IMPLEMENTAREA CONCURENTEI IN LIMBAJE DE PROGRAMARE

CONCURENTA IN JAVA

Ioana Leustean

https://docs.oracle.com/javase/tutorial/essential/concurrency/

➢ **Thread**

Orice fir de executie (thread) este un obiect al clasei Thread

➢ **Atributele unui thread**

▪ **ID** - unic pentru fiecare thread
   - este accesat cu getId, nu poate fi modificat

▪ **Name** - este un String
   - este accesat cu : getName, setName

▪ **Priority** - un numar intre 1 si 10
   - este accesata cu: getPriority, setPriority
   - in principiu thread-urile cu prioritate mai mare sunt executate primele
   - setarea prioritatii nu ofera garantii in privinta executiei

▪ **Status** - este accesat cu: getState
   - nu poate fi modificat direct (e.g. nu exista setState)

> **Enum Thread.State**

public static enum Thread.State
extends Enum<Thread.State>

Starile posibile ale unui thread:

- NEW: create dar care nu si-a inceput executia

- RUNNABLE: in executie

- BLOCKED: blocat de lacatul unui monitor

- WAITING: asteapta ca un alt thread sa execute o actiune

- TIMED_WAITING: asteapta un alt thread, dar numai un timp limitat

- TERMINATED: thread-ul si-a terminat executia

> **Ciclul de viata al unui thread**

- exemplu – HowToDoInJava

- exemplu – javatpoint.com

https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.State.html

https://docs.oracle.com/javase/tutorial/essential/concurrency

➢ **Crearea obiectelor de tip Thread:**

▪ Metoda directa
   o   ca subclasa a clasei Thread
   o   implementarea interfetei Runnable

▪ Metoda abstracta
   o   folosind clasa Executors

➢ Definirea unui thread ca subclasa a clasei Thread

```java
public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
            HelloThread t = new HelloThread();
            t.start();
    }}
```

definim o subclasa a clasei Thread
cu propria metoda run()

definim thread-ul ca un obiect din noua clasa
si il pornim folosind metoda start()

- metoda start() porneste thread-ul creand un fir de exeutie separat
- metoda run() contine ceea ce executa thread-ul dupa apelul metodei start()

➢ Definirea unui thread ca subclasa anonima a clasei Thread

```java
public class HelloThreadAn  {

    public static void main(String args[]) {

        Thread t = new Thread() { public void run() {
                        System.out.println("Hello from a thread!");
                        }};
            t.start();
}}
```

➢ Definirea unui thread prin implementarea interfetei Runnable

Interfata Runnable contine o singura metoda: run()

```java
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        Thread t = new Thread (new HelloRunnable());
        t.start();
    }}
```

apelul metodei start() creaza un fir de executie separat, in care este apelata metoda run() a obiectului respectiv

➢ Clasa Thread

```
public class Thread
extends Object
implements Runnable
```

- Metodele ale instantelor:

  - **run()**
  - **start()**
  - **join()**
  - **join(long millisecunde)**
  - **interrupt()**
  - **boolean isAlive()**

- Metode statice (se aplica thread-ului current):

  - **yield()**
  - **sleep(long milisecunde)**
  - **Thread.currentThread()**

➤ Clasa Thread

public class Thread
extends Object
implements Runnable

Metodele ale instantelor:

- **start()**
  porneste thread-ul intr-un fir de executie separate si invoca **run()**
- **run()**
  este suprascrisa sau apelata din Runnable
- **join(), join(long millisecunde)**
  este invocata de thread-ul curent pe un alt doilea thread;
  thread-ul current este blocat pana cand al doilea thread isi termina executia sau
  pana cand expira timpul (milisecunde)
- **interrupt()**
  intrerupe executia thread-ului;
  este folosit in situatia in care un thread cere altui thread sa isi intrerupa executia
- **boolean isAlive()**
  intoarce true atata timp cat thread-ul nu si-a incetat executia

➢ Clasa Thread

public class Thread
extends Object
implements Runnable

Metode statice (se aplica thread-ului current):

▪ **yield()**
   thread-ul cedeaza randul altui thread care are aceeasi prioritate

▪ **sleep(long milisecunde)**
   thread-ul este blocat pentru numarul de milisecunde precizat

▪ **Thread.currentThread()**
   intoarce o referinta la thread-ul care invoca metoda

https://docs.oracle.com/javase/tutorial/essential/concurrency

➤ Multi-threading in Java

- O aplicatie Java porneste (prin apelul functiei main) un thread, numit thread-ul principal.
  Acesta poate porni alte thread-uri.

