

BAB VIII

Thread

A. Definisi Thread dan Multithreading

Thread di Java adalah unit dasar dari eksekusi yang memungkinkan sebuah program untuk melakukan beberapa tugas secara bersamaan (concurrently)..

Multithreading adalah kemampuan sebuah program untuk menjalankan beberapa tugas secara bersamaan pada saat yang sama, dengan mengalokasikan beberapa thread pada sebuah proses. Dalam pemrograman, multithreading memungkinkan sebuah aplikasi untuk melakukan beberapa tugas secara bersamaan dalam satu waktu, sehingga dapat meningkatkan kinerja dan efisiensi aplikasi tersebut.

Sebagai contoh, dalam sebuah aplikasi pengolahan data, sebuah thread dapat bertanggung jawab untuk membaca data dari sumber, sementara thread lain dapat bertanggung jawab untuk mengolah data tersebut dan thread lainnya lagi dapat bertanggung jawab untuk menampilkan hasilnya. Dengan menggunakan multithreading, ketiga tugas tersebut dapat dijalankan secara bersamaan dalam satu waktu, sehingga waktu yang dibutuhkan untuk menyelesaikan tugas tersebut dapat lebih efisien dan cepat.


B. Membuat Thread

Di Java, terdapat dua cara untuk membuat thread yaitu dengan meng-extend kelas Thread atau dengan mengimplementasikan interface Runnable. Berikut adalah penjelasan mengenai kedua cara tersebut:

1. Meng-extend kelas Thread:
Langkah-langkah untuk membuat thread dengan cara ini adalah sebagai berikut:
 - a. Buat sebuah kelas baru yang meng-extend kelas Thread.

- b. Override method `run()` untuk menentukan kode program yang akan dieksekusi oleh thread.
- c. Buat sebuah objek dari kelas tersebut dan panggil method `start()` untuk memulai thread.

Berikut adalah contoh kode program untuk membuat thread dengan cara ini:



```
1 public class MyThread extends Thread {  
2     public void run() {  
3         System.out.println("Hello from MyThread!");  
4     }  
5  
6     public static void main(String args[]) {  
7         MyThread thread = new MyThread();  
8         thread.start();  
9     }  
10 }
```

2. Mengimplementasikan interface Runnable

Langkah-langkah untuk membuat thread dengan cara ini adalah sebagai berikut:

- a. Buat sebuah kelas baru yang mengimplementasikan interface Runnable.
- b. Override method `run()` untuk menentukan kode program yang akan dieksekusi oleh thread.
- c. Buat sebuah objek dari kelas tersebut dan lewatkan objek tersebut ke dalam konstruktor kelas Thread.
- d. Panggil method `start()` pada objek thread untuk memulai thread.

Berikut adalah contoh kode program untuk membuat thread dengan cara ini :



```
1 public class MyRunnable implements Runnable {
2     public void run() {
3         System.out.println("Hello from MyRunnable!");
4     }
5
6     public static void main(String args[]) {
7         MyRunnable runnable = new MyRunnable();
8         Thread thread = new Thread(runnable);
9         thread.start();
10    }
11 }
```

C. Menjalankan dan Menghentikan Thread di Java

1. Menjalankan thread di java

Untuk menjalankan thread yang sudah dibuat, Anda dapat menggunakan method start() pada objek thread tersebut. Method start() akan menjalankan thread secara asynchronous, sehingga thread tersebut dapat berjalan secara bersamaan dengan thread utama. Berikut adalah contoh cara menjalankan thread dengan menggunakan method start():



```
1 class SampleThread extends Thread {
2     public void run() {
3         System.out.println("Hello from MyThread ");
4     }
5
6 }
7
8 public class Main {
9     public static void main(String args[]) {
10         SampleThread thread = new SampleThread();
11         thread.start();
12
13         try {
14             Thread.sleep(5000);
15         } catch (InterruptedException e) {
16             e.printStackTrace();
17         }
18     }
19 }
20
```

