



CSE 2202 **Object Oriented Programming** **(OOP)**

Java File and I/O

2018



Computer Science & Engineering Program
The School of EE & Computing
Adama Science & Technology University

CSE Program

Objectives

After studying this chapter, students should be able to learn:

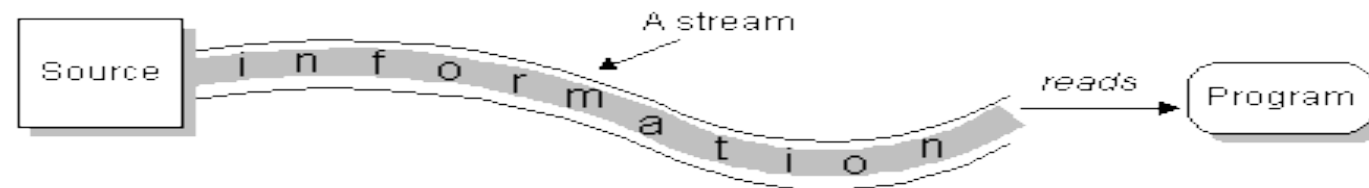
- Streams
 - Byte Stream
 - Text Stream
- Files
 - Creating a file
 - Writing to a file
 - Reading from a file
- Object Serialization
- File Class

Input/output Basics

- Different sources and sinks of I/O that you want to communicate with are files, the console, network connections, etc., but you need to talk to them in a wide variety of ways (sequential, random-access, buffered, binary, character, by lines, by words, etc.).
- Input/Output(I/O) communication between a computer program and external sources and destinations of information.
- Involves *Reading* and *Writing*
 - Reading input from a source
 - Writing output to a destination
- Example Sources and Destinations:
 - Files
 - Network connections
 - Other programs

- Java uses an I/O system called *streams* which is a *flow (sequence) of data*.
- Java provides `java.io` package to implement *streams*
- *Streams* treat all external source and destinations of data the same way: as "streams" of information.

- **Reading from an Input Stream**



- **Writing to an Output Stream**



- Streams are of two type:
 - ***Input Stream (InputStream):*** is an abstract super class for all classes representing an input stream of bytes
 - Input can be from keyboard, File and memory.
 - ***Output Stream (OutputStream):*** is also an abstract super class for all classes representing an output stream of bytes.
 - output can be from monitor, memory or file.
 - For each input/output devices there is a corresponding class which belongs to the InputStream/OutputStream abstract super class.
- Inputs and outputs can be given into two formats:
 - Byte format
 - Character Format

Byte vs. Character Streams

- **Byte Streams** are used to read and write data in binary format (1's and 0's)

Example data: *images, sounds, executable programs, word-processing documents, etc.*

- **Character Streams** are used to read and write data in text format (characters)

Example data: *plain text files (txt extension), web pages, user keyboard input, etc.*

- Package `java.io` offers classes to connect to streams
- To connect to a stream, instantiate a subclass of one of these abstract superclasses:

	input	output
byte	<code>InputStream</code>	<code>OutputStream</code>
character	<code>Reader</code>	<code>Writer</code>

Byte Streams

- Handle data in the form of bits and bytes.
- Byte streams are used to handle any characters (text), images, audio and video files.
- For example, to store an image file (.gif or.jpg), we should go for a byte stream.
- To handle data in the form of 'bytes' the abstract classes: InputStream and OutputStream are used.
- The important classes of byte streams are:
 - **FileInputStream/FileOutputStream:** They handle data to be read or written to disk files.
 - **FilterInputStream/FilterOutputStream:** They read data from one stream and write it to another stream.
 - **ObjectInputStream/ObjectOutputStream:** They handle storage of objects and primitive data.

Java BufferedOutputStream Class

- Java BufferedOutputStream class is used for buffering an output stream. It internally uses buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.
- For adding the buffer in an OutputStream, use the BufferedOutputStream class. Let's see the syntax for adding the buffer in an OutputStream:

```
OutputStream os= new BufferedOutputStream(new FileOutputStream("D:\\IO Package\\testout.txt"));
```

- Java BufferedInputStream class is used to read information from stream. It internally uses buffer mechanism to make the performance fast.

