

INTRODUCTION A LA PROGRAMMATION

Semester 1 Exam

Instructions :

- You have one hour and fourty five minutes to do this exam (8:15 - 10:00).
- The maximum number of points is 110 (about 20 of which is optional).
- All the documentation is allowed during the exam.
- Be sure to **process only one exercise per sheet**, and **write your SCIPER number** on each of the sheets. A sheet without identification will not be corrected.
- The exam consists of 4 exercises. These exercises are independent from each other.
- The exercises are not all of the same difficulty. Start with those that involve concepts you best have knowledge of.

CONTINUE TO THE NEXT PAGE

Exercice 1: Programming [15 points]

In your first mini-project, a method called

```
public static int[] [] shrink(int[] [] image, int[] seam)
```

has been provided to you to remove a given **seam** (path of pixels) from a given image. Write the code for a method called

```
public static int[] [] enlarge(int[] [] image, int[] seam)
```

that duplicates the given **seam** in a given image. The new **seam** will be juxtaposed (i.e., put side by side) after the given seam with the same pixel values. The final image will have an additional column of pixels. As a reminder, the following convention is used: **seam[i]** gives the column of the pixel on the *i*'th line of the image.

Assume that the provided parameters will have correct values.

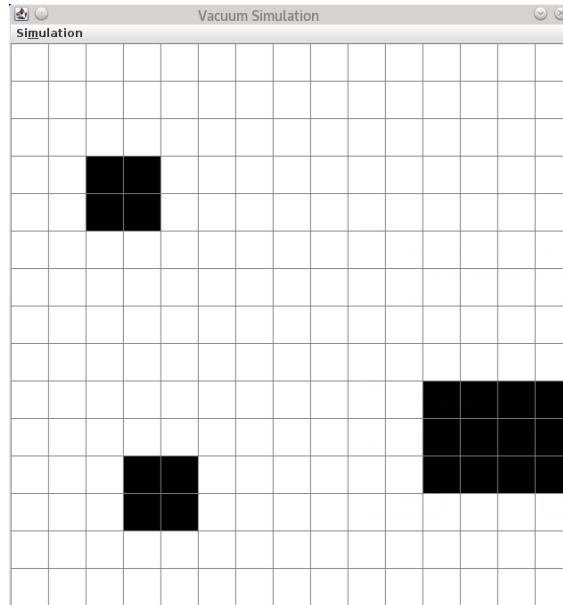
Suite au verso ➞

Exercise 2: Object Oriented Design [42 points + 8 bonus]

It is important to read all the instructions before starting to answer.

The architecture that was given to you for implementing your second mini-project is provided in the appendix.

You are asked to extend it to simulate the navigation of automatic vacuum cleaners in a room that looks like this:



where it is assumed that:

- The room is partitioned to floor tiles (i.e., the room is gridded; each square in white is a tile).
- All the tiles have the same surface size.
- Some of the tiles (shown in black) are inaccessible because they are covered by furniture.
- The room may contain a variable number of vacuum cleaners.
- They move randomly tile by tile.
- A vacuum cleaner has an identifier (an integer) and an intake capacity (also an integer) given at construction. It is smaller than a tile and it sucks the tile it is over. The tile then becomes clean.
- Some tiles contain small forgotten objects that risk to be sucked up by the vacuum cleaners.
- A small object can be sucked up by a vacuum cleaner if this vacuum cleaner has not lost more than 50% of its intake capacity. Otherwise, the vacuum cleaner moves on the tile without cleaning it and the tile is left dirty.
- A vacuum cleaner loses 3% of its intake capacity each time it cleans a dirty tile.
- A tile becomes clean once sucked and stays clean until the end of the simulation.
- It must be possible to place an object or a vacuum cleaner at a given position.
- The vacuum cleaner has no knowledge of the room it is cleaning. It is not smart enough to bypass already cleaned tiles (if it moves on a clean tile, it does not clean it and loses no intake capacity).

The simulation must stop when all the open tiles (i.e., those not occupied with furniture) are clean or that all vacuum cleaners have less than 10% of their intake capacities.

We then wish to know the number of tiles that remain dirty and the positions of the objects that are not sucked by the vacuum cleaners.

The simulation can be restarted. In that case, the vacuum cleaners will resume with the same position and the same capacities they had from the previous simulation. Already sucked objects will not reappear. All the tiles not occupied by furniture will again be considered as dirty.

For simplicity, we assume that a small object does not overlap two tiles and collisions between vacuum cleaners are neglected.

All classes used in the project, such as `Vector2D`, are available at your disposal. The API of the classes `World` and `Animal` is annexed as a reminder. You will consider these classes as already coded and available.

