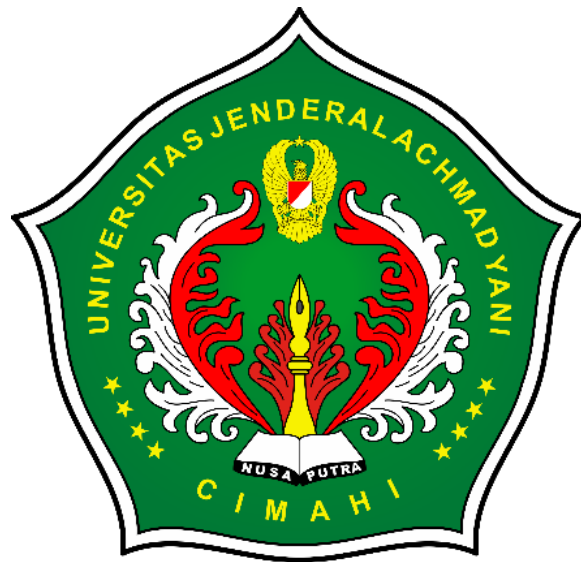


**LAPORAN PRAKTIKUM  
PEMOGRAMAN OBJEK 2**

**MODUL 9  
CONCURRENCY (MULTITHREADING)**

**DISUSUN OLEH :**

**SANDRO ANUGRAH TAMBUNAN      2250081136**



**PROGRAM STUDI INFORMATIKA  
FAKULTAS SAINS DAN INFORMATIKA  
UNIVERSITAS JENDERAL ACHMAD YANI  
TAHUN 2024**



# DAFTAR ISI

DAFTAR GAMBAR .....	iii
BAB I. HASIL PRAKTIKUM.....	1
I.1 SimpleThread.java.....	1
I.1.A. Source Code .....	1
I.1.B. Hasil .....	2
I.1.C. Analisa .....	2
I.2 TwoThreadsDemo.java .....	3
I.2.A. Source Code .....	3
I.2.B. Hasil .....	3
I.2.C. Analisa .....	4
I.3 ThreeThreadsDemo.java .....	5
I.3.A. Source Code .....	5
I.3.B. Hasil .....	5
I.3.C. Analisa .....	6
I.4 CountdownGUI.java.....	7
I.4.A. Source Code .....	7
I.4.B. Hasil .....	8
I.4.C. Analisa .....	9
I.5 Clock.java.....	10
I.5.A. Source Code .....	10
I.5.B. Hasil .....	12
I.5.C. Analisa .....	13
I.6 PrintNameThread.java.....	14
I.6.A. Source Code .....	14
I.6.B. Hasil .....	15

I.6.C. Analisa .....	15
I.7 PrintNameThread.java.....	16
I.7.A. Source Code .....	16
I.7.B. Hasil .....	17
I.7.C. Analisa .....	17
I.8 TwoStrings.java.....	18
I.8.A. Source Code .....	18
I.8.B. Hasil .....	21
I.8.C. Analisa .....	21
I.9 SharedData.java.....	22
I.9.A. Source Code .....	22
I.9.B. Hasil .....	26
I.9.C. Analisa .....	26
BAB II. TUGAS PRAKTIKUM .....	28
BAB III. KESIMPULAN .....	29

## DAFTAR GAMBAR

Gambar 1. 1 Hasil output SimpleThread.java.....	2
Gambar 1. 2 Hasil output TwoThreadsDemo.java .....	3
Gambar 1. 3 Hasil output ThreeThreadsDemo.java .....	5
Gambar 1. 4 Hasil output CountdownGUI.java.....	8
Gambar 1. 5 Hasil output Clock.java.....	12
Gambar 1. 6 Hasil output PrintNameThread.java.....	15
Gambar 1. 7 Hasil output FileReading.java.....	17
Gambar 1. 8 Hasil output TwoStrings.java.....	21
Gambar 1. 9 Hasil output SharedData.java.....	26



# BAB I. HASIL PRAKTIKUM

## I.1 SimpleThread.java

### I.1.A. Source Code

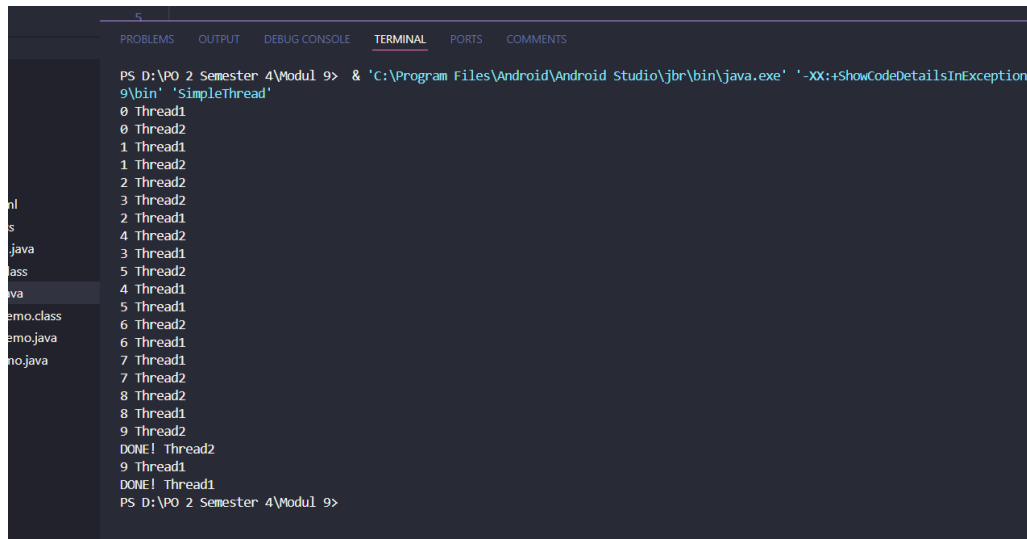
Sebelum diperbaiki:

```
public class SimpleThread extends Thread {  
    public SimpleThread(String str) {  
        super(str);  
    }  
  
    public void run() {  
        for (int i = 0; i < 10; i++) {  
            System.out.println(i + " " + getName());  
            try {  
                sleep((long) (Math.random() * 1000));  
            } catch (InterruptedException e) {  
            }  
        }  
        System.out.println("DONE! " + getName());  
    }  
  
    public static void main(String[] args) {  
        new SimpleThread("Thread1").start();  
        new SimpleThread("Thread2").start();  
    }  
}
```

