Create a project using Eclipse+Maven+Selenium+Robot Framework using Page Object Model

Pre-requisites:

- Java JDK (version 1.8.0_131 used in example project).
- Python (version 3.4 used in example project).
- Eclipse IDE (Mars.2 used in example project).
- If Eclipse version doesn't include Maven, then in Eclipse go to Help > Install new software > Add > Name: "M2E", Location: http://download.eclipse.org/technology/m2e/releases or http://download.eclipse.org/technology/m2e/milestones/1.0 > OK > Check Maven Integration for Eclipse > Next > Accept license > Finish.
- If project will be run outside the IDE, then install Maven as well. MAVEN_HOME environment variable should be set to the Maven folder path and Path environment variable should have an entry to Maven's bin folder path.
- Chrome/Firefox browser.
- RED editor for Eclipse (optional) (https://marketplace.eclipse.org/category/free-tagging/robotframework)

Example project

Example project can be downloaded from
https://github.com/patr1c1a/robotFramework-bloggerWebsite-
The example project was created using Windows 10 - 64 bit, with Chrome browser version 63.0.3239.132 and Firefox Quantum browser version 57.0.4. Chromedriver and Geckodriver for Windows 64 bit.

Brief description of steps

1. Create Maven project
2. Add Selenium and Robot dependencies, and the Robot Framework Maven Plugin to pom.xml
3. Create a suitable project structure
4. Download browser drivers and save them in the project
5. Create custom keyword to set the path to browser drivers
6. Create setup files
7. Write keywords and tests

Project creation

To avoid having to install Robot Framework and Selenium libraries, Maven will be used to manage dependencies automatically.

1. Within Eclipse, go to New > Project > Maven > Maven Project. Check "create a simple project", add a group ID, artifact ID and create project. The new project structure will contain some default folders and a pom.xml file. We'll be using src/main/java, src/test/resources and we'll create a new one: src/test/robotframework.

2. Open pom.xml and add <dependencies> </dependencies> tags before the closing </project> tag. Also create <build><plugins> </plugins></build> tags before the closing </project> tag.
    a. Go to http://www.seleniumhq.org/download/maven.jsp. Copy the latest selenium 3 dependency and paste inside <dependencies> tags in pom.xml.
    b. Go to https://github.com/Hi-Fi/robotframework-seleniumlibrary-java and find the selenium 3 dependency, then copy and paste it inside <dependencies> tags in pom.xml.
    c. Go to http://robotframework.org/MavenPlugin/ and copy the robot plugin information and paste it inside <build/><plugins/> tags in pom.xml.
    d. Save pom.xml and right click on project > Maven > Update project. Maven dependencies will now show up in project explorer.

The SeleniumLibrary by Hi-Fi is a fork of the Selenium2Library by Markus Bernhardt (currently deprecated) and provides a Java implementation of the Robot Framework.

3. Browser drivers need to be downloaded separately, as they are exe files and thus won't be managed by Maven. Download your browser drivers from Selenium downloads page (http://www.seleniumhq.org/download) and place the exe files in /src/test/resources. Keep in mind that Selenium 3 uses Geckodriver as an implementation for the Firefox browser driver. If Firefox needs to be used, then download the Geckodriver from https://github.com/mozilla/geckodriver/releases and place it in your src/test/resources folder. In this example, ChromeDriver version 2.33 and GeckoDriver version 0.19.1 were used.

4. To enforce UTF-8 encoding, go to Window > Preferences > General > Workspace and change "Text file encoding" to UTF-8.

5. In project explorer, right click on /src/main/java and create a package with your project name (in this case it's called automated_tests). Then right click on it and create a .java file named CustomKeywords.java. This will allow us to implement keywords.

6. In project folder src/main/test create a new directory called robotframework/acceptance. The actual structure of directories and files contained in robotframework/acceptance will depend on the application structure. In the case of a simple website, one directory per website page would be suitable to follow the Page Object Model. In this example, a directory to store tests for the home page is created in src/main/test/robotframework/acceptance/homepage. This will hold a file containing all tests to be run in the "homepage" suite, as well as some resource and setting files that will affect those specific tests. Whenever this project needs to be extended to other pages (or other components in the application under test) then more directories will have to be created under the acceptance directory, each one with their own tests, resources and settings files.

7. Right click on the src/main/test/robotframework/acceptance directory and add New > File to add robot tests. Name this file CommonResources.robot. This will provide references to libraries being used globally.

8. Right click on the src/main/test/robotframework/acceptance/homepage directory and add New > File named __init__.robot. This file is needed to perform test and suite setup.

9. Right click on the src/main/test/robotframework/acceptance/homepage directory and add New > File named Resources.robot. This will be used to store variables and high level keywords.

