# 1.Page Objects

## Introduction

A Page Object models the user interactions with the page as objects within the code. This reduces the amount of duplicated code and means that if the UI changes, the fix need only be applied in one place.

## Principle

Page Objects methods should return other PageObjects, page objects being an object oriented class that serves as an interface to a page from the automated tests. With this we can effectively model the users steps through the application, having a method defined for each user interaction with the page.  The tests then use the methods of this page object class whenever they need to interact with that page of the UI. The main benefit of this approach is the fact that when something is changed on a given page, the tests doesn't need to be changed, just change the method in the Page Object class. All changes that needs to be made for a page can be found in the same place.

The main benefits of a PageObject:

- A PageObject represents the entire page.
- Clean separation between the test code and page specific code.
- Single repository for the page actions.
- Page actions do not contain assertions.
- Different results are modeled as different methods.

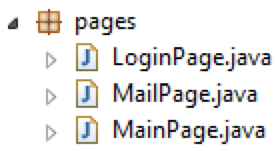
We take as an example the below code from a Yahoo login functionality:

```
@Before
public void setUp() throws Exception {
driver = new FirefoxDriver();
baseUrl = "http://ro.yahoo.com/";
driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
}
@Test
public void testContYahoo() throws Exception {
driver.navigate().to(baseUrl + "/");
driver.findElement(By.xpath("//span[@class='tab-icon tuc-spr']")).click();
driver.findElement(By.id("username")).sendKeys("mirimiramoii");
driver.findElement(By.name("passwd")).sendKeys("Aalndala99");
driver.findElement(By.id(".save")).click();
try {
assertEquals("Bun, ola",
driver.findElement(By.xpath("//div[@id='mediafpwave3']/div/a/em")).getText());
} catch (Error e) {
verificationErrors.append(e.toString());
}
driver.findElement(By.xpath("//span[@class='icon']/following::span[contains(text(),'Ma
il')]")).click();
driver.findElement(By.xpath("//li[@id='Compose']/a/span[contains(text(),'Compunere')]"
)).click();
}
@After
public void tearDown() throws Exception {
driver.quit();
```

The code is added and generated as Selenium IDE generates the code. This can be a little bit confusing and hard to read because it contains a lot of hardcoded data and also actions from different pages. The locators are not very clear what they suppose to do. When trying to add a new similar test, the code needs to be duplicated and hard to maintain in time.

The above test implies the actions on 3 pages: Main Yahoo page, Yahoo login page and Yahoo email page. Using and applying the Page Object pattern will conduct in the creation of a new package, containing the Page classes.

We start by adding a new Package containing 3 class pages:



```
pages
  LoginPage.java
  MailPage.java
  MainPage.java
```

Each class will contain the main actions that the user will perform over the page, grouped by action methods:

Example LoginPage will look like:

```
public class LoginPage {
public LoginPage enterUsername(String username){
PageDriver.driver.findElement(By.id("username")).sendKeys(username);
return this;
}
public LoginPage enterPassword(String password){
PageDriver.driver.findElement(By.name("passwd")).sendKeys(password);
return this;
}
public LoginPage authenticate(){
PageDriver.driver.findElement(By.id(".save")).click();
return this;
}
```

and the test class will look like:

```
public class SendAMail {
  MainPage mainPage = new MainPage();
  LoginPage loginPage = new LoginPage();
  MailPage mailPage = new MailPage();

@Test
public void testContYahoo() throws Exception {
 mainPage.clickSignIn();
 loginPage.enterUsername(TestEnv.Env.ADMIN);
 loginPage.enterPassword(TestEnv.Env.ADMINPASS);
 loginPage.authenticate();
 mainPage.clickEmail();
 mailPage.createNewMail();
}
```