TP 3 - listes et ensembles

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

Sujets POO: Christelle Chaudet & Jean-Paul Bodeveix

Ce TP fait suite au TP 2. Vous devez donc avoir terminé le TP 2 pour passer au TP 3.

Pour rappel, les règles du jeu sont ici.
On se place maintenant dans un package jeu gérant la boucle de jeu et le comportement des joueurs.

1. Comme indiqué dans les règles du jeu, un joueur dispose d'une zone de jeu devant lui comportant une **pile de limites/fin de limites** de vitesse, une **pile dite de bataille** contenant les attaques et parades, une **collection de bornes** et un **ensemble de bottes**. Écrire la classe **ZoneDeJeu** sachant que les piles seront représentées par des listes.
2. Un joueur a un **nom** et une zone de jeu. Définir la classe **Joueur** avec son constructeur et un accesseur sur son nom.
3. Deux joueurs sont considérés identiques s'ils ont le même nom. Définir la ou les méthodes nécessaires. Définir également la méthode **toString** renvoyant son nom.
4. Définir l'interface **IMain** introduite pour gérer les cartes que le joueur a en main. Cette interface déclare deux méthodes : **void prendre** et **void joue**r prenant en paramètre une carte. Elle fournit aussi un itérateur sur les cartes qu'elle gère.
5. Proposer une implantation **MainAsListe** de **IMain** par une liste. **prendre** ajoute une carte à la liste. **jouer** supprime une carte en supposant la précondition affirmant sa présence (**assert**). Ajouter également la méthode **toString** affichant la main.
6. On revient à la classe **Joueur**. Ajouter la main au joueur (attribut, accesseur en lecture). Définir la méthode **donner** permettant de donner une carte au joueur. Il l'aura ainsi dans sa main.
7. Définir la méthode **Carte prendreCarte(List<Carte> sabot)**. La première carte du sabot est donnée au joueur qui la met dans sa main. La méthode renvoie cette carte ou renvoie **null** si le sabot est vide.
8. Définir la méthode **deposer(Borne borne)** qui ajoute une carte borne dans la zone du joueur.
9. Définir la méthode **int donnerKmParcourus()** renvoyant le nombre de km déposés dans la zone du joueur.
10. Écrire un programme de test dans la classe **TestJoueur** du paquetage **tests_fonctionnels** en ajoutant des bornes au joueur et en vérifiant le kilométrage atteint.
11. Définir l'interface **Cartes** contenant les constantes PRIORITAIRE, FEU_ROUGE, FEU_VERT qui sont des cartes de type FEU, respectivement instances de **Botte**, **Attaque** et **Parade**.

12. Définir la méthode **int donnerLimitationVitesse()** renvoyant la limite de vitesse en cours pour le joueur sachant que si la pile de limites est vide ou si le sommet de la pile est une fin de limite ou si le joueur est prioritaire (il a la botte de type FEU), la limite est de 200. Sinon elle est de 50.
13. Surcharger les méthodes ajouter de Joueur et de Zone avec les cartes de type Limite, Bataille et Botte. Ecrire un programme réalisant des tests de getLimite dans différentes configurations.
14. Définir la méthode **boolean estBloque()** renvoyant **true** si le joueur ne peut pas avancer. Il peut avancer dans l'une des situations suivantes :

    - la pile de bataille est vide et il est prioritaire,
    - le sommet est une parade de type FEU,
    - le sommet est une parade et il est prioritaire,
    - le sommet est une attaque de type FEU et il est prioritaire,
    - le sommet est une attaque d'un autre type pour lequel il a une botte et il est prioritaire.

15. Ecrire les tests correspondants. Définir une classe **TestZoneDeJeu** dans le paquetage tests_fonctionnels. On ajoutera successivement aux listes de la zone un feu rouge, un véhicule prioritaire, un accident, un as du volant, une panne d'essence, de l'essence. On efface les bottes et on ajoute un feu vert. On testera **estBloque** après chaque ajout/suppression afin d'obtenir :

    true, false, true, false, true, false, true, false