INPUT DAN OUTPUT LANJUTAN

12.1. I/O Stream

Stream adalah proses membaca data dari suatu sumber (input) atau mengirimkan data ke suatu tujuan (output). Stream juga dikategorikan berdasarkan apakah mereka digunakan untuk membaca atau menulis stream. Walaupun ini sudah cukup nyata, kita diperbolehkan untuk membaca dari input stream tapi tidak diperbolehkan untuk menulisnya. Di lain pihak, kita diperbolehkan untuk menulis output stream tapi tidak diperbolehkan untuk membacanya.

Class InputStream dan class Reader adalah superclass-superclass dari semua input stream. Class OutputStream dan class Writer adalah class-class root dari semua output stream.

Input stream juga dikenal sebagai stream sumber (source stream) sejak kita memperoleh informasi dari stream ini. sementara itu output stream disebut juga stream hasil (sink stream).

System.out.println() adalah contoh stream, yang berfungsi menampilkan data (informasi) ke layar.

Berikut ini adalah variabel standard stream :

a. `System.in (default: keyboard)`
b. `System.out (default: layar)`
c. `System.err (default: console)`

12.1.1. Byte Stream

Byte Stream adalah abstraksi file atau alat untuk data biner. Byte Stream digunakan untuk menulis atau membaca data biner.

InputStream dan OutputStream adalah dua class abstrak tertinggi dari Byte Stream. Class InputStream adalah abstraksi class root untuk semua input byte stream sedangkan class OutputStream adalah class root abstraksi dari semua output byte stream.

Berikut ini adalah beberapa class turunan dari byte stream :

a. `BufferedInputStream`
   Sebuah subclass dari FilterInputStream yang memungkinkan penyimpanan input sementara untuk menyediakan pembacaan byte yang lebih efisien.
b. `BufferedOutputStream`
   Sebuah subclass dari FilterOutputStream yang memungkinkan penyimpanan output sementara untuk proses penulisan byte yang lebih efisien. Memungkinkan penulisan byte ke bentuk dasar output stream tanpa menyebabkan diperlukannya pemanggilan dasar sistem untuk setiap penulisan byte.
c. `FilterInputStream`
   Untuk membaca byte stream yang telah terfilter, yang mungkin memindahkan source dasar dari data sepanjang proses dan menyediakan fungsi tambahan.
d. `FilterOutputStream`
   Untuk menulis stream byte yang telah difilter, yang mana mungkin dipindahkan ke source dasar dari data sepanjang proses dan menyediakan fungsi tambahan.

e. `ObjectInputStream`

Digunakan untuk serialisasi object. Deserialisasi obyek dan data primitif yang telah tertulis sebelumnya menggunakan sebuah ObjectOutputStream.

f. `ObjectOutputStream`
Digunakan untuk serialisasi object. Serialisasi object dan data primitif untuk sebuah OutputStream.

g. `DataInputStream`
Sebuah subclass dari FilterInputStream yang memerintahkan sebuah aplikasi membaca data primitif Java dari sebuah input stream dasar dalam sebuah Mesin yang berjalan secara bebas (machine-independent way).

h. `DataOutputStream`
Sebuah subclass dari FilterOutputStream yang menjalankan aplikasi penulisan data primitif ke output stream dasar ke dalam sebuah mesin yang bebas berjalan (machine independent way).

i. `PrintStream`
Sebuah subclass dari FilterOutputStream yang menyediakan kemampuan untuk mencetak representasi dari nilai data yang bermacam-macam dengan tepat.

j. `FileInputStream`
Untuk membaca baris byte dari sebuah file

k. `FileOutputStream`
Untuk menulis byte ke sebuah file.

l. `BufferedArrayOutputStream`
Mengimplementasikan sebuah penyimpan sementara berupa byte, yang mana mungkin akan dituliskan ke bentuk stream-nya.

