IO Streams

In any programming language, in any application, providing input to the Applications and getting output from the Applications is essential.

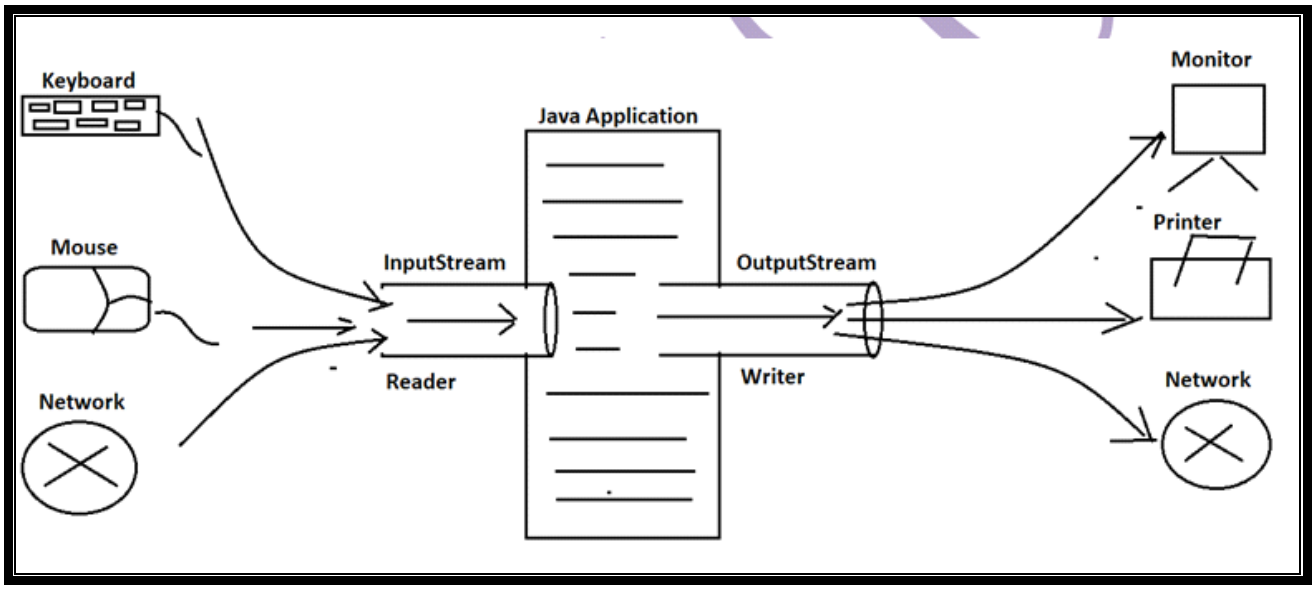
In case of C and C++ applications, we are able to perform input and output operations by using some predefined library in the form of printf(), scanf(), cin>>, cout<<,.....

Similarly in Java Applications, to perform input and output operations we have to use streams.

Java has represented all the streams in the form of predefined classes in "java.io" package.

Stream:

Stream is medium or channel; it will allow the data in continuous flow from input devices to java program and from Java program to output devices.



In Java IOStreams are divided into following ways:

- **Byte oriented Streams.**
- **Character-Oriented Streams**

• Byte-Oriented Streams:

These are Streams, which will allow the data in the form of bytes from input devices to Java program and from java program to output devices.
The length of the data in byte-oriented streams is 1 byte.

There are two types of Byte-Oriented Streams

- **InputStream**

- OutputStream

• InputStream:

It is a byte-oriented Stream, it will allow data in the form of bytes from input devices to Java Applications.

EX:
ByteArrayInputStream
FilterInputStream
DataInputStream
ObjectInputStream
FileInputStream
StringBufferInputStream
BufferedInputStream….

• OutputStream:

It is a byte-oriented Stream, it will allow the data in the form of bytes from Java applications to output devices.

EX:
ByteArrayOutputStream
FilterOutputStream
DataOutputStream
FileOutputStream
PrintStream
BufferedOutputStream..

NOTE: All the ByteOrientedStream classes are terminated with "Stream" word.

NOTE: The length of data items in Byte Oriented Streams is 1 byte.

• Character-Oriented Streams:

These are the Streams, which will allow the data in the form of characters from input devices to java program and form java program to output devices.

There are two bytes of character-oriented streams
- **Reader**
- **Writer**

Reader:

It is a character-oriented stream, it will allow the data in the form of characters from input devices to java program.

EX:
CharArrayReader
FilterReader
BufferedReader
FileReader
InputStreamReader….

Writer:

It is a character-oriented stream, it will allow the data in the form of characters from java program to output devices.

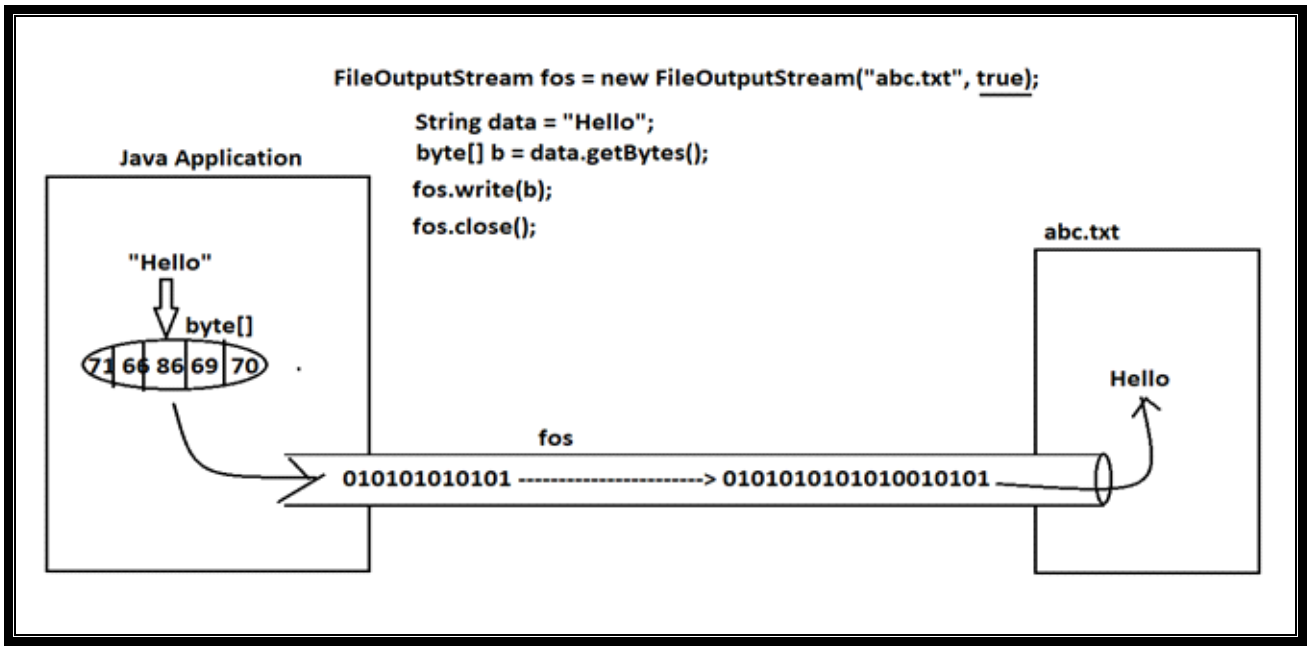
EX:
CharArrayWriter
FilterWriter
FileWriter
PrintWriter
BufferedWriter….

