

INTRODUCTION A LA PROGRAMMATION

Examen semestriel

Instructions :

- Vous disposez de deux heures pour faire cet examen (10h00 - 12h00).
- Nombre maximum de 90 points (+ 15 bonus).
- Attention : il y a aussi des énoncés sur le verso.
- Vous devez **écrire à l'encre noire ou bleu foncé**, pas de crayon ni d'autre couleur. N'utilisez **pas non plus de stylo effaçable** (perte de l'information à la chaleur).
- Toute documentation papier est autorisée ; En revanche, vous ne pouvez pas utiliser d'ordinateur personnel, ni de téléphone portable, ni aucun autre matériel électronique.
- Répondez sur les feuilles qui vous sont distribuées et **aux endroits prévus**. **Ne répondez pas sur l'énoncé**.
- Ne joignez aucune feuilles supplémentaires ; **seules les feuilles de réponses distribuées seront corrigées**.
- Vous pouvez répondre aux questions en français ou en anglais.
- L'examen compte 3 exercices indépendants. **Vous pouvez commencer par celui que vous souhaitez**
 - Exercice 1 : **55** points.
 - Exercice 2 : **35** points.
 - Exercice 3 : **15** points (bonus).

suite au verso ↗

Exercice 1 : Conception OO [55 points]

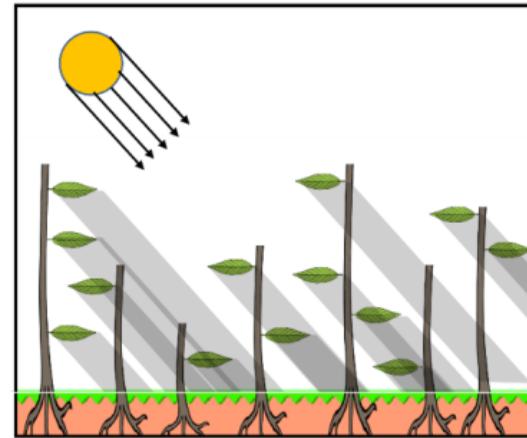
Il est important de lire chaque question de cet exercice dans son intégralité avant d'y répondre. Prenez aussi le temps de lire les contraintes imposées et de consulter le code fourni en annexe avant de commencer l'exercice.

Cet exercice contient 4 questions. Les questions 1 et 2 sont indépendantes.

On s'intéresse ici à simuler, de façon très simplifiée, des plantes qui poussent, se reproduisent, et meurent, dans un environnement impacté par des sources de lumière.

Exemple : plantes sous la lumière du soleil

- elles acquièrent de l'énergie en absorbant la lumière avec leurs feuilles ;
- et en dépensent pour la croissance et la reproduction.



Contraintes de conception Lors de la résolution de cet exercice, les contraintes suivantes seront à respecter :

1. Lorsque l'on vous demande d'écrire le corps d'une classe vous le ferez en syntaxe Java et y écrirez les attributs et les méthodes mais sans les corps (sauf pour ceux explicitement demandés).
2. Vous supposerez qu'un programme principal existe, qu'il crée un objet **Environment** qui y crée des plantes et des sources de lumière et appelle en boucle leur méthode **update**. Cette méthode permet de faire évoluer les entités impliquées au cours du temps (au moyen de méthodes **update**, telles que dictées par l'interface **Updatable**).
3. Les classes devront contenir les membres nécessaires pour mettre en oeuvre toutes les fonctionnalités souhaitées, **qu'elles soient explicitement demandées ou suggérées par les spécifications de l'énoncé** (en particulier par la mise en oeuvre des méthodes **update**) ;
4. Pour les méthodes qui ne sont pas triviales, **vous noterez au moyen d'un bref commentaire la signification du type de retour et ce qu'elles font dans les grandes lignes ainsi que les fonctionnalités des autres classes qu'elles utilisent**.
5. **Ne négligez ni les constructeurs, ni les droits d'accès.**
6. Le droit **protected** ne doit pas être utilisé pour les attributs.
7. Votre conception doit être bien modularisée.
8. Vous ne proposerez de getter/setter que lorsqu'ils sont nécessaires à la mise en oeuvre de la simulation décrite.
9. Il n'est pas nécessaire d'écrire des directives d'importation.

