Setelah mempelajari modul 2 ini, mahasiswa diharapkan :

- mampu memahami konsep stream
- mampu menerapkan class stream

Komunikasi pada jaringan, dengan file, atau antar aplikasi direpresentasikan dengan Stream oleh JAVA. Komunikasi berbasis Stream adalah inti dari hampir semua komunikasi aplikasi yang dikembangkan menggunakan JAVA. Konsep stream merupakan dasar yang penting dalam mengembangkan aplikasi jaringan.

3.1. Review Stream

Komunikasi pada level byte, direpresentasikan oleh JAVA menggunakan stream, sebagai wadah tempat informasi dikirim ataupun tempat informasi diterima. Stream dapat dianalogikan dengan pipa air, yang apabila terpasan dengan baik, air (dalam hal ini informasi atau data) dapat mengalir dari suatu tempat ke tempat lain.

Dalam membangun sebuah aplikasi, kita harus dapat memilih jenis class stream yang tepat untuk aplikasi kita. Stream dapat digabungkan bersama-sama untuk mendapatkan kemudahan pengolahan informasi.

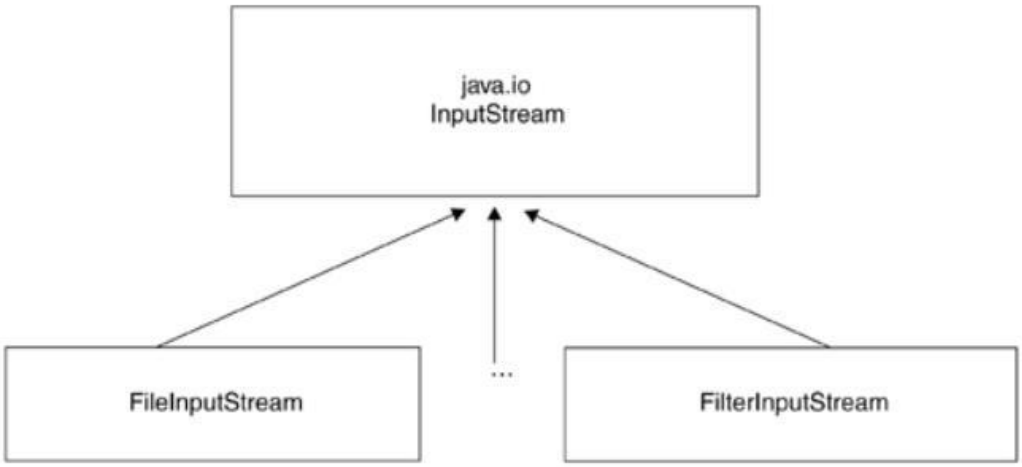
Ada 2 macam kategori besar Stream :

- input stream : dimana informasi dapat kita baca / ambil

- output stream : dimana informasi dapat kita tulis / kirim

3.2. Input Stream

Paket Input Stream terletak pada java.io.InputStream. Merupakan stream dimana kita bisa membaca aliran informasi dari suatu sumber.



Ada 6 macam input stream yang bisa kita gunakan dari paket tersebut. Saat input stream dibuat, objeknya akan membaca informasi dari sumber informasi.

Low-level Input Stream	***Fungsi***
byteArrayInputStream	Membaca data byte dari array di memory
FileInputStream	membaca data byte dari sebuah file pada sistem file lokal
PipedInputStream	Membaca data byte dari pipe thread
StringBufferInputStream	Membaca data byte dari sebuah string

SequenceInputStream	Membaca data byte dari stream input lain
System.in	Membaca data byte dari konsol user

Kelas abstrak InputStream memiliki method-method umum yang diwariskan pada semua class turunannya, yang mana bersifat public. Method-method tersebut adalah :

