ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communication
Cours Introduction à la Programmation, (SIN/SSC)
Sam J.

INTRODUCTION A LA PROGRAMMATION

Examen semestriel

Instructions :

Instructions :

- You have two hours to do this exam (10h00 - 12h00).
- Maximum 85 points (+ 20 bonus, free to choose).
- 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**
  - Exercice 1 : **56** points.
  - Exercice 2 : **34** points.
  - Exercice 3 : **15** points.

suite au verso ☞

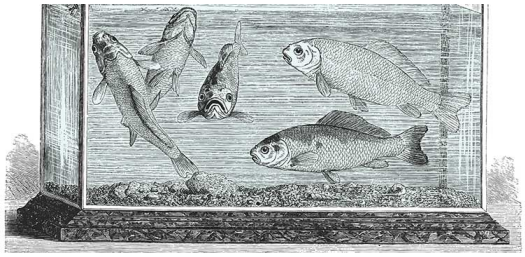
Exercice 1 : Conception OO [56 points]

It is important to read each question of this exercise in its entirety before answering it. Also, take the time to read the imposed constraints and to consult the code provided in the appendix before starting the exercise. This exercise contains 4 questions. Questions 1 and 2 are independent of question 3.

This exercise focuses on simulating, in a very simplified way, the reproduction of goldfish in an aquarium.

Goldfish in an aquarium that :

- are distinguished as males and females ;
- feed and reproduce under certain conditions.



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

1. When asked to write the body of a class, do so in Java syntax and include attributes and methods but without the bodies (except for those explicitly requested).
2. A draft `Aquarium` class is provided which will need to be completed. You will assume that a main program exists, that it creates an `Aquarium` object, adds fish and food sources to it, and calls their `update` method in a loop. This method allows the involved entities to evolve over time (using `update` methods, as dictated by the `Updatable` interface).
3. Classes must contain the necessary members to implement all the desired functionalities, **whether they are explicitly requested or suggested by the specifications of the statement** (especially by the implementation of the `update` methods) ;
4. For methods that are not trivial, **briefly comment on the meaning of the return type and what they do in broad terms, as well as the functionalities of other classes they use**.
5. **Do not neglect constructors, nor access rights**.
6. The `protected` right must not be used for attributes.
7. Your design must be well modularized.
8. Propose getters/setters only when they are necessary for the implementation of the simulation described.
9. It is not necessary to write import directives.

suite ☞

Question 1 : Aquarium (11.5 points)

An aquarium is characterized by its *volume* (a `Vec3d` giving its length, width, and height), its *filling percentage*, its *PH*, and the *temperature* of its water. These data are initialized at construction through parameters. If an aquarium is constructed using only a volume, the other attributes will have specific default values : 85% for the filling level, 7 for the PH, and 22 for the temperature. These default values correspond to the ideal living conditions for fish.

The characteristics of the aquarium evolve over time (water evaporates, PH changes, etc.). **The precise modalities of these changes are not of interest to us but are taken care of by the `update` method (provided in the appendix).**

The following additional functionalities are desired for the aquarium at this stage :

1. access its PH, temperature, and filling level,
2. return the aquarium to ideal living conditions (temperature, PH, and filling level) ;
3. increase or decrease the filling level (this function must return the filling level resulting from this operation) ;
4. and transfer its fish to another aquarium.

Provide the Java code to add to the `Aquarium` class to meet this specification. **Provide only the method headers and attributes**.

Question 3 : Nutrients (21 points)

It is now about completing the nutrients' conception (`Food`). A nutrient for fish can be either one of their *eggs* or a *flake*.

— Each type of nutrient has an *energy level* and a *position* (`Vec3d`). These data are provided at construction.

— Nutrients are distinguished from each other in their way of being consumed (i.e., losing energy) : for example, an *egg* if eaten is gobbled up entirely at once (energy reduced to zero), a flake is nibbled on little by little. The precise consumption modalities are not of interest to us.

— An *egg* can be fertilized and it must be possible to inquire whether it is. It is created unfertilized. At the moment it is fertilized, its incubation begins. The incubation time is a constant common to all eggs. A new fish will replace the egg once this time has elapsed. We are not interested in the exact details of how the new fish is created. You will consider that an egg has access to its aquarium (as an attribute passed at construction). **However, it must only be able to access the addFish and removeFood methods**.

Over time, nutrients move randomly in the water. The exact modalities of these movements are not of interest.

