Bab 12

Stream I/O Lanjut

12.1 Tujuan

Dalam module sebelumnya, Anda telah mempelajari bagaimana untuk mendapatkan input user dan memanipulasi file-file menggunakan strea. Kini Anda akan mempelajari lebih banyak tentang stream dan class-class stream yang lain.

Setelah melengkapi pembahasan ini, Anda harus dapat :

1. Tahu tipe-tipe stream yang umum digunakan
2. Menggunakan class File dan methodnya
   - Karakter dan Stream byte
   - Input dan Output Streams
   - Node dan Filter Streams
3. Menggunakan class-class Input/Output yang berbeda
   - *Reader*
   - *Writer*
   - *InputStream*
   - *OutputStream*
4. Memahami konsep dari stream chaining
5. Mendefinisikan serialisasi
6. Memahami penggunaan dari kata kunci *transient*
7. Menulis dan membaca dari sebuah object stream

12.2 Tipe-Tipe Stream yang Umum Digunakan

12.2.1 Stream Karakter dan Byte

Seperti yang elah disebutkan sebelumnya, secara umum ada dua tipe dari stream, yaitu stream karakter dan byte. Kita hanya mengulang perbedaan mendasar antara keduanya. Stream byte adalah abstraksi file atau alat untuk data biner sedangkan stream karakter adalah untuk karakter Unicode.

Class *InputStream* adalah abstraksi class root untuk semua input stream byte sedangkan class *OutputStream* adalah class root abstraksi dari semua output stream byte. Untuk stream karakter, superclasss yang sesuai dari semua class-class secara berturut-turut adalah class *Reader* dan the *Writer.* Kedua class-class ini adalah abstraksi class-class untuk membaca dan menulis stream karakter.

12.2.2 Input dan Output Stream

Stream juga dikategorikan berdasarkan apakah mereka digunakan untuk membaca atau menulis stream.Walaupun ini sudah cukup nyata, izinkan saya untuk mendefinisikan tipe

stream ini. Anda diijinan untuk membaca dari input stream tapi tidak menulisnya. Di lain pihak, Anda diijinkan untuk menulis output streams tapi tidak 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).

12.2.3 Node dan Stream Filter

Kini package *java.io* membedakan antara node dan stream filter. Sebuah stream node adalah sebuah stream dengan fungsi dasar berupa fungsi membaca atau menulis dari sebuah lokasi khusus seperti pada disk atau dari jaringan. Tipe-tipe dari stream node terdiri atas file, memory dan jalur data. Stream filter, di lain pihak, diletakkan pada layer stream node diantara threads atau proses untuk menyediakan fungsi tambahan yang tidak dapat ditemukan dalam stream node oleh stream node itu sendiri. Penambahan lapisan pada sebuah stream node disebut dengan stream chaining.

Sesi ini berturut-turut mempunyai sebuah tujuan dariclass-class stream yang berbeda. Untuk melengkapi daftar dari class-class ini, silahkan melihat dokumentasi Java's API.

12.3 Class File

Walaupun class *File* bukan merupakan class stream, ini sesuatu yag penting bahwa kita mempelajari ini sejak class-class stream merupakan file-file yang telah dimanipulasi. Class adalah sebuah perwakilan dari abstraksi dari file-file nyata dan nama path direktori.

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

Sebuah Constructor File
`File(String pathname)`
Instantiate sebuah object *File* dengan nama path khusus sebagai nama filenya. Nama filenya mungkin salah satu menjadi penuh(sebagai contoh, isi dengan path yang lengkap) atau mungkin terdiri atas namafile itu sendiri dan diasumsikan menjadi diisi dalam direktori tersebut.

Table 1.2a: File constructor

Class *File* menyediakan beberapa method untuk memanipulasi file dan direktori. Berikut ini beberapa dari method-method tersebut.

Method-method File
`public String getName()`
Mengembalikan nilai nama file atau nama direktori dari object *File* ini.
`public boolean exists()`
Menguji apakah sebuah file atau sebuah direktori masih ada atau tidak
`public long length()`
Mengembalikan nilai ukuran dari file.
`public long lastModified()`
Mengembalikan nilai tanggal dalam milidetik ketika file terakhir kali dimodifikasi.
`public boolean canRead()`
Mengembalikan nilai true jika dijinkan untuk membaca dari file. Sebaliknya, nilai pengembaliannya bernilai false.
`public boolean canWrite()`
mengembalikan nilai true jika diijinkan untuk menulis ke sebuah file. Sebaliknya, nilai pengembaliannya bernilai false.
`public boolean isFile()`
Menguji apakah object ini berupa sebuah file, yaitu persepsi normal kita tentang apa itu sebuah file (bukan sebuah direktori) atau bukan.
`public boolean isDirectory()`
menguji apakah object ini adalah sebuah direktori atau bukan.

