Programmation orientée objet avec Java

5.1. Java2d

DEI - ENSG

2020-2021

Java2d

Java2d

Dessiner en Java

Plusieurs librairies permettent de faire du dessin en Java (création de jeux, affichage d'images, animations, dessin d'objets, de formes, etc.). **Java2D** est une bibliothèque bas niveau qui permet de dessiner toutes sortes de figures géométriques en 2D. Pour la 3D ou pour des préoccupations de performances, il est préférable d'utiliser plutôt une libraire utilisant OpenGL comme JOGL.
Java2D a aussi l'avantage d'être parfaitement intégré à Swing, la bibliothèque graphique de base en Java pour la création d'interfaces graphiques remplacée depuis 2014 par JavaFX (mais reste toujours très utilisée).

Java2d

Très brève introduction à Swing

Swing utilise trois catégories d'objets.
Les **conteneurs** sont des objets destinés à contenir d'autres objets. Ils sont responsables de la disposition des objets qu'il délèguent à un gestionnaire de disposition (*LayoutManager*) :

- *JWindow* : fenêtre basique sans barre de titre, sans boutons, etc.
- *JDialog* : fenêtre pour boite de dialogue.
- **JFrame** : fenêtre la plus utilisée, destinée à être la fenêtre principale d'une application. Possède une barre de titre, un bouton de fermeture et de redimensionnement, peut accueillir un menu, etc.
- **JPanel** : conteneur utilisé pour regrouper des *composants* ou d'autres conteneurs.

Java2d

Très brève introduction à Swing

Les **composants** sont les éléments graphiques de base : bouton, boite à cocher, élément de liste déroulante, label, etc :

- JRadioButton
- JComboBox,
- JMenu,
- JLabel,
- etc.

Les **classes de liaisons** permettent d'interagir avec l'utilisateur (événements), de disposer les composants dans un conteneur (Layout), etc.

Une fenêtre très simple

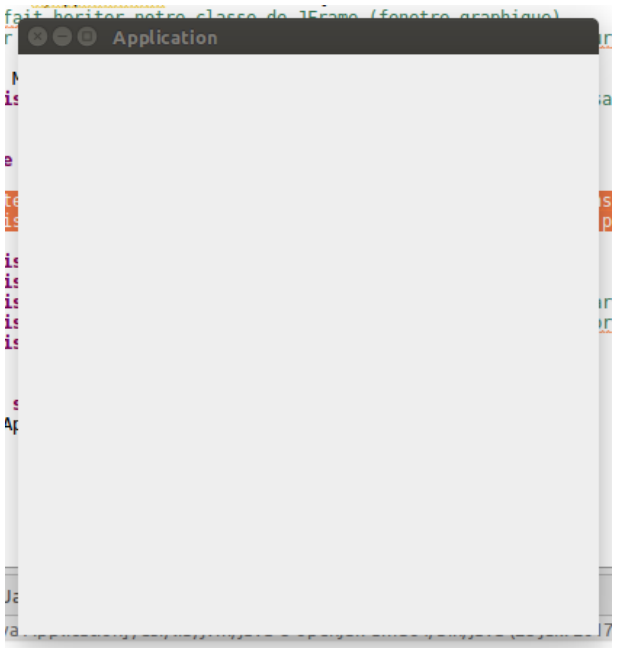
Il peut être intéressant de créer une classe héritant de JFrame plutôt que d'instancier directement un nouvel objet JFrame, afin de personnaliser son comportement via la surcharge éventuelle des ses méthodes.

Une fenêtre très simple

```
1  public class MyApplication extends JFrame{
2     // on fait heriter notre classe de JFrame (fenetre graphique)
3     // pour pouvoir eventuellement personnaliser son comportement (surcharge)
4
5    public MyApplication(){
6       this.init(); // on separe le constructeur du code d' initialisation
7       // des parametres graphiques (bonne pratique)
8    }
9
10   private void init(){
11      this.setTitle("Test fenetre"); // titre de la fenetre
12      this.setSize(400, 400); // taille de la fenetre. On utilise plutot
13      // setPreferredSize si le composant parent utilise un LayoutManager.
14      this.setLocationRelativeTo(null); // positionnement centre
15      // par rapport a l'ecran
16      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // comportement
17      // lors d'un clic sur la croix rouge
18      this.setVisible(true); // on la rend visible
   }
```

Une fenêtre très simple

```
1 public static void main(String args []) {
2     MyApplication myApp = new MyApplication();
3 }
```



Une fenêtre simple.

