Sincronizarea firelor de execuție prin metodele clasei Thread.

1. Metodele clasei Thread

1.1 Constructorii clasei Thread

- Crearea unui nou obiect Thread
  public Thread();
- Crearea unui nou obiect Thread cu indicarea obiectului pentru care va fi activată metoda run().
  public Thread(Runnable target);
- Analogic constructorului precedent, dar se indică și numele obiectului Thread
  public Thread(Runnable target, String name);
- Crearea unui nou obiect Thread cu indicarea numelui acestuia
  public Thread(String name);
- Crearea unui nou obiect Thread, cu indicarea numelui grupului căruii îi aparține și obiectul pentru care va fi activată metoda run().
  public Thread(ThreadGroup group, Runnable target);
- Analogic constructorului precedent, dar se indică și numele obiectului Thread
  public Thread(ThreadGroup group, Runnable target, String name);
- Crearea unui nou obiect Thread cu indicarea numelui grupului căruii îi aparține și numele acestuia.
  public Thread(ThreadGroup group, String name);

1.2. Metodele clasei Thread

- activeCount – determină numărul de thread-uri active din grupul căruia îi aparține
  public static int activeCount();
- currentThread – determină thread-ul curent activ
  public static Thread currentThread();
- getName – returnează numele thread-ului.
  public final String getName();
- getPriority returnează prioritatea curentă a thread-ului
  public final int getPriority();
- getThreadGroup – returnează grupul căruii îi aparține thread-ul
  public final ThreadGroup getThreadGroup();
- interrupt – întrerupe realizarea thread-ului
  public void interrupt();
- Interrupted – determină dacă thread-ul este întrerupt
  public static boolean interrupted();
- isAlive – determină dacă thread-ul se realizează sau nu
  public final boolean isAlive();
- isDaemon – determină este thread-ul daemon
  public final boolean isDaemon();
- join – așteaptă realizarea complectă a thread-ului sau așteaptă realizarea thread-ului pe parcursul timpului indicat în milisecunde sau nanosecunde
  public final void join();
  public final void join(long millis);
  public final void join(long millis, int nanos);
- run – metoda este activată pentru a realiza thread-ul
  public void run();
- setDaemon – indică că thread-ul creat va fi daemon
  public final void setDaemon(boolean on);

- setName – atribue nume thread-ului
  public final void setName(String name);
- yield – oprește temporar thread-ul activ și permite execuția altui thread
  public static void yield();
- setPriority – setarea priorității
  public final void setPriority(int newPriority);
- sleep – oprește execuția (”adormirea”) thread-ului pe timpul indicat în milisecunde și nanosecunde
  public static void sleep(long millis);
  public static void sleep(long millis, int nanos);
- start – activarea thread-ului
  public void start();
- stop – dezactivarea finală a thread-urilor
  public final void stop();

