Fantaisie Travaux Pratiques n°4

« hashCode, HashSet, LinkedHashSet, HashMap»

Ce TP poursuit le TP3 sur les monstres. Afin d'avoir toujours une version stable par sujet de TP :
- Ouvrir Eclipse, sélectionner le même workspace que pour le premier TP (par exemple : Z:\Licence3\Semestre1\POO2\workspace).
- Copier le projet fantaisieTP3 (en le sélectionnant puis ctrl+c),
- Coller le projet (ctrl+v), et renommer le fantaisieTP4.

Dans ce TP vous travaillerez donc uniquement dans ce nouveau projet.

1. Les salles

Dans le paquetage « bataille » créer la classe « Salle » comportant :
- Deux attributs : numSalle (un entier) et zoneDeCombat :
  - si la salle est haute elle sera parfaite pour un monstre de type aérien,
  - si la salle comporte de l'eau souterraine elle sera parfaite pour un monstre de type aquatique
  - sinon ce sera un monstre de type terrestre qui pourra l'occuper.
- Un constructeur initialisant ces deux attributs,
- Les getters sur ces deux attributs,
- La méthode *toString* permettant l'affichage correspondant à l'exemple ci-dessous :

```
Salle n°1 de type combat TERRESTRE
```

2. La grotte

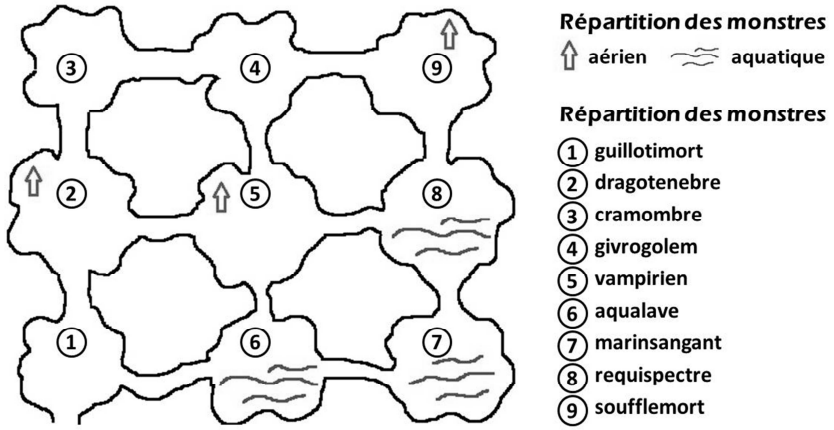
La grotte possède quatre attributs :
- planGrotte de type « LinkedHashMap » associant une salle à une liste de salle (salles accessibles à partir de la salle d'origine). Nous voulons rapidement avoir les différentes salles accessibles à partir de la salle en cours d'où « HashMap » ; mais nous voulons aussi avoir le plan de la grotte toujours de la manière dont il a été entré d'où « Linked ».
- batailles de type « HashMap » associant une salle à une bataille : dans chacune des salles il y aura une bataille (avec le camp des monstres et le camp des humains).
- sallesExplorees de type « HashSet » : contient les salles que les humains ont déjà traversées. Un « Set » pour ne pas avoir de doublons.
- numeroSalleDecisive de type entier, indiquant dans quelle salle se trouve la pierre de sang.

Ajout d'une salle

Ecrire la méthode *ajouterSalle* possédant deux paramètres d'entrée :
- une zone de combat, permettant de savoir selon les caractéristiques de la salle si on aura plutôt une bataille aérienne, terrestre ou aquatique.
- Un vararg pour le ou les monstres qui gardent la salle.

Cette méthode crée la salle avec comme numéro la taille de la « LinkedHashMap » planGrotte et la zone de combat. Un nouvel objet de type bataille est créé et le ou les monstres rejoignent la bataille dans le camp des monstres. La salle est ajoutée aux attributs :
- planGrotte avec pour clé la salle et pour valeur une « ArrayList ».
- batailles avec comme clé la salle et comme valeur la bataille.

Afficher le plan de la grotte



Des méthodes sont téléchargeables depuis Moodle. Copier dans la classe « Camp » les méthodes nbCombattants et selectionner. Copier dans la classe « Grotte » la méthode *afficherPlanGrotte*. Vérifiez que vous avez bien noté les attributs / classes comme décrit dans l'énoncé.

Dans le paquetage « *testsfonctionnels* » créer la classe « TestGestionGrotte » dans lequel vous copierez les monstres disponibles sous Moodle.

