BAB II
MULTITHREADING

2.1. Mengenal dan Menggunakan *Thread*

Dalam pemrograman Java, sebuah *thread* merupakan jalur atau utas (*path*) yang dapat diikuti saat menjalankan sebuah program. Sebuah `thread` dapat memiliki prioritas masing-masing, mulai dari 1 (*minimum priority*), 5 (*normal priority*), sampai dengan 10 (*maximum priority*). Terdapat dua macam *thread* dalam bahasa pemrograman Java, yaitu:

1) `Daemon`, merupakan *thread* (dengan prioritas rendah) yang dibuat oleh *Java Virtual Machine* (JVM) untuk melakukan pekerjaan tertentu, seperti membuat objek, memanggil *method*, dan lain sebagainya. *Thread* ini berjalan di latar belakang (*run in background*, seperti `carbage collection`) selama program masih berjalan. Ada beberapa properti dalam *thread* tipe ini, diantaranya:
   - ✓ *Thread* ini tidak dapat mencegah JVM keluar ketika semua *thread* lain milik pengguna telah menyelesaikan tugas/eksekusi mereka.
   - ✓ JVM tetap akan menghentikan dirinya sendiri ketika semua *thread* milik pengguna telah menyelesaikan tugasnya.
   - ✓ Jika JVM menemukan masih ada `daemon thread` yang masih berjalan, maka JVM akan menghentikannya kemudian mematikan dirinya sendiri.
2) `Non-Daemon`, merupakan suatu *thread* yang ketika telah selesai mengerjakan tugasnya (*exit*), maka JVM juga dapat keluar atau mematikan dirinya sendiri (exit). Jadi, ketika semua *thread* non-daemon selesai, maka program diakhiri. Sebaliknya, jika ada *thread* non-daemon yang masih berjalan, maka program tidak akan berhenti.

Saat JVM dijalankan, biasanya terdapat sebuah *non-daemon thread* tunggal (yang biasanya menjalankan *main method*). JVM akan terus menjalankan semua *thread* sampai salah satu dari hal berikut dipenuhi/terjadi:

1. Metode exit (*exit method*) pada kelas `Runtime` dipanggil, dan *security manager* telah mengizinkan operasi keluar untuk dilakukan.
2. Semua *non-daemon thread* telah mati/dihentikan, baik dengan kembali dari pemanggilan metode `run` maupun dengan cara melemparkan pengecualian (*throwing an exception*) yang merambat di luar metode `run`.

Selain *main thread*, secara default JVM akan menjalankan beberapa *thread* pada setiap program Java, diantaranya:

1. `DestroyJavaVM` → *thread* yang bertugas untuk menutup/menghentikan JVM (sampai semua *non-daemon thread* dijalankan hingga selesai).
2. `Signal Dispatcher` → merupakan *thread* yang menangani sinyal asli yang dikirim oleh OS ke JVM.
3. `Finalizer` → merupakan thread yang bertugas menarik berbagai objek dari antrian finalisasi.
4. `Reference Handler` → merupakan *thread* yang memiliki prioritas tinggi yang bertugas untuk meminta *reference* yang tertunda.

Java menyediakan dukungan *built-in* untuk pemrograman *multithreading*. Pada suatu kondisi, kita menginginkan aplikasi yang kita kembangkan dapat menjalankan komputasi atau *event* tertentu secara bersamaan. Contoh kasus, pada saat aplikasi menjalankan pekerjaan (misal, meminta dan memproses beberapa paket data yang diminta oleh user dari sebuah server), dan pada saat yang bersamaan, aplikasi menjalankan *event* tertentu pada *Graphical User Interface* (GUI) untuk berinteraksi dengan user. Untuk mengakomodir kasus tersebut, maka kita dapat menggunakan beberapa *thread* secara bersamaan (*multithreading*).

Di Java, untuk membuat *thread* dapat dilakukan dengan cara mengimplementasikan (*implements*) sebuah *interface* dan memperluas (*extends*) kelas. Setiap *thread* Java, dibuat dan dikendalikan oleh kelas *Thread* yang ada pada paket `java.lang`. Di bawah ini merupakan contoh penggunaan *thread* untuk menampilkan tanggal dan waktu pada saat program dijalankan.

```
import java.text.SimpleDateFormat;
import java.util.Calendar;

public class MyThread implements Runnable{
    @Override
    public void run() {
        try {
            while (true) {
                Calendar c = Calendar.getInstance();
                SimpleDateFormat df = new
                   SimpleDateFormat("dd-MM-yyyy HH:mm:ss");
                String tanggal = df.format(c.getTime());
                System.out.println("Jalan pada "+tanggal);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
        }
    }
}
```

Untuk menggunakan *thread* di atas pada *entry point*, kita cukup melakukan instansiasi kelas `MyThread` pada objek `Runnable`. Selanjutnya, objek *runnable* kita

masukkan ke dalam objek `Thread`. Untuk menjalankan thread ini, dapat dilakukan dengan memanggil *method* `start()`.

