

# INTRODUCTION A LA PROGRAMMATION

## Test Semestre I

### Instructions :

- Vous disposez de une heure quarante cinq minutes pour faire cet examen (8h15 - 10h);
- Nombre maximum de points 125 points (dont environ 30 facultatifs);
- Toute documentation est autorisée;
- Veuillez à **ne traiter *qu'un* exercice par feuille**, et à **indiquer votre numéro sciper** sur *chacune* des feuilles. Une feuille sans identification ne sera pas corrigée;
- L'examen compte 4 exercices. Ces exercices sont indépendants.
- Les exercices ne sont pas tous de même difficulté. Commencez par ceux dont vous maîtrisez le mieux les concepts impliqués.

SUJET À LA PAGE SUIVANTE

---

## Exercice 1 : Concepts [33 points]

Répondez clairement et succinctement aux questions suivantes :

1. [10 points] Soit le code suivant :

```
1. class X
2. {
3.     public X m() {
4.         return new X(this);
5.     }
6.     public X() {}
7.     public X(X o) {}
8. }


9. class Y extends X
10. {
11.     public Y m() {
12.         return new Y(this);
13.     }

14.     public Y() {}

15.     public Y(Y o) {super(o);}
16. }

17. class Test
18. {
19.     public static void main(String[] args) {
20.         Y y1 = new Y();
21.         Y y2 = new Y(y1);
22.         System.out.println( y1.equals(y2));
23.     }
24. }
```

- (a) Qu'affiche t-il ? Justifiez brièvement.
- (b) Compilerait-il toujours si l'on supprimait l'instruction `super(o);` à la ligne 15 ? Justifiez brièvement.
- (c) La méthode `m` de `Y` est-elle une redéfinition (overriding) de la méthode `m` de `X` ? Justifiez.
- (d) Compilerait-il toujours si l'on ajoutais la méthode suivante dans la classe `Y` :
- ```
public Object m() {
    return new Y(this);
}
```
- Justifiez brièvement.
- (e) Peut-on compter sur un constructeur par défaut si l'on supprime le constructeur de copie de la classe `Y` ? Justifiez brièvement.

Suite au verso 

---

2. [3 points] Soit le programme suivant :

```
1. class X
2. {
3.     private String x = "X";
4. }

5. class Y extends X
6. {
7.     public Y() {
8.         super("X");
9.     }

10.    public Y(String s) {
11.        this(s);
12.    }
13. }
```

Indiquez la ou les faute(s) empêchant ce programme de compiler.

3. [4 points] Soit la portion de programme suivante :

```
class A {
    private int a1;
    private int a2;
    private int a3;

    public A(int val1, int val2, int val3) {
        a1 = val1;
        a2 = val2;
        a3 = val3;
    }
}
```

Comment proposez-vous de le compléter, sans dupliquer de lignes de codes, afin qu'il soit possible de construire des instances de A comme suit :

```
A a1 = new A(1,2); // l'attribut a3 prend 10 comme valeur par défaut
A a2 = new A(1);   // les attributs a2 et a3 prennent 10 comme valeur par défaut
```

- 
4. [3 points] Soit le code suivant :

```
interface Dessinable
{
    void draw();
}

class Ballon implements Dessinable
{
    void draw() {
        // on dessine le ballon ici
    }
}
```

- (a) Pourquoi ne compile t-il pas ?
- (b) Comment proposez-vous de le corriger ?
5. [3 points] Un attribut statique d'une classe doit-il être initialisé par l'un des constructeurs de la classe ? Justifiez brièvement votre réponse.
6. [3 points] Pourquoi les `RuntimeException` ne sont-elles pas soumises à la règles «déclarer-ou-traiter» ?

