IUT de Paris Descartes – Base de la Programmation Objet

Travaux Pratiques – Sujet n°8

Objectifs

— Définir une hiérarchie de classes
— Définir des constructeurs dans une telle hiérarchie
— Employer des méthodes héritées

Animation d'une "chenille"

Il s'agit de réaliser une application permettant d'animer une ou plusieurs "chenilles" se déplaçant dans une fenêtre sur l'écran, chaque chenille se déplaçant de manière aléatoire et "rebondissant" sur les bords de la fenêtre. Notre animation adoptera une représentation textuelle.

Un programme principal réalisant l'animation d'une chenille vous est fourni. Créez un projet et importez le fichier `AppliChenille.java` se trouvant sur le disque commun.

1. Étudiez le programme principal et les méthodes statiques qui l'accompagnent. Faites en sorte qu'il compile (les correctifs rapides proposés par *eclipse*, et accessibles au clavier par `Ctrl-Shift-1`, peuvent largement accélérer cette tâche).

Solution: Le programme principal instancie un objet de type `Chenille` (du paquetage `chenille`). Le constructeur de cette classe fixe sa taille et la position de sa tête. Cette classe doit aussi proposer deux méthodes. La première doit permettre de dessiner la chenille dans un tableau de caractère. La seconde doit permettre de déplacer la chenille.

Les méthodes statiques `effacer` et `afficher`, définies dans `AppliChenille.java`, sont triviales. La première remplit le tableau de caractère avec des espaces alors que la seconde affiche son contenu à l'écran. Une étude de la méthode d'affichage permet de déterminer que l'axe des x est horizontal alors que celui des y est vertical et que le point de coordonnée $(0, 0)$ est en bas à gauche. Enfin, la méthode statique `pause` permet de stopper le programme pendant 200 millième de secondes. Les exceptions (et les blocs `try/catch`) seront vues dans la suite du cours.

Le code minimal que doit contenir la classe `Chenille` pour une compilation sans erreur est le suivant :

```
// Chenille.java
package chenille;

public class Chenille {
  // x et y représente les coordonnées de la tête de la chenille
  public Chenille(int nbAnneaux, int x, int y) {
  }

  public void déplacer(int xMax, int yMax) {
  }

  public void dessiner(char[][] t) {
```

```
  }
}
```

Représentation d'une chenille

Une chenille est constituée d'une tête suivie d'une suite d'anneaux. Une tête sera représentée à l'écran par le caractère 't' et un anneau par le caractère 'a'.

Le corps de la chenille est constitué de N anneaux, numérotés de 0 à $N-1$; l'anneau n° 0 est celui qui suit la tête et l'anneau n° $N-1$ est celui de queue.

Le cap de la chenille indique la direction dans laquelle se déplace la tête de la chenille. C'est l'angle entre l'axe Ox et le vecteur indiquant la direction de déplacement de la tête. Dans notre animation textuelle, le cap de la chenille prendra des valeurs multiples de 45° donnant ainsi 8 directions possibles et les coordonnées des anneaux et de la tête prendront des valeurs entières.

A l'instant initial, la chenille est parallèle à l'axe Ox, orientée vers l'est, et sa tête est située au milieu de la fenêtre de dessin (représentée par le tableau de caractère dans le programme principal). En d'autres termes, sa tête est en $(XMAX/2, YMAX/2)$, l'anneau n° 0 en $((XMAX/2)-1, YMAX/2)$, etc.

À chaque étape de l'animation le déplacement de la chenille s'effectue de la manière suivante :
- Si la chenille est sur un bord de la fenêtre, la chenille effectue un demi-tour sur elle même.
- Dans le cas contraire, la chenille effectue une déviation de cap d'un angle tiré au hasard parmi l'ensemble $\{-45°, 0°, +45°\}$.
- Les anneaux de la chenille se décalent d'une position :
  - l'anneau n° i prend la position de l'anneau n° $i-1$ ($i = N-1, N-2, \ldots, 1$)
  - l'anneau n° 0 prend la position de la tête
- La tête se déplace d'une case selon la direction définie par le nouveau cap.

