TD10

Exceptions personnalisées

Dans la suite, les exceptions que vous allez **lever** ne seront pas celles existant déjà dans Java, mais des instances de classe que vous allez écrire vous même. Par exemple, dans une classe `Vaisseau`, lorsqu'on essaie de déplacer le vaisseau vers la gauche alors que le vaisseau est déjà à l'extrémité gauche de l'écran, on pourrait lever une exception `DeplacementImpossible` :

```
public class DeplacementImpossible extends Exception
{

}

public class Vaisseau
{
  private int position;
  private int nb_projectiles;
  private int nb_points_de_vie;
  ...
  public void deplacer (int deplacement) throws DeplacementImpossible
  {
      if (this.position + deplacement < 0){
          throw new DeplacementImpossible();
      }
  }
      else
      ...
}
```

1. Écrivez un executable dans lequel :
   - vous créez un vaisseau
   - vous écrivez une boucle for permettant de déplacer le vaisseau 20 fois vers la gauche (deplacement = -1)

   Gérez l'exception éventuellement levée par la méthode deplacer. Dans le cas où l'exception est levée, on affiche dans le terminal "Le vaisseau est déjà tout à gauche".
2. Écrivez une classe `PlusDeProjectilesException` qui **étend** `Exception`. Utilisez cette exception dans une méthode `tirer` qui renvoie un nouveau `Projectile` à la position du vaisseau si le nombre de projectiles est supérieur à 0 et lève l'exception sinon.
3. Écrivez une classe `PlusDePointsDeVieException` qui **étend** `Exception`. Utilisez cette exception dans une méthode `retirerPointsDeVie` qui ne renvoie rien mais lève l'exception si le vaisseau n'a plus de points de vie.
4. Écrivez un exécutable pour : créer un vaisseau, le déplacer 3 fois vers la gauche, tirer 5 projectiles, et lui enlever 10 points de vie. Gérez les exceptions.

Héritage et Interface

Informations

La classe `Hashset` permet d'implémenter des ensembles. Elle possède entre autres les méthodes suivantes :

- `add(E e)` : ajoute un élément à l'ensemble s'il n'est pas déjà présent.
- `contains(Object o)` : retourne vrai si l'élément spécifié est présent dans l'ensemble.
- `remove(Object o)` : supprime de l'ensemble l'élément spécifié.
- `size()` : retourne le nombre d'éléments de l'ensemble.

Dans cet exercice on veut modéliser une partition de musique. Aucune implémentation n'est demandée. Certaines méthodes peuvent ne pas avoir de code. Pour cela on suppose que l'on a la classe `Note` suivante :

```
public class Note{
   public Note(int frequence, int duree);
   public int getDuree();
   public int getFrequence();
   public void jouer();
   public int hashCode(); // ceci permet de les mettre dans un HashSet.
}
```

Remarque : une note de musique est plus que cela, cette modélisation est bien incomplète !

Un **accord** est un ensemble de notes ; il a une durée égale à la durée de la plus longue de ses notes ; je le trouve harmonieux si toutes les fréquences des notes sont des puissances de 2.

1. Créez une classe `Accord` qui hérite de `HashSet<Note>`. Ajoutez une méthode `duree`, une méthode `estHarmonieux` et une méthode `jouer` (qui peut n'être qu'un affichage dans le terminal dans notre exemple).
2. Dans un exécutable, créez un accord de 3 notes.
3. Une partition est la donnée d'une suite de notes et d'accords. On voudrait donc qu'une partition soit l'héritier d'un ArrayList de notes et d'accords. Quel est le problème ? Pour le résoudre, proposez une interface qu'Accord et Note pourraient implémenter.
4. Ceci implique de devoir toucher le code de `Note`. Malheureusement, on ne le peut pas : `Note` nous a été donné compilé ! Essayez de résoudre le problème en créant une classe `NotePerso` qui hérite de `Note` et qui implémente votre interface.
5. Écrivez une classe `Partition` ; donnez lui une méthode `jouer()`. Dans un exécutable, créez une Partition et *jouez*-là..