— **Write the classes and interfaces that can be used to represent the nutrients model. You can use any values you like for the constants involved. Comment on the role of the `update` methods when they are redefined**.

— **Describe any additions to be made to the `Aquarium class`**.

suite au verso ☞

Question 3 : Fish (13 points)

A *fish* is endowed with a *position* (`Vec3d`) given at construction. At each time step, it moves around the aquarium (the exact modalities are not of interest to us). Fish are categorized into *males* and *females*. As this simulation is mainly interested in reproduction, these categories represent types (classes). Female fish randomly lay eggs over time.

If a fish collides with a nutrient, it *interacts* with it. The modalities of interaction are not of interest to us for the moment but depend on its type. You will consider that it is the aquarium that tests collisions and calls up the interaction methods.

Moreover, fish are aware of the aquarium in which they live (attribute transmitted at construction). However, they should only have access to its PH, and temperature. They must also be able to lay in it a given number of eggs.

— **Write the classes and interfaces allowing to represent the above model. You can use the values of your choice for the constants involved.**

— **Describe any additions to be made to the `Aquarium` class.**

Question 4 : Interactions [10.5 points]

The modalities of a fish interaction with a nutrient depend on its type :

— if it is a female, she consumes it.

— if itis s a male : he consumes it if it is a flake and fertilizes it if it is an egg.

We wish that the method of interaction takes place without type testing, by using the mechanism of method overloading. **Indicate the methods to add in the existing hierarchies to code the interactions. You will also provide the bodies of the methods**

Exercice 2 : Concepts de base [34 points]

Answer the following questions clearly and concisely :

1. **[5 points]** The code is as follows :

```
1  class A {
2      private String a;
3      public A() { init("better"); }
4      private void init(String a) { this.a = a; }
5      public String toString() { return " a " + a + " " ; }
6  }
7  class B extends A {
8      private String b = "day";
9      public B() { }
10     public B(String b) { this(); this.b += " ?"; }
11     public String toString() { return super.toString() + b; }
12 }
13 class Pgme {
14     public static  void main(String[] args) {
15         B b = new B("");
16         System.out.println(b);
17     }
18 }
```

   (a) What does it display ?

   (b) Can we delete the empty default constructor on line 9 ?

   (c) Can we set the method `toString` as protected in `A` ?

   (d) Can we replace `this()` with `this("day")` on line 10 ?

   (e) Can we replace `this()` with `super("day")` on line 10 ?

   Briefly justify each of your answers.

2. **[6 points]** Consider the following class declarations :

```
1
2  class A {
3      void m1(){}
4  }
```

(1)

```
1
2  class B extends A {
3      void m2(){}
4  }
```

(2)

```
1
2  class C {
3      void m3(){}
4  }
```

(3)

   For each of these classes, add the missing access rights and package declarations so that the following constraint is satisfied : `m3` is accessible in `A` but not in `B`, `m1` is accessible in `B` and `C`, `m2` is accessible nowhere but in `B`.

3. **[12.5 points]** Consider the following two statements :

```
1 interface I {
2     default void m() {}
3 }
4 abstract class C {
5     public abstract void m();
6 }
```

Indicate for each of the following statements, each taken separately, whether they can be added to the program without causing a compiler error :

(a) `class D extends C {}`

(b) `class D extends C extends I {}`

(c) `class D  implements I {}`

(d) `class D extends I {}`

(e) `interface F extends I {}`

(f) `interface F implements I {}`

(g) `abstract class D extends C { enum E { ONE, TWO } }`

(h) `abstract class D extends C {}`

(i) `class D extends C implements I { public void m() {} }`

(j) `class D implements I { public void m() {I.super.m();} }`

(an invalid answer causes a penalty of -0.25, but the mark cannot be below zero for the question)

suite ☞

4. **[10.5 points]** The code is as follows :

```
1  class Collector extends ArrayList<Integer> {
2      static int k = 0;
3      boolean add (int i) {
4          ++k;
5          if (i%2 != 0) {
6              return super.add(i);
7          }
8          return false;
9      }
10     public String print() {
11         String res = "";
12         for (Integer i : this)  res+=i + " ";
13         return res;
14     }
15 }
16 class Garbage extends Collector {
17     boolean add (int i) {
18         super.add(i);
19         if (i%4 == 0) {
20             return super.add( i+1 );
21         }
22         return false;
23     }
24     boolean add (int i, boolean b) {  add(i); return b; }
25 }
26 class Pgme {
27     public static void main(String[] args) {
28         Collector tray = new Garbage();
29         tray.add(2); tray.add(7); tray.add(4);
30         System.out.println(tray.print());
31         System.out.println(Garbage.k);
32     }
33 }
```

(a) what does it display ?

