PRAKTIKUM 25

INPUT DAN OUTPUT

A. TUJUAN PEMBELAJARAN

1. Memahami konsep Input/Output di Java
2. Mengenal kelas – kelas yang berhubungan dengan IO.
3. Mampu membuat program yang menerapkan konsep Input Output.

B. DASAR TEORI

Dalam pemrograman Java, Stream and File bukanlah istilah yang asing didengar. Stream and File merupakan proses penulisan dan pembacaan data sering yang di sebut dengan proses input dan output, dimana penulisan data berarti mengalirkan data ke output dan menerima atau mendapatkan data dari input.

Penerapan dalam pemrograman java ini selalu melakukan proses input dan output yaitu memindahkan byte data dari satu system ke system lain. Data yang dibaca dari server yang mengirim data tidak berbeda dengan membaca data dari sebuah file. Java mengangani I/O secara berbeda dari bahasa-bahasa pemrograman yang lainnya.

STREAM

Stream merupakan dasar operasi input-output (I/O) dalam Java yang menggunakan package java.io sebagai package utama. Stream adalah representasi abstrak dari input dan output device, dimana aliran bytes akan ditransfer seperti file dalam harddisk, file pada sistem remote atau printer. Kita dapat membaca data dari input stream, yang dapat berupa file, keyboard atau komputer remote. Dalam Stream ini terdapat proses input dan output sebagai berikut :

1. Input Stream

Kelas java.io.InputStream adalah:

```
public abstract class InputStream
```

Adapun 2 method utama dari InputStream adalah :

- **Read**

  Method ini digunakan untuk membaca stream.

- **Close**

  Method ini digunakan untuk menutup koneksi input stream.

Dalam proses Penginputan Stream ini pun terdapat pembagian dalam kelasnya, yaitu :

a) Byte Stream

Merupakan kelas dan interface ini digunakan untuk menangani data biner.

b) Character Stream

Merupakan kelompok kelas ini digunakan untuk menangani proses baca tulis karakter Unicode. Kelas ini merupakan pengembangan dari kelas Byte Stream sehingga lebih efisien.

Data input dalam Stream ini berfungsi untuk saling melengkapi dengan DataOutputStream, yaitu untuk mendapatkan data yang telah ditulis.

2. Output Stream

Subclass-subclass dari outputStream adalah :

- **ByteArrayOutputStream** : digunakan untuk menuliskan stream menjadi byte array.
- **FileOutputStream** : digunakan untuk menulis pada file
- **FilterOutputStream** :merupakan superclass dari subclass-subclass seperti DataOutputStream, BufferOutputStream, PrintStream, CheckedOutputStream
- **ObjectOutputStream** : digunakan untuk menuliskan objek pada OutputStream.
- **PipedOutputStream** : digunakan untuk menjadi output dari PipedInputStream.

Data Output dalam stream ini merupakan class yang menyediakan cara praktis untuk menuliskan tipe data primitif ke output stream yang lebih mudah digunakan dalam penyelesaian program dalam java.

FILE

File merupakan data yang siap diinput dan diproses dalam Stream yang merupaka data operasi dalam pemrograman. Keterkaitan antara keduanya, proses Input dan Ouput tetap dilakukan walau dengan cara yang berbeda, dari subclass maupun method yang digunakan.

File Input Stream dan File Output Stream

FileInputStream digunakan untuk membaca data dari file yang merupakan turunan langsung dari class InputStream dan FileOutputStream untuk menuliskan data ke file merupakan turunan langsung dari class OutputStream.

Dalam file pun terdapat subclass – subclass dan method, sama halnya dengan Stream, seperti :

1. Class File

Class File merupakan langkah awal dalam mempelajari proses input-output dengan Java, karena File merupakan objek yang mewakili path, file, atau direktori pada harddisk. Ada tiga cara membuat objek File, yaitu :

➔ Menggunakan objek string sebagai argumen yang menginformasikan path untuk file atau direktori.

➔ Menggunakan dua langkah, dimana yang pertama untuk mendefinisikan direktori dan yang kedua untuk file.

➔ Menggunakan dua argumen, dimana yang pertama adalah argumen string yang mendefinisikan direktori, dan yang kedua adalah argumen string yang mendefinisikan nama file.

2. File Writer

Di dalam aplikasi web, disamping database, penggunaan file untuk menyimpan data cukup banyak dilakukan karena kebutuhan penyimpanan data yang sederhana cukup dengan menggunakan file. File Writer merupakan subclass dari OutputStreamWriter yang merupakan subclass dari class abstract Writer. Class FileWriter memiliki konstruktor yang umum seperti berikut :

a) FileWriter (File objekfile);

b) FileWriter (String pathkefile);

c) FileWriter (String pathkefile, boolean append);

3. File Reader

File Reader merupakan class yang dapat digunakan untuk membaca file teks. Konstruktor dari FileReader :

- FileReader(File objekfile);
- FileReader(String pathkefile);

Method yang digunakan :

- Read(char[] array);
- Read(char[] array, int offset, int length);

C. TUGAS PENDAHULUAN

Pelajari konsep IO Java pada Java API documentation

D. PERCOBAAN

Percobaan 1: Melihat direktori

```
// Using directories.
import java.io.File;

class DirList {
  public static void main(String args[]) {
    String dirname = "/java";
    File f1 = new File(dirname);
    if (f1.isDirectory()) {
      System.out.println("Directory of " + dirname);
      String s[] = f1.list();

      for (int i=0; i < s.length; i++) {
        File f = new File(dirname + "/" + s[i]);
        if (f.isDirectory()) {
          System.out.println(s[i] + " is a directory");
        } else {
          System.out.println(s[i] + " is a file");
        }
      }
    } else {
      System.out.println(dirname + " is not a directory");
    }
  }
}
```

