SCIENCES SORBONNE UNIVERSITÉ

LU2IN002-2021oct
Éléments de programmation
par objets avec Java

Examen du 10 janvier 2022 – Durée : 1 heure 30 minutes

Seul document autorisé : **feuille A4 manuscrite, recto-verso.**
Pas de calculatrice ou téléphone. Barème indicatif sur 40.

Remarques préliminaires : *la visibilité des variables et des méthodes à définir, ainsi que leurs aspects statique, ou final ne sera pas précisé. C'est à vous de décider de la meilleure solution à apporter. Si la méthode* `toString` *n'est pas demandée dans la question, il n'est pas nécessaire de la fournir. Sans aucune précision donnée dans la question, une méthode ne rend rien.*

Remarque : Le sujet est tel qu'il a été distribué à l'examen, il y a 2 bugs :
- les questions sont numérotées 2.x alors qu'il n'y a pas de 1.x
- plus grave : dans la question 2.10, il faut remplacer "une nouvelle version de la méthode inscritTournoi() de la classe Serveur" par "une nouvelle version de la méthode **inscription()** de la classe Serveur". Cela a été normalement annoncé dans les amphis.

On souhaite écrire des classes pour gérer une version simplifiée de serveur de tournois entre joueurs. Plusieurs classes composent ce programme :
- les classes de représentation des joueurs : `Joueur`, `JoueurAbo`, et `JoueurFTP`. Les joueurs sont soit des joueurs abonnés (`JoueurAbo`), soit des joueurs "Free to play" (`JoueurFTP`).
- la classe `Serveur`. Un serveur est unique et il gère les tournois. Il s'occupe de créer un tournoi, d'enregistrer des joueurs dans un tournoi, de lancer un tournoi (quand celui-ci contient suffisamment de joueurs) et de le clôturer quand celui-ci est terminé.
- la classe `Tournoi` qui est dédiée à l'organisation d'un tournoi entre joueurs. Les joueurs y sont inscrits par le serveur, puis le tournoi est lancé pour désigner le joueur vainqueur du tournoi.

Question 2.1 (4 points) Écrire les classes suivantes :

La classe `Joueur` qui possède un unique constructeur avec un argument donnant le `nom` du joueur (String) et un assesseur (getter) permettant d'obtenir ce nom. Elle possède un attribut `serveur` qui contient la référence d'un `Serveur`. Cette variable est initialisée à `null`. Ajouter une méthode `enregistrement()` qui prend un argument pour donner une valeur à cet attribut et ne rend rien. Cette classe possède aussi une méthode `statut()` qui rend un booléen : `true` si le joueur est abonné, `false` sinon. Cette classe ne doit pas être instanciable.

La classe `JoueurAbo` qui possède un attribut `nbPoints` (entier) donnant le nombre de points que le joueur a gagné lors des tournois auxquels il a participé, un constructeur à 2 arguments (nom et nombre de points de tournoi initiaux), un constructeur avec un seul argument (nom, le nombre de points étant alors initialisé à 0), un assesseur (getter) fournissant le nombre de points obtenus par le joueur et une méthode `addPoints()` qui prend un argument un entier qui est ajouté au nombre de points du joueur.

La classe `JoueurFTP`, sans attribut mais avec une variable permettant de compter le nombre de joueurs FTP créés. Le nom d'un tel joueur est créé automatiquement de la forme "j1", "j2",... où le nombre correspond au numéro de création du joueur. Cette classe ne contient qu'un constructeur sans argument et une méthode pour connaître le nombre de joueurs FTP qui ont été créés.

Classe Joueur :
- 1 point pour la bonne définition et retirer 0.5pt si pas abstraite
- 0.5 pt pour la méthode statut() abstraite (et correctement définie, 0 sinon)

— 0.5 pt pour enregistrement()

Classe JouerAbo : 1pt pour la classe. Retirer 0.25pt si pas d'utilisation de this() ou super() dans le 2e constructeur.

