

INTRODUCTION A LA PROGRAMMATION


Test Semestre I

Instructions :

- Vous disposez de une heure quarante cinq minutes pour faire cet examen (13h15 - 15h) ;
- Nombre maximum de 105 points (dont environ 25 facultatifs) ;
- Toute documentation sur papier est autorisée ;
- Veillez à **ne traiter *qu'un* exercice par feuille** ;
- L'examen compte 4 exercices indépendants.

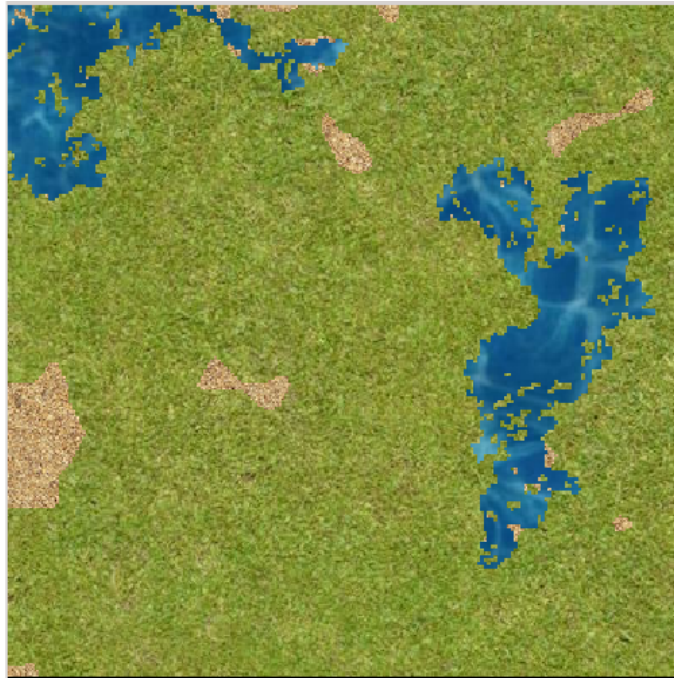
Vous pouvez commencer par celui que vous souhaitez

- Exercice 1 : **55** points.
- Exercice 2 : **20** points.
- Exercice 3 : **15** points.
- Exercice 4 : **15** points.

Suite au verso 
--

Exercice 1 : Conception OO [55 points]

Il s'agit dans cet exercice de simuler l'évolution, en fonction des saisons, des surfaces cultivables sur des territoires ressemblant à ceci :



Une partie du programme est fournie en annexe ; à savoir : une classe `Positionable` et une interface `Updatable`.

Prenez connaissance de ce matériel avant de commencer l'exercice. Notez que nous ne nous préoccupons pas dans cet exercice de la représentation graphique de la simulation.

Un territoire (classe `Territory`) est représenté au moyen d'une grille où chaque case contient une couche minérale (sol) qui peut être couverte par une couche d'herbe, par une couche d'eau ou par les deux. La couche minérale existe toujours.

Les cases bleues sont celles ayant une quantité d'eau dépassant un seuil donné `WATER_TRESHOLD`. Ce seuil est une constante identiques pour tous les territoires.

Les cases vertes sont celles contenant de l'herbe et où l'eau est présente en quantité inférieure à `WATER_TRESHOLD`.

La surface cultivable d'un territoire est donnée par le nombre de cases vertes.

Une description de la conception souhaitée pour les cases de la grille est donnée plus loin.

Un territoire est caractérisé par la *zone climatique* à laquelle il appartient, tropicale ou tempérée, ainsi que la *saison* pendant laquelle il est simulé.

Il doit être possible de simuler les quatre saisons : automne, hiver, printemps et été.

Les fonctionnalités souhaitées sont de pouvoir :

1. Initialiser un territoire en fournissant en paramètre son climat et un nom de fichier. Ce fichier est supposé contenir les données permettant d'initialiser la grille à savoir ses dimensions (vous supposerez que c'est un carré) et la nature de chaque case de la grille. Les modalités exactes d'initialisation de la grille depuis un tableau ne nous intéressent pas, mais peuvent lancer une `IOException` (de type «checked»).
2. Simuler l'évolution du territoire sur un pas de temps `dt` (méthode `simulate`).
3. Modifier la saison pour laquelle on souhaite simuler l'évolution territoire.

