Thread Programming di Java

Thread merupakan kemampuan yang disediakan oleh Java untuk membuat aplikasi yang tangguh, karena thread dalam program memiliki fungsi dan tugas tersendiri. Dengan adanya thread, dapat membuat program yang lebih efisien dalam hal kecepatan maupun penggunaan sumber daya, karena kita dapat membagi proses dalam aplikasi kita pada waktu yang sama. Thread umumnya digunakan untuk pemrograman multitasking, networking, yang melibatkan pengaksesan ke sumber daya secara konkuren.

- Sebuah thread memungkinkan untuk memiliki beberapa state:
  1. Running
     Sebuah thread yang pada saat ini sedang dieksekusi dan didalam control dari CPU.
  2. Ready to run
     Thread yang sudah siap untuk dieksekusi, tetapi masih belum ada kesempatan untuk melakukannya.
  3. Resumed
     Setelah sebelumnya di block atau diberhentikan sementara, state ini kemudian siap untuk dijalankan.
  4. Suspended
     Sebuah thread yang berhenti sementara, dan kemudian memperbolehkan CPU untuk menjalankan thread lain bekerja.
  5. Blocked
     Sebuah thread yang di-block merupakan sebuah thread yang tidak mampu berjalan, karena ia akan menunggu sebuah resource tersedia atau sebuah event terjadi.
- Thread memiliki delapan constructor. Marilah kita lihat bersama beberapa constructor :

Thread Constructors
`Thread()`
Membuat sebuah object *Thread* yang baru.
`Thread(String name)`
Membuat sebuah object thread dengan memberikan penamaan yang spesifik.
`Thread(Runnable target)`
Membuat sebuah object *Thread* yang baru berdasar pada object *Runnable*. Target menyatakan sebuah object dimana method *run* dipanggil.
`Thread(Runnable target, String name)`
Membuat sebuah object *Thread* yang baru dengan nama yang spesifik dan berdasarkan pada object *Runnable*.

Tabel 1.2.1: Constructor dari Thread

- Class *Thread* juga menyediakan beberapa constants sebagai nilai prioritas. Tabel berikut ini adalah rangkuman dari class *Thread*.

Thread Constants
`public final static int MAX_PRIORITY`
Nilai prioritas maksimum, 10
`public final static int MIN_PRIORITY`
Nilai prioritas minimum, 1.
`public final static int NORM_PRIORITY`
Nilai default prioritas, 5.

Tabel 1.2.2:Konstanta dalam Thread

- Method- method inilah yang disediakan dalam class *Thread*.

Thread Methods
`public static Thread currentThread()`
Mengembalikan sebuah reference kepada thread yang sedang berjalan.
`public final String getName()`
Mengembalikan nama dari thread.
`public final void setName(String name)`
Mengulang pemberian nama thread sesuai dengan argument *name*. Hal ini dapat menyebabkan *SecurityException*.
`public final int getPriority()`
Mengembalikan nilai prioritas yang telah diberikan kepada thread tersebut.
`public final boolean isAlive()`
Menunjukkan bahwa thread tersebut sedang berjalan atau tidak.
`public final void join([long millis, [int nanos]])`
Sebuah overloading method. Sebuah thread yang sedang berjalan, harus menunggu sampai thread tersebut selesai (jika tidak ada parameter-parameter spesifik), atau sampai waktu yang telah ditentukan habis.
`public static void sleep(long millis)`
Menunda thread dalam jangka waktu milis. Hal ini dapat menyebabkan *InterruptedException*.
`public void run()`
Eksekusi thread dimulai dari method ini.

Thread Methods
`public void start()`
Menyebabkan eksekusi dari thread berlangsung dengan cara memanggil method run.

Tabel 1.2.3: Method-method dari Thread

- Ada dua cara yang bisa digunakan dalam membuat sebuah thread, yaitu :
  1. Membuat subclass dari thread
     Untuk menjalankan thread, dapat dilakukan dengan memanggil method start(). Saat start() dijalankan, maka sebenarnya method run() dari class akan dijalankan. Jadi untuk membuat thread, harus mendefinisikan method run() pada definisi class. Konstruktor dari cara ini adalah :

     ClassThread namavar = new ClassThread();
     Namavar.start();

Atau dapat juga langsung dengan cara:

New ClassThread().start();

2. Mengimplementasikan interface Runnable

Cara ini merupakan cara yang paling sederhana dalam membuat thread. Runnable merupakan unit abstrak, yaitu kelas yang mengimplementasikan interface ini hanya cukup mengimplementasikan fungsi run(). Dalam mengimplementasi fungsi run(), kita akan mendefinisikan instruksi yang membangun sebuah thread. Konstruktor dari cara ini adalah :

ObjekRunnable objek = new ObjekRunnable();

Thread namavar = new Thread(Objek Runnable);

Atau dengan cara singkat seperti :

New Thread(new ObjekRunnable());

A. Daemon Dan User Thread

Ada dua Macam thread dalam Java, yaitu ***daemon*** dan ***user*** thread. ***Daemon*** thread merupakan thread yang siklus hidupnya tergantung pada thread utama atau induk, sehingga apabila thread induk berakhir, maka otomatis thread-thread daemon juga ikut berakhir. Sedangkan ***user*** thread memiliki sifat berbeda, dimana apabila thread utama sudah selesai, maka user thread akan terus dijalankan.

