158. What is Page object model? Advantages of its implementation

We have have one standalone test where you are creating all locators in that one single

test and completing the execution.

So when edit box login user name id got changed, So you have to update that ID in all the tests, So you have to go to each and every test and update it.

So if you don't use page object mechanism and if you're directly run standalone test, then you will have some potential maintenance problem here.

Developers keep changing locators frequently, and every day you will end up updating all the files where those locators are defined. So instead, can't we put all these locators at one single place so that all our tests simply blindly access.

if you maintain only one copy of login locator, let's say ID=login

Tomorrow if ID record change, you will just update here so that it will automatically

update it in all files because all files are pointing to that one single variable where we updated it. So this is how we reduce maintenance by bringing them into common places like this and it can be used across multiple tests.

Problem here for this theory, your test might have around 200 locators in one single file

if you maintain all locators in one single file. If you identified one issue

You know, you have to go and search complete file to see where you have to update.

All the locators are in one single file, if you touch anything it will impacts other line in the file, that means not just one single locator.

The entire test will start failing if this file is messed up because all the tests are relying on only one single file.

And if you make any random mistake where it ran into some compilation issue or what, then all tests will break. You cannot dump everything in one single file.

Here page object model comes into picture.

how many pages you have - in your app we have login page, product catalog, checkout, confirmation.

create 4 Java classes instead of one single file, and all the locators which belong to login class will go into login java class and all the tests. If they are willing to use any locators which are present on login screen, then directly they will call this login class object and then they get their objects.

So like this we are reducing the size of our one single maximum file and we are breaking nicely.

If somebody says login, edit box is breaking/not working, then you will really not worry about searching all the classes because you know that ID is present in the login page class so you will directly go there and update it.

If they want to use login our ID, they'll obviously use this login class only.

like this you are solving maintenance issue.

And this side(page classes)also you are solving the maintenance issue by breaking them into different classes. The way we are breaking them depends upon which pages.

So this kind of design pattern we call a page object design pattern.

159. Creating Page object Classes for Login Screen and migrate the test

we have developed one end-to-end test. first how many pages it involved

so that we can create that many classes & move all these objects to those respective pages.

Landing page - create one separate class called landing page class. all the objects which belongs to this landing page, will go into that class.

So like this, we are segregating the pieces instead of having everything in one single file.

Landing page – url,username,password,login button these are the details for the login page

Note : if you put dot, while rename the package this will be created as a separate child folder

inside this parent folder.

In the landing page class

WebElement userEmail = driver.findElement(By.id(“userEmail”);

There is one problem driver, showing some error

where is this driver? Landing page do not have any clue about how this driver is initialized.

We have created driver in standalone class you have to bring that driver to this landing page.

I created one local object ( WebDriver driver; ) in landing page class

Standalone driver you have to send to this landing page class.

So I will create one constructor.

WebDriver driver; // local variable

Public LandingPage(WebDriver driver)

{

// Initialization

This.driver=driver;

}

constructor takes the same name as a class name. good thing of the constructor is

that will be the first method to execute when you touch your class.

If anybody try to initialize the object of this class, then first this method will execute.

After that only anything will happen.

Now in our case, what is our initialization required? - So we have local variable called driver,

this.driver.

So when you say ( this.driver ) this refers to the current class instance variable,

this.driver = driver; ( right side driver is coming from the standalone test )

is not really coming from that standalone test. We need to do

In standalone test, create one object for that landing page.

This is how you create. ( LandingPage landingpage = new LandingPage(driver); )

Aboveline driver is the argument.

when you are sending argument to any classes, all the arguments you can catch in the constructor.

So now when you are calling an object with one argument then that argument you have to define here also.

Public LandingPage(WebDriver driver)

bcze this is WebDriver in standalone test

you are sending WebDriver in your constructor also you need to tell that

somebody will call your class with one variable(driver) That information you have to tell to your constructor.

if I create object of this class ( LandingPage landingpage = new LandingPage(driver) )

Now this above driver, what you have in this file

that will go and sit here ( public LandingPage(WebDriver driver) )

whatever driver life you got from your standalone test that you are carefully passing to your local variable ( WebDriver driver; )

Now there is another nice way to declare this. So that mechanism we called as a Page Factory. There is one design pattern called Page Factory. So using that you can reduce the syntax of creating your web element.

There is an annotation called @findBy() here you have to tell attribute & attribute value

// page factory design pattern

@findBy(Id=”userEmail”)

WebElement userEmails; // userEmail – is a variable

At runtime, this will be constructed like d.fE(By.id=” “).sendkeys(“ “) this. You can write this way or above way.

how this guy know about driver? - in the constructor only we will mention

PageFactory.initElements(driver, this); // we add this in constructor

When initializing it needs driver, right? - It takes the driver's argument & then uses this driver to initialize.

Interview que : how this annotation knows our driver?

there is one method called initElements which you have to write first which will take care of constructing that using driver argument, what you send in the method.

@findBy(Id=”userPassword”)

WebElement passwordele;

@findBy(Id=”login”)

WebElement submit;

160. Implementing Action methods for Page factory web elements to implement logic

Instead of writing them in 3 steps, you can write one action method explaining what that method does, and you can push all these three lines of code into that method.