Classe JoueurFTP : 1pt pour la classe avec la bonne gestion du compteur (static cpt, etc.) et la bonne génération du nom (avec appel à super()).

Retirer 0.25pt si le premier nom généré est "j0" au lieu de "j1"

Rappel : `super()` doit apparaître forcément en première position.

Remarque : la solution `return this instanceof JoueurAbo` pour statut() dans la classe Joueur n'est pas convenable car cela implique de devoir modifier cette fonction si on rajoute par la suite une nouvelle classe de joueurs, non dérivée de JoueurAbo, et qui pourraient avoir aussi le statut d'abonné. Ce n'est donc pas "propre".

```
1  public abstract class Joueur {
2          private String nom;
3          private Serveur serveur;
4          public Joueur(String nom) {
5                  this.nom = nom;
6          }
7          public String getNom() {
8                  return nom;
9          }
10         public void enregistrement(Serveur s) {
11                 this.serveur = s;
12         }
13         public abstract boolean statut();
14 }
15 // ---------------------------------------------
16 public class JoueurAbo extends Joueur {
17         private int nbPoints;
18
19         public JoueurAbo(String nom, int nbPoints) {
20                 super(nom);
21                 this.nbPoints = nbPoints;
22         }
23         public JoueurAbo(String nom) {
24                 this(nom,0);
25         }
26         public void addPoints(int nb) {
27                 nbPoints+=nb;
28         }
29         public boolean statut() {
30                 return true;
31         }
32 }
33 // ---------------------------------------------
34 public class JoueurFTP extends Joueur {
35         private static int cpt =0;
36         public JoueurFTP() {
37                 super("j"+(++cpt));
38         }
39         public static int getCpt() {
40                 return cpt;
41         }
42         public boolean statut() {
43                 return false;
44         }
45 }
```

Question 2.2 (5 points) On souhaite doter les 3 classes précédentes d'une capacité de clonage mais, à cause de la classe `Joueur` qui n'est pas instanciable, il n'est pas possible de définir une méthode `clone()`. On utilise donc des constructeurs par recopie. Donner le code des constructeurs par recopie pour chacune des classes `Joueur`, `JoueurAbo`, et `JoueurFTP`.

2pts pour la classe Joueur

2pts pour la classe JouerAbo

1pt pour la classe JoueurFTP

Ne mettre que la moitié des points pour les classes JoueurAbo et JoueurFTP s'il n'y a pas d'appel à super().

dans `Joueur` :

```
1         public Joueur(Joueur j) {
2                 this(j.nom);
3                 this.serveur = j.serveur;
4         }
```

dans `JoueurAbo` :

```
1         public JoueurAbo(JoueurAbo j) {
2                 super(j);
3                 this.nbPoints = j.nbPoints;
4         }
```

dans `JoueurFTP` :

```
1  public JoueurFTP(JoueurFTP j) {
2                 super(j);
3         }
```

Question 2.3 (4 points) Écrire la classe `Tournoi` contenant 2 constantes entières `MAXJOUEURS` qui donnent le nombre maximum de joueurs pouvant participer à un tournoi (tous les tournois ont le même maximum), initialisée à la valeur 12, et `PTSVAINQUEUR` qui donne le nombre de points de tournoi affecté au vainqueur d'un tournoi, initialisée à 10. Ces 2 constantes sont valables pour tout tournoi créé.

Cette classe contient 4 attributs : un entier `ident` qui est l'identifiant du tournoi créé. La valeur de cet identifiant est donné en argument de l'unique constructeur de cette classe ; un tableau de `Joueur` de nom `tabJoueurs` ; un entier `nbJoueurs` qui donne le nombre de joueurs inscrits dans le tournoi, et qui vaut donc 0 au départ ; une variable `vainqueur` de type `Joueur` qui contiendra le joueur qui aura gagné le tournoi à l'issue de son lancement, au départ cette variable contient la valeur `null`.

