# OPERASI FILE

## Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:

* + - 1. **Memahami** konsep **operasi file** yang ada dalam java
      2. **Mengerti** konsep **operasi** dalam Java dan dapat **mengimplementasikannya** dalam sebuah program sederhana

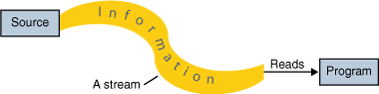
## Input Output

### Java I/O

Java io adalah **paket** untuk **menangani pembacaan** dan **penulisan informasi** untuk pemrograman java. Dalam java io terbagi dalam 2 tipe. **Byte Stream** dan **Character stream**. Sebelum lebih jauh lagi kita membahasnya akan di jelaskan terlebih dahulu konsep i/o dan stream.

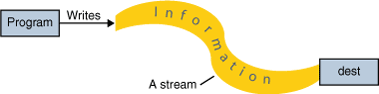
I/O adalah **input** dan **output**. Yang digunakan untuk **mengambil informasi** data dari perangkat keras untuk **metoda input**, sedangkan **output** untuk **mengirim informasi** data ke perangkat keras.

**Stream** adalah **aliran proses** **informasi** data yang **direpresentasikan** secara **abstrak** untuk sebuah **input** atau **output**. Kita dapat **menuliskan data** ke sebuah **stream** dan **membaca** data dari sebuah **stream.**



Gambar 7‑1 Alur membaca stream

Dari gambar 4.2.1.sebuah program **membuka stream** dari sebuah **sumber informasi (source)** seperti file, memory, socket. Kemudian membaca informasi secara **sekuensial.**

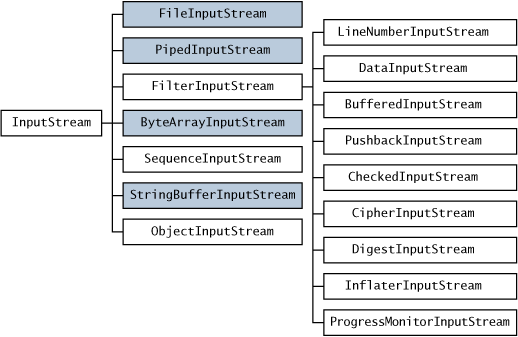


Gambar 7‑2 Alur menulis stream

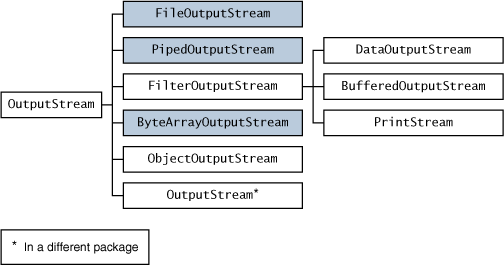
Dari gambar 4.2.2 sebuah program dapat **mengirim informasi** dengan **membuka stream** ke sebuah **tujuan informasi (destination)**. Kemudian menuliskan informasi ke tujuan secara **sekuensial.**

### Byte Stream

Byte Stream atau binari stream berisikan **binari data**. Pada java byte stream mempunyai 2 buah superclass yaitu ***InputStream*** dan ***OutputStream*** yang merupakan class **abstract.**



Gambar 7‑3 Hirarki class dari class InputStream



Gambar 7‑4 Hirarki class dari class OutputStream

Gambar 4.2.3 dan gambar 4.2.4 merupakan hirarki class dari class InputStream dan class OutputStream.

Karena keterbatasan, maka di modul ini hanya akan mengulas class **DataInputStream** dan **DataOutputStream**.

* + DataInputStream dalam jdk 1.5 hanya mempunyai **1 kostruktor dan 17 method** yang ada dalam classnya **selebihnya turunan** dari class **parent** nya. Berikut ini akan dijelaskan konstruktor dan method yang sering digunakan.
    - **DataInputStream(InputStream in)**

Untuk **membuat** sebuah **objek DataInputStream** dengan spesifikasi **InputStream** yang diinginkan.

