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 :

- Vous disposez de deux heures pour faire cet examen (10h00 - 12h00).
- Nombre maximum de 85 points (+ 20 en bonus, à choisir librement).
- Attention : il y a aussi des énoncés sur le verso.
- Vous devez **écrire à l'encre noire ou bleu foncée**, pas de crayon ni d'autre couleur. N'utilisez **pas non plus de stylo effaçable** (perte de l'information à la chaleur).
- Toute documentation papier est autorisée ; En revanche, vous ne pouvez pas utiliser d'ordinateur personnel, ni de téléphone portable, ni aucun autre matériel électronique.
- Répondez sur les feuilles qui vous sont distribuées et **aux endroits prévus**. **Ne répondez pas sur l'énoncé**.
- Ne joignez aucune feuilles supplémentaires ; **seules les feuilles de réponses distribuées seront corrigées**.
- Vous pouvez répondre aux questions en français ou en anglais.
- L'examen compte 3 exercices indépendants. **Vous pouvez commencer par celui que vous souhaitez**

  - Exercice 1 : **56** points.
  - Exercice 2 : **34** points.
  - Exercice 3 : **15** points.

suite au verso ☞

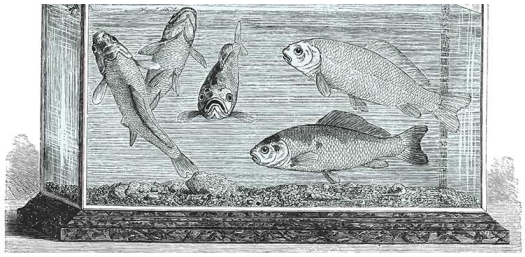
Exercice 1 : Conception OO [56 points]

Il est important de lire chaque question de cet exercice dans son intégralité avant d'y répondre. Prenez aussi le temps de lire les contraintes imposées et de consulter le code fourni en annexe avant de commencer l'exercice. Cet exercice contient 4 questions. Les questions 1 et 2 ne dépendent pas de la question 3.

On s'intéresse ici à simuler, de façon très simplifiée, la reproduction de poissons rouges dans un aquarium.

Poissons rouges dans un aquarium qui :

- se distinguent en mâles et femelles ;
- se nourissent et se reproduisent sous certaines conditions.



Contraintes de conception Lors de la résolution de cet exercice, les contraintes suivantes seront à respecter :

1. Lorsque l'on vous demande d'écrire le corps d'une classe vous le ferez en syntaxe Java et y écrirez les attributs et les méthodes mais sans les corps (sauf pour ceux explicitement demandés).
2. Une ébauche de classe `Aquarium` est fournie qu'il s'agira de compléter. Vous supposerez qu'un programme principal existe, qu'il crée un objet `Aquarium` y ajoute des poissons et des sources de nourriture et appelle en boucle leur méthode `update`. Cette méthode permet de faire évoluer les entités impliquées au cours du temps (au moyen de méthodes `update`, telles que dictées par l'interface `Updatable`).
3. Les classes devront contenir les membres nécessaires pour mettre en oeuvre toutes les fonctionnalités souhaitées, **qu'elles soient explicitement demandées ou suggérées par les spécifications de l'énoncé** (en particulier par la mise en oeuvre des méthodes `update`) ;
4. Pour les méthodes qui ne sont pas triviales, **vous noterez au moyen d'un bref commentaire la signification du type de retour et ce qu'elles font dans les grandes lignes ainsi que les fonctionnalités des autres classes qu'elles utilisent**.
5. **Ne négligez ni les constructeurs, ni les droits d'accès**.
6. Le droit `protected` ne doit pas être utilisé pour les attributs.
7. Votre conception doit être bien modularisée.
8. Vous ne proposerez de getter/setter que lorsqu'ils sont nécessaires à la mise en oeuvre de la simulation décrite.
9. Il n'est pas nécessaire d'écrire des directives d'importation.

suite ☞

Question 1 : Aquarium (11.5 points)

Un aquarium est caractérisé par son *volume* (un `Vec3d` donnant sa longueur, largeur et hauteur), son *pourcentage de remplissage*, son *PH* et la *température* de son eau. Ces données sont intialisées à la construction au moyen de paramètres. Si l'on construit un aquarium en donnant uniquement son volume, les autres attributs auront des valeurs par défaut spécifiques : 85% pour le niveau de remplissage, 7 pour la PH et 22 pour la température. Ces valeurs par défaut correspondent aux conditions de vie idéales pour les poissons.

Les caractéristiques de l'aquarium évoluent au cours du temps (l'eau s'évapore, le PH change etc.). **Les modalités précises de ces changements ne nous intéressent pas mais sont prises en charge par la méthode `update` déjà présente (voir l'annexe).**

Les fonctionnalité additionnelles suivantes sont souhaitées pour l'aquarium à ce stade :

1. accéder à son PH sa température et son niveau de remplissage,
2. ramener l'aquarium aux conditions de vie idéales (température, PH et niveau de remplissage) ;
3. augmenter ou diminuer le niveau de remplissage (cette fonction doit retourner le niveau de remplissage résultant de cette opération) ;
4. et transférer ses poissons dans un autre aquarium.

Donnez le code Java à ajouter à la classe `Aquarium` pour répondre à la spécification précédente. **Donnez uniquement les entêtes des méthodes et les attributs**.

Question 2 : Nutriments (21 points)

Il s'agit maintenant de compléter la modélisation des nutriments (`Food`). Un nutriment pour les poissons peut être soit un de leurs *oeufs*, soit un *flocon*.

