Laporan Contoh Program Java

I/O Stream

Dosen pengajar:

RUDHI WAHYUDI FEBRIANTO

Disusun Oleh:

Nama : Ade Hikmat Pauji Ridwan

Kelas : 222K

NPM : 22552011130

PROGRAM STUDI TEKNIK INFORMATIKA

UNIVERSITAS TEKNOLOGI BANDUNG

2024

BAB I
PENDAHULUAN

1.1. Latar Belakang

Java adalah salah satu bahasa pemrograman yang paling populer dan serbaguna yang digunakan dalam berbagai jenis aplikasi, mulai dari aplikasi desktop, web, hingga aplikasi seluler. Dengan perubahan zaman yang membawa segala sesuatu menjadi global, teknologi juga mengalami perkembangan yang pesat, dan pemrograman komputer menjadi salah satu aspek yang sangat krusial dalam era digital ini.

Bahasa pemrograman Java dikembangkan oleh James Gosling pada tahun 1995 sebagai bagian dari Sun Microsystems Java Platform. Java sangat dipengaruhi oleh sintaksis bahasa C dan C++, tetapi dengan penyederhanaan dan keketatan yang lebih tinggi, serta akses yang lebih terbatas ke sistem operasi. Hal ini menjadikan Java sebagai bahasa pemrograman yang mudah dipelajari dan dipahami. Sebagai bahasa pemrograman tingkat tinggi (High Level Language), Java dirancang tidak hanya untuk dikompilasi oleh mesin, tetapi juga agar dapat dimengerti oleh manusia.

Dalam penggunaannya, Java menggunakan berbagai konsep dan operator pemrograman seperti Class, Object, Constructor, Inheritance, dan lain-lain. Konsep-konsep ini memungkinkan pengembang untuk membuat kode yang lebih terstruktur dan modular, sehingga memudahkan dalam pengembangan dan pemeliharaan aplikasi. Hingga saat ini, mayoritas produk yang diperkenalkan ke industri teknologi informasi dan telekomunikasi modern menggunakan bahasa pemrograman Java sebagai komponen utama dari arsitektur mereka, menunjukkan betapa pentingnya peran Java dalam perkembangan teknologi informasi global.

1.2. Tujuan

Adapun tujuan dari praktikum ini adalah:

1. Mampu memahami konsep *I/O Stream*.
2. Mampu membuat program yang mengalirkan data melalui stream.

BAB II
DASAR TEORI

2.1 Pengertian Stream

Stream adalah konsep yang memungkinkan rangkaian item dapat dibaca atau ditulis dari atau ke sebuah file atau device. Dalam bahasa pemrograman Java, stream dibagi menjadi dua jenis utama: byte stream dan character stream.

- **Byte Stream**: Byte stream digunakan untuk operasi input dan output (I/O) yang berhubungan dengan data biner. Contoh penggunaan byte stream adalah ketika membaca atau menulis file gambar, audio, atau video. Kelas utama yang digunakan untuk byte stream adalah InputStream dan OutputStream, dengan subclass seperti FileInputStream, FileOutputStream, BufferedInputStream, dan BufferedOutputStream.
- **Character Stream**: Character stream digunakan untuk operasi I/O yang berhubungan dengan data karakter. Ini berarti bahwa character stream lebih cocok untuk membaca dan menulis data teks. Kelas utama yang digunakan untuk character stream adalah Reader dan Writer, dengan subclass seperti FileReader, FileWriter, BufferedReader, dan BufferedWriter.

2.1.1 InputStream dan OutputStream

Terdapat dua kelas abstrak utama yang dirancang sebagai kelas induk untuk kelas-kelas yang termasuk dalam kategori byte stream, yaitu InputStream dan OutputStream. Kedua kelas ini berperan penting dalam menangani operasi input/output (I/O) dalam Java. Karena merupakan kelas abstrak, InputStream dan OutputStream tidak dapat digunakan secara langsung. Namun, mereka memiliki banyak kelas turunan (subclass) yang dapat digunakan untuk berbagai kebutuhan I/O.