Example : **BufferedOutputStream.**

```
import java.io.*;
class Create2
{
    public static void main(String args[]) throws IOException
    {
        //attach keyboard to DataInputStream
        DataInputStream dis = new DataInputStream (System.in);
        //attach file to FileOutputStream, if we use true then it will open in append mode
        FileOutputStream fout = new FileOutputStream ("D:myfile", true);
        BufferedOutputStream bout = new BufferedOutputStream (fout, 1024);
        //Buffer size is declared as 1024 otherwise default buffer size of 512 bytes is
        used.
        //read data from DataInputStream and write into FileOutputStream
        char ch;
        System.out.println ("Enter # at end : " );
        while ( (ch = (char) dis.read() ) != '#' )
            bout.write (ch);
        bout.close ();
        fout.close ();
    }
}
```

Example: **BufferedInputStream**

```
import java.io.*;
class Read2
{   public static void main(String args[]) throws IOException
    { //attach the file to FileInputStream
        FileInputStream fin = new FileInputStream ("D:myfile");
        BufferedInputStream bin = new BufferedInputStream (fin);
        //read data from FileInputStream and display it on the
monitor
        int ch;
        while ( (ch = bin.read() ) != -1 )
            System.out.print ( (char) ch);
        fin.close ();
    }
}
```

Java SequenceInputStream Class

- Java SequenceInputStream class is used to read data from multiple streams. It reads data sequentially (one by one).

SequenceInputStream

```
import java.io.*;
class InputStreamExample {
    public static void main(String args[])throws Exception{
        FileInputStream input1=new FileInputStream("D:testin.txt");
        FileInputStream input2=new FileInputStream("D:testout.txt");
        SequenceInputStream inst=new SequenceInputStream(input1, input2);
        int j;
        while((j=inst.read())!=-1){
            System.out.print((char)j);
        }
        inst.close();
        input1.close();
        input2.close();
    }
}
```

Java ByteArrayOutputStream

- Java ByteArrayOutputStream class is used to **write common data** into multiple files. In this stream, the data is written into a byte array which can be written to multiple streams later.
- The ByteArrayOutputStream holds a copy of data and forwards it to multiple streams.
- The ByteArrayInputStream is composed of two words: ByteArray and InputStream. As the name suggests, it can be used to read byte array as input stream.
- Java ByteArrayInputStream class contains an internal buffer which is used to **read byte array** as stream. In this stream, the data is read from a byte array.
- The buffer of ByteArrayOutputStream automatically grows according to data.