Pour modéliser une chenille dans notre programme, quatre classes d'objets vont être employées :
- Une classe `géométrie.Direction` qui vous est fournie et qui décrit le cap que peut prendre la tête de la chenille. Cette classe propose les méthodes utiles pour le déplacement de la tête. La partie publique de cette classe est donnée en annexe et le fichier Java est sur le serveur commun.
- Une classe `Anneau` qui décrit les objets représentant chaque anneau de la chenille.
- Une classe `Tête` qui décrit l'objet représentant la tête de la chenille. La tête peut être vue comme un cas particulier d'anneau. La classe `Tête` sera donc définie en héritant de la classe `Anneau` (autrement dit la classe `Tête` sera une sous-classe de la classe `Anneau`).
- une classe `Chenille` qui décrit un objet modélisant une chenille dans son ensemble et donc qui utilisera les services des classes `Tête` et `Anneau`.

La classe **Anneau**

Un objet de type `Anneau` est défini par deux attributs (les coordonnées x et y). Il doit être capable de se placer à une position donnée et de se dessiner au sein d'un tableau de caractère.

2. Écrivez la classe `Anneau` au sein d'un paquetage nommé `chenille`.

Solution:

```
// Anneau.java
package chenille;

public class Anneau {
  private int x, y;
  private static final char SYMBOLE = 'a';

  public Anneau(int x, int y) {
    placerA(x, y);
```

```
  }

  public void placerA(int x, int y) {
    this.x = x;
    this.y = y;
  }

  public void dessiner(char[][] t) {
    t[x][y] = SYMBOLE;
  }
}
```

La classe `Tête`

La classe `Tête` est définie comme une sous-classe de la classe `Anneau`. En effet comme un anneau, une tête est définie par deux attributs (sa position) et est capable de se placer à une position donnée et de se dessiner. Les différences sont les suivantes :

— La classe `Tête` enrichit la classe `Anneau` d'un nouvel attribut : `cap` une référence vers un objet de type `géométrie.Direction` qui définit la direction de déplacement de la tête.
— La position et le cap initial d'une tête sont fixés à sa création (donnés en paramètre du constructeur) mais la valeur du cap pouvoir être optionnelle (`Direction.EST` par défaut).
— La méthode de dessin est spécialisée de manière à afficher le caractère propre à une tête (`'t'`).

De plus, une nouvelle méthode `déplacer(int xMax, int yMax)` est ajoutée. Cette méthode réalise le traitement suivant :

1. Le cap est modifié selon la position actuelle de la tête.
   — Si elle est sur les bords de la fenêtre (sa position en x est égale à 0 ou `xMax` ou sa position en y est égale à 0 ou `yMax`), son cap est inversé (voir la méthode `Direction.inverser()`).
   — Dans le cas contraire, son cap est modifié aléatoirement en lui ajoutant une déviation de $-45°$, $0°$ ou $45°$ (voir la méthode `Direction.dériver()`).
2. La position de la tête est mise à jour en lui appliquant un déplacement dans la direction définie par le cap (voir les méthodes `getDx()` et `getDy()` de la classe `Direction`).

3. Programmez la classe `Tête` au sein d'un paquetage `chenille`. Cela peut vous conduire à enrichir la classe `Anneau` de nouvelles méthodes.

