SCIENCES SORBONNE UNIVERSITÉ

LU2IN002-2021oct
Éléments de programmation
par objets avec Java

Partiel du 16 novembre 2021 – Durée : 1 heure 30 minutes

Documents non autorisés. Appareils électroniques éteints et rangés.
Le barème sur 40 points est donné à titre indicatif.

Remarque préliminaire : *la visibilité des variables et des méthodes à définir, ainsi que leurs aspects statique, ou final ne sera pas précisé. C'est à vous de décider de la meilleure solution à apporter. De même, penser à compléter les classes par l'écriture de la méthode* `toString` *qui leur correspond.*

Partie 1 Questions (15 points)

Question 1.1 (2 points) Soient les deux classes suivantes qui compilent sans erreur. Pour chacune des lignes 12, 13, 14 et 15, écrire le numéro de la ligne, puis indiquer l'affichage obtenu pour la ligne.

```
1  public class Quest {
2      public Quest() {}
3      public void methQ(int x) { System.out.println("int"); }
4      public void methQ(double y) { System.out.println("double"); }
5      public void methQ(int a, double b) { System.out.println("int,double"); }
6      public void methQ(double c, double d) { System.out.println("double,double"); }
7  }
8  public class Test {
9      public static void main(String [] args) {
10         double di=5;
11         Quest q=new Quest();
12         q.methQ(di);
13         q.methQ((int)3.7);
14         q.methQ(8,9);
15         q.methQ((double)1,2);
16     }
17 }
```

2 points : -0.5 point réponse erronée/manquante. Rappel : aucune explication n'est demandée

l. 12 `double` : c'est le type de la variable *di* qui est pris en compte
l. 13 `int` : l'expression `(int)3.7` est de type `int`
l. 14 `int,double` : il n'existe pas de méthode de signature `methQ(int, int)`,
comme Java peut convertir un `int` en `double` implicitement
deux méthodes sont possibles `methQ(int, double)` et `methQ(double, double)`
la méthode choisie est celle de signature la plus proche `methQ(int, double)`
l. 15 `double,double` : il n'existe pas de méthode de signature `methQ(double, int)`,
un `double` ne peut pas être converti implicitement en `int`
il n'y a donc qu'une seule méthode possible `methQ(double, double)`

Question 1.2 (3 points) Tableau à deux dimensions avec nombre variable de colonnes.

Q. 1.2a Donner la déclaration et l'initialisation de la variable *tab* qui fait référence à un tableau d'entiers à 2 dimensions de 5 lignes. Le nombre de colonne de chaque ligne est choisi aléatoirement entre 1 et 10 compris.

2 point : -0.5 par faute (on met quand même 0.5/2 si seule la première ligne est correcte)
On ne compte pas comme "faute" les petites fautes qui ne concerne pas l'allocation du tableau : erreur sur les bornes de l'aléatoire, manque le cast, parcours de boucle un peu faux...

```
1          int [][] tab=new int [5][];
2          for(int i=0;i<tab.length;i++) {
3              int a=(int)(Math.random()*10+1); // [1,10]
4              tab[i]=new int [a];
5          }
```

Q. 1.2b Écrire une boucle sans indice qui, pour chaque ligne du tableau *tab*, affiche le nombre de colonnes qui lui correspond.

1 point : -0.5 par faute (0/1 si pas d'utilisation de la boucle sans indice)

```
1          for(int [] sousTab : tab) {
2              System.out.println(sousTab.length);
3          }
```

Question 1.3 On considère un programme pour modéliser des bulles de savon. Soit la classe suivante qui compile sans erreur.

```
1  public class Bulle {
2      public static final double DIAMETRE_MAX=30;
3      private static int nbBulles=0;
4      public final int numero;
5      private double diametre ;
6
7      public Bulle(double diametre) {
8          nbBulles++;
9          numero=nbBulles;
10         this.diametre=diametre;
11     }
12     public Bulle() {
13         this(Math.random()*DIAMETRE_MAX);
14     }
15     public int getNbBulles() { return nbBulles; }
16 }
```