B. Sleep

Mengatur thread untuk menghentikan prosesnya sejenak dan member kesempatan pada thread atau proses lain. Sleep dilakukan dengan cara memanggil method :

Sleep(long waktu);

Waktu untuk method ini merupakan tipe long dalam milisekon.

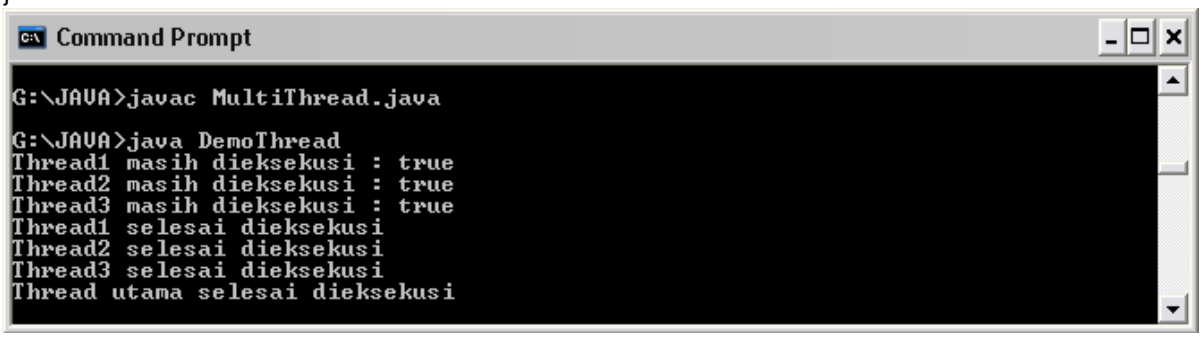
C. Interrupt

Apabila menginginkan suatu thread untuk menghentikan proses, maka perlu memanggil method interrupt. Interrupt digunakan untuk memberi signal pada thread untuk menghentikan prosesnya.

Contoh **Thread.java**

```
class ThreadBaru extends Thread {
   public ThreadBaru(String id) {
       super(id);
       start(); //Mulai eksekusi thread baru
   }
   public void run() {
       for(int i=0;i<5;i++){
         try{
             Thread.sleep(100);
         }catch(InterruptedException e) {}
       }
   }
}
class DemoThread {
   public static void main(String[] args) {
       ThreadBaru thread1 = new ThreadBaru("Thread1");
       ThreadBaru thread2 = new ThreadBaru("Thread2");
       ThreadBaru thread3 = new ThreadBaru("Thread3");
       System.out.println("Thread1 masih dieksekusi : " + thread1.isAlive());
       System.out.println("Thread2 masih dieksekusi : " + thread2.isAlive());
       System.out.println("Thread3 masih dieksekusi : " + thread3.isAlive());
       //tunggu hingga semua child thread selesai dieksekusi
```

```
        try{
            thread1.join();
            System.out.println("Thread1 selesai dieksekusi");
            thread2.join();
            System.out.println("Thread2 selesai dieksekusi");
            thread3.join();
            System.out.println("Thread3 selesai dieksekusi");
        }catch(InterruptedException e) {
            System.out.println("Thread utama diinterupsi " + e);
        }
        System.out.println("Thread utama selesai dieksekusi");
    }
}
```



D. Synchronized

Sinkronisasi adalah method atau blok yang memiliki tambahan keyword synchronized, sehingga apabila dijalankan maka hanya satu thread pada suatu waktu yang dapat menjalankan method atau blok program. Thread lain akan menunggu thread yang sedang mengeksekusi method ini hingga selesai. Mekanisme sinkronisasi penting apabila terjadi pembagian sumber daya maupun data di antara thread-thread. Sinkronisasi juga melakukan penguncian pada sumber daya atau data yang sedang diproses.

Contoh **ThreadSinkronisasi.java**

```
class TestSinkronisasi {
    private java.util.Random random = new java.util.Random();
    public void callMe(String data) {
        System.out.print("[");
        try{
            Thread.sleep(random.nextInt(200));
        }catch(InterruptedException e) {
            e.printStackTrace();
        }
        System.out.print(data);
        try{
            Thread.sleep(random.nextInt(200));
        }catch(InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("]");
    }
}
```

```
class ThreadBaru extends Thread {
    private String data;
    private TestSinkronisasi obj;
    public ThreadBaru(TestSinkronisasi obj,String data) {
        this.obj = obj;
        this.data = data;
        start();
    }
    public void run() {
        obj.callMe(data);
    }
}
class DemoThread {
    public static void main(String[] args) {
        TestSinkronisasi obj = new TestSinkronisasi();
        ThreadBaru thread1 = new ThreadBaru(obj,"Superman");
        ThreadBaru thread2 = new ThreadBaru(obj,"Batman");
        ThreadBaru thread3 = new ThreadBaru(obj,"Spiderman");
        //tunggu hingga semua child thread selesai dieksekusi
        try{
            thread1.join();
            thread2.join();
            thread3.join();
        }catch(InterruptedException e) {
            System.out.println("Thread utama diinterupsi " + e);
        }
    }
}
```