NOTE: All the predefined Classes of character-oriented streams are terminated with either Reader or Writer.

NOTE: The length of the data in characters-oriented stream is 2 bytes.

FileOutPutStream:

It is byte-oriented Stream, it can be used to transfer the data from Java program to a particular target file.
To transfer the data from Java program to a particular target file by using FileOutPutstream we have to use the following Steps.



- **Create FileOutPutStream between Java program and target file:**

  If we want to create FileOutPutStream class object then we have to use the following constructors

  public FileOutPutStream(String target_File)
  public FileOutPutStream(String target_File,boolean b)

EX:

FileOutPutStream fos = new FileOutPutStream("abc.txt");
It will override the existed data in the target file at each and every write operation.

FileOutPutStream fos=new FileOutPutStream("abc.txt",true);
It will not override the existed data in the target file, it will append the specified new data to the existed data in the target file.

When JVM encounter the above instruction, JVM will perform the following tasks.

- JVM will take the specified target file.
- JVM will search for the specified target file at the respective location.
- If the specified target file is available then JVM will establish FileOutPutStream from java program to target file.
- If the specified target file is not available then JVM will create a file with the target file name and establish FileOutPutStream from Java program to target file.

Declare the data and convert into byte[]:

```
String data = "Hello";
byte[] b = data.getBytes();
```

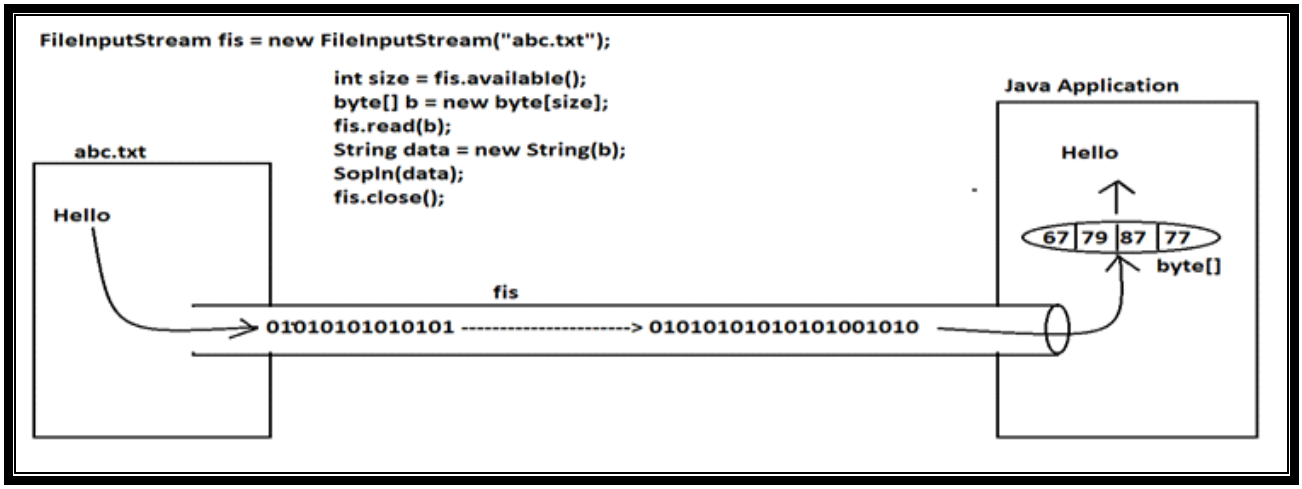
- **Write Byte Array data into FileOutPutStream:**

To write byte[] data into FileOutPutStream,we have to use the following method.
public void write(byte[] b) throws IOException

EX:
fos.write(b);
4)Close FileOutPutStream:
fos.close();

2) FileInputStream:

It is a byte-oriented Stream, it can be used to transfer the data from a particular source file to Java Program.
If we want to transfer the data from source file to java program by using FileInputStream,



We have to use the following Steps:

- ## Create FileInputStream class Object:

  To create FileInputStream class object,we have to use the following constructor from java.io.FileInputStream class.
  public FileInputStream(String file_name) throws FileNotFoundException

  EX: FileInputStream fis = new FileInputStream("abc.txt");

When JVM encounter the above instruction then JVM will perform the following actions.

- JVM will take the specified source file name.
- JVM will search for the specified source file at the respective location.
- If the source file is not available at the respective location then JVM will raise an excpetion like "java.io.FileNotFoundException".
- If the required source file is available then JVM will establish FileInputStream from source file to JAVA program.
- After creating FileInputStream, JVM will transfer the data from source file to FileInputStream in the form bytes.

- ## Get the Size of the Data from FileInputStream and prepare byte[] with the Data Size:

  To get the size of the data from FileInputStream, we have to use the following method
  public int available()

  EX: int size = fis.available();
  byte[] b = new byte[size];

- ## Read the data from FileInputStream into byte[]:

  To read the data from FileInputStream into byte[],we have to use the following method.
  public void read(byte[] b)throws IOException
  EX: fis.read(b);

- ## Convert data from byte[] to String:

  String data = new String(b);
  System.out.println(data);

Close FileInputStream:

fis.close();

Write a Java program to display particular file content on command prompt by taking filename as command line input?

```
1)  import java.io.*;
2)  class DisplayEx {
3)        public static void main(String args[]) throws Exception {
4)              String file_Name = args[0];
5)              FileInputStream fis = new FileInputStream(file_Name);
6)              int size = fis.available();
7)              byte b[] = new byte[size];
8)              fis.read();
9)              String data = new String(b);
10)             System.out.println(data);
11)             fis.close();
12)       }
13) }
```

Write a Java program to count no of words available in a particular text file and how many times the word "Durga" is repeated?

```
1)  import java.io.*;
2)  import java.util.*;
3)  class Word_Count_Ex {
4)        public static void main(String args[]) throws Exception {
5)              FileInputStream fis = new FileInputStream("abc.txt");
6)              int size = fis.available();
7)              byte b[] = new byte[size];
8)              fis.read();
9)              String data = new String(b);
10)             StringTokenizer st = new StringTokenizer(data);
11)             int tokens = st.countTokens();
12)             System.out.println("No of words :"+tokens);
13)             int count = 0;
14)             while(st.hasMoreTokens()) {
15)                     String token = st.nextToken();
16)                     if(token.equals("Durga")) {
17)                             count = count+1;
18)                     }
19)             }
20)             System.out.println("'Durga' is repeated :"+count);
21)             fis.close();
22)     }
23) }
```

