**Складові системи запуску тестів:**

1. Сервер безперервної інтеграції – **Jenkins.** (Для цього необхідний ще репозиторій **git**)
2. Складальник – **Maven.**
3. Фрейворк запуска тестів – **TestNG, jUnit…**
4. **Тести** (ЯП: java).
5. Драйвер системи, що тестується – **WebDriver.**
6. Браузер – **Firefox.**
7. Система, що тестується.

**Налаштування середовища java на локальному комп’ютері.**

Завантажити дистрибутив **JDK** та інсталювати на локальній машині. Бажано інсталювати в корінь диску С. Назва папки «java» без пробілів(для змінного середовища). Змінне середовище налаштовується для того, щоб була можливість працювати з додатком безпосередньо з консолі не вказуючи повний шлях до виконуваного файлу додатка.

Налаштування змінного середовища КП для java:

1. Перейти в *Система\Додаткові параметри системи\Змінне середовище\Системні змінні.*
2. Створити: «Ім’я змінної»=«JAVA\_HOME»; «Значення змінної»= «**C:\Java\jdk1.8.0\_92**»(шлях до встановленої java).
3. Редагувати системну змінну “Path”. В поле «Значення змінної» через «;» без пробіла додати запис «**%JAVA\_HOME%\bin**»

Перевірити інсталювання та налаштування **java**,а також визначення версії: в консоль ввести команду «**java -version»**.

**Налаштування Maven.**

Завантажити дистрибутив **apache-maven** та інсталювати на локальній машині. Бажано інсталювати в С:\java. Назва папки «java» без пробілів(для змінного середовища).

Налаштування змінного середовища КП для Maven:

1. Перейти в *Система\Додаткові параметри системи\Змінне середовище\Системні змінні.*
2. Створити: «Ім’я змінної»=«M3\_HOME»; «Значення змінної»= **«C:\Java\apache-maven-3.3.9»(**шлях до встановленого Maven).
3. Редагувати системну змінну “Path”. В поле «Значення змінної» через «;» без пробіла додати запис «**%M3\_HOME%\bin**».

Перевірити інсталювання та налаштування **maven**: в консоль ввести команду «**mvn -version»**.

**Створення проекту із архітипу.**

1. Необхідно перейти на git та клонувати з репозиторія Баранцева архітип:

git clone git://github.com/barancev/Selenium2-Java-QuickStart-Archetype.git

cd Selenium2-Java-QuickStart-Archetype (перейти в діректорю)

mvn install (виконати команду – інсталювати архітип)

1. Генеруємо проект на основі архітипу:

mvn archetype:generate -DarchetypeGroupId=com.sebarmeli -DarchetypeArtifactId=selenium2-java-quickstart-archetype -DarchetypeVersion=0.3 -DgroupId=<mygroupId> -DartifactId=<myartifactId>,

де ми вказуємо DgroupId=ua.bms.test(назва групи розробки, унікальний ідентифікатор групи проектів в рамках компанії) та DartifactId=selenium\_java\_training(назва проекту, тобто унікальний ідентифікатор проекту в рамка групах проектів).

1. В результаті створюється структура підкаталогів.

1-рівень: папка «selenium\_java\_training»

2-рівень: папка “src” та файл “pom.xml”(опис проекту, де описані різні залежності від фреймворка(TestNG, jUnit), від браузера, селеніума…)

3-рівень папки src: папки “test”(будуть знаходитися самі тести, плагіни, які будуть запускати ці тести) та “main”(розміщують допоміжний код, також тут знаходиться фабрика ВебДрайвера)

4-рівень папки «test»: папки “resources” та ”java”

5-рівень папки «resources»: файл testng.xml – конфігураційний файл, який перелічує тести, що будуть запускатися

5-рівень папка “java”: ua/bms/test/pages/ - знаходяться самі тести з базовим тестом для всіх класів TestBase.java.

Після створення проекту, необхідно створити для нього репозиторій на git.

