SCIENCES SORBONNE UNIVERSITÉ

LU2IN002-2022oct
Éléments de programmation
par objets avec Java

Examen du 16 juin 2023 – Durée : 2 heures

Seul document autorisé : **feuille A4 manuscrite, recto-verso.**
Pas de calculatrice ou téléphone. Barème indicatif sur 43.

Partie 1 Exercices (14 points)

Exercice 1 (5 points) Compter des trucs...

Soit le programme suivant :

```
1  public class A { }
2  public class Truc {
3      private A a1, a2;
4      public Truc() {a1=new A(); a2=new A
           ();}
5      public Truc(A a) { a1=a; a2=a; }
6      public void empty() {a1=null; a2=
           null;}
7  }
```

```
8  public class Test {
9      public static void main(String []
           args){
10         Truc t1=new Truc();
11         Truc t2=t1;
12         Truc t3=new Truc(new A());
13         t2.empty();
14         t3=null;
15     }
16 }
```

Q1.1 Combien y-a-t-il d'instances de la classe `Truc` créées ? Donner le numéro des lignes où sont créées chacune de ces instances.

Q1.2 Mêmes questions pour les instances de la classe `A` créées.

Q1.3 Faire un schéma des *handles* et des objets dans la mémoire après l'exécution de la ligne 12.

Q1.4 Après l'exécution de la ligne 14 et avant la fin du programme, donner le nombre d'instances de la classe `Truc` et le nombre d'instances de la classe `A` présentes. Expliquer.

Exercice 2 (5 points) Cocotiers et noix de coco.

On considère les classes suivantes :

```
1  public class NoixDeCoco {
2      private double poids;
3      public NoixDeCoco(){ poids=Math.random()*1000+500; } }
4  public class Cocotier {
5      private int taille;
6      private NoixDeCoco [] tab;
7      public Cocotier(int taille,int nbNoix) {
8          this.taille=taille;
9          tab=new NoixDeCoco[nbNoix];
10         for(int i=0;i<tab.length;i++) {
11             tab[i]=new NoixDeCoco();
12         }
13     } }
```

Q2.1 Donner le code – au choix – du constructeur par copie ou de la méthode `clone()` de la classe `NoixDeCoco`.

Q2.2 Donner le code – au choix – du constructeur par copie ou de la méthode `clone()` de la classe `Cocotier`.

Q2.3 Soit l'instruction `Cocotier c1=new Cocotier(10,50);` donner l'instruction pour créer une copie de l'objet référencé par `c1`.

Exercice 3 (4 points) Gérer un échiquier

Soient les classes suivantes d'un jeu d'échecs :

```
1  public abstract class Piece {
2    public abstract void afficher();
3  }
4  public class Tour extends Piece {
5      public void afficher() {
6          System.out.println("Je suis
               une tour"); }
7  }
```

```
8  public class Cavalier extends Piece {
9      public void afficher() {
10         System.out.println("Je suis un
               cavalier"); }
11 }
```

On suppose que l'on se trouve dans la méthode `main` d'une classe `TestEchiquier`. On rappelle qu'un échiquier est constitué de 64 cases (correspondant à 8 lignes de 8 colonnes de cases) et qu'une case peut être soit vide soit occupée par une pièce (et une seule).

- déclarer la variable `echiquier` comme un tableau de `Piece` à 2 dimensions de 8 cases sur 8 cases.
- ajouter un cavalier sur la première ligne, deuxième colonne de l'échiquier.
- ajouter une tour dans la première colonne, le numéro de la ligne étant choisi aléatoirement.
- écrire les instructions pour afficher toutes les pièces présentes sur l'échiquier (on suppose que d'autres pièces peuvent se trouver sur l'échiquier).

