Université Grenoble Alpes
Licence 2 MIASHS - Inff3 – Algorithmique et programmation par objets
2023-2024

TD 8

Chenilles : encore des classes et un peu d'héritage

Exercice 1 : Cases, chenilles et espaces

On se propose de définir un ensemble de classes pour représenter des chenilles évoluant dans des espaces clos. Une chenille occupe des cases contigües sur un espace et grandit lorsqu'elle se déplace si elle n'a pas encore atteint sa taille adulte. Pour définir les 3 classes de cet exercice, il est conseillé de lire l'énoncé de l'exercice dans son intégralité.

1.1 **Classe `Case`**

Une instance de la classe `Case` permet de repérer une case dans un espace. Chacune de ces instances est représentée à l'aide de 2 attributs d'instance de type `int` correspondant respectivement à l'abscisse et l'ordonnée de la case concernée. Définir les constructeurs et les méthodes suivantes dans la classe `Case`.

- Un constructeur prenant en paramètres 2 `int` correspondant à l'abscisse et l'ordonnée.
- Deux accesseurs en lecture permettant d'obtenir respectivement la valeur de l'abscisse et de l'ordonnée.
- Quatre méthodes `est()`, `nord()`, `ouest()`, `sud()` renvoyant chacune une `Case` et permettant d'obtenir la case voisine située à l'est, au nord, à l'ouest ou au sud sachant que dans un espace les cases sont indicées de gauche à droite et de haut en bas.
- Une méthode `voisines()` renvoyant un tableau de 4 `Case` correspondant aux 4 cases voisines.
- Une méthode `equals(Object o)` renvoyant un `boolean` et permettant de verifier que `o` est équivalent à la case courante.
- Une méthode `toString()` renvoyant dans une `String` la représentation textuelle de la case courante. Par exemple, le résultat pour la case d'abscisse 2 et d'ordonnée 4 est "`(2, 4)`".

1.2 **Classe `Espace`**

Une instance de la classe `Espace` représente un espace rectangulaire de cases. La case de coordonnées (0, 0) se situe en haut à gauche de cet espace et toutes les cases de l'espace ont des coordonnées positives. Pour représenter un espace, on utilise 4 attributs d'instance : deux `int` correspondant à la largeur et la hauteur (en nombre de cases), un tableau de `Chenille` destiné à accueillir les `Chenille` présentes sur l'espace, et un `int` donnant le nombre effectif de `Chenille` actuellement présentes sur l'espace. Définir les constructeurs et les méthodes suivantes dans la classe `Espace`.

- Un constructeur sans paramètres, un avec un paramètre de type `int` et un avec deux paramètres de type `int` : les deux paramètres correspondent à la largeur et la hauteur de l'espace, si l'un d'eux est omis on considère un espace carré, si les deux sont omis un espace 10x10 est initialisé. Le tableau des `Chenille` doit pouvoir accueillir 5 `Chenille`, aucune n'est initialement présente sur l'espace.
- Une méthode sans retour `addChenille(Chenille c)` ajoutant `c` à l'espace. On suppose que cet ajout peut se faire sans problèmes (`c` ne chevauche pas une chenille déjà présente dans l'espace). S'il n'y a plus de place dans le tableau des chenilles présentes, on le réalloue avec 5 éléments supplémentaires.
- Une méthode `contient(Case c)` renvoyant un `boolean` et permettant de vérifier que `c` est bien dans les limites de l'espace.
- Une méthode `caseOccupee(Case c)` renvoyant un `boolean` et permettant de vérifier que `c` est occupée (par l'une des chenilles présentes dans l'espace qui peuvent aider à déterminer la réponse).
- Une méthode `caseAuHasard()` retournant une case prise au hasard dans l'espace.
- Une méthode `caseLibreAuHasard()` retournant une case non occupée prise au hasard dans l'espace.