Write a Java program to copy an image from a source file to a particular target file?

```
1)  import java.io.*;
2)  public class Image_Copy_Ex {
3)        public static void main(String args[]) {
4)              FileInputStream fis = new FileInputStream();
5)              int size = fis.available();
6)              byte[] b = new byte[size];
7)              fis.read(b);
8)              FileOutPutStream fos = new FileOutPutStream("abc.jpg");
9)              fos.write(b);
10)             fis.close();
11)             fos.close();
12)       }
13) }
```

FileWriter:

This character-oriented Stream can be used to transfer the data from Java Application to a particular target File.
If we want to transfer the data from java applications to a particular target file by using

FileWriter then we have to use the following steps:

- **Create FileWriter Object:**

To create FileWriter class object,we have to use the following constructor.
public FileWriter(String target_File)

EX: FileWriter fw = new FileWriter("abc.txt");
It will override the existed content with the new content at each and every write operation.

public FileWriter(String target_File,boolean b)
EX: FileWriter fw = new FileWriter("abc.txt",true);

It will append new content to the existed content available in the file at each and every write operation.
When JVM encounter the above instructions, JVM will take the specified file and JVM search for the specified file at the respective location, if the required target file is available then JVM will establish FileWriter from Java application to the target file. If the required target file is not available at the respective location then JVM will create a new file with the same specified file name and establish FileWriter from Java application to the target file.

- **Declare the data which we want to transfer and convert that data into char[]:**

```
String data = "Hello";
char[] ch = data.toCharArray();
```

- **Write char[] data into FileWriter:**

To write char[] data into FileWriter,we have to use the following method.
public void write(char[] ch)throws IOException
EX: fw.write(ch);

- **Close FileWriter:**

```
fw.close();
```

EX:

```
1)  import java.util.*;
2)  public class FileWriterEx {
3)          public static void main(String args[]) throws Exception {
4)                  FileWriter fw = new FileWriter("abc.txt",true);
5)                  String data = "DurgaSoftwareSolutions";
6)                  char[] ch = data.toCharArray();
7)                  fw.write(ch);
8)                  fw.close();
9)          }
10) }
```

FileReader:

This character-oriented stream can be used to transfer the data from a particular source file to Java program.

If we want to transfer the data from a particular source file to Java program by using FileReader then we have to use the following steps:

- ## Create FileReader Class Object:

To create FileReader class object,we have to use the following constructor.

public FileReader(String file_Name) throws FileNotFoundException

EX: **FileReader fr = new FileReader("abc.txt");**

When JVM encounter the above instruction, JVM will perform the following steps.

- **JVM will take source file name from FileReader constructor.**
- **JVM will check whether the specified file is available or not at the respective location.**
- **If the specified source file is not available at the respective location then JVM will rise an exception like "java.io.FileNotFoundException".**
- **If the specified file is existed at the respective location then JVM will establish FileReader from source file to Java program.**
- **After creating FileReader, JVM will transfer the content of source file to FileReader object in the form of characters.**

- ## Read Data from FileReader:

To read data from FileReader, we have to use the following steps.

- Read character by character from FileReader in the form of ASCII values.
- Convert that ASCII values into the respective characters.
- Append the converted characters to a String variable.

Repeat the above steps up to all the characters which are available in the respective source file or up to the end-of-file character i.e "-1".

To read an ASCII value from FileReader, we have to use the following method.
public int read() throws IOException

- **Close FileReader:**
  fr.close();

EX:

```
1)  import java.util.*;
2)  public class FREx {
3)      public static void main(String args[])throws Exception {
4)          FileWriter fr = new FileWriter("abc.txt");
5)          String data="";
6)          int val = fr.read();
7)          while(val != -1) {
8)              data = data+(char)val;
9)              val = fr.read();
10)         }
11)         System.out.println(data);
12)         fr.close();
13)     }
14) }
```

Write A Java Program To Copy A Document From One File To Another File By Using Character Oriented Streams?

```
1)  import java.io.*;
2)  public class FileCopyEx {
3)      public static void main(String args[])throws Exception {
4)          FileReader fr = new FileReader("hibernatecgf.xml");
5)          String data="";
6)          int val = fr.read();
7)          while(val != -1) {
8)              data = data+(char)val;
9)              val = fr.read();
10)         }
11)         char[] ch = data.toCharArray();
12)         FileWriter fw = new FileWriter("abc.xml");
```

```
13)             fw.write(ch);
14)             fr.close();
15)             fw.close();
16)         }
17) }
```

Approaches to provide dynamic Input:

There are three approaches to provide dynamic input in java applications.

- BufferedReader
- Scanner
- Console

BufferedReader:

If we want to take dynamic input by using BufferedReader in java applications then we have to use the following statement.

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
Where "in" is static variable,it will refer a predefined "InputStream" object which is connected with command prompt.

If we provide data on command prompt then that data will be transferred to InputStream object in the form of binary data.

where "InputStreamReader can be used to convert the data from binary representation to character representation.

where BufferedReader can be used to improve the performance of Java application while performing input operation.

To read the data from BufferedReader, we will use the following method

- readLine()
- read()

Q) What is the difference between readLine() Method and

read() Method?

readLine() method will read a line of text from command prompt [BufferedReader] and it will return that data in the form of String.

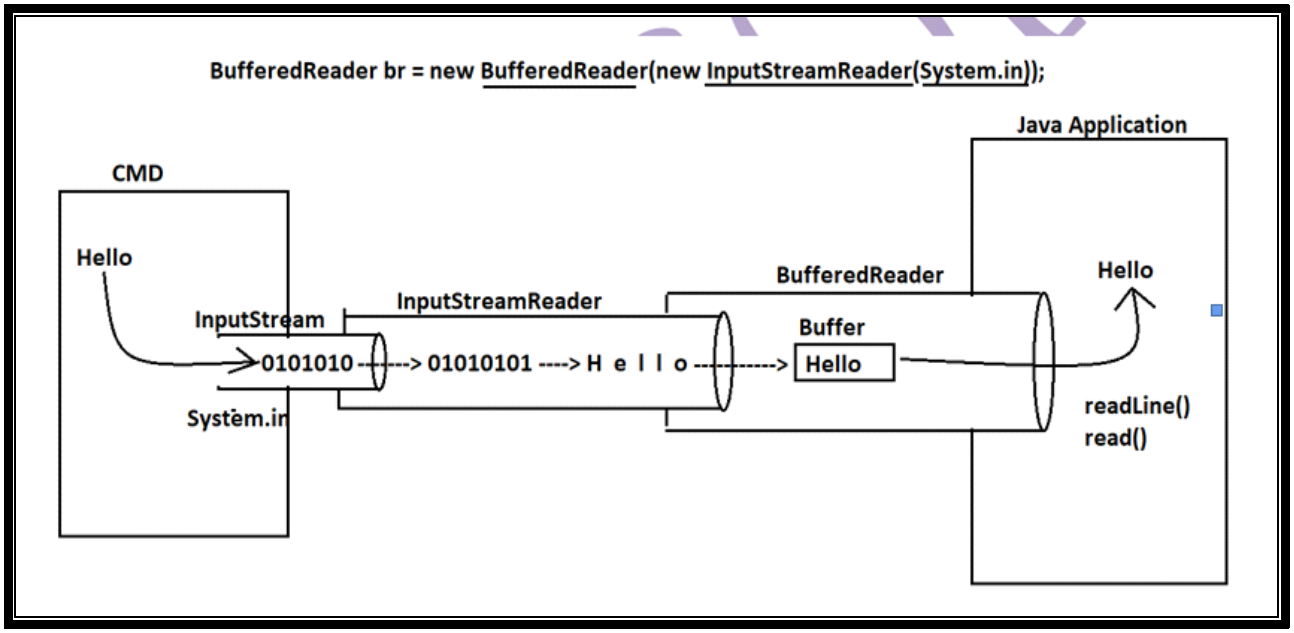
public String readLine() throws IOException

