УДК 004.054
DOI https://doi.org/10.32689/maup.it.2022.4.2

Віра ГАРАСИМІВ
кандидат технічних наук, доцент кафедри комп'ютерних систем і мереж, Івано-Франківський національний технічний університет нафти і газу, вул. Карпатська, 15, Івано-Франківськ, індекс 76019 (vira.harasymiv@nung.edu.ua)
ORCID: 0000-0002-6613-3549

Тарас ГАРАСИМІВ
асистент кафедри комп'ютерних систем і мереж, Івано-Франківський національний технічний університет нафти і газу, вул. Карпатська, 15, Івано-Франківськ, індекс 76019 (tarikksm@gmail.com)
ORCID: 0000-0003-1731-0286

Vira HARASYMIV
Candidate of Technical Sciences, Associate Professor at the Computer Systems and Networks Department, Ivano-Frankivsk National Technical University of Oil and Gas, 15 Karpatska str., Ivano-Frankivsk, Ukraine, postal code 76019 (vira.harasymiv@nung.edu.ua)

Taras HARASYMIV
Assistant at the Computer Systems and Networks Department, Ivano-Frankivsk National Technical University of Oil and Gas, 15 Karpatska str., Ivano-Frankivsk, Ukraine, postal code 76019 (tarikksm@gmail.com)

Бібліографічний опис статті: Гарасимів, В., Гарасимів, Т. (2022). Особливості структури проєкту із використанням патернів проєктування Page Object та PageFactory. *Інформаційні технології та суспільство,* 4 (6), 14–21. DOI: https://doi.org/10.32689/maup.it.2022.4.2

Bibliographic description of the article: Harasymiv, V., Harasymiv, T. (2022). Osoblyvosti struktury proyektu iz vykorystannyam paterniv proyektuvannya Page Object ta PageFactory [Features of the project structure with using the Page Object and Page Factory design patterns]. *Informatsiini tekhnolohii ta suspilstvo – Information technology and society*, 4 (6), 14–21. DOI: https://doi.org/10.32689/maup.it.2022.4.2

ОСОБЛИВОСТІ СТРУКТУРИ ПРОЄКТУ ІЗ ВИКОРИСТАННЯМ ПАТЕРНІВ ПРОЄКТУВАННЯ PAGE OBJECT ТА PAGE FACTORY

Стаття присвячена дослідженню особливостей структури проєктів із використанням патернів проєктування Page Object та Page Factory для написання авто-тестів. У якості мови програмування обрано об'єктно-орієнтовану мову програмування Java, а проєкти створено в IntelliJ IDEA.

Розглянуто тест кейс створення нового акаунту на сайті. Написано відповідні авто-тести із використанням патернів проєктування PageObject та PageFactory та із застосуванням програмних бібліотек Selenium-java, WebDriverManager та фреймворку TestNG. Інформація, яка є необхідною для успішного створення нового акаунту на сайті, зчитується з файлу, який має розширення properties та зберігається у папці resources. Сформано звіт успішності проходження представленого авто-тесту.

Кожну сторінку веб-додатку представлено окремим класом, в якому знаходяться методи, що будуть працювати з ними. Веб-елементи є прихованими (private) та зберігаються окремо у проєкті, де використано лише патерн проєктування Page Object. Методи класів сторінок повертають нові об'єкти класів сторінок відповідно до написаного тестового сценарію. Скрипт авто-тестів відокремлено від веб-елементів та методів, що імітують дії користувача.

Також використовуючи анотації @BeforeSuite, @AfterSuite та @BeforeClass, створені методи, які відповідають початку та завершенню кожного тестового набору.

Наведено структури проєктів із використанням патерну Page Object та Page Factory. Так як патерн проєктування Page Factory є добре оптимізованим та використовується для ініціалізації об'єктів сторінки або для створення об'єкта самої сторінки, він спрощує структуру проєкту. Веб-елементи відповідних сторінок зберігаються у відповідних класах, в не окремо від них.

Ключові слова: *проєкт, патерн проєктування, Page Object, Page Factory, авто-тест, веб-елемент, веб-додаток, фреймворк.*

FEATURES OF THE PROJECT STRUCTURE WITH USING THE PAGE OBJECT AND PAGE FACTORY DESIGN PATTERNS

The article is devoted to the study of the features of the project structure using the Page Object and Page Factory design patterns for writing auto-tests. The object-oriented programming language Java was chosen as the programming language, and the projects were created in IntelliJ IDEA.

