Listes

Resp. UE : Érik Martin-Dorel & Jean-Paul Bodeveix

Sujets POO: Christelle Chaudet & Jean-Paul Bodeveix

Ce TP fait suite au TP 1. Vous devez donc avoir terminé le TP 1 pour passer au TP 2.

Pour rappel, les règles du jeu sont ici.

Méthode equals

1. Définir les méthodes **equals** sur les cartes sachant que le nombre d'exemplaires devra être ignoré dans la comparaison. On évitera autant que possible les copier/coller de code.
2. Ecrire dans un package tests un programme de test créant quelques cartes et vérifiant le résultat de leurs comparaisons

Mélange de cartes, listes

1. Ajouter au package cartes la classe **JeuDeCartes** contenant le tableau **Carte[]** typesDeCartes de tous les types de cartes avec le nombre d'exemplaires comme déclaré dans les règles du jeu. On considère comme dans le TP 1 que **VéhiculePrioritaire** est une botte de type **FEU**. On ignore les cartes Memo qui ne sont pas utiles dans le jeu.

   Le constructeur devra créer une **listeCartes** de toutes les cartes, chacune présente avec le nombre d'exemplaires souhaité. Par exemple, la carte FeuRouge devra être présente 5 fois dans cette liste.
2. Ecrire un programme de test affichant toutes les cartes du jeu.
3. Ecrire un programme de test vérifiant que le nombre d'exemplaires souhaité pour chaque type de carte a bien été respecté. On ajoutera une méthode **checkCount**() dans JeuDeCartes.
4. Définir dans un package nommé utils une classe **Utils** comprenant les méthodes génériques sta- tiques :
   a. **extraire** prenant en argument une liste supposée non vide : elle choisit de manière aléatoire un élément, le supprime et le retourne. On écrira 2 versions : l'une travaillant directement sur la liste et l'autre exploitant un ListIterator.

b. **mélanger** prenant en argument une liste et retournant la liste obtenue en extrayant tous les éléments de la liste passée en argument. Cette liste devient donc vide après l'appel.
c. **verifierMelange** prenant en argument 2 listes et renvoyant **true** ssi tout élément a le même nombre d'occurrences dans chacune des 2 listes. On utilisera **Collections.frequency**.
d. **rassembler** prenant en argument une liste et retournant la liste où les éléments identiques sont consécutifs. On ne cherchera pas à trier la liste.
e. **verifierRassemblement** prenant en argument une liste et retournant un booléen indiquant si les éléments identiques sont bien consécutifs. On introduira 2 ListIterator, le premier balayant la liste et le deuxième la fin de la liste à la recherche d'une duplication de la dernière des copies consécutives.

5. Ajouter un test pour ces méthodes. On vérifiera que le nombre d'occurrences de chaque élément présent initialement est inchangé dans la liste mélangée. On prendra comme liste le jeu de cartes créé précédemment. Attention de ne pas le détruire ! On utilisera une méthode statique de Collections pour compter le nombre d'éléments d'une liste. On introduira une méthode statique générique pour réaliser le test. On vérifiera le rassemblement sur les listes [], [1;1;2;1;3], [1;4;3;2], [1;1;2;3;1].
6. Modifier le code de JeuDeCartes pour que les cartes de **listeCartes** soient mélangées par le constructeur.
7. Ajouter un test vérifiant que le nombre d'occurences de chaque carte est bien respecté dans le jeu mélangé.