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 trois heures pour faire cet examen (9h00 - 12h00).
- Nombre maximum de 105 points (dont 15 en bonus).
- 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.
- 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**. Vous disposez de pages additionnelles à la fin du fornulaire de réponse.
- 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**

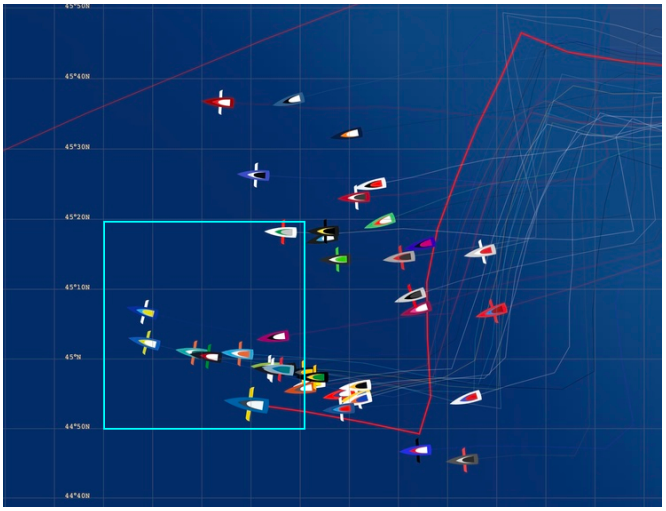
  - Exercice 1 : **50** points.
  - Exercice 2 : **38** points.
  - Exercice 3 : **17** points.

suite au verso ☞

Exercice 1 : Conception OO [50 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 3 questions.

Le Vendée Globe est une célèbre course nautique en solitaire. L'organisation a mis en place un outil de visualisation permettant de suivre les skippers en direct. On se propose de structurer cet outil en utilisant une approche orientée-objet.



L'outil de visualisation doit permettre typiquement d'afficher des données caractéristiques des bateaux ou des projections de trajectoire pour ces derniers.

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. Vous supposerez qu'il existe une classe `API` utilisable par une directive d'importation et qui permet de connaitre les informations relatives aux bateaux. Vous supposerez aussi qu'un programme principal existe, qu'il crée les objets nécessaires, appelle en boucle une méthode de mise à jour, ainsi qu'une méthode de dessin sur une objet `Canvas`.
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** ;
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 : Bateaux et carte (28.5 points)

Un *bateau* est caractérisé par le *nom de son skipper*, ses *coordonnées nautiques* (un `NavigationCoord`), sa *vitesse* (un `double`), son *rang* dans la course, sa *distance* au bateau en tête de course et un *identifiant* (un entier qui lui est propre). Ces données sont initialisées à la construction au moyen de paramètres. Le rang, la vitesse et la distance sont nuls au départ.

Les bateaux peuvent être de différents types, pour simplifier : des *Monocoque* ou des *Catamarans*. Ces derniers peuvent être équipés de voiles ou pas. Une carte, à visualiser par l'outil de suivi de la course, comporte à ce stade un *ensemble de bateaux*, et la coordonnée nautique (`NavigationCoord`) sur laquelle est centrée sa vue (c'est le milieu de la carte tel que visualisé par l'outil). Ces données sont aussi fournies à la construction. Si la coordonnée du centre n'est pas fournie elle aura une valeur par défaut correspondant à la constante fournie par `API`.

Les fonctionnalités de base suivantes sont souhaitées à ce stade :

1. accéder individuellement à l'ensemble des informations caractéristiques des bateaux ;
2. changer le centre de la carte ;
3. dessiner une carte sur un `Canvas` ; ce qui implique de dessiner tous ses bateaux ;
4. mettre à jour, en une seule méthode, les données caractéristiques des bateaux sur la carte au moyen des outils de l'`API` ;
5. supprimer un bateau d'identifiant donné de la carte.

On considère que le dessin des bateaux ne peut être implementé concrètement que pour des types de bateaux spécifiques. On souhaite également que la méthode de mise à jour puisse utiliser des sous-classes éventuelles de `API` (ce qui serait décidé dans le programme principal) et que l'`API` ne soit pas communiquée aux bateaux. **Écrivez les classes et éventuelles interfaces permettant de représenter le modèle décrit ci-dessus. Vous donnerez aussi le corps complet des méthodes de mise à jour**.