Example of Java ByteArrayOutputStream

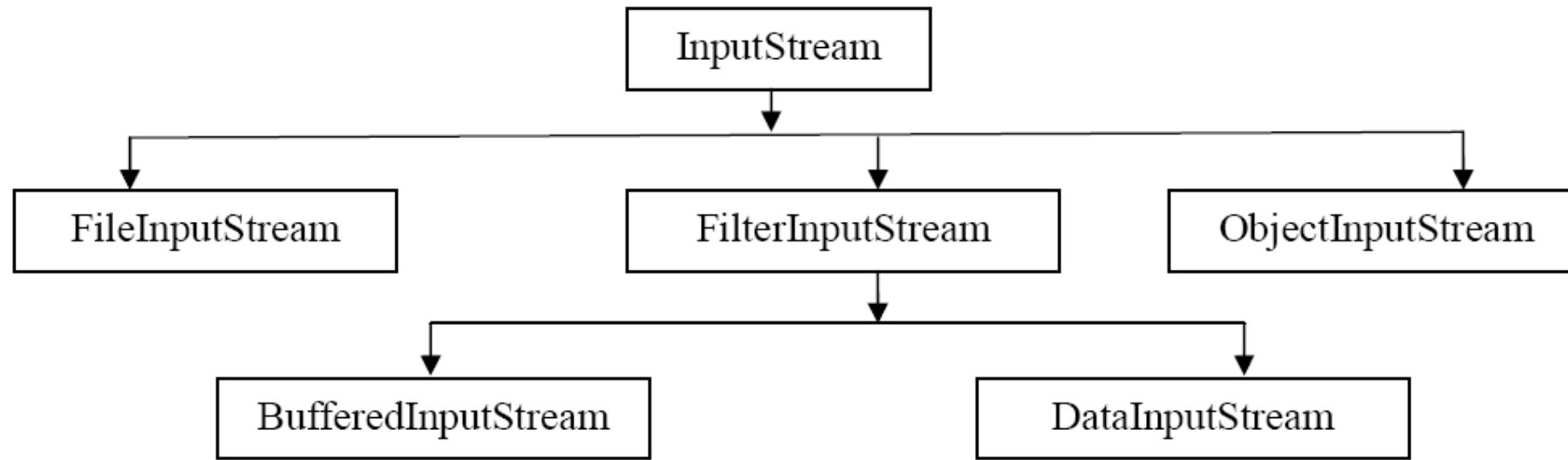
```
import java.io.*;
public class DataStreamExample {
    public static void main(String args[])throws Exception{
        FileOutputStream fout1=new FileOutputStream("D:\\
f1.txt");
        FileOutputStream fout2=new FileOutputStream("D:\\
f2.txt");

        ByteArrayOutputStream bout=new ByteArrayOutputStream();
        bout.write(65);
        bout.writeTo(fout1);
        bout.writeTo(fout2);

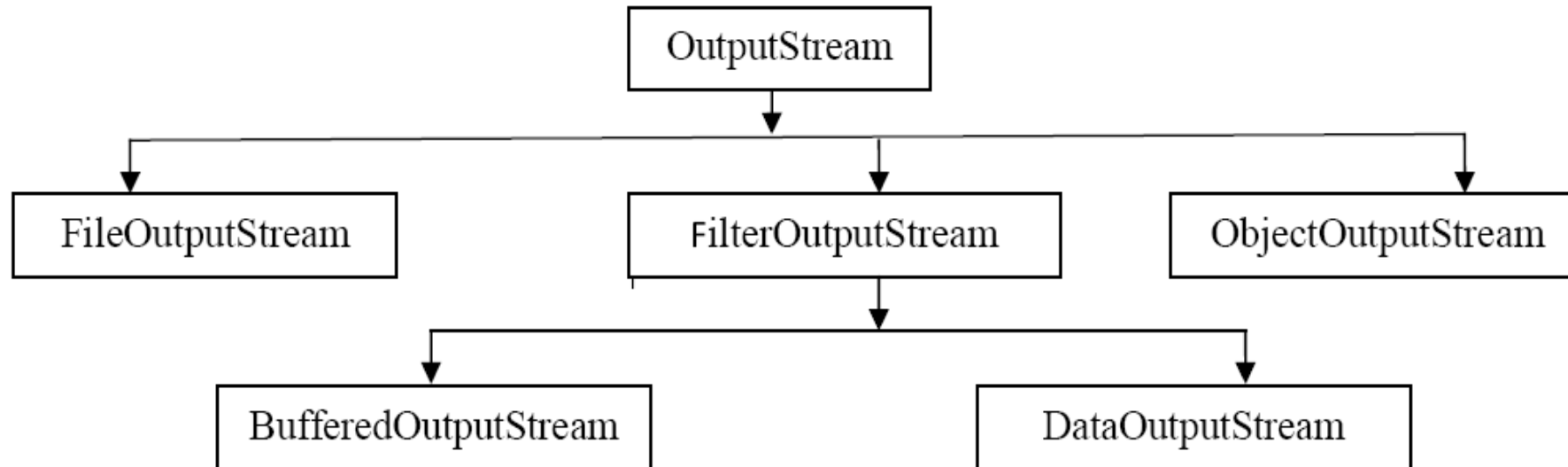
        bout.flush();
        bout.close();//has no effect
        System.out.println("Success...");
    }
}
```

ByteArrayInputStream example

```
import java.io.*;
public class ReadExample {
    public static void main(String[] args) throws IOException {
        byte[] buf = { 35, 36, 37, 38 };
        // Create the new byte array input stream
        ByteArrayInputStream byt = new ByteArrayInputStream(buf);
        int k = 0;
        while ((k = byt.read()) != -1) {
            //Conversion of a byte into character
            char ch = (char) k;
            System.out.println("ASCII value of Character is:" + k + "; Special character
is: " + ch);
        }
    }
}
```

