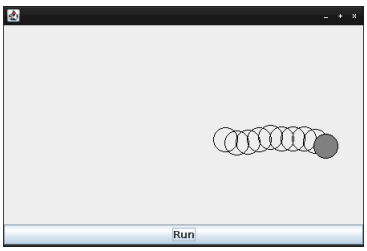
Licence MASS – INF F3 – TP n° 4

2012–2013

Ce TP s'intéresse au dessin dans une fenêtre graphique et utilise l'héritage pour spécialiser un comportement.



1 But du TP

Le but de ce TP est d'animer une chenille à l'intérieur d'une fenêtre. La chenille est constituée d'anneaux liés les uns aux autres et guidés par une tête. La tête étant donc un anneau particulier, elle sera une sous classe des anneaux.

Pour gérer le dessin, nous nous intéresserons aux capacité de la classe `java.awt.Canvas`.

2 La chenille

2.1 Les anneaux

Les éléments de base constituant la chenille sont ses anneaux. Ils peuvent se déplacer en suivant les directions données par la tête. À chaque déplacement, un anneau prend la place de l'anneau le précédant. Chaque anneau doit donc être capable de donner sa position et de se déplacer à une position donnée.

Écrivez une classe Anneau qui implémente ces contraintes. Un anneau étant défini par les coordonnées de son centre ainsi que son rayon.

2.2 La tête

La tête est un anneau particulier. En plus des contraintes liées aux anneaux, la tête est responsable du déplacement de la chenille ainsi que de sa direction. Le déplacement se fait à l'intérieur d'une surface rectangulaire.

Écrivez une classe Tete qui étend Anneau. Cette classe permet de gérer un cap en plus d'une position.

Écrivez la méthode `avancer(int Xmax, int Ymax)` pour la classe tête qui va déplacer la tête de la chenille tout en veillant à rester à l'intérieur du rectangle défini par les coordonnées maximales. Le déplacement s'effectue comme suit :

> Si la tête est trop près d'un bord, elle change de cap de 90 ° , sinon elle change d'un cap aléatoire entre -30 ° et +30 ° . Ensuite elle avance selon la formule `x += cos(cap) * rayon; y += sin(cap) * rayon`.

2.3 La chenille

Une chenille est composée d'un tête et d'un corps constitué d'un nombre fini d'anneaux.

Écrivez la classe chenille qui possède ces attributs ainsi qu'une méthode lui permettant d'avancer. Ce déplacement fonctionne comme suit :

La tête avance. Puis, pour tous les anneaux suivants, l'anneau prend la position que l'anneau précédent avait avant de se déplacer (n'oubliez pas que la tête est aussi un anneau).

La méthode `avancer(...)` de la tête nécessitant les bornes de la zone de dessin, cette méthode doit aussi les prendre en paramètre.

3 Canvas

Un canvas représente une zone de dessin. Il s'agit d'un composant qu'il est possible d'inclure dans une fenêtre. Pour faciliter le dessin des chenilles, nous allons définir un nouveau type de canvas qui va gérer un nombre (variable, mais on va commencer par un) de chenilles.

Écrivez une classe CanvasChenille qui étend Canvas et qui permet de stocker une (ou plusieurs) chenille.

3.1 Dessins

Pour faciliter le travail pour notre Canvas, nous allons déléguer le dessin des anneaux aux Anneaux eux mêmes. Pour ce faire, nous allons utiliser la méthode `drawOval(int x, int y, int width, int height)` de la classe Graphics. Cette méthode dessine un ovale dans un rectangle de taille width×height dont le coin suppérieur gauche est aux coordonnées (x, y). Attention, l'origine du canvas est en haut à gauche et l'axe y est vers le bas.

Écrivez la méthode `dessine(Graphics g)` pour la classe Anneau qui dessine le cercle correspondant à cet anneau.

Surchargez cette méthode dans la classe Tete pour dessiner un anneau plein. Les méthodes `fillOval` et `setColor` de la classe Graphics seront utiles.

Ajoutez la méthode `dessine(Graphics g)` dans la classe Chenille pour dessiner tous les anneaux la composant.

Dans la classe CanvasChenille, surchargez la méthode `paint(Graphics g)` pour ajouter au comportement originel le dessin de la chenille stockée dans cette classe. Avant ou après le dessin, la chenille doit être déplacée pour que le prochain dessin ne représente pas tout à fait la même scène. Pour fournir les bornes de la zone de dessin à la méthode avancer de la chenille, les méthodes `getWidth()` et `getHeight()` vous seront utiles.

4 Interface graphique

Créez une classe visuelle qui contient un CanvasChenille et un bouton pour lancer l'animation. Si vous n'arrivez pas à ajouter directement le CanvasChenille à la fenêtre, vous pouvez ajouter un Canvas simple et remplacer, ensuite, Canvas par CanvasChenille dans le code qui a été généré.

Afin de ne pas complètement figer la fenêtre en lançant une animation infinie, le code de l'actionPerformed associé au bouton devra ressembler à :

```
public void actionPerformed(ActionEvent arg0) {
  Thread t = new Thread() {
    public void run() {
      while (true) {
        canvas.repaint();
        try {
          Thread.sleep(500);
        } catch (InterruptedException e) {
          e.printStackTrace();
        }
      }
    }
  };
  t.start();
```

```
}
```

5 Et ensuite ?

Pour aller plus loin, vous pouvez créer différents types de chenilles : des chenilles colorées, des chenilles qui gagnent ou perdent des anneaux quand elles touchent un bord.

Vous pouvez aussi gérer l'ajout d'une nouvelle chenille à chaque clic sur le bouton. Pour ce faire, écrivez une méthode `ajouterChenille(...)` dans la classe CanvasChenille et modifiez l'actionPerformed pour ne lancer le thread que lors de la création de la première chenille.