Question 1.3.1 (2 points) Pour chaque instruction des lignes 23 à 26 (4 instructions) ci-dessous, écrire le numéro de la ligne, puis indiquer si l'instruction compile (OK) ou pas (FAUX). Si l'instruction ne compile pas, justifier brièvement pourquoi (justification courte suffisante, exemples : privé, pas statique...).

```
20 public class Test1 {
21     public static void main(String [] args) {
22         Bulle b=new Bulle();
23         System.out.println(b.diametre);
24         System.out.println(b.DIAMETRE_MAX);
25         System.out.println(Bulle.nbBulles);
26         System.out.println(Bulle.getNbBulles());
27     }
28 }
```

2 point : 0.5 par ligne (pour les instructions fausses, pas de justification => 0/0.5)

l. 23 : `b.diametre` Faux, car `diametre` est private
l. 24 : `b.DIAMETRE_MAX` OK, mais c'est mieux d'utiliser `Bulle.DIAMETRE_MAX`
l. 25 : `Bulle.nbBulles` Faux, car `nbBulles` est private
l. 26 : `Bulle.getNbBulles()` Faux `getNbBulles()` pas static, l'accesseur d'une variable static devrait être static, la signature de cette méthode devrait être : `public static int getNbBulles()`

Question 1.3.2 (3 points) Soit la classe suivante qui compile sans erreur.

```
50 public class Test2 {
```

```
51      public static void main(String [] args) {
52          Bulle [] tab=new Bulle[5];
53          Bulle b1=new Bulle();
54          Bulle b2=b1;
55          tab[0]=b1;
56          tab[1]=b2;
57          tab[2]=tab[0];
58          tab[3]=new Bulle();
59          // Diagramme mémoire
60          tab[3]=null;
61          // Le programme continue...
62      }
63  }
```

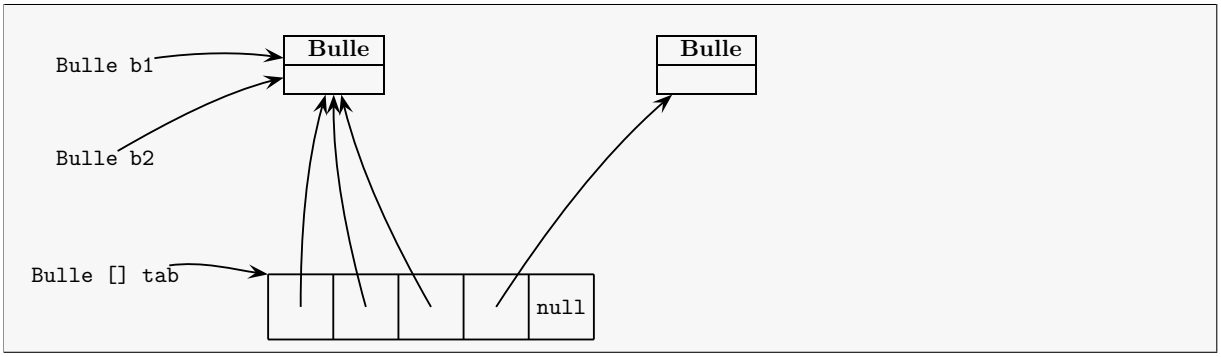
Q. 1.3.2a Dans la méthode `main`, au total, combien d'instances de la classe `Bulle` ont été créées ?

1 point pour le chiffre 2

2 instances (aux lignes 53 et 58) de `Bulle` ont été créées.
Remarque : il y a 3 variables créées : `tab`, `b1`, `b2` (+ `args`)

Q. 1.3.2b Dessiner le diagramme mémoire à la ligne 59 (sans indiquer les attributs ni les méthodes).

1 point : -0.5 par faute (flèche manquante/erronée, objet manquant/ajouté...)
On ne compte pas comme "faute", si ils n'ont pas écrit null dans la cinquième case



Q. 1.3.2c Quel est l'effet de l'instruction en ligne 60 sur l'état de la mémoire ?

1 points pour idée de "libération de la mémoire d'un objet"

Le *garbage collector* libère la mémoire pour l'objet créé à la ligne 58, car il n'est plus référencé dans le programme.

