Laborator 3 - 4

Fire de execuție. Aplicatii implementate in Java.

Continut laborator:
1. Obiectivul lucrarii.
2. Notiuni teoretice:
a. Crearea firelor de execuţie
b. Crearea unui fir de execuţie prin extinderea clasei Thread/ implementarea interfeţei Runnable
c. Prioritatile firelor de executie
3. Cerinte laborator.
4. Tema.

1. Obiectivul lucrarii.

Lucrarea de faţă are rolul de a prezenta şi familiariza studentul cu modul de construire şi utilizare a firelor de execuţie în Java.
La sfârşitul acestei lucrări, studentul va avea posibilitatea să scrie programe Java în care să utilizeze noţiunile învăţate.

2. Notiuni teoretice

a. Crearea firelor de execuţie

"Multithreading" înseamnă capacitatea unui program de a executa mai multe secvenţe de cod în acelaşi timp. O astfel de secvenţă de cod se numeşte fir de execuţie sau **thread**. Limbajul Java suportă multithreading prin clase disponibile în pachetul **java.lang**. În acest pachet există 2 clase **Thread** şi **ThreadGroup,** şi interfaţa **Runnable**. Clasa **Thread** şi interfaţa **Runnable** oferă suport pentru lucrul cu thread-uri ca entităţi separate, iar clasa **ThreadGroup** pentru crearea unor grupuri de thread-uri în vederea tratării acestora într-un mod unitar.

Există 2 metode pentru crearea unui fir de execuţie: se creează o clasă derivată din clasa **Thread**, sau se creează o clasă care implementeză interfaţa **Runnable**.

b. Crearea unui fir de execuţie prin extinderea clasei Thread

Se urmează etapele:

- se creează o clasă derivată din clasa **Thread**
- se suprascrie metoda **public void run()** moştenită din clasa **Thread**
- se instanţiază un obiect thread folosind **new**
- se porneşte thread-ul instanţiat, prin apelul metodei **start()** moştenită din clasa Thread. Apelul acestei metode face ca maşina virtuală Java să creeze contextul de program necesar unui thread după care să apeleze metoda **run()**.

Exemplu:

```
public class Fir
{
public static void main(String args[])
{
```

```
FirdeExecutie fir=new FirdeExecutie();
fir.start();
System.out.println("Revenim la main");
}
}

class FirdeExecutie extends Thread{

public void run()
{
for(int i=0;i<10;i++)
System.out.println("Pasul "+i);
System.out.println("Run s-a terminat");
}
}
```

Observaţie:

Metoda main() are propriul fir de execuţie. Prin apelul **start()** se cere JVM crearea şi pornirea unui nou fir de execuţie. Din funcţia **start()** se va ieşi imediat. Firul de execuţie corespunzător metodei **main()** îşi va continua execuţia independent de noul fir de execuţie creat.

Să considerăm un alt exemplu simplu în care se vor crea două fire de execuţie ce rulează concomitent. **Metoda sleep()** cere oprirea rulării firului de execuţie curent pentru un interval de timp specificat.

```
public class Fir1
{
public static void main(String args[])
{
FirdeExecutie fir1=new FirdeExecutie();
FirdeExecutie fir2=new FirdeExecutie();
fir1.start();
fir2.start();
System.out.println("Revenim la main");
}
}
class FirdeExecutie extends Thread
{
public void run()
{
for(int i=0;i<10;i++)
{
System.out.println("Pasul "+i);
try{
sleep(500);//oprirea pt. 0,5 secunde a firului de executie
}
catch(InterruptedException e) {System.err.println("Eroare");}
}
System.out.println("Run s-a terminat");
}
}
```

c. Prioritatile firelor de executie

Java defineşte 3 constante pentru selectarea priorităţilor firelor de execuţie:

```
public final static int MAX_PRIORITY; // 10
public final static int MIN_PRIORITY; // 1
public final static int NORM_PRIORITY; // 5
```

O metodă importantă în contextul utilizării priorităţilor este

```
public static native void yield()
```

care scoate procesul curent din execuţie şi îl pune în coada de aşteptare.

Iată un exemplu simplu care demonstrează cum se lucrează cu priorităţile firelor de execuţie. Metoda **getName()** (moştenită din clasa **Thread**) returnează numele procesului curent.

```
public class Fir2
    {
public static void main(String args[])
{
FirdeExecutie fir1=new FirdeExecutie("Fir 1");
FirdeExecutie fir2=new FirdeExecutie("Fir 2");
FirdeExecutie fir3=new FirdeExecutie("Fir 3");
fir1.setPriority(Thread.MIN_PRIORITY);
fir2.setPriority(Thread.MAX_PRIORITY);
fir3.setPriority(7);
fir1.start();
fir2.start();
fir3.start();
System.out.println("Revenim la main");
}
}
class FirdeExecutie extends Thread
{
public FirdeExecutie(String s)
{
super(s);
}
public void run()
{
String numeFir=getName();
for(int i=0;i<5;i++)
{
//if(numeFir.compareTo("Fir 3")==0) yield();
System.out.println(numeFir+ " este la pasul "+i);
try{
sleep(500);
}
catch(InterruptedException e) {System.err.println("Eroare");}
}
System.out.println(numeFir+ " s-a terminat");
}
}
```

