Rapport Projet

Erwan Guichard

Novembre 2018

Table des matières

Résumé

Ce rapport est en LaTeX, il contient les explications détaillées de la réalisation de mon programme pour l'exercice 3.2.

1 Introduction

Pour le projet de **Programmation Concurrente** j'ai choisi de faire l'exercice de balles en mouvement en Java car c'est un exercice très intéressant pour s'entrainer à programmer avec des objets. Je ne mettrai pas tout le code dans ce rapport, je montrerai seulement les parties importantes du programme qui nécessite une explication. Pour la réalisation de ce projet je me suis principalement inspiré d'OpenClassRoom [3].

2 La classe Fenetre :

Tout d'abord commençons par la classe qui crée la fenêtre de l'application.

2.1 Les variables

La classe **Fenetre** héritée de *JFrame* contient les variables principales les variables d'instances de type *JPanel* et de type *JButton*. Nous verrons leurs utilités plus tard.

```
static float afficheScore  = 0;
static double tmpDebut = 0;
static double tmpFin = 0;
static float tmpExec;
static float tmpTot;
private JPanel entete = new JPanel();
static JLabel score = new JLabel("Score: "+afficheScore);
static JLabel temps = new JLabel("Temps: 0 s");
private JButton startStop = new JButton("Start");
private JButton plus = new JButton("  +  ");
private JButton moins = new JButton("  -  ");
private JPanel boutons = new JPanel();
static Panneau panneau = new Panneau();
static boolean movement = false;
static boolean CanIncTmpTot = true;
```

2.2 Le conctructeur

Après cela, on ajoute dans le constructeur les boutons, les labels et le panel grâce aux layout managers. Ici, j'utilise l'objet *BorderLayout*.

```
public Fenetre() {

    super("Balles");
    add(entete, BorderLayout.NORTH);
    entete.add(score);
    entete.add(temps);
    panneau.setBackground(Color.lightGray);
    add(panneau, BorderLayout.CENTER);
    add(boutons, BorderLayout.SOUTH);
    boutons.add(startStop);
    boutons.add(plus);
```

```
boutons.add(moins);
    ...
```

La partie ci-dessus, permet d'agencer le positionnement des composants. Je commence par appeler ma fenêtre **Balles** ensuite je positionne mes composants de la façon suivante :

- un *JPanel* **entete**, positionnée dans le haut du panneau avec *BorderLayout.NORTH*, qui contient :
  - le label **score** qui affiche le score incrémenté à chaque collision (initialement à 0)
  - le label **temps** qui affiche le temps écoulé pendant que les balles sont en mouvement (initialement à 0)
- un *JPanel* **panneau**, positionnée dans le centre du panneau avec *BorderLayout.CENTER* avec une couleur d'arrière-plan gris clair, destinée à afficher des balles
- un *JPanel* **boutons**, positionnée dans le bas du panneau avec *BorderLayout.SOUTH*, qui contient les boutons :
  - **startStop** pour démarrer ou arreter le déplacement des balles
  - **plus** pour ajouter une balle
  - **moins** pour retirer une balle

ensuite, toujours dans le constructeur, j'ajoute les actions pour chaque bouton :

- le **startStop** est en deux partis :
  - si *movement* est vrai, c'est-à-dire si la balle est en mouvement alors il faut stopper le mouvement et pour cela : on passe le booléen *CanInctmpTot* à vrai pour autoriser l'incrémention du temps, on enresgistre la valeur du temps de fin de mouvement dans *tmpFin* qui récupère l'heure courante (qu'on vera à la section 4) , on change le texte dans le bouton Start pour qu'il devienne Stop et finalement on passe le booléen de *movement* à faux.
  - sinon on enresgistre la valeur du temps de début de mouvement dans *tmpDebut*, on change le texte dans le bouton Stop pour qu'il devienne Start et finalement on passe le booléen de *movement* à vrai.
- le **plus** appelle la méthode *ajoutBalle()* qu'on vera dans la section 2.3
- le **moins** appelle la méthode *panneau.retrait()* qu'on vera dans la section 5.2

```
    ...
startStop.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
     if(movement) {
     CanIncTmpTot=true;
     tmpFin = System.currentTimeMillis();

     startStop.setText("Start");
     movement=false;
     }else {
     tmpDebut = System.currentTimeMillis();
     startStop.setText("Stop");
     movement=true;
     }
    }
 });

plus.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
    ajoutBalle();
    repaint();
   }
```

```
        });

        moins.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent e) {
               panneau.retrait();
               repaint();
           }
        });
            ...
```

