Programmation Objet II

Georges Côté

420-2CP-BB - Hiver 2021

TP3 v1.0

Objectifs du travail

- Familiarisation avec les principes plus avancés de la programmation (classes, héritage, polymorphisme, structure de données, tri, test)
- Documentation.
- Réalisation d'un travail d'une certaine envergure de façon structuré

Dates de remise

Groupe 01 : Le 18 mai à minuit
Groupe 02 : Le 18 mai à minuit

Le projet au complet compressé remis à travers Léa. Le projet vaut pour 30% de la note finale. Des pénalités s'appliquent en cas de retard. Vous pouvez utiliser IntelliJ ou NetBeans.

Quand vous remettez votre projet dans Léa, compressez-le en **ZIP**. Le format .RAR n'est pas supporté par Léa et je dois faire quelques manipulations pour l'extraire correctement.

Remise en .RAR ou en .7z : -10% pénalité

Remise des fichiers .JAVA seulement : -15% pénalité si vous ne remettez que les fichiers .java

Remise du projet au complet dans le bon format : **0**% pénalité

Yahtzee (30%)

J'espère que vous connaissez le jeu « Yahtzee ». Ce n'est pas compliqué et c'est un beau petit projet.

Règles du jeu

Voici une page Wikipédia qui explique les règles du jeu qui se joue à plusieurs joueurs à l'aide de cinq dés.

Chaque joueur essaie de former des figures avec les cinq dés. Une fois qu'une figure est choisie, elle ne peut pas être choisie à nouveau.

Combinaison	Points	Exemple
As	1 × le nombre de dés 1 obtenus	1-1-2-4-5 donne 2x1
Deux	2 × le nombre de dés 2 obtenus	1-2-2-2-4 donne 3x2
Trois	3 × le nombre de dés 3 obtenus	1-3-4-5-6 donne 1x3
Quatre	4 × le nombre de dés 4 obtenus	1-2-3-5-6 donne 0x4
Cinq	5 × le nombre de dés 5 obtenus	1-2-5-5-6 donne 2x5
Six	6 × le nombre de dés 6 obtenus	5-6-6-6-6 donne 4x6
Sous-total	**Somme des points obtenus ci-dessus**	
Prime	35 points si le sous-total est supérieur ou égal à 63 points ou 0	
Total I	**Somme du sous-total et de l'éventuelle prime**	

Combinaison	Description	Points
Brelan	Trois dés identiques par exemple 2-2-2-4-5	3 fois la valeur des dés identiques
Carré	Quatre dés identiques par exemple 4-4-4-4-6	4 fois la valeur des dés identiques
Full	Trois dés identiques + deux dés identiques par exemple 2-2-2-3-3	25 points
Petite suite	1,2,3,4; 2,3,4,5 ou 3,4,5,6	30 points
Grande suite	1,2,3,4,5 ou 2,3,4,5,6	40 points
Yahtzee	Cinq dés identiques	50 points
Chance	Rien de spécial	Somme de tous les dés

Pour résumer très rapidement, chaque joueur joue à tour de rôle. Chaque joueur a jusqu'à trois essais par tour pour essayer de compléter une combinaison.

Premier essai : le joueur lance les cinq dés. Dépendant du résultat, il choisit quels dés il garde et quels dés il lance à nouveau.

Par exemple, un joueur lance les cinq dés.

2-2-3-4-4.

Il a la possibilité de compléter la combinaison « Full » (à moins qu'elle a déjà été complétée). Il garde 2-2-4-4 et lance le dernier dé. S'il obtient un 2 ou un 4 au deuxième essai, il choisit la « Full » et obtient 25 points. S'il n'obtient pas une de ces deux valeurs, il peut essayer une troisième fois. Si jamais il n'obtient

pas un 2 ou un 4 au troisième essai, il peut choisir une autre combinaison (par exemple les Deux et obtenir 4 points quand même).

Une fois que la figure a été choisie, elle n'est plus disponible.

Design

Le développement de logiciel demande une bonne planification. Avant de s'attaquer à l'interface, nous devons nous assurer que notre code fonctionne bien.

Vous devez écrire une classe (disons YahtzeeEvaluation) qui évalue les résultats des 5 dés. Et vous devez valider chaque cas possible.

Par exemple, vous avez le cas du Brelan (3 dés de même valeur). Si vos dés sont 3-2-3-4-3, le joueur obtient 3 x la valeur des dés identiques. Si vos dés sont 3-2-3-4-4, la méthode retourne 0.

Tests (50% du TP)

Vous devez lire un fichier qui contient une série de tests. La classe qui évalue les cinq dés par rapport à la combinaison et doit retourner le bon résultat.

```
4 1 2 3 5 Quatre 4
5 5 5 1 2 Cinq 15
6 6 1 6 3 Six 18
1 2 3 4 4 Six 0
3 2 3 3 2 Brelan 9
3 2 1 3 1 Brelan 0
3 2 1 3 1 Carre 0
3 3 3 1 3 Carre 12
3 3 3 3 3 Carre 12
3 2 3 2 1 Full 0
3 2 3 2 3 Full 25
```

Chaque ligne contient cinq valeurs entre 1 et 6, une combinaison et un résultat.

```
3 2 3 3 2 Brelan 9
```

Pour cette ligne, la valeur des dés est 3 2 3 3 2, la combinaison est Brelan et le test doit donner 9 comme résultat.

