



GESTION DES TESTS

2 jours



Sommaire

- Introduction
- Qualité logicielle
- Les tests et difficultés
- Définitions des tests
- Le plan de tests
- Tests unitaires avec Java
- Selenium



MODULE 1

Introduction

Qu'est-ce qu'un logiciel ?

- il est fait pour des **utilisateurs**
- il est **complexe** et de **très grande taille**
- il fait intervenir **plusieurs participants**
 - travail en équipe(s),
 - organisation,
 - planification
- il est **long** et **coûteux** à développer

Définition du logiciel

Un logiciel (*software*) est l'ensemble :

- des programmes,
- des procédures
- et des documentations

nécessaires au fonctionnement d'un système informatique

Caractéristiques du logiciel

- le logiciel est **facile à reproduire**
 - tout le coût se trouve dans le développement
- le logiciel est **immatériel et invisible**
 - on ne peut l'observer qu'en l'utilisant
 - la qualité n'est pas vraiment apparente
 - difficile d'estimer l'effort de développement
- développement **difficile à automatiser**
 - beaucoup de main-d'œuvre nécessaire ...

Caractéristiques du logiciel

le logiciel ne s'use pas, mais il vieillit

- **Détérioration suite aux changements :**

- complexification induite
- duplication de code
- introduction d'erreurs

- **Mal conçu au départ :**

- Inflexibilité
- manque de modularité
- documentation insuffisante

Evolution du matériel

Différentes catégories de logiciels

- **sur mesure** : pour un client spécifique
- **générique** : vendu sur un marché
- logiciel **temps-réel**
- logiciel **embarqué**
- logiciel **distribué**

Problématique historique du logiciel

« D'une part le logiciel n'est **pas fiable**, d'autre part il est incroyablement difficile de réaliser dans les **délais prévus** des logiciels **satisfaisant leur cahier des charges** »

Le logiciel n'est pas fiable ...

Quelques **exemples célèbres**

- **1ère sonde Mariner vers Vénus**
 - perdue dans l'espace (erreur Fortran)
- **navette spatiale**
 - lancement retardé (problème logiciel)
- **ATT**
 - appels longue distance suspendus (changement de version)
- **avion F16**
 - retourné à l'équateur (non prise en compte hémisphère sud)
- **bug de l'an 2000**

Tout système comporte des **bugs** ...

Délais et exigences non satisfaits ...

Quelques **exemples célèbres**

- OS 360 d'IBM
 - en retard, dépassement mémoire et prix, erreurs
- aéroport de Denver (système de livraison des bagages)
 - 1 an de retard
- SNCF (système Socrate)
 - difficultés importantes

Abandons ...

Quelques **exemples célèbres**

- **EDF (contrôle-commande centrales 1400 MWatts)**
 - renonce après plusieurs années d'efforts

- **bourse de Londres (projet d'informatisation)**
 - abandon : 4 années de travail + 100 ML perdus

- **Etats-Unis (système de trafic aérien)**
 - Abandon

La non rencontre des exigences et les dépassements de délais sont **fréquents** : 300 à 400 %

Méthodologies de développement

Comme pour tout produit manufacturé complexe :

- on **décompose** la production en « **phases** »
- l'ensemble des phases constitue un « **cycle de vie** »
- les phases font apparaître des **activités** clés

Activités du développement de logiciel

- analyse des besoins
- spécification
- conception
- programmation
- intégration
- vérification et validation

Analyse des besoins

- **But :**
 - déterminer **ce que doit faire** (et ne pas faire) **le logiciel**
 - déterminer les **ressources**, les **contraintes**
- **Caractéristiques :**
 - parler **métier** et non info
 - entretiens, questionnaires
 - **observation** de l'existant, **étude** de situations similaires
- **Résultat :** ensemble de documents
 - **rôle** du système
 - future **utilisation**
 - aspects de l'**environnement**
 - (parfois) un manuel d'utilisation préliminaire

Spécification

- **But :**
 - établir une 1ère description du futur système
 - consigner dans un document qui fait référence
- **Données :**
 - résultats de l'analyse des besoins + faisabilité informatique
- **Résultat :** Spécification Technique de Besoin (STB)
 - **ce que fait** le logiciel, mais **pas comment**
- **Remarques:**
 - pas de choix d'implémentation
 - (parfois) un manuel de référence préliminaire

Conception

- **But :**
 - décrire **comment** le logiciel est construit
 - décider **comment** utiliser la techno. pour répondre aux besoins
- **Travail :**
 - enrichir la description de détails d'implémentation
 - pour aboutir à une description très proche d'un programme
- **2 étapes :**
 - **conception architecturale**
 - **conception détaillée**

Conception architecturale

- **But** : décomposer le logiciel en composants
 - mettre au point l'**architecture** du système
 - définir les **sous-systèmes** et leurs **interactions**
 - concevoir les **interfaces** entre composants
- **Résultat** :
 - **description** de l'architecture globale du logiciel
 - **spécification** des divers composants

Conception détaillée

- **But** : élaborer les éléments internes de chaque sous-syst.
- **Résultat** :
 - pour **chaque composant**, description des
 - **structures de données, algorithmes**
- **Remarque** :
 - si la **conception** est **possible**, la **faisabilité** est **démontrée**
 - sinon, la **spécification** est **remise en cause**

Programmation

- **But :**
 - passer des structures de données et algorithmes
 - à un ensemble de programmes
- **Résultat :**
 - ensemble de **programmes**
 - ensemble de **bibliothèques / modules**
 - **documentés** (commentaires)
- **Remarques:**
 - activité la mieux **maîtrisée** et **outillée** (parfois automatisée)

Gestion de configurations et Intégration

- **Gestion de configurations :**
 - **gérer les composants** logiciels (programmes, bibliothèques, ...)
 - **maîtriser leur évolution** et leurs mises à jour
- **Intégration :**
 - **assembler** les composants
 - pour obtenir un système exécutable

Vérification

- **But** : vérifier par rapport aux spécifications
 - **vérifier** que les descriptions successives
 - (et *in fine* le logiciel) **satisfont la STB**
- **Moyens** : revues de projet, tests
 - test = **chercher des erreurs** dans un programme
 - exécution sur un sous-ensemble fini de données
- **4 types de tests** :
 - **test unitaire** : composants isolés
 - **test d'intégration** : composants assemblés
 - **test système** : système installé sur site
 - **test d'acceptation** : testé par l'utilisateur

Validation

- **But** : vérifier par rapport aux utilisateurs
- **Moyen** : revues de projet

Disposition de titre et de contenu avec liste

Question ?



MODULE 2

Gestion des Tests

Sommaire

- Qualité logicielle
- Les tests et difficultés
- Définitions des tests
- Le plan de tests

Qualité logicielle

- Différentes perceptions de la qualité ...



Facteurs de qualité

Qualité externe :

- **complétude fonctionnelle**
 - réalise toutes les tâches attendues
- **ergonomie / convivialité**
 - facile d'utilisation
 - apprentissage aisé
- **fiabilité / robustesse**
 - tâches effectuées sans problème
 - fonctionne même dans des cas atypiques

Facteurs de qualité

Qualité **interne** :

- **adaptabilité**
 - facile à modifier, à faire évoluer
- **Réutilisabilité**
 - des parties peuvent être réutilisées facilement
- **traçabilité / compréhension**
 - le fonctionnement du logiciel est facile à comprendre
- **efficacité / performance**
 - bonne utilisation des ressources (mémoire, cpu, ...)
- **Portabilité**
 - capacité à marcher dans un autre contexte ou env.

Définitions

- **Anomalie** : manifestation observée d'un comportement différent du comportement attendu ou spécifié (la norme)
- **Défaut** : manque, insuffisance : l'anomalie est constatée parce qu'il y a un défaut
- **Erreur** : faute commise en se trompant, méprise : l'erreur provoque un défaut se manifestant par une anomalie

C'est parce qu'on peut commettre des erreurs en développant des logiciels que les tests sont nécessaires

Les tests et difficultés

- Tout programme est faux, à moins qu'on ait prouvé le contraire
- Les tests servent à montrer la présence d'erreurs, jamais à prouver leur absence
(Dijkstra 1969)
- Le programmeur est la personne la moins bien placée pour tester son programme
- Faible rendement Durée importante
- Tâche difficile

Les tests et difficultés

- Un programme n'est jamais faux
- Mais il ne fait pas ce que l'on attend de lui !
 - Les tests sont estimés à 33% du coût d'un projet !
- Plus un programme est compliqué, plus il est difficile à tester

Les tests et difficultés

Techniques:

- **Mauvaise synchronisation** entre écriture et tests
- **Problèmes de disponibilités de ressources** (modules, systèmes, fichiers non disponibles)
- Nécessité de création d'outils spécifiques (génération de cas, interprétation des résultats)
- Besoin d'environnement(s) particulier(s)
- Problèmes liés à la stabilité des environnements (quelle version de système, de SGBD, etc.)