The test case of creating a new account on the site is considered. Corresponding auto-tests are written using the PageObject and PageFactory design patterns and using Selenium-java, WebDriverManager and the TestNG framework. Information that is necessary for successfully creating a new account on the site is read from a file with the extension properties and stored in the resources folder. A report on the success of passing the presented auto-test is created.

Each page of the web application is represented by a separate class that contains the methods that work with them. Web elements are hidden (private) and stored separately in the project, where only the Page Object design pattern is used. Page class methods return new page class objects according to the written test script. The auto-test script is separated from web elements and methods simulating user actions.

Also using the @BeforeSuite, @AfterSuite and @BeforeClass annotations, methods are created that correspond to the start and end of each test suite.

The structures of projects using the Page Object and Page Factory patterns are given. Since the Page Factory design pattern is well-optimized and is used to initialize page objects or to create the page object itself, it simplifies the project structure. The web elements of the corresponding pages are stored in the corresponding classes, not separately from them.

Key words: *project, design pattern, Page Object, Page Factory, auto-test, web element, web application, framework.*

Постановка проблеми. Написанна авто-тестів є важливим інструментом пошуку дефектів на ранніх етапах розробки ПЗ (програмного забезпечення). Саме завдяки автоматизації тестування можливо зменшити вартість виправлення дефектів та покращити процес забезпечення якості ПЗ. Хоча написання авто-тестів може здатися легким завданням для розробників та інженерів, існує велика ймовірність одержання неефективної реалізації тестових сценаріїв та, як наслідок, поганої їхньої підтримки. Іноді, для того щоб змінити один веб-елемент на веб-сторінці, на який покладалася велика кількість тестів, потрібно перевірити та оновити увесь скрипт цих тестів. Це забирає багато часу та зменшує ефективність вровадження авто-тестів на ранній стадії розробки ПЗ [1]. Тому набули свого розвитку різноманітні патерни проєктування для написання підтримуваних та багаторазових авто-тестів.

Аналіз останніх досліджень та публікацій. Розробка ПЗ на даний час у своїй більшості реалізується із застосуванням гнучних підходів та інкрементальних ітеративних моделей розробки ПЗ, які побудовані на можливості постійного внесення змін [2, с. 43–44]. Це призвело до збільшення кількості регресійних тестів та недоречності їхнього проходження вручну. Тому автоматизація тестування ПЗ набула широко використання.

Для кращого розуміння авто-тестів, а також для їхньої підтримки існує ряд патернів проєктування. Найпоширенішими серед них є Page Object, Page Factory та Page Element [3, с. 50–53]. Основна проблема патернів заключається в їхньому коректному використанні та у розумінні, що застосування конкретного патерну повинно, в першу чергу, вирішити поставлене завдання [4, с. 62–63].

Постановка завдання. Page Object – один із найпопулярніших архітектурних рішень в автоматизації тестування. Даний патерн проєктування допомагає інкапсулювати роботу з окремими елементами сторінки, що, у свою чергу, дає можливість зменшити обсяг коду та його підтримку [5, с. 44–46]. При створенні проєкту із використанням патерну Page Object необхідно врахувати те, що скрипт авто-тестів повинен бути розділений від класів сторінок, які містять відповідні їм методи. Веб-елементи сторінок зберігаються окремо. Із появою патерну проєктування Page Factory, процес написання авто-тестів значно полегшився, так як даний патерн дає можливість зберігати веб-елементи сторінки у відповідному їй класі, що, у свою чергу, вносить певні зміни у структуру самого проєкту [6]. Тому метою даної роботи є врахування особливостей структури проєкту із використанням даних патернів проєктування.

