TP 5 : Interagir avec le jeu

Le but de ce TP est de permettre l'interaction des joueurs avec le jeu et notamment avec leurs pokemons. Nous allons voir comment on peut lire et interpréter des consignes tapées dans la console et on va permettre une interaction adaptative très primitive entre le joueur et les pokemons.

Lien utile : https://pokemondb.net/pokedex/all .

Préambule

Créez un nouveau projet Java qui s'appelle TP5. Comme d'habitude créez dedans un nouveau package tp5, puis copiez dans ce package le contenu de votre TP4. Vérifiez que cette opération a bien changé les packages et que le programme s'exécute bien encore.

Exercice 1

Le premier pas sera d'arriver à faire Java interpréter le texte écrit par l'usager dans la console. Cette étape se fera exclusivement dans la classe `ChasseAuxPokemons`.

Nous pouvons interpréter le code écrit sur une console en utilisant un objet d'un type spécial, notamment le type `Scanner`. Un des meilleurs aspects de Java étant sa documentation, nous pouvons aller chercher les spécifications des `Scanners` en ligne :

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

Un scanner est un outil qui peut analyser une source en entrée (un fichier, un flux de caractères, etc.). Il peut vérifier si la source a encore du contenu à parcourir, il peut trouver le prochain morceau de contenu, etc. Nous allons utiliser le scanner de la façon suivante : nous allons créer un objet de type `Scanner` qui prend en entrée un flux, notamment `System.in`. Puis, nous allons utiliser l'objet créé pour chercher les entrées mises dans la console par un utilisateur. En fonction de l'entrée nous allons soit continuer à le laisser taper, sur une ligne suivante, soit arrêter le programme.

1. Pour créer un objet de type Scanner il nous faut un constructeur. Celui qui nous intéresse prendra en paramètre seulement un flux en entrée (`InputStream`). Pouvez-vous trouver ce constructeur dans la liste de constructeurs ? Et si on voulait prendre en entrée plutôt un fichier ?
2. Dans votre méthode `main` de la classe `ChasseAuxPokemons` nettoyez votre code : enlevez tout le code du TP dernier sauf : les objets créés (y compris les tableaux créés) et le code qui génère

aléatoirement de la nourriture. Commentez le code qui génère de la nourriture. Vérifiez que le programme fonctionne encore.

3. En utilisant `System.in` en tant que paramètre pour le constructeur, créez un objet scanner de type `Scanner`. Normalement cela vous donne une erreur de compilation. Laquelle ?
4. La raison pour laquelle vous avez une erreur de compilation est le fait que la partie du code Java qui inclut la classe `Scanner` n'est pas mise à la disposition de chaque projet par défaut. Pour bien l'inclure dans le vôtre il va falloir ajouter un package Java qui s'appelle `java.util.Scanner`. Au dessus de votre classe TP5 ajoutez la ligne de code suivante :

   ```
   import java.util.Scanner ;
   ```

   Normalement votre code doit bien fonctionner maintenant.

5. Nous allons commencer avec un exercice très simple d'utilisation d'un scanner. Nous allons poser une question sur la console, vous allez mettre une réponse et nous allons confirmer que la réponse a bien été aperçue par le scanner.
   a. Dans votre méthode `main`, faites afficher la question « `Etes-vous etudiant de premiere annee a l'IUT de Limousin ?` »
   b. Nous allons utiliser la méthode `next()` de la classe `Scanner`. En utilisant la documentation Java de cette classe, décrivez son fonctionnement.
   c. Dans une variable `reponse` de type `String` nous allons stocker la réponse que vous allez mettre dans la console. Vous aurez notamment besoin de l'instruction :

      ```
      String reponse = scanner.next() ;
      ```

   d. Finalement, nous allons tester si le scanner a bien compris la réponse. Vous allez utiliser un `System.out.println` pour afficher la réponse reçue.
   e. Essayez d'écrire une réponse à plusieurs mots. Qu'est-ce qui se passe ?