* + - **xxx readXxx()**

**xxx** disini dapat **diganti** dengan **tipe data primitif** seperti int, float, byte, boolean, byte, char, dll. Digunakan untuk **membaca** dari **stream tipe data** tertentu secara langsung.

* + - * **DataOutputStream** dalam jdk 1.5 hanya mempunyai **1 kostruktor dan 15 method** yang ada dalam classnya **selebihnya turunan** dari class **parent** nya. Berikut ini akan dijelaskan konstruktor dan method yang sering digunakan.
    - **DataOutputStream(OutputStream out)**

Untuk membuat sebuah **objek DataOutputStream** dengan spesifikasi **OutputStream** yang diinginkan.

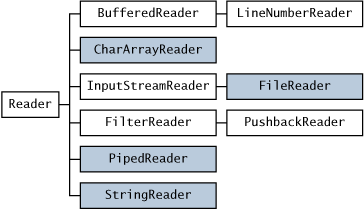
* + - **void writeXxx(xxx v)**

**xxx** disini dapat **diganti** dengan **tipe data primitif** seperti int, float, byte, boolean, byte, char, dll.

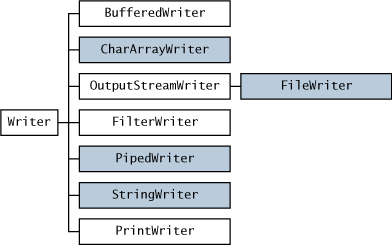
|  |
| --- |
| import java.io.\*;  public class DemoStream  {  public static void main(String[] args) {  byte[] data = new byte[10];  int panjang=0;  System.out.print("Masukkan data : ");  try {  panjang=System.in.read(data);  System.out.print("Yang anda ketik : ");  System.out.write(data);  System.out.println("Panjang Karakter : "+panjang);  }  catch (IOException e) {  System.out.print("Terjadi Exception");  }  }  } |

### Character Stream

Character stream merupakan sebuah **objek** yang dapat **membaca** dan **menuliskan byte stream** seperti **character stream**. Jadi sebenarnya character stream adalah sebuah **byte stream** yang diteruskan oleh **class *Reader*** dan***Writer*** yang merupakan class **abstract**.



Gambar 7‑5 Hirarki class dari class Reader



Gambar 7‑6 Hirarki class dari class Writer

Disini akan dibahas class **BufferedReader** dan class **BufferedWriter** saja.

* + - * **BufferedReader** dalam jdk 1.5 hanya mempunyai **2 kostruktor** dan **9 method** yang ada dalam classnya selebihnya **turunan** dari **class parent** nya. Berikut ini akan dijelaskan konstruktor dan method yang sering digunakan.
      * **BufferedReader(Reader in)** dan **BufferedReader(Reader in,int size)**. Membuat **objek BufferedReader** dengan karakter **buffer inputstream** atau **objek Reader** yang lain. **Int size** digunakan untuk **ukuran buffer** yang digunakan.
      * **String readLine()**. **readLine** digunakan untuk **membaca satu baris** penuh **text,** yaitu **mengembalikan String** dari **objek BufferedReader** yang digunakan.
      * **BufferedWriter** dalam jdk 1.5 hanya mempunyai **2 kostruktor** dan **6 method** yang ada dalam classnya selebihnya turunan dari class parent nya. Berikut ini akan dijelaskan konstruktor dan method yang sering digunakan.
      * **BufferedWriter (Writer in)** dan **BufferedWriter(Writer in,int size)**. **Membuat** objek **BufferedWriter** dengan **karakter buffer outputstream** atau **objek Writer** yang lain. **Int size** digunakan untuk **ukuran buffer** yang digunakan.
      * **void write(String s, int off, int len)**. Digunakan untuk **menuliskan** sebuah **String s** **ke** sebuah **media** yang telah **dispesifikasikan** oleh objek **BufferedWriter**. **Dimulai** dari **karakter ke off(integer)**, **ke** berapa banyak **karakter** yang akan ditulis **len(integer).**

**Pastikan** jika ingin **menginstan objek** dari paket java io, **error** harus selalu **ditangkap** oleh **Exception,** seperti IOException yang digunakan untuk menangkap kesalahan dari error IO.

