

# INTRODUCTION A LA PROGRAMMATION

## Examen semestriel

### Instructions :

- Vous disposez de une 1 :45mn pour faire cet examen (13h15 - 15h).
- Nombre maximum de 105 points.
- 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 : **50** points.
  - Exercice 2 : **42** points.
  - Exercice 3 : **13** points.

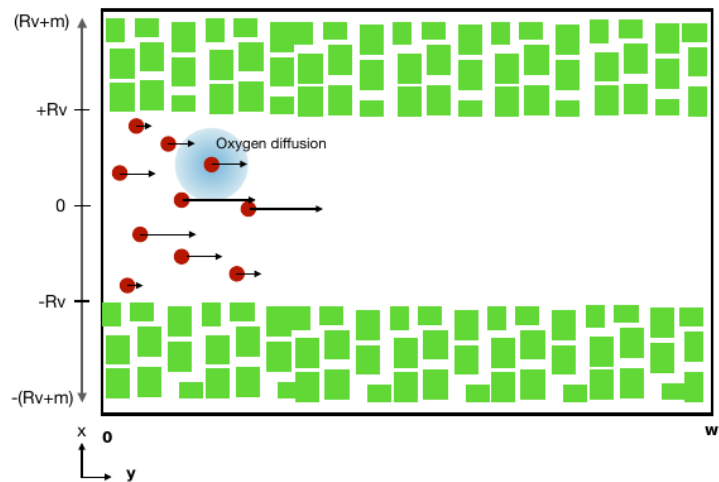
suite au verso ➞

## Exercice 1 : Conception OO [50 points]

Il est important de parcourir l'exercice dans son intégralité, notamment les contraintes imposées, et de consulter le code fourni en annexe avant de commencer à répondre aux questions.

On s'intéresse ici à simuler graphiquement une portion de vaisseau sanguin. Les parois du vaisseau sont constituées de cellules musculaires qui composent sa structure. La figure ci-dessous représente le vaisseau par une découpe en 2D sur une longueur donnée. A l'intérieur du vaisseau, un flux de liquide fait circuler les cellules sanguines.

- Cercle rouge : cellule sanguine. Transporte et diffuse de l'oxygène autour de leur position.
- Rectangle vert : cellule musculaire. Ces cellules forment la structure même du vaisseau sanguin. Elles ont besoin de l'oxygène fourni par cellules sanguines.
- Nous considérerons la position 0 au milieu du vaisseau, avec  $\pm R_v$  étant les bordures inférieures/supérieures du vaisseau,  $m$  la largeur de la couche de cellules musculaires lisses et  $w$  la longueur de la portion de vaisseau simulée.



Dans ce qui suit, vous supposerez que la classe `Vec2d` fournie en annexe permet de représenter des coordonnées ou des vitesses. Vous supposerez également que les entités impliquées sont dessinables (selon des modalités connues mais dont le détail ne nous intéresse pas ici).

**Vaisseau sanguin** La portion de vaisseau sanguin à simuler (classe `Vessel`) est caractérisée par les données :  $R_v$ ,  $m$  et  $w$  ainsi que par le *coefficient de viscosité* du liquide qui y circule (des `double`). Ces données seront initialisées à la construction du vaisseau au moyen de données fournies en paramètres. Toutes les cellules intervenant dans le vaisseau sont caractérisées par une position (de type `Vec2d`). Vous supposerez également, par simplification, que toute cellule a connaissance du vaisseau auquel elle appartient. Cette information lui est transmise à sa construction. Il est enfin possible de calculer ce que l'on appelle le *gradient de pression*, à une position  $p$  du vaisseau sanguin, sous la forme d'un `double`. **Les modalités précises de ce calcul ne nous intéressent pas ici.**

**Cellules sanguines** une cellule de ce type est dotée d'une position. A sa création, elle se verra attribuer une *vitesse* qui sera calculée en fonction du *gradient de pression* à sa position et du *coefficient de viscosité* dans le vaisseau. Cette vitesse demeurera constante après son initialisation. A chaque pas de temps  $dt$ , la cellule se déplacera d'une distance  $dz$  calculée en fonction de sa position courante et de sa vitesse. La direction de déplacement est toujours  $(0,1)$  dans le système de coordonnées  $(x,y)$ . Quand une *cellule sanguine* arrive à l'extrémité droite de la portion de vaisseau, elle doit être supprimée du vaisseau.