Method-method File
`public String[] list()`
Mengembalikan nilai daftar file dan subdirektori yang ada dalam object ini. Object ini haruslah berupa sebuah direktori.
`public void mkdir()`
Membuat sebuah direktori yang merupakan abstraksi nama path ini.
`public void delete()`
Membuang file atau direktori yang sebenarnya diwakili oleh object File tersebut.

Table 1.2a: method-method File

Mari kita melihat bagaimana method ini bekerja berdasarkan contoh berikut ini :

```
import java.io.*;

public class FileInfoClass {
   public static void main(String args[]) {
      String fileName = args[0];
      File fn = new File(fileName);
      System.out.println("Name: " + fn.getName());
      if (!fn.exists()) {
         System.out.println(fileName + " does not exists.");
         /* membuat sebuah temporary directory . */
         System.out.println("Creating temp directory...");
         fileName = "temp";
         fn = new File(fileName);
         fn.mkdir();
         System.out.println(fileName +
                    (fn.exists()? "exists": "does not exist"));
         System.out.println("Deleting temp directory...");
         fn.delete();
         System.out.println(fileName +
                    (fn.exists()? "exists": "does not exist"));
         return;
      }
      System.out.println(fileName + " is a " +
                      (fn.isFile()? "file." :"directory."));
      if (fn.isDirectory()) {
         String content[] = fn.list();
         System.out.println("The content of this directory:");
         for (int i = 0; i < content.length; i++) {
            System.out.println(content[i]);
         }
      }
      if (!fn.canRead()) {
         System.out.println(fileName + " is not readable.");
         return;
      }
      System.out.println(fileName + " is " + fn.length() +
                                              " bytes long.");
      System.out.println(fileName + " is " + fn.lastModified()
                                             + " bytes long.");
```

```
                if (!fn.canWrite()) {
                   System.out.println(fileName + " is not writable.");
                }
             }
          }
```

12.4 Class Reader

Bagian ini menggambarkan stream karakter yang digunakan untuk membaca.

12.4.1 Method Reader

Class *Reader* terdiri atas beberapa method untuk membaca karakter. berikut ini adalah beberapa method class :

Method Reader
`public int read(-) throws IOException`
Sebuah method overload, yang mana memiliki tiga versi. Membaca karakter, segala karakter array atau sebuah porsi untuk sebuah karakter array.
`public int read()` - membaca sebuah karakter tunggal.
`public int read(char[] cbuf)`- Membaca karakter dan menyimpannya dalam karakter array *cbuf*.
`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.
`public abstract void close() throws IOException`
Menutup Stream ini. Memanggil method *Reader* yang lain setelah menutup stream akan menyebabkan suatu *IOException* dijalankan.
`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.
`public boolean markSupported()`
mengindikasikan apakah sebuah stream mendukung operasi pemberian tanda (mark) atau tidak Tidak didukung oleh default. Seharusnya bersifat overid subclass.
`public void reset() throws IOException`
Reposisi stream ke posisi akhir stream yang telah ditandai

Table 1.3.1: Reader methods

12.4.2 Class Node Reader

Berikut ini adalah beberapa dasar class *Reader*:

Class-class Node Reader
`FileReader`
Untuk membaca file-file karakter.
`CharArrayReader`
Mengimplementasikan suatu karakter buffer yang dapat dibaca.
`StringReader`
Untuk membaca dari sebuah sumber string.
`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.

Table 1.3.2: Class-class Node Reader Classes

12.4.3 Class-Class Filter Reader

Untuk menambah fungsi ke class-class dasar *Reader*, Anda dapat menggunakan class stream filter. Berikut ini adalah beberapa dari class-class tersebut :

Class-Class Filter Reader
`BufferedReader`
mengizinkan penyimpanan sementara karakteryang bertujuan untuk menyediakan fasilitas pembacaan karakter, arrays, dan bais yang lebih efisien.
`FilterReader`
Untuk membaca stream karakter yang telah terfilter.
`InputStreamReader`
Menkonversi pembacaan byte ke bentuk karakter.
`LineNumberReader`
Sebuah subclass dari class *BufferedReader* yang dapat menjaga memori penyimpanan untuk nomor baris.
`PushbackReader`
Sebuah subclass dari class *FilterReader* yang memungkinkan karakter dikembalikan atau tidak terbaca oleh stream.

