TRAVAUX PRATIQUES 8
– JAVA POO

Université Sultan Moulay Slimane **Beni-Mellal**	
École nationale des sciences appliquées **Khouribga**	

COMPTE RENDU :

TP8 JAVA POO

Filière : Informatique et Ingénierie des données (iid1)

Réalisé par :	• LAHMAMA Fatima-Zahraa • EL FATHI Zakaria
Encadré par :	• Mr. GHERABI Noreddine

AU : 2021/2022

TRAVAUX PRATIQUES 8
– JAVA POO

EXERCICE 1 :

Un programme dont le thread principal lance et nomme trois threads. Chaque thread ainsi créé doit effectuer :

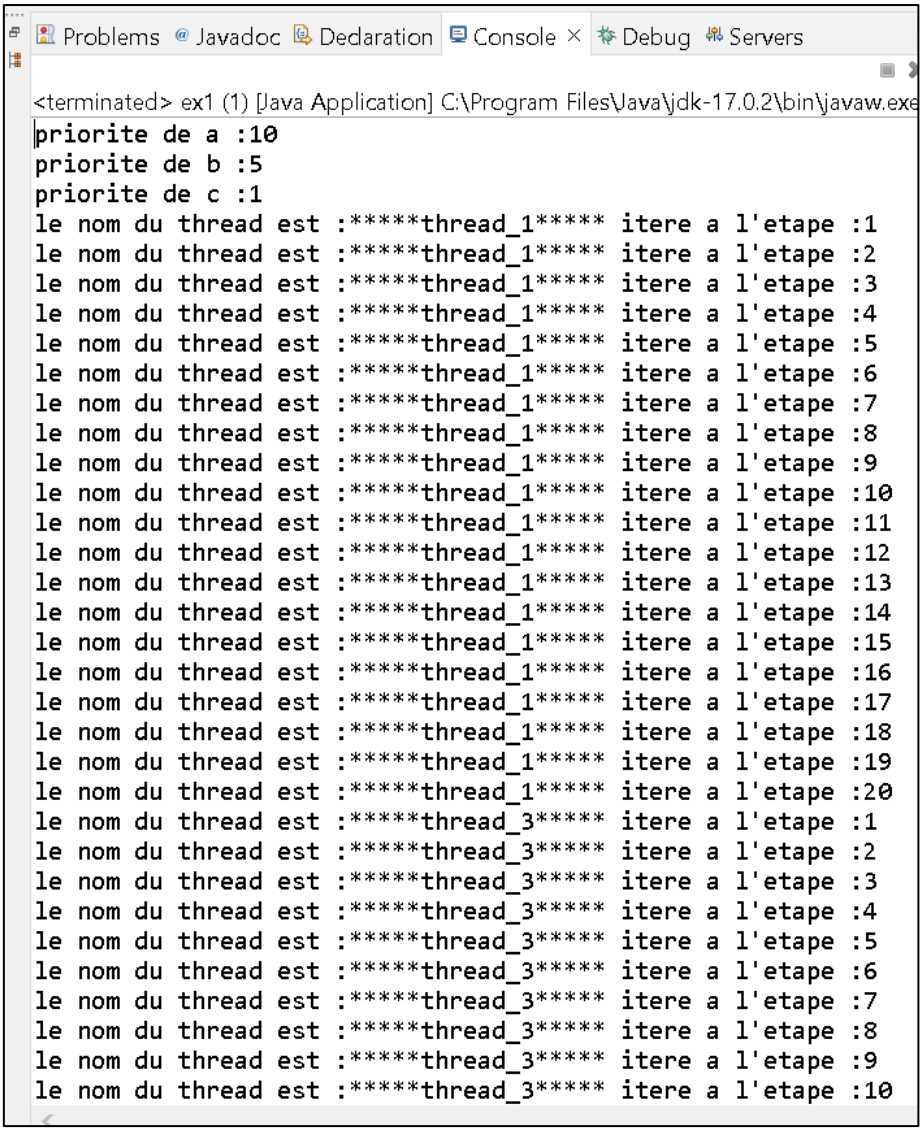
- 20 fois les actions suivantes :
- Attendre un temps aléatoire compris entre 0 et 200 ms,
- Afficher son nom
  - Le premier Thread le plus prioritaire et le troisième ayant une priorité minimale.
  - Le 2ème Thread devra attendre la fin de l'exécution des deux autres threads avant de terminer son exécution.

