Requirement

Install HomeBrew

(is like one marketplace of the mac)

How to install HomeBrew:

- /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"

Blade CLI

(is one command helpful to Liferay products)

Java JDK 1.8+ must be installed.

- brew install wget
- wget https://github.com/jpm4j/jpm4j.installers/raw/master/dist/biz.aQute.jpm.run.jar
- sudo java -jar biz.aQute.jpm.run.jar -g init
- sudo /usr/local/bin/jpm install -f https://releases.liferay.com/tools/blade-cli/latest/blade.jar
- Test if Blade is installed:
    - blade version

iTerm2 with OhMyZSH

ITerm2 Page Download:

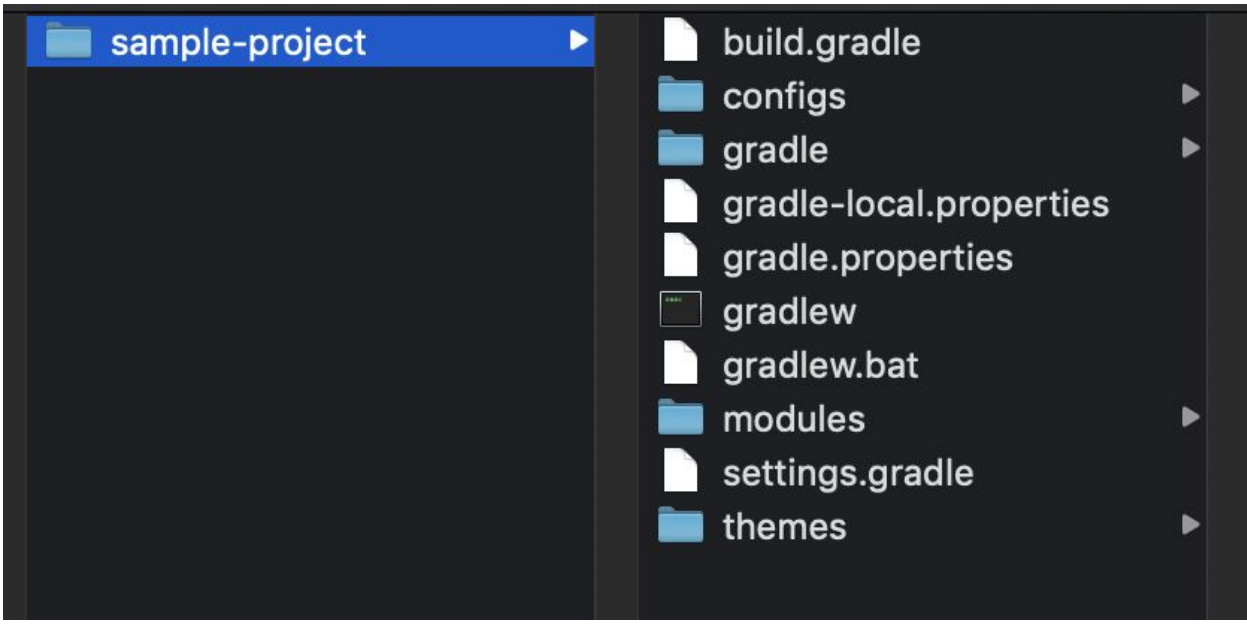
- https://www.iterm2.com/
- How to install OhMYZSH
    - sh -c "$(curl -fsSL https://raw.github.com/robbyrussell/oh-my-zsh/master/tools/install.sh)"

Full Project

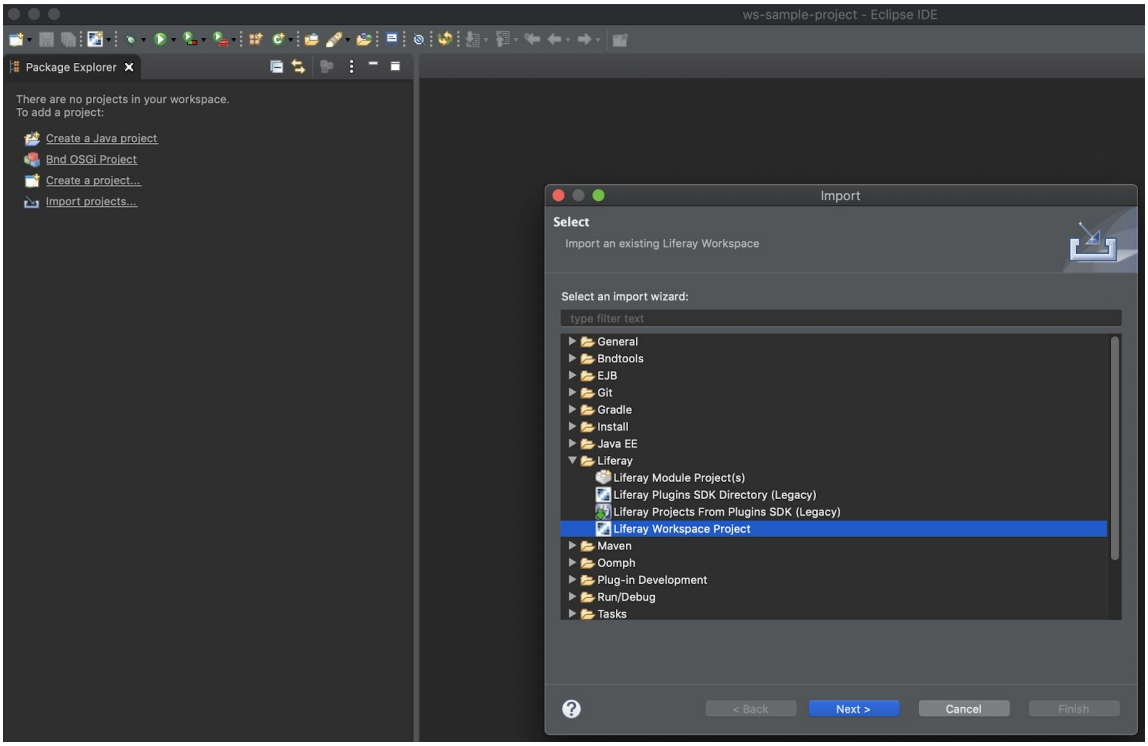
You can find this source project on: https://github.com/diegotomfurtado/sample-project

Create and Prepare the Environment

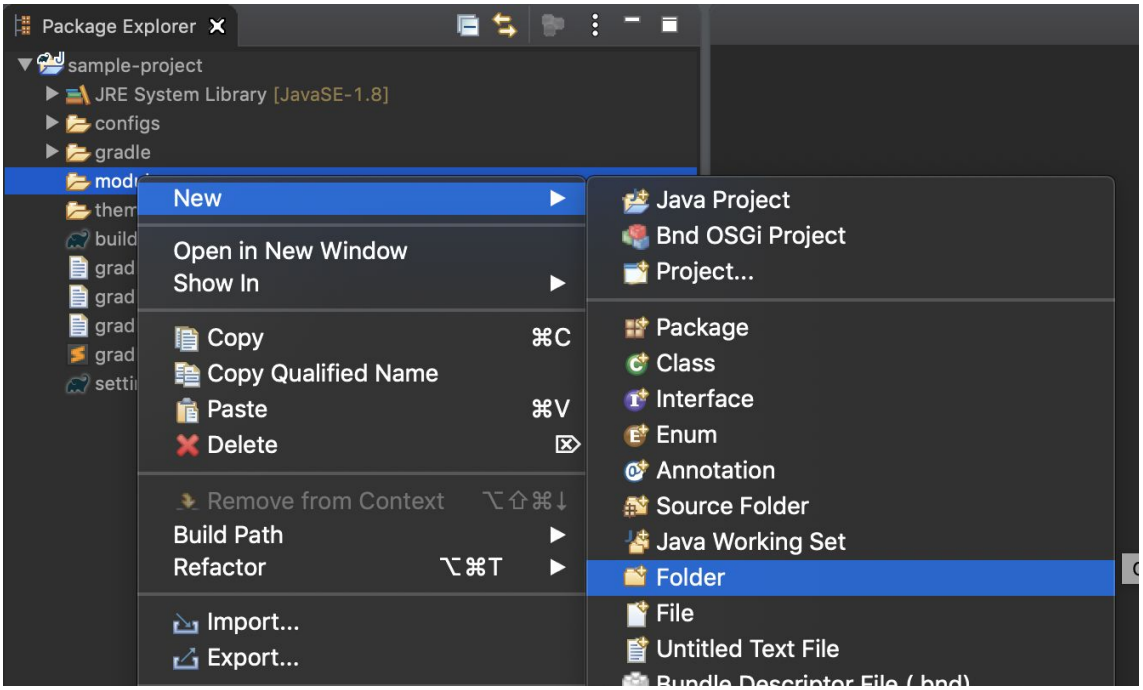
1. **Open your terminal / iTerm**
2. **Create the Liferay workspace**
    a. Go to the place where will be your workspace
    b. Type: blade init -v 7.2 sample-project
    c. And then your workspace will be created like this:

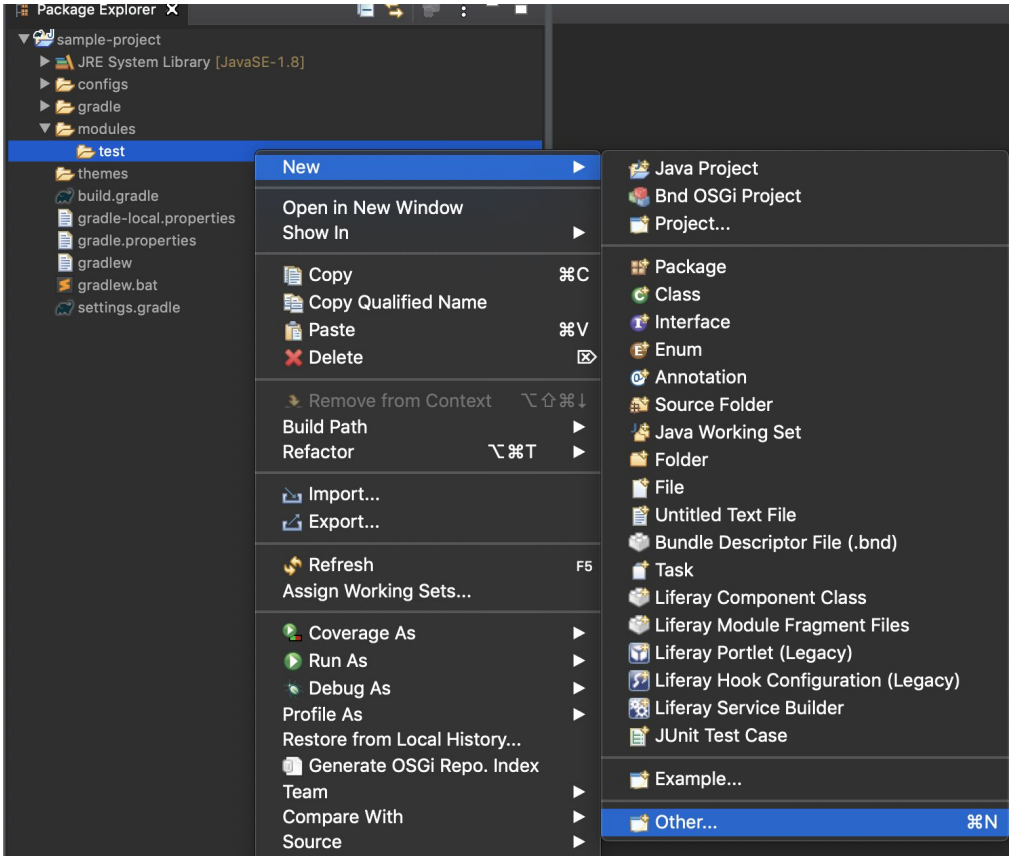


3. **Open your IDE and then import your Liferay workspace.** If you don't see this option, you should download the Liferay Plugin into Marketplace.

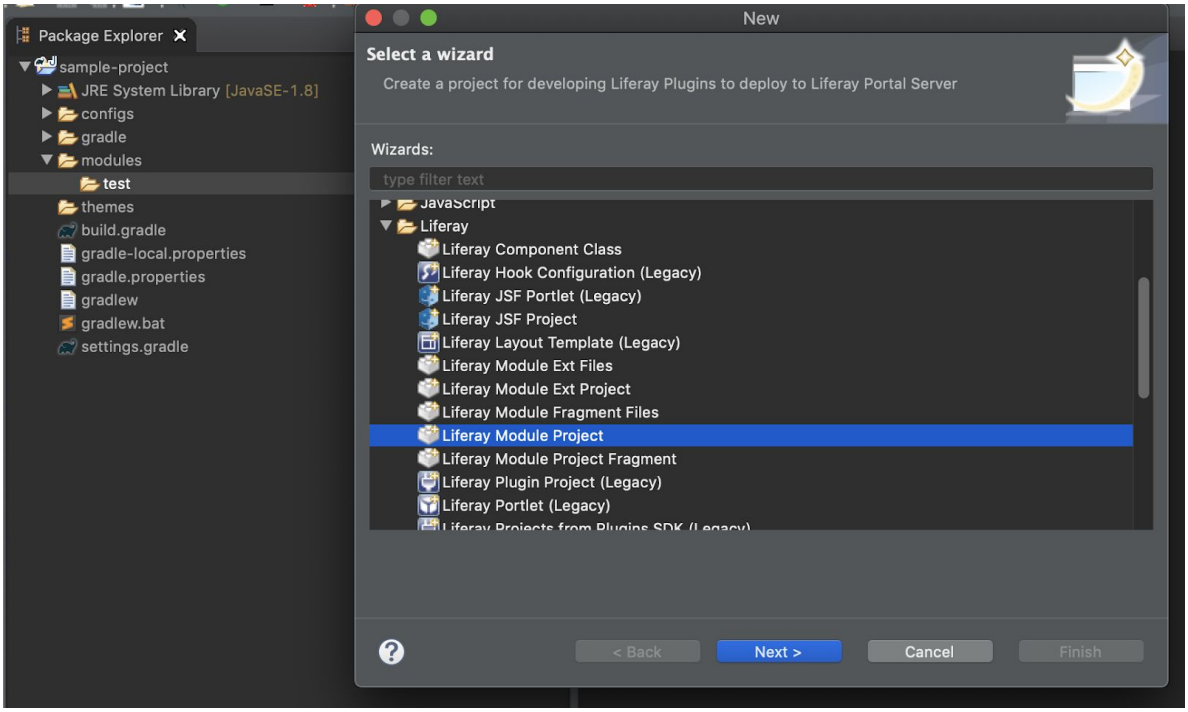
a. 

4. Go to "modules" folder and create another folder called: test

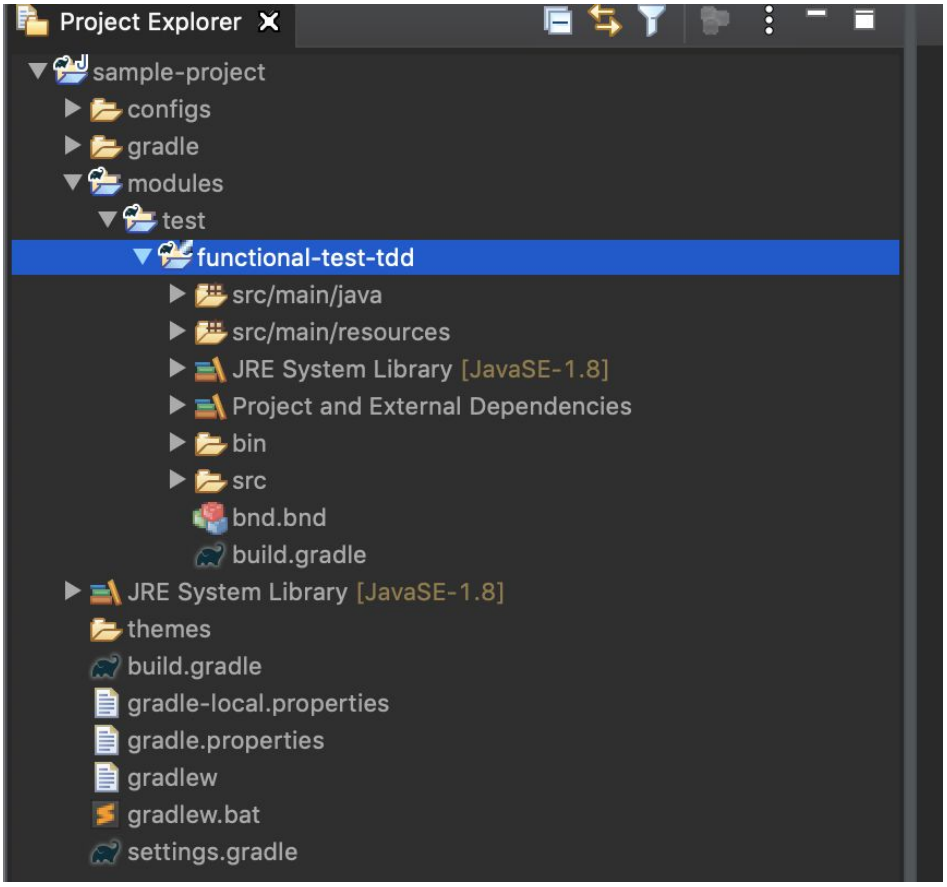
a. 

