MULTITHREADING DAN SINKRONISASI

OBJEKTIF :

1. Mahasiswa Mampu Memahami *Multithreading* pada Java.
2. Mahasiswa Mampu Menggunakan *Software* IntelliJ IDEA untuk Membuat Program Penciptaan *Thread*, Sinkronisasi, dan Komunikasi Antar *Thread*.

PENDAHULUAN

Java adalah bahasa pemrograman *multithreaded* yang berarti kita dapat mengembangkan program *multithreaded* menggunakan Java. Program *multithreaded* berisi dua atau lebih bagian yang dapat berjalan secara bersamaan. Masing-masing bagian dapat menangani tugas yang berbeda pada saat yang bersamaan dengan memanfaatkan secara optimal sumber daya yang tersedia khususnya ketika komputer memiliki banyak CPU.

Menurut definisi, *multitasking* adalah ketika banyak proses berbagi sumber daya pemrosesan yang sama seperti CPU. *Multithreading* memperluas gagasan *multitasking* ke dalam aplikasi dimana *programmer* dapat membagi operasi tertentu dalam satu aplikasi ke dalam masing-masing *thread*. Setiap *thread* dapat berjalan secara paralel. Sistem Operasi membagi waktu pemrosesan tidak hanya di antara aplikasi yang berbeda, tetapi juga di antara setiap *thread* dalam suatu aplikasi.

12.1 KEMAMPUAN MULTITHREADING

Multithreading memungkinkan *programmer* untuk menulis dengan cara di mana beberapa aktivitas dapat dilanjutkan secara bersamaan dalam program yang sama.

12.1.1 THREAD DAN PROSES

Program *multithread* dapat berisi bagian satu, dua atau lebih bagian yang berjalan secara konkuren. Masing-masing bagian disebut *thread*. Masing-masing *thread* mendefinisikan satu alur eksekusi sendiri. *Multithreading* merupakan bentuk special *multitasking*. Pada multitasking, proses merupakan unit kepemilikan dan penjadwalan (aktivitas).

Pada bahasa Java, proses ini masih dapat mempunyai banyak aktivitas independen, sehingga:

1. Thread adalah abstraksi dari unit aktivitas (penjadwalan).

2. Proses adalah unit sumber daya.

Proses adalah lingkungan eksekusi, unit manajemen sumber daya dimana *thread-thread* dapat mengaksesnya.

Pada lingkungan *multithreading, thread* merupakan unit penjadwalan terkecil. Artinya satu program dapat mempunyai lebih dari satu *thread* yang berjalan secara konkuren. Contohnya, editor teks, pada saat bersamaan:

- Melakukan format terhadap teks, serta
- Menunggu interaksi pemakai.

Semua *thread* mengimplementasikan *init (), start (), stop ()* dan *run(). Thread* ini dapat didefinisikan menggunakan metode tersebut secara eksplisit di kode, atau menggunakan *default* di kelas *Thread*.

1. Method *init()* dipanggil saat *thread* dimulai dan merupakan lokasi yang bagus untuk menempatkan kode inisialisasi.
2. Method *start()* dipanggil setiap kali *thread* dimulai.
3. Method *stop()* untuk menghentikan *thread* dan biasanya berisi kode yang akan mengakhiri badan *thread*.
4. Method *run(),* dimulai oleh *start()* dan biasanya berisi badan kode *thread*.

Thread sering disebut *lightweight process* (LWP), yaitu unit dasar utilisasi pemroses dan berisi *program counter, register set* dan *stack space*. *Thread-thread* di satu proses saling berbagi (memakai bersama) bagian kode, data dan sumber daya sistem operasi seperti *file* dan *signal*. Pemakaian ekstensif menyebabkan alih pemroses antara *thread-thread* di satu proses tidak mahal dibanding alih konteks antar proses. Meski alih thread masih memerlukan alih himpunan register, namun tidak ada melibatkan manajemen memori.

Multithreading merupakan upaya meningkatkan kinerja sistem komputer, disebabkan :

1. Penciptaan *thread* baru lebih cepat dibanding penciptaan proses baru,
2. Terminasi thread lebih cepat dibanding pengakhiran proses.
3. Alih ke *thread* lain di satu proses lebih cepat dibanding alih dari satu proses ke proses lain karena tanpa adanya *context switching.*
4. *Thread-thread* di satu proses dapat berbagi kode, data dan sumber daya lain secara nyaman dan efisien dibanding proses-proses terpisah.

12.1.2 KEGUNAAN THREAD

Thread adalah *class library* yang mengatur setiap aliran eksekusi pada suatu program di bahasa pemrograman Java. Pada dasarnya suatu program , memiliki setidaknya minimal satu single *Thread*. Akan menjadi masalah, ketika suatu program menerima banyak *action* atau eksekusi di waktu yang bersamaan, lalu program tidak dapat mengatasinya. *Thread* akan mengatur tentang, kapan suatu eksekusi program di jalankan, di hentikan sementara, dimatikan (diselesaikan) atau dijalankan bersamaan. Untuk menjalankan sebuah *thread* kita bisa menggunakan *keyword extends* (mewariskan) pada *class library Thread*, atau menggunakan *keyword implements* (Mengimplementasi) dari *interface Runnable* .

