TP 3 — Héritage, polymorphisme

Exercice 1 – Comptes bancaires

Question 1

On veut créer :

— une classe nommée `Compte` permettant de définir un compte en banque par son identificateur (une chaîne de caractères) et son solde (un réel). L'identificateur sera non-modifiable. La classe est dotée de constructeurs, des méthodes d'accès aux attributs et d'une méthode `getNatureCompte()` qui retourne la chaîne de caractères "compte simple".

— une sous-classe `CompteEpargne` (héritant donc de la classe `Compte`). Cette classe redéfinit la méthode `getSolde()` pour retourne le solde original auquel sont ajoutés des intérêts capitalisés. Cette classe possède deux attributs privés supplémentaires qui sont le taux d'intérêt annuel (réel) et le nombre d'années d'épargne (entier) et des méthodes d'accès aux nouveaux attributs. La méthode `getNatureCompte()` retourne "compte avec intérêt".

Écrire le code des classes `Compte` et `CompteEpargne` pour qu'il soit compatible avec le code de la classe ci-dessous (nommée `TestCalculInterets`). Tester votre programme.

```
class TestCalculInterets
{
  public static void main(String [] args)
  {
    Compte compte1 = new Compte ("A01", 1000f);
    CompteEpargne compte2;
    compte2 = new CompteEpargne ("E99", 1000f, 0.1f);
    compte2.setAnnees(5);
    Compte c;
    String message = "L'argent qui dort ne rapporte rien :";
    c = compte1;
    message += "\n Solde compte "
               + c.getIdentifiant() + " : "
               + c.getSolde() + " "+c.getNatureCompte();
    c = compte2;
    message += "\n Solde compte "
               + c.getIdentifiant() + " : "
               + c.getSolde() + " "+c.getNatureCompte();
    System.out.println(message);
  }
}
```

Question 2

Écrire une méthode `creerCompte()` permettant de saisir un compte ou un compte épargne avec les valeurs correspondantes. Cette méthode retournera systématiquement un objet de la classe `Compte`.

Question 3

Créer un tableau de `Compte` qui contiendra des références sur des objets de la classe `Compte` et des objets de la classe `CompteEpargne`.

Parcourir le tableau et afficher pour chaque compte le solde et la nature du compte. Vérifier que l'affichage est correcte en fonction de la nature du compte

Exercice 2 – Bâtiments et impôts

Une mairie veut informatiser le calcul des impôts locaux pour chaque bâtiment (maison individuelle ou usine).

Question 1

Créer une classe `Batiment`. Ses attributs sont le nom du propriétaire, l'adresse du bâtiment et la surface. Ses méthodes sont le calcul de l'impôt (5 euros/m^2) et l'affichage des attributs (méthode `affiche`). Le constructeur permet d'initialiser les attributs. Le constructeur doit obliger de spécifier l'ensemble des valeurs des attributs. La classe propose une méthode `getCategorie()` qui renverra la valeur 0.

Question 2

Créer la classe `Usine`. Une usine est un bâtiment qui est occupé par une entreprise (d'où un attribut supplémentaire pour le nom de l'entreprise). Écrire le constructeur (tous les attributs obligatoires) et la nouvelle méthode `affiche`. La classe propose une méthode `getCategorie()` qui renverra la valeur 1.

Question 3

Créer la classe `Villa`. Elle spécialise un bâtiment avec deux attributs supplémentaires : le nombre de pièces et la présence (ou non) d'une piscine. Le calcul de l'impôt est différent : 100 € par pièce et 500 € pour une piscine. Les caractéristiques supplémentaires doivent être affichées dans la méthode `affiche`. Un second constructeur permet d'initialiser le nombre de pièces à 4 et la présence d'une piscine à faux par défaut. La méthode this(...) qui permet d'appeler le constructeur de la même classe **doit être utilisée** avec ce second constructeur. La classe propose une méthode `getCategorie()` qui renverra la valeur 2.

Question 4

Ajouter deux attributs à la classe `Usine` : le nombre d'employés et le nombre de livraisons par jour. Écrire une méthode `FluxVehicule` permettant de calculer le flot moyen de véhicule que l'usine induit (on estime à 75% le nombre d'employés venant en voiture personnelle).

Question 5

Écrire chaque méthode `equals` pour ces 3 classes. Elles doivent avoir obligatoirement les prototypes suivants :

```
public boolean equals (Object obj) {\ldots }
```

La mairie veut faire des statistiques sur l'impôt perçu. Réaliser une nouvelle classe permettant de réaliser les actions suivantes :

- Créer un ensemble de constructions en affectant les valeurs que vous voulez. Les constructions seront stockées dans un tableau de bâtiment (`Batiment[]`). Par exemple un tableau de 6 éléments dans lequel il y aura : 2 instances de `Batiment`, 2 instances d'`Usine` et deux instances de `Villa`.
- Écrire une méthode retournant l'impôt moyen perçu sur les constructions dans le tableau.
- Écrire une méthode retournant la référence de la construction rapportant le plus d'argent.
- Écrire une méthode affichant l'impôt moyen perçu par catégorie de construction.
- Écrire une méthode testant s'il y a des instances dans le tableau qui sont égales.

Réfléchir au mécanisme mis en œuvre lors de l'exécution du programme.