Rapport CCTP: Programmation concurrente

Ḱamarouzamane Combo, L3 informatique

10 octobre 2017

Résumé

Dans ce rapport, nous allons manipuler des Threads et des Interfaces graphiques.

1 Introduction

Ecrivons un programme (en Java et en Python) qui affche des balles en mouvement dans une fenêtre.

1. Les balles ne peuvent sortir du cadre de la fenêtre et ricochent contre les bords
2. Un bouton permet de démarrer/arrêter le mouvement de toutes les balles. Lorsque les balles sont en mouvement, ce bouton a pour étiquette *"stop"* et lorsque les balles sont à l'arrêt il a pour étiquette *"start"*.
3. Un bouton *"+"* permet d'ajouter une nouvelle balle de couleur aléatoire. Lorsqu'un seuil fixé au préalable est atteint, plus aucune balle ne peut être ajoutée.
4. Un bouton *"-"* permet de supprimer une balle.
5. Lorsqu' une collision se produit, les balles impliquées sont supprimées.
6. Un score est affiché en permanence et est incrémenté à chaque collision.
7. Une horloge affiche le temps écoulé pendant que les balles sont en mouvement ; cette horloge doit stopper son décompte lorque les balles sont arrêtées et reprendre son décompte lorsque les balles sont à nouveau en mouvement.

2 Programmation en Java

Dans cette section, nous allons voire comment gérer les interfaces graphique et les Threads en Java.

2.1 L'interface Graphique

Java possède des librairies de classe dédiées à la conception d'interface graphique :
— **AWT** [1] (Abstract Windowing Toolkit)
— **SWING** [1]
Pour utiliser des classes dans ces librairies il faut placer la commande d'import en début du fichier du code source :
— **import java.awt.*;**
— **import javax.swing.*;**
Il y a 2 types d'éléments qui interviennent dans une interface graphique :
— Les conteneurs : qui servent à contenir d'autres éléments pour les regrouper et les agencer.
— Les composants : qui sont des éléments d'affichage (texte, dessin,...), ou d'interaction (bouton, zone de saisie,...)

NB : un conteneur peut contenir des conteneurs (permet de créer des agencements complexes).

Les fenêtres sont des conteneurs particuliers qui peuvent directement (sans être placés dans autre chose) s'afficher sur l'écran.

Exemple : fenetre vide

```
import javax.swing.*;

public class MaFenetre extends JFrame {
	public MaFenetre (String titre, int x, int y, int w, int h) {
		super(titre);
		this.setDefaultCloseOperation(EXIT_ON_CLOSE);
		this.setBounds(x,y,w,h);
		this.setVisible(true);
	}
	public static void main(String[] args) {
		new MaFenetre ( "Ma première fenêtre ",300,200,500,400);
	}
}
```

Les gestionnaire de mise en page :

• Pour agencer les composants dans un conteneur il faut utiliser un gestionnaire spécifique appelé : LayoutManager

• Il existe plusieurs LayoutManager (gestionnaire layout), chacun offrant des possibilités d'agencement particulier

• On spécifie le gestionnaire d'un conteneur à l'aide de la méthode : **setLayout**(LayoutManager lm)

• Attention les gestionnaires sont dans la librairie AWT, pour les utiliser il faut donc, en début de code, placer la commande : **import java.awt.*;**

• Dans ce gestionnaire la zone est organisée en 5 régions : north, sud, west, east, center.

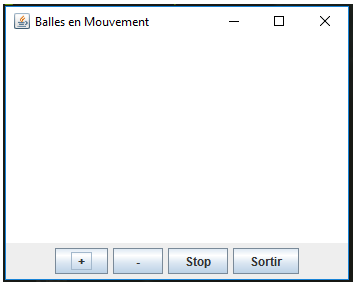
• La position d'insertion est indiquée par une constante passé en argument de la méthode **add** du conteneur :

add(Component c, Object position)

```
avec position =
                    BorderLayout.NORTH
                    BorderLayout.SOUTH
                    BorderLayout.WEST
                    BorderLayout.EAST
                    BorderLayout.CENTER
```

• FlowLayout organise les composants en les plaçant les uns après les autres de la gauche vers la droite, en passant à la ligne suivante quand cela est nécessaire.

- Il n'y a pas d'indication explicite de la position d'insertion.
- Dans le conteneur on ajoute les composants avec la méthode : **add**(Component c)



Gestion des évènements :

- Pour gérer les évènements (les actions) que produit l'utilisateur sur l'interface graphique, il faut en général utiliser des écouteurs d'évènement. On les appelle des Listener
- Dans la librairie **java.awt.event** il y a un certains nombres de classes et d'interfaces qui définissent les principaux listener
- En pratique il faut :

1. instancier, spécialiser ou implémenter le listener dont on a besoin (en fonction du type d'évènement à gérer).
2. enregistrer le listener auprès du composant sur lequel se produit l'évènement méthode du type : **addXXXListener**(Listener l) avec **XXX** un nom relatif au type d'évènement à écouter.

Exemple : les actions sur les boutons

```
import javax.swing.*; import java.awt.*; import java.awt.event.*;

public class MaFenetre extends JFrame implements ActionListener {
       public MaFenetre (String titre, int x, int y, int w, int h) {
              super(titre); this.setDefaultCloseOperation(EXIT_ON_CLOSE);
              Container pane=getContentPane(); pane.setLayout(new FlowLayout());

              JButton b=new JButton("Start");
              b.addActionListener(this);
              pane.add(b); this.setBounds(x,y,w,h); this.setVisible(true);
       }
       public static void main(String[] args) { new MaFenetre ( "Test",300,200,500,400); }
       public void actionPerformed(ActionEvent e){
              JOptionPane.showMessageDialog(this, "ACTION !") ;
       }
}
```

2.2 Thread

Dès que vous cliquez sur le bouton " + ", le programme lance la balle à partir du coin supérieur gauche de l'écran et celle-ci commence à rebondir. Le gestionnaire du bouton " + " appelle la méthode **ajoutBalle()**. Cette méthode contient une boucle progressant sur 1000 déplacements. Chaque appel à la méthode **deplace()** déplace légèrement la balle, ajuste la direction si elle rebondit sur un mur, puis redessine l'écran.

