# Homework/Exercise: Selenium IDE and WebDriver

This document defines the homework assignments from the ["QA Fundamentals" Course @ Software University](https://softuni.bg/trainings/1166/qa-fundamentals-july-2015). Please submit as homework a single zip / rar / 7z archive holding the source code or whatever approach is needed to finish your tasks

# Introduction

Selenium is a project with a set of test automation tools primarily focused on web applications. An IDE - an extension for Firefox web-browser and an API for the most popular programming languages. In the exercises below your task will be to use both IDE and WebDriver to automate some tasks. The basic setup and examples will be shown detailed with explanations and screenshots. Afterwards you will have to complete a similar task without any graphical examples.

## Install Selenium IDE

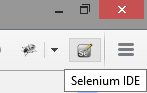
In order to use Selenium IDE you will need Firefox web-browser. If you don’t have one, before proceeding to Selenium IDE, take one from <http://getfirefox.com>.

Then go to Google -> type “Selenium IDE download” -> click on the first result -> find the section IDE on the page -> click on the latest version (e.g. 2.9.0). Or simply click on the link which should directly try to install the extension for you (not recommended) <http://release.seleniumhq.org/selenium-ide/2.9.0/selenium-ide-2.9.0.xpi>

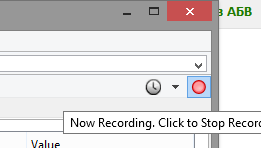
## Record and playback a login functionality in ABV.BG

<http://abv.bg> was once the most popular mailbox amongst Bulgarians and its glory lasted more than a decade. If you don’t use it frequently try to remember your credentials from the times when internet was young and virgin. If you don’t have an account there, this is your chance to register one.

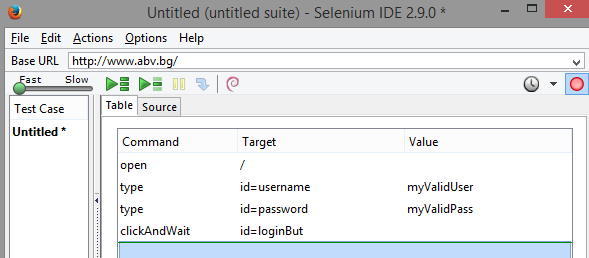
1. First thing you need to do is to start Selenium IDE and the recording action. Click the IDE icon in the upper right corner of your Firefox web-browser



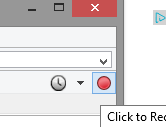
1. Navigate to <http://abv.bg>
2. Check your IDE if it’s in state “Now recording”



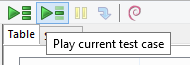
1. Enter your credentials and click the “Вход” button
2. Once you are successfully logged in, check the IDE. It should have recorded your actions so far



1. Click the recording button to ensure you have stopped recording



1. Logout from your account.
2. Replay the actions by clicking “Play current test case”

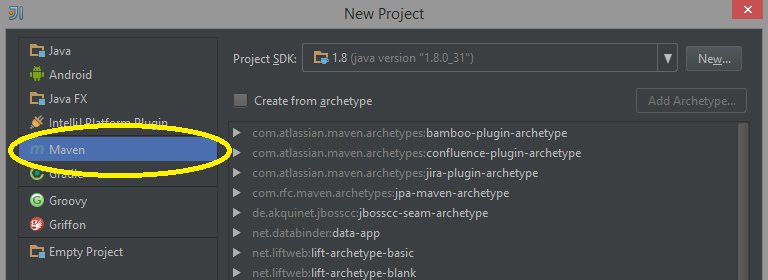


1. Ensure the played version have opened the site and logged you successfully.
2. Save your test case in the filesystem named “01. Automate ABV.BG login.html” as it will be part of your homework submission (you will later need to add it to the archive which you will submit as a homework)