**Налаштування IDE.**

1. Необхідно інсталювати Eclipse.
2. За допомогою Maven створюємо проект для Eclipse IDE:
   1. В консолі переходимо в папку з проектом: selenium\_java\_training;
   2. Вводимо команду: mvn eclipse:eclipse;
   3. Таким чином в директорії проекту створюються додаткові файли для імпорту проекту в Eclipse.
3. Перейти в Eclipse ->File->Import->General->Existing project into Workspace. Обираємо папку selenium\_java\_training та додаємо проект в Eclipse.
4. Налаштовуємо всі залежності бібліотек в Eclipse(якщо наявна червона позначка на директорії проекту в Eclipse):
   1. Window->Preferences->Java->Build Path->Classpath Variables. Повинна бути змінна M2\_REPO. Якщо її нема, то необхідно додати. New -> Name=M2\_REPO, Path= «users\Vova\.m2\repository»(Шлях до робочої директорії Мавен .m2, куди Мавен складає свої тимчасові файли – локальний репозиторій. Зазвичай знаходиться в users\Vova\.m2\repository).

**Створення заготовки для тесту з допомогою рекордера.**

1. Використовується Mozilla Firefox.
2. Завантажуємо плагін Selenium IDE для Mozilla з <http://www.seleniumhq.org/> або з <https://addons.mozilla.org/en-US/firefox/addon/selenium-ide/> .
3. Переходимо на сайт, який ми будемо тестувати.
4. Запускаємо Selenium IDE -> Option->Option->General “Enable experimental features”=TRUE.
5. Для того, щоб експортований файл (запис рекордера) включити в наш тест(в наш проект, що був згенерований з архітипу), потрібно змінити налаштування форматування:
   1. Optional переходимо на закладку Formats копіюємо налаштування, які вже є(обираємо наприклад Java/Junit 4/WebDriver -> Source -> копіюємо все);
   2. Add -> вставляємо раніше скопійоване, називаємо, наприклад, «My Format» -> Save -> OK

Або правимо існуючий (змінюємо Header, Footer, package)

* 1. Заходимо знову в Option->Option-> Formats та обираємо наш створений формат.

1. Копіюємо текст тесту та вставляємо в пакет, що розташований в директоріях src/test/java/назва пакету.
2. В тексті тесту змінюємо назву пакета та назву класу.
3. Необхідно також завантажити та встановити додатковий плагін для запуску TestNG Test: Help -> “Install New Software” Work with=” http://beust.com/eclipse ”, “Contact all update sites…”=FALSE.
4. Зайти на сторінку сайту, який буде тестуватися, з формою авторизації, виконати авторизацію із записуванням з допомогою рекордера;
5. Скопіювати код з закладки «Source»;
6. Створити новий клас в src/test/java з назвою LoginTest. В цьому класі виконати правку назви пакета та назви класу(клас повинен наслідувати клас TestBase)
7. Необхідно налаштувати src/main/resources/application.properties:

site.url=http://10.100.7.71:57775/osop/LoginForm.csp???????? (не працює)

browser.name=firefox

1. Вказуэмо клас(тест) який потрібно буде запускати: src/test/resources/testng.xml -> class name=….
2. Запускаємо тести. Run As – TestNG Test класу LoginTest.java АБО в консолі в папці проекту – mvn test.

Перевірка результатів тестування зберігається html файлі в target/surefire-reports/index.html

**Особливості тестування:**

1. В класі TestBase можна встановити час виконання тесту:

driver.manager().timeouts()……30

Перегляд змінної PATH: в консолі set PATH.

**Запуск тестів на браузерах Opera та IE**

Для цього потрібно завантажити вебдайвери цих браузерів: IEDriverServer.exe і chromedriver.exe.

Ці файли необхідно розташувати в директорії, яка наявна в змінній PATH.

Рекомендація: Створити директорію (наприклад C:\Tools) та присвоїти їй змінну PATH(за аналогією з java і maven).

Для безпосереднього запуску тестів на цих браузерах потрібно зазначити їх у файлі конфігурації:

src\main\resources\application.properties, де browser.name=chrome або IE.

Для Opera та Safari потрібно тільки прописати src\main\resources\application.properties, де browser.name=opera або safari (не потрібно додаткові плагіни-драйвера).

**Запуск тестів на віддаленій машині.**

1. Розгорнути віртуальну машину з Linux.
2. Інсталювати java:
   1. **APT** - засіб управління пакетами.

sudo - sudo (англ. substitute user and do, дословно «подменить пользователя и выполнить») — программа для системного администрирования UNIX-систем, позволяющая делегировать те или иные привилегированные ресурсы пользователям с ведением протокола работы.

