POO

Principes de Base de la POO :

Encapsulation :

Définition : Regroupement des données et des méthodes qui les manipulent.

Objectif : Protéger l'état interne de l'objet, exposer uniquement ce qui est nécessaire.

Héritage :

Définition : Mécanisme par lequel une classe peut hériter des caractéristiques d'une autre.

Objectif : Réutilisation du code, extension et modification des comportements existants.

Polymorphisme :

Définition : Capacité d'une fonction à agir différemment selon l'objet sur lequel elle opère.

Objectif : Flexibilité dans l'utilisation des objets, écriture de code plus générique.

Classe définition

Une classe en POO est un concept qui sert de modèle ou de plan pour créer des objets. Elle est constituée de deux éléments principaux :

- Attributs (ou Propriétés) : Ce sont des variables qui définissent les caractéristiques d'un objet. Par exemple, dans une classe Voiture, les attributs pourraient être la marque, le modèle, la couleur, etc.

- Méthodes (ou Fonctions) : Ce sont des fonctions définies à l'intérieur d'une classe. Elles déterminent le comportement ou les actions que les objets de cette classe peuvent exécuter. Par exemple, dans la classe Voiture, on pourrait avoir des méthodes pour démarrer la voiture, accélérer, freiner, etc.

En résumé, une classe en JAVA est un cadre structurel qui combine des données (attributs) et des fonctions (méthodes) en un seul bloc constructif logique. Elle est la pierre angulaire de la POO, permettant de créer des objets complexes avec des comportements spécifiques et une interaction structurée.

Encapsulation

L'encapsulation est un principe fondamental de la programmation orientée objet (POO). Elle implique le regroupement des données (attributs) et des méthodes (fonctions) qui opèrent sur ces données au sein d'une même unité, la classe. L'encapsulation permet de contrôler l'accès aux données d'un objet en rendant les attributs privés et en fournissant des méthodes publiques pour acceder (getters) et modifier (setters) ces données.

Pourquoi l'Encapsulation?

1. **Sécurité des Données** : Empêche l'accès direct aux attributs internes d'un objet, protégeant ainsi les données contre des modifications inattendues ou incorrectes.
2. **Contrôle des Données** : Permet de valider les données avant de les modifier, évitant ainsi des états invalides ou incohérents dans l'objet.
3. **Flexibilité et Maintenance** : Facilite la modification et la maintenance du code, car les changements dans la représentation des données n'affectent pas les parties du programme qui utilisent ces données.

Constructeurs

Un constructeur est une méthode spéciale d'une classe en JAVA qui est automatiquement appelée lorsqu'un objet de cette classe est créé. Son rôle principal est d'initialiser l'objet, c'est-à-dire de définir un état initial cohérent pour ses attributs.

Caractéristiques Clés des Constructeurs :

- Ils portent le même nom que la classe à laquelle ils appartiennent.
- Ils ne possèdent pas de type de retour, pas même void.
- Ils peuvent prendre des paramètres pour permettre l'initialisation de l'objet avec des valeurs spécifiques.
- Ils peuvent être surchargés, ce qui signifie qu'une classe peut avoir plusieurs constructeurs, chacun ayant un ensemble différent de paramètres.

Pourquoi Utiliser les Constructeurs ?

Les constructeurs sont essentiels pour plusieurs raisons :

- **Initialisation Sûre et Contrôlée** : Ils garantissent que chaque objet démarre sa vie dans un état valide et cohérent.
- **Simplicité de Création d'Objets** : Ils simplifient le code en automatisant l'initialisation, rendant ainsi le code plus propre et plus compréhensible.
- **Flexibilité** : Grâce à la surcharge, différents types d'initialisation peuvent être définis pour un même objet, offrant ainsi une grande flexibilité dans la manière dont les objets sont créés.

Types de Constructeurs en JAVA

Constructeur par Défaut

Définition :

- Un constructeur par défaut est un type de constructeur qui ne prend aucun paramètre. Il est automatiquement appelé lorsqu'un objet d'une classe est créé sans spécifier de paramètres.

