Programmation concurrente

ḾOHSIN Aly Asgar, L3 informatique

10 octobre 2017

Résumé

Afin que les programme ne s'exécute plus de manière séquentielle, il existe un moyen qui permet d'exécuter plusieurs programme en parallèle, ce sont les threads. Lors d'un lancement d'un programme, l'un des threads s'occupe de l'interface graphique. A travers ces deux exercices (2.3 et 3.2) , nous allons étudier comment les créer et les manipuler

1 Introduction

Un processus est un programme (fichier exécutable) en cours d'exécution sur un processeur. Dans chaque processus, Il y a des threads. Un thread, c'est un fil d'exécution de code à l'intérieur d'un processus et qui a la possibilité d'être ordonnancé (c'est un « sous-processus »). Ce sont des fils d'exécution de notre programme. Lorsqu'on en crée plusieurs, on peut exécuter des tâches simultanément. Parmi tous ses tâches, une, c'est de s'occuper de l'interface graphique. L'interface graphique, appelée communément la GUI (pour Graphical User Interface), est le lien entre l'utilisateur et la machine. Il permet d'afficher à l'écran des éléments que l'utilisateur comprend et interprète.

Nous allons voir en premier lieu les interface graphique en java et en python puis en deuxième partie les threads dans ces mêmes langages.

1.1 Java

Pour que les programme ne réagissent plus à des saisies au clavier mais à des événements provenant d'un composant graphique (un bouton, une liste, un menu), nous allons voir la notion d'interface graphique

Le langage Java propose différentes bibliothèques pour programmer des IHM (Interface-Hommes Machine), mais nous utiliserons essentiellement les packages `javax.swing, java.awt` présents dans Java.

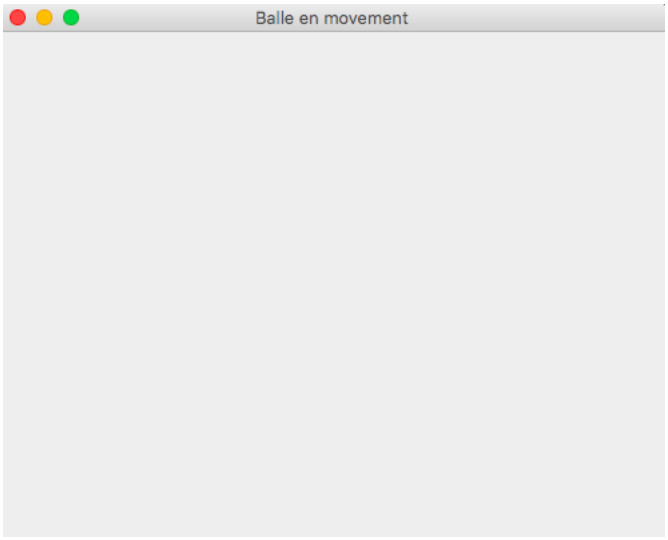
La classe qui représente les fenêtres est la classe `JFrame`. Pour créer une fenêtre avec un certain titre, on utilise un constructeur qui prend un objet `String` comme unique paramètre. Une fois la fenêtre créée, il ne reste plus qu'à lui donner une dimension avec la méthode `setSize` et ensuite de la rendre visible avec la méthode `setVisible` qui prend un booléen en paramètre et la fenêtre s'affiche lors de l'exécution du programme. Lorsque l'on travaille avec les interfaces graphiques, on peut également demander que le programme se termine lorsque l'utilisateur ferme la fenêtre en cliquant sur la croix dans la barre de titre en utilisant la méthode `setDefaultCloseOperation` de la classe `JFrame`. Nous avons désormais notre fenêtre

Voici le code de l'affichage de l'exercice 3.2

```
import javax.swing.*;

public class Fenetre extends JFrame {
public Fenetre() {
```

```
        super("Balle en movement");
        setSize(500, 400);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setVisible(true);
    }
}
```



1.2 Python

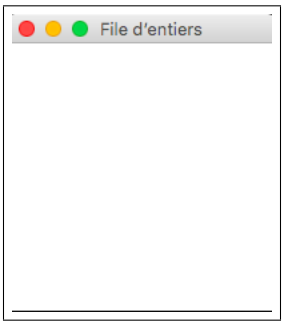
Contrairement à Java, pour créer une interface graphique en python c'est un peu plus simple. En python, on utilise la bibliothèque `Tkinter` pour créer des interface graphique.