* + - * Jadi jika ingin membaca dari console dengan **Character Streams,** dapat ditulis dengan cara :

InputStreamReader input = new InputStreamReader(System.in);

BufferedReader buff = new BufferedReader(input);

* + - * Atau dengan cara :

BufferedReader buff = new BufferedReader(

new InputStreamReader(System.in));

## Operasi File

File digunakan sebagai **media penyimpan**, untuk **mengakses file** kita harus **menspesifikasikan** **dimana** **file** yang akan **kita akses,** atau **membuat file baru** yang akan **disimpan**. Dalam java kita dapat **mengakses file menulis** dan **membaca** dengan **character stream** atau dengan **byte stream**, dan dapat juga kita hanya **menciptakan** sebuah **file**. Untuk **menciptakan** sebuah **file** dengan **mengakses** class **java.io.File** dan **menciptakan** **objek** dari **class** tersebut kita sudah dapat **mencipktakan file**. **Tanpa** harus **menangkap error** io. **Berbeda** dengan **menciptakan file** yang **langsung diakses** oleh **stream**. **File** tersebut **harus** dapat **menangkap error io** ketika penciptaan objek class file.

### java.io.File

terdapat **4 atribut, 4 konstruktor** dan **39 method** yang ada didalam class untuk menspesifikasikan file yang dibuat.

**File(),File(String path),File(String dir,String nm)**

Konstruktor diatas adalah yang sering digunakan yaitu **membuat objek file** kemudian digunakan dengan **pengesetan methodnya,** atau **menginstan** langsung dengan **nama file** beserta **pathnya.**

**boolean createNewFile(), boolean delete(),boolean exists()**

method-method diatas untuk **mengeset** dengan **pengecekan,** untuk **createNewFile** digunakan untuk **menciptakan file** kemudian mengembalikan **nilai true** jika **file dibuat.**

### java.io.FileOutputStream

terdapat **5 konstruktor** dan **7 method** untuk membuat **file** yang akan **diakses** menggunakan **byte stream**.

**FileOutputStream(File of), FileOutputStream(File of,boolean append).**

Digunakan untuk **penciptaan objek file** yang akan **diakses** dengan **byte stream**, dan untuk **variabel append** digunakan untuk **apakah isi** file akan **dilanjutkan ke akhir** dari isi **file.**

### java.io.FileWriter

terdapat **5 konstruktor** dan **tidak ada method** yang dideskripsikan didalam class ini.

**FileWriter(File of), FileWriter(File of,boolean append)**

Digunakan untuk **penciptaan objek file** yang akan **diakses** dengan **character stream,** dan untuk **variabel append** digunakan untuk **apakah isi file** akan dilanjutkan **ke akhir** dari isi **file.**

### java.io.FileInputStream

terdapat **3 konstruktor** dan **9 method** yang ada. Digunakan untuk **mengambil file** yang telah dideskripsikan untuk dibaca dengan **byte stream.**

**FileInputStream(File of), FileInputStream(String nama).**

Digunakan untuk **mengambil file** untuk dibaca secara **byte stream,** bisa **memasukan** **deskripsi** **file** yang telah ada dengan **String**, atau dengan **file** yang telah **diinstan** dengan jelas.

### java.io.FileReader

terdapat **3 konstruktor** dan **tidak ada method** yang dideskripsikan didalam class ini.