Pour compléter, la classe contient 2 accesseurs (getter) pour obtenir l'identifiant et le nombre de joueurs du tournoi. Elle contient aussi une méthode `ajoute` qui prend un joueur en argument et l'ajoute dans le tableau des joueurs inscrits au tournoi, s'il reste encore de la place dans ce tournoi (ie. le nombre de joueurs maximum n'a pas été atteint).

1 point pour la définition correcte des constantes (retirer 0.5pt par constante mal définie).

0.5 point pour les attributs corrects.

0.5 point pour les 2 getters (0.25 chaque, et doit être parfait sinon 0).

1 point pour le constructeur.

1 point pour la bonne définition de ajoute().

```
1  public class Tournoi {
2         private static final int MAXJOUEURS = 12;
3         private static final int PTSVAINQUEUR = 10;
4         private int ident;
```

```
5         private Joueur[] tabJoueurs;
6         private int nbJoueurs;
7         private Joueur vainqueur;
8
9         public Tournoi(int n) {
10                ident = n;
11                tabJoueurs = new Joueur[MAXJOUEURS];
12                nbJoueurs = 0;
13                vainqueur = null;
14        }
15
16        public int getIdent() { return ident; }
17        public int getNbJoueurs() {return nbJoueurs;}
18
19        public void ajoute(Joueur j) {
20                if (nbJoueurs < tabJoueurs.length) {
21                        tabJoueurs[nbJoueurs++] = j;
22                }
23        }
```

Question 2.4 (4 points) Donner le code de la méthode `clone()` qui permet de cloner un tournoi.

Attention : on rappelle qu'un `Joueur` n'est pas instanciable.

Remarque : il n'est pas possible d'écrire `new Joueur(this.tabJoueurs[i]))` car le compilateur refuse car la classe Joueur est abstraite (d'où le "Attention"...).

Remarque 2 : il faut considérer un clonage "profond" et cloner aussi les Joueurs. Ce n'est pas forcément clair mais le "Attention" devait attirer l'attention sur le fait qu'il fallait sûrement faire quelque chose. Donner au total pour cette question au maximum 2pts à une solution de clonage qui ne clone pas les Joueurs.

1 point pour les 3 premières instructions.

3 points pour le clonage des joueurs répartis ainsi :
- — 0.5 point pour la boucle
- — 0.5 point pour le if avec statut
- — 1 point pour chaque appel du constructeur par recopie de chacune des 2 classes JoueurAbo et JoueurFTP. Ne mettre que la moitié des points s'il n'y a pas tous les cast.

```
1         public Tournoi clone() {
2                 Tournoi t = new Tournoi(this.ident);
3                 t.tabJoueurs = new Joueur[MAXJOUEURS];
4                 t.nbJoueurs = this.nbJoueurs;
5                 t.vainqueur = null;
6
7                 for (int i=0; i<this.nbJoueurs;i++) {
8                         if (this.tabJoueurs[i].statut())
9                                 t.tabJoueurs[i] = new JoueurAbo((JoueurAbo)(this
                                      .tabJoueurs[i]));
10                        else
11                                t.tabJoueurs[i] = new JoueurFTP((JoueurFTP)this.
                                      tabJoueurs[i]);
12                        // Si jamais il y a un vainqueur de tournoi:
13                        if (this.vainqueur == this.tabJoueurs[i])
14                                t.vainqueur = t.tabJoueurs[i];
15                }
16                return t;
17        }
```

Question 2.5 (4 points) Écrire la classe `Serveur`. Cette classe doit garantir qu'il ne peut exister qu'une unique instance de serveur dans le programme. Elle contient un attribut qui est un `ArrayList` de tournois

de nom `tabTournois` ; une méthode `getInstance()`, sans argument, qui rend la référence du serveur ; une méthode `creeTournoi()` sans argument qui créé un nouveau tournoi et l'ajoute à `tabTournois`. L'identifiant qui est donné au tournoi créé correspond au nombre de tournois qui ont été créés depuis le lancement du programme.

