

MARI BELAJAR

PEMRGORAMAN JAVA

CONCURRENCY

UNTUK PEMULA

OLEH IZZAT ARRAMSYAH

KONTEN

| | |
|-----------------------------------|----|
| Pendahuluan | 03 |
| Bagian I Thread | 04 |
| Bagian II Runnable Interface | 06 |
| Bagian III Executor Service | 08 |
| Bagian IV Synchronized Methods | 10 |

PENGENALAN

APA ITU CONCURRENCY PADA JAVA ?

Concurrency adalah sebuah konsep dasar pada java yang memungkinkan untuk menjalankan beberapa Thread berjalan secara bersamaan, memungkinkan untuk memanfaatkan sumber daya CPU secara efisien dan meningkatkan kinerja aplikasi Java. Java sendiri menyediakan beberapa tools untuk mengelola Concurrency ini. Berikut adalah beberapa tools yang berkaitan dengan Concurrency pada Java :

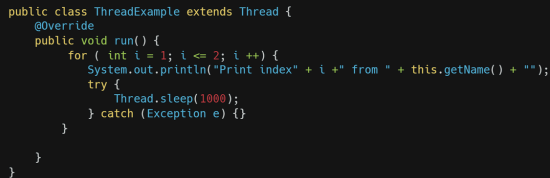
- 1.Thread
- 2.Runnable Instance
- 3.Executor Service
- 4.Syncronized Methods

BAGIAN I

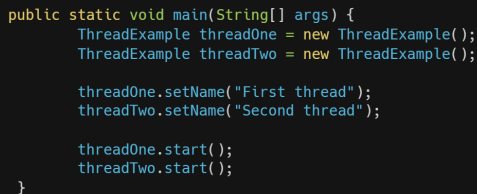
THREAD

'Thread' merupakan class dasar yang menjadi bagian dari Concurrency pada Java. Hal ini memungkinkan kita untuk mengelola thread yang merupakan jalur eksekusi independent dalam program Java. Thread memungkinkan aplikasi melakukan banyak tugas secara bersamaan (multithreading).

Contoh Kode :



```
public class ThreadExample extends Thread {
    @Override
    public void run() {
        for (int i = 1; i <= 2; i++) {
            System.out.println("Print index" + i + " from " + this.getName() + "");
            try {
                Thread.sleep(1000);
            } catch (Exception e) {}
        }
    }
}
```



```
public static void main(String[] args) {
    ThreadExample threadOne = new ThreadExample();
    ThreadExample threadTwo = new ThreadExample();

    threadOne.setName("First thread");
    threadTwo.setName("Second thread");

    threadOne.start();
    threadTwo.start();
}
```

BAGIAN I

Hasil :

```
Print index1 from Second thread
Print index1 from First thread
Print index2 from Second thread
Print index2 from First thread
```

Contoh kode diatas adalah bagaimana kita mengimplementasikan class Thread menggunakan extends Thread. Jika dilihat pada hasil kedua thread tersebut berjalan secara independent.

BAGIAN II


RUNNABLE INTERFACE

Runnable Interface dirancang untuk menyediakan mekanisme agar kelas dapat di-eksekusi oleh sebuah Thread. Dengan kata lain, Runnable Interface digunakan untuk mendefinisikan tugas yang akan dijalankan oleh Thread.

Contoh Kode :



```
public class RunnableExample implements Runnable {  
    private String name;  
  
    public RunnableExample( String name) {  
        this.name = name;  
    }  
  
    @Override  
    public void run() {  
        for ( int i = 1; i <= 2; i++) {  
            System.out.println("Print index" + i +" from " + name + "");  
            try {  
                Thread.sleep(1000);  
            } catch (Exception e) {}  
        }  
    }  
}
```



```
public static void main(String[] args) {  
    Thread threadOne = new Thread(new RunnableExample("First thread"));  
    Thread threadTwo = new Thread(new RunnableExample("Second  
thread"));  
    threadOne.start();  
    threadTwo.start();  
}
```

Hasil :

```
Print index1 from First thread  
Print index1 from Second thread  
Print index2 from Second thread  
Print index2 from First thread
```

Contoh kode diatas adalah bagaimana cara untuk mengimplementasikan runnable interface. secara penulisan code tidak banyak berubah dari cara mengimplementasikan dengan menggunakan `extends Thread`. Hanya saja ada sedikit perubahan pada main class karena kita perlu mengirim object yang mengimplementasikan **Runnable Interface** `Thread threadOne = new Thread(new RunnableExample("First thread"))`.

