Université de Paris **Programmation Orientée Objet**
L2 **Année 2021-2022**

TD - Séance n°2

Révisions – Classes, Modélisation

Exercice 1 *Variables et méthodes statiques*
On définit une classe A par le code suivant.

```
1  public class A {
       public static int a = 3;
3      public int b;

5      public A (int c) {
           this.b = c;
7      }

9      public void g() {
           a = a+1;
11         b = b+1;
       }
13 }
```

1. On définit une méthode statique h dans la classe A. Quels champs peut-elle utiliser ? Même question pour une méthode non statique.
2. On utilise la classe A dans une classe Test, comme ci-dessous. Qu'obtient on à l'exécution ?

```
1  public class Test {
       public static void main(String[] args) {
3          A u = new A(0);
           A v = new A(0);
5          u.g();
           System.out.println("u: a=" + u.a + "; b=" + u.b);
7          v.g();
           System.out.println("v: a=" + v.a + "; b=" + v.b);
9          u.g();
           System.out.println("u: a=" + u.a + "; b=" + u.b);
11     }
   }
```

Exercice 2 *Passage d'argument*

1. Qu'obtient-on à l'exécution du code suivant ?

```
public class Test2 {
    public static int g(int i) {
        i = i+1;
        return i;
    }
    public static void main(String[] args) {
        int i = 0;
        g(i);
        System.out.println(i);
    }
}
```

2. On suppose maintenant avoir la classe C suivante (elle encapsule un entier).

```
public class C {
    private int a;

    public C(int a) {
        this.a = a;
    }

    public String toString() {
        return Integer.toString(a);
    }
    public void setNumber(int j) {
        this.a = j;
    }
}
```

— Qu'obtient on si on exécute le code suivant ?

```
public class Test3a {
    public static void h(C k) {
        k.setNumber(5);
    }
    public static void main(String[] args) {
        C k = new C(0);
        h(k);
        System.out.println(k);
    }
}
```

— Et si on exécute le code suivant ?

```
public class Test3b {
    public static C h(C k) {
        k = new C(5);
        return k;
    }
    public static void main(String[] args) {
```

```
        C k = new C(0);
        C l = h(k);
        System.out.println("k=" + k + ", l=" + l);
    }
}
```

3. Expliquez en quoi les exemples précédents illustrent le passage par valeur utilisé par java.

Exercice 3 *L'horloge – Compteur cyclique*

Le but de cet exercice et du suivant est de définir une classe pour créer des horloges.

Un *compteur cyclique* est un compteur de nombres entiers ayant une valeur maximale **fixe** qu'il peut atteindre. Il s'initialise à zéro ; il possède la fonctionnalité `incrementer` qui lui fait augmenter sa valeur courante d'une unité. (Du moins tant qu'il n'a pas atteint sa valeur maximale, et si c'est le cas le compteur revient à 0).

1. Déterminer les champs de la classe `CompteurCyclique`. Quels champs doivent être définis `final` ?
2. Créer un constructeur `public CompteurCyclique(int max)` qui produit une instance de valeur maximale `max` et initialise sa valeur courante à 0. Ecrivez un autre constructeur qui permette en plus d'initialiser la valeur courante du compteur cyclique à une autre valeur passée en argument. Remarquez que vous pouvez faire dépendre les constructeurs l'un de l'autre.
3. Ajouter une méthode de type *getter*, en choisissant un nom adéquat, qui permette d'obtenir la valeur courante du compteur.
4. Peut-on définir une méthode de type *setter* pour la valeur maximale ?
5. Définir les méthodes suivantes :
   — `public void reinitialiser()` qui met la valeur courante du compteur à zéro.
   — `public boolean incrementer()` qui augmente la valeur courante du compteur de la manière décrite dans la définition ci-dessus. La méthode retourne `true` dans le cas de réinitialisation, `false` autrement.
6. Redéfinir la méthode `public String toString()` pour qu'elle affiche la valeur du compteur en utilisant deux chiffres (si la valeur est <10, ajouter un 0 devant la valeur).

   • Facultatif : Plutôt que 2 chiffres, on peut utiliser le nombre de chiffres de la valeur maximale, en rajoutant autant de zéros que nécessaire.

   • Remarque : Penser à utiliser la méthode `String.format`. Par example, `String.format("%05d", 24);` affiche 00024. Pour trouver le nombre de chiffres d'un entier n, utiliser soit `Integer.toString(n).length()`, soit `String.valueOf(n).length()`, soit `(int)Math.log10(n)+1`.

Exercice 4 *L'horloge – La classe principale*

On peut voir une horloge comme une combinaisons des objets que l'on vient de définir. En effet les *heures* et les *minutes*, sont des cas particuliers de `CompteurCyclique`.