read() method will read a single character from command prompt [BufferedReader] and it will return that character in the form of its ASCII value.

public int read()throws IOException

EX:

```
1)  import java.io.*;
2)  public class BufferedReaderEx {
3)        public static void main(String args[]) throws Exception {
4)     BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
5)              System.out.println("Enter Text :");
6)              String data1 = br.readLine();
7)              System.out.println("Enter the same text again :");
8)              int data2 = br.read();
9)              System.out.println("First Entered :"+data1);
10)             System.out.println("Second Entered :"+data2+"--->"+(char)data2);
11)     }
12) }
```



Consider the following Program:

```
1)  import java.io.*;
2)  public class BufferedReaderEx {
3)        public static void main(String args[]) throws Exception {
4)     BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
5)              System.out.println("First value :");
6)              String val1 = br.readLine();
7)              System.out.println("Second value :");
8)              String val2 = br.readLine();
9)              System.out.println("Addition :"+val1+val2);
10)       }
11) }
```

If we provide 10 and 20 as dynamic input to the above program then the above program will display "1020" value instead of 30 that is the above program has performed String concatenation instead of performing Arithmetic Addition because br.readLine() method has return 10 and 20 values in the form String data.
In the above program, if we want to perform Arithmetic operations over dynamic input then we have to convert String data into the respective primitive data, for this we have to use Wrapper Classes.

ThereFore, BufferedReader dynamic input approach is depending on wrapper classes while reading primitive data as dynamic input.

EX:

```
1)  import java.io.*;
2)  public class BufferedReaderEx {
3)        public static void main(String args[]) throws Exception {
4)     BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
5)              System.out.println("First value :");
6)              String val1 = br.readLine();
7)              System.out.println("Second value :");
8)              String val2 = br.readLine();
9)              int f_Val = Integer.parseInt(val1);
10)             int s_Val = Integer.parseInt(val2);
11)             System.out.println("Addition :"+(f_Val+s_Val));
12)       }
13) }
```

Scanner:

- This class is provided by Java in java.util package along with JDK5.0 Version.
- In java applications, if we use BufferedReader to dynamic input then we must use wrapper classes while reading primitive data as dynamic Input.
- In java applications, if we use "Scanner" to read dynamic input then it is not required to use wrapper classes while reading primitive data as dynamic input, scanner is able to provide environment to read primitive data directly from command prompt.

- If we want to use scanner in Java applications then we have to use the following steps.

- ## Create Scanner class Object:

To create Scanner class Object, we have to use the following constructor
public Scanner(InputStream is)
EX: Scanner s=new Scanner(System.in);

- ## Read dynamic Input:

To read String data as Dynamic input, we have to use the following method.
public String next()
To read primitive data as Dynamic input, we have to use the following method.
public xxx nextXXX()
where xxx may be byte,short,int,float.....

EX:

```
1)  import java.util.*;
2)  public class ScannerEx
3)  {
4)     public static void main(String[] args)throws Exception
5)     {
6)        Scanner s=new Scanner(System.in);
7)        System.out.print("Employee Number  :");
8)        int eno=s.nextInt();
9)        System.out.print("Employee Name    :");
10)       String ename=s.next();
11)       System.out.print("Employee Salary  :");
12)       float esal=s.nextFloat();
13)       System.out.print("Employee Address :");
14)       String eaddr=s.next();
15)
16)       System.out.println("Employee Details");
17)       System.out.println("--------------------");
18)       System.out.println("Employee Number   :"+eno);
```

```
19)      System.out.println("Employee Name    :"+ename);
20)      System.out.println("Employee Salary  :"+esal);
21)      System.out.println("Employee Address :"+eaddr);
22)   }
23) }
```

EX:

```
1)  import java.util.*;
2)  public class ScannerEx1
3)  {
4)     public static void main(String[] args)throws Exception
5)     {
6)        Scanner s=new Scanner(System.in);
7)        System.out.print("Enter Text Data :");
8)        String data1=s.nextLine();
9)        System.out.print("Enter The same text data again :");
10)       String data2=s.next();
11)       System.out.println("First Entered   :"+data1);
12)       System.out.print("Second Entered    :"+data2);
13)    }
14) }
```

Console:

This class is provided by Java in java.io package along with JAVA 6 Version.
In Java applications, to take dynamic input, if we use BufferedReader and Scanner then we are able to get the following drawbacks:

- We have to consume 2 instructions for each and every dynamic input [s.o.pln(..) and readLine() or nextXXX() methods]
- These are not providing security for the data like password data, pin numbers.....

- To overcome the above problems, we have to use "Console" dynamic input approach.
- If we want to use Console in Java applications then we have to use the following Steps:

Create Console Object:

To get Console object, we have to use the following method from "System" class.
public static Console console()
EX: Console c = System.console();

Read Dynamic Input:

To read String data, we have to use the following method.
public String readLine(String msg)
To read password data,we have to use the following method.
public char[] readPassword(String msg)

EX:

```
1)  import java.io.*;
2)  public class ConsoleEx {
3)        public static void main(String args[]) throws Exception {
4)              Console c = System.console();
5)              String uname = c.readLine("User Name :");
6)              char[] pwd = c.readPassword("PassWord :");
7)              String upwd = new String(pwd);
8)              if(uname.equals("durga") && upwd.equals("durga")) {
9)                    System.out.println("Valid User");
10)             }
11)             else {
12)                  System.out.println("InValid User");
13)       }
14)    }
15) }
```

Serialization and Deserialization:

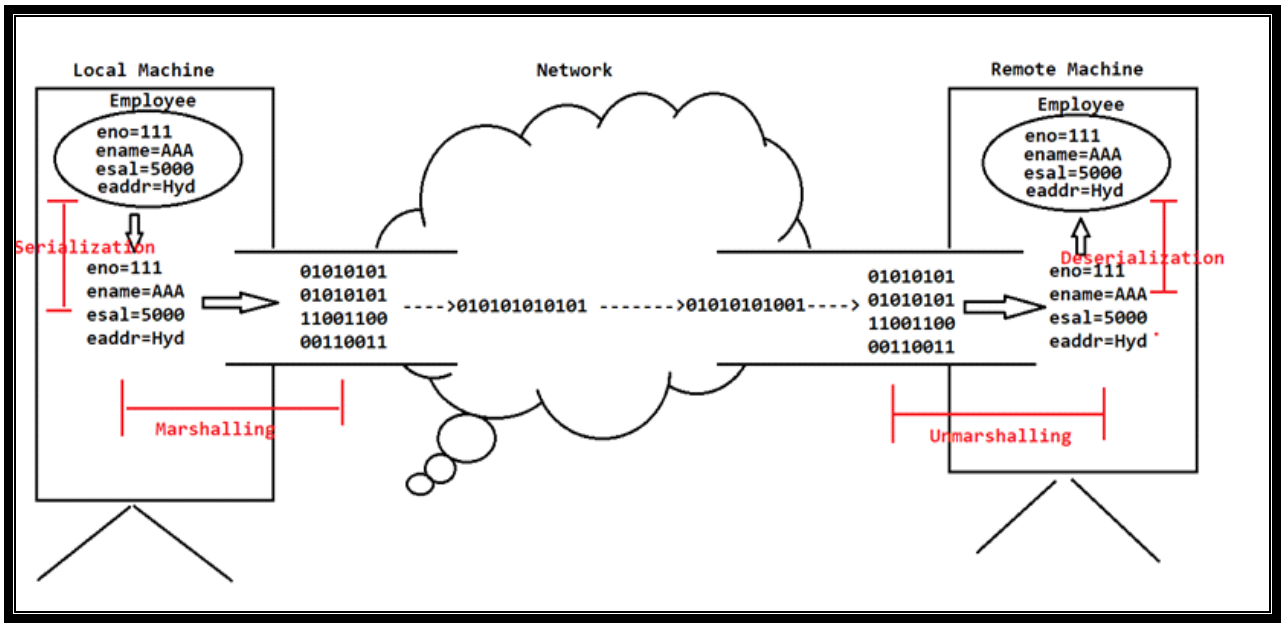
If we design Java applications by distributing application logic over multiple [JVMS] then that Java application is called as Distributed Application.

In general, in Distributed applications, it is frequent requirement to transfer an object [Distributed Object] from one machine to another machine.

In Java, Object is a block of memory, it is not possible to transfer the object through network, where we have to transfer object data from one machine to another machine through network.

To transfer an Object through network from one machine to another machine, first we have to separate the data from an object at local machine and convert the data from system representation to network representation then transfer the data to network.

At remote machine, we have to get the data from network and convert the data from system representation to System representation and reconstruct an object on the basis of data.



The process of converting the data from System representation to network representation is called as "Marshalling".

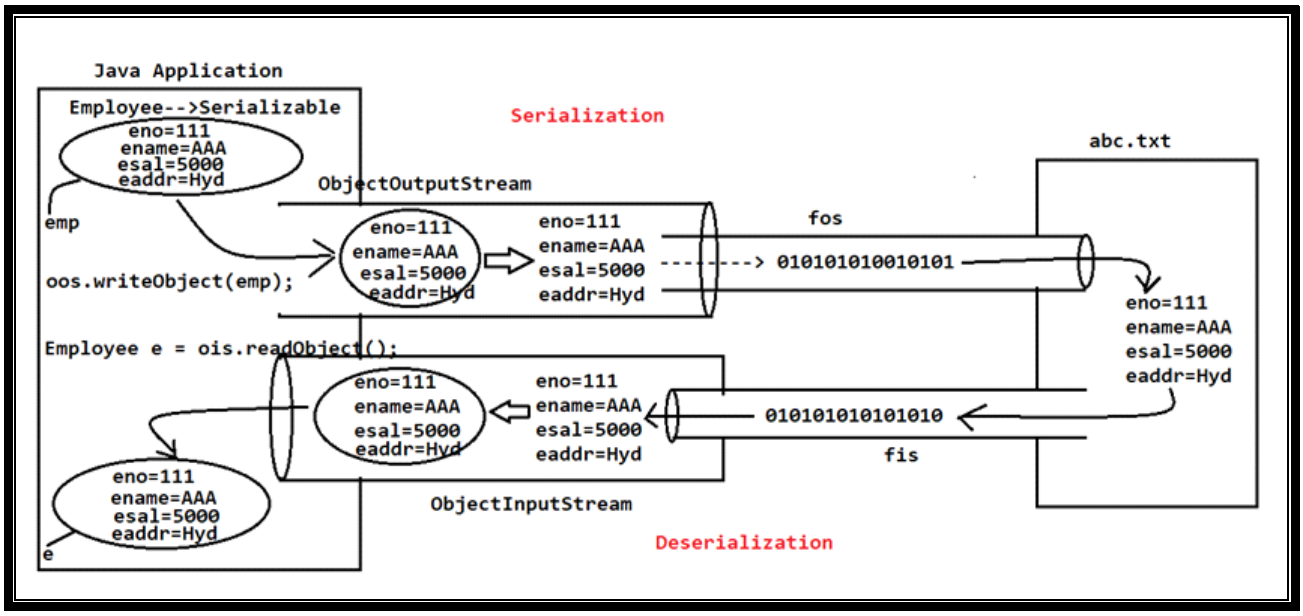
The process of converting the data from Network representation to System representation is called as "UnMarshalling".

The Process of separating the data from an Object is called as "Serialization".

The process of reconstructing an object on the basis of data is called as "Deserialization".

To perform Serialization and Deserialization in Java applications, JAVA has given two predefined byte-oriented streams like java.io.ObjectOutputStream for Serialization java.io.ObjectInputStream for Deserialization

In Standalone applications,if we want to perform Serialization and Deserialization over an object then we have to take a file[text file] to store serialized data.