Part 1: Design (36 points)

You are required:

1. To clearly state the members (attributes and method headers) of all the classes necessary for the implementation of the physical world. The role of each class or member must be explained by means of a comment (if not trivial to understand).
2. To write (in French) which class will be in charge of the simulation, how it will fit into the existing architecture and what its role is.

For the first two points, you can answer in Java pseudo-code or using class diagrams. The body of the methods is not required.

You can make analogies with the classes in your project to explain the role of each class added.

Design constraints

1. The constructors should not be forgotten as well as possible useful constants.
2. You should pay special attention to the access rights of the attributes and methods and must clearly indicate what is abstract, final or static in your design.
3. Methods giving the result of the simulation (number of remaining dirty tiles and the positions of the objects not sucked) must be coded in the physical world.

Part 2: Programming (6 points)

Write the code for the method that **resets** the physical world.

Part 3: Design (bonus, 8 points)

The `Daedalus` class offers to `DaedalusSimulation` the list of predators and the list of preys of the physical world through getters (`getPredators()` and `getPreys()`).

1. What would be needed to change in the code of `DaedalusSimulation` if there were a getter `getAnimals()` returning a list containing both predators and preys? Give an explanation in French.
2. If this change were to be made without doing a type test, what impact would that have had on the hierarchy of animals (give a brief explanation in French of what must be changed)?

Suite au verso ➞

Exercise 3: Concepts [35 points]

Answer the following questions clearly and briefly:

1. [6 points] Consider the following code:

```
0. class A {
1.     protected int a1 = 1;
2.     protected int a2 = 1;
3.     public A(int a1, int a2) {
4.         this.a1=a1;
5.         this.a2=a2;
6.     }
7. }

8. class B extends A {
9.     private int a1 = 1;
10.    private int a2 = 1;
11.    public B(int a1, int a2) {
12.        super(1,1);
13.        this.a1 = a1;
14.        this.a2 = a2;
15.    }
16. }
17. }
```

- (a) How would you initialize by default the values of **a1** and **a2** in **A** to 1 by a default constructor that has only one statement? (Provide the code)
- (b) Is calling **super(1,1)** on line 13 useful since we initialize **a1** and **a2** in the next two instructions?
- (c) How can we write a method in **B** that prints on the console the values of all the attributes of an instance of **B**? What would this method print if it is invoked on an instance of **B** that was created using **new B(2,3)**?

2. [4 points] Given the following program:

```
class T {
    public static int THRESHOLD = 5;

    public static boolean small(int number) {
        return number < THRESHOLD;
    }

    public static boolean small(int number, int threshold) {
        return number < threshold;
    }
}
```

-
- (a) Is the second `small` method an overload of the first one. Justify briefly;
- (b) Propose the header of a static method `small` for `T`, that will not be an overload of `boolean small(int number)`.

3. [6 points] Consider the following piece of program:

```
1. class E extends Exception {  
2. }  
  
3. class A {  
4.     private int a;  
5.     public A(int a) throws E {  
6.         if (a < 0) {  
7.             throw new E();  
8.         }  
9.         this.a = a;  
10.    }  
11. }  
12. class B extends A {  
13.     private int b;  
14.     public B(int a, int b) {  
15.         super(a);  
16.         this.b = b;  
17.     }  
18. }
```

Compiling this program generates the error message

`unreported exception E; must be caught or declared to be thrown.`

- (a) Explain why this happens.
- (b) How can we make this program compile? Indicate what code to add and where.
- (c) What should the class `E` contain at minimum in terms of good practice.

Suite au verso ➞

4. [5 points] Consider the following code:

```
1. interface I {  
2.     default void m() {  
3.         System.out.println("I::m");  
4.     }  
5. }  
  
6. class A implements I {  
7.     public void m() {  
8.         I.m();  
9.     }  
10. }
```

- (a) Why will it not compile?
- (b) Propose a correction that only modifies the class A.
- (c) Is it possible to declare the method `m` of the interface as `protected`?
- (d) Is it possible to declare the method `m` of A as `protected`?

-
5. [5 points] What will the following program print when executing the line `A a = new B(2,3):`

```
abstract class A {
    protected int a;
    public A(int a) {
        this.a = a;
        System.out.println(this);
    }
}
class B extends A {
    private int b;
    public B(int a, int b) {
        super(a);
        this.b = b;
    }
    public String toString() {
        return "L'objet B vaut : " + a + " " + b;
    }
}
```

Briefly justify.

This program does something in the constructor of **A** that is not recommended. What is it? Explain briefly.