Kelas untuk menjalankan *thread*:

```
public class MainThread {
    public static void main(String[] args) {
        Runnable obj = new MyThread();
        Thread t = new Thread(obj);
        t.start();
    }
}
```

2.2. Mengenal dan Menggunakan Objek *Thread*

Salah Dalam Java, objek *thread* berada pada paket `java.lang`. `Thread` juga dapat digunakan untuk membantu aplikasi dalam menjalankan *event* atau perintah dibelakang layar (*run in background*). Dalam objek *thread*, terdapat beberapa *method* yang dapat digunakan untuk keperluan *multit-hreading*, diantaranya adalah sebagai berikut:

Tabel 2.1 Method dalam objek *Thread*

Nama Method	Kegunaan
getId()	Mendapatkan informasi ID *thread*
getName()	Memperoleh informasi nama *thread*
setName(String name)	Mengisi/mengubah nama *thread*
interrupt()	Melakukan interupsi pada *thread*
isAlive()	Mengecek status *thread* (hidup/tidak)
isDaemon()	Mengecek status *thread* (daemon/tidak)
setDaemon(Boolean b)	Mengubah status *thread* (daemon/tidak)
isInterrupted()	Mengecek status *thread* (diinterupsi/tidak)
join()	Menunggu *thread* untuk mati
setPriority(int newPriority)	Mengubah prioritas *thread*
getPriority()	Mengambil informasi prioritas *thread*
start()	Menjalankan *thread*

2.2.1. Kontruksi Thread

Dalam melakukan kontruksi objek `Thread`, kita dapat langsung meletakkan objek target yang akan dieksekusi. Objek target harus mengimplementasikan `Runnable`, contohnya seperti kode program di bawah ini:

```
public class MyTargetThread implements Runnable{
    @Override
```

```
    public void run(){
        //implementasi abstract method "run"
    }
}
```

Ketika objek `Thread` target dikontruksi, kita dapat langsung memasukkan ke dalam objek `Thread`, contoh ilustrasinya sebagai berikut:

```
Thread thread1 = new Thread(new MyTargetThread());
```

Kita juga dapat memberikan nama *thread* ketika melakukan kontruksi. Contohnya kita melakukan instansiasi objek *thread* dengan nama "`Thread1`" sebagai berikut:

```
Thread thread1 = new Thread(new MyTargetThread(), "Thread1");
```

2.2.2. Menjalankan Thread

Untuk menjalankan sebuah *thread*, kita dapat menggunakan *method* `start()` yang ada di dalam objek `Thread`. Contoh kode untuk menjalankan *thread* adalah sebagai berikut:

```
Thread thread1 = new Thread(new MyTargetThread(), "Thread1");
thread1.start();
```

Pada contoh kode di atas, kita membuat sebuah `Thread` dengan nama "`Thread1`", selanjutnya kita jalankan *thread* tersebut dengan menggunakan *method* `start()`.

Catatan: Kelas yang dapat dijadikan target adalah kelas yang mengimplementasikan *interface* `Runnable`, atau kelas yang mewarisi kelas yang mengimplementasikan `Runnable`, seperti kelas **`SwingWorker`**. Contoh:

```
import javax.swing.SwingWorker;
public class ContohThread {
    public static void main(String[] args) {
        Thread thread2 = new Thread(new MyClass(), "Thread2");
        thread2.start();
    }

    static class MyClass extends SwingWorker<Object, Object> {
        @Override
        protected Object doInBackground() throws Exception {
            System.out.println("Run in background");
            return "";
        }
    }
}
```

2.2.3. Menghentikan Thread

Perlu kita ketahui, bahwa sebuah thread itu berjalan sendiri tanpa harus melakukan interferensi dan untuk menghentikan thread dapat menggunakan 2 cara berikut:

1. Memaksa *thread* untuk dihentikan melalui *thread* utama atau program utama.

   Penghentian paksa dapat dilakukan dengan menggunakan method `interrupt()` dari objek `Thread`. Ketika melakukan penghentian *thread* secara paksa, jangan lupa untuk menggunakan blok *try-catch* untuk *error handling*.