Suite au verso ➞

---

7. [7 points] Soit le code suivant :

```
class Disque
{
    private int rayon;

    public Disque(int unRayon) {
        rayon = unRayon;
    }

    boolean equals(Disque o) {
        return rayon == o.rayon;
    }
}

class equals
{
    public static void main(String[] args) {
        Disque d1 = new Disque(3);
        Disque d2 = new Disque(1+2);
        System.out.println(d1.equals(d2));
        System.out.println(d1.equals("Un joli disque de rayon 3"));
    }
}
```

- (a) Pourquoi la méthode `equals` de la classe `Disque` peut-elle accéder au rayon de l'objet `o` sans passer par un getter public ?
- (b) La méthode `main` compile t-elle ?
  - i. si oui, dites pourquoi et ce qu'elle affiche.
  - ii. si non, dites pourquoi et comment la corriger.

SUITE DU SUJET À LA PAGE SUIVANTE

---

## Exercice 2 : Conception OO et programmation [50 points + 15 bonus]

Il est important de lire l'énoncé complètement avant de commencer à répondre.

Il s'agit dans cet exercice de simuler des bactéries évoluant sur un substrat.

**Le substrat** Un substrat est caractérisé par une *température* que l'on supposera constante (une fois qu'une valeur lui est attribuée cette valeur ne changera pas par la suite). Il contient un *ensemble de bactéries* ainsi qu'un *ensemble de sources de nourriture* pour ces bactéries. Il occupe une forme circulaire (assiette de Petri) caractérisée par un *rayon*, également constant.

Les fonctionnalités souhaitées relatives au substrat sont :

- pouvoir construire un substrat en lui affectant une température et un rayon ;
- pouvoir y ajouter une bactérie donnée ;
- pouvoir y ajouter une source de nourriture donnée ;
- faire évoluer la population de bactéries au cours du temps.

**Les sources de nourriture** Une source de nourriture occupe une *position* sur le substrat et est caractérisée par une *quantité* (un entier pour simplifier).

**Les bactéries** Chaque bactérie :

- occupe une *position* sur le substrat,
- a un *niveau d'énergie* (un entier) ainsi qu'une vitesse de déplacement;
- peut se déplacer, se diviser et mourir;
- évolue au cours du temps;

Vous supposerez qu'il existe des classes prédéfinies **Position** et **Speed** utilisable pour modéliser la position et la vitesse.

Une bactérie meurt lorsque son niveau d'énergie est à zéro.

Deux types de bactéries seront cultivées : des *bactéries à flagelles* et des *bactéries à tentacule*. Une bactérie à tentacule est caractérisée par la longueur de sa tentacule.

Les deux types de bactéries se distinguent entre elle par le mode de déplacement : les premières se déplacent à vitesse constante en changeant aléatoirement de direction, les secondes se déplacent au moyen d'une tentacule.

**La mise en oeuvre exacte des modalités de déplacement et de division ne nous intéressent pas ici.**

L'algorithme simulant l'évolution des bactéries est le même pour tout type de bactéries. Il met en oeuvre les étapes suivantes à chaque pas de simulation:

1. la bactérie se déplace. Si elle atteint les limites du substrat elle inverse sa direction;
2. si une source de nourriture est suffisamment proche, elle en consomme ce qui augmente son niveau d'énergie de la quantité de nourriture consommée. Elle en consomme une quantité fixe à chaque fois et cette quantité est caractéristique de la bactérie (chaque bactérie aura une quantité fixée à sa naissance) ;
3. elle contrôle son niveau d'énergie et décide de se diviser en fonction de ce niveau;
4. elle perd une certaine quantité d'énergie (fixe et identique pour tout type de bactérie).

La division est conditionnée par la température du substrat et nécessite des niveaux seuil d'énergie distincts d'un type de bactérie à l'autre. La nature de l'influence de la température sur la division ne nous intéresse pas.

Les sources de nourriture entièrement consommées et les bactéries mortes doivent disparaître du substrat.



