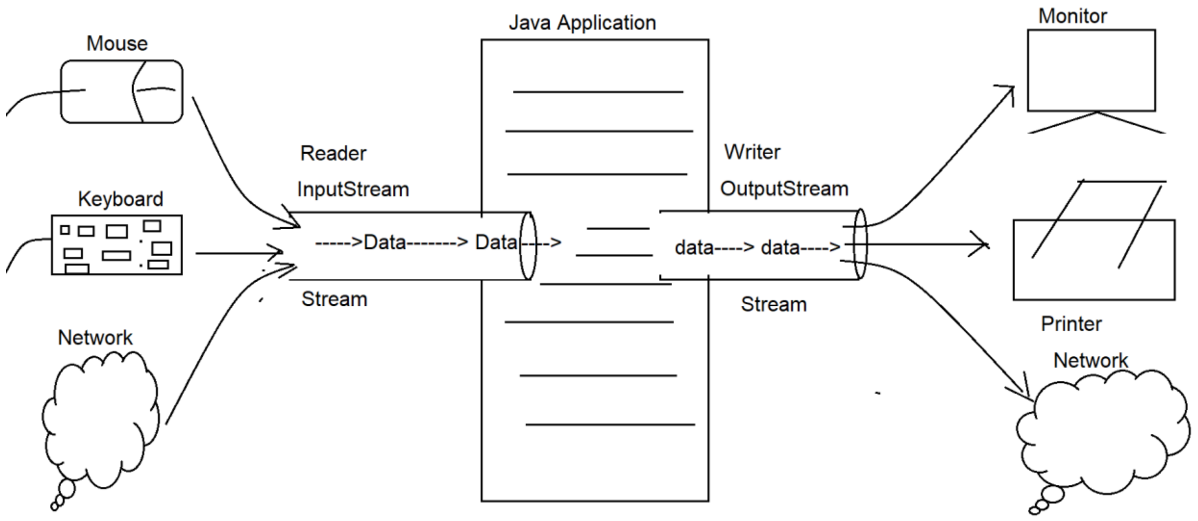
Workshop Link: https://attendee.gotowebinar.com/register/1011365310966029397

IOStreams:

In C and C++ applications, we are able to perform input and Output operations by using the predefined functions like printf() and scanf(), cin<<, cout>>.

In Java applications, to perform input and output operations we have to use "Streams".

Stream: Stream is a channel or medium , it is able to carry data in continuous flow from input devices to the java applications and from java applications to the output devices.



In Java applications, all the streams are provided by JAVA in the form of predefined classes in java.io package.

There are two types of Streams.

1. Byte Oriented Streams
2. Character Oriented Streams

Byte Oriented Streams:
These streams are able to carry data in the form of bytes in continuous flow from input devices to java applications and from java applications to output devices.

```
The length of each and every data item in Byte oriented streams is 1 byte.

There are two types of Byte oriented streams.
1. InputStream
2. OutputStream

InputStream:
It is able to carry data from Input devices to the java applications in the
form of bytes in continuous flow.

EX:   ByteArrayInputStream
      FileInputStream
      BufferedInputStream
      DataInputStream
      ObjectInputStream
      —--
      —---

OutputStream:
It is able to carry data from Java applications to the output devices in the
form of bytes in continuous flow.

EX:   ByteArrayOutputStream
      FileOutputStream
      BufferedOutputStream
      PrintStream
      DataOutputStream
      —---
      —---
Note: IN the case of Byte oriented streams, all the stream classes are ended
with the word Stream.

Character Oriented Streams:
It is able to carry data in the form of characters in continuous flow from
input devices to java applications and from Java applications to the output
devices.

The data length in character oriented streams is 2 bytes.

There are two types of character oriented streams.

1. Reader
2. Writer
```

Reader:
It is able to carry data in the form of characters in continuous flow from input devices to the java applications.

```
EX:   CharArrayReader
      FileReader
      BufferedReader
      InputStreamReader
      —----
      —-----
```

Writer:
It is able to carry data in the form of characters in continuous flow from Java applications to the output devices.

```
EX:   CharArrayWriter
      FileWriter
      PrintWriter
      BufferedWritere
      —---
      —--
```

Note: All the character oriented stream classes are ended either with Reader or with Writer.

