CST256: Object Oriented Programming

UNIT - III

Kanak Kalyani

Streams in JAVA

File

It deals directly with files and the file system. That is, the **File** class does not specify how information is retrieved from or stored in files; it describes the properties of a file itself. A **File** object is used to obtain or manipulate the information associated with a disk file, such as the permissions, time, date, and directory path, and to navigate subdirectory hierarchies.

A directory in Java is treated simply as a **File** with one additional property—a list of filenames that can be examined by the **list( )** method.

The following constructors can be used to create **File** objects:

File(String *directoryPath*)

Example: File f1 = new File("/");

File(String *directoryPath*, String *filename*)

Example: File f2 = new File("/","autoexec.bat");

File(File *dirObj*, String *filename*)

Example: File f3 = new File(f1,"autoexec.bat");

Example:

```
import java.io.File;
import java.io.IOException;
import java.util.Date;
class FileDemo {
 public static void main(String args[]) throws IOException {
File f1 = new File("d:/abc");
        System.out.println ("File Name: " + f1.getName());
        System.out.println ("Abs Path: " + f1.getAbsolutePath());
        System.out.println ("Parent: " + f1.getParent());
        System.out.println (f1.exists() ? "exists" : "does not exist");
       System.out.println (f1.canWrite() ? "is writeable" : "is not writeable");
       System.out.println (f1.canRead() ? "is readable" : "is not readable");
       System.out.println ("is " + (f1.isDirectory() ? "" : "not" )+ " a
      directory");
       System.out.println (f1.isFile() ? "is normal file" : "not a file");
       System.out.println (f1.isAbsolute() ? "is absolute" : "is not absolute");
       System.out.println ("File last modified: "     +  new
      Date(f1.lastModified()).toString());
       System.out.println ("File size: " + f1.length() + " Bytes");
```

Output

```
File Name: abc
Abs Path: d:\abc
Parent: d:\
exists
is writeable
is readable
is  a directory


not a file
is absolute
File last modified: Mon Feb 23
11:52:04 IST 2015
File size: 0 Bytes
```

```
File f;
 f=new File("myfile.txt");
 if(!f.exists()){
   f.createNewFile();
   System.out.println("New file \"myfile.txt\" has been created
to the current directory");
      System.out.println("abs path"+ f.getAbsolutePath());
  }   //deleting the file
 System.out.println ("file deleted: " + f.delete());
 if (f1.isDirectory()) {
   System.out.println("Directory of " + f1.getParent());
String s[] = f1.list();
for (int i=0; i < s.length; i++)
{
            f = new File(f1.getAbsolutePath()+ "/" + s[i]);
            if (f.isDirectory()) {
                        System.out.println(s[i] + " is a directory");
            }
            else{
                        System.out.println(s[i] + " is a file");
            }
}}}}
```

Output

New file "myfile.txt" has been created to the current directory

abs path

C:\Users\user\Documents\NetBeansProjects\2013-14\myfile.txt

file deleted: true

Directory of d:

demo.txt is a file

fol is a directory

demo.txt is a file

fol is a directory

```
public class filerename {
    public static void main(String[] args) {
    File f = null;
    File f1 = null;
    boolean bool = false;

      f1=new File("d:\\demo");
      if (f1.isDirectory())
      {
       System.out.println("Directory of " + f1.getParent());
       File s[] = f1.listFiles();
       for (int i=0; i < s.length; i++)
       {
          String filename="d:\\demo\\a" + i;
          f=new File(filename);
          bool =s[i].renameTo(f);
          System.out.print("File renamed? "+bool);
       }
      }}}
```

Do it Yourself

- Write a program to list all the files and directories of D:\. If the file in consideration is a directory then display all the files and directories also.