Percobaan 2 : Menggunakan FileInputStream

```
// Demonstrate FileInputStream.
// This program uses try-with-resources. It requires JDK 7 or later.

import java.io.*;
class FileInputStreamDemo {
  public static void main(String args[]) {
    int size;

    // Use try-with-resources to close the stream.
    try ( FileInputStream f =
           new FileInputStream("FileInputStreamDemo.java") ) {

      System.out.println("Total Available Bytes: " +
                         (size = f.available()));

      int n = size/40;
      System.out.println("First " + n +
                         " bytes of the file one read() at a time");
      for (int i=0; i < n; i++) {
        System.out.print((char) f.read());
      }

      System.out.println("\nStill Available: " + f.available());

      System.out.println("Reading the next " + n +
                         " with one read(b[])");
      byte b[] = new byte[n];
      if (f.read(b) != n) {
        System.err.println("couldn't read " + n + " bytes.");
      }


    System.out.println(new String(b, 0, n));
    System.out.println("\nStill Available: " + (size = f.available()));
    System.out.println("Skipping half of remaining bytes with skip()");
    f.skip(size/2);
    System.out.println("Still Available: " + f.available());

    System.out.println("Reading " + n/2 + " into the end of array");
    if (f.read(b, n/2, n/2) != n/2) {
      System.err.println("couldn't read " + n/2 + " bytes.");
    }

    System.out.println(new String(b, 0, b.length));
    System.out.println("\nStill Available: " + f.available());
  } catch(IOException e) {
    System.out.println("I/O Error: " + e);
  }
 }
}
```

Percobaan 3 : Contoh menggunakan BufferedInputStream

```
// Use buffered input.
// This program uses try-with-resources. It requires JDK 7 or later

import java.io.*;

class BufferedInputStreamDemo {
  public static void main(String args[]) {
    String s = "This is a &copy; copyright symbol " +
      "but this is &copy not.\n";
    byte buf[] = s.getBytes();

    ByteArrayInputStream in = new ByteArrayInputStream(buf);
    int c;
    boolean marked = false;

    // Use try-with-resources to manage the file.
    try ( BufferedInputStream f = new BufferedInputStream(in) )
    {
      while ((c = f.read()) != -1) {
        switch(c) {
        case '&':
          if (!marked) {
            f.mark(32);
            marked = true;
          } else {
            marked = false;
          }
          break;
        case ';':
          if (marked) {
            marked = false;
            System.out.print("(c)");
          } else

            System.out.print((char) c);
          break;
        case ' ':
          if (marked) {
            marked = false;
            f.reset();
            System.out.print("&");
          } else
            System.out.print((char) c);
          break;
        default:
          if (!marked)
            System.out.print((char) c);
          break;
        }
      }
    } catch(IOException e) {
      System.out.println("I/O Error: " + e);
    }
  }
}
```

Percobaan 4 : Contoh menggunakan PushbackInputStream

```
// Demonstrate unread().
// This program uses try-with-resources. It requires JDK 7 or later.

import java.io.*;

class PushbackInputStreamDemo {
  public static void main(String args[]) {
    String s = "if (a == 4) a = 0;\n";
    byte buf[] = s.getBytes();
    ByteArrayInputStream in = new ByteArrayInputStream(buf);
    int c;
    try ( PushbackInputStream f = new PushbackInputStream(in) )
    {
      while ((c = f.read()) != -1) {
        switch(c) {
        case '=':
          if ((c = f.read()) == '=')
            System.out.print(".eq.");
          else {
            System.out.print("<-");
            f.unread(c);
          }
          break;
        default:
          System.out.print((char) c);
          break;
        }
      }
    } catch(IOException e) {
      System.out.println("I/O Error: " + e);
    }
  }
}
```

Percobaan 5 : Contoh menggunakan Serialization

```
// A serialization demo.
// This program uses try-with-resources. It requires JDK 7 or later.

import java.io.*;

public class SerializationDemo {
  public static void main(String args[]) {

    // Object serialization

    try ( ObjectOutputStream objOStrm =
            new ObjectOutputStream(new FileOutputStream("serial")) )
    {
      MyClass object1 = new MyClass("Hello", -7, 2.7e10);
      System.out.println("object1: " + object1);

      objOStrm.writeObject(object1);
    }
    catch(IOException e) {
      System.out.println("Exception during serialization: " + e);
    }

    // Object deserialization

    try ( ObjectInputStream objIStrm =
            new ObjectInputStream(new FileInputStream("serial")) )
    {
      MyClass object2 = (MyClass)objIStrm.readObject();
      System.out.println("object2: " + object2);
    }
    catch(Exception e) {
      System.out.println("Exception during deserialization: " + e);
    }
  }
}

class MyClass implements Serializable {
  String s;
  int i;
  double d;

  public MyClass(String s, int i, double d) {
    this.s = s;
    this.i = i;
    this.d = d;
  }
  public String toString() {
    return "s=" + s + "; i=" + i + "; d=" + d;
  }
}
```

E. LAPORAN RESMI

Kumpulkan hasil percobaan di atas dan tambahkan analisa untuk tiap percobaan, latihan, dan tugas yang telah dibuat.