I will just put returntype – void

…  whatever email you want to send, send it from your test case and that email, I will put it here

Public void loginApplication() // Action method

Public void loginApplication(String email,String password)

{

userEmail.sendkeys(email);

passwordele.sendkeys(password);

submit.click();

}

Now, let's go back to our test.

remove this all three lines of code. Emailed,password,submit in the standalone test class.

Now you create an object for that class name.So you can call methods using this object.

Object dot. And you can call all the methods which are present inside that class.

landingpage. loginApplication([harisankar722@gmail.com,”H@rish777](mailto:harisankar722@gmail.com,)”); // in standalone test

I'll create another action method.

public void goTo()

{

driver.get(“url”);

}

landingpage.goTo();

 based upon different environments(LOCAL,QA,UAT), you work in your projects,

You can tweak the URL on global level there and it will apply to all your tests.

we have successfully completed login application.

Create one new class for product catalog page

//List<WebElement> products = d.fES(By.css(“.mb-3”);

@findBy(css=”.mb-3”)

List<WebElement> products;

if you if I want to start working on this product catalog page, this is the first step we have is that explicit wait in standlone test we have this explicit wait in so many places.

161. Creating Abstract Components to reuse the common methods/code in framework

 main rule - in any framework is wherever you see some reusable code, make sure

that you have it in one proper class and reuse it across all your test cases without duplicating the code again and again.

But there are some code like switching child windows, waiting for element to be located.

So all this may be reusable many times in our Selenium tests.

Create new package in main – package name ( rahulshettyacademy.abstractcomponents )

but the name itself says that, you know, all reusable components we are using here.

Inside this package – create one new class ( abstract component )

Let's say if you are landing page needs to utilize some reusable code written in this abstract component.

How you want to use here in your class one way is creating object of that class.

But let's not create object.

We will use concept called inheritance in landingPage class add extends and write your class name abstractcomponent.

Whatever methods variables fields declared in abstarctcomponent class will be automatically applicable in your landing page class. Also, that means you can reuse the code defined here.

In Productcatalog class also you need to add this extends abstractcomponent

open Abstractcomponent class

public void waitForElementToAppear(By findBy)

{

WebDriver wait = new WebDriverwait(driver,Duration.ofSeconds(5));

Wait.unitl(Expectedcoditions.visibiltyofelementslocated(findBy));

}

So this is our reusable method which have this code and this should be generic.

So you cannot hard code, you are located here.

see the entire driver dot will call as a web element. Means d.fE(By.css(“ “);

But if you see code only this piece(By.css(“ ”), then we call it as a by locator and return type for this is by.

Landingpage extends abstractcomponent

Child extends parent

Fom child class to parent class u can send variables

Add ( super(driver); ) Keyword in the landingpage child class if we add super keyword in the child class constructor then that will fall in to the abstract component parent class.

It is telling that create constructor because from child to class you are sending driver to parent.

But in parent to catch the driver, you should declare constructor. Only constructors can catch the variables.

So this is how we are first to catching the driver from actual test where it have life to the page object classes from page object. We are smartly sending to its parent with the help of super keyword.

@FindBy(css=”.mb-3”)

List<WebElement> products;

BY productsBy = cssSelector(“.mb-3”);

Public List<WebElement> getProductList() {

waitForElementToAppear(productsBy);

return products;

}

162. Page object Class implementation for Product catalogue page and update test

Productcatalog class

By addToCart = By.css(“.card-body button:last-of-tyep”);

Public WebElement getproductByName(String productName)

{

webElement prod = getProductList().stream().filter(product->product.fE(By.css(“b”).getText().equals(productsName)).findFirst().orElse(null);

return prod;

}

Public void addproductToCart(String productName)

{

WebElement prod = getproductByName(productName);

Prod.findElement(addTocart).click();

}

Interview que : "Can you apply page factory within WebElement.findElement?"

No, it's not possible.

Note wherever page factory is there (ex; d.fE(By.css(“ “); then you can use

@findBy(css=” “)

List<WebElement> products; // this reduces the whole code

164. Creating common methods to Abstract component and extending it in page classes

we have to write clicking on the cart, on the header.

So I don't want to write this in product catalog. bcze the header, what you see on top, is common for all pages. So I don't want to restrict this only for this page.

I have to put it in the abstract component bcze that is something which can be used

by all page object files, with a common header.

Note : we are using inheritance, as per inheritance, child classes have access

to parent class methods as well.

Validations cannot go in page object files. Page object files should only have the code

to perform actions. But if you are validating something which can make your test case pass or fail, that kind of validation should be inside your test case only.

every page creation, we are ending up creating one object So many objects we are created in the test project. To handle this – go to landing page after submit you now it will go to the productcatlogue page so after product catalogue object you can cut and past after the submit button and write the return type.

165. Wrapping up the whole test with complete refactor into Page object model

Created one new class – checkout page

first we will extend from the parent class where we can use reusable things and then we have to create constructor. Because that gives life of the driver.

Next WebDriver driver;

Then next you need to initialize elements.

Based upon these 3 pagefactory elements you have to write actions.

So our main action here is to select country

we have successfully completed the complete page object migration.

And now you look at the code, how simple it is that we are landing on the page first and logging in and getting the product list and we are adding the products to cart, right?

And we are going to cart the page and then we are verifying the product display and then we are checking

out, selecting this country, submitting the order, getting the confirmation page and having the assertion.