- Chaque type de nutriment a un *niveau d'énergie* et une *position* (`Vec3d`). Ces données sont transmises comme paramètre à la construction.
- Les nutriments se distinguent les uns des autres dans leur façon d'être consommé (c'est à dire de perdre de l'énergie) : par exemple un *oeuf* s'il est mangé se fait gober intégralement en une fois (énergie réduite à zéro), un flocon se fait grignoter par petit bout. Les détails précis de mise en oeuvre ne nous intéressent pas.
- Un *oeuf* peut être fertilisé et il doit être possible d'interroger le fait qu'il le soit. Il est créé non fertilisé. À l'instant où il est fertilisé, son incubation commence. Le temps d'incubation est une constante commune à tous les oeufs. L'oeuf se transformera en poisson une fois ce temps écoulé. Les modalités exactes de création du nouveau poisson ne nous intéressent pas ici. Vous considérerez qu'un oeuf a accès à son aquarium (attribut transmis à la construction). **Il ne doit cependant pouvoir accéder qu'aux méthodes addFish et removeFood)** .

Au cours du temps, les nutriments se déplacent de manière aléatoire dans l'eau. Les modalités exactes de ces déplacements ne nous intéressent pas.

- **Écrivez les classes et interfaces permettant de représenter le modèle les nutriments. Vous pouvez utiliser les valeurs de votre choix pour les constantes impliquées. Commentez le rôle des méthodes `update` lorsqu'elles sont redéfinies.**
- **Décrivez les éventuels ajouts à apporter à la classe `Aquarium`.**

suite au verso ☞

Question 3 : Poissons (13 points)

Un *poisson* est doté d'une *position* (`Vec3d`) donnée à la construction. À chaque pas de temps il se déplace dans l'aquarium (les modalités exactes ne nous intéressent pas). Les poissons sont catégorisés en *mâles* et *femelles*. Comme cette simulation s'intéresse essentiellement à la reproduction, ces catégories représentent des types (classes). Les poissons femelles pondent aléatoirement des oeufs au cours du temps.

Si un poisson est en collision avec un nutriment il *interagit* avec. Les modalités d'interaction ne nous intéressent pas pour le moment mais dépendent de son type. Vous considérerez que c'est l'aquarium qui teste les collisions et appelle les méthodes d'interaction.

Par ailleurs, les poissons ont connaissance de l'aquarium dans lequel ils vivent (attribut transmis à la construction). Ils ne doivent cependant avoir accès comme informations sur ce dernier qu'à son PH et sa température. Ils doivent aussi pouvoir y pondre un nombre donné d'oeufs.

- **Écrivez les classes et interfaces permettant de représenter les poissons. Vous pouvez utiliser les valeurs de votre choix pour les constantes impliquées.**
- **Indiquez les éventuels ajouts à apporter à la classe `Aquarium` en relation avec la spécification ci-dessus.**

Question 4 : Interactions [10.5 points]

Les modalités d'interaction d'un poisson avec un nutriment dépendent de son type :

- si c'est une femelle elle le consomme.
- si c'est un mâle : il en consomme si c'est un flocon et le fertilise si c'est un oeuf.

On souhaite que la méthode d'interaction se fasse sans tests de type en utilisant le mécanisme de la surcharge de méthodes. **Indiquez les méthodes à ajouter dans les hiérarchie existantes pour coder les interactions. Vous donnerez également le corps des méthodes**

Exercice 2 : Concepts de base [34 points]

Répondez clairement et succinctement aux questions suivantes :

1. **[5 points]** Soit le code suivant :

```
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) Qu'affiche t-il ?

   (b) Peut-on supprimer le constructeur par défaut vide de la ligne 9 ?

   (c) Peut-on marquer la méthode `toString` comme protected dans `A` ?

   (d) Peut-on remplacer `this()` par `this("day")` à la ligne 10 ?

   (e) Peut-on remplacer `this()` par `super("day")` à la ligne 10 ?

   Justifiez brièvement chacune de vos réponses.

2. **[6 points]** Soient les déclarations de classes suivantes :

```
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)

   Pour chacune de ces classes, ajoutez les droits d'accès manquant ainsi que des déclarations de paquetages pour que la contrainte suivante soit satisfaite : `m3` est accessible dans `A` mais pas dans `B`, `m1` est accessible dans `B` et `C`, `m2` n'est accessible nulle part ailleurs que dans `B`. Le nom des paquetages peut être librement choisi.

3. **[12.5 points]** Soient les deux déclarations suivantes :

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

Indiquez pour chacune des déclarations suivantes, chacune prise séparément, si elles peuvent être ajoutée au programme sans causer de faute de compilation :

(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();} }`

(une réponse invalide cause une pénalité de -0.25, mais la note ne peut être en dessous de zéro pour la question)

suite ☞

4. **[10.5 points]** Soit le code suivant :

```
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) qu'affiche t-il ?

(b) peut-on mettre le mot clé `abstract` au début de la ligne 1 ?

(c) peut-on ajouter une méthode `String add(int i)` à la classe `Garbage` ?

(d) peut on ajouter la ligne `System.out.println(tray.k);` à la fin de `main` ?

(e) donnez un exemple de surcharge («overloading») de méthode dans le code ci-dessus.

(f) donnez un exemple de redéfinition («overriding») de méthode dans le code ci-dessus.

(g) pourquoi ne peut-on pas ajouter `tray.add(2, true);` à la fin de `main`. Comment faudrait-il changer cette instruction pour qu'il soit possible de l'ajouter sans causer d'erreur de compilation ?

Justifiez brièvement vos réponses aux points (b), (c), (d) et (g).

Exercice 3 : Déroulement de programme [15 points]

Le programme suivant s'exécute sans erreurs (vous supposerez les directives d'importation présentes). Qu'affiche t-il ? expliquez succinctement son déroulement **sans paraphraser le 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  }
```