La méthode statique sleep() de la classe Thread effectue une pause du nombre de millisecondes donné.

L'appel à **Thread.sleep()** ne crée pas un nouveau thread, **sleep()** est une méthode statique de la classe Thread qui met le **thread** courant en sommeil (5 ms). Bien sûr, vous avez la possiblité

de changer la durée du sommeil pour obtenir l'apparence d'un déplacement plus ou moins rapide. La méthode **sleep()** peut déclencher une exception **InterruptedException**, il faut donc utiliser un bloc **try-catch** si nous désirons la capturer.

Nous reviendrons sur cette exception et sur son gestionnaire un peu plus tard. Pour l'instant, nous terminons simplement le rebond si l'exception survient.

Si vous étudiez attentivement le code, vous remarquez l'appel **panneau.paintComponent(panneau.getGraphics()** dans la méthode **ajoutBalle()**, vous devriez normalement appeler la méthode **repaint()** et laisser le soin à AWT d'obtenir le contexte graphique et de redessiner l'écran. Toutefois, si vous essayez de réaliser cet appel dans ce programme, vous découvrirez que l'écran n'est jamais redessiné puisque la méthode **ajoutBalle()** a totalement pris le pas sur le traitement.

Remarquez également que le composant de balle étend **JPanel**, ce qui simplifie l'effacement de l'arrière-plan. Dans le programme suivant, nous utiliserons un autre thread pour calculer la position de la balle et nous emploierons à nouveau la méthode **repaint()** de **JComponent**.

Chaque thread ayant une occasion de s'exécuter, le thread de répartition d'événements peut savoir si l'utilisateur clique sur le bouton **"Sortir"** alors que les balles rebondissent. Il peut alors traiter l'action correspondante.

Procédure à suivre pour mettre en oeuvre un thread séparé

Voici une procédure très simple pour exécuter du code dans un thread séparé :

1. Placez le code dans la méthode run() d'une classe qui implémente l'interface Runnable. Cette interface est très simple, elle ne possède que la méthode run() : L'interface java.lang.Runnable

   ```
   void run() :
   Cette méthode doit être redéfinie et vous devez y ajouter les instructions qui doivent être ex
   ```

   Cela peut-etre implémenté comme ceci :

   ```
   class MaClasse implements Runnable {
          public void run() {
                 // code de la tâche à réaliser
          }
   }
   ```

2. Construisez un objet de votre classe :

   ```
   Runnable interface = new MaClasse();
   ```

3. Construisez un objet de la classe Thread à partir de l'interface Runnable :

   ```
   Thread tâche = new Thread(interface);
   ```

4. Démarrer le thread :

   ```
   tâche.start();
   ```

Mise en oeuvre sur les balles rebondissantes

Pour exécuter notre programme dans un thread séparé, nous devons simplement implémenter une classe **BalleSeparee** et placer le code de l'animation dans la méthode **run ()**, comme dans le code suivant :

```
private class BalleSeparee implements Runnable
     {
        private Balle balle;

        public BalleSeparee(Balle balle)
        {
            this.balle = balle;
        }

        public void run()
        {
```

```
            try
            {
               while(true)
               {
                  balle.deplace(panneau.getBounds());
                  panneau.repaint();
                  Thread.sleep(20);
               }
            }
            catch (InterruptedException ex) { }
         }
      }
```

Pause et reprise d'une animation

Nous allons reprendre notre application sur les balles rebondissantes. Cette fois-ci, nous contrôlons parfaitement l'animation des balles déjà présentes. Il est possible maintenant de faire une pause et ainsi de figer l'ensemble des balles et de les relancer par la suite lorsque nous le désirons.

Un sémaphore avec un compteur d'autorisation de 1 peut servir à synchroniser l'ensemble des threads représentant le mouvement des balles. Si nous demandons de mettre en pause avec le bouton **"Pause"**, chaque thread de l'animation appelle la méthode **acquire()**. Quand plus tard l'utilisateur clique sur le bouton "Relancer", le thread d'interface utilisateur appelle la méthode **release()**, en spécifiant le nombre de threads à libérer (correspondant au nombre de balles déjà présentes). Ainsi toutes les balles redeviennent animées.

3 Programmation en Python

Cette fois-ci, nous allons voire comment gérer les interfaces graphique et les Threads en Python.

3.1 L'interface Graphique

Interface graphique avec le module **Tkinter** [2]

Le module **Tkinter** ("Tk interface") de Python permet de créer des interfaces graphiques (GUI : graphical user interface). De nombreux composants graphiques (ou widgets) sont disponibles : fenêtre (classe Tk), bouton (classe Button), case à cocher (classe Checkbutton), étiquette (classe Label), zone de texte simple (classe Entry), menu (classe Menu), zone graphique (classe Canvas), cadre (classe Frame)... On peut gérer de nombreux événements : clic sur la souris, déplacement de la souris, appui sur une touche du clavier, top d'horloge...

4 Conclusion

Références

[1] Swing. https://docs.oracle.com/javase/tutorial/uiswing/index.html.

[2] Tkinter. https://docs.python.org/2/library/tkinter.html.