TP évalué #3

Introduction

Consignes générales

- Le tp doit être compilable avec la commande `gradle build`.
- Pour avoir la note maximale il faut que votre programme passe tous les tests unitaires (pré-définis, ne les modifiez pas!).
- Ceci est un travail *individuel*.
- Vous avez les 2h de la session à disposition.
- A la fin de la session: déposer votre TP, i.e., tout le projet (zippé) sur Moodle (format .zip uniquement, les autres formats ne seront pas acceptés !)

Remarque importante: ne modifiez **pas** les tests, il seront utilisés pour évaluer votre travail.

Instructions

- Commencez par télécharger le squellette fourni sur Moodle.
- Un script *build.gradle* est fourni ainsi que l'arborescence (respectant les conventions de Gradle) et un certain nombre de classes.
- A vous de compléter le squelette en écrivant les fonctions demandées dans les exercices.

Description

En préambule, nous attirons votre attention sur le fait que nous avons conscience que ce TP est long et que nous allons en tenir compte. Le nombre de tests qui passent vous donnent une idée de la note. Si tous les tests passent, la note de 6 est garantie. Si vous rendez un TP qui ne compile pas, la note 1.0 est garantie.

Le TP est à nouveau basé sur le jeu de rôle (RPG) que nous avons commencé lors des précédentes sessions. Il n'y aura que le concept de potions magique dans ce TP. Il s'agira essentiellement d'utiliser les streams pour filtrer des listes de potions selon certains critères. Les consignes des exercices sont en principe suffisamment claires et dans tous les cas nous ne répondrons qu'à des questions relatives à la compréhension de l'énoncé et non directement liées au code. Il peut sembler, au premier abord, qu'il y ait beaucoup d'exercices, mais chacun d'entre eux est relativement court.

Exercice 1

Dans la classe Game, completer la fonction `storeFileContentsForGame` qui se charge de lire un fichier (fourni) potions.txt et de stocker son contenu dans une variable statique `items`. La fonction de lecture (`readFile`) vous est déjà fourni (c'est la même que celle vue en cours). (Si jamais, le fichier texte potions.txt se trouve dans `/src/main/resources/`, mais il ne faut *pas* le modifier).

Exercice 2

Compléter la fonction `createManaPotionStream` qui va lire le contenu de la variable `items` (cela implique que l'exercice 1 fonctionne), et créer un stream d'objets `ManaPotion` à partir des informations contenues dans le fichier. Il vous faut donc *parser* chaque ligne du fichier. Il s'agit d'instancier des potions de mana avec leur points de mana et leur prix: tous deux doivent être de type `int`. Le fichier potions.txt contient des lignes strucurées selon la convention suivante:

- nom du type de potion (Mana ou Health)
- points de mana ou de vie
- prix
- le séparateur est un espace, i.e. chacun de ces 3 éléments est séparé par un espace

En résumé: Il faut donc créer un stream qui instancie une potion de mana pour chaque ligne du fichier qui commence par "Mana".

Remarque: le parsing et le reste peuvent se faire en une ligne, i.e., dans un seul stream, séquentiellement.

Exercice 3

Compléter la fonction `createHPPotionStream`: il s'agit essentiellement du même exercice que le précédent, mais c'est un stream de potions de vie: "Heath" est le mot-clef utilisé dans potions.txt. Il faut donc créer un stream qui instancie une potion de vie pour chaque ligne du fichier qui commence par "Health".

Exercice 4

Implémenter la fonction `filterManaPotionsByPrice(ArrayList<Item> items, int price)`. Celle-ci doit filtrer d'`items` les potions de mana dont le prix est *strictement* inférieur à `price`.

Indice: vous pouvez utiliser l'instruction `instanceof` pour tester si tel object est une `ManaPotion` ou une `HealingPotion`

Exercice 5

Implémenter la fonction `filterManaPotionsByMana(ArrayList<Item> items, int mana)`. Cela ressemble beaucoup à l'exercice précédent, mais il y a un petite complication: pensez au à *caster* si besoin... et aussi cette fois on veut filtrer uniquement les potions dont la valeur de mana est *strictement* supérieure à l'argument `mana`.

Exercice 6

Implémenter la fonction `public static <T> void copy_a_b(ArrayList<? extends T> a, ArrayList<? super T> b)`. Cette fonction copie une des deux arrayList dans l'autre. Selon sa signature, vous devriez pouvoir trouver laquelle est le ''producteur" et laquelle est le ''consommateur". (Le code de cette fonction peut être écrit en essentiellement deux lignes, mais vous pouvez l'écrire comme vous voulez tant que les tests passent, évidemment.)

Indice: PECS