- Fiecare thread are o prioritate, care poate fi setata de programator.
  Thread-urile cu prioritate mai mare se executa primele, iar cele cu prioritati egala se executa in ordine FIFO.

- Fiecare thread are propria stiva dar poate accesa si date partajate.

- Exista thread-uri utilizator si thread-uri deamon (thread-uri cu prioritate mica, care au rolul de a servi thread-urile utilizator, de exemplu garbadge collector thread; acestea sunt pornite in mare parte de JVM).

- JVM continua sa execute thread-uri pana cand:
    o   este apelata metoda exit a clasei Runtime
    o   toate thread-urile utilizator si-au terminat executia (normal sau printr-o exceptie)

➢ Executia este nedeterminista

```java
public class HelloThread implements Runnable {
    public void run() {
    for (int x = 0; x < 5; x = x + 1)
    System.out.println("Hello from the new thread!");
}

    public static void main(String args[]) {
    Thread t = new Thread (new HelloThread());
     t.start();
     for (int x = 0; x < 5; x = x + 1)
     System.out.println("Hello from the main thread!");
    }}
```

```
C:\myjava>java HelloThread
Hello from the main thread!
Hello from the main thread!
Hello from the main thread!
Hello from the main thread!
Hello from the main thread!
Hello from the new thread!
Hello from the new thread!
Hello from the new thread!
Hello from the new thread!
Hello from the new thread!

C:\myjava>java HelloThread
Hello from the main thread!
Hello from the main thread!
Hello from the main thread!
Hello from the main thread!
Hello from the main thread!
Hello from the new thread!
Hello from the new thread!
Hello from the new thread!
Hello from the new thread!
Hello from the main thread!
```

➢ join() si InterruptedException

```java
public class HelloThread implements Runnable {
    public void run() {
        for (int x = 0; x < 1000; x = x + 1)
        System.out.println("Hello from the new thread!");
    }

    public static void main(String args[]) throws InterruptedException {
        Thread t = new Thread (new HelloThread());
        t.start();
        for (int x = 0; x < 1000; x = x + 1)
        System.out.println("Hello from the main thread!");
        t.join();
}}
```

Thread-ul current (in exemplu thread-ul principal) va astepta pana cand
thread-ul t isi va termina executia; daca threadul t este intrerupt va arunca o exceptie.

➢ Transmiterea unui parametru catre un thread

```java
public static void main(String args[]) {
    Scanner myInput = new Scanner( System.in );
    int n;
    System.out.print( "Enter n " );
    n=myInput.nextInt();
    Thread t = new Thread (new HelloThread(n));
    t.start();
    for (int x = 0; x < n; x = x + 1)
        System.out.println("Hello from the main thread");
}}
```

➤ Transmiterea unui parametru catre un thread

```java
import java.util.Scanner;
public class HelloThread implements Runnable {
    private int n;
    public HelloThread(int n){this.n=n;}
    public void run() {
        for (int x = 0; x < n; x = x + 1)
        System.out.println("Hello from " + Thread.currentThread().getId()+"!");
    }

    public static void main(String args[]) {
        Scanner myInput = new Scanner( System.in );
        int n;
        System.out.print( "Enter n " );
        n=myInput.nextInt();
        Thread t = new Thread (new HelloThread(n));
        t.start();
        for (int x = 0; x < n; x = x + 1)
        System.out.println("Hello from the main thread!");  }}
```

parametrul este transmis constructorului

➢ sleep(ms) si InterruptedException

```java
public class SleepMessages {

    public static void main(String args[]) throws InterruptedException {

        String importantInfo[] = { "This", "is ", "important"};

        for (int i = 0;  i < importantInfo.length; i++) {

            //Pause for 4 seconds
            Thread.sleep(4000);

            System.out.println(importantInfo[i]);
}}}
```

opreste executia threadului curent pentru ms milisecunde si arunca exceptie daca threadul este intrerupt