m. `PipedInputStream`
Seharusnya terhubung ke sebuah PipedOutputStream. Stream ini secara khusus digunakan oleh dua urutan yang di dalamnya satu dari urutan tersebut membaca data dari sumber ini sementara urutan yang lain menulis ke PipedOutputStream tujuan.

n. `PipedOutputStream`
Seharusnya tersambung ke sebuah PipedInputStream. Stream ini secara khusus digunakan oleh dua urutan dimana di dalamnya satu dari urutan tersebut menulis data ke bentuk stream-nya sementara urutan yang lain membaca dari PipedInputStream tujuan.

o. `LineNumberInputStream`
Sebuah subclass FilterInputStream yang memungkinkan pemeriksaan posisi dari nomor baris tertentu.

p. `PushbackInputStream`
Sebuah subclass dari class FilterInputStream yang memungkinkan byte diproses balik atau tidak dibaca ke bentuk stream-nya.

q. `BufferedArrayInputStream`
Mengimplementasikan sebua penimpan sementara yang terdiri atas data byte, yang mungkin dpat dibaca dari stream-nya.

Sedangkan metode dari Class InputStream antara lain :

a. `public int read(-) throws IOException`
Method overloaded, juga memiliki tiga versi seperti class Reader tersebut.

b. `public abstract int read()`

Membaca byte selanjutnya dari data dari stream ini.

c. `public int read(byte[] bBuf)`
Membaca sejumlah byte dan menyimpannya dalam byte array bBuf.

d. `public abstract int read(char[] cbuf, int offset, int length)`
Membaca panjang sejumlah length byte dan menyimpannya dalam array byte bBuf dimulai dari offset tertentu.

e. `public abstract void close() throws IOException`
Menutup stream. Memanggil metode InputStream yang lain setelah menutup streamnya akan menyebabkan sebuah IOException dijalankan.

f. `public void mark(int readAheadLimit) throws IOException`
Menandai posisi tertentu dalam stream. Setelah menandainya, panggil untuk menjalankan fungsi reset() akan mencoba untuk mengatur posisi streamnya pada titik tertentu kembali. Tidak semua stream input-byte mendukung operasi ini.

g. `public boolean markSupported()`
Mengindikasikan apakah suatu stream mendukung operasi pemberian tanda (mark) dan reset. Yang tidak didukung secara default. Seharusnya diubah menjadi overide oleh subclass.

h. `public void reset() throws IOException`
Merubah posisi stream pada posisi akhir yang diberi tanda (mark).

Berikut ini adalah beberapa contoh penggunaan Class InputStream :

A. Membaca Input dari Console

```
import java.io.*;
class ContohInputStream{
  public static void main(String[] args) throws IOException{
    byte[] data = new byte[10];

    System.out.println("Ketik 10 buah karakter:");
    System.in.read(data);

    System.out.println("Karakter yang anda ketik adalah:");
    for(int i=0; i<data.length; i++){
      System.out.print((char) data[i]);
    }
  }
}
```

B. Membaca Input dari File

```
import java.io.*;
class ContohFileInputStream{
  public static void main(String[] args) {
    if (args.length == 0) { System.out.println("Masukkan nama
file sebagai parameter!"); }

    byte data;
    FileInputStream fin=null;

    try {
      fin = new FileInputStream(args[0]);
      do {
        data = (byte) fin.read();
        System.out.print((char)data);
      } while(data != -1);
```

```
    } catch(FileNotFoundException e) {
      System.out.println("File: " + args[0] + "tidak
ditemukan.");
    } catch(IOException e) {
      System.out.println("Ekspresi tidak diketahui : " + e) ;
    } finally {
      if (fin != null) {
        try {
            fin.close();
        } catch(IOException err) {
            System.out.println("Ekspresi tidak diketahui : " +
err);
        }
      }
    }
  }
}
```