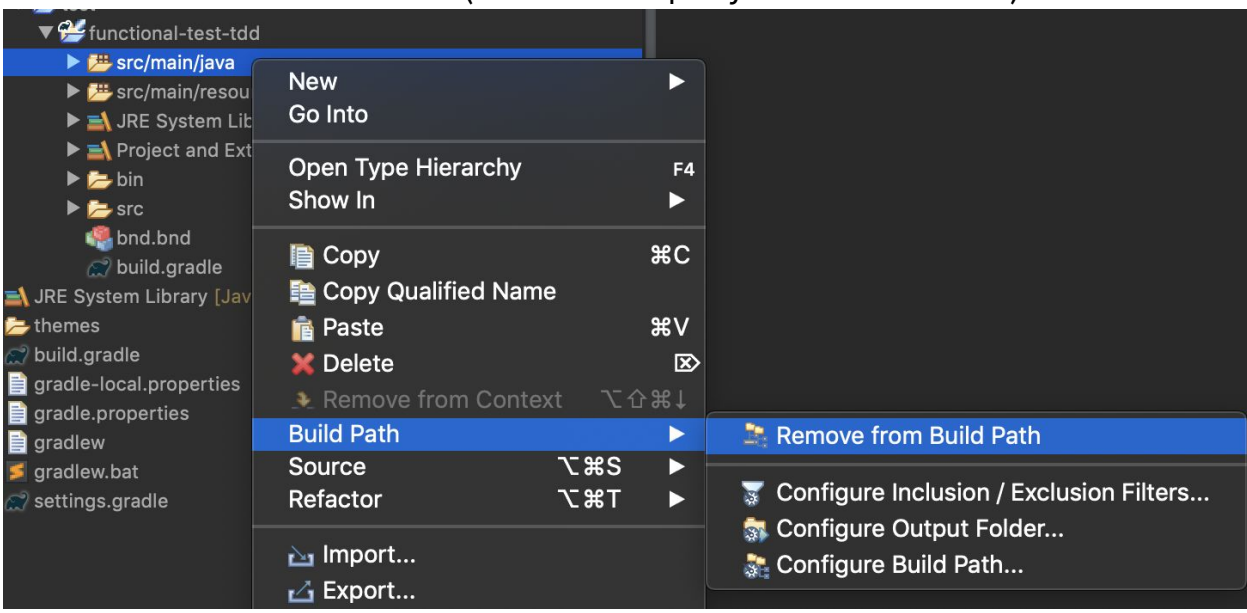
5. **Go to test folder and then create a Liferay Modules Project:**
    a. click on [test] with right button > new > other > choose Liferay Modules Project > on Project Name type: functional-test-tdd > move to [test] level > click on Next > click on Finish > Go to [sample-project] with the right button > Gradle > Refresh Gradle Project.
    b. 

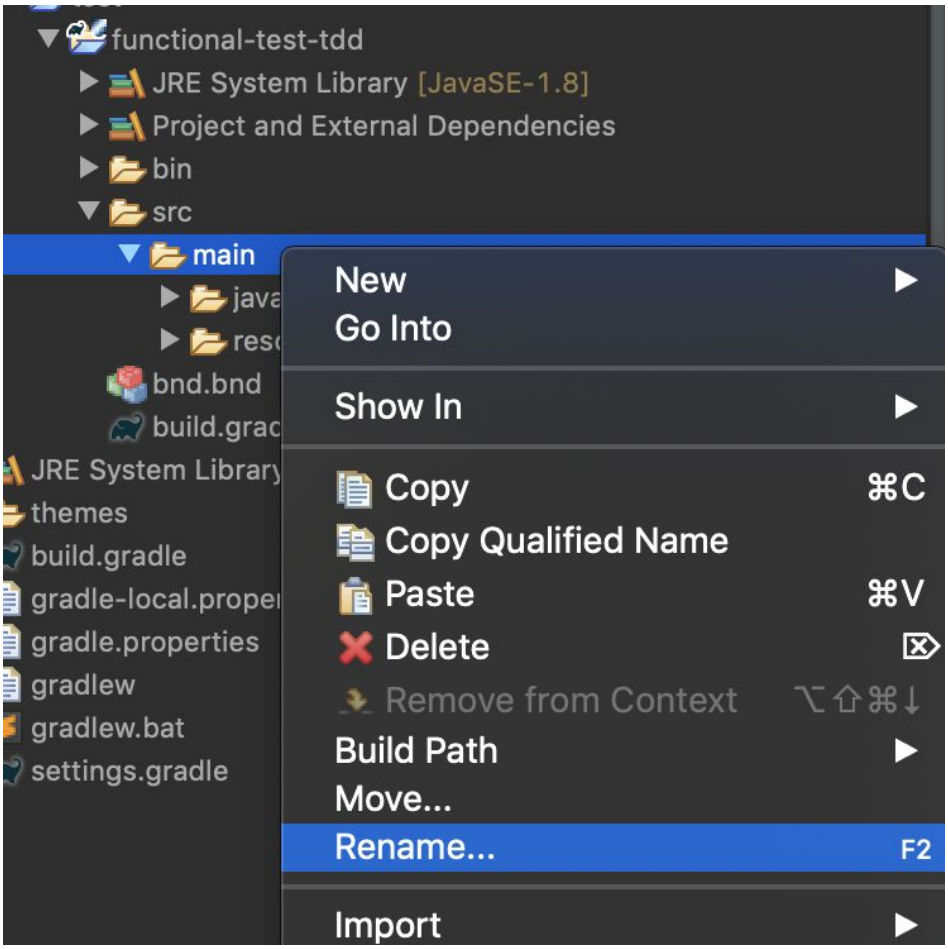
c.


Package Explorer
sample-project
JRE System Library [JavaSE-1.8]
configs
gradle
modules
test
themes
build.gradle
gradle-local.properties
gradle.properties
gradlew
gradlew.bat
settings.gradle
New
Select a wizard
Create a project for developing Liferay Plugins to deploy to Liferay Portal Server
Wizards:
type filter text
JavaScript
Liferay
Liferay Component Class
Liferay Hook Configuration (Legacy)
Liferay JSF Portlet (Legacy)
Liferay JSF Project
Liferay Layout Template (Legacy)
Liferay Module Ext Files
Liferay Module Ext Project
Liferay Module Fragment Files
Liferay Module Project
Liferay Module Project Fragment
Liferay Plugin Project (Legacy)
Liferay Portlet (Legacy)
< Back
Next >
Cancel
Finish

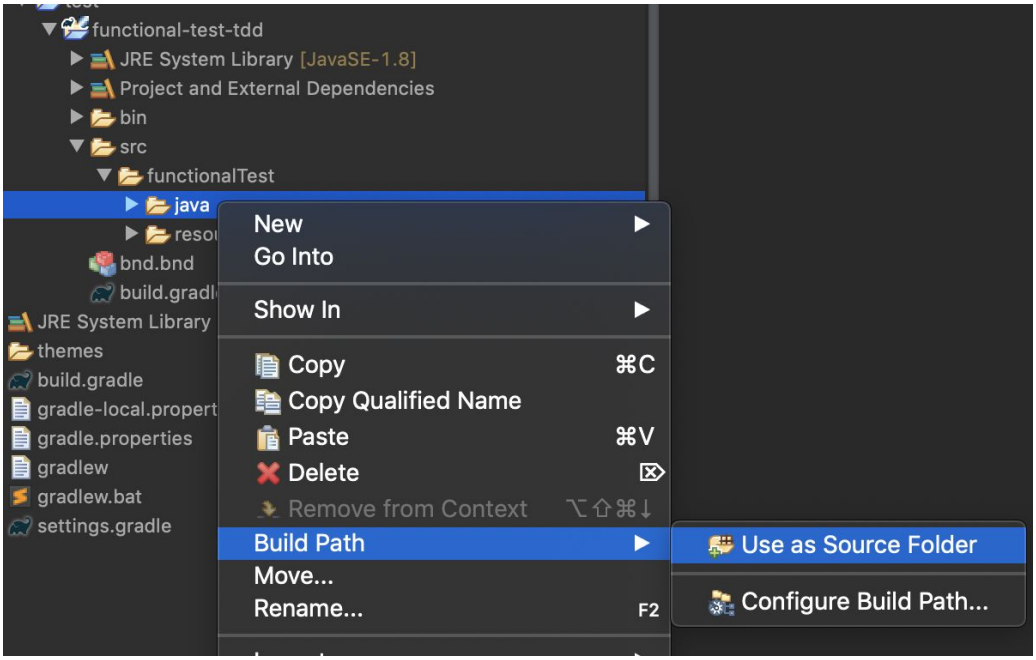
d.


