

Selenium Webdriver

Pré-requis du module

- Savoir trouver un objet dans une page Web avec XPath
- Connaître le langage JAVA
- Connaître les concepts de la POO et leur implémentation en JAVA
 - Système de type (notion de classe ; notion d'objet)
 - Encapsulation (notion de visibilité ; notion d'interface)
 - Héritage (notion de hiérarchie de classes)
 - Polymorphisme



Sommaire

1. Qu'est-ce que Selenium Webdriver ?
2. Interactions basiques
3. Synchronisation
4. Modèle Page/objects
5. Checkboxes, bouton radios et menus déroulants
6. Tableaux
7. Interactions complexes (Drag&Drop, Mouse Hover)



Sommaire

- 8. Changement de pages, frames et alertes
- 9. Dates et chaînes de caractères
- 10. Vérifications en Base de Données
- 11. Inclure un script Auto IT (?)
- 12. Inclure du JavaScript (?)
- 13. Événements (?)
- 14. Reporting, temps de réponses et copies d'écrans (?)



QU'EST-CE QUE SELENIUM WEBDRIVER?



Le Framework Selenium






QU'EST-CE QUE SELENIUM WEBDRIVER ?

- Selenium est un framework d'automatisation de tests d'applications web
- C'est un projet Open Source (sous licence Apache 2.0)
- Le projet Selenium se compose de plusieurs briques dont certaines sont aujourd'hui dépréciées.

Selenium Core	Basé sur JavaScript et un langage de commandes en HTML (appelé Selenese)
Selenium RC ou Selenium 1	Basé sur Selenium Core, il permet d'écrire les tests dans d'autres langages tels que C#, Java, PHP Python, Perl et Ruby
Selenium WebDriver	Successeur de Selenium RC. N'est pas basé sur JavaScript mais sur le support natif de chaque navigateur pour l'automatisation.
Selenium IDE	Extension Firefox permettant d'enregistrer des tests
Selenium Builder	Autre extension Firefox plus évoluée
Selenium Server	Serveur d'exécution Selenium
Selenium Grid	Grille d'exécution Selenium

Selenium Webdriver

QU'EST-CE QUE SELENIUM WEBDRIVER ?

- Selenium WebDriver fournit une API unifiée ainsi que des implémentations pour un ensemble de navigateurs
 -  Firefox
 -  Internet Explorer
 -  Safari
 -  Opera
 -  Chrome
- Des drivers pour d'autres navigateurs sont également disponibles mais non maintenus par le projet Selenium.
- L'API est disponible dans plusieurs langages. On présente ici sa version Java. On exécute les tests Selenium Webdriver via JUnit.

Selenium Webdriver

UTILISER SELENIUM WEBDRIVER DANS UN PROJET MAVEN

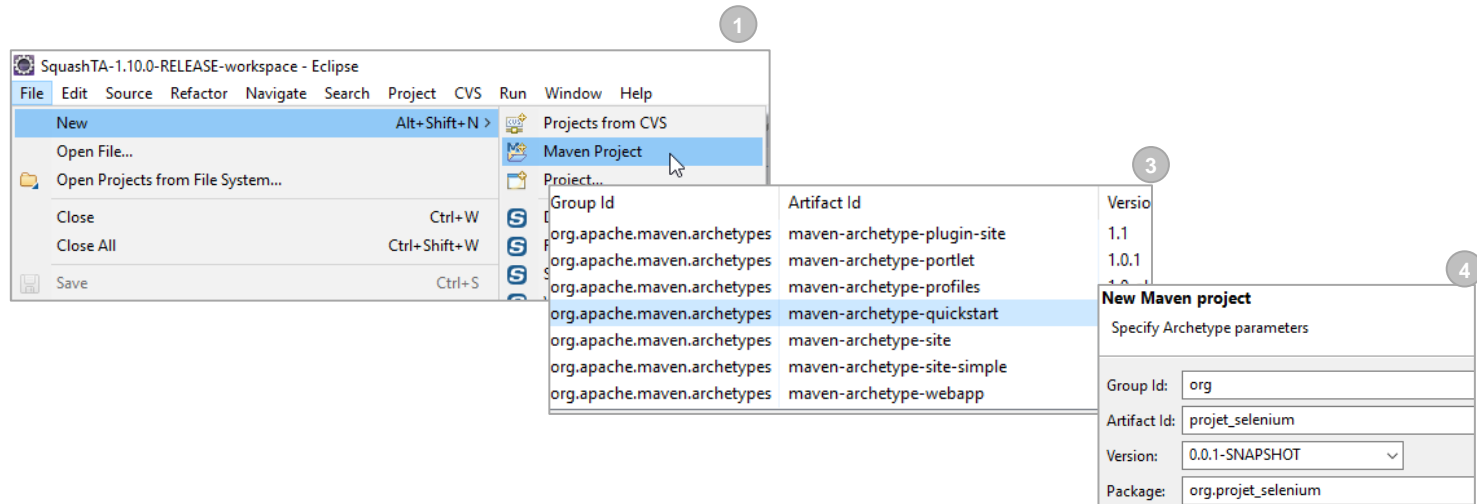
Dans Eclipse, créer un nouveau projet Maven

Dans le menu **File**, choisir **New > Maven Project**

Sélectionner « Use default Workspace location » et cliquez sur **Next**.

La fenêtre suivante affiche les différents archétypes de projets préconfigurés disponibles. Ces archétypes sont organisés par groupe (group id) et identifiés par uid (artifact id). Choisissez l'archetype suivant : **Group Id** = org.apache.maven.archetypes / **Artifact Id** = maven-archetype-quickstart

Spécifiez un groupe id pour votre projet (ex : **org**) et un Artefact ID qui sera le nom de votre projet (ex : **projet_selenium**).



Selenium Webdriver

CONFIGURER LE PROJECT OBJECT MODEL (POM.XML) POUR SELENIO WEBDRIVER

- Pour pouvoir utiliser les classes du framework Selenium WebDriver, nous devons obligatoirement déclarer deux dépendances dans le pom.xml :
 - JUnit
 - Selenium
- Sauvegardez les changements (ctrl + S) ou rafraîchir la configuration du projet (clic droit > Maven > Update Project).
- Le projet Maven télécharge ainsi les dépendances (fichiers .jar) nécessaires au projet.

Ainsi, nos tests automatisés d'IHM web écrits avec Selenium WebDriver seront structurés à partir du framework de test unitaire Java, JUnit

```
<dependencies>
```

```
<dependency>
```

```
<groupId>junit</groupId>
```

```
<artifactId>junit</artifactId>
```

```
<version>4.12</version>
```

```
</dependency>
```

```
<dependency>
```

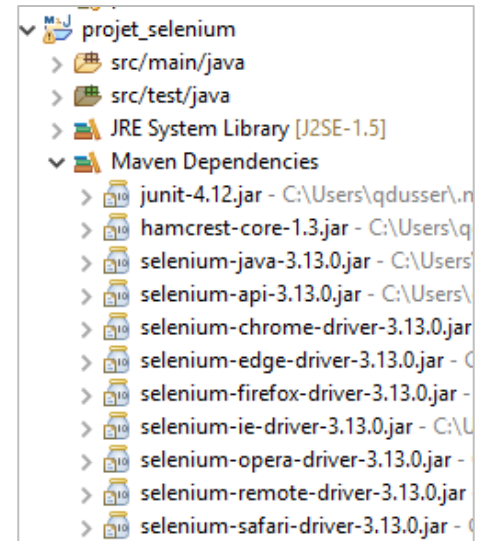
```
<groupId>org.seleniumhq.selenium</groupId>
```

```
<artifactId>selenium-java</artifactId>
```

```
<version>3.13.0</version>
```

```
</dependency>
```

```
</dependencies>
```



Rappel sur JUnit

QU'EST-CE QUE JUNIT ?

- **JUnit** est un framework de tests unitaires standard en Java.
- *JUnit* contient en plus des instructions Java habituelles des instructions de validation appelées **assertions**.
- La version la plus récente de JUnit est JUnit 5... nous utiliserons **JUnit 4**, notamment caractérisé par la présence d'**annotations**.

STRUCTURE D'UNE CLASSE JUNIT

```
package mon.package.test;

import org.junit.Test;
import org.junit.Assert.*;

public class MaClasseDeTest {

    @Test
    public void fonctionnaliteTest() {
        // implémentation du test
        String string1 = "Hello" + " " + "World« ;
        assertEquals("Hello World" , string1);
    }
}
```

@Test **@Annotations JUnit**

assertEquals **assertions JUnit**

```
package mon.package.test;

import org.junit.Before;
import org.junit.After;
import org.junit.Test;

public class MaClasseDeTest {

    @Before
    public void setUp() {
        // initialisation : exécuté avant chaque test
    }

    @After
    public void tearDown() {
        // Nettoyage : code exécuté après chaque test
    }

    @Test
    public void fonctionnaliteTest() {
        // implémentation du test
    }
}
```

INTERACTIONS BASIQUES



Selenium WebDriver : Interactions basiques

APPROCHE DE L'AUTOMATISATION AVEC SELENIUM WEBDRIVER

La démarche de l'automatisation suit toujours le même processus pas à pas :



1. Vérifier que l'ensemble des prérequis sont satisfaits et que les éléments sont dans l'état attendu (visibles, actifs...).
2. Réaliser des actions sur ces éléments.
3. Vérifier que le résultat des actions sur ces éléments correspondent au résultat attendu.



Selenium WebDriver : Interactions basiques

L'INTERFACE WEBDRIVER

- **WebDriver** est l'interface par laquelle Selenium va pouvoir interagir avec les différents navigateurs (envoyer des commandes)
- L'interface **WebDriver** présente les méthodes que tout driver Selenium WebDriver doit implémenter.
- Cette interface est implémentée par différentes classes tels que:
 - FirefoxDriver
 - ChromeDriver
 - InternetExplorerDriver



Selenium WebDriver : Interactions basiques

PREMIERS PAS : INSTANCIER UN DRIVER POUR LE NAVIGATEUR DE TEST

WebDriver *driver* = new ???????;

- Permet d'instancier le driver à utiliser selon le navigateur cible.
- Pour que WebDriver puisse interagir avec le navigateur cible, l'utilisateur doit télécharger et spécifier l'emplacement du driver adéquat
- La variable **driver** peut ainsi appeler toutes les méthode implémentées par la classe WebDriver.

➤ Chrome

```
// préciser le chemin du driver (ici relatif à la racine du projet)
System.setProperty("webdriver.chrome.driver", "src/main/resources/driver/chromedriver.exe");
// Initialisation du navigateur Chrome
WebDriver driver = new ChromeDriver();
```

➤ Firefox

```
System.setProperty("webdriver.gecko.driver", "src/main/resources/driver/geckodriver.exe");
WebDriver driver = new FirefoxDriver();
```

➤ Edge

```
System.setProperty("webdriver.edge.driver", "src/main/resources/driver/msedgedriver.exe");
WebDriver driver = new EdgeDriver();
```

➤ Internet Explorer

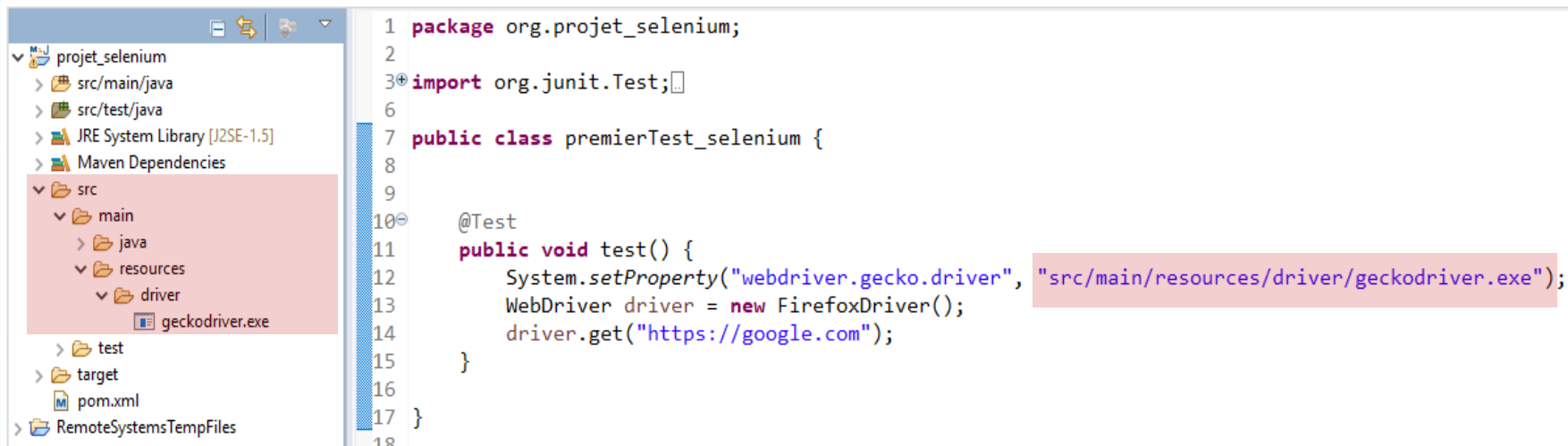
```
System.setProperty("webdriver.ie.driver", "src/main/resources/driver/IEDriverServer.exe");
WebDriver driver = new InternetExplorerDriver();
```

Autre solution : enregistrer les driver dans le PATH system

Selenium WebDriver : Interactions basiques

PREMIERS PAS : INSTANCIER UN DRIVER POUR LE NAVIGATEUR DE TEST

Exemple d'instanciation du driver pour lancement d'un test sur Firefox



The screenshot shows an IDE with a project named 'projet_selenium'. The project structure on the left includes 'src/main/java', 'src/test/java', 'JRE System Library [J2SE-1.5]', 'Maven Dependencies', 'src' (expanded), 'main' (expanded), 'java' (expanded), 'resources' (expanded), 'driver' (expanded), and 'geckodriver.exe'. The code editor on the right shows the following Java code:

```
1 package org.projet_selenium;
2
3 import org.junit.Test;
4
5
6
7 public class premierTest_selenium {
8
9
10 @Test
11 public void test() {
12     System.setProperty("webdriver.gecko.driver", "src/main/resources/driver/geckodriver.exe");
13     WebDriver driver = new FirefoxDriver();
14     driver.get("https://google.com");
15 }
16
17 }
18
```

À partir de la variable **driver**, nous pouvons désormais appeler les méthodes de la classe **WebDriver** (ex : **get()**)

```
driver.get("https://google.com");
```

Les méthodes WebDriver

QUELLES SONT LES MÉTHODES DE LA CLASSE WEBDRIVER ?

Parmi les grandes familles de méthodes de la classe WebDriver figurent :

- Les actions sur les fenêtres du navigateur
- La navigation
- La récupération d'information globales sur les pages
- La récupération d'objets dans le DOM



Selenium WebDriver : Interactions basiques

PREMIERS PAS : INTERAGIR AVEC LES FENÊTRES ET LES PAGES

Les méthodes de navigation de l'interface WebDriver

- **get(String url) : void**
Permet de naviguer vers l'url indiquée
- **getTitle() : String**
Renvoie le titre de la page affichée
- **getCurrentUrl() : String**
Renvoie l'URL de la page affichée
- **getPageSource() : String**
Renvoie le code source de la page affichée
- **close() : void**
Ferme la fenêtre active contrôlée par le WebDriver
- **quit() : void**
Ferme toutes les fenêtres contrôlées par le WebDriver
- **navigate() : Navigation**
*Accès à l'interface navigation et aux méthodes **to()**, **forward()**, **back()**, **refresh()**... void*



Selenium WebDriver : Interactions basiques



EXERCICE : Premiers pas avec Selenium



1. À partir des méthodes de l'interface WebDriver naviguez sur l'url suivante : « <https://latavernedutesteur.fr> »
2. Récupérez le titre de la page et utilisez une assertion pour vérifier que le titre de la page est bien celui attendu (à savoir : « [La](#) [taverne](#) [du](#) [testeur](#) »)
3. Fermez le navigateur

Selenium WebDriver : Interactions basiques

INTERAGIR AVEC DES OBJET D'UNE PAGE : MAPPER LES ÉLÉMENTS D'UNE PAGE

L'interface WebDriver offre deux méthodes pour mapper les éléments d'une page web :

- `findElement(By by) : WebElement`
- `findElements(By by) : List<WebElement>`

➤ La méthode « `findElement` »

```
WebElement element = driver.findElement(By.className("formation"));
```

- Prend un objet « By » en paramètre qui localise l'élément web.
- Retourne un objet « WebElement » représentant le 1^{er} élément web mappé par l'objet « By » passé en paramètre.
- Si le driver ne trouve pas d'élément web correspondant à cet objet « By », une exception « NoSuchElementException » est levée.

➤ La méthode « `findElements` »

```
List<WebElement> elements = driver.findElements(By.xpath("//.[@id='formation']"));
```

- Retourne une liste d'objets « WebElement ».
- Si aucun élément ne correspond à l'objet « By » passé en paramètre, la liste retournée est vide.

Selenium WebDriver : Interactions basiques

LOCALISER LES ÉLÉMENTS WEB PAR LEURS ATTRIBUTS HTML

➤ HTML : `<input id="user" name="login" class="content">`

- Localisation par **id**
`WebElement element = driver.findElement(By.id("user"));`
- Localisation par **className**
`WebElement element = driver.findElement(By.className("content"));`
- Localisation par **name**
`WebElement element = driver.findElement(By.name("login"));`

➤ HTML : `<h1>Squash</h1>`

- Localisation par nom de balise **tagName**
`WebElement element = driver.findElement(By.tagName("h1"));`

➤ HTML : `Déconnexion : Florence`

- Localisation par **texte visible**
`WebElement element = driver.findElement(By.linkText("Déconnexion : Florence"));`
`WebElement element = driver.findElement(By.partialLinkText("Déconnexion"));`

Selenium WebDriver : Interactions basiques

LOCALISER LES ÉLÉMENTS WEB PAR LEURS ATTRIBUTS HTML

➤ HTML :

```
<div id="reptiles">
  <span class="product">Iguana</span>
  <span class="product soldOut">Rattlesnake</span>
  <span class="product">Crocodile</span>
</div>
```

- Localisation par **Xpath**

```
WebElement element = driver.findElement(By.xpath("//div[@id='reptiles']/span[.='Iguana']"));
```

• QU'EST-CE QUE LE LANGAGE XPATH ? RAPPEL :

- Le langage Xpath est employé pour identifier un ensemble de nœuds dans les document XML
- Nous l'utiliserons pour localiser les éléments web dans l'arborescence HTML

Selenium WebDriver : Interactions basiques

SELENIUM : INTERAGIR AVEC LES OBJETS (INTERFACE WEBELEMENT)

- **click()** : void
Permet de cliquer sur un élément
- **clear()** : void
Permet de vider le contenu d'un champ
- **sendKeys()** : void
Permet de saisir du texte dans un champ
- **submit()** : void
Permet de soumettre un formulaire
- **getText()** : String
Retourne l'innerText de l'élément
- **getAttribute(String)** : String
Retourne la valeur d'un attribut HTML étant donné son nom
- **Is Displayed()** : boolean
Retourne true si l'élément est affiché sur la page
- **isSelected()** : boolean
Retourne true si l'élément est sélectionné (bouton radio / checkbox)
- **isEnabled()** : boolean
Retourne true si l'élément est actif sur la page

Selenium WebDriver : Interactions basiques

SELENIUM : INTERAGIR AVEC LES OBJETS (INTERFACE WEBELEMENT)

Quelques exemples...

- Effacer un champ texte prérempli
`driver.findElement(By.id("input_r54")).clear();`
- Ecrire dans un champ texte
`driver.findElement(By.id("input_r54")).sendKeys("Hello world !");`
- Récupérer la valeur texte d'un titre et l'assigne à la variable 'titre'
`String titre = driver.findElement(By.tagName("h1")).getText();`
- Cliquer sur un élément web
`driver.findElement(By.id("button_e1")).click();`
- Récupérer la valeur l'attribut 'type' d'un élément
`driver.findElement(By.id("tutu")).getAttribute("type");`
- Vérifier qu'un élément web est sélectionné (checkbox, radio)
`boolean married = driver.findElement(By.id("checkbox01")).isSelected();`
- Vérifie qu'un élément web est affiché à l'écran
`driver.findElement(By.XPath("//input[contains(.,'submit')]")).isDisplayed();`



Selenium WebDriver : Interactions basiques



EXERCICE : Automatiser un cas de test fonctionnel avec Selenium



1. Ouvrir l'application JpetStore et automatisez le cas de test suivant :

Actions	Résultat attendu
Accéder à l'application Jpetstore et se connecter en tant que j2ee/j2ee	L'utilisateur « ABC » est bien connecté (apparition d'un message de bienvenu et du lien « Sign out »)
Sélectionner la catégorie Fish	Affichage de la liste des produit disponible pour la catégorie Fish
Sélectionner le produit de votre choix	
Ajouter un item au panier (« add to cart »)	Affichage du panier
Passer la quantité commandée à 2 et cliquer sur « update cart »	Le prix total est égal au double du prix à l'unité

TIPS : Les méthodes à utiliser

WebDriver

```
.get();  
.findElement(By.xpath(« »)); : WebElement
```

WebElement

```
.click();  
.clear();  
.sendKeys(« »);
```

Assert JUnit

```
assertEquals(WebElement.getText(), « »);
```


SYNCHRONISATION



La synchronisation

- GÉRER LA SYNCHRONISATION AVEC LES CLASSES WAIT

Trois types d'actions « attendre / wait » avec Selenium Webdriver :

- Implicit Wait



10'



- Instruction 1
- Instruction 2
- Instruction 3
- Instruction 4
- ...

GLOBAL

- Explicit Wait

instruction



condition

- Element.isDisplayed
- Element.isSelected
- Element.isEnabled
- ...

LOCAL

- Fluent Wait

Try

instruction

catch

exception



10'

LOCAL

GÉRER LA SYNCHRONISATION AVEC LES CLASSES WAIT – IMPLICIT WAIT

➤ Implicit wait (contexte global)

L'*implicit wait* définit un timeout global sur toutes les méthodes WebDriver d'une classe. Il permet pour chaque action du driver :

- de jouer l'instruction autant de fois qu'il sera nécessaire à son succès, dans la limite du timeout défini (pooling par défaut : 250 millisecondes)
- de lever une exception en cas de d'échec de l'instruction au terme du timeout (**NoSuchElement**)

➤ Exemple

```
WebDriver driver = new FirefoxDriver();  
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);  
driver.findElement (By.id("button_e1")); ⌚ 10'
```

Explicit Wait

GÉRER LA SYNCHRONISATION AVEC LES CLASSES WAIT – EXPLICIT WAIT

➤ Explicit wait (contexte restreint)

Permet d'attendre qu'une condition particulière soit remplie avant d'exécuter l'instruction

Exemples de cas d'utilisation

- La validation d'un formulaire qui prend un délai variable en fonction des informations à traiter pour charger la page suivante.
- Globalement, l'ensemble des pages qui traitent un certain volume de données – dont les formulaires – peuvent entraîner une désynchronisation.
- Certains éléments sont cachés sur la page et attendent qu'une condition spécifique soit remplie pour s'afficher.

➤ Exemple

Instanciation de l'explicit wait

```
WebDriverWait wait = new WebDriverWait(driver, 10);
```

Ex : attendre qu'un élément soit cliquable

```
WebElement webElement = wait.until(ExpectedConditions.elementToBeClickable(By.id("btnLogin")));
```

Ex : attendre qu'un élément soit visible

```
boolean visible = wait.until(ExpectedConditions.visibilityOf(webElementBouton));
```

Explicit Wait

❑ Différentes conditions portant sur :

- La présence d'une alerte
- La présence d'un attribut sur un élément
- La sélection d'un élément
- La présence d'un élément cliquable
- La présence d'une frame disponible
- La visibilité d'un élément
- Le texte d'un élément
- La présence ou absence de certains éléments
- L'URL ou le titre courant de la page

<https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/support/ui/ExpectedConditions.html>

❑ Possibilité de combiner logiquement les conditions

- *and(ExpectedCondition<?> conditions) : Boolean*
- *or(ExpectedCondition<?> conditions) : Boolean*
- *not(ExpectedCondition<?> conditions) : Boolean*

Fluent Wait

GÉRER LA SYNCHRONISATION AVEC LES CLASSES WAIT – FLUENT WAIT

➤ Fluent wait (contexte restreint)

Conceptuellement proche de l'Implicit Wait, **MAIS** :

- S'applique sur un contexte restreint (n'est pas une configuration globale)
- Permet de contrôler le polling et d'ignorer des exceptions

➤ Exemple

// Waiting 30 seconds for an element to be present on the page, checking for its presence once every 5 seconds.

```
Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
```

- .withTimeout(Duration.ofSeconds(30))
- .pollingEvery(Duration.ofMillis(250))
- .ignoring(NoSuchElementException.class);

// affectation de la variable foo à partir d'une recherche avec Fluent wait

```
WebElement foo = wait.until(new Function<WebDriver, WebElement>()  
{  
    public WebElement apply(WebDriver driver) {  
        return driver.findElement(By.id("foo"));  
    }  
});
```

// action sur l'élément foo

```
foo.sendKeys("Une chaîne de caractère");
```

TP PetStore – Synchronisation



EXERCICE : Mettre en place une synchronisation des instructions de Selenium WebDriver

Créer un test de connexion à l'application suivante :

<https://katalon-demo-cura.herokuapp.com/> (login = John Doe ; pwd = ThisIsNotAPassword)

(une classe par méthode : Implicit / Explicit / Fluent **wait**)



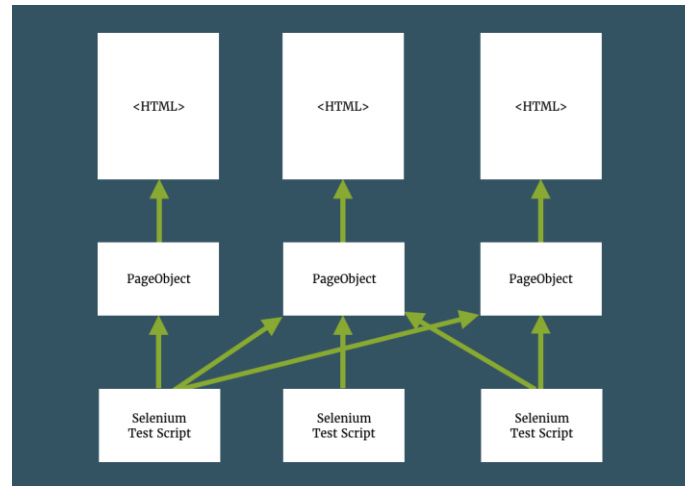
MODÉLISER LES PAGES AVEC PAGE/OBJECT



Le pattern Page Object

• PRINCIPLE

Modéliser chaque écran (ou partie d'écran) sous forme d'une classe Java. Comme toute classe java, la Page Object se compose d'attributs (WebElement) et de méthodes (actions réalisables sur l'écran) que l'on peut appeler dans les scripts WebDriver. L'idée est de **reproduire une logique d'utilisation - de cheminement - au sein du script de test automatisé**



➤ Les avantages

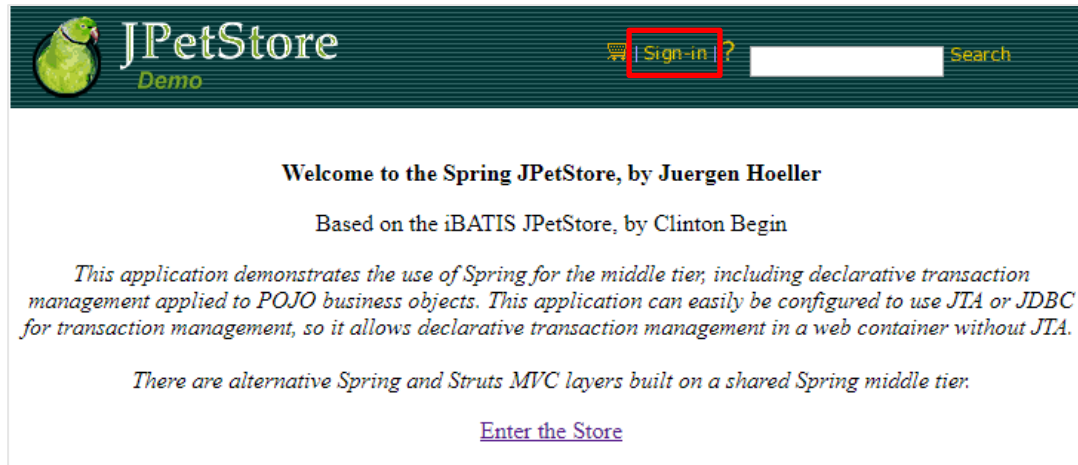
- **Mutualiser le code** pour faciliter la maintenance (ex : si le xpath d'un élément change → à changer uniquement dans le page object)
- **Simuler les règles de gestion** : sur un écran donné, seules les actions et vérifications possibles sur cet écran sont disponibles
- **Définir la navigation** d'une page à une autre (ex: après le login, la page d'accueil est affichée)
- **Séparer l'implémentation du test** :
 - Le code technique dans les pages objets
 - Le déroulement des pas de test et les données dans les scripts de test
- **Possibilité d'utiliser l'héritage** ou la composition pour les zones présentes sur plusieurs écrans (ex : menu)

➤ Remarque

Ce principe peut s'appliquer avec n'importe quel automate, dès lors que les scripts sont codés dans un **langage objet**

Le pattern Page Object

PAGE OBJECT – LES CLASSE OBJET (ECRAN)



```
public class PageIndex {
```

```
    @FindBy (xpath="//img[@name='img_signin']")  
    WebElement bouton_signin;
```


```
    public PageLogin clicSignInButton(WebDriver driver) {  
        bouton_signin.click();  
        return PageFactory.initElements(driver, PageLogin.class);  
    }  
}
```

Avec le pattern Page Objet on préférera rechercher les WebElement avec l'annotation **@FindBy** plutôt qu'avec la méthode **findElement** du WebDriver → cela permet de ne pas avoir à déclarer de WebDriver dans les classes objet.
(Le WebDriver sera déclaré dans la classe de test)

La méthode de clic sur le bouton Sign-in initialise la page suivante (**PageLogin**). Pour ce faire, on déclare un type de retour **PageLogin** et on utilise la méthode :
PageFactory.initElements(driver, class)

Le pattern Page Object

PAGE OBJECT – LES CLASSE OBJET (ECRAN)



JPetStore Demo

[Fish](#) | [Dogs](#) | [Reptiles](#) | [Cats](#) | [Birds](#)

Please enter your username and password.

Username: 1

Password: 2

3

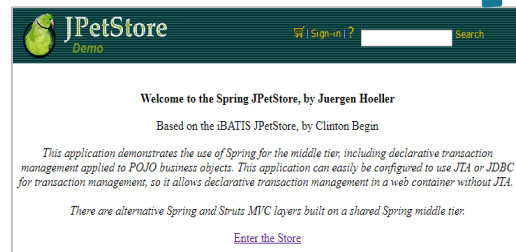
```
public class PageLogin {  
  
    1 @FindBy(xpath = "//input[@name='username']")  
      private WebElement username_field;  
  
    2 @FindBy(xpath = "//input[@name='password']")  
      private WebElement password_field;  
  
    3 @FindBy(xpath = "//input[@name='update']")  
      private WebElement submit_button;  
  
    public PageAccueil logIn(WebDriver driver, String username, String password) {  
        username_field.clear();  
        username_field.sendKeys(username);  
        password_field.clear();  
        password_field.sendKeys(password);  
        submit_button.click();  
        return PageFactory.initElements(driver, PageAccueil.class);  
    }  
}
```

Le pattern Page Object

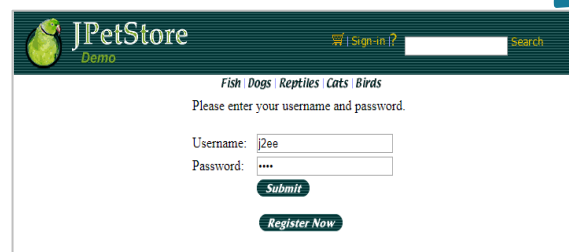
PAGE OBJECT – LA CLASSE DE TEST

```
public class TestPageObject {  
  
    @Test  
    public void test() {  
  
        // instantiation du driver  
        System.setProperty("webdriver.gecko.driver", "src/main/resources/driver/geckodriver.exe");  
        WebDriver driver = new FirefoxDriver();  
  
        // accès à l'url de l'application  
        driver.get("http://localhost:8090/jpetstore-1.0.5-env2");  
  
        // instantiation de la pageIndex  
        PageIndex page_index = PageFactory.initElements(driver, PageIndex.class);  
  
        // appel de la méthode "clicSignInButton" --> instantiation de la page de connexion  
        PageLogin page_login = page_index.clicSignInButton(driver);  
  
        // appel de la méthode "logIn" --> instantiation de la page d'accueil  
        PageAccueil page_accueil = page_login.logIn(driver, "j2ee", "j2ee");  
  
        ...  
    }  
}
```

PageIndex



PageLogin



PageAccueil



Gérer les héritages en Page Object Pattern

PAGE OBJECT – LES HÉRITAGES

Si des éléments (et donc des méthodes) sont accessibles depuis plusieurs écrans de l'application, il est possible de les regrouper au sein d'une classe abstraite de laquelle hériteront les pages concernées

➤ Exemple

```
public abstract class BandeauMenu {  
  
    @FindBy(xpath = "//input[@name='search']")  
    private WebElement search_field;  
  
}
```

```
public class PageIndex extends BandeauMenu {  
    ...  
}
```

```
public class PageLogin extends BandeauMenu {  
    ...  
}
```

```
public class PageAccueil extends BandeauMenu {  
    ...  
}
```

Le pattern Page Object



EXERCICE : Créer les « Page Objects » correspondants – 30 min



1. Appliquez le principe Page/Object à l'application JpetStore.
2. Réécrivez votre test en utilisant le parcours de pages et en appelant les éléments et les méthodes propres à ces pages

CHECKBOXES, BOUTONS RADIO ET MENUS DÉROULANTS



Checkboxes

➤ CODE HTML

```
<label>
  <input type="checkbox" id="cbox1" value="checkbox1"/>
  Voici la première case.
</label>
<br/>
<label>
  <input type="checkbox" id="cbox2" value="checkbox2"/>
  Voici la seconde case.
</label>
```

➤ RENDU HTML

☐ Voici la première case.

☐ Voici la seconde case.

Boutons radio

➤ CODE HTML

```
<input type="radio" name="fruit" value="pomme"/> Pomme <br/>  
<input type="radio" name="fruit" value="poire"/> Poire <br/>  
<input type="radio" name="fruit" value="banane"/> Banane <br/>
```

➤ RENDU HTML



☒ Pomme
☐ Poire
☐ Banane

Checkboxes et boutons radio

➤ INTÉRAGIR AVEC LES CHECKBOXES ET BOUTONS RADIO

- Les checkboxes et boutons radio peuvent être manipulées sur deux formes avec Selenium Webdriver :

En tant que WebElement unitaire

```
WebElement checkbox = driver.findElement(By.id("myid"));
checkbox.click();
```

En tant que collection (List) d'options

```
List<WebElement> checkboxOptions = driver.findElements(By.xpath("/../.."));
checkboxOptions.get(0).isSelected();
checkboxOptions.get(1).getAttribute("value");
checkboxOptions.get(1).click();
```

- Il est également possible de sélectionner directement l'option en Xpath pour interagir avec.

Menus déroulants

➤ CODE HTML

```
<select name="cars">
  <option value="volvo"> Volvo </option>
  <option value="saab"> Saab </option>
  <option value="fiat"> Fiat </option>
  <option value="audi"> Audi </option>
</select>
```

➤ RENDU HTML



Volvo ▼

Volvo

Saab

Fiat

Audi

Manipuler les menus déroulants

➤ INSTANCER UN SELECT

Selenium Webdriver fournit une classe d'aide pour interagir avec les menus déroulants : **Select**
Cette classe prend un WebElement comme paramètre de construction.

```
WebElement menu = driver.findElement(By.id("categorie"));
Select select = new Select(menu);
```

➤ MANIPULER UN SELECT

- **isMultiple(): boolean**
Retourne vrai sur le menu autorise une sélection multiple
- **deselectAll(): void**
Désélectionne tous les options
- **selectByIndex(int arg0) : void // deselectByIndex(int arg0) : void**
Sélectionne // Désélectionne l'option à la position passée en paramètre
- **selectByValue(String arg0) : void // deselectByValue(String arg0) : void**
Sélectionne // Désélectionne l'option dont l'attribut « value » est passé en paramètre
- **selectByVisibleText(String arg0) : void // deselectByVisibleText(String arg0) : void**
Sélectionne // Désélectionne l'option dont le texte visible est passé en paramètre
- **getOptions(): List<WebElement>**
Récupère la liste des options en tant que WebElements.
- **getAllSelectedOptions(): List<WebElement>**
Récupère la liste des options sélectionnées en tant que WebElements.
- **getFirstSelectedOption(): WebElement**
Récupère la première option sélectionnée en tant que WebElements

TP PetStore – checkbox et menu déroulant



EXERCICE : Gérer les checkboxes et les menus déroulants

1. Automatisez ce test :

Actions	Résultat attendu
Accéder à l'application Jpetstore et se connecter en tant que j2ee/j2ee	L'utilisateur « ABC » est bien connecté (apparition d'un message de bienvenu et du lien « Sign out »)
Accéder à la page MyAccount	Affichage de la page de préférence de compte
Sélectionner « japanese » comme langage de préférence et les « Reptiles » comme animal favori	Les options s'affichent dans les champs correspondant
Vérifier que les checkboxes « Enable My list » et « Enable My Banner » sont bien sélectionnées par défaut	
Désélectionner « Enable My list »	La checkbox est désélectionnée



TABLEAUX



Tableaux HTML

➤ CODE HTML

```
<table>
  <tbody>
    <tr>
      <th>NOM</th>
      <th>PRENOM</th>
      <th>DATE DE NAISSANCE</th>

    </tr>
    <tr>
      <td>Dupont</td>
      <td>Hervé</td>
      <td>20 déc. 1954</td>
    </tr>
    <tr>
      <td>Durant</td>
      <td>Danielle</td>
      <td>24 avr. 1982</td>
    </tr>
  </tbody>
</table>
```

➤ RENDU HTML

NOM	PRENOM	DATE DE NAISSANCE
Dupont	Hervé	20 déc. 1954
Durant	Danielle	24 avr. 1982

Tableaux HTML – Numéro de ligne

➤ MANIPULER UNE CELLULE

Implémentation

Cette méthode retourne une cellule comme WebElement

```
public WebElement getCellule(row, col){  
    WebElement element = driver.findElement(By.xpath("//table/tbody/tr["+row+"]/td["+col+"]"));  
    return element;  
}
```

Exemple d'utilisation

Ici, la cellule concernée est celle de la ligne 2, colonne 2

```
WebElement cellule = getCellule(2, 2);  
cellule.getText();  
cellule.click();
```


Tableaux HMTL – Tableaux dynamiques

➤ GÉRER LES TABLEAUX DYNAMIQUE

Les tableaux dynamiques

- Colonnes stables
- Contenu des lignes variables

Stratégie

1. Trouver la ligne sur laquelle agir sur la base d'une valeur de référence
2. Conserver l'information sur le numéro de ligne
3. Effectuer les opérations voulues sur les autres cases de la même ligne



Tableaux HMTL – Numéro de ligne

➤ PARCOURIR UN TABLEAU

Cette méthode recherche la valeur `s` dans la chaque colonne (`for(WebElement case : l_cases)` de chaque ligne d'un tableau (« `for(WebElement ligne : l_lignes){...}` »)

```
public int retournerNumeroDeLigne(String s){
    int ligneCourante = 1;
    List<WebElement> l_lignes = driver.findElements(By.xpath("//table/tbody/tr"));
    for(WebElement ligne : l_lignes){
        List<WebElement> l_cell = ligne.findElements(By.xpath("td"));
        for(WebElement cell:l_cell){
            if(cell.getText().equals(s)){
                return ligneCourante;
            }
        }
        ligneCourante++;
    }
    return -1;
}
```

Tableaux HMTL – Numéro de ligne

➤ MANIPULER UNE CELLULE DE MANIÈRE DYNAMIQUE

Rappel implémentation

```
public WebElement getCellule(row, col){  
    WebElement element = driver.findElement(By.xpath("/table/tbody/tr["+row+"]/td["+col+"]"));  
    return element;  
}
```

Exemple d'utilisation dynamique

```
WebElement cellule = getCellule(retournerNumeroDeLigne(«Dupont»), 2);
```

```
WebElement cellule = getCellule(2, 2);  
cellule.getText();  
cellule.click();
```



EXERCICE : Utiliser les parcours tableaux

1. Automatisez ce test et trouver un WebElement dynamiquement à partir d'un parcours tableaux

Actions	Résultat attendu
Accéder à l'application Jpetstore et se connecter en tant que j2ee/j2ee	L'utilisateur « ABC » est bien connecté (apparition d'un message de bienvenu et du lien « Sign out »)
Entrer « dog » dans la bar de recherche et valider	La page des résultats de recherche s'affiche
Cliquer sur le lien de la colonne « Product ID » correspondant à la ligne dont le nom de l'animal « Dalmation »	La page du produit Dalmation s'affiche



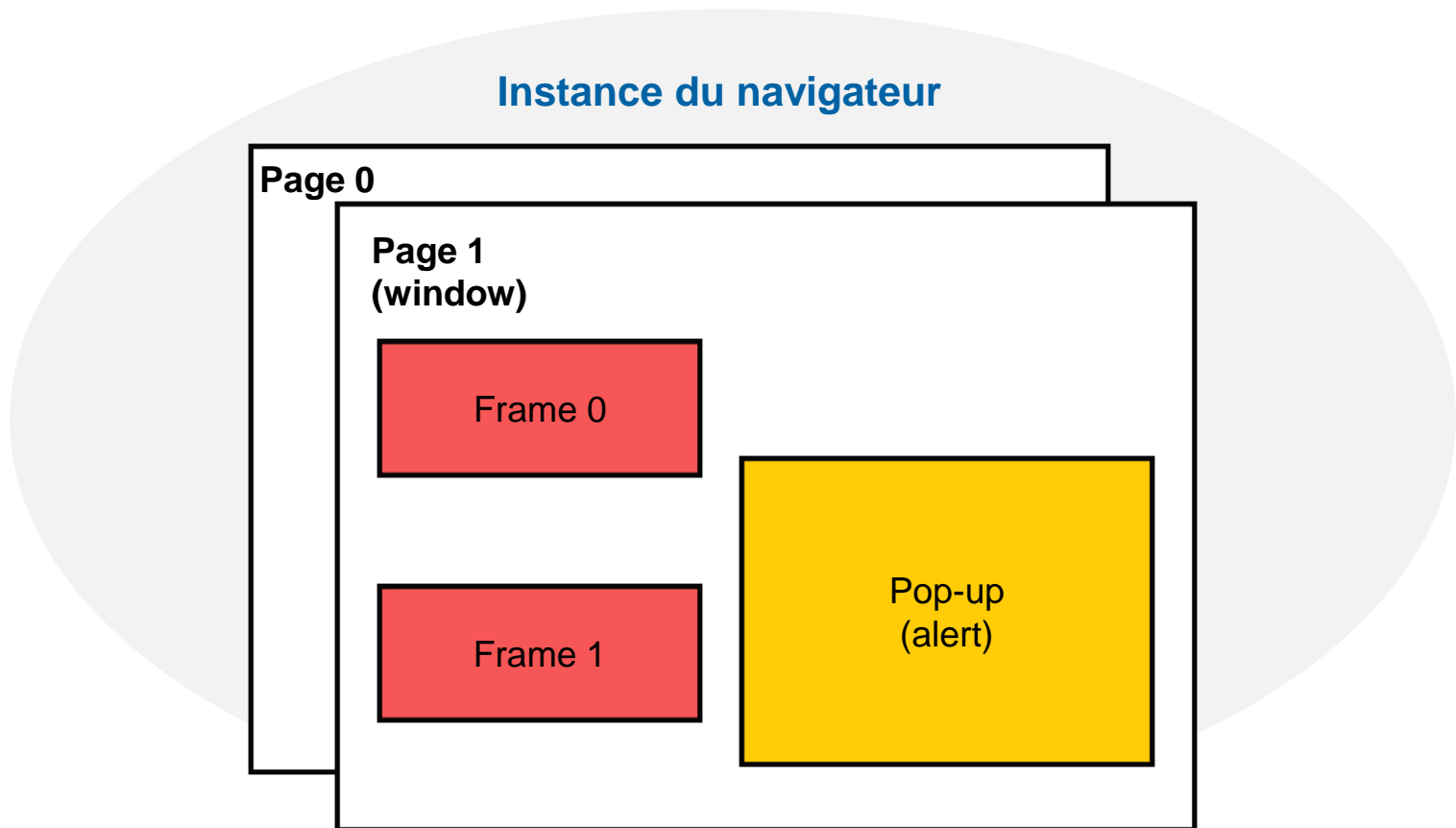
CHANGEMENTS DE PAGE, FRAMES ET ALERTES



Navigation dans les iFrames

➤ GÉRER LES FOCUS

Dans certains cas, les éléments ciblés par nos tests ne sont pas accessibles. Ils peuvent se trouver au sein de frames, de pop-up, ou de pages sur lesquels notre driver n'a pas le focus. Les méthodes **switchTo()** nous permettent de modifier ce focus et d'accéder aux éléments souhaités.



Navigation dans les iFrames

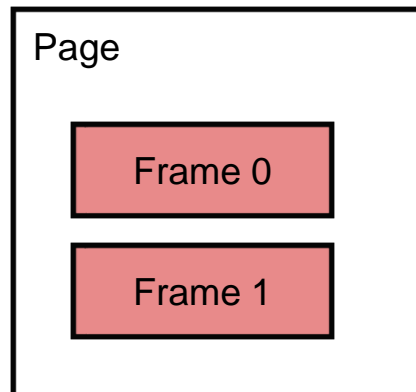
➤ SWITCHTo().FRAME

Interface WebDriver().

- `switchTo().frame(int frameNumber) : void`
Aller dans le contexte de l'iFrame en utilisant son index (ordre des iFrames dans la page)
- `switchTo().frame(String frameNameOrId) : void`
Aller dans le contexte de l'iFrame en utilisant l'id ou le nom de l'iFrame
- `switchTo().frame(WebElement frameElement) : void`
Aller dans le contexte de l'iFrame en utilisant un WebElement correspondant à l'iFrame
- `switchTo().defaultContent() : void`
Sortir de l'iFrame et revenir dans le contexte global de la page



Attention: Le **`switchTo().frame`** prend comme base le contexte dans lequel il est appelé. Si la frame appelée n'existe pas dans le contexte une exception est lancée.



```
// contexte global
driver.switchTo().frame(0);
// contexte frame 0
driver.switchTo().frame(1);
// erreur frame 1 n'existe pas dans la frame 0
driver.switchTo().defaultContent();
// contexte global
driver.switchTo().frame(1);
```

Alertes et changements de page

➤ SWITCHTo().ALERT

Interface WebDriver().

`switchTo().alert()` : Alert

Permet d'accéder aux méthodes de l'interface Alert

Les méthodes de l'interface Alert():

`.accept()` : void

Accepter l'alerte

`.dismiss()` : void

Fermer l'alerte

`.getText()` : String

Récupérer le texte de l'alerte

`.sendKeys()` : void

Saisir du texte dans l'alerte

➤ SWITCHTo().WINDOW

Interface WebDriver().

- `switchTo().getWindowHandles()` : Set<String>
- `switchTo().getWindowHandle()` : String
- `switchTo().window(String handle)` : void



TP Hotel Booking - Interactions avancées et IFrames



EXERCICE : Utiliser les switchTo()

1. Automatisez ce test sur l'application TutorialHtml5HotelPhp

Actions	Résultat attendu
Se connecter à l'application TutorialHtml5HotelPhp	Affichage de la page "HTML5 Hotel Room Booking"
cliquer sur la première cellule du planning	Affichage de la pop-up de création d'une nouvelle réservation
Donner le nom « resa 1 » et sauvegarder	La réservation apparaît bien sur le planning

INTERACTIONS COMPLEXES



Drag and Drop

➤ LES DRAG-AND-DROP

Méthodes

Actions().

- `clickAndHold(WebElement element)` : Actions
Simule un clic actif (non relâché)
- `moveToElement(WebElement element)` : Actions
Simule le déplacement du curseur jusqu'à un élément cible
- `release(WebElement element)` : Actions
Simule le relâchement du clic
- `build()` : Action
Construit l'action

Action().

- `perform(WebElement element)` : void
Réalise l'action

Exemple

```
Actions a = new Actions(driver);  
a.clickAndHold(element1).moveToElement(element2).release(element2).build().perform();
```

Il existe aussi la méthode `.dragAndDrop(WebElement source, WebElement target)`
À utiliser avec `build().perform();`

TP Hotel Booking - Interactions avancées et IFrames



EXERCICE : Utiliser les interactions avancées

1. Effectuer un « drag and drop » sur l'application TutorialHtml5HotelPhp

Actions	Résultat attendu
Se connecter à l'application TutorialHtml5HotelPhp	Affichage de la page "HTML5 Hotel Room Booking"
Bouger la réservation sur la chambre 1 au lendemain	Le message « update success » s'affiche
Attendre 7 secondes	Le message disparaît

Mouse Hover

➤ LES MOUSE HOVER

Méthodes

Actions().

- `moveToElement(WebElement element)` : Actions
Simule le déplacement du curseur jusqu'à un élément cible
- `build()` : Action
Construit l'action

Action().

- `perform(WebElement element)` : void
Réalise l'action

Exemple

- *En deux temps*

```
Actions a = new Actions(driver);  
a.moveToElement(element).build().perform();  
driver.findElement(...).click();
```

- *En une action unique*

```
Actions a = new Actions(driver);  
a.moveToElement(element).moveToElement(driver.findElement(...)).click().build().perform();
```

TP Hotel Booking - Interactions avancées et IFrames



EXERCICE : Utiliser les interactions avancées

1. Effectuer un « mouse hover » sur l'application TutorialHtml5HotelPhp

Actions	Résultat attendu
Se connecter à l'application TutorialHtml5HotelPhp	Affichage de la page "HTML5 Hotel Room Booking"
Supprimer la réservation sur la chambre 1 (en utilisant la croix en haut et à droite de la cellule)	Le message « Deleted. » s'affiche
Attendre 7 secondes	Le message disparaît

DATES ET CHAÎNES DE CARACTÈRES



Dates avec Joda Time

```
<dependency>
  <groupId>joda-time</groupId>
  <artifactId>joda-time</artifactId>
  <version>2.9.7</version>
</dependency>
```

```
DateTimeFormatter formatter = DateTimeFormat.forPattern("dd-MM-yyyy");
DateTime dateTime = formatter.parseDateTime("01-01-2000");
SimpleDateFormat format = new SimpleDateFormat("dd/MM/yy");
String date = format.format(dateTime.toDate());
Assert.assertEquals("Comparaison des dates", "01/01/00", date);
```



Dates avec Java time

```
LocalDateTime date = LocalDateTime.now();  
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy MM dd");  
String text = date.format(formatter);
```



VÉRIFICATIONS EN BASE DE DONNÉES



Principe de la vérification en base de données

L'IMPORTANCE DE LA GESTION DES DONNÉES DE TEST

Dans un projet d'automatisation, établir une connexion avec la base de donnée est essentiel, et ce pour différentes raisons :

- **Contrôler les jeux de données en entrée et sortie de test**
 - **Charger des jeux de données nécessaires au test** (Souvent les tests dépendent des données en bas. Ex : la connexion implique l'existence d'un utilisateur enregistré en base)
 - **Nettoyer la BDD après le test** (certains tests génèrent des données en base et risquent de corrompre le résultat des tests suivants)
- **Pousser le principe du test**

Comparer le contenu réel de la base de données à l'attendu (plus complet qu'une simple vérification en IHM)

Principe de la vérification en base de données

AJOUT DE DEUX DÉPENDANCE DANS LE POM...

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>3.6.0</version>
  </dependency>
  <dependency>
    <groupId>org.dbunit</groupId>
    <artifactId>dbunit</artifactId>
    <version>2.6.0</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.6</version>
  </dependency>
</dependencies>
```



Principe de la vérification en base de données

DÉMARCHE À SUIVRE

- Etablir une connexion avec la base de données
- Charger des jeux de données et/ou réinitialiser les données de test.
- Comparer le contenu réel de la base de données à l'attendu.

Le format de données que nous utiliserons est le Dbunit (flat xml)

```
<dataset>
  <category catid="RABBITS" name="Rabbits" descn=""/>
  <category catid="MICE" name="Mice" descn=""/>
  <category catid="RATS" name="Rats" descn=""/>
</dataset>
```

NOM DE
TABLE

NOM DE
COLONNE

VALEUR

Etablir une connexion avec la base de données

CRÉATION DES ATTRIBUTS DE CONNEXION À LA BDD

Quatre informations nécessaires :

- Le driver de base de données à utiliser
- La localisation de la base de données
- Le compte à utiliser
- Le mot de passe associé à ce compte

Dans une classe java (ex : `BddConnexion.java`), définir les attributs suivants :

```
private static final String DRIVER = "com.mysql.jdbc.Driver";  
private static final String JDBC_URL = "jdbc:mysql://localhost:3306/jpetstore_env2";  
private static final String USER = "jpetstore_2_user";  
private static final String PASSWORD = "squash";
```

DB Unit pour la manipulation des jeux de données

INTÉGRATION DBUNIT À JAVA

Méthode de construction d'un dataset (interface **IDataset**) à partir d'un fichier au format xml Dbunit

```
private IDataset readDataSet (String filename) throws Exception{  
    return new FlatXmlDataSetBuilder().build(new File(filename));  
}
```

Méthode d'insertion d'un dataset (type **IDataset**) à partir d'une connexion à une BDD (interface **IDatabaseTester**)

```
public void insertData(String path_to_file) throws Exception {  
    IDataset dataset = readDataSet(path_to_file);  
    IDatabaseTester databaseTester = new JdbcDatabaseTester(DRIVER, JDBC_URL, USER, PASSWORD);  
    databaseTester.setSetUpOperation(DatabaseOperation.CLEAN_INSERT);  
    databaseTester.setDataSet(dataset);  
    databaseTester.onSetup();  
}
```

Méthode de suppression d'un dataset (type **IDataset**) à partir d'une connexion à une BDD (interface **IDatabaseTester**)

```
public void deleteAllData(String path_to_file) throws Exception {  
    IDataset dataset = readDataSet(path_to_file);  
    IDatabaseTester databaseTester = new JdbcDatabaseTester(DRIVER, JDBC_URL, USER, PASSWORD);  
    databaseTester.setSetUpOperation(DatabaseOperation.DELETE_ALL);  
    databaseTester.setDataSet(dataset);  
    databaseTester.onSetup();  
}
```

DB Unit pour la manipulation des jeux de données

INTÉGRATION DBUNIT À JAVA

Méthode de construction d'un dataset (interface **IDataset**) à partir d'un fichier au format xml Dbunit

```
private IDataset readDataSet (String filename) throws Exception{
    return new FlatXmlDataSetBuilder().build(new File(filename));
}
```

Méthode de comparaison de deux dataset (sous forme **ITable**) . La première table est créée à partir du fichier xml et la seconde est la table telle quelle est en BDD

```
public void compareData(String table, String path_to_file, String... col) throws SQLException, Exception {
    IDatabaseTester databaseTester = new JdbcDatabaseTester(DRIVER, JDBC_URL, USER, PASSWORD);
    IDataset databaseDataSet = databaseTester.getConnection().createDataSet();
    ITable actualTable = databaseDataSet.getTable(table);
    IDataset expectedDataSet = readDataSet(path_to_file);
    ITable expectedTable = expectedDataSet.getTable(table);
    String tab[] = col;
    Assertion.assertEqualsIgnoreCols(expectedTable, actualTable, tab);
}
```

Permet d'exclure certaines colonnes de la comparaison



EXERCICE : Gérer les données de test



Pour chacun des trois tests créés précédemment pour l'application Hotel Room, mettre en place une gestion des données en entrée et sortie de test.

(insert et suppression de données avec fichier xml flatDbUnit).

1. Créer une classe BddOutils.java avec :

- Informations de connexion à la BDD
- Implémentation de la méthode « readDataSet »
- Implémentation de deux méthodes d'insertion et de suppression de données en BDD

2. Constituer les jeux de données xml.

3. Appeler les méthodes dans les tests pour les rendre indépendant

TIPS : Le fichier xml dbUnit de suppression des données d'une table s'écrit :

```
<dataset>
  <nomTableAVider/>
</dataset>
```

FAIRE DES COPIES D'ECRAN



Support Selenium

- Selenium fournit un support pour effectuer des copies d'écran
- *getScreenshotAs(OutputType<X> target) throws WebDriverException : <X>*
 - *Retourne une copie d'écran de la page par ordre de préférence:*
 - *Page entière*
 - *Fenêtre courante*
 - *Portion visible de la frame courante*
 - *L'ensemble de l'écran*
- *OutputType :*
 - *FILE*
 - *BASE64*
 - *BYTES*



Exemple d'usage

Dans le pom.xml, ajouter :

```
<!-- https://mvnrepository.com/artifact/ru.yandex.qatools.ashot/ashot -->
<dependency>
<groupId>ru.yandex.qatools.ashot</groupId>
<artifactId>ashot</artifactId>
<version>1.5.3</version>
</dependency>
```

Dans une classe outil :

```
public static void takeSnapShot(WebDriver webdriver,String filePath) throws Exception{
    //Convert web driver object to TakeScreenshot
    TakesScreenshot scrShot =((TakesScreenshot)webdriver);
    //Call getScreenshotAs method to create image file
    File SrcFile=scrShot.getScreenshotAs(OutputType.FILE);
    //Move image file to new destination
    File DestFile=new File(filePath);
    //Copy file at destination
    FileUtils.copyFile(SrcFile, DestFile);
}
```

Utilisation de la méthode

```
takeSnapShot(driver, ".\\src\\test\\target\\snapshots\\nameOfFile.png");
```

INCLURE UN SCRIPT DANS SELENIUM (EX: AUTOIT)



Inclure un script dans Selenium (ex: Autolt)

Objectif

Ecrire dans les tests qui interagissent avec des éléments qui ne se trouvent pas dans le navigateur (par exemple pour saisir un mot de passe, accepter une alerte ou effectuer un téléchargement)

Méthode proposée

- Réaliser un script **Autolt** qui interagit avec les fenêtres Windows
- Appeler ce script depuis un test Selenium

Outillage

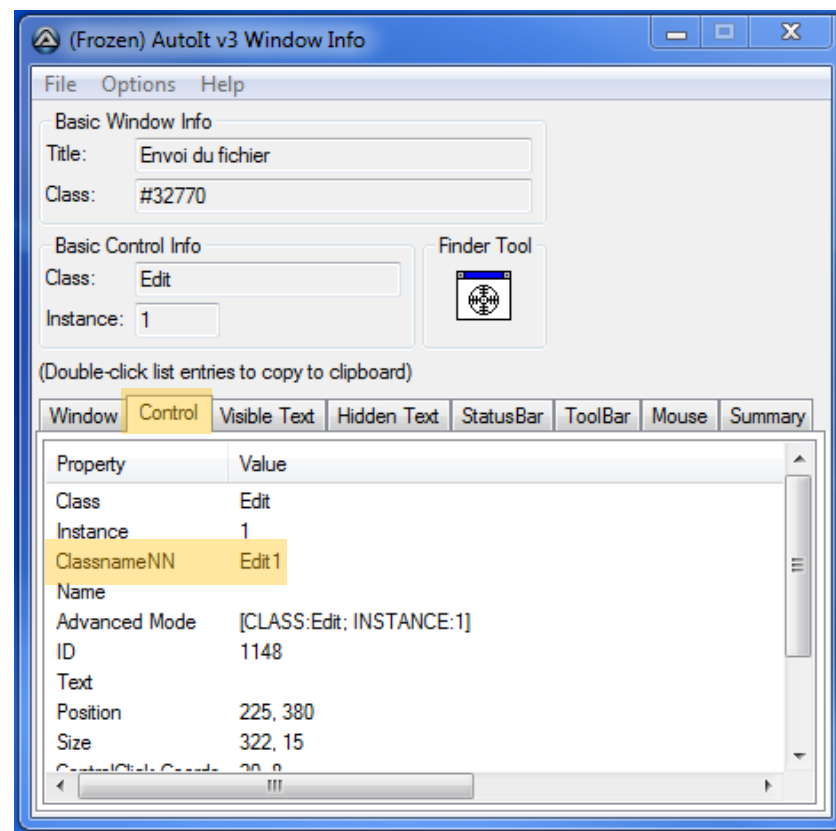
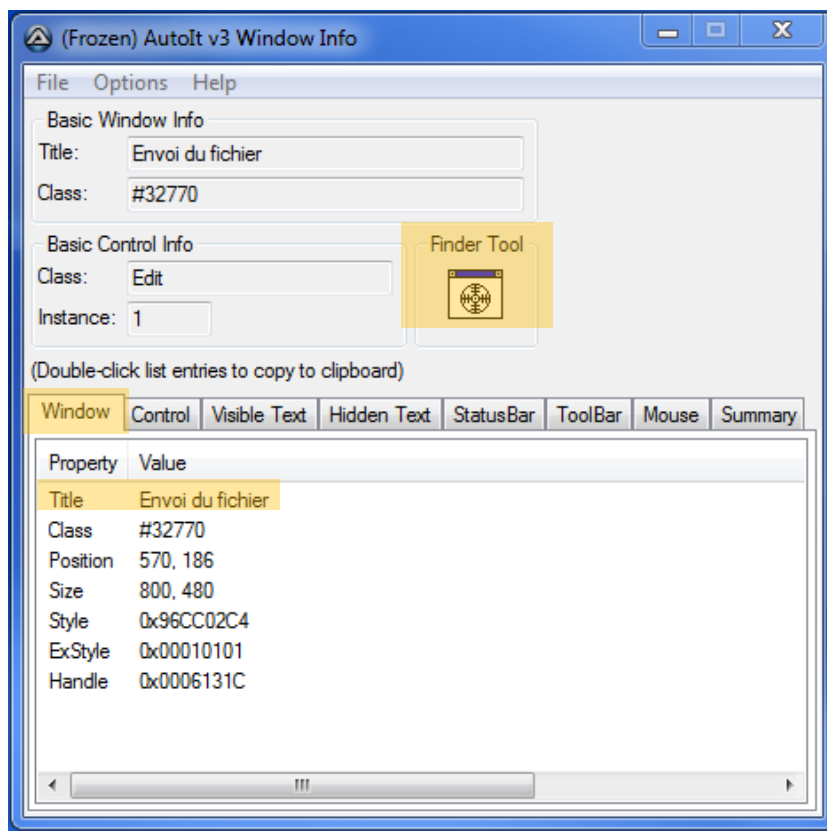
- SciTE : pour éditer les scripts
- Au3Info : pour obtenir des informations sur les objets Windows
- Autolt3 : pour compiler le script en exécutable Windows



Inclure un script dans Selenium (ex: Autolt)

TP : Automatiser un chargement de fichier

- Ouvrir SciTE
- Ouvrir Auto3Info



Inclure un script dans Selenium (ex: Autolt)

1

Contro

ControlClick

ControlCommand

ControlDisable

ControlEnable

ControlFocus

ControlGetFocus

ControlGetHandle

ControlGetPos

ControlGetText

ControlHide

ControlListView

ControlMove

ControlSend

ControlSetText

ControlShow

ControlTreeView

WinWai

WinWait

WinWaitActive

WinWaitClose

WinWaitNotActive

Inclure un script dans Selenium (ex: Autolt)

ControlClick(

ControlClick ("title", "text", controlId [, button = "left" [, clicks = 1 [, x [, y]]])
Sends a mouse click command to a given control.

ControlFocus(

ControlFocus ("title", "text", controlId)
Sets input focus to a given control on a window.

ControlSetText(

ControlSetText ("title", "text", controlId, "new text" [, flag = 0])
Sets text of a control.

Sleep(

Sleep (delay)
Pause script execution.

Raccourcis :

- F7 pour compiler
- F5 pour exécuter

Inclure un script dans Selenium (ex: Autolt)

Intégration à Selenium WebDriver :

```
Runtime.getRuntime().exec("D:\\AutoIt\\AutoItTest.exe");  
Thread.sleep(5000);
```



- Avec AutoIT, écrire un script qui :
 - ouvre l'application Bloc-notes (notepad.exe)
 - écrit « Ceci est mon test »
 - enregistre le document sous le nom de « monFichier.txt »
- Avec Selenium, écrire un test qui :
 - se connecte à un compte de messagerie (s'en créer un pour les besoins du test)
 - envoie un mail à l'adresse : testAutom34life@gmail.com comprenant en PJ « monFichier.txt » (pour réaliser l'action d'attacher une pièce jointe, il faudra évidemment écrire un nouveau script AutoIT, intégré au test seleniumS).



INCLURE DU JAVASCRIPT



Inclure un script Javascript dans Selenium

```
WebDriver driver = new FirefoxDriver();

if (driver instanceof JavascriptExecutor) {
    ((JavascriptExecutor)
    driver).executeScript("alert('hello world');");
}
```



GESTION DES ÉVÉNEMENTS



EventFiringWebDriver et WebDriverEventListener

```
public class Ecouteur implements WebDriverEventListener{  
    ...  
}
```

```
WebDriver driver = new FirefoxDriver();  
EventFiringWebDriver eDriver = new EventFiringWebDriver(driver);  
Ecouteur ecouteur = new Ecouteur();  
eDriver.register(ecouteur);
```



RAPPORTS ET COPIES D'ÉCRAN



Générer un rapport avec JUnit

```
public class JUnitTestReporter {
```

@Rule

```
public TestRule junitWatcher = new TestWatcher() {
```

@Override

```
public Statement apply(Statement base, Description description) {  
    return super.apply(base, description);  
}
```

@Override

```
protected void succeeded(Description description) {  
    System.out.println(description.getDisplayName() + " " + "Test Passed!");  
}
```

@Override

```
protected void failed(Throwable e, Description description) {  
    System.out.println(description.getDisplayName() + " "  
        + e.getClass().getSimpleName());  
}
```

```
};
```

```
}
```

Générer un rapport avec JUnit

```
import static org.junit.Assert.assertTrue;  
import org.junit.Test;  
  
public class JUnitSampleTest extends JUnitReporter {  
    ...  
}
```



Générer un rapport HTML avec JUnit

```
public class JUnitHTMLReporter {  
  
    static File junitReport;  
    static BufferedWriter junitWriter;  
  
    @BeforeClass  
    public static void setUp() throws IOException {  
        ...  
    }  
  
    @AfterClass  
    public static void tearDown() throws IOException {  
        ...  
    }  
  
    @Rule  
    public TestRule watchman = new TestWatcher() {  
        ...  
    }  
}
```



Exercice

- *Ajoutez des captures d'écran dans selenium :*
 - *Systematiquement avant et après une opération donnée*
 - *Uniquement en cas d'erreur*
 - *(par exemple sur la détection d'un objet)*
- *Faites en sorte que les images soient affichées ou accessibles depuis des logs*



MESURER LES TEMPS DE RÉPONSE



Chronométrer des temps

- Usage marginal dans le cadre de Selenium (Il existe de meilleurs outils pour la performance)
- Le calcul des temps se fait à l'aide de la méthode :
 - `System.currentTimeMillis()` : long
 - Retourne une représentation du moment actuel en millisecondes (= la différence entre le moment présent et le 1er janvier 1970 à 00:00 UTC)



Chronométrer des temps - Exercice

- Créez une interface IChronometre permettant de répondre aux exigences suivantes :
 - Il faut pouvoir démarrer le chronomètre, arrêter le chronomètre, remettre le chronomètre à 0, lire le temps écoulé au chronomètre
- Implémentez l'interface IChronometre
- Utilisez le chronomètre dans un test selenium
 - En tant que simple log informatif
 - En tant que condition de succès du test