---

Vous supposerez :

- que la méthode d'évolution des bactéries a pour entête `evolue(Substrat s)` où `Substrat` est la classe modélisant le substrat.
- que **le substrat n'offre pas de getter donnant accès à toutes ses sources de nourriture ou toutes ses bactéries** (Indication : pour se nourrir, par exemple, une bactérie doit simplement connaître la source de nourriture la plus proche de sa position dans le substrat).
- que les méthodes de déplacement sont spécifiques à chaque catégorie de bactéries et ne peuvent pas être définies concrètement pour une bactérie quelconque;

## Partie 1 : Conception (43 points)

Dessinez un hiérarchie de classes permettant de mettre en oeuvre les fonctionnalités souhaitées. Vous spécifierez dans votre diagramme les classes, les interfaces éventuelles, les attributs et les entêtes des méthodes (**sans les corps**).

### Contraintes de conception

1. Votre conception devra utiliser le polymorphisme. Elle ne doit pas nécessiter de dupliquer du code, ne doit pas comporter de getters/setters inutiles ni de tests de type.
2. Les constructeurs ne seront pas oubliés. On supposera qu'ils initialisent les attributs au moyen de valeurs passées en paramètres.
3. Vous porterez une attention particulière aux droits d'accès des attributs et des méthodes et indiquerez clairement ce qui est abstrait, final ou statique dans votre conception.

## Partie 2 : Programmation (7 points + 3 bonus)

Donnez le code de la méthode permettant de faire évoluer une bactérie.

(**Bonus** : qu'aurait-on du modifier au niveau de la conception si le substrat n'était pas donné en argument de la méthode d'évolution ?)

## Partie 3 : Conception (bonus, 12 points)

Supposons maintenant que l'on souhaite simuler des interactions entre les bactéries. Une bactérie peut donc en rencontrer une autre et potentiellement la tuer en cas de rencontre. Supposons que :

- les bactéries à flagelles ne peuvent tuer aucune bactérie;
- les bactéries à tentacule tuent les bactéries à flagelles, et seulement ces dernières, dès qu'elles les rencontrent.

Expliquez comment vous procéderiez pour implémenter la méthode

`void interact(Bacteria other)` (adaptez le nom de la classe si vous l'avez nommée autrement) permettant de gérer la rencontre d'une bactérie avec une autre **sans faire de test de type**.

Vous donnerez le code Java nécessaire en indiquant dans quelle(s) classe(s) le placer.

Suite au verso ➞

---

## Exercice 3 : Analyse de programme [20 points]

Une entreprise de développement logiciel vous demande d'évaluer son programme de gestion des ressources (humaines et matérielles). Les employés y sont catégorisés schématiquement comme suit :

```
abstract class Employe
{
    // plein de choses
    public abstract double salaire();
}

class Consultant extends Employe
{
    // plein de choses
    public double salaire();
}

class Commercial extends Employe
{
    // plein de choses
    public double salaire();
}

class Technicien extends Employe
{
    // plein de choses
    public double salaire();
}

class Developpeur extends Employe
{
    // plein de choses
    public double salaire();
}
```

### 3.1 Paiement de bonus (6 points)

Le directeur des ressources humaines vous demande d'ajouter une fonctionnalité permettant d'attribuer un bonus de fin d'année à tous les employés permanents de l'entreprise :

```
void payerBonus(Employe[] desEmployes, double bonus)
```

(parmi la liste `desEmployes`, seuls les permanents doivent recevoir un bonus)

Les employés permanents sont les `Developpeur` et les `Technicien` (mais aucune information dans la hiérarchie actuelle ne permet de le savoir).

Comment procéderiez-vous pour coder une telle fonctionnalité **sans faire de test** de type ? Vous avez le droit d'enrichir la hiérarchie de classes existante.

---

### 3.2 Partage d'employés (7 points)

En observant le constructeur de la classe principale (qui permet de recruter pleins d'employés d'un coup) :

```
class Entreprise
{
    private Employe[] staff;

    public Entreprise(Employe[] unStaff) {
        staff = unStaff;
    }

    //plein de choses
}
```

vous constatez une/des faille(s) d'encapsulation (les instances d'employés sont mutables : un employé peut changer de catégorie salariale, de lieu de résidence etc..)