10. Right click on the src/main/test/robotframework/acceptance/homepage directory and add New > File named PageObjects.robot. This is where all “locators” and some other variables will be stored, to reference each element of the home page.

11. Right click on the src/main/test/robotframework/acceptance/homepage directory and add New > File named Tests.robot. This is where the actual Robot Framework tests will be stored.

Custom Keywords

To be able to use the browser driver you need to implement your own keyword that sets to the driver path. For this, in CustomKeywords.java within /src/main/java/automated-tests, add the following method (a method will match a keyword using the method name, in this case, “Browser Setup”) :

```
public class Setup {
   public void browserSetup() {
    System.setProperty("webdriver.chrome.driver",
   "src\\test\\resources\\chromedriver.exe");
    System.setProperty("webdriver.gecko.driver",
   "src\\test\\resources\\geckodriver.exe");
    }
}
```

Adding tests

Libraries

To write our tests we’ll be using keywords provided by Hi-Fi’s SeleniumLibrary (keyword documentation can be found at https://github.com/Hi-Fi/robotframework-seleniumlibrary-java) and our own high-level robot keywords.

In order to use SeleniumLibrary and our own keyword library (CustomKeywords.java, which we added during project creation), we need to indicate this in a Settings section. We could do this in any file that contains tests, but it’s a good practice to place settings outside these files so they can affect a suite globally. For this purpose, we’ll have a file named CommonResources.robot inside the src/test/robotframework/acceptance directory (since the libraries will need to be accessed by all tests). There will be only one CommonResources.robot file in the project. Each Tests.robot files will need to reference CommonResources.robot using the `Resource` keyword.

A simple example would be as follows:

- *Example CommonResources.robot contents*:

```
*** Settings ***
Library                         SeleniumLibrary
Library                         automated_tests.CustomKeywords
```

- *Example Tests.robot contents*:

```
*** Settings ***
Resource                        Resources.robot
```

```
Resource                        PageObjects.robot
Resource                        ../CommonResources.robot
```

In this case, CommonResources.robot is being used to determine which libraries must be globally included (SeleniumLibrary and our own, CustomKeywords.java).

Test suite settings

__init__.robot

Other settings (like actions to be performed before and after each test run) will be placed in another file: __init__.robot. There will be one of these per each directory that represents a component of the system under test (in this case, a page in the website). So we'll have one of these files in the src/main/test/robotframework/acceptance/homepage directory, and the same will apply whenever we add more website pages to be tested by our automation suite.

In this case, we don't need to reference the __init__ file like we did with CommonResources through the Resource keyword, as it will be automatically detected.

- *Example __init__.robot contents*:

```
*** Settings ***
Test Setup                      Run Keywords  Browser Setup  AND  Open
Browser  ${HOMEPAGE URL}  ${BROWSER}
Test Teardown                   Close Browser
```

In __init__.robot we're placing setup actions that must be performed before each test is run: in this example, our own keyword, included in CustomKeywords.java, called "Browser Setup" (which will set up the browser driver path) and opening a browser to the homepage url stored in a variable in the PageObjects.robot file. Also, we indicate which actions must be performed after each test finishes running: in this example, "Close Browser" which is a SeleniumLibrary keyword. We could add more settings if needed, for example, the Test Timeout keyword (also from the SeleniumLibrary) allows to indicate the timeout before a test is considered as failed.

PageObjects.robot

This file will only contain variables that represent each element in the page under test (in this case, the home page). This is where we'll add our locators and other variables that will be used in higher level keywords.

We can use `$(variablename)` to include variables that can be used as arguments. For instance, the browser can be set using a variable here (either "chrome" or "firefox" can be used, since those are the drivers we set up). This variable will be used as an argument when calling our custom Open Browser keyword implemented in CustomKeywords.java. Also, the url of the homepage under test can be also set in a variable, as well as other urls to be tested in links or menus.

- *Example PageObjects.robot contents*:

```
*** Variables ***
#objects
${BROWSER}                      chrome
${HOMEPAGE URL}              http://www.blogger.com
```

```
${HOMEPAGE TITLE}                    Blogger.com - Create a unique and beautiful
blog. It's easy and free.

#locators
${HEADER MAIN LOGO}                  css = div.header--logo.logo a.ga-header-logo
${HEADER LOGO}                       css = div.header--logo.logo svg.logo-icon
${HEADER SITE BRAND}                 css = svg.logo-type use
```

Resources.robot

This is where definitions for higher level keywords used by the Home Page tests will live.

