INITIATION À LA PROGRAMMATION 2 (IP2)

TP4 bis : réseaux d'espionnage

Dans ce TP, vous allez modéliser le comportement des réseaux d'espionnage qui sont organisés comme suit. Chaque `Espion` vit seul dans une maison sécurisée. De chaque `Maison`, un canal de communication unidirectionnel est disponible vers une autre maison sécurisée. Un `Reseau` est modélisé en spécifiant une maison sécurisée qui figure comme le siège de ce réseau. Les espions peuvent être `loyal` ou `déloyal` au gouvernement. Dans la deuxième partie de ce TP, vous allez écrire des méthodes de traitement des espions (dé)loyaux...

1. Écrivez une classe `Espion` qui contient les éléments suivants :
   - **attributs** privés : une chaîne de caractères `vraiNom`, un booléen `loyal`, et un identifiant entier unique `id`. Le premier espion créé aura l'identifiant `1` ;
   - deux **constructeurs** d'espions : l'un prend en argument un `vraiNom` et un booléen `loyal`, l'autre ne prend qu'un `vraiNom` en argument, et suppose que le nouvel espion est loyal au début ;
   - une méthode `description` qui retourne une chaîne de caractères de la forme "`espion 1 (loyal)`" ou "`espion 2 (déloyal)`".
2. Testez votre code : sauvegardez le fichier `Test.java` fourni sur Moodle dans le même dossier que le fichier `Espion.java` que vous avez créé. En Eclipse, vous devez aussi importer le fichier dans le projet, avec le menu File, Import, General - File System. Si Eclipse s'y oppose, vous pouvez créer une classe `Test` comme d'habitude, et après copier-coller le contenu du fichier sur Moodle dedans.
3. Écrivez une classe `Maison` qui contient les éléments suivants :
   - **attributs** privés : un `habitant` espion, une `ligneVers` une autre maison (éventuellement `null`), et un identifiant `id` de type `char`. La première maison aura l'identifiant '`a`', la deuxième '`b`', etc. ;
   - deux **constructeurs** : l'un prend en argument un `habitant` et une `ligneVers`, et l'autre prend en argument seulement un `habitant`.
   - une méthode `description` qui retourne une chaîne de caractères de la forme

     `maison c : habitée par espion 2 (déloyal), ligne non connectée`

     ou, le cas échéant,

     `maison a : habitée par espion 3 (loyal), ligne vers maison b`

4. Écrivez une classe `Reseau` qui satisfait aux critères suivants :
   - deux **attributs** privés : un `nom` et un `siege`, qui est une maison (éventuellement `null`).
   - deux **constructeurs** : l'un permet de créer un réseau avec un nom et avec un siège habité par un espion donné en argument, et l'autre permet de créér un réseau avec un nom et avec siège `null`.
   - une méthode `afficher` qui permet d'afficher un diagramme 'anonymisé' du réseau. Par exemple, un diagramme d'un réseau nommé `monReseau` pourrait être le suivant :

```
monReseau
---------
Maison        c  ->  b  ->  a
Habitant      3      1      2
Loyal?        O      O      N
```

5. Écrivez une méthode `estPresent(Espion e)` de la classe `Reseau` qui renvoie `true` si l'espion donné en argument est dans le réseau, et `false` sinon.
6. Écrivez une méthode `ajouterEspion(Espion e)` de la classe `Reseau` qui teste d'abord si l'espion donné en argument est dans le réseau. Si c'est le cas, on affiche un texte comme "`l'espion 3 est déjà dans le réseau dans la maison c`", et on ne change pas le réseau. Si ce n'est pas le cas, on crée une nouvelle maison pour cet espion, et l'ajoute au réseau. Si le réseau est vide, la nouvelle maison devient le siège. Sinon, on crée une ligne de la dernière maison actuelle vers la nouvelle maison.
7. Testez la classe `Reseau` en utilisant la classe `Test.java`. Ajoutez un autre réseau qui correspond exactement au réseau affiché ci-dessus - notez bien l'ordre des noms des maisons. Pour faire cela, ajoutez une méthode `setLigne` à la classe `Maison`.
8. Écrivez une méthode `desertion`, qui prend en argument un espion, et, s'il est présent dans le réseau, change son attribut `loyal` en `false`.
9. Écrivez une méthode `insertion`, qui prend en argument un espion, et insère l'espion dans une nouvelle maison dans le réseau, qui vient juste avant le premier espion déloyal.

   Veillez à tester que la méthode que vous écrivez fonctionne dans les deux cas extrêmes : si l'espion au siège n'est pas loyal, l'espion donné en argument devient le nouveau siège ; si tous les espions sont loyaux, l'espion donné en argument s'ajoute à la fin du réseau.
10. Écrivez une méthode `enlever`, qui prend en argument un espion, et enlève du réseau la maison qui contient cet espion, s'il y en a une. Veillez à ce que les lignes des autres maison restent connectées : lorsqu'on enlève une maison, la ligne de la maison avant (s'il y en a une) est reliée à la maison suivante.
11. Écrivez une méthode `premierLoyal`, qui renvoie la première maison qui contient un espion loyal, ou `null` s'il y en a pas.
12. Écrivez une méthode `nettoyer`, qui enlève tous les espions déloyaux du réseau actuel, et qui retourne un nouveau réseau contenant les maisons des espions déloyaux, dans le même ordre que dans l'ancien réseau. Le réseau retourné par `nettoyer` a comme nom la concatenation du nom de l'ancien réseau avec le mot `Deloyal`.

   Donnons un exemple de l'effet d'appeler la méthode `nettoyer`. Voici un réseau `monReseau` avant nettoyage :

```
monReseau
---------
Maison      a -> b -> d -> c
Habitant    3    1    4    2
Loyal?      O    N    O    N
```

   Après un appel à la méthode `nettoyer` :

```
monReseau                          monReseauDeloyal
---------                          ----------------
Maison      a -> d                 Maison      b -> c
Habitant    3    4                 Habitant    1    2
Loyal?      O    O                 Loyal?      N    N
```