Kelas Turunan dari InputStream

InputStream adalah kelas induk abstrak yang menyediakan metode dasar untuk membaca byte dari sumber data. Beberapa kelas turunan dari InputStream meliputi:

- **ByteArrayInputStream**: Digunakan untuk membaca array byte sebagai aliran input.
- **FileInputStream**: Digunakan untuk membaca data dari file.
- **FilterInputStream**: Kelas abstrak yang digunakan untuk membuat aliran input yang bisa di-filter.
- **ObjectInputStream**: Digunakan untuk membaca objek yang telah diserialisasi.
- **PipedInputStream**: Digunakan untuk membaca data dari aliran pipa (pipe) yang memungkinkan komunikasi antar thread.

Kelas Turunan dari OutputStream

OutputStream adalah kelas induk abstrak yang menyediakan metode dasar untuk menulis byte ke tujuan data. Beberapa kelas turunan dari OutputStream meliputi:

- **ByteArrayOutputStream**: Digunakan untuk menulis data ke array byte sebagai aliran output.
- **FileOutputStream**: Digunakan untuk menulis data ke file.

- **FilterOutputStream**: Kelas abstrak yang digunakan untuk membuat aliran output yang bisa di-filter.
- **ObjectOutputStream**: Digunakan untuk menulis objek ke aliran output sebagai byte yang telah diserialisasi.
- **PipedOutputStream**: Digunakan untuk menulis data ke aliran pipa (pipe) yang memungkinkan komunikasi antar thread.

BAB III
METOLOGI PERCOBAAN

3.1 Alat dan Bahan

1. PC / Laptop
2. Visual Studio Code
3. Java

3.2 Algoritma

3.2.1 Algoritma Rata-rata Dari InputStream

1. Mulai.
2. Lakukan perintah pemanggilan java.io.
3. Buka stream untuk membaca input dari konsol.
4. Minta pengguna memasukkan deretan angka yang dipisahkan oleh spasi.
5. Baca input dari pengguna.
6. Pisahkan input berdasarkan spasi dan ubah menjadi array string.
7. Konversi setiap string menjadi integer dan hitung jumlah totalnya.
8. Hitung rata-rata dengan membagi jumlah total dengan jumlah elemen dalam array.
9. Tampilkan nilai rata-rata angka.
10. Tangani pengecualian jika terjadi kesalahan input/output atau format angka tidak valid.
11. Tutup semua stream yang dibuka.
12. Selesai.

3.2.2 Algoritma Menambahkan Input ke File dengan OutputStream

1. Mulai.
2. Lakukan perintah pemanggilan java.io.
3. Buka stream untuk menambahkan data ke file.
4. Minta pengguna memasukkan teks.
5. Baca input dari pengguna.
6. Tambahkan input ke dalam file.
7. Tangani pengecualian jika terjadi kesalahan input/output.
8. Tampilkan pesan sukses atau gagal.
9. Tutup stream yang dibuka.
10. Selesai.

BAB IV
HASIL DAN ANALISA

4.1 Hasil Percobaan

4.1.1 Percobaan Menghitung Rata-rata Dari InputStream dari InputStream

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class AverageCalculator {
    public static void main(String[] args) {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        try {
            System.out.println("Masukkan deretan angka yang dipisahkan oleh spasi:");
            String input = reader.readLine();
            String[] numberStrings = input.split(" ");

            int sum = 0;
            for (String numberString : numberStrings) {
                sum += Integer.parseInt(numberString);
            }

            double average = (double) sum / numberStrings.length;
            System.out.println("Rata-rata dari angka yang dimasukkan adalah: " +
average);

        } catch (IOException e) {
            System.err.println("Terjadi kesalahan input/output: " + e.getMessage());
        } catch (NumberFormatException e) {
            System.err.println("Format angka tidak valid: " + e.getMessage());
        } finally {
            try {
                reader.close();
            } catch (IOException e) {
                System.err.println("Gagal menutup stream: " + e.getMessage());
            }
        }
    }
}
```

Output:

```
Masukkan deretan angka yang dipisahkan oleh spasi:
1 2 4 5
Rata-rata dari angka yang dimasukkan adalah: 3.0
```

4.2.2 Percobaan Memasukan Input ke File dengan OutputStream

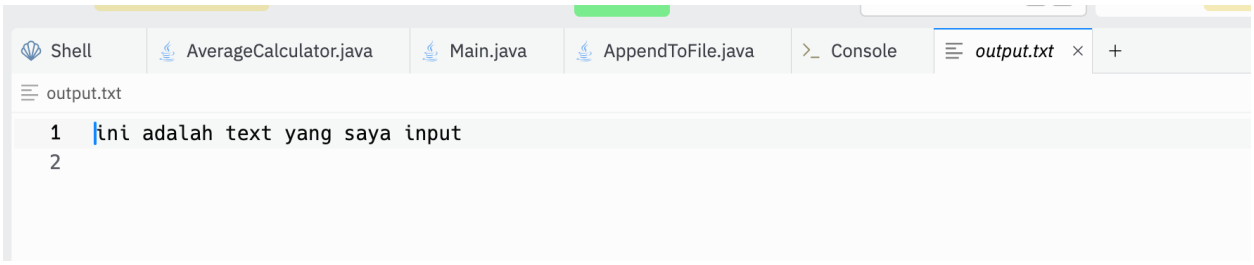
```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Scanner;

public class AppendToFile {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Masukkan teks yang ingin ditambahkan ke dalam file:");
        String userInput = scanner.nextLine();

        PrintWriter out = null;
        try {
            FileWriter fw = new FileWriter("output.txt", true);
            BufferedWriter bw = new BufferedWriter(fw);
            out = new PrintWriter(bw);
            out.println(userInput);
            System.out.println("Teks berhasil ditambahkan ke dalam file.");
        } catch (IOException e) {
            System.err.println("Terjadi kesalahan input/output: " + e.getMessage());
        } finally {
            if (out != null) {
                out.close();
            }
            scanner.close();
        }
    }
}

OUTPUT:
```

```
Masukkan teks yang ingin ditambahkan ke dalam file:
ini adalah text yang saya input
Teks berhasil ditambahkan ke dalam file.
```


Shell
AverageCalculator.java
Main.java
AppendToFile.java
Console
output.txt
output.txt
1 ini adalah text yang saya input
2

BAB V
KESIMPULAN

Kedua program di atas menunjukkan penggunaan dasar dari InputStream dan OutputStream di Java untuk membaca dan menulis data. InputStream digunakan untuk membaca data dari sumber input (misalnya, konsol), sementara OutputStream digunakan untuk menulis data ke tujuan output (misalnya, file). Penggunaan konsep stream ini memungkinkan pemrosesan data yang efisien dan fleksibel dalam aplikasi Java.

Poin Penting:

1. **Pemrosesan Input:** Program InputStream mengilustrasikan bagaimana cara membaca dan memproses input teks dari pengguna. Input tersebut dipisahkan menjadi bagian-bagian berdasarkan spasi, kemudian dikonversi menjadi integer, dan akhirnya dijumlahkan. Ini menunjukkan kemampuan Java dalam memanipulasi dan mengolah data teks yang dimasukkan melalui konsol.
2. **Penanganan Output:** Program OutputStream menunjukkan bagaimana cara menulis teks yang dimasukkan pengguna ke dalam file. Dengan membungkus FileOutputStream dengan OutputStreamWriter dan BufferedWriter, program ini memastikan efisiensi dalam penulisan data ke file, yang penting untuk kinerja aplikasi yang baik.
3. **Penanganan Kesalahan:** Kedua program menunjukkan pentingnya menangani pengecualian (IOException dan NumberFormatException). Penanganan pengecualian ini memastikan bahwa program dapat berjalan dengan lancar dan memberikan umpan balik yang sesuai kepada pengguna saat terjadi kesalahan, meningkatkan keandalan dan pengalaman pengguna.
4. **Pengelolaan Sumber Daya:** Menutup semua stream yang dibuka sangat penting untuk mencegah kebocoran sumber daya dan memastikan integritas data. Pengelolaan sumber daya yang baik adalah aspek kritis dalam pengembangan perangkat lunak untuk menjaga performa aplikasi dan mencegah masalah yang berkaitan dengan penggunaan memori dan file.

   Secara keseluruhan, pemahaman dan penggunaan yang tepat dari InputStream dan OutputStream adalah dasar yang kuat untuk mengembangkan aplikasi Java yang efisien dan efektif dalam menangani berbagai operasi I/O.

Code Link:
https://replit.com/@adehikmat1/Tugas-PBO-2-Ade-Hikmat-IO-Stream?v=1