Table 1.3.3: Class-class Filter Reader

12.5 Class-Class Writer

Sesi ini menggambarkan stream karakter yang digunakan untuk menulis.

12.5.1 Writer Method

Class *Writer* terdiri atas beberapa method untuk menulis karakter. Berikut ini adalah beberapa method class :

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

Table 1.4.1: Method Writer

12.5.2 Node Writer Classes

Berikut ini beberapa dasar class *Writer*:

Node Writer Classes
`FileWriter`
Untuk menulis karakter ke sebuah file.
`CharArrayWriter`
Menggunakan karakter penyangga yang dapat dituliskan juga.
`StringWriter`
Untuk menulis source string
`PipedWriter`

Node Writer Classes
Digunakan dengan berpasangan(dengan menghubungkan *PipedReader*) oleh dua thread yang ingin berkomunikasi. Satu dari thread ini menulis karakter ke stream ini.

Table 1.4.2: Node Writer classes

12.5.3 Filter Writer Classes

Untuk menambah fungsionalitas ke dasar class Writer, Anda dapat menggunakan class stream filter.Terdapat beberapa class-class:

Filter Writer Classes
`BufferedWriter`
Menyediakan penyangga karakter bertujuan untuk menyediakan efisiensi penulisan karakter, array, dan garis.
`FilterWriter`
Untuk menulis stream karakter yang difilter.
`OutputStreamWriter`
Mengkodekan karakter yang ditulis ke dalam byte.
`PrintWriter`
Mencetak representasi yang diformat dari object ke dala stream text-output.

Table 1.4.3: Filter Writer classes

12.6 Contoh Dasar Reader/Writer

Contoh penggantian menggunakan class *FileReader* dan *FileWriter.* Dalam contoh ini, program membaca dari file yang khusus oleh user dan mengkopi isi dari file ke file lain.

```
import java.io.*;

class CopyFile {
   void copy(String input, String output) {
      FileReader reader;
      FileWriter writer;
      int data;
      try {
         reader = new FileReader(input);
         writer = new FileWriter(output);
         while ((data = reader.read()) != -1) {
            writer.write(data);
         }
         reader.close();
         writer.close();
      } catch (IOException ie) {
         ie.printStackTrace();
      }
```

```
    }

    public static void main(String args[]) {
       String inputFile = args[0];
       String outputFile = args[1];
       CopyFile cf = new CopyFile();
       cf.copy(inputFile, outputFile);
    }
}
```

Cobalah program tersebut sendiri dan amatilah apa ang terjadi pada file yang dimanipulasi.

12.7 Merubah Contoh Reader/Writer

Contoh pengganti hampir sama dengan contoh sebelumnya tetapi lebih efisien. Walaupun membaca dan menulis ke stream sekali saja, karakter membaca yang pertama disimpan dalam buffer sebelum penulisan karakter baris per baris. Program menggunakan teknik dari perangkaian stream dimana clas *FileReader* dan *FileWriter* didekorasi dengan class *BufferedReader* dan *BufferedWriter,* berurutan.

```
import java.io.*;

class CopyFile {
   void copy(String input, String output) {
      BufferedReader reader;
      BufferedWriter writer;
      String data;
      try {
         reader = new BufferedReader(new FileReader(input));
         writer = new BufferedWriter(new FileWriter(output));
         while ((data = reader.readLine()) != null) {
            writer.write(data, 0, data.length());
         }
         reader.close();
         writer.close();
      } catch (IOException ie) {
         ie.printStackTrace();
      }
   }

   public static void main(String args[]) {
      String inputFile = args[0];
      String outputFile = args[1];
      CopyFile cf = new CopyFile();
      cf.copy(inputFile, outputFile);
   }
}
```

Bandingkan kode ini dengan sebelumnya. Apakah hasil dari menjalankan program ini?

12.8 Class InputStream

Bagian ini memberikan gambaran perbedaan stream byte yang digunakan membaca.

