PATRICK CLOUDY

DEVELOPPEZ S'IL VOUS PLAIT...

EDITIONS
FAMILINEG

Table des matières

Certes, nous allons échouer comme nous l'a dit notre vaillant maître de conférences, mais faisons de notre mieux !

Concepts clés de la POO:

- **Classe:** Un "type objet" qui définit la structure et le comportement d'un objet.
    - Attributs : variables qui définissent la structure d'un objet.
    - Méthodes : fonctions qui définissent le comportement d'un objet.
- **Objet:** Une instance d'une classe.
- **Héritage:** Mécanisme permettant à une classe de hériter des attributs et des méthodes d'une autre classe.
- **Polymorphisme:** Mécanisme permettant d'invoquer la même méthode sur des objets de différentes classes.

Création et initialisation d'objets:

- **Opérateur new:** Permet de créer une instance d'une classe.
- **Constructeur:** Méthode spéciale qui définit ce qui doit être fait pour créer une instance à partir d'une classe donnée.
- **Initialisation des attributs:**
    - Par défaut :
        - Numériques : 0
        - Caractères : '\0'
        - Booléens : false
        - Chaînes de caractères et autres objets : null
    - Explicite : Lors de la déclaration d'un attribut ou dans le constructeur.

Mécanisme d'affichage:

- **Méthode toString():** Méthode qui permet de convertir un objet en une chaîne de caractères.
    - Si elle n'est pas définie, l'adresse mémoire de l'objet est affichée.
    - Si elle est définie, la chaîne de caractères définie par la méthode est affichée.

Conventions à respecter:

- **Nommage des classes, méthodes, attributs et variables:**
    - Pascal case (ex: MyFirstObject)
    - Camel case (ex: myFirstMethod)
    - Snake case (ex: my_first_variable)
- **Structure du projet:**

- Fichiers sources dans le répertoire src
- Fichiers binaires dans le répertoire bin
- . . .

Exemple de code en Java pour illustrer les concepts de la POO :

```
public class Point {
  private int x;
  private int y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  public int getX() {
    return x;
  }
  public void setX(int x) {
    this.x = x;
  }
  public int getY() {
    return y;
  }
  public void setY(int y) {
    this.y = y;
  }
  public String toString() {
    return "(" + x + ", " + y + ")";
  }
}
public class Main {
  public static void main(String[] args) {
    Point p1 = new Point(1, 2);
    Point p2 = new Point(3, 4);
```

```
    System.out.println(p1); // Affiche "(1, 2)"
    System.out.println(p2); // Affiche "(3, 4)"
  }
}
```

Explication du code:

- La classe Point définit deux attributs privés : x et y.
- La classe Point possède un constructeur qui permet d'initialiser les attributs x et y lors de la création d'une instance.
- La classe Point possède deux méthodes getX() et setX() pour accéder et modifier l'attribut x.
- La classe Point possède deux méthodes getY() et setY() pour accéder et modifier l'attribut y.
- La classe Point possède une méthode toString() qui permet de convertir un objet Point en une chaîne de caractères.
- La classe Main crée deux instances de la classe Point : p1 et p2.
- La classe Main utilise la méthode System.out.println() pour afficher les instances p1 et p2.

Cet exemple illustre les concepts de base de la POO en Java :

- **Classes et objets:** La classe Point définit un type d'objet et les instances p1 et p2 sont des objets de ce type.
- **Encapsulation:** Les attributs x et y sont privés et ne sont accessibles qu'à travers les méthodes de la classe.

Gestion des références en Java à partir d'un exemple de diagramme de classes

Introduction

Ce cours présente les concepts fondamentaux de la gestion des références en Java, en s'appuyant sur un exemple de diagramme de classes. Nous aborderons les notions suivantes :

- **Allocation mémoire et références**
- **Passage par valeur ou par référence**
- **Comparaison d'objets**
- **Contrôle d'accès**
- **Visibilité et interface publique**

Exemple de diagramme de classes

Considérons le diagramme de classes suivant :

+----------------+

| Article |

+----------------+

| reference |

| description |

| price |

+----------------+

| toString() |

| setPrice() |

| moreExpensiveThan(Article) |

| lessExpensiveThan(Article) |

+----------------+

+----------------+

| BankAccount |

+----------------+

| number |

| balance |

+----------------+

| credit(double) |

| withdraw(double) |

| getBalance() |

+----------------+

+----------------+

| Person |

+----------------+

| name |

| address |

+----------------+

| openBankAccount() |

| closeBankAccount() |

+----------------+

+-----------------------------+

| BankAccount <=> Person |

+-----------------------------+

| owners |

| accounts |

+-----------------------------+

Ce diagramme représente trois classes : Article, BankAccount et Person. Les classes Article et BankAccount ont des attributs et des méthodes, tandis que la classe Person n'a que des attributs.