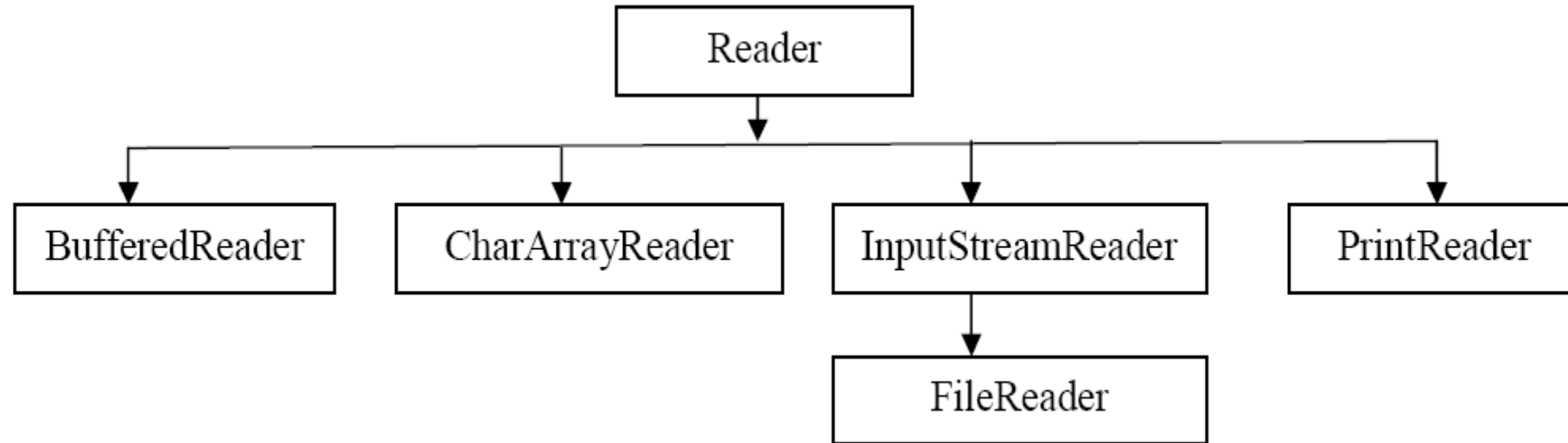


Byte Stream Classes for Reading Data

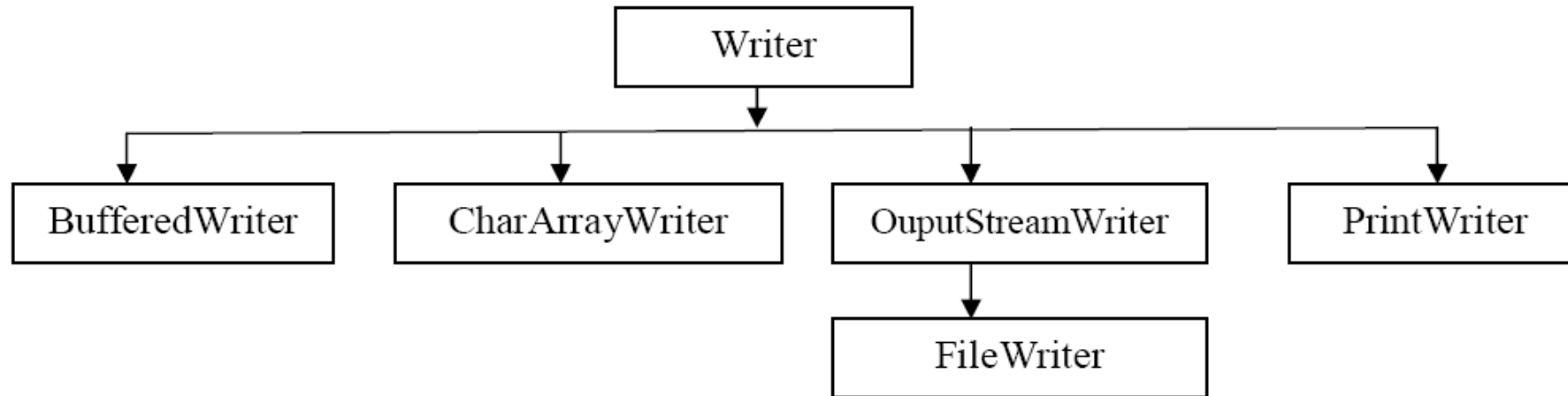


Byte Stream Classes for Writing Data

- Handle data in the form of characters.
- Character or text streams can always store and retrieve data in the form of characters (or text) only.
 - It means text streams are more suitable for handling text files like the ones we create in Notepad.
 - They are not suitable to handle the images, audio or video files.
- To handle data in the form of 'text', the abstract classes: **Reader** and **Writer** are used.
- The important classes of character streams are:
 - **BufferedReader/BufferedWriter:** - Handles characters (text) by buffering them. They provide efficiency.
 - **CharArrayReader/CharArrayWriter:** - Handles array of characters.
 - **PrintReader/PrintWriter:** - Handle printing of characters on the screen.



Text Stream Classes for Reading Data



Text Stream Classes for Writing Data

Let's see the simple example of writing the data to a text file **testout.txt** using Java BufferedWriter.

```
import java.io.*;
public class BufferedWriterExample {
public static void main(String[] args) throws Exception {
    FileWriter writer = new FileWriter("D:/testout.txt");
    BufferedWriter buffer = new BufferedWriter(writer);
    buffer.write("Welcome to java IO Class.");
    buffer.close();
    System.out.println("Success");
}
}
```

Java BufferedReader Example

In this example, we are reading the data from the text file **testout.txt** using Java BufferedReader class.

```
import java.io.*;
public class BufferedReaderExample {
    public static void main(String args[])throws Exception{
        FileReader fr=new FileReader("D:\\testout.txt");
        BufferedReader br=new BufferedReader(fr);
        int i;
        while((i=br.read())!=-1){
            System.out.print((char)i);
        }
        br.close();
        fr.close();
    }
}
```

Java BufferedReader Example

In this example, we are reading the data from the text file **testout.txt** using Java BufferedReader class.

```
import java.io.*;
public class BufferedReaderExample {
    public static void main(String args[])throws Exception{
        FileReader fr=new FileReader("D:\\testout.txt");
        BufferedReader br=new BufferedReader(fr);
        int i;
