Architecture Logicielle – TP 2 2022–2023

Les patrons de *construction* – partie 1

Exercice 1 – *Affichage de dessins vectoriels*

Dans l'archive associée à ce sujet, vous trouverez un programme permettant d'afficher un dessin sous forme vectoriel. Ces dessins peuvent être composés de lignes, cercles et rectangles. Ces différents concepts sont déjà implémentés. Pour former un dessin, on créera une liste contenant des instances des classes `Line, Rectangle, Circle` que l'on passera à la méthode `draw()` d'une instance de `GraphicViewer` qui ouvre une fenêtre et y affiche le graphique.
Pour ajouter de nouvelles formes dessinables par un `GraphicViewer`, celles-ci doivent implémenter l'interface `Drawable` :

```
1  package shapes;
2  import java.awt.Graphics2D;
3
4  public interface Drawable {
5    public void draw(Graphics2D screen);
6  }
```

La classe Test contient une méthode `getDemo()` qui renvoie un dessin construit de manière procédurale que vous pouvez admirer dès à présent en l'exécutant :

```
1  package client;
2  import java.awt.event.*;
3  import java.awt.image.*;
4  import java.awt.geom.*;
5  import java.awt.Color;
6  import java.util.*;
7
8  import shapes.*;
9  import viewer.GraphicViewer;
10
11 public class Test {
12   public static java.util.List<Drawable> getDemo() {
13     java.util.List<Drawable> ls=new ArrayList<Drawable>();
14     ls.add(new Line(0, 500, 800, 500, Color.GREEN));
15     ls.add(new Line(300, 0, 0, 300, Color.YELLOW));
16
17     ls.add(new Line(30, 300, 180, 200, Color.BLUE));
18     ls.add(new Line(330, 300, 180, 200, Color.BLUE));
19     ls.add(new Rectangle(30, 300,330, 500, Color.RED));
20
21     double sunX = 600;
22     double sunY = 120;
23     double sunRad = 60;
24     ls.add(new Circle(sunX, sunY, sunRad, Color.BLACK));
25     int sunRay = 20;
26     for (int i=0; i<sunRay; ++i) {
```

```
27          double tau=i*2*Math.PI/sunRay;
28          ls.add(new Line(sunX+(sunRad+5)*Math.cos(tau),
29            sunY-(sunRad+5)*Math.sin(tau),
30            sunX+(1.5*sunRad+5)*Math.cos(tau),
31            sunY-(1.5*sunRad+5)*Math.sin(tau),
32            Color.BLACK));
33        }
34
35        double manX=600;
36        double manY=450;
37        ls.add(new Line(manX, manY-70, manX-40, manY-110, Color.RED));
38        ls.add(new Line(manX, manY-70, manX+40, manY-110, Color.RED));
39        ls.add(new Circle(manX, manY-120, 20, Color.GRAY));
40        ls.add(new Line(manX, manY, manX, manY-100, Color.BLUE));
41        ls.add(new Line(manX, manY, manX-20, manY+50, Color.BLACK));
42        ls.add(new Line(manX, manY, manX+20, manY+50, Color.BLACK));
43
44        return ls;
45      }
46
47      public static void main(String[] args)  {
48        GraphicViewer gv = new GraphicViewer();
49        java.util.List<Drawable> demo=getDemo();
50        gv.draw(demo);
51      }
52  }
```

Problème : On souhaite maintenant ajouter un mode de rendu supplémentaire qui donnerait un style de dessin à main levée lors du rendu en introduisant des approximations dans les paramètres des composants de dessin. Pour cela, nous allons ajouter aléatoirement des approximations dans les paramètres des figures (points, rayons, etc.) en nous aidant de la classe utilitaire `Noise`. Celle-ci contient deux méthodes statiques :

- `getNoise()` : renvoie une valeur entière aléatoire comprise entre -5 et 5,
- et `getNoise(double l)` : renvoie une valeur entière aléatoire comprise entre $-\frac{l}{5}$ et $\frac{l}{5}$.

Le bruit (l'aléa) ne peut pas être généré au moment du rendu, sinon la figure changerait d'aspect à chaque affichage. Par conséquent, il nous faut mémoriser ces nouveaux paramètres dans chaque (chaque objet de type) figure. Ainsi, le nouveau style de rendu sera obtenu en introduisant les classes suivantes :

- `HandLine` : celle-ci utilisera la classe `QuadCurve2D` (courbe de Bézier) pour le tracé tel que l'on déplacera aléatoirement de quelques pixels (`getNoise(void)`) l'origine et l'arrivée du segment de droite et on placera un point de contrôle au milieu du segment, lui même décalé de plus ou moins un nombre aléatoire de pixels en fonction de la longueur du segment (`getNoise(longueurX)`).
- `HandRectangle` : on utilisera simplement quatre `HandLine` pour cela,
- `HandCircle` : on utilisera une ellipse à la place d'un cercle (deux rayons différents) et on ajoutera sur à chaque rayon un nombre aléatoire de pixels fonction de la longueur du rayon (avec `getNoise(rayonX)` et `getNoise(rayonY)`).

Évidemment, afin de pouvoir utiliser les nouvelles classes, il sera nécessaire de modifier la méthode `getDemo()`. Cependant on souhaite pouvoir faire en sorte que toute fonction produisant un dessin, à l'instar de `getDemo()`, puisse à l'avenir construire son dessin indépendamment du mode de rendu (classique ou à main levée ou bien encore un mode de rendu à venir).

Question 1.1 : Donnez un diagramme UML du code existant.

Question 1.2 : Quel patron de conception proposez-vous d'utiliser pour répondre à notre problème. Rappelez son schéma de principe.

Question 1.3 : Donnez le schéma UML de votre solution en identifiant tous les acteurs. Pour cela, commencez

par déterminer le nombre et le type de produits différents, puis leurs sous-variants. Note : il sera (très) sûrement souhaitable d'apporter des changements à l'architecture actuelle pour appliquer le patron.
Question 1.4 : Implémentez la solution correspondante.