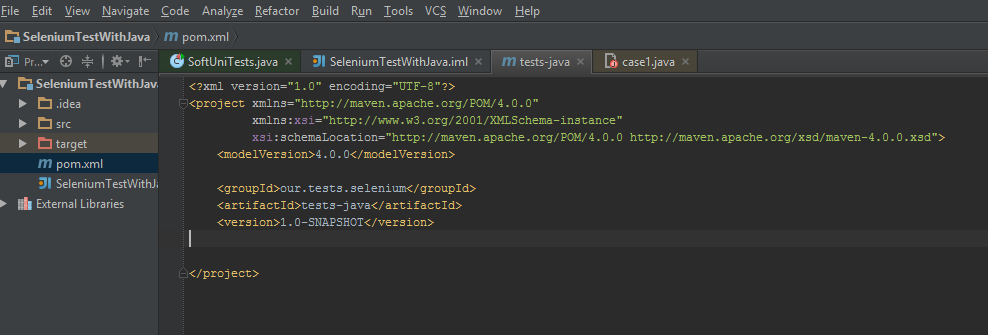
## Playback the login via WebDriver

Your task slightly evolves. You need to do the same, but this time using Selenium WebDriver as a part of a Java Maven project. The recorded IDE script can easily be exported as a Java JUnit Selenium WebDriver test case, but in this exercise we will do it from scratch.

1. Setup a Java Maven project using IntelliJ IDEA
   1. Open IntelliJ IDEA and click on Create new project. Choose maven from the right side menu.



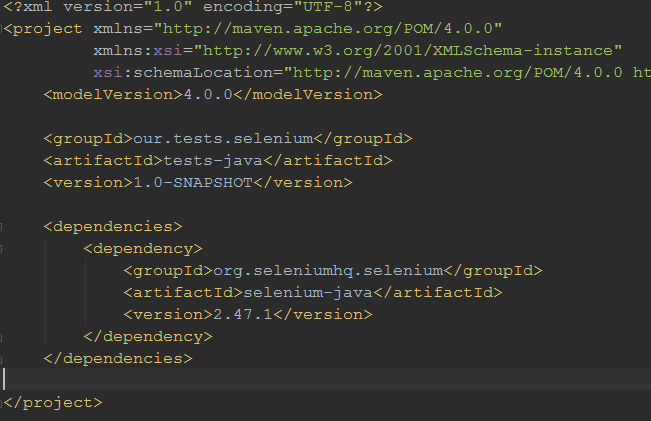
* 1. Click Next
  2. Write some dummy Group ID and Artifact ID like tests.abv and click Next
  3. Name your project somehow and choose where it should be stored in your filesystem. You will later FIND IT THERE in order to ADD IT TO THE HOMEWORK ARCHIVE.
  4. After clicking Finish you will Result into a new project with added pom.xml file



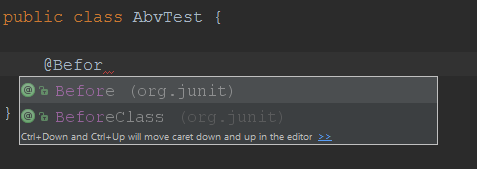
* 1. This file is responsible for a lot of the metainformation about the project, including the libraries it depends on. We will strongly depend on Selenium WebDriver, thus we need to add this dependency.
  2. Open dependencies tags with empty contents in the pom.xml file

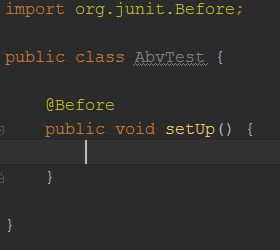


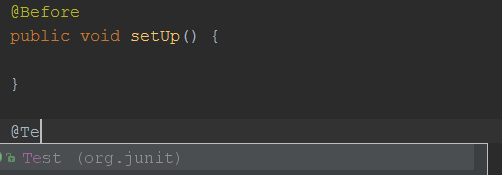
* 1. Now we need to describe the WebDriver dependency. In order to find its metainfo we can query google for “Seneium WebDriver maven”. Then open the first result. Then copy the dependency block given there. Finally resulting in pom.xml with