Setelah diperbaiki

-

### I.1.B. Hasil



Gambar 1. 1 Hasil output SimpleThread.java

### I.1.C. Analisa

Pada kode di atas, kelas `SimpleThread` adalah subclass dari kelas `Thread`. Kelas ini memiliki konstruktor yang menerima sebuah string sebagai parameter dan memanggil konstruktor dari superclass (`Thread`) dengan string tersebut untuk menetapkan nama thread. Metode `run()` dari kelas ini berisi sebuah loop yang berjalan sebanyak 10 kali. Di dalam loop, metode ini mencetak indeks loop beserta nama thread, kemudian membuat thread tidur (sleep) untuk waktu acak antara 0 hingga 1000 milidetik. Jika thread terinterupsi saat tidur, `InterruptedException` akan ditangkap, meskipun tidak ada tindakan yang dilakukan saat pengecualian ini ditangkap.

Di dalam metode `main`, dua instance dari `SimpleThread` dibuat dengan nama "Thread1" dan "Thread2", kemudian kedua instance tersebut dijalankan secara bersamaan menggunakan metode `start()`. Ini akan memulai eksekusi metode `run()` pada kedua thread tersebut secara paralel. Setelah loop selesai, setiap thread akan mencetak pesan "DONE!" beserta nama threadnya. Perilaku sleep dengan waktu acak menyebabkan output dari kedua thread akan muncul dalam urutan yang tidak dapat diprediksi, menampilkan interleaving dari dua thread yang berjalan bersamaan.



## I.2 TwoThreadsDemo.java

### I.2.A. Source Code

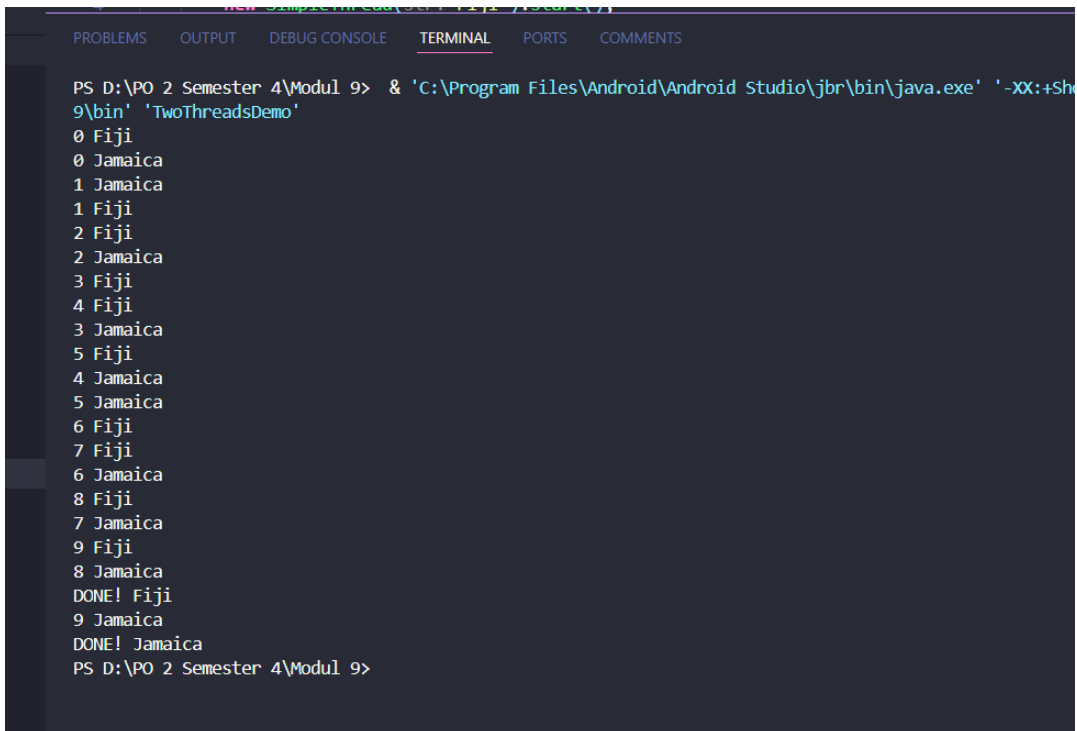
Sebelum diperbaiki:

```
public class TwoThreadsDemo {  
  
    public static void main(String[] args) {  
  
        new SimpleThread("Jamaica").start();  
  
        new SimpleThread("Fiji").start();  
  
    }  
  
}
```

Setelah diperbaiki

-

### I.2.B. Hasil



```
PS D:\PO 2 Semester 4\Modul 9> & 'C:\Program Files\Android\Android Studio\jbr\bin\java.exe' '-XX:+Sh  
9\bin' 'TwoThreadsDemo'  
0 Fiji  
0 Jamaica  
1 Jamaica  
1 Fiji  
2 Fiji  
2 Jamaica  
3 Fiji  
4 Fiji  
3 Jamaica  
5 Fiji  
4 Jamaica  
5 Jamaica  
6 Fiji  
7 Fiji  
6 Jamaica  
8 Fiji  
7 Jamaica  
9 Fiji  
8 Jamaica  
DONE! Fiji  
9 Jamaica  
DONE! Jamaica  
PS D:\PO 2 Semester 4\Modul 9>
```

Gambar 1. 2 Hasil output TwoThreadsDemo.java

### **I.2.C. Analisa**

Pada kode di atas, kelas `TwoThreadsDemo` berfungsi sebagai kelas utama yang menjalankan program. Di dalam metode `main()`, dua instance dari kelas `SimpleThread` dibuat, masing-masing dengan nama "Jamaica" dan "Fiji". Setelah pembuatan instance, metode `start()` dipanggil pada setiap instance tersebut untuk memulai eksekusi thread.

