

# Loading the Components

This lesson introduces a class called `LoadableComponent` and explains why and how to use it.

## WE'LL COVER THE FOLLOWING ^

- What is `LoadableComponent`?
- Without `LoadableComponent`
- With `LoadableComponent`

## What is `LoadableComponent`? #

`LoadableComponent` is a base class that aims to ensure:

- Pages are loaded.
- The initial state of the page is asserted.
- The elements are intractable before we use them.

It helps in debugging the failure of a page to load and in reducing flakiness due to page load issues.

## Without `LoadableComponent` #

Following is a sample `LoginPage.java` (Page Object) without `LoadableComponent`.

```
package com.example.webdriver;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;

public class LoginPage {

    private final WebDriver driver;
```

```

public LoginPage(WebDriver driver) {

    this.driver = driver;
}

public void setName(String name) {
    WebElement field = driver.findElement(By.id("name"));
    clearAndType(field, name);
}

public void setPassword(String pwd) {
    WebElement field = driver.findElement(By.name("password"));
    clearAndType(field, pwd);
}

public HomePage submit() {
    driver.findElement(By.id("submit")).click();
    return new HomePage(driver);
}

private void clearAndType(WebElement field, String text) {
    field.clear();
    field.sendKeys(text);
}
}

```

Creating an instance of the page object:

```
LoginPage page = new LoginPage(driver)
```

## With **LoadableComponent** #

Now, to turn this code into a loadable component, **LoginPage** has to extend the **LoadableComponent** class and implement two methods **load()** and **isLoading()**.

```

public class LoginPage extends LoadableComponent<LoginPage> {

    // rest of class ignored for now

    @Override
    protected void load() {
        driver.get("http://www.facebook.com");
    }
}

```

```

        driver.get("https://www.facebook.com/");
    }

    @Override
    protected void isLoading() throws Error {
        String url = driver.getCurrentUrl();
        assertTrue(url.contains("facebook"), "current url " + url + " expected to contain 'facebook'");
    }
}

```

Creating an instance of the page object:

```
LoginPage page = new LoginPage(driver).get()
```

Please notice the calling `get()` method on the created page object instance. This will internally call the `isLoading()` method, to ensure that the page is loaded properly by satisfying the condition in the `isLoading()` method. When the URL is not correct, the control goes to `load()` function for the mitigation because the assertion failed. The `load()` function simply reloads the page. This will double-check that the page is indeed loaded completely.

One more implementation of `LoadableComponent`, called `SlowLoadableComponent`, is used in place of `LoadableComponent` when the page may load slowly. Here, we give a maximum **timeout** before the page could load completely.

```

public class LoginPage extends SlowLoadableComponent<LoginPage> {

    public LoginPage() {
        super(java.time.Clock.systemDefaultZone(), 60000);
    }

    // rest of class ignored for now

    @Override
    protected void load() {
        driver.get("https://www.facebook.com/");
    }

    @Override
    protected void isLoading() throws Error {

```

```

String url = driver.getCurrentUrl();
assertTrue(url.contains("facebook"), "current url " + url + " expected to contain 'facebook'");
}

@Override
protected void isError() throws Error {
    super.isError();
}
}

```

Behavior is just the same as `LoadableComponent` except that the `isLoading()` method is checked until the given maximum timeout is reached while checking for `isLoading()` and `isError()` at certain intervals.

---

In the next lesson, you'll learn to implement a `WebDriver` manager for handling `WebDriver` instances.