TD n°8 – Tableaux, collections

Services de la classe Arrays

https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html

La classe Arrays du package java.util offre des services pour les tableaux (tri, recherche, remplissage, égalité, …) La liste des méthodes définies dans la classe Arrays sont static et s'appellent donc en utilisant le nom de la classe suivi de celui de la méthode : Arrays.sort(Object[] a).

Soient les tableaux instanciés suivants :
int[] T1 = new int[10] ;
String[] T2 = new String[7] ;
int[] T3 = {0, 6, 2, 4, 3} ;
String[] T4 = {« bleu », « rouge », « blanc », « vert », « mauve », « indigo »} ;

Consulter les nombreuses méthodes disponibles dans la classe Arrays et trouver les méthodes permettant de faire les actions suivantes :

- Remplir T1 avec l'entier 5 dans toutes les cases.
- Remplir T2 avec la chaîne de caractères « bleu » dans les cases d'indices 1 et 2.
- Afficher les tableaux
- Trier T3 dans l'ordre croissant
- Trier T4 dans l'ordre alphabétique
- Vérifier si T1 et T3 sont égaux
- Copier les 5 premiers éléments de T4 dans un nouveau tableau T5
- Dupliquer T4 dans un tableau T6 et vérifier s'ils sont égaux
- Transformer T4 en List<String> et afficher la liste obtenue

Créer une classe avec une méthode main afin de tester les méthodes demandées ci-dessus.

Collection de Pokemons

Nous allons maintenant considérer la classe `Joueur` qui contient (au moins) :

- Un nom ;
- Un niveau ;
- Un nombre de points ;
- Une collection de Pokemons ;

Utiliser les listes pour représenter la collection du joueur ? (« Généraliser » le plus possible le type déclaré)

- La méthode `vitesseMoyenne()` retournant la vitesse moyenne des Pokemons de la collection ;
- Des méthodes `vitesseMoyenneTYPE(String t)` calculant la vitesse moyenne des Pokemons de type `t` de la collection.
  Lever des exceptions !
- La méthode `toString()` affichant l'état d'un joueur.

En plus de ces fonctionnalités de base, un joueur peut :

- Attraper un Pokemon en l'ajoutant à sa collection ;
- Transférer un Pokemon en l'enlevant de sa collection ;

Combat de Pokemons entre Joueurs : méthode aléatoire

Enfin, un joueur peut attaquer avec l'un des Pokemons de sa collection un autre joueur. Le joueur attaqué choisit alors un Pokemon de sa collection qui combattra contre le Pokemon de l'attaquant.

Nous souhaitons définir une méthode `void attaquer(Joueur adversaire)`.

- Etant donnée l'existence d'une interface `IAttaque`, ajouter l'implémentation de l'interface dans la classe `Joueur`.

L'attaque la plus simple est de choisir aléatoirement les Pokemons qui se combattent dans les collections respectives des 2 joueurs.

La méthode `attaquer` de la classe `Joueur` doit donc :

- Choisir aléatoirement un Pokemon dans la collection de chacun des joueurs ;
- Appeler la méthode `attaquer` de la classe `Pokemon` afin de modifier les points de vie des Pokemons qui combattent.

Parking

On veut écrire une classe `Parking` qui sert à stocker des voitures. Les voitures seront stationnées sur des places numérotées entre 1 et sa capacité. Le nombre de places disponible est fixé une fois pour toute à la construction du parking.

La structure de donnée choisie pour modéliser le parking est une table associative de type `HashMap` qui fait correspondre un numéro de place avec la voiture qui y stationne.

Réutiliser le type `Voiture` vu en TD4 et 5 (des instances vous sont également proposées dans le main de la classe `GestionVehicules` du TD4).

- Créer la classe `Parking` avec ses attributs et son constructeur ;
- Ecrire une méthode *garer* qui prend en paramètre une voiture, un numéro de place, et qui gare la voiture à la place donnée si elle est libre.
- Ecrire une méthode *liberer* qui prend en paramètre un numéro de place et qui retire du garage la voiture à cette place. La méthode retourne l'objet `Voiture` récupéré.
- Ecrire une méthode chercher qui prend en paramètre une voiture, et retourne le numéro de place associée.
- Ecrire une méthode *toString* retournant l'état du garage : pour chaque place, donner son numéro ainsi que les caractéristiques de la voiture qui l'occupe si elle existe,

Tester votre classe dans un main.

Extension : Ajout des exceptions

- Lorsque le numéro de place donné en paramètre n'existe pas dans le parking, lever une exception de type `IndexOutOfBoundsException`
- Lever une exception de type `IllegalStateException`
    - Si la voiture veut se garer sur une place déjà occupée,
    - Si la place à libérer est vide,
    - Si la voiture cherchant sa place n'est pas dans le parking.

Gérer ces exceptions dans le *main* dans des *catch* différents.