

INTRODUCTION A LA PROGRAMMATION

Final Exam

Instructions :

- You have 1 :45mn to complete this exam (13 :15 - 15 :00).
- There are 105 points in total.
- Note that there are also instructions on the back side.
- You must **write using black or dark-blue ink**, do not use pencils or other colors. Do note use **erasable pens** (loss of information due to heat).
- Any paper document is permitted : However, you are not allowed to use a personal computer, nor a cellphone, nor any other electronic device.
- Reply on the answering sheets that were distributed to you **in the dedicated places. Do not answer on the statement.**
- Do not attach any additional piece of paper ; **only the distributed answering sheets will be graded.**
- You can reply to the questions in English or in French.
- This exam consists of 3 independent exercises.

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

- Exercise 1 : **50** points.
- Exercise 2 : **42** points.
- Exercise 3 : **13** points.

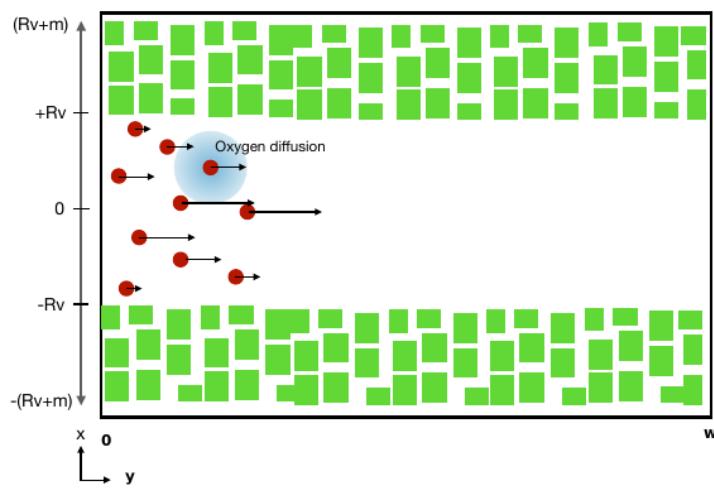
suite au verso ↗

Exercise 1 : Modeling OO [50 points]

It is important to review the exercise entirely, in particular the imposed constraints, and to consult the code provided in the appendix before starting answering questions.

Here, we are interested in graphically simulating a slice of blood vessel. The vessels' borders are composed of muscle cells which make up their structure. The figure below represents a 2D slice of a blood vessel, with a given length. Inside the vessel, liquid flows which carries blood cells.

- Red circle : blood cells. Transports and diffuses oxygen around their position.
- Green rectangle : muscle cells. These cells form the structure of the blood vessel. They need oxygen provided by blood cells
- We will consider position 0 at the middle of the vessel, with $+/- R_v$ being the lower/higher vessel limits, m the width of muscle cells layer and w the length of the slice of our simulated blood vessel.



For the next steps, you will suppose that a `Vec2d` class exists, as provided in the appendix, which allows to represent coordinates or speeds. You will also assume that the objects are drawables (we are not interested in the exact drawing procedures).

Blood vessel The slice of blood vessel to simulate (class `Vessel`) is characterized by the following variables : R_v , m et w and by *viscosity coefficient* of the liquid which circulates (all of type `double`). These characteristics will be initialized when the vessel is built (through the parameters of constructor(s)). Every cell in the vessel are characterized by a position (type `Vec2d`). You will also assume, for the sake of simplicity, that all cells are aware of the vessel they belong to. This information will be transmitted when cells are constructed. You will assume that the *pressure gradient* at a position p of the blood vessel, is calculable (as a `double`). **We are not interested into the precise modalities involved in its calculation .**

Blood cells The blood cells are characterized by a position. Upon their creation, they will be given a *speed* calculated through the *pressure gradient* at their position and the *viscosity coefficient* in the vessel. This speed remains constant after initialization. At each time step dt , cells travels a distance dz calculated in function of their current position and speed. Travel direction always follows $(0,1)$ in the coordinate system (x,y) . Moreover, when an *blood cell* arrives at the right extremity of the vessel, it must be destroyed.