Allocation mémoire et références

En Java, la mémoire est allouée via l'opérateur new. L'opérateur new crée un objet à partir d'une classe et retourne sa référence. La référence est l'adresse mémoire de l'objet.

Par exemple, le code suivant crée un objet de type Article et stocke sa référence dans la variable a1 :

```
Article a1 = new Article("BQ45A21", "chair", 75.25);
```

Passage par valeur ou par référence

Lorsque vous passez un paramètre à une méthode, il peut être passé par valeur ou par référence.

- **Passage par valeur:** une copie de la valeur est transmise à la méthode. La modification de la valeur dans la méthode n'affecte pas la valeur d'origine.
- **Passage par référence:** la référence à l'objet est transmise à la méthode. La modification de l'objet dans la méthode affecte l'objet d'origine.

En Java, les types primitifs sont passés par valeur et les objets sont passés par référence.

Comparaison d'objets

L'opérateur == compare les références des objets et non les valeurs des attributs. Pour comparer les valeurs des attributs, il faut utiliser la méthode equals().

La méthode equals() est définie par défaut dans la classe Object. Cette méthode compare les attributs de l'objet courant avec les attributs de l'objet passé en paramètre.

Contrôle d'accès

Le contrôle d'accès permet de restreindre la visibilité des attributs et des méthodes d'une classe.

En Java, les mots-clés suivants sont utilisés pour contrôler l'accès :

- private : accessible uniquement depuis la classe
- default : accessible uniquement depuis le package
- public : accessible depuis n'importe quelle classe

Visibilité et interface publique

L'interface publique d'une classe est la liste des attributs et des méthodes accessibles à tous depuis l'extérieur de la classe.

Il est important de bien choisir la visibilité des attributs et des méthodes pour garantir la cohérence et la sécurité du code.

Exemple de code

```
public class Main {
  public static void main(String[] args) {
    // Création de deux articles
    Article a1 = new Article("BQ45A21", "Chaise", 75.25);
    Article a2 = new Article("BQ45A22", "Table", 120.50);
    // Comparaison des prix
    if (a1.moreExpensiveThan(a2)) {
      System.out.println("L'article " + a1.getName() + " est plus cher que l'article " + a2.getName());
    } else {
      System.out.println("L'article " + a2.getName() + " est plus cher que l'article " + a1.getName());
    }
    // Passage par référence
    modifierArticle(a1);
    // Affichage du prix après modification
    System.out.println("Nouveau prix de l'article " + a1.getName() + " : " + a1.getPrice());
  }
  public static void modifierArticle(Article article) {
    article.setPrice(article.getPrice() * 1.1); // Augmentation de 10%
  }
}
class Article {
  private String reference;
  private String description;
  private double price;
  public Article(String reference, String description, double price) {
```

```
    this.reference = reference;
    this.description = description;
    this.price = price;
  }
  public String getName() {
    return reference + " - " + description;
  }
  public double getPrice() {
    return price;
  }
  public void setPrice(double price) {
    this.price = price;
  }
  public boolean moreExpensiveThan(Article article) {
    return this.price > article.price;
  }
}
```

Explication du code:

- La classe Article possède des attributs pour la référence, la description et le prix.
- La méthode moreExpensiveThan() compare le prix de l'objet courant avec le prix de l'objet passé en paramètre.
- La méthode modifierArticle() modifie le prix de l'objet passé en paramètre.
- La fonction main() crée deux objets de type Article et compare leurs prix.
- La fonction main() appelle ensuite la méthode modifierArticle() avec l'un des objets en paramètre.
- Le prix de l'objet est ensuite affiché après modification.

Points importants à retenir:

- Les objets sont passés par référence en Java.
- La méthode equals() est utilisée pour comparer les valeurs des attributs de deux objets.
- Le contrôle d'accès permet de restreindre la visibilité des attributs et des méthodes d'une classe.

Attributs, méthodes et classes statiques

I. Introduction

Ce cours présente les notions d'attributs, de méthodes et de classes statiques en Java. Il est important de comprendre ces concepts pour écrire du code Java efficace et robuste.

II. Les constantes

A. Définition et avantages

Une constante est une variable dont la valeur ne peut pas être modifiée après sa déclaration. En Java, on utilise le mot-clé final pour créer des constantes.

