Тема №11

Потоки (threads)

Базові поняття

- ***Потоки*** (threads) - підтримка ***багатопоточності*** на рівні самої мови.
- В одній програмі можна запустити декілька потоків, які будуть працювати паралельно.

Деякі застосування

- **“Чутливі” графічні застосування**: в одному потоці виконується анімація та інші операції, а в іншому – кнопка для зупинки та інші елементи керування.
- Сокети.

Потоки: основи

- **Кожний потік є екземпляром класу Thread**.
- **Функціональність потоку визначається методом run().** Потік завершується, коли завершується виконання коду цього методу (або виникає неперехоплене виключення).
- **Запуск потоку здійснюється методом start().**

Організація потоків: два способи

- створення екземпляру класу, **похідного від класу Thread**;
- **реалізація інтерфейсу Runnable;** досить типовим є використання анонімних класів.

Приклад запуску потоку

```
Thread t = new Thread (new Runnable(){
        public void run() {
            for (int i=0;i<1000;i++) {
                result+=Math.random()-0.5;
            }
        }
    });
        t.start(); //Запуск потоку
        …
    t.join(); //Без цього виведення може бути некоректним
        System.out.println("Result is "+result);
```

Більш простий запуск (якщо достатньо анонімного потоку)

```
new Thread () {

        @Override
        public void run() {
            for (int i=0;i<1000;i++) {
                result+=Math.random()-0.5;
            }
            System.out.println("Test indicator is "+result);
        }

    }.start();
```

Клас, похідний від Thread

- Основна функціональність потоку визначається в методі **run().**
- Щоб запустити потік, потрібно **викликати метод start(),** який викликає **run().**
- **Самого лише створення екземпляру недостатньо для запуску потоку.**
- **Якщо просто викликати run() - потік не запускається.**
- Часто **start()** викликається в конструкторі, і тоді потік запускається при створенні екземпляра.

Реалізація інтерфейсу Runnable

- Наприклад, якщо клас уже є підкласом деякого іншого класу.
- Функціональність визначається методом **run().**
- **Типова схема запуску потоку**: створюється екземпляр класу Thread, конструктору якого передається вказівник на екземпляр нашого класу. Далі викликається start().
- Знову-ж таки - часто в конструкторі, і тоді:

Thread tr = new Thread(this);

tr.start();

- Досить типово – як **анонімний клас**.

Приклад програми з кількома потоками

- Програма **demopotok** в каталозі **threads**.
- Створюються три потоки; вони реалізують той самий код, але роблять різні затримки, і тому працюють з різними швидкостями.
- Кожний потік видає повідомлення про поточні значення змінних.
- Кожний потік зупиняється після заданої кількості проходів.

Коментарі: функціональність потоку

```
public void run() {
while (Passes<MaxPasses) {
Passes++;
TotalPasses++;
System.out.println(Passes+"-th pass of thread number
    "+Nomer+"; total passes - "+TotalPasses);
try {sleep(Delay);
}
catch (InterruptedException e) {
System.out.println("Something strange...");
}}
System.out.println ("Thread number "+Nomer+" stopped");
}
```

Коментарі: конструктор

```
public potok(int Delay) {
this.Delay=Delay;
this.start();
Number++;
Nomer=Number;
System.out.println("Thread number
   "+Nomer+ " started");
}
```

Коментарі: початкова ініціалізація

```
static int MaxPasses;
static {System.out.println("Enter amounts of passes");
try {BufferedReader br=new BufferedReader( new
    InputStreamReader
(System.in));
MaxPasses = Integer.parseInt(br.readLine());
}
catch (Exception e) {System.exit(1);}
}
static int Number=0;
static int TotalPasses=0;
int Nomer=0;
int Passes=0;
int Delay;
```

Коментарі: запуск потоків

```
potok tr1=new potok(1000);
potok tr2=new potok(2000);
potok tr3=new potok(5000);
```

Використання змінних

- Кожний потік отримує власні копії локальних змінних та полів екземпляра, але статичні поля спільно використовуються всіма потоками

Пріоритети потоків

- Потоки можуть виконуватися з різними пріоритетами.
- Константи класу Thread: **MIN_PRIORITY=1; NORM_PRIORITY=5; MAX_PRIORITY=10**
- Пріоритет може бути встановлений методом ***setPriority***, хоча ручне управління пріоритетами не дуже рекомендується.

Зупинка потоку

- Є метод для явної зупинки, але ним користуватися не рекомендується.
- Краще - цикл, в якому задати умову завершення потоку.

Метод interrupt

- Запит на переривання потоку.
- Встановлює в потоці статус переривання.
- **Не є примусовим перериванням! Потік має сам вирішити, як обробляти цей запит!**

Приклад застосування interrupt

```
Thread t = new MyThread();
        t.start();
        t.interrupt();
        try {
        t.join();
        }
        catch (InterruptedException ir) {
            System.out.println("InterruptedException
    happened");
        }
    }
```

1-й варіант потоку: interrupt не працює

```
class MyThread extends Thread {
    @Override
    public void run() {
        boolean b=true;
       while (b) {
            if (isInterrupted()) System.out.println("Request for
    interruption received");
            System.out.println("I am still working!");
       }
        System.out.println("Thread finished");
    }
}
```

2-й варіант: потік зупиняється

```
public void run() {
        boolean b=true;
       while (b) {
          if (isInterrupted()) {
                System.out.println("Request for interruption
received");
                b=false;
           }
           System.out.println("I am still working!");
       }
        System.out.println("Thread finished");
}
```

Затримка потоку

- Метод ***sleep().***
- Або викликається для конкретного потоку, або ***Thread.sleep(мс);***

Метод join

- Дозволяє дочекатися завершення іншого потоку. Наприклад, якщо потік ***tr1*** робить виклик ***tr2.join(),*** то він призупиняється і чекає, поки tr2 не завершить роботу.

Стани потоків

- **New** – створений, але не запущений.
- **Runnable** – запущений методом **start**.
- **Dead** – зупинений.
- **Block** – заблокований. Переходить до заблокованого стану після методу **sleep** або **wait**, або якщо він чекає завершення операції введення-виведення, або якщо він чекає завершення синхронізованого методу.

Метод yield

- Передає управління іншому потоку з тим же пріоритетом.

Синхронізація потоків

- Проблема полягає в тому, що часто потрібно заборонити одночасний доступ різних потоків до певних об’єктів або одночасний виклик певних методів.
- Основні методи синхронізації – за ресурсами та за подіями.

Синхронізація за ресурсами

- Ключове слово ***synchronized*** - у формі оператора та у формі модифікатора.

Оператор synchronized

synchronized (expression) {
statements
}

Вираз ***expression*** повертає об’єкт або масив. Перед виконанням критичної секції (***statements***) цей об’єкт ***блокується***, і ніякий інший потік не може мати до нього доступу, поки виконується критична секція.

Приклад: сортування масиву

```
public static void sortIntArray( int[] a) {
…
synchronized (a) {
Sorting
}

}
```

Synchronized як модифікатор

- Якщо модифікатор **synchronized** використовується в описі методу - весь метод оголошується критичною секцією.

Синхронізовані колекції

- **Collections.synchronizedList(new ArrayList());**
- Concurrent API.

Координація потоків: ще один спосіб

- Методи ***wait()*** і ***notify()*** класу **Object**. Ідея - потрібно, щоб потік призупинився і дочекався настання певної події. Якщо потік при роботі з об’єктом викликає метод ***wait(),*** він зупиняється і чекає, поки не буде викликаний метод ***notify()*** цього ж об’єкта.