Metode dari Class OutputStream antara lain :

a. `public void write(-) throws IOException`
   Sebuah metode overloaded untuk menulis bentuk byte ke bentuk stream.
b. `public abstract void write(int b)`
   Menulis nilai byte khusus b ke bentuk output stream-nya.
c. `public void write(byte[] bBuf)`
   Menulis isi dari array byte bBuf ke bentuk stream-nya.
d. `public void write(byte[] bBuf, int offset, int length)`
   Menulis sejumlah length byte dari array bBuf ke bentuk stream-nya, dimulai pada offset khusus ke stream-nya.
e. `public abstract void close() throws IOException`
   Menutup stream ini dan mengeluarkan beberapa sumber dari sistem digabungkan dengan stream-nya. Penggunaan metode lain setelah memanggil metode ini akan menyebabkan sebuah IOException dijalankan.
f. `public abstract void flush()`
   Mengganti stream (sebagai contoh, data byte tersimpan dalam buffer akan segera ditulis dalam tujuan yang dimaksud).

Berikut ini adalah beberapa contoh penggunaan Class OutputStream :

A. Menulis Output ke Console

```
import java.io.*;
class ContohOutputStream{
  public static void main(String[] args) throws IOException{
    byte[] data = {'a','b','c','d','e','f','g'};

    System.out.write(data,3,4);
    System.out.write('\n');
    System.out.write(data);
  }
}
```

B. Menulis Output ke File

```
import java.io.*;
class ContohFileOutputStream{
  public static void main (String[] args) {
```

```
     if (args.length==0) {System.out.println("Error: tulis nama
file!");}

     byte data;
     FileOutputStream fout=null;

     try {
       fout = new FileOutputStream(args[0]);
       System.out.println ("Ketik data yang ingin Anda tulis ke
file. Ketik \"Q\" untuk mengakhiri");
       data = (byte)System.in.read();

       while (data != (byte)'Q') {
         fout.write(data);
         data = (byte) System.in.read();
       }
     } catch(FileNotFoundException e) {
       System.out.println("file : " + args[0] + " tidak dapat
dibuka atau dibuat.");
     } catch(IOException e) {
       System.out.println("Ekspresi tidak diketahui : " + e);
     } finally {
       if (fout != null) {
         try {
             fout.close();
         } catch(IOException err) {
             System.out.println("Ekspresi tidak diketahui : " +
err);
           }
       }
     }
   }
 }
```

C. Program Copy Isi File (Byte)

```
 import java.io.*;
 public class FileCopyBytes {
   public static void main(String[] args) throws IOException {
     if (args.length<2) {System.out.println("Error: tulis nama
file sumber dan tujuan !");}

     FileInputStream in = null; FileOutputStream out = null;

    try {
       in = new FileInputStream(args[0]);
       out = new FileOutputStream(args[1]);

       int c;
       while ((c = in.read()) != -1) {
         out.write(c);
       }
     } finally {
       if (in != null) {
         in.close();
       }
       if (out != null) {
         out.close();
       }
     }
   }
 }
```

12.1.2. Character Stream

Character Stream digunakan untuk menulis dan membaca data karakter (unicode). Class Reader dan Writer adalah dua class abstrak tertinggi dari Character Stream.

Berikut ini adalah beberapa class turunan dari character stream :

