Back End Development 1

Sesi 8

Multithreading - Synchronized Java

Introduction

Konsep untuk dapat menjalankan task atau tugas lebih dari satu secara paralel, sehingga dengan konsep ini task yang banyak akan cepat selesai karena tidak saling tunggu untuk menyelesaikan task.

Untuk menciptakan Thread di Java dapat dilakukan dengan dua cara, yaitu mengimplement interface Runnable dan meng-extends class Thread.

```
public class ThreadHello implements Runnable {

    private final String name;

    public ThreadHello(String name) {
        this.name = name;
    }

    @Override
    public void run() {
        System.out.println(name + ": Hello");
        try {
            Thread.sleep(250);
        } catch (InterruptedException ex) {
            Logger.getLogger(ThreadHello.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

}
```

Class ThreadHello merupakan sebuah Thread yang mengimplement interface Runnable, ada satu methode yang harus dioverride yaitu run(). Di dalam methode tersebut kita bisa memberikan task atau tugas untuk thread yang Anda buat, sedangkan attribut name digunakan untuk pelabelan thread kelak jika thread tersebut berjalan atau running.

Untuk sleep(250) sebenarnya hanya digunakan untuk mendelay sedikit agar nanti kelihatan urutan pengerjaan task, karena jika tidak diberikan sleep() task berjalan terlalu cepat dan susah untuk memperlihatkan prosesnya.

```
@Test
    public void testRun() {
        for (int i = 0; i < 5; i++) {
            ThreadHello instance = new ThreadHello(String.valueOf(i));
            Thread t = new Thread(instance);
            t.start();
        }
    }
```

Kode test diatas yaitu akan membuat sebanyak 5 thread yang akan mengeksekusi perintah di dalam methode run() class ThreadHello, instance dari ThreadHello harus dilewatkan dalam kontruktor class Thread. Untuk menjalankan panggil method start().

```
T E S T S
Running com.odeng.maven.multithreading.ThreadHelloTest
0: Hello
3: Hello
2: Hello
4: Hello
1: Hello
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.147 sec
```

Ouput dari kode test menunjukkan bahwa urutan “Hello” tidak dijalankan serial atau bergantian, terbukti dengan label yang secara random tampil.

```java
public class ThreadWorld extends Thread {

    @Override
    public void run() {
        System.out.println(getName() + ": World");
        try {
            Thread.sleep(250);
        } catch (InterruptedException ex) {
            Logger.getLogger(ThreadHello.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

}
```

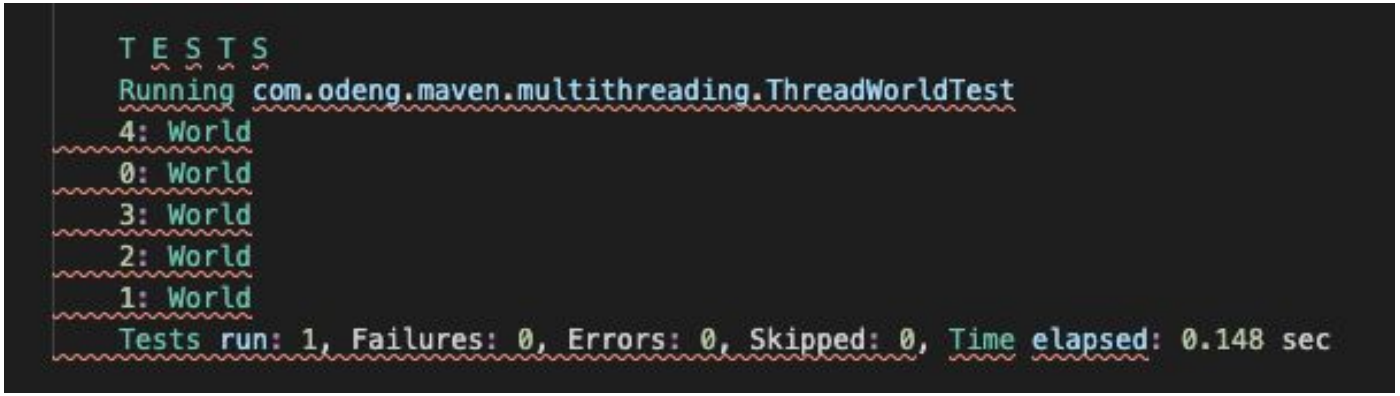
Cara yang kedua yaitu dengan extends class Thread, sama halnya cara yang pertama kita harus melakukan override methode run().

Di dalam methode run() kita berikan perintah atau tugas sesuai yang kita inginkan, dalam contoh menampilkan pesan “World”. Ketika menggunakan cara ini, kita bisa memanfaatkan semua methode yang terdapat pada class Thread, misalkan getName() untuk mengambil label/nama thread yang diset atau diatur ketika memanggil thread.

```
@Test
    public void testRun() {
        for (int i = 0; i < 5; i++) {
            ThreadWorld instance = new ThreadWorld();
            instance.setName("" + i);
            instance.start();
        }

    }
```

Karena sebenarnya class ThreadWorld adalah extends Thread sehingga kita bisa langsung memanfaatkan semua methode yang dimilikinya, setName() digunakan untuk memberikan label/nama pada thread. start() dipanggil untuk menjalankan sebuah thread.