Kelas `SimpleThread` (yang didefinisikan sebelumnya) akan menjalankan metode `run()` pada setiap thread, yang mencetak angka dari 0 hingga 9 bersama dengan nama thread, kemudian membuat thread tidur untuk waktu acak hingga 1000 milidetik di setiap iterasi loop. Setelah loop selesai, setiap thread akan mencetak pesan "DONE!" bersama dengan nama threadnya. Karena dua thread berjalan secara paralel, output dari kedua thread akan tampil secara berselang-seling dalam urutan yang tidak dapat diprediksi, menunjukkan bagaimana dua thread bekerja bersamaan.

## I.3 ThreeThreadsDemo.java

### I.3.A. Source Code

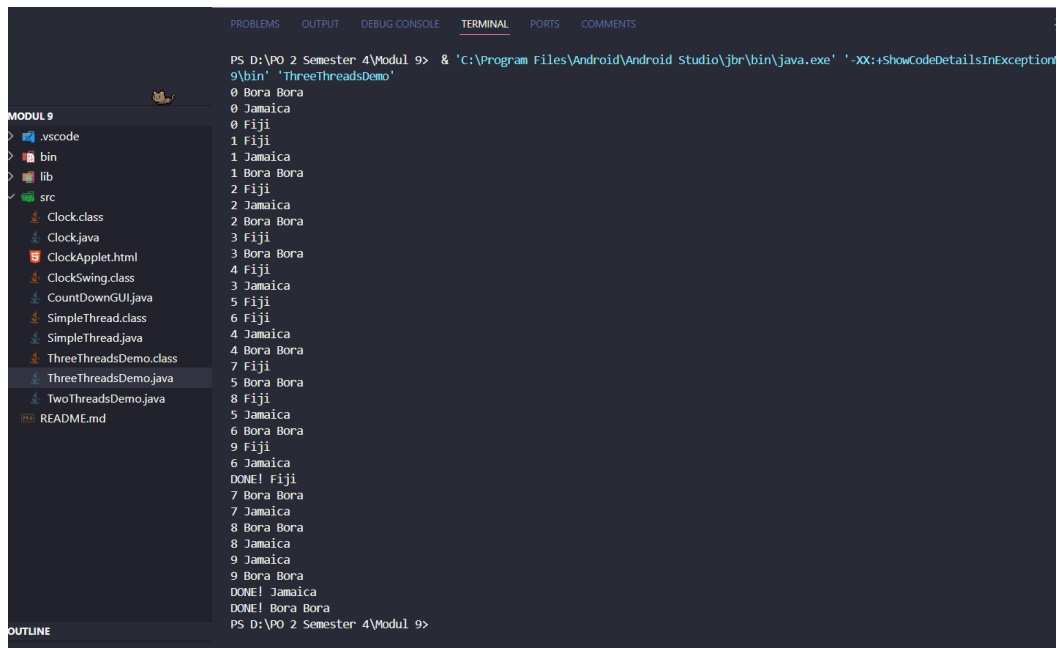
Sebelum diperbaiki:

```
public class ThreeThreadsDemo {  
  
    public static void main(String[] args) {  
  
        new SimpleThread("Jamaica").start();  
  
        new SimpleThread("Fiji").start();  
  
        new SimpleThread("Bora Bora").start();  
  
    }  
  
}
```

Setelah diperbaiki

-

### I.3.B. Hasil



```
PS D:\PO 2 Semester 4\Modul 9> & 'C:\Program Files\Android\Android Studio\jbr\bin\java.exe' '-XX:+ShowCodeDetailsInException...' '9\bin' 'ThreeThreadsDemo'
0 Bora Bora
0 Jamaica
0 Fiji
1 Fiji
1 Jamaica
1 Bora Bora
2 Fiji
2 Jamaica
2 Bora Bora
3 Fiji
3 Bora Bora
4 Fiji
3 Jamaica
5 Fiji
6 Fiji
4 Jamaica
4 Bora Bora
7 Fiji
5 Bora Bora
8 Fiji
5 Jamaica
6 Bora Bora
9 Fiji
6 Jamaica
DONE! Fiji
7 Bora Bora
7 Jamaica
8 Bora Bora
8 Jamaica
9 Jamaica
9 Bora Bora
DONE! Jamaica
DONE! Bora Bora
PS D:\PO 2 Semester 4\Modul 9>
```

Gambar 1. 3 Hasil output ThreeThreadsDemo.java

### **I.3.C. Analisa**

Pada kode di atas, kelas `ThreeThreadsDemo` berfungsi sebagai kelas utama yang menjalankan program. Di dalam metode `main()`, tiga instance dari kelas `SimpleThread` dibuat, masing-masing dengan nama "Jamaica", "Fiji", dan "Bora Bora". Setelah pembuatan instance, metode `start()` dipanggil pada setiap instance tersebut untuk memulai eksekusi thread.

Kelas `SimpleThread` (yang didefinisikan sebelumnya) akan menjalankan metode `run()` pada setiap thread, yang mencetak angka dari 0 hingga 9 bersama dengan nama thread, kemudian membuat thread tidur untuk waktu acak hingga 1000 milidetik di setiap iterasi loop. Setelah loop selesai, setiap thread akan mencetak pesan "DONE!" bersama dengan nama threadnya. Karena tiga thread berjalan secara paralel, output dari ketiga thread akan tampil secara berselang-seling dalam urutan yang tidak dapat diprediksi, menunjukkan bagaimana ketiga thread bekerja bersamaan.

## I.4 CountdownGUI.java

### I.4.A. Source Code

Sebelum diperbaiki:

```
import javax.swing.*;

import java.awt.*;

public class CountdownGUI extends JFrame {

    JLabel label;

    CountdownGUI(String title) {

        super(title);

        label = new JLabel("Start count!");

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        getContentPane().add(new Panel(), BorderLayout.WEST);

        getContentPane().add(label);

        setSize(300, 300);

        setVisible(true);

    }

    void StartCount() {

        try {

            for (int i = 10; i > 0; i++) {

                Thread.sleep(1000);

                label.setText(i + "");

            }

            Thread.sleep(1000);

        }

    }

}
```

```

        label.setText("Count down complete.");

        Thread.sleep(1000);
    } catch (InterruptedException ie) {

    }

    label.setText(Thread.currentThread().toString());
}

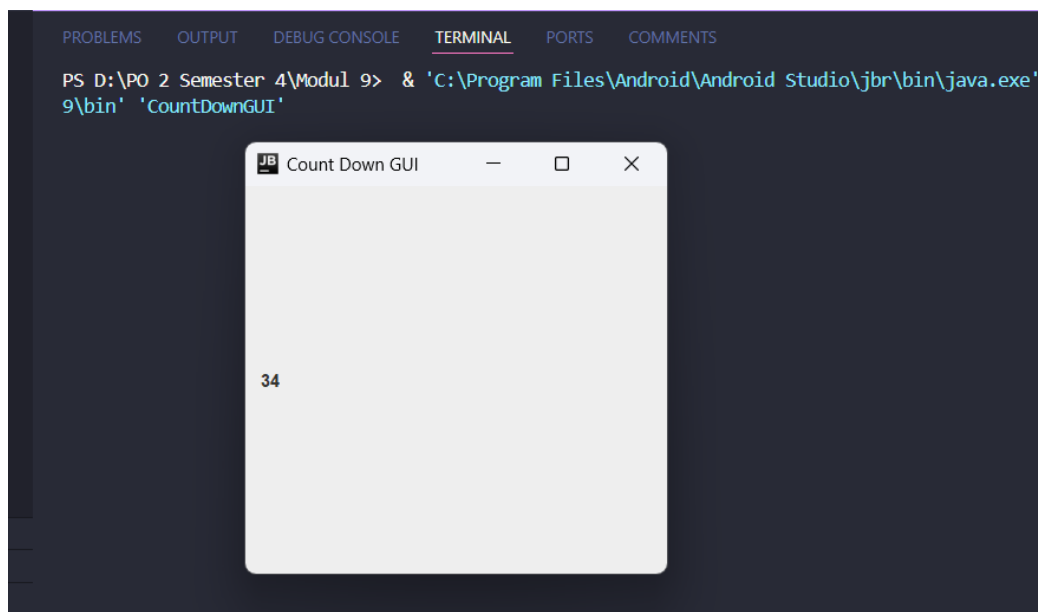
public static void main(String[] args) {
    CountdownGUI cdg = new CountdownGUI("Count Down GUI");
    cdg.StartCount();
}
}

```

Setelah diperbaiki

-

#### I.4.B. Hasil



Gambar 1. 4 Hasil output CountdownGUI.java

#### **I.4.C. Analisa**

Pada kode di atas, kelas `CountDownGUI` adalah subclass dari `JFrame` yang membuat antarmuka grafis untuk melakukan hitung mundur. Di dalam konstruktor `CountDownGUI`, jendela GUI diinisialisasi dengan judul yang diberikan sebagai parameter. Label `JLabel` dengan teks "Start count!" ditambahkan ke jendela, dan pengaturan dasar jendela dilakukan seperti menutup aplikasi saat jendela ditutup, menambahkan panel ke bagian barat layout, dan menambahkan label ke jendela. Ukuran jendela diatur menjadi 300x300 piksel dan jendela dibuat terlihat dengan `setVisible(true)`.

Metode `StartCount` adalah metode yang menjalankan hitung mundur dari 10 hingga 1. Setiap detik, nilai label diperbarui untuk menampilkan angka saat ini. Setelah hitung mundur selesai, label diubah menjadi "Count down complete." dan kemudian setelah satu detik lagi, diubah untuk menampilkan informasi thread saat ini yang menjalankan hitung mundur. Penundaan satu detik dilakukan menggunakan `Thread.sleep(1000)` dan pengecualian `InterruptedException` ditangkap tetapi tidak ada tindakan yang dilakukan jika pengecualian ini terjadi.

Di dalam metode `main`, instance dari `CountDownGUI` dibuat dengan judul "Count Down GUI" dan metode `StartCount` dipanggil untuk memulai hitung mundur. Kode ini akan menampilkan jendela GUI dan menjalankan hitung mundur yang ditampilkan pada label di jendela tersebut.

## I.5 Clock.java

### I.5.A. Source Code

Sebelum diperbaiki:

```
import javax.swing.*;

import java.awt.*;

import java.text.DateFormat;

import java.util.Calendar;

import java.util.Date;


public class Clock extends JFrame implements Runnable {

    private JLabel timeLabel;

    private Thread clockThread = null;


    public Clock(String title) {

        super(title);

        timeLabel = new JLabel();

        timeLabel.setFont(new Font("Serif", Font.BOLD, 24));

        setLayout(new FlowLayout());

        add(timeLabel);

        setSize(200, 100);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        setVisible(true);

    }


    public void start() {

        if (clockThread == null) {
```



```

        clockThread = new Thread(this, "Clock");
        clockThread.start();
    }
}

public void run() {
    Thread myThread = Thread.currentThread();
    while (clockThread == myThread) {
        SwingUtilities.invokeLater(() -> {
            Calendar cal = Calendar.getInstance();
            Date date = cal.getTime();
            DateFormat dateFormatter =
DateFormat.getTimeInstance();
            timeLabel.setText(dateFormatter.format(date));
        });
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            // the VM doesn't want us to sleep anymore, so
get back to work
        }
    }
}

public void stop() {
    clockThread = null;
}

```

```

    }

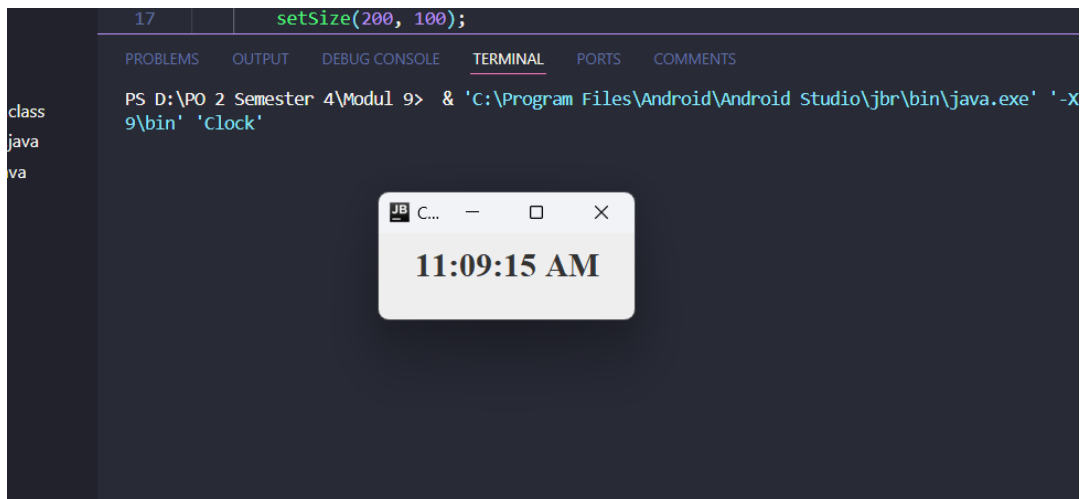
    public static void main(String[] args) {
        Clock clock = new Clock("Clock");
        clock.start();
    }
}

```

Setelah diperbaiki

-

### I.5.B. Hasil



*Gambar 1. 5 Hasil output Clock.java*

### I.5.C. Analisa

Pada kode di atas, kelas `Clock` adalah subclass dari `JFrame` yang juga mengimplementasikan antarmuka `Runnable`. Kelas ini membuat antarmuka grafis yang menampilkan jam digital yang terus diperbarui setiap detik. Konstruktor `Clock` menginisialisasi jendela GUI dengan judul yang diberikan sebagai parameter. Label `JLabel` digunakan untuk menampilkan waktu, dan font label diatur agar lebih besar dan tebal. Layout jendela diatur menggunakan `FlowLayout`, kemudian label ditambahkan ke jendela. Ukuran jendela diatur menjadi 200x100 piksel, pengaturan untuk menutup aplikasi saat jendela ditutup ditetapkan, dan jendela dibuat terlihat dengan `setVisible(true)`.

Metode `start` memeriksa apakah `clockThread` bernilai null, jika iya, maka thread baru dibuat dengan `Clock` sebagai target dan diberi nama "Clock". Thread tersebut kemudian dimulai dengan memanggil `start()`.

Metode `run` adalah metode yang dijalankan oleh thread tersebut. Di dalam metode ini, sebuah loop berjalan terus-menerus selama `clockThread` sama dengan thread saat ini. Di dalam loop, `SwingUtilities.invokeLater` digunakan untuk memastikan bahwa pembaruan GUI dilakukan di thread Event Dispatch Thread. Di dalam `invokeLater`, waktu saat ini diperoleh menggunakan `Calendar.getInstance()`, kemudian diformat menggunakan `DateFormat.getTimeInstance()` dan diatur ke label `timeLabel`. Setelah memperbarui waktu, thread tidur selama satu detik menggunakan `Thread.sleep(1000)`. Jika thread terinterupsi, pengecualian `InterruptedException` ditangkap tetapi tidak ada tindakan khusus yang dilakukan.

Metode `stop` digunakan untuk menghentikan thread dengan mengatur `clockThread` ke null.

Di dalam metode `main`, instance dari `Clock` dibuat dengan judul "Clock", kemudian metode `start` dipanggil untuk memulai thread yang akan memperbarui waktu pada label setiap detik. Kode ini akan menampilkan jendela GUI dengan jam digital yang terus diperbarui.

## I.6 PrintNameThread.java

### I.6.A. Source Code

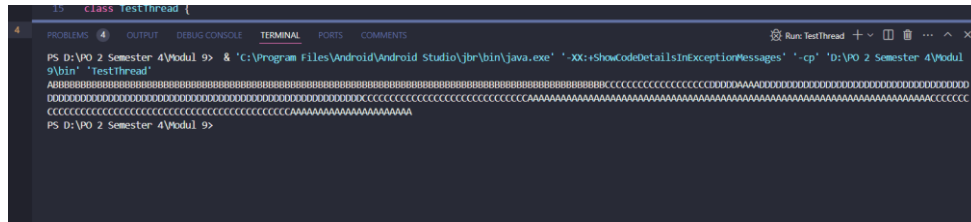
Sebelum diperbaiki:

```
class PrintNameThread extends Thread {  
    PrintNameThread(String name) {  
        super(name);  
        // menjalankan thread dengan satu kali instantiate  
        start();  
    }  
    public void run() {  
        String name = getName();  
        for (int i = 0; i < 100; i++) {  
            System.out.print(name);  
        }  
    }  
}  
  
class TestThread {  
    public static void main(String args[]) {  
        PrintNameThread pnt1 = new PrintNameThread("A");  
        PrintNameThread pnt2 = new PrintNameThread("B");  
        PrintNameThread pnt3 = new PrintNameThread("C");  
        PrintNameThread pnt4 = new PrintNameThread("D");  
    }  
}
```

Setelah diperbaiki

-

### I.6.B. Hasil



Gambar 1. 6 Hasil output PrintNameThread.java

### I.6.C. Analisa

Pada kode di atas, kelas `PrintNameThread` adalah subclass dari `Thread` yang mencetak nama thread sebanyak 100 kali. Konstruktor `PrintNameThread` menerima parameter berupa nama dan memanggil konstruktor superclass `Thread` dengan nama tersebut. Setelah itu, thread langsung dijalankan dengan memanggil `start()` di dalam konstruktor, sehingga thread mulai dieksekusi setelah satu kali instansiasi.

Metode `run` dalam kelas `PrintNameThread` mengambil nama thread menggunakan `getName()` dan menyimpannya dalam variabel `name`. Selanjutnya, dalam sebuah loop yang berjalan sebanyak 100 kali, nama thread dicetak menggunakan `System.out.print(name)`.

Kelas `TestThread` berfungsi sebagai kelas utama yang menjalankan program. Di dalam metode `main`, empat instance dari `PrintNameThread` dibuat dengan nama "A", "B", "C", dan "D". Saat setiap instance dibuat, konstruktor `PrintNameThread` memanggil metode `start()` sehingga masing-masing thread langsung mulai berjalan dan mencetak namanya masing-masing sebanyak 100 kali.

Karena keempat thread berjalan secara paralel, output dari thread-thread tersebut akan muncul secara berselang-seling dalam urutan yang tidak dapat diprediksi, menunjukkan bagaimana beberapa thread dapat bekerja bersamaan untuk mencetak hasil ke konsol.

## I.7 PrintNameThread.java

### I.7.A. Source Code

Sebelum diperbaiki:

```
public class PrintNameThread extends Thread {

    PrintNameThread(String name) {

        super(name);

        start();

    }

    public void run() {

        String name = getName();

        for (int i = 0; i < 100; i++) {

            System.out.println(name);

        }

    }

    public static void main(String[] args) {

        PrintNameThread pnt1 = new PrintNameThread("A");

        PrintNameThread pnt12 = new PrintNameThread("B");

        PrintNameThread pnt13 = new PrintNameThread("C");

        PrintNameThread pnt14 = new PrintNameThread("D");

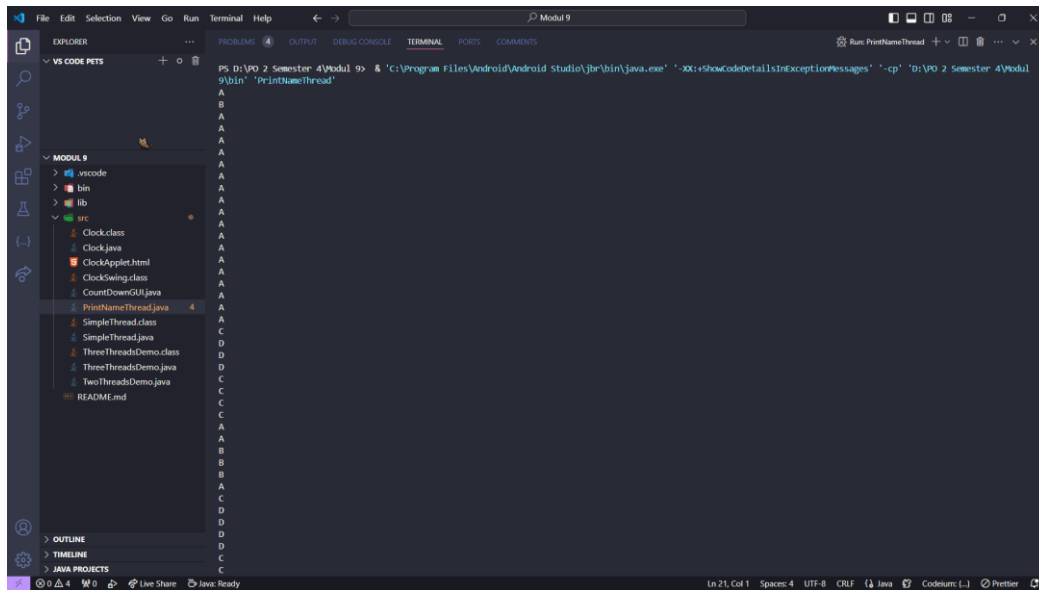
    }

}
```

Setelah diperbaiki

-

### I.7.B. Hasil



*Gambar 1. 7 Hasil output FileReading.java*

### I.7.C. Analisa

Pada kode di atas, kelas `PrintNameThread` adalah subclass dari `Thread` yang mencetak nama thread sebanyak 100 kali. Konstruktor `PrintNameThread` menerima parameter berupa nama dan memanggil konstruktor superclass `Thread` dengan nama tersebut. Setelah itu, thread langsung dijalankan dengan memanggil `start()` di dalam konstruktor, sehingga thread mulai dieksekusi segera setelah diinstansiasi.

Metode `run` dalam kelas `PrintNameThread` mengambil nama thread menggunakan `getName()` dan menyimpannya dalam variabel `name`. Selanjutnya, dalam sebuah loop yang berjalan sebanyak 100 kali, nama thread dicetak menggunakan `System.out.println(name)`, yang mencetak nama thread pada baris baru setiap kali iterasi.

Di dalam metode `main`, empat instance dari `PrintNameThread` dibuat dengan nama "A", "B", "C", dan "D". Saat setiap instance dibuat, konstruktor `PrintNameThread` memanggil metode `start()` sehingga masing-masing thread langsung mulai berjalan dan mencetak namanya masing-masing sebanyak 100 kali.

Karena keempat thread berjalan secara paralel, output dari thread-thread tersebut akan muncul secara berselang-seling dalam urutan yang tidak dapat diprediksi, menunjukkan bagaimana beberapa thread dapat bekerja bersamaan untuk mencetak hasil ke konsol.

## I.8 TwoStrings.java

### I.8.A. Source Code

Sebelum diperbaiki:

```
public class TwoStrings {  
    static void print(String str1, String str2) {  
        System.out.println(str1);  
        try {  
            Thread.sleep(500);  
        } catch (InterruptedException e) {  
        }  
        System.out.println(str2);  
    }  
}  
  
class printStringThread implements Runnable {  
    Thread thread;  
    String str1, str2;  
    printStringThread(String str1, String str2){  
        this.str1 = str1;  
        this.str2 = str2;  
        thread = new Thread(this);  
        thread.start();  
    }  
    public void run() {  
        TwoStrings.print(str1, str2);  
    }  
}
```



```

    }

    class testThread {
        public static void main(String[] args) {
            new printStringThread("Hello", "THere")
            new printStringThread("how are", "yout")
            new printStringThread("THank you", "veri mych")
        }
    }

```

Setelah diperbaiki

```

    public class TwoStrings {
        static void print(String str1, String str2) {
            System.out.println(str1);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println(str2);
        }
    }
}

```

```

class PrintStringThread implements Runnable {
    Thread thread;
    String str1, str2;
    PrintStringThread(String str1, String str2) {

```

```
        this.str1 = str1;

        this.str2 = str2;

        thread = new Thread(this);

        thread.start();

    }

    public void run() {

        TwoStrings.print(str1, str2);

    }

}

class TestThread {

    public static void main(String[] args) {

        new PrintStringThread("Hello", "There");

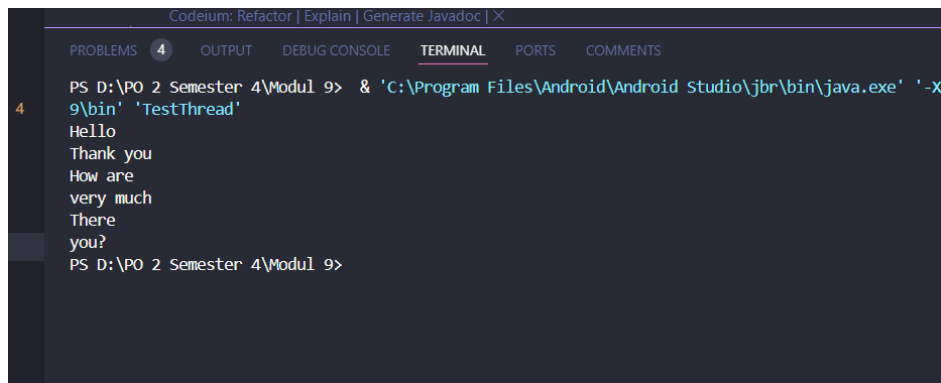
        new PrintStringThread("How are", "you?");

        new PrintStringThread("Thank you", "very much");

    }

}
```