Thread: MyThread

```
import java.text.SimpleDateFormat;
import java.util.Calendar;

public class MyThread implements Runnable{

    @Override
    public void run() {
        try {
            while (true) {
                Calendar c = Calendar.getInstance();
                SimpleDateFormat df = new
                SimpleDateFormat("dd-MM-yyyy HH:mm:ss");
                String tanggal = df.format(c.getTime());
                System.out.println("Jalan pada "+tanggal);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
        }
    }
}
```

Menjalankan dan Menghentikan Thread MyThread:

```
public class MainThread {
    public static void main(String[] args) {
        try {
            Runnable obj = new MyThread();
            //instansiasi objek Thread
            Thread t = new Thread(obj);
            //menjalankan thread
            t.start();
            //memberikan jeda 2 detik
            Thread.sleep(2000);
           //menghentikan thread dengan paksa
            t.interrupt();
        } catch (InterruptedException e) {
             System.err.println("Error: "+e.getMessage());
        }
    }
}
```

2. Menunggu *thread* tersebut untuk mematikan diri sendiri.

Cara ini dapat dilakukan dengan cara memodifikasi kode program agar dapat memberikan perintah pada objek *thread* untuk menghentikan dirinya sendiri. Contoh kode program adalah sebagai berikut:

Thread: MyThread

```
import java.text.SimpleDateFormat;
import java.util.Calendar;

public class MyThread implements Runnable{
    private boolean isFinish = false;

    public void setIsFinish(boolean isFinish) {
        this.isFinish = isFinish;
    }

   @Override
    public void run() {
        try {
            while (!isFinish) {
                Calendar c = Calendar.getInstance();
                SimpleDateFormat df = new
                SimpleDateFormat("dd-MM-yyyy HH:mm:ss");
                String tanggal = df.format(c.getTime());
                System.out.println("Jalan pada "+tanggal);
                Thread.sleep(1000);
            }
        } catch (InterruptedException e) {
        }
    }
}
```

Menjalankan dan Menghentikan Thread MyThread:

```
public class MainThread {
    public static void main(String[] args) {
        try {
            MyThread obj = new MyThread();
            //instansiasi objek Thread
            Thread t = new Thread(obj);
            //menjalankan thread
            t.start();
            //memberikan jeda 2 detik
            Thread.sleep(2000);
            //menghentikan thread dengan paksa
            t.setIsFinish(true);
        } catch (InterruptedException e) {
             System.err.println("Error: "+e.getMessage());
        }
    }
}
```

2.2.4. Informasi Thread

Saat kita menggunakan *thread*, maka kita perlu melakukan monitoring untuk mengetahui informasi yang ada di dalamnya saat sedang berjalan, seperti: apakah sebuah *thread* masih berjalan, nama *thread* saat ini, id *thread* saat ini, dan sebagainya. Berikut merupakan beberapa contoh penggunaan beberapa method untuk monitoring informasi dalam sebuah thread saat berjalan.

Contoh:

```
public class MainThread {
    public static void main(String[] args) {
        try {
            MyThread obj = new MyThread();

            Thread t = new Thread(obj);
            t.setName("Thread1");
            t.start();

            System.out.println("ID Thread: "+ t.getId());
            System.out.println("Nama Thread: "+ t.getName());
            System.out.println("Interupsi : "+ t.isInterrupted());
            System.out.println("Prioritas : "+ t.getPriority());
            t.setName("Thread2");
            System.out.println("Nama Thread: "+ t.getName());
            Thread.sleep(2000);

            obj.setIsFinish(true);
        } catch (InterruptedException e) {
            System.err.println("Error: "+e.getMessage());
        }
    }
}
```

2.3. Mengenal dan Menggunakan Objek Timer

Pada suatu kondisi, dalam aplikasi yang kita kembangkan ingin kita tambahkan suatu aksi yang dilakukan secara terus-menerus pada suatu periodik tertentu tanpa mengganggu aplikasi utama. Hal ini dapat dilakukan dengan memanfaatkan objek `Timer` yang berada paket `java.util`. Untuk menjalankan aksi yang akan dieksekusi oleh objek `Timer`, kita harus memperluas (*extends*) objek `TimerTask` pada kelas yang kita buat. Selain itu, kita juga harus melakukan *override method* `run()` kerena kelas `TimerTask` juga mengimplementasikan *interface* `Runnable`. Contoh penggunaan objek `Timer` secara lengkap seperti kode di bawah ini:

```
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Timer;
import java.util.TimerTask;

/**
 *
 * @author nishom
 */

public class MyTask extends TimerTask{

    @Override
    public void run() {
        Calendar c = Calendar.getInstance();
        SimpleDateFormat df = new
        SimpleDateFormat("dd-MM-yyyy HH:mm:ss");
        String waktu = df.format(c.getTime());
        System.out.println(waktu);
    }
    public static void main(String[] args) {
         Timer timer = new Timer();
         timer.schedule(new MyTask(), 0, 1000);
    }
}
```

2.4. Sinkronisasi *Thread*

Setiap *thread* yang kita buat, akan berjalan secara otomatis tanpa adanya interferensi dari kita. Pada kondisi tertentu, *thread* yang kita buat mungkin akan mengakses suatu *resource* seperti data variabel atau *file*. Jika hanya terdapat satu *thread*, maka hal ini tidak akan menjadi masalah. Tetapi, jika *thread*-nya lebih dari satu (*multi-threading*) maka akan menimbulkan masalah karena mengakses *resource* secara bersamaan biasanya akan menimbulkan *error*. Oleh karena itu, kita memerlukan sinkronisasi antar *thread*. Manfaat sinkroniasi adalah untuk mencegah interferensi *thread* dan masalah konsistensi.