Pour ces tests, je vous impose trois choses.

1) Pour représenter chacune des cinq valeurs, vous devez créer une classe qui hérite de ma classe « Valeur »

```
public class Valeur {
    protected int valeur;

    public int get() {
        return valeur;
    }
}
```

2) Les méthodes qui servent à tester chacun des cas doivent avoir un paramètre de type ArrayList<Valeur>.
   a. `ArrayList<Valeur> liste`.
   b. La liste contient en fait cinq instances d'une sous-classe de « Valeur ».
3) Vous devez toujours trier la liste en ordre croissant avant d'évaluer les dés. Nous avons vu (ou nous allons voir) trois algorithmes de tri (tri par sélection, tri à bulle et tri par insertion). Vous devez coder les trois algorithmes. À chaque appel de tri, le code choisit aléatoirement un des trois algorithmes.
   a. Au moment de la remise de cet énoncé, vous n'aurez probablement pas vu le tri. Pendant le développement, vous pouvez utiliser quelque chose comme ceci:

```
private static void tri(ArrayList<Valeur> liste) {
    liste.sort((valeur1, valeur2) -> {
        if (valeur1.get() == valeur2.get())
            return 0;
        if (valeur1.get() > valeur2.get())
            return 1;
        return -1;
    });
    System.out.println(liste);
}
```

   ou chercher pour Collections.sort. Mais, lors de la remise finale, vous devez utiliser votre propre code de tri.

Pour la sous-classe qui hérite de la classe « Valeur », elle reçoit un entier qui provient du fichier et elle l'affecte à la variable d'instance.

[Suggestion] Il est possible d'avoir deux listes qui contiennent les mêmes instances mais dont l'une est

```
ArrayList<Valeur>
```

Et l'autre est

```
ArrayList<ClasseQuiHériteDeValeur>
```

Dans le document sur les ArrayList, il est mentionné qu'il n'est pas possible d'avoir une liste de « SousClasse » et d'appeler une méthode qui s'attend à recevoir une liste de « SuperClasse ».

Par exemple, il n'est pas possible d'écrire

```
ArrayList<SousClasse> liste = new ArrayList<>();
appellerMethode(liste);
```

si la signature de la méthode est :

```
appellerMethode(ArrayList<SuperClasse> liste)
```

Il faut que la signature soit quelque chose comme

```
appellerMethode(ArrayList liste)
```

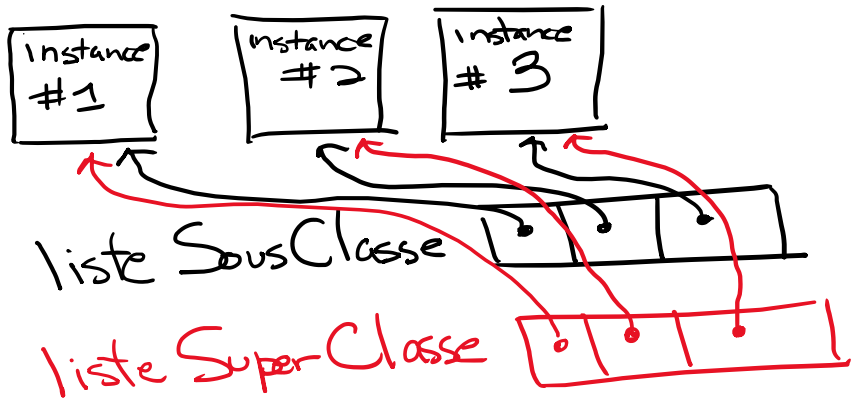
Une façon de contourner cette limitation est d'avoir deux listes.

```
ArrayList<SousClasse> listeSousClasse = new ArrayList<>();
ArrayList<SuperClasse> listeSuperClasse = new ArrayList<>();
```

Chaque instance est ajoutée aux deux listes. On travaille avec la liste « listeSousClasse » mais quand il est le temps d'appeler la méthode appellerMethode(ArrayList<SuperClasse> liste), on l'appelle en utilisant

```
appellerMethode(listeSuperClasse);
```

En réalité, les deux listes contiennent exactement les mêmes instances.



Donc, si on revient aux tests, idéalement, vous créez une classe de test (disons YahtzeeTest) qui utilise la classe (disons YahtzeeEvaluation) pour évaluer les cas de Yahtzee pour chaque ligne. Dans le main, vous appelez votre code qui lit le fichier « test.txt ».

```
public static void main(String[] args) {

    if (YahtzeeTest.valider( fichier: "test.txt")) {
        System.out.println("Tout est beau");
    } else {
        System.out.println("Ah non");
    }
```

Le jeu (50%)

Pour le jeu même, vous faites votre propre interface mais elle doit être facile d'utilisation (conviviale). Le jeu doit avoir de deux à quatre joueurs. Vous pouvez demander le nom de chaque joueur ou vous pouvez utiliser « Joueur 1 », « Joueur 2 », etc.