        while((i=br.read())!=-1){
            System.out.print((char)i);
        }
        br.close();
        fr.close();
    }
}
```

- The CharArrayReader is composed of two words: CharArray and Reader. The CharArrayReader class is used to read character array as a reader (stream). It inherits Reader class.

```
import java.io.CharArrayReader;
public class CharArrayExample{
    public static void main(String[] ag) throws Exception {
        char[] ary = { 'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't' };
        CharArrayReader reader = new CharArrayReader(ary);
        int k = 0;
        // Read until the end of a file
        while ((k = reader.read()) != -1) {
            char ch = (char) k;
            System.out.print(ch + " : ");
            System.out.println(k);
        }
    }
}
```

CharArrayWriterExample

```
import java.io.CharArrayWriter;
import java.io.FileWriter;
public class CharArrayWriterExample {
public static void main(String args[])throws Exception{
    CharArrayWriter out=new CharArrayWriter();
    out.write("Welcome to javaTpoint");
    FileWriter f1=new FileWriter("D:|a.txt");
    FileWriter f2=new FileWriter("D:|b.txt");
    FileWriter f3=new FileWriter("D:|c.txt");
    FileWriter f4=new FileWriter("D:|d.txt");
    out.writeTo(f1);
    out.writeTo(f2);
    out.writeTo(f3);
    out.writeTo(f4);
    f1.close(); f2.close();
    f3.close(); f4.close();
    System.out.println("Success...");
}
}
```

PrintWriter Example

```
import java.io.File;
import java.io.PrintWriter;
public class PrintWriterExample {
    public static void main(String[] args) throws Exception {
        //Data to write on Console using PrintWriter
        PrintWriter writer = new PrintWriter(System.out);
        writer.write("Javatpoint provides tutorials of all technology.");
        writer.flush();
        writer.close();
        //Data to write in File using PrintWriter
        PrintWriter writer1 = null;
        writer1 = new PrintWriter(new File("D:/testout.txt"));
        writer1.write("Like Java, Spring, Hibernate, Android, PHP etc.");

        writer1.flush();
        writer1.close();
    }
}
```

- A file represents organized collection of data.
- Data is stored permanently in the file.
- Once data is stored in the form of a file we can use it in different programs.

- Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc.
- If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class .
- You can also read/write character-stream data. But, for reading / writing streams of characters, it is recommended to use FileReader class.

Object Serialization

- Serialization is the process of storing object contents into a file.
- The class whose objects are stored in the file should implement 'Serializable' interface of java.io package.
 - Serializable interface is an empty interface without any members and methods, such an interface is called 'marking interface' or 'tagging interface'.
 - Marking interface is useful to mark the objects of a class for a special purpose.
 - For example, 'Serializable' interface marks the class objects as 'serializable' so that they can be written into a file.
 - If serializable interface is not implemented by the class, then writing that class objects into a file will lead to `NotSerializableException`.
- static and transient variables cannot be serialized.
- De-serialization is the process of reading back the objects from a file.

Object Serialization

- Serialization is the process of storing object contents into a file.
- The class whose objects are stored in the file should implement 'Serializable' interface of java.io package.
 - Serializable interface is an empty interface without any members and methods, such an interface is called 'marking interface' or 'tagging interface'.
 - Marking interface is useful to mark the objects of a class for a special purpose.
 - For example, 'Serializable' interface marks the class objects as 'serializable' so that they can be written into a file.
 - If serializable interface is not implemented by the class, then writing that class objects into a file will lead to `NotSerializableException`.
- static and transient variables cannot be serialized.
- De-serialization is the process of reading back the objects from a file.

```
import java.io.*;
import java.util.*;
class Employ implements Serializable
{   private int id;
    private String name;
    private float sal;
    private Date doj;
    Employ (int i, String n, float s, Date d)
    {   id = i;
        name = n;
        sal = s;
        doj = d;
    }
    void display ()
    {
        System.out.println (id+ "\t" + name + "\t" + sal + "\t" + doj);
    }
}
```

```
static Employ getData() throws IOException
{
    BufferedReader br = new BufferedReader (new InputStreamReader
(System.in));
    System.out.print ("Enter employ id : ");
    int id = Integer.parseInt(br.readLine());
    System.out.print ("Enter employ name : ");
    String name = br.readLine ();
    System.out.print ("Enter employ salary : " );
    float sal = Float.parseFloat(br.readLine ());
    Date d = new Date ();
    Employ e = new Employ (id, name, sal, d);
    return e;
}
```

//ObjectOutputStream is used to store objects to a file

```
import java.io.*;
import java.util.*;
class StoreObj
{ public static void main (String args[]) throws IOException
  { BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
    FileOutputStream fos = new FileOutputStream ("objfile");
    ObjectOutputStream oos = new ObjectOutputStream ( fos );
    System.out.print ("Enter how many objects : ");
    int n = Integer.parseInt(br.readLine () );
    for(int i = 0;i<n;i++)
    { Employ e1 = Employ.getData ();
      oos.writeObject (e1);
    }
    oos.close ();
    fos.close ();
  }
}
```

```
//ObjectInputStream is used to read objects from a file
import java.io.*;
class ObjRead
{ public static void main(String args[]) throws Exception
  {   FileInputStream fis = new FileInputStream ("objfile");
      ObjectInputStream ois = new ObjectInputStream (fis);
      try
      {   Employ e;
          while ( (e = (Employ) ois.readObject() ) != null)
              e.display ();
      }
      catch(EOFException ee)
      {
          System.out.println ("End of file Reached...");
      }
      finally
      {   ois.close ();
          fis.close ();
      }
  }
}
```

Thank you staying with me!!!

Q?

End of course !!!

Good time with CSE Dept. !!

If you have any comment/suggestion?

You well come