CAPITOLUL 2

2.1. Gruparea thread-urilor (firelor de execuție). Prioritatea firelor de execuție.

2.1.1. Gruparea firelor de execuție

Gruparea firelor de execuție pune la dispoziționeun mecanism pentru manipularea acestora ca un întreg. De exemplu, putem să lansăm sau să suspendăm toate firele dintr-un grup cu un singur apel la metodă. Gruparea firelor de execuție se realizează prin intermediul clasei `ThreadGroup`.

Fiecare fir de execuție Java este membru al unui grup, indiferent dacă specificăm explicit acest lucru. Afilierea unui fir de execuție la un anumit grup, se realizează la crearea acestuea şi devine permanentă, deci nu vom putea muta un fir de execuție dintr-un grup în altul, după ce acesta a fost creat. În cazul în care, creăm un fir de execuție fără a specifica în constructor din ce grup face parte, el va fi plasat automat în acelaşi grup cu firul de execuție care l-a creat. La pornirea unui program Java se creează automat un obiect de tip **`ThreadGroup`** cu numele **`main`**, care va reprezenta grupul tuturor firelor de execuție create direct din program, şi care nu au fost ataşate explicit altui grup. Se poate ignora complet plasarea firelor de execuție în grupuri şi lăsînd sistemul să se ocupe de aceasta, plasîndu-le pe toate în grupul **`main`**. Există situații când programul creează o mulțime de fire de execuție, iar gruparea lor poate uşura substanțial manevrarea acestora. Crearea unui fir de execuție şi plasarea lui într-un grup (altul decât cel implicit) se realizează prin următorii constructori ai clasei `Thread`:

```
public Thread(ThreadGroup group, Runnable target)
public Thread(ThreadGroup group, String name)
public Thread(ThreadGroup group, Runnable target,
                                          String name)
```

Fiecare din aceşti constructori creează un fir de execuție, îl inițializează şi îl plasează într-un grup specificat în argumentul acestuea. În exemplul următor vor fi create două grupuri, primul cu două fire de execuție, iar al doilea cu trei fire de execuție:

```
ThreadGroup grup1 = new ThreadGroup("Producatori");
Thread p1 = new Thread(grup, "Producator 1");
Thread p2 = new Thread(grup, "Producator 2");
ThreadGroup grup2 = new ThreadGroup("Consumatori");
Thread c1 = new Thread(grup, "Consumator 1");
Thread c2 = new Thread(grup, "Consumator 2");
Thread c3 = new Thread(grup, "Consumator 3");
```

Pentru a afla cărui grup aparține un anumit fir de execuție putem folosi metoda `getThreadGroup` a clasei `Thread`. Un grup poate avea ca părinte un alt grup, ceea ce înseamnă că firele de execuție pot fi plasate într-o ierarhie de grupuri, în care rădăcina este grupul implicit `main`, după cum este arătat în Figura 3.

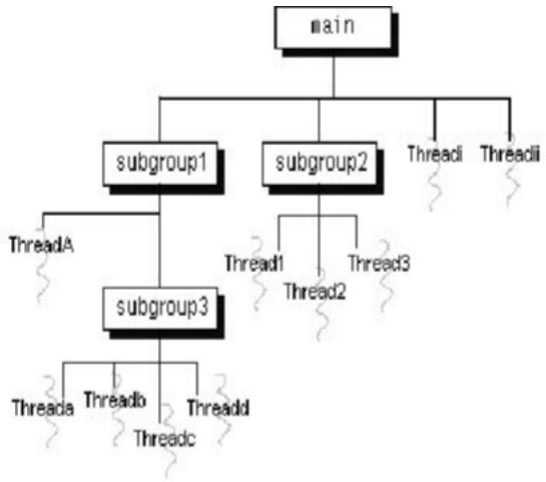

Figura 3 Gruparea firelor de execuție

Exemplu: Enumerarea firelor de execuție active

```
     public class EnumerateTest {
        public void listCurrentThreads()
            { ThreadGroup currentGroup =
          Thread.currentThread().getThreadGroup();
     //aflu numarul firelor de execuţie active
            int numThreads = currentGroup.activeCount();
     //pun intr-un vector referinţe la firele de exec.
//active
          Thread[] listOfThreads = new Thread[numThreads];
          currentGroup.enumerate(listOfThreads);
             //le afisez pe ecran
          for (int i = 0; i < numThreads; i++)
            System.out.println("Thread #" + i + " = " +
            listOfThreads[i].getName());
        }
     }
```

