# MULTITHREADING

## Tujuan

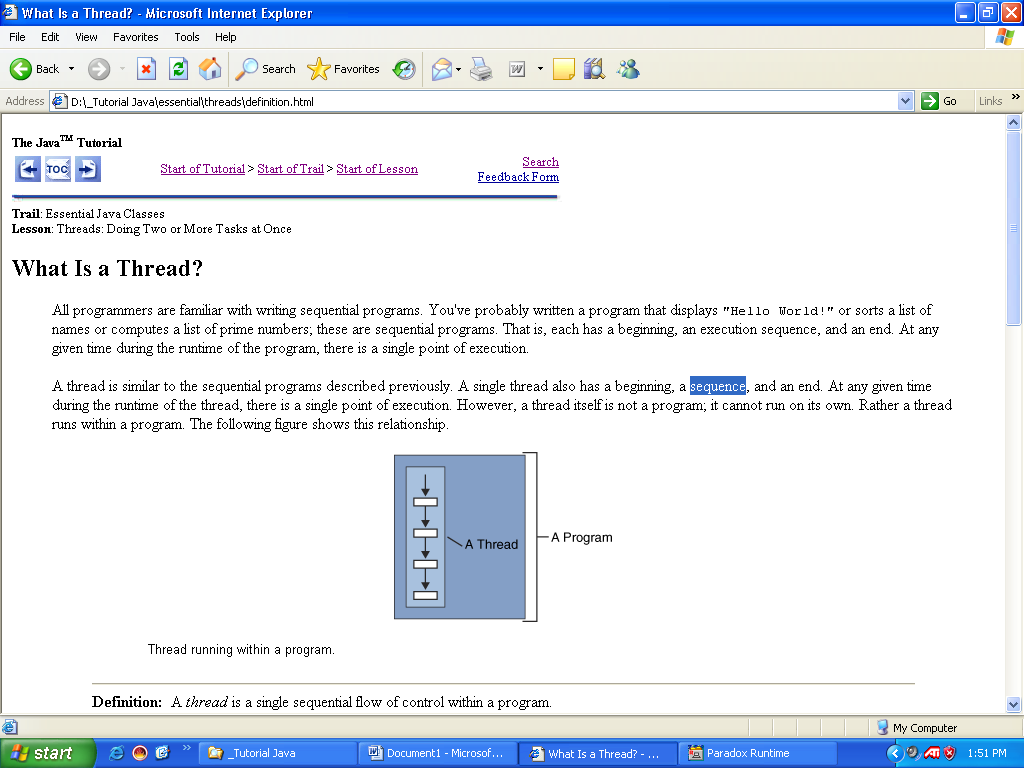
Setelah mengikuti praktikum ini mahasiswa diharapkan dapat :

1. **Mengenal** konsep-konsep multithreading dalam java dan dapat **mengimplementasikannya** dalam kasus-kasus pemrograman.
2. **Mengerti** dan **memahami sinkronisasi** pada multithreading dan dapat **mengimplementasikannya** dalam kasus-kasus pemrograman.

## Pendahuluan

Para programmer pada umumnya lebih mengenal membuat **program** secara **sequensial**. Anda tentu pernah membuat program yang menampilkan "Hello World!" atau program yang mengurutkan sejumlah angka acak. Kedua hal tersebut dapat dikategorikan sequential programs. Sebuah **program sequensial** berarti sebuah **program yang hanya memiliki satu aliran eksekusi.** Untuk setiap eksekusinya, ia mempunyai **titik awal,** sebuah **sekuen eksekusi**, dan **titik akhir.** Dalam sebuah runtime hanya terdapat satu proses yang telah dieksekusi.