File Operations:
—---------------
In Java applications, it is a frequent requirement to send data to a particular target file and to get data from a particular Source file.

To perform the above specified file operations Java has provided the following predefined classes.

1. FileOutputStream
2. FileInputStream
3. FileWriter
4. FileReader

FileOutputStream:
It can be used to send data from Java applications to a particular target file.

If we want to send data to a particular target file by using FileOutputStream then we have to use the following steps.

1. Create FileOutputStream class object:

```
public FileOutputStream(String targetFile)
public FileOutputStream(String targetFile, boolean state)
```

Where state values may be true or false values, where true value will make the FileOutputStream to append present data to the old data existing in the file, Where false value will make the FileOutputStream to override the existing data in the file with the present data.

Note: By Default FileOutputStream will take false value as state, that is override operation.

```
FileOutputStream fos = new FileOutputStream("welcome.txt", true);
```

When we execute the above instruction, JVM will perform the following actions.

   a. JVM will take the name and location of the target file.
   b. JVM will check whether the specified target file exists or not.
   c. If the specified target file does not exist then JVM will create a new file with the same name and JVM will create FileOutputStream.
   d. If the specified file exists then JVM will create FileOutputStream.

2. Declare the data which we want to send to the target file:

```
String data = "Durgasoft";
```

3. Convert the data into byte[]:

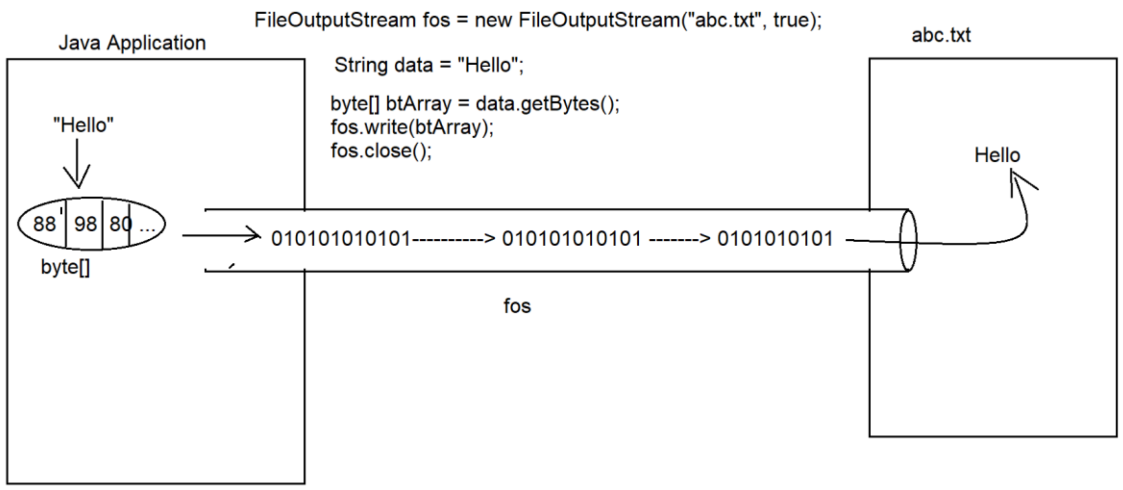
```
byte[] btArray = data.getBytes();
```

4. Write byte[] data into FileOutputStream:

```
public void write(byte[] data)

fos.write(btArray);
```

5. Close FileOutputStream:

```
    fos.close();
```



EX:
```
import java.io.FileOutputStream;

public class Main {
   public static void main(String[] args)throws Exception
{
        FileOutputStream fileOutputStream = new
FileOutputStream("E:/abc/xyz/welcome.txt", true);
        String data = "\nWelcome To Durga Software
Solutions";
        byte[] byteArray = data.getBytes();
        fileOutputStream.write(byteArray);
        System.out.println("Data Send to
E:/abc/xyz/welcome.txt");
        fileOutputStream.close();
    }
}
```

