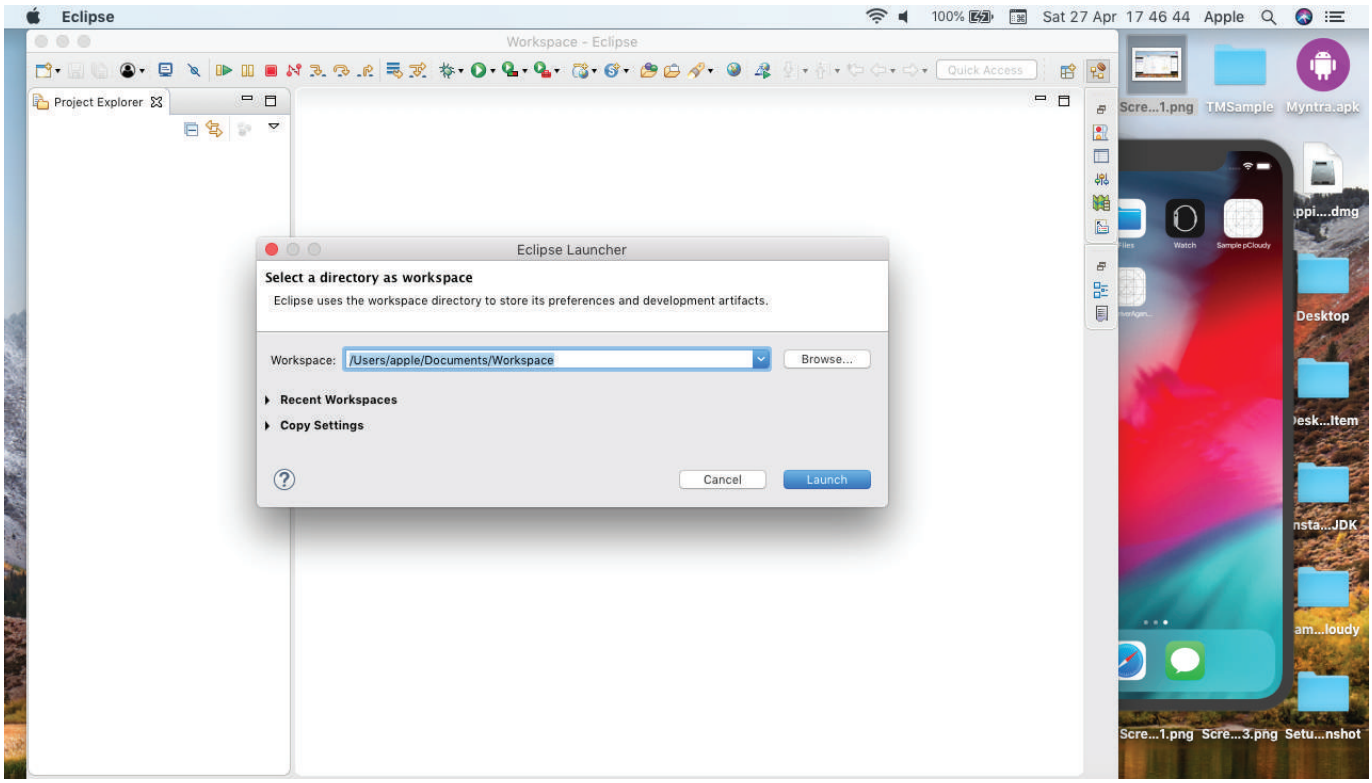
Once you are done with the recording you can copy the script and paste in eclipse editor.