Question 2 : Champs de données maritimes (16.5 points)

On souhaite maintenant compléter l'outil en permettant de visualiser une estimation de trajectoire optimale pour un bateau donné.

Cette estimation peut être faite par des algorithmes que l'on supposera connus mais spécifiques à chaque type de bateau. Ces algorithmes dépendent de *champs de données maritimes* (`Field`, grille turquoise dans le figure) fournissant des informations sur le vent, les courant marins etc. Un `Field` est une grille carrée caractérisée par la coordonnée nautique (`NavigationCoord`) de son point extrême en haut à droite, et sa taille (un entier pour simplifier). Chaque cellule de sa grille contient une donnée du type fourni `Data`. Deux types de `Field` seront considérés : des *champs de vent* (avec des données relatives au vent, type fourni `WindData`) et des `champs de courants marins` (avec des données relatives au courant marins, type fourni `OceanCurrentData`). Les fonctionnalités souhaitées pour les `Field` sont les suivantes :

1. construire un champ en lui attribuant des coordonnées, une taille et une grille vide ;
2. retourner la coordonnée nautique correspondant au centre de la cellule de coordonnée `[i][j]` de la grille ;
3. remplir la grille de données en garantissant qu'elle ne contiennent que des données de type `WindData` pour les champs de vent et que des données de type `OceanCurrentData` pour des champs de courants marin. Ce remplissage se fera au moyen des méthode `getWindValue()` et `getOceanCurrentValue()` de l'API. On souhaite de plus que cette fonctionnalité de remplissage n'accède qu'à une vue restreinte de l'API qui n'offre que les méthodes `getWindData` et `getOceanCurrentData()`.

Écrivez les classes et éventuelles interfaces permettant de représenter le modèle décrit ci-dessus. Vous donnerez aussi le corps complet des méthodes de mise à jour.

Question 3 : Trajectoires optimales (5 points)

La fonctionnalité additionnelle dont on souhaite pouvoir disposer pour les bateaux est le calcul d'une estimation de trajectoire optimale (type fourni `Trajectory`) sur la base d'un ensemble **variable**, potentiellement vide, de `Field`. Par exemple on souhaite pouvoir calculer la trajectoire optimale en tenant compte d'un `Field` de type *"vent"* et d'un `Field` de type *"courant océanique"* (de même taille et position) ou encore calculer la trajectoire en tenant compte uniquement d'un `Field` de type *"vent"* ou uniquement d'un `Field` de type *"courant océanique"*. La méthode calculant une estimation de trajectoire optimale appliquera des algorithmes spécifiques aux types de bateaux (et qui utilisent les données des champs).

Donnez les entêtes de méthodes à ajouter à votre conception pour permettre le calcul des trajectoires optimales en indiquant où les placer.

Vous considérerez que de nouveaux types de champs peuvent être ajoutés dans la conception par la suite.

Exercice 2 : Concepts de base [38 points]

Répondez clairement et succinctement aux questions suivantes :

1. **[6 points]** Soient le code suivant :

```
1  public class BitManipulation {
2
3      public static void main(String[] args) {
4        //12 = 00000000_00000000_00000000_00001100 en binaire
5        int value = 12;
6        System.out.println(getXthBit(value, 3));
7        System.out.println(getXthBit(value, value));
8
9        int newValue = embedInXthBit(value, false, 2);
10       print(value, newValue);
11       // -12 = 11111111_11111111_11111111_11110100 en binaire
12       value = -12;
13       newValue = embedInXthBit(value, false, 2);
14       print(value, newValue);
15     }
16
17     public static void print(int value, int newValue){
18       System.out.println("Original Value: " + value);
19       System.out.println("Modified Value: " + newValue);
20     }
21
22     public static boolean getXthBit(int value, int pos) {
23       assert 0 <= pos && pos < Integer.SIZE;
24       final int lsb = value & (0x1 << pos);
25       return lsb != 0;
26      }
27
28     public static int embedInXthBit(int value, boolean m, int pos
          ) {
29       assert 0 <= pos && pos < Integer.SIZE;
30       final int mask = 0x1 << pos;
31       return m ? value | mask : value & ~mask;
32     }
33
34 }
```

(a) Expliquez brièvement les étapes de fonctionnement de la méthode `getXthBit`.