Structura firelor de execuție şi grupurilor de execuție poate fi reprezentată în următoarea forma:

```
     MainGrupa{subGrupa1{subGrupa3{Tha, Thb, Thc, Thd},
ThA},subGrupa2{Th1, Th2, Th3},Th1, Th2}.
```

2.1.2. Prioritatea firelor de execuție.

Prioritatea firelor de execuție este un număr întreg, care determină prioritatea relativă a unui fir față de altul.

Fiecare fir de execuție are o prioritate cuprinsă între valorile MIN_PRIORITY şi MAX_PRIORITY. Aceste două variabile finale sunt declarate în clasa **`Thread`** . În mod implicit, un fir de execuție are prioritatea NORM_PRIORITY, de asemenea definită în clasa **`Thread`** .

Mediul de execuție Java ordoneaza firele de execuție pentru preluarea controlul unității centrale în funcție de prioritatea lor. Dacă există mai multe fire cu prioritate maximă, acestea sunt ordonate după un

algoritm numit ”round-robin”. Firele cu prioritate mai mică sunt prelucrate doar atunci când toate firele cu prioritate mare sunt în starea **”Nu Rulează”** .

Prioritatea unui fir de execuţie poate fi determinată prin metoda **`getPriority().`** Metoda **`getPriority()`** returnează prioritatea firului de execuţie, care este un număr întreg şi reprezintă prioritatea curentă a firului de execuţie. Pentru a stabili prioritatea, se foloseşte metoda **`setPriority()`** şi primeşte ca parametru un număr întreg, care reprezintă prioritatea dorită.

Pentru a modifica prioritatea grupului de fire de execuţie se foloseşte metoda setMaxPriority().

Schimbarea priorităţii unui fir de execuţie este periculoasă, dacă metoda cu prioritate mare nu se termină foarte repede sau dacă nu are opriri dese. În caz contrar, celelalte metode nu vor mai putea primi controlul unităţii centrale.

Există însă situaţii în care putem schimba această prioritate fără pericol, de exemplu când avem un fir de execuţie care nu face altceva decât să citească caractere de la utilizator şi să le memoreze într-o zonă temporară. În acest caz, firul de execuţie este în cea mai mare parte a timpului în starea `Nu Rulează` din cauză că aşteaptă terminarea unei operaţii de intrare/ieşire. În clipa în care utilizatorul tastează un caracter, firul va ieşi din starea de aşteptare şi va fi primul planificat la execuţie din cauza priorităţii sale ridicate. În acest fel utilizatorul are senzaţia că aplicaţia răspunde foarte repede la comenzile sale.

În alte situaţii, avem de executat o sarcină cu prioritate mică. În aceste cazuri, putem stabili, pentru firul de execuţie, care execută aceste sarcini o prioritate redusă.

Exemplu de creare a grupelor cu fire de execuţie şi modificarea priorităţilor.

```
public class ThreadGroup {
public static void main(String[] args)
          { String name = null;
   ThreadGroup sys =
    Thread.currentThread().getThreadGroup();

   Thread curr = Thread.currentThread();
   curr.setPriority(curr.getPriority() + 1);
   sys.list();
    ThreadGroup g1 = new ThreadGroup(" g1");

   Thread t=null;

   t= new Thread(g1,new Fir("  A"));
   t.setPriority(Thread.MAX_PRIORITY);

   Thread t1=null;

   t1= new Thread(g1,new Fir("  B"));
   t.setPriority(Thread.MAX_PRIORITY-6);

   Thread t2=null;

  t2= new Thread(g1,new Fir("  C"));
  t.setPriority(Thread.MAX_PRIORITY-6);
  g1.list();

   ThreadGroup g2 = new ThreadGroup(g1, " g2");

for (int i = 0; i < 5; i++){
          name=String.valueOf(  Integer.toString(i));
```

```
System.out.println(name);
   Thread thread = new Thread(g2,new Fir(name));
}
g2.list();
   System.out.println("Starting all threads:");
}
}
class Fir extends Thread
 { String name;
   public Fir(String name) {
            this.name=name;
            start();
   }
          public void run() {
                   System.out.println("Salut de la firul  "+ name);
                   try {
                            Thread.sleep(100);
                    } catch (InterruptedException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                   }
          }

}
```