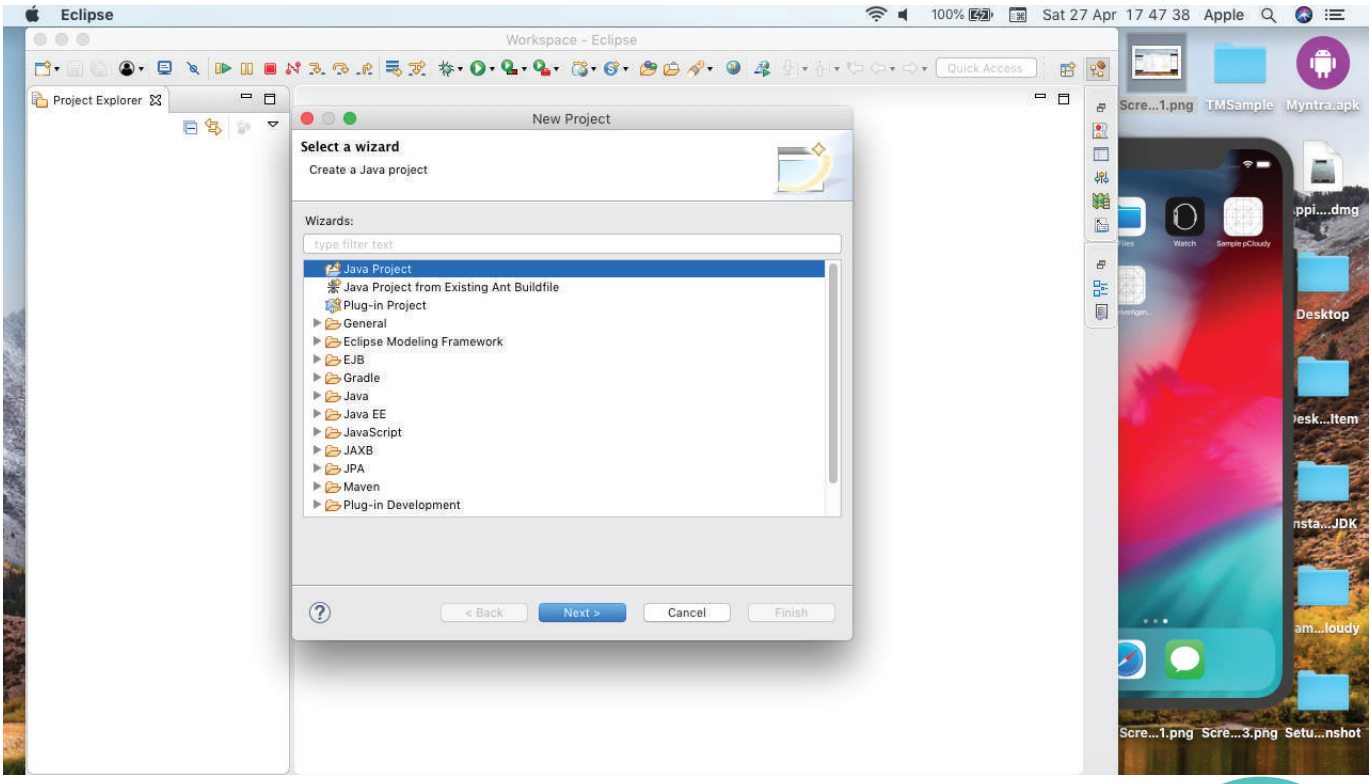
YOUR FIRST APPIUM SCRIPT IN IOS

Now let's try to launch an app with code. To write code in Java you need Eclipse, Selenium Standalone JAR, Appium Java Client. First, go to eclipse.org and download Eclipse. Now download Selenium JAR file from the seleniumhq.org/download page.

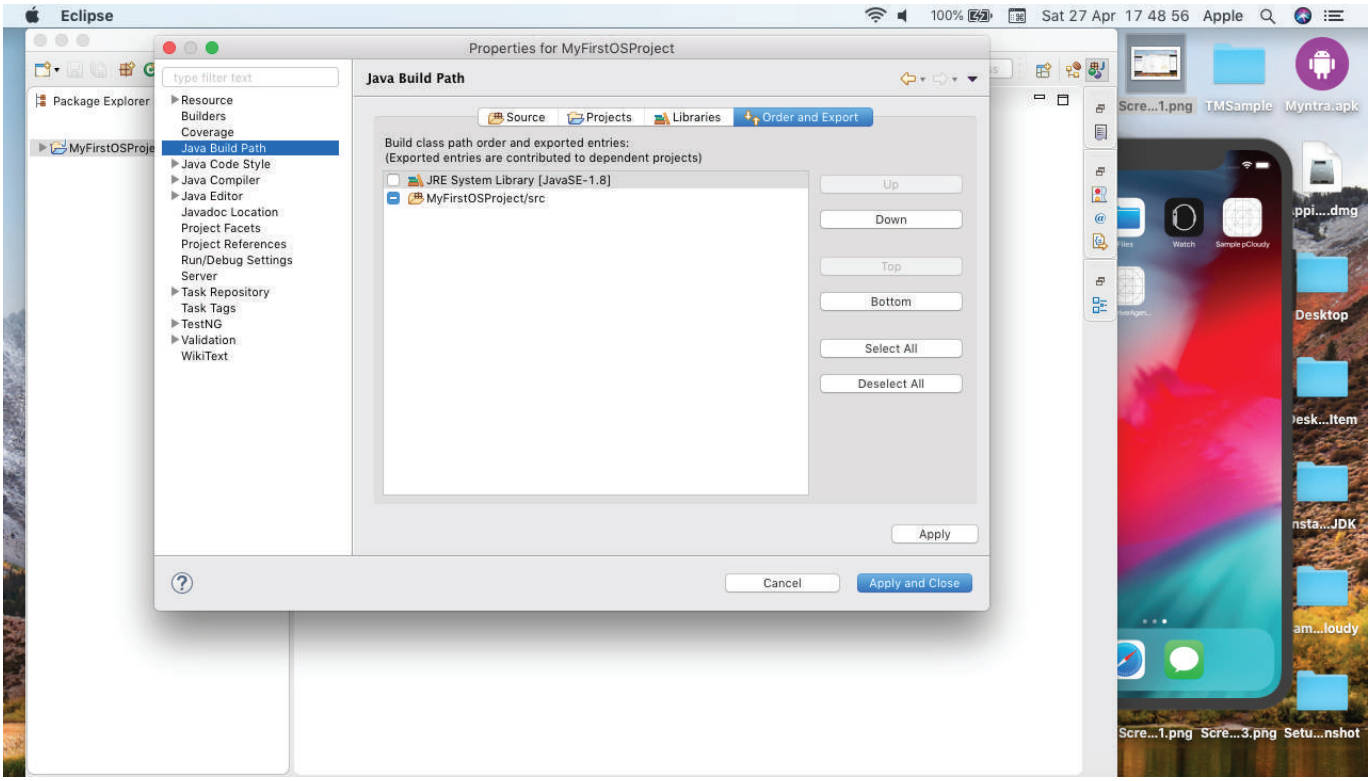
Next, you need to get Appium client libraries based on the programming language you use. Now you can launch Eclipse and select a workspace location.