**FileReader(File of), FileReader(String nama).**

Digunakan untuk **mengambil file** untuk dibaca secara **character stream**,bisa **memasukan deskripsi file** yang telah ada dengan **String**, atau dengan **file** yang telah **diinstan** dengan jelas.

|  |
| --- |
| import java.io.\*;  public class CopyFile {  public static void main(String[] args) throws IOException {  System.out.print("Masukkan isi file : ");  BufferedReader br=new BufferedReader(new InputStreamReader(System.in));  String isi=br.readLine();  /\* buat objek File\*/  File FAwal = new File("Fawal.txt");  File FAkhir = new File("Fakhir.txt");  /\* Menuliskan isi ke dalam file \*/  FileOutputStream fos=new FileOutputStream(FAwal);  DataOutputStream dos=new DataOutputStream(fos);  dos.writeBytes(isi);  /\* Membaca isi file awal\*/  FileInputStream fis=new FileInputStream(FAwal);  DataInputStream dis=new DataInputStream(fis);  String isiFawal= dis.readLine();  System.out.println("isi file awal: "+isiFawal);  /\* Mengcopy isi file awal yang dibaca ke file akhir \*/  FileOutputStream fos2=new FileOutputStream(FAkhir);  DataOutputStream dos2=new DataOutputStream(fos2);  dos2.writeBytes(isiFawal);  System.out.println("isi file akhir: "+isiFawal);  }  } |

## Komunikasi antar objek

Penggunaan bahasa **pemrograman berbasis objek** mengharuskan kita merancang program untuk dapat **saling berkomunikasi antar objeknya**. **Komunikasi** ini dapat berupa **pemanggilan method** **dari objek lain** ataupun secara langsung **mengakses atribut** dari **objek lain**. Agar satu objek dengan objek lain dapat **saling berkomunikasi**, maka setiap **objek** harus **dibuat** **dari** suatu ***interface***. Dengan kata lain setiap **objek pasti memiliki** definisi ***interface*.**

Pada program yang jalan di atas suatu jaringan, **objek** dapat **berjalan** pada **mesin** yang **berbeda.** Dengan kata lain **objek** yang **digunakan** oleh **satu** **proses** di **satu mesin** dapat **mengakses objek** di **proses** pada **mesin lain.** **Pengaksesan** ini **didasarkan** pada **interface** yang telah disepakati. Dan **pemanggilan** ke **objek lain** hanya dimungkinkan dengan cara **memanggil method** yang tersedia.

Setiap **proses berisi sekumpulan objek**, beberapa diantara **objek** tersebut dapat **menerima baik pemanggilan** secara **lokal** atau secara **remote.** **Pemanggilan method** **antar objek** pada **proses** yang **berbeda,** disebut ***remote method******invocation***. Sedangkan **pemanggilan method** yang terjadi **antar objek** dalam **satu proses** yang **sama** disebut ***local method invocation***. **Objek** yang dapat **menerima remote invocation** disebut sebagai ***remote object*.**

Suatu **remote object** dapat **diakses** atau **dipanggil methodnya** dari **objek** pada **proses** yang **berbeda** melalui ***remote object references*.** **Remote object references** adalah **ID** yang **dapat digunakan** untuk **menunjuk remote objek** tertentu, baik pada **satu mesin** atau **mesin berbeda.** Pada satu **ID remote object references,** juga terdapat **informasi alamat host** dimana **remote objek** yang **ditunjuk berjalan**. Suatu **remote object** memiliki sebuah ***remote interface***yang **mendefinisikan method** mana yang dapat **dipanggil** secara **remote**. Berdasar remote interface tersebut, setiap **objek** baik **lokal** maupun **remote** hanya dapat **memanggil method** yang **terdefinisi**.