Exemplu de realizare a mai multor mrtode de sincronizare din clasa Thread:

```
class Lab3 {
    static Thread1 primu = new Thread1();
    static Thread2 al_doilea = new Thread2();
    static Thread3 al_treilea = new Thread3();
    static Thread4 al_patrulea = new Thread4();
    static int counter = 0;
    static int counter2 = 0;
    static String info1 = "\nNumele";
    static String info2 = "\nFamilia";
    static String info3 = "\nProgramarea concurenta si distribuita";
    static String info4 = "\nGrupa";

    static class Thread1 extends Thread {
        @Override
        public void run() {
            System.out.println("");
            while (true) {
                if (!(primu.isInterrupted())) {
                    System.out.println("Starting Thread 1");
                    int p_term = 0;
                    int d_term = 0;
                    int p_term1 = 0;
                    int d_term2 = 0;
                    int n = 0;
                    for (int i = 200; i <= 300; i += 2) {
                        n++;
                        if (n == 1) {
                            p_term = i;
                        }
                        if (n == 2) {
                            d_term = i;
                        }
                        if (n == 3) {
                            p_term1 = i;
                        }
                        if (n == 4) {
                            d_term2 = i;
```

```
                            n = 0;
                            System.out.println(this + "(" + p_term + "*" + d_term +
"+" + p_term1 + "*" + d_term2 + ")=" +
                                    (p_term * d_term + p_term1 * d_term2));
                        }

                    }
try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                    }

                    primu.interrupt();
                }


                char afisare;

                if (al_doilea.getPriority() == 9 && al_patrulea.getPriority() == 9 &&
primu.getPriority() == 9) {
                    for (int i = 0; i < info1.length(); i++) {
                        afisare = info1.charAt(i);
                        try {
                            Thread.sleep(100);
                        } catch (InterruptedException e) {
                        }
                        System.out.print(afisare);
                    }
                    al_treilea.setPriority(9);
                    break;

                }
            }
        }
    }


    static class Thread2 extends Thread {
        @Override
        public void run() {
            System.out.println("");
            while (true) {
                if (!(al_doilea.isInterrupted())) {
                    System.out.println("Starting Thread 2");
                    int p_term = 0;
                    int d_term = 0;
                    int p_term1 = 0;
                    int d_term2 = 0;
                    int n = 0;
                    for (int i = 106; i >= 6; i -= 2) {
                        n++;
                        if (n == 1) {
                            p_term = i;
                        }
                        if (n == 2) {
```

```
                            d_term = i;
                        }
                        if (n == 3) {
                            p_term1 = i;
                        }
                        if (n == 4) {
                            d_term2 = i;
                            n = 0;
                            System.out.println(this + "(" + p_term + "*" + d_term +
"+" + p_term1 + "*" + d_term2 + ")=" +
                                    (p_term * d_term + p_term1 * d_term2));

                        }
                    }


                    al_doilea.interrupt();

                }
                System.out.println();
                al_doilea.setPriority(9);
                char afisare;
                if (al_doilea.getPriority() == 9) {
                    for (int i = 0; i < info2.length(); i++) {
                        afisare = info2.charAt(i);
                        try {
                            Thread.sleep(100);
                        } catch (InterruptedException e) {
                        }
                        System.out.print(afisare);
                    }
                    al_patrulea.setPriority(9);
                    break;
                }

            }
        }
    }

    static class Thread3 extends Thread {

        @Override
        public void run() {
            while (true) {
                if (!(al_treilea.isInterrupted())) {
                    System.out.println("Starting Thread 3");
                    for (int i = 234; i <= 1000; i += 1) {
                        System.out.print(al_treilea.getName() + ":" + i + " ");
                        counter++;
                        if (counter == 10) {
                            System.out.println(" ");
                            counter = 0;
                        }
                        if (i == 1000) {
                            System.out.println("\n ");
```

```
                        }
                    }
                    al_treilea.interrupt();
                }
                char afisare;
                if (al_doilea.getPriority() == 9 && al_patrulea.getPriority() == 9 &&
primu.getPriority()== 9 && al_treilea.getPriority() == 9) {
                    for (int i = 0; i < info3.length(); i++) {
                        afisare = info3.charAt(i);
                        try {
                            Thread.sleep(100);
                        } catch (InterruptedException e) {
                        }
                        System.out.print(afisare);
                    }

                    break;


                }
            }
        }
    }

    static class Thread4 extends Thread {
        @Override
        public void run() {
            while (true) {
                if (!(al_patrulea.isInterrupted())) {
                    System.out.println("Starting Thread 4");
                    for (int i =1234; i >= 457; i -= 1) {
                        System.out.print(al_patrulea.getName() + ":" + i + " ");
                        counter2++;
                        if (counter2 == 10) {
                            System.out.println(" ");
                            counter2 = 0;
                        }
                    }

                    al_patrulea.interrupt();
                }
                char afisare;
                if (al_doilea.getPriority() == 9 && al_patrulea.getPriority() == 9) {
                    for (int i = 0; i < info4.length(); i++) {
                        afisare = info4.charAt(i);
                        try {
                            Thread.sleep(100);
                        } catch (InterruptedException e) {
                        }
                        System.out.print(afisare);
                    }
                    primu.setPriority(9);
                    break;
                }
```

```
            }
        }
    }
    public static void main(String[] args) throws InterruptedException {
        primu.start();

        al_doilea.start();
        al_treilea.start();

        al_treilea.setName("Th3");
       // al_treilea.join(1000);
        al_patrulea.start();
        al_patrulea.setName("Th4");
    }
```