2. Menghentikan thread di java

Untuk menghentikan thread yang sedang berjalan, Anda dapat menggunakan method `stop()`. Namun, method `stop()` tidak direkomendasikan karena dapat menyebabkan kesalahan pada data atau mengganggu kestabilan aplikasi. Sebagai gantinya, Anda dapat menggunakan sebuah flag boolean untuk memberikan tanda pada thread untuk berhenti secara sukarela. Berikut adalah contoh cara menghentikan thread dengan menggunakan flag boolean:

```
1 class SampleThread extends Thread {
2     private boolean running = true;
3
4     public void run() {
5         while (running) {
6             System.out.println("Hello from MyThread");
7         }
8     }
9
10    public void stopThread() {
11        running = false; // memberikan tanda untuk menghentikan thread
12    }
13 }
14
15 public class Main {
16     public static void main(String args[]) {
17         SampleThread thread = new SampleThread();
18         thread.start();
19
20         // berhenti setelah 5 detik
21         try {
22             Thread.sleep(5000);
23         } catch (InterruptedException e) {
24             e.printStackTrace();
25         }
26
27         thread.stopThread();
28     }
29 }
30
```

Beberapa method thread yang sering digunakan :

- 1) `start()`: method yang digunakan untuk memulai thread dan mengeksekusi method `run()`. Jika method `start()` dipanggil pada objek thread yang sama lebih dari satu kali, maka akan terjadi `IllegalThreadStateException`.
- 2) `run()`: method yang digunakan untuk menjalankan kode program pada thread. Ketika sebuah thread dimulai, kode program yang dieksekusi ada pada method `run()`.

- 3) `sleep(long millis)`: method yang digunakan untuk memperlambat atau menghentikan sementara thread saat berjalan selama milis milidetik. Pemanggilan method ini membutuhkan penanganan `InterruptedException` karena dapat terjadi gangguan selama thread sedang tertidur.
- 4) `join()`: method yang digunakan untuk menunggu hingga thread yang ditunjukkan oleh objek thread selesai dieksekusi sebelum melanjutkan eksekusi program pada thread yang sedang berjalan. Method ini juga dapat menerima argumen `long millis` untuk menentukan waktu maksimal thread yang ditunggu, setelah itu program akan tetap melanjutkan eksekusi meskipun thread belum selesai dieksekusi.

Sebenarnya masih ada beberapa method lain seperti `interrupt()`, `yield()`, dll. Jadi, silahkan explore sendiri di <https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html>

D. Sinkronisasi Thread

1. Race condition

Race condition adalah situasi di mana beberapa thread mengakses dan memodifikasi sumber daya bersamaan, yang mengakibatkan hasil yang tidak konsisten atau tidak terduga. Race condition dapat terjadi ketika ada beberapa thread yang bekerja pada kode yang sama dan menggunakan variabel bersama. Hal ini dapat mengakibatkan data yang salah atau tidak terduga.

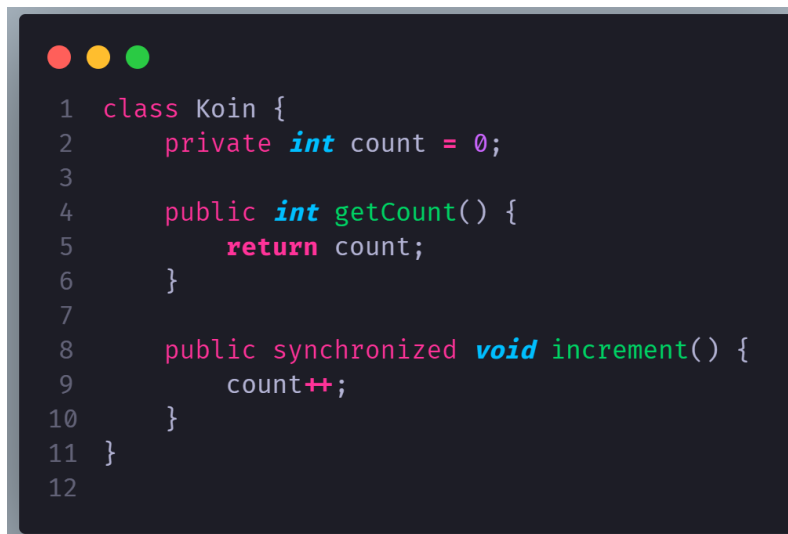
2. Deadlock

Deadlock terjadi ketika dua atau lebih thread saling menunggu untuk sumber daya yang sama. Dalam situasi ini, setiap thread terus menunggu sumber daya yang dibutuhkan oleh thread lainnya, sehingga tidak ada thread yang dapat melanjutkan eksekusi kode. Deadlock dapat terjadi ketika terdapat dua atau lebih thread yang saling berkomunikasi dengan sumber daya yang sama dan kedua thread tersebut memegang sumber daya yang diperlukan oleh thread lainnya secara bersamaan. Akibatnya, kedua thread tersebut terus menunggu dan tidak ada yang dapat melanjutkan eksekusi.