a. `BufferedReader`
   Mengizinkan penyimpanan sementara karakter yang bertujuan untuk menyediakan fasilitas pembacaan karakter, arrays, dan baris yang lebih efisien.
b. `BufferedWriter`
   Menyediakan penyangga karakter bertujuan untuk menyediakan efisiensi penulisan karakter, array, dan baris.
c. `CharArrayReader`
   Mengimplementasikan suatu karakter buffer yang dapat dibaca.
d. `CharArrayWriter`
   Menggunakan karakter penyangga yang dapat dituliskan juga.
e. `FileReader`
   Untuk membaca file-file karakter.
f. `FileWriter`
   Untuk menulis karakter ke sebuah file.
g. `FilterReader`
   Untuk membaca stream karakter yang telah terfilter.
h. `FilterWriter`
   Untuk menulis stream karakter yang difilter.
i. `InputStreamReader`
   Menkonversi pembacaan byte ke bentuk karakter.
j. `LineNumberReader`
   Sebuah subclass dari class BufferedReader yang dapat menjaga memori penyimpanan untuk nomor baris.
k. `OutputStreamWriter`
   Mengkodekan karakter yang ditulis ke dalam byte.
l. `PipedReader`
   Digunakan untuk pasangan (dengan sebuah PipedWriter yang sesuai) oleh dua urutan yang ingin berkomunikasi. Salah satu dari urutan tersebut membaca karakter dari sumber tertentu.
m. `PipedWriter`
   Digunakan dengan berpasangan (dengan menghubungkan PipedReader) oleh dua thread yang ingin berkomunikasi. Satu dari thread ini menulis karakter ke stream ini.
n. `PrintWriter`
   Mencetak representasi yang diformat dari object ke dala stream text-output.
o. `PushbackReader`
   Sebuah subclass dari class FilterReader yang memungkinkan karakter dikembalikan atau tidak terbaca oleh stream.
p. `StringReader`
   Untuk membaca dari sebuah sumber string.
q. `StringWriter`
   Untuk menulis source string.

Sedangkan metode dari Class Reader antara lain :

a. `public int read(-) throws IOException`
   Sebuah metode overload, yang mana memiliki tiga versi. Membaca karakter, segala karakter array atau sebuah porsi untuk sebuah karakter array.
b. `public int read()`
   Membaca sebuah karakter tunggal.
c. `public int read(char[] cbuf)`
   Membaca karakter dan menyimpannya dalam karakter array cbuf.
d. `public abstract int read(char[] cbuf, int offset, int length)`
   Membaca karakter sejumlah panjang karakter tertentu dan menyimpannya dalam karakter cbuf dimulai pada tanda offset khusus yang telah ditentukan.
e. `public abstract void close() throws IOException`
   Menutup Stream ini. Memanggil metode Reader yang lain setelah menutup stream akan menyebabkan suatu IOException dijalankan.
f. `public void mark(int readAheadLimit) throws IOException`
   Menandai posisi tertentu pada stream. Setelah menandai, panggil untuk melakukan reset() kemudian stream akan mencoba mengatur posisinya kembali pada titik ini. Tidak semua stream input karakter mendukung operasi ini.
g. `public boolean markSupported()`
   Mengindikasikan apakah sebuah stream mendukung operasi pemberian tanda (mark) atau tidak Tidak didukung oleh default. Seharusnya bersifat overide subclass.
h. `public void reset() throws IOException`
   Reposisi stream ke posisi akhir stream yang telah ditandai.

Berikut ini adalah beberapa contoh penggunaan Class Reader :

A. Membaca Input dari Console (Karakter)

```
import java.io.*;
class ContohCharReader{
  public static void main(String[] args) throws IOException{
    char data;
    String str = "";
    BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));

    System.out.println("Ketik sejumlah karakter, akhiri dengan
\"Q\"");
    data =(char) br.read();
    while (data != 'Q') {
      str += data;
      data = (char) br.read();
    }
    System.out.println("Karakter yang anda ketik : " + str);
  }
}
```

B. Membaca Input dari Console (Baris) -1-

```
import java.io.*;
class ContohLineReader{
  public static void main(String[] args) throws IOException{
    String hasil = "";
```

```
    String str;
    BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));

    System.out.println("Ketik sejumlah string dan akhiri dengan
KELUAR");
    str = br.readLine();
    while (!str.equals("KELUAR")){
      hasil += str + '\n';
      str = br.readLine();
    }
    System.out.println();
    System.out.println("String yang anda ketik: " + hasil);
  }
 }
```

C. Membaca Input dari Console (Baris) -2-

```
 import java.io.*;
 public class ContohLineReader2{
   public static void main( String[] args ){
     BufferedReader data = new BufferedReader(new
InputStreamReader( System.in));
     String nama = "";

     System.out.print("Masukkan Nama Anda : ");
     try {
       nama= data.readLine();
     } catch( IOException e ){
       System.out.println("Ada Kesalahan!");
     }
     System.out.println("Halo " + nama +"!");
   }
 }
```

D. Membaca Input dari File

```
 import java.io.*;
 class ContohFileReader{
   public static void main (String args[]) {
     if (args.length==0) {System.out.println("Error: tulis nama
file!");}

     String data;
     BufferedReader br = null;
     try {
       Reader fin = new FileReader(args[0]);
       br = new BufferedReader(fin);

       do {
         data = br.readLine();
         System.out.println(data);
       } while(data != null);
     } catch(FileNotFoundException e) {
       System.out.println("File : " + args[0] + "tidak
ditemukan.");
     } catch(IOException e) {
       System.out.println("Ekspresi tidak diketahui : " + e);
     } finally {
       if (br != null) {
         try {
           br.close();
```

```
          } catch(IOException err) {
            System.out.println("Ekspresi tidak diketahui : " +
err);
          }
        }
      }
    }
  }
```

Metode dari Class Writer antara lain :

a. `public void write(-) throws IOException`
   Sebuah metode overloading dalam lima versi:
b. `public void write(int c)`
   Menulis sebuah karakter tunggal yang diwakili oleh pemberian nilai integer.
c. `public void write(char[] cbuf)`
   Menulis isi dari karakter array cbuf.
d. `public abstract void write(char[] cbuf, int offset, int length)`
   Menulis sejumlah length karakter dari array cbuf, dimulai pada offset tertentu.
e. `public void write(String str)`
   Menulis string string.
f. `public void write(String str, int offset, int length)`
   Menulis sejumlah length karakter dari string str, dimulai pada offset tertentu.
g. `public abstract void close() throws IOException`
   Menutup stream ini setelah flushing beberapa karakter yang tidak tertulis. Invocation method lain setelah menutup stream ini akan menyebabkan terjadinya IOException.
h. `public abstract void flush()`
   Mengganti stream (yaitu karakter yang disimpan dalam buffer dengan segera ditulis ke tujuan yang dimaksud).

Berikut ini adalah beberapa contoh penggunaan Class Writer :

A. Menulis Output ke Console

```
import java.io.*;
class ContohWriter{
  public static void main(String[] args) throws IOException{
    PrintWriter pw = new PrintWriter(System.out,true);

    pw.println("Menulis ke console dengan karakter stream");
  }
}
```

B. Menulis Output ke File

```
import java.io.*;
class ContohFileWriter{
  public static void main (String[] args) {
    if (args.length==0){ System.out.println("Error: tulis nama
file!");}

    String data; FileWriter fout=null;
    BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
```

```
     try {
       fout = new FileWriter(args[0]);
       System.out.println("Ketik teks yang akan disimpan ke file.
Ketik STOP untuk berhenti");

       data = br.readLine();
       while(!data.equals("STOP")) {
       fout.write (data + "\r\n");
       data = br.readLine();}
     } catch(FileNotFoundException e) {
       System.out.println("File : " + args[0] + " tidak dapat
dibuka atau dibuat.");
     } catch(IOException e) {
       System.out.println("Ekspresi tidak diketahui : " + e);
     } finally {
       if (fout != null) {
         try {
             fout.close();
         } catch(IOException err) {
           System.out.println("Ekspresi tidak diketahui : " +
err);
         }
       }
     }
   }
 }
```

C. Program Copy Isi File (Character)

```
 import java.io.*;
 public class FileCopyCharacters {
   public static void main(String[] args) throws IOException {
     if (args.length<2){ System.out.println("Error: tulis dua
nama file!");}

     FileReader inputStream = null;
     FileWriter outputStream = null;
     try {
       inputStream = new FileReader(args[0]);

       outputStream = new FileWriter(args[1]);
       int c;
       while ((c = inputStream.read()) != -1) {
       outputStream.write(c); }
     } finally {
       if (inputStream != null) {
         inputStream.close();
       }
       if (outputStream != null) {
         outputStream.close();
       }
     }
   }
 }
```