PPA - персональный архив пакетов (Personal Package Arhive - PPA)

<http://ubuntovod.ru/instructions/add-remove-disable-ppa-ubuntu.html>

* 1. **sudo -s** \*\*\*(Отримання прав супер адміністратора. Після цього в подальших командах не потрібно використовувати слово "sudo" )
  2. **sudo java -version** \*\*\*(інформація щодо версії встановленої java)
  3. **sudo apt-get update && apt-get remove openjdk**\* \*\*\*(Видалення пакетів java(openjdk\*), встановлених в систему зі збереженням файлів конфігурації)
  4. **sudo add-apt-repository ppa:webupd8team/java** \*\*\*(додавання репозиторія)
  5. **sudo apt-get update** \*\*\*(Отримати оновлені списки пакетів)
  6. **sudo apt-get install oracle-java8-installer** \*\*\*(Встановити / обновити пакет oracle-java8-installer)

1. Інсталювати Selenium Standalone Server:
   1. Завантажити інсталяційний jar файл.
   2. Переглянути наявність попереднього файлу ls;
   3. Запуск: **sudo java –jar ~/Downloads/selenium-server-standalone-2.30.0.jar**
2. Дізнаємося адресу сервера: **ifconfig** (в мене 192.168.12.132)
3. Переходимо до файлу **src\main\resources\application.properties** та присвоюємо значення атрибуту:

**grid2.hub=http://192.168.12.132:4444/wd/hub**

**site.url= http://10.100.7.71:57775/osop/LoginForm.csp**

**Налаштування профіля в maven.**

Якщо тести будуть запускатися через maven, то необхідно налаштування зазначати не в **src\main\resources\application.properties, а в pom.xml**(в кореневій директорії проекта).

1. Необхідно повернути файл src\main\resources\application.properties в початковий стан за допомогою git:

**git status**

**git checkout – src/main/resources/application.properties**

1. Налаштуэмо профілі в файлі **pom.xml**:

<id>firefof\_local</id>

<properties>

<env>local</env>

<user.username></user.username>

<user.password></user.password>

<grid2.hub></grid2.hub>

<site.url>http://10.100.7.71:57775/osop/LoginForm.csp</site.url>

<browser.name>firefox</browser.name>

<browser.version></browser.version>

<browser.platform></browser.platform>

</properties>

<activation>

<activeByDefault>true</activeByDefault>

</activation>

</profile>

<profile>

<id>firefox\_linux</id>

<properties>

<env>prod</env>

<user.username></user.username>

<user.password></user.password>

<grid2.hub>http://192.168.12.132:4444/wd/hub</grid2.hub>

<site.url>http://10.100.7.71:57775/osop/LoginForm.csp</site.url>

<browser.name>firefox</browser.name>

<browser.version></browser.version>

<browser.platform></browser.platform>

</properties>

</profile>

1. В консолі переходимо в директорію з проектом та виконуємо:

mvn test –P firefox\_local

**Сервер безперервної інтеграції Jenkins.**

1. Завантажуємо файл jankins.war в каталог C:\Tools
2. З консолі запускаємо командою(попередньо потрібно перейти в папку Tools):

java –jar Jenkins.war

1. Зберігаємо всі тести в git:

git add .

git commit –m “next step”

git push

1. Заходимо на локал хост: <http://localhost:8080/>
2. Запровнюємо назву, обираєм перший пункт, вказуємо Subversion= <https://github.com/forelock7/selenium_java_training_barancev.git>

**Основні команди.**

Об’єкти:

1. Браузер – запуск, зупинка.
2. Вікно браузера – відкрити, закрити, переключитися, розмір.
3. Діалогові вікна.
4. Сторінка – відкрити. попередня, наступна, виконати JS-код.
5. Елементи – знайти (емуляція – миші, клавіатури; текст, атрибути, стилі, розмір).
6. Фрейми – знайти, переключитися.

**Запуск і зупинка, відкриття сторінок**

Запуск браузера:

**WebDriver driver = new FirefoxDriver();** - WebDriver - інтерфейс

…new ChromeDriver();

…new InternetExplorerDriver();

…new OperaDriver();

..new SafariDriver();

Налаштування додаткових властивостей браузера. Особливо актуально коли на віддаленій машини чи різні версії браузерів, тому вказується назва браузера чи версія……

**DesiredCapabilities caps = new DesiredCapabilities();**

**caps.setCapability(HAS\_NATIVE\_EVENTS, true);**

**WebDriver driver = new FirefoxDriver(caps);**

Атрибути:

**browserName**

**version**

**platform**

Зупинка браузера.

