Université de La Reunion

Brice VIRASSAMY 36004977

File d'entier

Session 2018/2019

-

Résumé

En résumé, cet exercice du TP consiste à modéliser un producteur et des consommateurs (d'un domaine quelconque) avec un système de file d'attente. Le producteur fournit à la file d'attente jusqu'à ce qu'elle soit pleine et les consommateurs consomment jusqu'à l'indisponibilité de la ressource, autrement dit lorsqu'il n'y a plus de produit dans la file d'attente. j'ai choisi de rajouter la possibilité de mettre en pause les consommateurs et le producteur pour que même s'il y ait disponibilité de ressources ou même si la file n'est pas pleine, le producteur et les consommateurs arrêtent leurs activités. Une interface graphique a aussi été rajouter pour l'utilisateur du programme (interface en java et python).

Table des matières

1 Introduction **1**
1.1 Objectifs 1
1.2 Plan 1

2 Les préliminaires **2**
2.1 En Java 2
2.2 En Python 2

3 Points importants **3**
3.1 Le test primordial 3
3.2 La mise en pause 3

4 Architecture de l'interface graphique en Java **5**

5 Architecture de l'interface graphique en Python **7**

6 Conclusion **8**

7 Bibliographie **9**

Chapitre 1

Introduction

1.1 Objectifs

Les objectifs à réaliser sont les suivants :
— Faire en sorte que le producteur/consommateur ne puisse pas produire/consommer quand la file est pleine/vide.
— Rajouter la fonctionnalité de pause qui a été énoncer.
— Proposer une interface graphique cohérente avec le sujet.

1.2 Plan

— Dans un 1er temps : Nous parlerons des recherches et apprentissages préliminaires à la rédaction du code
— Dans un 2nd temps : Nous traiterons les points importants communs aux deux langages.
— Dans un 3eme temps : Nous examinerons l'interface graphique en java.
— Dans un 4eme temps : Nous examinerons l'interface graphique en python.

Chapitre 2

Les préliminaires

2.1 En Java

En amont du programme java, je me suis donc renseigné sur comment modéliser l'idée que j'avais concernant l'interface graphique. J'avais des connaissances, solides sur certains points où je savais exactement quel code effectuer mais sur d'autres points notamment le traitement des "JLabels" après leurs créations, j'avais des lacunes donc j'ai consulté différents sites qui traités points par points des éléments possibles d'une interface graphique, mais je ne savais pas comment assembler tout cela donc j'ai décidé de reprendre depuis le début l'apprentissage de l'interface graphique java à travers le cours de java sur le site openclassroom.

2.2 En Python

En amont du programme python, comme pour le programme en python je me suis renseigneé sur comment modéliser l'idée que j'avais. Le problème était que je n'ai jamais programmé d'interface graphique en python, c'était donc une découverte pour moi. Le prof nous a indiqué quel était le nom de l'interface graphique (TKinter) et un cours sur le site openclassroom pour nous permettre de manière autonome de s'approprier cette librairie. Je suis donc parti de zéro par rapport à java concernant l'interface graphique et j'ai suivi rigoureusement le cours, mais en conséquent j'ai pris plus de temps sur la programmation en python par rapport à ce point.

Chapitre 3

Points importants

3.1 Le test primordial

En lien avec les objectifs, les producteurs et les consommateurs ne doivent pas manipuler la file si elle est vide pour le consommateur ou si elle est pleine pour le producteur. Ceci a été réaliser grâce aux codes suivants : En java, dans la file, dans les méthodes enfiler et defiler un element) :

```
while(tabFile.size()==MAX) wait();
tabFile.add(e);
notifyAll();
```

```
while(tabFile.size()==0) wait();
int e = tabFile.get(0);
tabFile.remove(0);
notifyAll();
```

En Python, dans la file, dans les méthodes enfiler et defiler un element) :

```
with self.cond:
    while(len(self.tabFile)==20):
        self.cond.wait()
    self.tabFile.append(e)
    self.cond.notifyAll()
```

```
with self.cond:
    while(len(self.tabFile)==0):
          self.cond.wait()
    e = self.tabFile[0]
    del self.tabFile[0]
    self.cond.notifyAll()
```

Ces codes testent si la file est vide ou si elle est pleine. Si elle est pleine le producteur ne peut plus rajouter. Si elle est vide le consommateur ne peut plus consommer.