Pada Java, cara untuk melakukan sinkronisasi *thread* dapat dilakukan dengan menggunakan *keyword* `synchronized`. Pada Java, terdapat 3 jenis sinkronisasi *thread* yaitu: Sinkronisasi *method*, *block*, dan *static*.

Contoh Thread tidak Tersinkronisasi

```
package thread.notsync;

public class WithoutSynchronization {
    public static void main(String args[]){
        Table obj = new Table();//hanya satu objek
        MyThread1 t1 = new MyThread1(obj);
        MyThread2 t2 = new MyThread2(obj);
        t1.start();
        t2.start();
    }
}
```

```
class Table{
    void printTable(int n){//method tidak tersinkronisasi
        for(int i=1; i<=5; i++){
          System.out.println(n*i);
          try{
               Thread.sleep(400);
          }catch(InterruptedException e){
              System.out.println(e);
          }
        }
    }
}

class MyThread1 extends Thread{

     Table t;
    MyThread1(Table t){
        this.t=t;
    }
   @Override
    public void run(){
        t.printTable(5);
    }
}

class MyThread2 extends Thread{
    Table t;

    MyThread2(Table t){
        this.t=t;
    }
     @Override
    public void run(){
        t.printTable(100);
    }
}
```

Contoh Sinkronisasi Method

```
package thread.syncmethod;

public class SynchronizedMethod{
    public static void main(String args[]){
         Table obj = new Table();//hanya satu objek
         MyThread1 t1 = new MyThread1(obj);
         MyThread2 t2 = new MyThread2(obj);
         t1.start();
         t2.start();
    }
}

class Table{
    synchronized void printTable(int n){
        for(int i=1; i<=5; i++){
            System.out.println(n*i);
            try{
                Thread.sleep(400);
            }catch(InterruptedException e){
                System.out.println(e);
            }
        }
    }
}
```

```
class MyThread1 extends Thread{
    Table t;
    MyThread1(Table t){
        this.t=t;
    }
    @Override
    public void run(){
        t.printTable(5);
    }
}

class MyThread2 extends Thread{
    Table t;
    MyThread2(Table t){
        this.t=t;
    }
    @Override
    public void run(){
        t.printTable(100);
    }
}
```

Jika kita melakukan sinkronisasi *method* dengan menggunakan kelas *annonymous*, maka akan menggunakan kode yang lebih sedikit.

Contoh Sinkronisasi Method dengan Kelas Annonymous

```
package thread.syncmethodannonymousclass;

/**
 *
 * @author nishom
 */

public class SynchronizedMethodAnnonymousClass{
    public static void main(String args[]){
    final Table obj = new Table();//hanya satu objek

    Thread t1 = new Thread(){

        @Override
        public void run(){
            obj.printTable(5);
        }
    };

    Thread t2 = new Thread(){
        @Override
        public void run(){
            obj.printTable(100);
        }
    };

    t1.start();
    t2.start();
    }
}
```

```
class Table{
    synchronized void printTable(int n){
       for(int i=1; i<=5; i++){
         System.out.println(n*i);
         try{
            Thread.sleep(400);
         }catch(InterruptedException e){
             System.out.println(e);
         }
       }
    }
}
```

Contoh Sinkronisasi Block

```
package thread.syncblock;

public class SynchronizedBlock{
    public static void main(String args[]){
        Table obj = new Table();//only one object
        MyThread1 t1 = new MyThread1(obj);
        MyThread2 t2 = new MyThread2(obj);
        t1.start();
        t2.start();
    }
}

class Table{
    void printTable(int n){
        synchronized(this){//sinkronisasi block
            for(int i=1; i<=5; i++){
                System.out.println(n*i);
                try{
                    Thread.sleep(400);
                }catch(InterruptedException e){
                    System.out.println(e);
                }
            }
        }
    }
}

class MyThread1 extends Thread{
    Table t;
    MyThread1(Table t){
        this.t=t;
    }
    @Override
    public void run(){
        t.printTable(5);
    }
}
class MyThread2 extends Thread{
    Table t;
    MyThread2(Table t){
        this.t=t;
    }
    @Override
    public void run(){
        t.printTable(100);
    }
}
```