4. Faire chuter un volume donné v de pluie sur le territoire. Chaque case reçoit alors une quantité d'eau calculée en fonction de v . Les modalités précises du calcul ne nous intéressent pas.
5. Connaître la surface cultivable (à n'importe quel moment de la simulation)

Zones climatiques Une zone climatique est caractérisée par le fait qu'elle offre comme fonctionnalité le calcul de la *température moyenne en fonction d'une saison* : `double computeAverageTemp(Saison s)`.

On distingue deux catégories de zones climatiques les *tempérées* et les *tropicales*. Pour chacune de ces catégories le calcul de la température moyenne se fait différemment. Les modalités exactes des calculs ne nous intéressent pas.

Cases et cellules Chaque case de la grille sera modélisée comme la superposition de trois *cellules* (cellule de sol, d'eau et d'herbe). Une cellule a une position sur la grille. Les cellules qui n'existent pas dans une case vaudront `null` (lorsqu'une case ne contient qu'une cellule de sol par exemple, les autres cellules vaudront `null`).

Les *cellules d'eau* sont caractérisées par la quantité d'eau qui y est présente. Les *cellules d'herbe* par la quantité d'herbe.

La quantité d'eau présente sur une case de la grille est donc la quantité d'eau présente dans sa cellule d'eau. En cas de pluie, la quantité d'eau reçue s'ajoute à la cellule «eau» et s'il n'y en avait pas, il faut donc en créer une.

Les cellules d'eau évoluent au cours du temps en diffusant 20% de leur quantité d'eau sur les cases qui sont immédiatement ses voisines sur la grille. Si la température moyenne du territoire est supérieure à un certain seuil `EVAPORATION_TRESHOLD` (constant et commun à tous les territoires), elle s'évapore d'une certaine quantité dépendant du temps écoulé. Vous doterez ce type de cellules d'une méthode `evaporate` chargée de simuler l'évaporation de l'eau à proprement parler. S'il n'y a plus d'eau, la cellule d'eau disparaît (devient `null` dans la case correspondante).

Les cellules d'herbe voient leur quantité d'herbe diminuer ou augmenter en fonction du temps écoulé, de la température moyenne et de la quantité d'eau présente sur la case correspondant à la cellule. Vous doterez ce type de cellules d'une méthode `grow` chargée du calcul de l'augmentation ou de la diminution de la quantité d'herbe. S'il n'y a plus d'herbe (quantité nulle), la cellule d'herbe disparaît.

SUJET À LA PAGE SUIVANTE

Question 1 : Conception [50 points]

On vous demande d'écrire la/les classe(s) permettant de compléter les éléments manquants et permettant de mettre en oeuvre les fonctionnalités souhaitées ; seules les entêtes des méthodes sont demandés. **On ne vous demande pas d'écrire de programme complet, ni le corps des méthodes, uniquement les attributs et les entêtes de méthodes.** Vous pouvez décrire les classes et leur contenu au moyen de diagrammes ou en pseudo-code Java. Les contraintes suivantes seront respectées :

1. vous supposerez notamment qu'un programme principal existe, qu'il crée un objet `Territory`, et appelle en boucle sa méthode `simulate`. Cette méthode permet de faire évoluer les entités impliquées au cours du temps (au moyen de méthodes `update`, telle que dictées par l'interface `Updatable`) ;
2. l'initialisation de la grille au travers d'un fichier sera fait au moyen d'une méthode utilitaire `readFromFile` de la classe `Territory` ;
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.**
5. **Ne négligez ni les constructeurs, ni les droits d'accès et les modificateurs.**
6. le droit `protected` ne doit pas être utilisé pour les attributs ;
7. votre conception ne doit pas nécessiter de dupliquer du code ;
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.

Question 2 : Programmation [5 points]

Donnez le corps de la méthode `update` des cellules d'eau.

SUJET À LA PAGE SUIVANTE

Exercice 2 : Concepts de base [20 points]

Répondez clairement et succinctement aux questions suivantes :

1. [3 points] Qu'affiche le code suivant :

```

0. class A {
1.     private int a;
2.
3.     public void setA(int a) {
4.         this.a = a;
5.     }
6.     public int getA() {
7.         return a;
8.     }
9. }
10.
11. class Question {
12.     public static void main(String[] args) {
13.         A a = new A();
14.         m(a);
15.         System.out.println(a.getA());
16.     }
17.
18.     public static void m(A a) {
19.         A a1 = new A();
20.         a1.setA(2);
21.         a = a1;
22.     }
23. }
```

Justifiez brièvement votre réponse.