Steps to Perform Serialization:

- ## Create Serializable Object:
- **To create Serializable Object we have to implement java.io.Serializable marker interface to the respective class.**
- **Serializable interface is marker interface, it will make eligible any object for Serialization and Deserialization.**

EX:

```
class Employee implements Serializable {
        int eno = 111;
        String ename = "AAA";
        float esal = 5000;
}
Employee e1 = new Employee();
```

- ## Prepare FileOutPutStream with a particular target File:

FileOutPutStream fos=new FileOutPutStream("abc.txt");

- ## Create ObjectOutPutStream:

To create ObjectoutputStream,we have to use the following constructor.
public ObjectOutputStream(FileOutputStream fos)

EX: ObjectOutputStream oos=new ObjectOutputStream(fos);

- **Write Serializable object to ObjectOutputStream:**

To write Serializable object to ObjectOutputStream,we have to use the following method.

public void writeObject(Object obj)throws NotSerializableException

EX: oos.writeObject(e1);

Steps To perform DeSerialization:

- **Create FileInputStream object:**

FileInputStream fis = new FileInputStream("emp.txt");

- **Create ObjectInputStream:**

To create ObjectInputStream class object,we have to use the Following constructor.

public ObjectInputStream(FileInputStream fis)

EX: ObjectInputStream ois=new ObjectInputStream(fis);

- **Read DeSerialized Data from ObjectInputStream:**

To read DeSerialized object from ObjectInputStream,we have to use the following method.

public Object readObject()

EX: Employee e2 = (Employee)ois.readObject();

EX:

```
1)  import java.io.*;
2)  class Employee implements Serializable
3)  {
4)     int eno;
5)     String ename;
6)     float esal;
7)     String eaddr;
8)
9)     Employee(int eno, String ename, float esal, String eaddr)
10)    {
11)       this.eno=eno;
12)       this.ename=ename;
13)       this.esal=esal;
14)       this.eaddr=eaddr;
15)    }
16)
17)    public void getEmpDetails()
18)    {
19)       System.out.println("Employee Details");
```

```
20)      System.out.println("----------------------");
21)      System.out.println("Employee Number  :"+eno);
22)      System.out.println("Employee Name    :"+ename);
23)      System.out.println("Employeed Salary :"+esal);
24)      System.out.println("Employee Address :"+eaddr);
25)   }
26) }
27) class SerializationEx
28) {
29)   public static void main(String[] args)throws Exception
30)   {
31)      FileOutputStream fos=new FileOutputStream("emp.txt");
32)      ObjectOutputStream oos=new ObjectOutputStream(fos);
33)      Employee emp1=new Employee(111, "Durga", 50000, "Hyd");
34)      System.out.println("Employee Details before Serialization");
35)      emp1.getEmpDetails();
36)      oos.writeObject(emp1);
37)      System.out.println();
38)      FileInputStream fis=new FileInputStream("emp.txt");
39)      ObjectInputStream ois=new ObjectInputStream(fis);
40)      Employee emp2=(Employee)ois.readObject();
41)      System.out.println("Employee Details After Deserialization");
42)      emp2.getEmpDetails();
43)   }
44) }
```

- In Object serialization, static members are not allowed.
- If we serialize any object having static variables then compiler will not rise any error and JVM will not rise any exception but static variables will not be listed in the serialized data in the text file.
- In object serialization, if we do not want to allow any variable in serialization and deserialization then we have to declare that variable as "transient" variable.
- transient int eno=111;

EX:

```
1)  import java.io.*;
2)  class User implements Serializable {
3)     String uname;
4)     transient String upwd;
5)     String uemail;
6)     long umobile;
7)     public static final int MIN_AGE=18;
8)     public static final int MAX_AGE=25;
9)
10)    User(String uname, String upwd, String uemail, long umobile)
```

```
11)   {
12)      this.uname=uname;
13)      this.upwd=upwd;
14)      this.uemail=uemail;
15)      this.umobile=umobile;
16)   }
17) }
18) class Test
19) {
20)   public static void main(String[] args)throws Exception
21)   {
22)      FileOutputStream fos=new FileOutputStream("abc.txt");
23)      ObjectOutputStream oos=new ObjectOutputStream(fos);
24)      User u=new User("abc", "abc123", "abc@duurgasoft.com", 998877);
25)      oos.writeObject(u);
26)   }
27) }
```

In Java applications, if we serialize an object which is not implementing java.io.Serializable interface then JVM will rise an exception like "java.io.NotSerializableException".

```
1)  import java.io.*;
2)  class A {
3)       int I = 10;
4)       int j = 20;
5)  }
6)  class Test {
7)       public static void main(String args[])throws Exception {
8)              FileOutputStream fos = new FileOutputStream("abc.txt");
9)              ObjectOutputStream oos = new ObjectOutputStream(fos);
10)             A a = new A();
11)             oos.writeObject(a);
12)      }
13) }
```

Status:"java.io.NotSerializableException".[Exception]

In Java applications, if we implement Serializable interface to the super class then automatically all the sub class objects are eligible for Serialization and Deserialization.

EX:

```
1)  import java.io.*;
2)  class A implements Serializable {
3)        int I = 10;
4)        int j = 20;
5)  }
6)  class B extends A {
7)        int k = 30;
8)        int I = 40;
9)  }
10) class Test {
11)       public static void main(String args[]) throws Exception {
12)             FileOutputStream fos = new FileOutputStream("abc.txt");
13)             ObjectOutputStream oos = new ObjectOutputStream(fos);
14)             B b = new B();
15)             oos.writeObject(b);
16)       }
17) }
```

In Java applications, if we implement Serializable interface in sub class then only sub class properties are allowed in Serialization and deserialization, the respective super class members are not allowed in the Serialization and deserialization.

EX:

```
1)  import java.io.*;
2)  class A  {
3)        int i = 10;
4)        int j = 20;
5)  }
6)  class B extends A implements Serializable {
7)        int k = 30;
8)        int I = 40;
9)  }
10) class Test {
11)       public static void main(String args[]) throws Exception {
12)             FileOutputStream fos = new FileOutputStream("abc.txt");
13)             ObjectOutputStream oos = new ObjectOutputStream(fos);
14)             B b = new B();
15)             oos.writeObject(b);
16)       }
17) }
```

In Java applications, while Serializing an object if any associated object is available then JVM will serialize the respective associated object also but the respective associated

object must implement Serializable interface otherwise JVM will rise an exception like "java.io.NotSerializableException".

EX:

```
1)  import  java.io.*;
2)  class Branch implements Serializable {
3)        String bid;
4)        String bname;
5)        Branch(String bid, String bname) {
6)              this.bid = bid;
7)              this.bname = bname;
8)        }
9)  }
10) class Account implements Serializable {
11)         String accNo;
12)         String accName;
13)         Branch branch;
14)         Account(String accNo, String accName, Branch branch) {
15)                 this.accNo = accNo;
16)                 this.accName = accName;
17)                 this.branch = branch;
18)         }
19) }
20) class Employee implements Serializable {
21)         String eid;
22)         String ename;
23)         Account acc;
24)         Employee(String eid, String ename, Account acc) {
25)                 this.eid = eid;
26)                 this.ename = ename;
27)                 this.acc = acc;
28)         }
29) }
30) class Test {
31)       public static void main(String args[]) throws Exception {
32)                 FileOutputStream fos = new FileOutputStream("abc.txt");
33)                 ObjectOutputStream oos = new ObjectOutputStream(fos);
34)                 Branch branch = new Branch("B-111","S R Nagar");
35)                 Account acc = new Account("abc123","Durga",branch);
36)                 Employee emp = new Employee("E-111","Durga",acc);
37)                 oos.writeObject(emp);
38)       }
39) }
```

Externalization:

As part of object serialization and deserialization we are able to separate the data from Object and stored in a text file and we are able to retrieve that object data from text file to Object in Java application.