3. Sinkronisasi thread

Sinkronisasi thread dalam Java adalah teknik yang digunakan untuk menghindari situasi di mana dua atau lebih thread saling bersaing untuk mengakses sumber daya bersama, seperti variabel atau objek, yang dapat menyebabkan *deadlock* atau *race condition*. Dalam Java, sinkronisasi thread dapat dicapai dengan menggunakan kata kunci `synchronized`.

Sebagai contoh, perhatikan class koin berikut :



```
1  class Koin {  
2      private int count = 0;  
3  
4      public int getCount() {  
5          return count;  
6      }  
7  
8      public synchronized void increment() {  
9          count++;  
10     }  
11 }  
12
```

Pada contoh di atas, method `increment()` telah di-synchronize menggunakan kata kunci `synchronized`. Ketika dua thread atau lebih mencoba memanggil method `increment()` pada waktu yang sama, hanya satu thread yang dapat mengakses method tersebut pada satu waktu.

Semisal terdapat kode main seperti berikut:

```

1  public class Main {
2      public static void main(String[] args) {
3          Koin koin = new Koin();
4
5
6          Thread thread1 = new Thread(() → {
7              for (int i = 0; i < 10; i++) {
8                  koin.increment();
9              }
10         });
11
12
13         Thread thread2 = new Thread(() → {
14             for (int i = 0; i < 10; i++) {
15                 koin.increment();
16             }
17         });
18
19         thread1.start();
20         thread2.start();
21
22         try {
23             thread1.join();
24             thread2.join();
25         } catch (InterruptedException e) {
26             e.printStackTrace();
27         }
28
29         System.out.println("Count: " + koin.getCount());
30     }
31 }

```

dapat dilihat bahwa thread1, dan thread2 sama-sama mengakses method increment di objek koin yang sama namun karena method increment di class Koin telah diberikan keyword synchronized jadi program dapat terhindar dari Deadlock ataupun race condition.

Tapi sebenarnya hal ini juga dipengaruhi dari kecepatan Processor, jumlah thread yang jalan bersamaan dll, jadi bisa jadi meski tanpa synchronized sekalipun programnya dapat tetap jalan.

E. Menggunakan Executor

Executor di Java adalah sebuah mekanisme untuk menjalankan sebuah tugas secara asynchronous. Executor memungkinkan pengguna untuk memisahkan logika tugas dari cara pelaksanaannya,

sehingga pengguna dapat fokus pada pemrograman aplikasi dan menghindari kompleksitas dalam manajemen thread.

Dalam penggunaannya, Executor dapat digunakan untuk mengelola thread secara lebih efisien dan menghindari masalah yang mungkin terjadi ketika penggunaan thread diimplementasikan secara manual. Hal ini karena Executor menyediakan thread pool yang dapat digunakan untuk mengelola thread secara lebih efektif, sehingga memungkinkan pengguna untuk mengatur jumlah thread yang digunakan dan menghindari kesalahan dalam manajemen thread seperti deadlock dan race condition.

```
1 import java.util.concurrent.ExecutorService;
2 import java.util.concurrent.Executors;
3
4 public class Main {
5     public static void main(String[] args) {
6         // membuat executor dengan fixed thread pool
7         ExecutorService executor = Executors.newFixedThreadPool(3);
8
9         // menjalankan tiga tugas secara bersamaan
10        for (int i = 0; i < 3; i++) {
11            executor.execute(new Runnable() {
12                public void run() {
13                    System.out.println("Tugas sedang dijalankan oleh thread: "
14                        + Thread.currentThread().getName());
15
16                    try {
17                        Thread.sleep(1000);
18                    } catch (InterruptedException e) {
19                        e.printStackTrace();
20                    }
21
22                    System.out.println("Tugas selesai dijalankan oleh thread: "
23                        + Thread.currentThread().getName());
24                }
25            });
26        }
27
28        // menutup executor setelah selesai menjalankan tugas
29        executor.shutdown();
30    }
31 }
```

Pada contoh di atas, digunakan ExecutorService dengan newFixedThreadPool(3) untuk membuat thread pool dengan tiga thread yang tetap. Kemudian, tiga tugas dijalankan secara bersamaan dengan executor.execute(). Setelah selesai, executor dihentikan dengan executor.shutdown().

Method shutdown() akan memberikan sinyal ke executor untuk menghentikan menerima tugas baru, namun executor akan menyelesaikan semua tugas yang sudah berjalan sebelum menutup thread. Setelah semua tugas selesai, executor akan menutup semua thread yang digunakan.