



SELENIUM

2 jours





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

Un 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**

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

Validation

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

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

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

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.

Disposition de titre et de contenu avec liste

Question ?



MODULE 2

Introduction à la programmation JAVA

Sommaire

- ❖ **Qu'est-ce que JAVA ?**
- ❖ **La notion d'objet**
- ❖ **L'environnement de développement**
- ❖ **Les classes**
 - Les variables d'instances
 - Les types de données
 - Le constructeur
- ❖ **Les méthodes**
 - Les paramètres
 - Les types de retour
 - Les instructions
 - Les modificateurs d'accessibilité
 - Les accesseurs
- ❖ **Ecrire des tests en JAVA**
- ❖ **Concepts de la programmation orientée objet**
 - Encapsulation
 - Héritage
 - Interface
 - Polymorphisme
 - Enumération

Sommaire

❖ Sélections et décisions

- if... else
- switch...case

❖ Tableaux et boucles

- tableaux
- for
- while

❖ Collections

- Les types de Collection
- Typage des éléments d'une collection
- Parcours d'une collection

❖ Exceptions

Qu'est-ce que Java?

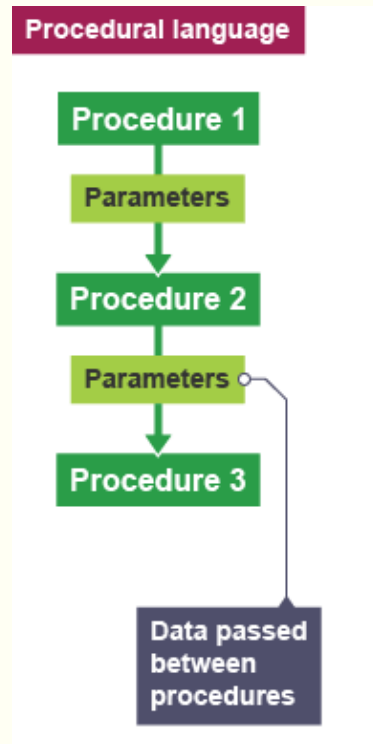
- Java est un langage de programmation mis au point par Sun Microsystems dans les années 90.
- La première version sort en 1995

Qu'est-ce que Java?

- Un langage de programmation objet basé sur la notion de classe (les objets sont décrits/regroupés dans des classes)
- Langage sûr, parce que fortement « typé »
 - Mots réservés (class, package, if, else, while, int, void, ...)
 - Syntaxe (nom_type nom_variable; => déclaration de variable, par ex: int a;)
 - Normes de codage et de nommage
- Java fournit également une librairie standard de classes (API)
- Les applications Java ne sont pas compilées en langage machine mais en bytecode. Ce bytecode nécessite un programme spécial appelé (JVM – Java Virtual Machine) pour être interprété.

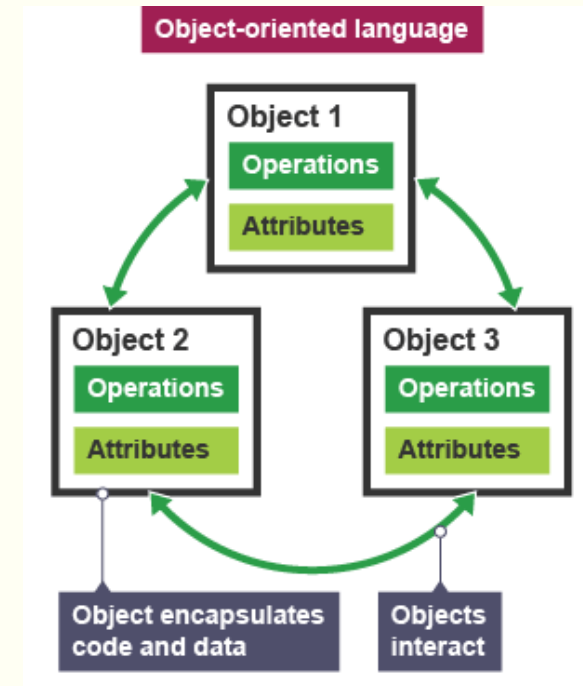
Vous avez dit « orienté objet » ?

Programmation procédurale



VS

Programmation orientée objet



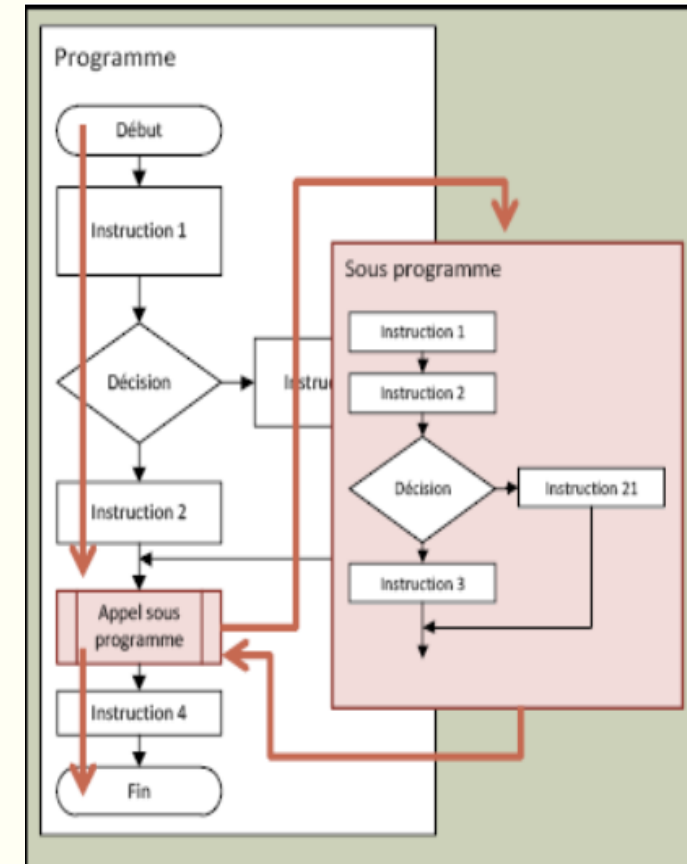
Vous avez dit « orienté objet » ?

- Une suite d'instructions s'exécutant les unes après les autres
- Avec des procédures ou des fonctions (sous-programmes)

Cette approche a permis de décomposer les fonctionnalités d'un programme en procédures qui s'exécutent séquentiellement

Inconvénients

- Le **procédural** est très éloigné de notre manière de penser
- Le **développeur doit penser de manière algorithmique** (proche du langage de la machine).
- Le **code** est **peu lisible** => Et **difficile** à **modifier**, à **maintenir**
- L'**ajout** de fonctionnalités **difficile** => Réutilisation du code incertaine
- Le **travail d'équipe** est **délicat**



Vous avez dit « orienté objet » ?