Dessiner dans un JPanel

JPanel et dessin

Le dessin avec Java2D se fait via la méthode **paintComponent()**, méthode définie dans la classe *JComponent* dont hérite entre autre JPanel (mais aussi tous les autres composants à l'exception des conteneurs de haut niveau : JFrame,n JDialog,etc.). Elle permet d'ajouter des instructions de dessins dans un composant. Elle est appelée à chaque manipulation du panel : redimensionnement, clic, déplacement, etc.

Dessiner dans un JPanel

JPanel et dessin

L'outil de dessin est le **contexte graphique**, objet de la classe *Graphics* ou de sa classe dérivée *Graphics2D* qui encapsule toute l'information nécessaire au dessin :

- le trait, la couleur,
- les formes,
- une transformation affine des coordonnées,
- etc.

Pour dessiner, on va créer une classe héritant de JPanel et surcharger sa méthode *paintComponent*. Puis il suffira d'ajouter ce nouveau conteneur dans notre fenêtre JFrame !

Dessiner dans un JPanel

```
1  public class MyPanel extends JPanel {
2     // notre classe sera un conteneur, destine a accueillir
3     // pleins d'autres composants graphiques
4
5     @Override
6     public void paintComponent(Graphics g) {
7        // on redefinit la methode paintComponent,
8        // qui precise la facon dont le dessin est effectue dans le panel
9
10       super.paintComponent(g); // Appel de la methode
11       // paintComponent de la classe mere
12
13       // Graphics est un objet fourni par le systeme
14       // qui est utilise pour dessiner les composant du conteneur
15       Graphics2D g2d = (Graphics2D) g; // on cast en Graphics2D,
16       // objet permettant des manipulations plus evoluees
```

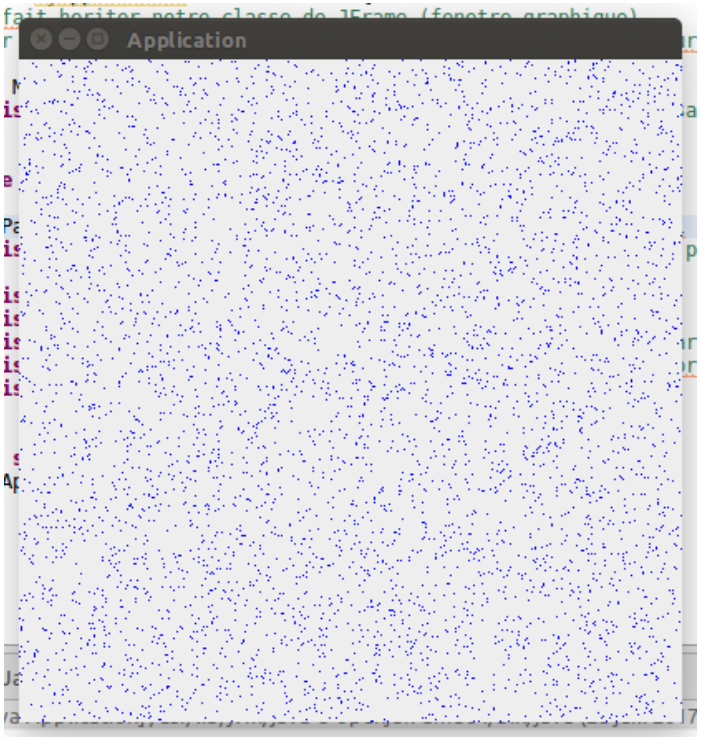
Dessiner dans un JPanel

```
1
2         g2d.setPaint(Color.blue); // on va dessiner en bleu
3         int w = this.getWidth(); // on recupere la largeur du conteneur
4         int h = this.getHeight(); // on recupere sa hauteur
5
6         for(int i=0; i< 5000; i++){
7             // on va tirer aleatoirement 5000 points
8             Random r = new Random();
9             int x = Math.abs(r.nextInt()) % w;
10            int y = Math.abs(r.nextInt()) % h;
11
12            // un point est en fait une ligne de longueur nulle
13            g2d.drawLine(x, y, x, y);
14        }
15    }
16 }
```

Dessiner dans un JPanel

```
1
2  public class MyApplication extends JFrame {
3
4      private void init () {
5
6          // on instancie un nouveal objet MyPanel
7          MyPanel mainPanel = new MyPanel();
8
9          // on redefinit le conteneur principal de notre fenetre
10         this.setContentPane(mainPanel);
11
12         // ...
13     }
14
15 }
```