Expliquez la/les faille(s) et comment la/les réparer (il n'est pas nécessaire de donner le code. Expliquez juste le principe).

Peut-il être nécessaire d'apporter des modifications au contenu de la hiérarchie de classes pour réparer cette/ces failles(s) ?

**Suite au verso** ➞

---

### 3.3 Sources de dépenses (7 points)

Le programme de gestion comporte en outre les composants suivants:

```
// le matériel acheté et utilisé par l'entreprise
abstract class Materiel
{
    public abstract double prixAchat();
}

class Ordinateur extends Materiel
{
    public double prixAchat() { ... }
}

class Licence extends Material
{
    public double prixAchat() { ... }
}

// les logiciels fabriqués et vendus par l'entreprise
class Logiciel
{
    public double prixVente();
}
```

Il existe dans le programme existant une fonctionnalité :

```
double estimationGain(Logiciel[] s, Employe[] e, Materiel[] m)
```

estimant les gains de l'entreprise lors de la vente des logiciels *s*, sachant que leur développement a nécessité de payer les employés *e* et d'acheter les ressources matérielles *m* (schématiquement, gain = prix de vente des logiciels - salaires des employés - coût du matériel lié au développement).

Le directeur de l'entreprise souhaiterait que cette fonctionnalité n'expose pas aussi explicitement ses sources de dépenses et permette d'en ajouter d'autres sans que cela ne nécessite de changements ultérieurs.

Quelle solution lui proposeriez-vous?

SUITE DU SUJET À LA PAGE SUIVANTE

---

## Exercice 4 : Exceptions [22 points]

Soit le programme suivant :

```
1. import java.util.Scanner;
2. import java.util.ArrayList;

3. class MyReader
4. {
5.     private final static Scanner clavier = new Scanner(System.in);
6.     public static int readInt() {
7.         int lu;
8.         System.out.println("Donnez un entier");
9.         lu = clavier.nextInt();
10.        return lu;
11.    }
12. }

13. class Container
14. {
15.     private ArrayList<Integer> collection;

16.     public Container() {
17.         collection = new ArrayList<Integer>();
18.     }

19.     public void read(int aSize) {
20.         for (int i = 0; i < aSize; ++i) {
21.             collection.add(1/MyReader.readInt());
22.         }
23.     }
24. }

25. class Exceptions
26. {
27.     public static void main(String[] args) {
28.         Container myContainer = new Container();
29.         myContainer.read(20);
30.     }
31. }
```

- 
1. Comment ce programme se comportera t-il si l'utilisateur saisit :
    - (a) autre chose qu'un entier ?
    - (b) un zéro ?
  2. Comment proposez-vous de le modifier, en utilisant la gestion des exceptions, pour que l'utilisateur :
    - (a) ait 10 droits à l'erreur en cas de saisie d'autre chose qu'un entier. Une fois ce droit à l'erreur dépassé, le programme doit lancer une `InputMismatchException` qui sera traitée dans le programme principal.
    - (b) ait 5 droits à l'erreur en cas de saisie de 0 pour une entrée du tableau (il a le droit de recommencer 5 fois au cas où il saisit 0 pour une des entrées du tableau). Un fois ce droit à l'erreur dépassé, le programme doit lancer une `InputMismatchException` qui sera traitée dans le programme principal.

**Important** : pour la méthode `readInt`, la lecture d'un zéro doit être considérée comme normale. `readInt` peut être utilisée dans d'autres contextes.

Le programme principal devra traiter les exceptions lancées et permettre l'affichage d'un message approprié ("Remplissage du tableau : trop de zéros !" ou "Vous avez saisi n'importe quoi trop de fois !").

Vous indiquerez quelles lignes de code ajouter (éventuellement modifier) et à quel(s) endroit(s).