Les exceptions

Objectif(s)

✯ lancer, gérer et exploiter des exceptions dans des contextes que vous avez déjà rencontrés

Exercice 1 – Jeu de Bataille - v3.0

D'abord, nous rendons le jeu de bataille plus facile : au lieu de toujours utiliser un jeu de 32 cartes, nous allons aussi utiliser des jeux de 52 cartes.

— Heureusement, nous avons déjà l'énumération Hauteurs que nous pouvons facilement étendre.

— Pour bien gérer les paquets de cartes, nous ajoutons un attribut qui stocke la taille maximale du paquet. Cet attribut sera initialisé avec la une valeur passée au constructeur de la classe.

Si la valeur est différente de 32 ou 52, le constructeur lève une `IllegalAttributeException` indiquant le problème.

Après avoir modélisé le jeu de bataille, plusieurs problèmes peuvent se produire au cours de l'exécution :

— un joueur qui n'a plus de cartes pour continuer le jeu ;

— quelqu'un peut essayer d'ajouter une carte à un paquet de cartes qui ne peut pas être incluse.

Afin de pouvoir réagir à ces cas et rattraper toutes les erreurs d'exécution sans sortie brusque du programme, nous allons définir :

1. deux nouvelles classes d'exceptions personnalisées nommées :
   — la classe `ExceptionDeTasVide` qui, quant à elle, hérite de `Exception`. Cette exception a un attribut de type `Joueur` qui stocke la référence du joueur dont le tas est vide et qui est initialisé dans le constructeur.
   — `ExceptionDeCarteInvalide` qui hérite de la classe `IllegalAttributeException`.

   Le message de ces exceptions doit décrire la raison pour laquelle l'exécution a échoué.
2. Tester le cas d'un tas vide :
   — Modifier dans la classe `PaquetCartes`, la méthode `Carte tirerCarteDessus()` de telle sorte de lever l'exception du tas vide ;
   — Dans une classe `TestExceptions`, créer un jeu de 52 cartes, remplissez-le, puis le battre 5 fois. Tester l'exception de tas vide en essayant de retirer les n premières cartes comme suit (la saisie sera faite en utilisant une instance de la classe `Scanner`) :

```
[52 cartes restantes]. Saisir le nombre des cartes à retirer du jeu (0
pour quitter): 2
2 cartes retirées:
Dame de Carreau- 8 de Carreau-

[50 cartes restantes]. Saisir le nombre des cartes à retirer du jeu (0
pour quitter): 10

10 cartes retirées:
Roi de Trèfle- 10 de Pique- Roi de Pique- 4 de Pique- 2 de Trèfle- As
de Trèfle- 3 de Coeur- Roi de Coeur- 5 de Coeur- 8 de Coeur-

[40 cartes restantes]. Saisir le nombre des cartes à retirer du jeu (0
pour quitter): 45
```

`ExceptionDeTasVide: Attention, Vous demandez un nombre > à ce qui est existe dans le tas!!!`

Retirer les cartes est fait en créant un paquet de carte vide qui est rempli au fur et à mesure avec les cartes récupérées avec `tirerCarteDessus()`

— Exploiter la levée de l'exception afin d'exiger l'utilisateur de saisir un n inférieur au nombre de cartes dans le tas.

Attention : N'oubliez pas de remettre les cartes dans le paquet original, si on ne peut pas en rétirées n.

3. Cas d'une carte invalide :
   — Modifier la méthode `ajouterCarte` et la méthode `mettreCarteDessous` de la classe `PaquetCartes` afin de vérifier :
     — Que la carte en train d'être ajoutée n'est pas déjà incluse dans le paquet. **Attention :** vous allez avoir besoin d'une implémentation de la méthode `equals()` pour la classe `Carte`.
     — Que la carte en train d'être ajoutée n'a pas d'hauteur `DEUX` – `SIX` pour un paquet de 32 cartes.

   Dans les deux cas, la méthode doit lever une `ExceptionDeCarteInvalide` ayant de message approprié.
   — Dans la classe `TestExceptions`, créer un paquet de 32 cartes, remplissez-le et essayer d'ajouter une deux de cœur. Attraper l'exception afficher un message qui indique le problème.
   — Dans la classe `TestExceptions`, créer un paquet de 52 cartes, remplissez et essayer d'ajouter une deux de trefle un utilisant la méthode `mettreCarteDessous`. Attraper l'exception afficher un message qui indique le problème.
4. Exploiter l'exception au cas du tas vide :
   — Apporter les modifications nécessaires à la méthode `Carte prendreCarteDessus()` de la classe Joueur