Code :

```
package tp8;

class ppalThread extends Thread{
    String nom;
    public ppalThread(String nom) {
        super(nom);
        }
    public void run() {

        for(int i=0;i<20;i++) {
            try {sleep((int)Math.random()*200);
        }catch(InterruptedException e) {
            e.printStackTrace();
        }
            System.out.println("le nom du thread est
:*****"+this.currentThread().getName()+"***** itere a l'etape :"+(i+1));
        }
    }
}
public class ex1 {

    public static void main(String[] args) {

        ppalThread t1=new ppalThread("thread_1");
        ppalThread t2=new ppalThread("thread_2");
        ppalThread t3=new ppalThread("thread_3");
        //priorite maximale
        t1.setPriority(10);
        //priorite minimale
        t3.setPriority(1);
        System.out.println("priorite de a :"+t1.getPriority());
        System.out.println("priorite de b :"+t2.getPriority());
        System.out.println("priorite de c :"+t3.getPriority());

        t1.start();
```

```
        // t2.start();
        t3.start();
        try {
            t1.join();
            t3.join();
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        t2.start();
    }
}
```

Exécution :



EXERCICE 2 :

L'objectif de cet exercice est de réaliser des opérations de deux threads qui partagent un même objet (ici un compte en banque commun).

1. On crée la classe (Compte) qui comprend :
   - Un attribut privé solde initialisé à 100 représentant le solde courant du compte
   - Les getters et les setters
   - Une méthode retirer(float montant) permettant de retirer un certain montant du compte.

Code :

```
class Compte{
    private float solde;
    public Compte() {
        super();
        this.solde=100;
    }

    public Compte(float solde) {
        super();
        this.solde=solde;
    }

    public float getSolde() {
        return solde;
    }

    public void setSolde(float solde) {
        this.solde = solde;
    }

    public void retirer(float montant) {
            this.solde-=montant;
    }
}
```

Ensuite on crée la classe (Personne) qui implémente « Runnable » et représente le comportement que deux personnes (Ahmed et Amal) ont tous les deux. Leur comportement est assez particulier puisque Said et Ali réfléchissent (s'endorment 2 secondes), et en particulier pendant qu'ils effectuent un retrait. Cette classe contient :

- Un attribut compte de type Compte représentant le compte en banque de Ahmed et Amal.
- Une méthode Retrait(float montant) permettant à Ahmed et Amal d'effectuer un retrait sur leur compte en banque. Le comportement de cette méthode est le suivant : La personne voulant effectuer le retrait vérifie le solde, puis s'endort 2 secondes, puis à son réveil effectue le retrait. Le nom de la personne effectuant le retrait doit être signalé.
- Une méthode run() décrivant le comportement de Ahmed et Amal. Il s'agit d'effectuer en boucle (par exemple 10retraits de 200DH).

Code :

```
class Person implements Runnable{
    Compte comp;
    String proprietaire;
    public Person(Compte compte,String proprietaire ) {
        super();
        this.comp=compte;
        this.proprietaire=proprietaire;
    }

    public void Retrait(float montant) {
        if(montant <= comp.getSolde()) {
        System.out.println("------Solde suffisant!(wait 2 seconds please!)------");
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        comp.retirer(montant);
        System.out.println("------retrait effectue avec succes par: "+ proprietaire +"--");

        }

        else System.out.println(" ------Solde insuffisant pour cette operation !-------");
    }

    @Override
    public void run() {
        System.out.println("**********operation commencee!**********");
    for(int i=0;i<10;i++) {
                this.Retrait(200);
                System.out.println("total de retrait a cet instant:"+(200*(i+1))+"Dh!");
        }
    System.out.println("\n\n-----votre solde maintenant est de :"+comp.getSolde()+"\n\n");
```

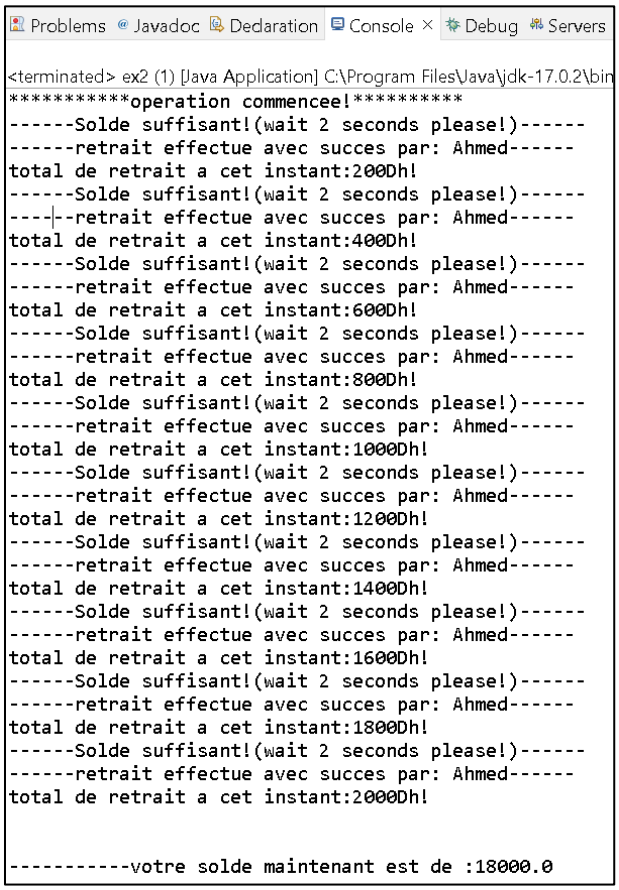
```
    }

}
```

En fin la méthode main(String[] args) crée deux threads (Ahmed et Amal) avec le même Runnable (ici de type Personne), puis les lance.

