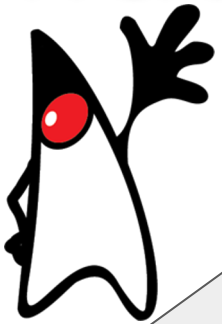
Université Ibn Tofail
Faculté des sciences de Kenitra

Licence d'Université Spécialisé
Développement des Applications Web et Mobile

جامعة ابن طفيل Université Ibn Tofaïl

Programmation Java EE



JAKARTA EE

Année universitaire
2020/2021

Pr. Yassine Ait Hsain
yassine.aithsain@uit.ac.ma

Plan du cours

Rappel

- Introduction
- Les concept d'objet, Classe et d'encapsulation
- L'héritage
- Le polymorphisme
- Les opérateurs et les expressions
- Les instructions de contrôle
- La gestion des exceptions

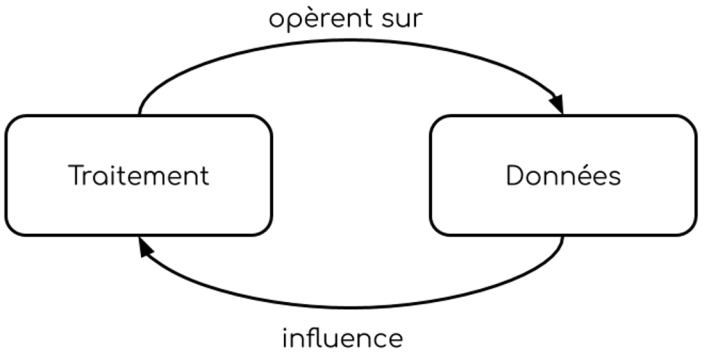
Introduction

Introduction

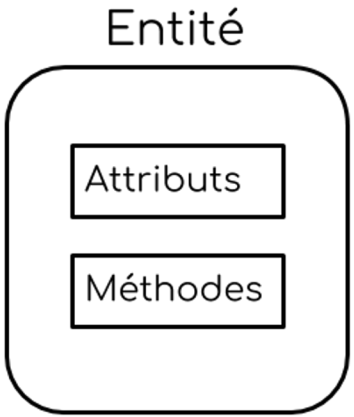
- ➔ est né en 1995 chez Sun Microsystems
  - ◆ Version actuelle Java 15.0.1 , actuellement Oracle
- ➔ est orienté objet
- ➔ est fortement typé
  - ◆ Toute variable doit être déclarée avec un type
  - ◆ Le compilateur vérifie que les utilisations des variables sont compatibles avec leur type (notamment via un sous-typage correct)
  - ◆ Les types sont d'une part fournis par le langage, mais également par la définition des classes
- ➔ est compilé
  - ◆ En bytecode, i.e., code intermédiaire indépendant de la machine
- ➔ est interprétée
  - ◆ Le bytecode est interprété par une machine virtuelle Java

Introduction

Programmation Structurée



Programmation Oriente Objet



Introduction

La programmation orientée objet offre, en réalité, quatre concepts fondamentaux:

- ➔ Encapsulation
- ➔ Abstraction
- ➔ Héritage
- ➔ Polymorphisme

qui permet de mieux organiser les programmes, dans le sens de la robustesse, lisibilité, modularité, et maintenabilité.

Les concept d'objet et d'encapsulation

Les concept d'objet et d'encapsulation

Principe d'encapsulation:

- regrouper dans le même objet informatique ("concept"), les données et les traitement qui lui sont spécifiques:
    - **Attributs**: les données incluses dans un objet
    - **Méthodes**: les fonctions définies dans un objet.

Le concept d’Abstraction

- Le processus d’abstraction consiste à identifier pour un ensemble d'éléments:
  - des caractéristiques communes à tous les éléments
  - des mécanismes communs à tous les éléments

```
String designation1 = “p1”;
int qte1 = 10;
```

```
String designation2 = “p2”;
int qte2 = 10;
```

```
afficher(designation1, qte1)
afficher(designation2, qte2)
```