(b) can we put the keyword `abstract` at the beginning of line 1 ?

(c) can we add a `String add(int i)` method to the `Garbage` class ?

(d) can we add the line `System.out.println(tray.k);` at the end of `main` ?

(e) give an example of method («overloading») in the above code.

(f) give an example of method («overriding») in the above code.

(g) why can't we add `tray.add(2, true);` to the end of `main`. How should this instruction be changed so that it can be added without causing a compilation error.

Briefly justify your answers to (b), (c), (d) and (g) questions.

Exercice 3 : Déroulement de programme [15 points]

The following program runs without errors (you'll assume the import directives are present). What does it display ? Briefly explain how it works, **without rephrasing the code**.

```
1  interface PowerProduct extends BasicProduct {
2      double POWER = 400;
3      default double power(double rpm) { return rpm; }
4      default double power() { return 0; }
5  }
6  interface  BasicProduct {
7      default double weight() { return 0; }
8  }
9  abstract class Product implements BasicProduct {
10     public double weight;
11     public Product (double w) { weight = w; }
12     public double weight() { return weight; }
13 }
14 class GasEngine extends Product implements PowerProduct {
15     private double maxPower = PowerProduct.POWER*10;
16     private double rpm;
17     public GasEngine(double rpm) { this(500,300, rpm); }
18     public GasEngine(double w, double p, double r) {
19         super(w); maxPower = p; rpm = r;
20     }
21     public String toString() {
22        return "GasEngine(" + weight() + "," + power() + ")";
23    }
24     public double power() { return power(rpm); }
25     public final double power(double rpm) {
26         return maxPower * PowerProduct.super.power(rpm) /
              PowerProduct.POWER;
27     }
28 }
29 class ElectricEngine extends Product implements PowerProduct {
30     private double maxPower;
31     public ElectricEngine() {
32         this(100, 200); maxPower *=2;
33     }
34     public ElectricEngine(double w, double p) { super(w); maxPower=p;
           }
35     public String toString() {
36         return "ElectricEngine(" + weight() + "," + power() + ")";
37     }
38     public final double power() { return maxPower; }
39 }
40 class Car extends Product implements PowerProduct {
41     private List <PowerProduct> engines = new ArrayList<>();
42     private Chassis chassis = new Chassis();
43     private String id;
44     private double rpm; // SUITE  -->>
45
```

```
46      public Car(String id, double rpm) {
47           super(0); this.id = id; this.rpm=rpm;
48      }
49      public String getId() { return id; }
50      public String toString() {
51          String str =  "Car(" + id + ")\n";
52          for (PowerProduct e : engines) str += e.toString() + "\n";
53          str+= chassis.toString();
54          str+= "performance : " + performance();
55          return str;
56      }
57      public double performance() { return power(rpm) / weight(); }
58      public final double weight() {
59          double s = chassis.weight();
60          for (PowerProduct e : engines) s += e.weight();
61          return s;
62      }
63      public double power(double rpm) {
64          double s = 0.0;
65          for (PowerProduct e : engines) s += e.power(rpm);
66          return s;
67      }
68      public void add(PowerProduct p) { engines.add(p); }
69      public void addChassis() {  chassis= new Chassis(); }
70      public void addChassis(double w) { chassis= new Chassis(w); }
71
72      private class Chassis  extends Product implements BasicProduct {
73        private Chassis(double w) { super(w); }
74          private Chassis() { this(1000); }
75          public double weight() { return weight; }
76          public String toString() {
77              return "Chassis(" + weight() + ")\n";
78          }
79      }
80  }
81  class Deroulement {
82      public static void  main(String[] args) {
83          double rpm = PowerProduct.POWER*20;
84          Car a = new Car("Alice's car", rpm);
85          a.add(new GasEngine(rpm)); a.addChassis();
86          System.out.println(a);  System.out.println();
87          rpm = PowerProduct.POWER*10;
88          Car b = new Car("Bob's car", rpm);
89          b.add(new GasEngine(400, 200, rpm) ); b.addChassis(1500);
90          b.add(new ElectricEngine());
91          System.out.println(b);
92          Car faster = (a.performance() > b.performance()) ? a : b;
93          System.out.println(faster.getId() +  " is faster\n");
94      }
95  }
```