Виклад основного матеріалу. Припустимо, що об'єкт тестування (test object) – це сайт http://practice.bpbonline.com/index.php. Як приклад, розглянемо тест кейс перевірки успішної реєстрації. Процес декомпозиції даного тест кейсу на класи (сторінки) показано на рисунку 1. Спочатку ми

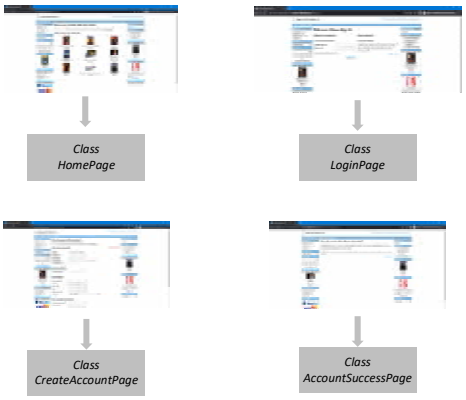

Рис. 1. Процес декомпозиції тестового сценарію на класи згідно патерну проєктування PageObject

заходимо на головну сторінку сайту – клас HomePage. Для реєстрації вибираємо меню «My Account» і переходимо на іншу сторінку (клас LoginPage). У колонці «New Customer» натискаємо на кнопку «Continue». Переходимо на сторінку заповнення необхідної інформації для створення нового акаунту (клас AccountPage). Підтверджуємо введену інформацію. Акаунт успішно створено (клас AccountSuccessPage). Для створення проєкту в IntelliJ IDEA додано залежності Selenium-java, TestNG та WebDriverManager у файл POM нашого проєкту. Структура нашого проєкту показана на рисунку 2.

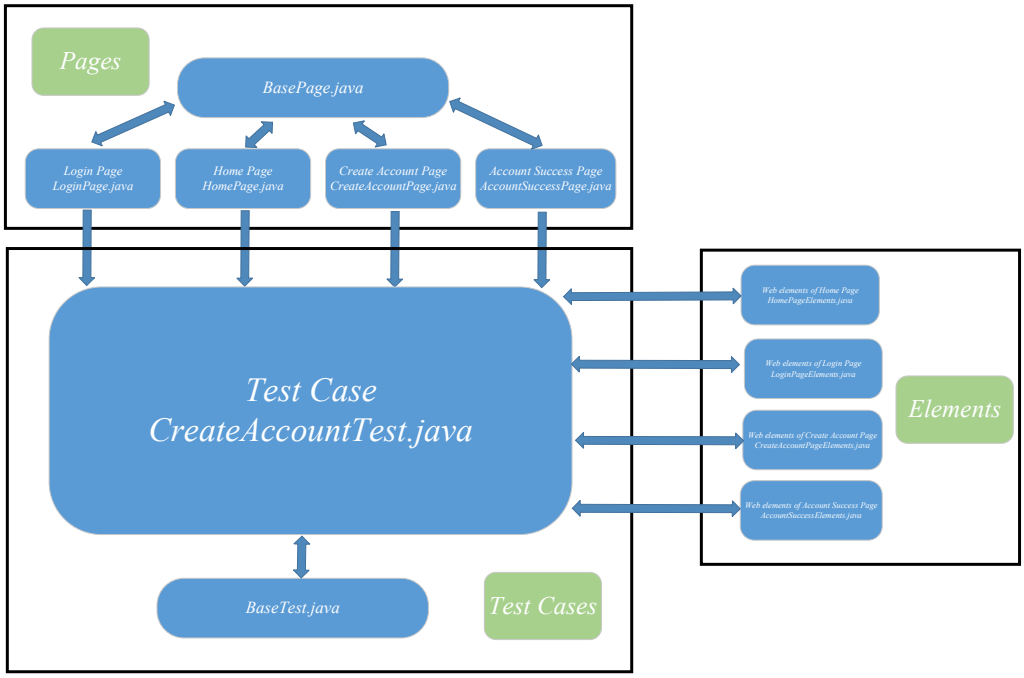

Рис. 2. Структура проєкту із використанням патерну PageObject

Клас BasePage є базовим класом, нащадками якого будуть класи наших сторінок. Конструктор відповідає за ініціалізацію WebDriver та WebDriverWait [7]. Крім того, до даного класу можна додати методи різних очікувань:

```
public class BasePage {
  private static final int TIMEOUT = 30;

  protected WebDriver driver;
  private WebDriverWait wait;

  public BasePage(WebDriver driver) {
    this.driver = driver;
    wait = new WebDriverWait(driver, Duration.ofSeconds(TIMEOUT));
  }

  protected void waitForElementToAppear(By locator) {
    wait.until(ExpectedConditions.visibilityOfElementLocated(locator));
  }
}
```