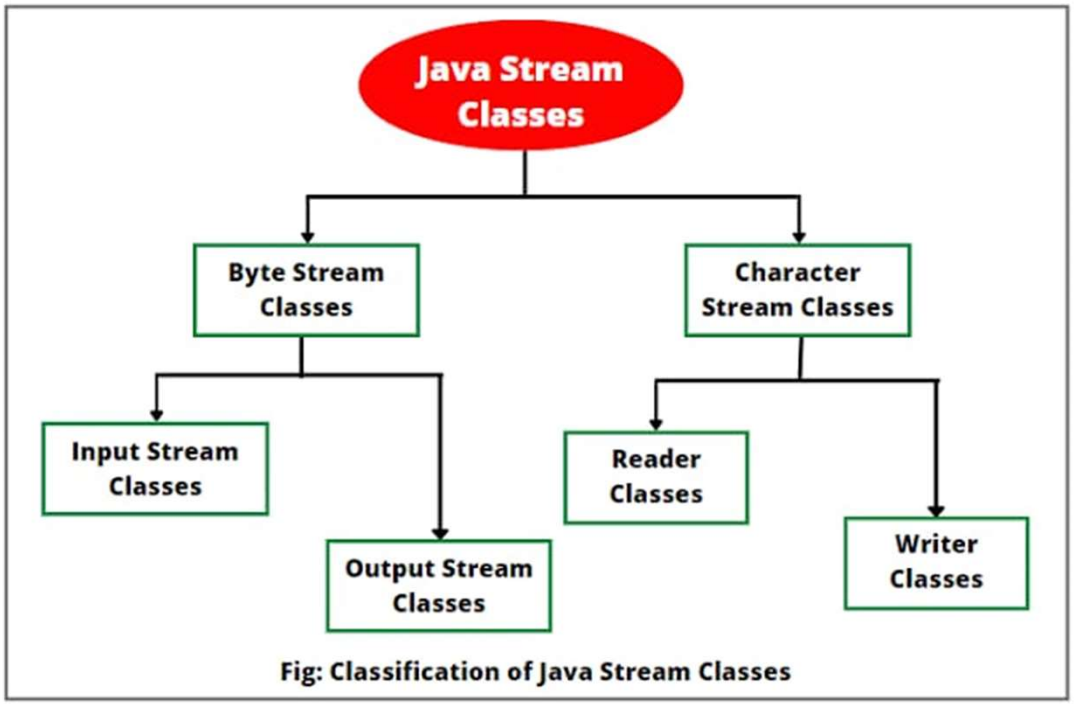
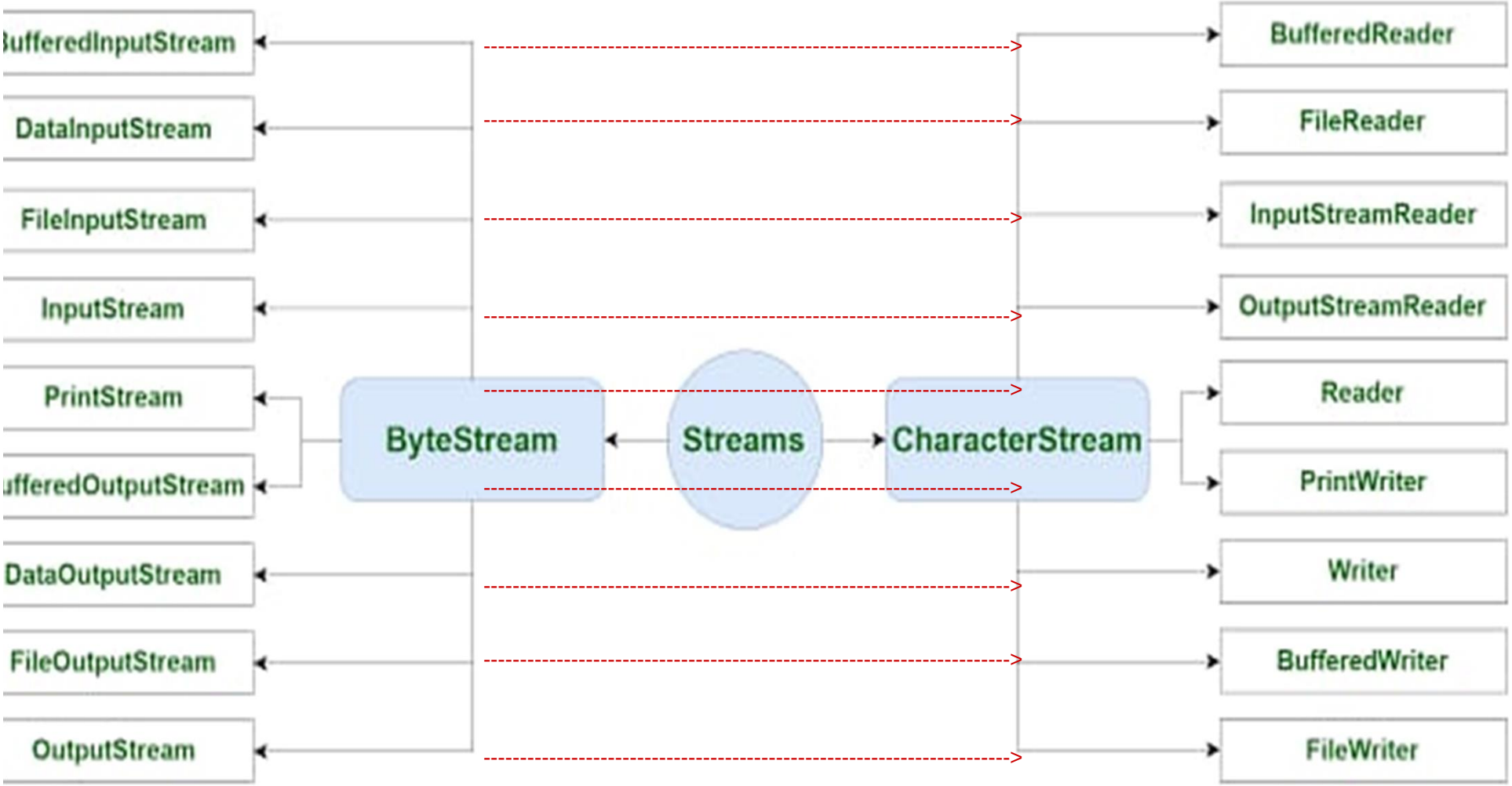