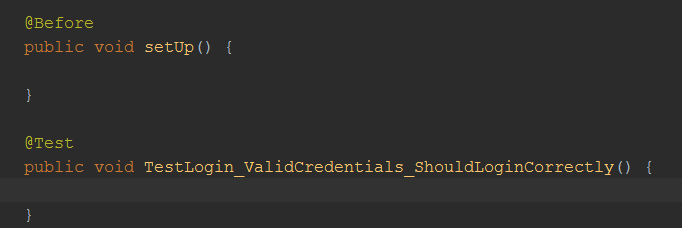


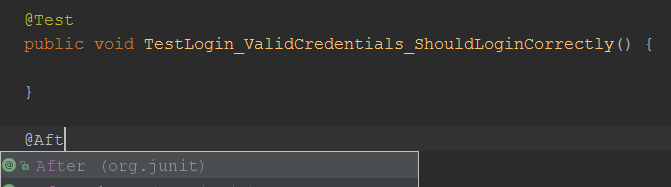
1. Create class named AbvTest in src/test/java
2. Create an initialize method (annotated with Before), one test method and one teardown method, all of them still empty. Use the IntelliJ default suggestions or force them with ALT + Enter in order to import the needed libraries (e.g. @Before needs JUnit)

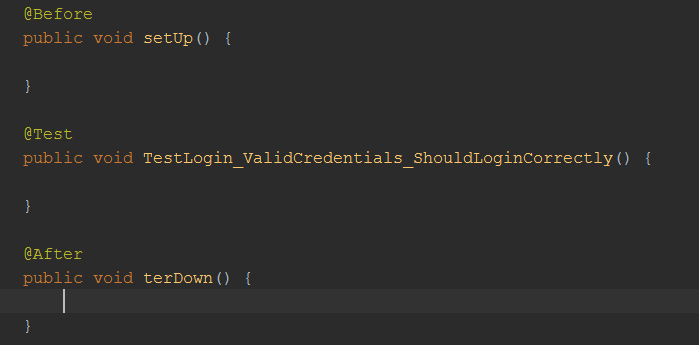




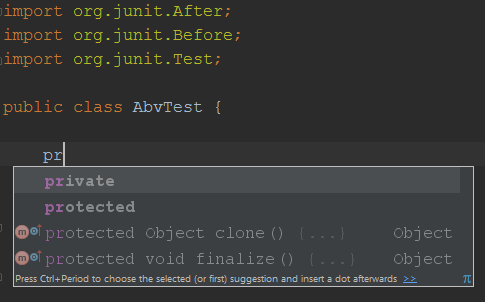


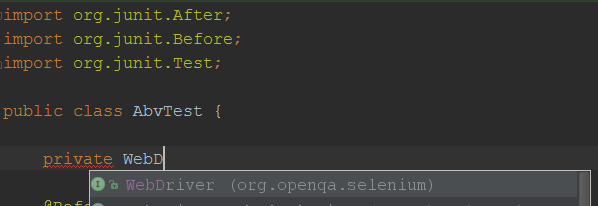


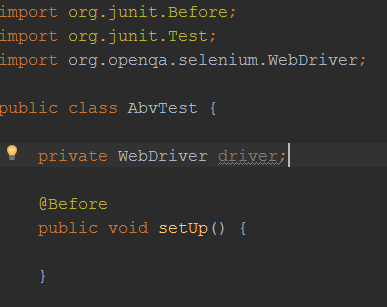




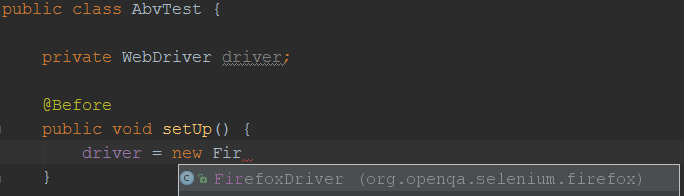
1. Create a class field named “driver” of type “WebDriver” with private access