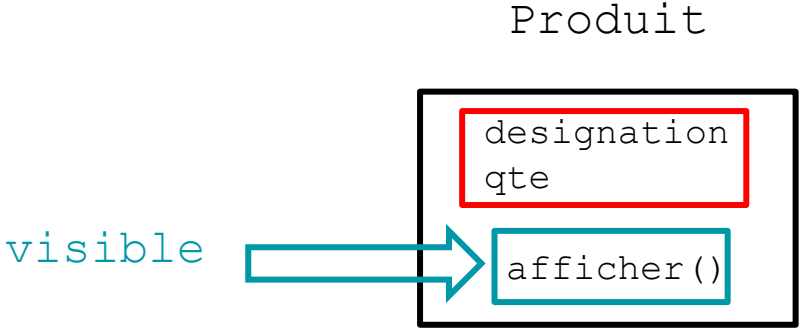
```
designation
qte

affciher()
```

Exemple

- L'interface d'une voiture
  - Volant, accélérateur, pédale de frein, etc.
  - Tout ce qu'il faut savoir pour la conduire (mais pas forcément comprendre comment ça marche)
  - L'interface ne change pas, même si l'on change de moteur... et même si on change de voiture (dans une certaine mesure): abstraction de la notion de voiture (en tant qu'objet a conduire)

Encapsulation et Interface

- L'encapsulation permet de définir deux niveaux d'abstraction:
  - Niveau **externe**: partie visible par les **programmeurs-utilisateurs**:
    - l'interface: en tête de quelque méthodes bien choisis.
  - Niveau **interne** : détails d'implémentation
    - corps: method et attributs accessible uniquement depuis l'intérieur de l'objet



Encapsulation et Interface

Il y a donc deux facettes à l'encapsulation:

- regroupement de tout ce qui caractérise l'objet: donnees (attribut) et traitements (méthodes)
- isolement et dissimulation de s détails d'implémentation
- interface = ce que le **programmeur-utilisateur** (hors de l'objet) peut utiliser.
- Ex: JDBC, SWT, etc.

Les concept d'objet et d'encapsulation

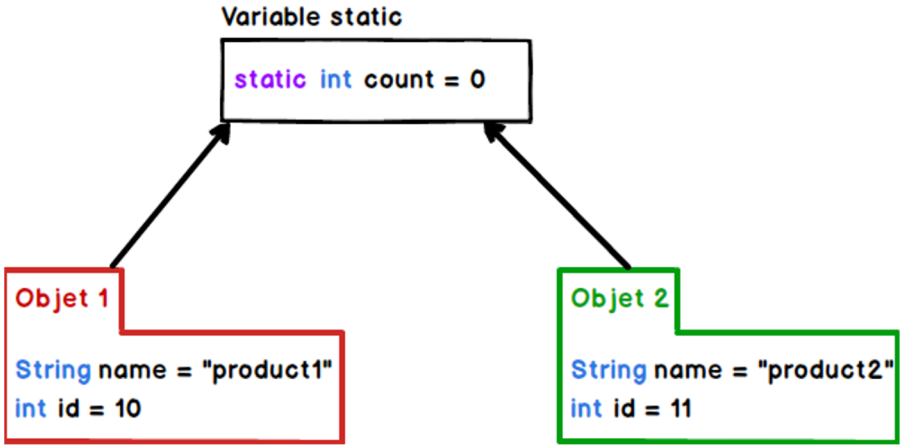
(droits d'accès)

- Droit d'accès:

Modificateur	Class	Package	SubClass	Monde
public	oui	oui	oui	oui
protected	oui	oui	oui	non
sans (package)	oui	oui	non	non
private	oui	non	non	non

Les concept d'objet et d'encapsulation (static)

- Le mot clé **static** en Java est un modificateur utilisé pour économiser l'espace mémoire. Cela aide à gérer la mémoire occupée par les objets, les variables et les définitions de méthodes. Le mot-clé static garantit qu'une seule instance d'une méthode d'objet ou d'une variable concernée est créée en mémoire.



