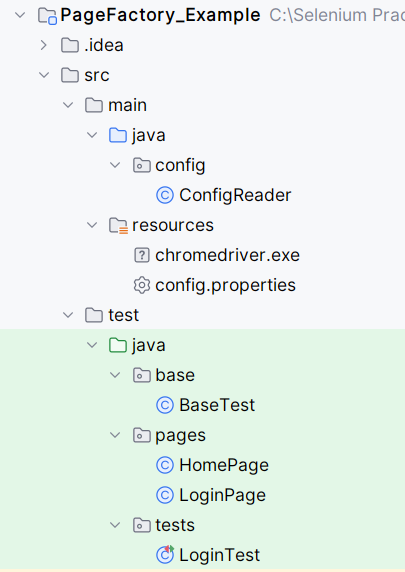
**Page Object Model (POM) using PageFactory and ConfigReader**

Using a **config.properties** file and a **ConfigReader** class in your Selenium project centralizes configuration settings such as URLs, browser type, and timeouts. Here's how you can design these components:

**Code Structure Overview**

The project structure follows the modular approach:

****

**1. config.properties File**

Create a config.properties file in your src/main/resources directory.

*# Application URL*baseUrl=http://www.automationpractice.pl/index.php  
  
*# LoginPage URL*loginUrl=http://www.automationpractice.pl/index.php?controller=authentication&back=my-account  
  
searchQuery=dress  
  
*# Test Credentials*email=gihec51991@ronete.com  
password=123456789

### ****2.**** ConfigReader ****Class****

This class will read values from the config.properties file using Java's Properties class.

package config;  
  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.util.Properties;  
  
*//This class will read values from the config.properties file using Java's Properties class.*public class ConfigReader {  
 private static Properties *properties*;  
  
 static {  
 try {  
 *// Load the config.properties file* FileInputStream fileInputStream = new FileInputStream("src/main/resources/config.properties");  
 *properties* = new Properties();  
 *properties*.load(fileInputStream);  
 } catch (IOException e) {  
 e.printStackTrace();  
 throw new RuntimeException("Failed to load configuration file.");  
 }  
 }  
  
 *// Method to fetch property values* public static String getProperty(String key) {  
 return *properties*.getProperty(key);  
 }  
}

Using a **config.properties** file and a **ConfigReader** class in your Selenium project centralizes configuration settings such as URLs, browser type, and timeouts. Here's how you can design these components:

**1. config.properties File**

Create a config.properties file in your src/resources or src/test/resources directory.

properties

Copy code

# Application URL

url=http://www.automationpractice.pl/index.php

# WebDriver properties

browser=chrome

driver\_path=path/to/chromedriver

# Implicit wait time (in seconds)

implicit\_wait=10

# Test credentials

username=testuser@example.com

password=password123

**2. ConfigReader Class**

This class will read values from the config.properties file using Java's Properties class.

package config;  
  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.util.Properties;  
  
*//This class will read values from the config.properties file using Java's Properties class.*public class ConfigReader {  
 private static Properties *properties*;  
  
 static {  
 try {  
 *// Load the config.properties file* FileInputStream fileInputStream = new FileInputStream("src/main/resources/config.properties");  
 *properties* = new Properties();  
 *properties*.load(fileInputStream);  
 } catch (IOException e) {  
 e.printStackTrace();  
 throw new RuntimeException("Failed to load configuration file.");  
 }  
 }  
  
 *// Method to fetch property values* public static String getProperty(String key) {  
 return *properties*.getProperty(key);  
 }  
}

**3. Using the ConfigReader in BaseTest**

Update the BaseTest class to use ConfigReader for fetching configuration values.

package base;

import config.ConfigReader;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import java.time.Duration;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.BeforeMethod;

public class BaseTest {

protected WebDriver driver;

// Set up the WebDriver and navigate to the URL

@BeforeMethod

public void setUp() {

System.setProperty("webdriver.chrome.driver", "src/main/resources/chromedriver.exe");

driver = new ChromeDriver();

driver.manage().window().maximize();

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));

//using ConfigReader for fetching configuration values.

String baseUrl = ConfigReader.getProperty("baseUrl");

driver.get(baseUrl);

}

// Close the WebDriver after test execution

@AfterMethod

public void tearDown() {

if (driver != null) {

driver.quit();

}

}

}

**4. The HomePage and LoginPage classes will contain the following java code.**

package pages;  
  
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.support.FindBy;  
import org.openqa.selenium.support.PageFactory;  
  
public class HomePage {  
 WebDriver driver;  
  
 *// Locate the search box using @FindBy* @FindBy(id = "search\_query\_top")  
 WebElement searchBox;  
  
 *// Locate the search button using @FindBy* @FindBy(name = "submit\_search")  
 WebElement searchButton;  
  
 *// Constructor to initialize WebElements* public HomePage(WebDriver driver) {  
 this.driver = driver;  
 *// initElements initializes all @FindBy-annotated fields in this class* PageFactory.*initElements*(driver, this);  
 }  
  
 *// Perform a search operation* public void search(String query) {  
 searchBox.sendKeys(query);  
 searchButton.click();  
 }  
}

package pages;  
  
import config.ConfigReader;  
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.support.FindBy;  
import org.openqa.selenium.support.PageFactory;  
  
public class LoginPage {  
 WebDriver driver;  
  
 *// Locate the email input field using @FindBy* @FindBy(id = "email")  
 private WebElement emailField;  
  
 *// Locate the password input field using @FindBy* @FindBy(id = "passwd")  
 private WebElement passwordField;  
  
 *// Locate the login button using @FindBy* @FindBy(id = "SubmitLogin")  
 private WebElement loginButton;  
  
 *// Constructor to initialize WebElements* public LoginPage(WebDriver driver) {  
 this.driver = driver;  
 PageFactory.*initElements*(driver, this); *// Initialize WebElements using PageFactory* }  
  
 *// Method to perform login* public void login(String email, String password) {  
 emailField.sendKeys(email); *// Enter email* passwordField.sendKeys(password); *// Enter password* loginButton.click(); *// Click the login button* }  
}

**5. Using the ConfigReader in LoginTest**

package tests;  
  
import base.BaseTest;  
import config.ConfigReader;  
import org.testng.Assert;  
import org.testng.annotations.Test;  
import pages.HomePage;  
import pages.LoginPage;  
  
public class LoginTest extends BaseTest {  
  
 *// Test login functionality* @Test  
 public void testLogin() {  
 HomePage homePage = new HomePage(driver); *// Initialize HomePage;* LoginPage loginPage= new LoginPage(driver); *// Initialize LoginPage;  
  
 // Using Values from config.properties* String loginUrl = ConfigReader.*getProperty*("loginUrl");  
  
 String email = ConfigReader.*getProperty*("email");  
 String password = ConfigReader.*getProperty*("password");  
  
 String query = ConfigReader.*getProperty*("searchQuery");  
  
 homePage.search(query); *// Search for "dress"* driver.navigate().to(loginUrl);  
 loginPage.login(email, password); *// Perform login* Assert.*assertTrue*(driver.getCurrentUrl().contains("my-account"), "Login failed!"); *// Assert login success* }  
}

### ****Benefits of This Approach****

1. **Centralized Configuration**: All configurations are stored in one place, making updates easy.
2. **Flexibility**: Supports dynamic configurations for different environments (e.g., staging, production).
3. **Scalability**: New properties can be added to config.properties without modifying the code.
4. **Readability**: Configuration settings are separated from the test logic.

This approach is widely used in Selenium projects for better maintainability and scalability.