Project Explorer
sample-project
configs
gradle
modules
test
functional-test-tdd
src/main/java
src/main/resources
JRE System Library [JavaSE-1.8]
Project and External Dependencies
bin
src
bnd.bnd
build.gradle
JRE System Library [JavaSE-1.8]
themes
build.gradle
gradle-local.properties
gradle.properties
gradlew
gradlew.bat
settings.gradle

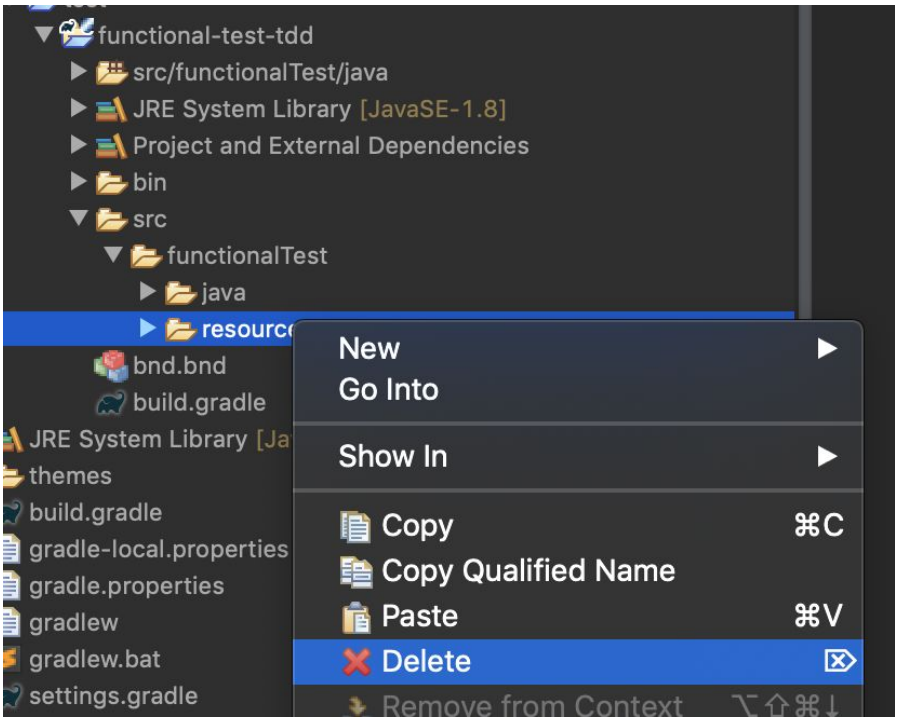
6. **Rename source folder name**
    a. From: src/main/java to src/functionalTest/java
    b. Remove the resource folder (in this example you will not use it)
    c. 

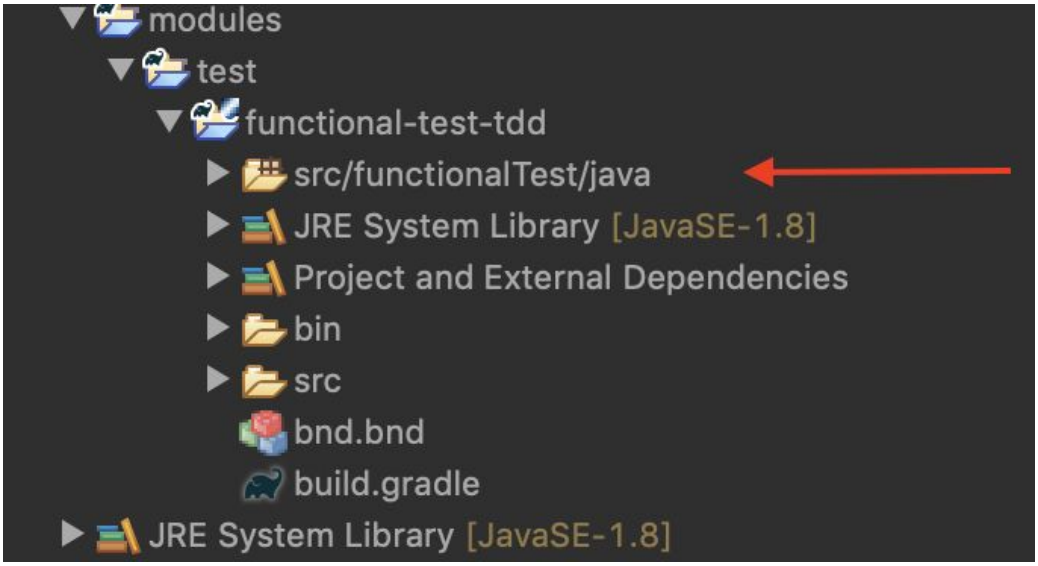
    d. 

e.

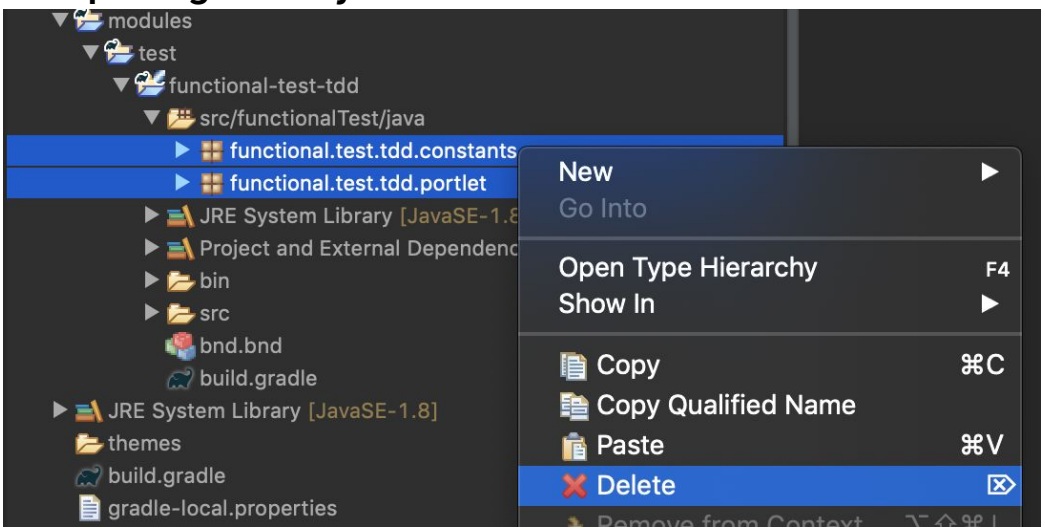

functional-test-tdd
JRE System Library [JavaSE-1.8]
Project and External Dependencies
bin
src
functionalTest
java
bnd.bnd
JRE System Library
themes
build.gradle
gradle-local.propert
gradle.properties
gradlew
gradlew.bat
settings.gradle
New
Go Into
Show In
Copy ⌘C
Copy Qualified Name
Paste ⌘V
Delete
Remove from Context
Build Path
Move...
Rename... F2
Use as Source Folder
Configure Build Path...

f.

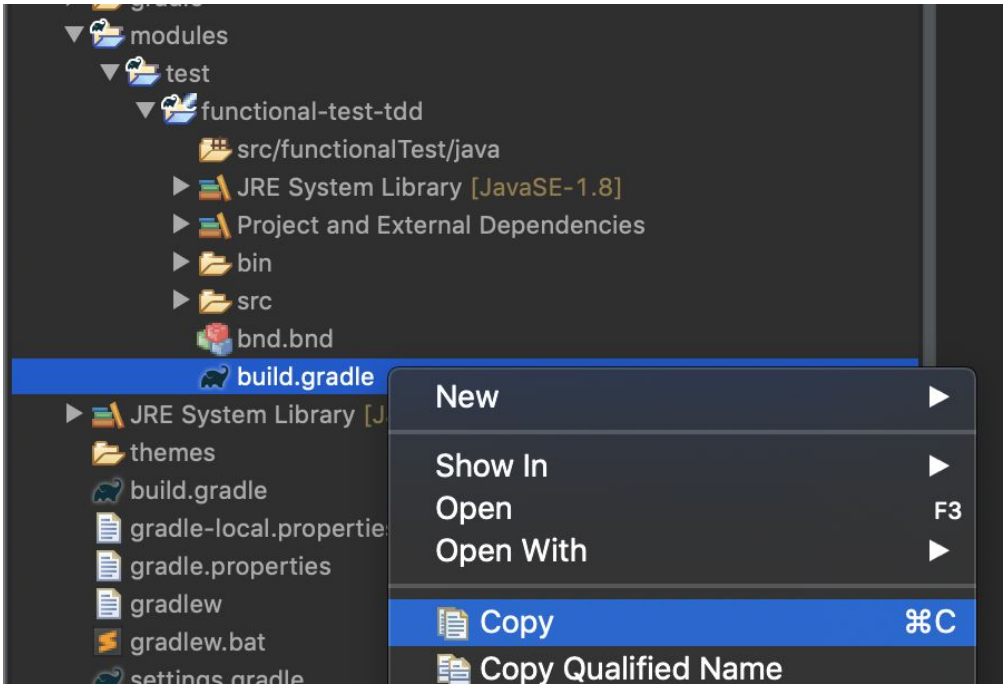

functional-test-tdd
src/functionalTest/java
JRE System Library [JavaSE-1.8]
Project and External Dependencies
bin
src
functionalTest
java
bnd.bnd
build.gradle
JRE System Library
themes
build.gradle
gradle-local.properties
gradle.properties
gradlew
gradlew.bat
settings.gradle
New
Go Into
Show In
Copy ⌘C
Copy Qualified Name
Paste ⌘V
Delete

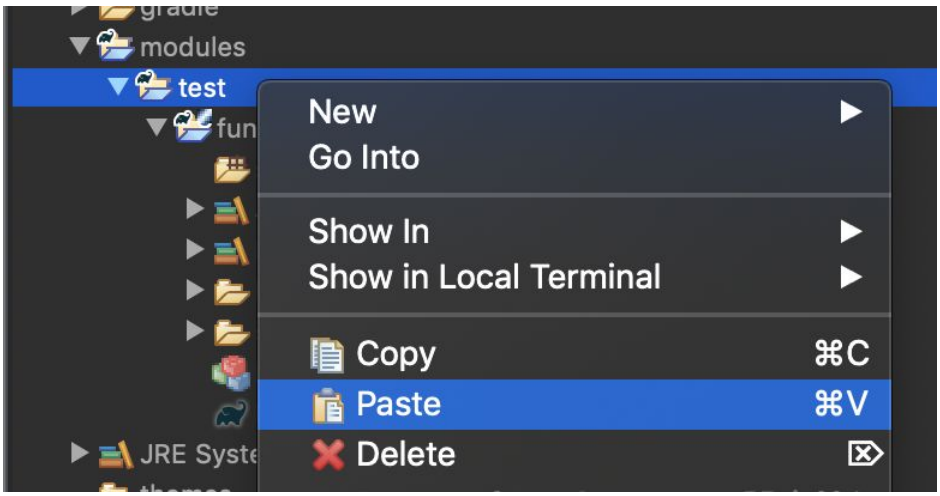
g. 