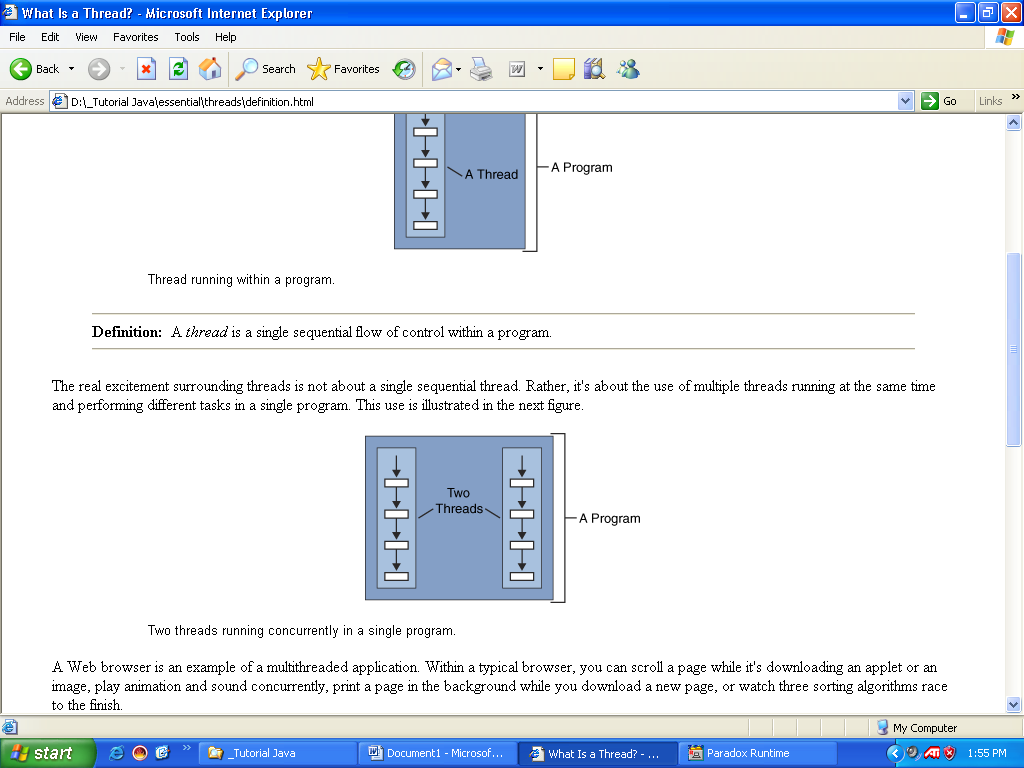
**Thread** dapat diartikan **aliran eksekusi** dari **suatu task** dalam sebuah **program.** Sebuah thread mempunyai **awal,** **tahapan,** dan **akhir** serta membutuhkan **waktu** untuk **pelaksanaannya.** **Thread bukan**lah sebuah **program** tetapi merupakan **bagian dari program** dan **tidak dapat menjalankan** **dirinya sendiri** tetapi berjalan dalam sebuah program. Gambar dibawah ini mengambarkan cara kerja single thread.



Gambar 8‑1 Cara kerja sebuah thread pada sebuah program

Seperti yang sudah dijelaskan bahwa thread merupakan aliran eksekusi dari suatu *task* dalam program. Adakalanya suatu **program** harus **melakukan beberapa *task*** dalam **waktu** yang **bersamaan** untuk menghasilkan program yang **lebih interaktif** **dan responsif**, misalnya program **winamp** yang pada saat yang bersamaan program harus mengeluarkan **voice**, menampilkan **timer,** dan menampilkan **visualisasi.** Oleh karena itu diperlukan suatu mekanisme agar program dapat menjalankan **beberapa *task*** pada saat yang bersamaan yang dikenal dengan **multithreading**. **Multithreading** adalah **kemampuan menjalankan** **beberapa thread** pada **waktu** yang **bersamaan** yang **masing-masing mengerjakan *task*** yang **berbeda-beda** dalam satu **program.**

**Keuntungannya** menggunakan **multithreading** adalah sebuah **program** akan **tampak stabil dan memiliki waktu respond user** yang **baik.** Skema multithreading dapat dilihat pada gambar dibawah ini.



Gambar 8‑2 Beberapa thread pada sebuah program / multithreading

## Membuat dan Mengunakan Threads

Java menyediakan class **java.lang.Thread** untuk membuat thread. Ada 2 cara untuk membuat thread:

### Subclassing Thread

Yaitu dengan **mendeklarasikan** sebuah **subclass** dari **class Thread** dan **meng-override** method **run.**

|  |
| --- |
| class NamaClass extends Thread {  //konstruktor  NamaClass() {  }  public void run() {  // isi task  . . .  }  } |

Untuk **menjalankan thread** maka kita harus membuat **objeknya** dan **memanggil** method **start().** Contoh :

Membuat thread untuk menampilkan angka 0 sampai 100

|  |
| --- |
| class SubclassThread extends Thread  {  public void run()  {  for(int i=0;i<=100;i++)  {  System.out.println(i);  }  }  }  class PanggilThread  {  public static void main(String[] args)  {  SubclassThread Thread1 = new SubclassThread();  Thread1.start();  }  } |

Membuat program simulasi lari marathon dimana 1 grup lari diwakili oleh sebuah thread.

