

Question de cours

Question 1 : Soit `s1` et `s2` deux variables de type `String`. Comment peut-on tester que les deux chaînes de caractères contiennent les mêmes caractères dans le même ordre ?

Il faut utiliser la méthode `equals` et `s1.equals(s2)` permet de tester si les deux chaînes de caractères sont égales, c'est-à-dire contiennent les mêmes caractères dans le même ordre.

Question 2 : Quelle instruction est rendue impossible (en dehors du code des constructeurs ou de l'initialisation par défaut) par l'utilisation du mot réservé `final` dans la déclaration d'un attribut `myAttribute` ?

L'affectation (de type `myAttribute = ...`) est rendue impossible (en dehors du code des constructeurs ou de l'initialisation par défaut) par l'utilisation du mot réservé `final` dans la déclaration d'un attribut `myAttribute`.

Question 3 : Écrire une expression pour le cosinus de $\pi/4$ en Java.

```
Math.cos(Math.PI/4)
```

Classe Player

Question 4 : Compléter la classe `Player`, représentant les joueurs. Chaque joueur possède un nom de famille (*lastName*) et un prénom (*firstName*). La méthode `getName()` permet de récupérer le nom complet (prénom et nom de famille séparé par un espace), la méthode `toString()` retourne simplement le nom de famille.

```
public class Player {  
    private final String firstName;  
    private final String lastName;  
  
    public Player(String firstName, String lastName) {
```

```

    this.firstName = firstName;
    this.lastName = lastName;
}
public String getName() { // exemple : "Ashleigh Barty"
    return firstName + " " + lastName;
}
public String toString() { // exemple : "Barty"
    return lastName;
}
public boolean equals(Object o) {
    if (!o instanceof Player) {
        return false;
    }
    Player other = (Player) o;
    return this.firstName.equals(other.firstName)
        && this.lastName.equals(other.lastName);
}
}

```

Question 5 : Écrire la méthode addScore.

```

public void addScore(int points){
    this.score += points;
}

```

Classe ScoreRecorder

Question 6 : Compléter les méthodes de la classe ScoreRecorder.

```

public class ScoreRecorder {
    private final List<PlayerScore> scores = new ArrayList<>();

    public int getScore(Player player) {
        return this.find(player).getScore();
    }
    public void givePoints(Player player, int points) {
        this.find(player).addScore(points);
    }
    public void givePoint(Player player) {
        this.givePoints(player, 1);
    }
}

```

```

private PlayerScore find(Player player) {
    for (PlayerScore pair : this.scores) {
        if (pair.getPlayer().equals(player)) {
            return pair;
        }
    }
    PlayerScore pair = new PlayerScore(player);
    scores.add(pair);
    return pair;
}

public int getTotalPoints() {
    int sum = 0;
    for (PlayerScore pair : scores) {
        sum += pair.getScore();
    }
    return sum;
}

public List<Player> getPlayers() {
    List<Player> players = new ArrayList<>();
    for (PlayerScore pair : score) {
        Player player = pair.getPlayer();
        if (!players.contains(player)) {
            players.add(player);
        }
    }
    return players;
}
}

```

La classe StandardGame

Question 7 : Dessiner le diagramme de classe pour pour chacune des classes Player, StandardGame et ScoreRecorder.