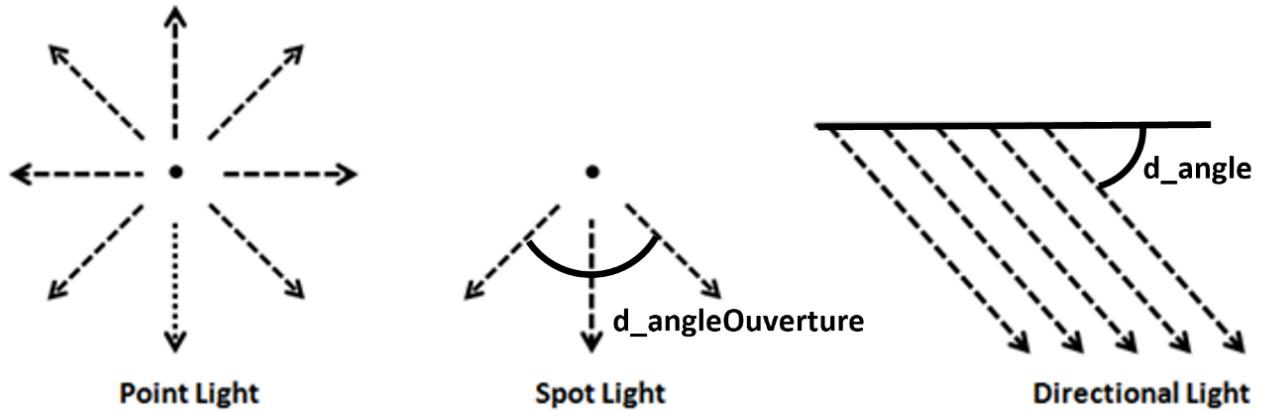


FIGURE 1 – Sources de lumière

Question 1 : Sources de lumière [23 points]

Les sources de lumière peuvent être soit *en forme de point*, de *spot* ou *directionnelles* (voir la figure 1).

- Une source de lumière en forme de point éclaire partout. Elle est caractérisée par son intensité (*double*) ainsi que par sa position (*Vec2d*).
- Une source de lumière en forme de spot éclaire dans une direction donnée avec un *angle d'ouverture* donné. Elle est donc caractérisée par son intensité (*double*), sa position (*Vec2d*), sa direction de diffusion et son angle d'ouverture.
- Une source de lumière directionnelle éclaire l'espace selon un *angle d'inclinaison* donné, et est donc caractérisée par son intensité (*double*) ainsi que son angle d'inclinaison (*double*).

Les caractéristiques de chaque type de source de lumière doivent être initialisées lors de leur construction au moyen de données fournies en paramètres.

Évolution au cours du temps : Les sources de lumière en forme de point et en forme de spot voient leur intensité évoluer au cours du temps selon des algorithmes qui leur sont spécifiques. Les sources de lumière directionnelles quant à elles, voient uniquement leur angle varier, par rotation, en fonction du temps. Les modalités précises de mise en oeuvre des rotations et changement d'intensité ne nous intéressent pas.

Les sources de lumière émettent de l'énergie. La quantité d'énergie (*double*) émise sur un pas de temps *dt*, en un point donné (*Vec2d*) est calculable selon des modalités spécifiques à chaque source.

Écrivez des classes permettant de représenter le modèle. Vous supposerez que les données nécessaires à l'initialisation des objets sont passés en paramètre des constructeurs.

suite au verso ↗

Question 3 : Feuilles et plantes (17 points)

Une *feuille* est dotée d'une position (`Vec2d`). Une *plante* a un ensemble de feuilles. Elle est caractérisée par une position (`Vec2d`), un niveau d'énergie, et un âge. Ces données sont toutes de type `double`. L'âge maximal possible est une donnée caractéristique *commune à toutes les plantes*.

Une plante peut être créée sans feuille et avec un âge nul ou avec un nombre donné de feuilles et un âge donné. Le reste des attributs doit être initialisé au moyen de valeurs passées en paramètres.

Évolution au cours du temps : à chaque pas de temps `dt`, les plantes acquièrent de l'énergie en absorbant celle émise par toutes les sources de lumière de l'environnement.

Pour une source de lumière donnée, une plante de position `p` absorbe, par feuille et en fonction de la position de celle-ci, un pourcentage de l'énergie émise par la source en `p`. Ce pourcentage est une constante commune à toutes les plantes.

Si le niveau d'énergie d'une plante dépasse un certain seuil (une constante commune pour toutes les feuilles), une feuille est ajoutée à une position aléatoire sur la plante, et le niveau d'énergie de la plante est divisé par 2. Les modalités du positionnement aléatoire ne nous intéressent pas.