|  |
| --- |
| class GrupLari extends Thread{  private String[] nama;  private int waktuTotal=0;  int lama=(((int)(Math.random()\*1000))+1000);  public GrupLari(String[] p){  nama=p;  }  @Override  public void run(){  /\* aktivitas lari dimulai dari pelari pertama sampai pelari terakhir \*/  for(int i=0;i<nama.length;i++){  System.out.println(nama[i]+" Mulai berlari");  if(i<nama.length-1)  System.out.println(nama[i]+" Memberikan tongkat kepada "+nama[i+1]+" ("+lama/100+" detik)");  else  System.out.println(nama[i]+" Sampai garis finish ("+lama/100+" detik)");  System.out.println("Waktu Total : "+waktuTotal+" detik");  try{  sleep(lama);  waktuTotal=waktuTotal+(lama/100);  }  catch(InterruptedException e){}  }  }  }  //kurang mainnya  class PanggilThread  {  public static void main(String[] args)  {  String[] nama = {"tegar","tiara","haluan","tia"};  GrupLari Thread1 = new GrupLari(nama);  Thread1.start();  }  } |

### Implementasi Runnable interface

**Mendeklarasikan** sebuah class yang **mengimplementasikan** method ***run()*** pada **interface Runnable.**

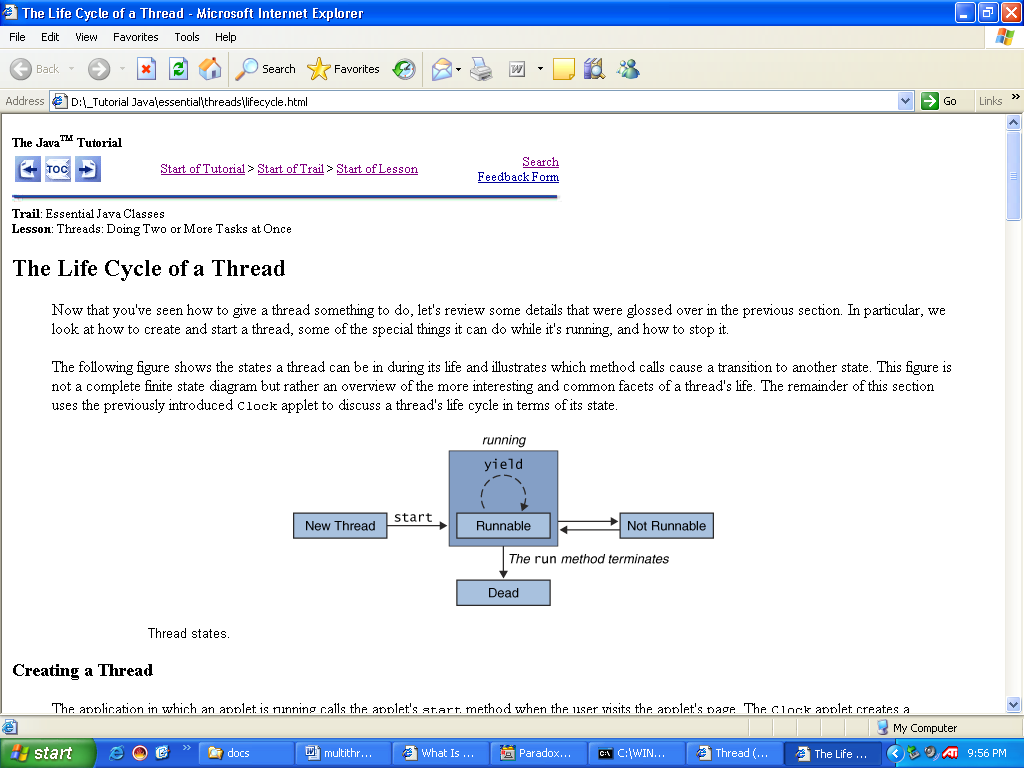
|  |
| --- |
| class NamaClassimplements Runnable{  public void run() {  // isi task  . . .  }  } |

Contoh :

|  |
| --- |
| class impRunnable implements Runnable{  public void run(){  for(int i=0;i<=100;i++){  System.out.println(i);  }  }    public static void main(String[] args){  impRunnable Thread1 = new impRunnable();  Thread t=new Thread(Thread1);  t.start();  }  } |

Setiap thread mempunyai prioritas. **Prioritas thread** berupa **integer** dalam **range 1** (MIN\_PRIORITY) sampai **10** (MAX\_PRIORITY). Dengan method ***setPriority()*** kita dapat **memodifikasi prioritas thread** tertentu. Thread yang mempunyai **prioritas lebih tinggi** akan **dieksekusi lebih dulu** daripada thread dengan prioritas lebih rendah. Jika **ada dua thread mempunyai prioritas** yang **sama,** maka **CPU** akan **memilih** di antara kedua thread tersebut.