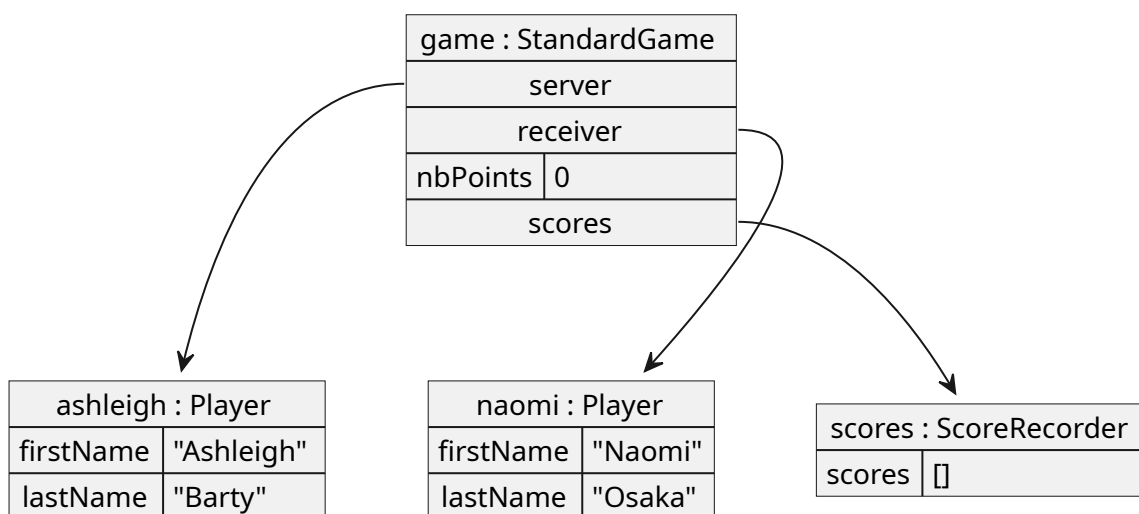
C StandardGame
<div> <div>server : Player</div> <div>receiver : Player</div> <div>scores : ScoreRecorder</div> <div>nbPoints : int</div> </div>
<div> <div>StandardGame(server : Player, receiver : Player)</div> <div>scoreNextPoint(player : Player)</div> <div>getScore() : ScoreRecorder</div> <div>getServer() : Player</div> <div>call() : String</div> </div>

C Player
<div> <div>firstName : String</div> <div>lastName : String</div> </div>
<div> <div>Player(firstName : String, lastName : String)</div> <div>getName() : String</div> <div>toString() : String</div> <div>equals(o : Object) : boolean</div> <div>hashCode() : int</div> </div>

C ScoreRecorder
<div> <div>scores : List<PlayerScore></div> </div>
<div> <div>getScore(player : Player) : int</div> <div>givePoint(player : Player)</div> <div>givePoints(player : Player, points : int)</div> <div>find(player : Player) : PlayerScore</div> <div>getTotalPoints() : int</div> <div>getPlayers() : Set<Player></div> </div>

Diagramme d'objets

Question 8 : Donnez l'état des objets créés (valeurs des attributs des objets) après l'exécution des trois premières instructions de la méthode `main`.



Question 9 : Afin de pouvoir écrire toute sorte de conditions de victoire, écrire l'interface `WinDecider` dont `PetanqueDecider` est une implémentation.

```
public interface WinDecider {  
    boolean isWinner(Player player, ScoreRecorder scores);  
}
```

Question 10 : Réaliser une implémentation `AdvantageDecider` de `WinDecider`. Pour cette implémentation, dont le constructeur aura deux paramètres entiers `minPoints` et `delta`, un joueur est vainqueur s'il possède au moins `minPoints` points, et au moins `delta` points de plus que n'importe lequel de ses adversaires.

```
public class AdvantageDecider implements WinDecider {  
    private final int minPoints;  
    private final int delta;  
    public AdvantageDecider(int minPoints, int delta) {  
        this.minPoints = minPoints;  
        this.delta = delta;  
    }  
    public boolean isWinner(Player player, ScoreRecorder scores) {  
        if (scores.getScore(player) < this.minPoints) {  
            return false;  
        }  
        for (Player opponent : scores.getPlayers()) {  
            if (!opponent.equals(player)  
                && scores.getScore(opponent) + this.delta > scores.getScore(player)) {  
                return false;  
            }  
        }  
        return true;  
    }  
}
```

Question 11 : Écrire 3 nouvelles méthodes publiques dans `ScoreRecorder` :

- `isWinner(Player player)` décide si un joueur donné est vainqueur,
- `getWinner()` retourne un joueur vainqueur s'il y en a au moins un, et `null` sinon,
- `isOver()` décide si la partie est terminée, c'est-à-dire s'il y a au moins un vainqueur.

```

public class ScoreRecorder{
    /* ... */
    public boolean isWinner(Player player) {
        return decider.isWinner(player, this);
    }
    public Player getWinner() {
        for (Player player : this.getPlayers()) {
            if (this.isWinner(player)) {
                return player;
            }
        }
        return null;
    }
    public boolean isOver() {
        return this.getWinner() != null;
    }
}

```