À chaque pas de temps  $dt$ , un nombre aléatoire de cellules positionnées au hasard sont créés dans le vaisseau. Les modalités exactes de cette génération ne nous intéressent pas mais nécessitent de pouvoir tester si une position  $(x,y)$  se trouve bien dans la zone adéquate du vaisseau :

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

**Cellules musculaires** Une cellule de ce type est caractérisée par un niveau d'oxygène (un double) et sa position. Elle ne se déplace pas mais doit être "fournie" en oxygène par les *cellules sanguines* qui circulent. A chaque pas de temps  $dt$ , son niveau d'oxygène baisse selon un algorithme dépendant de  $dt$  (et dont les modalités de calcul ne nous intéressent pas). En contrepartie, son niveau d'oxygène est aussi augmenté en fonction du nombre de cellules sanguines présentes dans leur voisinage. Pour simplifier, vous supposerez :

- qu'il n'est pas nécessaire de simuler la diffusion de l'oxygène par les cellules sanguines ;
- qu'il existe un algorithme permettant de calculer la quantité d'oxygène présente dans le vaisseau au voisinage d'un position  $p$  ; ce calcul dépendant uniquement du nombre de cellules sanguines présentes dans le voisinage de  $p$  ;
- et que le calcul du gain en oxygène par une cellule musculaire dépend uniquement de la quantité d'oxygène présente dans son voisinage.

Les cellules musculaires qui ont un niveau d'oxygène nul meurent et doivent disparaître de la simulation. Enfin, les cellules musculaires seront toutes créées lors de la création du vaisseau sanguin, selon un algorithme connu mais non spécifié. Vous supposerez que cet algorithme génère un nombre aléatoire de cellules musculaires et les ajoute au vaisseau à chaque fois qu'il est invoqué.

### Question 1 : Conception [40 points]

On vous demande d'écrire la/les classe(s) permettant de mettre en oeuvre les fonctionnalités souhaitées ; seules les entêtes des méthodes sont demandés. **On ne vous demande pas d'écrire de programme complet, ni le corps des méthodes, uniquement les attributs et les entêtes de méthodes.** Vous pouvez décrire les classes et leur contenu au moyen de diagrammes ou en pseudo-code Java. Les contraintes suivantes seront respectées :

1. Vous supposerez notamment qu'un programme principal existe, qu'il crée un objet **Vessel**, et appelle en boucle sa 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**).
2. 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**) ;
3. 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.**
4. **Ne négligez ni les constructeurs, ni les droits d'accès et les modificateurs.**
5. Le droit **protected** ne doit pas être utilisé pour les attributs et vous utiliserez des **ArrayList** pour modéliser les collections.
6. Votre conception doit être bien modularisée.
7. Vous ne proposerez de getter/setter que lorsqu'ils sont nécessaires à la mise en oeuvre de la simulation décrite.
8. Il n'est pas nécessaire d'écrire des directives d'importation.

### Question 2 : Programmation/Extensions [10 points]

1. **(6 points)** Donnez le corps de la méthode **update** de la classe **Vessel**. On rappelle l'existence de la méthode **Object remove(index i)** des **ArrayList**.
2. **(4 points)** Supposons que l'on veuille modéliser le fait qu'un vaisseau qui a perdu plus de la moitié de ses cellules musculaires (par rapport au nombre présent à sa création) est un vaisseau mort (c'est un état que l'on aimerait pouvoir tester). Que proposez-vous d'ajouter à votre conception pour mettre en oeuvre cette fonctionnalité sans faire de test de type ? Vous donnerez le code Java à ajouter (attributs, méthodes, corps compris) en indiquant où il doit être ajouté. Il n'est pas nécessaire de tenir compte de cet ajout dans le codage de la méthode **update** de la classe **Vessel**.