}

Au fost utilizate metodele **`setPriority(), interrupt(), join()`**

Lucrarea de laborator nr. 3

Tema lucrării: ***Sincronizarea firelor de execuție utilizînd metodele clasei Thread.***

Scopul lucrării:

Studierea mecanismelor de sincronizare a firelor de execuție în limbajul Java, utilizând metodele puse la dispoziție de clasa `Thread`, pentru a înțelege modul în care se poate asigura execuția corectă și ordonată a proceselor paralele.

Obiectivele lucrării:

1. Familiarizarea cu conceptele de **thread**, **concurență** și **sincronizare**.
2. Studierea metodelor clasei `Thread` (ex.: `sleep()`, `join()`, `interrupt()`, `isAlive()`).
3. Implementarea unor exemple simple de sincronizare între două sau mai multe fire de execuție.
4. Analiza comportamentului programului în situații de concurență fără sincronizare și cu sincronizare.
5. Aplicarea metodelor clasei `Thread` pentru rezolvarea unor probleme practice.
6. Formarea deprinderilor practice de programare concurentă și gestionare a problemelor de sincronizare.

***Exemplu de realizare*:**

```
public class Lab3PCD {
    static int size = 100;
    static int counter = 0;
    static int[] b = new int[size];
    static int[] a;

    static Straight first = new Straight();
    static Reverse second = new Reverse();
    static ReverseInterval third = new ReverseInterval();
    static StraightInterval fourth = new StraightInterval();

    static intpairone = 0;
    static intpairtwo = 0;
```

```
    static class Straight extends Thread {
int result = 0;
        @Override
        public void run(){
System.out.println("Starting Thread 1");
            for (inti = 0; i< counter; i+=4){
try{
Thread.sleep(100);
                }
                catch (InterruptedExceptione){
e.printStackTrace();
                }
                if ((i+4) >= counter){
                }
                else {
pairone = a[i] + a[i+1];
pairtwo = a[i+2] + a[i+3];
                    result = pairone + pairtwo;
System.out.println ("Current value for Thread 1 is: " + pairone + " + " +
pairtwo + " = " + result);
                }
            }
            while(fourth.isAlive()){
try{
Thread.sleep(1000);
                } catch (InterruptedException e) {
e.printStackTrace();
                }
            }
System.out.println("1: Name");
        }
    }

    static class Reverse extends Thread {
int result = 0;
        @Override
        public void run(){
System.out.println("Starting Thread 2");
            for (inti = counter-1; i>= 0; i-=4){
try{
Thread.sleep(100);
                }
                catch (InterruptedExceptione){
e.printStackTrace();
                }
               // if (a[i] <= 106 && a[i] >= 16) {
                    if ((i - 4) <= 0) {
                    }
                    else {
pairone = a[i] + a[i-1];
pairtwo = a[i-2] + a[i-3];
                        result = pairone + pairtwo;
System.out.println ("Current value for Thread 2 is: " + pairone + " + " +
pairtwo + " = " + result);
                    }
```

```
            }
System.out.println("2: Surname");
        }
    }

    public static class StraightIntervalextends  Thread {
        @Override
        public void run(){
System.out.println("Starting Thread 4 ");
            for (inti = 200; i<= 300; i++){
System.out.print(i + " ");
            }
System.out.println(" ");
            while(second.isAlive()){
try{
Thread.sleep(1000);
                } catch (InterruptedException e) {
e.printStackTrace();
                }
            }
System.out.println("4: Group");
        }

    }

    public static class ReverseInterval extends Thread {
        @Override
        public void run(){
System.out.println("Starting Thread 3 ");
            for (inti = 1000; i<= 1100; i++){
System.out.print(i + " ");
            }
System.out.println(" ");
            while(first.isAlive()){
try{
Thread.sleep(1000);
                } catch (InterruptedException e) {
e.printStackTrace();
                }
            }
System.out.println("3: P C D");
        }
    }

    public static void main(String[] args) {
System.out.println("Printing Array: ");
        for (inti = 0; i< 100; i++) {
            b[i] = (int) Math.round((Math.random() * 100) + 15);
System.out.print (b[i] + " ");
            if (i == 50){
System.out.println (b[i] + " ");
            }
            if (i == 99)
System.out.println();
            if (b[i]%2==0){
                counter++;
            }
```

```
        }
        a = new int[counter+1];
int k = 0;
        for (inti = 0; i< 100; i++){
            if (b[i]%2==0){
                a[k] = b[i];
                k++;
            }
        }

first.start();
second.start();
third.start();
fourth.start();
try{
first.join();
second.join();
third.join();
fourth.join();
        } catch (InterruptedException e) {
e.printStackTrace();
        }

    }
}
```