Manfaat utama banyak *thread* di satu proses adalah memaksimumkan tingkat konkurensi antara operasi-operasi yang terkait erat. Aplikasi jauh lebih efisien dikerjakan sebagai sekumpulan *thread* dibanding sebagai kumpulan proses.

Kegunaan lingkungan *multithread* di Java, satu *thread* dapat berhenti tanpa menghentikan bagian-bagian program yang lain sehingga cara ini lebih menghemat CPU.

12.2 PENCIPTAAN THREAD

Thread dapat diciptakan dengan beberapa cara, yaitu:

1. Memperluas kelas *Thread (extends Thread)*
2. Mengimplementasikan *interface Runnable*.
3. Mengimplementasikan *interface Callable*.

Interface Runnable digunakan untuk mengidentifikasikan kode yang dieksekusi merupakan bagian *Thread* aktif. *Interface* ini hanya berisi satu metode run() yang dieksekusi saat *Thread* diaktivasi. *Runnable interface* diimplementasikan oleh kelas *Thread* dan kelas-kelas lain yang mendukung eksekusi secara *multithread*. Sebagai langkah pertama, yaitu perlu mengimplementasikan metode run() yang disediakan oleh *interface Runnable*. Berikut ini adalah sintaks sederhana dari metode run() :

```
public void run( )
```

Kemudian kita dapat melakukan instansiasi objek *Thread* menggunakan konstruktor berikut :

```
Thread(Runnable threadObj, String threadName);
```

threadObj adalah turunan dari kelas yang mengimplementasikan *interface Runnable* dan *threadName* adalah nama yang diberikan ke *thread* baru. Setelah objek *Thread* dibuat, dapat memulainya dengan memanggil metode start(), yang mengeksekusi metode run(). Berikut ini adalah sintaks sederhana metode start() :

```
void start();
```

Berikut adalah contoh penciptaan *thread* dengan memperluas kelas *Thread*:

```
class ContohThread extends Thread;
```

Berikut adalah contoh penciptaan *thread* dengan memperluas kelas *Thread*:

```
class ContohThread implements Runnable;
```

Kelas *Thread* digunakan untuk membangun dan mengakses eksekusi *Thread* individu yang dieksekusi sebagai bagian program *multithread*. Setiap *thread* di Java memiliki prioritas yang membantu sistem operasi menentukan penjadwalan *thread*, yaitu *MIN PRIORITY, MAX PRIORITY*, dan *NORM PRIORITY.*

12.3 SINKRONISASI

Ketika memulai dua *thread* atau lebih dalam suatu program, memungkinkan adanya situasi beberapa *thread* mencoba mengakses sumber yang sama dan akhirnya thread tersebut dapat menghasilkan hasil yang tidak terduga karena masalah konkurensi. Misalnya, jika beberapa *thread* mencoba menulis dalam file yang sama maka thread dapat merusak data karena salah satu *thread* dapat menimpa data atau sementara satu *thread* lain membuka file yang sama, dan thread lain mungkin akan menutup file yang sama pada saat yang bersamaan.

Jadi, diperlukan sinkronisasi untuk menyinkronkan aksi beberapa *thread* dan memastikan bahwa hanya satu *thread* yang dapat mengakses sumber daya pada titik waktu tertentu. Sinkronisasi ini diimplementasikan menggunakan konsep yang disebut monitor. Setiap objek di Java dikaitkan dengan monitor, yang dapat dikunci atau dibuka oleh *thread*. Hanya satu thread pada satu waktu yang dapat menahan kunci pada monitor.

Kata kunci *synchronized* digunakan untuk mengunci objek saat mengeksekusi suatu blok kode. Tidak ada *Thread* lain yang dapat mengubah objek yang dispesifikasikan selama blok kode dieksekusi. Berikut ini adalah bentuk umum dari pernyataan yang disinkronkan :

```
synchronized(objectIdentifier) {
}
```

Di sini, *ObjectIdentifier* adalah referensi ke objek yang kuncinya terkait dengan monitor yang diwakili oleh pernyataan yang disinkronkan. Sekarang kita akan melihat dua contoh, di mana kita akan mencetak penghitung menggunakan dua *thread* yang berbeda. Ketika *thread* tidak disinkronkan, akan mencetak nilai penghitung yang tidak berurutan. Tetapi ketika kita mencetak penghitung dengan menempatkan blok di dalam *synchronized()*, maka akan mencetak nilai penghitung yang sangat banyak dan berurutan untuk kedua *thread*.