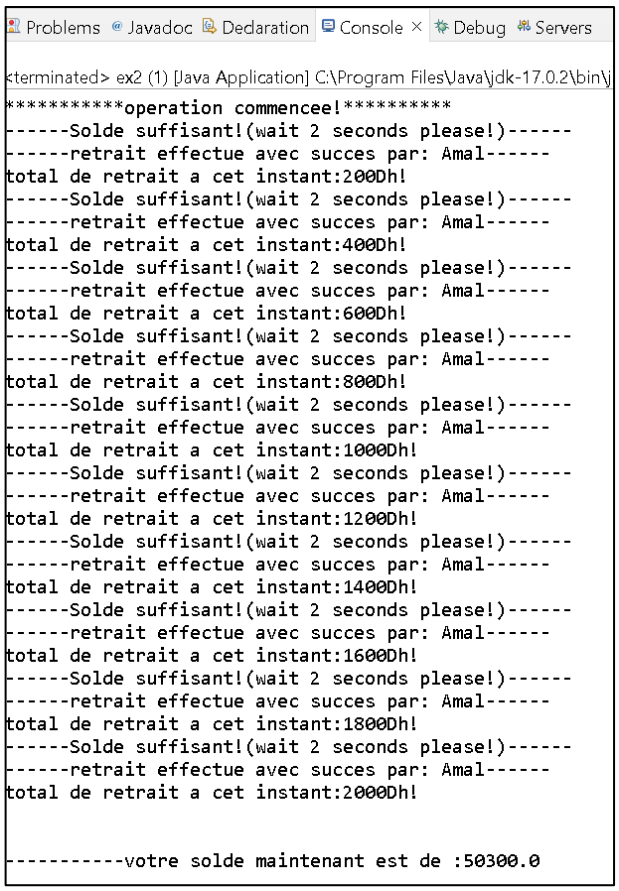
Code :

```
public class ex2 {

    public static void main(String[] args) {


        //on cree deux compte pour les attribuer a Amal et Ahmed
        Compte compte1 =new Compte(20000);
        Compte compte2 =new Compte(52300);

        // on instancie deux personnes Ahmed et AMAL avec les deux comptes
        Person Ahmed =new Person(compte1,"Ahmed");
        Person Amal =new Person(compte2,"Amal");

        //lancement
        Thread Ahmed1=new Thread(Ahmed);
        Thread Amal1= new Thread(Amal);
        Ahmed1.start();
        try {
            Ahmed1.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        Amal1.start();}
```