Rezultatul realizării:

```
java.lang.ThreadGroup[name=main,maxpri=10] Thread[main,6,main]
java.lang.ThreadGroup[name= g1,maxpri=10]
Salut de la firul     B
Salut  de  la  firul   A
Salut  de  la  firul   C
Starting all threads:
java.lang.ThreadGroup[name=      g2,maxpri=10]
Salut de la firul  0
Salut  de  la  firul  2
Salut  de  la  firul  1
Salut  de  la  firul  4
Salut  de  la  firul  3
```

Lucrare de laborator nr. 2

Tema lucrării: Gruparea firelor de execuție. Modificarea priorității.

Scopul lucrării:

Studierea mecanismelor de grupare a firelor de execuție (thread groups) și a modificării priorității acestora în cadrul unui program, pentru a înțelege modul în care sistemul de operare și limbajul de programare gestionează concurența și resursele procesorului.

Obiectivele lucrării:

- **Familiarizarea cu noțiunile de bază** privind:
  - grupurile de fire (thread groups);
  - prioritățile grupelor și a firelor și influența lor asupra planificării.
- **Crearea și gestionarea grupurilor de fire**, demonstrând:
  - cum se organizează mai multe fire într-un grup comun;
  - cum se aplică metodele de control asupra unui grup de fire.
- **Studierea mecanismului de prioritizare**, prin:
  - modificarea priorității firelor individuale;
  - analizarea comportamentului programului atunci când diferite fire au priorități diferite.
- **Aplicarea practică** a conceptelor prin scrierea și rularea unor programe care:
  - creează și rulează mai multe fire în paralel;
  - utilizează grupuri de fire pentru organizare;
  - modifică și compară prioritățile firelor.
- **Analiza rezultatelor** pentru a înțelege:
  - cum influențează prioritățile ordinea de execuție;
  - diferențele dintre execuția firelor cu prioritate ridicată și scăzută;
  - limitările reale ale priorității în funcție de planificatorul sistemului de operare.

Exemplu de realizare:

```
publicclass ThreadGroup1
{ publicstaticvoid main(String[] args) {
    ThreadGroup sys =
      Thread.currentThread().getThreadGroup();
    sys.list();
    sys.setMaxPriority(Thread.MAX_PRIORITY - 1);
    Thread curr = Thread.currentThread();
    curr.setPriority(curr.getPriority() + 1);
    sys.list();
     ThreadGroup g1 = new ThreadGroup("g1");
    g1.setMaxPriority(Thread.MAX_PRIORITY);
    Thread t = new Thread(g1, "A")
          t.start();
          t.setPriority(Thread.MAX_PRIORITY);
    g1.list();
    g1.setMaxPriority(Thread.MAX_PRIORITY - 2);
    g1.setMaxPriority(Thread.MAX_PRIORITY);
    g1.list();
          t = new Thread(g1, "B");
          t.start();
          t.setPriority(Thread.MAX_PRIORITY);
    g1.list();
    g1.setMaxPriority(Thread.MIN_PRIORITY + 2);
          t = new Thread(g1, "C");
          t.start();
          t.setPriority(t.getPriority() -1);
    g1.list();
```

```
    ThreadGroup g2 = new ThreadGroup(g1, "g2");
    g2.list(); // (8)
    g2.setMaxPriority(Thread.MAX_PRIORITY);
    g2.list(); // (9)
for (int i = 0; i < 5; i++)
new Thread(g2, Integer.toString(i)).start();;
    sys.list(); // (10)
    System.out.println("Starting all threads:");

}} ///:~
```