## Object Serialization

Pada pemrograman **io** dan **socket**, yang **dikirimkan** hanya **data stream** ke **file** atau ke **jaringan.** Lalu bagaimana jika yang dikirimkan suatu Objek? D**iperlukan** suatu **mekanisme** untuk **membuat** sebuah **objek** dapat **dikirimkan** seperti **mengirimkan data.** Dan ini yang lebih **dikenal** dengan **Serialisasi Objek(Object serialization).**

**Serialisasi** objek **perluasan** dari **inti** class **java io** yang **digunakan** untuk **objek.** Serialisasi objek bisa **digunakan** untuk **pengkodean** (encoding) untuk **objek** dan **membuat objek** tersebut dapat **diraih** atau **digunakan,** **melalui bit-bit stream**, kemudian dapat digunakan untuk **penyusunan kembali objek** dari **bit-bit stream** yang **dikodekan,** dan **saling melengkapi**. Serialisasi merupakan **mekanisme** yang **ringan** dan kuat untuk **komunikasi** dengan **sockets** atau **RMI**(Remote Method Invocation). Selain untuk komunikasi dengan sockets teknik ini dapat digunakan juga untuk **menyimpan keadaan** suatu **status** dari suatu objek **ke** dalam **file**, seperti yang sudah dijelaskan di pendahuluan.

Jika ingin membuat sebuah objek yang dapat diserialisasikan, maka kita harus mengimplementasikan salah satu interface **java.io.Serializable** atau **java.io.Externalizable** pada class yang ingin dibuat objek yang dapat diserialisasikan. Untuk lebih jelas tentang serialisasi objek dibawah ini akan diberikan sedikit penjelasan yang terkait dengan objek serialisasi.

### Interface java.io.Serializable

**Interface serializable** harus di**implementasika**n jika ingin **membuat objek** yang dapat **diserialisasi.** Implementasinya tergolong sederhana karena **tidak** terdapat method yang harus **didefinisikan** untuk **di override**.

Tujuan **mengimplementasikan interface serializable** adalah untuk **memberitahukan** kepada **JVM** (Java Virtual Machine), bahwa **objek** yang **menerapkan serializable** merupakan objek yang **dapat diserialisasikan.**

### Class java.io.ObjectOutputStream

Class **objectoutputstream** digunakan untuk **mengirimkan objek** menjadi **stream** yang lalu dapat **dikirimkan** ke **file** atau ke **socket (jaringan).** Class ObjectOutputStream **mempunyai 2 constructor** dan **31 method** untuk versi jdk 1.5.

Adapun method dan constructor yang sering digunakan adalah:

* **ObjectOutputStream(OutputStream out)**

Membuat **ObjectOutputStream** yang akan **menuliskan** ke **spesifik OutputStream** yang dikehendaki.

* void**writeObject(Object obj)**

menuliskan **objek** yang akan **dikirimkan** ke **ObjectOutputStream.**

### Class java.io.ObjectInputStream

Class **ObjectInputStream** adalah kelas untuk **mengambil objek** dari **stream** yang **dikirimkan** **melalui** file atau socket (jaringan). Class **ObjectInputStream** mempunyai **2 Constructors, 31 methods** untuk versi jdk 1.5.

Method dan constructor yang sering digunakan adalah

* **ObjectInputStream(InputStream in)**

Membuat ObjectInputStream yang **mengambil spesifik stream** dari **InputStream** yang dikehendaki.

* *Object* **readObject()**

**Membaca** objek yang telah didefinisikan.

Contoh penerapan Serialisasi objek:

|  |
| --- |
| import java.io.\*;  public class BarangSer implements Serializable{  private String nama;  private int jumlah;  public BarangSer(String nm,int jml) {  nama=nm;  jumlah=jml;  }  public void tampil(){  System.out.println("nama barang : "+nama);  System.out.println("jumlah barang : "+jumlah);  }  public void simpanObject(BarangSer ob){  try{  FileOutputStream fos=new FileOutputStream("dtBrg.txt");  ObjectOutputStream oos=new ObjectOutputStream(fos);  oos.writeObject(ob);  oos.flush();  }catch(IOException ioe){  System.err.println("error "+ioe);  }  }  public void bacaObject(BarangSer obb){  try{  FileInputStream fis =new FileInputStream("dtBrg.txt");  ObjectInputStream ois=new ObjectInputStream(fis);  while((obb=(BarangSer)ois.readObject())!=null)  obb.tampil();  }catch(IOException ioe){  System.exit(1);  }catch(Exception e){  System.exit(1);  }  }  public static void main(String[] args) {  BarangSer a1=new BarangSer("Baju",5);  a1.simpanObject(a1);  a1.bacaObject(a1);  }  } |