12.8.1 Method InputStream

Class *InputStream* terdiri atas beberapa method untuk membaca byte. Beberapa method class:

Method InputStream
`public int read(-) throws IOException`
Method overloaded, juga memiliki tiga versi seperti class Reader tersebut.
`public abstract int read()` - Membaca byte selanjutnya dari data dari stream ini.
`public int read(byte[] bBuf)`- Membaca sejumlah byte dan menyimpannya dalam byta array *bBuf*.
`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.
`public abstract void close() throws IOException`
Menutup stream in. Memanggil method *InputStream* yang lain setelah menutup streamnya akan menyebabkan sebuah *IOException* dijalankan.
`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.
`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.
`public void reset() throws IOException`
Merubah posisi stream pada posisi akhir yang diberi tanda (mark)

Table 1.7.1: Method InputStream

12.8.2 Class-Class Node InputStream

Berikut ini merupakan beberapaclass-class dasar *InputStream* :

Class-class Node InputStream
`FileInputStream`
Untuk membaca baris byte dari sebuah file
`BufferedArrayInputStream`

Class-class Node InputStream
Mengimplementasikan sebua penimpan sementara yang terdiri atas data byte, yang mungkin dpat dibaca dari streamnya.
`PipedInputStream`
Seharusnya terhubung ke sebuah *PipedOutputStream*. Stream ini secara khusus digunakan oleh dua urutan yang didalamnya satu dari urutan tersebut membaca data dari sumber ini sementara urutan yang lain menulis ke *PipedOutputStream* tujuan.

Table 1.7.2: class-class Node InputStream

12.8.3 Class-class Filter InputStream

Untuk menambah fungsi ke class dasar *InputStream*, Anda dapat menggunakan class stream filter. Berikut ini adalah beberapa dari class-class tersebut :

Class-class Filter InputStream
`BufferedInputStream`
Sebuah subclass dari *FilterInputStream* yang memungkinkan penyimpanan input sementara untuk menyediakan pembacaan byte yang lebih efisien.
`FilterInputStream`
Untuk membaca byte stream yang telah terfilter, yang mungkin memindahkan source dasar dari data sepanjang proses dan menyediakan fungsi tambahan.
`ObjectInputStream`
Digunakan untuk serialisasi object. Deserialisasi object dan data primitif yang telah tertulis sebelumnya menggunakan sebuah*ObjectOutputStream*.
`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).
`LineNumberInputStream`
Sebuah subclass *FilterInputStream* yang memungkinkan pemeriksaan posisi dari nomor baris tertentu.
`PushbackInputStream`
Sebuah subclass dari class *FilterInputStream* yang memungkinkan byte diproses balik atau tidak dibaca ke bentuk sreamnya.

Table 1.7.3: Class-class Filter InputStream

12.9 Class-Class OutputStream

Sesi ini memberikan sebuah pandangan tentang byte stream yang berbeda yang digunakan dalam proses penulisan.

12.9.1 Method OutputStream

Class *OutputStream* terdiri atas beberapa method untuk menulis data byte. Berikut ini adalah beberapa dari class methodnya :

Method OutputStream
`public void write(-) throws IOException`
Sebuah method overloaded untuk menulis bentuk byte ke bentuk stream. Ada tiga versi :
`public abstract void write(int b)` – Menulis nilai byte khusus *b* ke bentuk output stream nya.
`public void write(byte[] bBuf)` – Menulis isi dari array byte *bBuf* ke bentuk stream nya.
`public void write(byte[] bBuf, int offset, int length)` – Menulis sejumlah *length* byte dari array *bBuf* ke bentuk streamnya, dimulai pada *offset* khusus ke streamnya.
`public abstract void close() throws IOException`
Menutup stream ini dan mengeluarkan beberapa sumber dari sistem digabungkan dengan streamnya. Penggunaan method lain setelah memanggil method ini akan menyebabkan sebuah*IOException* dijalankan.
`public abstract void flush()`
Mengganti stream (sebagai contoh, data byte tersimpan dalam buffer akan segera ditulis dalam tujuan yang diamksud).

Table 1.8.1: Method OutputStream

12.9.2 Class-Class Node OutputStream

Berikut ini adalah beberapa dari class dasar *OutputStream* :

Clas-class Node OutputStream
`FileOutputStream`
Untuk menulis byte ke sebuah file.
`BufferedArrayOutputStream`
Mengimplementasikan sebuah penyimpan sementara berupa byte, yang mana mungkin akan dituliskan ke bentuk streamnya.
`PipedOutputStream`

Clas-class Node OutputStream
Seharusnya tersambung ke sebuah *PipedInputStream*. Stream ini secara khusus digunakan oleh dua urutan dimana didalamnya satu dari urutan tersebut menulis data ke bentuk streamnya sementara urutan yang lain membaca dari *PipedInputStream* tujuan.