## Exercice 2 : Concepts de base [42 points]

Répondez clairement et succinctement aux questions suivantes :

1. [6 points] Soit le code suivant :

```

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) Qu'affiche t-il ?  
 (b) Qu'affiche t-il si l'on supprime le `this.` à la ligne 5.

Justifiez brièvement vos réponses.

2. [9 points] Soit le code suivant :

```

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 }
```

Complétez-le de sorte à ce que le programme principal affiche la valeur des attributs de l'objet `a` sous le format : 5 10

Une seule méthode additionnelle pourra être ajoutée à chacune des classes et le programme principal doit rester inchangé. Vous donnerez le code Java à ajouter et indiquerez où il doit être ajouté.

3. [15 points] Soit la portion de code suivante (on suppose que toutes les inclusions nécessaires sont faites) :

```

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 }

```

(vous justifierez brièvement chacune des réponses ci-dessous).

- Qu'affiche t-il ?
- Quelle(s) erreur(s) sont signalées par le compilateur si l'on supprime le constructeur de la ligne 4 ? (vous supposerez que les numérotations de lignes demeurent inchangées).
- Qu'affiche t-il si l'on supprime uniquement la ligne 5 ?
- Pourquoi le compilateur émettra t-il un message d'erreur, si l'on ajoute la ligne  
`w = 12;`  
dans le corps du constructeur de la ligne 18 du programme donné ci-dessus. Que faut-il faire dans ce constructeur pour que l'attribut de W soit initialisé à 12 ?
- Dans le programme fourni au départ, peut-on ajouter la déclaration :  
`int w = 12;`  
juste après la ligne 17. Si non pourquoi ? Si oui qu'affiche le programme ?

suite au verso ➞

4. [12 points] Soit le programme suivant (on suppose que toutes les inclusions nécessaires sont faites) :

```

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) Pourquoi supprimer la méthode `sing` dans la classe `Artist` provoque une faute de compilation ?
- (b) Pourquoi transformer la méthode `sing` en méthode abstraite dans la classe `Artist` provoque une faute de compilation ?
- (c) Si l'on part de l'hypothèse que d'autres types d'artistes pourront être ajoutés, qui feront autre chose que chanter (`sing`) ou peindre (`paint`), quelles critiques pouvez-vous émettre sur la méthode `perform` et la modélisation de la hiérarchie d'artistes. Proposez les modifications à faire pour résoudre les problèmes soulevés. Vous indiquerez le cas échéant quelles parties réécrire et comment.
- (d) Comment proposez-vous de modéliser le fait que certains types d'artistes produisent des œuvres matérielles (peintures, sculpture par exemple) et que dans ce cas, l'on doit pouvoir trouver où ces œuvres sont exposées (un ensemble de lieux dont chacun est décrit sous la forme d'une `String`) ?

suite au verso ➞

### Exercice 3 : Déroulement de programme [13 points]

Le programme suivant compile et s'exécute sans erreurs (vous supposerez les directives d'importation présentes). Qu'affiche t-il ? Expliquez succinctement son déroulement. **Il ne s'agit pas ici de paraphraser le code, mais bien d'expliquer les étapes et le déroulement du programme.**

```

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     private boolean s;
14
15     Status(boolean s) { this.s = s; }
16     boolean getS() { return s; }
17 }
18
19 abstract class A implements D
20 {
21     private String n;
22     private Status s;
23
24     public A(String n, Status s) {
25         this.n = n;
26         this.s = s;
27     }
28
29     public boolean e(A other) { return s.getS(); }
30     public abstract boolean f(B other);
31     public abstract boolean f(C other);
32
33     @Override
34     public String d() { return (n + s.toString()); }
35 }
36
37 class B extends A
38 {
39     public B() { super("small", Status.GREEN); }
40
41     public boolean e(A other) { return other.f(this); }
42     public boolean f(C other) { return true; }
43     public boolean f(B other) { return false; }
44 }
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 }

```