Daur hidup sebuah thread adalah sebagai berikut :

  
Gambar 8‑3 Daur hidur sebuah thread

1. **New Thread: kondisi awal** dari sebuah thread. Ini terjadi **ketika** sebuah **objek thread** dibuat.
2. **Runnable:** Terjadi ketika **thread** sedang **mengeksekusi task** (menjalankan method run).
3. **Not Runnable:** Merupakan **kondisi** pada **thread** di mana **thread tidak** melakukan **eksekusi** apa apa tetapi **thread** masih dalam keadaan **running**.
4. **Dead:** Kondisi thread di mana thread telah **selesai mengeksekusi *task*.**

## Method pada Thread

Java menyediakan beberapa method pada class **java.lang.Thread** untuk mendukung multithreading,antara lain:

* 1. **setName(String name):** Method yang digunakan untuk **merubah nama thread.**
  2. **setPriority(int priority):**Digunakan untuk **merubah prioritas thread.**
  3. **getName():** Digunakan untuk **mengembalikan nama thread**
  4. **getPriority():**Digunakan untuk **mengembalikan prioritas thread**
  5. **isAlive():** Digunakan untuk **mengetahui apakah thread masih hidup.**
  6. **start():** Digunakan untuk **menjalankan thread.**
  7. **stop():** Digunakan untuk **menghentikan thread.**
  8. **sleep(long ms):** Digunakan untuk **menghentikan aktivitas thread** selama **waktu** yang **ditentukan** melalui parameter (dalam milisecond)
  9. **suspend():** Digunakan untuk **menghentikan aktivitas** **thread** sementara **sampai method** ***resume()*** **dipanggil.**
  10. **resume():** Digunakan untuk **menjalankan aktivitas thread** yang **tersuspend.**

Methods pada thread tidak hanya seperti yang telah disebutkan saja. Methods yang lain dapat dipelajari sendiri.

## Sinkronisasi Thread

Pada suatu keadaan tertentu beberapa **thread membutuhkan** sebuah **resource** yang sama. Walaupun **thread** dapat **berjalan bersama sama** tetapi untuk **menggunakan** sebuah **resource** diperlukan suatu **aturan** dalam **pengaksesannya** sehingga dalam **satu waktu** hanya ada **sebuah thread** yang dapat **menggunakan resource**. Hal ini dikenal dengan **sinkronisasi thread**.

Contoh kasus sederhana adalah penulisan file. Dalam satu waktu hanya ada satu thread yang diperbolehkan menulisi file. Jika file ditulisi secara bersama sama maka isi file bias menjadi tidak valid. Didalam proses penulisann file inilah diperlukan sinkronisasi.

Jika **suatu method** atau **objek** dilakukan **sinkronisasi,** maka **otomatis objek** atau **method** tersebut **dilock** dan jika ada **thread** yang akan **mengakses** harus **menunggu sampai method** tersebut **selesai dijalankan** atau **objek tersebut direlease**. Sintaks penggunaannya sinkronisasi pada java dapat dibagi menjadi 2 :

1. **Sinkronisasi Method**

Sinkronisasi hanya dilakukan terhadap method yang mempunyai identifier ***shynchronized.***

|  |
| --- |
| public **synchronized** int namaMethod()  {  // isi method  **…**  } |

1. **Sinkronisasi Objek**

Sinkronisasi dilakukan terhadap seluruh method pada objek*.*

|  |
| --- |
| synchronized(object)  {  // isi method  …  } |

Contoh :

1. Membuat program multithreading dengan memanfaatkan sinkronisasi thread dimana satu thread berfungsi menambah angka ke resource sedangkan thread lain mengambil angka di resource.

|  |
| --- |
| public class SinkronisasiThread{  private static Resources r;  public static void main(String[] args){  r = new Resources();  new Ambil().start();  new Tambah().start();  }  static class Ambil extends Thread{  public void run(){  System.out.println("Ambil berjalan");  for (int a = 0; a < 10; a++){  r.ambil();  }  }  }  static class Tambah extends Thread{  public void run(){  System.out.println("Tambah berjalan");  for (int a = 0; a < 10; a++){  r.tambah();  }  }  }  static class Resources {  private boolean b = false;  private byte nilai = 0;  private synchronized void tambah(){  while (b==true){  try{  System.out.println("Menunggu untuk Ambil");  wait();  }  catch (InterruptedException e){}  }  System.out.println("Tambah nilai "+ ++nilai);  b = true;  notifyAll();  }  private synchronized void ambil(){  while (b==false){  try{  System.out.println("Menunggu untuk Tambah");  wait();  }  catch (InterruptedException e){}  }  System.out.println("Ambil nilai "+ nilai);  b = false;  notifyAll();  }  }  } |

|  |  |
| --- | --- |
| 1. Sebutkan kegunaan dari method run( )? 2. Apa kekurangan dari synchronized? 3. Apa kekurangan dari Thread? 4. Bagaimana kita menjalankan thread baru? | tanda tanya |