D. Program Copy Isi File (Line)

```
 import java.io.*;
 public class FileCopyLines {
   public static void main(String[] args) throws IOException {
     if (args.length<2){ System.out.println("Error: tulis dua
nama file!");}
```

```
    BufferedReader inputStream = null;
    PrintWriter outputStream = null;
    try {
      inputStream = new BufferedReader(new FileReader(args[0]));
      outputStream = new PrintWriter(new FileWriter(args[1]));
      String l;
      while ((l = inputStream.readLine()) != null) {
        outputStream.println(l);
      }
    } finally {
      if (inputStream != null) {
        inputStream.close();
      }
      if (outputStream != null) {
        outputStream.close();
      }
    }
  }
}
```

12.2. Operasi File

Walaupun class file bukan merupakan class stream, ini sesuatu yang penting bahwa kita mempelajari ini sejak class-class stream merupakan file-file yang telah dimanipulasi. Class file adalah sebuah perwakilan dari abstraksi dari file-file nyata dan nama direktori. Class File adalah class yang mendukung berbagai operasi yang berhubungan dengan berkas (file) dan direktori (folder). Penggunaannya misalnya untuk mengganti nama atau membuat file atau direktori.

Untuk meng-instantiate sebuah object File, Anda dapat menggunakan constructor berikut ini:

a. File(String nama)
   Membuat file instance dengan mengkonversi nama path yang diberikan menjadi nama path abstrak.
b. File(String induk, String anak)
   Membuat file instance baru dari nama path induk dan nama path anak.
c. File(File induk, String anak)
   Membuat file instance baru dari nama path abstrak induk dan nama path anak.

Metode dari Class File antara lain :

a. `public String getName()`
   Mengembalikan nilai nama file atau nama direktori dari obyek File ini.
b. `public boolean exists()`
   Menguji apakah sebuah file atau sebuah direktori masih ada atau tidak.
c. `public long length()`
   Mengembalikan nilai ukuran dari file.
d. `public long lastModified()`
   Mengembalikan nilai tanggal dalam milidetik ketika file terakhir kali dimodifikasi.
e. `public boolean canRead()`
   Mengembalikan nilai true jika dijinkan untuk membaca dari file. Sebaliknya, nilai pengembaliannya bernilai false.

f. `public boolean canWrite()`
mengembalikan nilai true jika diijinkan untuk menulis ke sebuah file. Sebaliknya, nilai pengembaliannya bernilai false.

g. `public boolean isFile()`
Menguji apakah obyek ini berupa sebuah file, yaitu persepsi normal kita tentang apa itu sebuah file (bukan sebuah direktori) atau bukan.

h. `public boolean isDirectory()`
Menguji apakah obyek ini adalah sebuah direktori atau bukan.

i. `public String[] list()`
Mengembalikan nilai daftar file dan subdirektori yang ada dalam obyek ini. Obyek ini haruslah berupa sebuah direktori.

j. `public void mkdir()`
Membuat sebuah direktori yang merupakan abstraksi nama path ini.

k. `public void delete()`
Membuang file atau direktori yang sebenarnya diwakili oleh obyek File tersebut.