Rezultatul rulării programului:

```
(1) ThreadGroup[name=system,maxpri=10]
      Thread[main,5,system]
(2) ThreadGroup[name=system,maxpri=9]
      Thread[main,6,system]
(3) ThreadGroup[name=g1,maxpri=9]
      Thread[A,9,g1]
(4) ThreadGroup[name=g1,maxpri=8]
      Thread[A,9,g1]
(5) ThreadGroup[name=g1,maxpri=8]
      Thread[A,9,g1]
      Thread[B,8,g1]
(6) ThreadGroup[name=g1,maxpri=3]
      Thread[A,9,g1]
      Thread[B,8,g1]
      Thread[C,6,g1]
(7) ThreadGroup[name=g1,maxpri=3]
      Thread[A,9,g1]
      Thread[B,8,g1]
      Thread[C,3,g1]
(8) ThreadGroup[name=g2,maxpri=3]
(9) ThreadGroup[name=g2,maxpri=3]
(10)ThreadGroup[name=system,maxpri=9]
      Thread[main,6,system]
      ThreadGroup[name=g1,maxpri=3]
        Thread[A,9,g1]
        Thread[B,8,g1]
        Thread[C,3,g1]
        ThreadGroup[name=g2,maxpri=3]
          Thread[0,6,g2]
          Thread[1,6,g2]
          Thread[2,6,g2]
          Thread[3,6,g2]
          Thread[4,6,g2]
Starting all threads:
All threads started
```

Etapele de realizare a lucrării:

1. Studenții grupei se divizează în echipe a câte doi. Lucrarea practică va fi realizată în echiră.
2. Implementați aplicația (din ***Sarcina lucrării***). În echipă decideți cine ce nivel a aplicației va implementa.
3. După ce ați primit instrucțiuni de la profesor, introduceți modificările în ramura dvs. separată de pe GitHub. Efectuați commit pentru modificări în codul aplicației (comanda git commit).
4. La finalizarea lucrării, elaborați un raport care trebuie să conțină - numele, prenumele, grupa, sarcina și opțiunea dvs. pentru implementarea sarcinii, scurtă descriere, un link către codul sursă de pe GitHub. Salvați raportul în format PDF sau WORD și trimiteți pe ELSE.

Sarcinile propuse spre realizare:

1.Problema penrtu nivelul mediu:

Creaţi o structură, conform formulei date în tabelul 1, conform variantei. Enumeraţi toate firele de execuţie din grupa principală şi subgrupele ce le conţine. Afişaţi informaţia despre numele firului de execuţie, numele grupei din care face parte, şi prioritatea sa. Prioritatea fiecărui Thread este indicată în paranteze () pentru fieacare fir de execuţie (vezi tabelul 1). Nu suprascriţi meroda run() din clasa Thread.

2. Problema penrtu nivelul avansat:

Creaţi o structură, conform formulei date în tabelul 1, conform variantei. Enumeraţi toate firele de execuţie din grupa principală şi subgrupele ce le conţine. Prioritatea fiecărui Thread este indicată în paranteze () pentru fieacare fir de execuţie (vezi tabelul 1). Suprascriţi meroda run() din clasa Thread în care firul de execuţie să extragă la consolă numele său şi grupul la care aparţine. Extrageţi prioritatea firelor de execuţie.

Tabelul 1. Structura firelor de execuţe în dependenţă de variantă

Nr. Variantei	**Formula structurii, compusă din fire şi grupuri de fire**
1	Main{G1{G3{Tha(3), Thb(3), Thc(3), Thd(3)}},G2{Th1(4), Th2(5), Th3(5)},Th1(7), Th2(7), ThA(3)}
2	Main{G2{G4{G1{Tha(1), Thb(3), Thc(8), Thd(3)}}, ThA(1)},G3{Th1(4), Th2(3), Th3(5)},Th1(3), Th2(6)}
3	Main{GN{GH{Tha(4), Thb(3), Thc(6), Thd(3)}, ThA(3)},GM{Th1(2), Th2(3), Th3(3)},Th1(8), Th2(3)}
4	Main{GO{GZ{Tha(1), Thb(3), Thc(3), Thd(7)}},GV{ ThA(3)},GF{Th1(5), Th2(3), Th3(9)},Th1(3), Th2()}
5	Main{Th1(3),GE{GH{Tha(4),Thb(3),Thc(2),Thd(1)}, ThA(3)},GK{Th1(3), Th2(6), Th3(3)}, Th2(7)}
6	Main{G1{ThA(3)}{G3{Thf(3), Thb(7), Thc(3), Thd(3)} G2{Th8(3), Th9(4), Th3(3)},Th1(3), Th2(3)}
7	Main{ThA(3)},G2{Th1(5), Th2(3), Th33(7)},Th11(3), Th22(3), G1{G3{Thaa(2), Thbb(3), Thcc(8), Thdd(3)}}
8	Main{G4{G3{Tha(2), Thb(8), Thc(3), Thd(3), G2{Th1(3), Th2(3), Th3(3), ThA(3)},Th1(8), Th2(3)}} }
9	Main{G6{ ThA(3)},Th1(4), Th2(3), G2{Th1(2), G3{Tha(2), Thb(3), Thc(4), Thd(3)}, Th2(3), Th3(3)}}
10	Main{G7{G3{Tha(6), Thb(3), Thc(6), Thd(3)}, ThA(7), ThB(6)}, Th2(3), G2{Th1(7), Th2(3), Th3(3)}}