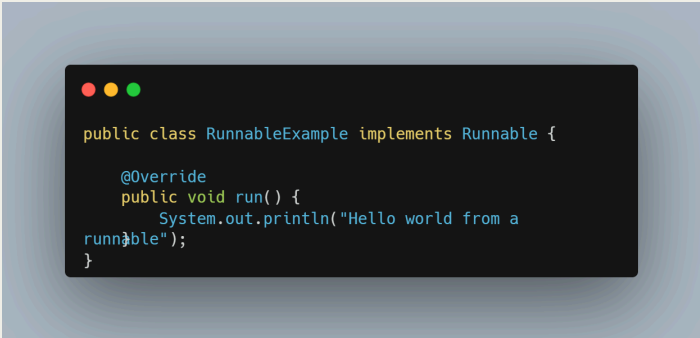
Lalu mana yang lebih baik antara `Thread` atau `Runnable Interface` ? kembali lagi bahwa semua itu tergantung kebutuhan aplikasi kita. Jika kita menggunakan **extends Thread** kita tidak bisa menambahkan `extends` dari class lain. Karena pada java tidak mengizinkan multiple inheritance (Tidak bisa menjadi subclass dari class lain selain `Thread`). Hal ini menyebabkan penggunaan **extends Thread** sangat terbatas. Namun jika kita menggunakan **Impelemnts Runnable** kita bisa melakukan `extends` dari class lain bahkan kita bisa menambahkan implementasi interface. Hal ini membantu kita jika aplikasi yang kita kembangkan cukup kompleks karena bisa lebih flexible dibandingkan dengan menggunakan **extends Thread**.

BAGIAN III


EXECUTOR SERVICE

Executor Service menyediakan cara yang lebih flexible dan powerful untuk mengelola thread dibanding menggunakan Thread atau Runnable secara langsung. Dengan Executor Service kita dapat dengan mudah mengelola eksekusi thread.

Contoh Kode :



```
public class RunnableExample implements Runnable {  
    @Override  
    public void run() {  
        System.out.println("Hello world from a  
runnable");  
    }  
}
```



```
public static void main(String[] args) {  
    Runnable runnable = new RunnableExample();  
  
    ExecutorService executorService = Executors.newFixedThreadPool(2);  
    executorService.submit(new RunnableExample());  
    executorService.submit(() -> System.out.println("Hello from a runnable running in an  
ExecutorService"));  
  
    executorService.shutdown();  
}
```


BAGIAN III

Hasil :

```
Hello world from a runnable  
Hello from a runnable running in an ExecutorService
```

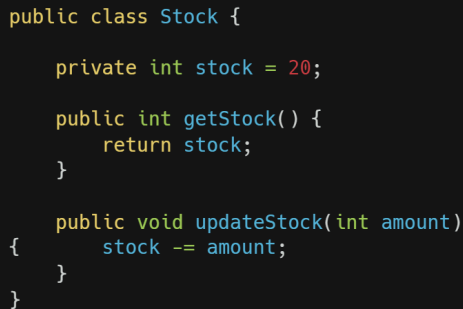
Pada contoh kode diatas merupakan contoh sederhana dari Exectuor Service. Jika kita lihat method *newFixedThreadPool* digunakan untuk mendefinisikan jumlah thread pool. *submit* digunakan untuk memberikan tugas ke dalam Executor Service. Sementara *shutdown* digunakan untuk menghentikan Executor Service setelah semua tugas selesai.

BAGIAN IV

SYNCHRONIZED METHODS

Synchronized Method digunakan untuk mengotrol akses terhadap satu method oleh beberapa thread sekaligus. Saat menggunakan Synchronized Method, hanya ada satu thread yang dapat mengakses method tersebut.

Contoh Kode :



```
public class Stock {  
    private int stock = 20;  
  
    public int getStock() {  
        return stock;  
    }  
  
    public void updateStock(int amount)  
{  
        stock -= amount;  
    }  
}
```

BAGIAN IV

```
public class Store {  
  
    static synchronized void purchase(String username, Stock stock, int amount)  
    {  
        System.out.println(username + " want to purchase " + amount);  
        int stock = stock.getStock();  
        if(stock - amount < 0) {  
            System.out.println("Not enough stock");  
        } else {  
            System.out.println("Item is in stock");  
            stock.updateStock(amount);  
            System.out.println(amount + " items purchased");  
        }  
        System.out.println("Current stock: " + stockChecker.getStock());  
    }  
  
    public static void main(String[] args) {  
        Stock stock = new Stock ();  
        ExecutorService executorService = Executors.newFixedThreadPool(5);  
        executorService.submit(() -> purchase("user 1", stock, 10));  
        executorService.submit(() -> purchase("user 2", stock, 10));  
        executorService.submit(() -> purchase("user 3", stock, 20));  
        executorService.shutdown();  
    }  
}
```

Hasil:

```
user 1 want to purchase 10  
Item is in stock  
10 items purchased  
Current stock: 10  
user 3 want to purchase 20  
Not enough stock  
Current stock: 10  
user 2 want to purchase 10  
Item is in stock  
10 items purchased  
Current stock: 0
```

BAGIAN IV

Pada contoh kode diatas kita membuat sebuah studi kasus pembelian sebuah barang ditoko. Dimana stok barang hanya tersisa 20pcs namun ada 3 user yang mengakses fitur pembelian secara bersamaan. Jika dilihat pada hasil running code diatas, User 3 tidak dapat membeli stok sebanyak 20pcs dikarenakan user 1 sudah lebih dulu melakukan pembelian sebanyak 10pcs. *synchronized void purchase* ini merupakan Synchronized Methods dimana seluruh method disinkronkan dan jika satu thread mengeksekusi method tersebut maka thread lain harus menunggu.

SOURCECODE

<https://github.com/izzatarramsyah/Concurrency-Example-Java>