➢ sleep(ms) cu tratarea InterruptedException

https://docs.oracle.com/javase/tutorial/essential/concurrency/simple.html

```java
public class MessageLoop implements Runnable {

    public void run() {
        String importantInfo[] = {"This", "is", "important"};
        try {
            for (int i = 0;  i < importantInfo.length;i++) {
                Thread.sleep(4000);
                 threadMessage(importantInfo[i]);
            }
        } catch (InterruptedException e) {
            threadMessage("I wasn't done!");
}}}
```

- Thread-ul secundar este creat prin implementarea interfetei Runnable

➢ **sleep(ms) cu tratarea InterruptedException**

- Thread-ul principal creaza un al doilea thread si asteapt ca acesta sa isi termine executia.

```java
public static  void threadMessage(String message) {
    String threadName = Thread.currentThread().getName();
     System.out.format("%s: %s%n", threadName, message);    }


 public static void main(String args[])   throws InterruptedException {

     threadMessage("Starting MessageLoop thread");
     Thread t = new Thread(new MessageLoop());
     t.start();
     threadMessage("Waiting for MessageLoop thread to finish");
     t.join();
    threadMessage("Finally!");}
```

```
C:\Users\igleu\Documents\DIR\ICLP22\Curs 2022\Java2022\pg>java MessageLoop
main: Starting MessageLoop thread
main: Waiting for MessageLoop thread to finish
Thread-0: This
Thread-0: is
Thread-0: important
main: Finally!
```

```java
public static void main(String args[])   throws InterruptedException {

    threadMessage("Starting MessageLoop thread");
    Thread t = new Thread(new MessageLoop());
    t.start();
    threadMessage("Waiting for MessageLoop thread to finish");

    t.interrupted();

    t.join();
    threadMessage("Finally!");}
```

```
C:\Users\igleu\Documents\DIR\ICLP22\Curs 2022\Java2022\pg>java MessageLoop
main: Starting MessageLoop thread
main: Waiting for MessageLoop thread to finish
Thread-0: I wasn't done!
main: Finally!
```

https://docs.oracle.com/javase/tutorial/essential/concurrency

➢ Comunicarea intre thread-uri

■ doua thread-uri care incrementeaza acelasi contor

```java
public class Interference  implements Runnable {
 static Integer counter = 0;


    public void run () {
       for (int i = 0; i < 5; i++) {
         performTask();
       }}


private void performTask () {
     int temp = counter;
     counter++;
     System.out.println(Thread.currentThread()
                   .getName() + " - before: "+temp+" after:" + counter);}
public static void main (String[] args) {.. }}
```

➢ Comunicarea intre thread-uri – data race

▪ doua thread-uri care incrementeaza acelasi contor

```java
public static void main (String[] args) {
    Thread thread1 = new Thread(new Interference());
    Thread thread2 = new Thread(new Interference());
    thread1.start(); thread2.start();
    thread1.join(); thread2.join();  }
```

```
Thread-1 - before: 1 after:2
Thread-0 - before: 0 after:1
Thread-1 - before: 2 after:3
Thread-0 - before: 3 after:4
Thread-1 - before: 4 after:5
Thread-0 - before: 5 after:6
Thread-1 - before: 6 after:7
Thread-0 - before: 7 after:8
Thread-1 - before: 8 after:9
Thread-0 - before: 9 after:10
```

```
Thread-0 - before: 0 after:2
Thread-1 - before: 1 after:2
Thread-0 - before: 2 after:3
Thread-0 - before: 4 after:5
Thread-1 - before: 3 after:4
Thread-0 - before: 5 after:6
Thread-1 - before: 6 after:7
Thread-0 - before: 7 after:8
Thread-1 - before: 8 after:9
Thread-1 - before: 9 after:10
```

data race

https://docs.oracle.com/javase/tutorial/essential/concurrency