Contoh Sinkronisasi Static

```
package thread.syncstatic;

/**
 *
 * @author nishom
 */
public class SynchronizedStatic{
    public static void main(String t[]){
        MyThread1 t1 = new MyThread1();
        MyThread2 t2 = new MyThread2();
        MyThread3 t3 = new MyThread3();
        t1.start();
        t2.start();
        t3.start();
    }
}

class Table{
    synchronized static void printTable(int n){
        for(int i=1; i<=10; i++){
            System.out.println(n*i);
            try{
                Thread.sleep(400);
            }catch(InterruptedException e){
              //error handling
            }
        }
    }
}

class MyThread1 extends Thread{

    @Override
    public void run(){
        Table.printTable(1);
    }
}

class MyThread2 extends Thread{

   @Override
    public void run(){
        Table.printTable(10);
    }
}

class MyThread3 extends Thread{

   @Override
    public void run(){
        Table.printTable(100);
    }
}
```

2.5. Praktik

Pada bab ini, kita akan praktik membuat sebuah aplikasi sederhana yang berfungsi untuk mendapatkan informasi kurs atau nilai tukar (*exchange rate*) mata uang. Dalam praktikum ini juga diimplementasikan kelas `Timer` dan `TimerTask` untuk menampilkan waktu/jam serta `SwingWorker` untuk menjalankan *thread* di belakang layar (*run in background*) saat menunggu respon *exchange rate* dari provider https://api.exchangeratesapi.io/. Pada saat bersamaan, aplikasi juga menjalankan *thread* loading yang berfungsi sebagai notifikasi kepada pengguna aplikasi bahwa informasi *exchange rate* yang diminta masih diproses oleh *thread* lainnya.

2.5.1. Membuat Projek Baru

Untuk membuat *project* baru di Netbeans ikuti langkah-langkah berikut:

1. Klik menu **File → New Project**, selanjutnya akan terbuka dialog **New Project**
2. Pada dialog **New Project**, untuk *Categories* pilih **Java**, dan untuk jenis **Projects** pilih **Java Application →** klik **Next** untuk melanjutkan ke tahapan berikutnya.
3. Pada dialog **New Java Application**, isi *field* dan pilihan yang ada sebagai berikut:
   A. Project Name : **`P02-Multithreading`**
   B. Project Location : Lokasi opsional, sesuai dengan keinginan anda
   C. Project Folder : Otomatis oleh Netbeans IDE
   D. Create Main Class : **Hilangkan centang pada checkbox**
4. Klik **Finish**

2.5.2. Membuat Paket Baru

Untuk membuat paket baru dalam sebuah *project*, ikuti langkah-langkah berikut:

1. Klik kanan pada Node **Source Package → New → Java Package**. Selanjutnya Netbeans akan membuka dialog **New Java Package**.
2. Pada dialog **New Java Package**, isikan "***thread***" pada kotak isian nama paket (*package name*)
3. Klik **Finish**.

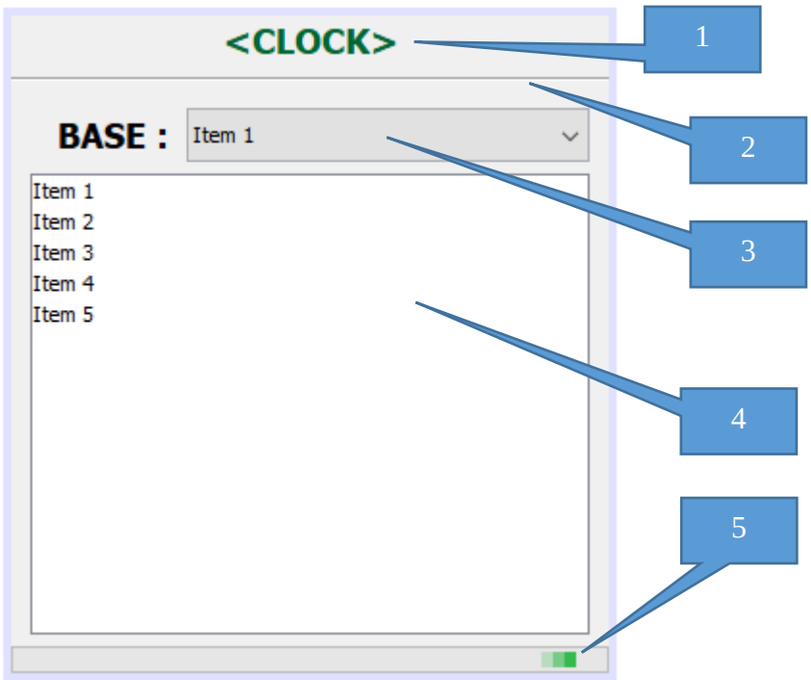
2.5.3. Membuat Form Baru

Untuk membuat *form* baru dalam sebuah *project*, ikuti langkah-langkah berikut:

1. Klik kanan pada *package* **thread → New → JFrame Form**. Selanjutnya Netbeans akan menampilkan dialog untuk membuat form baru (*dialog* **New JFrame Form**).
2. Pada field **Class Name**, isikan **`ExchangeRate`**.
3. Klik **Finish**