On retrouve ici notre singleton... :-)

2pt pour la bonne gestion du singleton (une instance et une seule).

1pt pour la bonne définition et initialisation de l'ArrayList.

1pt pour la méthode creeTournoi().

```
1  import java.util.ArrayList;
2  public class Serveur {
3          private static final Serveur INSTANCE = new Serveur();
4          private static int nbCrees = 0;
5          private ArrayList<Tournoi> tabTournois;
6          private Serveur() {
7                  tabTournois = new ArrayList<Tournoi>();
8          }
9          public static Serveur getInstance() {
10                 return INSTANCE;
11         }
12         public void creeTournoi() {
13                 tabTournois.add(new Tournoi(nbCrees++));
14         }
15 }
```

Question 2.6 (2 points) Pour qu'un joueur s'inscrive dans un tournoi, donner le code de la méthode `inscription()` de la classe `Serveur`. Cette méthode prend en argument un joueur et l'inscrit dans un des tournois de `tabTournois`. Le choix du tournoi est fait de façon aléatoire.

0.25pt pour la bonne signature de la fonction.

1 point pour la bonne définition de la valeur aléatoire (avec size()). Retirer 0.5pt si ce n'est pas size() qui est utilisé.

0.5pt pour l'appel de get().

0.25pt pour ajoute(j).

```
1          public void inscription(Joueur j) {
2                  int alea = (int)(Math.random()*tabTournois.size());
3                  tabTournois.get(alea).ajoute(j);
4          }
```

Question 2.7 (2 points) Donner le code de la méthode `inscritTournoi()`, sans argument, de la classe `Joueur` qui permet à un joueur de demander au serveur de l'inscrire dans un tournoi.

2 points si tout est parfait, aucun point dans le cas contraire.

Moitié des points seulement si pas de test sur la nullité de serveur (solution 1).

```
1          public void inscritTournoi() {
2                  if (serveur != null)
3                          serveur.inscription(this);
4          }
```

une solution alternative qui utilise le fait que le serveur est un singleton :

```
1          public void inscritTournoi() {
2                  Serveur.getInstance().inscription(this);
3          }
```

Question 2.8 (3 points) Donner le code de la méthode `lance()`, sans argument, de la classe `Tournoi`. Cette méthode simule (très simplement) le déroulé du tournoi : le vainqueur est choisi aléatoirement parmi les joueurs qui sont inscrits dans ce tournoi. Ce joueur est alors affecté à l'attribut `vainqueur`. De plus, si le vainqueur est un joueur abonné, il faut ajouter `PTSVAINQUEUR` à son nombre de points.

Remarque : s'il existe déjà un vainqueur, ou s'il n'y a aucun joueur inscrit, cette méthode ne fait rien.

0.5 pt pour la bonne valeur aléatoire

0.5pt pour la vérification que le vainqueur n'est pas connu.

0.5pt pour le bon choix d'un gagnant.

1.5 point pour une bonne gestion du vainqueur (l'ajout des points gagnés ne se fait que si le vainqueur est un JoueurAbo). Enlever 0.75pt si vainqueur n'est pas converti en JoueurAbo par un cast (ou équivalent).

```
1  public void lance() {
2      if (nbJoueurs >0) {
3          if (vainqueur == null) {
4              int alea = (int)(Math.random()*nbJoueurs);
5              vainqueur = tabJoueurs[alea];
6              if (vainqueur.statut()) {
7                  JoueurAbo j = (JoueurAbo)(vainqueur);
8                  j.addPoints(PTSVAINQUEUR);
9              }
10         }
11     }
12 }
```

Question 2.9 (3 points) Donner le code de la méthode `lanceTournoi()` de la classe `serveur`. Cette méthode prend en argument l'**identifiant d'un tournoi**. Le tournoi demandé, s'il existe dans `tabTournois`, est lancé. Une fois que ce tournoi s'est terminé, il est alors supprimé de la liste des tournois du serveur.

