Développement en JAVA

Héritage, Polymorphisme et Exceptions

Lydia YATAGHENE
lydia.yataghene@junia.com

Lire/afficher sur la console

Affichage sur la console

System.out.println(“chaine de caractères”) ;

Lire depuis la console

```
Scanner scan = new Scanner(System.in) ;
 int n = scan.nextInt() ;
 double x = scan.nextDouble() ;
 String s = scan.nextLine() ;
```

Principes de Java

- **Encapsulation**
  - Regrouper les éléments fonctionnels
  - Protéger l'accès depuis l'extérieur au code et aux données
  - Représenter les classes comme des boîtes noires (wrappers)
- **Polymorphisme**
  - Modifier le comportement de fonctions selon les classes filles, d'après un comportement par défaut.
- **Héritage**
  - Décliner le comportement d'une classe en plusieurs variantes selon ses classes filles

Héritage

Héritage

```
public class Personnage {
 public String nom;
 // constructeur par défaut
 public Personnage(){
 nom = "Inconnu";
 }

public Personnage(String name){
 this.nom = name;
 }
 }
```

- On veut maintenant faire des classes Gaulois et Romain pour avoir des comportements plus spécifiques. Comment s'y prendre ?
- **Java propose l'héritage comme solution.**

Héritage

- L'héritage permet:
  - A un objet d'acquérir les propriétés d'un autre objet (évite de dupliquer le code).
  - Centraliser les définitions de méthodes.
  - Augmenter la modularité
  - Simplifier l'architecture

- La classe **mère** est plus générale.
  - Elle contient les propriétés communes à toutes les classes **filles.**
- Les classes filles ont des propriétés plus spécifiques.
  - On obtient une hiérarchie de classes.

Héritage

- **Principe:**
  - Pour exprimer qu'une classe est une classe fille, on utilise le mot-clé **extends** dans la déclaration d'une classe :

```
class <nom classe fille> extends <nom classe mère>
```

  - En Java, on hérite d'une **seule** et **unique** classe.
  - (une classe qui n'hérite d'aucune classe héritent en fait implicitement de la classe Object)

Exemple

```
public class Personnage {
public String nom;
// constructeur par défaut
 public Personnage(){
 nom = "Inconnu";
 }
 }
 public String presentation(){
 return "mon nom est” + nom;
 }
 }
```

```
public class Gaulois extends Personnage {
// constructeur par défaut
 public Gaulois(){
//…
}

}
```

Scope et héritage

Les attributs et méthodes:

- **public** sont toujours accessibles par une classe fille (bien sûr !)
- **private** restent inaccessibles, même pour une classe fille. Les attributs et méthodes définis en private ne sont pas hérités.
- **protected** sont hérités (même dans un package différent).

Superclasses

- Il est possible de faire appel au constructeur d'une classe mère depuis sa fille.
- Le mot-clé **super** permet de faire un appel direct au constructeur de la classe mère associée.
- Pour appeler le constructeur de la classe mère : la méthode se nomme **super**(liste des arguments) tout simplement.

Exemple

```
public class Personnage {
private String nom;
 // Constructeur
 public Personnage(String name){
 this.nom = name;
 }

public String presentation(){
 return ”Je m’appelle” + nom;
 }
 }
```

```
 public class Gaulois extends Personnage {

 public Gaulois(String name){
 super(name);
 }
 }

 public static void main(String[] args){
 Gaulois asterix = new Gaulois(”Astérix”);
 System.out.println( asterix.nom);
 }
```

La classe Object

- En Java, toute classe hérite implicitement de la classe Object.
- Une classe fille a donc toujours une seule classe mère.
- L'implémentation de toute méthode de la classe Object que vous ne redéfinissez pas sera évidemment l'implémentation de Object !

La classe Object

- **Méthodes:**
  - clone : Crée et retourne une copie d’un objet.
  - Equals : Indique si un objet est égal à un autre.
  - finalize: Appelé par le garbage collector sur un objet quand le garbage collector determine qu'il n'y a plus de références à l'objet .
  - getClass : Renvoie la classe d'exécution d’un objet.
  - hashCode : Renvoie un code de hachage pour l'objet.
  - toString : Renvoie une représentation de l'objet sous forme de chaîne.

Polymorphisme

Le polymorphisme

- Le polymorphisme est un mécanisme important dans la programmation objet. Il permet de modifier le comportement d'une classe fille par rapport à sa classe mère.
- Le polymorphisme permet d'utiliser l'héritage comme un mécanisme d'extension en adaptant le comportement des objets.
- Le polymorphisme permet de proposer un comportement de méthode différent selon les classes et instances appelantes.
- Une même méthode définie dans une classe va être redéfinie dans une autre classe.

Polymorphisme et héritage

- Pour les méthodes de la classe mère, on a le choix :
  - Soit le comportement est le même : on peut omettre la réécriture de la méthode.
  - Soit le comportement est différent : on peut réécrire la méthode (Polymorphisme).
- On peut utiliser une annotation @Override pour souligner que l'on redéfinit une méthode de la classe mère.
- Il existe deux références pour parcourir la hiérarchie :
  - **this** : est une référence sur l'instance de la classe.
  - **super** : est une référence sur l'instance mère.
- Evidemment, on peut ajouter des méthodes spécifiques à la classe fille !

Exemple

```
public class Personnage {
private String nom;
 // Constructeur
 public Personnage(String name){
 this.nom = name;
 }

 public Personnage(String name){
 this.nom = name;
 }

public String presentation(){
 return ”Je m’appelle” + nom;
 }
 }
```

```
 public class Gaulois extends Personnage {

 public Gaulois(String name){
 super(name);
 }

@Override
public String presentation(){
 return super.presentation() + ” je suis un gaulois”;
 }
 }

 public static void main(String[] args){
 Gaulois asterix = new Gaulois(”Astérix”);
 System.out.println( asterix.presentation());
 }
```

@Override

- En Java, tout méthode héritée et redéfinie dans une classe fille doit être spécifiée avec le tag @Override.
- Cela permet de spécifier qu'il s'agit d'un cas de polymorphisme et de générer automatiquement la documentation liée.

Polymorphisme

- On redéfinit dans une classe fille une des fonctions de la classe mère. On parle aussi de spécialisation.
- Le mot-clé super appelé au sein d'une méthode permet d'appeler la méthode équivalente dans la classe mère
- Polymorphisme paramétrique appelé "surcharge", il permet de redéfinir une méthode avec un prototype différent, il propose différents degrés de généricité et propose un nombre variable d'arguments.

Polymorphisme et constructeurs

- Comme toute méthode, on peut surcharger un constructeur.
  - Redéfinir un constructeur adapté aux contextes.
  - Définir des niveaux d'abstraction.

```
public class Personnage {

public String presentation(){ }

public String presentation(String name, int age){}

}
```

Polymorphisme et Arguments

- Il est possible de ne pas spécifier le nombre précis d'arguments attendus par une méthode
- Tous ces arguments devront néanmoins partager un type commun

```
public class Personnages {

public String presentation( String ... caracteristiques) {

 for ( String elt : caracteristiques ) {

  // Instructions
  }

 }

 }
```

Polymorphisme

- Le polymorphisme et le transtypage implicite nous permettent de manipuler des objets qui sont issus de classes différentes, mais qui partage un même type.

```
Gaulois obelix = new Gaulois(”Obélix”);

Gaulois asterix = new Gaulois(”Astérix”);

Personnage cleopatre = new Personnage(”Cléopâtre”) ;

Personnage[] distribution= new Personnage[3];

distribution[0]= asterix;

distribution[1]= obelix;

distribution[2]= cleopatre;
```

Polymorphisme

```
Personnage asterix = new Gaulois("Astérix");
```

- Dans l'exemple asterix est déclaré comme un Personnage, même si l'objet est en fait un Gaulois.
- Comme la variable est déclarée comme un Personnage, on ne peut pas appeler une méthode spécifique d'une classe dérivée comme Gaulois.

- **Si un objet O est déclaré avec un type T, on ne peut appeler que des méthodes du types T sur l'objet O !**

Polymorphisme

- Le polymorphisme ad-hoc est une définition d'une fonction commune entre classes ne partageant pas d'héritage commun.
- Le polymorphisme ad hoc permet de définir des opérateurs communs à des classes différentes

Exemples : toString, compareTo, ...

Opérateur instanceof

- Il est possible de tester la nature de chaque instance.
- Il est possible de vérifier si un objet est bien d'un certain type.

```
public class Personnage { . . . }
public class Gaulois extends Personnage { . . . }
public class IrreductibleGaulois extends Gaulois { . . . }
public class Romain extends Personnage { . . . }
. . .
 public static void main(String[] args){
 IrreductibleGaulois asterix = new IrreductibleGaulois();
 System.out.println( asterix instanceof Personnage); //true
 System.out.println( asterix instanceof Gaulois); //true
 System.out.println( asterix instanceof Romain); //false
```

Final

- Le mot-clé final indique que la définition d'un élément est terminale et que sa valeur ne peut être modifiée.
- Pour une classe : Empêcher l'héritage d'une classe, une classe **final** n'aura pas de classe.
- Pour une méthode : Son utilisation permet de bloquer la surcharge d'une méthode, cette méthode ne pourra pas être redéfinie dans une classe dérivée.
- Pour un attribut: Définir une valeur fixe pour un attribut.

```
public final class Personnage {}
public final String Presentation() {}
private final int age ;
final int test = 42;
```

Final

- Le mot-clé final peut être appliqué à n'importe quelle variable (Arguments de méthode, variables locales)
- Paramètres
  - Protection contre les changements
  - Arguments ignorés à la compilation
  - Transmissible aux classes anonymes et/ou internes
- Variables locales
  - Clarté du code
  - Optimisation du temps de compilation (constantes)
  - Transmissible aux classes anonymes et/ou internes

```
public int add ( final int a, final int b) {
 final int c = 5;
return a + b + c;
 }
```