TP6 : un peu de collections et d'encapsulation

Dans les TP précédents, nous avons d'abord implanté une simulation macroscopique d'un écosystème (modèle de Lotka-Volterra), dans laquelle nous avons modélisé les deux populations (proies / prédateurs). Ensuite nous avons modélisé un terrain à une (ou deux) dimension(s) mais sans modéliser explicitement les êtres vivants eux-même. Enfin nous avons modélisé des êtres vivants qui évoluaient indépendemment les uns des autres.

Dans ce TP, on se propose d'implanter des simulations où plusieurs êtres vivants évoluent sur un même terrain, mais sans se préoccuper de leur position. Un terrain sera donc essentiellement une collection d'êtres vivants. Réciproquement, un être vivant sera associé de manière unique à un terrain.

Exercice 1 (Un terrain plein de mulots)**.** – Complétez la classe `Terrain` fournie, en utilisant un tableau d'êtres vivants de taille 100 comme structure de donnée. Des tests unitaires sont fournis. Notez que certains tests ne fonctionneront qu'une fois la classe `Mulot` implantée.
- Consultez la classe `EtreVivant` fournie. Décrire ce que fait le constructeur.
- Écrivez une classe `Mulot` dérivant de `EtreVivant`. Un mulot a une méthode `se_reproduit` qui construit un nouveau mulot ; de plus à chaque appel à `évolue`, il se reproduit. Des tests unitaires sont fournis.
- Ajouter une fonction `Mulot.main` qui construit un terrain avec un mulot et le fait évoluer pendant 10 itérations en affichant le nombre de mulots. Que constatez-vous ?

Exercice 2 (Un terrain plein de souris)**.** – Écrivez une classe `Souris` similaire à `Mulot`, à une différence prêt : une souris devient adulte qu'après une itération, et ne se reproduit qu'une fois adulte.
- (Optionnel) Éviter la duplication de code en déplaçant la méthode `reproduit` de `Mulot` dans `EtreVivant` et en l'adaptant. Vous aurez besoin d'utiliser l'*introspection* (voir `java.lang.reflect`) pour récupérer la classe de l'animal qui se reproduit.
- Ajouter une fonction `Souris.main` qui construit un terrain avec une souris et le fait évoluer pendant 10 itérations en affichant le nombre de souris. Que constatez-vous ?

Exercice 3 (Tableaux dynamiques et encapsulation)**.** – Faites évoluer le terrain précédent sur 20 itérations. Que constatez-vous ?
- Ajoutez un test le plus simple possible à `TerrainTest` caractérisant le problème constaté. Le test ne doit pas passer.
- Consulter l'API de la classe Vector sur le web. Donner le nom des méthodes pour rajouter un élément, supprimer un élément, accéder à un élément en position i.

  Vous pouvez ignorer pour l'instant le fait que cette classe utilise les notions d'interface et de classe générique, notions que l'on étudiera plus tard.
- Écrire une mini application qui :
  - Construit un vecteur de chaînes de caractères (`Vector<String>`) vide.
  - Rajoute successivement les chaînes "bonjour", "ça", "va", "bien"
  - Utilise une boucle « for » pour afficher toutes les chaînes.
  - Supprime la chaîne "ça".
  - Insère la chaîne "Alfred" à la même position.
  - Utilise une boucle « foreach » pour afficher toutes les chaînes.

- Changez l'implantation de `Terrain` pour utiliser un tableau dynamique de type `Vector<EtreVivant>`.
- Vérifiez que les tests passent à nouveau.
- Faites évoluer le terrain avec les souris sur 30 itérations.
- Lors du changement d'implantation de `Terrain`, avez-vous dû modifier les autres classes ? Est-ce souhaitable ? Pourquoi ?

Exercice 4 (Système proie-prédateur à l'échelle microscopique (Optionnel))**.**

- Ajoutez une méthode `terrain.aleatoire()` qui renvoie un être vivant au hasard présent sur le terrain. La tester.
- Ajoutez une méthode `terrain.enleve(etre_vivant)` qui enlève l'être vivant indiqué du terrain. La tester.
- Ajoutez une méthode `etre_vivant.meurt()` qui supprime l'être vivant du terrain. La tester.
- Ajoutez une méthode `terrain.nombre_souris()` qui compte le nombre de souris sur le terrain (indication : utiliser `instanceof`). La tester. Idem pour `terrain.nombre_fennecs()`.
- (Optionnel) Éviter la duplication de code en implantant une méthode `terrain.nombre(C)` où `C` est une classe comme `Souris` ou `Fennec`.
- Implanter une classe `Fennec` dérivant de `EtreVivant`. Un fennec évolue à chaque itération en chassant. Avec une probabilité dépendant du nombre de souris, il capture l'une d'entre elles et la mange. S'il a mangé, il se reproduit avec une certaine probabilité. Dans tous les cas, il meurt avec une certaine probabilité.
- Construisez un terrain avec quelques souris et quelques fennecs et regardez l'évolution du nombre de souris et de fennecs. On pourra tracer le nombre de souris et de fennecs en affichant à chaque itération une ligne de la forme `" s f"` où la position de 's' et de 'f' reflète respectivement le nombre de souris et de fennecs.
- Faites varier les paramètres pour voir comment change l'évolution. Essayez de reproduire des cycles périodiques comme dans le modèle Lotka-Volterra.