7. **Delete the packages that you will not use:**

a. 

8. **Create another build.gradle on [test] level.**
   a. Go to [functional-test-tdd] and copy the build.gradle
   b. Go to [test] and paste

c. 

d. 

9. **Update the content of build.gradle, from [test]**
    a. Replace everything that was there, with:

```
plugins {
    id "de.undercouch.download" version "3.1.2"
}

repositories {
    maven {
```

```
                url "https://repository-cdn.liferay.com/nexus/content/groups/public"
        }
}

apply plugin: 'java'
apply plugin: 'maven'
apply plugin: 'maven-publish'
apply plugin: 'eclipse'

sourceCompatibility = "1.8"
targetCompatibility = "1.8"

ext {
        frwSeleniumCommonsVersion = '3.0.0'
        seleniumVersion = '3.3.1'
}

configure(subprojects.findAll { !it.subprojects }) {
        afterEvaluate { project ->
                dependencies {

                        compile                'com.liferay.gs:frw-selenium-commons:'                +
frwSeleniumCommonsVersion
                        compile 'org.codehaus.groovy:groovy-all:3.0.3'

                        compile group: "org.json", name: "org.json", version: "2.0"
                        compile group: 'org.easetech', name: 'easytest', version: '0.6.3'
                        compile group: 'org.easetech', name: 'easytest-core', version: '1.4.0'
                        compile group: 'com.google.guava', name: 'guava', version: '22.0'

                        testCompile group: "junit", name: "junit"
                        testCompile group: 'pl.pragmatists', name: 'JUnitParams', version: '1.0.4'
```

```
                testCompile group: 'org.hamcrest', name: 'java-hamcrest', version: '2.0.0.0'

                compile 'org.seleniumhq.selenium:selenium-api:' + seleniumVersion
                compile 'org.seleniumhq.selenium:selenium-java:' + seleniumVersion
                compile 'org.seleniumhq.selenium:selenium-remote-driver:' + seleniumVersion
                compile 'org.seleniumhq.selenium:selenium-support:' + seleniumVersion
            }

            deploy {
                enabled = false
            }
        }

        tasks.withType(Test) {
            testLogging {
                events "passed", "skipped", "failed", "standardOut"
                showExceptions true
                exceptionFormat "full"
                showCauses true
                showStackTraces true

                afterSuite { desc, result ->
                    if (!desc.parent) {
                        def output = "Results: ${result.resultType} (${result.testCount} tests, ${result.successfulTestCount} successes, ${result.failedTestCount} failures, ${result.skippedTestCount} skipped)"
                        def startItem = '|  ', endItem = '  |'
                        def repeatLength = startItem.length() + output.length() + endItem.length()
                        println('\n' + ('-' * repeatLength) + '\n' + startItem + output + endItem + '\n' + ('-' * repeatLength))
                    }
                }
```

```
        }
    }

    tasks.withType(Test) {
        reports.html.destination = file("${reporting.baseDir}/${name}")
    }
}
```

10. **Update the content of build.gradle, from [functional-test-tdd]**
    a. Replace everything that was there, with:

```
import org.apache.tools.ant.taskdefs.condition.Os

jar.enabled = false

group = 'com.liferay.samples'
version = '1.0.0-SNAPSHOT'

repositories {
   jcenter()
   mavenCentral()
   mavenLocal()
}

dependencies {
    testCompile group: "io.takari.junit", name: "takari-cpsuite", version: "1.2.7"
}

sourceSets {
   functionalTest {
```

```
        java {
            compileClasspath += main.output + test.output
            runtimeClasspath += main.output + test.output
            srcDir file('src/functionalTest/java')
        }
        resources.srcDir file('src/functionalTest/resources')
    }
}

configurations {
    functionalTestCompile.extendsFrom testCompile
    functionalTestRuntime.extendsFrom testRuntime
}

// Command to run this task: ./gradlew performTest
task performTest(type: Test) {
    description 'Runs the functional Tests without the BDD flow.'

    testClassesDirs = sourceSets.functionalTest.output.classesDirs
    classpath = sourceSets.functionalTest.runtimeClasspath

            filter {
                    includeTestsMatching "RunAll*"
            }

    outputs.upToDateWhen { false }
}

// Command to run this task: ./gradlew performTestClass -PclassToBeTested=<Class Name>
// Example: ./gradlew performTestClass -PclassToBeTested=SampleTest
task performTestClass(type: Test) {
            description 'Run the Specific Functional Tests without the BDD flow.'
```

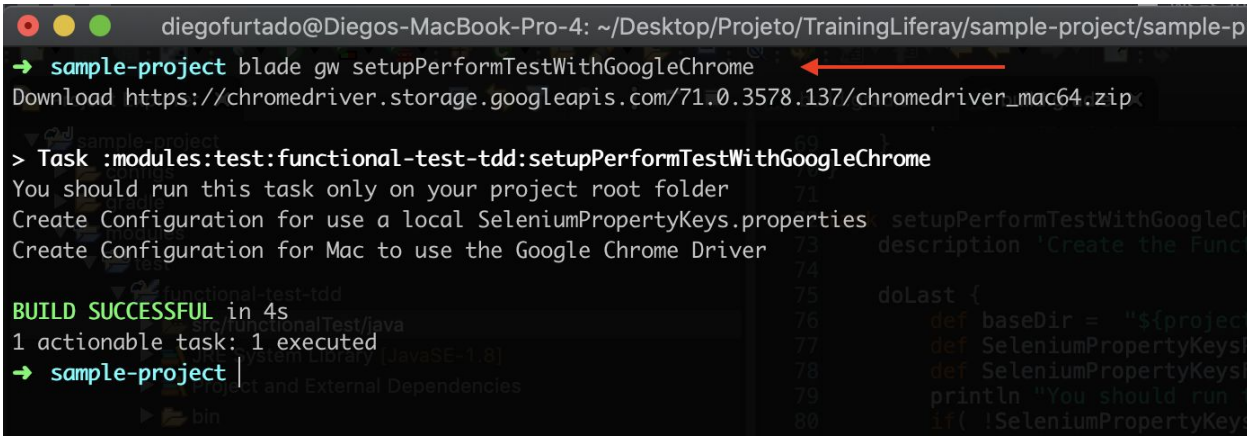
```
        testClassesDirs = sourceSets.functionalTest.output.classesDirs
        classpath = sourceSets.functionalTest.runtimeClasspath

        scanForTestClasses = false

        if (project.hasProperty('classToBeTested')==false){
        } else {
                includes = ['**/'+project.getProperty('classToBeTested')+'.class']
        }

        outputs.upToDateWhen { false }
}

gradle.taskGraph.whenReady { graph ->
   if (graph.hasTask(build)) {
      performTest.enabled = false
   }
}

task setupPerformTestWithGoogleChrome {
    description 'Create the Functional Tests properties and Chrome Driver, for Functional Test without BDD flow, but only work if run on project root folder.'

        doLast {
                def baseDir = "${project.hasProperty('liferay.workspace.home.dir') ? project.property('liferay.workspace.home.dir') : 'modules'}"
                def SeleniumPropertyKeysPath = new File( 'modules/test/functional-test-tdd/SeleniumProperties' )
                def SeleniumPropertyKeysFile = new File( 'modules/test/functional-test-tdd/SeleniumProperties/SeleniumPropertyKeys.properties' )
                println "You should run this task only on your project root folder"
                if( !SeleniumPropertyKeysFile.exists() ) {
```

