Threads of Control

Iwan Setyawan Adji, 32438-TE
Jurusan Teknik Elektro FT UGM,
Yogyakarta

1 PENDAHULUAN

Threads adalah segala sesuatu yang terdapat dalam halaman dari suatu website yang biasanya ditampilkan melalui web browser. Web browser ini dapat melakukan pengontrolan terhadap threads (Threads of Control). Web browser menggunakan sistem threads dimana semua pengolahan data digital untuk ditampilkan pada halaman web dilakukan secara multiprosessing atau paralel sehingga halaman website tersebut seakan-akan dipotong-potong terlebih dahulu agar memperingan proses penampilan suatu halaman. Halaman web dapat menampilkan data yang sekiranya cepat untuk ditampilkan terlebih dahulu, misalnya saja teks. Jadi kita dapat menikmati informasi meskipun belum semuanya secara totalitas dengan membaca halaman web tersebut sambil menunggu proses lain selesai, misalnya saja data berupa image/gambar yang pastinya berukuran lebih besar daripada teks dan membutuhkan bandwidth yang cukup tinggi agar prosesnya download gambar pada web cepat selesai.

1.1 Perbedaan Multithreading dan Multiprocessing

Multiprocessing adalah sebuah proses yang mengacu pada dua atau lebih program yang dieksekusi dalam waktu bersamaan, di bawah kontrol sistem operasi. Program tidak perlu hubungan dengan satu sama lain karena proses yang dilakukan memang secara terpisah dan bersamaan untuk memperingan proses transfer informasi namun tetap menempatkan diri sesuai tempatnya. Dan ketika telah selesai maka kita dapat melihat satu kesatuan yang utuh kembali.

Multithreading adalah proses dimana terjadi dua atau lebih tugas yang dieksekusi secara bersamaan, dalam satu program. Hal ini sering disebut sebagai prosesan ringan, hal tersebut terjadi mungkin karena beban yang terlibat dalam membuat semua pekerjaan tersebut tidak begitu berat. Multithreading dapat memproduksi program yang dapat menyelesaikan lebih banyak pekerjaan pada waktu yang sama walaupun keadaan CPU dibagi-bagi antar tugas. Sedangkan multiprosesing biasanya sudah diimplementasikan di dalam sistem operasi. Dua atau lebih program mampu melakukan eksekusi pada platform dengn multiprocessing tanpa perlu menggunakan requirement pada planning.

2 Tipe Program Thread

Dua cara untuk membuat suatu program thread dalam bahasa Java antara lain:

- Implement the Runnable interface
- Extend the Thread class

2.1 Implement the Runnable Interface

Hal-hal yang diperlukan untuk menghasilkan sebuah thread di Java adalah

- Menginisiasikan Obyek berbentuk thread

- Menjalankan run methode

Code untuk menyelesaikan objektif (do the work) dari thread di tempatkan pada method run(), atau pada method lain yang dilibatkan pada method run().

Salah satu cara (yang akan menjadi topik bagian selanjutnya) untuk instantiate objek dari kelas thread adalah untuk

- Memperluas kelas Thread ke dalam kelas anda sendiri
- Kemudian instantiate objek kelas anda.

Penting: Untuk memulai thread benar-benar berjalan, Anda tidak memanggil run () method. Sebaliknya, Anda memanggil start () method pada objek.

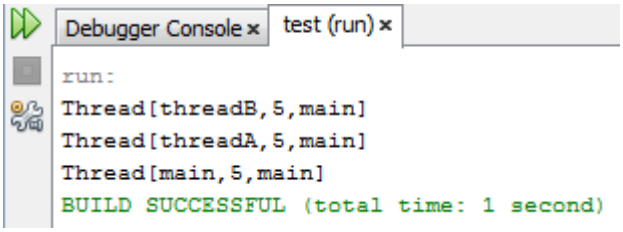
Namun, kadang-kadang tidak mungkin untuk memperluas kelas Thread, karena Anda harus memperluas beberapa kelas yang lain. Ingat bahwa Java tidak mendukung pewarisan berganda. Dalam pemrograman applet, misalnya, Anda harus memperpanjang kelas Applet. Dalam kasus-kasus di mana tidak mungkin untuk memperluas kelas Thread, Anda dapat merancang kelas untuk memperluas beberapa kelas yang diperlukan, dan juga untuk mengimplementasikan interface Runnable.