Partie 2 Problème : livraisons de colis (29 points)

Remarque : *Si la méthode* ***`toString`*** *n'est pas demandée dans la question, il n'est pas nécessaire de la fournir, par contre, n'oubliez pas de définir les méthodes* ***`abstract`*** *quand cela est nécessaire.*

On veut modéliser un système de livraisons qui permet de livrer toutes sortes de choses livrables (colis, meubles livrables, repas livrables...). On suppose que l'on a des camions qui récupèrent des livrables dans des usines ou dans des dépôts, et qui les livrent dans des maisons de particulier ou dans d'autres dépôts. Les seuls bâtiments de la modélisation sont les usines, les maisons et les dépôts. Un dépôt permet de stocker des livrables. Un camion stocke les livrables le temps du transport. Un colis payant est un colis qui a en plus un prix. On suppose qu'un meuble ne peut pas être livré sauf si il est livrable.

On veut pouvoir connaître le numéro, la dimension et le poids de tous les livrables. On considère pour cela l'interface `Livrable` ci-contre, ainsi que la classe `Dimension` ci-dessous.

```
public interface Livrable {
   public int getNumero();
   public Dimension getDimension();
   public double getPoids();
}
```

```
1  public final class Dimension {
2     public final int longueur;
3     public final int largeur;
4
5     public Dimension(int lng, int lrg) {
6        longueur=lng;
7        largeur=lrg;
8     }
9     public Dimension() {
10       this((int)(Math.random()*100)+1,
11            (int)(Math.random()*100)+1);
12    }
13    public String toString() {
14       return longueur+"x"+largeur;
15    }
16 }
```

Q4.1 (2 points) (a) Dessiner le diagramme de classes UML (sans les attributs ni les méthodes) entre les classes `Batiment`, `Camion`, `Colis`, `ColisPayant`, `Depot`, `Maison`, `Meuble`, `MeubleLivrable`, `Usine` et l'interface `Livrable`. **(b)** Indiquer sur le schéma la (ou les) classe(s) qui pourraient être abstraites.

Q4.2 (3 points) Écrire la classe `Colis` qui implémente l'interface `Livrable`. Le numéro du colis sera généré automatiquement à l'aide d'un compteur et commencera à 10001 (le premier colis aura le numéro 10001, le deuxième 10002...). Cette classe contient un constructeur prenant en paramètre la dimension et le poids du colis, ainsi qu'un deuxième constructeur prenant en paramètre seulement le poids. Écrire aussi une méthode `toString()`.

Q4.3 (3 points) Un colis payant est un colis qui a un prix. Écrire la classe `ColisPayant` avec :
- une constante `PRIX_STANDARD` initialisée à 10 euros correspondant au prix standard d'un colis payant (le prix standard peut être connu par tous),
- un attribut `prix` (`double`),
- un constructeur à 3 paramètres,
- un constructeur à 1 paramètre qui initialise le prix au prix standard,
- une méthode `toString()`.

Interfaces `Receveur` et `Livreur` Les maisons, les dépôts et les camions peuvent recevoir des livrables. Les usines, les dépôts et les camions peuvent livrer des livrables.

Q4.4 (2 point) (a) Écrire l'interface `Receveur` qui contient une méthode `recevoir` qui prend en paramètre un livrable et qui retourne vrai si le livrable a bien été reçu, faux sinon. **(b)** Écrire l'interface `Livreur` qui contient une méthode `livrer` sans paramètre qui retourne le premier livrable accessible ou `null` sinon.

Q4.5 (4 points) Un camion peut recevoir et livrer des livrables. Le chargement du camion est représenté par un tableau de livrables. Quand un camion reçoit un livrable, il est inséré dans la première case libre du tableau seulement si le livrable n'est pas `null`, si le poids maximal du chargement n'est pas dépassé et s'il reste de la place. Écrire la classe `Camion` avec notamment les attributs et méthodes suivants :
- `tabLiv` : un tableau de livrables,
- `poidsMax` : le poids maximal que le camion peut transporter,
- constructeur prenant en paramètre `nbMax` le nombre maximal de livrables qu'il peut transporter et `poidsMax` le poids maximal qu'il peut transporter,
- méthode `double getPoidsChargement()` qui retourne la somme des poids de tous les livrables dans le camion.