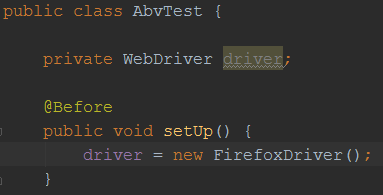




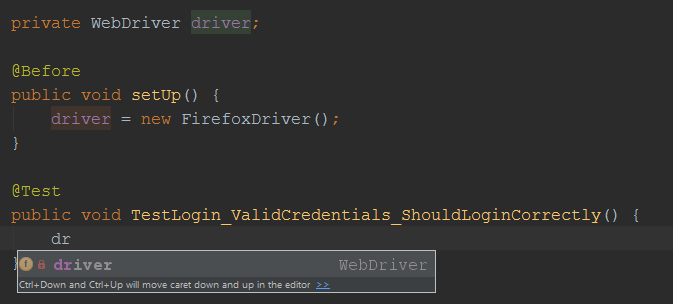


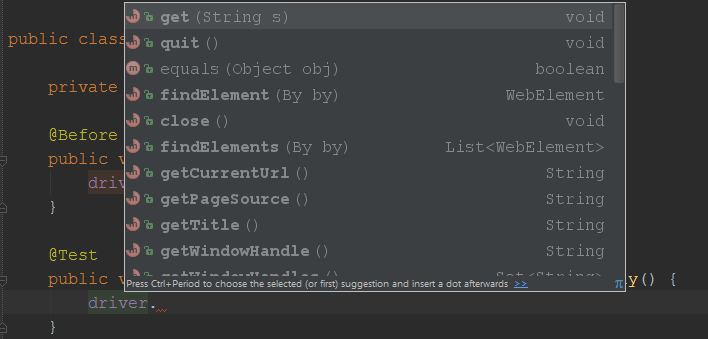
1. Initialize the “driver” with a concrete driver type (e.g. Firefox driver, Chrome driver, etc…). We will use Firefox driver all the way. The initialization is done in the initialize method annotated with @Before. In our case have named it “setUp()”.

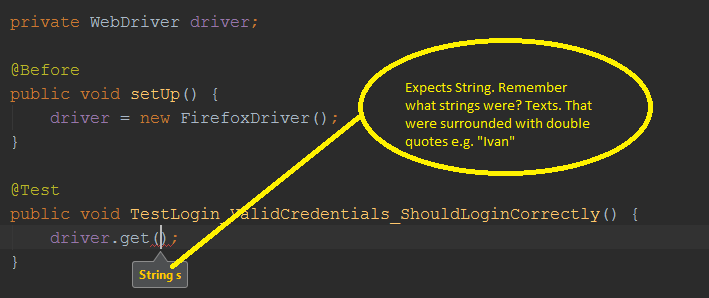


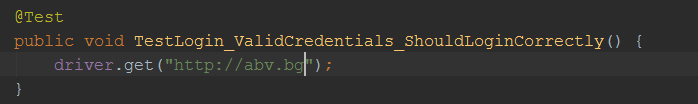


1. Let’s open <http://abv.bg> in our test method using the already initialized “driver” field. The method “get()” from the standard WebDriver library navigates to a certain url