Utilisation :

- Ce constructeur est utilisé pour initialiser un objet avec des valeurs par défaut. Si aucun constructeur par défaut n'est explicitement défini dans une classe, le compilateur en JAVA en fournit un implicitement.
- Il est particulièrement utile pour garantir que les objets d'une classe commencent leur existence dans un état valide et prédictible.

```
public class Voiture {
    private String marque;
    private String modele;

    public Voiture() {
        this.marque = "Inconnue";
        this.modele = "Inconnu";
    }
}
```

Types de Constructeurs en JAVA

Constructeur Paramétré

Définition :

- Un constructeur paramétré est un constructeur qui accepte un ou plusieurs paramètres. Ces paramètres permettent de passer des valeurs spécifiques pour initialiser les attributs de l'objet lors de sa création.

Utilisation :

- Ce type de constructeur est utilisé pour créer des objets avec des états initiaux spécifiques. Il offre une flexibilité accrue en permettant la personnalisation des objets dès leur création.
- Il est indispensable dans des situations où les valeurs par défaut ne suffisent pas ou ne sont pas souhaitables pour l'initialisation des objets.

```
public class Voiture {
    private String marque;
    private String modele;

    public Voiture(String m,
String mod) {
        this.marque = m;
        this.modele = mod;
    }
}
```

Exemple d’utilisation

```java
import java.util.List;

public class Voiture {
    private String marque;
    private String modele;

    public Voiture() {}

    public void setMarque(String m) {
        marque = m;
    }

    public void setModele(String mod) {
        modele = mod;
    }

    public void afficherInfos() {
        System.out.println("Marque: " + marque + ", Modèle: " + modele);
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        List<Voiture> voitures = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);
        String marque, modele;

        for (int i = 0; i < 3; i++) {
            Voiture voiture = new Voiture();
            System.out.println("Saisir les informations pour la voiture " + (i + 1) + ":");
            System.out.print("Marque: ");
            marque = scanner.next();
            voiture.setMarque(marque);

            System.out.print("Modèle: ");
            modele = scanner.next();
            voiture.setModele(modele);

            voitures.add(voiture);
        }

        System.out.println("\nInformations des voitures saisies :");
        for (Voiture voiture : voitures) {
            voiture.afficherInfos();
        }
    }
}
```

Exemple d'utilisation

```java
public class Voiture {
    private String marque;
    private String modele;
    public Voiture(String m, String mod) {
        this.marque = m;
        this.modele = mod;
    }
    public void afficherInfos() {
        System.out.println("Marque: " + marque + ", Modèle: " + modele);
    }
}
```

```java
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        List<Voiture> voitures = new ArrayList<>();
        Scanner scanner = new Scanner(System.in);
        String marque, modele;

        for (int i = 0; i < 3; i++) {
            System.out.println("Saisir les informations pour la voiture " + (i + 1) + ":");
            System.out.print("Marque: ");
            marque = scanner.next();
            System.out.print("Modèle: ");
            modele = scanner.next();
            voitures.add(new Voiture(marque, modele));
        }

        System.out.println("\nInformations des voitures saisies :");
        for (Voiture voiture : voitures) {
            voiture.afficherInfos();
        }
    }
}
```

Exercice d’application

Développer une classe Personne en JAVA avec quatre attributs : id, nom, prenom, et email. L'id doit s'incrémenter automatiquement pour chaque nouvelle instance. L'email doit être généré automatiquement sous la forme prenom.nom@mail.com. Implémenter une fonction pour saisir les informations sans utiliser de constructeur avec arguments. Ensuite, créer une liste de quatre personnes, saisir leurs informations, et afficher ces informations.

Instructions

Définir la Classe Personne :

- Attributs privés : id, nom, prenom, et email.
- Constructeur par défaut.
- Getters pour chaque attribut.
- Méthode saisirInfos() de type **void** pour la saisie des informations.
- Variable statique pour l'auto-incrémentation de l'id.
- Génération automatique de l'email dans saisirInfos().