- Tout d'abord, on commence par importer `Tkinter`.
- On crée ensuite un objet de la classe `Tk`. Cet objet sera la fenêtre principale de notre interface.
- On crée un `Label`, c'est-à-dire un objet graphique affichant du texte
- On appelle la méthode `pack` de notre `Label`. Cette méthode permet de positionner l'objet dans notre fenêtre (et, par
- conséquent, de l'afficher).
- Enfin, on appelle la méthode `mainloop` de notre fenêtre racine. Cette méthode ne retourne que lorsqu'on ferme la fenêtre.

Code de l'affichage de l'exercice 2.3

```
from tkinter import *

fen = Tk()
fen.title('File d'entiers')
fen.mainloop()
```



Ainsi, nous venons de voir comment créer des interface graphique en Java et Python

2 Threads

2.1 Java

Pour écrire des applications concurrentes en Java, on va utiliser la classe `Thread` qui se trouve dans le package `java.lang`. Celle-ci représente deux choses à la fois : la tâche qui doit être exécutée en parallèle et le thread. Pour définir une nouvelle tâche, il suffit d'écrire une nouvelle classe qui étend la classe `Thread` et de redéfinir la méthode `run`. Pour créer un nouveau thread et donc avoir la tâche exécutée en parallèle, il faut appeler la méthode `start` de la classe `Thread`. Cette méthode fait deux choses : elle crée tout d'abord un nouveau thread d'exécution et elle y exécute ensuite la méthode `run`.

2.1.1 Problème

Dans cet exercice, deux chose ma poser problème et que je n'ai pas su faire :

1. L'horloge
2. La collision

2.2 Python

Tout comme on java, crée des threads est python est plus ou moins pareille.Pour créer un thread, il suffit de surcharger la méthode `run` de la classe Thread. Si le thread a besoin de données lors de son exécution, il faut surcharger son constructeur sans oublier d'appeler le constructeur de la classe mère. L'exécution de thread commence par la création d'une instance et l'appel à la méthode `start`. En résumé, il faut retenir les éléments suivants :

- surcharger la classe `threading.Thread`,
- surcharger le constructeur sans oublier d'appeler le constructeur `threading.Thread.__init__`
- surcharger la méthode `run`, c'est le code que devra exécuter le thread,
- créer une instance de la nouvelle classe et appeler la méthode `start` pour lancer le thread secondaire qui formera le second fil d'exécution.

La portion de code de l'exercice 2.3 montre comment créer des threads en python

```
class ThreadConsommateur(Thread):
    def __init__(self,texte, q, tempsSommeil, nom):
        Thread.__init__(self)
        self.texte = Label(fen, text="Thread Consommateur: Retrait de effectue!")
        self.texte.pack()
        self.message= texte
        self.q = q
```

```
        self.tempsSommeil = tempsSommeil
        self.name = nom
        self.daemon = True

    def run(self):
        while True:
            self.retrait = self.q.get(block=True, timeout=None)
            self.texte["text"] = "Thread Consommateur: Retrait de {} effectue!".format(self.retrait)
            self.texte.pack(expand=True, fill='both')
            time.sleep(self.tempsSommeil)

c = ThreadConsommateur(texte,q, 1, "c")
    c.start()
```

3 Conclusion

En bref, a travers ces deux exercices, on constate que pour avoir une meilleur expérience utilisateur et un bon programme, il faut avoir une interface graphique intuitive et diviser nos programme en plusieurs threads.