- **int available ( )** throws java.io.IOException : untuk mengetahui jumlah byte data yang saat ini tersedia untuk dibaca.
- **void close ( )** throws java.io.IOException : menutup input stream dan melepaskan sumber daya yang terkait dengan input stream tersebut (contoh : file yang menjadi sumber input stream)
- **void mark(int readLimit) :** mencatat posisi input stram saat ini, agar nanti bisa kembali ke titik yang dicatat dengan memanggil method InputStream.reset() . Tidak semua input stream mendukung method ini.
- **boolean markSupported ( ) :** mengembalikan nilai “true” apabila input stream mendukung mark() dan reset().
  - **int read( ) throws java.io.IOException :** mengambil byte data berikutnya dari stream. Subkelas dari InputStream biasanya menuliskan ulang method ini untuk menyesuaikan fungsinya (seperti untuk membaca dari string spt pada StringBufferInputStream, atau untuk membaca dari file seperti pada FileInputStream). Input stream akan mengeblok penggunaan I/O, dan akan memblok seterusnya hingga ada byte data berikutnya untuk dibaca. Apabila akhir dari stream tercapai, nilai -1 akan dimunculkan.
    - **int read(byte[ ] byteArray)** throws java.io.IOException : membaca serangkaian byte dan menempatkannya pada array byte yang ditunjuk dalam paramete input, dengan berulang-ulang

memanggil methode read() hingga array byte tersebut penuh terisi atau tidak ada lagi data yang tersedia untuk di baca. Method ini akan mengembalikan nilai integer jumlah byte data yang dibaca secara sukses, atau -1 apabila akhir stream telah tercapai.

– **int read(byte [ ] byteArray, int offset, int length)** throws java.io.IOException, java.lang.IndexOutOfBoundsException
– **void reset( )** throws java.io.IOException : mengembalikan posisi stream ke posisi yang telah ditandai oleh method mark().
– **long skip(long amount)** throws java.io.IOException : membaca data byte, tapi diabaikan selama sejumlah nilai yang diinputkan dalam parameter (long amount). Data yang dibaca tidak disimpan, tetapi proses pembacaan terus mengalir maju.

3.3. Penggunaan Input Stream

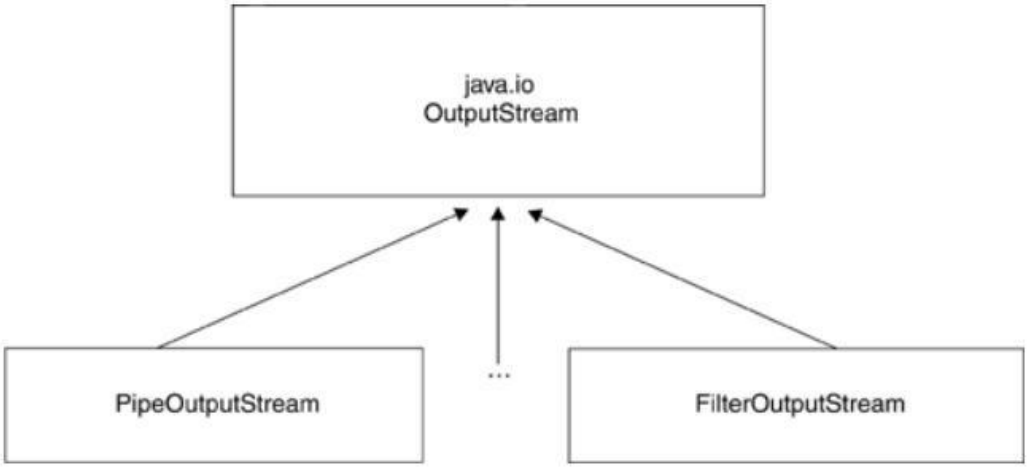
Contoh penggunaan kelas turunan InputStream untuk menampilkan isi dari sebuah file dapat dilihat pada kode di bawah ini. Di mana file dimasukkan pada parameter awal pemanggilan program ini (nama file dimasukkan sebagai argumen).

```
import java.io.*;
public class FileInputStreamDemo
{
      public static void main(String args[])
      {
            if (args.length != 1)
            {
            System.err.println ("Syntax - FileInputStreamDemo
file");
                  return;
            }
            try
            {
            // Membuat input stream yang membaca dr file
            InputStream fileInput = new FileInputStream ( args[0]
);

            int data = fileInput.read(); // Baca byte ke 1

            while (data != -1) // ulangi : hingga end of
file (EOF) dicapai
                  {
                  System.out.write ( data ); // menampilkan byte
data ke console
                  data = fileInput.read(); // baca byte
                  berikutnya }
```

```
            fileInput.close(); // Close the file
            }
            catch (IOException ioe)
            {
                  System.err.println ("I/O error - " + ioe);
            }
}}
```

3.4. Output Stream

Paket OutputSteam terletak pada java.io.OutputStream. Output stream merupakan stream dimana kita bisa menuliskan infomasi untuk dikirim ke sesuatu penerima informasi.