Criterii de evaluare:

1. Crearea schemei în comformitate cu variantul corespunzător.
2. Crearea şi iniţializarea grupelor şi a thread-urilor din problema penrtu nivelul mediu .
3. Crearea şi iniţializarea grupelor şi a thread-urilor din problema penrtu nivelul avansat .
4. Crearea interfeţei grafice a programului.
5. Corectitudinea codului - verificarea dacă codul este corect, fără erori de funcţionare.
6. Respectarea instrucţiunilor şi cerinţelor - verificarea corectitudenii cerinţelor sarcinii, cum ar fi numărul de fire de execuţie.
7. Optimizarea codului - Evaluarea eficienţei codului în utilizarea resurselor şi evitarea codului.
8. Respectarea termenului de susţinere - evaluarea punctajului în funcţie de punctualitate, dacă lucrarea a fost predată în termenul stabilit.

9. Evaluarea cunoștințelor - explicațiile oferite despre procesul de realizare a lucrării, ceea ce poate include descrierea funcțiilor principale și a logicii utilizate.

10. Utilizarea de către student a IA.

Pentru obținerea notei 5 - 6 sunt obligatorii criteriile 1,2,5,6,8,9

Pentru obținerea notei 7 - 8 sunt obligatorii criteriile 1,3,5,6,8,9

Pentru obținerea notei 9 - 10 sunt obligatorii criteriile 1-9

Dacă a fost utilizat criteriul 10 nota este scăzută cu 2 baluri, numai dacă studentul sa lămurit în funcționarea codului. În caz contrar lucrare de laborator nu este susținută.

***Întrebări de verificare*:**

- Ce este un **grup de fire (ThreadGroup)** și ce rol are în gestionarea concurenței?
- Cum se creează și se asociază un fir de execuție la un anumit grup?
- Ce metode de lucru cu grupurile de fire sunt disponibile în limbajul Java (sau în limbajul de programare utilizat la laborator)?
- Ce reprezintă **prioritatea unui fir de execuție** și cum influențează ea planificarea?
- Care este intervalul valorilor de prioritate pentru firele de execuție și care este valoarea implicită?
- Care sunt constantele standard de prioritate definite în Java (`MIN_PRIORITY`, `NORM_PRIORITY`, `MAX_PRIORITY`)?
- Cum se poate modifica prioritatea unui fir în program?
- De ce modificarea priorităților nu garantează întotdeauna ordinea exactă de execuție a firelor?
- Cum influențează sistemul de operare comportamentul planificării firelor, indiferent de prioritățile setate în cod?
- Cum poate fi vizualizat sau verificat comportamentul firelor cu priorități diferite într-un program experimental?

Lista de literatură recomandată:

1. Petre Dini – *Programare avansată în Java*, Editura Polirom, Iași.
2. Cursuri universitare de „Sisteme de operare” și „Programare concurentă” de la universitățile tehnice din România și Republica Moldova (materiale online).
3. Блох Дж. – *Java. Эффективное программирование*, 3-е изд., Вильямс.
4. Хорстманн К. – *Java. Библиотека профессионала. Том 1–2*, Вильямс.
5. Herbert Schildt – *Java: The Complete Reference*, 12th Edition, McGraw-Hill.
6. Brian Goetz et al. – *Java Concurrency in Practice*, Addison-Wesley.