Клас HomePage містить метод clickOnMyAccountButton(), який реалізує процес вибору користувачем меню «MyAccount» та повертає об'єкт класу LoginPage, оскільки після вибору меню «MyAccount» ми переходимо на дану сторінку:

```
public class HomePage extends BasePage{

  public HomePage(WebDriver driver) {
    super(driver);
  }

  public LoginPage clickOnMyAccountButton(By myAccountMenu){
    driver.findElement(myAccountMenu).click();
    return new LoginPage(driver);
  }
}
```

Аналогічно клас LoginPage містить відповідний йому метод clickOnContinueButton(), що реалізує натискання кнопки «Continue» на даній сторінці. Клас CreateAccountPage містить методи, що реалізують дії користувача при заповненні усіх необхідних полів реєстрації. Інформацію, яку вводить користувач, зчитується з файлу, який має розширення properties та зберігається у папці resources:

```
testdata.user_name = Test User First Name
testdata.user_last_name = Test User Last Name
testdata.dateOfBirthday = 07/30/1988
testdata.userCompany = Test User Company
testdata.userStreet = Test User street
testdata.postCode = 12300
testdata.userCity = User City
testdata.userState = User state
testdata.userTelephoneNumber = 380661234590
testdata.userPassword = 123456789
```

Клас AccountSuccesPage містить метод getActualMessage(), який повертає повідомлення, яке користувач отримає після введення та підтвердження необхідних даних для реєстрації:

```
public class AccountSuccessPage extends BasePage{

    public AccountSuccessPage(WebDriver driver) {
        super(driver);
    }

    public String getActualMessage(By actualMessage){
        waitForElementToAppear(actualMessage);
        return driver.findElement(actualMessage).getText();
    }
}
```

Елементи сторінок HomePage, LoginPage, CreateAccountPage та AccountSuccessPage зберігатимо окремо, оскільки при їхньому збереженні у відповідних їм класах сторінок можливо одержати виключення. Веб-елементи об'являємо приватними, а для їхнього використання застосовуємо спеціальні методи.

Клас BaseTest містить усі загальні функціональні можливості і змінні тестових класів, тому усі тестові класи є нащадками даного класу [8]:

```
public class BaseTest {
  protected WebDriver driver;
  private static final String OSCOMMERCE_URL =
"http://practice.bpbonline.com/index.php";

  @BeforeSuite
  public void beforeSuite(){
    WebDriverManager.chromedriver().setup();
  }

  @BeforeClass
  public void setDriver(){
    driver = new ChromeDriver();
    driver.manage().window().maximize();
    driver.get(OSCOMMERCE_URL);
  }

  @AfterSuite
  public void afterSuite(){
    if(driver != null){
      driver.quit();
    }
  }
}
```

Клас CreateAccountTest містить авто-тест для перевірки створення нового акаунту. Тест успішно пройде, якщо користувач після введення та підтвердження необхідних даних одержить повідомлення про успішну реєстрацію. Скрипт авто-тесту наведено нижче:

```
 public class CreateAccountTest extends BaseTest{
  private static final String MESSAGE = "Your Account Has Been Created";

  @Test
  public void createAccountTest() {
    String actualMessage = new HomePage(driver)
        .clickOnMyAccountButton(HomePageElements.getMyAccountMenu())
        .clickOnContinueButton(LoginPageElements.getContinueButton())
        .selectGender(CreateAccountPageElements.getGenderRadioButton())
        .enterFirstName(CreateAccountPageElements.getFirstNameField())
```

```
        .enterLastName(CreateAccountPageElements.getLastNameField())
        .enterDateOfBirth(CreateAccountPageElements.getDateOfBirth())
        .enterUserEmail(CreateAccountPageElements.getEmailField())
        .enterCompanyName(CreateAccountPageElements.getCompanyNameField())
        .enterStreetAddress(CreateAccountPageElements.getStreetAddressField())
        .enterPostCode(CreateAccountPageElements.getPostCodeField())
        .enterUserCity(CreateAccountPageElements.getCityField())
        .enterUserState(CreateAccountPageElements.getStateField())

.selectCountry(CreateAccountPageElements.getSelectCountryMenu(),CreateAccountPag
eElements.getUkraineOption())
        .enterTelephoneNumber(CreateAccountPageElements.getTelephoneNumber())

.checkNewsLetterCheckBox(CreateAccountPageElements.getNewsletterCheckbox())

.createPassword(CreateAccountPageElements.getPasswordField(),CreateAccountPageEl
ements.getPasswordFieldConfirmation())
        .submitEnteredInformation(CreateAccountPageElements.getSubmitButton())
        .getActualMessage(AccountSuccessPageElements.getActualMessage());
    Assert.assertTrue(actualMessage.contains(MESSAGE));
  }
}
```

Так як для написання авто-тесту ми використовували фреймворк TestNG [9], то після запуску нашого авто-тесту, ми одержали звіт успішності виконання тестового сценарію, який показано на рисунку 3.