2. [5 points] Soit le code suivant :

```

0. public static void main(String[] args) {
1.     for (SemaphoreColor color : SemaphoreColor.values()) {
2.         String frName = color.frenchName();
3.         System.out.print(frName);
4.         if (color.canPass()) {
5.             System.out.print(" peut ");
6.         }
7.         else {
8.             System.out.print(" ne peut pas ");
9.         }
10.        System.out.println("passer");
11.    }
12. }
```

Il affiche en sortie :

```

rouge ne peut pas passer
vert peut passer
jaune ne peut pas passer
```

Donnez le code Java du type énuméré `SemaphoreColor` permettant d'obtenir ces affichages.

Suite au verso ➞

3. [7 points] Soit le code suivant :

```

0. class A {
1.     private int a = 1;
2.     public A() {}
3.     public A(int a) { this.a = a; }
4.     public int getA() { return a; }
5.
6.     public String toString() { return "A: " + a; }
7. }
8.
9. class B extends A {
10.     private int a = 2;
11.     private int b = 3;
12.
13.     public String toString() {
14.         return "B: " + getB() + " " + getA();
15.     }
16.
17.     public int getA() { return a; }
18.     public int getB() { return b; }
19. }
20.
21. class Construct {
22.     public static void main(String[] args) {
23.         A a1 = new A();
24.         B b1 = new B();
25.         A b2 = new B();
26.
27.         System.out.println(a1);
28.         System.out.println(b1);
29.         System.out.println(b2);
30.     }
31. }

```

- (a) Combien d'attribut(s) a un objet de type B ? citez-les.
- (b) Combien de constructeur(s) a la class A ? citez-les
- (c) Combien de constructeur(s) a la class B ? citez-les
- (d) Qu'affiche le programme ?
- (e) Qu'affiche t-il si on supprime la ligne 17 ?
- (f) Compile t-il si l'on remplace la ligne 3 par :

```
public A(int a) { this(); this.a = a; }
```

Justifiez brièvement votre réponse
- (g) Que se passe t-il si l'on supprime la ligne 2 ? (on supposera que l'on n'a pas fait la modification du point précédent)

4. [5 points] Soit le code suivant :

```
0.  abstract class A {  
1.    private int a;  
2.    public A () { this(100); }  
3.    public A(int unA) { a = unA; }  
4.    abstract public A m();  
5.  }  
6.  
7.  class B extends A  
8.  {  
9.      private int b;  
10.     public B(int unA, int unB) { a = unA; b = unB ; }  
11.     @Override  
12.     public void m(int i) { A a = new A(); }  
13. }
```

Il contient trois (3) fautes qui l'empêchent de compiler.

- (a) Expliquez quelles sont ces fautes.
- (b) Comment rendre le programme compilable en ne retouchant que la ligne 10 et 12 ?

Suite au verso ➞

Exercice 3 : Gestion des exception [15 points]

Le programme de la page suivante compile et s'exécute sans erreur. Sachant que la méthode `remove` des `ArrayList` renvoie une `IndexOutOfBoundsException` si on l'applique à une liste vide :

1. Qu'affiche le programme ?
2. Qu'affiche t-il si à la ligne 91 on ajoute au paquet un article de poids 10 au lieu de celui de poids 9 ?
3. Peut-on remplacer la ligne 71/72 par

```
public void accept(Parcel parcel) throws RuntimeException {  
    ?
```

4. Qu'affiche t-il si l'on remplace la ligne 84 par

```
PostOffice office = null;
```

Justifiez brièvement vos réponses.

```

0. import java.util.ArrayList;
1. import java.util.List;
2.
3. // Article à mettre dans un paquet
4. class Item {
5.     private double weight; // poids
6.     private double price; //prix
7.     public Item(double w, double p) {
8.         weight = w;
9.         price = p;
10.    }
11.    public String toString() {
12.        return "Item(" + weight + ","
13.            + price + ")";
14.    }
15.    public double getWeight() {
16.        return weight;
17.    }
18. }
19. // Paquet
20. class Parcel {
21.     boolean processed = false;
22.     private List <Item> items =
23.         new ArrayList<Item>();
24.     protected void removeItem() {
25.         items.remove(0);
26.     }
27.     protected void addItem(double w, double p) {
28.         items.add(new Item(w,p));
29.     }
30.
31.     public void reset() {
32.         processed = false;
33.     }
34.     public void sendTo(PostOffice office) {
35.         do {
36.
37.             try {
38.                 displayContent();
39.                 office.accept(this);
40.                 processed = true;
41.                 System.out.println("Sent");
42.             } catch(RuntimeException e) {
43.                 removeItem();
44.             } catch(Exception e) {
45.                 System.out.println("Cannot be sent : "
46.                     + e.getMessage());
47.                 processed = true;
48.             }
49.         } while (! processed);
50.     }