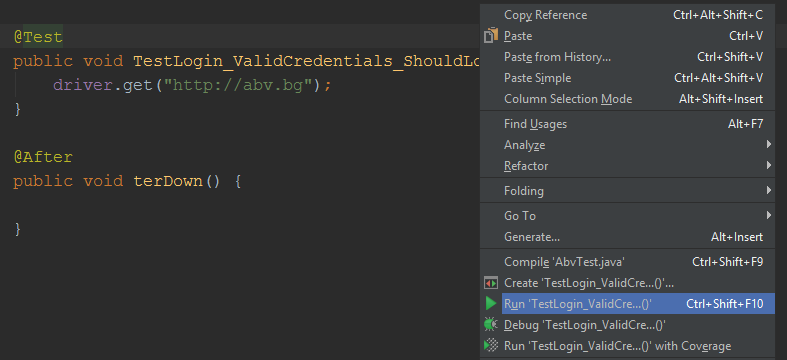




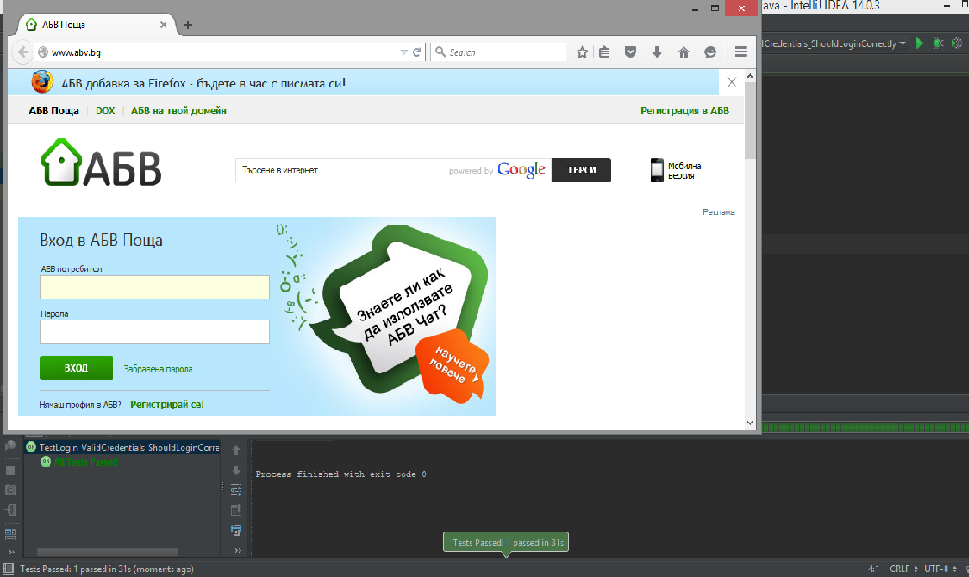




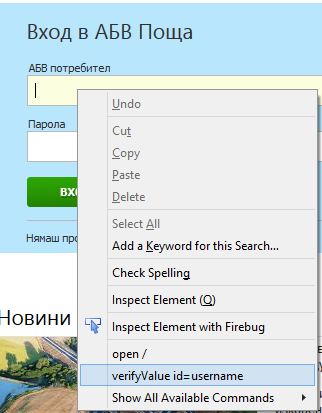
1. Run this test method via right click in the editor and “Run”. We expect it to compile correctly, force opening Firefox and navigating to ABV.BG



Result:

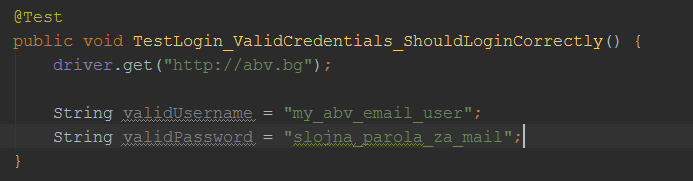


1. Let’s find the Username and Password fields by a certain selectors (Each of them with type WebElement) and populate valid username and password using the “sendKeys()” method.
   1. Find the most concrete selector for the username field. Selenium IDE helps for that. Open Selenium IDE just to have more helpers in the right-click context menu. Then right click on the username field. You will find the available commands that IDE suggests you. It includes the selector it uses. Obviously it’s the selector ID and the value of the ID is “username”

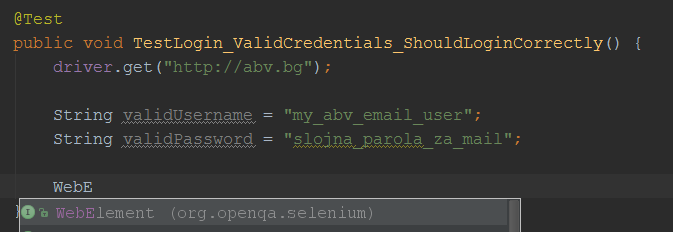


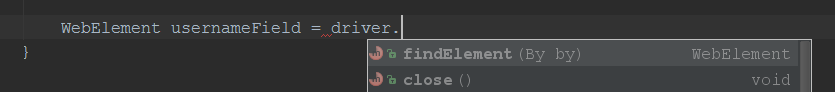
* 1. Do the same for the password field. Again the selector is ID and its value is “password”
  2. Do the same for the “Вход” button. The selector there is ID again and the value is “loginBut”. Now we will use these selectors and values to find the elements in our code. We will use “findElement()” method by giving it “By” static class which receives different selectors. E.g. By.Xpath. In our case it’s By.Id.

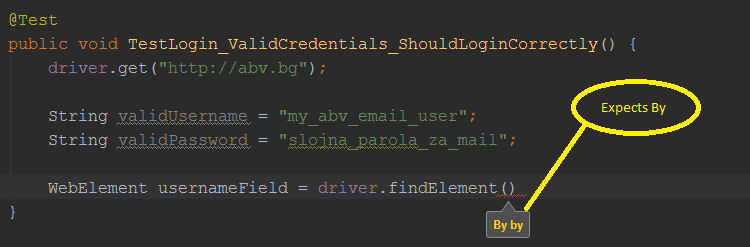
1. Declare our valid username and password in the code as String variables since they are textual