Create a new Java project and enter your project name.

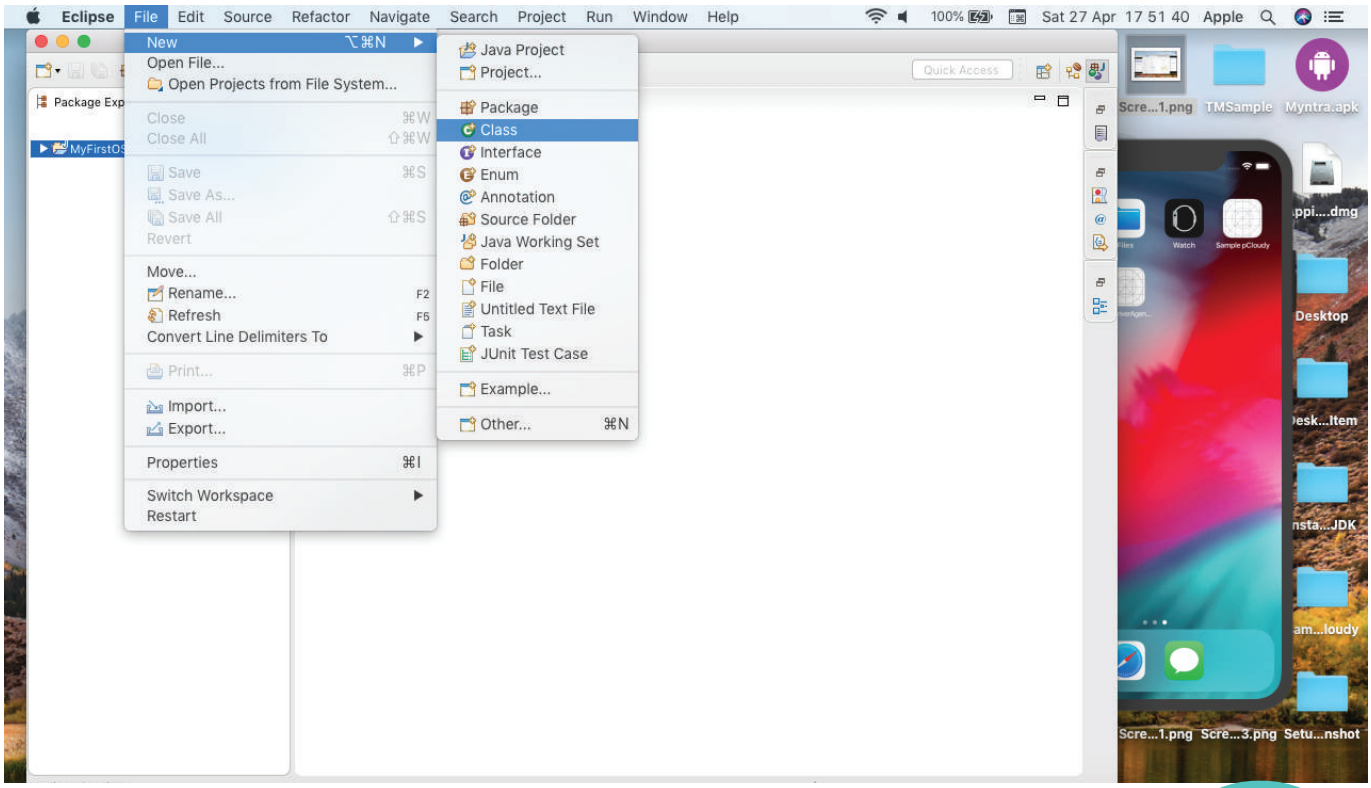


*Now go to project properties and then select the **Java build path** option.*

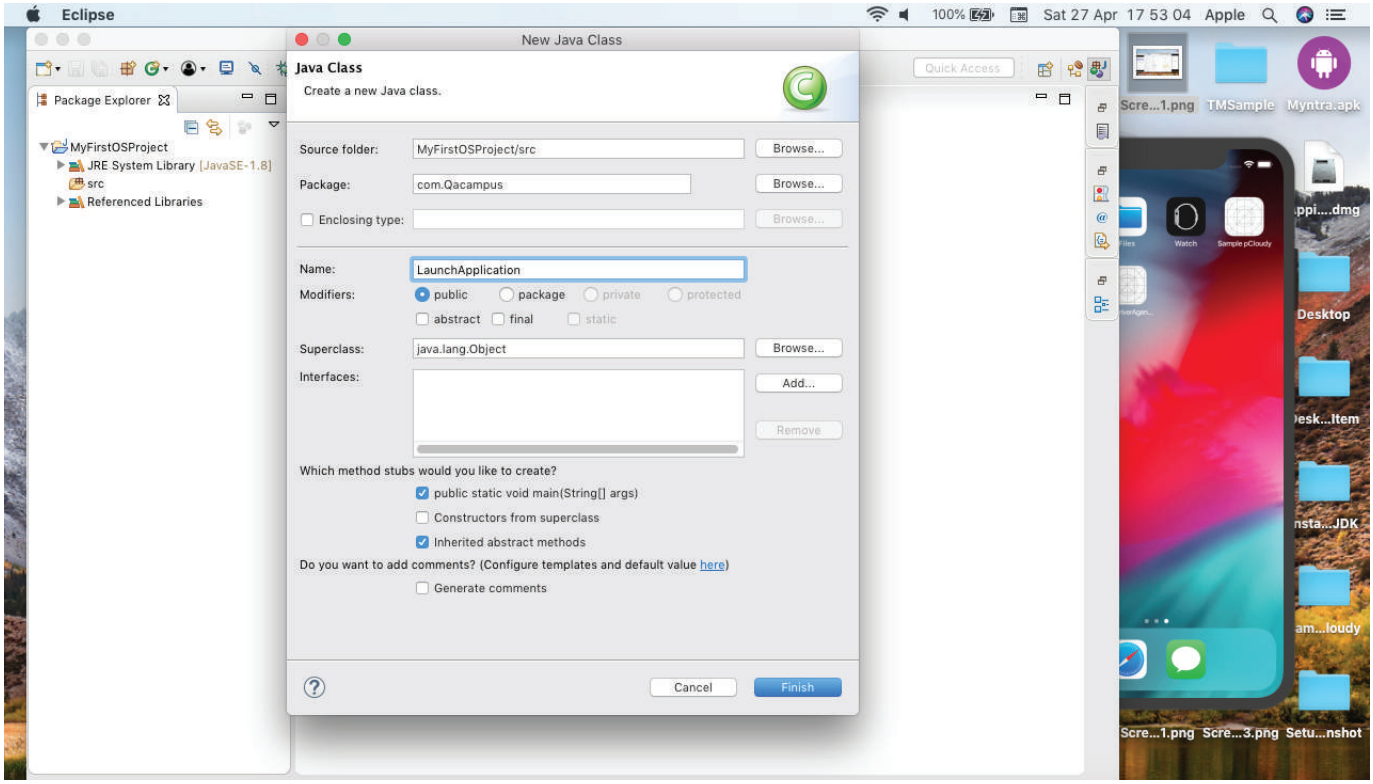


Then click on the "Add External JARs" option. Now add both the Selenium server and the Java client files.

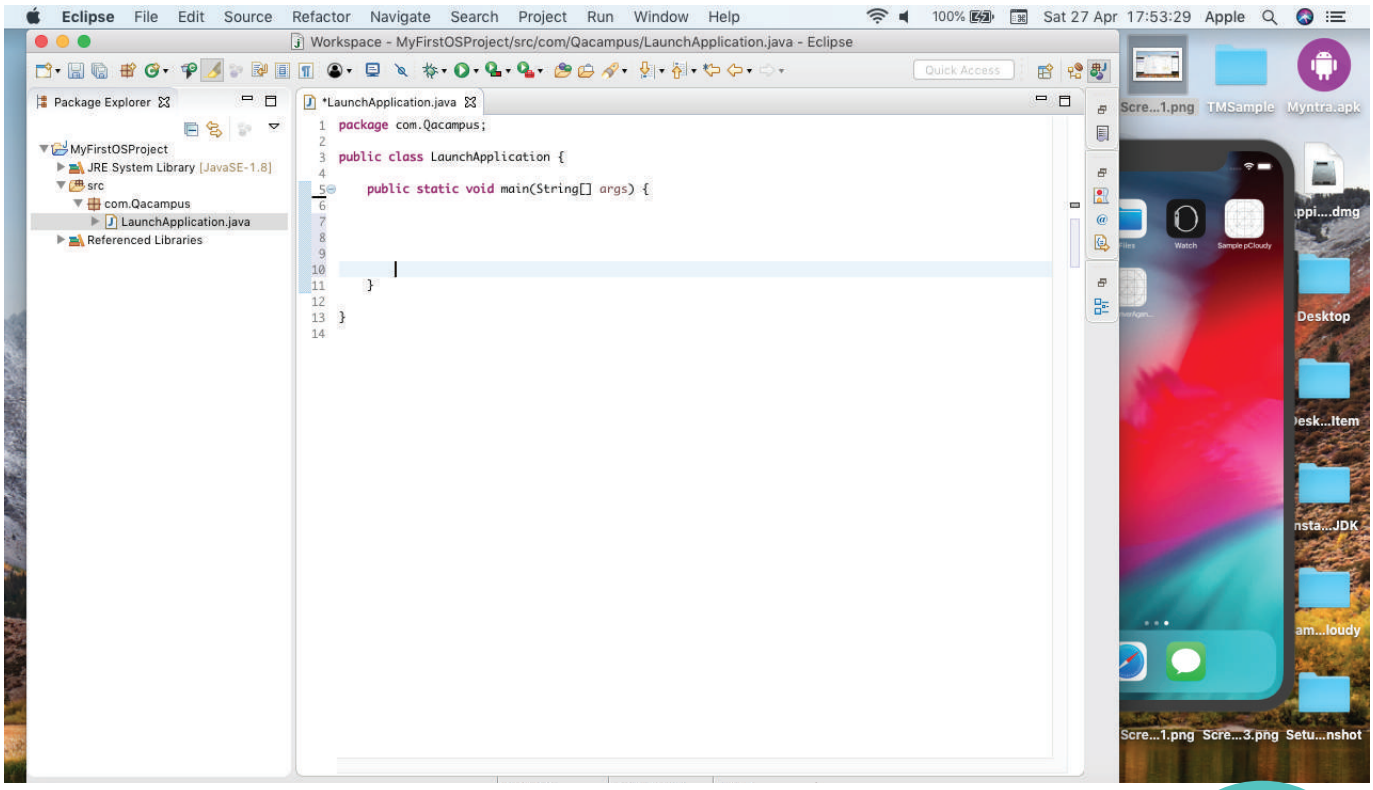
Now go to the file menu and create a new class.



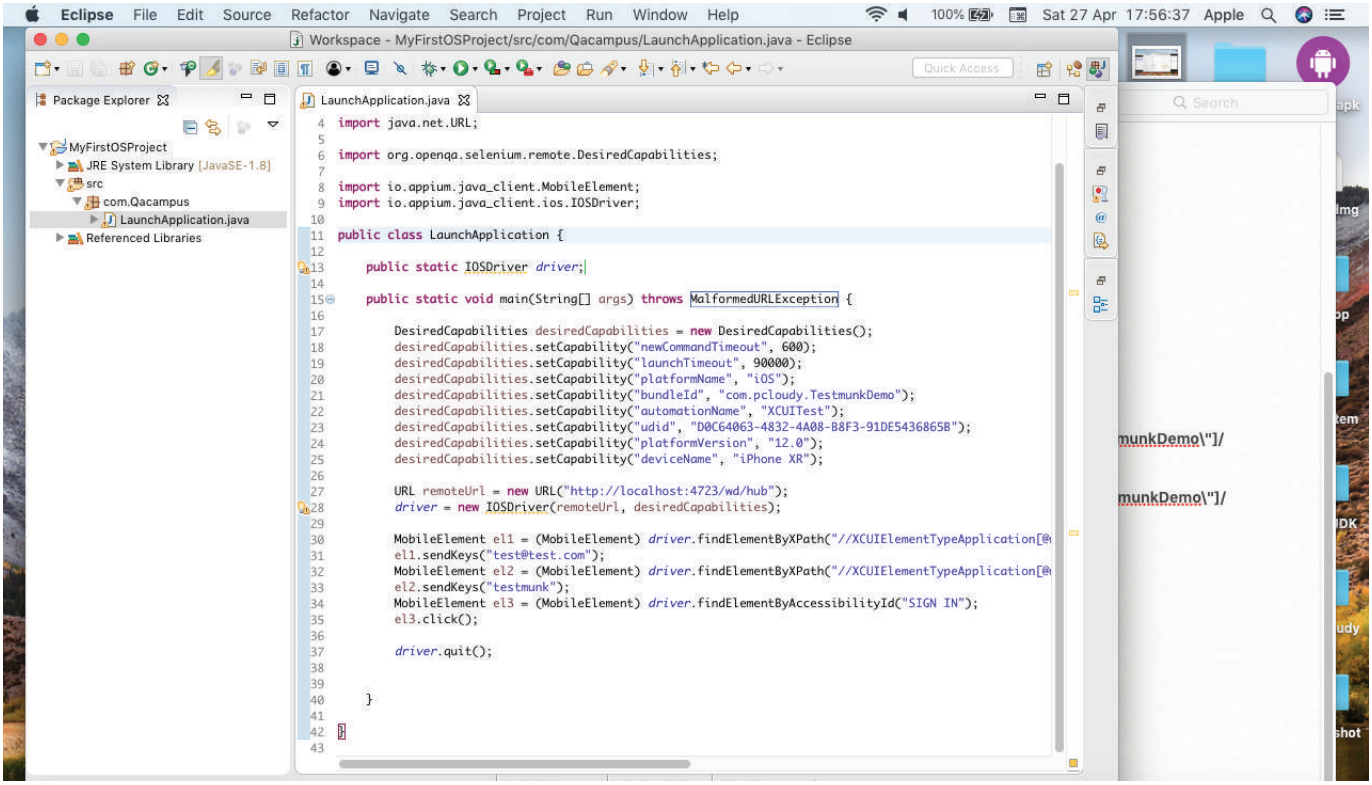
*In the **Java class** dialogue box enter the package name, class name and select the main class checkbox.*



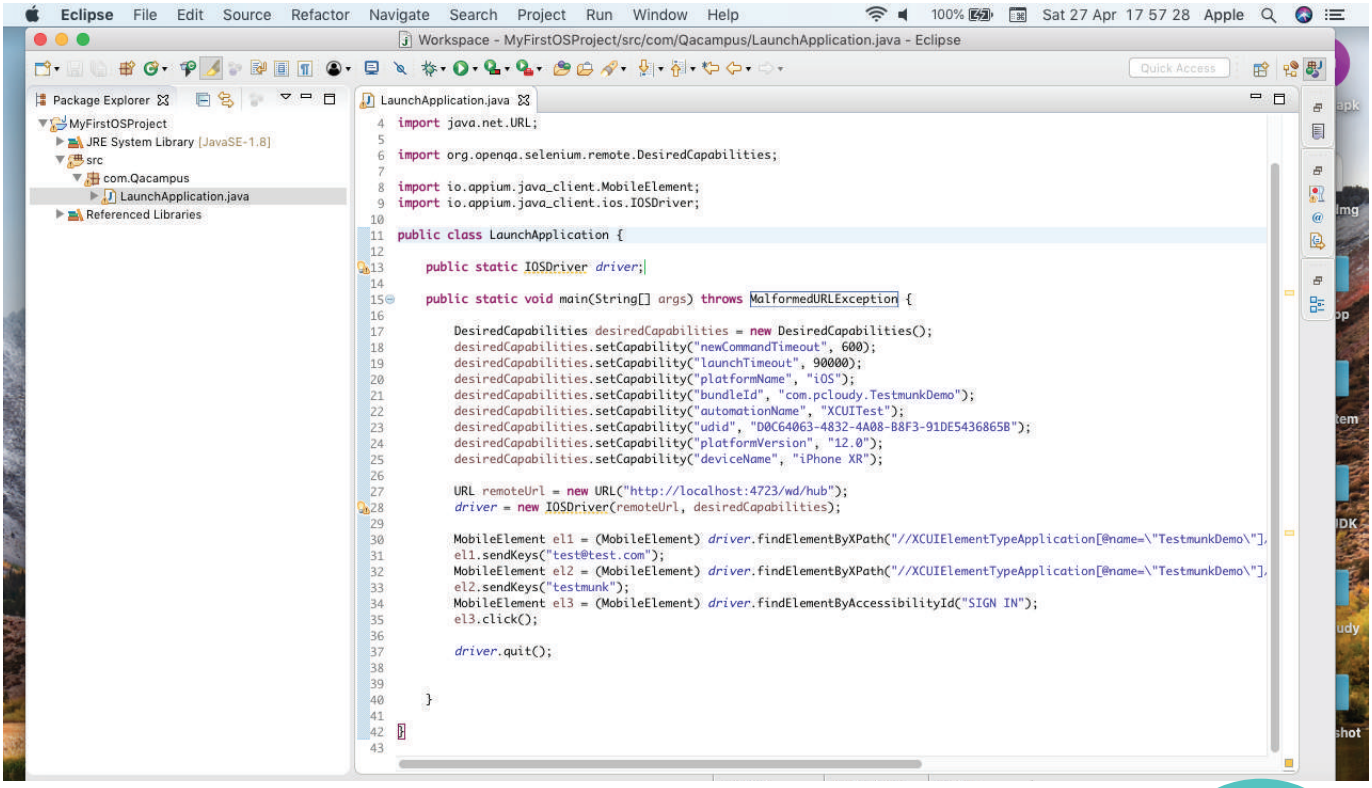
A class has been created with main method.



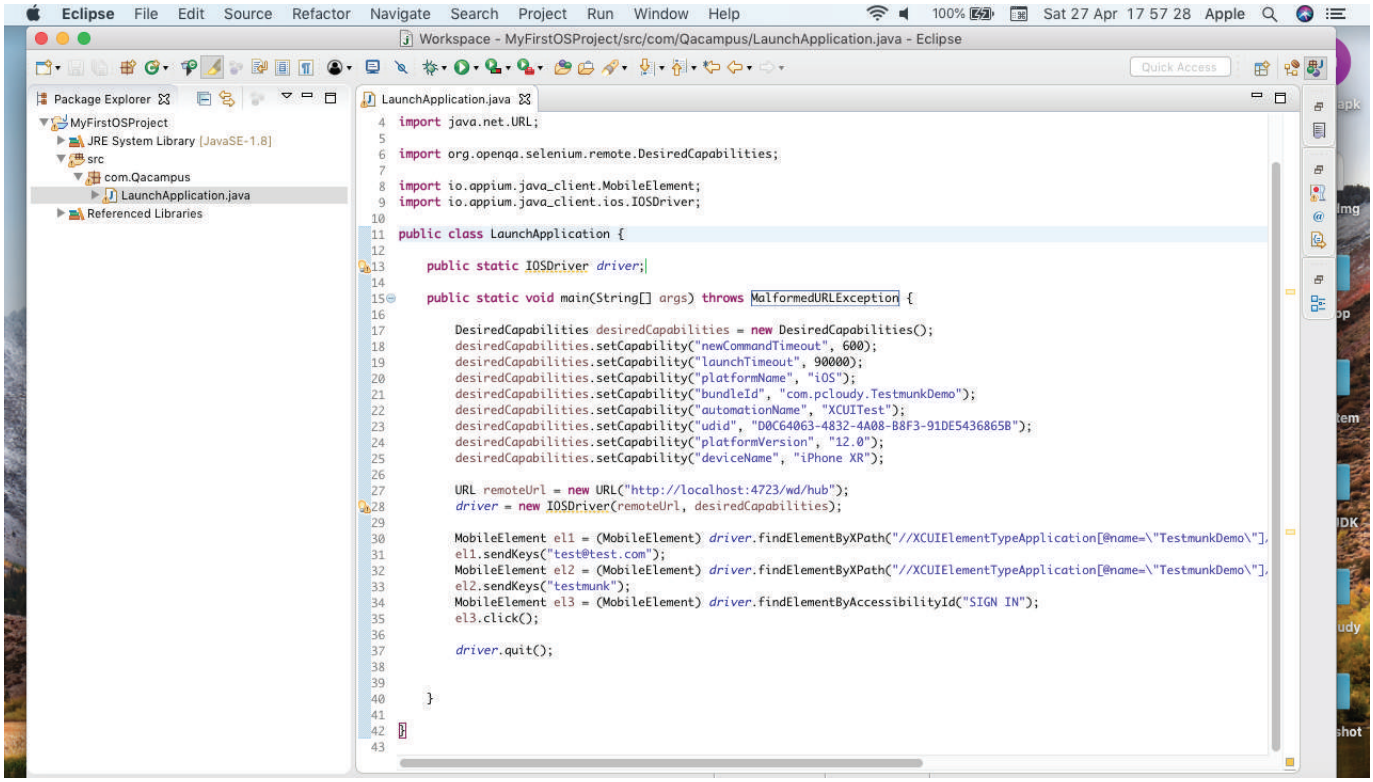
We have already shown how to record a script using Appium inspector. Now copy the recorded script and paste it into the class we have created.