Tester votre méthode *ajouterSalle* en affichant la grotte.

```
Grotte grotte = new Grotte();
grotte.ajouterSalle(ZoneDeCombat.TERRESTRE, guillotimort);
grotte.ajouterSalle(ZoneDeCombat.AERIEN, dragotenebre);
grotte.ajouterSalle(ZoneDeCombat.TERRESTRE, cramombre);
grotte.ajouterSalle(ZoneDeCombat.TERRESTRE, givrogolem);
grotte.ajouterSalle(ZoneDeCombat.AERIEN, tableauVampirien);
grotte.ajouterSalle(ZoneDeCombat.AQUATIQUE, aqualave);
grotte.ajouterSalle(ZoneDeCombat.AQUATIQUE, marinsangant);
grotte.ajouterSalle(ZoneDeCombat.AQUATIQUE, requispectre);
grotte.ajouterSalle(ZoneDeCombat.AERIEN, soufflemort);
```

Configuration des accès
Ecrire la méthode *configurerAcces*

Cette méthode utilisera le fait que nous sommes dans un « **Linked**HashMap » avec des salles dont le numéro a commencé à 1.
Pour spécifier cette méthode nous utiliserons la méthode privée *trouverSalle* qui permet de retrouver une salle dans la map planGrotte à partir de son numéro. La méthode *configurerAcces* utilise la méthode *trouverSalle* pour :
- trouver la salle d'origine (celle dont le rang est passé en paramètre),
- trouver l'ensemble des salles accessibles (dont les numéros sont passés en paramètre par un vararg).

Elle permet donc de configurer les accès en associant à la salle d'origine celles passées en paramètre.

Remarque : dans la méthode *trouverSalle* nous ne sommes pas efficaces car nous devons parcourir l'ensemble des salles de la grotte ; mais il s'agit de l'initialisation. Pour l'utilisation future nous utiliserons bien une salle comme clé pour accéder à la valeur de la map, donc on reste sur une bonne utilisation du « LinkedHashMap ».

trouverSalle : possède un paramètre d'entrée `numeroSalle` : numéro de la salle que l'on veut retourner. Pour implémenter cette méthode :
- récupérer les salles de la grotte,
- créer un entier `indice` qui permettra de savoir combien de salle on a parcouru.
- parcourir les salles jusqu'à trouver la salle et la retourner.

configurerAcces : possède deux paramètres d'entrée :
- le numéro de la salle d'origine depuis laquelle on veut passer dans d'autres salles.
- Un vararg contenant les numéros des salles accessibles depuis la salle d'origine.

Trouver la salle d'origine à l'aide de son numéro donné en paramètre d'entré, puis récupérer la liste des salles qui lui sont accessibles (vide au début de l'initialisation). Parcourir l'ensemble des numéros des salles accessibles donnés en paramètre d'entrée (vararg). Pour chacun des numéros, trouver la salle qui lui correspond et l'ajouter à la liste des salles accessibles.

Tester votre méthode *configurerAcces* en affichant la grotte.
```
grotte.configurerAcces(1, 2, 6);
grotte.configurerAcces(2, 1, 3, 5);
grotte.configurerAcces(3, 2, 4);
grotte.configurerAcces(4, 3, 5, 9);
grotte.configurerAcces(5, 2, 4, 6, 8);
grotte.configurerAcces(6, 1, 5, 7);
grotte.configurerAcces(7, 6, 8);
grotte.configurerAcces(8, 5, 7, 9);
grotte.configurerAcces(9, 4, 8);
```


Choix de la salle décisive
Ajouter un setter sur l'attribut numeroSalleDecisive.

Ecrire la méthode salleDecisive qui à partir d'une salle passée en paramètre d'entrée retourne un booléen permettant de savoir si elle contient la pierre de sang ou non.

Tester votre méthode *salleDecisive* en affichant la grotte.

Trouver l'entrée de la grotte
Ecrire la méthode *premiereSalle* qui retourne la première salle de la grotte et l'ajoute dans les salles explorées.

Tester votre méthode en demandant la première salle de la grotte.
```
System.out.println("\n**** TP4 ****");
System.out.println("Plan de la grotte\n" + grotte.afficherPlanGrotte());
System.out.println("\nParcours de la grotte\n");
Salle salle = grotte.premiereSalle();
System.out.println(salle + "\n");
```

Choisir la prochaine salle à explorer
Ecrire la méthode *salleSuivante* qui retourne la prochaine salle à explorer. Pour cela elle reçoit en paramètre d'entrée la salle que le groupe d'humain veut quitter. Grâce à l'attribut planGrotte elle récupère la liste des salles accessibles. Ensuite il vous faut enlever les salles déjà explorées sallesExplorees. S'il ne vous reste plus de salle alors remettre toutes les salles accessibles en jeu. La salle destination sera placée dans la liste sallesExplorees. Retourner une salle prise au hasard dans la liste.

Tester votre méthode en parcourant toutes les salles jusqu'à celle contenant la pierre de sang.
```
if (!grotte.salleDecisive(salle)) {
  do {
     salle = grotte.salleSuivante(salle);
     System.out.println(salle + "\n");

  } while (!grotte.salleDecisive(salle));
  System.out.println("C'est la salle contenant la Pierre de Sang");
}
```