Fig: Classification of Java Stream Classes

Ending with stream is byte type :
Ending with reader is of character type.



Stream Classifications based on file types

Methods in INPUTSTREAM

Method	Description
int available( )	Returns the number of bytes of input currently available for reading.
void close( )	Closes the input source. Further read attempts will generate an **IOException**.
void mark(int *numBytes*)	Places a mark at the current point in the input stream that will remain valid until *numBytes* bytes are read.
boolean markSupported( )	Returns **true** if **mark( )**/**reset( )** are supported by the invoking stream.
int read( )	Returns an integer representation of the next available byte of input. –1 is returned when the end of the file is encountered.
int read(byte *buffer*[ ])	Attempts to read up to *buffer.length* bytes into *buffer* and returns the actual number of bytes that were successfully read. –1 is returned when the end of the file is encountered.
int read(byte *buffer*[ ], int *offset*, int *numBytes*)	Attempts to read up to *numBytes* bytes into *buffer* starting at *buffer*[*offset*], returning the number of bytes successfully read. –1 is returned when the end of the file is encountered.
void reset( )	Resets the input pointer to the previously set mark.
long skip(long *numBytes*)	Ignores (that is, skips) *numBytes* bytes of input, returning the number of bytes actually ignored.

TABLE 19-1 The Methods Defined by **InputStream**

Methods in OUTPUTSTREAM

Method	Description
void close( )	Closes the output stream. Further write attempts will generate an **IOException**.
void flush( )	Finalizes the output state so that any buffers are cleared. That is, it flushes the output buffers.
void write(int *b*)	Writes a single byte to an output stream. Note that the parameter is an **int**, which allows you to call **write( )** with expressions without having to cast them back to **byte**.
void write(byte *buffer*[ ])	Writes a complete array of bytes to an output stream.
void write(byte *buffer*[ ], int *offset*, int *numBytes*)	Writes a subrange of *numBytes* bytes from the array *buffer*, beginning at *buffer*[*offset*].

TABLE 19-2 The Methods Defined by **OutputStream**

FileInputStream

The FileInputStream class creates an InputStream that you can use to read bytes from a file.

Its two most common constructors are shown here:

FileInputStream(String filepath)

FileInputStream(File fileObj)

FileOutputStream

FileOutputStream creates an OutputStream that you can use to write bytes to a file.

