Fiche 9 : affichage et interaction

IPO – Introduction à la programmation objet

Thibaut Balabonski @ Université Paris-Saclay
https://www.lri.fr/~blsk/IPO/

V1, automne 2022

Les bibliothèques Swing et AWT permettent de bâtir en Java une interface graphique interactive. Premier tour d'horizon.

Fenêtre graphique Une application graphique est matérialisée une *fenêtre*, s'ouvrant dans l'interface graphique de votre système d'exploitation, généralement constituée d'une barre de titre avec quelques boutons (fermeture, agrandissement, réduction) et d'un cadre. La classe `JFrame` matérialise une telle fenêtre en Java. L'aspect extérieur de la fenêtre (forme du cadre, bouton, barre de titre) est lié au système d'exploitation. On ne s'intéressera ici qu'au contenu.

Le constructeur d'une `JFrame` prend en paramètre la chaîne à afficher dans la barre de titre.

```
public class Bulles {
  public static void main(String[] args) {
    JFrame frame = new JFrame("Bulles");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    ...
```

Le contenu principal de la fenêtre est regroupé dans un *conteneur* que l'on peut récupérer avec la méthode `getContentPane()`. Ce contenu est consituté de différents éléments graphiques : boutons, zones de texte, affichage...

Un élément graphique multi-fonctions : JPanel La classe `JPanel` définit l'un des éléments pouvant être affiché dans une fenêtre, qui est pratique à la fois pour dessiner directement et pour regrouper et organiser d'autres éléments. On peut inclure un `JPanel` dans une fenêtre en l'ajoutant à son `contentPane`.

```
    JPanel panel = ...
    frame.getContentPane().add(panel);
```

On termine ensuite la mise en place de la fenêtre en la dimensionnant autour de son contenu (`pack()`), et en la déclarant visible.

```
    frame.pack();
    frame.setVisible(true);
```

Il n'y a plus qu'à inclure du contenu dans notre `JPanel` pour l'afficher dans la fenêtre.

Une technique simple pour réaliser un affichage arbitraire consiste à définir une nouvelle classe étendant `JPanel`, en y incluant les éléments qui nous intéressent.

```
class Aquarium extends JPanel {
  private ArrayList<Bulle> bulles;
  ...
```

Parmi les premiers éléments que l'on peut configurer dès la construction d'une zone d'affichage : la taille de la zone, et sa couleur de fond.

```
  public Aquarium(int l, int h) {
    this.bulles = new ArrayList<>();
    setPreferredSize(new Dimension(l, h));
    setBackground(new Color(161, 202, 241));
  }
```

Les éléments graphique comme `JPanel` possèdent une méthode `paintComponent` définissant leur rendu à l'écran. Cette méthode prend en paramètre un objet `g` de la classe `Graphics`, que l'on peut considérer comme un « pinceau » :

— ses attributs mémorisent la configuration de dessin actuelle (notamment : la couleur sélectionnée),
— ses méthodes dessinent des traits ou des formes.

Pour définir l'affichage d'un panneau personnalisé, on redéfinit cette méthode. On peut alors se servir des différentes méthodes de la class `Graphics` pour réaliser des tracés arbitraites.

```
  public void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.setColor(new Color(255, 255, 255, 127));
    for (Bulle b : bulles) {
      g.fillOval(b.x - b.r, b.y - b.r, 2*b.r, 2*b.r);
    }
  }
```

Note : pour ne pas perdre le comportement général d'affichage de la classe mère `JPanel`, on commence par faire un appel à sa méthode d'affichage d'origine avec **`super`**`.paintComponent`. Ceci va notamment mettre en place la couleur de fond, au-dessus de laquelle seront tracés les éléments suivants.

Dans le code principal, on peut alors utiliser cette nouvelle classe pour incarner le `JPanel` à afficher.

```
    JPanel panel = new Aquarium(800, 600);
```

Interaction avec la souris Les interactions en temps réel en Java sont basées sur un système d'*événements* et de *notifications*. Certains objets sont des *sources* d'événements, et d'autres sont des *récepteurs*, qui vont réagir aux événements des sources. Chaque objet récepteur s'inscrit auprès de la source dont il doit suivre les événements. En retour, une source diffuse auprès des récepteurs inscrits sur sa liste les informations de chaque nouvel événement.