Розглянемо процес написання даного тест кейсу із використанням патерну Page Factory, який являє собою розширення патерну Page Object, є добре оптимізованим та використовується для ініціалізації об'єктів сторінки або для створення об'єкта самої сторінки. Page Factory ініціалізує веб-елементи класу сторінки із використанням анотації @FindBy, тому вони зберігаються у відповідному класі сторінки, а не окремо від неї [10]. Це, у свою чергу, полегшує процес написання авто-тестів та впливає на саму структура проєкту (рис. 4).

Test	# Passed	# Skipped	# Retried	# Failed	Time (ms)	Included Groups	Excluded Groups
Smoke							
Creating new account test	1	0	0	0	8 162		

Class	Method	Start	Time (ms)
Smoke			
Creating new account test — passed			
com.IFNTUNG.edu.tests.CreateAccountTest	createAccountTest	1669586950220	2876

Creating new account test

com.IFNTUNG.edu.tests.CreateAccountTest#createAccountTest

Рис. 3. Звіт успішності виконання тестового сценарію для перевірки створення нового акаунту

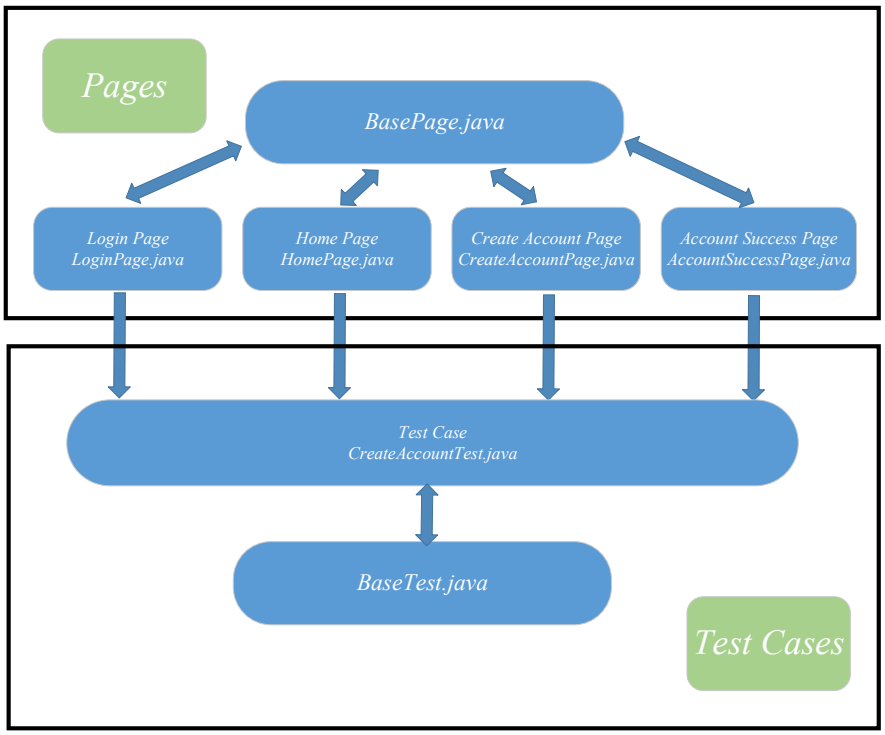

Рис. 4. Структура проєкту із використанням патернів PageObject та PageFactory

Також для ініціалізації всіх елементів змінемо конструктор класу BasePage таким чином:

```
public BasePage(WebDriver driver) {
    this.driver = driver;
    wait = new WebDriverWait(driver, Duration.ofSeconds(TIMEOUT));
    PageFactory.initElements(driver,this);
}
```