Question 1.4 (2 points) Lorsqu'une bulle rencontre une autre bulle cela crée une troisième bulle qui a pour diamètre la somme des diamètres des deux bulles. Donner le code de la méthode `rencontrer` de la classe `Bulle` qui implémente ce comportement.

2 points : -0.5 par faute (paramètre faux, type retour faux, return manquant, new absent...)

```
    public Bulle rencontrer(Bulle b2) {
        return new Bulle(diametre+b2.diametre);
    }
```

Question 1.5 (3 points) On souhaite écrire une classe `UsineABulles` pour créer des bulles de diamètre compris dans [1,10] de la façon suivante : la première bulle créée aura un diamètre de 1 cm, la deuxième de 2 cm, la troisième de 3 cm... la dixième de 10 cm, la onzième de 1 cm, la douzième de 2 cm... la vingtième de 10 cm, la vingt-et-unième de 1 cm... et ainsi de suite. Cette classe `UsineABulles` contient seulement trois membres : une variable `diametre` initialisée à 0, un constructeur, et une méthode `Bulle creerBulle()` qui permet de créer une bulle en suivant le protocole indiqué. On ne doit pas pouvoir créer

d'instance d'`UsineABulles`. Écrire la classe `UsineABulles` et donner un exemple d'utilisation pour créer des bulles.

3 points : -0.5 par faute (par static manquant, private faux, création de Bulle faux ...)
On n'enlève pas de points si le diamètre des bulles n'est pas tout à fait 1, 2, ..., 10, 1, 2,... (par exemple, si `diametre++` n'est pas au bon endroit).

```
1  public class UsineABulles {
2      private static double diametre=0;
3      private UsineABulles() {} // private : Empeche la creation d'instance
4      public static Bulle creerBulles() {
5          diametre++;
6          if (diametre>10) {
7              diametre=1;
8          }
9          return new Bulle(diametre);
10     }
11 }
```

Remarque : au lieu du `if`, on peut ajouter `diametre=diametre%10;` avant `diametre++;`

Partie 2 Problème (27 points)

On souhaite écrire un programme pour gérer des véhicules. Un véhicule possède un nombre variable de roues (typiquement, 2 roues (vélo), 3 roues (tricycle), 4 roues (voiture), etc.).

Une roue est définie par son diamètre, en centimètres, dont la valeur est dans l'intervalle $[50, 100]$.

Remarque générale : pour les boucles sur les tableaux, sanctionner la non-utilisation de length.

Question 2.1 (3.5 points) Écrire la classe `Roue` qui possède un unique constructeur avec un argument donnant la valeur du diamètre de la roue. Afin de vérifier que le diamètre se trouve bien dans l'intervalle requis, la classe doit contenir deux constantes `DIAMETREMIN` et `DIAMETREMAX` (valant respectivement 50 et 100). Lors de la création d'une roue, si le diamètre donné est inférieur au diamètre minimum alors la roue est créée avec le diamètre minimum, de même si le diamètre donné est supérieur au diamètre maximum alors elle est créée avec le diamètre maximum.

Bonne définition des constantes (static et final) : 2pt (moitié des points s'il manque l'un ou l'autre).

Bonne définition générale (constructeur, et attribut) : 1pt

Méthode toString : 0.5pt

```
1  public class Roue {
2          public static final int DIAMETREMIN = 30;
3          public static final int DIAMETREMAX = 100;
4
5          private int diametre; // diamètre en centimètre
6
7          public Roue(int d) {
8                  if (d<DIAMETREMIN)
9                          diametre = DIAMETREMIN;
10                 else if (d>DIAMETREMAX)
11                         diametre = DIAMETREMAX;
12                 else
13                         diametre = d;
14         }
15         public String toString() {
16                 return "Roue de "+diametre+" cm";
```

```
17          }
18  }
```

Question 2.2 (4.5 points) Pour modéliser un véhicule avec un nombre de roues variable, on utilise pour attribut un tableau de roues qui doit être initialisé lors de la création du véhicule. Un deuxième attribut d'un véhicule est un numéro (entier) qui l'identifie de façon unique. Ce numéro est géré en utilisant un compteur de création de véhicule.