- Finalement, il est peut-être plus simple de **s'inspirer du monde réel**
 - **Le monde réel est composé d'objets**, d'êtres vivants, de matière
 - Pourquoi ne pas programmer de manière plus réaliste ?
- **Les objets ont des propriétés, des attributs**
 - Un chat a 4 pattes, un serpent aucune
- **Les objets ont une utilité**, une ou plusieurs fonctions = **des méthodes**
 - Une voiture permet de se déplacer

Alors, plutôt que de focaliser sur les procédures, intéressons-nous d'avantage aux données

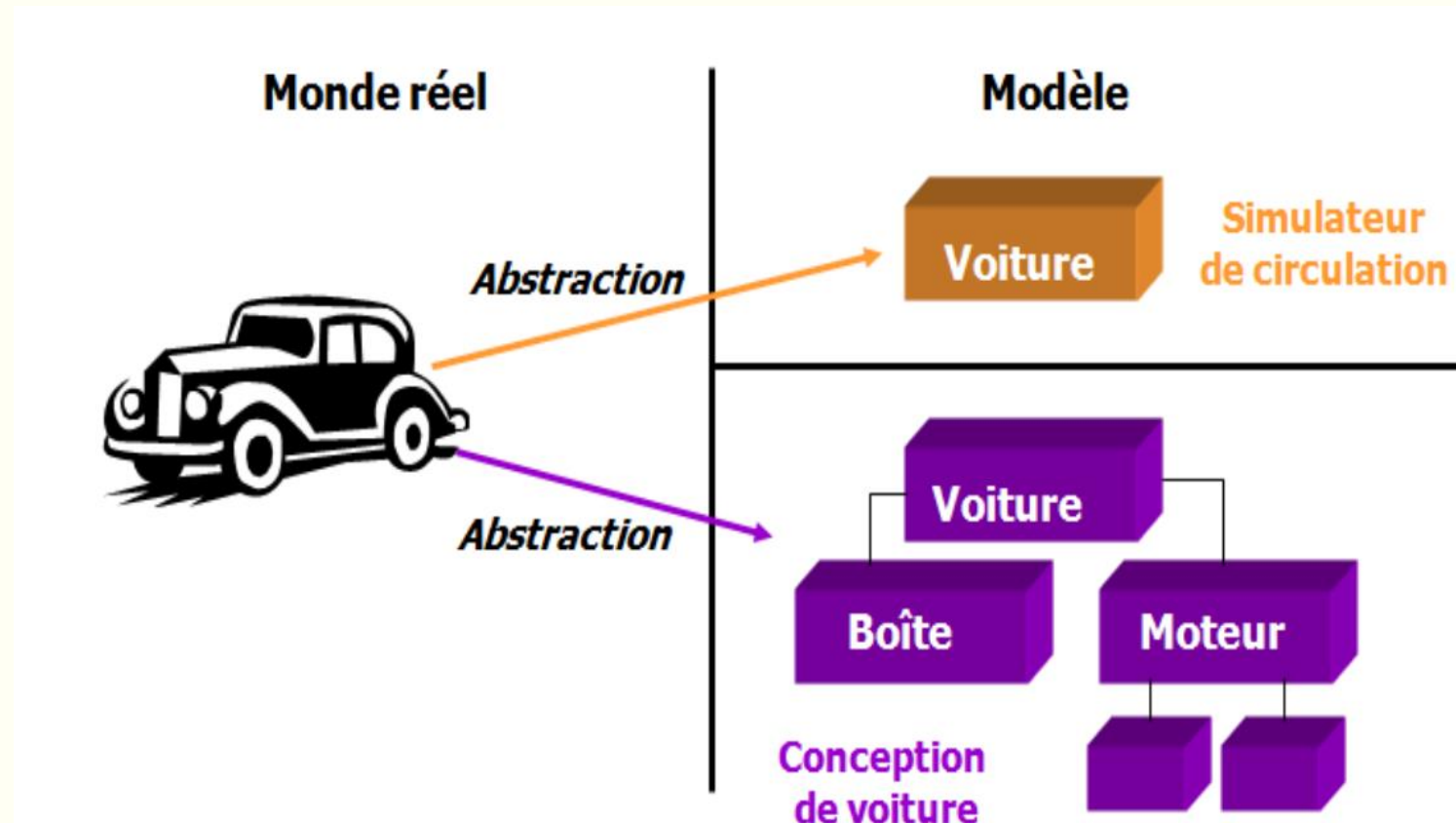
Vous avez dit « orienté objet » ?

- Finalement, il est peut-être plus simple de **s'inspirer du monde réel**
 - **Le monde réel est composé d'objets**, d'êtres vivants, de matière
 - Pourquoi ne pas programmer de manière plus réaliste ?
- **Les objets ont des propriétés, des attributs**
 - Un chat a 4 pattes, un serpent aucune
- **Les objets ont une utilité**, une ou plusieurs fonctions = **des méthodes**
 - Une voiture permet de se déplacer

Alors, plutôt que de focaliser sur les procédures, intéressons-nous d'avantage aux données

Qu'est-ce qu'un objet ?

Un objet est l'abstraction d'une entité du monde réel

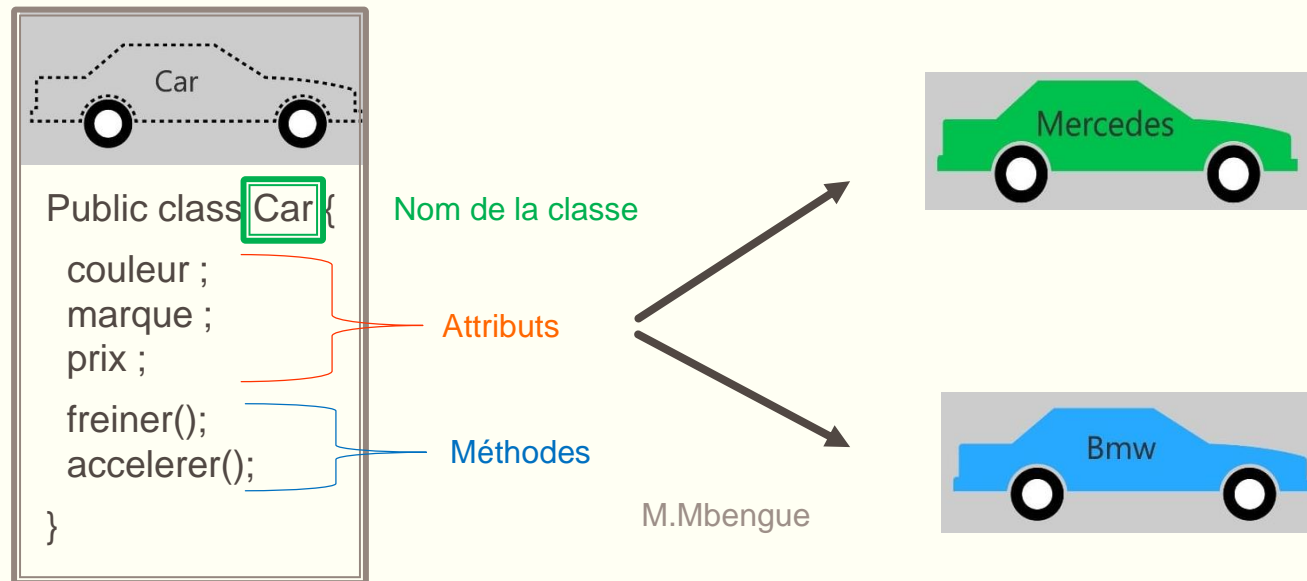


Qu'est-ce qu'un objet ?

- **Un élément qui modélise toute entité, concrète ou abstraite, manipulée par le logiciel...**
 - un élément avec un état interne donné par des valeurs **d'attributs** appelées aussi « **variables internes** »
 - *Exemple : L'objet voiture a plusieurs attributs : 4 roues, un volant, un moteur, etc*
- **Un élément qui réagit à certains messages qu'on lui envoie de l'extérieur**
 - C'est son comportement, ce qu'il sait faire : **on parlera de méthodes**
 - *Exemple : Avec cette voiture, je peux tourner, accélérer, freiner, ...*
- **Un élément qui ne réagit pas toujours de la même manière**
 - Son comportement dépend de l'état dans lequel il se trouve. **On parlera d'instance**
 - *Exemple : Essayez de démarrer sans essence !!?*

Des classes aux objets

- Avant de créer des objets, il faut définir un modèle
- Des objets pourront être créés à partir de ce modèle
- Ce modèle s'appelle **une classe**
- Les objets fabriqués à partir du modèle sont **des instances**



Notion de classe

→ Un programme Java est composé de classes (mot clé: **class**)

→ Tout le code permettant d'exécuter des instructions se trouve dans des *classes*

Nom du package

{

```
package fr.eql.autom.java;
```

Corps de classe avec

- (des propriétés)
- (des méthodes)

{

```
class PremiereClasse {  
  
}
```

En-tête de classe avec

- le mot réservé class
- le nom de la classe
- (un modifieur d'accessibilité)
- (des annotations)

- Le nom d'une *classe* débute par une majuscule et est écrit en **CamelCase**
- Une *classe* appartient à un **package**. Les *packages* permettent de ranger les *classes* (comme une suite de dossiers)
- Le nom d'un package est une suite de caractères en minuscule séparés par des points « . »

Notion de méthode

corps de méthode avec :
• (des instructions)

```
package fr.eql.autom.java;  
  
class DeuxiemeClasse {  
    public void disBonjour() {  
        System.out.println("Bonjour!");  
    }  
}
```

En-tête de méthode avec :

- (un modifieur d'accessibilité)
- un type de résultat/retour
- un nom
- des () contenant les paramètres séparés par des virgules

Instruction
(ici un appel de la méthode
println(String arg0) : void)

Exercice

L'IDE Eclipse

1/ Ouvrir Eclipse

2/ Créer un projet Maven

3/ Ouvrir le fichier pom.xml

Apache Maven – A quoi sert Maven?

Maven est un outil qui permet de gérer la production des logiciels Java ... de l'écriture du code au déploiement d'un exécutable.

- Il introduit des conventions dans la structure du projet Java
 - src/main
 - src/test
 - src/main/java
 - src/test/java
 - src/main/ressources
 - src/test/ressources
 - target
- Il permet de gérer les dépendances de code
- Il introduit des conventions dans la structure du projet Java
- Il permet de gérer les dépôts de dépendances
- Il définit un cycle de vie logiciel
 - Compile
 - Test
 - Package
 - Install
 - deploy

Maven – pom.xml d'un projet Selenium WebDriver / JUnit

POM = Project Object Model

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>fr.eql.autom</groupId>
  <artifactId>simple</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
    </dependency>
  </dependencies>
</project>
```

JUnit est un framework de tests unitaires standard en Java

Parallèlement à notre développement java, nous créerons des tests pour vérifier la qualité de notre code

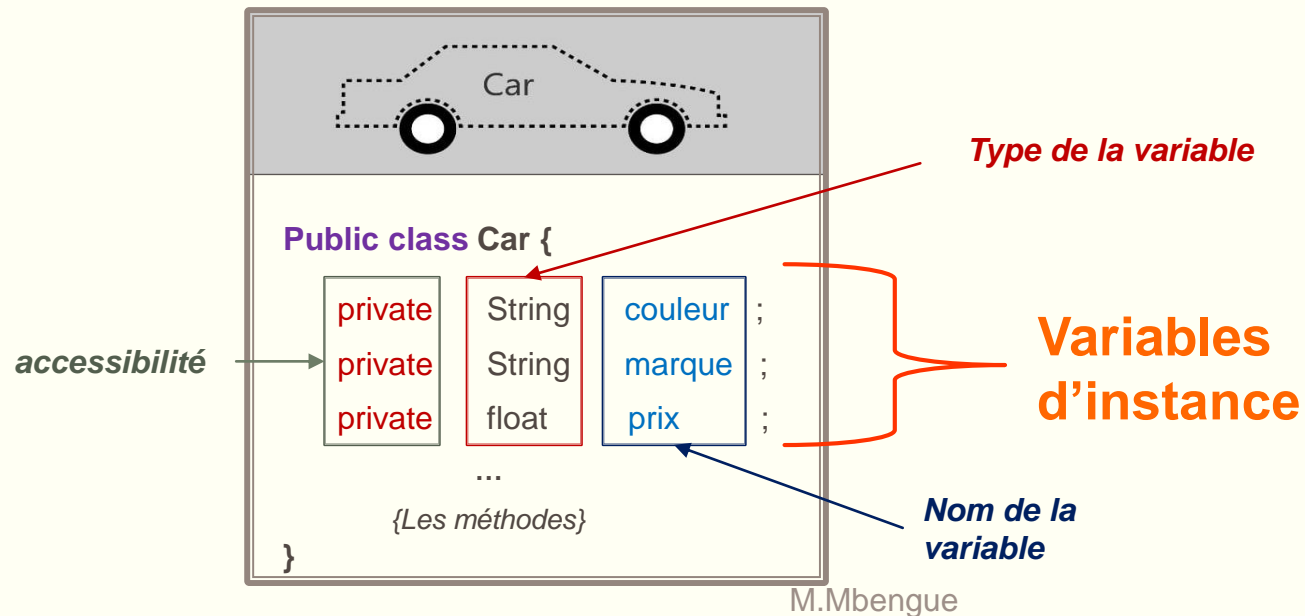
Disposition de titre et de contenu avec liste

TP

Créer sa première classe

Les variables d'instance

- Les variables d'instance seront déclarées dans le corps de la classe (au début, de préférence...).
- Leur création répond à la question « quels sont les attributs de mon objet ? »
→ ex : *une voiture* sera caractérisée par *sa marque, sa couleur, son modèle, son prix, etc ...*
- Comme pour n'importe quelle autre variable, on déclare une variable d'instance en déclarant son **type** et son nom.
- Les variables d'instances ont la particularité d'être définie en tant qu'élément « **private** » (on verra l'utilité de ce mot réservé plus tard...)



Les types de données

Parmi les données on distingue :

- Les objets, qui ont pour type la classe qui les définit
- Les données de type primitif, qui appartiennent à l'un des huit types primitifs

Les huit types primitifs

Booléens

- **boolean** (true /false)

Entiers

- **byte** (de -128 à 127)
- **short** (de -32768 à 32767)
- **int** (de -2147483648 à 2147483647)
- **long** (de -9223372036854775808 à 9223372036854775807)

Nombres à virgule

- **float** (précision de 7 chiffres après la virgule)
- **double** (précision de 15 chiffres après la virgule)

Caractères

- **char**
(« balblabla »)

String n'est pas un type primitif !

String est une classe, utilisée comme type représentant une chaîne de caractères. Ce n'est pas un type primitif mais c'est un type spécial

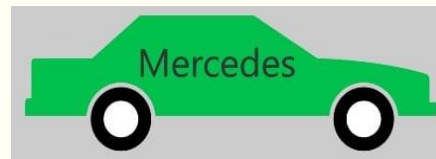
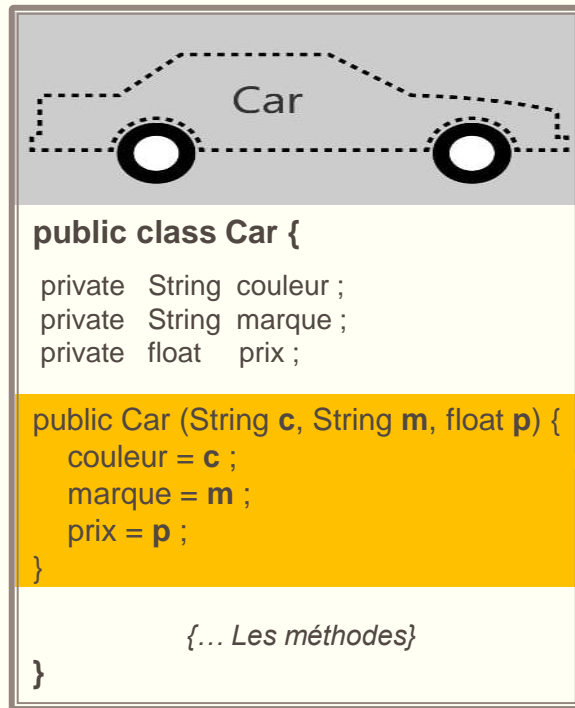
Disposition de titre et de contenu avec liste

TP

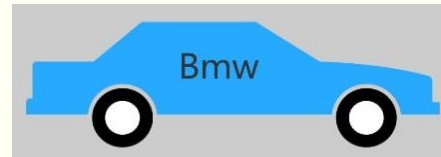
**Définir des variables
d'instance**

Le constructeur(Créer des instances de ma classe)

- Une classe a un ou plusieurs constructeurs qui servent à créer les instances et à initialiser leur état
- Un constructeur a **le même nom** que la classe et n'a pas de type retour



Car **voiture1** = **new** Car (vert, mercedes, 20000) ;



Car **voiture2** = **new** Car (bleu, bmw, 22000) ;

Constructeur de
la classe « Car »

Par défaut, si aucun constructeur n'est défini explicitement dans une classe, il existe toujours un constructeur sans arguments.

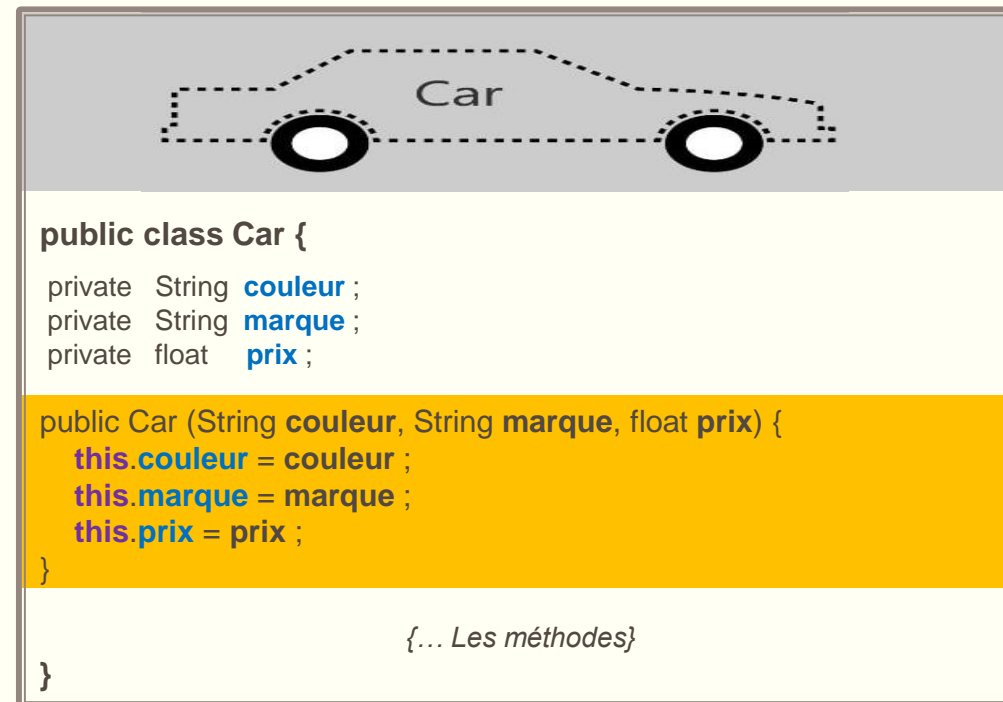
L'instance créée :

- **A son propre état interne** (les valeurs des variables d'instance) .
- **Partage le code qui détermine son comportement** (les méthodes) avec les autres instances de la classe.

Utilisation du « **this** »

Il est parfois commode d'utiliser le même nom pour les variables d'instance et les paramètres du constructeur.

Dans un tel cas, il est alors nécessaire de rendre le code explicite et non-ambigu, en utilisant le mot clé **this** pour désigner l'attribut.



Disposition de titre et de contenu avec liste

TP

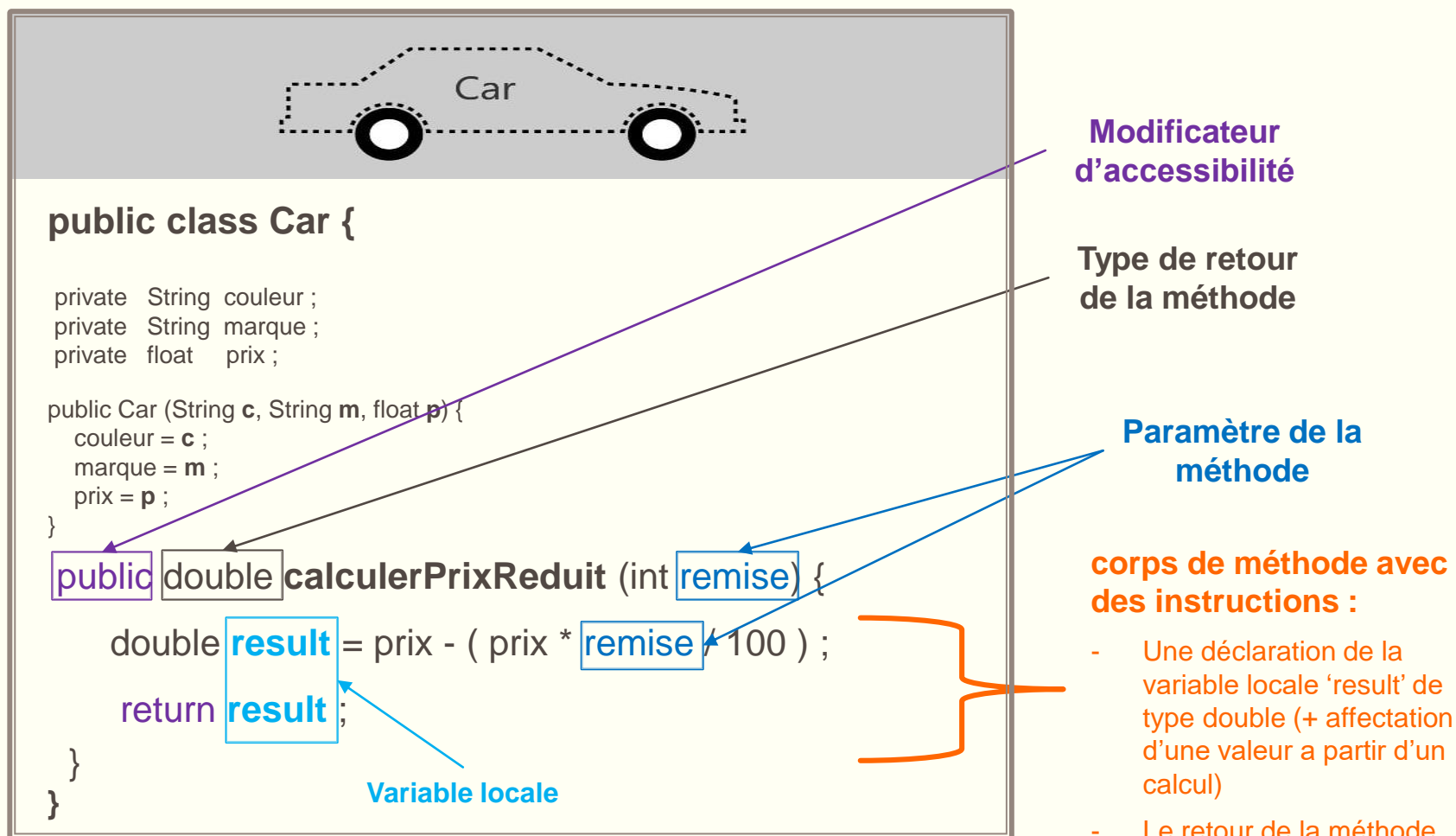
Ecrire un constructeur

Les différents éléments d'une méthode



```
public class Car {  
  
    private String couleur ;  
    private String marque ;  
    private float  prix ;  
  
    public Car (String c, String m, float p) {  
        couleur = c ;  
        marque  = m ;  
        prix    = p ;  
    }  
  
    public double calculerPrixReduit (int remise) {  
        double result = prix - ( prix * remise / 100 ) ;  
        return result ;  
    }  
}
```

Les différents éléments d'une méthode



Les paramètres d'une méthode

- Souvent, les méthodes (comme les constructeurs) ont besoin qu'on leur passe **des données initiales sous la forme de paramètres**
- Le type des paramètres doit être indiqué dans la déclaration de la méthode

Ex 1

```
public int addition (int a, int b) {  
    return a + b ;  
}
```

Ex 2

```
public String passerEnMinuscule (String phrase)  
{  
    return phrase.toLowerCase() ;  
}
```

- Quand la méthode ou le constructeur n'a pas de paramètre, on ne met rien entre les parenthèses

Ex 3

```
public void disBonjour () {  
    System.out.println(« Bonjour ») ;  
}
```

Le type de retour d'une méthode

- Quand la méthode renvoie une valeur, on doit indiquer le type de la valeur renvoyée dans la déclaration de la méthode

Ex 1

```
public int addition (int a, int b) {  
    return a + b ;  
}
```

Ex 2

```
public String passerEnMinuscule (String phrase) {  
    return phrase.toLowerCase() ;  
}
```

- Quand la méthode ou le constructeur n'a pas de paramètre, on ne met rien entre les parenthèses

Ex 3

```
public void disWhatever (String text) {  
    System.out.println(text) ;  
}
```

Méthodes de la classe String

- `length() : int`
retourne le nombre de caractères dans une chaîne
- `isEmpty() : boolean`
retourne vrai si la chaîne est de longueur 0
- `equals(String str) : boolean`
- `substring(int fromIndex) : String`
- `substring(int fromIndex, int endIndex) : String`
- `toLowerCase() : String`
- `toUpperCase() : String`
- `trim() : String`

... quelques exemples

```
public int calculeNombreLettre(String mot) {  
    return mot.length();  
}
```

```
public boolean isVariableNonAffecte(String var) {  
    return var.isEmpty();  
}
```

```
public boolean verifierMessageBienvenu(String msg) {  
    return msg.substring(0, 5).equals("Welcome");  
}
```

```
public String passerEnMajuscule(String phrase) {  
    return phrase.toUpperCase();  
}
```

Opérateurs dans le langage Java

Assignation (=)

- `int i = 0;`

Arithmétique (+ - * / ++ --)

- `int j = 1 + 1;`
- `int k = 1 - 1;`
- `int m = 2 * 2;`
- `int n = 2 / 1;`
- `n++;`
- `m--;`

Opérations sur les chaînes (+)

- `String var = "de" + "but";`

Comparaison (> < >= <= == !=)

- `boolean res1 = 1 > 0;`
- `boolean res2 = 1 < 1;`
- `boolean res3 = 1 >= 1;`
- `boolean res4 = 1 <= 0;`
- `boolean res5 = 1 == 0;`
- `boolean res6 = 1 != 0;`

Logique (&& || !)

- `boolean res7 = true && false`
- `boolean res8 = true || false`
- `boolean res9 = ! true`

Grands types d'instructions

- **déclaration de variables**

- `int i;`
- `String s;`
- `Voiture v;`

- **affectation d'une valeur à une variable**

- `i = 2;`
- `s = "bonjour"; s = s.length();`
- `v = new Voiture(); v = contrat.getVoiture();`

- **instanciation d'objet**

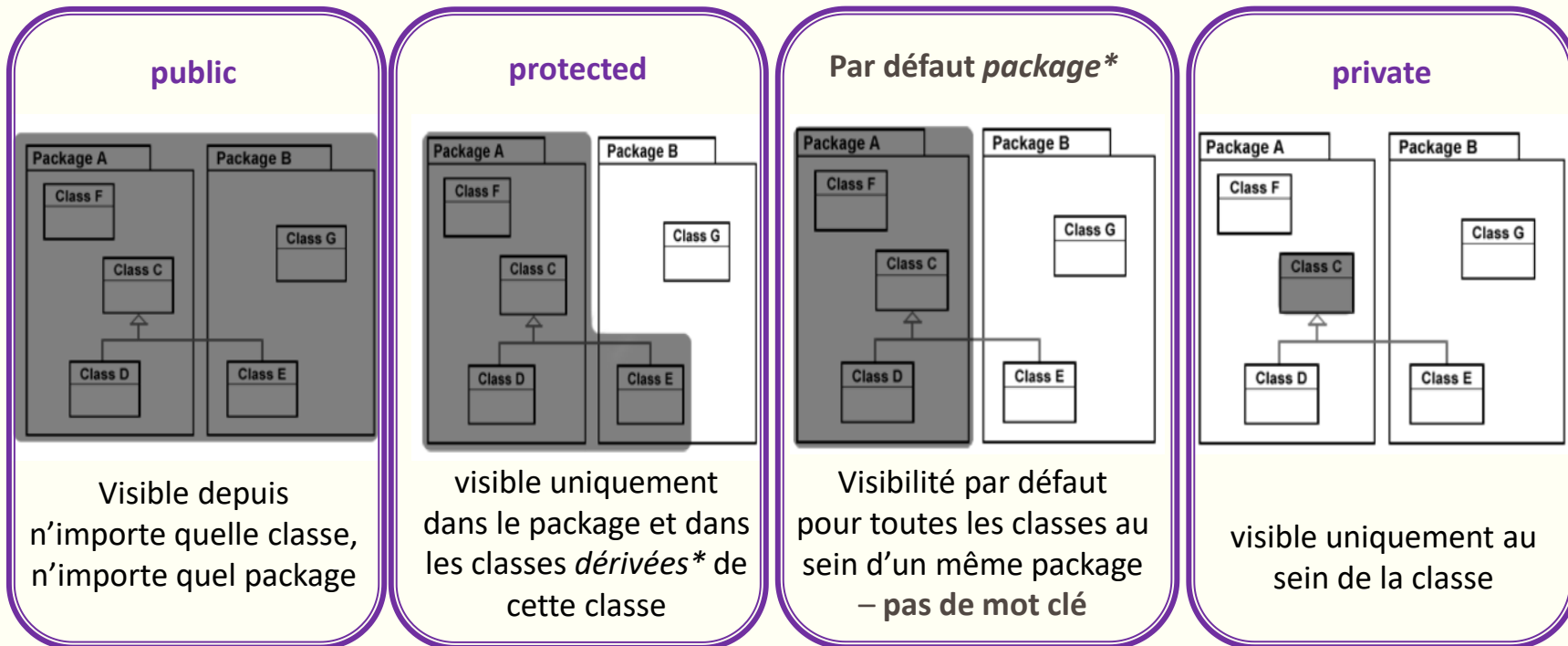
- `new Voiture("Ford");`

- **appel de méthode**

- `s.length();`

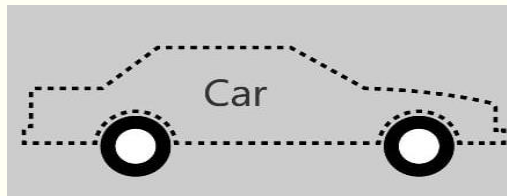
Les modificateurs d'accessibilité

- Toutes les méthodes et données membres définies au sein d'une classe sont utilisables par toutes les méthodes de la classe.
- Lors de la conception d'une classe, il faut décider des méthodes/variables qui seront visibles à l'extérieur de cette classe (principe d'encapsulation).
- Java implémente la protection des 4 P (**public**, **package**, **protected**, **private**)



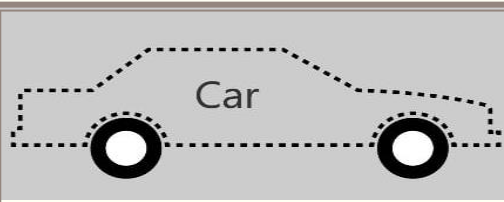
Les accesseurs (*getter* et *setter*)

- Deux types de méthodes servent à donner accès aux variables d'instance (**private**) depuis l'extérieur de la classe
 - ❖ Les **accesseurs en lecture** pour lire les valeurs des variables → **getter** en anglais
 - ❖ Les **accesseurs en écriture** pour modifier leur valeur → **setter** en anglais



```
public class UneAutreClasse {  
    ...  
    Car voiture1 = new Car ();  
    voiture1.setCouleur(« bleue »);  
    voiture1.setMarque(« Bmw »);  
    System.out.println(« Quelle belle voiture » + voiture1.getCouleur());  
    ...  
}
```

Les accesseurs (*getter* et *setter*)



```
public class Car {  
    private String couleur ;  
    private String marque ;  
    private float  prix ;  
  
    // public Car () {}  
  
    Public void setCouleur(String c){  
        couleur = c ;  
    }  
  
    Public String getCouleur(){  
        return couleur ;  
    }  
  
    ...  
}
```



```
// création de l'instance voiture1 à partir du constructeur implicite de la classe Car  
Car voiture1 = new Car () ;  
  
// appel de la fonction setCouleur(String) de la classe Car  
voiture1.setCouleur(« bleue »);  
  
// appel de la fonction setMarque(String) de la classe Car  
voiture1.setMarque(« Bmw »);  
  
// affichage en console de la concaténation de « Quelle belle voiture » et du  
résultat de la méthode getCouleur de la classe « Car » renvoyant la valeur de la  
variable 'couleur'  
System.out.println(« Quelle belle voiture » + voiture1.getCouleur();
```

Consol

Quelle belle voiture bleue

M.Mbengue

Méthodes et propriétés statique

- Le mot-clé **static** permet de déclarer une méthode – ou une propriété – qui n'est pas liée à une instance de classe mais à la classe elle-même.
- Cela signifie que ces éléments ne sont pas dans le contexte d'un objet lorsqu'ils sont utilisés mais dans le contexte de la classe.
- On les appelle en utilisant le nom de la classe et non le nom d'une instance (nom de variable).

Exemple de méthode statique



```
public class Car {
```

```
    private String couleur ;  
    private String marque ;  
    private float  prix ;
```

```
    public Car (String c, String m, float p) {  
        couleur = c ;  
        marque = m ;  
        prix = p ;  
    }
```

```
    public double calculerPrixRduit (int remise) {  
        double result = prix - ( prix * remise / 100 ) ;  
        return result ;  
    }
```

Cette méthode est dynamique,
puisque'elle dépend de la
valorisation de la variable **prix**
par une instance de classe

```
    public static double calculerAcceleration (float vitesselnit, float vitessFinal, float duree) {  
        double result = (vitessFinal - vitesselnit) / duree  
        return result;  
    }
```

Cette méthode est '**static**',
puisque'elle n'est liée à aucune
instance de la classe.

Les méthodes et les mots-clés

- **final** (classes, méthodes et propriétés)

- Sert à déclarer que l'élément qu'il marque est immuable.
- S'il s'agit d'une classe, il n'est pas possible d'en définir une sous-classe.
- S'il s'agit d'une méthode, il n'est pas possible de la redéfinir / la surcharger.
- S'il s'agit d'une propriété, sa valeur ne peut pas être modifiée.

- **abstract** (classes et méthodes)

- Sert à déclarer qu'un élément n'est pas complètement défini.
- Une classe peut être déclarée comme abstraite.
- Elle ne peut alors pas être instanciée
- Elle peut en lieu et place d'une méthode contenir une simple déclaration de signature de méthode portant elle-même le mot-clé **abstract**

Héritage

- **Objectif** : raccourcir les temps d'écriture et de mise au point d'une application en réutilisant le code déjà implémenté.
- **La méthode** : réunir des objets possédant des caractéristiques communes dans une nouvelle classe, plus générale, appelée super-classe.
- On parle alors d'**héritage**
- En Java, **chaque classe a une et une seule classe mère** (pas d'héritage multiple), dont elle hérite les variables et les méthodes
- Le mot clé est **extends**.

Hérédité et surcharge des méthodes

- Si la classe « B » hérite de la classe « A », on dira que « B » **extends** « A ».
- « B » hérite des méthodes de « A » et **peut les surcharger**

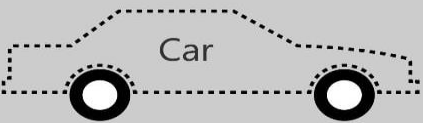
?

```
public class Vehicule {  
    private int nb_roues ;  
    public Vehicule (int nb_roues) {  
        this.nb_roues = nb_roues ;  
    }  
    public void avance() {  
        ...  
    }  
}
```

*La classe « Car » peut surcharger le constructeur de « Vehicule » grâce au mot clé **super()**.*



La classe « Car » hérite de la classe « Vehicule »



```
public class Car extends Vehicule {  
    private String couleur ;  
    private String marque ;  
    private float prix ;  
    public Car (String c, String m, float p) {  
        super(4)  
        couleur = c ;  
        marque = m ;  
        prix = p ;  
    }  
}
```

Une instance de « Car » peut utiliser les méthodes de « Vehicule ».

→ Polymorphisme*

```
public class UneAutreClasse {  
    ...  
    Car bmv = new Car(« vert », « bmv », 20000);  
    bmv.avance();  
    ...  
}
```

Interfaces

- Une classe peut implémenter une interface. Contrairement à une classe, une interface ne spécifie pas de comportement mais uniquement des définitions de méthodes.
- L'idée est d'imposer aux classes qui implémentent cette interface, une manière d'interagir avec l'extérieur.
- Il faut voir l'implémentation comme **un contrat** : la classe qui implémente une interface s'engage à surcharger toutes les méthodes définies dans cette interface.
- Une classe peut implémenter autant d'interfaces qu'elle le souhaite (...et hérité d'une classe).
- En Java, le mot clé est **implements**.

Attention!

Une interface ne peut contenir que des **variables constantes** ou **statiques** et des **entêtes de méthodes**


```
int UNE_VARIABLE_CONSTANTE = 1 ;
```


Interface et implémentation des méthodes


- Si la classe « B » implémente la classe « A, » on dira que « **B** » **implements** « A ».
- « B » s'engage surcharger toutes les méthodes de « A »

?

```
public interface Vehicule {  
    void rouler();  
    void freiner();  
}
```



```
public class Car implements Vehicule {  
    //Constructeurs  
    public Car(... , ..) { ...}  
  
    //Methodes  
    public void rouler() {  
        //Coder ici la manière dont l'auto roule  
    }  
  
    public void freiner() {  
        //Coder ici la manière dont l'auto freine  
    }  
  
    //Autres méthodes propres à Car.
```



```
public class Bike implements Vehicule {  
    //Constructeurs  
    public Bike(... , ..) { ...}  
  
    //Methodes  
    public void rouler() {  
        //Coder ici la manière dont le velo roule  
    }  
  
    public void freiner() {  
        //Coder ici la manière dont le velo freine  
    }  
  
    //Autres méthodes propres à Velo.
```

Polymorphisme

- **Concept** : Un objet n'est pas nécessairement manipulé selon la spécification de la classe dont il est une instance. Il peut être manipulé selon la spécification d'une **classe** dont il **hérite** (= qu'il étend) ou encore d'une **interface** qu'il **implémente**.
- On parle de **polymorphisme**.

Exemple :

// création d'une instance de PageAccueil à partir du type PageGenerique -> Polymorphisme

PageGenerique **page1** = new PageAccueil();

// création d'une instance de PageCatalogue à partir de son type -> OK

PageCatalogue **page2** = new PageCatalogue();

// création d'une liste du type de l'interface Page

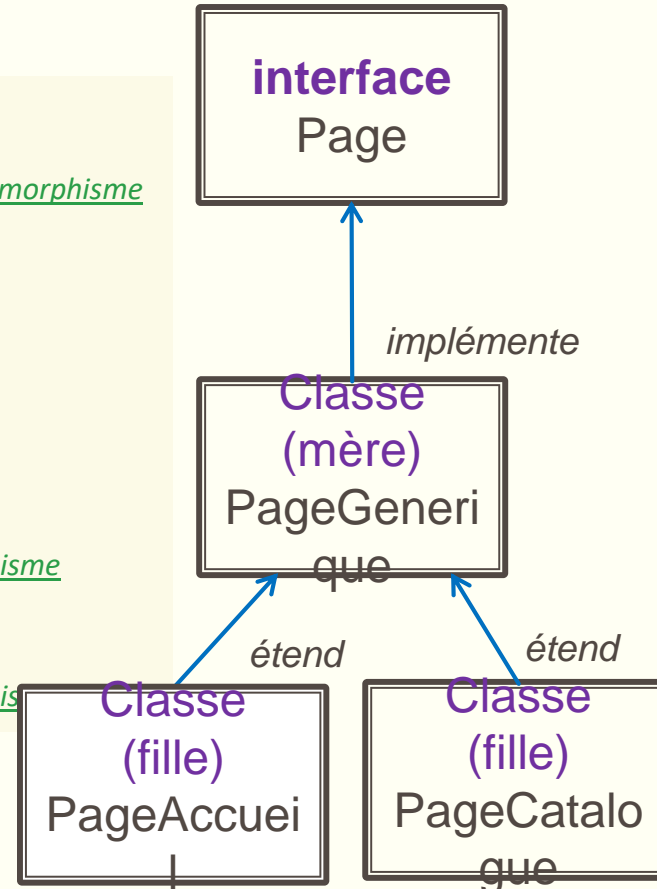
List<**Page**> **listeDePages** = new ArrayList<**Page**>();

// ajout de l'instance page1 (PageGenerique) dans la liste de pages -> Polymorphisme

listeDePages.add(page1);

// ajout de l'instance page2 (PageCatalogue) dans la liste de pages -> Polymorphisme

listeDePages.add(page2);




Énumérations

- Une énumération est un type de données particulier, dans lequel une variable ne peut prendre qu'un nombre restreint de valeurs. Ces valeurs sont des constantes nommées.
- **Exemple :** une énumération « Civilité » aura pour données : MADAME, MONSIEUR, MADEMOISELLE ;
- Une énumération se déclare comme une classe mais avec le mot-clé **enum** au lieu de **class**.



```
public enum Color {  
    BLEU,  
    VERT,  
    JAUNE,  
    ROUGE;  
}
```



```
public class Car {  
    private Color couleur ;  
    private String marque ;  
    private float prix ;  
  
    // public Car () {}  
  
    Public void setCouleur(Color c){  
        couleur = c ;  
    }  
  
    Public Color getCouleur(){  
        return couleur ;  
    }  
  
    ...  
}
```

```
public class UneAutreClasse {  
    ...  
  
    Car voiture1 = new Car();  
    voiture1.setCouleur(Color.BLEU);  
    ...  
}
```

Décisions

- Mots-réservés : **if**, **else**

Une **condition** doit retourner « vrai » ou « faux »
➔ Peut être toute méthode de type retour **boolean**

<pre>if (condition){ instructions }</pre>	<pre>If (condition) { instructions } else { instructions }</pre>	<pre>If (condition 1) { instructions } else if (condition 2) { instructions }</pre>
---	---	---

Comparaison de types primitifs ou de référence d'objets	Comparaison de valeurs d'objets	Comparaisons d'objet avec <i>null</i>
<pre>if (<i>a</i> == <i>b</i>) if (<i>a</i> != <i>b</i>) if (<i>a</i> <= <i>b</i>) if (<i>a</i> > <i>b</i>)</pre>	<pre>if (<i>a</i>.equals(<i>b</i>)) if (!<i>a</i>.equals(<i>b</i>)) if ("".equals(<i>b</i>))</pre>	<pre>if(<i>a</i> == null) if(<i>a</i> != null)</pre>

Sélections

- Mots-réservés : **switch, case**

- Structure :

```
switch (variable de type primitif ou String){  
  
    case valeurDeLaVariable : instructions ; break;  
    case valeurDeLaVariable : instructions ; break;  
    case valeurDeLaVariable : instructions ; break;  
  
    ...  
    default : instructions;  
  
}
```

Besoin d'un
exemple avec la
voiture ?