**driver.quit();** - закрити всі вікна і завершити роботу(все чистить). Якщо виконується тестування віддалено, то браузер не закривається, а просто завершується сесія. І потім дозволяється виконати нову сесію.

**driver.close();** - закрити поточне вікно(якщо воно одне. то завершається робота, але не чистяться тимчасові файли). Не можна використовувати для віддаленого тестування.

Відкриття сторінок.

**driver.get(“http://selenium2.ru/”);** - відкрити сторінку та зачекати, поки вона завантажиться. Кінець завантаження – коли браузер встановив спеціальний флажок, а тим часом може виконуватися підвантажування даних чи виконуватися певний скріпт.

**driver.navigate().to(“http://selenium2.ru/”);** - теж саме.

Навігація:

**driver.navigate().refresh();** - оновити

**driver.navigate().back();** - назад

**driver.navigate().forward();** - вперед

**Пошук елементів і очікування**

Базові команди пошуку

**WebElement element = driver.findElement(byLocator);** - знайти перший елемент за заданою умовою НА ВСІЙ СТОРІНЦІ

**List <WebElement> elements = driver.findElements(byLocator);** - знайти всі елементи за заданою умовою.

Типи локаторів (в порядку використання):

**By.id**

**By.name**

**By.cssSelector**

**By.xpath**

**By.tagName**

**By.class**

**By.linkText** (краще не використовувати!!!!!)

**By.partialLinkText**

Пошук в середині елемента:

**WebElement element2 = element1.findElement (byLocator);** - знайти перший елемент за заданою умовою, що знаходиться в середині елемента element1.

**List <WebElement> elements = element1.findElements(byLocator);** - знайти всі елементи за заданою умовою, які знаходяться в середині елемента element1.

Приклад:

Потрібно дістатися до комірки 4-1(рядок 4, стовпчик 1)

Необхідно спочатку знайти таблицю table.t1(селектор CSS – table (тег), t1(клас)), потім 4 рядок та 1 комірка.

**WebElement cell41 =**

**driver.findElement(By.cssSelector(“table.t1”))**

**.findElements(By.tagName(“tr”)).get(3)**

**.findElement(By.tagName(“td”)).get(0);**

Якщо нічого не знайшлося:

* findElement – викидається виключення NoSuchElementException.
* findElements – повертає пустий список.

Неявне очікування.

**driver.manager().timeouts().imlicitlyWait(30, TimeUnit.SECONDS);** - Протягом 30 секунд findElement чекає, поки елемент з’явиться; a findElements чекає, поки хоча б один елемент з’явиться(якщо там ще з’являться після 30 сек, то він вже їх не знайде).

**Явне очікування.**

1. Існує цикл (коли ми за 30 сек 30 раз визиваємо елемент)

**for (int count = 0;; count ++) {**

**if (count >= 30)**

**throw new TimeoutException();**

**try {**

**driver.findElement(locator);**

**break;**

**} catch (NoSuchElementException e) { }**

**Thread.sleep(1000);**

**}**

1. Існує реалізація цього циклу в Selenium.

**WebDriverWait wait =  
new WebDriverWait(driver, 30);**

**WebElement element = wait.until(ExpectedConditions  
 .presenceOfElementLocated(By.id(“dialog-confirm-delete”)));**

Готові класи з реалізаціями:

wait.until(titlels(“New page title”)); - завантаження елемента

wait.until(presenceOfElementLocated(locator)); - поява елемента

wait.until(visibilityOfElementLocated(locator)); - видимість елемента

wait.until(invisibilityOfElementLocated(locator)); - невидимість елемента

wait.until(stalenessOf(element)); - зникнення елемента з DOM

Бонус. Що ще можна почекати.

* visibilityOfElementLocated
* elementToBeClickable
* elementToBeSelected
* textToBePresentInElement
* stalenessOf – зникнення елемента
* alertIsPresent
* titleContains

Порівняння:

|  |  |
| --- | --- |
| Явні очікування | Неявні очікування |
| * на стороні клієнта (не блокується браузер) | * на стороні браузера |
| * чекати можна будь чого | * очікування появи DOM (очікує тільки того, щоб появився елемент) |
| * потрібно писати явно (багато писанини) | * працюють автоматично |
| * TimeoutException | * NoSuchElementException |
| * багато мережевих запитів (велике навантаження) | * Один мережевий запит |

Якщо використовуються явні очікування, то неявні потрібно відключити(поставити в нуль).

Рекомендується явні.