Exemple:

```
public class Constantes {
  public static final double PI = 3.1415926535;
  public static final int MAX_VALUE = 100;


}
```

Avantages des constantes:

- Amélioration de la lisbilité du code
- Facilitation de la maintenance du code
- Réduction des risques d'erreurs

B. Attributs de classe et constantes

Les attributs de classe sont déclarés avec le mot-clé static. Ils sont partagés par toutes les instances de la classe.

Exemple:

```
public class Compteur {


  private static int count = 0;


  public Compteur() {
    count++;
  }


  public static int getCount() {
```

```
    return count;
  }

}
```

III. Méthodes statiques

A. Définition et fonctionnement

Les méthodes statiques sont déclarées avec le mot-clé static. Elles n'ont pas besoin d'une instance de la classe pour être appelées.

Exemple:

```
public class MathUtils {

  public static int add(int a, int b) {
    return a + b;
  }

  public static int multiply(int a, int b) {
    return a * b;
  }

}

// Utilisation de la méthode statique add()
int result = MathUtils.add(10, 20);
```

B. Méthodes statiques vs. méthodes d'instance

Fonctionnalité	Méthodes statiques	Méthodes d'instance
Accès	Accessible sans instance	Nécessite une instance
Modificateur d'accès	public, private, protected, default	public, private, protected, default

Utilisation	Opérations génériques sur la classe	Opérations spécifiques à une instance

IV. Classes statiques

A. Définition et caractéristiques

Une classe statique est une classe qui ne peut pas être instanciée. Elle ne contient que des membres statiques (attributs et méthodes).

Exemple:

```
public class StringUtils {

  public static String toUpperCase(String str) {
    return str.toUpperCase();
  }

  public static String toLowerCase(String str) {
    return str.toLowerCase();
  }

}
```

Les méthodes de classes et les boîtes à outils en Java

I. Méthodes de classes

A. Définition

Une méthode de classe est une méthode qui appartient à une classe et non à un objet particulier de cette classe. Elle est accessible à toutes les instances de la classe.

B. Exemple

```
public class StaticExample {
  public static void staticMethod() {
    System.out.println("This is a static method");
  }
  public static void main(String[] args) {
    StaticExample.staticMethod();
  }
}
```

Dans cet exemple, la méthode staticMethod() est une méthode de classe car elle est déclarée avec le mot-clé static. Elle est accessible depuis la méthode main() sans avoir besoin de créer une instance de la classe StaticExample.

C. Cas particulier : la méthode main()

La méthode main() est une méthode de classe particulière qui est le point d'entrée de chaque programme Java. Elle est toujours statique et ne peut pas être surchargée.

D. Règles d'utilisation

- N'utilisez des méthodes de classe que lorsque c'est indispensable.
- Privilégiez les méthodes d'instance pour les opérations qui dépendent de l'état d'un objet particulier.

II. Boîtes à outils

A. Définition

Une boîte à outils est une classe qui fournit des méthodes utilitaires pour un type de données particulier. Les méthodes de la boîte à outils sont généralement statiques.

B. Exemple : la classe Arrays

La classe Arrays fournit des méthodes utilitaires pour manipuler des tableaux.

C. Fonctionnalités des boîtes à outils

Les boîtes à outils offrent généralement des fonctionnalités pour :

- L'affichage
- Le remplissage
- Le test d'égalité
- La recherche de valeurs
- Le tri

D. Avantages des boîtes à outils

- Elles simplifient le code.
- Elles améliorent la lisbilité du code.
- Elles réduisent le risque d'erreurs.

E. Limites des boîtes à outils

- Elles ne sont pas toujours disponibles pour les types non primitifs.
- Elles peuvent ne pas être aussi performantes que des solutions codées en dur.

III. Conclusion

Les méthodes de classes et les boîtes à outils sont des outils puissants qui peuvent vous aider à écrire du code Java plus efficace et plus lisible. Il est important de les utiliser à bon escient et de connaître leurs limites.

IV. Exemple de code

L'exemple de code suivant illustre les notions abordées dans ce cours:

```
public class Main {
  public static void main(String[] args) {
    // Création de deux tableaux
    int[] intTab1 = new int[]{1, 5, 2, 0, 7, 3};
    int[] intTab2 = new int[]{1, 5, 2, 0, 7, 3};
    // Comparaison des tableaux
    System.out.println(Arrays.equals(intTab1, intTab2)); // true
    // Tri des tableaux
    Arrays.sort(intTab1);
    // Affichage des tableaux
    System.out.println(Arrays.toString(intTab1)); // [0, 1, 2, 3, 5, 7]
  }
}
```

Comparaison de tableaux

Il existe deux méthodes principales pour comparer des tableaux en Java :

- Arrays.equals(int[] a, int[] a2) : Cette méthode compare deux tableaux d'entiers et retourne true si les deux tableaux sont de même longueur et si chaque élément correspondant est égal.
- Arrays.deepEquals(int[][] a, int[][] a2) : Cette méthode compare deux tableaux d'entiers à deux dimensions et retourne true si les deux tableaux ont la même taille et si chaque élément correspondant est égal.