Its most commonly used constructors are shown here:

FileOutputStream(String filePath)

FileOutputStream(File fileObj)

FileOutputStream(String filePath, boolean append)

FileOutputStream(File fileObj, boolean append)

Copy Content of one file to another using File Input and Output Stream.

```
public class Files {
   public static void main(String args[])
{
   try
            {
               File f=new File("d:\\abc\\demo1.txt");
               FileInputStream is=new    FileInputStream("d:\\abc\\demo.txt");
               FileOutputStream os=new FileOutputStream(f);
               System.out.println("available bytes: " +   is.available());
               int s=is.available();
    for(int i=0;i<s;i++)
    {
        int b=is.read();
        System.out.print((char)b);
        os.write(b);

    }
    os.close();
    is.close();
}catch(FileNotFoundException e)
{ System.out.println(" File Demo.txt not found ");}
catch(IOException e)
{ e.printStackTrace();}
}
}
```

Do it Yourself

- Copy the contents of file d1.txt into d2.txt but alternative bytes.

- Copy the contents of file d1.txt into d2.txt, the contents should be in Upper case

Copy the contents of file d1.txt into d2.txt, toggle the case in d2.txt

```
public class Files {
   public static void main(String args[]) throws FileNotFoundException, IOException
   {
      File f=new File("d:\\abc\\demo1.txt");
      BufferedInputStream bis=new BufferedInputStream(new
FileInputStream("d:\\abc\\demo.txt"));
      BufferedOutputStream bos=new BufferedOutputStream(new FileOutputStream(f));
      System.out.println("available bytes: " + bis.available());
      int s=bis.available();
      for(int i=0;i<s;i++)
      {
         int b;
         b=bis.read();
         if(Character.isUpperCase((char)b))
         {
            char c=Character.toLowerCase((char)b);
            System.out.print(c);
            bos.write(c);
         }
         else
         {
            char c=Character.toUpperCase((char)b);
            System.out.print(c);
            bos.write(c);
         }
      }
      bos.close();
}
}
```

Character Based Stream

The Methods Defined by **Reader**

int read( )	Returns an integer representation of the next available character from the invoking input stream. –1 is returned when the end of the file is encountered.
int read(char *buffer*[ ])	Attempts to read up to *buffer.length* characters into *buffer* and returns the actual number of characters that were successfully read. –1 is returned when the end of the file is encountered.
abstract int read(char *buffer*[ ], int *offset*, int *numChars*)	Attempts to read up to *numChars* characters into *buffer* starting at *buffer*[*offset*], returning the number of characters successfully read. –1 is returned when the end of the file is encountered.
boolean ready( )	Returns **true** if the next input request will not wait. Otherwise, it returns **false**.
void reset( )	Resets the input pointer to the previously set mark.

Character Based Stream

The Methods Defined by **Writer**

abstract void	close()	It closes the stream, flushing it first.
abstract void	flush()	It flushes the stream.
void	write(char[] cbuf)	It writes an array of characters.
abstract void	write(char[] cbuf, int off, int len)	It writes a portion of an array of characters.
void	write(int c)	It writes a single character.
void	write(String str)	It writes a string.
void	write(String str, int off, int len)	It writes a portion of a string.

```
public class FileWriterDemo {
     public static void main(String args[]) {
   String source = "Now is the time for all good men  to come to the aid of their country
and pay their due taxes.";

char buffer[] = new char[source.length()];
source.getChars(0, source.length(), buffer, 0);

try {
        FileWriter f0 = new FileWriter("file1.txt");
        FileWriter f1 = new FileWriter("file2.txt");
        FileWriter f2 = new FileWriter("file3.txt") )

    // write to first file
    for (int i=0; i < buffer.length; i += 2) {
      f0.write(buffer[i]);
    }

    // write to second file
    f1.write(buffer);

    f2.write(buffer,buffer.length-buffer.length/4,buffer.length/4);
   } catch(IOException e) {
    System.out.println("An I/O Error Occurred");
   }
```

```
try {
FileReader fr = new FileReader("FileReader.txt") )
 int c;

    // Read and display the file.
    while((c = fr.read()) != -1)
            System.out.print((char) c);

  }
catch(FileNotFoundException e) {
    System.out.println("I/O Error: " + e);
  }

catch(IOException e) {
    System.out.println("I/O Error: " + e);
  }


 }
}
```