Table 1.8.2: Class-class Node OutputStream

12.9.3 Class-Class Filter OutputStream

Untuk menambah fungsi ke class dasar *OutputStream*, Anda dapat menggunakan class stream filter.berikut ini beberapa dari class tersebut :

Class-Class Filter OutputStream
`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.
`FilterOutputStream`
Untuk menulis stream byte yang telah difilter, yang mana mungkin dipindahkan ke source dasar dari data sepanjang proses dan menyediakan fungsi tambahan.
`ObjectOutputStream`
Digunakan untuk serialisasi object. Serialisasi object dan data primitif untuk sebuah *OutputStream*.
`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).
`PrintStream`
Sebuah subclass dari *FilterOutputStream* yang menyediakan kemampuan untuk mencetak representasi dari nilai data yang bermacam-macam dengan tepat.

Table 1.8.3: Class-Class Filter OutputStream

12.10 Contoh Dasar InputStream/OutputStream

Contoh berikut ini menggunakan class *FileInputStream* dan *FileOutputStream* untuk membaca dari sebuah file khusus dan mengcopy isi dari file ini ke file yang lain.

```
import java.io.*;

class CopyFile {
   void copy(String input, String output) {
      FileInputStream inputStr;
      FileOutputStream outputStr;
      int data;
      try {
         inputStr = new FileInputStream(input);
         outputStr = new FileOutputStream(output);
         while ((data = inputStr.read()) != -1) {
            outputStr.write(data);
         }
         inputStr.close();
         outputStr.close();
      } catch (IOException ie) {
         ie.printStackTrace();
      }
   }

   public static void main(String args[]) {
      String inputFile = args[0];
      String outputFile = args[1];
      CopyFile cf = new CopyFile();
      cf.copy(inputFile, outputFile);
   }
}
```

12.11 Contoh Modifikasi InputStream/OutputStream

Contoh berikutnya menggunakan class *PushbackInputStream* yang memanfaatkan sebuah object *FileInputStream* dan class *PrintStream*.

```
import java.io.*;

class CopyFile {
   void copy(String input) {
      PushbackInputStream inputStr;
      PrintStream outputStr;
      int data;
      try {
         inputStr = new PushbackInputStream(new
                                    FileInputStream(input));
         outputStr = new PrintStream(System.out);
         while ((data = inputStr.read()) != -1) {
            outputStr.println("read data: " + (char) data);
            inputStr.unread(data);
            data = inputStr.read();
            outputStr.println("unread data: " + (char) data);
         }
         inputStr.close();
         outputStr.close();
      } catch (IOException ie) {
         ie.printStackTrace();
      }
   }

   public static void main(String args[]) {
      String inputFile = args[0];
      CopyFile cf = new CopyFile();
      cf.copy(inputFile);
   }
}
```

Uji kode ini pada sebuah file yang mengandung sedikit baris atau karakter.

12.12 Serialisasi

Java Virtual Machine (JVM) mendukung kemampuan untuk membaca atau menulis sebuah object ke bentuk stream. kemampuan ini disebut dengan serialisasi, proses "flattening" sebuah object sehinnga data tersebut dapat disimpan ke beberapa penyimpanan permanen atau dilewatkan ke object lain melalui class *OutputStream*. Ketika menulis sebuah object, ini merupakan hal yang penting bahwa keadaan tersebut sudah tertulis dan telah diserialisasi dari setiap objectnya dapat dibangun kembali sebagaimana dtaa tersebut dibaca. Menyimpan sebuah objectke beberapa tipe penyimpanan permanen yang dikenal sebagai persistence.

Stream yang digunakan untuk deserialisasi dan serialisasi secara berurutan adalah class *ObjectInputStream* dan *ObjectOutputStream* .

Untuk memungkinkan sebuah object diserialisasi (sebagai contoh dapat disimpan dan diurutkan), Class tersebut harus mengimplementasikan interface yang dapat diserialisasi. Class ini seharusnya juga menyediakan default constructor atau sebuah constructor tanpa argumen. Satu hal yang baik mengenai kemampuan untuk melakukan serialisasiyang dapat diturunkan, yang berarti kita tidak memiliki implementasi serialisasi pada setiap class. Ini berarti mengurangi pekrjaan untuk programmer. Anda hanya dapat mengimplementasikan serialisasi sekali sepanjang hirarki class.