      En fait la méthode `next()` prendra seulement le premier des mots utilisés. Si on veut utiliser plusieurs mots il va falloir utiliser une boucle et la méthode `hasNext` de la classe `Scanner`. Mais pour l'instant on va s'arrêter ici.

   f. Le bon fonctionnement d'un scanner demande sa clôture à la fin de son usage. On va donc finir par fermer le scanner en utilisant la méthode `close()` de la classe `Scanner`.

Exercice 2

Dans ce deuxième exercice nous allons changer la partie du code qui génère aléatoirement de la nourriture. Le nouveau fonctionnement sera le suivant : à chaque fois on va donner à l'utilisateur de dire s'il veut s'arrêter (en écrivant stop sur la console) ou s'il veut continuer (toute autre valeur). Tant que l'utilisateur veut continuer, on va générer un nombre aléatoire à chaque itération et en fonction de ce nombre nous allons générer de la nourriture ou non. Le texte qui sera affiché à chaque fois qu'une

nourriture est générée sera : `Vous avez trouve un.e/du/de la <nom de la nourriture>.` Nous travaillerons principalement dans la classe `ChasseAuxPokemons`.

1. Premièrement : faites ajouter à votre tableau de nourriture la gourmandise et la potion magique que vous avez créés la dernière fois.
2. Modifiez le code qui assure la génération de la nourriture pour qu'à chaque fois on génère un nombre aléatoire, et, pour chaque élément du nouveau tableau de nourriture, on génère la même nourriture si le nombre aléatoire est inférieur à la fréquence de la nourriture. Si on génère de la nourriture d'un certain type, alors on affiche le texte mentionné ci-dessus. Sinon, on n'affiche rien. Faites exécuter le programme résultant.
3. Modifiez votre code encore. Cette fois-ci il va falloir intégrer le scanner dans votre code. A chaque fois on pose la question si l'utilisateur veut continuer (il doit taper n'importe quelle lettre ou mot) ou s'il veut s'arrêter (il doit alors écrire stop dans la console). Tant que l'utilisateur ne dit pas stop on continue générer la nourriture selon le fonctionnement de l'exercice précédent.
4. N'oubliez pas de fermer le scanner à la fin.

Exercice 3

Maintenant nous allons créer un joueur qui portera votre nom. Il sera initialisé avec trois pokemons différents. A chaque itération il aura la possibilité de faire une action, par exemple de regarder ses pokemons, les caresser ou, plus tard, regarder la nourriture qu'il a réussi trouver, accumuler plus de nourriture ou nourrir un de ses pokemons. Cet exercice concernera en mesure égale les classes `Joueur` et `ChasseAuxPokemons`.

1. Premièrement nous allons permettre à un joueur de caresser l'un de ses pokemons. Ceci fera sa loyauté augmenter par 1. Dans la classe Joueur écrivez une méthode `void caresserPokemon(Pokemon pokemon)` qui vérifie premièrement si le pokemon indiqué appartient au joueur. Si c'est le cas, la méthode fait monter la loyauté du pokemon, toutefois sans dépasser 100. Si après la modification la loyaute reste inférieure à 100, on affiche « Mmmm, ca sent bon. Et sous mon oreille gauche ? ». Si la loyauté est de 100, on affiche le texte : « Oui, moi je t'aime aussi ! ».
2. Dans la classe `ChasseAuxPokemons` créez un joueur qui porte votre nom et prénom et qui a au moins trois pokemons différents.
3. Dans la boucle qui demande au joueur de choisir de continuer ou arrêter le jeu, modifiez le code pour demander à chaque tour si le joueur veut : regarder ses pokemons, en tapant « `1` », ou caresser l'un des pokemons, en tapant « `2` », ou arrêter le jeu, en tapant « stop ». Faites exécuter le code pour voir qu'il fonctionne bien.
4. Continuez à raffiner votre code. Cette fois-ci, si le joueur tape « `1` », faites afficher tous ses pokemons non-nulls, précédés par leur index dans le tableau de pokemons. Donc pour chaque pokemon on affiche « `Index <index> : <pokemon affiche>` ».
5. Maintenant, si le joueur tape « `2` », demandez-lui l'index du pokemon qu'il veut caresser (de 0 à 4). Pour interpréter cette valeur comme un entier et non un `String`, utilisez la méthode `nextInt()` plutôt que la méthode `next()`. Vérifiez que la valeur donnée en entrée est entre