- *Example Resources.robot contents*:

```
*** Keywords ***
Open Home Page
  [Documentation]                    Home page is displayed, with its page title.
  Go To                              ${HOMEPAGE URL}
  Maximize Browser Window
  Title Should Be                    ${HOMEPAGE TITLE}

Main Logo Has Image And Text
  [Documentation]                    Main logo is composed of image and brand.
  Page Should Contain Element        ${HEADER MAIN LOGO}
  Page Should Contain Element        ${HEADER LOGO}
  Page Should Contain Element        ${HEADER SITE BRAND}
```

Tests.robot

This is where the actual home page tests will be placed. Also, this is where the other files will be referenced (except for __init__.robot, which is automatically detected in the same directory). Tests will be implemented using Robot Framework keywords or our own high level keywords, contained in Resources.robot.

Each test case can also have optional tags (0, 1 or more tags). These are used to group tests in order to categorize tests (and run only some of them, depending on your needs, by changing the value of robot-tag within the Maven Plugin set up in pom.xml). There are different ways to add a tag: only for a particular test, by adding `[tags] tagA tagB` below the test name or forcing a tag to all testcases in your file by placing a "force tags" in settings: `Force Tags tagA tagB`. To run tests from a specific tag, edit Maven's pom.xml and add a <properties /> first-level element (within <project />) with a property name of your choosing and the tag you want to run, like this:

```
<properties>
  <robot-tag>mytag</robot-tag>
</properties>
```

Then, in the plugins section, within the robotframework-maven-plugin plugin element, add this:

```
<configuration>
  <includes>
    <include>${robot-tag}</include>
  </includes>
</configuration>
```

- *Example Tests.robot contents:*

```
*** Settings ***
Resource          Resources.robot
Resource          PageObjects.robot
Resource          ../CommonResources.robot

*** Test Cases ***
Main Header Displayed In Homepage
  [Tags]          regression  smoke
  Open Home Page
  Main Header Shows Up
  Main Logo Has Image And Text
```

Example project structure

```
automated_tests
      |-- src
      | |-- main
      | |    |-- java
      | |    |      +-- automated_tests
      | |    |            +-- CustomKeywords.java
      | |    +-- resources
      | +-- test
      |      |-- java
      |      |-- resources
      |      |      |-- chromedriver.exe
      |      |      |-- geckodriver.exe
      |      +-- robotframework
      |            +-- acceptance
      |                  +-- homepage
      |                        |-- __init__.robot
      |                        |-- PageObjects.robot
      |                        |-- Resources.robot
      |                        +-- Tests.robot
      |                  +-- CommonResources.robot
      +-- target
      +-- test-output
      +-- pom.xml
```

- **pom.xml**: used by Maven to determine which libraries and settings the project will be running. These settings include a property called <robot-tag> that will allow to choose which tests to run, according to their tags.
- **target directory**: it will include the Robot Framework reports (html format) after tests are run, with full description of executed keywords, pass and fail results and screenshots of failed keywords. It will also include details such as at what time each keyword was ran, how long it took, etc.
- **src/main/java/automated_tests/CustomKeywords.java**: used to implement custom keywords using Java. In this case, there is a method to set the path to browser driver needed by Selenium.
- **src/test/resources**: directory that stores browser drivers needed by Selenium.
- **src/test/robotframework/acceptance**: directory where tests are stored. It's advisable to have different test suites related to each application component or feature. In this case, since only the home page of a website is being tested, all tests are included in a folder called "homepage", following the Page Object Pattern.

- **src/test/robotframework/acceptance/CommonResources.robot**: contains references to libraries that are needed by more than one test suite.
- **src/test/robotframework/acceptance/homepage/__init__.robot**: file used by the homepage test suite to set up and tear down every test run.
- **src/test/robotframework/acceptance/homepage/PageObjects.robot**: contains variables (for homepage test suite), locators used to select elements from the DOM and language identifiers. There is a variable that allows to choose in which browser to execute tests (only Chrome and Firefox are supported in the present project), should more browser drivers be added.
- **src/test/robotframework/acceptance/homepage/Resources.robot**: contains high level keyword implementation. These keywords provide a higher level of abstraction so the tests are simple to understand.
- **src/test/robotframework/acceptance/homepage/Tests.robot:** contains all test cases, along with a settings section to reference resource files.

Running project

Instructions to run project within Eclipse:

1. Within Eclipse: File > Import > Existing project into workspace > "Select root directory": [choose automated_tests project directory] > Finish
2. Right click on project folder > Run As > Maven install. If dependencies are not fully downloaded, right click on project directory > Maven > Update Project.

Instructions to run project from command line (Maven must be installed):

1. Open console/terminal (Windows: CMD)
2. Change directory to the project directory: **cd C:/directory/automated_tests**
3. Run command: **mvn clean install**