Implementing Page Object Model

Now that you're familiar with POM design, loadable components, and WebDriver manager, now it's time to learn implementation of a POM.

WE'LL COVER THE FOLLOWING

Page object model implementation

We will learn to implement the page object model in accordance with the design discussed in Designing Page Object Model.

First, we will create a base class `AbstractBasePage` that extends `org.openqa.selenium.support.ui.LoadableComponent` and associate a file containing locators. We initialize `AbstractBasePage` with `org.openqa.selenium.WebDriver` and `Locators` file in the constructor of the class.

Anatomy of locators file

Locators file could be of any format, such as `.properties`, `.xml`, `.json`, `.yaml`, `.ini`, `.toml`. For demonstration purposes, `.properties` is considered.

Keys present in the locators file should be suffixed with the type of locator. Selenium supports:

- ***id*** – id attribute of a HTML tag.

- ***linkText*** – text of <*a*> tag.
- ***partialLinkText*** – substring text of <*a*> tag
- ***name*** – name attribute of a HTML tag.
- ***tagName*** – name of the HTML tag.
- ***xpath*** – querying nodes or values from XML.
- ***className*** – class attribute of a HTML tag.
- ***cssSelector*** – querying nodes or values from HTML using CSS.

And the value of keys is the locator itself.

```
search_field.name = q

next_page.xpath = //span[contains(text(), 'Next')]
```

The above code snippet shows how to represent the locator key having suffix of one of the locator types.

Using appropriate libraries, the locator file is read and stored in appropriate data structures for querying. As we are using a properties file, we will use `java.util.Properties` for storing the *key-value* pairs.

Anatomy of AbstractBasePage

All page objects we will be creating will be extending `AbstractBasePage`. This class is initialized with an instance of `org.openqa.selenium.WebDriver` and the locators file associated with that page.

As this class is extending `LoadableComponent`, we implement the methods `isLoaded()` and `load()` to ensure the page load is complete, and if something goes wrong during the load, what needs to be done to mitigate the same and ascertain the page load is done.

Apart from these methods, we have another method `getLocator(String key)` to query the locator's value for the given key. This method is basically a factory method where the different types of locators are created based on the suffix of the locator's key found in the locators file.

In the above example of locators file, we have `search_field.name = q`

```
By locator = getLocator("search_field");
```

This above example will create a `By.name(...)` object for the matching key present in the locators file. By using this strategy, we can make our code independent of the changes in the locators and hard-coding in Java file.

In the future, let's say if any of the locators change, all we need to do is, change in the respective locators file with the value and suffix the key with the correct locator type.

A sample implementation of `AbstractBasePage`:

```
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.io.Reader;
import java.util.Optional;
import java.util.Properties;

import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.support.ui.LoadableComponent;

public abstract class AbstractBasePage<T extends AbstractBasePage<T>> exte
nds LoadableComponent<T> {

    private static final String BASE_LOCATORS_DIR = "src/main/resources/lo
cators";

    protected final WebDriver driver;
    private final File locatorFile;
    private final Properties properties = new Properties();

    public AbstractBasePage(WebDriver driver, String locatorsFile) {
        this.driver = driver;
        this.locatorFile = new File(BASE_LOCATORS_DIR, locatorsFile);
        try {
            Reader reader = new FileReader(this.locatorFile);
            properties.load(reader);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
    /* check whether the page load is complete */
    @Override
    protected void isLoaded() throws Error {
        String state = (String) ((JavascriptExecutor) driver).executeScript("return document.readyState");
        assert "complete".equalsIgnoreCase(state);
    }

    /* if page not loaded in time or any problem occurred, refresh the page */
    @Override
    protected void load() {
        driver.navigate().refresh();
    }

    protected final By getLocator(String key) {
        Optional<String> locatorKey = properties.keySet().stream().filter(e -> e.toString().startsWith(key))
                .map(Object::toString).findFirst();
        if (!locatorKey.isPresent()) {
            throw new RuntimeException(
                    "unable to find locator with key " + locatorKey + " in " + locatorFile.getAbsolutePath());
        }
        String locatorType = locatorKey.get().substring(locatorKey.get().lastIndexOf('.') + 1);
        String value = properties.getProperty(locatorKey.get());
        switch (locatorType) {
        case "id":
            return By.id(value);
        case "name":
            return By.name(value);
        case "xpath":
            return By.xpath(value);
        case "css":
            return By.cssSelector(value);
        case "tag":
            return By.tagName(value);
        case "link":
            return By.linkText(value);
        case "partialLink":
            return By.partialLinkText(value);
        case "class":
            return By.className(value);
        default:
```

```
            return null;
        }

    }

}
```