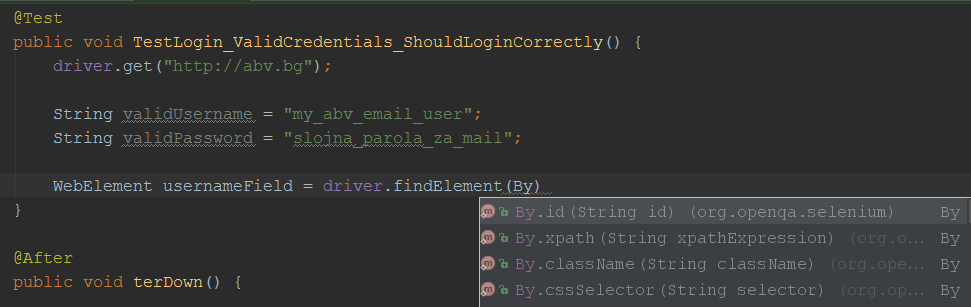


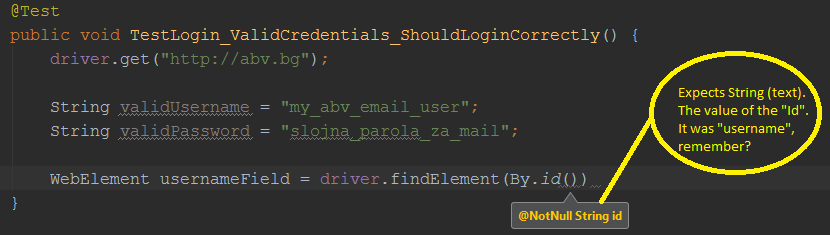
1. Find username field by id













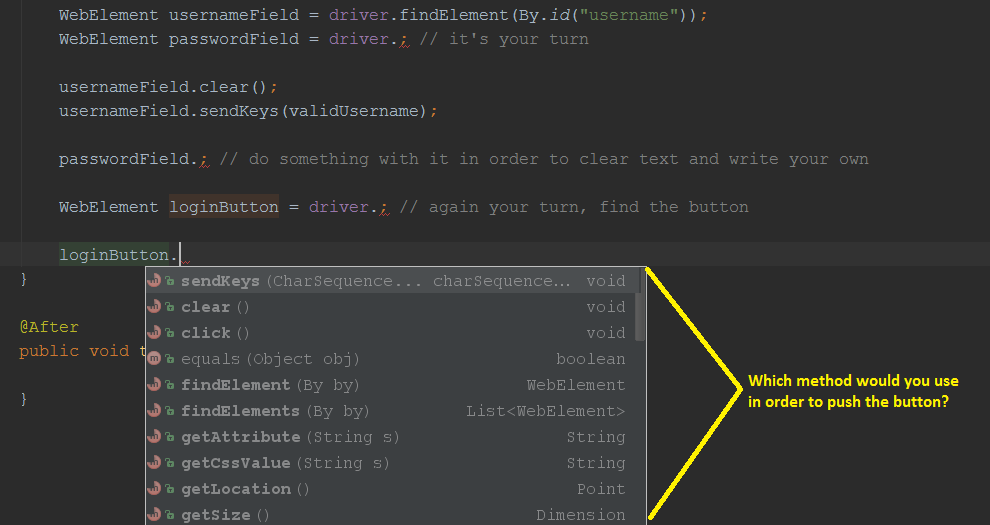
1. Do the same for the password field.



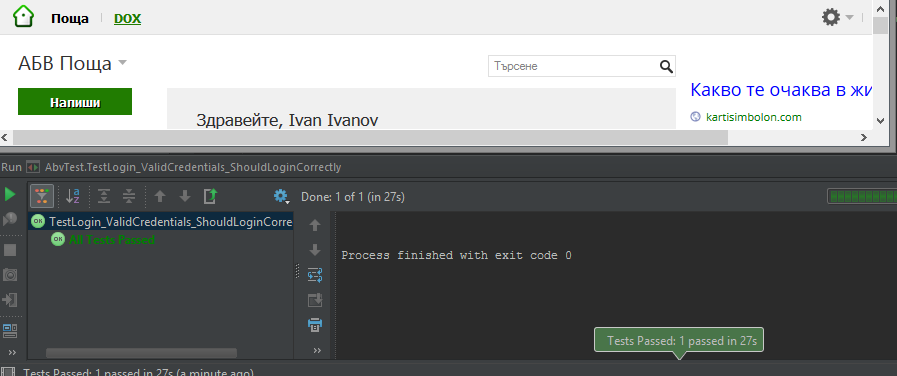
1. Clear the current text in the usernameField and type your valid username in it.