3.2 La mise en pause

Toujours en accord avec les objectifs, la mise en pause doit pouvoir s'effectuer correctement, la file ne doit pas se remettre dans son état initial à la mise en pause des threads, c'est à dire avec zéro élément (sauf si tout a été consommer au préalable). je vais expliquer concrètement comment cela fonctionne. Au lancement du programme je lance une méthode, startThread() qui lance les threads qui ont été crée : En java :

```
public void startThread() {
createThread();
prod.start();
for(int i = 0; i<consos.length;i++) { consos[i].start(); }
}
```

En python :

```
def startThread(self):
    self.createThread()
    self.prod.start()
    for threadConso in self.consos:
        threadConso.start()
```

Ensuite si je clique sur stop, la méthode stopThread() est lancée et stop les threads, mais on ne veut pas stopper, on veut mettre en pause. Ce qu'il faut savoir, c'est que les éléments des Threads initiaux sont stockés en attribut de la classe fenêtre donc jamais effacé, quand je reclique sur startThread() cela recrée des nouveaux Threads mais avec les éléments des Threads initiaux donc cela permet de reprendre avec les "mêmes threads". Le bouton start/stop simule donc bien une pause. Voilà la méthode stopThread() : En java :

```
public void stopThread(){
prod.interrupt();
for(int i = 0; i<consos.length;i++) { consos[i].interrupt(); }
labProd.setText("");
labConso.setText("");
repaint();
}
```

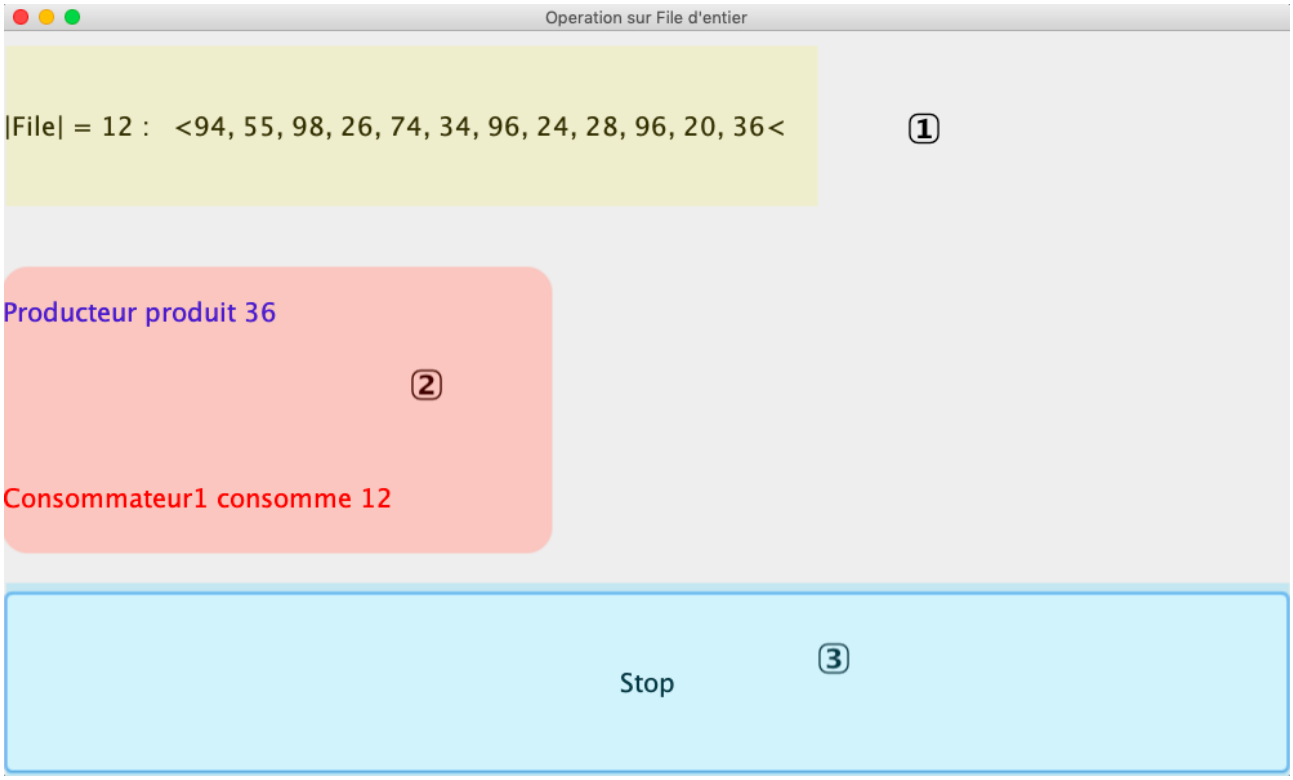
En python :

```
def stopThread(self):
    self.prod.interrupt()
    for threadConso in self.consos:
        threadConso.interrupt()
    self.labConso.config(text="")
    self.labProd.config(text="")

    self.labFile.update()
    self.labConso.update()
    self.labProd.update()
```