Berikut ini adalah beberapa contoh penggunaan Class File :

A. Menampilkan Atribut File

```
import java.io.*;
public class InfoFile{
  public static void main(String[] args){
    BufferedReader StreamTeks=new BufferedReader(new
InputStreamReader (System.in));
    System.out.println("Masukan nama file:"); String namaBerkas
= "";
    try {
      namaBerkas = StreamTeks.readLine();
    } catch (IOException i){
    }

    File berkas = new File(namaBerkas);
    if (!berkas.exists()){ System.out.println("Berkas ini tak
ada");}
    if ( berkas.isDirectory()) System.out.println("Direktori");
    if (berkas.isFile()) System.out.println("Berkas biasa");
    if (berkas.isHidden()) System.out.println("Tersembunyi");
    if (berkas.canRead()) System.out.println("Bisa dibaca");
    if (berkas.canWrite()) System.out.println("Bisa ditulisi");
    if (berkas.isAbsolute()) System.out.println("path absolut");
    else System.out.println("path relatif");

    System.out.println("Induk : " + berkas.getParent());
    System.out.println("Path : " + berkas.getPath());
    System.out.println("Path Absolut : " +
berkas.getAbsolutePath());
    System.out.println("Nama : " + berkas.getName());
    System.out.println("Ukuran : " + berkas.length() +" byte");
  }
}
```

B. Menghapus File

```
import java.io.*;
public class HapusFile{
  public static void main (String[] args) {
```

```
     if (args.length==0) {System.out.println("Error: tulis nama
file yang akan dihapus!");}

     String namaFile = args[0];

     File berkas = new File (namaFile);
     if (berkas.exists()) {
       System.out.println("Berkas " + namaFile + " ada");

       berkas.delete();

       System.out.println("Setelah penghapusan....");
       if (berkas.exists()) System.out.println("Berkas " +
namaFile + " ada");
       else System.out.println("Berkas " + namaFile + " sudah
dihapus");
     }
     else {
       System.out.println("Berkas " + namaFile + " tidak ada");
     }
   }
 }
```

C. Mengganti Nama File

```
 import java.io.*;
 public class GantiNamaFile {
   public static void main(String[] args) {
     if (args.length<2) {System.out.println("Error: tulis nama
file lama dan baru!");}

     File berkasSemula= new File(args[0]);
     File berkasBaru = new File(args[1]);
     if (berkasSemula.exists()) {
       if (berkasBaru.exists()) {
         System.out.println("Sudah ada file dengan nama : "
+berkasBaru);
       }
       else {
         berkasSemula.renameTo(berkasBaru);
           System.out.println("Nama file " + berkasSemula + "
sudah diganti menjadi " + berkasBaru);
       }
     }
     else {
       System.out.println("Nama file " + berkasSemula + " tidak
ada");
     }
   }
 }
```

D. Membuat Direktori

```
 import java.io.*;
 public class BuatDir {
   public static void main(String [] args) {
     if (args.length==0) {System.out.println("Error: tulis nama
direktori yang akan dibuat!");}

     String namaDir = args[0];
     File dir = new File (namaDir);
     dir.mkdir();
```

```
  }
}
```

E. Menampilkan Isi Direktori

```
import java.io.*;
public class ListDir {
  public static void main(String[] args) {
    String namaDir = ".";

    File dir = new File(namaDir);
    String dafFile[] = dir.list();
    for (int i=0; i < dafFile.length; i++)
      System.out.println(dafFile[i]);
  }
}
```

F. Mengakses Data NonSekuensial

```
import java.io.*;
public class ContohRandomAccessFile{
  public static void main(String [] args) throws IOException{
    RandomAccessFile berkas = new
RandomAccessFile("latihan.txt", "rw");
    berkas.writeBytes("ABCDEFGHIJKLMNOPQRSTUVW");
    char kar = ' ';

    berkas.seek(0); System.out.println("isi berkas: ");
    while(berkas.getFilePointer() < berkas.length()) {
    kar = (char) berkas.readByte(); System.out.print(kar);}
    System.out.println();

    berkas.seek(3); berkas.writeByte((int)'z');
    System.out.println("Sesudah penggantian");
    berkas.seek(0); System.out.println("isi berkas: ");
    while (berkas.getFilePointer() < berkas.length()) {
      kar = (char) berkas.readByte(); System.out.print(kar);
    }
  }
}
```