

# INTRODUCTION TO PROGRAMMING

## Semester exam

### Instructions :

- You have two hours to do this exam (10h00 - 12h00).
- Maximum 90 points (+ 15 bonus).
- Attention : There are also statements on the back of the pages.
- You must **write in black or dark blue ink**, not with a pencil and not with any other color. Do **not use an erasable pen** (heat might cause part of your writing to be lost).
- Any paper documentation is allowed ; However, you may not use a personal computer, cell phone, or other electronic equipment.
- Answer on the sheets given to you and in the places provided. **Do not write your answers in this document**.
- Do not attach any additional sheets ; **only answer sheets that were distributed to you will be graded**.
- You can answer the questions in French or in English.
- The exam consists of 3 independent exercises. **You can start with whichever you want**
  - Exercise 1 : **55** points.
  - Exercise 2 : **35** points.
  - Exercise 3 : **15** points (bonus).

continue at the back of the page ↗

## Exercise 1 : OO design [55 points]

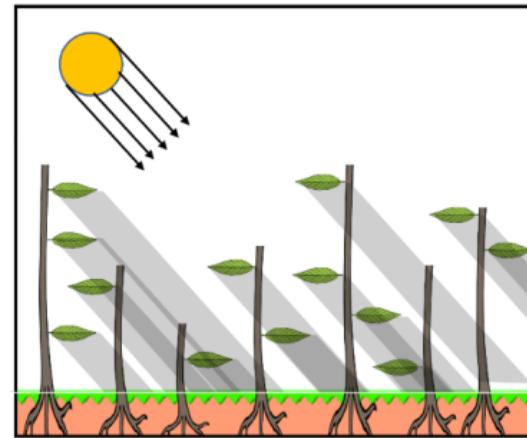
**It is important to read fully each question in this document before answering it. Also, take the time to read the design constraints and consult the code provided in the appendix before starting the exercise.**

This exercise contains 4 questions. Questions 1 and 2 are independent.

We are interested here in simulating, in a very simplified way, plants that grow, reproduce, and die, in an environment impacted by light sources.

Example : plants under sunlight

- they gain energy by absorbing light with their leaves ;
- and spend it on growth and reproduction.



**Design constraints** When solving this exercise, the following constraints must be respected :

1. When you are asked to write the body of a class you must do so in Java syntax and write their attributes and methods but without bodies (except for the methods where writing the body is explicitly requested).
2. You'll assume that a main program exists, that it creates an **Environment** object, that it creates plants and light sources there, and calls their **update** methods in a simulation loop. This method allows the entities involved to evolve over time (using the **update** methods, according to the **Updatable** interface).
3. The classes must contain the members necessary to implement all the desired functionalities, **whether explicitly requested or implied by the specifications of this statement** (in particular by the implementation of the methods **update**) ;
4. For the non-trivial methods, **you'll note with a brief comment what the return type means and what they basically do, and what other classes they use**.
5. **Don't forget the constructors or the access modifiers.**
6. **protected must not be used** as an access modifier for attributes.
7. Your design should be well-modularized.
8. You should only propose getters/setters when they are necessary for the implementation of the simulation described.
9. There is no need to write the import statements.

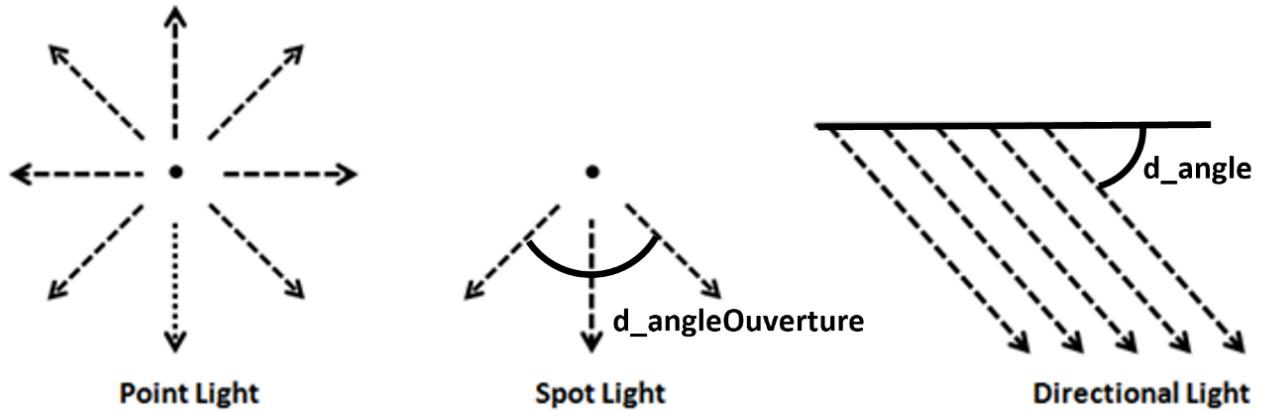


FIGURE 1 – Light Sources

**Question 1 : Light Sources [23 marks]**

Light sources can be either *point-shaped*, *spot*, or *directional* (see figure1).

- A point-shaped light source spreads light everywhere. It is characterized by its intensity (`double`) as well as by its position (`Vec2d`).
- A spot-shaped light source spreads the light in a given direction with a given *opening angle* (mentioned in Figure 1 as *d\_angleOuverture*). It is therefore characterized by its intensity (`double`), its position (`Vec2d`), the direction in which it shines and its opening angle.
- A directional light source spreads the light according to a given *angle of inclination*, and is therefore characterized by its intensity (`double`) as well as its angle of inclination (`double`).

The characteristics of each type of light source must be initialized during their construction by providing the necessary data as parameters.

**Evolution over time :** For the point-shaped and spot-shaped light sources, the intensity evolves according to algorithms that are specific to them. Directional light sources, on the other hand, only see their angle vary, by rotation, as a function of time. The precise details of the implementation of the rotations and change of intensity do not interest us.

Light sources emit energy. The quantity of energy (`double`) emitted over a time step *dt*, at a given point (`Vec2d`) can be calculated according to specific methods for each source.

**Write classes to represent this model. You will assume that the data necessary for the initialization of the objects are passed as parameters to the constructors.**

continue at the back of the page ↗

## Question 2 : Leaves and plants (17 points)

A *leaf* has a position (`Vec2d`). A *plant* has a set of leaves. It is characterized by a position (`Vec2d`), an energy level, and an age. These data are all of type `double`. The maximum possible age is a variable *shared by all plants*.

A plant can be created without a leaf, with zero age. It can also be created with a given number of leaves and a given age. The rest of the attributes must be initialized using values passed as parameters.

**Evolution over time :** at each time step `dt`, plants acquire energy by absorbing energy emitted by all light sources in the environment.

For a given light source, a plant in position `p` absorbs, per leaf and according to the position of this one, a percentage of the energy emitted by the source in `p`. This percentage is a constant common to all plants.

If a plant's energy level exceeds a certain threshold (a common constant for all leaves), a leaf is added to a random position on the plant, and the plant's energy level is halved. The methods for the random positioning do not interest us.

Moreover, at each time step `dt`, the plant ages (sees its age increase by `dt`) and loses a time-dependent amount of energy. Plants with zero energy die and must disappear from the simulation. When a plant exceeds the maximum age it also dies. Before disappearing from the environment, a plant leaves around its position some new shoots (the exact details do not interest us).

- Write the classes to represent the model described above. You will assume that the data necessary for the initialization of the objects are passed as parameters to the constructors.
- Give all the method signatures that needed to be added to the `Environment` class to perform the required processing, commenting on their role.

## Question 3 : Programming (6 marks)

Write the body of the plant's `update` method. The body of any other method called by `update` is not necessary. However, you must have commented their header to explain what they do and what important methods they invoke. You can give any name you want to any constants that are not explicitly mentioned in the problem statement.

## Question 4 : Analysis [9 marks]

1. Why does the class `Environment` not implement the `update` interface ?
2. What criticism would you have of adding the following constructor to the `Environment` class :  
`public Environment(double width, double height, List<Plant>)`
3. Assume all classes are in the same package. How can we, by adding an interface, prevent the `addLight` method of `Environment` from accidentally making plants evolve (by invoking their `update` method) ? Give a brief explanation in natural language indicating what should be changed in the existing code.

## Exercise 2 : Basic Concepts [35 points]

Clearly and succinctly answer the following questions :

1. [5 marks] Consider the following code :

```

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) What does line 22 print ? Justify briefly.
- (b) Line 23 displays 5.0. Re-write the code in line 17 to make it display the attribute x of the superclass that has value 10.
2. [10 marks] Indicate for each of the following assertions whether they are correct or incorrect :
- An abstract class can only have abstract methods.
  - An inner class can access a private member of its outer class.
  - A class cannot directly extend multiple classes.
  - A class can implement multiple interfaces.
  - An abstract class cannot implement an interface.
  - A method in an abstract class cannot call an abstract method.
  - The constructors in an interface can only be default constructors.
  - An abstract class can have only default constructors.
  - An interface can extend another interface.
  - All members of an interface are public.

continue at the back of the page ↗

3. [11 points] Given the following code :

```

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) What does it display ? Briefly justify.
- (b) Does it still compile if we modify the header of method `m` like this :  
`public static void m (final A a, B b)`  
 Briefly justify.
- (c) Does it still compile if we declare class `B` as `final` ? Same question if we declare class `A` as `final` ? Justify briefly.

4. [9 points] Consider the following class declarations :

```

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)

Indicate for each of the following assertions whether they are correct or incorrect :

- (a) The import clause is not necessary in program (2) for the program to compile.
- (b) The import clause is not necessary in program (4) for the program to compile.
- (c) The method `m1()` can be invoked on the object `c4` (via a method of `C4`) without compiling problem.
- (d) The method `m1()` can be invoked on the object `c3` (via a method of `C3`) without compilation problems.
- (e) The method `m2()` can be called by the method `m1()` without compilation problems.
- (f) The method `m3()` can be called on the object `c2` (via a method of `C2`) without compilation error.

**continue at the back of the page ↗**

### Exercise 3 : Program flow [15 points]

The following program compiles and runs without errors (assuming the import directives are present).

```

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. For each of the following lines, specify whether it can be added to `main` :
  - (a) `System.out.println(r1.c(2));`
  - (b) `System.out.println(r0.c(2));`
  - (c) `System.out.println((R1)r0.c(2));`

Briefly justify.
2. What does the program display ? Briefly explain the program's control flow. **The point here is not to paraphrase the code, but to *explain* the steps and the flow of the program.**