12.4 KOMUNIKASI

Komunikasi antar *thread* penting ketika *programmer* ingin mengembangkan aplikasi dimana terdapat dua atau lebih *thread* untuk bertukar informasi. Ada tiga metode sederhana yang memungkinkan komunikasi antar *thread*. Berikut contoh metode yang dapat digunakan :

No	Method	Deskripsi
1	public void wait()	Menyebabkan *thread* saat ini menunggu sampai thread lainnya memanggil metode notify ().
2	public void notify()	Membangunkan satu *thread* yang menunggu di monitor pada objek yang sama.
3	public void notifyAll()	Membangunkan semua thread yang menunggu di monitor pada objek yang sama.

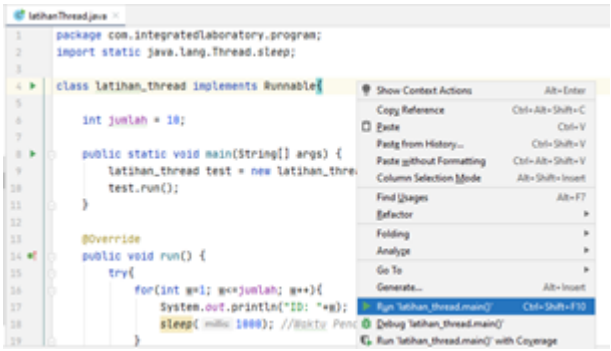
Metode-metode ini diimplementasikan sebagai metode akhir dalam *Object*, sehingga metode tersebut tersedia di semua kelas. Ketiga metode hanya dapat dipanggil dari dalam konteks yang telah disinkronkan .

Berikut contoh pembuatan thread dengan mengimplementasikan *interface Runnable :*

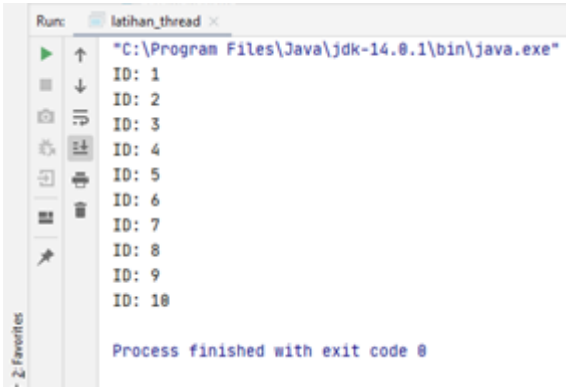
```
package com.integratedlaboratory.program;
import static java.lang.Thread.sleep;

class latihan_thread implements Runnable {
    int jumlah = 10;

    public static void main(String[] args) {
        latihan_thread test = new latihan_thread();
        test.run();
    }
    public void run() {
        try{
            for(int w=1; w<=jumlah; w++){
                System.out.println("ID: "+w);
                sleep(1300); //Waktu Pending
            }
        }catch(InterruptedException ex){
            ex.printStackTrace();
        }
    }
}
```

Perintah :

Tekan tombol Ctrl+Shift+F10 untuk melakukan Run pada IntelliJ IDEA atau dengan melakukan *klik* kanan pada *file* java seperti berikut:



Output :



Penjelasan :

Pada contoh kode di atas kita akan mencoba menjalankan suatu perintah *loop*, dengan menggunakan *Thread*. Di dalam *method void run()*, kita akan mengeksekusi *handling try catch{}* . Di dalam blok *try,* kita menggunakan *looping* untuk menampilkan *output text* yang tampil sebanyak 10 kali. Lalu kita menggunakan *method Thread.sleep ()* untuk menjeda atau *mendelay*, setiap tampilan output pada perulangan pertama atau ke 1 sampai terakhir atau ke 10. Lamanya jeda atau *delay*, tergantung seberapa satuan milidetik yang kita tentukan. Dalam contoh kode di atas kita menggunakan 1200 milidetik. Karena kita menggunakan *Thread.sleep*, kita wajib menggunakan *exception* di blok *catch*, dengan menggunakan *InteruptedException* atau *Exception*. Selanjutnya jalankan *method* tersebut, dengan menginstansiasi objek dengan *keyword new*, lalu menjalankan *method run().*

REFERENSI

[1] Hariyanto, Bambang. 2010. Esensi-Esensi Bahasa Pemrograman Java Revisi Ketiga. Bandung: Informatika Bandung.

[2] Tutorialspoint. Java-Multithreading. Diambil dari : https://www.tutorialspoint.com/java/java_multithreading.htm . (1 Agustus 2020)

[3] Fathurrahman. 12 October 2017 . Belajar Mengenal Apa itu Thread pada Program Java. Diambil dari : https://www.okedroid.com/2017/10/belajar-mengenal-apa-itu-thread-pada-program-java.html?m=1 .(3 Agustus 2020)