Operasi File

File digunakan sebagai media penyimpan. Untuk mengakses file kita harus menspesifikasikan dimana file yang akan kita akses, atau file baru yang akan disimpan. Dalam java kita dapat melakukan operasi file, yaitu membuat file baru atau menulis dan membaca file dengan character stream atau dengan byte stream.

Untuk menciptakan sebuah file dengan mengakses class java.io.File dan menciptakan objek dari class tersebut, ini 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.Filewriter → dengan character stream

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 → dengan byte stream

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.Fileoutputstream → dengan byte stream

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.Filereader → dengan character stream

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.

Kira-kira seperti apa yah implementasinya di Kode? Any idea?

input file dengan byte stream

try this at home

```
import java.io.*;
public class DemoStream5 {
   public static void main(String[] args) {
      byte data;
      String namaFile = "test.txt";
      FileOutputStream fout = null;
      try   {
        fout = new FileOutputStream(namaFile, true);
       //true artinya menambahkan kedalam file, tidak menimpa
        System.out.print("Ketik : ");
        data = (byte)System.in.read();
        while (data!=(byte)'\r') {
           fout.write(data);
           data = (byte)System.in.read();
        }
      }
      catch (FileNotFoundException e) {
        System.out.println("File "+namaFile+" tidak dapat  dicreate");
      }
      catch (IOException e) {
        System.out.println("Terjadi Exception");
      }
      finally {
         if (fout!=null) {
            try {
                  fout.close();
            }
            catch (IOException e) {
                  System.out.println("Terjadi  Exception");
            }
         }
      }
   }
}
```

enter

Outputnya:

```
Ketik : commonlabz
```

input file dengan byte stream

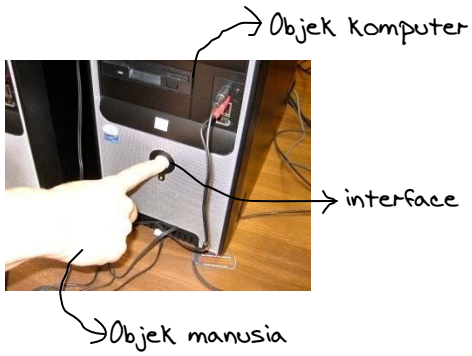
try this at home

```
import java.io.*;
public class DemoStream4 {
  public static void main(String[] args) {
   byte data;
   String namaFile = "test.txt";
   FileInputStream fin = null;
   try      {
      fin = new FileInputStream(namaFile);
      do {
            data = (byte)fin.read();
            System.out.print((char)data);
      }
      while (data!=-1);
            //pada saat eol/eof nilai yang dikembalikan -1
   }
   catch (FileNotFoundException e) {
      System.out.println("File "+namaFile+" tidak ditemukan");
   }
   catch (IOException e) {
      System.out.println("Terjadi Exception");
   }
   finally {
      if (fin!=null) {
            try {
                  fin.close();
            }
            catch (IOException e) {
                  System.out.println("Terjadi Exception");
            }
      }
   }
  }
}
```

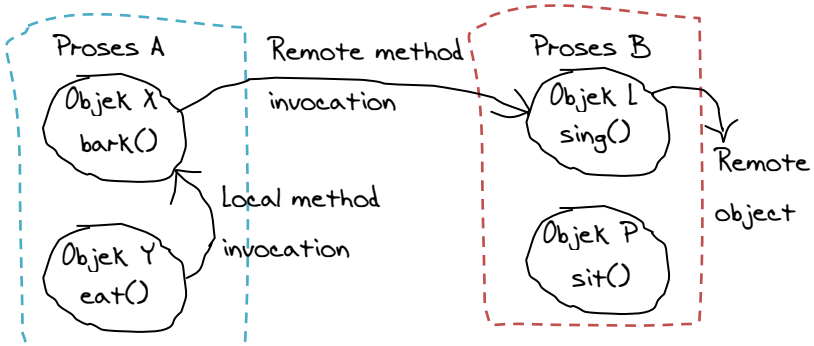
Outputnya:

```
commonlabz?
```

Komunikasi antar objek



Agar satu objek bisa berkomunikasi dengan objek lain maka setiap objek pasti memiliki definisi interface.



Remote object dipanggil oleh objek lain melalui remote object references yang merupakan sebuah ID yang dapat digunakan untuk menunjuk pada suatu remote object tertentu, di dalam ID ada info alamat host di mana remote object yang ditunjuk lagi jalan. Method mana aja yang bisa dipanggil secara remote didefinisiin di remote interface

Object Serialization

Apa itu object serialization?