Rezultatul realizării:

Printing Array:
44 25 18 115 103 83 53 67 95 56 107 17 82 16 85 59 55 30 16 114 40 20 112 39 101 102 59 20 27 60 68 21 86 106 87 81 26 107 68 52 16 92 113 79 62 80 109 67 56 100 63 63 105 114 47 39 34 94 59 26 43 17 40 23 81 104 94 55 56 57 20 18 65 88 81 43 77 103 101 48 40 82 88 91 29 19 28 53 19 51 77 20 38 56 54 33 57 20 70 103 108
Starting Thread 1
Starting Thread 2
Starting Thread 4
Starting Thread 3
1000 1001 1002 1003 200 1004 201 202 203 1005 204 1006 205 1007 206 1008 207 1009 208 1010 209 1011 210 211 1012 212 213 1013 214 215 1014 216 217 1015 218 1016 219 1017 220 1018 221 1019 222 1020 223 1021 224 1022 225 1023 226 1024 227 1025 228 1026 229 1027 230 1028 231 1029 232 1030 233 1031 234 1032 235 1033 236 1034 237 1035 238 1036 239 1037 240 1038 241 1039 242 1040 243 1041 244 1042 245 1043 246 1044 247 248 1045 249 1046 250 1047 251 252 253 1048 254 255 256 1049 1050 257 258 259 260 1051 1052 1053 261 262 263 264 265 266 267 1054 268 1055 269 1056 270 1057 271 1058 272 1059 273 1060 274 275 276 277 278 1061 279 1062 280 1063 281 1064 1065 282 1066 283 1067 284 1068 285 1069 1070 286 1071 287 1072 288 1073 1074 289 290 1075 291 1076 292 293 294 295 1077 296 1078 297 1079 298 1080 299 300 1081 1082 1083 1084 1085 1086 1087 1088 1089 1090 1091 1092 1093 1094 1095 1096 1097 1098 1099 1100
Current value for Thread 2 is: 178 + 74 = 252
Current value for Thread 1 is: 62 + 138 = 200
Current value for Thread 2 is: 94 + 48 = 142
Current value for Thread 1 is: 46 + 130 = 176

Current value for Thread 1 is: 60 + 214 = 274
Current value for Thread 2 is: 170 + 88 = 258
Current value for Thread 2 is: 106 + 154 = 182
Current value for Thread 1 is: 80 + 154 = 234
Current value for Thread 1 is: 132 + 66 = 252
Current value for Thread 2 is: 198 + 66 = 264
Current value for Thread 1 is: 128 + 214 = 342
Current value for Thread 2 is: 128 + 214 = 342
Current value for Thread 2 is: 136 + 154 = 290
Current value for Thread 1 is: 156 + 154 = 304
Current value for Thread 1 is: 68 + 94 = 212
Current value for Thread 2 is: 68 + 94 = 162
Current value for Thread 1 is: 192 + 128 = 188
Current value for Thread 2 is: 192 + 128 = 320
Current value for Thread 1 is: 136 + 122 = 258
Current value for Thread 2 is: 136 + 122 = 254
Current value for Thread 1 is: 116 + 58 = 174
Current value for Thread 2 is: 154 + 46 = 200
Current value for Thread 1 is: 110 + 90 = 200
2: Surname
4: Group
1: Name
3: P C D