2.5.4. Mendesain Form

Setelah *form* selesai dibuat, berikutnya desainlah *form* tersebut sehingga menjadi seperti gambar berikut:



Gambar 2.1 Desain *form* (*Multithreading*)

Keterangan:

1. Nama palette : JLabel
   Nama variabel : **lblClock**
2. Nama palette : JSeparator
   Nama variabel : **pembatas**
3. Nama palette : JComboBox
   Nama variabel : **cmbBase**
4. Nama palette : JList
   Nama variabel : **listRates**
5. Nama palette : JProgressBar
   Nama variabel : **loading**

Catatan: Buatlah agar desain tampilan menjadi *responsive*

2.5.5. Membuat Kelas CurrencyConversionResponse

Kelas ini digunakan untuk menampung data yang dihasilkan dari respon provider https://api.exchangeratesapi.io/. Untuk membuat kelas tersebut, ikuti langkah-langkah di bawah ini:

1. Klik kanan pada paket ***thread*** yang telah anda buat sebelumnya.
2. Pilih **new → Java Class**. Selanjutnya Netbeans akan menampilkan dialog untuk membuat kelas baru (dialog **New Java Class**).
3. Pada field **Class Name**, isikan **CurrencyConversionResponse**.
4. Klik **Finish**.

Selanjutnya, ubah kode program sehinggan menjadi seperti di bawah ini:

```
import java.util.Map;
import java.util.TreeMap;

/**
 *
 * @author nishom
 */
public class CurrencyConversionResponse {
    private String base;
    private String date;
    private Map<String, String> rates = new TreeMap<>();

    public Map<String, String> getRates() {
        return rates;
    }
    public void setRates(Map<String, String> rates) {
        this.rates = rates;
    }
    public String getBase() {
        return base;
    }
    public void setBase(String base) {
        this.base = base;
    }
    public String getDate() {
        return date;
    }
    public void setDate(String date) {
        this.date = date;
    }
}
```

2.5.6. Menambahkan *Method, Inner Class,* dan *Event*

Setelah *form* dan kelas yang dibutuhkan selesai dibuat, selanjutnya adalah membuat objek, *method*, dan *event* yang akan digunakan dalam aplikasi. Adapun langkah-langkahnya adalah sebagai berikut:

1. Tambahkan *library* yang dibutuhkan ke dalam *project*. *Library* dapat anda *download* di sini: https://drive.google.com/open?id=1-l8WfpDBzuNlg_we9reXEz2pA19LD1bz.
2. Setelah anda menambahkan library, selanjutnya buka jendela ***Source*** pada kelas **ExchangeRate.java**
3. Untuk memastikan anda tidak salah *import* objek, *import* beberapa *class* atau *interface* yang akan digunakan dalam *project*.

```
import com.google.gson.Gson;
import java.io.IOException;
import java.io.InputStream;
import java.net.HttpURLConnection;
```

```
import java.net.MalformedURLException;
import java.net.URL;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;
import java.util.Timer;
import java.util.TimerTask;
import javax.swing.DefaultListModel;
import javax.swing.SwingWorker;
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;
import javax.swing.plaf.nimbus.NimbusLookAndFeel;
```

4. Deklarasikan beberapa objek berikut (baris 32-40) pada kelas **ExchangeRate,** Sehingga kode program terlihat seperti di bawah ini:

```
String API_PROVDIER =
“https://api.apilayer.com/exchangerates_data”;
String[]  code  =  {"MXN","AUD","HKD","RON","HRK","CHF","IDR",
"CAD","USD","JPY","BRL","PHP","CZK","NOK","INR","PLN","ISK","M
YR","ZAR","ILS","GBP","SGD","HUF","EUR","CNY","TRY","SEK","RUB
","NZD","KRW","THB","BGN","DKK"};

DefaultListModel<String> model = new DefaultListModel<>();
List<String> rates = new ArrayList<>();
```