At each time step dt , a random number of cells, with random positions are created inside the vessel. **We are not interested into the exact modalities of this random generation here.** Nevertheless, it must be guaranteed that positions (x,y) of generated cells lies in the following area of the vessel :

$$0 \leq y < w \text{ et } -R_v < x < R_v$$

Muscle cells a muscle cell is characterized by its position and its oxygen level a (`double`). It does not move and is simply supplied in oxygen by the circulating *blood cells*. At each time step dt , their oxygen level decreases following a known algorithm that depends on dt . In return, their oxygen level increases

by a quantity dependent on the number of blood cells in their neighborhood. You are allowed to make the following simplifications :

- It is not necessary to simulate oxygen diffusion by blood cells;
- There exists an algorithm that calculates the oxygen quantity present in the vessel in the neighborhood of a position p ; this calculation only depends on the number of red blood cells in the neighborhood of p ;
- The calculus of the oxygen gain by a muscle cell only depends on the oxygen quantity presents around its position.

When the oxygen level of muscle cells goes past zero, they die and must disappear of the simulation. Finally, all muscle cells will be created when the vessel is being constructed, following a known but unspecified algorithm. You will suppose that this algorithm generates a random number of cells each time it is being called.

Question 1 : Model [40 points]

You have to write the class(es) to represent the above problem in an object-oriented Java program, and which implement the desired functionalities. **You are not asked to write the entire program, or method bodies. We only ask for class attributes and method signatures.**

1. You can assume that a main program exists, that it creates the `Vessel`, and calls its `update` 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. 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);
3. Regarding the non trivial methods, you should rely on **brief comments** to clarify the method's functionality, its return types and which other methods it calls, where needed.
4. **Do not forget constructors, or access rights and modifiers.**
5. You are not allowed to use the `protected` modifier for attributes. The type `ArrayList` will be used to model collections.
6. Your design must be well modularized.
7. Only the getters and setters necessary to the simulation must be provided.
8. You do not have to write import statements.

Question 2 : Programming/Extensions [10 points]

1. **(6 points)** Give the body of the `update` of the class `Vessel`. Recall that there exists a method `Object remove(index i)` in the class `ArrayList`.
2. **(4 points)** Suppose we want to model the fact that a vessel which has lost more than half of its muscle cells (compared to the number at the vessel creation) is a dead vessel ("dead" is a state we want to be able to test). What do you propose to add to your model to execute this functionality without testing for types? Give the Java code to add (attributes and methods including bodies) while specifying where it has to be added.

Exercise 2 : Basic Concepts [42 points]

Give a brief and clear answer to the following questions :

1. [6 points] Consider the following code snippet :

```

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

- (a) What does it print ?
- (b) What does it print if we remove `this.` at line 5.

Justify briefly your answers.

2. [9 points] Consider the following code snippet :

```

1 abstract class A {
2     private int a=5;
3 };
4
5 class B extends A {
6     private int b=10;
7 };
8
9 class ToString
10 {
11     public static void main(String[] args)
12     {
13         A a = new B();
14         System.out.println(a);
15     }
16 }
```

Complete the code so that the main program prints the values of the attributes following the format : 5 10 . One single method must be added in each class. The main program must remain unchanged.

3. [15 points] Consider the following code snippet :

```

1  class W
2  {
3      private int w = 5;
4      public W() {
5          this(10);
6          System.out.println(w * 2);
7      }
8      public W(int w)
9      {
10         this.w = w;
11         System.out.println(w);
12     }
13 }
14
15 class X extends W
16 {
17     private int x = 2;
18     public X() { System.out.println(x); }
19 };
20
21 class Test
22 {
23     public static void main(String[] args)
24     {
25         W w = new W();
26         X x = new X();
27     }
28 }
```

(each of the answers below should be briefly justified).

- (a) What does the program print ?
- (b) Which error(s) are detected by the compiler if we remove the constructor at line 4 ? (You may assume that the line numbers remain unchanged).
- (c) What does it print if we remove only line 5 ?
- (d) Why will the compiler give an error message, if we add the line
`w = 12;`
in the body of the constructor at line 18 in the above code. What should be added in this constructor so that the attribute of W is initialized to 12 ?
- (e) In the original program, is it possible to add the statement
`int w = 12;`
just after line 17. If so, what does the program print ? if not, why ?

suite au verso ↗

4. [12 points] Consider the following code snippet :