Remarque : il faut faire attention à ce que le idTournoi ne soit pas utilisé comme indice dans l'ArrayList !

0.5pt pour la signature de la méthode

1.5pt pour le parcours de l'ArrayList pour trouver letournoi demandé.

0.5pt pour l'appel de lance() sur le tournoi trouvé SI on l'a trouvé (pas de point sinon).

0.5pt pour le remove() à l'issue du lancement.

```
1  public void lanceTournoi (int idTournoi) {
2    int i= 0;
3    while ((i<tabTournois.size())&&(tabTournois.get(i).getIdent() != idTournoi)) {
4        i++;
5    }
6    if (i<tabTournois.size()) {
7        tabTournois.get(i).lance();
8        tabTournois.remove(i);
9    }
10 }
```

Question 2.10 (7 points) Afin de gérer la saturation d'un tournoi (nombre de joueurs maximum atteint), on choisit d'utiliser une solution à base d'exception.

1. Donner le code de la classe `PlusDePlaceException` pour créer une exception. Cette classe ne contient qu'un unique constructeur qui prend en argument une chaîne de caractères.
2. Donner une nouvelle version de la méthode `ajoute()` afin qu'elle lève une exception de type

`PlusDePlaceException` dans le cas où l'on demande d'ajouter un joueur dans le tournoi qui est déjà au maximum. Le message associé à cette levée d'exception doit être "Tournoi X plein !" où X est l'identifiant du tournoi.

3. Donner une nouvelle version de la méthode `inscritTournoi()` de la classe `Serveur`. Cette méthode doit récupérer l'exception `PlusDePlaceException` susceptible de se produire. Si tel est le cas, un nouveau tournoi est créé par le serveur et une demande d'inscription est relancée pour le joueur.

Remarque : l'énoncé contient un bug dans le 3e item : ce n'est pas `inscritTournoi()` de la classe `Serveur` mais `inscription()` qu'il faut modifier. On a signalé ça dans les amphis.

1.5pt pour la définition correcte de la classe PlusDePlaceException : 1pt pour le constructeur et 0.5pt pour l'entête de la classe.

2.5pts pour la méthode ajoute() de Tournoi :
- mettre 1.5pt pour l'ajout de throws dans la signature ;
- mettre 1pt pour la levée de l'exception correctement faite.

3pts pour la méthode inscription() :
- 1pt pour la bonne structure try...catch ;
- 0.5pt pour le bon catch() ;
- 1.5pt pour la bonne gestion : création de tournoi et nouvel appel à inscription(). La solution n'est pas nécessairement un appel récursif, cela peut se faire par une boucle ou autre.

Retirer 1pt si en plus du try...catch un "throws PlusDePlaceException" a été ajouté à la signature de la méthode inscription().

```java
public class PlusDePlaceException extends Exception {

        public PlusDePlaceException(String s) {
                super(s);
        }
}
```

dans `Tournoi` :

```java
public void ajoute(Joueur j) throws PlusDePlaceException {
                if (nbJoueurs < tabJoueurs.length) {
                        tabJoueurs[nbJoueurs++] = j;
                }
                else
                        throw new PlusDePlaceException("Tournoi "+ident+" plein
                            !");
        }
```

dans `serveur` :

```java
        public void inscription(Joueur j) {
                int alea = (int)(Math.random()*tabTournois.size());
                try {
                        tabTournois.get(alea).ajoute(j);
                }
                catch (PlusDePlaceException e) {
                        this.creeTournoi();
                        this.inscription(j);
                }

        }
```

Question 2.11 (2 points) Donner le schéma UML **fournisseur** des classes réalisées.

1pt pour la bonne mention de tout ce qui doit être indiqué dans un schéma fournisseur : tous les attributs, méthode et leur visibilité. Mettre 0pt en cas d'oubli systématique.

1pt pour les bonnes flèches (bon sens et bonne tête de flèche).