IN201 : Conception et Programmation Orientées Objet

Examen : première partie

Cet examen de 1h15 est composé de 8 exercices indépendants. Le barême indiqué pour chaque exercice l'est à titre indicatif et peut être modifié.
Les seuls documents autorisés pour cet examen sont :
– les notes distribuées en cours
– vos notes manuscrites
Les annales des examens des années précédentes sont interdites. Les téléphones portables doivent être éteints et rangés. L'utilisation d'un ordinateur durant l'examen est interdite.
Vous avez normalement la place de répondre sur ce document. Si vous devez utiliser une copie libre, n'oubliez pas d'y indiquer vos noms, prénoms et groupe de PC, ainsi que le numéro précis de la question à laquelle vous répondez.

Nom : _____

Prénom : _____

Groupe : _____

Question	Points	Score
1	2	
2	1	
3	1	
4	1	
5	2	
6	1	
7	1	
8	1	
Total :	10	

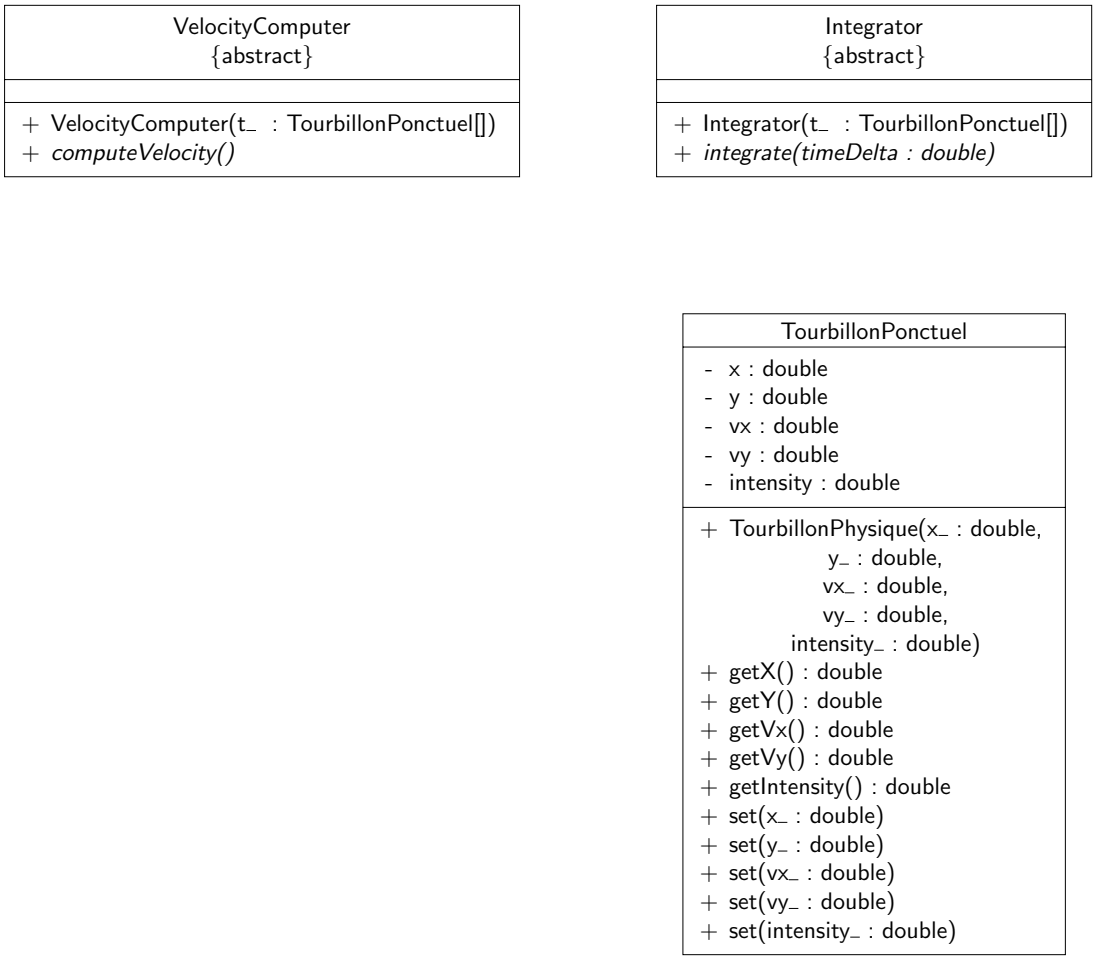
Commentaires éventuels :