(b) Expliquez brièvement les étapes de fonctionnement de la méthode `embedInXthBith`.

(c) Indiquez ce qu'il affiche.

2. **[6 points]** Soit le code suivant :

```
1  class W {
2      private String w = "colugo";
3      public String toString(){ return w;}
4      public void setW(String w){ this.w = w; }
5  }
6  class Pgme {
7    public static void main(String[] args){
8      W w1 = new W(); W w2 = new W();
9      m(w1,w2);
10     System.out.println(w1);
11     System.out.println(w2);
12   }
13    public static void m(W w1, W w2){
14     w1 = w2;
15     w2.setW("hyrax");
16    }
17 }
```

(a) Qu'affiche t-il ?

(b) Qu'affiche t-il si on insère l'instruction `w1=w2;` entre les ligne 8 et 9 ?

(c) Qu'affiche t-il si on supprime `this.` à la ligne 4 ?

Justifiez brièvement chacune de vos réponses.

3. **[12.5 points]** Soit le code suivant :

```
1  class A {
2      private String a = "A";
3      public A() { System.out.println(a); }
4      public String toString(){ return a;}
5  }
6  class B extends A {
7     private String b = "B";
8     public B() { System.out.println(b); }
9     public B(String b) { this.b  = b; }
10    public String toString(){ return super.toString() + b; }
11 }
12 class C extends B {
13    private String c = "C";
14    public String toString(){ return super.toString() + c; }
15 }
16
17 class Ctor {
18     public static void main(String[] args){
19        C c = new C();
20        System.out.println(c);
21     }
22 }
```

(a) Qu'affiche t-il ?

(b) Que se passe t'il si on remplace `super` par `this` à la ligne 10 ?

(c) Le programme compile t-il toujours si on supprime le constructeur de `B` de la ligne 8 ?

(d) Est-ce que le programme continue de pouvoir s'exécuter si on supprime la méthode `toString` de `B` ?

(e) Est il possible d'ajouter `this();` comme première instruction du constructeur par défaut de `B` ?

(f) Est il possible d'ajouter `this("b")` comme comme première instruction du constructeur par défaut de `B` ?

(g) Est il possible d'ajouter `super();` comme première instruction du constructeur par défaut de `B` ?

(h) Est il possible d'ajouter `super("b")` comme comme première instruction du constructeur par défaut de `B` ?

Justifiez brièvement vos réponses.

suite au verso ☞

4. **[13.5 points]** Soit le programme suivant dont on souhaite qu'il affiche une description des articles stockés dans une bibliothèque, leur nombre et le prix de chacun d'eux.

```
1  abstract class ShelfItem {
2    private String title;
3    public ShelfItem (String title){ this.title = title;}
4    public abstract String getId();
5    public int getPages() {return -1; }
6    public String getTitle() {return title; }
7  }
8
9  class Book extends ShelfItem {
10   private final int nbPages;
11   public Book(String title, int nbPages) {
12   super(title); this.nbPages= nbPages;
13   }
14   public String getId() { return "book::" + getTitle(); }
15   public int getPages() { return nbPages; }
16 }
17
18 class DVD extends ShelfItem {
19   public DVD(String name) { super(name); }
20   public String getId() { return "dvd::" + getTitle(); }
21 }
22
23 class Shelf {
24   public static int CAPACITY = 200;
25   private ShelfItem[] items = new ShelfItem[CAPACITY];
26   private int numItems = 0;
27
28   public void add(ShelfItem item) {
29     if (numItems < CAPACITY-1)
30       items[numItems++] = item;
31   }
32
33   public String toString(){
34     String description = "";
35     for (int i=0; i < numItems; ++i){
36       description += items[i].getId();
37       int pages = items[i].getPages();
38       if (pages != -1) {
39         description += "," + Integer.toString(pages)
40                     + " pages";
41       }
42       description += "\n"; // new line
43     }
44     return description;
45   }
46   public ShelfItem[] getAllItems() {return items;}
47 }
48
```

```
49
50 class PriceCalculator {
51   public static float price(Shelf shelf) {
52     float price = 0.0f;
53     for (ShelfItem item : shelf.getAllItems())
54       price += price(item);
55       return price;
56   }
57
58   public static float price(ShelfItem item) { return 0.0f; }
59
60   public static float price (Book book) { return 5.0f; }
61
62   public static float price(DVD dvd) { return 2.0f; }
63 }
64
65
66 public class StaticPolym {
67   public static void main(String[] args) {
68     Shelf shelf = new Shelf();
69     Book book = new Book("The old man and the waves", 250);
70     shelf.add(book);
71     DVD dvd = new DVD("the-gleaming");
72     shelf.add(dvd);
73     System.out.println(" The content of the shelf is:\n" + shelf)
          ;
74     System.out.println("The Shelf has : "
75               + shelf.getAllItems().length + " items");
76     float totalPrice = PriceCalculator.price(shelf);
77     System.out.println("The shelf price is : " + totalPrice);
78   }
79 }
```