Une fenêtre très simple



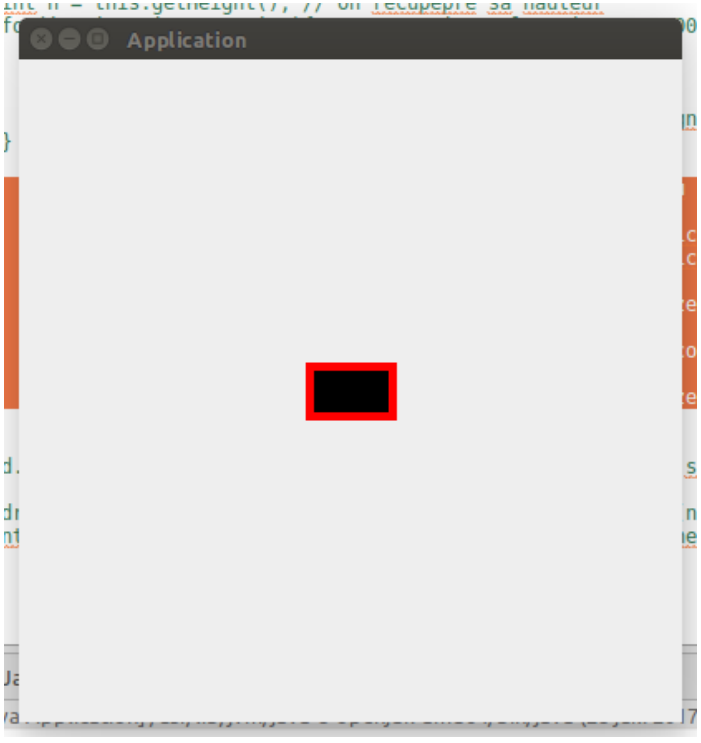
Une dessin simple : 5000 points bleus tirés aléatoirement.

Dessiner dans un JPanel

Affichage d'un rectangle un peu particulier : dessin du contour et de l'intérieur fait séparément !

```
1  public class MyPanel extends JPanel {
2      @Override
3      public void paintComponent(Graphics g) {
4          // ...
5          int w = this.getWidth();
6          int h = this.getHeight();
7          int sizeW = 50; // largeur du rectangle qu'on va afficher
8          int sizeh = 30; // hauteur du rectangle qu'on va afficher
9          g2d.setPaint(Color.black); // couleur de l'interieur
10         // dessin de l'interieur
11         g2d.fillRect(w/2 - sizeW/2, h/2 - sizeh/2, sizeW, sizeh);
12         g2d.setPaint(Color.red); // couleur du contour
13         BasicStroke bs1 = new BasicStroke(5); // pinceau du contour : taille 5
14         g2d.setStroke(bs1);
15         g2d.drawRect(w/2 - sizeW/2, h/2 - sizeh/2, sizeW, sizeh); // dessin du cont
16             }
}
```

Une fenêtre très simple



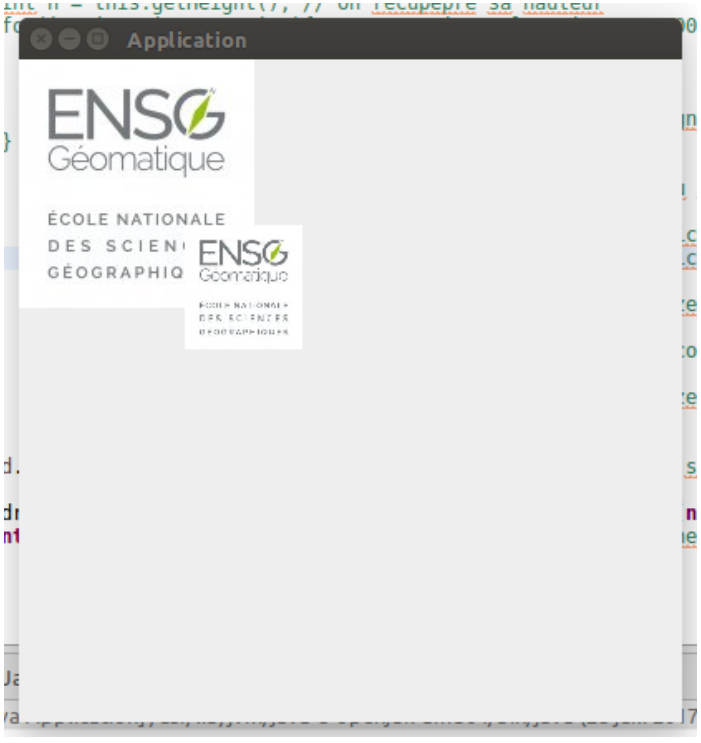
Affichage d'un rectangle un peu particulier.