La gestion des joueurs se fait grâce à un **ArrayList**.

Je vous impose quelques petits trucs. Vous allez avoir besoin d'une classe « Dé » qui permet de choisir une valeur entre 1 et 6 au hasard. Cette classe **doit** hériter de la classe « Valeur ».

Vous devez utiliser la même classe (disons YahtzeeEvaluation) qui reçoit un ArrayList<Valeur> pour le jeu aussi.

Lors de la remise, assurez-vous que votre code n'appelle pas List.sort, Collections.sort ni Collections.swap.

Vous devez garder les statistiques de chacun des joueurs et les afficher à chaque tour.

Si une combinaison a déjà été utilisée (par exemple la petite suite) et que l'utilisateur choisit cette combinaison-là, vous devez redemander.

Documentation

Vous n'avez pas besoin d'écrire du Javadoc mais vous devez écrire un petit document (à l'allure professionnel) qui explique comment jouer (pas les règles mais comment utiliser l'interface). Vous pouvez remettre un fichier pdf.

Restrictions sur les instructions

Essayez de vous limiter aux instructions vues en classe. Si vous voulez utiliser des concepts plus avancés, confirmez avec moi si c'est acceptable.

Je m'attends quand même à des commentaires dans le code même si JavaDoc n'est pas requis.

Critères d'évaluation

Votre programme doit respecter les normes de programmation. Vous devez réaliser ce travail **seul**. Si vous aidez un autre étudiant, donnez-lui des conseils, expliquez-lui des concepts mais ne partagez pas votre code.

Veuillez formater les résultats correctement.

Ça prend des classes. Le faire de façon procédurale n'est pas acceptable.

Utiliser les try/catch correctement. Je vais essayer de faire planter votre programme. Si je réussis, vous allez perdre des points.

Critère	Nombre de points accordés
Programmation ▪ Clarté, simplicité ▪ Paramètres de méthodes bien choisis ▪ Conception Orientée Objet ▪ Respect des normes	/20
Documentation	/3
Fonctionnement du programme ▪ Résultats obtenus ▪ Affichage des résultats ▪ Respect des consignes	/7
Total	**/30**

Autres suggestions

Vous devriez ré-utiliser la classe Saisie ici. Aucune obligation…

Ceci est un bel exemple pour l'utilisation des enums.

Un petit truc pour associer une chaine à un enum.

En écrivant votre enum comme ceci :

```
public enum MonEnum {
    VALEUR1( s: "Valeur1"),
    VALEUR2( s: "Valeur2");

    private final String nom;

    private MonEnum(String s) {
        nom = s;
    }

    public String toString() {
        return this.nom;
    }
}
```

Quand vous écrivez

```
MonEnum monEnum = MonEnum.VALEUR1;
System.out.println(monEnum);
```

Au lieu d'avoir VALEUR1, la chaine associée avec cet enum est affichée.

```
Valeur1
```

Un enum, c'est vraiment similaire à une classe.

Et si vous nommez vos enums correctement, vous devriez être capable de convertir la combinaison qui se trouve dans le fichier directement en un enum en utilisant MonEnum.valueOf(…).

```
3 2 3 3 2 Brelan 9
```

Habituellement, les enums sont tous en majuscule mais un petit chaine.toUpperCase() devrait vous aider avec ça.

Pour l'interface, c'est un peu difficile en utilisant que le clavier. Voici comment j'ai fait pour spécifier quels dés je voulais rouler et quels dés je voulais garder. Un petit x, je garde, un espace, je roule à nouveau. Un « xxxxx » pour terminer le tour avant le troisième essai.

```
[4, 6, 1, 4, 6]
Entrez un x pour chaque dé que vous voulez garder: xx xx
[4, 6, 1, 4, 6]
Entrez un x pour chaque dé que vous voulez garder: xx xx
[4, 6, 2, 4, 6]
```

J'ai aussi affiché les résultats de façon horizontale sinon c'est un peu illisible.

```
|   1   |   2   |   3   |   4   |   5   |   6   | Bonus | Brelan |  Carré |  Full  | Petite | Grande |   Yam  | Chance | total  |
|       |       |       |       |       |       |   0   |        |        |        |        |        |        |   22   |   22   |
```

Cas spéciaux

J'imagine que ces cas-là sont possibles.

Combinaison	Possibilités
Brelan	Si vous obtenez 4 ou 5 valeurs identiques, c'est quand même 3x
Carré	Si vous obtenez 5 valeurs identiques, c'est quand même 4x
Full	5 valeurs identiques respectent cette combinaison 5-5-5 et 5-5

Youtube

Voici un lien vers une démonstration de mon programme : https://youtu.be/Ex9dQMpTaT0