Output dari kode test ingin menjelaskan bahwa message “World” ditampilkan secara acak dan tidak teratur, inilah konsep thread yang mengerjakan tugas dengan paralel.

Synchronized Java

Introduction

Contoh penggunaan method synchronized misalkan ketika mengakses resource secara bersama-sama, hal ini akan sangat berbahaya ketika thread tidak bisa dikontrol. Misalkan ada tugas untuk menulis ke sebuah file yang sama, proses ini thread harus dipastikan selesai terlebih dahulu kemudian bisa dilanjutkan thread yang lain untuk menulis juga.

```
public class NumberGenerator {

    private final int low;
    private final int high;

    public NumberGenerator(int low, int high) {
        this.low = low;
        this.high = high;
    }

    public int randomNumber() {
        Random r = new Random();
        int result = r.nextInt(high - low) + low;
        return result;
    }
}
```

Class di atas digunakan untuk membuat random number dengan batas atas dan batas bawah tertentu, sehingga ketika methode randomNumber() dipanggil akan menghasilkan nilai random. Methode tersebut rencananya akan dipanggil oleh beberapa thread.

Selanjutnya class ThreadNumber yang merupakan turunan dari class Thread adalah sebuah thread yang di dalam methode run() akan memanggil callGenerator(), class NumberGenerator menjadi attribut pada class tersebut agar dapat memanggil methode randomNumber. callGenerator() berisi baris perintah menampilkan 3 random number, yaitu dengan memanggil methode randomNumber() pada object NumberGenerator.

```
public class ThreadNumber extends Thread {

    private final NumberGenerator ng;

    public ThreadNumber(NumberGenerator ng) {
        this.ng = ng;
    }

    @Override
    public void run() {
        callGenerator();
    }

    private void callGenerator() {
//          synchronized (ng) {
              for (int i = 0; i < 3; i++) {
                  System.out.println(getName() + " " + ng.randomNumber());

              }
//          }

    }
```

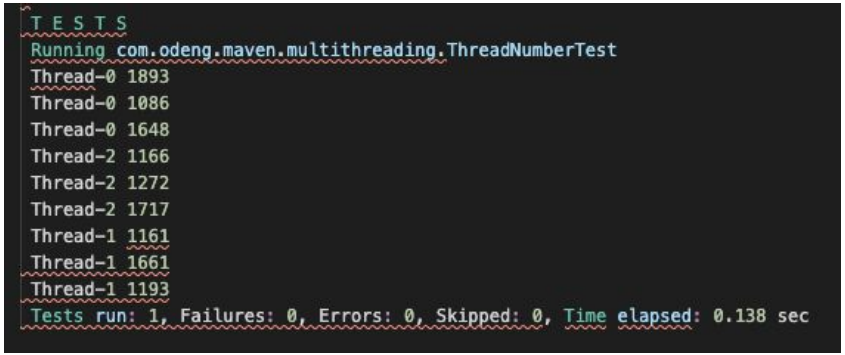
```java
    @Test
    public void testRun() {
        NumberGenerator generator = new NumberGenerator(1000, 2000);
        for (int i = 0; i < 3; i++) {
            new ThreadNumber(generator).start();
        }
    }
```

Script test di atas yaitu dengan membuat object dari class NumberGenerator dan parameter batas bawah 1000 serta batas atasnya 2000.

Skenarionya adalah membuat 3 thread menggunakan perulangan kemudian masing-masing langsung dijalankan ketika dibuat dengan memanggil method start().

```
T E S T S
Running com.odeng.maven.multithreading.ThreadNumberTest
Thread-2 1049
Thread-0 1523
Thread-2 1234
Thread-1 1494
Thread-2 1371
Thread-0 1025
Thread-1 1880
Thread-1 1585
Thread-0 1945
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.152 sec
```

Terlihat bahwa output dari script test antara thread-0, thread-1, dan thread-2 masih tidak teratur, ini menandakan bahwa thread berjalan paralel tidak saling tunggu. Selanjutnya silakan comment pada method callGenerator() dibuka agar mengaktifkan synchronized, jangan lupa dijalankan kembali script testnya.



Ouput di atas adalah hasil penerapan synchronized, terlihat Thread-0 menampilkan 3 random number kemudian disusul dengan Thread-2, dan yang terakhir adalah Thread-1.

"Walaupun menggunakan thread akan jauh lebih efisien dalam melakukan pengolahan atau pemrosesan, hati-hati juga ketika menggunakannya karena untuk melakukan control terhadap thread tidak mudah."