EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communication
Cours de programmation aux sections IN/SC
Sam J.

INTRODUCTION A LA PROGRAMMATION

Test Semestre I

Instructions :

- You have an hour and forty-five minutes to complete this exam (13 :15 - 15 :00) ;
- There are 105 points in total (among which approximately 25 are optional) ;
- This exam is open-book ;
- Please, be sure to **use the appropiate answer sheet for each question**
- This exam consists in 4 independent exercises.

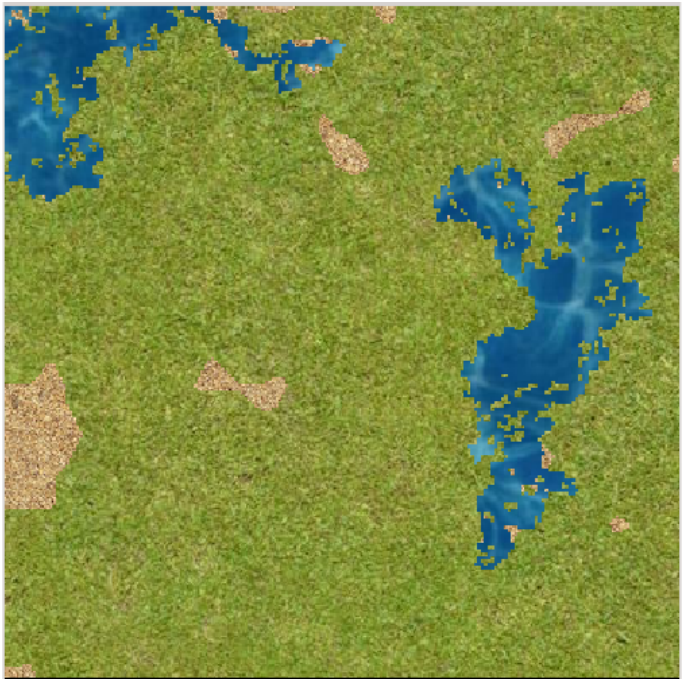
  You can start with the one that you feel the most comfortable with.

  - 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]

The goal of this exercise is to simulate the evolution of cultivable areas through the seasons. The territory of the cultivable areas resembles the following :



A part of the program that you need to develop is given in the annex. Specifically, you may find in the annex the class `Positionable` and the interface `Updatable`.
Please, read the entire material for this exercise before writing your answers Note that in this exercise, we do not care about the graphical part of the simulation.

A territory (class `Territory`) is represented by a grid where each grid box contains a layer of soil which can be covered either by a layer of grass, or by a layer of water, or by both. The soil always exists.

The blue boxes of the grid are defined as the boxes for which the water quantity exceeds the threshold `WATER_TRESHOLD`. This threshold is constant for all territories.

The green boxes of the grid are the boxes that contain grass, and for which the water quantity is less than `WATER_TRESHOLD`.
The cultivable area of a territory is given by the number of green boxes.
You will find below a more detailed description on how to model grid boxes and their layers.

A territory is characterized by the *climatic zone* it belongs to. The *climatic zone* is either tropical or temperate.

In addition, you must simulate the four seasons : autumn, winter, spring and summer.

The required functionality is to :

1. Initialize a territory given as parameter its climate and a filename. The file associated to the territory contains the information required to initialize the grid, namely its dimensions (you can assume that the grid is a square) and the nature of each box of the grid. The exact initialization of the grid is out of the scope of the exercise, but might throw a `IOException` (of type "checked").
2. Simulate the evolution of the territory given a time step `dt` (method `simulate`).
3. Modify the season for which we want to simulate the territory evolution.

4. Drop a given volume `v` of rain on the territory. Each grid box will receive a quantity of water which is calculated based on `v`. The specific calculation is out of the scope of the exercise.
5. Know the cultivable area (at any time during the simulation).

Climatic zones A climatic zone is characterized by the fact that it offers, as a functionality, the calculation of the *average temperature of the season* :`double computeAverageTemp(Saison s)`.

We distinguish two categories of climatic zones : the *temperate* and the *tropical*. For each category, the average temperature is computed differently. The exact method of the calculation is out of the scope of this question.

Boxes and cells Each grid box can be modeled as a superposition of three *cells* (the soil cell, the water cell, and the grass cell). Each cell has a position on the grid. The cells that do not exist in a box will take the value `null`. (when a box contains only a soil cell for example, the other cells will take `null`).