1. En aérodynamique, on peut modéliser des phénomènes de *tourbillons physiques* par un ensemble de *tourbillons ponctuels* définis par leur position, leur vitesse et leur intensité. Nous considérons ici des tourbillons dans le plan 2D et nous cherchons à simuler l'évolution de tourbillons physiques. Les positions et vecteurs vitesses seront représentés en coordonnées cartésiennes. Les tourbillons ponctuels interagissant les uns avec les autres, un tourbillon physique utilise pour calculer son évolution après un certain pas de temps

– un objet de type `VelocityComputer` pour mettre à jour la vitesse des tourbillons
– un objet de type `Integrator` pour calculer la position des tourbillons à partir de leur vitesse

On suppose que le calcul de l'évolution d'un tourbillon physique est demandé de façon externe par un utilisateur en donnant le pas de temps nécessaire.

($\frac{1}{2}$ pt) (a) compléter le diagramme de classe UML suivant notamment en introduisant la classe `TourbillonPhysique` et des associations entre les classes. Si vous souhaitez introduire des accesseurs et modifieurs sur des attributs qui sont nombreux, vous pouvez en écrire un seul et mettre « ... » pour les autres.



Dans la question suivante, on écrira des commentaires javadoc *succincts* permettant de comprendre l'utilisation des méthodes, paramètres etc.

(1 pt) (b) donner le code Java de la classe `TourbillonPhysique`. On écrira une méthode `toString` pour `TourbillonPhysique` en supposant que la classe `TourbillonPonctuel` en possède une permettant de représenter les caractéristiques des tourbillons par une chaîne de caractères.

($\frac{1}{2}$ pt) (c) avec ce qui est vous est donné, peut-on écrire un programme (méthode `main`) créant un tourbillon physique composé de 3 tourbillons ponctuels de vitesse initiale nulle et calculant la position des tourbillons après un pas de temps de 1s ? Si oui, écrire la méthode `main` correspondante.

(1 pt) **2.** Lors d'un match de football opposant l'équipe A à l'équipe B, le numéro 6 de l'équipe A récupère la balle en tacklant le numéro 10 de l'équipe B qui se roule de douleur par terre. Le numéro 6 fait ensuite un demi-tour sur lui-même, passe la balle à son numéro 8 qui court avec sur 15m avant de centrer pour son numéro 10 qui marque de la tête.

Représenter le scénario précédent par un diagramme de séquence.

(1 pt) **3.** On dispose d'une classe `Processor` possédant une méthode `process` prenant un entier en paramètre et renvoyant un double. On souhaite tester la méthode `process`. Pour cela, on dispose d'un certain nombre de valeurs attendues en sortie en fonction de la valeur du paramètre de `process` :

entrée	résultat
1	5.9
2	-3.475
10	13
16	-23.675

Écrire une classe de test JUnit pour tester la méthode `process`. On suppose que l'on souhaite avoir une précision à 10^{-6} près pour `process` et que le constructeur de `Processor` ne prend pas d'argument. On évitera d'écrire « à la main » toutes les assertions permettant de vérifier que le résultat de l'appel à `process` est bien conforme à ce que l'on attend.

(1 pt) **4.** Une pile est une structure de données possédant les opérations suivantes :
- `push(element)` qui permet de mettre un élément dans la pile
- `pop()` qui permet de récupérer l'élément sur le haut de la pile et de l'enlever de la pile
- `isEmpty()` qui permet de savoir si la pile est vide ou pas

On considère ici une pile contenant des valeurs de type **`double`**. Les éléments de la pile seront stockés dans un tableau. Lorsque l'on construit une pile, celle-ci est vide et on donne sa capacité maximale. Il est impossible d'empiler plus d'éléments que la capacité maximale. Écrire une classe `LIFO` représentant une pile et introduire les spécifications JML des méthodes de la classe (pré et postconditions, on n'écrira pas d'invariants). Pour éviter des problèmes de visibilité dans les spécifications JML, on supposera que les attributs ont une visibilité publique, ce qui permet de les utiliser dans les spécifications.

5. Une personne est représentée par la classe donnée ci après. La méthode `behave` décrit le comportement d'une personne et son paramètre représente l'heure à laquelle on désire avoir le comportement de la personne.

Personne
- nom : String - prenom : String
+ Personne(nom_ : String, prenom_ : String) + behave(heure : int)

(1 pt) (a) on souhaite spécialiser `Personne` en une classe `Employe` de telle sorte que :

- un employé possède un salaire représenté par un réel ;
- un employé se comporte comme un employé entre 8h et 12h et 13h et 18h, comme une personne normale sinon. On représentera le fait de se comporter comme un employé par l'affichage d'une chaîne de caractères « `Je travaille...` »

Écrire le code Java de la classe `Employe`.