1. Définissez et donnez des constructeurs à une classe Horloge.
2. Définissez une méthode setter `public void setHeure(int h, int m)`.
3. Surchargez la méthode `setHeure` pour qu'elle modifie les heures. Peut-on surcharger la méthode à nouveau pour modifier les minutes ?
4. Définissez une méthode `public void ticTac()` qui simule le passage d'un minute. (Utilisez la méthode `public void incrementer()` de la classe `CompteurCyclique`)
5. Redéfinir la méthode `public String toString()` pour qu'elle affiche l'heure actuelle de l'horloge.

Si vous avez le temps

Ces exercices de modélisation vous sembleront ouvert. L'intérêt est de ne pas avoir de solution à priori et d'explorer des approches différentes, en discutant leurs avantages, leurs inconvénients et leur réalisme. Vous pourrez terminez chez vous.

Exercice 5 *Le parking - Ticket*

On veut modéliser le fonctionnement des parkings automobiles, qui se caractérisent par leurs :

— nombre de places
— nombre de places libres
— emplacements (l'endroit où se trouvent les places)
— pourcentage de remplissage

Lorsqu'une voiture entre dans un parking, elle reçoit un ticket qu'elle devra rendre en sortant.

1. Discutez la nature des emplacements et proposez plusieurs modélisations dans le cadre du parking. Une fois fixée la nature des emplacements, que pouvez vous dire des autres caractéristiques d'un parking ?
2. Si on s'intéresse plus particulièrement aux objets qui représentent les tickets, on peut décider que c'est lorsqu'elle rentre dans un parking qu'une voiture reçoit un ticket, et que s'il n'y a pas de place elle n'en recevra pas. Quelles propositions parmi les suivantes pourriez vous retenir ?
   (a) avoir un constructeur `Ticket(int n,Parking p)`
   (b) avoir un constructeur `Ticket(Parking p)`
   (c) avoir une méthode `Ticket getTicket()` dans la classe *Voiture*
   (d) avoir une méthode `Ticket entreParking(Parking p)` dans la classe *Voiture*

(e) avoir une méthode `Ticket entreParking(Voiture v)` dans la classe *Parking*

(f) proposez une autre méthode et l'évaluer.

Exercice 6 *Le parking - Places libres*

Dans cet exercice, on supposera la classe `Voiture` déjà écrite et on ne se préoccupera pas du ticket. Notre objectif est d'optimiser la recherche d'une place libre. Ainsi on aura :

```
import java.util.LinkedList;
public class Parking {
    private final Voiture[] places;
    private final LinkedList<Integer> libres;
}
```

L'emplacement d'une voiture sera confondu avec l'indice qu'elle occupe dans le tableau `places`, et la liste `libres` contiendra l'ensemble des emplacements encore libres.

Rappels sur `LinkedList<T>` et `Integer` :

— Les listes sont associées à un type, ce qui assure l'uniformité des objets contenus

— La classe qui permet de manipuler des `int` non pas comme type primitif mais comme des objets est `Integer`. Vous pouvez utiliser le constructeur `Integer(int)` et la méthode `int intValue()` de la classe `Integer` pour passer explicitement d'un `int` à un `Integer` et réciproquement.

— Sur les `LinkedList<Integer>` vous pourrez utiliser les méthodes : `boolean add(Integer)`, `boolean isEmpty()`, `Integer removeFirst()` et le constructeur `LinkedList()`

Dans la classe `Parking` :

1. Pourquoi les champs sont-ils déclarés `final` ? Le nombre de voiture dans le parking pourra-t-il quand même varier ?
2. Proposez un constructeur `Parking (int n)` où l'entier n désigne la taille du parking
3. Ecrivez une méthode `public int prendPlace(Voiture x)` qui place la voiture x à un endroit libre, et retourne cet emplacement
4. Une méthode `public Voiture exitPlace(int n)` qui fait sortir la voiture qui se trouve à l'emplacement n

Exercice 7 *Le parking - Emplacements*

Dans cet exercice on choisit de ne pas utiliser de `LinkedList` mais de coder la liste des places libres dans l'objet `Emplacement`.

En voici le fonctionnement : un emplacement se définit par une référence vers un véhicule, et par le numéro du prochain emplacement libre. Un parking se compose

d'un tableau d'emplacements, et d'une variable indiquant un emplacement libre particulier, duquel on peut déduire un second emplacement libre etc...

On pourra utiliser -1 pour identifier un numéro d'emplacement non valable.

1. Ecrivez une modélisation de la classe `Emplacement` :
   - un constructeur `Emplacement(int x)` qui prend en argument le numéro du prochain emplacement libre.
   - une méthode `void setNextLibre(int x)` qui permet de modifier cet attribut
   - un accesseur `int getNextLibre()`
   - une méthode `void positionne(Voiture v)`
   - une méthode `Voiture setFree(int n)` qui en plus de libérer la voiture prend n pour prochain emplacement libre.
2. Ecrivez une modélisation de la classe `Parking` :
   - un constructeur `Parking(int taille)`.
   - `int prendPlace(Voiture x)`
   - `Voiture exitPlace(int n)`