Faculté des Sciences
Aix*Marseille Université

Année universitaire 2017/2018

Site : ☒ Luminy ☐ St-Charles ☐ St-Jérôme ☐ Cht-Gombert ☒ Aix-Montperrin ☐ Aubagne-SATIS
Sujet de : ☒ 1er semestre ☐ 2ème semestre ☒ Session 2 Durée de l'épreuve : 2h
Examen de : L3 Nom du diplôme : Licence d'Informatique
Code du module : SIN5U2TL Libellé du module : Programmation et conception orientées objet
Calculatrices autorisées : NON Documents autorisés : OUI, notes de Cours/TD/TP

Exercice 2 : Canards (`Duck` = canard, `Flock` = groupe/volée de canards, `fly` = voler, `land` = atterrir)

Considérons les classes suivantes :

```
public class Duck {
    private boolean isFlying;
    public Duck()             { isFlying = false; }
    public boolean isFlying() { return isFlying; }
    public void fly()         { this.isFlying = true; }
    public void land()        { this.isFlying = false; }
}
```

```
public class Flock {
    private Duck[] ducks;

    public Flock(Duck... ducks) { this.ducks = ducks; }

    public void fly() {
      for (Duck duck : ducks) { duck.fly(); }
      assert isFlying();
    }

    public boolean isFlying() {
      for (Duck duck : ducks) { if (!duck.isFlying()) return false; }
      return true;
    }
}
```

```
public class ElectricDuck extends Duck {
    boolean isSwitchOn;

    public void switchOn() { isSwitchOn = false; }

    public void switchOff() {
      if (isFlying()) { land(); }
      isSwitchOn = false;
    }

    @Override
    public void fly() {
      if (!isSwitchOn) { return; }
      super.fly();
    }
}
```

Q1. Dans quel cas l'assertion de la méthode `fly` peut être fausse ? Pourquoi ?

Q2. Quel principe SOLID est violé par la classe `ElectricDuck` ?

Actuellement, les spécifications de la classe `Flock` imposent que la méthode `fly` fasse obligatoirement voler tous les canards. Nous souhaitons changer cela et remplacer le code de la méthode `fly` de `Flock` par celui-ci :

```
public void fly() {
  for (Duck duck : ducks) { if (!duck.canFly()) return; }
  for (Duck duck : ducks) { duck.fly(); }
  assert isFlying();
}
```

Q3. Implémentez la méthode `canFly` correctement dans la classe `Duck` et `ElectricDuck` de façon à respecter le contrat et faire disparaître le problème de conception évoqué à la question 2.

Exercice 1 : Coupe du monde

Le but de cet exercice est d'écrire un programme qui calcule le classement des groupes de la coupe du monde de football. Dans cet exercice, nous implémenterons seulement les trois premiers des 8 critères du règlement de la coupe du monde :

— Le plus grand nombre de points sur tous les matchs (victoire = 3 points, match nul = 1 point) ;

— La différence de buts sur tous les matchs (diff. de buts = buts marqués - buts encaissés) ;

— Le plus grand nombre de buts marqués sur tous les matchs.

En d'autres termes, on regarde en premier critère les nombres de points pour départager deux équipes. Si les deux équipes ont le même nombre de points, on regarde en second critère les différences de buts des deux équipes. Si les deux équipes ont la même différence de buts, on regarde en troisième critère les nombres de buts marqués.

Vous trouvez ci-dessous un exemple de classement respectant l'ordre :

Pays	Victoires	Nuls	Défaites	Buts pour (3)	Buts contre	Diff. buts (2)	points (1)
USA	2	1	0	7	3	4	7
Italie	2	1	0	6	2	4	7
Chili	0	1	2	2	4	-2	1
Pays-Bas	0	1	2	1	7	-6	1

Nous souhaitons pouvoir classer les pays de la façon suivante :

```
Team chili =       new Team("Chili");
Team italy =       new Team("Italie");
Team netherlands = new Team("Pays-Bas");
Team usa =         new Team("USA");
Team[] teams = { chili, italy, netherlands, usa };
Match[] matches = {
  new Match(usa,   italy,       2, 2),
  new Match(usa,   chili,       2, 1),
  new Match(usa,   netherlands, 3, 0),
  new Match(italy, chili,       1, 0),
  new Match(italy, netherlands, 3, 0),
  new Match(chili, netherlands, 1, 1)
};
Group group = new Group(teams, matches);
group.sort();
for (Team team : group.team) System.out.println(team.name);
```

Les classes `Team`, `Group` et `Match` sont déjà implémentées à l'exception de la méthode `sort` de la classe `Group` :

```
public class Team {
    public final String name;

    public Team(String name) { this.name = name; }
}
```

```
public class Group {
    public final Team[] teams;
    public final Match[] matches;

    public Group(Team[] teams, Match[] matches) {
      this.teams = teams; this.matches = matches;
    }

    public void sort() { /* à faire à la question 6. */ }
}
```

```
public class Match {
    public final Team team1, team2;
    public final int goalCount1, goalCount2;

    public Match(Team team1, Team team2, int goalCount1, int goalCount2) {
        this.team1 = team1; this.team2 = team2;
        this.goalCount1 = goalCount1; this.goalCount2 = goalCount2;
    }

    public boolean isWonBy(Team team) {
        return (team1 == team && goalCount1 > goalCount2)
            || (team2 == team && goalCount1 < goalCount2);
    }

    public int pointsFor(Team team) {
        if (isWonBy(team))     { return 3; }
        if (isADrawWith(team)) { return 1; }
        return 0;
    }

    public boolean isADrawWith(Team team) {
        return (team1 == team || team2 == team) && goalCount2 == goalCount1;
    }

    public int goalDifferenceFor(Team team) {
        if (team == team1) { return goalCount1 - goalCount2; }
        if (team == team2) { return goalCount2 - goalCount1; }
        return 0;
    }

    public int goalCountFor(Team team) {
        if (team == team1) { return goalCount1; }
        if (team == team2) { return goalCount2; }
        return 0;
    }
}
```

Q1. Écrivez le code de l'interface `Comparator<T>` de Java qui possède la méthode `int compare(T t1, T t2)`. Dans les questions suivantes, les implémentations de cette méthode devront retourner un nombre négatif, zéro ou un nombre positif si le premier argument est respectivement inférieur, égal ou supérieur au deuxième argument.

Q2. Écrivez le code de l'interface `BiFunction<T, U, R>` qui possède l'unique méthode `R apply(T t, U u)`.

Q3. Écrivez le code de la classe `ScoreComparator<T, U>` qui implémente l'interface `Comparator<U>`. Cette classe :

- possède un constructeur qui prend un tableau `array` de `T` et une référence `scoreFunction` de type `BiFunction<T, U, Integer>` en paramètres.
- implémente la méthode `compare(U u1, U u2)` de façon à retourner :

$$\sum_{t \in \text{array}} \text{scoreFunction.apply}(t, u_2) - \sum_{t \in \text{array}} \text{scoreFunction.apply}(t, u_1)$$

Chaque instance de la classe `ScoreComparator` va permettre de représenter un critère de tri. Par exemple, `new ScoreComparator<Match,Team>(matches,(match,team)->match.pointsFor(team))` construit une instance qui représente le premier critère de tri.

Q4. Écrivez le code de la classe `CompositeComparator<T>` qui implémente l'interface `Comparator<T>`. Cette classe :

- possède un constructeur qui prend en paramètres deux références `comparator1` et `comparator2` de type `Comparator<T>`.
- implémente la méthode `compare(T t1, T t2)` de façon à retourner :
  - `comparator1.compare(t1, t2)` si le résultat est différent de zéro ;
  - `comparator2.compare(t1, t2)` sinon.

Notez qu'une instance de la classe `CompositeComparator` permet pour le moment de composer deux critères de tri.

Q5. Comme nous souhaitons composer trois critères de tri, proposez une version de `CompositeComparator<T>` permettant de composer un nombre variable de comparateurs. Notez que cette classe doit nous permettre à la question suivante de composer les trois critères de tri afin d'obtenir le classement.

Q6. Donnez l'implémentation de la méthode `void sort()` de la classe `Group` qui trie le tableau `teams` en considérant les matchs du tableau `matches` et en composant les fonctions de score suivantes dans cet ordre :

- nombre de points (`pointsFor`) ;
- différence de buts (`goalDifferenceFor`) ;
- nombre de buts marqués (`goalCountFor`).

Vous pouvez utiliser la méthode statique `Arrays.sort(T[] array, Comparator<T> comparator)` de Java qui trie le tableau `array` en respectant le comparateur `comparator`.

Q7. Est-il facile d'ajouter de nouveaux critères si les trois premiers critères ne permettent pas de départager les équipes ? Quel principe SOLID est donc respecté ?