Suite au verso ➞

6. [6 points] What will the following program output:

```
1. class passref
2. {
3.     public static void main (String[] args) {
4.         String s1 = "est-ce ainsi";
5.         String s2 = "cent hordes sauvages";
6.         String[] s3 = {s2};
7.
8.         f(s1, " meurent ?");
9.         g(s3, " vivent ?");
10.        System.out.println(s1);
11.        System.out.println(s3[0]);
12.    }
13.
14.    private static void f(String s1, String s2)
15.    {
16.        s1 = s1.substring(4, 6);
17.        s1 = s1.replace("e ", "val");
18.        s1 = "que les " + s1;
19.        s1 += s2;
20.    }
21.
22.    private static void g(String[] s1, String s2) {
23.        s1[0] = s1[0].substring(5, 11);
24.        s1[0] = s1[0].replace("rd", "mm");
25.        s1[0] = "que les " + s1[0];
26.        s1[0] += s2;
27.    }
28. }
```

Briefly justify.

Reminders:

- `String.replace(String s1, String s2)` replaces all the occurrences of `s1` by `s2` in `this`;
 - `String.substring(int beginIndex, int endIndex)` extracts the substring that starts at the index `beginIndex` (included) and terminates at `endIndex` (excluded).
7. [3 points] Name one important rationale for the introduction of methods with default definition in interfaces.

CONTINUE TO THE NEXT PAGE

Exercice 4: Program Analysis [18 points]

The owner of a small car park has programmed an application to calculate himself a report on the vehicles that are parked there.

The constraints he wants to take into account are as follows:

- The car park has space for 10 cars and 2 motorcycles.
- A car place costs 2 francs a day, a motorcycle place 1 franc.
- A vehicle parked for more than n days is entitled to a discount (n and the amount of discount are specific to the type of vehicle).

If all the places dedicated to motorcycles are taken, then an additional motorcycle can be parked on a place designated for cars. In such a case, the price of the occupied space remains same (i.e., the rider pays the price as if he parked a car, if he occupies a car place with a motorcycle).

Here is the program the owner produced:

```
0. import java.util.ArrayList;
1. import java.util.List;
2. public class ParkingCorrige {
3.     public static void main(String[] args) {
4.         Parking parking = new Parking();
5.         parking.parquerMoto(new Motocycle(), 0);
6.         parking.parquerMoto(new Motocycle(), 1);
7.         //pas de places moto vide
8.         parking.parquerVoiture(new Voiture(), 0);
9.         parking.parquerVoiture(new Motocycle(), 1);
10.
11.         System.out.println(parking.calculerPrixTotal(4));
12.     }
13. }
```

Suite au verso ➞

```
14. class Parking {
15.     private Voiture[] placesVoiture = new Voiture[10];
16.     private Motorcycle[] placesMoto = new Motorcycle[2];

17.     public void parquerVoiture(Voiture v, int i) {
18.         if(i >= 0 && i < 10) {
19.             placesVoiture[i] = v;
20.         }
21.     }

22.     public void parquerMoto(Motorcycle m, int i) {
23.         if(i >= 0 && i < 2) {
24.             placesMoto[i] = m;
25.         }
26.     }

27.     private double calculerPrix(Voiture[] vehicules, int jourActuel) {
28.         double somme = 0.0;
29.         for (Voiture v : vehicules) {
30.             if (v != null)
31.                 somme += v.calculerPrix(jourActuel)
32.                     - v.reduction(jourActuel);
33.         }
34.         return somme;
35.     }

36.     public double calculerPrixTotal(int jourActuel) {
37.         double somme = calculerPrix(placesVoiture, jourActuel)
38.             + calculerPrix(placesMoto, jourActuel);
39.         return somme;
40.     }
41. }
```

```
42. class Voiture {
43.     int jourDebut;
44.     public Voiture(int j) {
45.         jourDebut = j;
46.     }

47.     public double calculerPrix(int jourActuel) {
48.         return (jourActuel - jourDebut)* 2;
49.     }

50.     public double reduction(int jourActuel) {
51.         if ((jourActuel - jourDebut) > 15)
52.             return 1.5;
53.         return 0;
54.     }
55. }

56. class Motocycle extends Voiture {

57.     public Motocycle(int j) {
58.         super(j);
59.     }

60.     public double calculerPrix(int jourActuel) {
61.         return super.calculerPrix(jourActuel)/2.;
62.     }

63.     public double reduction(int jourActuel) {
64.         if ((jourActuel - jourDebut) > 20)
65.             return 1.0;
66.         return 0;
67.     }
68. }
```

Questions

1. Explain why the total price will not be calculated correctly.
2. What criticisms can you make regarding the way methods `parquerVoiture` and `parquerMoto` are coded (lines 17 and 22)?
3. Propose an alternative design that fixes these problems. Your proposal can be made in French and/or by using pseudo-code or class diagrams.