   Nous allons exploiter l'`ExceptionDeTasVide` de deux manières différentes afin de gérer le déroulement d'une partie :
   (a) Pour la première, dans `jouer1Tour()`, on ne teste pas s'il reste de carte dans le tas d'un joueur. Si une `ExceptionDeTasVide` arrive pendant le tirage de cartes, on l'attrape et en fonction du joueur stocké dans l'exception on retourne le joueur gagnant.
   (b) Pour la deuxième, `jouer1Tour()`, on on ne teste pas s'il reste de carte dans le tas d'un joueur, mais on n'attrape pas l'exception non plus. On permet à `jouer1Tour()` de propager l'exception, on l'attrape dans la méthode `jouer()` et on exploite l'information par rapport au joueur afin de savoir qui a gagné. **Attention :** la partie peut toujours finir si 100 tours se sont passés.

Exercice 2 – Fichier I/O

On veut écrire un programme qui lit un fichier qui contient une liste d'étudiant. Chaque étudiant est décrit par son nom de famille, le nombre d'heures de cours par semestre et le nombre de crédit accumulé, séparé par des espaces.

Le nom du fichier à lire va être passé au programme comme paramètre à la ligne de commande pendant l'appel du programme.

1. Créez une classe `LireEtudiants` qui ouvre un `BufferedReader` en utilisant un `Path` créé à partir du nom de fichier passé en paramètre.
2. Téléchargez le fichier `etudiants.dat` et lancer le programme sur ce fichier.
3. Créez une classe `Etudiant` qui a trois attribut : `nom`, une chaîne de caractère, `nbrDHeures`, un entier, et `credits`, un double.
4. Le constructeur de cette classe prend une chaîne de caractère en paramètre, la coupe en plusieurs parties en utilisant la méthode `split`. Le paramètre passée à `split` sera une chaîne de caractère contenant un seul espace.

   Les trois parties résultant sont censés être le nom, le nombre d'heures et le nombre de crédit. Les deux derniers doivent être traduit de chaîne de caractère à entier/double en utilisant les méthodes `parseInt`/`parseDouble` des classes `Integer`/`Double`, respectively.

5. Dans la classe `LireEtudiants`, lisez le `BufferedReader` ligne par ligne, créez des étudiants et ajoutez-les à une liste d'étudiants.
6. Créez un `BufferedWriter` dans la classe `LireEtudiants`. Le nom du fichier sera le deuxième paramètre passé pendant l'appel du programme.
7. Appelez le programme en utilisant `java LireEtudiants etudiants.dat avertissement.txt`.
8. Nous voulons calculer la note moyenne — le nombre de crédit divisé par le nombre d'heures — et écrire les données des étudiants pour lesquels cette note est inférieure à 3.0 dans un fichier.
   On ajoute une méthode `noteMoyenne` à la classe `Etudiant` qui fait ce calcul.
9. Écrivez l'en-tête "Nom NoteMoyenne" dans le fichier ouvert avec le `BufferedWriter`.
10. Parcourez la liste d'étudiants, calculez leur note moyenne et écrivez leurs informations dans le fichier si la note moyenne < 3.0.
11. Fermez le `BufferedReader`/`BufferedWriter`.

Exercice 3 – Gérer les exceptions pendant le traitment des fichiers

Jusqu'ici, tout c'est bien passé. Maintenant, nous allons voir des problèmes possibles et gérer les exceptions qui peuvent arriver.

1. Lancez le programme en appelant `java LireEtudiants nomdefichier avertissement.txt`. Prenez note de l'exception qui est levée et ajouter un `catch` qui gère cette exception en affichant un message qui indique le problème et arrête le programme avec un appel à la méthode `exit` de la classe `System`. **N'affichez pas** le "stacktrace" !
2. Téléchargez le fichier `etudiantsCorrompus.dat`. Lancer le programme en appelant `java LireEtudiants etudiantsCorrompus.dat avertissement.txt`. Notez l'exception qui arrive. On va gérer cette exception où elle arrive (dans le constructeur) ou dans la méthode qui appelle le constructeur ?
3. Ajoutez le code pour attraper l'exception, indiquer à quelle ligne du fichier elle est arrivée et ignorer l'étudiant pour lequel elle arrive.
4. Changez les permissions du fichier `avertissement.txt` pour qu'il est lecture-seul. Notez l'exception qui arrive et gérez-la dans votre programme en affichant un message informatif et arrêtant le programme.
5. Nettoyage final : pour toutes les exceptions attrapées, mais non pas déjà explicitement gérées, affichez un message qui note le type de l'exception attrappé et arrêtez le programme. **Nous ne voulons pas** voir des "stacktrace"s.
6. Lancez le problème sans paramètre. Devrait-on attraper et gérer ces exceptions ?