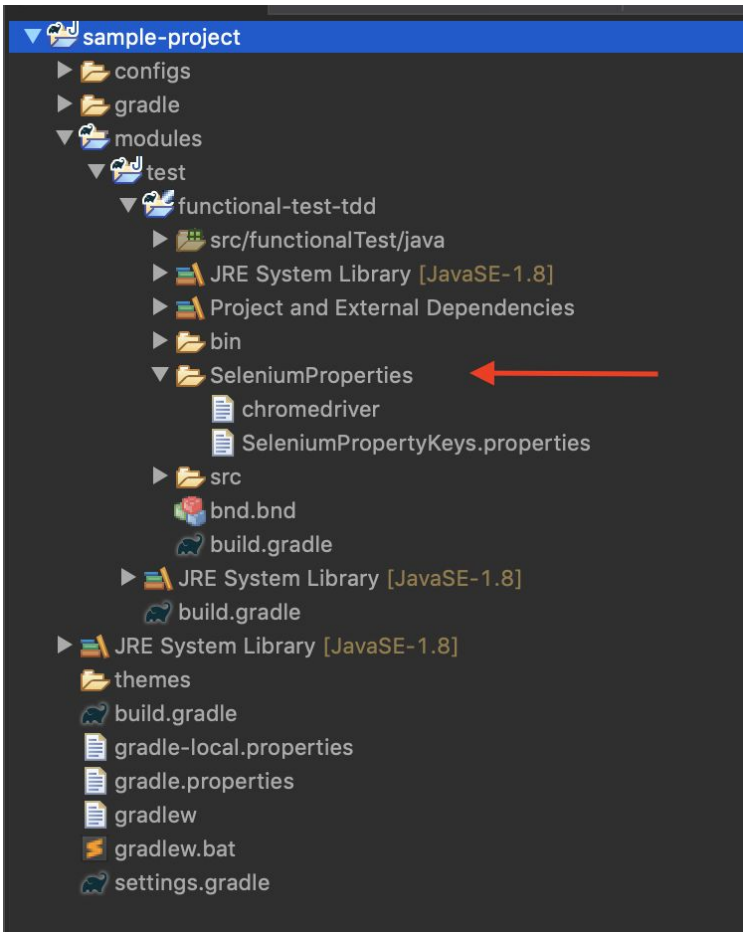
```
            println "Create Configuration for use a local SeleniumPropertyKeys.properties"
            SeleniumPropertyKeysPath.mkdirs()
            SeleniumPropertyKeysFile.withWriterAppend { w ->
                w <<  "browser=defaultGC\ntime-out=10\nusername=test"
            }
        } else {
            println "You already had the local SeleniumPropertyKeys.properties created"
        }

        if ( (Os.isFamily(Os.FAMILY_WINDOWS)) && (!new File( baseDir + '/test/functional-test-tdd/SeleniumProperties/chromedriver.exe').exists()) ) {
            println "Create Configuration for Windows to use the Google Chrome Driver"
            download {
                src 'https://chromedriver.storage.googleapis.com/71.0.3578.137/chromedriver_win32.zip'
                dest new File(SeleniumPropertyKeysPath, 'chromedriver_win32.zip')
            }

            copy {
                def DriverPath = new File ('/SeleniumProperties/')
                from zipTree('/SeleniumProperties/chromedriver_win32.zip')
                into "SeleniumProperties/"
            }
            delete 'SeleniumProperties/chromedriver_win32.zip'

        } else if ( (Os.isFamily(Os.FAMILY_MAC)) && (!new File( baseDir + '/test/functional-test-tdd/SeleniumProperties/chromedriver').exists()) ) {
            println "Create Configuration for Mac to use the Google Chrome Driver"
            download {
                src 'https://chromedriver.storage.googleapis.com/71.0.3578.137/chromedriver_mac64.zip'
                dest new File(SeleniumPropertyKeysPath, 'chromedriver_mac64.zip')
            }
```

```
                copy {
                        def DriverPath = new File ('/SeleniumProperties/')
                        from zipTree('SeleniumProperties/chromedriver_mac64.zip')
                        into "SeleniumProperties/"
                }
                delete 'SeleniumProperties/chromedriver_mac64.zip'

        } else if ( (Os.isFamily(Os.FAMILY_UNIX)) && (!new File( baseDir + '/test/functional-test-tdd/SeleniumProperties/chromedriver').exists()) ) {
                println "Create Configuration for Unix to use the Google Chrome Driver"
                download {
                        src 'https://chromedriver.storage.googleapis.com/71.0.3578.137/chromedriver_linux64.zip'
                        dest new File(SeleniumPropertyKeysPath, 'chromedriver_linux64.zip')
                }

                copy {
                        from zipTree(new File('SeleniumProperties/', 'chromedriver_linux64.zip'))
                        into "SeleniumProperties/"
                }
                delete 'SeleniumProperties/chromedriver_linux64.zip'
        } else {
                println "You already had the local Google Chrome Driver created"
        }
    }
}
```

11. **Go to [sample-project], click on the right button > Gradle > Refresh Gradle Project**

12. **Open you Terminal / iTerm**
    a. Go to the root of the project [sample-project]

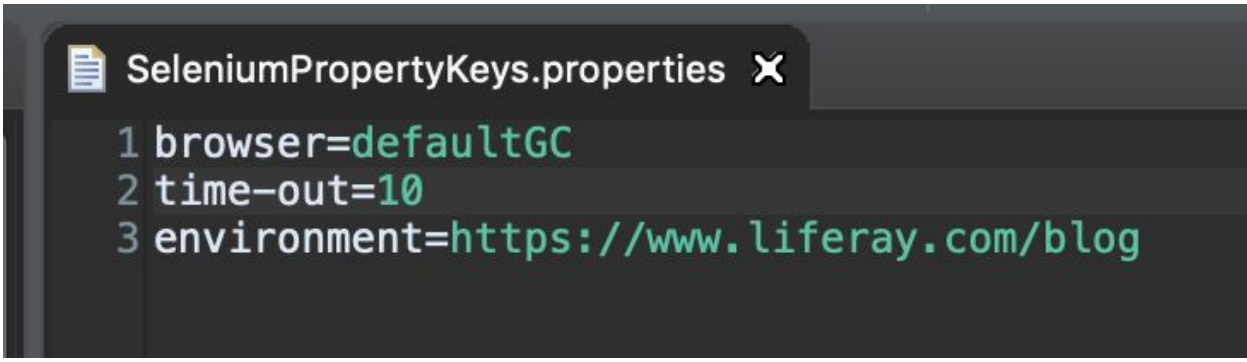
b. Type: blade gw setupPerformTestWithGoogleChrome
c. Return to your IDE and then Refresh the project
d. 

e. 

13. Update the properties with basic information of which website will be tested

a. Go to [test] > [functional-test-tdd] > SeleniumProperties > SeleniumPropertyKey.properties

b. Insert: environment=https://www.liferay.com/blog

c. 

d. PS.: If you want to run as Headless, just comment the first line with "#": #browser=defaultGC

e. P.S.: if you want to know more about how parameters and values you could use into this properties, go to
   i. Project and External Dependencies > frw-selenium-commons-3.0.0.jar > com.liferay.gs.testFramework.core > SeleniumReadPropertyKeys.class

What we will automate

1. Website tested: Liferay Blog: https://www.liferay.com/blog
2. Check if there is content published for "Tests";

Prepare the Web Automation Architecture

1. **Create the packages (com.liferay.sample.functional.test.) to:**
   a. pages
   b. tests
   c. utils
   d. Suite

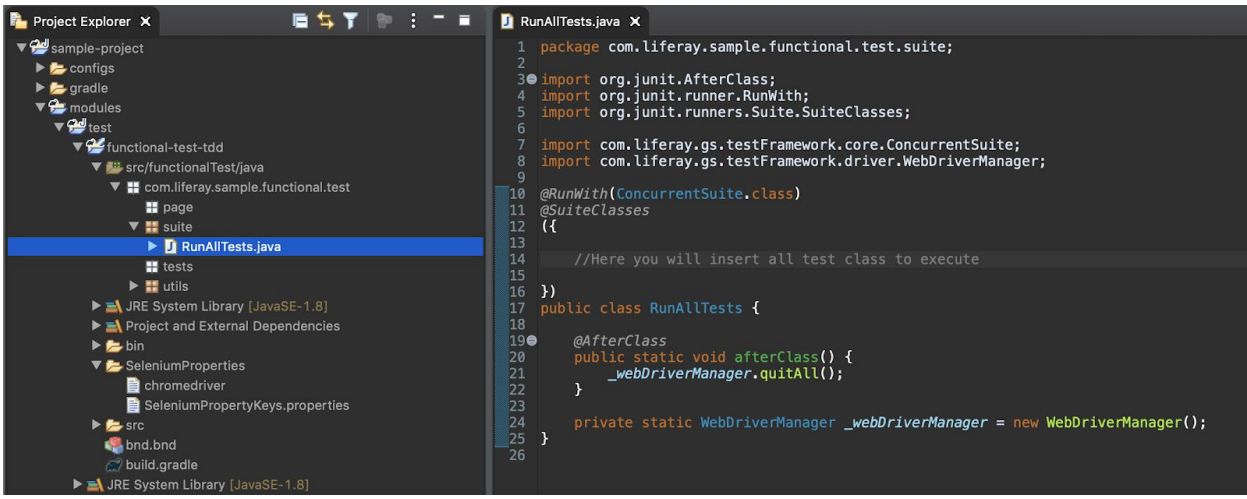
e. 

