

INTRODUCTION A LA PROGRAMMATION

Test Semester I

Instructions :

- The exam's duration is one hour and forty-five minutes (from 8h15 to 10h00);
- The maximum number of points is 110, of which approximately 15 are optional;
- All paper documentation is allowed;
- Please put **only one exercise per sheet**, and **write your SCIPER number** on *every* sheet.
A sheet without SCIPER number will not be corrected;
- This exam is made of 3 independent exercises.
- The exercises are of varying difficulty. Start with those you are most comfortable with.

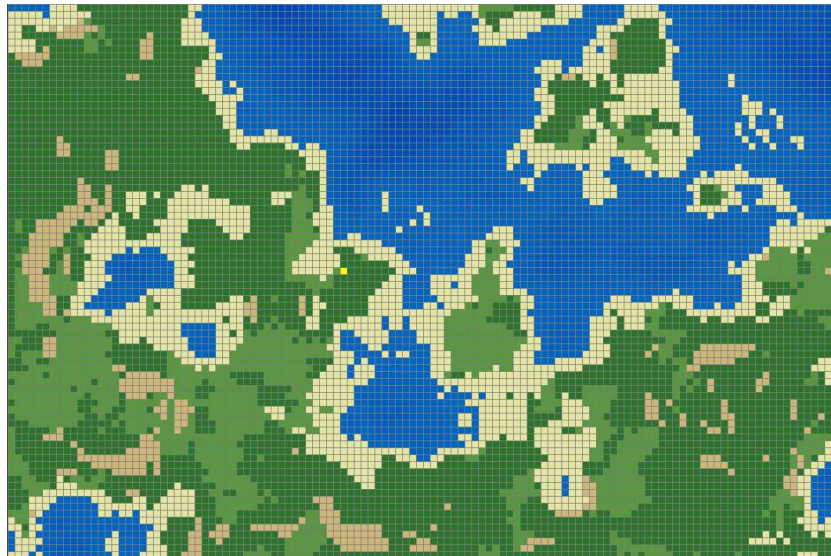
DESCRIPTION ON THE NEXT PAGE

DESCRIPTION ON THE NEXT PAGE

Exercice 1 : Object Oriented Conception [52 points]

Different kind of trees grow on a land (**Terrain**). We wish to graphically simulate the growth of the trees, in order to compute what parts of the land are below trees.

The land is rectangular, and tiled in cells. The nature of the cells will influence the growth of the trees.



A part of the program, inspired by your second mini-project, is given in the annex. It consists of :

- a minimal modelization of a land;
- a main program `Program`;
- an interface `Simulable`;
- an enumeration `Ground`.

Familiarize yourself with this material before starting, including the comments in the material..

The land's cells

We distinguish four cell types: chalk cells, clay cells, water cells, and rock cells.

Each cell knows its own position in the land (you will assume that you have a class `Vector` similar to your project which represents a pair of `int`). A cell can potentially contain a tree; we assume that there can only be a *single* tree per cell.

Cells are colored given their nature (i.e. there is a specific color for water cells, another for rock cells, etc). The presence of a tree and its length (see below) also impacts the cell's color. The precise computation of a color is not of interest for this exercise.

Trees

Trees belong to a specific category. They are all characterized by a *size*, which is bound by a maximum. This maximum is specific to the category of the tree, and cannot be determined otherwise. For instance, the maximal size of a conifer is fixed at 10m, and 15m for the hardwood trees (see below for the different trees). When created, a tree can have a non-zero size.

Tree categories : trees are either conifers or hardwood trees. They cannot occupy a water cell, nor a rock cell, nor a cell already occupied by another tree.

If a tree is in a cell, it grows in a way specific to its category, also depending on the time delta `deltaTime` from the simulation, and the nature of the cell in which it is. Again, the precise computation of the growth of one tree is not of interest for this exercise.

Expansion of the tree canopy Every simulation cycle (call to `update`), each tree has a given probability to replicate (and produce a tree of the same category) next to it. The conifers replicate in a cell chosen randomly in a radius around the cell. The hardwood trees also replicate the same way, but in addition, have also the same probability to replicate anywhere on the land. The radius is specific to the category of the tree; assume they are computed given a method `getExpansionRadius()`. The computation of probabilities will happen in a method `boolean mustExpand()` returning true if the tree must replicate, and false otherwise. The implementation of `mustExpand` are specific to the category of the tree.

When replicating, if the cell chosen for the new tree is a water cell, a rock cell, or already contains a tree, nothing happens. Otherwise, a new tree of the chosen type is created in the cell.

Assume that a tree knows the cell in which it grows, and a cell knows the land to which it belongs. The use of protected getters such as `getOwner()` is authorized, and you will assume that all classes that you will create belong to the same package as the class `Terrain`.

Question 1 : Conception [44 points]


1. Please write the class(es) needed to complete the class `Terrain`, and needed to create the functionalities wanted. Write only the headers of the methods. **We do not ask you to write the complete program, nor the methods body, only the attributes and the methods headers.** You may describe the classes and their content via class diagrams, or in Java pseudo-code.
2. The classes need to contain the appropriate members to implement all functionalities **explicitely or implicitely** described by the instructions;
3. For methods in which some details are unknown, **you'll write a brief comment about the meaning of the return type and the goal of the method.**
4. Do not forget constructors, access rights and modifiers.

The constraints are:

1. the access right `protected` cannot be used for attributes;
2. your conception will avoid code deduplication;
3. you will assume that a graphic toolbox is already available, use the classes provided in the annex, and assume the existence of a class `Vector` representing a pair of coordinates;
4. you will only write getters and setters when they are needed by the simulation;
5. you will ignore all import directives;

Question 2 : Programmation [8 points]

Now, please complete the body of the method used to replicate a hardwood tree. The newly-created trees have a height of 1.0m.

Turn the page 

Exercise 2 : Concepts [36 points]

Please answer clearly and briefly to the following questions :

1. [6 points] Given the following code :

```

0.  class A {
1.      private int a = 1;
2.
3.      public A() {
4.          System.out.println("A: " + a);
5.      }
6.
7.      public A (int a) {
8.          this();
9.          this.a = a;
10.     }
11.
12. }
13.
14. class B extends A {
15.     private int a;
16.
17.     public B(int a) {
18.         super(a);
19.         System.out.println("B: " + this.a);
20.     }
21.
22.     public B() {
23.         this.a = 4;
24.         System.out.println("B: " + a);
25.     }
26. }
27.
28. class P {
29.
30.     public static void main (String[] args) {
31.         A a = new A(2);
32.         B b1 = new B(2);
33.         B b2 = new B();
34.     }
35. }

```

- (a) What is printed? Please justify briefly.
- (b) On line 19, could we replace `this.a` par `a` without breaking the compilation ? Justify briefly.
- (c) Could we add `this()` just before line 23? Justify briefly.
- (d) Could we swap lines 8 et 9? Justify briefly.

2. [6 points] What is printed? Please justify briefly.

```

0. import java.util.ArrayList;
1. import java.util.List;
2.
3. class P {
4.     public static void f(List<Integer> list) {
5.         for (Integer i : list) {
6.             System.out.print(i + " ");
7.         }
8.         System.out.println();
9.     }
10.
11.     public static void g(List<Integer> list) {
12.         for (Integer i : list) {
13.             ++i;
14.         }
15.         list.remove(0);
16.     }
17.
18.     public static void h (List<Integer> list) {
19.         List<Integer> other = new ArrayList<Integer>();
20.
21.         for (Integer i : list) {
22.             other.add(i+2);
23.         }
24.         list = other;
25.     }
26.
27.     public static void main(String[] args) {
28.         List<Integer> list = new ArrayList<Integer>();
29.         list.add(1);
30.         list.add(5);
31.         list.add(22);
32.
33.         f(list);
34.         g(list);
35.         f(list);
36.         h(list);
37.         f(list);
38.     }
39. }

```

Turn the page ➞

3. [7 points] Given the following code :

```

0. class A {
1.     public final static A I = new A(2);
2.     private int a;
3.
4.     private A(int a) {
5.         this.a = a;
6.     }
7.
8.     public A() {
9.         this.a = 1;
10.    }
11.
12.    @Override
13.    public String toString() {
14.        return "value " + a;
15.    }
16.
17. }
18.
19. class S {
20.     A a = new A();
21.
22.     public static void main(String[] args) {
23.
24.     }
25. }

```

For all following cases, indicate if we could add that line (and only that line) in the method `main` without breaking the compilation. Justify briefly in each case.