1. Do the same for the password field.
2. Find the button like you have found the username field and the password field. It was still by selector ID and the value of the ID we found above using the IDE, remember?



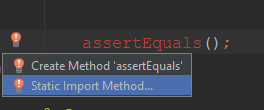
1. If you have done the proper actions, run the test again from the right-click context menu. If everything if ok, you should be navigated to the welcome screen and test would have passed successfully.

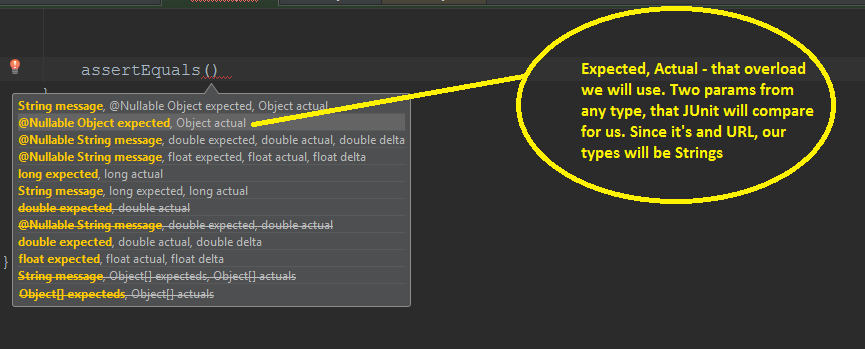


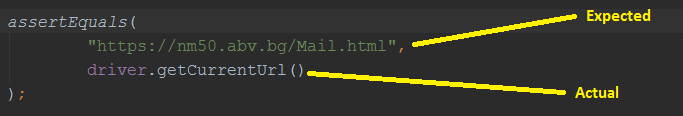
## Check if you are on the right URL logged in with right user

One of the biggest advantages of test automation is the script to tell you whether something works or not, since the reporting is the most important thing of the software development process so far. Luckily JUnit gives us a whole framework for that – the assertions. assertEquals() method will tell you whether the test passed successfully. It requires an expected result and the actual result (the current state).

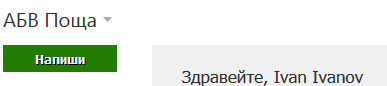
1. Once you have logged successfully, you should be on the url <https://nm50.abv.bg/Mail.html> . Make an assertion which checks whether the case navigated successfully to this url after the login button was pushed.
   1. assertEquals() is not recognized by default from the standard junit imported packages. It requires static import. In order to make IntelliJ to understand this, you need to write the whole method and use (), so the IDE will understand it’s a function/method. The click ALT + Enter



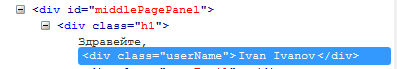




1. The next thing we would like to check is, if we were logged with the right user. We can log with an username, but at the end of the day we have also our actual fullname, in our case it’s Ivan Ivanov. So let’s check if it’s true.
   1. First thing we need to find is where “Ivan Ivanov” is located

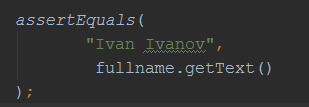


* 1. Use again the Selenium IDE to find the selector and its value..
  2. It’s a text in a subelement. The “Здравейте, “ part is irrelevant for us, so we will check only the name as a text