Par ailleurs, à chaque pas de temps `dt`, la plante vieillit (voit son âge augmenter de `dt`) et perd une quantité d'énergie dépendant du temps. Les plantes ayant une énergie nulle meurent et doivent disparaître de la simulation. Lorsqu'une plante dépasse l'âge maximal elle meurt aussi. Avant de disparaître de l'environnement une plante laisse autour de sa position quelques nouvelles pousses (les modalités exactes ne nous intéressent pas).

- Écrivez les classes permettant de représenter le modèle ci-dessus. Vous supposerez que les données nécessaires à l'initialisation des objets sont passés en paramètre des constructeurs.
- Donnez tous les prototypes de méthodes à ajouter à la classe `Environment` pour réaliser les traitements requis en commentant leur rôle.

Question 2 : Programmation (6 points)

Écrivez le corps de la méthode `update` des plantes. Le corps de toute autre méthode appelée par `update` n'est pas nécessaire. Cependant, vous devez avoir commenté leur entête pour expliquer ce qu'elles font et quelles méthodes importantes elles invoquent. Vous pouvez donner les noms de votre choix à toutes les constantes qui ne sont pas explicitement mentionnées dans l'énoncé du problème.

Question 4 : Analyse [9 points]

1. Pourquoi la classe `Environment` n'implémente t-elle pas l'interface `update` ?
2. Quelle critique susciterait de votre part l'ajout d'un constructeur à la classe `Environment` :


```
public Environment(double width, double height, List<Plant>)
```
3. On suppose que toutes les classes sont dans le même paquetage. Comment peut-on, en ajoutant une interface, empêcher la méthode `addLight` de `Environment` de faire évoluer accidentellement les plantes (en invoquant leur méthode `update`) ? Donnez une explication succincte en français en indiquant ce qu'il faudrait modifier dans le code existant.

Exercice 2 : Concepts de base [35 points]

Répondez clairement et succinctement aux questions suivantes :

1. [5 points] Soit le code suivant :

```

1  class X {
2      private double x = 2;
3
4      public X() {
5          System.out.println("X:" + getX());
6          x = 10.0;
7      }
8      public double getX(){ return x; }
9  }
10
11 class Y extends X {
12     private double x;
13     public Y() {
14         System.out.println("Y:" + getX());
15         x = 5.0;
16     }
17     public double getX(){ return x; }
18 }
19
20 class Test {
21     public static void main(String[] args) {
22         X y = new Y();
23         System.out.println(y.getX());
24     }
25 }
```

- (a) Qu'affiche l'exécution de la ligne 22 ? Justifiez brièvement.
- (b) La ligne 23 affiche 5.0. Re-écrivez le code de la ligne 17 pour faire en sorte que cela affiche la valeur 10 du x de la super-classe.
2. [10 points] Indiquez pour chacune des assertions suivantes si elles sont correctes ou incorrectes :
- (a) Une classe abstraite ne peut avoir que des méthodes abstraites.
 - (b) Une classe imbriquée peut accéder à un membre privé de sa classe englobante.
 - (c) Une classe ne peut pas étendre directement plusieurs classes.
 - (d) Une classe peut implémenter plusieurs interfaces.
 - (e) Une classe abstraite ne peut pas implémenter une interface.
 - (f) Une méthode dans une classe abstraite ne peut pas appeler de méthode abstraite.
 - (g) Les constructeurs dans une interface ne peuvent être que des constructeurs par défaut.
 - (h) Une classe abstraite ne peut pas avoir que des constructeurs par défaut.
 - (i) Une interface peut étendre une autre interface.
 - (j) Tous les membres d'une interface sont public.

suite au verso ↗

3. [11 points] Soit le code suivant :

```

1  class A {
2      private int a=1;
3      public void add(int delta) { a+=delta; }
4      public int getV() { return a; }
5  }
6
7  class B extends A {
8      private int b=2;
9      public void add(int delta) { b+=delta; }
10     public int getV() { return b; }
11 }
12
13 class Test {
14
15     public static void main(String[] args) {
16         A a = new A();
17         B b = new B();
18         print(a,b);
19         a.add(5);
20         b.add(5);
21         print(a,b);
22         m(a,b);
23         print(a,b);
24     }
25
26     public static void m (A a, B b) {
27         a = b;
28         b = new B();
29         a.add(2);
30         b.add(3);
31     }
32
33     public static void print (A a, B b) {
34         System.out.println(a.getV() + " " + b.getV());
35     }
36 }
```

- (a) Qu'affiche t-il ? justifiez brièvement.
- (b) Compile t-il toujours si on modifie l'entête de la méthode `m` comme ceci :
`public static void m (final A a, B b)`
Justifiez brièvement.
- (c) Compile t-il toujours si l'on déclare la classe `B` comme `final` ? Même question si l'on déclare la classe `A` comme `final` ? Justifiez brièvement.