2. **Let's create all classes necesseries:**
    a. Suite:
        i. RunAllTests()
    b. Utils:
        i. CommonMethods() -> it's like a base path

3. **Now, update the RunAllTest with:**

```
package com.liferay.sample.functional.test.suite;

import org.junit.AfterClass;
import org.junit.runner.RunWith;
import org.junit.runners.Suite.SuiteClasses;

import com.liferay.gs.testFramework.core.ConcurrentSuite;
import com.liferay.gs.testFramework.driver.WebDriverManager;

//This will be responsible to run with concurrence
@RunWith(ConcurrentSuite.class)
@SuiteClasses
({
        //Here you will insert all test class to execute
})
public class RunAllTests {

        @AfterClass
```

```
        public static void afterClass() {
                _webDriverManager.quitAll();
        }

        private static WebDriverManager _webDriverManager = new WebDriverManager();
}
```



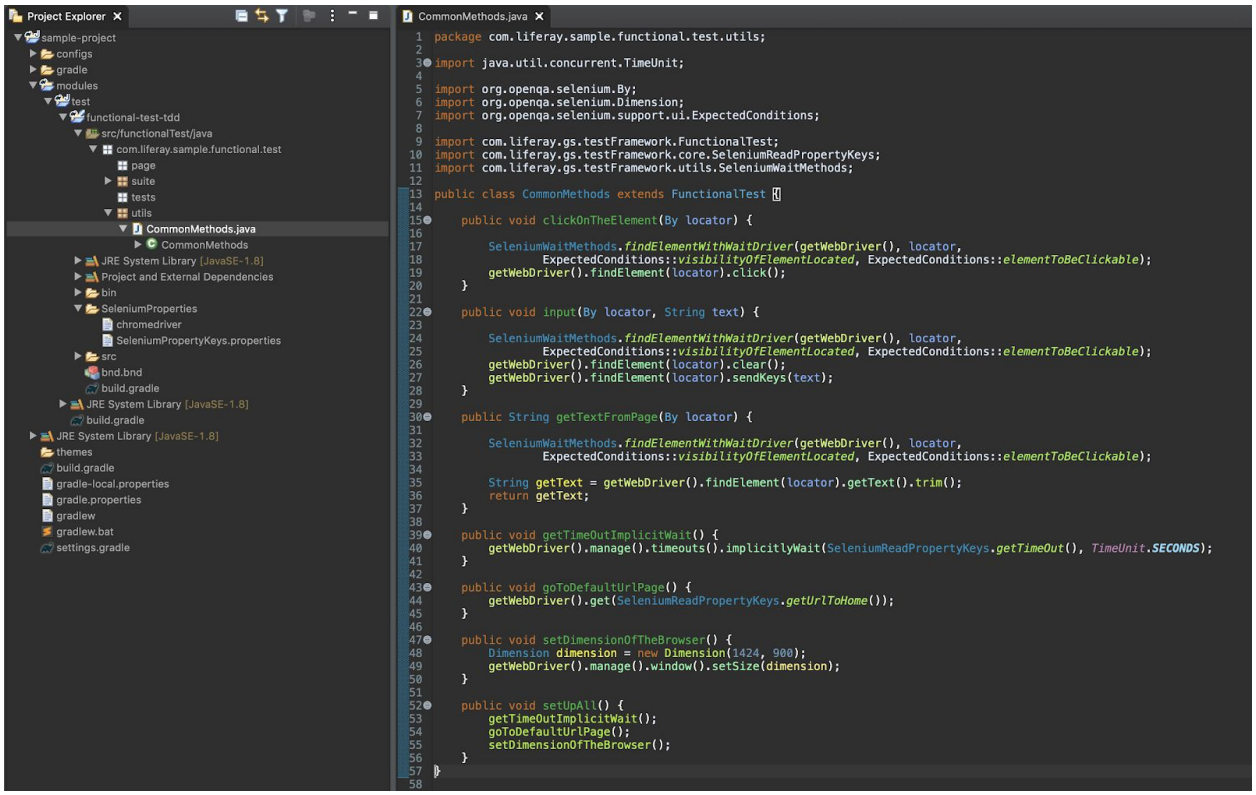
4. **Now, update the CommonMethods() with:**

```
package com.liferay.sample.functional.test.utils;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;
import org.openqa.selenium.Dimension;
import org.openqa.selenium.support.ui.ExpectedConditions;

import com.liferay.gs.testFramework.FunctionalTest;
import com.liferay.gs.testFramework.core.SeleniumReadPropertyKeys;
import com.liferay.gs.testFramework.utils.SeleniumWaitMethods;

/*
```

```
 * When you will need to use anything from the Liferay Framework
 * you must extends the FunctionalTest
 *
 * In this class you will see the example as:
 * getWebdriver()
 * findElementWithWaitDriver()
 *
 * */
public class CommonMethods extends FunctionalTest {

	public void clickOnTheElement(By locator) {

		// Improve your findElements using the implicitWaits

SeleniumWaitMethods.findElementWithWaitDriver(getWebDriver(), locator,

ExpectedConditions::visibilityOfElementLocated,
ExpectedConditions::elementToBeClickable);
		// use getWebdriver() as the driver to interact with the
browser
		getWebDriver().findElement(locator).click();
	}

	public void input(By locator, String text) {


SeleniumWaitMethods.findElementWithWaitDriver(getWebDriver(), locator,

ExpectedConditions::visibilityOfElementLocated,
ExpectedConditions::elementToBeClickable);
		getWebDriver().findElement(locator).clear();
		getWebDriver().findElement(locator).sendKeys(text);
	}
```

```
public String getTextFromPage(By locator) {

SeleniumWaitMethods.findElementWithWaitDriver(getWebDriver(), locator,

ExpectedConditions::visibilityOfElementLocated,
ExpectedConditions::elementToBeClickable);

        String getText =
getWebDriver().findElement(locator).getText().trim();
        return getText;
    }

    public void getTimeOutImplicitWait() {

getWebDriver().manage().timeouts().implicitlyWait(SeleniumReadPropertyKe
ys.getTimeOut(), TimeUnit.SECONDS);
    }

    public void goToDefaultUrlPage() {

getWebDriver().get(SeleniumReadPropertyKeys.getUrlToHome());
    }

    // It's important to define the dimension, specially when you run as
Headless.
    public void setDimensionOfTheBrowser() {
        Dimension dimension = new Dimension(1424, 900);
        getWebDriver().manage().window().setSize(dimension);
    }

    public void setUpAll() {
        getTimeOutImplicitWait();
        goToDefaultUrlPage();
```

```
                    setDimensionOfTheBrowser();
            }
        }
```

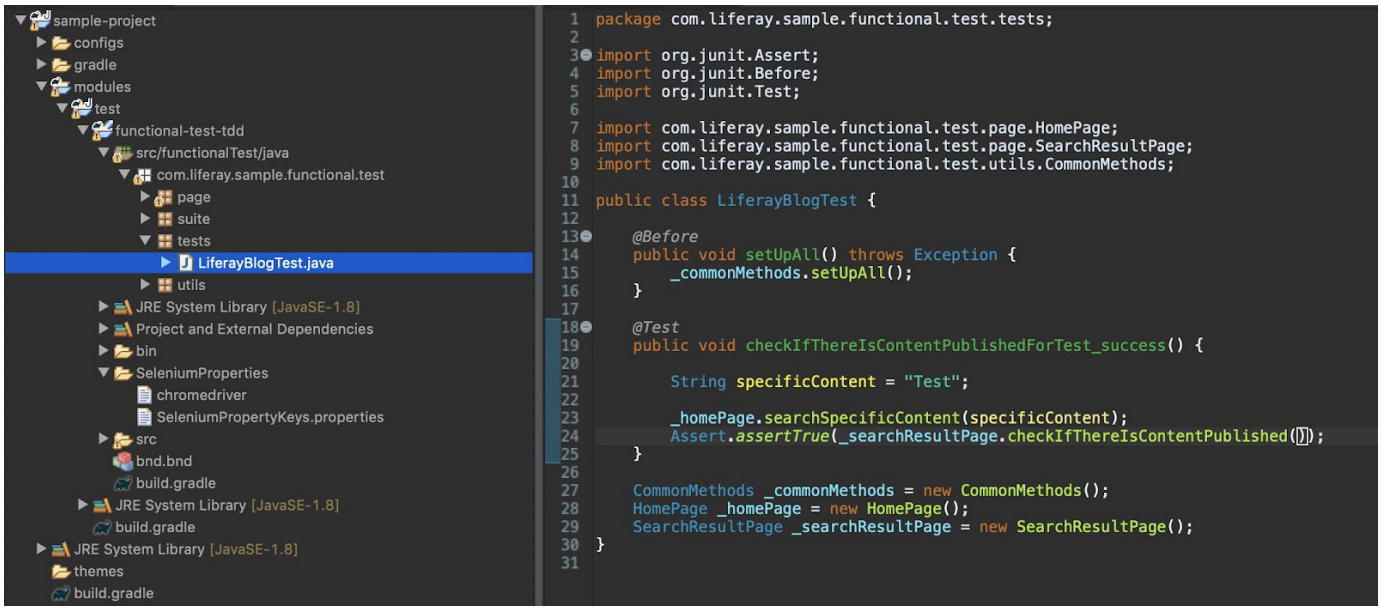


Create Automated Scenarios

1. **Scenario:** Check if there is content published for "Tests";