5. Tambahkan beberapa *method* di bawah ini ke dalam kelas **ExchangeRate**.

```
private void initBase(){
    listRates.setModel(model);
    cmbBase.removeAllItems();
    cmbBase.addItem("--Select Base--");
    for (String str : code) {
        cmbBase.addItem(str);
    }
}


private void startClock(){
      Timer t = new Timer("Thread-Jam");
      t.schedule(new TimerTask() {
          @Override
          public void run() {
              Calendar c = Calendar.getInstance();
              SimpleDateFormat df = new
              SimpleDateFormat("HH:mm:ss");
              String waktu = df.format(c.getTime());
              lblClock.setText(waktu);
          }
       }, 0, 1000);
  }
  private void loading(boolean b){
      loading.setVisible(b);
  }

  private void addResponseToList(String base){
      CurrencyConversionResponse response =
       getResponse(API_PROVDIER+””
       + "/latest?"
       + "apikey=YOUR_API_KEY"
       + "&base=" + base + ""
       );
```

```
            if(response != null) {
                rates.clear();
                for (String str : code) {
                    String rate = response.getRates().get(str);
                    String item = str + "\t: "+ rate;
                    rates.add(item);
                }
            }
        }
        private static CurrencyConversionResponse
             getResponse(String strUrl) {
             CurrencyConversionResponse response = null;
             Gson gson = new Gson();
             StringBuilder sb = new StringBuilder();
             if(strUrl == null || strUrl.isEmpty()) {
                 System.out.println("Application Error");
                 return null;
             }

             URL url;
             try {
                 url = new URL(strUrl);
                 HttpURLConnection connection =
                (HttpURLConnection) url.openConnection();

                 try (InputStream stream =
                     connection.getInputStream()) {
                     int data = stream.read();
                     while (data != -1) {
                         sb.append((char) data);
                         data = stream.read();
                     }
                 }
                 response = gson.fromJson(sb.toString(),
                  CurrencyConversionResponse.class);
             } catch (MalformedURLException e) {
                 System.out.println(e.getMessage());
             } catch (IOException e) {
                 System.out.println(e.getMessage());
             }

             return response;
        }
```

6. Membuat kelas **Exchange** di dalam kelas **ExchangeRate** (*inner class*). Kelas **Exchange** ini merupakan kelas yang mewarisi kelas SwingWorker (kelas SwingWorker mengimplementasikan kelas **RunnableFuture**, dan kelas **RunnableFuture** mengimplementasikan kelas **Runnable**). Kelas **Exchange** ini kita digunakan untuk menjalankan 1 (satu) *thread* untuk mengambil dan menampilkan nilai tukar mata uang yang telah kita ambil dari https://api.exchangeratesapi.io ke dalam sebuah list (*run in background*).

```
class Exchange extends SwingWorker<Object, Object> {
        public Exchange(String name){
            setName(name);
            System.out.println(name+" => Dijalankan!");
```

```
        }
        @Override
        protected Object doInBackground() throws Exception {
            String base =
             cmbBase.getSelectedItem().toString();
            addResponseToList(base);
            return 0;
        }

        @Override
        protected void done(){
            model.clear();
            rates.forEach((rate) -> {
                model.addElement(rate);
            });
            loading(false);
             System.out.println(getName()+" => Dihentikan!");
        }
    }
```

7. Modifikasi kode pada konstruktor sehingga menjadi seperti di bawah ini:

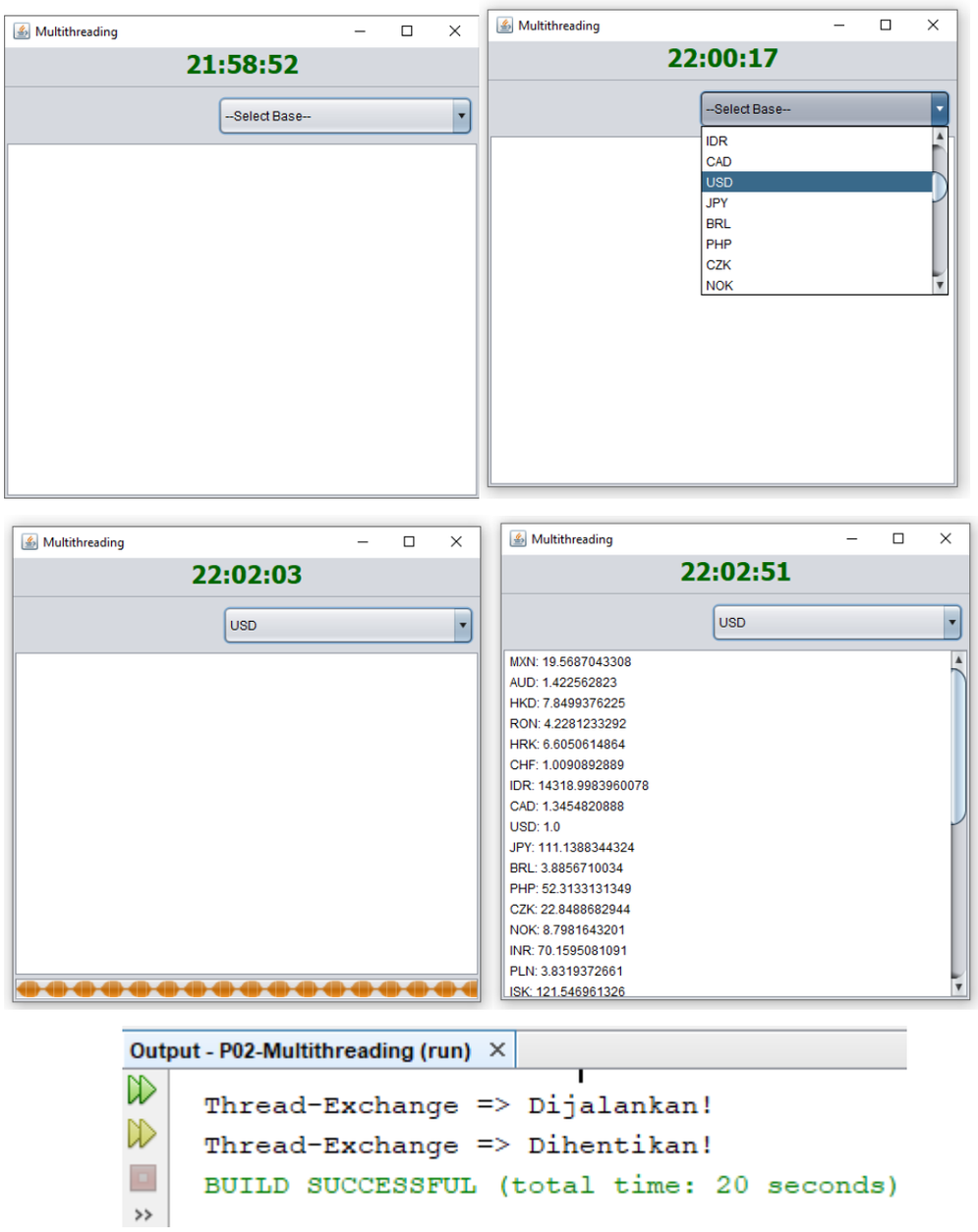
```
42      public ExchangeRate() {
43          initComponents();
44
45          initBase();
46          startClock();
47          loading(false);
48      }
```