Example

```
public class Files {
   public static void main(String args[]) throws FileNotFoundException, IOException
   {
      File f=new File("d:/abc\\demo.txt");
      BufferedReader br=new BufferedReader(new FileReader(f));
      BufferedWriter bw=new BufferedWriter(new FileWriter("d:/abc\\fw1.txt"));
      String str;
      BufferedReader console=new BufferedReader(new
InputStreamReader(System.in));
      while((str=br.readLine())!=null)
      {
          bw.append(str);
          str=console.readLine();
          str=str.toUpperCase();
          bw.append(str);

      }
      bw.close();
      br.close();
```

Difference Between Scanner And BufferedReader

BufferedReader	Scanner
Synchronous and should be used with multiple threads	Not synchronous and not used with multiple threads
Buffer memory is larger	Buffer memory is smaller
Faster than Scanner	Slower because it does parsing of the input data
Uses buffering to read characters from the character-input stream	It is a simple text scanner which parses primitive types and strings

Example

Consider a file which contains data in the format

Item_name Price Quantity

Read the data from the file and store it in item class

- **Sample Input**

  T-shirt 500 20
  Mug 100 40
  Juggling_Dolls 200 10
  Pin 10 200
  Key_Chain 50 500

- **Sample Output**

  You ordered 20 units of T-shirt at $500.00 and bill is 10000.00
  You ordered 40 units of Mug at $100.00 and bill is 4000.00
  You ordered 10 units of Juggling_Dolls at $200.00 and bill is 2000.00
  You ordered 200 units of Pin at $10.00 and bill is 2000.00
  You ordered 500 units of Key_Chain at $50.00 and bill is 25000.00

```
6
7  import java.io.*;
8  class order{
9      double price;
10     int unit;
11     String desc;
12     order(double price,int unit, String desc){
13         this.price=price;
14         this.unit=unit;
15         this.desc=desc;
16     }
17     double getBill(){
18         return unit*price;
19     }
20     void display(){
21          System.out.format("You ordered %d" + " units of %s at $%.2f and bill is %.2f \n",
22             unit, desc, price, unit*price );
23     }
24 }
```

```
public class datainput {
    public static void main(String a[]) throws FileNotFoundException, IOException
    {
        File f=new File("C:\\Users\\Kanak\\Desktop\\College\\OOP\\notes\\NetBeansP
        BufferedReader in = new BufferedReader(new FileReader(f));
        double price;
        int unit;
        String desc;
        order ord[]=new order[5];
        int i=0;
        String str;
        try {
            while ((str=in.readLine())!=null) {
                String[] ssplit=str.split(" ");
                desc=ssplit[0];
                price=Double.parseDouble(ssplit[1]);
                unit=Integer.parseInt(ssplit[2]);
                ord[i]=new order(price, unit, desc);
                ord[i].display();
            }
        } catch (EOFException e) {
            e.printStackTrace();
        }catch(IOException e){
            e.printStackTrace();
        }
     }}
```

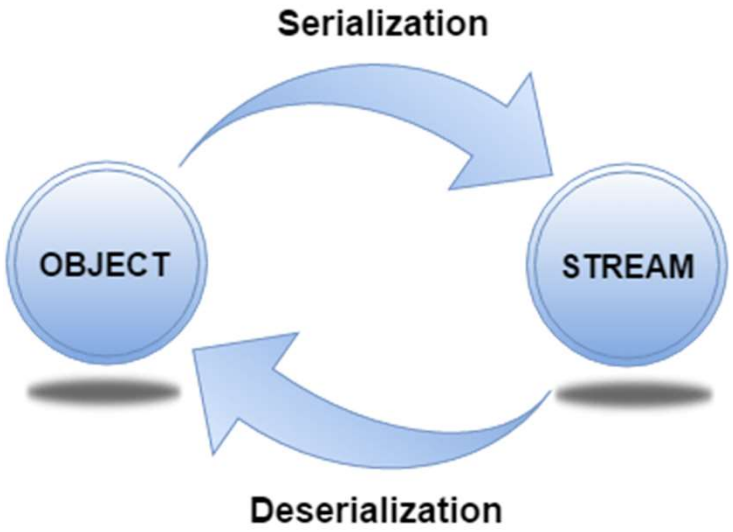
Do it Yourself

- Write a java program that reads a file containing data for 5 employees having empid, name and type of employee (permanent or adhoc). The file contains data in a comma separated format, one line per employee. Create the employee class. Store the data employee array. If employee is permanent assign Rs 5000 as his perks and if employee is adhoc assign Rs 1000 as his perks. Display the data