In Java applications, to perform serialization and deserialization Java has given "ObjectOutputStream" and "ObjectInputStream" two byte-oriented Streams.

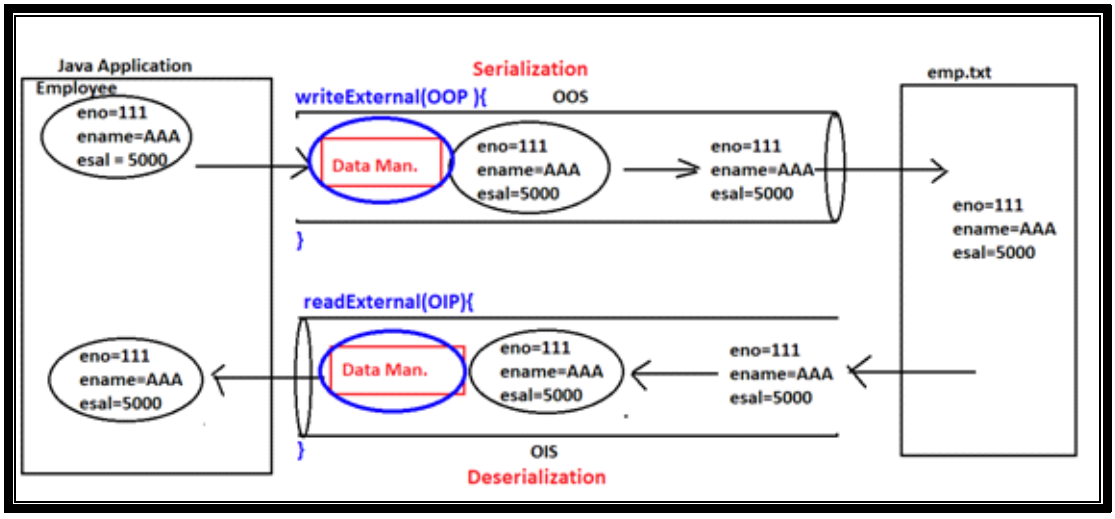
In Java applications, to perform Serialization just we have to send Serializable object to ObjectOutputStream,it will perform serialization internally, where Developers are not having controlling over serialization process.

In Java applications, to perform deserialization just we have to create ObjectInputStream and we have to read deserialized object, where ObjectInputStream will perform deserialization internally, where developers are not having controlling over deserialization process.

In the above context to have controlling over serialization and deserialization processes inorder to provide the services like security, data compression, data decompression, data encoding.....over serialized and deserialized data we have to go for "Externalization".

If we want to perform Externalization in java applications, we have to use the following steps.

1) Prepare Externalizable object
2) Perform Serialization and Deserialization over Externalizable object.



Prepare Externalizable Object:

In Java applications,if we want to create Serializable object then the respective class must implement java.io.Serializable interface.
Similarly, if we want to prepare Externalizable object then the respective class must implement java.io.Externalizable interface.
java.io.Externalizable is a sub interface to java.io.Serializable interface.

java.io.Serializable interface is a marker interface, which is not having abstract methods but java.io.Externalizable interface is not marker interface, which includes the following methods.

```
public void writeExternal(ObjectOutput oop)throws IOException
public void readExternal(ObjectInput oip)throws IOException, ClassNotFoundException
```

where writeExternal(--) method will be executed just before performing serialization in ObjectOutputStream,
where we have to perform manipulations on the data which we want to serialize.

where readExternal(--) method will be executed immediately after performing Deserializationin ObjectInputStream, where we can perform manipulations over the deserialized data.
where ObjectOutput is stream, it will carry manipulated data for Serialization.

To put data in ObjectOutput,we have to use the following methods.
public void writeXXX(xxx data)
where xxx may be byte, short, int, UTF[String].....
where ObjectInput will get serialized data from text file to perform manipulations.
To read data from ObjectInput we have to use the following method
public void readXXX(xxx data)
where xxx may be byte, short, int, UTF[String].....

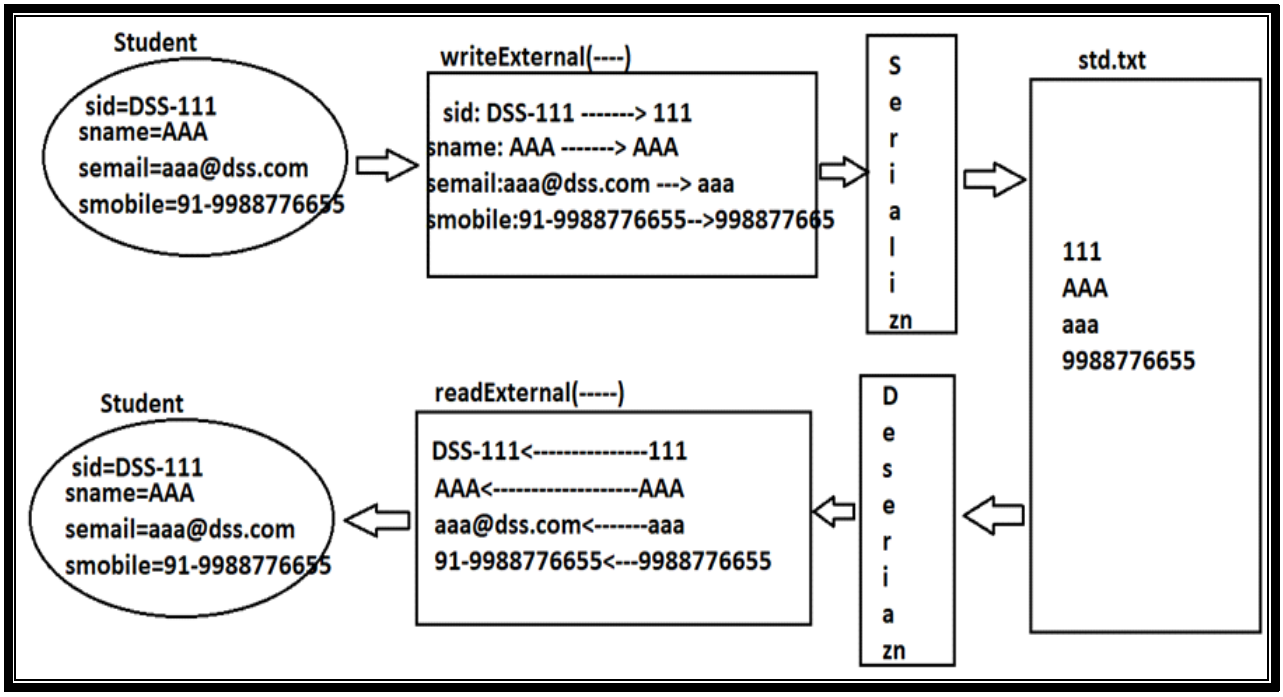
If we want to prepare Externalizable object we have to use the following steps.