***Probleme propuse spre realizare**:*

Problema:

Scrieţi un program care creează patru fire de execuţie. Toate firele vor citi datele din diapazonul indicat în sarcină. Perechile de thread-uri 1-2 și 3-4 au sarcini commune. Primul fir Th1 va realiza: Sarcina 1 din tabelul 3, al doilea fir Th2 va afișa Sarcina 2 din tabelul 3,al treilea fir Th3 va afișa Sarcina 3 din tabelul 3, al patrulea fir Th4 va afișa Sarcina 4 din tabelul 3.

După finalizarea realizării sarcinilor firelor de execuţie thread-ul Th2 va afişa Numele studentului care a efectuat lucrarea dată de laborator, Th4 va afișa grupa, Th1 va afișa Prenumele studentului, Th3 va afișa denumirea disciplinei (pe lung). Literele textului vor apărea pe ecran cu un interval de 100 milisecunde.

Un membru a echipei efectuează realizarea și sincronizarea firelor 1 - 2. Al doilea membru a echipei efectuează realizarea și sincronizarea firelor 3 – 4. Fiecare membru a echipei folosește câte 2 metode de sincronizare diferite.

Se utilizează numai metodele clasei Thread

Tabelul 3 Condiţiile pentru realizarea sarcinii problemei conform variantelor

	Sarcina 1	Sarcina 2	Sarcina 3	Sarcina 4
1	Sumele numerelor pare două câte două începând căutarea şi sumarea de la primul element	Sumele numerelor pare două câte două începând căutarea şi sumarea de la la ultimul element	De parcurs de la început intervalul [100,500]	De parcurs de la sfârșit intervalul [300,700]

2	Suma poziţiilor numerelor pare două câte două începând căutarea şi sumarea de la primul element	Suma poziţiilor numerelor pare două câte două începând căutarea şi sumarea de la ultimul element	De parcurs de la început intervalul [120, 690]	De parcurs de la sfârșit intervalul [1000, 1567]
3	Sumele numerelor impare două cate două începând căutarea şi sumarea de la primul element	Sumele numerelor impare două cate două începând căutarea şi sumarea de la ultimul element	De parcurs de la început intervalul [0, 798]	De parcurs de la sfârșit intervalul [1456, 2111]
4	Sumele poziţiilor numerelor impare două câte două începând căutarea şi sumarea de la primul element	Sumele poziţiilor numerelor impare două câte două începând căutarea şi sumarea de la ultimul element	De parcurs de la început intervalul [234, 987]	De parcurs de la sfârșit intervalul [123, 890]
5	Sumele produselor numerelor de pepoziţii pare două câte două începând căutarea și sumarea cu primul element	Sumele produselor numerelor de pepoziţii pare două câte două începând căutarea și sumarea cu ultimul element	De parcurs de la început intervalul [567, 1002]	De parcurs de la sfârșit intervalul [567, 1100]
6	Sumele produselor numerelor de pe poziţii impare două câte două începând căutarea și sumarea cu primul element	Sumele produselor numerelor de pe poziţii impare două câte două începând căutarea și sumarea cu ultimul element	De parcurs de la început intervalul [654, 1278]	De parcurs de la sfârșit intervalul [123, 908]
7	Sumele produselor numerelor pare două câte două începând căutarea și sumarea cu primul element	Sumele produselor numerelor pare două câte două începând căutarea și sumarea cu ultimul element	De parcurs de la început intervalul [234, 1000]	De parcurs de la sfârșit intervalul [456,1234]
8	Sumele produselor numerelor impare două câte două începând căutarea și sumarea cu primul element	Sumele produselor numerelor impare două câte două începând căutarea și sumarea cu ultimul element	De parcurs de la început intervalul [126, 987]	De parcurs de la sfârșit intervalul [213, 899]
9	Diferența produselor numerelor de pepoziţii pare două câte două începând căutarea și sumarea cu primul element	Diferența produselor numerelor de pepoziţii pare două câte două începând căutarea și sumarea cu ultimul element	De parcurs de la început intervalul [222, 999]	De parcurs de la sfârșit intervalul [3333, 3999]
10	Diderența produselor numerelor de pe poziţii impare două câte două începând căutarea și sumarea cu primul element	Diderența produselor numerelor de pe poziţii impare două câte două începând căutarea și sumarea cu ultimul element	De parcurs de la început intervalul [11,548]	De parcurs de la sfârșit intervalul [1234, 678]