### I.8.B. Hasil



```
Codeium: Refactor | Explain | Generate Javadoc | X
PROBLEMS 4 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
PS D:\PO 2 Semester 4\Modul 9> & 'C:\Program Files\Android\Android Studio\jbr\bin\java.exe' '-X
9\bin' 'TestThread'
Hello
Thank you
How are
very much
There
you?
PS D:\PO 2 Semester 4\Modul 9>
```

*Gambar 1. 8 Hasil output TwoStrings.java*

### I.8.C. Analisa

Pada kode di atas, kelas `TwoStrings` memiliki metode statis `print` yang mencetak dua string dengan jeda waktu setengah detik (500 milidetik) di antara kedua cetakan. Metode ini menggunakan `Thread.sleep(500)` untuk menunda eksekusi selama setengah detik, dan jika terjadi interupsi, pengecualian `InterruptedException` ditangkap dan ditangani dengan mencetak stack trace.

Kelas `PrintStringThread` mengimplementasikan antarmuka `Runnable`, memungkinkan objek dari kelas ini untuk dieksekusi oleh thread. Kelas ini memiliki konstruktor yang menerima dua string sebagai parameter. Di dalam konstruktor, dua string tersebut disimpan dalam variabel instance, dan sebuah thread baru dibuat dan dimulai dengan `thread.start()`.

Metode `run` dalam kelas `PrintStringThread` memanggil metode `print` dari kelas `TwoStrings`, meneruskan dua string yang diberikan sebagai parameter ke konstruktor.

Kelas `TestThread` adalah kelas utama yang menjalankan program. Di dalam metode `main`, tiga instance dari `PrintStringThread` dibuat dengan pasangan string yang berbeda: "Hello" dan "There", "How are" dan "you?", serta "Thank you" dan "very much". Setiap instance secara otomatis memulai thread yang memanggil metode `run`, yang pada gilirannya memanggil metode `print` dari kelas `TwoStrings`.

Karena setiap instance dari `PrintStringThread` berjalan pada thread terpisah, output dari ketiga thread dapat muncul secara berselang-seling. Ini menunjukkan bagaimana beberapa thread dapat bekerja bersamaan untuk mencetak hasil ke konsol, meskipun ada jeda waktu di antara dua cetakan string dalam metode `print`.

## I.9 SharedData.java

### I.9.A. Source Code

Sebelum diperbaiki:

```
public class SharedData {  
    int data;  
  
    synchronized void set(int Value ) {  
        System.out.println("Ganarete " + Value);  
        data = Value;  
    }  
  
    synchronized int get() {  
        System.out.println("Get " + data);  
        return data;  
    }  
  
    class Producer implements Runnable {  
        SharedData sd;  
  
        Producer(SharedData sd) {  
            new Thread(this, "Producer").start();  
        }  
  
        public void run() {  
            for(int i = 0; i < 10; i++) {  
                sd.set((int) (Math.random()*100));  
            }  
        }  
    }  
}
```

```

class Cosumer implements Runnable {
    SharedData sd ;

    Cosumer(SharedData sd) {
        this.sd = sd;
        new Thread(this, "Consumer").start();
    }

    public void run() {
        for(int i = 0; i < 10; i++) {
            sd.get();
        }
    }
}

class TestProducerconsumer {
    public static void main(String[] args) {
        SharedData sd = new SharedData()
        new Producer(sd)
        new Cosumer(sd)
    }
}
}

```

Setelah diperbaiki

```

public class SharedData {
    int data;
}

```

```

synchronized void set(int value) {
    System.out.println("Generate " + value);
    data = value;
}

synchronized int get() {
    System.out.println("Get " + data);
    return data;
}

class Producer implements Runnable {
    SharedData sd;

    Producer(SharedData sd) {
        this.sd = sd;
    }

    public void run() {
        for(int i = 0; i < 10; i++) {
            sd.set((int) (Math.random() * 100));
        }
    }
}

class Consumer implements Runnable {
    SharedData sd ;

```

```

        Consumer(SharedData sd) {
            this.sd = sd;
        }

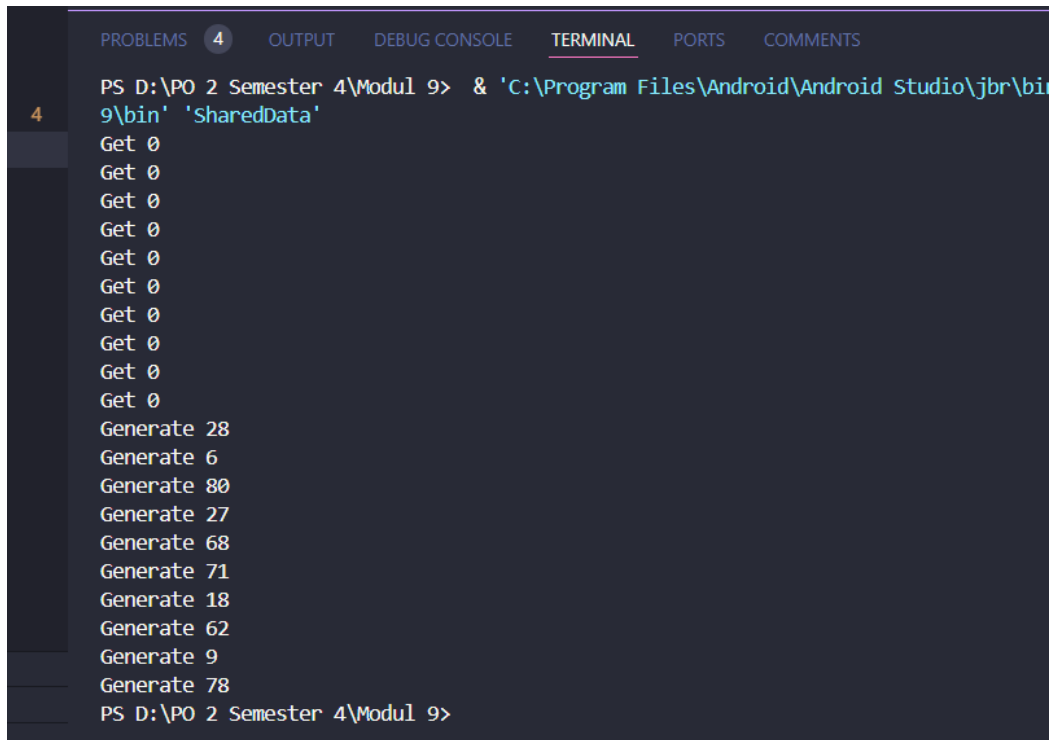
        public void run() {
            for(int i = 0; i < 10; i++) {
                sd.get();
            }
        }
    }

    public static void main(String[] args) {
        SharedData sd = new SharedData();
        SharedData.Producer producer = sd.new Producer(sd);
        SharedData.Consumer consumer = sd.new Consumer(sd);

        new Thread(producer, "Producer").start();
        new Thread(consumer, "Consumer").start();
    }
}