Et finalement ci-dessous, toujours dans le constructeur, les parametre de base pour le lancement d'une fenêtre telle que sa taille, sa position sur l'écran, la fermeture de la fenêtre quand on clique sur la croix rouge et sa visibilité.

```
            ...
        setSize(500, 450);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setVisible(true);
    }
```

2.3 La méthode ajoutBalle()

la méthode *ajoutBalle()* permet :

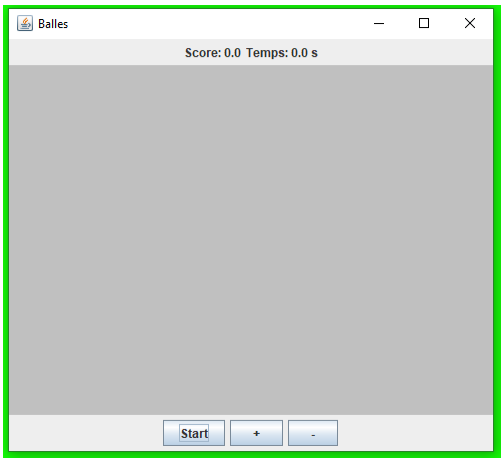
— de créer une balle (créer une instance de classe Balle)

— d'ajouter cette balle au panneau (qui mémorise toutes les balles ajoutées dans une liste (*ArrayList<Balle>*)

— de lancer un thread dont le code est dans la classe *BalleSeparee*(section 4) (donc un nouveau thread par balle)

```
 private void ajoutBalle() {
     Balle balle = new Balle();
     panneau.ajout(balle);
     new Thread(new BalleSeparee(balle)).start();
  }
```

Visuellement, cela donne :



Fenêtre vide

3 La classe Balle :

La classe **Balle** permettra, quand on l'appelle, de créer une balle avec des coordonnées x et y aléatoires dans le panneau ainsi que une couleur aléatoire(code de couleurs aléatoire inspire du JournalDuNet [2].

3.1 Les variables

```
public class Balle {
    private int rad=30;
    private int randx = (int) (Math.random()*((Fenetre.panneau.getWidth()-rad)-1))+1;
    private int randy = (int) (Math.random()*((Fenetre.panneau.getHeight()-rad)-1))+1;
    private int x = randx, y = randy;
    boolean backX, backY = false;
    Color balleCol = new Color((int)(Math.random() * 0x1000000));
             ...
```

Elle contient une méthode *move()* qui permet de la déplacer en ajoutant 1 à ses coordonnées x et y actuels. Elle teste aussi si la balle va sortir du champ du panneau (peu importe sa taille) et par conséquent changer sa coordonnée x ou y en inversent le sens de déplacement en décrémentant de 1. Il en résulte que la balle rebondira sur les bords du panneau. La balle est représentée graphiquement par un disque de 30 pixels de rayon.

3.2 La méthode move()

```
             ...
    public void move() {
        if(x < 1)
            backX = false;
        if(x > Fenetre.panneau.getWidth()-rad)
            backX = true;
        if(y < 1)
            backY = false;
        if(y > Fenetre.panneau.getHeight()-rad)
            backY = true;
        if(!backX)
            ++x;
        else
            --x;
        if(!backY)
            ++y;
        else
            --y;
    }
             ...
```

3.3 Les accesseurs et mutateurs

la classe contient aussi ses accesseurs et mutateurs qui permette de récupérer ou modifier les valeurs de x, y et rad

```
             ...
    public int getPosX() {return x;}
    public int getPosY() {return y;}
    public int getRad() {return rad;}
```

```
    public void setPosX(int x) {this.x = x;}
    public void setPosY(int y) {this.y = y;}
    public void setRad(int y) {this.rad = rad;}
             ...
```

3.4 La méthode paintComponent()

En finalement, il y a la méthode *painComponent* qui dessine la balle avec la couleur aléatoire généré par la variable *balleCol*.

```
             ...
    public void paintComponent(Graphics2D g){
        g.setColor(balleCol);
        g.fillOval(x,y,rad,rad);
        g.drawOval(x,y,rad,rad);
    }
}
```

Visuellement, cela donne :

Fenêtre avec les balles créées chacune d'une couleur aléatoire

4 La classe BalleSeparee :

La classe **BalleSeparee** implémentation de l'interface *Runnable* est la classe qui va exécuter les mouvements de chaque balle. Elle contient l'unique méthode *run()* qui est exécuté par le thread. Cette méthode *run()* contient une boucle infinie qui fait sans arrêt (jusqu'à ce qu'on arrête le programme) :