Les tests et difficultés

Humaines

- Tâche fastidieuse
- Les tests débutent souvent en retard (tendance à cumuler les retards sur les tests)
- Blocages psychologiques (erreurs ressenties comme des fautes par le programmeur qui les a commises)
 - C'est la première fois que l'on montre quelque chose :
remise en cause des spécifications
Conflits
- Manque de formation à la technique des tests
- Manque d'objectifs précis
- Manque de rigueur

Les tests et difficultés

Il existe (malheureusement) de nombreuses causes ou possibilités d'erreurs lors du développement d'applications informatiques

Pendant les phases de conception

- mauvaise expression des besoins
- mauvaises solutions techniques
- etc.

Pendant la conception de la logique détaillée :

- mauvaise interprétation des spécifications logique incomplète
- cas particulier omis cas d'erreurs négligés
- etc.

Les tests et difficultés

Pendant le codage :

- erreurs de syntaxe
- erreurs d'initialisation
- erreurs de paramètres
- erreurs de compteur de boucle
- définition multiple ou non-définition de variables
- déclarations de type ou de dimension incorrects
- erreur d'écriture du nom d'une variable etc.

Pendant la translation (link):

- erreur de compilation restante
- mauvaise résolution des références externes
- confusion des noms de membres ou de bibliothèques

Comment agir sur la qualité logicielle ?

La **qualité** est atteinte ou **améliorée** en appliquant certains **principes** :

- rigueur et formalisme
- séparation des préoccupations
- modularité
- généralité / abstraction
- incrémentalité
- anticipation des changements

Rigueur et formalisme

- **rigueur** = précision, exactitude (*confiance en la fiabilité*)
- **formalisme** = le plus haut degré de rigueur (*mathématiques*)
 - nécessaire pour les parties critiques (haut risque)
 - peut être utilisé dans chaque phase
 - spécification formelle
 - vérification formelle (preuve)
 - analyse de complexité d'algorithmes
 - ...

Séparation des préoccupations

- **principe** : traiter séparément les \neq aspects d'un problème
 - **diviser pour régner**
- **résultat** : réduit la quantité de complexité à contrôler

Séparation des préoccupations

Différentes sortes de séparations :

- **séparation de domaine**
 - domaine de problème : quoi résoudre ?
 - domaine de solution : comment résoudre ?
- **séparation de temps** : phases du cycle de vie
- **séparation de qualité**
 - maquettes, prototypes
 - conception globale, détaillée
- **vues séparées sur le logiciel** : modélisation en UML
 - cas d'utilisation, structure statique
 - comportement dynamique, architecture
- **séparation de responsabilités** : org. en équipes projet

Modularité

- **principe** : séparer le système en composants logiques
 - modules
- **avantages** :
 - réduction de complexité
 - les modules peuvent être
 - conçus et construits séparément
 - réutilisés
 - système modifié en changeant un nombre limité de modules

Généralité / Abstraction

- **principe :**

- généraliser un problème particulier
- le rendre réutilisable dans d'autres contextes

- **exemple :**

- **logiciel générique** vs **logiciel sur mesure**
- **design patterns** : des solutions généralisées pour des problèmes typiques de conception

Incrémentalité

- **principe :**
 - développer le logiciel en une série d'incrément
 - se rapprocher de la solution par raffinements successifs

- **exemple :**
 - phase de conception
 - cycle de vie en spirale
 - méthode agile

Définitions des tests

- **IEEE/ANSI 1990 (Standard 610-12)**

- Processus consistant à exécuter un système ou un composant dans des conditions spécifiées, en observant ou en enregistrant les résultats, et à faire une évaluation de certains des aspects du système ou du composant

- **IEEE/ANSI 1983 (Standard 823)**

- Processus consistant à analyser un élément logiciel pour détecter les différences entre les conditions requises et celles existantes, et permettant d'évaluer les caractéristiques de l'élément logiciel

- **Autre définition préférée**

- **Test** : action qui consiste à exécuter un programme (une partie de programme ou un système) avec l'intention de trouver des erreurs

Définitions des tests

Vérification

- Processus d'évaluation d'un système ou d'un composant pour déterminer si le produit satisfait, à sa phase de développement, les conditions imposées au début de la phase
on vérifie qu'on a bien fait un logiciel

Validation :

- Processus d'évaluation d'un système ou d'un composant durant ou à la fin de son développement pour déterminer s'il satisfait ses spécifications
on valide qu'on a fait le bon logiciel

Test = vérification + validation

Définitions des tests

- **Boîte noire**

- On ne connaît que les spécifications (l'aspect externe)
- On valide le programme par rapport à ses spécifications
- Test fonctionnel

- **Boîte blanche**

- On connaît à la fois les spécifications et l'implémentation (pseudo- code ou code)
- valide la réalisation du programme
- Test structurel

- **Boîte grise**

- Combinaison des deux techniques

- **Non régression**

- Un test de non régression consiste vise à s'assurer qu'une évolution n'a pas introduit de nouveaux défauts

Le plan de tests

Le plan de test décrit toutes les procédures permettant de conduire les tests jusqu'à leur terme

Un plan de tests doit, au minimum :

- établir les objectifs de chaque phase de tests,
- définir les responsabilités et le planning pour chacune des activités des tests,
- s'assurer de la disponibilité des outils de tests et données de tests
- établir les normes de conduite des tests,
- définir les critères de fin et d'acceptabilité de chaque test.

Le plan de tests

Composants à tester

- faire une liste exhaustive des composants à tester
-
- préciser pour chaque composant :
 - les objectifs des activités de test les types de test et les étapes
 - les méthodes de test choisies pour chaque type et chaque étape
 - les outils de test utilisés
 - les règles à suivre pour :
 - définir chaque type de test rédiger les procédures de test
 - exécuter les tests définir et décider les critères d'arrêt des tests
 - enregistrer les résultats de test et rédiger les comptes-rendus
 - exploiter les résultats des tests

Objectif de chaque test

Le plan devra détailler pour chaque test l'objectif à atteindre

- validité des fonctionnalités
- contrôle de l'enchaînement des écrans
- ergonomie du produit
- performances etc.

Types de test

Le plan devra distinguer les différents types de tests

- test unitaire (module)
- test de programme ou de transaction
- test d'analyse
- test d'intégration (d'enchaînement)
- test de recette utilisateur
- recette exploitation
- tests système
- test inter-applications etc.

Procédures de tests

Pour chaque type de test, on doit indiquer les moyens utilisés

- test boîte blanche
- test boîte noire
- test de non régression
- contrôle des chemins
- contrôle de couverture
 - couverture de branches
 - couverture de conditions
 - conditions multiples
 - etc.

Critères d'arrêt

Pour chaque type de test, on doit préciser le critère d'arrêt du test

- taux de couverture atteint
- taux ou type d'erreurs détectés
- dépassement du temps imparti
- intervalle de variables
- performances atteintes
- volumes atteints
- etc.

Disposition de titre et de contenu avec liste

Question ?



MODULE 3

Développement dirigé par les tests

Test unitaire

- Test d'un bloc (unité) de programme (classe, méthode, etc.)
 - Vérification des comportements corrects
 - Vérification des comportements incorrects (valeurs incohérentes des paramètres, ...)
- Approche incrémentale
- Doivent être rejoués pour vérifier la non-régression

Test Driven Development(TDD)

- Initialement conceptualisé par Erich Gamma et Kent Beck
- Plus généralement intégré aux approches de développement agile :
 - eXtreme Programming,
 - Scrum,
 - etc.
- Utilisation de tests unitaires comme spécification du code
- De nombreux langages possède leur canevas de test unitaires
 - SUnit,
 - JUnit,
 - RUnit,
 - etc.

