IN201 : Conception et Programmation Orientées Objet

Examen : première partie

Cette première partie de l'examen est composée de 6 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 (avec vos notes manuscrites)
– les cartes de référence distribuées en cours
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	1½	
3	1	
4	2	
5	1	
6	2½	
Total :	10	

Commentaires éventuels :

1. Un arbre binaire est une structure de données contenant des nœuds reliés telle que :
- un et un seul nœud appelé la racine de l'arbre n'a pas de parent
- chaque nœud de l'arbre a un nœud parent unique (sauf la racine)
- un nœud a au plus deux fils

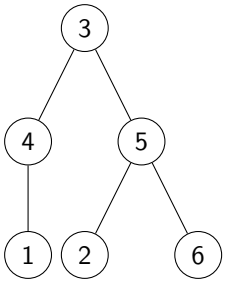
On suppose que les nœuds contiennent des entiers. On suppose que l'on dispose d'une méthode permettant de calculer la somme des entiers contenus dans les nœuds d'un arbre.

($\frac{1}{2}$ pt) (a) proposer un diagramme UML présentant les classes `Noeud` et `Arbre`, leurs éventuels attributs, leurs méthodes et les relations existant entre elles. 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. On réfléchira plus particulièrement à quelle est (sont) la classe (les classes) devant contenir la méthode permettant de calculer la somme des noeuds d'un arbre.

(½ pt) (b) écrire le code de la classe `Noeud`

(½ pt) (c) écrire le code de la classe `Arbre`

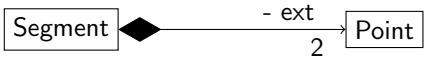
(½ pt) (d) écrire une classe possédant une méthode `main` construisant l'arbre présenté ci-dessous et affichant sur la console la somme des entiers contenus dans cet arbre.



(1½ pt) **2.** on cherche à modéliser le système immunitaire humain. Lorsque l'on attrape la grippe, le virus de la grippe entre dans le corps et ce dernier réagit dans un premier temps en produisant une inflammation. On suppose qu'un lymphocyte B a le bon récepteur antigène pour le virus. Il inactive le virus, puis phagocyte le virus et produit alors un complexe peptidique. Ce complexe est reconnu par un lymphocyte T qui stimule le lymphocyte B par l'envoi de cytokines. Le lymphocyte B se réplique alors en plasmocytes qui secrètent des anticorps permettant de neutraliser le virus.

Représenter le scénario précédent par un diagramme de séquence.

(1 pt) **3.** On considère les classes `Segment` et `Point` vues en cours liées par l'association suivante :



On rappelle de plus que la classe `Segment` possède un constructeur prenant deux instances de `Point` en paramètre et des méthodes `getExtremite1` et `getExtremite2` permettant de récupérer les points représentant les extrémités du segment.

Écrire une classe de test JUnit permettant de vérifier que les points passés en paramètres du constructeur de `Segment` sont différents des extrémités du segment. On supposera que l'on dispose des méthodes de `Segment` présentées tout au long du cours.

4. On souhaite écrire dans une classe `C` une méthode `m()` ne renvoyant rien qui effectue les opérations suivantes :

- affichage du texte `"Je demarre..."`
- appel à une méthode `pb()`
- affichage du texte `"J'ai fini..."`

On suppose que `pb` peut lever des exceptions de type `IOException` et `ArithmeticException`.

(1 pt) (a) écrire la méthode `m` en supposant que

- on veut traiter localement les exceptions de type `ArithmeticException` en affichant un message
- on veut propager les exceptions de type `IOException`

(b) soit le programme suivant utilisant `m` :

```
public class Main {
    public static void main(String[] args) {
        new C().m();
    }
}
```

(½ pt) (a) cette classe compile-t-elle ? Si non, expliquer pourquoi et corriger la classe pour qu'elle puisse être compilée.

(½ pt) (b) si l'on suppose que `pb` lève une `IOException`, que va-t-il s'afficher sur la console à l'exécution de `Main` ?

(1 pt) **5.** On s'intéresse ici à la modélisation d'un système ayant pour objectif la commande des gouvernes d'un avion en fonction de l'état courant de l'avion et des ordres du pilote et du copilote. On considére que l'avion possède un certain nombre de surfaces (gouvernes) qui sont contrôlées par le système de commandes de vol. Ces surfaces peuvent être les suivantes (on prend pour exemple un avion de type A320) :

- deux gouvernes de profondeur ;
- une gouverne de direction ;
- deux THS (*trimmable horizontal stabilizer*) qui sont des plans horizontaux réglables ;
- quatre volets ;
- plus de deux becs de bord d'attaque ;
- deux ailerons.

Le système de commande de vol est composé quant à lui des éléments suivants :

- un ADIRS (*Air Data and Inertial Reference System*) qui calcule les données décrivant l'état de l'avion. Ces données sont calculées à partir de capteurs situés sur l'avion ;
- un FMS (*Flight Management System*) qui élabore les consignes de vol à destination du pilote automatique ;
- un PA (pilote automatique) qui calcule les ordres à destination des gouvernes de l'avion en fonction des données fournies par le FMS ;
- un EFCS (*Electrical Flight Control System*) qui calcule les angles à appliquer aux gouvernes. L'EFCS est associé à un ADIRS pour pouvoir connaître l'état de l'avion. En réalité, il n'existe pas d'EFCS « général », mais deux EFCS spécialisés :
  - un EFCS pour le mode automatique, qui calcule les angles en fonction des ordres du PA ;
  - un EFCS pour le mode manuel, qui calcule les angles à partir des ordres de l'équipage de l'avion.

  Les éléments interagissent entre eux dans cet ordre (le FMS envoie des données au PA par exemple).

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.

6. Soient les classes suivantes :

```
public class AnimalHouse<E> {
  private E animal;
  public void setAnimal(E x) {
      animal = x;
  }
  public E getAnimal() {
      return animal;
  }
}

public class Animal{
}

public class Cat extends Animal {
}

public class Dog extends Animal {
}
```

Pour chacun des extraits de code suivant, dire si le code échouera à la compilation, s'exécutera correctement ou produira des erreurs à l'exécution. Expliquer également la réponse.

(½ pt) (a)
```
AnimalHouse<Cat> house = new AnimalHouse<Cat>();
house.setAnimal(new Cat());
```

(½ pt) (b)
```
AnimalHouse<Cat> house = new AnimalHouse<Cat>();
house.setAnimal(new Dog());
```

(½ pt) (c)

```
AnimalHouse<Cat> house = new AnimalHouse<Animal>();
house.setAnimal(new Cat());
```

(½ pt) (d)

```
AnimalHouse house = new AnimalHouse();
house.setAnimal(new Dog());
```

(½ pt) (e)

```
AnimalHouse<Animal> house = new AnimalHouse<Animal>();
house.setAnimal(new Cat());
Cat c = house.getAnimal();
```