```
switch (String voiture1.couleur){  
  
    case bleu : System.out.println (« belle voiture bleu ! ») ; break;  
    case vert : System.out.println (« belle voiture verte ! ») ; break;  
    case jaune : System.out.println (« belle voiture jaune ! ») ; break;  
  
    ...  
    default : System.out.println (« votre voiture n'a pas de couleur ? ») ;  
  
}
```

Tableaux

```
String tab[] = new String[10];
```

```
tab[0] = "premier élément";
```

```
tab[9] = "dernier élément";
```

For

- Mots-réservés : **for**
- Structure :

```
for (initialisation variable; condition; instruction){  
    instructions  
}
```

Un exemple ?

```
for (int i=0; i<10; i++){  
    System.out.println( i );  
}
```



À chaque répétition de la boucle (il y en aura 10), la console affichera la valeur de *i*

While

- Mots-réservés : **while** → (*tant que*)
- Structure :

```
while (condition){  
    instructions  
}
```

L'*instruction* se répète tant que la *condition* est vrai (*true*)

Les collections listes et leur parcours

Créer une collection et lui affecter des valeurs

```
List<String> liste = new ArrayList<String>();  
liste.add("premier élément");  
liste.add(« deuxième élément »);
```

Parcourir la collection

```
for(String s : liste){  
    assertTrue(s.contains(« élément »));  
}
```

Manipuler une liste

- `add(E element) : boolean`
- `addAll(List<E> elements) : boolean`
- `contains(Object o) : boolean`
- `isEmpty() : boolean`
- `size() : int`

...

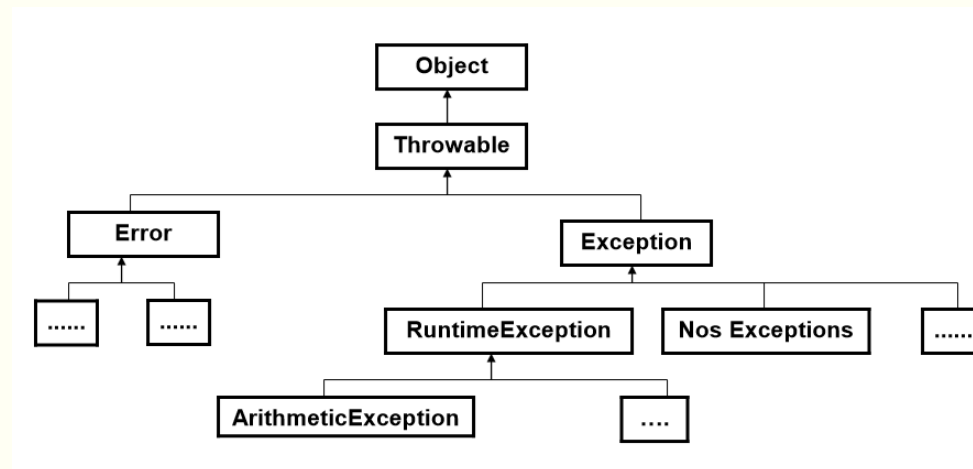
Maps

- L'interface `Map<Key, Value>` permet de stocker des valeurs associées à des clés uniques.
- `containsKey(Object key) : boolean`
 - Retourne vrai si la clé existe, sinon retourne faux
- `containsValue(Object value) ; boolean`
 - Retourne vrai si la valeur existe, sinon retourne faux
- `put(K key, V value) : V`
 - Ajoute une correspondance clé/valeur
- `get(Object key) : V`
 - Permet de récupérer la valeur associée à une clé
- `keySet() : Set<K>`
 - Permet de récupérer l'ensemble des clés

Exceptions

- Une **exception** représente une **erreur**.
- Une exception est un signal qui se déclenche en cas de problème.
- Il est possible de lancer une exception pour signaler une erreur.
- Lancer une exception si elle n'est pas gérée implique l'interruption du programme
- Pour éviter l'arrêt du programme on peut gérer les erreurs
- La gestion des exceptions se décompose en deux phases :
 - La levée d'exceptions,
 - Le traitement d'exceptions

Les exceptions sont
des classes Java



Créer et lever des exceptions : **throws**

- Il existe de nombreuses classes d'exception, dont beaucoup sont levées implicitement par la machine virtuelle.
- Il est néanmoins possible de créer des classes d'exception spécifiques que l'on entend lever sous certaines conditions
- Dans une méthode, on prévoit la levée de l'exception par les mots-clés **throws** et **throw**

X

```
public class SaisieErroneeEx extends Exception {  
  
    public SaisieErroneeEx(String s) {  
        super(s);  
    }  
  
}
```



```
public class voiture {  
    ...  
    public setMarque (String marque) throws SaisieErroneeEx {  
        if (marque.equals("") == true) {  
            throw new SaisieErroneeEx("Saisie erronee : chaine vide");  
        }  
        ...  
    }  
}
```

Exceptions

Le traitement des exceptions se fait à l'aide de la séquence d'instruction `try...catch...finally`.

```
try{  
    // instructions susceptibles de lever des exceptions  
  
} catch(Exception e){  
    // instructions si une exception est levée  
  
} finally {  
    // Sert à définir un bloc de code à exécuter dans tous les cas  
  
}
```

Les tests non-passant

`@Test (expected=Exception.class, timeout=1000)`

Permet de déclarer que le déclenchement d'une exception est une condition de succès du test, de fixer un timeout en millisecondes

Mots réservés – Types primitifs

abstract	assert	boolean	break	byte	case	catch
char	class	<i>const</i>	continue	default	do	double
else	enum	extends	false	final	finally	float
for	<i>goto</i>	if	implements	import	instanceof	int
interface	long	native	new	null	package	private
protected	public	return	short	static	strictfp	super
switch	synchronized	this	throw	throws	transient	true
try	void	volatile	while			

Mots réservés – Types primitifs

abstract	assert	boolean	break	byte	case	catch
char	class	<i>const</i>	continue	default	do	double
else	enum	extends	false	final	finally	float
for	<i>goto</i>	if	implements	import	instanceof	int
interface	long	native	new	null	package	private
protected	public	return	short	static	strictfp	super
switch	synchronized	this	throw	throws	transient	true
try	void	volatile	while			

Mots réservés – Types primitifs

abstract	assert	boolean	break	byte	case	catch
char	class	<i>const</i>	continue	default	do	double
else	enum	extends	false	final	finally	float
for	<i>goto</i>	if	implements	import	instanceof	int
interface	long	native	new	null	package	private
protected	public	return	short	static	strictfp	super
switch	synchronized	this	throw	throws	transient	true
try	void	volatile	while			



MODULE 3

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