(a) Quel affichage produit-il ?

(b) Quelle critique pouvez-vous émettre sur la mise en oeuvre de la méthode `getPages()` et quelle meilleure solution proposeriez-vous ?

(c) Quelle critique pouvez-vous émettre sur le getter `getAllItems()` et quelle meilleure solution proposeriez-vous ?

(d) Quelles autres modifications proposeriez vous pour produire l'affichage initialement voulu ?

Une explication en français suffit pour les questions (b) à (d), mais vous pouvez à la place donner le code que vous proposeriez pour remplacer celui fourni.

suite au verso ☞

Exercice 3 : Déroulement de programme [17 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  import java.lang.reflect.Array;
2  import java.util.ArrayList;
3
4  interface IdGenerator {
5      int generateId();
6  }
7  interface DefaultIdGenerator extends IdGenerator {
8      int START_ID = 0;
9      default int generateId() { return START_ID; }
10 }
11 class SequentialIdGenerator implements DefaultIdGenerator {
12     private static final int START_ID = 5;
13     private static final SequentialIdGenerator instance
14                                 = new SequentialIdGenerator();
15     public static SequentialIdGenerator getInstance() {
16                                             return instance; }
17     private int nextId;
18
19     protected SequentialIdGenerator(int startId) {
20                                             nextId = startId; }
21     private SequentialIdGenerator() {
22         this(DefaultIdGenerator.START_ID + START_ID);
23     }
24     @Override
25     public int generateId() {
26         System.out.println("Generating ID: " + (++nextId));
27         return nextId++;
28     }
29 }
30 interface Saluter {
31     default String salute(InformalUser user){
32         return "Hello " + user;
33     }
34     default String salute(FormalUser user){
35         return "Delighted to see you " + user;
36     }
37 }
38 interface Salutable {
39     String acceptSalutation(Saluter saluter);
40 }
41
42 abstract class User implements Salutable {
43     private int id;
44     private String name;
45     private String username;
46     public User() { this("User"); }
47
```

```
48      public User(String name) {
49          this.name = name;
50          this.username = name.toLowerCase();
51          id = SequentialIdGenerator.getInstance().generateId();
52      }
53      @Override
54      public String toString() { return getUsername() + "#" + id; }
55      public String getUsername() { return username; }
56  }
57
58  class InformalUser extends User {
59      private String name;
60      public InformalUser(String name) {
61          super();
62          this.name = name;
63      }
64      @Override
65      public String acceptSalutation(Saluter saluter) {
66          return saluter.salute(this);
67      }
68      @Override
69      public String toString() {
70          return name + " (" + getUsername() + ")";
71      }
72  }
73  class FormalUser extends User {
74      public FormalUser(String name) { super(name); }
75      @Override
76      public String acceptSalutation(Saluter saluter) {
77          return saluter.salute(this);
78      }
79  }
80  class Platform implements Saluter {
81      private ArrayList<User> users = new ArrayList<>();
82      public void add(User user) { users.add(user); }
83      public void welcomeUsers() {
84          for (User user : users) {
85                System.out.println(user.acceptSalutation(this)); }
86      }
87  }
88
89  public class Deroulement {
90      public static void main(String[] args) {
91          Platform platform = new Platform();
92          platform.add(new InformalUser("Bob"));
93          platform.add(new FormalUser("Alice"));
94          platform.add(new FormalUser("Charlotte"));
95          platform.welcomeUsers();
96      }
97  }
```