Test Automation
Lecture 16 –
JUnit HTML Reporting
Mobile Devices & Mobile Testing
Object Map Design Pattern

Lector: Milen Strahinski
Skype: strahinski
E-mail: milen.strahinski@pragmatic.bg
Facebook: http://www.facebook.com/LamerMan
LinkedIn: http://www.linkedin.com/pub/milen-strahinski/a/553/615

www.pragmatic.bg

Summary-overall

- Generating HTML test reports with JUnit
- Introduction of mobile testing
- Setting up the Android emulator for Selenium
- Setting up the Android device for Selenium
- Running tests using Selendroid
- Object Map Design Pattern

JUnit report – install ant

- Open the URL: http://ant.apache.org/bindownload.cgi
- On the Apache Ant Project page, find the heading Current Release of Ant.
- Download and unzip **apache-ant-X.X.X-bin.zip** [PGP] [SHA1] [MD5] in C:\ (for example)
- Add the ANT_HOME environment variable set to C:\ant
- Add the %ANT_HOME%\bin directory to your PATH environment variable.
- Check ant is working in CMD with: ant -version

JUnit report – Configure Eclipse

- 1. Navigate to Window -> Preferences -> Ant -> Runtime -> Under Runtime you'll see an entry for Ant Home Entries. Click the button "Ant Home" and browse to the directory where you have unzipped ANT distribution -> click “Apply”

- 2. The next thing you have to do is resolve your dependencies by making sure that the junit.jar is added to Ant’s “Global Entries”. Navigate in Eclipse to Window -> Preferences -> Ant-> Runtime -> Global Entries and click Add External JARs… Navigate to your “eclipse” directory (where eclipse is installed) under the “plugins” directory and verify the junit jar is under “org.junit_3.8.1” directory to avoid errors.

JUnit report – Configure Project (step 1)

- 1. Right click on the project and select Export –> Ant Buildfiles (this will be under General) –> Next –> Select the project that contains your JUnit tests –> Finish.
  - The default JUnit output directory is appropriately named “junit”.

JUnit report – Configure Project (step 2)

- 2. Finally, right-click on the Ant build file build.xml (this file is the one you created in Step 1) and “Run As -> Ant Build…” This will display a list of targets. All of the launch configurations you have previously configured will have a corresponding target in your Ant build file. Select the desired target(s), and also select the “junitreport” target (very important). Check the “Target execution order” text area to make sure the junitreport is last.

Introduction to mobile testing

- With the increasing adoption of smartphones and tablets, mobile applications have taken a center stage. Everyone is talking about iPhone, iPad, and Android. It has become essential to build/migrate and test applications for these platforms.

- We can run automated tests on a simulator/emulator or on a real device.

Installation & Configuration

- Download latest Android SDK
  - http://developer.android.com/sdk/index.html
- Installing Eclipse ADT Plugin (optional)
  - http://developer.android.com/sdk/installing/installing-adt.html
- Install Universal Adb Driver if your devices is not recognized:
  - http://download.clockworkmod.com/test/UniversalAdbDriverSetup6.msi
- Add Android platforms and other components to your SDK by using the SDK Manager
- Configure the ANDROID_HOME environment variable based on the location of the Android SDK. Additionally, consider adding ANDROID_HOME/tools, and ANDROID_HOME/platform-tools to your PATH.

Configure your mobile phone settings

- On your phone go to Settings -> Developer Options
  - Stay awake
  - USB Debugging
  - Allow mock locations
- For Android 4.4 KitKat it's hidden under About Phone -> taping few times on "Build Number"

adb (Android Debug Bridge)

- Open cmd and type: adb
- adb devices – checks the currently connected devices

We want emulator to work faster!

- With the latest SDK the HAXM is available through the SDK Manager, if it's not – you can find it as shown below:
- Tip: Set Intel (x86) processor for the device
- Tip: For Windows: install HAXM (Hardware Accelerated Execution Manager – Android SDK Manager extra)

You have HAXM already under:

C:\Path\To\AndroidSDK\sdk\extras\intel\Hardware_Accelerated_Execution_Manager

Required jars in Eclipse buildpath

Downloaded from http://selendroid.io:

- selendroid-client-0.13.jar
- selendroid-standalone-0.13.0-with-dependencies.jar

and the selenium library:

- selenium-server-standalone-2.44.0.jar

Selendroid - introduction

- Full compatibility with the JSON Wire Protocol that the deprecated AndroidDriver was based on.
- No modification of app under test required in order to automate it
- Testing the mobile web using built in Android driver webview app
- Same concept for automating native or hybrid apps
- UI elements can be found by different locator types
- Gestures are supported: Advanced User Interactions API
- Selendroid can interact with multiple Android devices (emulators or hardware devices) at the same time
- Existing Emulators are started automatically
- Selendroid supports hot plugging of hardware devices
- Full integration as a node into Selenium Grid for scaling and parallel testing
- Multiple Android target API support (10 to 19)
- Built in Inspector to simplify test case development.

Selendroid – tests support (part 1)

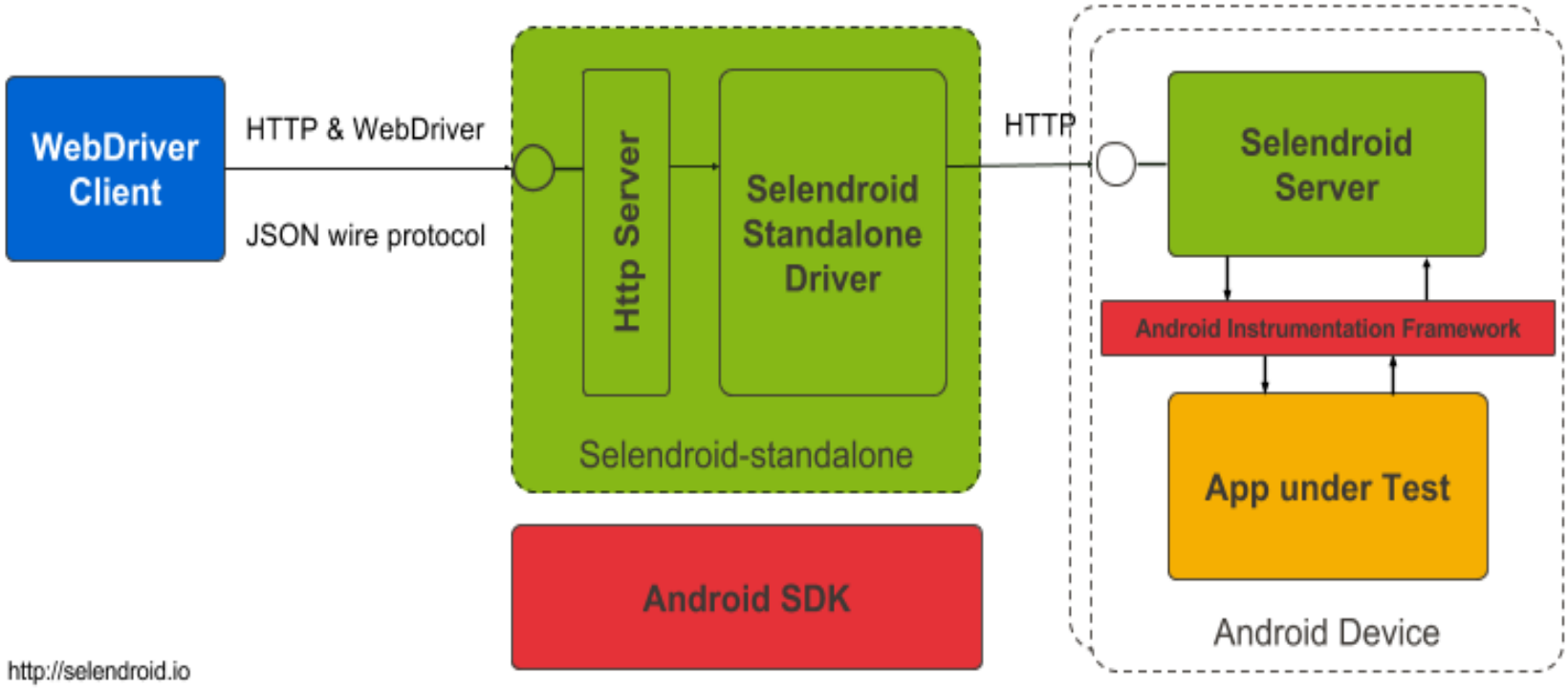
- **Native apps** live on the device and are accessed through icons on the device home screen. Native apps are installed through an application store (such as Google Play or Apple's App Store). They are developed specifically for one platform, and can take full advantage of all the device features — they can use the camera, the GPS, the accelerometer, the compass, the list of contacts, and so on.

- **Mobile Web apps** are not real applications, they are really **websites** that, in many ways, *look and feel* like native applications, but are not *implemented* as such. They are run by a browser and typically written in HTML5. Users first access them as they would access any web page: they navigate to a special URL and then have the option of "installing" them on their home screen by creating a bookmark to that page.

Selendroid – tests support (part 2)

- **Hybrid apps** are part native apps, part web apps. (Because of that, many people incorrectly call them “web apps”). Like native apps, they live in an app store and can take advantage of the many device features available. Like web apps, they rely on HTML being rendered in a browser, with the caveat that the browser is embedded within the app.

Selendroid architecture



Application Under Test (AUT)

- Selendroid can be used to test already built apps. Those Android apps (apk file) must exist on the machine, where the *selendroid-standalone* server will be started. The reason for this is that a customized *selendroid-server* for the app under test (AUT) will be created. Both apps (selendroid-server and AUT) must be signed with the same certificate in order to install the apks on the device.

Launching Selendroid

- To launch Selendroid for your AUT execute the cmd below:
  - java -jar selendroid-standalone-0.12.0-with-dependencies.jar -app selendroid-test-app-0.12.0.apk -forceReinstall
- Selendroid-standalone will start a http server on port 4444 and will scan all Android virtual devices (avd) that the user has created (~/.android/avd/). The Android target version and the screen size will be identified. If an emulator is running, it can be used since version *0.9.0*. Even an emulator that has been started manually after the selendroid-standalone got started can be used. If there are Android hardware devices plugged in, they will also be added to the device store.
- You can check that the application(s) and the devices are recognized by opening a browser and navigating to:
  http://localhost:4444/wd/hub/status

Java doc for mobile interactions

- org.openqa.selenium.interactions.touch
  - DoubleTapAction
  - DownAction
  - FlickAction
  - LongPressAction
  - MoveAction
  - ScrollAction
  - SingleTapAction
  - TouchActions
  - UpAction

Some Action examples


```
WebElement elem = driver.findelement(By.id("name"));
Action tchAct = new TouchActions(driver).doubleTap(elem).build();
tchAct.perform();


tchAct = new TouchActions(driver).singleTap(elem).build();
tchAct.perform();


tchAct = new TouchActions(driver) .flick(element,0,-
400,FlickAction.SPEED_NORMAL).build();
TchAct.perform();
```

Inspect Mobile Elements

- https://developers.google.com/chrome-developer-tools/docs/remote-debugging
- http://tech.gilt.com/post/49530971053/mobile-web-how-to-inspect-elements-on-androids
- In Chrome browser on your PHONE. Open Chrome -> Chrome Settings -> Developer Tools -> USB Web Debugging must be ON (if still available with new browser versions of Chrome, it might be enabled by default)

Selendroid Inspector

- As soon as your server is up and running you can find it under:
  - http://localhost:4444/inspector

Lets run the examples

- Lets run the example files – they are all based on the downloaded *.apk files from http://selendroid.io

Object Map (part 1)

- So far, we have seen how the Selenium WebDriver API needs locator information to find the elements on the page. When a large suite of tests is created, a lot of locator information is duplicated in the test code. It becomes difficult to manage locator details when the number of tests increases. If any changes happen in the element locator, we need to find all the tests that use this locator and update these tests. This becomes a maintenance nightmare.

- One way to overcome this problem is to use page objects and create a repository of pages as reusable classes.

Object Map (part 2)

- There is another way to overcome this problem — by using object map. An object or a UI map is a mechanism that stores all the locators for a test suite in one place for easy modification when identifiers or paths to GUI elements change in the application under test. The test script then uses the object map to locate the elements to be tested.

- Now lets check the ObjectMap.rar

Course Feedback

- Please, share your overall feedback at the bottom of the course page at:

http://pragmatic.bg/courses/automated-testing-course/

HIGHLY APPRECIATED! ☺

Course Certificates

☺ Congratulations! ☺