- **Declare an user defined class**
- **Implement java.io.Externalizable interface.**
- **Implement the methods writeExternal(--) and readExternal(--)**

of Externalizable interface at the user defined class.

```
class Employee implements Externalizable {
    public void writeExternal(ObjectOutput oop) throws IOException {
            ---implementation---
    }
    public void readExternal(ObjectInput oip) throws IOException,
                                                    ClassNotFoundException {
            ---implementation----
    }
```

```
}
Employee emp = new Employee();
```



Perform Serialization and Deserialization over Externalizable object by using ObjectOutputStream and ObjectInputStream:

Same as Serialization and DeSerialization

EX:

```
1)  import java.util.*;
2)  import java.io.*;
3)  class Employee implements Externalizable {
4)      String eid;
5)      String ename;
6)      String email;
7)      String emobile;
8)      //It will be used to construct object while performing deserialization in
9)        Externalization process
10)     public Employee() {
11)     }
12)     Employee(String eid, String ename, String email, String emobile) {
13)           this.eid = eid;
14)           this.ename = ename;
15)           this.email = email;
16)           this.emobile = emobile;
17)     }
18)     public void writeExternal(ObjectOutput oop) throws IOException {
```

```
19)         try {
20)                 StringTokenizer st1=new StringTokenizer(eid,"-");
21)                 st1.nextToken();
22)                 int no=Integer.parseInt(st1.nextToken());
23)                 StringTokenizer st2=new StringTokenizer(email,"@");
24)                 String mail=st2.nextToken();
25)                 StringTokenizer st3=new StringTokenizer(emobile,"-");
26)                 st3.nextToken();
27)                 String mobile=st3.nextToken();
28)                 oop.writeInt(no);
29)                 oop.writeUTF(ename);
30)                 oop.writeUTF(mail);
31)                 oop.writeUTF(mobile);
32)         }
33)         catch(Exception e) {
34)             e.printStackTrace();
35)         }
36)     }
37)     public void readExternal(ObjectInput oip) throws IOException,
    ClassNotFoundException {
38)          eid = "E-"+oip.readInt();
39)          ename = oip.readUTF();
40)          email = oip.readUTF()+"@durgasoft.com";
41)          emobile = "91-"+oip.readUTF();
42)     }
43)     public void getEmpDetails() {
44)          System.out.println("Employee Details");
45)          System.out.println("----------------");
46)          System.out.println("Employee Id  :  "+eid);
47)          System.out.println("Employee Name : "+ename);
48)          System.out.println("Employee Mail : "+email);
49)          System.out.println("Employee Mobile: "+emobile);
50)     }
51) }
52) class ExternalizableEx {
53)         public static void main(String args[]) throws Exception {
54)                 FileOutputStream fos = new FileOutputStream("emp.txt");
55)                 ObjectOutputStream oos = new ObjectOutputStream(fos);
56)                 Employee emp1 = new Employee("E- 111", "Durga",
    "durga@durgasoft.com","91-9988776655");
57)                 System.out.println("Employee Data before Serialization");
58)                 emp1.getEmpDetails();
59)                 oos.writeObject(emp1);
60)                 System.out.println();
61)                 FileInputStream fis = new FileInputStream("emp.txt");
```

```
62)                 ObjectInputStream ois = new ObjectInputStream(fis);
63)                 Employee emp2 = (Employee)ois.readObject();
64)                 System.out.println("Employee Data After Deserialization"):
65)                 emp2.getEmpDetails();
66)         }
67) }
```

Files in Java:

File is a storage area to store data.
There are two types of files in Java.

- Sequential Files
- RandomAccessFiles

• Sequential Files:

It will allow the user to retrieve data in Sequential manner.
To represent Sequential files, Java has given a predefined class in the form of java.io.File.

To create File class object we have to use the following constructor.
public File(String file_Name) throws FileNotFoundException

EX: File f = new File("c:/abc/xyz/emp.txt");

Creating File class object is not sufficient to create a file at directory structure we have to use the following method.
public File createNewFile()

To create a Directory, we have to use the following method.
public File mkdir()

To get file / directory name we have to use the following method.
public String getName()

To get file / directory parent location, we have to use the following method.
public String getParent()

To get file / directory absolute path, we have to use the following method.
public String getAbsolutePath()

To check whether the created thing File or not,we have to use the following method.

public boolean isFile()

To check whether the created thing is directory or not we have to use the following method.
public boolean isDirectory()

EX:

```
1)  import java.io.*;
2)  class Test {
3)      public static void main(String args[]) throws Exception {
4)          File f = new File("c:/abc/xyz/emp.txt");
5)          f.createNewFile();
6)          System.out.println(f.isFile());
7)          System.out.println(f.isDirectory());
8)          File f1 = new File("c:/abc/xyz/student");
9)          f1.mkdir();
10)         System.out.println(f.isFile());
11)         System.out.println(f.isDirectory());
12)         System.out.println("File Name :"+f.getName());
13)         System.out.println("Parent Name :"+f.getParent());
14)         System.out.println("Absolute Path :"f.getAbsolutePath());
15)         int size = fis.available();
16)         byte[] b = new byte[size];
17)         fis.read();
18)         String data = new String(b);
19)         System.out.println(data);
20)     }
21) }
```

RandomAccessFile:

It is a Storage area, it will allow the user to read data from random positions. To represent this file, java has given a predefined class in the form of "java.io.RandomAccessFile".

To create RandomAccessFile class object, we have to use the following constructor.
public RandomAccessFile(String file_name,String access_Privileges)
where access_Privileges may be "r" [Read] or "rw" [Read and Write]

To write data into randomAccessFile,we have to use the following method.
public void writeXXX(xxx value)
where xxx may be byte,short,int,UTF[String],.....

To read data from RandomAccessFile,we have to use the following method

public XXX readXXX()
where xxx may be byte,short,int,UTF[String],.....

To move file pointer to a particular position in RandomAccessFile, we have to use the following method.
public void seek(int position)

EX:

```
1)  import java.io.*;
2)  class Test {
3)        public static void main(String args[]) throws Exception {
4)              RandomAccessFile raf = new RandomAccessFile("abc.txt","rw");
5)              raf.writeInt(111);
6)              raf.writeUTF("Durga");
7)              raf.writeFloat(5000.0f);
8)              raf.writeUTF("HYD");
9)              raf.seek(0);
10)             System.out.println("Employee Number  :"+raf.readInt());
11)             System.out.println("Employee Name  :"+raf.readUTF());
12)             System.out.println("Employee Salary  :"+raf.readFloat());
13)             System.out.println("Employee Address  :"+raf. readUTF ());
14)       }
15) }
```