2. **What we should do:**

    a. **Create a test class**
        i. LiferayBlogTest()
            1. Method test: CheckIfThereIsContentPublishedForTest()
            2. @Before, to set up the environment;
            3. Should call the page's classes;
        ii. This class should not use logic stuff. Need to be clear and clean



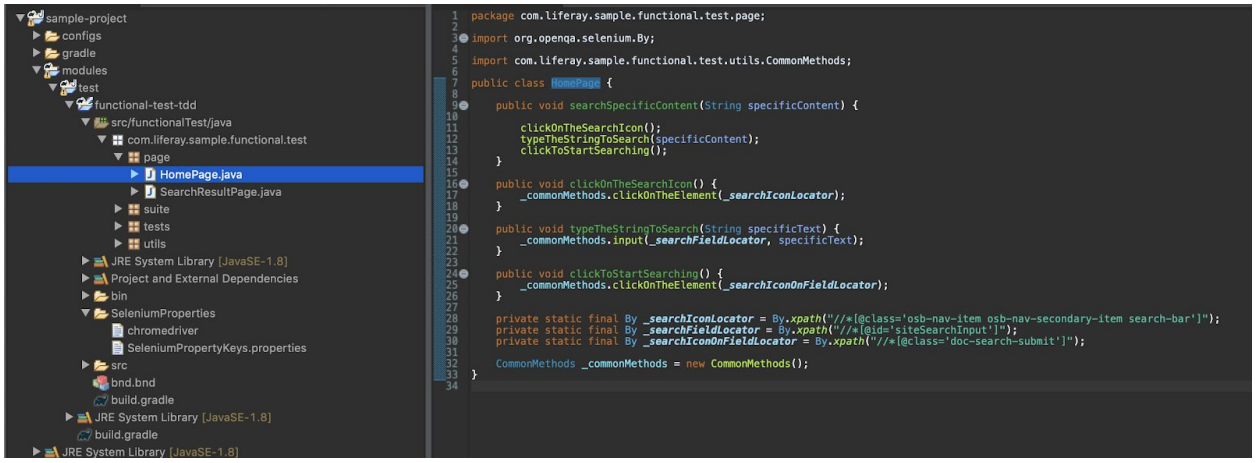
Code:

```
package com.liferay.sample.functional.test.tests;

import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

import com.liferay.sample.functional.test.page.HomePage;
import com.liferay.sample.functional.test.page.SearchResultPage;
import com.liferay.sample.functional.test.utils.CommonMethods;

public class LiferayBlogTest {

    @Before
    public void setUpAll() throws Exception {
        _commonMethods.setUpAll();
    }

    @Test
    public void checkIfThereIsContentPublishedForTest_success() {
```

```
        String specificContent = "test";

        _homePage.searchSpecificContent(specificContent);
        Assert.assertTrue(_searchResultPage.checkIfThereIsContentPublished());
    }

    CommonMethods _commonMethods = new CommonMethods();
    HomePage _homePage = new HomePage();
    SearchResultPage _searchResultPage = new SearchResultPage();
}
```

b. Create a page class

i. HomePage()

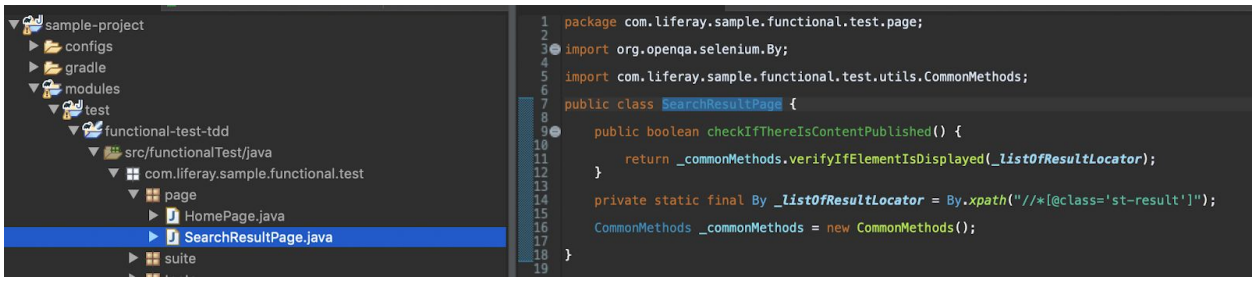


Code:

```
package com.liferay.sample.functional.test.page;

import org.openqa.selenium.By;

import com.liferay.sample.functional.test.utils.CommonMethods;

public class HomePage {
```

```
public void searchSpecificContent(String specificContent) {

        clickOnTheSearchIcon();
        typeTheStringToSearch(specificContent);
        clickToStartSearching();
}

public void clickOnTheSearchIcon() {
        _commonMethods.clickOnTheElement(_searchIconLocator);
}

public void typeTheStringToSearch(String specificText) {
        _commonMethods.input(_searchFieldLocator, specificText);
}

public void clickToStartSearching() {
        _commonMethods.clickOnTheElement(_searchIconOnFieldLocator);
}

private static final By _searchIconLocator = By.xpath("//*[@class='osb-nav-item osb-nav-secondary-item search-bar']");
private static final By _searchFieldLocator = By.xpath("//*[@id='siteSearchInput']");
private static final By _searchIconOnFieldLocator = By.xpath("//*[@class='doc-search-submit']");

CommonMethods _commonMethods = new CommonMethods();
}
```

ii. SearchResultPage()



Code:

```
package com.liferay.sample.functional.test.page;

import org.openqa.selenium.By;

import com.liferay.sample.functional.test.utils.CommonMethods;

public class SearchResultPage {

    public boolean checkIfThereIsContentPublished() {

        return _commonMethods.verifyIfElementIsDisplayed(_listOfResultLocator);
    }

    private static final By _listOfResultLocator = By.xpath("//*[@class='st-result']");

    CommonMethods _commonMethods = new CommonMethods();

}
```

c. Update the CommonMethod (insert in the end of the class)

```
69
70      public boolean verifyIfElementIsDisplayed(By locator) {
71
72          SeleniumWaitMethods.findElementWithWaitDriver(getWebDriver(), locator,
73                  ExpectedConditions::visibilityOfElementLocated, ExpectedConditions::elementToBeClickable);
74
75          result = (getWebDriver().findElements(locator).size() > 0);
76          return result;
77      }
78
79      boolean result = false;
80  }
81
```

Code:

public boolean verifyIfElementIsDisplayed(By locator) {

SeleniumWaitMethods.findElementWithWaitDriver(getWebDriver(), locator,

ExpectedConditions::visibilityOfElementLocated, ExpectedConditions::elementToBeClickable);

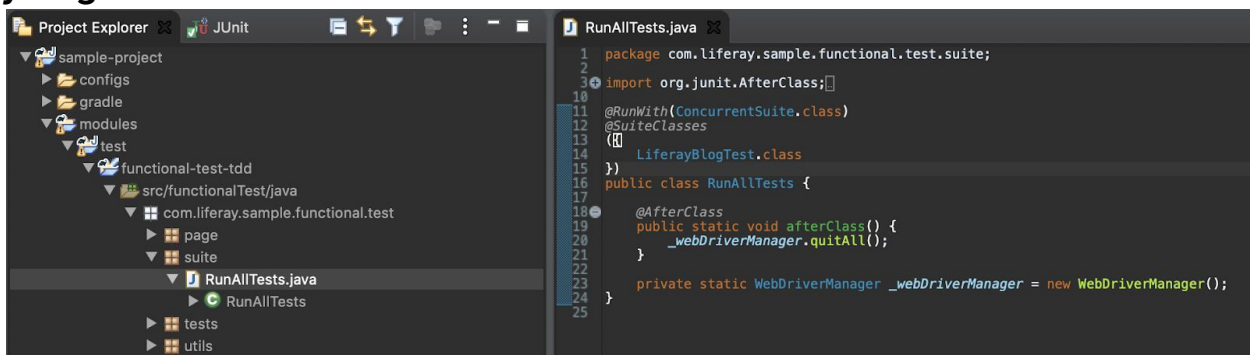
result = (getWebDriver().findElements(locator).size() > 0);

return result;

}

boolean result = false;

How to Run by Command line

1. Go to RunAllTest() class, and insert the test class that should run: **LiferayBlogTest.class**
   a. 

2. Open your Terminal / iTerm

a. From the root of the project, type: **blade gw performTest**

```
➜ sample-project blade gw performTest
> Task :modules:test:functional-test-tdd:performTest

com.liferay.sample.functional.test.suite.RunAllTests STANDARD_OUT
    Scheduled (class org.junit.runners.ParentRunner$4): org.junit.runners.ParentRunner$4@21fc35ce

com.liferay.sample.functional.test.suite.RunAllTests > com.liferay.sample.functional.test.tests.LiferayBlogT
est.checkIfThereIsContentPublishedForTest_success PASSED

-----------------------------------------------------------------
|  Results: SUCCESS (1 tests, 1 successes, 0 failures, 0 skipped)  |
-----------------------------------------------------------------

BUILD SUCCESSFUL in 19s
2 actionable tasks: 2 executed
Update available 4.0.0 -> 4.0.1
Run `blade update` to install
➜ sample-project
```