4. [9 points] Soient les déclarations de classes suivantes :

```

1 package p1;
2 public class C1 {
3     protected m1(){}
4     private m2(){}
5         m3(){}
6 }
```

(1)

```

1 package p1;
2 import p1.C1;
3 public class C2 {
4     private C1 c2;
5 }
```

(2)

```

1 package p2;
2 import p1.C1;
3 public class C3 extends C1 {
4     private C1 c3;
5 }
```

(3)

```

1 package p1.p3;
2 import p1.C1;
3 public class C4 {
4     protected C1 c4;
5 }
```

(4)

Indiquez pour toutes les assertions suivantes si elles sont correctes ou incorrectes :

- (a) La clause d'importation n'est pas nécessaire dans le programme (2) pour que le programme compile.
- (b) La clause d'importation n'est pas nécessaire dans le programme (4) pour que le programme compile.
- (c) La méthode `m1()` peut être invoquée sur l'objet `c4` (via une méthode de `C4`) sans problème de compilation.
- (d) La méthode `m1()` peut être invoquée sur l'objet `c3` (via une méthode de `C3`) sans problème de compilation.
- (e) La méthode `m2()` peut être appelée par la méthode `m1()` sans problème de compilation.
- (f) La méthode `m3()` peut être invoquée sur l'objet `c2` (via une méthode de `C2`) sans erreur de compilation.

Justifiez brièvement vos réponses.

suite au verso ↗

Exercice 3 : Déroulement de programme [15 points (bonus)]

Le programme suivant compile et s'exécute sans erreurs (vous supposerez les directives d'importation présentes).

```

1 interface I {
2     public int CST = 2;
3     public default int c() { return 10; }
4 }
5 class S implements I {
6     private String s;
7     public S(String s){
8         this.s = s;
9     }
10    @Override
11    public int c() { return I.super.c()*s.length(); }
12    @Override
13    public String toString() { return s; }
14 }
15 enum Rank {
16     HIGH(100, "***" ),
17     MEDIUM(50,  "**" ),
18     LOW(20,   "*" ),;
19
20     private int val;
21     private String stars;
22
23     private Rank(int v, String s){
24         val = v;
25         stars = s;
26     }
27     public String shining(){ return stars; }
28     public static List<Integer> getAll() {
29         List<Integer> lst = new ArrayList<>();
30         for (Rank r : values()){
31             lst.add(r.stars.length());
32         }
33         return lst;
34     }
35 }
36 abstract class R0 implements I {
37     private Rank r;
38     private List<S> c = new ArrayList<S>();
39     public R0(List<Integer> lst, Rank r){
40         this.r = r;
41         for(Integer i : lst){
42             String s="";
43             for(int j = 0; j < i; ++j){ s += "*"; }
44             s += "/" + r.shining();
45             c.add(new S(s));
46         }
47     }
}

```

```

48     // Suite classe R0
49     public abstract int p();
50
51     @Override
52     public int c(){
53         int val=0;
54         for (S s : c) { val += p() * s.c(); }
55         return val;
56     }
57     @Override
58     public String toString(){
59         String res="";
60         for (S s : c) { res += s.toString() + " " ; }
61         return res;
62     }
63 }
64
65 class R1 extends R0 {
66     public R1(Rank r) {
67         super(Rank.getAll(), r);
68     }
69
70     public int p(){ return I.CST; }
71     public int c(int i) { return c()*i; }
72 }
73
74 class Deroulement {
75     public static void main(String[] args){
76         R0 r0    = new R1(Rank.HIGH);
77         I   rint = new R1(Rank.MEDIUM);
78         R1 r1    = new R1(Rank.LOW);
79
80         p(r0);
81         p(rint);
82         p(r1);
83     }
84     public static void p(I obj){
85         System.out.println(obj);
86         System.out.println(obj.c());
87     }
88 }
```

1. Pour chacune des lignes suivantes, indiquez si on peut l'ajouter à `main` :

- (a) `System.out.println(r1.c(2));`
- (b) `System.out.println(r0.c(2));`
- (c) `System.out.println((R1)r0.c(2));`

Justifiez brièvement.

2. Qu'affiche le programme ? Expliquez succinctement son déroulement. **Il ne s'agit pas ici de paraphraser le code, mais bien d'expliquer les étapes et le déroulement du programme.**