Écrire les Méthodes de la Classe :

- Le constructeur initialise l'id et prépare l'email.
- saisirInfos() permet de saisir nom et prenom, et génère l'email.
- Les getters retournent les valeurs des attributs.

Utiliser la Classe dans main() :

- Créer une liste (ou un vecteur) pour stocker quatre objets Personne.
- Saisir les informations pour chaque personne.
- Afficher les informations de chaque personne.

Heritage

L'héritage en Java est un concept fondamental de la programmation orientée objet. Il permet à une classe de dériver des caractéristiques (comme les méthodes et les variables membres) d'une autre classe. Voici les points essentiels à comprendre sur l'héritage en Java :

Classes de Base et Dérivées : Dans l'héritage, on a une classe de base (parent) et une ou plusieurs classes dérivées (enfants). La classe dérivée hérite des propriétés et comportements de la classe de base.

Types d'Héritage :

- Héritage simple : Une classe dérivée hérite d'une seule classe de base.

Accès aux Membres Hérités : Les spécificateurs d'accès (public, protected, private) déterminent comment les membres de la classe de base sont accessibles dans la classe dérivée.

Heritage

Dans la programmation orientée objet en Java la classe de base (aussi appelée classe parente ou superclasse) est la classe dont d'autres classes (classes dérivées ou sous-classes) héritent des propriétés et des comportements. L'héritage permet aux classes dérivées d'utiliser et de modifier les caractéristiques définies dans la classe de base, facilitant ainsi la réutilisation du code et l'organisation hiérarchique des classes.

Classe de Base

- **Définition** : Contient des propriétés (variables membres) et des comportements (fonctions membres) communs qui peuvent être hérités par d'autres classes.
- **Rôle** : Fournir un ensemble de fonctionnalités de base que d'autres classes peuvent étendre ou modifier.
- **Exemple** : Une classe Véhicule qui définit des propriétés communes à tous les véhicules comme vitesseMax, couleur, et des méthodes comme démarrer() ou arrêter().

Heritage

Classe Dérivée

- **Définition** : Hérite des propriétés et comportements de la classe de base et peut ajouter ses propres propriétés et comportements ou modifier ceux hérités.
- **Rôle** : Spécialiser ou étendre les fonctionnalités de la classe de base.
- **Exemple** : Une classe Voiture dérivée de Véhicule, ajoutant des propriétés comme nombreDePortes et redéfinissant la méthode démarrer() pour inclure l'activation du système de divertissement.

Heritage

```
// Définition de la classe de base
class Base {
    // Code de la classe de base
}

// Définition de la classe dérivée
class Derived extends Base {
    // Code de la classe dérivée
}
```

Heritage

```
public class Vehicule {

   protected int vitesseMax;

   protected String couleur;


   public Vehicule(int vmax, String c) {

        vitesseMax = vmax;

        couleur = c;

   }

   public void demarrer() {

        System.out.println("Véhicule démarré");

   }


   public void arreter() {

        System.out.println("Véhicule arrêté");

   }

   public void afficherInformations() {

        System.out.println("Véhicule:\n  Couleur:
" + couleur + "\n  Vitesse Max: " + vitesseMax +
" km/h");

    }

}
```

Heritage

```
public class Voiture extends Vehicule {

   private int nombreDePortes;


      public  Voiture(int  vmax,  String  c,  int
portes) {

       super(vmax, c);

       nombreDePortes = portes;

   }


   @Override

   public void demarrer() {

       super.demarrer();

                System.out.println("Système   de
divertissement activé");

   }
```

```
   @Override

    public void afficherInformations() {

        super.afficherInformations();

         System.out.println("  Nombre de portes: "
+ nombreDePortes);

    }

}
```

Heritage

```
public class Main {

   public static void main(String[] args) {

        Vehicule monVehicule = new Vehicule(150,
"rouge");

       monVehicule.demarrer();

       monVehicule.afficherInformations();

       monVehicule.arreter();


          Voiture maVoiture = new Voiture(200,
"bleu", 4);

       maVoiture.demarrer();

       maVoiture.afficherInformations();

       maVoiture.arreter();

   }

}
```