2024 -2025

TP 4
Interfaces et polymorphisme

Exercice 1

Vous êtes chargé de concevoir un système de gestion des paiements dans un magasin en ligne. Le magasin accepte plusieurs types de paiements : par **carte de crédit**, par **PayPal**, et par **cryptomonnaie**.

1. Créez une **interface** Paiement qui définit une méthode ***effectuerPaiement(double montant)***. Cette méthode prend en entrée le montant du paiement et retourne un message indiquant que le paiement a été effectué.
2. Implémentez cette interface dans trois classes :
    - PaiementCarteCredit : Simule un paiement par carte de crédit: retourne un message

      "Paiement de " + montant + "€ effectué par Carte de Crédit.";

    - PaiementPayPal : Simule un paiement via PayPal.(retourne un autre message)
    - PaiementCryptomonnaie : Simule un paiement en cryptomonnaie (retourne un autre message)
3. Créez une classe de Test qui contient une méthode main. Créez un tableau d'objets de type Paiement, **où chaque élément est une instance de l'une des classes de paiement**. Utilisez ensuite une boucle pour effectuer un paiement avec chaque méthode. Interprétez le résultat de l'exécution de votre programme.

Exercice 2

PARTIE 1

1- Créez une **classe** Forme avec un attribut couleur de type String et une méthode abstraite calculerSurface() qui sera implémentée par les sous-classes.

2- Créez trois sous-classes qui héritent de la classe Forme.

- **Carre** : un attribut cote (double) pour représenter le côté du carré et attribut couleur.
- **Cercle** : un attribut rayon (double) pour représenter le rayon du cercle et attribut couleur.
- **Triangle** : Contient des attributs base (double) et hauteur (double) pour représenter la base et la hauteur du triangle et attribut couleur.

  3 - Implémentez la méthode calculerSurface() dans chaque sous-classe pour retourner la surface de la forme respective.

- Dans le programme principal (Créez une classe test qui contient la méthode main) : créez des objets de type Forme mais instanciés avec les sous-classes Carre, Cercle, et Triangle, puis utilisez le polymorphisme pour calculer et afficher les surfaces de chaque forme.

4- Testez le code qui permet de :
- créer 2 objets nommés: unCercle et unCarre.
- Les afficher directement avec:

  System.out.println (unCercle);

  System.out.println (unCarre) ;
- Notez le résultat affiché.

PARTIE 2 la méthode toString()

5 - Redéfinir la méthode *public String toString()* dans les deux classes Carre et Cercle qui affiche:
- “Le cercle est de couleur …… sa surface est ….. cm carré”
- ou “Le carré est de couleur …… sa surface est ….. cm carré”

6- Testez le même code de la question 4. Que remarquez-vous?

- Avant de redéfinir la méthode toString(), lorsque vous appelez System.out.println(unCercle);, **Java utilise la méthode toString() héritée de la classe Object,** ce qui affiche le nom complet de la classe et son identifiant mémoire.

- Après la redéfinition de la méthode toString() dans les classes Carre et Cercle, l'affichage devient plus lisible et personnalisé, avec des informations sur la couleur et la surface de la forme.

- Pensez à redéfinir la méthode toString() dans vos classes, c'est une bonne habitude.

PARTIE 3 La méthode equals()

7 - On veut comparer deux cercles en utilisant la méthode **equals.**

La méthode equals() en Java est utilisée pour comparer deux objets.

Dans la méthode main, ajoutez le code suivant:

```
Cercle cercle1 = new Cercle("Rouge", 5);
Cercle cercle2 = new Cercle("Rouge", 5);
Cercle cercle3 = new Cercle("Bleu", 7);

// Comparaison des cercles avant redéfinition de equals()
System.out.println("Comparaison cercle1 et cercle2 (avant redéfinition equals) : " +
cercle1.equals(cercle2));

System.out.println("Comparaison cercle1 et cercle3 (avant redéfinition equals) : " +
cercle1.equals(cercle3));
}
```

Notez le résultat de l'execution.

```
Comparaison cercle1 et cercle2 (avant redéfinition equals) : false
Comparaison cercle1 et cercle3 (avant redéfinition equals) : false
```

8 - Redefinir la méthode equals pour permettre de comparer deux cercles de la façon suivante:

deux cercles sont égaux s'ils ont la mêmes surfaces et la même couleur.

Testez le code dans la méthode main. le résultat d'execution attendu est le suivant:

Comparaison cercle1 et cercle2 (après redéfinition equals) : **true**
Comparaison cercle1 et cercle3 (après redéfinition equals) : false

La méthode equals() en Java est utilisée pour comparer deux objets afin de vérifier s'ils sont **égaux** en termes de contenu (plutôt qu'en termes de référence mémoire). Par défaut, la méthode equals() de la classe Object compare uniquement les références d'objets, c'est-à-dire si deux objets pointent vers la même zone mémoire.

- **Pourquoi redéfinir equals() ?**

La redéfinition de equals() permet de comparer deux objets en fonction de leurs attributs plutôt que de leurs références mémoire.

Par exemple, si deux objets Cercle ont le même rayon et la même couleur, nous pourrions vouloir les considérer comme égaux même s'ils sont des instances différentes en mémoire.