**Output:**

nama barang : Baju

jumlah barang : 5

## Modifier transient

Dalam contoh diatas objek yang **disimpan** dapat **dibaca** oleh siapapun yang **mempunyai hak akses.** Untuk mengantisipasinya, dapat dibuat **atribut tidak diserialisasikan**, yaitu dengan modifier **transient** pada **atribut** yang ingin **tidak** kita **serialisasikan.** Sebagai contoh, dapat diterapkan salah satu atribut tidak serialisasikan. Maka pada saat dijalankan hasil dari atribut tersebut akan mengeluarkan nilai defaultnya untuk yang tak mempunyai nilai.

Dalam kasus ini dapat dilakukan manipulasi, sehingga atribut **transient** dapat **diserialisasi** dengan **dienkripsikan** terlebih dahulu kemudian **didekripsikan** ketika **dibaca.** Untuk melakukan itu kita harus **mengoverriding** method **writeObject** yang ada pada kelas **ObjectOutputStream** dan **readObject** yang ada pada kelas **ObjectInputStream.**

Selain dengan cara diatas, dapat dengan mengimplementasikan interface **java.io.Externalizable**.

### Interface java.io.Externalizable

Interface **Externalizable** merupakan **subclass** dari interface **Serializable**. Akan tetapi interface Externalizable terdapat **2 method** yang harus **dioverriding** dan **didefinisikan** yaitu **writeExternal** dan **readExternal**. Berikut ini penjelasan kedua method tersebut.

* void **writeExternal(ObjectOutput out) throws IOException**

digunakan untuk **menyimpan objek** kedalam suatu **metode operasi,** seperti file/socket. Terdapat parameter **ObjectOutput** yang bernama **out,** **ObjectOutput** adalah jenis **interface** yang **dijadikan** **tipe variable** parameter. Dan berikut beberapa method yang dapat digunakan parameter tersebut.

|  |  |
| --- | --- |
| **Nama Method** | **Keterangan** |
| void writeBoolean(Boolean b) | Menuliskan nilai bertipe Boolean |
| void writeByte(int i) | Menuliskan nilai bertipe byte |
| void writeChar(int c) | Menuliskan nilai character dengan (byte)(0xff & (c >> 8))  (byte)(0xff & c) |
| void writeDouble(double d) | Menuliskan nilai bertipe double |
| void writeFloat(float f) | Menuliskan nilai bertipe float |
| void writeInt(int i) | Menuliskan nilai bertipe integer |
| void writeLong(long l) | Menuliskan nilai bertipe long |
| void writeShort(int s) | Menuliskan nilai bertipe short |
| void writeObject(Object o) | Menuliskan objek atau tipe data referensi |

Method-method ini yang sering dapat **digunakan** untuk **menuliskan tipe data**, yang ada dalam method **writeExternal** ini, meskipun banyak method lain lagi yang dapat digunakan untuk keperluan yang lain.

* void **readExternal(ObjectInput in) throws IOException, ClassNotFoundException**

Digunakan untuk **membaca objek** yang telah **dikirimkan** melalui suatu **metode operasi**, seperti file atau socket. Terdapat parameter bertipe **ObjectInput** yang bernama **in**, sama seperti ObjectOutput, **ObjectInput** merupakan **interface** yang **dijadikan tipe parameter**. Berikut beberapa method read yang dapat digunakan untuk membaca nilai dari input stream.