```

### I.9.B. Hasil



```
PS D:\PO 2 Semester 4\Modul 9> & 'C:\Program Files\Android\Android Studio\jbr\bin\java.exe' -cp 'C:\Program Files\Android\Android Studio\jbr\bin\java.exe' 'SharedData'
Get 0
Get 0
Get 0
Get 0
Get 0
Get 0
Get 0
Get 0
Get 0
Get 0
Generate 28
Generate 6
Generate 80
Generate 27
Generate 68
Generate 71
Generate 18
Generate 62
Generate 9
Generate 78
PS D:\PO 2 Semester 4\Modul 9>
```

Gambar 1. 9 Hasil output SharedData.java

### I.9.C. Analisa

Pada kode di atas, kelas `SharedData` mendefinisikan struktur data bersama yang dapat diakses oleh beberapa thread secara sinkron. Kelas ini memiliki variabel instance `data` yang menyimpan nilai integer, serta dua metode yang disinkronisasi: `set` dan `get`. Metode `set` menerima parameter `value`, mencetak pesan "Generate " diikuti oleh nilai tersebut, lalu menetapkan nilai ini ke variabel `data`. Metode `get` mencetak pesan "Get " diikuti oleh nilai `data`, kemudian mengembalikan nilai tersebut.

Kelas `Producer` adalah kelas dalam yang mengimplementasikan antarmuka `Runnable`. Kelas ini memiliki konstruktor yang menerima objek `SharedData` dan menyimpannya dalam variabel instance `sd`. Metode `run` dalam kelas `Producer` menghasilkan nilai acak antara 0 dan 99 sebanyak 10 kali dan mengaturnya ke dalam objek `SharedData` menggunakan metode `set`.



Kelas `Consumer` juga merupakan kelas dalam yang mengimplementasikan antarmuka `Runnable`. Kelas ini memiliki konstruktor yang menerima objek `SharedData` dan menyimpannya dalam variabel instance `sd`. Metode `run` dalam kelas `Consumer` mengambil nilai dari objek `SharedData` menggunakan metode `get` sebanyak 10 kali.

Pada metode `main`, sebuah objek `SharedData` dibuat. Kemudian, dua objek dari kelas dalam `Producer` dan `Consumer` dibuat dengan melewati objek `SharedData` yang telah dibuat sebelumnya. Dua thread baru kemudian dibuat, satu untuk `Producer` dan satu untuk `Consumer`, dan keduanya langsung dijalankan dengan memanggil `start`.

Karena metode `set` dan `get` disinkronisasi, mereka menjamin bahwa hanya satu thread yang dapat mengakses variabel `data` pada waktu yang sama. Ini mencegah terjadinya kondisi balapan (race condition) ketika thread `Producer` dan `Consumer` mencoba untuk menulis dan membaca nilai `data` secara bersamaan. Output dari program ini akan menunjukkan nilai-nilai yang dihasilkan oleh `Producer` dan diambil oleh `Consumer`, dengan pesan yang dicetak ke konsol setiap kali nilai diatur atau diambil.

## **BAB II. TUGAS PRAKTIKUM**

### **BAB III. KESIMPULAN**

Pada pertemuan praktikum modul 9 ini, saya mempelajari bagaimana cara mengimplementasikan threading dalam bahasa pemrograman Java. Dimulai dengan contoh sederhana dari kelas `SimpleThread` yang menjalankan thread untuk mencetak angka beserta nama thread secara paralel, saya memahami konsep dasar threading dan bagaimana thread dapat berjalan bersamaan dengan output yang muncul secara berselang-seling.

Selanjutnya, saya melihat penerapan threading dalam GUI dengan kelas `CountDownGUI` yang menampilkan hitung mundur menggunakan `JLabel` pada jendela JFrame, dan bagaimana thread diimplementasikan untuk memperbarui GUI setiap detik. Pada kelas `Clock`, saya mempelajari cara membuat jam digital yang memperbarui waktu setiap detik menggunakan thread dan memastikan pembaruan GUI dilakukan di thread Event Dispatch Thread.

Pada kelas `PrintNameThread`, saya memahami bagaimana membuat thread yang mencetak nama thread sebanyak 100 kali dengan konstruktor yang langsung memulai thread saat diinstansiasi. Selain itu, saya melihat bagaimana thread berjalan secara paralel dan mencetak hasil ke konsol dengan urutan yang tidak dapat diprediksi.

Dalam kelas `TwoStrings`, saya melihat bagaimana beberapa thread dapat dijalankan untuk mencetak dua string dengan jeda waktu, serta memahami pentingnya sinkronisasi dalam mengakses data bersama melalui contoh kelas `SharedData` yang memiliki metode `set` dan `get` yang disinkronisasi untuk mencegah kondisi balapan.

Secara keseluruhan, praktikum ini memberikan pemahaman mendalam tentang threading di Java, termasuk cara mengelola dan menyinkronisasi thread, serta penerapannya dalam berbagai skenario pemrograman.