8. Selanjutnya kembali ke jendela **Design**, kemudian klik kanan pada komponen ComboBox → pilih **Events** → pilih **Action** → pilih **actionPerformed** → tambahkan kode program sehingga kode menjadi seperti berikut:

```
private void cmbBaseActionPerformed(java.awt.event.ActionEvent evt) {
    int index = cmbBase.getSelectedIndex();
    if(index > 0){
        loading(true);
        new Exchange("Thread-Exchange").execute();
    }else{
        model.clear();
    }
}
```

Jalankan Program: Klik tool  atau tekan **F6**

Hasil Praktik:

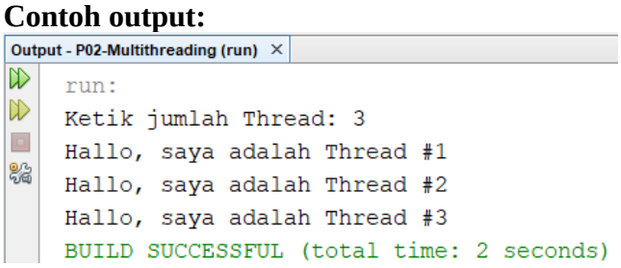


Gambar 2.2 Hasil Praktikum (*Multithreading*)

2.6. Latihan

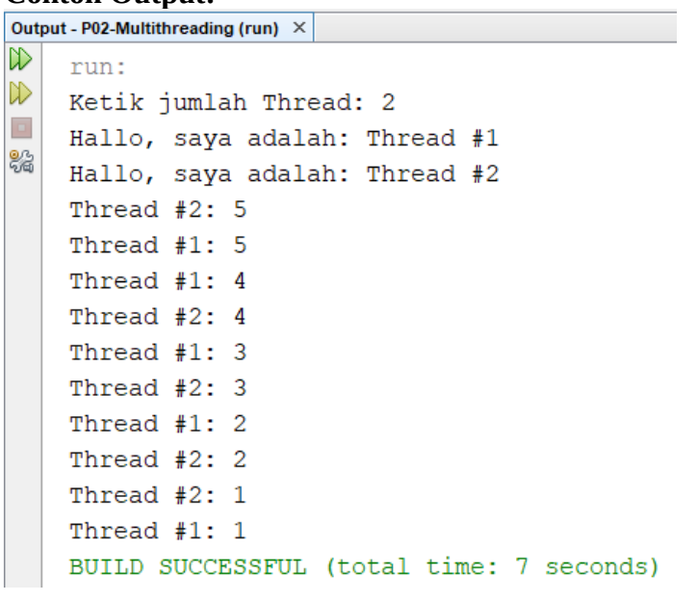
Latihan 1: Buatlah program yang dapat menerima inputan dari user, selanjutnya program akan menjalankan Thread sebanyak inputan user tersebut.

Contoh output:



Latihan 2: Modifikasi Thread yang telah anda buat sebelumya, agar masing-masing Thread dapat menampilkan angka 5 sampai 1.

Contoh Output:



Latihan 3: Modifikasi kembali Thread, agar Thread berjalan secara berurutan (thread berikutnya hanya dijalankan ketika thread yang sedang berjalan telah selesai).

Contoh Output: