# We are checking the return type and the parameter for By-

A screenshot of a computer

Description automatically generated with medium confidence

Sometimes common methods of all page class classes can be moved to a common package if needed.

# Use this assert class as the testng one is deprecated-

A screenshot of a computer

Description automatically generated with medium confidence

[Assert (JUnit API)](https://junit.org/junit4/javadoc/4.13/org/junit/Assert.html)

# When we run our test for first time we get assertion error-

Java is case sensitive. So expected and actual output not matching.

Selenium will always give preference to the visible text on the page and not the dom text, so we were getting the capitalization issue-

A screenshot of a computer

Description automatically generated

# Accounts page code and its run-

|  |
| --- |
| Accounts feature file  Feature: account page feature  # we will use background here because we want the precondition that the user should be logged in.  # if we change the hooks method and add the login there then out loginpage.feature will be impacted.  Background:  Given user has already logged into application  |userName|password|  |cucumbertesting1960@gmail.com|Malaravi@123|  #since we dont want to parameterise the above background so we used data table instead of examples.  Scenario: accounts page title  Given user is on accounts page  When user gets the title of the page  Then page title should be "My account - My Shop"  Scenario: account sections count  Given user is on accounts page  Then user gets accounts section on the page  #again we use data tables to pass in the data and check if all ok  # we dont want to parameterise, just verification of data  |ADD MY FIRST ADDRESS|  |ORDER HISTORY AND DETAILS|  |MY CREDIT SLIPS|  |MY ADDRESSES|  |MY PERSONAL INFORMATION|  |Home|  And account section page should have count as 6 |
| Account steps-  **package** stepDefinitions;  **import** java.util.List;  **import** java.util.Map;  **import** org.junit.Assert;  **import** com.pages.AccountPageObject;  **import** com.pages.LoginPageObject;  **import** com.qa.factory.DriverFactory;  **import** io.cucumber.datatable.DataTable;  **import** io.cucumber.java.en.Given;  **import** io.cucumber.java.en.Then;  **public** **class** AccountPageSteps {    **private** LoginPageObject loginPageObject=**new** LoginPageObject(DriverFactory.*getDriver*());  **private** AccountPageObject accountPageObject;  @Given("user has already logged into application")  **public** **void** user\_has\_already\_logged\_into\_application(DataTable credTable) {  List<Map<String, String>> credList=credTable.asMaps(); //as maps returns list of map which contains two string  //arguments.  //get(0) will give the first map pair of strings. from that first map pair we pass the key and get the value.  //in our case the first map pair will be username and password.  String userName=credList.get(0).get("userName");  String password=credList.get(0).get("password");  DriverFactory.*getDriver*().get("http://www.automationpractice.pl/index.php?controller=authentication&back=my-account");  accountPageObject=loginPageObject.doLogin(userName, password);  }  @Given("user is on accounts page")  **public** **void** user\_is\_on\_accounts\_page() {  String pageTitle= accountPageObject.getAccountPageTitle();  System.***out***.println("accounts page title is " + pageTitle);  }  @Then("user gets accounts section on the page")  **public** **void** user\_gets\_accounts\_section\_on\_the\_page(DataTable sectionsList) {  List<String> expectedAccountSectionList= sectionsList.asList();  System.***out***.println("expected account section list is " + expectedAccountSectionList);    List<String> actualAccountSectionList = accountPageObject.getAccountsSectionList();  System.***out***.println("actual account section list is " + actualAccountSectionList);    Assert.*assertTrue*(expectedAccountSectionList.containsAll(actualAccountSectionList));  }  @Then("account section page should have count as {int}")  **public** **void** account\_section\_page\_should\_have\_count\_as(Integer expectedAccountSectionCount) {  Assert.*assertTrue*(accountPageObject.getAccountSectionCount()==expectedAccountSectionCount);  }  } |
| Account page object-  **package** com.pages;  **import** java.util.ArrayList;  **import** java.util.List;  **import** org.openqa.selenium.By;  **import** org.openqa.selenium.WebDriver;  **import** org.openqa.selenium.WebElement;  **import** com.qa.factory.DriverFactory;  **public** **class** AccountPageObject {  **private** WebDriver driver;    //we will create a list of objects and capture the text from the account section.  **private** By accountsSection = By.*cssSelector*("div.center\_column span");    **public** AccountPageObject(WebDriver driver) {  **this**.driver=driver;  }    **public** **int** getAccountSectionCount() {  **return** driver.findElements(accountsSection).size();  }    **public** List<String> getAccountsSectionList() {  List<String> accountsList=**new** ArrayList<>();  List<WebElement> accountsHeadersList=driver.findElements(accountsSection);  //using for each we capture the text and store element in new array list  **for**(WebElement e:accountsHeadersList) {  String accountText=e.getText();  System.***out***.println(accountText);  accountsList.add(accountText);  }  **return** accountsList; //to return the list of strings which is easy to manipulate, first we created  //list of web elements and then captured the text, store in new list and return the new list.  }    /\*\*  \* this will return the title of the accounts page  \* **@return**  \*/  **public** String getAccountPageTitle() {  **return** driver.getTitle();  }  } |
| Login page object-  package com.pages;  import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  public class LoginPageObject {  // every page object will have by locators, constructor and page actions.  private WebDriver driver; // every class will have this webdriver.  // 1. by locators.  // by locators are also known as object repositories.  private By emailID = By.id("email");  private By password = By.id("passwd");  private By signInButton = By.id("SubmitLogin");  private By forgotPasswordLink = By.linkText("Forgot your password?1111");  // 2.constructor of the page class  public LoginPageObject(WebDriver driver) {  this.driver = driver;  }  // page classes should not have assertion.  // assertion should be written in test class or step def class.  // 3. page actions: features (behaviour) of the page in the form of methods.  public String getLoginPageTitle() {  return driver.getTitle();  }  public boolean isForgotPasswordLinkPresent() {  return driver.findElement(forgotPasswordLink).isDisplayed();  }  public void enterUserName(String userName) {  driver.findElement(emailID).sendKeys(userName);  }  public void enterPassword(String pwd) {  driver.findElement(password).sendKeys(pwd);  }  public void clickOnSignInButton() {  driver.findElement(signInButton).click();  }  // in step def do not maintain by locators and page methods. its ugly  // programming.  // in page object, selenium code should be written in page class.  // we will write one combine method for login which will take in username,  // password and login button click  public AccountPageObject doLogin(String un, String pwd) {  System.out.println("login with " + un + "pwd " + pwd);  driver.findElement(emailID).sendKeys(un);  driver.findElement(password).sendKeys(pwd);  driver.findElement(signInButton).click();  return new AccountPageObject(driver); // this is called page chaining concept.  //do login gives accounts page once logged in.  }    //according to page object model, when a method is landing you to next page from the current one  //then it is that methods responsibility to give you the object of that landing (new) page.  //this is page chaining concept used in frameworks.  } |
| Test runner-  package com.myTestRunner;  import org.junit.runner.RunWith;  import io.cucumber.junit.Cucumber;  import io.cucumber.junit.CucumberOptions;  @RunWith(Cucumber.class)  @CucumberOptions(  features = { "src/test/resources/appFeatures/AccountPage.feature" },  glue = { "stepDefinitions", "appHooks" },  plugin = { "pretty" }  )  public class MyTestRunner {  } |

output-

Junit output-

A screenshot of a computer error

Description automatically generated with medium confidence

Console output-

A screenshot of a computer screen

Description automatically generated with low confidence

A screenshot of a computer

Description automatically generated with low confidence

# Running login and accounts page together-

A screenshot of a computer

Description automatically generated

It runs in alphabetical order of page names. So, account page will run first followed by login page.

Now let’s run login and accounts page together-

In runner just modify the feature file line.

|  |
| --- |
| package com.myTestRunner;  import org.junit.runner.RunWith;  import io.cucumber.junit.Cucumber;  import io.cucumber.junit.CucumberOptions;  @RunWith(Cucumber.class)  @CucumberOptions(  features = { "src/test/resources/appFeatures/" },  glue = { "stepDefinitions", "appHooks" },  plugin = { "pretty" }  )  public class MyTestRunner {  } |

Junit console-

A screenshot of a computer

Description automatically generated

Console-

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with low confidence

A screenshot of a computer screen

Description automatically generated with low confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

Report link-

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

# Project structure-

A screenshot of a computer program

Description automatically generated with medium confidence

# Codes from this lecture-

|  |
| --- |
| Account page feature-  Feature: account page feature  # we will use background here because we want the precondition that the user should be logged in.  # if we change the hooks method and add the login there then out loginpage.feature will be impacted.  Background:  Given user has already logged into application  |userName|password|  |cucumbertesting1960@gmail.com|Malaravi@123|  #since we dont want to parameterise the above background so we used data table instead of examples.  Scenario: accounts page title  Given user is on accounts page  When user gets the title of the page  Then page title should be "My account - My Shop"  Scenario: account sections count  Given user is on accounts page  Then user gets accounts section on the page  #again we use data tables to pass in the data and check if all ok  # we dont want to parameterise, just verification of data  |ADD MY FIRST ADDRESS|  |ORDER HISTORY AND DETAILS|  |MY CREDIT SLIPS|  |MY ADDRESSES|  |MY PERSONAL INFORMATION|  |Home|  And account section page should have count as 6 |
| Account page steps-  **package** stepDefinitions;  **import** java.util.List;  **import** java.util.Map;  **import** org.junit.Assert;  **import** com.pages.AccountPageObject;  **import** com.pages.LoginPageObject;  **import** com.qa.factory.DriverFactory;  **import** io.cucumber.datatable.DataTable;  **import** io.cucumber.java.en.Given;  **import** io.cucumber.java.en.Then;  **public** **class** AccountPageSteps {    **private** LoginPageObject loginPageObject=**new** LoginPageObject(DriverFactory.*getDriver*());  **private** AccountPageObject accountPageObject;  @Given("user has already logged into application")  **public** **void** user\_has\_already\_logged\_into\_application(DataTable credTable) {  List<Map<String, String>> credList=credTable.asMaps(); //as maps returns list of map which contains two string  //arguments.  //get(0) will give the first map pair of strings. from that first map pair we pass the key and get the value.  //in our case the first map pair will be username and password.  String userName=credList.get(0).get("userName");  String password=credList.get(0).get("password");  DriverFactory.*getDriver*().get("http://www.automationpractice.pl/index.php?controller=authentication&back=my-account");  accountPageObject=loginPageObject.doLogin(userName, password);  }  @Given("user is on accounts page")  **public** **void** user\_is\_on\_accounts\_page() {  String pageTitle= accountPageObject.getAccountPageTitle();  System.***out***.println("accounts page title is " + pageTitle);  }  @Then("user gets accounts section on the page")  **public** **void** user\_gets\_accounts\_section\_on\_the\_page(DataTable sectionsList) {  List<String> expectedAccountSectionList= sectionsList.asList();  System.***out***.println("expected account section list is " + expectedAccountSectionList);    List<String> actualAccountSectionList = accountPageObject.getAccountsSectionList();  System.***out***.println("actual account section list is " + actualAccountSectionList);    Assert.*assertTrue*(expectedAccountSectionList.containsAll(actualAccountSectionList));  }  @Then("account section page should have count as {int}")  **public** **void** account\_section\_page\_should\_have\_count\_as(Integer expectedAccountSectionCount) {  Assert.*assertTrue*(accountPageObject.getAccountSectionCount()==expectedAccountSectionCount);  }  } |
| Accounts page object-  **package** com.pages;  **import** java.util.ArrayList;  **import** java.util.List;  **import** org.openqa.selenium.By;  **import** org.openqa.selenium.WebDriver;  **import** org.openqa.selenium.WebElement;  **import** com.qa.factory.DriverFactory;  **public** **class** AccountPageObject {  **private** WebDriver driver;    //we will create a list of objects and capture the text from the account section.  **private** By accountsSection = By.*cssSelector*("div.center\_column span");    **public** AccountPageObject(WebDriver driver) {  **this**.driver=driver;  }    **public** **int** getAccountSectionCount() {  **return** driver.findElements(accountsSection).size();  }    **public** List<String> getAccountsSectionList() {  List<String> accountsList=**new** ArrayList<>();  List<WebElement> accountsHeadersList=driver.findElements(accountsSection);  //using for each we capture the text and store element in new array list  **for**(WebElement e:accountsHeadersList) {  String accountText=e.getText();  System.***out***.println(accountText);  accountsList.add(accountText);  }  **return** accountsList; //to return the list of strings which is easy to manipulate, first we created  //list of web elements and then captured the text, store in new list and return the new list.  }    /\*\*  \* this will return the title of the accounts page  \* **@return**  \*/  **public** String getAccountPageTitle() {  **return** driver.getTitle();  }  } |
| Login page object-  package com.pages;  import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  public class LoginPageObject {  // every page object will have by locators, constructor and page actions.  private WebDriver driver; // every class will have this webdriver.  // 1. by locators.  // by locators are also known as object repositories.  private By emailID = By.id("email");  private By password = By.id("passwd");  private By signInButton = By.id("SubmitLogin");  private By forgotPasswordLink = By.linkText("Forgot your password?1111");  // 2.constructor of the page class  public LoginPageObject(WebDriver driver) {  this.driver = driver;  }  // page classes should not have assertion.  // assertion should be written in test class or step def class.  // 3. page actions: features (behaviour) of the page in the form of methods.  public String getLoginPageTitle() {  return driver.getTitle();  }  public boolean isForgotPasswordLinkPresent() {  return driver.findElement(forgotPasswordLink).isDisplayed();  }  public void enterUserName(String userName) {  driver.findElement(emailID).sendKeys(userName);  }  public void enterPassword(String pwd) {  driver.findElement(password).sendKeys(pwd);  }  public void clickOnSignInButton() {  driver.findElement(signInButton).click();  }  // in step def do not maintain by locators and page methods. its ugly  // programming.  // in page object, selenium code should be written in page class.  // we will write one combine method for login which will take in username,  // password and login button click  public AccountPageObject doLogin(String un, String pwd) {  System.out.println("login with " + un + "pwd " + pwd);  driver.findElement(emailID).sendKeys(un);  driver.findElement(password).sendKeys(pwd);  driver.findElement(signInButton).click();  return new AccountPageObject(driver); // this is called page chaining concept.  //do login gives accounts page once logged in.  }    //according to page object model, when a method is landing you to next page from the current one  //then it is that methods responsibility to give you the object of that landing (new) page.  //this is page chaining concept used in frameworks.  } |
| Test runner-  package com.myTestRunner;  import org.junit.runner.RunWith;  import io.cucumber.junit.Cucumber;  import io.cucumber.junit.CucumberOptions;  @RunWith(Cucumber.class)  @CucumberOptions(  features = { "src/test/resources/appFeatures/" },  glue = { "stepDefinitions", "appHooks" },  plugin = { "pretty" }  )  public class MyTestRunner {  } |