(½ pt) (b) que va produire l'exécution du programme suivant ? Pourquoi ?

```
public class MainEmploye {
    public static void main(String[] args) {
        Employe cg = new Employe("Garion", "Christophe", 0);
        Personne tf = cg;

        cg.behave(9);
        tf.behave(9);
    }
}
```

(½ pt) (c) on souhaite avoir une seconde méthode `behave` dans `Employe` qui prenne en paramètre un entier qui représente non pas l'heure « absolue » mais le nombre d'heures depuis lequel l'employé travaille. Est-ce possible en Java ?

(1 pt) **6.** Barney Stinson est un garçon bien compliqué. La plupart du temps, il est dans son état normal dans lequel il a plusieurs activités : il travaille et rencontre ses amis. Mais dès qu'une ou plusieurs jeunes filles approchent de son champ de vision, il entre dans un état « *Suit up !* ». Dans cet état, deux possiblités s'offrent à lui : soit il est accompagné par un *wingman* et alors il tente une approche du type « *Have you met Barney ?* » (que l'on abrégera par HYMB), sinon une approche du type « *Lorenzo von Matterhorn* » (que l'on abrégera par LVM). Dans tous les cas, qu'il soit normal ou pas, au bout de 25 minutes (la durée d'un épisode), Barney redeviendra « *awesome* ».

Modéliser le comportement de Barney par un diagramme d'états-transitions.

FIGURE 1 : « Suit up ! »
Neil Harris Patrick as Barney Stinson, *How I Met Your Mother*, ©CBS 2005-2012

(1 pt) **7.** Un système de contrôle aérien est un système qui permet de réguler et de sécuriser les vols. Dans ce système, les contrôleurs aériens (ou ATC pour *Air Traffic Controlers*) jouent un rôle primordial. Chaque avion qui est en vol possède un plan de vol qui est généré par les pilotes de l'avion. Les avions dont est en charge un ATC particulier contactent l'ATC pour fournir leur plan de vol. L'ATC a donc à sa disposition tous les plans de vol des avions qu'il doit gérer et construit un *Flight Progress Strip* (FPS) pour chacun de ces avions. Pour pouvoir connaître les conditions de vol, l'ATC a accès à plusieurs stations météorologiques. Parmi les contrôleurs, on trouve les contrôleurs locaux, les contrôleurs sol, les contrôleurs d'approche, les contrôleurs de départ et les contrôleurs radio. Les contrôleurs peuvent se passer le contrôle de l'avion suivant la procédure suivante :

- le contrôleur sol peut passer le contrôle de l'avion au contrôleur local et vice-versa
- le contrôleur local peut passer le contrôle de l'avion au contrôleur de départ
- le contrôleur de départ peut passer le contrôle de l'avion à un contrôleur radio
- les contrôleurs radio peuvent se passer le contrôle de l'avion entre eux. Ils peuvent également passer le contrôle de l'avion à un contrôleur d'approche
- le contrôleur d'approche peut passer le contrôle de l'avion à un contrôleur local

Modéliser le domaine précédent avec un diagramme de classes ne faisant pas apparaître les attributs et les méthodes des classes. On essayera tant que possible de donner des multiplicités et des noms aux associations ou des noms de rôles et on précisera la navigabilité des associations. On fera apparaître les éventuelles classes abstraites et interfaces

(1 pt) **8.** Soient les classes `ClasseA` et `ClasseB` suivantes :

```
1  package monpackage;
2
3  public class ClasseA {
4
5      private int unAttribut;
6
7      public void ClasseA(int unAttribut_) {
8          unAttribut = unAttribut_;
9      }
10
11     public int m1(double a) {
12         a = this.unAttribut * a;
13     }
14 }
```

```
1  public class ClasseB extends monpackage.ClasseA {
2
3      private double unAutreAttribut;
4
5      public ClasseB(double unAutreAttribut_) {
6          this.unAutreAttribut = unAutreAttribut_;
7      }
8
9      public void setAttributs(double att) {
10         this.unAutreAttribut = att
11         this.unAttribut = att;
12     }
13 }
```

Quelles sont les erreurs dans ces classes qui seront détectées par le compilateur Java ? On pourra soit indiquer **en rouge** directement dans le code source les erreurs ou se servir des numéros de ligne pour les préciser.