The *water cells* are characterized by the quantity of water they have. Similarly, the *grass cells* are characterized by the quantity of grass.

The quantity of water that exists in a box of the grid is therefore the quantity of water that exists in its water cell. In case of rain, the quantity of water received is added to the "water" cell, and if the water cell does not exist, it must be created.

The water cells evolve with time by diffusing 20% of their water quantity on their neighboring grid boxes. If the average temperature of the territory is more than a certain threshold `EVAPORATION_TRESHOLD` (constant and common for all territories), then it evaporates by a certain quantity depending on the elapsed time. You need to provide this type of cells with the method `evaporate` which simulates the evaporation of the water. If there is no more water, then the water cell disappears (becomes `null` in the corresponding box).

The grass quantity of the grass cells decreases or increases depending on the elapsed time, the average temperature, and the quantity of water present in the corresponding boxes. You will provide this type of cells with the method `grow` which computes the increase or decrease of the grass quantity. If there is no more grass (zero quantity), then the grass cell disappears.

SUJET À LA PAGE SUIVANTE

Question 1 : Conception [50 points]

You have to write the class(es) that fill the missing elements, and which implement the desired functionalities ; only method signatures are required. **You are not asked to write the entire program, or method bodies. We only ask for class attributes and method signatures.** You can describe classes and their content using diagrams or pseudo-code in Java. The following constraints must be satisfied :

1. You can assume that a main program exists, that it creates the `Territory`, and calls its `simulate` method in a loop. This method allows to advance the state of the simulation, i.e., all the entities involved (via the `update` method, as specified by the `Updatable` interface) ;
2. The initialization of the grid based on the input file is performed using the auxiliary method `readFromFile` of the class `Territory` ;
3. Classes must contain the necessary elements required to implement all the described functionalities, **including the ones that are subtly suggested in the text** (in particular those needed to implement the `update` methods) ;
4. Regarding the non trivial methods, you should rely on **brief comments** to clarify the method's functionality or its return types where needed.
5. **Do not forget constructors, or access rights and modifiers.**
6. You are not allowed to use the `protected` modifier for attributes ;
7. your design must avoid code duplication ;
8. Only the getters and setters necessary to the simulation must be provided ;
9. You do not have to write import statements ;

Question 2 : Programmation [5 points]

Write the body of the water cells' `update` method.

SUJET À LA PAGE SUIVANTE

Exercice 2 : Basic Concepts [20 points]

Answer clearly and briefly to the following questions :

1. [**3 points**] What does the following code displays :

```
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. }
```

   Justify your answer briefly.

2. [**5 points**] Consider the following code :

```
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. }
```

   It displays the following lines :

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

   Write the Java code for the enumeration `SemaphoreColor` needed to obtain these outputs.

Suite au verso ☞

3. [**7 points**] Consider the following code :

```
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) How many attribute(s) an object of type `B` has ? Name them.

(b) How many constructor(s) in the class `A` has ? Name them.

(c) How many constructor(s) in the class `B` has ? Name them.

(d) What does the program display ?

(e) What does the program display if we delete the line 17 ?

(f) Would it compile if we replace the line 3 by the following :

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

Justify your answer briefly.

(g) What happens if we delete the line 2 ? (We suppose that we did not do the modification to line 3 suggested by the previous question.)

4. **[5 points]** Consider the following code :

```
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. }
```

It contains three (3) faults which prevent it from compiling.

(a) Explain what these faults are.

(b) How to make the program compilable by modifying only line 10 and 12 ?

Suite au verso ☞

Exercice 3 : Exception handling [15 points]

The code in the next page compiles and runs without error. Knowing that the method `remove` of `ArrayList` throws an `IndexOutOfBoundsException` if we apply it to an empty array :

1. What does the program display ?
2. What does it display if an item of weight 10 is added to the package instead of that of weight 9 on line 91 ?
3. Can we replace the line 71/72 by :

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

   ?
4. What does it display if we replace the line 84 by :

   ```
   PostOffice office = null;
   ```

Justify your answers briefly.

```
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 : Program Flow [15 points]

The code in the next page compiles and runs without error. What does it display ? Explain its execution briefly and clearly. **Here you should bot paraphrase the code but *explain* the steps of the execution of the program.**

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