- (a) `a = new A(2);`
- (b) `A.I = new A(3);`
- (c) `A.I = new A();`
- (d) `System.out.println(a);`
- (e) `System.out.println(A.I);`
- (f) `System.out.println(new A().I);`
- (g) `System.out.println(A.a);`

4. [4 points] In your second mini-project, your main program has a line similar to:

```
Game game = new BikeGame();
```

What is the reason behind not using the following line instead:

```
ActorGame game = new BikeGame();
```


5. [7 points] Given the following code snippet:

```

0. import java.util.Scanner;
1.
2. class Exception2 {
3.     private static Scanner clavier = new Scanner(System.in);
4.
5.     public static void main(String[] args) {
6.         String a = clavier.nextLine();
7.         String b = clavier.nextLine();
8.
9.         try {
10.            System.out.print("** ");
11.            System.out.println(f(a,b) + " **");
12.            System.out.println("Voila !");
13.        }
14.        catch (Exception err) {
15.            System.out.println(err.getMessage());
16.        }
17.        finally {
18.            System.out.println("C'est parti !");
19.        }
20.    }
21.
22.    public static int f(String x, String y) throws Exception {
23.        if (x.length() < y.length()) {
24.            throw new Exception("Non !");
25.        }
26.
27.        int z = 0;
28.        for (int i = 0; i + y.length() < x.length(); ++i) {
29.            if (x.substring(i, i+y.length()).equals(y)) {
30.                System.out.print(i + " ");
31.                ++z;
32.            }
33.        }
34.        System.out.println();
35.        return z;
36.    }

```

What is printed if we input « HoHoHoHoHoHo » then « oH »?

What is printed if we input « oH » then « HoHoHoHoHoHo »? Briefly justify your answers.

Note :

- The call `s.substring(int begin, int end)` returns the sub-string of `s` between `begin` (inclusive) and `end` (non-inclusive).
- The lines 6 and 7 read the two afore mentioned strings in sequence (there is no trick about a carriage return symbol hidden in the reading buffer).

Turn the page ➞

6. [**6 points**] Suppose we wrote the following in the class `Truc` :

```
abstract int f();
```

For each following case, indicate whether it is correct or not, and justify each time briefly your answer.

- (a) The method `f` always returns zero.
- (b) It is not possible to declare a variable of type `Truc`.
- (c) It is not possible to instantiate a variable of type `Truc`.
- (d) It is not possible to redefine the method `f` in the sub-classes of `Truc`.
- (e) `Truc` must have several sub-classes.
- (f) The class `Truc` must be declared as `abstract`.

Exercise 3 : Program execution flow [22 points]

The following program compiles and executes without error.

```

0.  interface Builder {
1.      abstract void expand(Kingdom k);
2.  };
3.
4.  abstract class Kingdom {
5.      private int size; // surface du royaume
6.      private int money; // richesse
7.
8.      public abstract void launch(Builder b);
9.
10.     public Kingdom(int aSize,
11.                     int someMoney) {
12.         size = aSize;
13.         money = someMoney;
14.     }
15.
16.     public void grow(int aSize,
17.                      int someMoney) {
18.         size += aSize;
19.         money += someMoney;
20.     }
21.
22.     public String toString() {
23.         return ("Size: " + size + "\n" +
24.                "Money: " + money + "\n");
25.     }
26. }
27.
28. class Alfheim extends Kingdom {
29.     public Alfheim(int aSize,
30.                    int someMoney) {
31.         super(aSize, someMoney);
32.     }
33.
34.     public String toString() {
35.         return ("* Royaume des Elfes *\n" +
36.                super.toString());
37.     }
38.
39.     public void launch(Builder b) {
40.         b.expand(this);
41.     }
42. }
43.
44. class Nidavel extends Kingdom {
45.     public Nidavel(int aSize,
46.                    int someMoney) {
47.         super(aSize, someMoney);
48.     }
49.
50.     public String toString() {
51.         return ("* Royaume des Nains *\n" +
52.                super.toString());
53.     }
54.
55.     public void grow(int aSize,
56.                      int someMoney) {
57.         super.grow(2*aSize, 2*someMoney);
58.     }
59.
60.     public void launch(Builder b) {
61.         b.expand(this);
62.     }
63. }
64. }
65.
66. class CityBuilder implements Builder {
67.     public void expand(Kingdom kingdom) {
68.         kingdom.grow(10, 20);
69.     }
70. }
71.
72. class KingdomExpansion {
73.     public static void main(String[] args) {
74.         Kingdom elves = new Alfheim(20, 30);
75.         Kingdom dwarfs = new Nidavel(30, 10);
76.         System.out.println(elves);
77.         System.out.println(dwarfs);
78.
79.         System.out.println("-----");
80.         Builder cb = new CityBuilder();
81.         elves.launch(cb);
82.         dwarfs.launch(cb);
83.
84.         System.out.println(elves);
85.         System.out.println(dwarfs);
86.     }
87. }

```

What does it print ? Justify briefly. The goal here is not to paraphrase the code, but to *explain* the steps and the flow of the program.