| **Nama Method** | **Keterangan** |
| --- | --- |
| boolean readBoolean() | Membaca nilai boolean dan meng-embalikannya |
| byte readByte() | Membaca nilai byte dan meng-embalikannya |
| char readChar() | Membaca nilai karakter dan meng-embalikannya |
| double readDouble() | Membaca nilai double dan meng-embalikannya |
| float readFloat() | Membaca nilai float dan meng-embalikannya |
| int readInt() | Membaca nilai integer dan meng-embalikannya |
| long readLong() | Membaca nilai long dan meng-embalikannya |
| Object readObject() | Membaca objek dan mengembalik-an ke tipe Object |
| short readShort() | Membaca nilai boolean dan meng-embalikannya |

Method-method ini yang sering dapat digunakan untuk membaca tipe data, yang ada dalam method readExternal, meskipun banyak method lain lagi yang dapat digunakan untuk keperluan yang lain.

Jika kita ingin membuat kelas yang mampu **meserialisasikan** **objeknya** dan **dapat mengontrol** dan **memanipulasi atribut** yang **diserialisasikan** maka kita dapat membuat kelas tersebut mengimplementasikan interface **Externalizable**, kemudian **mengoverriding** kedua method diatas. **Kedua method** diatas akan **dipanggil** secara **otomatis** ketika **operasi baca** dan **tulis** **ke** atau **dari** **input/output stream** terjadi.

Contoh penerapan Externalizable pada Objek:

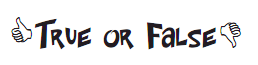
|  |
| --- |
| import java.io.\*;  public class BarangEx implements Externalizable{  private String nama;  private int jumlah;  public BarangEx() {  }  public BarangEx(String nm,int jml){  nama=nm;  jumlah=jml;  }  public void writeExternal(ObjectOutput out)throws IOException{  out.writeObject(nama);  out.writeInt(jumlah);  }  public void readExternal(ObjectInput in)throwsIOException,ClassNotFoundException{  this.nama=(String) in.readObject();  this.jumlah=in.readInt();  }  public String toString(){  return "data barang : "+nama+"\n"+"jumlah barang : "+jumlah;  }  public static void simpanObjek(BarangEx brg)throws IOException{  FileOutputStream fos=new FileOutputStream("dtEx.txt");  ObjectOutputStream oos=new ObjectOutputStream(fos);  oos.writeObject(brg);  oos.flush();  }  public static BarangEx bacaObjek()throwsClassNotFoundException,IOException{  FileInputStream fis=new FileInputStream("dtEx.txt");  ObjectInputStream ois=new ObjectInputStream(fis);  return (BarangEx)ois.readObject();  }  public static void main(String[] args)throws ClassNotFoundException,IOException{  BarangEx awal=new BarangEx("sepatu",2);  simpanObjek(awal);  System.out.println(bacaObjek());  }  } |

**Output :**

data barang : sepatu

jumlah barang : 2

Ini **contoh** dari penerapan **Externalizable** kalian dapat mempelajarinya sendiri perbedaannya dimana. Kami memberikan 2 contoh soal yang berbeda konsep agar menambah pengetahuan praktikan mengenai teknik pemrograman berorientasi objek. Kelebihan dan kekurangan mohon dimaafkan. Jika ada ketidak mengertian praktikan tanyakan ke asisten terdekat atau ke penulis.



|  |  |
| --- | --- |
| 1. Object hanya dapat disimpan menggunakan serialization saja 2. Ketika object di deserialized, dibaca dengan Last In First Out 3. File Objects merepresentasikan files, bukan direktori. 4. Transient Modifier memperbolehkan untuk instansiasi variable serializable 5. ObjectOutputStream adalah class yang digunakan untuk menyimpan serialization objek. | **TRUE | FALSE**  **TRUE | FALSE**  **TRUE | FALSE**  **TRUE | FALSE**  **TRUE | FALSE** |

|  |
| --- |
| *"Kita bukannya berhenti bermain karena kita makin tua; tetapi kita makin tua karena kita berhenti bermain." -* George Bernard Shaw |