Suatu mekanisme yang dapat membuat sebuah objek dapat dikirimkan seperti mengirimkan data.

Object serialization merupakan perluasan dari inti class java io yang digunakan untuk objek dan bisa digunakan untuk pengkodean (encoding) untuk objek dan membuat objek tersebut dapat diraih atau digunakan, dengan melalui bit-bit stream, kemudian dapat digunakan untuk penyusunan kembali objek tersebut 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 kita 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 implementasikan jika ingin membuat objek yang dapat diserialisasi, implementasi interface serializable 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 serializabel merupakan objek yang dapat diserialisasikan.

Class java.io.ObjectOutputStream
Class objectoutputstream adalah kelas yang digunakan untuk mengirimkan objek menjadi stream yang kemudian 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 yang digunakan 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 akan mengambil spesifik stream dari InputStream yang dikehendaki.

Object readObject()
Membaca objek yang telah diddefinisikan.

try this at home

```
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 program:

```
nama barang : Baju

jumlah barang : 5
```

Modifier Transient

Buat apa modifier transient ?

Pada contoh di atas, objek yang kita simpan itu bisa dibaca oleh siapapun yang punya hak akses. Nah buat antisipasi hal itu, kita bisa buat atribut tidak diserialisasikan. Makanya kita pake modifier transient pada atribut yang ngga pengen kita serialisasikan.

Misalnya, dari contoh diatas, salah satu atributnya nggak kita serialisasikan maka pada saat dijalankan hasil dari atribut tersebut mengeluarkan nilai defaultnya untuk yang tak mempunyai nilai. Dalam kasus ini kita dapat melakukan manipulasi, sehingga atribut transient dapat diserialisasikan 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 ada cara lain yang mungkin lebih mudah untuk dipahami yaitu 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 ObjecOutput 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 ObjecInput 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.

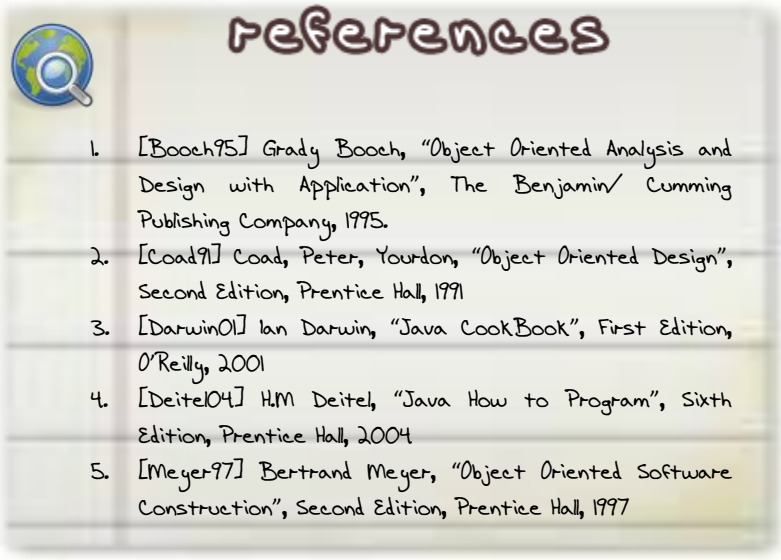
Nama Method	Keterangan
boolean readBoolean()	Membaca nilai boolean dan meng-embalikannya
byte readByte()	Membaca nilai byte dan meng-embalikannya
char readChar()	Membaca nilai karakter dan meng-embalikannya
double readDouble()	Membaca nilai double dan meng-embalikannya
float readFloat()	Membaca nilai 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 menserialisasikan 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.

Jika output programnya seperti di bawah ini, bagaimana bentuk kode programnya? Mari kita lihat kodenya di halaman sebelah

```
data barang : sepatu
jumlah barang : 2
```



try this at home

```
import java.io.*;

public class BarangEx implements Externalizable{
      private String nama;
      private int jumlah;

      public BarangEx(){                                    //konstruktor 1
      }

      public BarangEx(String nm, int jml){                  //konstruktor2
            nama=nm;
            jumlah=jml;
      }

      public void writeExternal(ObjectOutput out) throws IOException{
            out.writeObject(nama); //string adalah tipe data referensi
            out.writeInt(jumlah);
      }

      public void readExternal(ObjectInput in) throws IOException,
      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() throws
      ClassNotFoundException,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());
      }
}
```