11	Diferența produselor numerelor pare două câte două începând căutarea și sumarea cu primul element	Sumele produselor numerelor pare două câte două începând căutarea și sumarea cu ultimul element	De parcurs intervalul de la început [385, 1024]	De parcurs de la sfârșit intervalul [1000, 408]
12	Sumele produselor numerelor impare două câte două începând căutarea și sumarea cu primul element	Sumele produselor numerelor impare două câte două începând căutarea și sumarea cu ultimul element	De parcurs de la început intervalul [1111, 1748]	De parcurs de la sfârșit intervalul [2000, 1478]

Criterii de evaluare:

1. Crearea şi iniţializarea thread-urilor pentru realizarea sarcinilor.
2. Alegerea metodelor de sincronizare a firelor.
3. Sincronizarea thread-urilor cu metodele potrivite.
4. Crearea interfeţei grafice a programului.
5. Corectitudinea codului - verificarea dacă codul este corect, fără erori de funcționare.
6. Respectarea instrucțiunilor și cerințelor - verificarea corectitudenii cerințelor sarcinii, cum ar fi numărul de fire de execuție.
7. Optimizarea codului - Evaluarea eficienței codului în utilizarea resurselor și evitarea codului.
8. Respectarea termenului de susținere - evaluarea punctajului în funcție 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 1,2,3,5,8,9

Pentru obținerea notei 7 - 8 sunt obligatorii criteriile 1,2,3,4,5,6,8,9

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*:**

1. Care sunt metodele principale ale clasei `Thread` și ce funcționalități oferă acestea?
2. Care este diferența dintre metodele `start()` și `run()`?
3. Cum funcționează metoda `join()` și în ce situații se utilizează?
4. Ce efect are metoda `sleep()` asupra firului de execuție?
5. Cum poate fi întrerupt un fir de execuție utilizând `interrupt()`?

6. Care este rolul metodei `isAlive()` și cum poate fi folosită în controlul execuției?
7. Ce probleme pot apărea dacă mai multe fire accesează simultan aceeași resursă fără sincronizare?
8. Care sunt diferențele dintre sincronizarea prin metodele clasei `Thread` și utilizarea cuvântului cheie `synchronized`?
9. De ce este importantă sincronizarea firelor de execuție în aplicațiile multithreading?

Lista de literatură recomandată:

1. Oancea, M. – *Programare în Java. Ghid practic*, Editura Polirom, Iași, 2019.
2. Moldovan, D. – *Structuri de date și programare orientată pe obiecte în Java*, Editura Universității „Babeș-Bolyai", Cluj-Napoca, 2017.
3. Pîrvu, D. – *Tehnici avansate de programare în Java*, Editura MatrixRom, București, 2016.
4. Хорстманн, К., Корнелл, Г. – *Java. Библиотека профессионала. Том 1, Основы*, 11-е изд., СПб.: Питер, 2022.
5. Эккель, Б. – *Философия Java*, 4-е изд., СПб.: Питер, 2018.
6. Шилдт, Г. – *Java. Полное руководство*, 11-е изд., Москва: Вильямс, 2021.
7. Herbert Schildt – *Java: The Complete Reference*, 12th Edition, McGraw-Hill, 2022.
8. Cay S. Horstmann – *Core Java, Volume I–Fundamentals*, 12th Edition, Pearson, 2021.
9. Brian Goetz et al. – *Java Concurrency in Practice*, Addison-Wesley, 2006.