- Si le booléen *movement* est à vrai :
  - appelle la méthode *balle.move()*, donc déplace la balle (donc de 1 pixel en x et y) et fait le test du rebond
  - demande de redessiner le panneau (pour qu'on voit l'effet sur l'affichage du déplacement qui vient d'être effectué) par l'appel de repaint() sur le panneau.
  - attend 5 ms et recommence
- Sinon
  - il lance le calcule du temps total d'exécution du programme si le booléan *CanIncTmpTot* est à vrai.
  - actualise ensuite le label *score* avec la nouvelle valeur calculé

4.1 La méthode run()

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

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

    public void run() {
        try {
            while(true){
                Fenetre.panneau.collision();
                if(Fenetre.movement) {
                    balle.move();
                    Fenetre.panneau.repaint();
                    Thread.sleep(5);
                } else {
                    Fenetre.tmpExec =  (float) ((Fenetre.tmpFin - Fenetre.tmpDebut) /1000F);
                    if (Fenetre.CanIncTmpTot) {
                        Fenetre.tmpTot+=Fenetre.tmpExec;
                        Fenetre.CanIncTmpTot=false;
                    }
                    DecimalFormat df = new DecimalFormat ();
                    df.setMaximumFractionDigits (2);
                    Fenetre.temps.setText("Temps: "+df.format(Fenetre.tmpTot)+" s");
                }
            }
        } catch (InterruptedException e) { }
    }
}
```

5 La classe Panneau :

La classe **Panneau** héritée de *JPanel* contient les méthodes qui font interagir les balles avec lui-même. C'est grâce a cette classe que l'on peut dessiner les balles.

5.1 La méthode ajout()

La méthode *ajout()*, qui pour rappel est exécuter à l'appui sur le bouton **plus**, permet d'ajout une balle dans la liste *balles*. Il ajoute tant que la liste ne contient pas 10 balles.

5.2 La méthode retrait()

La méthode *retrait()*, qui pour rappel est exécuter à l'appui sur le bouton **moins**, permet d'enlever une balle dans la liste *balles*. Il enlève tant que la liste n'est pas vide.

5.3 La méthode collision()

La méthode *collision()* consiste à tester si un balle i entre en collision avec une balle j et si c'est le cas :

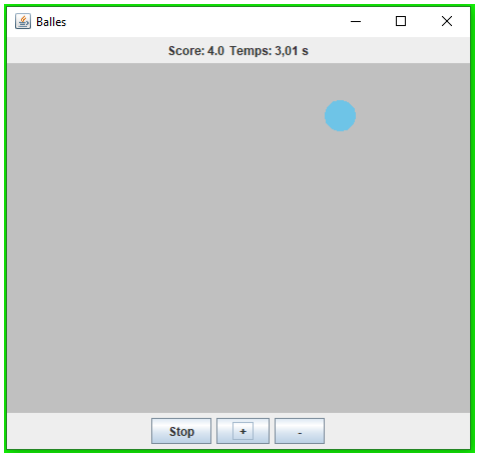
— incrémente *affichéScore* et met à jour la valeur dans le label

— supprime les balles i et j de la liste *balles*

(inspirer de post sur Comment ça Marche [1]).

5.4 La méthode paintComponent()

La méthode *paintComponent()* est la méthode qui fait l'affichage de Panneau, redéfinie pour le besoin, et qui est automatiquement appelée en conséquence de l'appel de repaint() dans Balle-Séparée.



Fenêtre avec le temps et le score incrémenté

6 Difficulté :

J'ai rencontré pas mal de difficulté pour créer ce programme une des difficultés que j'ai pu résoudre est le fait que mes couleurs de balles se mettaient à jour à chaque mouvement donc changeaient la couleur des balles. Cependant je n'ai pas pu corriger deux autres problèmes :

- le temps ne se met à jour que lorsque j'appuie sur le bouton **Stop** et pas durant tout le mouvement
- les collisions retour des erreurs car le faite de supprimer les balles faisait planter le mouvement et je pense aussi que c'est à cause des threads pour chaque balle.

7 Conclusion :

Ce projet était assez simple à première mais les difficultés que j'ai surmontées on fait que j'ai passé beaucoup de temps à chercher des solutions mais sans succès.

8 Bibliographie :

Références

[1] Commentcamarche. https://www.commentcamarche.net/.

[2] Journaldunet. https://www.journaldunet.fr/web-tech/developpement/1202399-comment-generer-un-nombre-aleatoire-random-en-java-compris-entre-deux-chiffres.

[3] Openclassroom. https://openclassrooms.com/fr/courses/26832-apprenez-a-programmer-en-java/23193-le-fil-rouge-une-animation.