Jadi, cara kedua (yang merupakan topik bagian ini) untuk instantiate objek dari kelas thread adalah untuk Menerapkan Runnable interface kedalam kelas Anda sendiri (memperluas beberapa kelas yang lain jika perlu), dan kemudian instantiate objek kelas Anda. Seperti sebelumnya, Anda akan menimpa run () method dari kelas Thread di kelas baru Anda. Pendekatan ini diilustrasikan pada contoh program berikut

```
/*File Thread01.java  Copyright 1997, R.G.Baldwin
Illustrates instantiation and running of threads using the
runnable interface instead of extending the Thread class.

Tested using JDK 1.1.3 under Win95.

Outputnya adalah:
Thread[threadA,5,main]
Thread[threadB,5,main]
Thread[main,5,main]


**********************************************************/

class Thread01{
  static public void main(String[] args){
    //Instantiate dua ojek threads
    Thread myThreadA =
      new Thread(new MyThread(),"threadA");
    Thread myThreadB =
      new Thread(new MyThread(),"threadB");

    //Memulai running
    myThreadA.start();
    myThreadB.start();

    try{
      //tunda satu detik
      Thread.currentThread().sleep(1000);
    }catch(InterruptedException e){}

    //Tampilkan info tentang thread utama
```

```
    System.out.println(Thread.currentThread());

  }//akhir main
}//akhir kelas Thread01
//=========================================================//

class DoNothing{
// Kelas ini ada hanya untuk diwariskan oleh MyThread
// kelas ini untuk mencegah bahawa kelas dapat diwariskan
// kelas Thread , jadi menuntutnya untuk implementasi
// Runnable interface instead.
}//akhir kelas DoNothing
//=========================================================//

class MyThread extends DoNothing implements Runnable{
  public void run(){
    //Menampilkan info tentang keterangan-keterangan Thread
System.out.println(Thread.currentThread());
  }//akhir run
}//akhir MyThread
```

Hasil capture compile:



Seperti yang anda lihat, program ini mendefinisikan kelas baru bernama MyThread yang membentang kelas bernama DoNothing dan mengimplementasikan Runnable interface. Kami menimpa run () method di kelas bernama MyThread untuk menampilkan informasi tentang thread dilaksanakan oleh sebuah objek dari kelas MyThread. Satu-satunya alasan untuk memperpanjang DoNothing dalam contoh sederhana ini adalah untuk mengkonsumsi warisan satu-satunya jalan yang tersedia untuk kelas Java. Tujuannya adalah untuk mensimulasikan situasi di mana Anda harus memperluas beberapa kelas yang lain. Memperluas kelas lain bukanlah suatu kebutuhan, dan pendekatan ini bekerja sama baiknya bila Anda tidak memperpanjang kelas lain.

Kode sedang main () instantiates dua jenis objek Thread menggunakan salah satu dari beberapa tersedia Thread konstruktor seperti yang ditunjukkan di bawah ini.

MyThreadA thread = new Thread (new MyThread (), "threadA");

Constructor khusus ini memerlukan dua parameter. Parameter pertama adalah obyek dari setiap kelas yang mengimplementasikan interface Runnable.

Parameter kedua adalah sebuah string yang menetapkan nama untuk Thread (Perhatikan bahwa nama yang diberikan kepada Thread independen dari nama variabel yang referensi referensi objek yang Thread.) Berbagai metode yang tersedia yang memungkinkan manipulasi status Thread dengan mengacu pada nama itu.Kode tambahan utama () mulai dua Thread berjalan dengan menerapkan start () method pada Thread masing-masing objek. Thread berhenti dan program berakhir ketika Thread ada lagi yang perlu dilakukan

(ini tidak terjadi di JDK 1.0.2.). Satu hal yang perlu diperhatikan adalah bahwa utama itu sendiri adalah suatu thread yang dimulai oleh penafsir.

Kode method run() menyebabkan nama masing-masing dari Thread (bersama dengan beberapa informasi lainnya) yang akan ditampilkan untuk kedua dari dua Thread instantiated dan mulai dalam program ini.

Kode sedang main () menyebabkan informasi serupa untuk ditampilkan untuk thread utama juga.

Sleep () method dipanggil pada thread utama untuk melaksanakan satu detik delay. Sleep () metode dan sejumlah metode Thread lain akan dibahas dalam bagian berikutnya.

Program ini menggambarkan pertama dari dua alternatif pendekatan untuk instantiate dan menjalankan Thread dalam sebuah program Java. Ini adalah pendekatan yang harus digunakan setiap kali kelas yang digunakan untuk instantiate objek Thread diperlukan untuk memperluas beberapa kelas yang lain.

Seperti disebutkan di atas, pendekatan ini juga dapat digunakan dalam kasus-kasus di mana tidak perlu untuk memperluas beberapa kelas yang lain baik begitu. Ini adalah yang paling umum dari dua pendekatan yang sedang dibahas.

Pendekatan berikutnya dapat digunakan ketika kelas digunakan untuk instantiate objek Thread tidak diperlukan untuk memperpanjang beberapa kelas lain, dan karena itu, dapat memperpanjang kelas Thread.

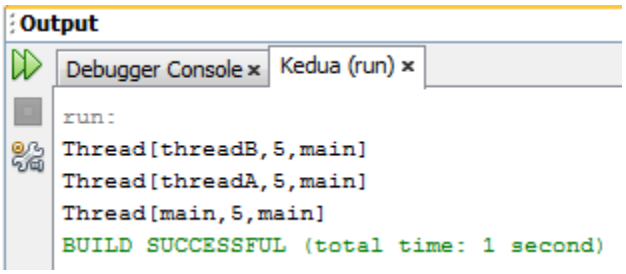
2.2 Extend the Thread class

Pada dasarnya sama dengan program sederhana yang mewarisi kelas Thread daripada untuk mengimplentasikan interface Runnable.

```
/*File Thread02.java  Copyright 1997, R.G.Baldwin
Illustrates instantiation and running of threads by
extending the Thread class instead of implementing the
Runnable class as was the case in the program named
Thread01.java

Tested using JDK 1.1.3 under Win95.

Outputnya adalah:
Thread[threadA,5,main]
Thread[threadB,5,main]
Thread[main,5,main]


**********************************************************/

class Thread02{
  static public void main(String[] args){
    //Instantiate dua objek thread baru
    Thread myThreadA =
      new Thread(new MyThread(),"threadA");
    Thread myThreadB =
      new Thread(new MyThread(),"threadB");

    //Mulai running
    myThreadA.start();
    myThreadB.start();

    try{//delay for one second
```

```
      Thread.currentThread().sleep(1000);
    }catch(InterruptedException e){}

    //Menampilkam info main thread
    System.out.println(Thread.currentThread());

  }//end main
}//akhir kelas Thread02
//=======================================================//

class MyThread extends Thread{
  public void run(){
    //Menampilkan info particular thread
    System.out.println(Thread.currentThread());
  }//akhir run
}//akhir MyThread
```

Hasil capture compile:



Dengan pendekatan ini, kelas baru bernama MyThreadextends di kelas Thread dan melaksanakan interface Runnable secara langsung. (Kelas yang mengimplementasikan Runnable Thread sehingga benar-benar MyClass implements Runnable tidak langsung.)

2.3 Another Approach which Extends the Thread Class

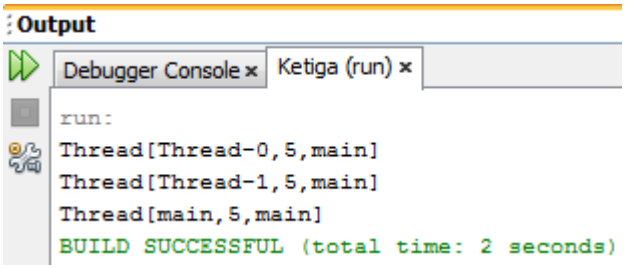
Program berikut ini juga memperpanjang kelas Thread tetapi agak sederhana daripada program sebelumnya. Thread individu memiliki nama default. Program ini menggunakan versi yang berbeda dari constructor Thread yang tidak memerlukan argumen.

Dengan demikian, pernyataan untuk instantiate objek yang Thread kurang kompleks. Jika tidak, pada dasarnya sama dengan program sebelumnya. Ini adalah sintaks yang akan paling sering melihat di Jawa buku-buku dan artikel yang membahas multithreading.

```
/*File Thread06.java  Copyright 1997, R.G.Baldwin
Illustrates another way to instantiate and run
threads by extending the Thread class instead of
implementing the Runnable class as was the case in
the program named Thread01.java
Tested using JDK 1.1.3 under Win95.

Outputnya adalah:
Thread[Thread-1,5,main]
Thread[Thread-2,5,main]
Thread[main,5,main]
```

```
**********************************************************/

class Thread06{
  static public void main(String[] args){
    //Instantiate two dua objek thread baru
    Thread myThreadA = new MyThread();
    Thread myThreadB = new MyThread();

    //Mulai running
    myThreadA.start();
    myThreadB.start();

    try{//tunda satu detik
      Thread.currentThread().sleep(1000);
    }catch(InterruptedException e){}

    //Menampilkan info manin thread
    System.out.println(Thread.currentThread());
    }//akhiri main()

}//akhiri kelas Thread06
//=======================================================//

class MyThread extends Thread{
  public void run(){
    //Menampilkan info particular thread
    System.out.println(Thread.currentThread());
  }//akhiri run
}//akhiri kelas MyThread
```

Hasil capture compile:



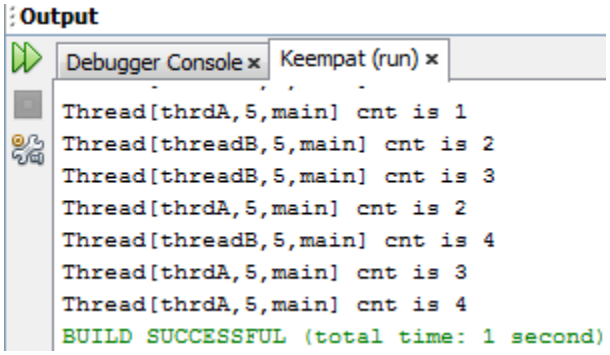
2.4 A Slightly More Substantive Sample Program

Mari kita memodifikasi salah satu program sampel sebelumnya menyebabkan Thread menjadi sesuatu yang sedikit lebih substantif sehingga dapat benar-benar melihat beberapa bukti yang bersamaan operasi.

```
/*File Thread03.java  Copyright 1997, R.G.Baldwin
Illustrates instantiation and running of threads using the
runnable interface.  Each of two threads counts from zero
to four and displays the count, sleeping a random amount
of time between counts.
Tested using JDK 1.1.3 under Win95.

Outputnya adalah:

Thread[thrdA,5,main] cnt is 0
```

```
Thread[threadB,5,main] cnt is 0
Thread[threadB,5,main] cnt is 1
Thread[threadB,5,main] cnt is 2
Thread[thrdA,5,main] cnt is 1
Thread[thrdA,5,main] cnt is 2
Thread[threadB,5,main] cnt is 3
Thread[thrdA,5,main] cnt is 3
Thread[threadB,5,main] cnt is 4
Thread[thrdA,5,main] cnt is 4

*********************************************************/

class Thread03{
  static public void main(String[] args){
    //Instantiate dua objek thread baru dengan panjang nama berbeda yang membantu
ketika melihat output.
    Thread threadA = new Thread(new MyThread(),"thrdA");
    Thread threadB = new Thread(new MyThread(),"threadB");

    //Start them running
    threadA.start();
    threadB.start();

    try{//tunda satu detik
      Thread.currentThread().sleep(1000);
    }catch(InterruptedException e){}

  }//akhiri main
}//akhir kelas Thread03
//=======================================================//

class MyThread implements Runnable{
  public void run(){
    for(int cnt = 0; cnt < 5; cnt++){
      System.out.println(Thread.currentThread() +
                                        " cnt is " + cnt);
      try{//delay a random amount of time
        Thread.currentThread().sleep(
                              (int)(Math.random() * 100));
      }catch(InterruptedException e){}
    }//akhiri loop
  }//akhiri run
}//akhiri kelas MyThread
```

Hasil capture compile:



Program ini Thread independen instantiates dua benda dan mulai menjalankannya asynchronously. Setiap thread menghitung dari nol sampai empat dan menampilkan hitungan beserta nama untuk setiap hitungan.

Setiap thread juga sleep jumlah acak waktu antara penting. Karena penggunaan waktu sleep acak, output dari setiap dijalankan akan berbeda. Output dari satu run ditampilkan dalam komentar pada awal program. Anda dapat melihat bagaimana setiap thread

- Mengaktifkan,
- Menghitung,
- Menampilkan, dan
- Sleep

Independen dari kegiatan Thread yang lain (kecuali bahwa mereka harus berbagi sumber daya yang tersedia ketika keduanya terjaga). Output dari setiap Thread bercampur dengan output dari yang lain. Hal ini karena kedua Thread berjalan bersamaan dan asynchronously.
Kedua, metode run ()

- Berjalan pada waktu yang sama,
- Bersaing untuk sumber daya jika tidak sleep, dan
- Menampilkan output mereka setiap kali mereka memiliki sesuatu untuk ditampilkan dan dapat mengontrol arus keluaran standar.

Sebuah program Java dapat memiliki banyak Thread, dan Thread tersebut dapat berjalan secara bersamaan, baik asynchronously atau serempak.

2.5 Method Run()

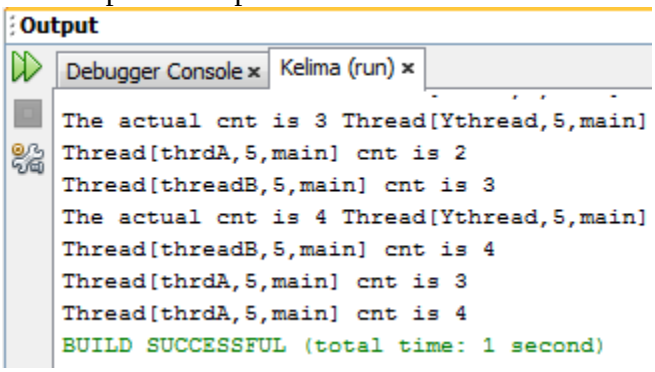
Ketika Anda melaksanakan program threaded, Anda akan menimpa run () method dari kelas Thread dan membangun fungsionalitas program threaded Anda ke run () method.
Tentu saja, run () method dapat meminta metode lainnya.
Anda akan memulai thread berjalan dengan menerapkan start () method pada obyek Thread. Start () akan memanggil metode run () method (dan menangani beberapa tugas-tugas lain yang diperlukan di latar belakang juga). Perhatikan bahwa tidak satu pun dari metode ini mengambil parameter apapun. Anda tidak terbatas pada penggunaan satu run () method dalam sebuah program threaded. Anda dapat menetapkan berbagai kelas dalam program yang baik extendThread atau mengimplementasikan Runnable. Anda dapat instantiate beberapa Thread objek dari masing-masing kelas. Masing-masing dari kelas-kelas ini ditimpa memiliki run () metode yang tidak tergantung pada run () metode dalam kelas-kelas lain.
Upgrade program berikut ini salah satu dari sedikit program-program sebelumnya dengan mendefinisikan dua kelas yang berbeda dari yang objek Thread instantiated. Setiap kelas memiliki run () metode yang berbeda dari run () method di kelas lain (meskipun mereka melakukan hal-hal yang serupa agar lebih mudah untuk melihat hasil). Thread dua objek dari masing-masing kelas instantiated dan mulai. Seperti sebelumnya, mereka menjalankan relatif asynchronously satu sama lain, dan output yang mereka hasilkan adalah bercampur di layar.

```
/*File Thread04.java  Copyright 1997, R.G.Baldwin
Illustrates instantiation and running of threads using the
runnable interface where two different classes are defined
each of which uses the Runnable interface.  The run()
method in one class is different from the run() method in
the other class.

Dua Thread objek instantiated dan mulai untuk masing-masing
thread kelas dua.
```

```
Masing-masing dari thread menghitung dari nol sampai empat dan menampilkan
penghitungan, sleep jumlah acak waktu penting.
Format tampilan berbeda antara dua
thread kelas.

Tested using JDK 1.1.3 under Win95.

Output untuk satu particular run adalah:

Thread[thrdA,5,main] cnt is 0
Thread[threadB,5,main] cnt is 0
The actual cnt is 0 Thread[Xthrd,5,main]
The actual cnt is 0 Thread[Ythread,5,main]
Thread[thrdA,5,main] cnt is 1
Thread[threadB,5,main] cnt is 1
Thread[thrdA,5,main] cnt is 2
The actual cnt is 1 Thread[Ythread,5,main]
The actual cnt is 1 Thread[Xthrd,5,main]
Thread[threadB,5,main] cnt is 2
The actual cnt is 2 Thread[Ythread,5,main]
Thread[thrdA,5,main] cnt is 3
The actual cnt is 3 Thread[Ythread,5,main]
Thread[thrdA,5,main] cnt is 4
The actual cnt is 2 Thread[Xthrd,5,main]
The actual cnt is 4 Thread[Ythread,5,main]
Thread[threadB,5,main] cnt is 3
The actual cnt is 3 Thread[Xthrd,5,main]
The actual cnt is 4 Thread[Xthrd,5,main]
Thread[threadB,5,main] cnt is 4

*********************************************************/

class Thread04{
  static public void main(String[] args){
    //Instantiate dua thread object baru dalam satu tipe
    Thread threadA = new Thread(
                            new OneThreadClass(),"thrdA");
    Thread threadB = new Thread(
                           new OneThreadClass(),"threadB");

    //Instantiate two new thread objects on another type
    Thread Xthread = new Thread(
                         new AnotherThreadClass(),"Xthrd");
    Thread Ythread = new Thread(
                        new AnotherThreadClass(),"Ythread");

    //Memulai running
    threadA.start();
    threadB.start();
    Xthread.start();
    Ythread.start();

    try{//tunda satu detik
      Thread.currentThread().sleep(1000);
    }catch(InterruptedException e){}
  }//akhir main
}//akhir kelas Thread04
```

```
//=========================================================//

class OneThreadClass implements Runnable{
  public void run(){
    for(int cnt = 0; cnt < 5; cnt++){
      System.out.println(Thread.currentThread() +
                                        " cnt is " + cnt);
      try{//tunda jumlah waktu random
        Thread.currentThread().sleep(
                               (int)(Math.random() * 100));
      }catch(InterruptedException e){}
    }//akhiri loop
  }//akhir run
}//akhiri kelas OneThreadClass
//=========================================================//

class AnotherThreadClass implements Runnable{
  public void run(){
    for(int cnt = 0; cnt < 5; cnt++){
      System.out.println("The actual cnt is " + cnt +
                             " " + Thread.currentThread());
      try{//delay a random amount of time
        Thread.currentThread().sleep(
                                (int)(Math.random() * 100));
      }catch(InterruptedException e){}
    }//akhiri loop
  }//akhiri run
}//akhiri kelas AnotherThreadClass
```

Hasil capture compile:



2.6 The Producer/Consumer Model

Hampir setiap buku tentang pemrograman Java menjelaskan penggunaan metode sinkronisasi menggunakan program yang melaksanakan produsen / konsumen model. Dalam model ini, satu thread adalah produsen data sementara thread yang lain adalah konsumen data yang sama. Ada tempat penyimpanan data yang umum di mana menempatkan data produsen dan konsumen mendapatkannya. Dalam rangka untuk mencegah masalah, perlu untuk mencegah konsumen dari berusaha untuk mendapatkan data dari Common area penyimpanan sementara produser adalah meletakkan data ke dalam gudang.

Hal ini juga diperlukan untuk mencegah produsen dari mencoba untuk memasukkan data ke dalam area penyimpanan umum ketika konsumen semakin data dari area penyimpanan. Kami akan mengikuti jejak

orang lain dan menggambarkan thread sinkronisasi dengan memberikan program sampel yang melaksanakan produsen / konsumen model.

Program contoh menggunakan standar FIFO queue sebagai penyimpanan data umum daerah.

Meskipun source code untuk antrian diberikan, kita akan mengasumsikan bahwa secara umum dipahami oleh semua orang, dan tidak akan menghabiskan waktu untuk mendiskusikan hal itu, selain untuk menyatakan bahwa antrian adalah sebuah struktur data yang biasa digunakan dalam pengolahan data dimana data pertama elemen dimasukkan ke dalam struktur adalah elemen pertama dibawa keluar.

Banyak buku pelajaran pemrograman menggambarkan antrean di supermarket sebagai antrian (dengan asumsi bahwa tidak ada yang memotong baris). Orang pertama yang masuk ke dalam garis ini adalah orang pertama untuk diperiksa, orang berikutnya sejalan cek keluar berikutnya, dll . Kami akan instantiate thread produsen yang menempatkan data ke dalam antrian, dan konsumen thread yang mendapatkan data dari antrian, satu byte pada suatu waktu dalam kedua kasus.

Kami akan menyebabkan setiap dari thread untuk sleep () untuk jangka waktu acak antara upaya untuk menempatkan atau memperoleh data. Kami akan menulis QueueManager mengelola kelas yang menempatkan data ke dalam antrian dan mendapatkan data keluar dari antrian dengan cara yang melaksanakan produsen / konsumen model. Anda mungkin berpikir dari QueueManager sebagai polisi lalu lintas mengatur lalu lintas di perempatan jalan yang ramai, berusaha untuk mencegah dua mobil dari pertemuan di tengah persimpangan.

Pemrograman setara dengan dua mobil rapat di tengah persimpangan seringkali hasil dalam kondisi biasa disebut sebagai kondisi ras. Sebagian besar buku Jawa berisi program contoh yang menggambarkan kondisi ras. Kau akan baik-disarankan untuk meninjau salah satu dari mereka. The QueueManager akan menggunakan menunggu () dan notify () metode untuk mencegah tabrakan. Ketika antrian penuh, produsen akan diminta untuk menunggu () sampai konsuken diberitahukan oleh ruang yang tersedia dalam antrian. Ketika antrian kosong, konsumen akan diminta untuk menunggu () sampai diberitahukan oleh produser bahwa data baru tersedia dalam antrian.

Setiap kali produsen menempatkan satu byte dalam antrian, itu akan memberitahu () konsumen ketersediaan data baru, untuk berjaga-jaga konsumen di tunggu () negara karena antrian yang kosong. Demikian pula, setiap kali mendapat konsumen byte dari antrian, itu akan memberitahu () produsen ruang yang sekarang tersedia dalam antrian untuk berjaga-jaga produsen dalam menunggu () negara karena antrian penuh. Meminjam yang memberitahukan () method tidak memiliki pengaruh jika tidak ada thread dalam menunggu () negara. Program ini diperbolehkan untuk berjalan selama satu detik dan kemudian berakhir. Program tidak menampilkan data aktual mengalir ke dalam dan keluar dari antrian. Itu akan membutuhkan beberapa halaman untuk bereproduksi.

Sebaliknya, program akan menampilkan pesan setiap kali produsen menemukan antrian penuh, atau konsumen menemukan antrian kosong. Karena penundaan acak antara mendapatkan dan menempatkan upaya, program menghasilkan output yang berbeda setiap kali dijalankan. Salah satu output seperti ditunjukkan dalam komentar pada awal program. Perhatikan bahwa antrian itu dibuat sangat kecil agar dapat lebih baik menggambarkan isu seputar antrian penuh dan kosong. Dalam situasi pemrograman nyata, antrean mungkin akan dibuat cukup besar dalam upaya untuk mencegah terjadinya antrian penuh atau kosong antrian. Sampai titik, dengan masalah semacam ini, kemungkinan bertemu penuh atau kosong antrian berkurang dengan ukuran keseluruhan antrian. Program sampel kami berikut. Program ini menggambarkan thread sinkronisasi yang merupakan konsep yang sangat penting.

Kecuali untuk kode untuk kelas Antrian, instruktur Anda akan menjelaskan pengoperasian program. Sekali lagi, diasumsikan bahwa semua siswa di kelas mengerti pengoperasian standar antrian FIFO.

```
/*File Synch01.java  Copyright 1997, R.G.Baldwin
This program illustrates the Producer/Consumer model using
wait() and notify()

Tested using JDK 1.1.3 under Win95.

Output untuk satu particular adalah:
Queue empty, waiting
Queue empty, waiting
Queue empty, waiting
Queue empty, waiting
Queue empty, waiting
Queue empty, waiting
Queue full, waiting
Queue full, waiting
Queue full, waiting
Queue empty, waiting
Queue empty, waiting
Queue empty, waiting
Queue empty, waiting
Queue empty, waiting
Queue empty, waiting
Queue empty, waiting
Terminating Consumer run method
Terminating Producer run method


**********************************************************/

class Synch01{
  //Instantiate sebuah kelas obyek bernama QueueManager yang
  // akan mengatur producer/consumer model.
  static QueueManager queueManager = new QueueManager();

  //digunakan untuk mengetahui thread yang terminate
  static boolean running = true;

  public static void main(String[] args){
    //instantiate dan memulai 2 thread
    Thread producer = new Producer();
    Thread consumer = new Consumer();
    producer.start();
    consumer.start();

    try{ //tunda 2 detik
      Thread.currentThread().sleep(2000);
    }catch(InterruptedException e){};

    running = false;//signal the threads to terminate
  }//akhiri main

}//akhiri kelas Synch01
//=========================================================//

class Producer extends Thread { //thread producer
  public void run() { //jalankan method untuk Producer thread
    byte byteToStore; //gunakan data simpanan untuk antrian
```

```
    //loop sampai berjalan salah
    while (Synch01.running){
      //dapatkan data byte
      byteToStore = (byte)(Math.random()*128);
      //memanggil method sinkronisasin untuk mengambil byte dalam antrian
Synch01.queueManager.putByteInQueue(byteToStore);
      //tunda waktu random
      try{
        Thread.currentThread().sleep(
                                (int)(Math.random()*100));
      }catch(InterruptedException e){};
    }//dan while statement
    System.out.println("Terminating Producer run method");
  }//akhir run method
}//akhir kelas producer
//=========================================================//

class Consumer extends Thread { //consumer thread
  public void run() { //jalankan method untuk Consumer thread

    // gunakan data yang tersimpan dibaca dari antrian
    byte dataFromQueue;

    //loop sampai berjalan salah
    while (Synch01.running) {
      //Invoke the synchronized method to get a byte
      // dari antrian
      dataFromQueue =
                  Synch01.queueManager.getByteFromQueue();

      //tunda jumlah waktu random
      try{
        Thread.currentThread().sleep(
                                (int)(Math.random()*100));
      }catch(InterruptedException e){};
    }//akhir while statement
    System.out.println("Terminating Consumer run method");
  }//akhirjalankan method
}//akhiri kelas consumer
//=========================================================//

//Kealas ini implementasi the Producer/Consumer model
// pengaturan antrian dengan resource yang disalurkan.
class QueueManager{
  Queue queue;
//---------------------------------------------------------//

  QueueManager(){//constructor
    queue = new Queue();//instantiate a queue object
  }//end constructor
//---------------------------------------------------------//

  synchronized void putByteInQueue(byte incomingByte){
    //This synchronized method places a byte in the queue
    // If the queue is full, wait(). If still full when
    // wait() terminates, wait again. Called by the
    // producer thread to put a byte in the queue.
    try{
```

```
      while(queue.isFull()){
        System.out.println("Queue full, waiting");
        wait();
      }//akhiri while loop
    }catch (InterruptedException E){
      System.out.println("InterruptedException: " + E);
    }//akhiri catch block

    //letakkan data byte dalam antrian
    queue.enQueue(incomingByte);

    //wake up getByteFromQueue() if it has invoked wait().
    notify();
  }//end method putByteInQueue()
  //--------------------------------------------------------//

  public synchronized byte getByteFromQueue(){
    //This synchronized method removes a byte from the
    // queue. If the queue is empty, wait(). If still
    // empty when wait() terminates, wait again. Called by
    // consumer thread to get a byte from the queue
    try{
      while(queue.isEmpty()){
        System.out.println("Queue empty, waiting");
        wait();
      }// akhir while
    }catch (InterruptedException E){
      System.out.println("InterruptedException: " + E);
    }//akhir catch block

    //dapatkan byte data dari antrian
    byte data = queue.deQueue();

    //wake up putByteInQueue() if it has invoked wait().
    notify();
    return data;
  }//end getByteFromQueue()

}//akhiri kelas QueueManager
//============================================================//

//Ini adalah standar FIFO kelas antrian
class Queue{
    //constant defining maximum queue size
    static final int MAXQUEUE = 4;
    byte[] queue = new byte[MAXQUEUE];
    int front, rear;

    Queue(){//constructor
      front = rear = 0;
    }//akhir constructor

    void enQueue(byte item){
      queue[rear] = item;
      rear = next(rear);
    }//akhir method enQueue

    byte deQueue(){
```

```
      byte temp = queue[front];
      front = next(front);
      return temp;
    }//akhir method deQueue

    boolean isEmpty(){
      return front == rear;
    }//akhir isEmpty

    boolean isFull(){
      return (next(rear) == front);
    }//akhir isFull

    int next(int index){
      return (index+1 < MAXQUEUE ? index+1 : 0);
    }//akhir next

}//akhir kelas Queue
//=========================================================//
```

Hasil capture compile:

```
Queue empty, waiting
Queue empty, waiting
Queue empty, waiting
Queue empty, waiting
Queue empty, waiting
Queue empty, waiting
Queue full, waiting
Queue full, waiting
Queue full, waiting
Queue empty, waiting
Queue empty, waiting
Queue empty, waiting
Queue empty, waiting
Queue empty, waiting
Queue empty, waiting
Queue empty, waiting
Terminating Consumer run method
Terminating Producer run method
```

REFERENCES

[1] G Baldwin, Richard.1997. **"Threads of Control".**http://www.dickbaldwin.com/java/Java058.htm