Test Automation Lecture 16 –

Maven HTML Reporting
Mobile Devices & Mobile Testing
Object Map Design Pattern

IT Learning & Outsourcing Center

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 Maven
- 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

Generate Maven HTML Test Report (part 1)

- We will use the maven-surefire-report-plugin in order to have Maven generate the report for us
- We will also use the maven-jxr-plugin for which will help us later on analyzing the failing tests
- In the next slide we will add the plugins into our pom.xml

Generate Maven HTML Test Report (part 2)

- 1) Put that under </dependencies> and save

добави след затварящия таг на </dependencies>

```
<reporting>
        <plugins>
                <plugin>
                        <groupId>org.apachea.maven.plugins</groupId>
                        <artifactId>maven-surefire-report-plugin</artifactId>
                </plugin>
                <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-jxr-plugin</artifactId>
                </plugin>
        </plugins>
</reporting>
```

Generate Maven HTML Test Report (part 3)

- 2) Now open Command Prompt and navigate into your workspace and then into the project you would like to have a HTML report generated.

- 3) Once you're inside the projects' directory, type:

mvn clean site

Which will first clean the "target" directory and then run all the tests in that project and then maven-surefire-report-plugin will populate the new test run results in HTML format under the "target" directory.

Generate Maven HTML Test Report (part 4)

- Now open the "target" directory under your project and then navigate into the "site" directory
- Open the file named "index.html" in a browser and you will be able to see the "Project Reports" link and the "Surefire Reports" under it.

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.

Selendroid & Appium - open source test automation framework

Installation & Configuration

- Download latest Android SDK
  - http://developer.android.com/sdk/index.html
- Install Universal Adb Driver if your devices is not recognized:
  - http://download.clockworkmod.com/test/UniversalAdbDriverSetup6.msi
  - http://developer.android.com/sdk/win-usb.html
  - http://developer.android.com/tools/extras/oem-usb.html#InstallingDriver
- 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.

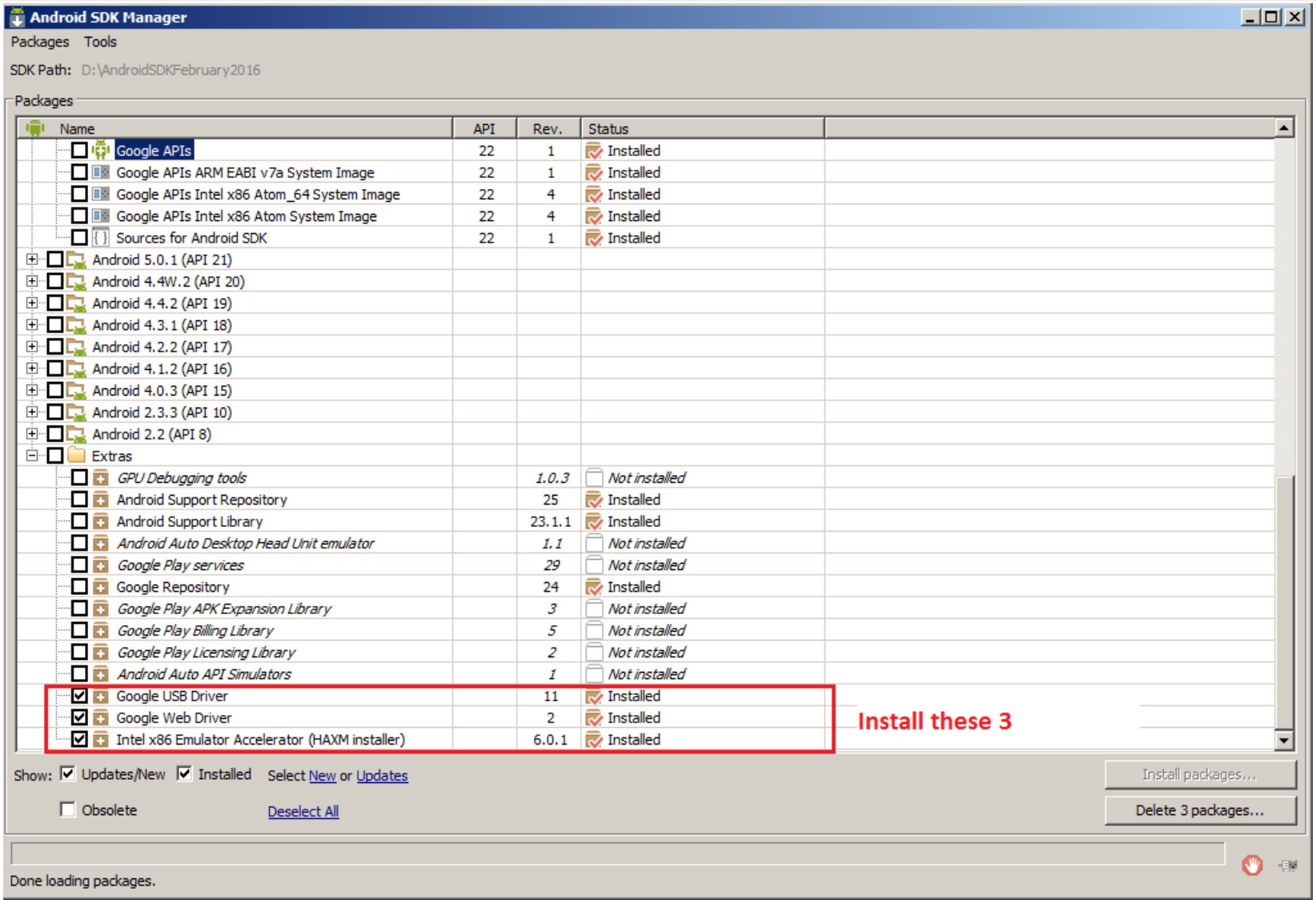
Configure your mobile phone settings

- 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 which have the USB degbugging option enabled

Accelerator and other drivers



Required jars in Eclipse buildpath

Downloaded from http://selendroid.io:

- selendroid-client-X.XX.jar
- selendroid-standalone-X.XX.X-with-dependencies.jar

and the selenium library:

- selenium-server-standalone-X.XX.X.jar

- Also make sure to download the APK files(which are the applications we test) from selendroid.io.

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)

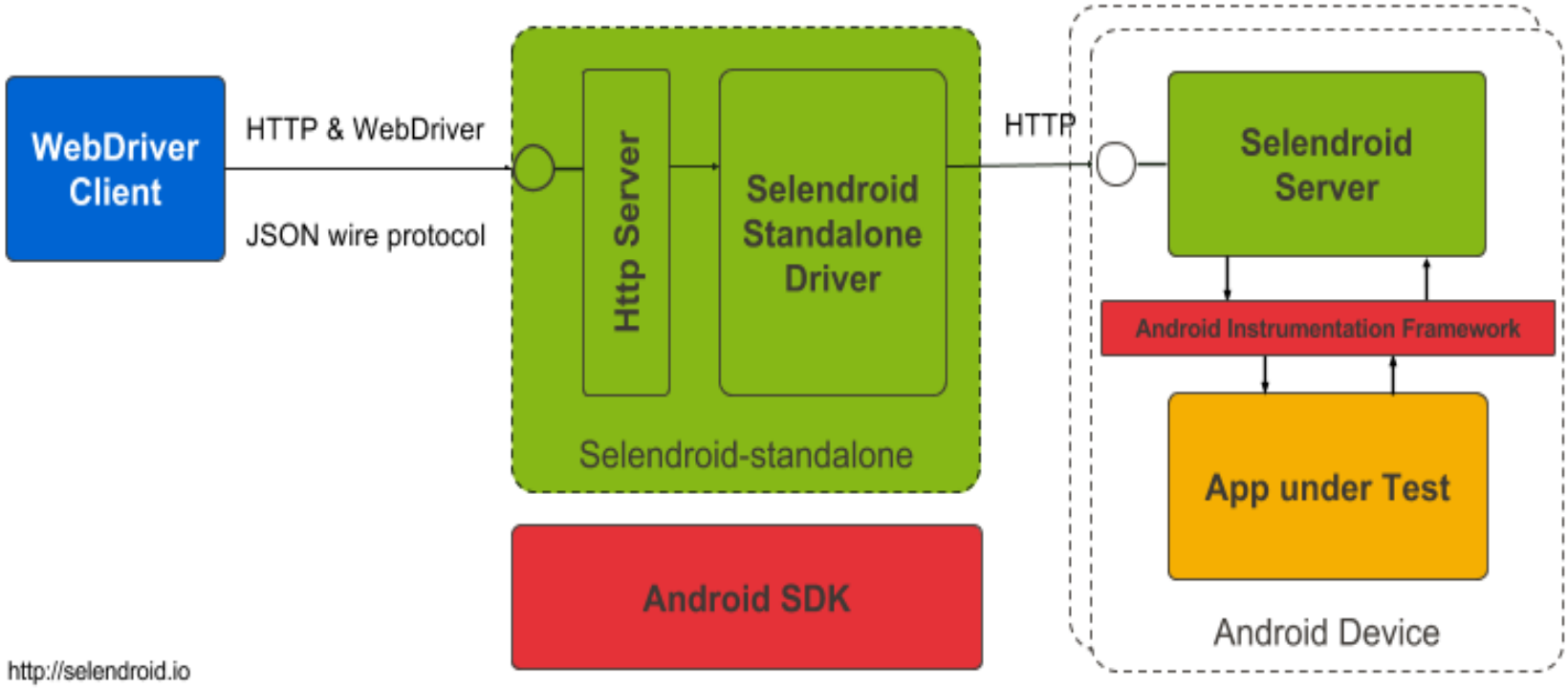
- **Native apps** live on the device and are accessed through icons on the device home screen. Native apps are installed through an application store (such as Google Play or Apple's App Store). They are developed specifically for one platform, and can take full advantage of all the device features — 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-X.X.X-with-dependencies.jar –app selendroid-test-app-X.X.X.apk
  -app employee-directory.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();
```

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! ☺