```

1 import java.util.List;
2 import java.util.ArrayList;
3
4 abstract class Artist
5 {
6     public String getType() { return "actor"; }
7     public void sing() {}
8     public void paint() {}
9 }
10
11 class Singer extends Artist
12 {
13     public void sing() {
14         System.out.println( "I'm singing'n the rain");
15     }
16     public String getType() {
17         return "singer";
18     }
19 }
20
21 class Painter extends Artist
22 {
23     public void paint() {
24         System.out.println("I'm painting Guernica");
25     }
26     public String getType() {
27         return "painter";
28     }
29 };
30
31 class RiveGauche
32 {
33     private List <Artist> artists = new ArrayList<Artist>();
34     public void perform() {
35         for (Artist artist : artists) {
36             if (artist.getType() == "singer") artist.sing();
37             if (artist.getType() == "painter") artist.paint();
38         }
39     }
40     void welcome(Artist artist) {
41         if (artist != null) artists.add(artist);
42     }
43 }
44
45 class Polymorph
46 {
47     public static void main(String[] args) {
48
49         RiveGauche scene = new RiveGauche();
50
51         scene.welcome(new Singer());
52
53         scene.welcome(new Painter());;
54
55         scene.perform();
56     }
57 }
```

-
- (a) Why would a compilation error arise from the removal of the `sing` method associated with the class `Artist` ?
 - (b) Why would a compilation error arise from changing the `sing` method associated with the class `Artist` into an abstract method ?
 - (c) If we assume that other categories of artists can be added who do not necessarily sing or paint, what kind of criticism would you voice of the method `perform` and the way the class hierarchy of artists is modeled ? Give the modifications that must be done and where to fix these issues.
 - (d) How would you propose to model the fact that some artists produce works that can be exposed (painting, sculptures etc.) and that, in that case, we must be able to find where these works are exposed (a set of locations described by `String`) ?

suite au verso ↗

Exercise 3 : Program Development [13 points]

The following program compiles and executes without errors (we assume the imports are done). What does it print ? Justify your answer by explaining *important* points (do not paraphrase the code, but show that you have understood!).

```

1 interface D {
2     String d();
3 }
4
5 interface E extends D {
6     boolean e(A a);
7 }
8
9 enum Status {
10     RED(true),
11     GREEN(false), ;
12 }
13
14 private boolean s;
15
16 Status(boolean s) { this.s = s; }
17 boolean getS() { return s; }
18 }
19
20 abstract class A implements D
21 {
22     private String n;
23     private Status s;
24
25     public A(String n, Status s) {
26         this.n = n;
27         this.s = s;
28     }
29
30     public boolean e(A other) { return s.getS(); }
31     public abstract boolean f(B other);
32     public abstract boolean f(C other);
33
34     @Override
35     public String d() { return (n + s.toString()); }
36 }
37
38 class B extends A
39 {
40     public B() { super("small", Status.GREEN); }
41
42     public boolean e(A other) { return other.f(this); }
43     public boolean f(C other) { return true; }
44     public boolean f(B other) { return false; }
45 }
46

```

```

47
48 class C extends A
49 {
50     int h;
51     public C(int h) {
52         super("big", Status.RED);
53         this.h = h;
54     }
55     public boolean e(A other) {
56         return super.e(other) && other.f(this);
57     }
58     public boolean f(B other) { return false; }
59     public boolean f(C other) { return other.h > h; }
60     @Override
61     public String d() { return super.d() + "(" + h + ")"; }
62 }
63
64 class Deroulement
65 {
66     public final static int NB = 1;
67     public final static int NC = 2;
68
69     public static void a(List<A> l, A a) { l.add(a); }
70
71     public static void main(String[] args) {
72         List<A> list = new ArrayList<>();
73         for (int i = 0; i < NB; ++i) {
74             a(list, new B());
75         }
76         for (int i = 0; i < NC; ++i) {
77             a(list, new C(i));
78         }
79         for (A a1 : list) {
80             for (A a2 : list) {
81                 System.out.print(a1.d());
82                 System.out.println(": " + a2.d() + " -> " + a1.e(a2));
83             }
84         }
85     }
86 }
87 }
```