Solution: Nous commençons par retravailler la classe `Anneau`. L'usage de la constante `SYMBOLE` ne nous permet pas de réemployer la méthode `dessiner` au sein de la classe `Tête`. Une refonte simple du code va nous permettre de la réemployer. Il est nécessaire de pouvoir accéder à la position d'un anneau, des accesseurs sont mis en place.

```
// Anneau.java
package chenille;

public class Anneau {
  ...
  // suppression de la constante SYMBOLE
  public char getSymbole() {
    return 'a';
  }

  public void dessiner(char[][] t) {
    t[x][y] = getSymbole();
  }
```

```
  public int getX() {
    return x;
  }

  public int getY() {
    return y;
  }
}

// Tête.java
package chenille;

import géométrie.Direction;

public class Tête extends Anneau {
  private Direction cap;

  public Tête(int x, int y, Direction cap) {
    super(x, y);
    this.cap = cap;
  }

  public Tête(int x, int y) {
    this(x, y, Direction.EST);
  }

  public void déplacer(int xMax, int yMax) {
    if (getX() == 0 || getX() == xMax ||
          getY() == 0 || getY() == yMax)
      cap = cap.inverser();
    else
      cap = cap.dériver(1);
    placerA(getX() + cap.getDx(), getY() + cap.getDy());
  }

  public char getSymbole() {
    return 't';
  }
}
```

La classe `Chenille`

La classe `Chenille` définit deux attributs pour une chenille :

— sa tête,
— la liste de ses anneaux stockée dans un tableau,

Elle possède un constructeur permettant à partir du nombre d'anneaux et de la position de la tête donnés en paramètre de créer une chenille en position initiale (horizontale avec les anneaux s'étendant vers l'ouest et la tête orientée à l'est).

Elle possède également 2 méthodes publiques :

— `dessiner(char[][] t)` pour dessiner la chenille,
— `déplacer(int xMax, int yMax)` pour faire effectuer à la chenille un déplacement élémentaire selon l'algorithme donné ci-dessus.

4. Complétez la classe `Chenille` obtenue dans la question 1.

Solution:

```
// Chenille.java
package chenille;

public class Chenille {
  private Tête tête;
  private Anneau[] anneaux;

  public Chenille(int nbAnneaux, int x, int y) {
    tête = new Tête(x, y);
    anneaux = new Anneau[nbAnneaux];
    for (int i = 0; i < nbAnneaux; ++i)
      anneaux[i] = new Anneau(x - (i + 1), y);
  }

  public void déplacer(int xMax, int yMax) {
    for (int i = anneaux.length - 1; i > 0; --i)
      anneaux[i].placerA(anneaux[i - 1].getX(), anneaux[i - 1].getY());
    anneaux[0].placerA(tête.getX(), tête.getY());
    tête.déplacer(xMax, yMax);
  }

  public void dessiner(char[][] t) {
    for (Anneau a : anneaux)
      a.dessiner(t);
    tête.dessiner(t);
  }
}
```

5. Exécutez le programme principal. Si vous en avez le temps, vous pourrez le modifier pour animer les déplacements de plusieurs chenilles.

Annexe – Extrait (partie publique utile) de la classe `Direction`

Vous pouvez étudier le code du type `Direction` qui vous est fournie. Ce type est une énumération. Ceci indique que seules les instances `Direction.EST`, `Direction.NORD_EST`, etc seront accessibles (et aucun autre objet de type Direction ne peut être instancié). Toutefois, des méthodes (`inverser`, `dériver`, etc) permettent quelques manipulations.

```
package géométrie;

public enum Direction {
  EST, NORD_EST, NORD, NORD_OUEST, OUEST, SUD_OUEST, SUD, SUD_EST;

  /** Retourne une direction inverse à la direction courante.
   * @return la direction opposée
   */
  public Direction inverser();

  /** Retourne une direction dérivée par rapport à la
   * direction courante d'un angle aléatoirement choisi et
   * compris entre (-marge * 45°) et (+marge * 45°). La dérivation
   * est choisie nécessairement pour être un multiple de 45°.
   * @param marge la tolérance
```

```
    * @return la nouvelle direction
    */
  public Direction dériver(int marge);

  /** Retourne le déplacement sur l'axe des x correspondant à la
    * direction courante.
    * @return le déplacement en x
    */
  public int getDx();

  /** Retourne le déplacement sur l'axe des y correspondant à la
    * direction courante.
    * @return le déplacement en y
    */
  public int getDy();
}
```

Un programme illustrant l'utilisation du type `Direction` vous est fourni.