In your first line of code, you need to define a public class variable of ***IOSDirver*** *and import the packages by hovering the mouse over them. You also need to import the packages for capabilities class. Delete the AppiumDriver and write* ***IOSDriver*** *instead.*



Click this button to execute the code; it will start the simulator and perform all the actions we have recorded.



STARTING APPIUM AND LAUNCHING THE APPLICATION FROM CODE - IOS

In this section, we will learn how to start the Appium server and how to provide the path of the .app file in the server to be installed on the device. Let's have a look at the code structure to handle the scenario.

```
public class AppiumserverTest {
public static void startAppiumServer()
{
}
public static void stopAppiumServer()
{
}
public static void main(String[] args)
{
startAppiumServer();
#Install application in device
#Launch App on device
stopAppiumServer();
}
}
```

The first section of the code takes care of starting and stopping the Appium server.

In this second part, which is the main method,

- We call the "**startAppiumServer()**" method to start the server
- We then provide the **path of .app** to be installed
- Set the capabilities of the application to be launched
- We finally call the "**stopAppiumServer()**" method

Now starting Appium from code requires the path of two files which are kept in the Appium folder.

Example:

C:\Program Files\Appium\resources\app\node_modules\nodejs\node.exe

C:\Program Files\Appium\resources\app\node_modules\appium\lib\main.js

Start and stop the Appium server

The Code to Start Appium Server

```
public class AppiumServer
{
public static void startAppiumServer() throws Exception
{
CommandLine command = new CommandLine
("/Applications/Appium.app/Contents/Resources/node/bin/node");
command.addArgument
("/Applications/Appium.app/Contents/Resources/node_modules/appium/bin/appium.js", false);
command.addArgument("--address", false);
command.addArgument("127.0.0.1");
command.addArgument("--port", false);
command.addArgument("4723");
```

In the code which is highlighted by the yellow box, we have called ***command line*** *class which is provided by Apache common's package. Within this command line, we have passed the Appium* ***"node"*** *and* ***"appium.js"*** *files which are highlighted in the green box. The box in blue contains the address of Apium server and the red box has the port of Appium server.*

This next picture is in continuation to the last picture of code we explained.

The Code to Start Appium Server

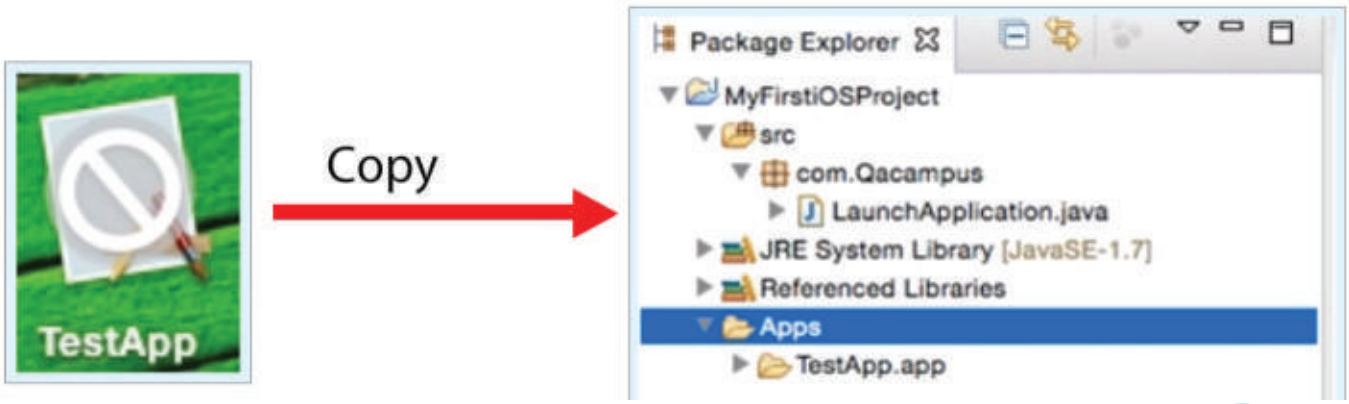
```
command.addArgument("4723");
command.addArgument("--no-reset", false);
DefaultExecuteResultHandler resultHandler = new DefaultExecuteResultHandler();
DefaultExecutor executor = new DefaultExecutor();
executor.setExitValue(1);
executor.execute(command, resultHandler);
Thread.sleep(7000);
}
public static void stopAppiumServer() throws Exception
{
Runtime.getRuntime().exec("killall node");
}
```