In the above code snippet, we are:

- reading the locators file in the constructor
- checking whether ***readyState*** of the document is ***complete*** in `isLoaded` method and reloading/refreshing the page in `load` method, incase `isLoaded` method fails. These methods will be used by `LoadableComponent` `get` method to create an instance of the page objects to ensure that indeed the page is in the right state.
- fetching locator from the locator property file based on key and creating a `By` object is done in `getLocator(String key)` method.

Sample page objects

Here, we shall perform a simple Google Search with some query. Here, we will create two pages : `GoogleSearch` and `SearchResults`.

`GoogleSearch` is the landing page. So, the base url needs to be loaded here. This is happening in `isLoaded()` method. Along with loading the base url, this method also checks for the page loading completeness.

One method for search for the given query which once done navigates to the next page `SearchResults`.

In `SearchResults`, we have methods for actions that are possible on that page.

With this chaining of actions and having assertions at every action, it makes it easy for anyone to read and understand the intent.

Please note that when creating instances of the pages, we need to call `get()` to take advantage of the `LoadableComponent` feature.

```
GoogleSearch search = new GoogleSearch(driver).get();
```

Implementation of GoogleSearch

```
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;

public class GoogleSearch extends AbstractBasePage<GoogleSearch> {

    private static final String URL = "https://www.google.com";

    public GoogleSearch(WebDriver driver) {
        super(driver, "search-home.properties");
    }

    @Override
    protected void isLoaded() throws Error {
        driver.get(URL);
        super.isLoaded();
    }

    public SearchResults search(String query) {
        driver.findElement(getLocator("search_field")).sendKeys(query, Key
s.ENTER);
        return new SearchResults(driver).get();
    }

}
```

In the above page object, we are:

- initializing the class with an instance of `WebDriver` and the locators file `search-home.properties` which is expected to be present in `src/main/resources/locators`.
- creating `isLoaded` method that gives the condition to validate the successful loading of a page by navigating to URL and waiting for its ***readyState*** to change to ***complete***. This method will be used by `LoadableComponent` class's method `get()` to create an instance of `GoogleSearch` class after checking the successful loading of the class
- creating a `search` method that is one of the actions that is possible on the Google Search page that takes a query to search as a parameter and performs typing text onto the search field and hitting ***ENTER*** button using methods from `AbstractBasePage` class

After performing the `search` operation, we will be moving to the next state

After performing the `search` operation, we will be moving to the next state, the `SearchResults` page.

Locator file for GoogleSearch page

`src/main/resources/locators/search-home.properties`

```
search_field.name = q
```

Implementation of SearchResults page

```
import org.openqa.selenium.WebDriver;

public class SearchResults extends AbstractBasePage<SearchResults> {

    public SearchResults(WebDriver driver) {
        super(driver, "search-results.properties");
    }

    @Override
    protected void isLoaded() throws Error {
        assert driver.getCurrentUrl().contains("search?");
    }

    public SearchResults nextPage() {
        driver.findElement(getLocator("next_page")).click();
        return this;
    }

}
```

In the above page object, we are:

- initializing the class with an instance of `WebDriver` and the locators file `search-results.properties`, which is expected to be present in `src/main/resources/locators`
- creating `isLoaded` method that gives the condition to validate the successful loading of a page by checking whether its URL contains ***"search?"***. This method will be used by `LoadableComponent` class's method `get()` to create an instance of `SearchResults` class after checking the successful loading of the class.

- creating the `nextPage` method which is one of the actions that is possible on the Search Results page and performs clicking on the next page using methods from `AbstractBasePage` class

After performing `nextPage` operation, we will be moving to the next state, `SearchResults` page.

Locator file for SearchResults page

`src/main/resources/locators/search-results.properties`

```
next_page.xpath = //span[contains(text(), 'Next')]
```

In the next lesson, you'll learn about managing static test data.