- Write a program to read data from console. Count number of words in the data given by user. Also count number of time character 'a' is used. Write the data given from user to file using **character based streams**. Also write the number of words and count of character 'a' to a new file.

Serialization

- **Serialization in Java** is a mechanism of *writing the state of an object into a byte-stream*. It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies.



Serializable Interface

java.io.Serializable interface

- **Serializable** is a marker interface (has no data member and method). It is used to "mark" Java classes so that the objects of these classes may get a certain capability.
- The **Serializable** interface must be implemented by the class whose object needs to be persisted.
- The String class and all the wrapper classes implement the *java.io.Serializable* interface by default.

Serialization Example

```
class MyClass implements Serializable {
String s;
int i;
double d;
public MyClass(String s, int i, double d) {
   this.s = s;
   this.i = i;
   this.d = d;
}
public String toString() {
   return "s=" + s + "; i=" + i + "; d=" + d;
}
}
```

```
import java.io.FileOutputStream;
import java.io.*;

public class SerializationDemo {

public static void main(String args[]) {
// Object serialization
try {
MyClass object1 = new MyClass("Hello", -7, 2.7e10);
System.out.println("object1: " + object1);
FileOutputStream fos = new FileOutputStream("serial.txt");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(object1);
oos.flush();
oos.close();
}
catch(IOException e) {

System.out.println("Exception during serialization: " + e);
System.exit(0);
}
```

```
// Object deserialization
try {
   MyClass object2;
   FileInputStream fis = new FileInputStream("serial.txt");
   ObjectInputStream ois = new ObjectInputStream(fis);
   object2 = (MyClass)ois.readObject();
   ois.close();
   System.out.println("object2: " + object2);
   //System.out.println("object2: " + object2);
}
catch(Exception e) {
   System.out.println("Exception during deserialization: " + e);
   System.exit(0);
}
}
}
```

Java Serialization with Inheritance

```
import java.io.Serializable;
class Person implements Serializable{
 int id;
 String name;
 Person(int id, String name) {
  this.id = id;
  this.name = name;
 }
}
class Student extends Person{
 String course;
 int fee;
 public Student(int id, String name, String course, int fee) {
  super(id,name);
  this.course=course;
  this.fee=fee;
 }
}
```

```
public class SerializeISA  {
 public static void main(String args[])   {
   try{
  Student s1 =new Student(211,"ravi","Engineering",50000);
   FileOutputStream fout=new FileOutputStream("f.txt");
 ObjectOutputStream out=new ObjectOutputStream(fout);
 out.writeObject(s1);
 out.flush();
 out.close();
 System.out.println("success");
 }
catch(Exception e){System.out.println(e);}
 try{
 //Creating stream to read the object
 ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt"));
 Student s=(Student)in.readObject();
 //printing the data of the serialized object
 System.out.println(s.id+" "+s.name+" "+s.course+" "+s.fee);
 //closing the stream
 in.close();
 }catch(Exception e){System.out.println(e);}    }   }
```

Java Serialization with Aggregation

```
class Address{
String addressLine,city,state;
public Address(String addressLine, String city, String state)
{
        this.addressLine=addressLine;
        this.city=city;
        this.state=state;
}
}
import java.io.Serializable;
public class Student implements Serializable{
 int id;
 String name;
 Address address;//HAS-A
 public Student(int id, String name) {
  this.id = id;
  this.name = name;
 }  }
```

Since Address is not Serializable, you cannot serialize the instance of the Student class.

Java Serialization with the static data member

```
class Employee implements Serializable{
 int id;
 String name;
 static String company="SSS IT Pvt Ltd";//it won't be serialized
  public Student(int id, String name) {
  this.id = id;
  this.name = name;
 }
}
```