Q. 2.2.1 Écrire la classe `Vehicule` qui possède un unique constructeur qui prend en argument un tableau de roues pour initialiser les roues du véhicules.

Q. 2.2.2 Ajouter un accesseur pour connaître le nombre de véhicules créés depuis le lancement du programme, ainsi qu'une méthode qui rend le nombre de roues du véhicules.

1pts pour le compteur et sa bonne gestion
0.5pt pour la méthode statique qui rend le nombre de véhicules créés
1pt pour la méthode qui rend le nombre de roues : SEULEMENT si elle utilise length. Toute autre solution (par exemple avec un attribut supplémentaire) vaut 0.
2pt pour le reste de la classe (avec toString)

Remarque : on sanctionne si l'accesseur ne respecte pas le standard : getNbVehicules (avec le nom de la variable).

```
1  public class Vehicule {
2          private static int nbVehicules = 0;
3
4          private Roue[] tabRoues;
5          private int numero;
6
7          public Vehicule(Roue[] lesRoues) {
8                  numero = nbVehicules++;
9                  tabRoues = lesRoues;
10         }
11         public int nombreDeRoues() {
12                 return tabRoues.length;
13         }
14         public static int getNbVehicules() {
15                 return nbVehicules;
16         }
17
18         public String toString() {
19                 String res = "Vehicule "+ numero + " [avec "+tabRoues.length+ "
                       roues =";
20                 for (Roue r : tabRoues)
21                         res += "("+r+")";
22                 return res + "]";
23         }
24 }
```

Question 2.3 (2 points) Écrire une classe `Test` avec une méthode `main` réalisant les traitements suivants :

1. création d'un tableau de 2 roues de diamètre 60 cm ;
2. création d'un vélo (nom de variable : `velo1`) avec les 2 roues créées précédemment ;
3. création d'un autre vélo (nom de variable : `velo2`) avec 2 roues de 60 cm ;
4. affichage des 2 vélos créés et du nombre de véhicules créés depuis le lancement du programme ;

2pt pour l'ensemble. Ne mettre que la moitié des points si le 2e vélo est créé avec les mêmes roues que le 1er.

```
public class TestClasses {
        public static void main(String[] args) {
                Roue[] tabRoues = new Roue[2];
                for (int i=0; i < tabRoues.length; i++) {
                        tabRoues[i] = new Roue(60);
                }
                Vehicule velo1 = new Vehicule(tabRoues);
                Vehicule velo2bad = new Vehicule(tabRoues); // A SANCTIONNER !
                Vehicule velo2 = new Vehicule( new Roue[]{new Roue(60), new Roue
                    (60)} ) ;
                System.out.println("Premier affichage: \n"+velo1);
                System.out.println(velo2);
                System.out.println("Il y a eu "+Vehicule.getNbVehicules()+" vé
                    hicules de créés.");

        }
}
```

Question 2.4 (4 points) Dans la classe `Vehicule`, écrire la méthode `clone` pour cloner un véhicule. Ne pas oublier de donner les modifications à apporter à d'autres classes du programme si nécessaire afin que la solution soit conceptuellement correcte.

Il faut écrire 2 fonctions clone() : une dans Vehicule, et l'autre dans Roue.

3pt : pour la méthode clone de Vehicule qui fait correctement appel à la méthode clone de Roue

1pt pour la méthode clone dans Roue.

Dans la classe `Vehicule` :

```
        public Vehicule clone()  {
                Roue[] tab = new Roue[tabRoues.length];
                for (int i=0; i<tabRoues.length; i++) {
                        tab[i] = tabRoues[i].clone();
                }
                return new Vehicule(tab);
        }
```

Dans la classe `Roue` :

```
        public Roue clone() {
                return new Roue(diametre);
        }
```

Question 2.5 (4 points) Deux roues sont dites compatibles si elles possèdent le même diamètre. Dans la classe `Roue`, écrire la méthode `compatible` qui prend en argument un `Objet obj` et qui rend le booléen vrai si cet objet correspond à une roue compatible avec la roue courante.