Observaţie:

Implementările Java depind de platformă. Un program Java care foloseşte fire de execuţie poate avea comportări diferite la execuţii diferite pentru aceleaşi date de intrare.

d. Crearea unui fir de execuţie folosind interfaţa Runnable

Este o modalitate extrem de utilă atunci când clasa de tip **Thread** care se doreşte a fi implementată moşteneşte o altă clasă (Java nu permite moştenirea multiplă). Interfaţa **Runnable** descrie o singură metodă **run()**.

Se urmează etapele:

- se creează o clasă care implementează interfaţa **Runnable**
- se implemetează metoda **run()** din interfaţă
- se instanţiază un obiect al clasei folosind **new**
- se creează un obiect din clasa **Thread** folosind un constructor care are ca parametru un obiect de tip **Runnable** (un obiect al clasei ce implementează interfaţa)
- se porneşte thread-ul creat la pasul anterior prin apelul metodei **start()**.

Exemplu:

```
public class Fir3
{
public static void main(String args[])
{
FirdeExecutie fir=new FirdeExecutie();
Thread thread=new Thread(fir);
thread.start();
System.out.println("Revenim la main");
}
}
class A
{
public void afis()
{
System.out.println("Este un exemplu simplu");
}
}
class FirdeExecutie extends A implements Runnable
{
public void run()
{
for(int i=0;i<5;i++)
System.out.println("Pasul "+i);
afis();
System.out.println("Run s-a terminat");
}
}
```

Observaţii utile:

Un fir de execuţie se poate afla la un moment dat în una din următoarele stări: **running** (rulează), **waiting** (adormire, blocare, suspendare), **ready** (gata de execuţie, prezent în coada de aşteptare), **dead** (terminat). Fiecare fir de execuţie are o prioritate de execuţie. În general thread-ul cu prioritatea cea mai mare este cel care va accesa primul resursele sistem.

O altă clasă aparte de thread-uri sunt cele **Daemon** care sunt thread-uri de serviciu (aflate în serviciul altor fire de execuţie). Când se porneşte maşina virtuală Java, există un singur fir de execuţie care nu este de tip Daemon şi care apelează metoda main(). JVM rămâne pornită cât există activ un thread care să nu fie de tipul Daemon.

3. Cerinte laborator.

Se va verifica fiecare din sursele java prezentate anterior si se vor testa pe mai multe exemple, pentru a observa modul de functionare.

Realizarea de implementari pentru problemele propuse.

4. Tema.

Probleme propuse spre rezolvare folosind fire de executie prin extinderea clasei Thread

1. Se da un tablou bidimensional cu m linii si n coloane cu componente intregi. Determinati numarul de componente impare.
   Exemplu
   ```
   3 4
   1 20 5 256
   51 4 23 55
   1 2 3 4
   ```
   Se va afisa 6.

2. Se da un tablou bidimensional cu m linii si n coloane cu component intregi. Ordonati crescator liniile tabloului. Apoi afisati tabloul.
   Exemplu
   ```
   3 4
   1 20 5 256
   51 4 23 55
   1 4 3 2
   ```
   se va afisa
   ```
   1 5 20 256
   4 23 51 55
   1 2 3 4
   ```

3. Se da un vector cu n component intregi. Inlocuiti fiecare componenta cu cel mai mare patrat perfect mai mic sau egal decat componenta. Apoi afisati vectorul.
   4
   80 23 100 125
   se va afisa
   64 16 81 121

Probleme propuse spre rezolvare folosind fire de executie prin interfaţa Runnable

1. Se da un tablou bidimensional cu m linii si n coloane cu componente intregi. Determinati numarul de componente cu prima cifra para.
   Exemplu
   ```
   3 4
   1 20 5 256
   50 4 23 55
   1 2 3 4
   ```
   Se va afisa 6.

2. Se da un tablou bidimensional cu m linii si n coloane cu component intregi. Pentru fiecare linie scrieti unul din mesajele: DA sau NU. DA, daca linia are toare componentele pare, respective NU contrar.. Apoi afisati tabloul.
   Exemplu
   ```
   3 4
   10 2 50 250
   51 4 23 55
   16 2 34 20
   ```
   se va afisa
   ```
   DA
   NU
   DA
   ```

3. Se da un vector cu n component intregi. Inlocuiti fiecare componenta cu produsul cifrelor nenule. Apoi afisati vectorul.
   4
   80 23 101 315
   se va afisa
   8 6 1 15