II. Fonctionnement des méthodes de comparaison

Les méthodes de comparaison de tableaux utilisent l'opérateur d'égalité == pour comparer les éléments des tableaux. L'opérateur == compare les références des objets, et non les valeurs des objets.

III. Limites des méthodes de comparaison

Les méthodes de comparaison de tableaux ne sont pas toujours suffisantes pour comparer des tableaux. Par exemple, si les tableaux contiennent des objets, les méthodes de comparaison de tableaux ne compareront que les références des objets, et non les valeurs des objets.

IV. Solutions alternatives

Si vous avez besoin de comparer des tableaux d'objets, vous pouvez utiliser la méthode equals() de la classe Object. La méthode equals() compare les valeurs des objets, et non les références des objets.

L'encapsulation, la surcharge, le chaînage et le polymorphisme

Encapsulation

Définition et objectifs

L'encapsulation consiste à masquer les détails d'implémentation d'une classe en limitant l'accès aux attributs et aux méthodes.

Objectifs:

- Améliorer la sécurité du code en limitant l'accès aux données sensibles.
- Faciliter la maintenance du code en masquant les détails d'implémentation.
- Augmenter la flexibilité du code en permettant de modifier l'implémentation sans affecter l'interface publique.

Exemple

Considérons une classe Personne avec les attributs nom et age.

```
public class Personne {

  private String nom;
  private int age;

  public Personne(String nom, int age) {
    this.nom = nom;
    this.age = age;
  }

  public String getNom() {
    return nom;
  }

  public void setNom(String nom) {
    this.nom = nom;
  }
```

```
  public int getAge() {
    return age;
  }

  public void setAge(int age) {
    this.age = age;
  }

}
```

Dans cet exemple, les attributs nom et age sont déclarés private. Cela signifie qu'ils ne sont accessibles qu'à l'intérieur de la classe Personne. Les méthodes getNom(), setNom(), getAge() et setAge() permettent d'accéder et de modifier les valeurs des attributs en respectant l'encapsulation.

Surcharge et chaînage

Surcharge

La surcharge permet de définir plusieurs méthodes de même nom mais avec des signatures différentes.

Exemple

Considérons une classe Math avec deux méthodes somme() :

```
public class Math {

  public static int somme(int a, int b) {
    return a + b;
  }

  public static double somme(double a, double b) {
    return a + b;
  }

}
```

Dans cet exemple, la classe Math possède deux méthodes somme() avec des signatures différentes. La première méthode prend deux entiers en arguments et retourne un entier. La deuxième méthode prend deux nombres décimaux en arguments et retourne un nombre décimal.

Chaînage

Le chaînage permet d'appeler une méthode depuis une autre méthode de la même classe.

Exemple

Considérons une classe String avec les méthodes toUpperCase() et toLowerCase():

```
public class String {
  public String toUpperCase() {
    // ...
  }
  public String toLowerCase() {
    // ...
  }
  public String toUpperCaseAndLowerCase() {
    return this.toUpperCase().toLowerCase();
  }

}
```

Dans cet exemple, la méthode toUpperCaseAndLowerCase() utilise le chaînage pour appeler les méthodes toUpperCase() et toLowerCase().

Array vs. ArrayList

Array

- Structure de données de taille fixe.
- Accès direct aux éléments.
- Déclaration : int[] tab = new int[10].

ArrayList

- Structure de données de taille dynamique.
- Offre de nombreuses méthodes pour manipuler les éléments.
- Déclaration : ArrayList<Integer> list = new ArrayList<>().

Exemple

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

tab[0] = 1;

tab[1] = 2;


ArrayList<Integer> list = new ArrayList<>();

list.add(1);

list.add(2);


// Accès aux éléments

int element1 = tab[0];

int element2 = list.get(0);


// Taille

int tailleTab = tab.length;

int tailleList = list.size();
```

Le polymorphisme

Définition

Le polymorphisme permet à une variable de référence de type générique de stocker une référence vers un objet de type spécifique.

Exemple

```
Animal animal1 = new Chien();

Animal animal2 = new Chat();


// Méthode commune aux deux classes

animal1.faireDuBruit();

animal2.faireDuBruit();
```

Dans cet exemple, les variables animal1 et animal2 de type Animal peuvent stocker des références vers des objets de type Chien et Chat.