Université Paris Cité **Programmation Orientée Objet**
L2 **Année 2023-2024**

TD - Séance n° 4

Héritage

Exercice 1 *Constructeurs et héritage*
On définit les classes suivantes :

```
1  class C {
       public C() { System.out.println("Hello"); }
3  }
   class D extends C {
5      private int x;
       public D(int x) { this.x = x; }
7  }
```

1. Quel est le résultat de `new C();` ? De `new D()` ?
2. Peut-on définir les classes suivantes ? Pourquoi ?

```
class E extends C {}
class F extends D {}
```

Exercice 2 *Héritage et surcharge*
On définit les classes `A` et `B` de la manière suivante :

```
public class A {
2    public void f(A x) {
       System.out.println("A.f(A)");
4    }
     public void g() {
6      f(new A());
     }
8  }
   public class B extends A {
10   @Override
     public void f(A x) {
12     System.out.println("B.f(A)");
     }
14   // Surcharge
     public void f(B x) {
16     System.out.println("B.f(B)");
     }
18 }
```

On construit des objets `a`, `b` et `c` :

```
A a = new A();
B b = new B();
A c = new B();
```

Qu'affichent les instructions suivantes ?

```
a.g();
b.g();
c.g();
```

```
a.f(a);
a.f(b);
a.f(c);
b.f(a);
b.f(b);
b.f(c);
c.f(a);
c.f(b);
c.f(c);
```

Que deviennent les questions précédentes si les classes `A` et `B` sont définies comme suit ?

```
1  public class A {
     public void f(A x) {
3      System.out.println("A.f(A)");
     }
5    //Surcharge
     public void f(B x) {
7      System.out.println("A.f(B)");
     }
9    public void g() {
       f(new A());
11   }
}
13 public class B extends A {
     @Override
15   public void f(A x) {
       System.out.println("B.f(A)");
17   }
     @Override
19   public void f(B x) {
       System.out.println("B.f(B)");
21   }
}
```

Exercice 3 *Héritage de méthodes statiques*
Le but de cet exercice est de modéliser une serre. Une serre est un tableau d'objets

de type `Plante`. Certaines plantes ont besoin d'un apport en engrais supplémentaire et seront de type `PlanteFleurie`. Des variables de classe vont stocker le nombre total de plantes qui ont besoin d'engrais et/ou d'être arrosées (en pratique ce n'est pas forcément la meilleure manière de procéder, mais nous ferons comme cela pour les besoins de l'exercice). Le début des classes est donné par :

```
public class Plante {
    private static int nbPlanteSoif;
    private boolean arrosee=false;
```

```
public class PlanteFleurie extends Plante {
    private static int nbPlanteSansEngrais;
    private boolean engrais=false;
```

1. Écrivez les constructeurs adéquats.
2. Écrivez les méthodes `toString` de chacune des classes.
3. Écrivez dans chaque classe une méthode `Etat` qui renseigne sur le nombre de plantes à arroser ou auxquelles il manque de l'engrais.
4. Dans chaque classe écrivez une méthode **statique** pour arroser un objet de la classe même. Lors de l'arrosage, contrairement à une `Plante` simple, une `PlanteFleurie` reçoit également de l'engrais.
5. Qu'affiche le code suivant ?

```
Plante[] serre=new Plante[5];
serre[0]=new Plante();
serre[1]=new PlanteFleurie();
serre[2]=new Plante();
serre[3]=new PlanteFleurie();
serre[4]=new Plante();
System.out.println(Plante.Etat());
System.out.println(PlanteFleurie.Etat());
for(int i=0;i<5;i++){
    Plante.arrosage(serre[i]);
    System.out.println(serre[i]);
}
System.out.println(Plante.Etat());
System.out.println(PlanteFleurie.Etat());
```

   Que se passe-t-il si `Plante.arrosage(serre[i])` est remplacé par `PlanteFleurie.arrosage(serre[i]);` ?

6. Proposez une solution pour que la méthode `arrosage` ait l'effet attendu (c'est-à-dire pour qu'elle arrose toutes les plantes et donne de l'engrais à toutes les plantes fleuries).

Exercice 4 *Héritage - Immobilier*

On cherche à modéliser un patrimoine immobilier. Tous les attributs seront privés, on créera des accesseurs selon les besoins.

1. Écrivez une classe `Batiment` avec deux variables `adresse` et `surfaceHabitable` (un `double`) et son constructeur `Batiment(String adresse, double surface)`. Implémentez la méthode `String toString()`, ainsi qu'un accesseur `getSurfaceHabitable()`.
2. Écrivez une classe `Maison` héritant de `Batiment` avec les attributs `nbPieces` et `surfaceJardin`. Écrivez le constructeur `Maison(String adresse, double surfaceH, double surfaceJ, int nbPieces)`, ainsi qu'un accesseur `getSurfaceJardin()`. Écrivez aussi la méthode `String toString()` en utilisant un appel à `super.toString()`.
3. Écrivez une méthode `main` dans une classe `TestBatiment`. Ce programme doit instancier un bâtiment, deux maisons et les afficher.

   Ensuite, créez un tableau de 10 bâtiments. Est-ce que les bâtiments sont instanciés ?

   Remplacez 2 éléments du tableau par les deux maisons précédemment créées. Qu'affichera `System.out.println` appelée sur chaque élément du tableau ?
4. Écrivez une méthode statique `surfaceHabitableTotale(Batiment[] tabBat)` dans la classe `TestBatiment` qui prend en argument un tableau de bâtiments (avec éventuellement des cases vides) et calcule la somme des surfaces habitables des bâtiments.
5. On souhaite ajouter dans la classe `TestBatiment` une méthode statique `surfaceJardinTotale(Batiment[] tabBat)` qui calcule, comme precedemment, la somme des surfaces des jardins des bâtiments donnés en argument. Quel problème rencontre-t'on ? Proposez une solution et implémentez là.
6. L'impôt local d'un bâtiment est calculé selon la formule

   $$\text{Impot} = \text{TauxA} \times \text{surfaceHabitable} + \text{TauxB} \times \text{surfaceJardin}$$

   Les valeurs de cette année étant TauxA = 5.6 et TauxB = 1.5.

   Où déclarer les variables TauxA et TauxB, et comment ? Dans quelle(s) classe(s) faut-il implémenter la méthode `double impot()` ?

Si vous avez le temps

Exercice 5 *Héritage - Immobilier (suite)*

1. Écrivez une classe Personne. Une personne a un nom et possède une certaine somme d'argent (en centimes d'euros).
2. Tous les bâtiments ont un propriétaire. Par contre, seules les maisons peuvent être louées, mais ce n'est pas automatique (une maison peut être `louable` ou non).

   Modifiez vos classes pour qu'on puisse trouver le propriétaire d'un bâtiment, le locataire d'une maison ainsi que les différents biens immobiliers dont est propriétaire ou locataire une personne.
3. Chaque maison a un loyer. Écrire une méthode qui fait payer à un locataire son loyer et le reverse au propriétaire. (On permettra aux personnes d'avoir des dettes)
4. Écrivez une méthode qui fait payer à un propriétaire ses impôts sur ses biens. L'impot sur un bien depend de la catégorie du bien : une maison louable a un impôt plus élevé qu'une maison non louable qui a un impôt plus élevé qu'un bâtiment autre qu'une maison.
5. Dans une classe de test écrivez une méthode `chercherLocation` qui reçoit en paramètre un tableau de maisons louables, un futur locataire et un budget. La méthode loge la personne dans une maison louable sans locataire actuel dont le loyer est inférieur au budget, ou renvoie `false` s'il n'y a pas de telle maison.