MODUL 5
Multi Threading 2

A. Tujuan :

1. Memahami tentang single thread
2. Memahami konsep dan penggunaan dari multi thread

B. Dasar Teori

Perbedaan kelas Thread dan Runnable pada java:

1. Didalam kelas thread pada java, kita tidak bisa memperpanjang thread yang telah kita buat. Dikarenakan java sendiri tidak mendukung banyak turunan layaknya bahasa pemrograman lain seperti C++. Maka dari itu, implementasi kelas Runnable lebih dianjurkan dikarenakan kita lebih diberi kebebasan untuk memperpanjang kelas sebanyak yang kita mau
2. Ketika kita menggunakan kelas Runnable, kita mendapatkan kebebasan untuk menggunakan kembali behavior class yang sudah tidak terpakai. Berbeda dengan thread class, behavior yang sudah tidak terpakai tidak akan dapat unutk di gunakan lagi.
3. Runnable thread lebih cocok bagi programmer yang berorientasi objek.

Sinkronisasi adalah suatu proses pengendalian akses dari sumber daya terbagi pakai (*shared resource*) oleh banyak thread sedemikian sehingga hanya satu thread yang dapat mengakses sumber daya tertentu pada satu waktu.
Dalam aplikasi multithreaded yang tidak tersinkronisasi, sangat mungkin terjadi adanya satu thread memodifikasi suatu obyek yang dipakai bersama pada saat thread lain sedangkan dalam proses menggunakan atau mengupdate nilai obyek tersebut. Sinkronisasi mencegah jenis kerusakan data demikian, jika tidak disinkronkan maka dapat mengakibatkan pembacaan yang buruk dan error yang signifikan. Secara umum bagian kritis (*critical sections*) dari kode biasanya ditandai dengan kata kunci *synchronized*.
Terdapat 2 (dua) bagian kode yang dapat dikenakan sinkronisasi di dalam Java:

- *synchronized method*
- *synchronized block*

Sinkronisasi pada method mencegah dua thread mengeksekusi method pada object dan dan waktu yang sama. Method yang disinkronisasi ditandai dengan kata kunci *synchronized*. Ketika method tersebut dipanggil maka thread akan mengaktivkan sebuah *object-lock* atau monitor. Jika thread lain mencoba untuk memanggil method tersebut maka akan didapati bahwa method tersebut terkunci dan berada dalam keadaan suspense sampai kunci/monitor terbuka.
Sinkronisasi pada method umumnya digunakan untuk mengsinkronisasi akses pada resource. Pada aplikasi multhithread, method dapat disinkronisasi untuk mencegah hilang dan rusaknya data. Contohnya adalah sebuah counter yang disimpan dalam sebuah file, misalkan counter nilai berapa kali sebuah aksi seperti mengakses web

terjadi. Counter ini dapat bertambah (dengan cara membaca nilai saat ini dan menulis yang baru) atau dibaca oleh beberapa thread. Jika satu thread mencoba untuk menambah nilai counter sebelum thread yang lain selesai memodifikasi counter, dengan kata lain nilai nilai tersebut di-*set* oleh satu thread dan di-*override* oleh yang lain. Itu berarti counter akan membaca nilai yang tidak valid. Terlebih lagi, jika ada dua upaya untuk meng-*override* nilai tersebut, maka file dapat rusak. Jika method untuk mengakses dan memodifikasi nilai counter disinkronisasikan, maka hanya ada satu thread yang bisa melakukan satu tindakan menulis pada satu waktu tertentu.

C. Tugas Pendahuluan

Buatlah desain flowchart untuk setiap soal dalam percobaan

D. Percobaan

D.1. Latihan

1. Menghentikan Thread

Sebagian thread dapat mengirimkan perintah stop kepada thread yang lain, dengan cara menggunakan perintah Thread.Stop().

```
class stoppingThread extends Thread {
   @Override
   public void run(){
      int count = 1;
      System.out.println ("Saya berhitung. Lihatlah!");
      for (;;) {
         System.out.print (count++ + " ");
         //sleep 0.5 detik
         try {
            Thread.sleep(500);
         } catch (InterruptedException ie) {
         }
      }
   }
}

public class StopThread {
   public static void main(String[] args) throws java.io.IOException
   {
      // TODO code application logic here
      Thread counter = new stoppingThread();
      counter.start();
      //menunggu input dari user
      System.out.println ("Tekan enter untuk menghentikan perhitungan thread");
      System.in.read();
      //interrupsi thread
      counter.stop();
   }
}
```

Amati hasilnya dan perhatikan bahwa proses Thread akan terhenti bila ditekan tombol “enter”.

2. Suspending/resuming Thread

Didalam java terdapat fungsi untuk memberhentikan kerja thread sementara, sehingga thread masih dapat digunakan kembali. Mekanisme pemberhentian thread sementara didalam java dikenal dengan fungsi “suspend” dengan menggunakan method Thread.Suspend(). Dan untuk membangkitakannya maka menggunakan method Thread.Resume().

```
class suspendingThread extends Thread {
   @Override
   public void run(){
      int count = 1;
      System.out.println ("Saya berhitung. Lihatlah!");
      for (;;) {
         System.out.print (count++ + " ");
         //sleep 0.5 detik
         try {
            Thread.sleep(500);
         } catch (InterruptedException ie) {
         }
      }
   }
}

public class SuspendThread {
   public static void main(String[] args)throws java.io.IOException {
      // TODO code application logic here
      Thread counter = new suspendingThread();
      counter.start();
      //menunggu input dari user
      System.out.println ("Tekan enter untuk SUSPEND perhitungan thread");
      System.in.read();
      counter.suspend();
      System.out.println ("Tekan enter untuk RESUME perhitungan thread");
      System.in.read();
      counter.resume();
      System.out.println ("Tekan enter untuk STOP perhitungan thread");
      System.in.read();
      counter.stop();
   }
}
```

Amati hasilnya dan perhatikan bahwa:

a. Begitu menekan tombol “enter” yang pertama maka thread akan terhenti sementara
b. Thread akan dilanjutkan setelah menekan tombol “enter” lagi
c. Dan thread akan berhenti, begitu menekan tombol “enter” lagi

3. Menentukan prioritas

Prioritas thread digunakan oleh penjadwal thread untuk menentukan thread mana yang seharusnya dikerjakan terlebih dahulu. Dalam hal ini rentang level dari MIN_PRIORITY sampai MAX_PRIORITY, yaitu 1 sampai 10. Dengan NORM_PRIORITY memiliki nilai 5. Prioritas pertama adalah yang nilainya paling besar.

```
import java.util.logging.Level;
import java.util.logging.Logger;

class HaloThread extends Thread {
    String s;
    public HaloThread (String ss) {
        this.s = ss;
    }
    @Override
    public void run() {
        for (int i=0;i<10;i++){
            System.out.println (s+" "+i+" : Haloo");
            try {
                sleep(1000);
            }
            catch (InterruptedException ex){
                Logger.getLogger(HaloThread.class.getName()).log(Level.SEVERE, null,ex);
            }
        }
    }
}
public class PriorityThread {
    public static void main(String[] args) {
        // TODO code application logic here
        HaloThread h1 = new HaloThread ("Thread 1");
        HaloThread h2 = new HaloThread ("Thread 2");
        HaloThread h3 = new HaloThread ("Thread 3");
        h1.setPriority(Thread.MIN_PRIORITY);
        h2.setPriority(Thread.NORM_PRIORITY);
        h3.setPriority(Thread.MAX_PRIORITY);
        h1.start();
        h2.start();
        h3.start();
    }
}
```

Amati hasilnya, apakah thread dengan prioritas paling tinggi selalu didahulukan, bila tidak, mengapa hal tersebut terjadi?

4. Sinkronisasi pada level method

Kelas berikut mendefinisikan sebuah counter dengan method akses dan modifikasi yang disinkronisasi.

```
class Counter {
   private int countValue;
   public Counter (){
      countValue = 0;
   }
   public Counter (int start){
      countValue = start;
   }
   //metode sinkronisasi utk perhitungan counter
   public synchronized void increaseCount() {
      int count = countValue;
      try {
         Thread.sleep(1000);
      } catch (InterruptedException ie){
      }
      count = count + 1;
      countValue = count;
   }
   public synchronized int getCount() {
      return countValue;
   }
}

class hitungThread implements Runnable {
   Counter myCounter;
   int countAmount;
   public hitungThread (Counter counter, int amount) {
      myCounter = counter;
      countAmount = amount;
   }
   @Override
   public void run(){
      for (int i=1;i<=countAmount;i++){
         myCounter.increaseCount();
         System.out.println (Thread.currentThread().getName()+" "+myCounter.getCount());
      }
   }
}

public class CountingThread {
   public static void main(String[] args) throws Exception {
      // TODO code application logic here
      Counter c = new Counter ();
      //instace Runnable akan menghitung 10x utk msg2 thread
      Runnable runner = new hitungThread (c,10);
      System.out.println ("Mulai perhitungan threads");
      Thread t1 = new Thread (runner);
      Thread t2 = new Thread (runner);
      Thread t3 = new Thread (runner);
```

```
        t1.start();
        t2.start();
        t3.start();
        //menunggu 3 thread selesai menjalankannya
        t1.join();
        t2.join();
        t3.join();
        System.out.println ("Nilai counter: " + c.getCount());
    }
}
```

Awal program, hilangkan kata **synchronized** pada program, amati bahwa program akan berjalan independent satu sama lain. Catat hasilnya.
Setelah itu tambahkan kata **synchronized** untuk mensinkronkan ketiga thread tersebut. Amati hasilnya dan catat perbedaannya.

D.2. Permasalahan

1. Mengontrol Thread dapat dilakukan dengan menggunakan method stop(), suspend(), ataupun resume(). Selain menggunakan method tersebut, juga bisa menggunakan variable bertipe boolean untuk pengontrolnya.
   Buatlah program menggunakan thread. Saat program dijalankan maka akan tampil counter bilangan genap, misal: 2 4 6 8 10 12 14 16….dst. Ketika user menekan “enter” maka dilakukan counter mundur bilangan ganjil, misal: 15 13 11 9 7 5 3 1…dst. hingga user kembali menekan “enter” sehingga counter dihentikan.
2. Terdapat 3 mahasiswa yang akan mengerjakan tugas bersama pada sebuah file. Tugasnya adalah menuliskan karakter huruf ‘A’ sampai ‘Z’ secara berurutan dan bersama-sama. Asumsikan mahasiswa sebagai thread, dan lakukan sinkronisasi tugas.

E. Laporan Resmi :

1. Analisalah semua program diatas dan buat kesimpulan.