— Un chronomètre (classe `Timer`) génère un tic d'horloge à un intervalle de temps périodique. Un récepteur inscrit auprès d'un chronomètre recevra donc une notification et pourra agir à chacun de ces tics périodiques.
— Une fenêtre graphique (classe `JFrame`) génère un événement à chaque frappe d'une touche ou action de la souris. Un récepteur inscrit auprès de la fenêtre pourra donc réagir à de tels événements.

Plusieurs interfaces décrivent les récepteurs capables de réagir à différentes sortes d'événements. Notamment :

— `MouseListener` pour les récepteurs réagissant aux actions de la souris,
— `KeyListener` pour les récepteurs réagissant à l'utilisation du clavier,
— `ActionListener` pour les récepteurs réagissant à un événement générique, comme l'utilisation d'un bouton d'une interface ou un tic d'horloge.

. La classe d'un récepteur doit donc implémenter l'interface correspondante

```
class Aquarium extends JPanel implements MouseListener {
  ...
```

et il faut, à un moment du code principal ou de la construction du système, inscrire chaque objet récepteur auprès de sa ou ses sources, à l'aide de méthodes dédiées de ces dernières.

```
    /* fin de Bulles.main */
    frame.addMouseListener(panel);
  }
```

À chaque nouvel événement, la source enverra des notifications à chaque récepteur ainsi inscrit.

Les « notifications » envoyées par une source à ses récepteurs sont matérialisées par l'appel de méthodes du récepteur. Plus précisément : par les méthodes déclarées dans l'interface du récepteur.

```
/* extrait de la bibliothèque java.awt */
interface MouseListener {
  /* Réaction aux événements de la source : */
  void mouseClicked(MouseEvent e);  /* clic */
  void mouseEntered(MouseEvent e);  /* arrivée du curseur dans la zone */
  void mouseExited(MouseEvent e);   /* sortie du curseur de la zone */
  void mousePressed(MouseEvent e);  /* bouton appuyé */
  void mouseReleased(MouseEvent e); /* bouton relâché */
}
```

Dans la classe d'un récepteur, on définit les méthodes correspondant aux différents événement auxquels on souhaite réagir. Ces méthodes de réaction prennent systématiquement un unique paramètre, de type adapté, décrivant précisément l'événement (par exemple : quelle touche du clavier a été utilisée, ou la position précisé du curseur de la souris). *Note :* si la réaction à un événement implique une modification des éléments à afficher, on peut rafraîchir l'affichage avec la méthode `repaint` de `JPanel`.

```
  public void mouseClicked(MouseEvent e) {
    bulles.add(new Bulle(e.getX(), e.getY(), 40));
    this.repaint();
  }
```

Rappel : pour concrétiser une interface, il faut fournir des définitions à toutes les méthodes de l'interface. Si l'on ne souhaite pas réagir à certains des événements prévus par l'interface, il suffit de définir la méthode correspondante avec un code vide.

```
  public void mousePressed(MouseEvent e) {}
  public void mouseReleased(MouseEvent e) {}
  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
```

Programme complet Tous les éléments précédents peuvent être regroupés dans un unique fichier `Bulles.java` (ou séparés en autant de fichiers que de classes).

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;
import java.util.ArrayList;

public class Bulles {
  public static void main(String[] args) {
    JFrame frame = new JFrame("Bulles");
    Aquarium panel = new Aquarium(800, 600);
    frame.getContentPane().add(panel);
    frame.addMouseListener(panel);
    frame.pack();
    frame.setVisible(true);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
}

class Aquarium extends JPanel implements MouseListener {
  private ArrayList<Bulle> bulles;
  public Aquarium(int l, int h) {
    this.bulles = new ArrayList<>();
    setBackground(new Color(161, 202, 241));
    setPreferredSize(new Dimension(l, h));
  }

  public void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.setColor(new Color(255, 255, 255, 127));
    for (Bulle b : bulles) {
      g.fillOval(b.x-b.r, b.y-b.r, 2*b.r, 2*b.r);
    }
  }

  public void mouseClicked(MouseEvent e) {
    bulles.add(new Bulle(e.getX(), e.getY(), 40));
    this.repaint();
  }

  public void mouseEntered(MouseEvent e) {}
  public void mouseExited(MouseEvent e) {}
  public void mousePressed(MouseEvent e) {}
  public void mouseReleased(MouseEvent e) {}
}

class Bulle {
  public final int x, y, r;
  public Bulle(int x, int y, int r) {
    this.x = x;
    this.y = y;
    this.r = r;
  }
}
```