Héritage

L'héritage

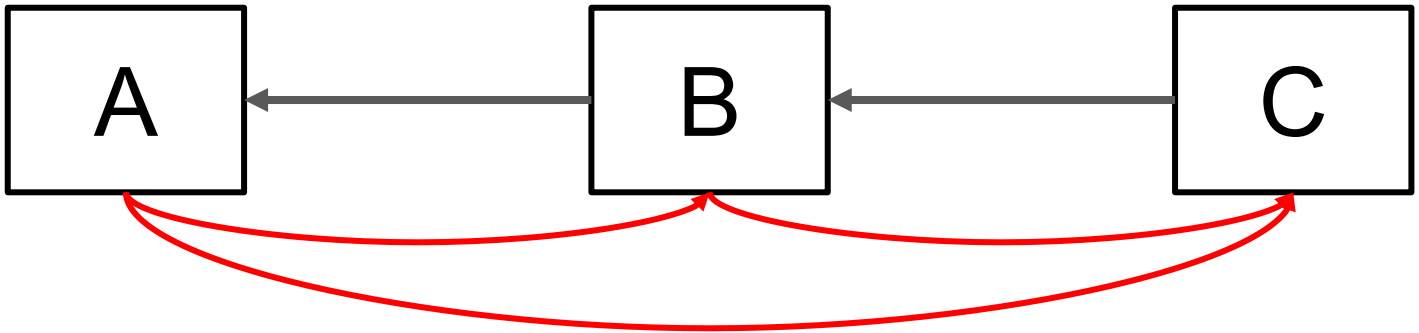
- Il permet de définir une nouvelle classe à partir d'une classe existante (qu'on réutilise en bloc !), à laquelle on ajoute de nouvelles données et de nouvelles méthodes.
- La conception de la nouvelle classe, qui hérite des propriétés et des aptitudes de l'ancienne, peut ainsi s'appuyer sur des réalisations antérieures parfaitement au point et les spécialiser à volonté.

Attention!

- l'héritage doit être utilisée pour décrire une relation "est-un" ("is-a")
- il ne doit jamais décrire une relation "a-un"/"possède-un" ("has-a")

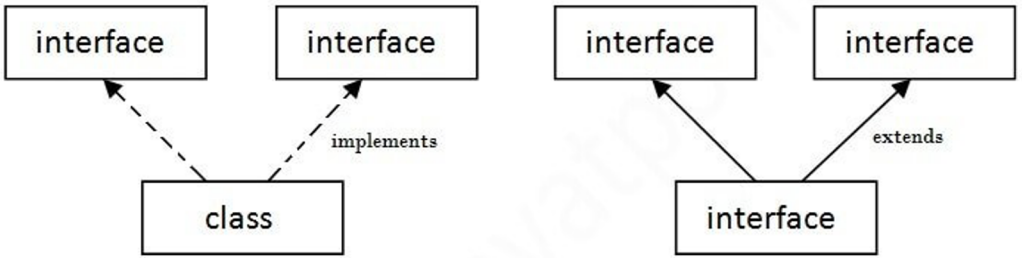
L'héritage (transitivité de l'héritage)

- Un sous-class qui hérite d'un super-class héritait c-a-d possédait, recevait les attributes et les méthodes (hors constructeurs) de l'ensemble des classes parentes (super-class, super-super-classe, etc.)
- Exemple:



Interfaces

- Une interface définit un comportement (d'une classe) qui doit être implémenté par une classe, sans implémenter ce comportement.
- C'est un ensemble de méthodes abstraites, et de constantes.



Multiple Inheritance in Java

Polymorphisme

Le polymorphisme

- Le mot polymorpsihm est apparu dans la Grèce antique. Il signifie quelque chose qui peut prendre plusieurs formes.
- Il permet de manipuler des objets sans en connaître (tout à fait) le type.
- Pour simplifier, le polymorphisme permet au développeur d'utiliser une méthode ou un attribute selon plusieurs manières, en fonction du besoin.
- Par exemple, on pourra construire un tableau d'objets (donc en fait de références à des objets), les uns étant de type Produit, les autres étant de type Ordinateur (dérivé de Produit) et appeler la méthode affiche pour chacun des objets du tableau. Chaque objet réagira en fonction de son propre type.

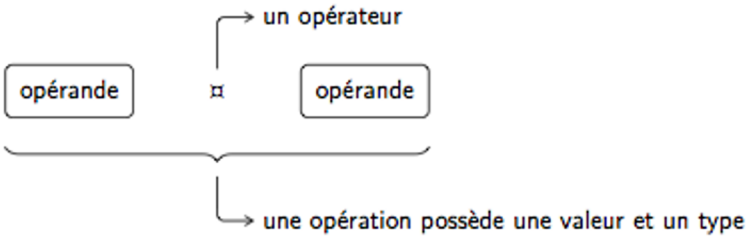
Le polymorphisme(Surcharge vs Redéfinition)