EX:
```
import java.io.FileOutputStream;
import java.io.IOException;

public class Main {
   public static void main(String[] args){

       FileOutputStream fileOutputStream = null;
       try{
           fileOutputStream = new
FileOutputStream("E:/abc/xyz/hello.txt", true);
           String data = "Hello User!";
           byte[] byteArray = data.getBytes();
           fileOutputStream.write(byteArray);
           System.out.println("Data Sent to
E:/abc/xyz/hello.txt");
       }catch(Exception e){
           e.printStackTrace();
       }finally{
           try {
               fileOutputStream.close();
           } catch (IOException e) {
               e.printStackTrace();
           }
       }
   }
}
```

EX:

```
import java.io.FileOutputStream;
import java.io.IOException;

public class Main {
   public static void main(String[] args){


       try(
               FileOutputStream fileOutputStream = new
FileOutputStream("E:/abc/xyz/hello.txt", true);
```

```
                ){

            String data = ", Welcome To Durgasoft.";
            byte[] byteArray = data.getBytes();
            fileOutputStream.write(byteArray);
            System.out.println("Data Sent to
E:/abc/xyz/hello.txt");
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

FileInputStream:

It can be used to get data from a particular source file to a Java application.

To read data from a source file to java applications by using FileInputStream then we will use the following steps.;

1. Create FileInputStream object:
   To create FileInputStream class objects we have to use the following constructor.

   public FileInputStream(String fileName)

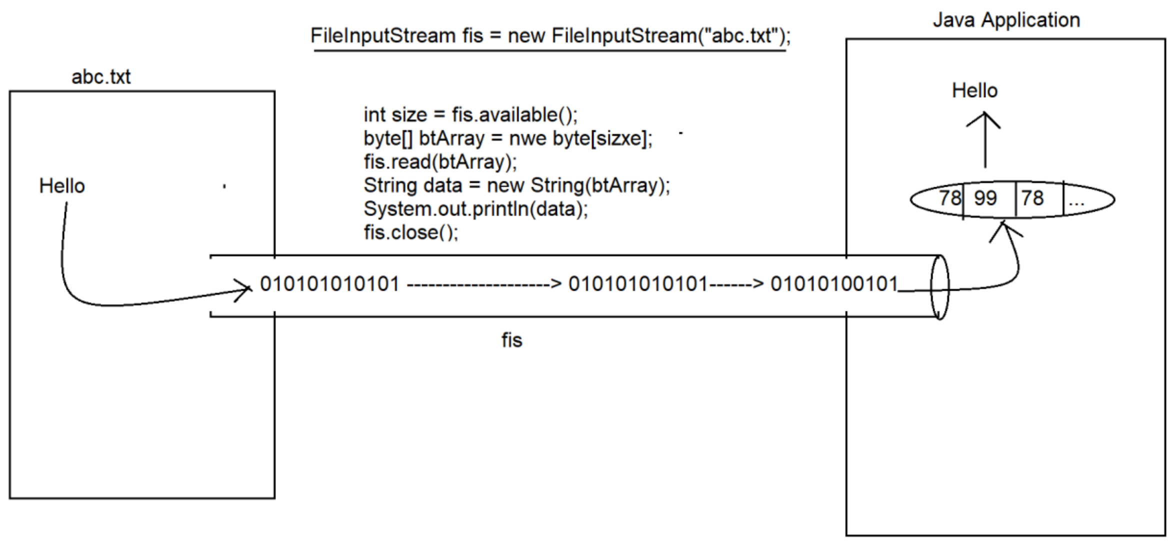
   EX: FileInputStream fis = new FileInputStream("abc.txt");

   If we execute the above instruction, JVM will perform the following actions.

   a. JVM will check whether the provided file exists or not.
   b. If the specified file does not exist then JVM will raise an exception like java.io.FileNotFoundException.
   c. If the specified file exists then JVM will create FileInputStram from the specified file to the java application.
   d. When FileInputStream is created, automatically the which is available in the file will come to FileInputSTream in the form of bytes.

2. Find data size in the FIleInputStream:
   ```
   int size = fis.available();
   ```
3. Create byte[] with the data size:
   ```
   byte[] byteArray = new byte[size];
   ```
4. Read data to byte[]:
   ```
   fis.read(byteArray);
   ```
5. Convert data from byte[] to String.
   ```
   String data = new String(byteArray);
   ```
6. Close the FIleInputStream:
   ```
   fis.close();
   ```

EX:
```
import java.io.FileInputStream;