```

```

51.     public void displayContent() {
52.         for (Item item : items) {
53.             System.out.print(item);
54.         }
55.         System.out.println();
56.     }
57.     public double getWeight() {
58.         double weight = 0.0;
59.         for (Item item : items) {
60.             weight += item.getWeight();
61.         }
62.         return weight;
63.     }
64.     public boolean isEmpty() {
65.         return (items.size() == 0);
66.     }
67. } // Fin classe Parcel
68. // Bureau de poste
69. class PostOffice {
70.     public final double MAX_WEIGHT = 10.0;
71.     public void accept(Parcel parcel)
72.         throws Exception
73.     {
74.         if (parcel.getWeight() >= MAX_WEIGHT){
75.             throw new RuntimeException("Heavy");
76.         }
77.         if (parcel.isEmpty()) {
78.             throw new Exception("Suspicious");
79.         }
80.     }
81. }
82. class Question {
83.     public static void main(String[] args) {
84.         PostOffice office = new PostOffice();
85.         Parcel parcel = new Parcel();
86.         parcel.sendTo(office);
87.         System.out.println("@@@@");
88.         parcel.reset();
89.         parcel.addItem(4.0, 2.5);
90.         parcel.addItem(5.0, 3.0);
91.         parcel.addItem(9.0, 5.5);
92.         parcel.sendTo(office);
93.     }
94. }

```

Suite au verso ➞

Exercice 4 : Déroulement de programme [15 points]

Le programme de la page suivante compile et s'exécute sans erreurs. Qu'affiche t-il ? 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.**

```

0.  interface Named {
1.      String getName();
2.  }
3.
4.  abstract class C implements Named {
5.      public static int count = 0;
6.      public C() {
7.          ++count;
8.      }
9.
10.     public abstract void a(I1 v);
11.
12.     @Override
13.     public String getName() {
14.         return prefix() + suffix();
15.     }
16.
17.     abstract String prefix();
18.
19.     String suffix() {
20.         return "C";
21.     }
22. }
23.
24. class C1 extends C {
25.     @Override
26.     String prefix() {
27.         return "C1::";
28.     }
29.     public void a (I1 v) {
30.         ((I2)v).m(this);
31.     }
32. }
33.
34. class C2 extends C {
35.     @Override
36.     String prefix() {
37.         return "C2::";
38.     }
39.     public void a (I1 v) {
40.         ((I2)v).m(this);
41.     }
42. }
43.
44.
45. class C3 extends C {
46.     @Override
47.     String prefix() {
48.         return "C3::";
49.     }
50.
51.     public void a (I1 v) {
52.         ((I2)v).m(this);
53.     }
54. }

```

```

55. interface I1 {
56.     default void m (C c) {
57.         System.out.println("nil");
58.     }
59. }
60. interface I2 extends I1 {
61.     default void m(C1 c) {
62.         System.out.println ("Nothing with "
63.                               + c.getName());
64.     }
65.     default void m(C2 c) {
66.         System.out.println ("Nothing with "
67.                               + c.getName());
68.     }
69. }
70. class A implements Named {
71.     private H h = new H();
72.
73.     @Override
74.     public String getName() {
75.         return "A";
76.     }
77.
78.     public void n(C c) {
79.         c.a(h);
80.     }
81.
82.     private class H implements I2 {
83.         public void m(C2 c) {
84.             System.out.println("In a world of "
85.                                   + c.count);
86.             System.out.println(getName() +
87.                                   " checks "
88.                                   + c.getName());
89.         }
90.     }
91. }
92. class Deroulement {
93.     public static void main(String[] args) {
94.         Named[] collect1 = {new C1(), new C2(),
95.                               new C3()};
96.
97.         for (Named n : collect1) {
98.             System.out.println(n.getName());
99.         }
100.
101.         C[] collect2 = {new C1(), new C2(),
102.                           new C3()};
103.
104.         A a = new A();
105.
106.         for (C c : collect2) {
107.             a.n(c);
108.         }
109.     }
110. }

```