0 et 4 et si c'est le cas et si le pokemon à cet index est non-null et faites exécuter la méthode `caresserPokemon`. Si l'index est incorrect, le joueur perdra son tour.

6. Faites exécuter le code pour vérifier qu'il marche bien.

Exercice 4

Dans ce dernier exercice on va permettre aux joueurs d'amasser de la nourriture et de donner à manger à ses pokemons. A ce but nous allons introduire un nouvel attribut qui s'appelle provisions. Cet attribut sera un tableau de taille 10 dont les éléments seront du type `Nourriture`. Attention : j'ai choisi de faire un tableau de cet attribut, mais il serait mieux en tant que `Collection`. Comme nous n'avons pas encore travaillé avec les collections, toutefois, nous allons utiliser un tableau. Dans cet exercice nous allons modifier principalement la classe `Joueur`.

1. Introduisez un nouvel attribut provisions de type `Nourriture[]` de taille 10.
2. A son instanciation, le joueur n'aura aucune provision. Dans le constructeur avec le plus de paramètres ajoutez du code qui instancie le tableau. Est-ce que cela affectera votre autre constructeur également ?
3. Ecrivez des getters pour les deux tableaux : celui de pokemons et celui de provisions.
4. Ecrivez une méthode privée avec la signature `int trouverProvision(Nourriture nourriture),` dont le fonctionnement est le suivant : on itère sur les positions du tableau provisions jusqu'à trouver la provision donnée en entrée. On retourne l'index de cette provision. Si la provision n'est pas dans le tableau on retourne la valeur par défaut -1.
5. En utilisant la méthode trouverProvision, écrivez une méthode `void ajouterProvision(Nourriture nourriture)`, qui ajoute une provision au tableau (ou envoie un message d'erreur s'il n'y a plus de place dans votre tableau).
6. Ecrivez une méthode `void nourrirPokemon(Pokemon pokemon, Nourriture nourriture),` dont le fonctionnement est le suivant. Si l'un des paramètres est `null` ou s'ils n'appartiennent pas au joueur, alors on retourne un message d'erreur. Sinon, on fait au pokemon manger la nourriture donnée. La nourriture disparaît du sac de provisions.

Exercice 5 (difficile)

Dans cet exercice le but sera de modifier l'interaction entre l'utilisateur et le jeu en lui donnant la possibilité d'ajouter de la nourriture à son sac de provisions, en lui permettant de voir le contenu de ce sac à provision et en lui permettant de nourrir l'un de ses pokemons. Si le joueur trouve de la nourriture il aura le droit à l'ajouter à son sac à provisions sans que cela compte comme une action séparée. Il pourra ajouter autant de nourriture qu'il trouve. On modifie le code de la méthode `main` dans la classe `ChasseAuxPokemons`.

1. Modifiez la partie du code où vous générez la nourriture. A chaque fois que vous en générez une, demandez au joueur s'il veut la prendre. Si c'est le cas (la réponse est oui), on ajoute la nourriture à son sac à provisions.

2. Modifiez la partie de votre code où vous donnez des choix à l'utilisateur. Ajoutez les options 3. Regarder votre sac à provisions et 4. Nourrir un pokemon. Pour l'option 3 on affiche le contenu du sac à provisions. Pour l'option 4. on demande l'index du pokemon et de la nourriture qu'il souhaite utiliser.
3. Faites exécuter votre code avec de diverses commandes pour vous assurez qu'il marche comme il faut.