public class Main {
   public static void main(String[] args){
       try(
               FileInputStream fileInputStream = new 
FileInputStream("E:/abc/xyz/welcome.txt");
               ){
           int size = fileInputStream.available();
           byte[] byteArray = new byte[size];
           fileInputStream.read(byteArray);
           String data = new String(byteArray);
           System.out.println(data);

       }catch(Exception e){
           e.printStackTrace();
       }
   }
}
```


FileInputStream fis = new FileInputStream("abc.txt");
Java Application
abc.txt
Hello
int size = fis.available();
byte[] btArray = nwe byte[sizxe];
fis.read(btArray);
String data = new String(btArray);
System.out.println(data);
fis.close();
Hello
78 99 78 ...
010101010101 -------------------> 010101010101------> 01010100101
fis

EX:

```
import java.io.FileInputStream;

public class Main {
   public static void main(String[] args){// java Main 
Test.java
       String fileName = args[0];
       try(
               FileInputStream fileInputStream = new 
FileInputStream(fileName);
               ){
           byte[] byteArray = new 
byte[fileInputStream.available()];
           fileInputStream.read(byteArray);
           String data = new String(byteArray);
           System.out.println(data);
       }catch(Exception e){
           e.printStackTrace();
       }
   }
}
```

```
PS D:\java6\INTELLIJ-APPS\app36> cd src
PS D:\java6\INTELLIJ-APPS\app36\src> javac Main.java
PS D:\java6\INTELLIJ-APPS\app36\src> java Main D:\java6\Test.java
```

EX:

```
import java.io.FileInputStream;

public class Main {
   public static void main(String[] args){// java Main
Test.java
       String fileName = "E:/abc/xyz/welcome.txt";
       try(
               FileInputStream fileInputStream = new
FileInputStream(fileName);
               ){
           byte[] byteArray = new
byte[fileInputStream.available()];
           fileInputStream.read(byteArray);
           String data = new String(byteArray);
           String[] tokens = data.split(" ");
           System.out.println("No of Words       :
"+tokens.length);
           int count = 0;
           for(String token: tokens){
               if(token.equals("Durga")){
                   count = count + 1;
               }
           }
           System.out.println("'Durga' Repeated   :
"+count);
       }catch(Exception e){
           e.printStackTrace();
       }
   }
}
```

EX:
```
import java.io.FileInputStream;
import java.io.FileOutputStream;

public class Main {
   public static void main(String[] args){
       try(
               FileInputStream fileInputStream = new
FileInputStream("E:/images/baba.jpg");
               FileOutputStream fileOutputStream = new
FileOutputStream("E:/abc/xyz/baba.jpg");
               ){
           byte[] byteArray = new
byte[fileInputStream.available()];
           fileInputStream.read(byteArray);
           fileOutputStream.write(byteArray);
           System.out.println("Image copied from
E:/images/baba.jpg to E:/abc/xyz/baba.jpg");
       }catch(Exception e){
           e.printStackTrace();
       }
   }
}
```

FileReader:

It can be used to read data from a particular source file to a Java application.

To read data from a particular source file to a Java application by using FileReader we have to use the following steps.

1. Create FileReader class object:

    FileReader fr = new FileReader("abc.txt");

    Where JVM will check whether the provided file exists or not, if the provided file does not exist then JVM will raise an exception like java.io.FileNotFoundException, if the specified file exists then JVM will create FileReader from the specified file to Java application with the complete data which exists in the specified file.

2. Read Data from FileReader:
   a. Read char by char in the form of its ASCII value.
   b. Convert char by char from ASCII value to character.
   c. Append char by char to a String variable.