12.12.1 Kata Kunci transient

Ketika suatu object diserialisasi, tempat hanya disediakan untuk data object. Method dan Constructor bukan merupakan bagian dari stream serialisasi. Ada beberapa object yang tidak diserialisasi kaena data yang diwakilinya berubah secara konstan. Beberapa contoh dari setiap object adalah object *FileInputStream* dan Object *Thread*. Sebuah *NotSerializableException* dijalankan jika operasi serialisasi gagal karena beberapa alasan.

Jangan berputus asa. Sebuah class yang mengandung object yang tidak diserialisasi dapat tetap diserialisasi jika penunjuk ke object non-serialisasi ditandai dengan katakunci *transient*. Pertimbangkan contoh berikut ini :

```
class MyClass implements Serializable {
   transient Thread thread; //try removing transient
   int data;
   /* beberapa data yang lain*/
}
```

Kata kunci *transient* mencagah data dari proses serialisasi. Object instantiasi dari class ini sekarang dapat ditulis ke sebuah *OutputStream*.

12.12.2 Serialisasi: Menulis Suatu Object Stream

Untuk menulis object ke sebuah stream, Anda perlu menggunakan class*ObjectOutputStream* class dan methodnya yaitu method *writeObject*. Method *writeObject* memiliki tanda sebagai berikut :

```
public final void writeObject(Object obj) throws IOException
```

dimana *obj* adalah object yang ditulis ke stream.

Contoh dibawah ini menuliskan sebuah object *Boolean* ke sebuah *ObjectOutputStream*. Class *Boolean* mengimplementasikan interface yang dapat di *Serialisasi*. Selanjutnya, Instantiasi object dari class ini dapat ditulis ke dan dibaca dari sebuah stream.

```
import java.io.*;

public class SerializeBoolean {
   SerializeBoolean() {
      Boolean booleanData = new Boolean("true");

      try {
         FileOutputStream fos = new
                              FileOutputStream("boolean.ser");
         ObjectOutputStream oos = new ObjectOutputStream(fos);
         oos.writeObject(booleanData);
         oos.close();
      } catch (IOException ie) {
         ie.printStackTrace();
      }
   }

   public static void main(String args[]) {
      SerializeBoolean sb = new SerializeBoolean();
   }
}
```

12.12.3 Deserialisasi: Membaca Sebuah Object Stream

Untuk membaca sebuah object dari sebuah stream, Anda perlu menggunakan class *ObjectInputStream* dan methodnya yaitu method *readObject*. Method *readObject* memiliki tanda sebagai berikut :

```
public final Object readObject()
             throws IOException, ClassNotFoundException
```

dimana where *obj* adalah object yang dibaca dari stream. tipe *Object* dikembalikan harus melalui proses typecast ke nama class yang sesuai sebelum method pada class tersebut dapat dieksekusi.

Contoh dibawah ini membaca sebuah object *Boolean* dari sebuah *ObjectInputStream*. Ini merupakan kesinambungan dari contoh sebelumnya pada serialisasi.

```
import java.io.*;

public class UnserializeBoolean {
    UnserializeBoolean() {
      Boolean booleanData = null;

      try {
         FileInputStream fis = new
                             FileInputStream("boolean.ser");
         ObjectInputStream ois = new ObjectInputStream(fis);
         booleanData = (Boolean) ois.readObject();
         ois.close();
      } catch (Exception e) {
```

```
            e.printStackTrace();
         }
         System.out.println("Unserialized Boolean from " +
                            "boolean.ser");
         System.out.println("Boolean data: " + booleanData);
         System.out.println("Compare data with true: " +
                     booleanData.equals(new Boolean("true")));
      }

      public static void main(String args[]) {
         UnserializeBoolean usb = new UnserializeBoolean();
      }
   }
```

12.13 Latihan

12.13.1 Enkripsi Sederhana

Baca dari sebuah file khusus oleh user dan encrypt isi file menggunakan teknik penggeseran yang sederhana. Juga, tanyakan pada user untuk menginput ukuran pergeseran. Output dari pesan yang telah di encrypt pada file yang lain yang memiliki nama yang juga dibuat oleh user sendiri.

Sebagai contoh,
Ukuran pergeseran: 1
Pesan yang dibaca dari file: Hello
Pesan ter-encrypt: Ifmmp