Attention : certaines cases du tableau peuvent être vides.

Q4.6 (3 points) Un bâtiment a une adresse (`String`) et un type (chaîne de caractères, par exemple, "habitation","industriel") qui dépend des classes filles. On ne veut pas définir de variables d'instance `type` dans la classe `Batiment` ni dans ses classes filles, mais on veut pouvoir connaître le type de toutes les classes filles de `Batiment`. Écrire une classe `Batiment` avec obligatoirement un seul attribut appelé `adresse` (l'adresse peut être connue par les classes filles, mais ne doit pas pouvoir être modifiée) et un constructeur prenant un seul paramètre l'adresse du bâtiment. Ajouter une méthode pour que l'on puisse connaître le type de tous les bâtiments. Écrire aussi la méthode `toString()` qui doit retourner une chaîne avec l'adresse et le type du bâtiment.

Q4.7 (2 points) Écrire une classe `Maison` sans attribut. Quand une maison reçoit un livrable, elle affiche un message avec son adresse (uniquement l'adresse, pas le type) et le numéro du livrable (uniquement le numéro).

Dépôts avec casiers Un dépôt contient des casiers. Chaque casier peut contenir un seul livrable.

Q4.8 (1 point) Un casier (défini question suivante) peut être vide, plein ou trop petit, ce qui dans certains cas peut poser un problème. Écrire une classe `CasierException` contenant un constructeur prenant en paramètre un casier et une information sur le type de problème (par exemple, "vide", "plein", "trop petit"). Le message de l'exception doit être, par exemple, `"«C» est vide"`, `"«C» est plein"`, `"«C» est trop petit"` où `«C»` doit être remplacé par le `tSotring()` du casier.

Q4.9 (3 points) Un casier a un numéro (`int`), une dimension (`Dimension`) et peut contenir un livrable (`Livrable`). Écrire une classe `Casier` avec un constructeur prenant en paramètre le numéro du casier et les méthodes (la méthode `toString()` n'est pas demandée) :
- méthode `stocker` dont le but est de stocker un livrable dans le casier. Cette méthode prend en paramètre un livrable, ne retourne rien, mais lève l'exception `CasierException` quand le casier est plein (contient déjà un livrable) ou trop petit (si la longueur du casier est plus petite que la longueur du livrable ou si la largeur du casier est plus petite que la largeur du livrable),

— méthode `obtenir` dont le but est d'obtenir le livrable dans le casier. Cette méthode sans paramètre enlève le livrable du casier et le retourne. Elle lève l'exception `CasierException` quand le casier est vide.

Q4.10 (4 points) Un dépôt est un bâtiment industriel qui peut recevoir et livrer des livrables, ces livrables sont stockés dans des casiers. Écrire une classe `Depot` avec :

— un (seul) attribut de type `ArrayList` de casiers,
— un constructeur prenant en paramètre une adresse et le nombre de casiers que doit contenir le dépôt au départ,
— une méthode `recevoir` qui stocke si possible le livrable dans le premier casier qui accepte le livrable. Pour cet examen, il ne vous ait pas demandé d'écrire la méthode `livrer`.
— On veut aussi écrire une méthode qui permet de charger plusieurs livrables dans un `Receveur` (par exemple, pour charger un camion). Écrire une méthode `charger(Receveur r)` qui parcourt tous les casiers et donne au receveur les livrables des casiers tant que le receveur accepte des livrables : si le receveur refuse un livrable, alors le livrable est remis dans le casier et la méthode s'arrête.

Q4.11 (2 points) Ajouter dans la classe `Camion` une méthode `livraison(ArrayList<Batiment> alBat)` qui contient en paramètre une liste de bâtiments que le camion doit tous visiter pour faire sa livraison. Le camion reçoit un livrable des bâtiments livreurs. Le camion livre un livrable aux bâtiments receveurs.