- Une méthode `toString()` retournant dans une `String` la représentation textuelle de l'espace sur plusieurs lignes. Les cases vides y sont dénotées par des `'.'` et les cases occupées par une `Chenille` par des lettres (une majuscule pour la tête de la chenille et des minuscules pour le reste du corps). Voici un exemple d'affichage d'une `String` résultant de cette méthode pour un espace contenant 2 `Chenille` (la première est dénotée par des `'a'` et la deuxième par des `'b'`).

```
......B
..A...b
..a...b
..aa..b
.......
```

1.3 **Classe `Chenille`**

Une instance de la classe `Chenille` représente une chenille pouvant évoluer sur un espace. Une chenille est représentée à l'aide d'un tableau de `Case` correspondant aux cases qu'elle occupe dans un espace, le premier élément de ce tableau correspondant à la position de la tête de la chenille. On retient également dans un attribut de type `Espace` l'espace pour lequel la chenille a été créée et dans un attribut de type `int` la taille effective de la chenille. Définir les constructeurs et les méthodes suivantes dans la classe `Chenille`.

- Un constructeur prenant un paramètre de type `Espace`, un autre prenant 2 paramètres de types `int` et `Espace` et encore un autre prenant 3 paramètres de type `Case`, `int` et `Espace` : l'`Espace` donné est celui pour lequel la chenille est créée, l'`int` donné correspond à la taille adulte (s'il est omis, on prend 10 par défaut) et la `Case` donnée correspond à la position de la tête de la chenille (si elle est omise, on prend une case libre au hasard dans l'espace). Si la position de la tête est incorrecte vis-à-vis de l'espace donné, une exception doit être levée. A sa création, une chenille se limite à une tête, elle grandit d'une case à chaque déplacement jusqu'à obtenir sa taille adulte qu'elle ne peut en aucun cas dépasser.
- Une méthode `estSur(Case c)` renvoyant un `boolean` et permettant de vérifier si la chenille occupe la case `c`.
- Une méthode `getCases()` renvoyant un tableau des `Case` effectivement occupée par la chenille.
- Une méthode sans retour `avance()` faisant avancer la chenille au hasard vers l'une des cases libres voisines de sa tête et la faisant grandir si elle n'a pas encore atteint sa taille adulte. Si la chenille n'a pas de possibilité de mouvement (toutes les cases voisines de la tête sont occupées), une exception est levée.
- Une méthode `toString()` renvoyant dans une `String` la représentation textuelle d'une chenille. Par exemple, pour la chenille A de l'exemple ci-dessous, si sa taille maximale est 5, on obtient `"Chenille(4/5):[(2, 1), (2, 2), (2, 3), (3, 3)]"`. Si elle avance au nord, on obtient ensuite `"Chenille(5/5):[(2, 0), (2, 1), (2, 2), (2, 3), (3, 3)]"`. Et si elle avance encore à l'ouest, on obtient ensuite `"Chenille(5/5):[(1, 0), (2, 0), (2, 1), (2, 2), (2, 3)]"`.

1.4 Programme principal

Ecrire un programme principal créant un espace 10x7 dans lequel on place 2 chenilles de taille 5 et 9 et qui leur fait effectuer 20 déplacements chacune en affichant leur progression sur la console.

Exercice 2 : Un peu plus loin avec de l'héritage

On se propose maintenant d'utiliser l'héritage pour affiner les classes obtenues dans l'exercice précédent. On souhaite pouvoir créer des espaces dans lesquels des obstacles fixes sont présents. On souhaite également disposer de chenilles ne se déplaçant pas au hasard mais qui vont par exemple tout droit tant qu'elles le peuvent et tournent systématiquement vers la gauche lorsqu'elles rencontrent un obstacle (qu'il soit fixe ou que ce soit une autre chenille). Discuter de la façon dont l'héritage peut être utilisé pour apporter ces améliorations, ainsi que d'autres éventuellement…