```
String data = “”;
int val = fr.read();
while(val != -1){
      data = data + (char)val;
      Val = fr.read();
}
```

3. Close FileReader:

```
fr.close();
```

EX:

```
import java.io.FileReader;

public class Main {
   public static void main(String[] args){
       try(
               FileReader fileReader = new
FileReader("E:/abc/xyz/welcome.txt");
               ){
           String data = "";
           int val = fileReader.read();
           while(val != -1){
               data = data + (char)val;
               val = fileReader.read();
           }
           System.out.println(data);
       }catch(Exception e){
           e.printStackTrace();
       }
   }
}
```

FileWriter:
It can be used to send data from Java applications to a particular target File.

Steps:
1. Create FileWriter:

```
FileWriter fw = new FileWriter(“abc.txt”, true);
```

JVM will check whether the specified file exists or not, if the specified file does not exist then JVM will create a new file with the same name, if the specified file exists then JVM will create FileWriter object from the java application to the source file.

2. Declare the data and convert that data to char[].

```
String data = “Durgasoft”;
char[] charArray = data.toCharArray();
```

3. Send data to FileWriter:

```
fileWriter.write(charArray);
```

4. Close FileWriter:

```
fileWriter.close();
```

EX:
```
import java.io.FileReader;
import java.io.FileWriter;

public class Main {
   public static void main(String[] args){
       try(
               FileWriter fileWriter = new
FileWriter("E:/abc/xyz/welcome.txt", true);
               ){
           fileWriter.write("Welcome To
Java".toCharArray());
           System.out.println("Data Sent
E:/abc/xyz/welcome.txt");
```

```
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

EX:

```
import java.io.FileReader;
import java.io.FileWriter;

public class Main {
   public static void main(String[] args){
       try(
               FileReader fileReader = new
FileReader("E:/documents/emp.csv");
               FileWriter fileWriter = new
FileWriter("E:/abc/xyz/emp.csv");
               ){
           String data = "";
           int val = fileReader.read();
           while(val != -1){
               data = data + (char)val;
               val = fileReader.read();
           }
           fileWriter.write(data.toCharArray());
           System.out.println("Data copied from
E:/documents/emp.csv to E:/abc/xyz/emp.csv");
       }catch(Exception e){
           e.printStackTrace();
       }
    }
}
```

Note: IN Java applications, byte oriented streams are able to carry both byte oriented data and character oriented data like images and documents,..... , but character oriented streams are able to carry only character oriented data but not byte oriented data.

EX:
```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;

public class Main {
   public static void main(String[] args){
       try(
               FileInputStream fileInputStream = new
FileInputStream("E:/documents/emp.csv");
               FileOutputStream fileOutputStream = new
FileOutputStream("E:/abc/xyz/empNew.csv");
               ){
           byte[] byteArray = new
byte[fileInputStream.available()];
           fileInputStream.read(byteArray);
           fileOutputStream.write(byteArray);
           System.out.println("Data copied from
E:/documents/emp.csv to E:/abc/xyz/empNew.csv");
       }catch(Exception e){
           e.printStackTrace();
       }
   }
}
```

Dynamic Input Approaches:

If we provide input data to the java application at runtime then that input data is called Dynamic Input.

In Java, there are three approaches to provide Dynamic input.

1. BufferedReader
2. Scanner
3. Console

1. BufferedReader:

If we want to read dynamic input from command prompt by using BufferedReader then we have to use the following statement.

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

Where “in” is a static reference variable in System class, it is able to refer to a predefined InputStream object and it is connected with the command prompt.

If we provide Dynamic input on command prompt then the provided data will be sent to the predefined InputStream object which is referred by System.in.

The data which is available in InputStream is in byte oriented data, which is not understandable , so we need to convert the data from bytes representation to character representations. To convert the data from bytes representation to character representation we have to use InputStreamReader.