Dessiner dans un JPanel

Affichage d'une image :

```
1  public class MyPanel extends JPanel {
2      private Image image; // attribut de type Image
3      public MyPanel() {
4          // on cree une icone de type image
5          ImageIcon ii = new ImageIcon("/home/glagaffe/logo-ensg.jpg");
6          this.image = ii.getImage(); // on recupere l'Image de l'icone
7      }
8      @Override
9      public void paintComponent(Graphics g) {
10         // ...
11         g2d.drawImage(this.image, 0, 0, null); // on affiche l'image sans
12         // redimentionnement, en haut a gauche du panel
13
14         g2d.drawImage(this.image, 0, 0, (int)(this.image.getWidth(null) * 0.5),
15         (int) (this.image.getHeight(null) * 0.5), null);
16         // on affiche l'image avec facteur d'echelle (*0.5)
17     }
}
```

Une fenêtre très simple



Affichage d'une image.

Programmation événementielle

Capture d'événements

Swing permet la capture et la gestion d'événements : clic souris, clavier, changement de valeur d'une liste déroulante, etc.
Les événements sont gérés par plusieurs interfaces *EventListener* permettant de définir les traitements en réponse à des événements générés par un composant :

- ActionListener : clic gauche de souris sur un élément ou enfoncement de la touche entrée,
- ItemListener : utilisation d'une liste déroulante ou d'une case à cocher,
- MouseMotionListener : déplacement du curseur de la souris,
- MouseListener : différents clics, clic enfoncé, relâché, etc.
- KeyListener : touches du clavier (appuyée, relâchée, etc.).

Programmation événementielle

Capture d'événements

Chacune de ces interfaces dispose de méthode qu'il faut redéfinir : mouseClicked(), keyPressed(), actionPerformed(), etc.
Pour demander à un composant d'écouter un événement, par exemple via ActionListener, il y a plusieurs méthodes :

- créer une classe interne implémentant ActionListener puis utiliser la méthode addActionListerner() du composant,
- utiliser directement addActionListener avec une classe anonyme en paramètre,
- spécifier que la classe manipulée implémente ActionListener.

Programmation événementielle

```
1  // implementation de l'interface ActionListener
2  public class MyButton extends JButton implements ActionListener {
3
4      public MyButton() {
5          this.setPreferredSize(new Dimension(20,20));
6          // on declare que le listener c'est l'objet lui-meme
7          this.addActionListener(this);
8      }
9
10     @Override
11     public void actionPerformed(ActionEvent e) {
12         System.out.println("Clic !");
13     }
14 }
```

Programmation événementielle

```
1  public class InteractivePanel extends JPanel implements KeyListener{
2          private int x, y; // coordonnees d'un point que l'on va pouvoir deplacer
3          public InteractivePanel(){
4                  this.setBackground(Color.BLACK); // fond noir
5                  this.setFocusable(true); // sinon par defaut le panel n'a pas le focus : on ne peut pas interagir av
6                  this.addKeyListener(this); // on declare que this ecoute les evenements clavier
7                  this.x = 0;
8                  this.y = 0;
9          }
10         public void paintComponent(Graphics g) {
11                 super.paintComponent(g);
12                 Graphics2D g2d = (Graphics2D) g;
13                 g2d.setPaint(Color.red); // dessin en rouge
14                 g.fillRect(this.x, this.y, 5,5); // pour mieux voir on fait un rectangle plutot qu'un point
15         }
16         @Override
17         public void keyPressed(KeyEvent e) { // pour implementer KeyListener
18                 int key = e.getKeyCode(); // on recupere un code associe a la touche sollicite
19                 if ((key == KeyEvent.VK_LEFT)) { // cas fleche de gauche
20                         x -= 1;
21                 }
22                 if ((key == KeyEvent.VK_RIGHT)) {
23                         x += 1;
24                 }
25                 if ((key == KeyEvent.VK_UP)) {
26                         y -= 1;
27                 }
28                 if ((key == KeyEvent.VK_DOWN)) {
                           y += 1;
                   }
                   repaint(); // repaint force l'appel a paintComponent()
```

Java2d

Dessiner en Java

Bien entendu il est possible de faire énormément de choses : gérer la transparence, effectuer des transformations mathématiques, détecter des collisions entre objets, faire des jeux, gérer des paramètres graphiques (anti-aliasing), etc.