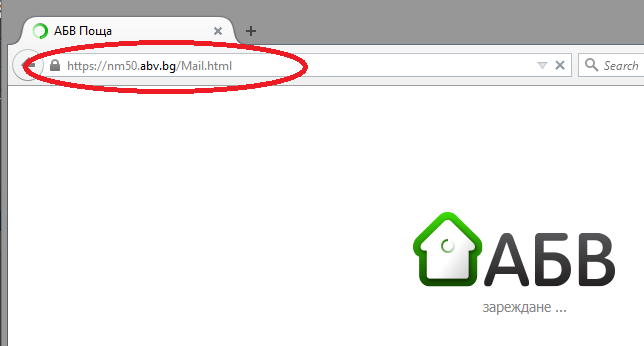




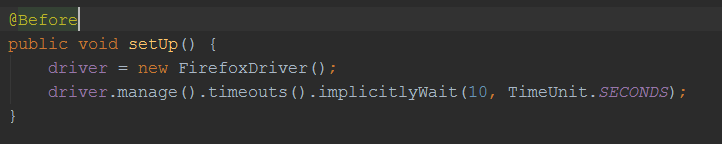
* 1. Assert if the text of the fullname element is the same as what we expect (Ivan Ivanov)



* 1. Run the test method from the context menu
  2. The test WILL fail. That’s because the WebDriver by defaults waits only the page synchronous requests to complete then scans the DOM for the selectors you have given. Unfortunately the modern web applications are doing a lot of asynchronous work, so does ABV. If you look closely right after logging in, the URL is changed to the desired one, but still nothing is present except one big waiting screen



* 1. Luckily, WebDriver supports Implicit and Explicit waits (<http://docs.seleniumhq.org/docs/04_webdriver_advanced.jsp> ). Explicit waits are waits for certain condition to be met (something like events), and are much harder, so we will not cover them in this topic. Instead, we will use implicit waits. These waits give us the opportunity to tell Selenium to wait some time, and retry to find the element by this selector. If it does not find it in the time given, it will still fail with timeout, but it will not do it immediately. We have seen that this transition from “АБВ зареждане...” to the real welcome screen is no more than 2-3 seconds with average internet connection. So we can set implicit wait of 10 seconds as stated in the documentation.
     1. Go to the setUp() method and set up the implicit wait time.

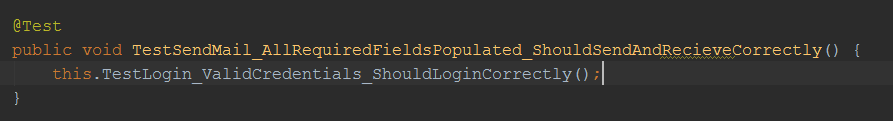


* 1. Rerun the test. Everything should pass successfully.

## Test mail sending

Extend the functionality to test not only the login, but the mail sending and receiving. For this purpose we will login and then compose a new mail, populate all required fields and send the mail to ourselves. Then we will check if the mail is in the folder “Inbox”.

1. Create a new test method. Call the previous test method in order to login correctly.



1. Find the compose mail button as WebElement
2. Click the compose mail button.
3. Find the “To” field and populate it with the same mail you are logged in
4. Find the “Subject” field and populate it with some dummy data e.g. “Test mail sending Selenium”
5. Find the textarea and populate it with some dummy data e.g. “Test body of the test mail”
6. Find the “Send” button and push it.
7. Go to inbox
8. Ensure you have received a new mail with the subject “Test mail sending Selenium” and with sender you (e.g. Ivan Ivanov)
9. Ensure the mails count in the folder “Inbox” have been incremented by one

## \* Self reporting test

Remember the Bug tracking systems we have talked about? JIRA or GitHub Issues? Make one of your tests to fail. Use some conditions to see if they have failed, and if so, before stopping the test, make Selenium to open you bug tracking system, login with your user and create new issue as a bug.

Some hints:

1. Use try/catch of if/else.
2. Navigate through your bug tracking system the same way you are navigating through abv.bg. By clicking on each link/button.
3. Use for the issue title, the test you are executing. E.g. if you are executing the login test, you issue title might start with “Login functionality does not work – “, then continue with the case you are asserting. If you are asserting username and password and you expect to login, but instead it says wrong username and password, append “ cannot login with valid credentials”. If you are expecting the url to change, but it does not change append “Url does not change”, or if you are expecting the name to be Ivan Ivanov, but it’s Stoyan Stoyanov, append “wrong fullname shown”. You may control that with string concatenation in each try/catch or if/else.
4. For the expected and actual result in the issue body you can send the same results as you are sending to assertEquals(). E.g. “Expected result: “Ivan Ivanov”, Actual result: “Stoyan Stoyanov”