Chapitre 4

Architecture de l'interface graphique en Java



L'interface graphique est une fenêtre JFrame. En 1, nous avons la file. En 2, les threads producteurs et consommateurs. En 3, le bouton start/stop pour mettre en pause ou relancer les threads. En 1 et 2 tout a été constitué avec des JLabels. En 3, c'est un bouton et il a été constitué avec un JButton.

Code de création des éléments :

```
this.labFile = new JLabel();
labFile.setText("< <");
labFile.setForeground(Color.BLACK);
labFile.setFont(new Font("Courrier New", Font.PLAIN, 20));

this.labProd = new JLabel();
labProd.setForeground(Color.BLUE);
labProd.setFont(new Font("Courrier New", Font.PLAIN, 20));

this.labConso = new JLabel();
labConso.setForeground(Color.RED);
```

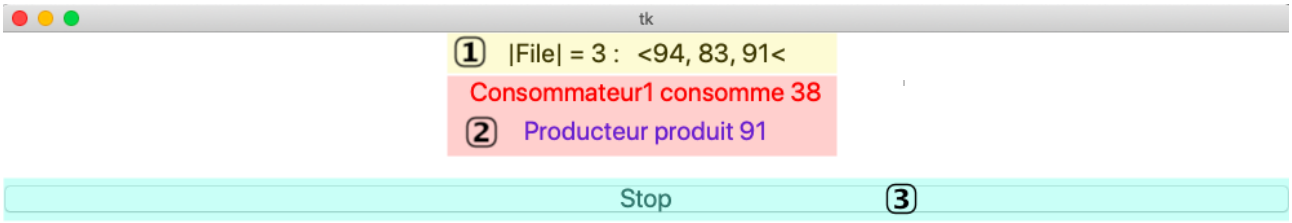
```
labConso.setFont(new Font("Courrier New", Font.PLAIN, 20));

JButton stopAndStart = new JButton("Start");
stopAndStart.setFont(new Font("Courrier New", Font.PLAIN, 20));
```

La methode setText() permet d'éditer le texte présent dans le Jlabel, setForeground() permet d'éditer la couleur et setFont() la couleur

Chapitre 5

Architecture de l'interface graphique en Python



L'interface graphique est une fenêtre Frame. Comme pour Java, en 1 nous avons la file, en 2 les threads et en 3 le bouton start/stop. En 1 et 2 tout a été constitué en Label (python) et 3 avec un Button (python)

Code de création des éléments :

```
self.labFile = Label(self,text="<  <",font=("Courrier New",20))
self.labProd = Label(self,text="", fg="blue",font=("Courrier New",20))
self.labConso = Label(self,text="", fg="red",font=("Courrier New",20))
self.stopAndStart = Button(self, text="Start", command =self.cliquer,font=("Courrier New",20))
self.stopAndStart.pack(side="bottom", fill=BOTH)
self.labProd.pack(side="bottom",fill=BOTH)
self.labConso.pack(side="bottom",fill=BOTH)
self.labFile.pack(side="bottom",fill=BOTH)
```

Les attributs text, fg (foreground en java), font sont équivalents aux méthodes avec le même nom en java. L'attribut command attend une fonction comme valeur, cette fonction sera exécutée au clic sur le bouton. La méthode pack() avec les attributs side et fill sert au positionnement dans la fenêtre Frame

Chapitre 6

Conclusion

Pour conclure, les objectifs énoncés ont tous été réaliser, la production et la consommation respectent ce qui a été imposer. La mise en pause est fonctionnelle. L'interface graphique est présente et cohérente avec le sujet. Pour améliorer le programme on pourrait améliorer l'interface graphique, principalement en python. On pourrait aussi rajouter une fonctionnalité pour saisir les threads directement dans l'interface graphique.

Chapitre 7

Bibliographie

— Cours Interface Graphique Java : https://openclassrooms.com/courses/apprenez-a-programmer-en-java/le-fil-rouge-une-animation
— Cours Interface Graphique Python : https://openclassrooms.com/fr/courses/235344-apprenez-a-programmer-e234859-des-interfaces-graphiques-avec-tkinter