Turunan output stream yang bisa langsung kita gunakan ada 6 kelas, yang memiliki fungsi masing-masing, seperti menuliskan string , menuliskan data ke file atau ke pipa komunikasi. Output stream merupakan produsen informasi, dia membuat byte dari informasi dan mengirimkannya ke tempat lain. Data dikomunikasikan secara sekuensial, data byte yang pertama masuk stream akan pertama kali keluar pula.

Low-Level Output Stream	Fungsi
ByteArrayOutputStream	Menuliskan data byte ke sebuah array byte

FileOutputStream	Menuliskan data byte ke sebuah file lokal
PipedOutputStream	Menuliskan data byte ke pipa komunikasi
StringBufferOutputStream	Menuliskan byte ke sebuah string buffer
System.err	Menuliskan data byte ke stream error pada konsol pengguna
System.out	Menuliskan data byte ke konsol pengguna

Method-method yang dimiliki oleh output stream antara lain adalah :

– **Void close( ) :** menutup output stream, dan memberitahu ujung stream satunya (penerima) bahwa stream telah berakhir.

– **Void flush( ) :** melakukan " *flushing*" data yang belum terkirim ke sisi penerima dari output stream. Untuk meningkatkan performasi aplikasi, sering kali stream diberi *buffer*, sehingga data-data disimpan terlebih dahulu sebelum dikirim. Method ini cukup penting bagi subkelas output stream yang digunakan pada operasi jaringan, dimana ' *flushing'* data selalu terjadi setelah terjadi operasi *request* atau *response* dikirimkan sehingga host lawan komunikasi tidak menunggu-nunggu data.

– **Void write(int byte) :** menuliskan byte yang diinputkan pada parameter method ini. Merupakan method abstrak yang di*override* oleh subklas-subklas turunan output stream.

– **Void write(byte[ ] byteArray)** : menuliskan isi dari array byte yang menjadi parameter method ini.

– **Void write(byte[ ] byteArray, int offset, int length)**

3.5. Penggunaan Output Stream

Di sini diberikan contoh penggunaan output stream pada sebuah kode aplikasi di mana aplikasi ini akan mengopy sebuah file dengan cara membaca isi dari suatu file dan menuliskannya ke file lain. Operasi dilakukan byte per byte.

```
import java.io.*;
public class FileOutputStreamDemo
{
      public static void main(String args[])
      {
            if (args.length != 2)          //  Two  parameters
are required, the source and destination
            {
                  System.err.println        ("Synta        -
                                               x
FileOutputStreamDemo src dest");
                  return;
            }
            String source = args[0];
            String destination = args[1];
            try
            {
                  // Open source file for input
                  InputStream  input  =  new  FileInputStream(
source );
                  System.out.println ("Opened " + source + "
for reading.");
                  OutputStream output = new FileOutputStream
( destination ); // Ouput output file for output
                  System.out.println ("Opened " +
destination + " for writing.");
                  int data = input.read();
                  while ( data != -1)
                  {
                        // Write byte of data to our
                        file output.write (data);
                        // Read next byte
                        data=input.read()
                        ;
                  }
                  // Close both streams
                  input.close();
                  output.close();
                  System.out.println ("I/O streams closed");
            }
            catch (IOException ioe)
            {
                  System.err.println ("I/O error - " + ioe);
            }
      }
}
```

3.6. Praktikum Modul 3

1. Tuliskan kode penggunaan Input Stream dan Output Stream yang ada pada sub bab 3.3 dan 3.5.

2. Buat modifikasi pada kode Input Stream di subbab 3.3, sehingga aplikasi tidak lagi membutuhkan argumen dalam eksekusinya, melainkan memunculkan dialog untuk membuka file yang ada pada komputer kita.

3. Buat modifikasi pada kode Output Stream di subbab 3.5 agar nama file input dan file output (hasil copy) ditampilkan dalam bentuk dialog.