Cycle de TDD

1. Ecrire un premier test
2. Vérifier qu'il échoue (car le code qu'il teste n'existe pas), afin de vérifier que le test est valide
3. Ecrire juste le code suffisant pour passer le test
4. Vérifier que le test passe
5. Réviser le code (refactoring), i.e. l'améliorer tout en gardant les mêmes fonctionnalités

Avantages

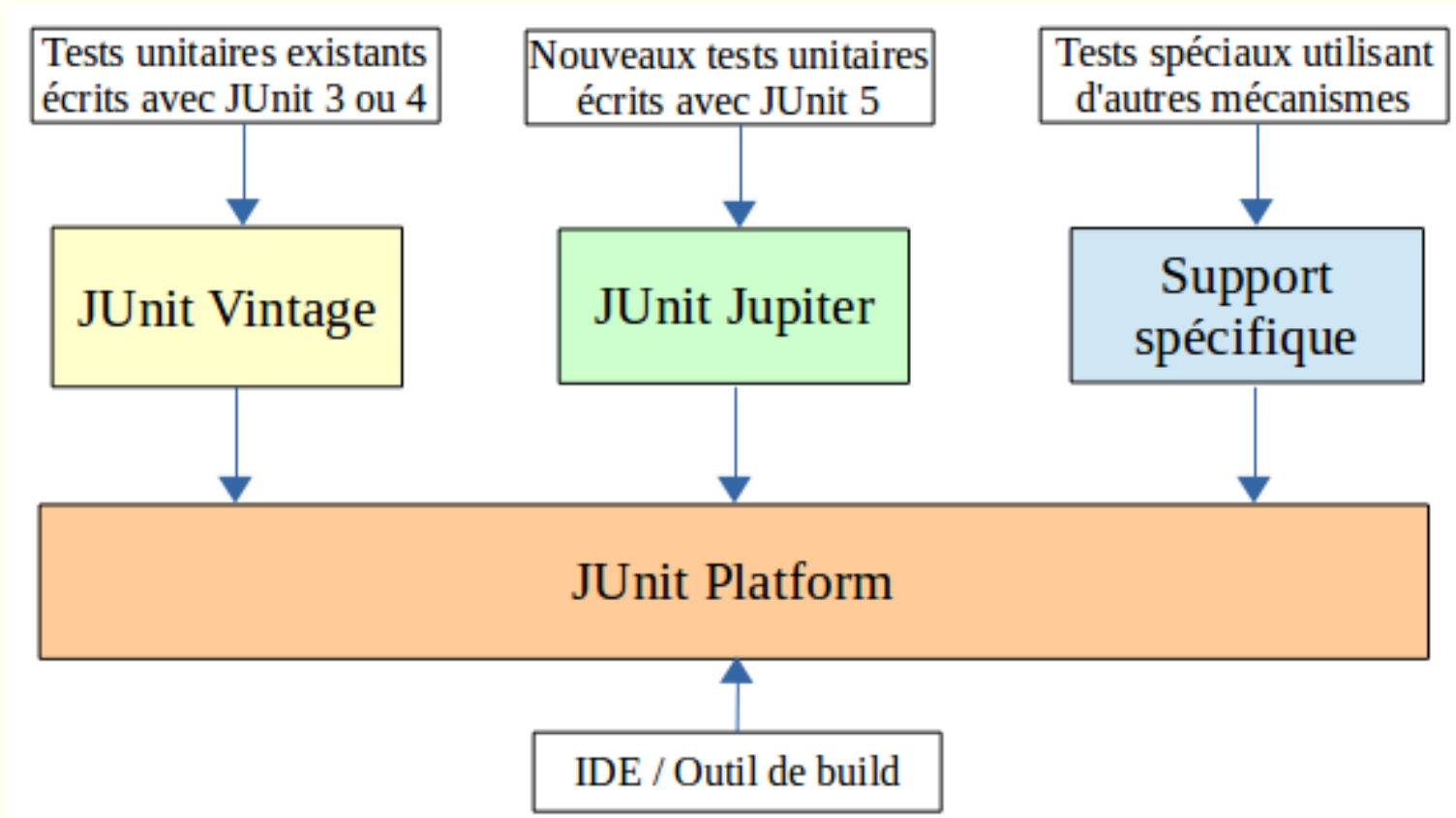
- Ecrire les tests d'abord \Rightarrow on utilise le programme avant même qu'il existe
- Diminue les erreurs de conception
- Augmente la confiance en soi du programmeur lors de la révision du code
- Construction conjointe du programme et d'une suite de tests de non-régression
- Estimer l'état d'avancement du développement d'un projet (vélocité)

JUnit 5

JUnit 5 sera divisé en 3 modules :

- ***JUnit Platform*** : contient tout l'aspect « moteur » de JUnit. On utilise ce module lorsqu'on veut faire exécuter les tests.
- ***JUnit Jupiter*** : combine une API et des mécanismes d'extension. Ce sont les éléments de ce module qu'on utilise dans les tests unitaires.
- ***JUnit Vintage*** : fournit un moyen d'exécuter les tests unitaires existants initialement écrits pour JUnit 3 et 4.

Utilisation



Projet Maven

Les dépendances peuvent être définies dans un projet Maven selon les besoins.

Exemple :

```
01. <properties>
02.   <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
03.   <java.version>1.8</java.version>
04.   <junit.version>4.12</junit.version>
05.   <junit.jupiter.version>5.0.0</junit.jupiter.version>
06.   <junit.vintage.version>${junit.version}.0</junit.vintage.version>
07.   <junit.platform.version>1.0.0</junit.platform.version>
08. </properties>
09. <dependencies>
10.   <dependency>
11.     <groupId>org.junit.jupiter</groupId>
12.     <artifactId>junit-jupiter-api</artifactId>
13.     <version>${junit.jupiter.version}</version>
14.     <scope>test</scope>
15.   </dependency>
16.   <dependency>
17.     <groupId>org.junit.jupiter</groupId>
18.     <artifactId>junit-jupiter-params</artifactId>
19.     <version>${junit.jupiter.version}</version>
20.     <scope>test</scope>
21.   </dependency>
22.   <!-- Pour executer des tests ecrits avec un IDE
23.        qui ne supporte que les versions precedentes de JUnit -->
24.   <dependency>
25.     <groupId>junit</groupId>
26.     <artifactId>junit</artifactId>
27.     <version>${junit.version}</version>
28.     <scope>test</scope>
29.   </dependency>
30.   <dependency>
31.     <groupId>org.junit.platform</groupId>
32.     <artifactId>junit-platform-runner</artifactId>
33.     <version>${junit.platform.version}</version>
34.     <scope>test</scope>
35.   </dependency>
36.   <dependency>
37.     <groupId>org.junit.jupiter</groupId>
38.     <artifactId>junit-jupiter-engine</artifactId>
39.     <version>${junit.jupiter.version}</version>
40.   </dependency>
41.   <dependency>
42.     <groupId>org.junit.vintage</groupId>
43.     <artifactId>junit-vintage-engine</artifactId>
44.     <version>${junit.vintage.version}</version>
45.   </dependency>
46. </dependencies>
```

M.Mbengue

Les annotations

Annotation	Rôle
@Test	La méthode annotée est un cas de test. Contrairement à l'annotation @Test de JUnit, celle-ci ne possède aucun attribut.
@ParameterizedTest	La méthode annotée est un cas de test paramétré
@RepeatedTest	La méthode annotée est un cas de test répété
@TestFactory	La méthode annotée est une fabrique pour des tests dynamiques
@TestInstance	Configurer le cycle de vie des instances de tests
@TestTemplate	La méthode est un modèle pour des cas de tests à exécution multiple
@DisplayName	Définir un libellé pour la classe ou la méthode de test annotée
@BeforeEach	<p>La méthode annotée sera invoquée avant l'exécution de chaque méthode de la classe annotée avec @Test, @RepeatedTest, @ParameterizedTest ou @Testfactory.</p> <p>Cette annotation est équivalente à @Before de JUnit 4</p>
@AfterEach	<p>La méthode annotée sera invoquée après l'exécution de chaque méthode de la classe annotée avec @Test, @RepeatedTest, @ParameterizedTest ou @Testfactory.</p> <p>Cette annotation est équivalente à @After de JUnit 4</p>

Les annotations

Annotation	Rôle
@BeforeAll	<p>La méthode annotée sera invoquée avant l'exécution de la première méthode de la classe annotée avec @Test, @RepeatedTest, @ParameterizedTest ou @Testfactory.</p> <p>Cette annotation est équivalente à @BeforeClass de JUnit 4.</p>
@AfterAll	<p>La méthode annotée sera invoquée après l'exécution de toutes les méthodes de la classe annotées avec @Test, @RepeatedTest, @ParameterizedTest et @Testfactory.</p> <p>Cette annotation est équivalente à @AfterClass de JUnit 4</p>
@Nested	Indiquer que la classe annotée correspond à un test imbriqué
@Tag	Définir une balise sur une classe ou une méthode qui permettra de filtrer les tests exécutés. Cette annotation est équivalente aux Categories de JUnit 4 ou aux groups de TestNG
@Disabled	<p>Désactiver les tests de la classe ou la méthode annotée.</p> <p>Cette annotation est similaire à @Ignore de JUnit 4</p>
@ExtendWith	Enregistrer une extension

La définition d'une méthode de test

```
1 package com.formation.test;
2
3 import static org.junit.jupiter.api.Assertions.assertEquals;
4
5 import org.junit.jupiter.api.Test;
6
7 class PremierJUnit5Test {
8
9     @Test
10     void premierTest() {
11         String message = "1+1 should be equal to 2";
12         System.out.println(message);
13
14         assertEquals(2, 1 + 1, message);
15     }
16 }
```


La définition d'un libellé

```
10 @DisplayName("A special test case")
11 class DisplayNameTest {
12
13     @BeforeAll
14     static void setupAll() {
15         System.out.println("BeforeAll");
16     }
17
18     @BeforeEach
19     void setup() {
20         System.out.println("BeforeEach");
21     }
22
23     @Test
24     @DisplayName("My test 1")
25     void testOne() {
26         System.out.println("TEST 1");
27     }
28 }
```

La désactivation de tests

- L'annotation `@org.junit.jupiter.api.Disabled` permet de désactiver un test.
- Il est possible de fournir une description optionnelle de la raison de la désactivation
- L'annotation `@Disabled` peut être utilisée sur une méthode ou sur une classe.
- L'utilisation sur une méthode désactive uniquement la méthode concernée.

```
@DisplayName("A special test case")
@Disabled("All test in this class will be skipped")
class DisabledTest {
```

```
@Test
@DisplayName("My test 1")
@Disabled
void testOne() {
    System.out.println("TEST 1");
}
```

Disposition de titre et de contenu avec liste

TP

01_StandardTest

Le cycle de vie des tests

```
9  class LifecycleJUnit5Test {
10
11  @BeforeAll
12  static void setupAll() {
13      System.out.println("BeforeAll");
14  }
15
16  @BeforeEach
17  void setup() {
18      System.out.println("BeforeEach");
19  }
20
21  @Test
22  void testOne() {
23      System.out.println("TEST 1");
24  }
25
26  @Test
27  void testTwo() {
28      System.out.println("TEST 2");
29  }
30
31  @AfterEach
32  void teardown() {
33      System.out.println("AfterEach");
34  }
35
36  @AfterAll
37  static void teardownAll() {
38      System.out.println("AfterAll");
39  }
40 }
```

Les assertions

- Les assertions ont pour rôle de faire des vérifications pour le test en cours.

Egalité	Nullité	Exceptions
assertEquals()	assertNull()	assertThrows()
assertNotEquals()	assertNotNull()	
assertTrue()		
assertFalse()		
assertSame()		
assertNotSame()		

Assertion standard

- L'assertion `assertAll`

```
@Test
void groupedAssertions() {
    Address address = new Address("Momadi", "Nassur");

    assertAll("address", () -> assertEquals("Momadi", address.getFirstName()),
              () -> assertEquals("Nassur", address.getLastName()));
}
```

- Les assertions `assertEquals` et `assertNotEquals`

```
assertEquals(2, 2);
assertNotEquals(2, 3);
```

- Les assertions `assertTrue` et `assertFalse`

```
assertTrue(true, "Un premier message");
assertFalse(false, () -> "Un " + "autre " + "message" + "." );
```

Assertion standard

- L'assertion `assertThrows`

```
@Test
void exceptionTesting() {
    Throwable e = assertThrows(IllegalArgumentException.class,
        () -> {
            throw new IllegalArgumentException("a message");
        });
    assertEquals("a message", e.getMessage());
}
```

- Les assertions `assertTimeout`

```
1 import static java.time.Duration.ofMillis;
2
3 class TimeoutExceededTest {
4
5     @Test
6     void timeoutNotExceeded() {
7         assertTimeout(ofMinutes(2), () -> {
8             // Perform task that takes less than 2 minutes
9         });
10    }
11
12    @Test
13    void timeoutExceeded() {
14        assertTimeout(ofMillis(10), () -> {
15            Thread.sleep(100);
16        });
17    }
18 }
19
20 }
```

L'utilisation d'assertions de bibliothèques tiers

- JUnit Jupiter propose un ensemble d'assertions qui peuvent suffire pour des tests simples
- Il est aussi possible d'utiliser d'autres bibliothèques d'assertions telles que :
 - AssertJ
 - Hamcrest
 - Truth
- Ces bibliothèques sont compatibles avec JUnit 5.
- JUnit 5 a fait le choix de ne pas fournir d'implémentation de l'assertion `assertThat()` qui attendait en paramètre un objet de type `Matcher` de la bibliothèque Hamcrest.
- JUnit 5 préfère laisser les développeurs utiliser ces bibliothèques tierces.

Hamcrest

- L'exemple ci-dessous utilise l'assertion `assertThat` de la bibliothèque Hamcrest.

```
import static org.hamcrest.CoreMatchers.containsString;
import static org.hamcrest.CoreMatchers.equalTo;
import static org.hamcrest.CoreMatchers.notNullValue;
import static org.hamcrest.MatcherAssert.assertThat;

import org.junit.jupiter.api.Test;

class HamcrestTest {

    @Test
    void assertWithHamcrestMatcher() {
        assertThat(2 + 1, equalTo(3));
        assertThat("Foo", notNullValue());
        assertThat("Hello world", containsString("world"));
    }
}
```

```
<dependency>
  <groupId>org.hamcrest</groupId>
  <artifactId>hamcrest-core</artifactId>
  <version>${hamcrest.version}</version>
  <scope>test</scope>
</dependency>
```

Disposition de titre et de contenu avec liste

TP

02_Assertion

Les tags

- Les classes et les méthodes de tests peuvent être tagguées pour permettre d'utiliser ces tags ultérieurement pour déterminer les tests à exécuter.
- Ils peuvent par exemple être utilisés pour créer différents scénarios de tests ou pour exécuter les tests uniquement sur des environnements dédiés.
 - Le libellé d'un tag ne doit pas :
 - être null ou une chaîne vide
 - contenir d'espace
 - contenir les caractères réservés : , () & | !

Exemple

```
4 import org.junit.jupiter.api.Tag;
5
6 @Tag("functional")
7 class FunctionalTest {
8
9     @Test
10     void testOne() {
11         System.out.println("Functional Test 1");
12     }
13
14     @Test
15     void testTwo() {
16         System.out.println("Functional Test 2");
17     }
18 }
```

```
@Tag("non-functional")
class NonFunctionalTest {

    @Test
    @Tag("performance")
    @Tag("load")
    void testOne() {
        System.out.println("Non-Functional Test 1 (Performance/Load)");
    }

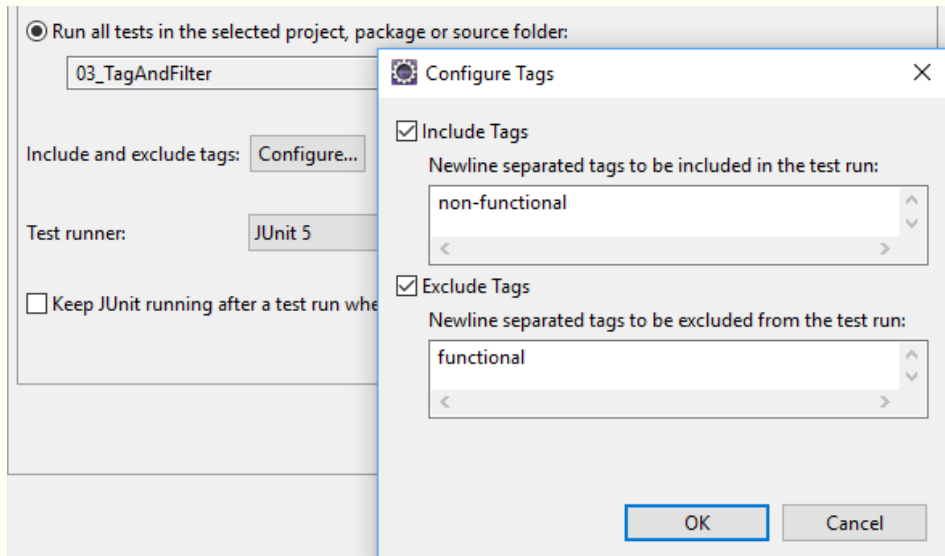
    @Test
    @Tag("performance")
    @Tag("stress")
    void testTwo() {
        System.out.println("Non-Functional Test 2 (Performance/Stress)");
    }

    @Test
    @Tag("security")
    void testThree() {
        System.out.println("Non-Functional Test 3 (Security)");
    }

    @Test
    @Tag("usability")
    void testFour() {
        System.out.println("Non-Functional Test 4 (Usability)");
    }
}
```

Exécution

- Exécuter dans Eclipse



- Click-droit sur le projet -> Run Configuration
- Select Run all test ... -> configure
- Inclure et/ exclure les tags

- Configurer le plugin maven-surefire-plugin

- Changer le pom.xml en ajoutant dans le plugin maven-surefire-plugin

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>${maven-surefire-plugin.version}</version>
      <dependencies>
        <dependency>
          <groupId>org.junit.platform</groupId>
          <artifactId>junit-platform-surefire-provider</artifactId>
          <version>${junit-platform.version}</version>
        </dependency>
      </dependencies>
    </plugin>
  </plugins>
</build>
```

Disposition de titre et de contenu avec liste

TP

03_TagAndFilter

Les tests imbriqués

- Les tests imbriqués permettent de grouper des cas de test pour renforcer le lien qui existent entre eux.
- JUnit 5 permet de créer des tests imbriquées (nested tests) en utilisant une annotation `@Nested` sur une classe interne.
- Seules les classes internes non statiques peuvent être annotées avec `@Nested`.

Les tests imbriqués : Exemple

```
class NestTest {  
  
    @BeforeEach  
    void setup1() {  
        System.out.println("Setup 1");  
    }  
  
    @Test  
    void topTest() {  
        System.out.println("Test 1");  
    }  
  
    @Nested  
    class InnerClass1 {  
        @BeforeEach  
        void setup2() {  
            System.out.println("Setup 2");  
        }  
  
        @Test  
        void innerTest1() {  
            System.out.println("Test 2");  
        }  
  
        @Nested  
        class InnerClass2 {  
            @Test  
            void innerTest2() {  
                System.out.println("Test 3");  
            }  
        }  
    }  
}
```

```
@DisplayName("A stack test")  
class StackTest {  
  
    @Test  
    @DisplayName("is instantiated")  
    void isInstantiated() {  
    }  
  
    @Nested  
    @DisplayName("when empty")  
    class WhenNew {  
  
        @Test  
        @DisplayName("is empty")  
        void isEmpty() {  
        }  
  
        @Test  
        @DisplayName("throws Exception when popped")  
        void throwsExceptionWhenPopped() {  
        }  
  
        @Test  
        @DisplayName("throws Exception when peeked")  
        void throwsExceptionWhenPeeked() {  
        }  
  
        @Nested  
        @DisplayName("after pushing an element")  
        class AfterPushing {  
  
            @Test  
            @DisplayName("it is no longer empty")  
            void isEmpty() {  
            }  
  
            @Test  
            @DisplayName("returns the element when popped")  
            void returnElementWhenPopped() {  
            }  
  
            @Test  
            @DisplayName("returns the element when peeked")  
            void returnElementWhenPeeked() {  
            }  
        }  
    }  
}
```


Disposition de titre et de contenu avec liste

TP

05_NestedAndRepeatTest

Les tests dynamiques

- Les tests dynamiques sont une nouvelle fonctionnalité de JUnit 5 qui permet la création dynamique de tests à l'exécution.
- Les tests standard définis avec l'annotation `@Test` sont statiques : l'intégralité de leur définition doit être fournie à la compilation.
- Les tests dynamiques sont des tests qui sont générés à l'exécution par une méthode de type fabrique qui est annotée avec `@TestFactory`.
- Les méthodes annotées avec `@TestFactory` ne sont donc pas des cas de tests mais des fabriques pour fournir un ensemble de cas de tests.
- Les tests dynamiques permettent par exemple d'obtenir les données requises par les cas de tests d'une source externe.

Les tests dynamiques

- Une méthode annotée avec `@TestFactory` ne peut pas être static ou private et peut retourner un objet de type :
 - `Stream<DynamicTest>`
 - `Collection<DynamicTest>`
 - `Iterable<DynamicTest>`
 - `Iterator<DynamicTest>`
- Si la méthode renvoie un objet d'un autre type alors une exception de type `JUnitException` est levée.

Les tests dynamiques

```
import java.util.stream.Stream;

import org.junit.jupiter.api.DynamicTest;
import org.junit.jupiter.api.TestFactory;

class DynamicExampleTest {

    @TestFactory
    Stream<DynamicTest> dynamicTestsFromStream() {
        Stream<String> inputStream = Stream.of("A", "B", "C");
        return inputStream.map(
            input -> dynamicTest("Display name for input " + input, () -> {
                System.out.println("Testing " + input);
            }));
    }
}
```

Les tests dynamiques

```
class CollectionTest {

    // Warning: this test will raise an exception
    @Disabled
    @TestFactory
    List<String> dynamicTestsWithInvalidReturnType() {
        return Arrays.asList("Hello");
    }

    @TestFactory
    Collection<DynamicTest> dynamicTestsFromCollection() {
        return Arrays.asList(
            dynamicTest("1st dynamic test", () -> assertTrue(true)),
            dynamicTest("2nd dynamic test", () -> assertEquals(4, 2 * 2)));
    }

    @TestFactory
    Iterable<DynamicTest> dynamicTestsFromIterable() {
        return Arrays.asList(
            dynamicTest("3rd dynamic test", () -> assertTrue(true)),
            dynamicTest("4th dynamic test", () -> assertEquals(4, 2 * 2)));
    }

    @TestFactory
    Iterator<DynamicTest> dynamicTestsFromIterator() {
        return Arrays.asList(
            dynamicTest("5th dynamic test", () -> assertTrue(true)),
            dynamicTest("6th dynamic test", () -> assertEquals(4, 2 * 2)))
        .iterator();
    }
}
```

Disposition de titre et de contenu avec liste

TP

06_DynamicTest

Les tests répétés

- JUnit Jupiter permet une exécution répétée un certain nombre de fois d'une méthode de test en l'annotant avec `@RepeatedTest`.
- L'annotation `@RepeatedTest` possède deux attributs :

Attribut	Rôle
<code>int value</code>	Le nombre de répétitions à opérer. Obligatoire
<code>String name</code>	Le libellé de chaque test exécuté

- Une méthode annotée avec `@RepeatedTest` doit respecter plusieurs contraintes :
 - ne pas être private
 - ne pas être static
 - doit obligatoirement retourner void

```
. @DisplayName("test addition répété")
. @RepeatedTest(3)
. void testRepete() {
.     Assertions.assertEquals(2, 1 + 1, "Valeur obtenue erronée");
. }
```

L'injection d'instances dans les constructeurs et les méthodes de tests

- Dans les versions antérieures à la version 5 de JUnit, les constructeurs et les méthodes de test des classes de tests ne pouvaient pas avoir de paramètres pour être exécutées par le Runner standard.
- JUnit 5 permet de mettre en œuvre l'injection de dépendances via des paramètres pour les constructeurs et les méthodes des classes de tests.

```
class TestInfoTest {  
    @BeforeEach  
    void init(TestInfo testInfo) {  
        String displayName = testInfo.getDisplayName();  
        System.out.printf("@BeforeEach %s %n", displayName);  
    }  
  
    @Test  
    @DisplayName("My test")  
    @Tag("my-tag")  
    void testOne(TestInfo testInfo) {  
        System.out.println(testInfo.getDisplayName());  
        System.out.println(testInfo.getTags());  
        System.out.println(testInfo.getTestClass());  
        System.out.println(testInfo.getTestMethod());  
    }  
  
    @Test  
    void testTwo() { }  
}
```

```
class RepetitionInfoTest {  
    @RepeatedTest(2)  
    void test(RepetitionInfo repetitionInfo) {  
        System.out.println("*** Test " + repetitionInfo.getCurrentRepetition()  
            + "/" + repetitionInfo.getTotalRepetitions());  
    }  
}
```


Disposition de titre et de contenu avec liste

TP

07_DependencyInjection



MODULE 4

Selenium

Qu'est-ce que Selenium WebDriver?

- **Selenium** est un ensemble d'outils conçus pour automatiser les navigateurs.
- Il est couramment utilisé pour les tests d'applications Web sur plusieurs plates-formes.
- Quelques outils sont disponibles sous le parapluie Selenium, tels que :
 - **Selenium WebDriver**
 - **Selenium IDE**
 - **Selenium Grid.**

Qu'est-ce que Selenium WebDriver?

- **WebDriver** est une interface de contrôle à distance qui vous permet de manipuler des éléments **DOM** dans des pages Web et de contrôler le comportement des agents utilisateurs.
- Cette interface fournit un **protocole de connexion indépendant du langage** qui a été implémenté pour diverses plates-formes telles que:
 - **GeckoDriver** (Mozilla Firefox)
 - **ChromeDriver** (Google Chrome)
 - **SafariDriver** (Apple Safari)
 - **InternetExplorerDriver** (MS InternetExplorer)
 - **MicrosoftWebDriver** ou **EdgeDriver** (MS Edge)
 - **OperaChromiumDriver** (navigateur Opera)

Installation ou configuration pour Java






- Pour écrire des tests en utilisant Selenium Webdriver et Java comme langage de programmation, vous devez télécharger les fichiers JAR de Selenium Webdriver sur le site Web de Selenium.
- Il existe plusieurs façons de configurer un projet Java pour le webdriver Selenium, l'un des plus faciles à utiliser est d'utiliser Maven.
- Maven télécharge les liaisons Java requises pour Selenium webdriver, y compris toutes les dépendances.
- L'autre méthode consiste à télécharger les fichiers JAR et à les importer dans votre projet.

Installation ou configuration pour Java

- Étapes pour configurer le projet Selenium Webdriver à l'aide de Maven:

```
<dependencies>
  <dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>3.14.0</version>
  </dependency>
</dependencies>
```

- Télécharger le driver Chrome:
 - <https://sites.google.com/a/chromium.org/chromedriver/>

	Name	Last modified	Size	ETag
	Parent Directory		-	
	chromedriver_linux64.zip	2019-03-12 19:25:26	4.83MB	3cd9e67808926bfba9a3f5946e2a994d
	chromedriver_mac64.zip	2019-03-12 19:25:27	6.69MB	de2aa78283af413100cddc2a4dee3ebc
	chromedriver_win32.zip	2019-03-12 19:25:29	4.41MB	9780b9b586e74253df9b58928b959861
	notes.txt	2019-03-14 18:17:49	0.00MB	d6180d1b525cf857b030077a525a9f47

M.Mbengue

Installation ou configuration pour Java

- Exemple:

```
package com.formation;

import org.openqa.selenium.WebDriver;

public class BrowserCheck {

    public static void main(String[] args) throws InterruptedException {

        // Indiquer le chemin vers le driver chrome

        String chromeDriverPath = "D:\\space\\selenium\\chromedriver.exe";
        System.setProperty("webdriver.chrome.driver", chromeDriverPath);

        // Ouvrir le navigateur chrome

        WebDriver webDriver = new ChromeDriver();

        Thread.sleep(5000);

        webDriver.quit();
    }
}
```

Disposition de titre et de contenu avec liste

TP

01_GettingStarted

WebDriver : La navigation

Naviguer () [Java]

Pour naviguer dans n'importe quelle URL:

```
driver.navigate().to("http://www.example.com");
```

Pour reculer:

```
driver.navigate().back();
```

Pour aller de l'avant:

```
driver.navigate().forward();
```

Pour rafraîchir la page:

```
driver.navigate().refresh();
```

WebDriver : Fenêtre

- Définir ou obtenir la taille de la fenêtre de n'importe quel navigateur lors de l'automatisation
- **Syntaxe**
 - `driver.manage (). window (). maxim ()`;
 - `driver.manage (). window (). setSize (DimensionObject)`;
 - `driver.manage (). window (). getSize ()`

WebDriver : Fenêtre

▪ Examples

Définir à la taille maximale de la fenêtre du navigateur:

```
//Initialize Browser
System.setProperty("webdriver.gecko.driver", "E:\\path\\to\\geckodriver.exe");
WebDriver driver = new FirefoxDriver();
driver.get("https://www.google.com/");

//Set Browser window size
driver.manage().window().maximize();
```

Définir la taille de la fenêtre spécifique:

```
//Initialize Browser
System.setProperty("webdriver.gecko.driver", "E:\\path\\to\\geckodriver.exe");
WebDriver driver = new FirefoxDriver();
driver.get("https://www.google.com/");

//Initialize Dimension class object and set Browser window size
org.openqa.selenium.Dimension d = new org.openqa.selenium.Dimension(400, 500);
driver.manage().window().setSize(d);
```

WebDriver : Fenêtre

- Examples

Obtenir la taille de la fenêtre du navigateur:

```
//Initialize Browser
System.setProperty("webdriver.gecko.driver", "E:\\path\\to\\geckodriver.exe");
WebDriver driver = new FirefoxDriver();
driver.get("https://www.google.com/");

//Get Browser window size and print on console
System.out.println(driver.manage().window().getSize());
```

WebDriver : Gérer une alerte

Il existe 3 types de popups.

1. **Alerte simple** : `alerte ("Ceci est une alerte simple");`
 2. **Alerte de confirmation** : `var popuResult = confirmer ("Confirmer avec les boutons OK et Annuler");`
 3. **Alerte d'invite** : `var person = prompt ("Aimez-vous stackoverflow?", "Oui / Non");`
- Son utilisateur doit savoir quel type de popup doit être manipulé dans son test élémentaire.
 - Soit vous pouvez
 1. **accepter()** accepter l'alerte
 2. **rejeter()** rejeter l'alerte
 3. **getText()** Pour obtenir le texte de l'alerte
 4. **sendKeys()** Pour écrire du texte dans l'alerte

WebDriver : Gérer une alerte

Pour une alerte simple:

```
Alert simpleAlert = driver.switchTo().alert();
String alertText = simpleAlert.getText();
System.out.println("Alert text is " + alertText);
simpleAlert.accept();
```

Pour une alerte de confirmation:

```
Alert confirmationAlert = driver.switchTo().alert();
String alertText = confirmationAlert.getText();
System.out.println("Alert text is " + alertText);
confirmationAlert.dismiss();
```

Pour une alerte rapide:

```
Alert promptAlert = driver.switchTo().alert();
String alertText = promptAlert.getText();
System.out.println("Alert text is " + alertText);
//Send some text to the alert
promptAlert.sendKeys("Accepting the alert");
Thread.sleep(4000); //This sleep is not necessary, just for demonstration
promptAlert.accept();
```

WebDriver : Changement de cadre

Java

- `driver.switchTo (). frame (nom de la chaîne);`
- `driver.switchTo (). frame (identifiant de chaîne);`
- `driver.switchTo (). frame (int index);`
- `driver.switchTo (). frame (WebElement frameElement);`
- `driver.switchTo (). defaultContent ();`

Paramètres

paramètre	détails
nameOrId	Sélectionnez un cadre par son nom.
indice	Sélectionnez un cadre par son index de base.
frameElement	Sélectionnez un cadre en utilisant son WebElement précédemment localisé

WebDriver : Changement de cadre

- **Pour basculer vers un cadre en utilisant Java**
- Pour une instance, si le code source html d'une vue ou d'un élément html est enveloppé par un iframe comme celui-ci:

```
<iframe src="../../../images/eightball.gif" name="imgboxName" id="imgboxId">  
  <p>iframes example</p>  
  <a href="../../../images/redball.gif" target="imgbox">Red Ball</a>  
</iframe><br />
```

- ... alors pour effectuer une action quelconque sur les éléments web de l'iframe, vous devez d'abord basculer le focus sur l'iframe, en utilisant l'une des méthodes ci-dessous:

WebDriver : Changement de cadre

Utiliser un identifiant de frame (doit être utilisé uniquement si vous connaissez l'id de l'iframe).

```
driver.switchTo().frame("imgboxId"); //imgboxId - Id of the frame
```

Utiliser le nom du cadre (doit être utilisé uniquement si vous connaissez le nom de l'iframe).

```
driver.switchTo().frame("imgboxName"); //imgboxName - Name of the frame
```

Utilisation de l'index d'images (doit être utilisé uniquement si vous ne possédez pas l'id ou le nom de l'iframe), où l'index définit la position de l'iframe parmi toutes les images.

```
driver.switchTo().frame(0); //0 - Index of the frame
```

WebDriver : Changement de cadre

Pour sortir d'un cadre en Java

Pour basculer le focus sur le document principal ou la première image de la page. Vous devez utiliser la syntaxe ci-dessous.

```
driver.switchTo().defaultContent();
```

WebDriver : Gestion de la fenêtre active

- Nous pouvons obtenir l'URL actuelle de la fenêtre active:

```
Path sampleFile = Paths.get("pages/window.html");  
driver.get(sampleFile.toUri().toString());
```

- Nous pouvons obtenir le handle pour la fenêtre en cours:

```
String parentWindowHandle = driver.getWindowHandle();
```

Disposition de titre et de contenu avec liste

TP

02_WebDriver

WebElement : Localisation d'éléments Web

- Les objets se trouvent dans Selenium grâce à l'utilisation de *localisateurs* et de la classe **By** .
- Afin de réaliser un projet d'automatisation robuste avec Selenium, il convient d'utiliser intelligemment les localisateurs pour Web Elements.
- Les localisateurs doivent être **descriptifs, uniques et peu susceptibles de changer** . Par exemple, vous n'obtiendrez pas de faux positifs dans les tests.
- La priorité est d'utiliser:
 1. **ID** - puisqu'il est unique et que vous obtenez exactement l'élément souhaité.
 2. **Nom de classe** - Il est descriptif et peut être unique dans un contexte donné.
 3. **CSS** (meilleures performances que xpath) - Pour les sélecteurs plus compliqués.
 4. **XPATH** - Où CSS ne peut pas être utilisé (axe XPATH), par exemple **div::parent**

WebElement : Localisation d'éléments Web

Localisation des éléments de page à l'aide de WebDriver

Pour interagir avec WebElements dans une page Web, nous devons d'abord identifier l'emplacement de l'élément.

■ Vous pouvez localiser les éléments par ..

1. **Par** *identifiant*
2. **Par** *nom de classe*
3. **Par** *tagName*
4. **Par** *nom*
5. **Par** *lien texte*
6. **Par** *sélecteur CSS*
7. **Par** *XPath*
8. **Utiliser** *JavaScript*

WebElement : Localisation d'éléments Web

Par identifiant

Exemple de recherche d'un élément à l'aide de l'ID:

```
<div id="coolestWidgetEvah">...</div>
```

```
Java      - WebElement element = driver.findElement(By.id("coolestWidgetEvah"));
C#        - IWebElement element = driver.FindElement(By.Id("coolestWidgetEvah"));
Python    - element = driver.find_element_by_id("coolestWidgetEvah")
Ruby      - element = driver.find_element(:id, "coolestWidgetEvah")
JavaScript/Protractor - var elm = element(by.id("coolestWidgetEvah"));
```

WebElement : Localisation d'éléments Web

- Par nom de classe

Exemple de recherche d'un élément en utilisant le nom de classe:

```
<div class="cheese"><span>Cheddar</span></div>
```

```
Java      - WebElement element = driver.findElement(By.className("cheese"));
C#        - IWebElement element = driver.FindElement(By.ClassName("cheese"));
Python    - element = driver.find_element_by_class_name("cheese")
Ruby      - cheeses = driver.find_elements(:class, "cheese")
JavaScript/Protractor - var elm = element(by.className("cheese"));
```


WebElement : Localisation d'éléments Web

Par nom de tag

Exemple de recherche d'un élément en utilisant le nom de tag:

```
<iframe src="..."></iframe>
```

```
Java      - WebElement element = driver.findElement(By.tagName("iframe"));
C#        - IWebElement element = driver.FindElement(By.TagName("iframe"));
Python    - element = driver.find_element_by_tag_name("iframe")
Ruby      - frame = driver.find_element(:tag_name, "iframe")
JavaScript/Protractor - var elm = element(by.tagName("iframe"));
```

WebElement : Localisation d'éléments Web

De nom

Exemple de recherche d'un élément en utilisant name:

```
<input name="cheese" type="text"/>
```

```
Java      - WebElement element = driver.findElement(By.name("cheese"));
C#        - IWebElement element = driver.FindElement(By.Name("cheese"));
Python    - element = driver.find_element_by_name("cheese")
Ruby      - cheese = driver.find_element(:name, "cheese")
JavaScript/Protractor - var elm = element(by.name("cheese"));
```

WebElement : Localisation d'éléments Web

Par lien texte

Exemple de recherche d'un élément à l'aide du texte du lien:

```
<a href="http://www.google.com/search?q=cheese">cheese</a>>
```

```
Java      - WebElement element = driver.findElement(By.linkText("cheese"));
C#        - IWebElement element = driver.FindElement(By.LinkText("cheese"));
Python    - element = driver.find_element_by_link_text("cheese")
Ruby      - cheese = driver.find_element(:link, "cheese")
JavaScript/Protractor - var elm = element(by.linkText("cheese"));
```

WebElement : Localisation d'éléments Web

Par sélecteurs CSS

Exemple de recherche d'un élément à l'aide des sélecteurs CSS:

```
<div id="food" class="dairy">milk</span>
```

```
Java      - WebElement element = driver.findElement(By.cssSelector("#food.dairy")); //# is  
used to indicate id and . is used for classname.
```

```
C#        - IWebElement element = driver.FindElement(By.CssSelector("#food.dairy"));
```

```
Python    - element = driver.find_element_by_css_selector("#food.dairy")
```

```
Ruby      - cheese = driver.find_element(:css, "#food span.dairy.aged")
```

```
JavaScript/Protractor - var elm = element(by.css("#food.dairy"));
```

WebElement : Localisation d'éléments Web

Par XPath

Exemple de recherche d'un élément à l'aide de XPath:

```
<input type="text" name="example" />
```

```
Java      - WebElement element = driver.findElement(By.xpath("//input"));
C#        - IWebElement element = driver.FindElement(By.XPath("//input"));
Python    - element = driver.find_element_by_xpath("//input")
Ruby      - inputs = driver.find_elements(:xpath, "//input")
JavaScript/Protractor - var elm = element(by.xpath("//input"));
```

WebElement : Localisation d'éléments Web

- **Utiliser JavaScript**
- Vous pouvez exécuter un javascript arbitraire pour trouver un élément et tant que vous retournez un élément DOM, il sera automatiquement converti en objet WebElement.
- Exemple simple sur une page chargée en jQuery:

```
Java      -  WebElement element = (WebElement)
             ((JavascriptExecutor)driver).executeScript("return $(' .cheese')[0]");

C#        -  IWebElement element = (IWebElement)
             ((IJavaScriptExecutor)driver).ExecuteScript("return $(' .cheese')[0]");

Python    -  element = driver.execute_script("return $(' .cheese')[0]");
Ruby      -  element = driver.execute_script("return $(' .cheese')[0]");
JavaScript/Protractor -
```

Disposition de titre et de contenu avec liste

TP

03_WebElement

WaitingElement: Types d'attentes

- Lors de l'exécution d'une application Web, il est nécessaire de prendre en compte le temps de chargement.
- Si votre code tente d'accéder à un élément qui n'est pas encore chargé, WebDriver lancera une exception et votre script s'arrêtera.
- Il existe trois types de Waits
 - **Attentes implicites**
 - **Attentes explicites**
 - **Attentes Courantes**
- Les attentes implicites sont utilisées pour définir le temps d'attente tout au long du programme, tandis que les attentes explicites ne sont utilisées que sur des parties spécifiques.

WaitingElement: Attente implicite

- Une attente implicite consiste à demander à WebDriver d'interroger le DOM pendant un certain temps lorsqu'il tente de trouver un élément ou des éléments s'ils ne sont pas immédiatement disponibles.
- Les attentes implicites sont essentiellement votre façon de dire à WebDriver la latence que vous souhaitez voir si l'élément Web spécifié n'est pas présent.
- Le paramètre par défaut est 0.
- Une fois définie, l'attente implicite est définie pour la durée de vie de l'instance de l'objet WebDriver.

WaitingElement: Attente implicite

- L'attente implicite est déclarée dans la partie instantiation du code à l'aide de l'extrait de code suivant.

Exemple en **Java** :

```
driver.manage().timeouts().implicitlyWait(15, TimeUnit.SECONDS);  
// You need to import the following class - import java.util.concurrent.TimeUnit;
```

WaitingElement: Attente explicite

- Vous pouvez rencontrer des cas où un élément prend plus de temps à charger.
- Définir l'attente implicite pour de tels cas n'a pas de sens car le navigateur attendra inutilement le même temps pour chaque élément, augmentant ainsi le temps d'automatisation.
- L'attente explicite aide ici en contournant l'attente implicite pour certains éléments spécifiques.
- Les attentes explicites sont des attentes intelligentes limitées à un élément Web particulier.

WaitingElement: Attente explicite

- En utilisant des attentes explicites, vous indiquez à WebDriver au maximum qu'il faut attendre X unités de temps avant d'abandonner.
- Les attentes explicites sont effectuées à l'aide des classes **WebDriverWait** et **ExpectedConditions**.
- .

WaitingElement: Attente explicite

- Dans l'exemple ci-dessous, nous allons attendre jusqu'à 10 secondes pour qu'un élément dont l'identifiant est un nom d'utilisateur soit visible avant de passer à la commande suivante

Exemple en **Java** :

```
//Import these two packages:
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

//Declare a WebDriverWait variable. In this example, we will use myWaitVar as the name of the
variable.
WebDriverWait myWaitVar = new WebDriverWait(driver, 30);

//Use myWaitVar with ExpectedConditions on portions where you need the explicit wait to occur.
In this case, we will use explicit wait on the username input before we type the text tutorial
onto it.
myWaitVar.until(ExpectedConditions.visibilityOfElementLocated(By.id("username")));
driver.findElement(By.id("username")).sendKeys("tutorial");
```

WaitingElement: Attentes Courantes

- Contrairement à l'attente implicite et explicite, l'attente fluide utilise deux paramètres.
- **Valeur de temporisation et fréquence d'interrogation.**
- Disons que nous avons une valeur de délai d'attente de 30 secondes et une fréquence d'interrogation de 2 secondes. WebDriver vérifie l'élément après toutes les 2 secondes jusqu'à la valeur du délai d'attente (30 secondes).
- Une fois que la valeur du délai d'expiration est dépassée sans aucun résultat, une exception est levée.

WaitingElement: Attentes Courantes

Exemple en **Java** :

```
Wait wait = new FluentWait(driver).withTimeout(30, SECONDS).pollingEvery(2,
SECONDS).ignoring(NoSuchElementException.class);

WebElement testElement = wait.until(new Function() {
    public WebElement apply(WebDriver driver) {
        return driver.findElement(By.id("testId"));
    }
});
```

- Un autre avantage d'utiliser l'attente fluide est que nous pouvons ignorer des types spécifiques d'exceptions (par ex. NoSuchElementExceptions) en attendant.

Disposition de titre et de contenu avec liste

TP

05_WaitingElement

Modèle d'objet de page

- Un rôle important dans l'automatisation des sites Web et des applications Web consiste à identifier les éléments à l'écran et à interagir avec eux.
- Les objets se trouvent dans Selenium grâce à l'utilisation de localisateurs et de la classe By .
- Ces localisateurs et interactions sont placés dans les objets de la page pour éviter le code en double et faciliter la maintenance.
- Il encapsule les WebElements et suppose de contenir le comportement et de renvoyer des informations sur la page (ou une partie d'une page dans une application Web).

Modèle d'objet de page

- Le modèle d'objet de page est un modèle où l'on écrit des classes orientées objet qui servent d'interface à une vue particulière de la page Web.
- Nous utilisons les méthodes de cette classe de page pour effectuer l'action requise.
- Le fait d'avoir votre code organisé de la même manière que le modèle d'objets de page fournit une API spécifique à l'application, vous permettant de manipuler les éléments de la page sans contourner le HTML.
- Le principe de base de la page dit: votre objet page doit avoir tout ce qu'un humain peut faire sur cette page Web.
- **Par exemple**, pour accéder au champ de texte d'une page Web, vous devez utiliser une méthode pour obtenir le texte et retourner la chaîne après avoir effectué toutes les modifications.

Modèle d'objet de page

Avantages du modèle d'objet de page:

1. Séparation propre entre le code de test et le code de page
2. En cas de modification de l'interface utilisateur de la page Web, il n'est pas nécessaire de modifier votre code à plusieurs endroits. Changer uniquement dans les classes de page.
3. Pas de localisateur d'éléments dispersés.
4. Facilite la compréhension du code
5. Entretien facile

Modèle d'objet de page : Exemples

```
public class AgeCalculatorPage {
    //WebElements
    private WebElement dayOfBirth = null;
    private WebElement monthOfBirth = null;
    private WebElement yearOfBirth = null;
    private WebElement age = null;
    private WebElement zodiacSign = null;
    private WebElement calculate = null;

    //WebDriver
    private WebDriver driver;
    Path sampleFile = Paths.get("pages/exercise_6_1.html");

    private String url = sampleFile.toUri().toString();

    //Class Constructor
    public AgeCalculatorPage(WebDriver webDriver) {
        driver = webDriver;
    }

    //Methods to open and close the WebDriver
    public void open() {
        this.driver.get(url);
    }
    public void close() {
        this.driver.quit();
    }

    //Method to execute the test
    public void calculate(String day, String month, String year) {
        getDayOfBirth().sendKeys(day);
        getMonthOfBirth().sendKeys(month);
        getYearOfBirth().sendKeys(year);
        getCalculate().click();
    }
}
```

```
//Methods to read values from required WebElements
public String getAge() {
    age = driver.findElement(By.id("age"));
    return age.getText();
}

public String getZodiacSign() {
    zodiacSign = driver.findElement(By.id("zodiacSign"));
    return zodiacSign.getText();
}

public WebElement getDayOfBirth() {
    dayOfBirth = driver.findElement(By.id("dayOfBirth"));
    return dayOfBirth;
}

public WebElement getMonthOfBirth() {
    monthOfBirth = driver.findElement(By.id("monthOfBirth"));
    return monthOfBirth;
}

public WebElement getYearOfBirth() {
    yearOfBirth = driver.findElement(By.id("yearOfBirth"));
    return yearOfBirth;
}

public WebElement getCalculate() {
    calculate = driver.findElement(By.id("calculate"));
    return calculate;
}
```

Modèle d'objet de page : Exemples

```
public class AgeCalculatorScript {  
    public static void main(String[] args) throws Exception {  
        checkAgeCalculator();  
    }  
  
    private static void checkAgeCalculator() throws Exception {  
        WebDriver driver = new ChromeDriver();  
        // Create an instance of AgeCalculatorPage class and open it  
        AgeCalculatorPage ageCalculatorPage = new AgeCalculatorPage(driver);  
        ageCalculatorPage.open();  
  
        // Start the test by means of the calculate method  
        ageCalculatorPage.calculate("11", "February", "1982");  
  
        // Verify results  
        if (ageCalculatorPage.getAge().equals("36")) {  
            System.out.println("Age was calculated correctly!");  
        } else {  
            System.out.println("There was an error in the age calculation");  
        }  
  
        if (ageCalculatorPage.getZodiacSign().equals("Aquarius")) {  
            System.out.println("Zodiac sign was calculated correctly!");  
        } else {  
            System.out.println("There was an error in the zodiac sign calculation");  
        }  
  
        ageCalculatorPage.close();  
    }  
}
```

Disposition de titre et de contenu avec liste

TP

06_PageObjetModel

xxx