Авто-тест для перевірки створення нового акаунту буде мати спрощений вигляд:

```
@Test
public void createAccountTest() {
  String actualMessage = new HomePage(driver)
      .clickOnMyAccountButton()
      .clickOnContinueButton()
      .selectGender()
      .enterFirstName()
      .enterLastName()
      .enterDateOfBirth()
      .enterUserEmail()
      .enterCompanyName()
      .enterStreetAddress()
      .enterPostCode()
      .enterUserCity()
      .enterUserState()
```

```
        .selectCountry()
        .enterTelephoneNumber()
        .checkNewsLetterCheckBox()
        .createPassword()
        .submitEnteredInformation()
        .getActualMessage();
    Assert.assertTrue(actualMessage.contains(MESSAGE));
}
```

Висновки. Отже, застосування патернів проєктування Page Object та Page Factory полегшують процес написання авто-тестів, їх нескладно підтримувати, а скрипт авто-тестів є зрозумілим для сприйняття. Необхідно врахувати, що структура самого проєкту із використанням патерну Page Object буде відрізнятися тим, що веб-елементи сторінок зберігтимуться окремо.

Список використаних джерел:

1. Yoni Flenner. Page Object Model-Make It Simple, Use Abstraction. URL: https://blog.testproject.io/2017/07/16/page-object-model/
2. Graham D., Black R., Erik van Veenendal. Foundations of Software Testing: ISTQB Certification, 4 th Edition. United Kingdom : EMEA, 2018. 273 p.
3. Anton Angelov, “Design Patterns for High-Quality Automated Tests: High-Quality Test Attributes and Best Practices”. United States: Kindle Edition, 2021. 348 p.
4. Erich Gamma, Richard Helm, Ralph Johnson, John Vissdes, "Design Patterns: Elements of Reusable Object-Oriented Software". United States: Addison-Wesley, 1994. 395 p.
5. Seretta Gamba, Dorothy Graham, “A Journey through Test Automation Patterns: One team’s adventures with the Test Automation Patterns”. United States: CreateSpace Independent Publishing Platform, 2018. 364 p.
6. Page Object Model (POM). URL: https://www.geeksforgeeks.org/page-object-model-pom/
7. Amir Ghahrai. Page Object Model Framework with Java and WebDriver. URL: https://devqa.io/page-object-framework-java-webdriver/
8. Krishna Rungta. Page Object Model (POM) & Page Factory in Selenium. URL: https://www.guru99.com/page-object-model-pom-page-factory-in-selenium-ultimate-guide.html
9. Selenium and TestNG. URL: https://testng.org/doc/selenium.html
10. Gunjan Kaushik, Ravinder Singh. Page Object Model using Page Factory in Selenium WebDriver. URL: https://www.toolsqa.com/selenium-webdriver/page-factory-in-selenium/

References:

1. Yoni Flenner. Page Object Model-Make It Simple, Use Abstraction. URL: https://blog.testproject.io/2017/07/16/page-object-model/ [in English]
2. Graham D., Black R., Erik van Veenendal. Foundations of Software Testing: ISTQB Certification, 4 th Edition. United Kingdom : EMEA, 2018. 273 p. [in English]
3. Anton Angelov, “Design Patterns for High-Quality Automated Tests: High-Quality Test Attributes and Best Practices”. United States: Kindle Edition, 2021. 348 p. [in English]
4. Erich Gamma, Richard Helm, Ralph Johnson, John Vissdes, "Design Patterns: Elements of Reusable Object-Oriented Software". United States: Addison-Wesley, 1994. 395 p. [in English]
5. Seretta Gamba, Dorothy Graham, “A Journey through Test Automation Patterns: One team’s adventures with the Test Automation Patterns”. United States: CreateSpace Independent Publishing Platform, 2018. 364 p. [in English]
6. Page Object Model (POM). URL: https://www.geeksforgeeks.org/page-object-model-pom/ [in English]
7. Amir Ghahrai. Page Object Model Framework with Java and WebDriver. URL: https://devqa.io/page-object-framework-java-webdriver/ [in English]
8. Krishna Rungta. Page Object Model (POM) & Page Factory in Selenium. URL: https://www.guru99.com/page-object-model-pom-page-factory-in-selenium-ultimate-guide.html [in English]
9. Selenium and TestNG. URL: https://testng.org/doc/selenium.html [in English]
10. Gunjan Kaushik, Ravinder Singh. Page Object Model using Page Factory in Selenium WebDriver. URL: https://www.toolsqa.com/selenium-webdriver/page-factory-in-selenium/ [in English]