**Прості дії.**

1. Клікнути

* посилання і кнопки
* радіо та чекбокси
* списки

1. Ввести текст
2. Прикріпити файли

**Складні діїю**

1. Клавіатурні поєднання
2. Наведення миші
3. Перетягування
4. Права кнопка миші
5. Подвійний клік

**click** – клік лівою клавішею миші

* любий видимий елемент
* клік в центрі елемента
* спрацьовують всі події

**sendKeys** – введення тексту

* любий видимий елемент
* додавання в кінець текста
* спрацьовують всі події
* прикріплює файли
* поєднання клавіш
* працюють стрілки

**select**- клік по елементу списка (два кліка –по списку та по пункту)

**toggle** – клік по чекбоксу

**check/uncheck** – клік з попередньою перевіркою

Спеціально для випадаючих списків створений допоміжний клас (знаходиться в пакеті **org.openqa.selenium.support.ui):**

**Select dropdown = new Select(element);**

**dropdown.selectByIndex(1);**

**dropdown.selectByValue("mon");**

**dropdown.selectByVisibleText ("Monday");**

**Приклади:**

1. Додати текст на початок

**element.click();**

**element.sendKeys(Keys.HOME);** - використання клавіші «HOME»

**element.sendKeys("some text");**

1. Видалити все, що наявне в полі ввода.

**element.sendKeys(Keys.CONTROL + "a**"); - виділити текст

**Thread.sleep(200);** - зачекати 200мсек

**element.sendKeys(Keys.DELETE); - натиснути DELETE**

1. Засабмітити форму, якщо нема кнопки (можна використовувати замість RETURN – ENTER, але є випадки, коли це може не спрацювати)

**element.sendKeys(  
"text to search" + Keys.RETURN);**

1. Гарячі клавіші потрібно відправляти в елемент body:

**body = driver.findElement(By.tagName("body"));**

**body.sendKeys(  
Keys.CONTROL + Keys.SHIFT + "1");**

1. Вставити багато тексту. Переміщення тексту в Clipboard, а потім вставка.

**setClipboardContents(longText);**

**textarea.sendKeys(Keys.CONTROL + "v");**

**Нативні та синтезовані події**

|  |  |
| --- | --- |
| Нативні | Синтезовані |
| На рівні ОС | В середині браузера |
| Реалізація на C/C++ | Реалізація на JS |
| Точніше емулюють | Не завжди точно емулюють |
| Інколи потребують фокус | Працюють в бекграунді |
| Не всі версії браузерів | Всі версії браузера |

windows – нативні події.

linux – синтетичні події.

Для їх ввімкнення та вимкнення використовується механізм:

**DesiredCapabilites caps = new DesiredCapabilites();**

**caps.setCapability(HAS\_NATIVE\_EVENTS, true);**

**WebDriver driver = new FirefoxDriver(caps);**

**Складні події. Actions.**

* click - клік
* sendKeys – ввести текст
* moveToElement – навести мишку на елемент
* clickAndHold – натиснути клавішу миші і не відпускати
* release – відпусти клавішу миші
* keyDown- натиснути клавішу
* keyUp – відпустити клавішу

Приклад дії.

new Actions(driver) – створюється новий елемент  
.moveToElement(drag) – перетягнути миш на такий елемент  
.keyDown(Keys.CONTROL) – зажати кнопку CONTROL  
.clickAndHold() – клікнути та тримати  
.moveToElement(drop) – не відпускаючи кнопки перетягти на інший елемент  
.release() – відпустити клавішу миші   
.keyUp(Keys.CONTROL) – відпустити клавішу CONTROL  
.perform(); - обов’язково. Виконання всієї серії зараз.

new Actions(driver)  
.moveToElement(el, 1, 1) – навести у визначену точку (верхній лівий кут)  
.click()   
.perform();

new Actions(driver)  
.moveToElement(el) – миш наводиться на центр елемента   
.moveByOffset(5,5) - зміщається трішки вправо на 5 пх і вниз на 5пх  
.click()   
.perform();

new Actions(driver)  
.moveToElement(menu) – навести на меню  
.moveToElement(submenu) – навести на підменю  
 .moveToElement(item) - навести на елемент  
.click()   
.perform();

new Actions(driver)  
.dragAndDrop(el1, el2) – створений метод для реалізації першого прикладу  
.perform();

**Отримання властивостей елементів. Методи.**

**getText** – отримати текст елемента.

* Видимий текст. Невидимий елемент має пустий текст.
* Нормалізація – видалення пробілів. Preformatted – зберігає пробіли.

**getAttribute** – отримання атрибутів.

* input.getAttribute(“value”); - отримання тектсу, який введено в поле input, оскільки це не просто текст, а значення атрибута value.
* textarea.getAttribute(“value”) - для поля <textarea>ТЕКСТ</textarea> - getAttribute повертає значення яке зараз наявне в полі. А метод getText – повертає значення ТЕКСТ, тобто те що було в полі до маніпуляцій.

**input.getAttribute("href");** - завжди в абсолютні посилання.

**button.getAttribute("disabled")** либо null, либо true disabled, selected, checked, readonly, …

**div.getAttribute("innerText");** - attribute или **property**? - повертає значення **property**.

**isDisplayed**

* видимий елемент
* якщо елемент знаходиться за лівим або верхнім краєм частково знаходиться за краєм (хоча елемент не видимий).
* скритий під іншим елементом, частково скритий під іншим елементом
* прозорий, або колір зливається з фоном
* **element.getCssValue("background-color"); -** знайти елемент за стилем CSS
* **element.getSize(); -** за розміром елемента
* **element.getLocation(); -** за розташуванням
* **element.getTagName();** - за тегом
* **element.isEnabled(); -** коли element.getAttribut(“Disabled=null”)
* **element.isSelected();** - коли element.getAttribut(“Selected=true”)

**Переключкення між вікнами:**

**driver.switchTo()**

* driver.switchTo().alert();
* driver.switchTo().frame();
* driver.switchTo().defaultContent();
* driver.switchTo().window();

**Діалогові вікна**

Alert alert = driver.switchTo().alert();

alert.getText();

alert.accept(); // либо alert.dismiss();

Якщо нема алерта - NoAlertPresentException

Якщо ми не очікуємо алерта, а він з’являється, то виникає помилка UnhandledAlertException. Дану помилку Selenium обробляє: спочатку закриває алерт; тест, в якому виникла помилка, припиняється(фейлиться) але браузер не висне, а розпочинається наступний тест.

**Фрейми**

driver.switchTo().frame(  
driver.findElement(By.tagName("iframe")));

// что-то сделали внутри фрейма

driver.switchTo().defaultContent();

Фрейми можуть бути вкладеними. Ми спочатку шукаємо елемент фрейма, а потім переключаємось у нього. driver.switchTo().defaultContent(); - повернення на самий верх ієрархії.

**Вікна**

allWindows = driver.getWindowHandles(); - повертає ідентифікатори всіх вікон, відкритих цим браузером

thisWindow = driver.getWindowHandle(); - повертає ідентифікатор поточного вікна, яке відкрите браузером

driver.switchTo().window(handle);

// что-то делаем в этом окне

driver.close(); - закриває вікно(натискає червоний хрестик), якщо воно саме не закривається після натискання якоїсь кнопки.

driver.switchTo().window(originalWindow); - перед тим як переключитися на нове вікно необхідно знати його ідентифікатор Handle.

Переключення між вікнами відбувається тільки по Handle – унікальне ім’я. яке присвоюється самим браузером чи ОС.

Послідовність:

// запоминаем идентификатор текущего окна

originalWindow = driver.getWindowHandle()

// запоминаем идентификаторы уже открытых окон

existingWindows = driver.getWindowHandles()

// кликаем кнопку, которая открывает новое окно

driver.findElement(By.id("button")).click()

// ждем появления нового окна, с новым идентификатором

newWindow = wait.until(anyWindowOtherThan(existingWindows))

// переключаемся в новое окно

driver.switchTo().window(newWindow)

// закрываем его

driver.close()

// и возвращаемся в исходное окно

driver.switchTo().window(originalWindow)

**Реалізація метода очікування нового вікна:**

public ExpectedCondition<String> anyWindowOtherThan(  
Set<String> oldWindows)

{

return new ExpectedCondition<String>() {

public String apply(WebDriver driver) {

Set<String> handles = driver.getWindowHandles();

handles.removeAll(oldWindows);

return handles.size() > 0 ? handles.iterator().next() : null;

}

};

}

**Розміри і розташування вікна:**

driver.window().getSize();

driver.window().setSize(new Dimension(800, 600));

driver.window().maximize(); - для розгортання вікна на весь екран

driver.window().getPosition();

driver.window().setPosition(new Point(0, 0));

Це потрібно для зняття скріншота, уникнення скролінга