C'est une version de méthode equals() qui est demandée ici.

2pt pour les 3 premiers tests (null, même objet et classes différentes).

2pt pour la comparaison des diamètres avec soit un cast, soit une variable locale.

moitié des points si la méthode prend une Roue et pas un Object.

```
        public boolean compatible(Object obj) {
                if (obj == null)
                        return false;
                if (this == obj)
                        return true;
                if (obj.getClass() != this.getClass()) {
                        return false;
```

```
8                   }
9                   return this.diametre == ((Roue)obj).diametre;
10          }
```

Question 2.6 (7 points) Un convoi est un ensemble de véhicules que l'on représente par la classe `Convoi` dont un des attributs est un tableau de véhicules. Dans notre programme, on veut garantir qu'il ne puisse exister qu'une seule instance de cette classe `Convoi` lors de l'exécution du programme.

En utilisant le mécanisme du singleton, écrire la classe `Convoi`. Dans cette classe, définir une constante `TAILLEMAX` (de valeur 4 ici) qui donne le nombre maximal de véhicules que peut contenir le convoi. Une méthode `ajoute` qui prend en argument un véhicule, permet d'ajouter un véhicule dans le convoi. Cette méthode rend le nombre de véhicules que contient le convoi à l'issue de cet ajout. Si le convoi est plein (la taille maximale de véhicules est atteinte), le nouveau véhicule n'est pas ajouté et la méthode rend la valeur -1. La classe doit fournir une méthode pour récupérer la référence du convoi créé et contenir un attribut qui donne le nombre de véhicules que contient le convoi.

4 pts pour la gestion complète et correcte du mécanisme du singleton (constructeur privé, attribut privé et final avec l'instance créée, méthode de récupération de l'instance créée).

2 pt pour l'ajout du véhicule correctement géré

0.5pt pour la bonne définition de la constante TAILLEMAX

0.5pt pour toString

```
1  public class Convoi {
2          public static final int TAILLEMAX = 4;
3
4          private static final Convoi INSTANCE = new Convoi();
5          private Vehicule[] tab;
6          private int nbVehicules;
7
8          private Convoi() {
9                  tab = new Vehicule[TAILLEMAX];
10                 nbVehicules =0;
11         }
12
13         public static Convoi getInstance() {return INSTANCE; }
14
15         public int ajoute(Vehicule v) {
16                 if (nbVehicules<TAILLEMAX) {
17                         tab[nbVehicules++] = v;
18                         return nbVehicules;
19                 }
20                 else {
21                         return -1;
22                 }
23         }
24
25         @Override
26         public String toString() {
27                 return "Convoi [tab=" + Arrays.toString(tab) + ", nbVehicules="
                       + nbVehicules + "]";
28         }
29
30
31 }
```

Question 2.7 (2.5 points) Donner les instructions à ajouter dans la méthode `main()` de `Test` pour réaliser les traitements suivants :

1. création d'une variable de nom `c` qui contient la référence d'un convoi ;

2. ajouter les 2 vélos créés précédemment dans le convoi ;
3. afficher le convoi ;
4. tester la compatibilité d'une roue avec un convoi ;
5. tester la compatibilité d'une roue avec une autre roue de même diamètre.

0.5pt par ligne

```
Convoi c = Convoi.getInstance();
c.ajoute(velo1);
c.ajoute(velo2);
c.ajoute(velo3);
System.out.println("Cinquieme affichage: \n"+c);

System.out.println("Compatibilité :");
System.out.println(" : "+tabRoues[0].compatible(tabRoues[1]));
System.out.println(" : "+tabRoues[0].compatible(tabRoues[0]));
System.out.println(" : "+tabRoues[0].compatible(c));
```