Exécution :



```
Problems  Javadoc  Declaration  Console ×  Debug  Servers
<terminated> ex2 (1) [Java Application] C:\Program Files\Java\jdk-17.0.2\bin
**********operation commencee!**********
------Solde suffisant!(wait 2 seconds please!)------
------retrait effectue avec succes par: Ahmed------
total de retrait a cet instant:200Dh!
------Solde suffisant!(wait 2 seconds please!)------
------retrait effectue avec succes par: Ahmed------
total de retrait a cet instant:400Dh!
------Solde suffisant!(wait 2 seconds please!)------
------retrait effectue avec succes par: Ahmed------
total de retrait a cet instant:600Dh!
------Solde suffisant!(wait 2 seconds please!)------
------retrait effectue avec succes par: Ahmed------
total de retrait a cet instant:800Dh!
------Solde suffisant!(wait 2 seconds please!)------
------retrait effectue avec succes par: Ahmed------
total de retrait a cet instant:1000Dh!
------Solde suffisant!(wait 2 seconds please!)------
------retrait effectue avec succes par: Ahmed------
total de retrait a cet instant:1200Dh!
------Solde suffisant!(wait 2 seconds please!)------
------retrait effectue avec succes par: Ahmed------
total de retrait a cet instant:1400Dh!
------Solde suffisant!(wait 2 seconds please!)------
------retrait effectue avec succes par: Ahmed------
total de retrait a cet instant:1600Dh!
------Solde suffisant!(wait 2 seconds please!)------
------retrait effectue avec succes par: Ahmed------
total de retrait a cet instant:1800Dh!
------Solde suffisant!(wait 2 seconds please!)------
------retrait effectue avec succes par: Ahmed------
total de retrait a cet instant:2000Dh!


----------votre solde maintenant est de :18000.0
```



```
Problems  Javadoc  Declaration  Console ×  Debug  Servers
<terminated> ex2 (1) [Java Application] C:\Program Files\Java\jdk-17.0.2\bin\j
**********operation commencee!**********
------Solde suffisant!(wait 2 seconds please!)------
------retrait effectue avec succes par: Amal------
total de retrait a cet instant:200Dh!
------Solde suffisant!(wait 2 seconds please!)------
------retrait effectue avec succes par: Amal------
total de retrait a cet instant:400Dh!
------Solde suffisant!(wait 2 seconds please!)------
------retrait effectue avec succes par: Amal------
total de retrait a cet instant:600Dh!
------Solde suffisant!(wait 2 seconds please!)------
------retrait effectue avec succes par: Amal------
total de retrait a cet instant:800Dh!
------Solde suffisant!(wait 2 seconds please!)------
------retrait effectue avec succes par: Amal------
total de retrait a cet instant:1000Dh!
------Solde suffisant!(wait 2 seconds please!)------
------retrait effectue avec succes par: Amal------
total de retrait a cet instant:1200Dh!
------Solde suffisant!(wait 2 seconds please!)------
------retrait effectue avec succes par: Amal------
total de retrait a cet instant:1400Dh!
------Solde suffisant!(wait 2 seconds please!)------
------retrait effectue avec succes par: Amal------
total de retrait a cet instant:1600Dh!
------Solde suffisant!(wait 2 seconds please!)------
------retrait effectue avec succes par: Amal------
total de retrait a cet instant:1800Dh!
------Solde suffisant!(wait 2 seconds please!)------
------retrait effectue avec succes par: Amal------
total de retrait a cet instant:2000Dh!


----------votre solde maintenant est de :50300.0
```

2. La classe ***« SoldeCompte »*** qui contient un attribut solde, une méthode « DeposerArgent(float montant) » qui ajoute un montant dans le compte, et une autre méthode « RetirerArgent (float montant) » pour Retirer un montant. Ces deux méthodes doivent être synchronisées.

- Si le solde est égal à 0 l'opération « RetirerArgent » doit attendre jusqu'à un

dépôt d'argent qu'il soit effectué. (utiliser wait(), notify)

Code :

```
class SoldeCompte{
    float solde;
    public SoldeCompte(float solde) {
        super();
        this.solde=solde;
    }
```

```
    public synchronized void RetirerArgent(float montant) {
        System.out.println("------Verification de solde!-----");
        if(montant>this.solde ) {
            System.out.println("Loading..\n------montant non disponible!------");
            try {
                //on attend
                wait(2000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        System.out.println("\n\n***retrait  en cours de "+montant+"***");
        this.solde-=montant;
        System.out.println("\n----- solde actuel: \t"+solde);
    notifyAll();
  }
 public synchronized void DeposerArgent(float montant) {


    System.out.println("\n***depot en cours de "+montant+"***");
    this.solde+=montant;
    System.out.println("\n\n ----solde actuel: \t"+solde);
//notify();
}
}
```

3. Une classe qui implémente l'interface « Runnable » et qui a pour tâche déposer et retirer une somme d'argent.

```
class depotRetraitClass implements Runnable  {
    SoldeCompte solde;
    public depotRetraitClass(SoldeCompte sold) {
        solde=sold;
    }

    @Override
    public void run() {


            float n=1000;
            float m=1500;
```

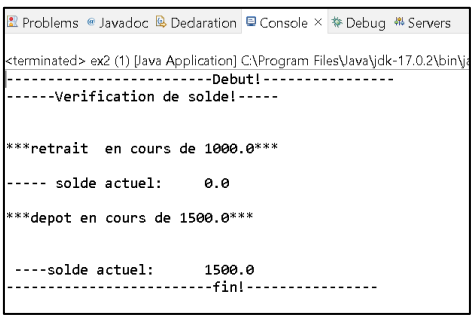
```
            solde.RetirerArgent(m);
            solde.DeposerArgent(n);
    }
    }
```

4. Dans la méthode principale on crée deux Threads, ces deux Threads effectuent plusieurs opérations de retrait et de dépôt d'argent simultanément. (le programme affiche à chaque fois le solde courant après chaque opération)

Code :

```
public class ex2 {

    public static void main(String[] args) {

    SoldeCompte s=new SoldeCompte(1000);
    depotRetraitClass dr= new   depotRetraitClass(s);
    Thread DR =new Thread(dr);

    System.out.println("-------------------------Debut!----------------");
    DR.start();
    try {
        DR.join();
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    System.out.println("-------------------------fin!----------------");
}
}
```

Exécution :



TRAVAUX PRATIQUES 8
– JAVA POO

AU : 2021/2022