# BAB 2

# *Class Thread*

## 2.1 Tujuan:

Setelah mempelajari modul ini peserta diharapkan dapat:

- Mengenal Kelas *Thread*
- Menggunakan Kelas *Thread* dalam aplikasi

## 2.2 Mengenal Kelas *Thread*

Salah satu fitur yang tertanam pada *environment Java* yaitu dukungan untuk menggunakan *thread*. *Thread* memungkinkan sebuah aplikasi *Java* untuk melakukan banyak aktifitas/operasi secara simultan (serentak). Dengan menggunakannya secara benar, *thread* memungkinkan *User Interface* sebuah aplikasi tetap responsif saat aplikasi tersebut melakukan operasi yang memakan waktu lama seperti komunikasi *networking* atau perhitungan-perhitungan yang kompleks.

*Thread* adalah unit fundamental dari eksekusi program. Setiap aplikasi minimal memiliki sebuah *thread* untuk menjalankan kode. Aplikasi yang memiliki dua *thread* atau lebih, biasa disebut dengan *multithreaded application*.

Fungsi seperti ini sering kita butuhkan dalam membuat program aplikasi/game nantinya. Misalnya, sebuah *thread* yang bertugas menjalankan operasi penghitungan nilai/skor game, kemudian *thread* yang lainnnya menjalankan operasi pendeteksi tabrakan antara obyek-obyek pada game. Kedua *thread* tersebut berjalan bersamaan dalam melakukan tugasnya masing-masing.

Setiap *thread* memiliki sebuah *konteks* yang berhubungan dengannya. *Konteks* tersebut memuat informasi tentang *thread*, seperti alamat dari instruksi yang sedang dieksekusi dan storage untuk variabel-variabel lokal. *Konteks* tersebut akan terupdate begitu *thread* dieksekusi. *Konteks* juga menyimpan *state* dari *thread*.

*Thread* bisa dalam *state* sebagai berikut:

- *Running*, saat dimana *thread* sedang menjalankan kode.
- *Ready*, saat dimana *thread* siap untuk mengeksekusi kode.

- *Suspended*, saat dimana *thread* sedang menunggu *external event.* Contohnya: menunggu data yang datang dari *device* lain. Begitu data datang dan *event* terjadi (dilakukan), maka *thread* kembali ke *state ready*.
- *Terminated*, saat dimana *thread* selesai mengeksekusi kode.

*Thread* yang dalam keadaan (*state*) *running, ready* atau *suspended*, adalah *thread* yang hidup (*live thread*). Sedangkan *thread* yang berstatus *terminated* adalah thread mati (*dead thread*).

Meskipun sebuah aplikasi boleh memiliki banyak *thread*, namun perlu memperhatikan kemampuan *device* yang ada (*mobile device*). Yang pada umumnya hanya memiliki kemampuan dan jumlah prosesor yang kecil (biasanya hanya mampu satu atau dua saja) untuk melakukan eksekusi kode.

### 2.2.1 Obyek Thread

Thread hanya bisa digunakan dalam sebuah aplikasi tidak bisa berdiri menjadi sebuah obyek tersendiri. Java runtime menghubungkan setiap *live thread* sebagai instance dari kelas *java.lang.Thread*. Kelas inilah yang digunakan untuk menjalankan thread baru serta mengambil dan mengeset prioritas dari thread itu sendiri. Pada J2ME Thread hanya mendukung method-method berikut:

- *activeCount()*
  Mengembalikan nilai sekarang yang aktif pada virtual mesin.
- *currentThread()*
  Akan mengembalikan nilai object uang sekarang.
- *getPriority()*
  Akan mengembalikan Prioritas thread.
- *isAlive()*
  Mengecek aktifnya thread.
- *join()*
  Menuggu hingga thread ini selesai.

- *run()*

  jika thread ini dibangun harus menggunakan implementasi Runnable dan menyantumkan object run maka otomatis object Runable akan memanggil method run.
- *setPriority()*

  Merubah prioritas pada thread ini.
- *sleep()*

  Untuk melaksanakan berhenti sementara dalam betuk milidetik.
- *start()*

  Thread ini mengawali eksekusi.
- *yield()*

  Thread ini untuk menunda dan mengijinkan thread lain untuk mengeksekusi.

### 2.2.2 Menggunakan Thread

Untuk diketahui, ada 2 cara dalam menggunakan class Thread ini:

- **Mendeklarasikan sebuah kelas yang menjadi turunan dari kelas Thread.**

  Sub kelas yang dibuat harus mengoverride method run dari kelas Thread, baru kemudian kita bisa membuat sebuah instance dari kelas tersebut untuk dialokasikan dan dijalankan. Berikut contoh *Thread* yang akan melakukan penghitungan scoring.

  *Pendeklarasian kelas:*

```
class ThreadSkoring extends Thread {
    ThreadSkoring() {
    }

    public void run() {
        // operasi menghitung skor game
         . . .
    }
}
```

  *Membuat obyek ThreadSkoring dan menjalankannya:*

```
ThreadSkoring p = new ThreadSkoring();
p.start();
```

- **Mendeklarasikan sebuah kelas yang mengimplementasikan interface Runnable.**

  Kelas baru yang dibuat harus mengoverride *method run* dari kelas *Runnable*, baru kemudian sebuah *instance* dari kelas tersebut bisa dialokasikan dan dijalankan melalui argumen saat membuat obyek baru dari kelas *Thread*.

  *Pendeklarasian kelas:*

```
class RunSkoring extends Thread {
    RunSkoring() {
    }

    public void run() {
        // operasi menghitung skor game
         . . .
    }
}
```

  *Membuat obyek RunSkoring dan menjalankannya:*

```
RunSkoring p = new RunSkoring();
new Thread(p).start();
```

### 2.2.3 Menghentikan Thread

Setelah melihat daftar method-method *Thread* yang didukung pada J2ME di atas, tampak tidak terdapat method *stop()* dan *interrupt()*. Keduanya didukung pada J2SE, namun karena konfigurasi J2ME yang minim, keduanya tidak didukung. Method *stop()* telah dihilangkan karena *inherently unreliable* dan tidak bisa diimplementasikan pada semua platform dengan aman dan konsisten. Sedangkan method *interrupt()* sudah diperkenalkan lagi pada CLDC versi 1.1 dan sepertinya akan segera diperkenalkan juga pada CDC revisi berikutnya.

Oleh karena itu, agar *thread* yang kita buat dan jalankan bisa berhenti, maka kita perlu memberikan *flag* agar tiap-tiap *live thread* bisa menghentikan dirinya sendiri. *Flag* ini sangat penting didefinisikan untuk setiap *thread* pada sebuah aplikasi agar bisa menghentikan dirinya sendiri. Cara paling mudah ialah agar *thread* secara periodik memeriksa sebuah variabel *flag* untuk menentukan dirinya berjalan terus atau sudah harus berhenti. Misalnya dengan mendefinisikan sebuah kelas *MyThread* sebagai berikut:

```
public class MyThread implements Runnable {
    private boolean selesai = false;

    public void run(){
      while( !selesai ){
          // lakukan operasi yang diperlukan
      }
    }

    public void selesai(){
      selesai = true;
    }
}
```

Kelas di atas akan terus melakukan operasi pada method *run()* sampai kita memanggil method *selesai()* yang sudah kita definisikan di dalam kelas itu.