The code line highlighted in yellow ensures that the application does not reset on the device. The red box contains the classes derived from the Appium common package, that handle the process. In the second method we have called ***"getRuntime()"*** *method and execute the* ***"Killall node"*** *command. This method will close running Appium server instances.*

Let's try to install and launch an application on a simulator/device.

We will also see how to install a .app file on an iOS simulator/device and launch it automatically without providing the path in Appium server.

To install and launch the application, create a folder with the name of the apps in your Eclipse project folder and copy the application ***"TestApp.app"*** *file into your project.*



Let's have a look at the code and understand the functioning.

```
public static void main(String[] args) throws Exception {
stopAppiumServer();
startAppiumServer();
File appDir =new File(System.getProperty("user.dir"),"/apps/");
File app =new File(appDir,"TestApp.app");
DesiredCapabilities capabilities = new DesiredCapabilities();
capabilities.setCapability(MobileCapabilityType.BROWSER_NAME, "iOS");
capabilities.setCapability(MobileCapabilityType.PLATFORM_VERSION, "8.2");
capabilities.setCapability(MobileCapabilityType.PLATFORM_NAME, "iOS");
capabilities.setCapability(MobileCapabilityType.DEVICE_NAME, "iPhone Simulator");
capabilities.setCapability(MobileCapabilityType.APP, app.getAbsolutePath());
capabilities.setCapability("automationName", "Appium");
driver = new IOSDriver(new URL("http://127.0.0.1:4723/wd/hub"), capabilities);
Thread.sleep(3000);
stopAppiumServer();
```

*First, we call the **"stopAppiumServer()"** method to verify if any instances of Appium server is already running, if so, then it closes that instance. Store the path of the .app file into a variable of "File" class, which is a Java class as highlighted in the yellow box. in the green box we created another variable of "File" class "app". It takes two parameters, the absolute path of the .app file and name of the .app file.*

*The absolute path is stored in the **"appDir"** variable with the name of the .app file in the second part. The code line highlighted in red is the capability which gets the .app file. The rest of the things are the same as discussed earlier. So with this code, the Appium server will be started and the app will be launched on the device.*

The complete code.

```
package com.exceldemo;
import io.appium.java_client.ios.IOSDriver;
import io.appium.java_client.remote.MobileCapabilityType;


import java.io.File;
import java.net.URL;

import org.apache.commons.exec.CommandLine;
import org.apache.commons.exec.DefaultExecuteResultHandler;
import org.apache.commons.exec.DefaultExecutor;
import org.openqa.selenium.remote.DesiredCapabilities;
```

```
public class AppiumServer {

public static void startAppiumServer() throws Exception{

CommandLine command = new CommandLine
("/Applications/Appium.app/Contents/Resources/node/bin/node");

command.addArgument("/Applications/Appium.app/Contents/
Resources/node_modules/appium/bin/appium.js", false);

command.addArgument("--address", false);

command.addArgument("127.0.0.1");

command.addArgument("--port", false);

command.addArgument("4723");

DefaultExecuteResultHandler resultHandler = new DefaultExecuteResultHandler();

DefaultExecutor executor = new DefaultExecutor();

executor.setExitValue(1);

executor.execute(command, resultHandler);

Thread.sleep(7000);

}
```

About pCloudy

pCloudy.com, based out of California is a leading mobile app testing platform with more than 50,000 users across the globe. It is a one-of-its-kind full life cycle testing platform for mobile apps developers, QA and mobile DevOps teams. It offers tools for mobile DevOps, including test automation, manual testing, performance testing for web & mobile applications. It also enables mobile application testing across a large section of real mobile devices and seamlessly integrates with Continuous Integration tools.

Useful Links

Capabilities

https://github.com/pankyopkey/pCloudy-sample-projects/tree/master/Java/Advanced(Continued...)

Appium tips and tricks

https://www.pcloudy.com/developer-and-tester-forum/question-and-answers/appium-tips-and-tricks/

Appium TestNG Whitepaper

https://www.pcloudy.com/appium-testNG-framework-and-multi-device-automation-execution/landing-page.php

Retrieve APIs

https://content.pcloudy.com/apidocs/index.html

Appium integration architecture with pCloudy

https://www.youtube.com/watch?v=lrA6zYikFA4&t=41s

pCloudy certification for Appium

https://pcloudyacademy.talentlms.com/

Appium integration architecture documentation

https://www.pcloudy.com/mobile-application-testing-documentation/automation-testing/running-appium-scripts.html

Sample projects

https://github.com/pankyopkey/pCloudy-sample-projects/tree/master/Java