Surcharge	Redéfinition
La **surcharge** de méthode est utilisée pour améliorer la lisibilité du programme.	La **redéfinition** de méthode est utilisée pour fournir l'implémentation spécifique de la méthode qui est déjà fournie par sa super classe.
En cas de **surcharge** de méthode, les paramètres doivent être différent.	La **redéfinition** de méthode se produit dans deux classes ayant une relation d'héritage.
La **surcharge** de méthode est l'exemple du polymorphisme au moment de la **compilation**.	La **redéfinition** de méthode est l'exemple du polymorphisme au moment de **l'exécution**.
En Java, la **surcharge** de méthode ne peut pas être effectuée en modifiant uniquement le type de retour de la méthode. Le type de retour peut être identique ou différent dans la surcharge de méthode. Mais vous devez changer le paramètre.	Le type de retour doit être identique ou covariant lors de la redéfinition de méthode.

Les opérateurs et les expressions

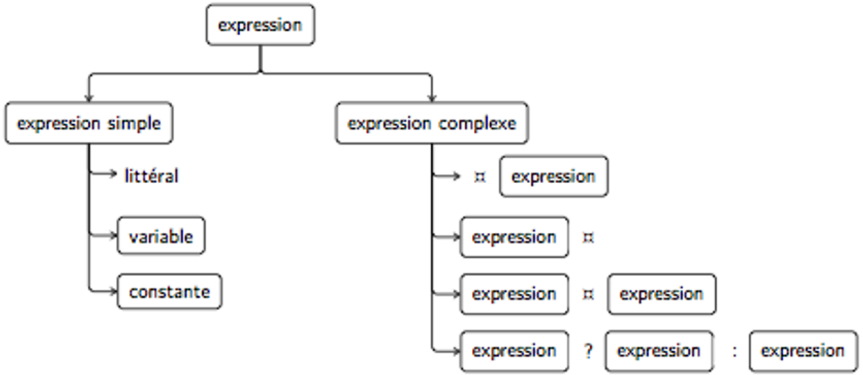
Les opérateurs et les expressions

- Les opérateurs permettent d'effectuer une opération bien définie sur des **valeurs** appelées opérandes, en produisant un résultat appelé **valeur** de l'opération. Cette valeur possède un type également bien définit par l'opération.



Les opérateurs et les expressions

- Une expressions, c'est tout simplement une combinaison d'opérateurs et d'opérandes.
- Une expression est évaluée, c'est-à-dire que sa valeur peut être calculée, celle-ci possédant toujours un type.
- Deux type:
  - Expression Simple : n'impliquent aucun opérateur
  - Expression Complexe : combine des opérandes avec des opérateurs.



La gestion des Exceptions

La gestion des exceptions

- Les exceptions représentent le mécanisme de gestion des erreurs intégré au langage Java.
- Il se compose d'objets représentant les erreurs et d'un ensemble de trois mots clés qui permettent de détecter et de traiter ces erreurs (**try, catch** et **finally**) mais aussi de les lever ou les propager (**throw** et **throws**).
- Lors de la détection d'une erreur, un objet qui hérite de la classe Exception est créé (on dit qu'une exception est levée) et propagé à travers la pile d'exécution jusqu'à ce qu'il soit traité.
- Ces mécanismes permettent de renforcer la sécurité du code Java.

La gestion des exceptions

- Si dans un bloc de code on fait appel à une méthode qui peut potentiellement générer une exception, on doit soit essayer de la récupérer avec try/catch, soit ajouter le mot clé throws dans la déclaration du bloc.
- Si on ne le fait pas, il y a une erreur à la compilation.

- Les erreurs et exceptions du paquetage java.lang échappent à cette contrainte. Throws permet de déléguer la responsabilité des erreurs à la méthode appelante

La gestion des exceptions

```
public class TestException {
  public static void main(java.lang.String[] args)
{
    int i = 3;
    int j = 0;
    System.out.println("résultat = " + (i / j));
  }
}
```

```
Exception in thread "main" java.lang.ArithmeticException: /
by zero
        at tests.TestException.main(TestException.java:23)
```

La gestion des exceptions

```
public class TestException {
  public static void main(java.lang.String[] args) {
    // Insert code to start the application here.
    int i = 3;
    int j = 0;
    try {
      System.out.println("résultat = " + (i / j));
    } catch (ArithmeticException e) {
      System.out.println(e.getMessage());
      e.printStackTrace();
    }
  }
}
```

`printStackTrace()` : affiche l'exception et l'état de la pile d'exécution au moment de son appel

Mettons cela en oeuvre