In InputStreamReader, the data is available in the form of individual characters, to read this type of data we have to provide more read operations, it will reduce application performance.

In the above context, to improve application performance we have to use BufferedReader, In BufferedReader all the individual characters are gathered as a single string in the form of a buffer.

To read data from BufferedReader we have to use the following methods.

1. public String readLine()
2. public int read()

Q)What is the difference between readLine() method and read() method?

Ans:
----

readLine() is able to read a line of data in the form of a String.
read() is able to read a single character in the form of its ASCII value.

EX:

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

```
public class Main {
   public static void main(String[] args){
       try(
               BufferedReader bufferedReader = new
BufferedReader(new InputStreamReader(System.in));
               ){
           System.out.print("Enter Data    : ");
           String data  = bufferedReader.readLine();
           System.out.print("Enter the same Data Again
: ");
           int val = bufferedReader.read();
           System.out.println("readLine()    : "+data);
           System.out.println("read()        :
"+val+"["+(char)val+"]");
       }catch(Exception e){
           e.printStackTrace();
       }
   }
}
```

```
Enter Data    : Durga Software Solutions
Enter the same Data Again    : Durga Software Solutions
readLine()    : Durga Software Solutions
read()        : 68[D]
```

In the case of BufferedReader, if we want to take primitive data as dynamic input then we have to use the following two steps.

1. Read data as String by using readLine() method.
2. Convert the data from String to primitive type by using wrapper classes.

EX:

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

```java
public class Main {
   public static void main(String[] args){
       try(
               BufferedReader bufferedReader = new
BufferedReader(new InputStreamReader(System.in));
               ){
           System.out.print("First Value     : ");
           String val1 = bufferedReader.readLine();
           System.out.print("Second Value    : ");
           String val2 = bufferedReader.readLine();
           System.out.println("ADD   : "+(val1 + val2));
           int fval = Integer.parseInt(val1);
           int sval = Integer.parseInt(val2);
           System.out.println("ADD   : "+(fval + sval));
       }catch(Exception e){
           e.printStackTrace();
       }
   }
}
```

```
First Value     : 10
Second Value    : 20
ADD   : 1020
ADD   : 30
```

2. Scanner:

Scanner is a class defined in the java.util package.
It is able to provide an environment to read primitive data directly without wrapper classes.

Steps:
1. Create Scanner class object:

```java
      Scanner scanner = new Scanner(System.in);
```

```
 2. Read Dynamic Input from Scanner:
   a. To read Primitive data :
      public xxx nextXxx()
      Where xxx may be byte, short, int,...

   b. To read String data :
      public String nextLine()
      public String next()

Q)What is the difference between nextLine() method and next()?
---------------------------------------------------------------
Ans:
----
nextLine() method is able to read the complete line of data as a String.
next() method is able to read a single word in the form of a String.

EX:
```

```java
import java.io.*;
import java.util.Scanner;

public class Main {
   public static void main(String[] args){
       try(
               Scanner scanner = new Scanner(System.in);
               ){
           System.out.print("Enter Data   : ");
           String data1 = scanner.nextLine();
           System.out.print("Enter the same data again
: ");
           String data2 = scanner.next();

           System.out.println("nextLine()    : "+data1);
           System.out.println("next()        : "+data2);
       }catch(Exception e){
           e.printStackTrace();
       }
```

```
    }
}
```

```
Enter Data    : Durga Software Solutions
Enter the same data again  : Durga Software Solutions
nextLine()    : Durga Software Solutions
next()        : Durga
```

EX:

```
import java.io.*;
import java.util.Scanner;

public class Main {
   public static void main(String[] args){
       try(
               Scanner scanner = new Scanner(System.in);
               ){

           System.out.print("Employee Number     : ");
           int eno = scanner.nextInt();
           System.out.print("Employee Name       : ");
           String ename = scanner.next();
           System.out.print("Employee Salary     : ");
           float esal = scanner.nextFloat();
           System.out.print("Employee Address    : ");
           String eaddr = scanner.next();

           System.out.println("Employee Details");
           System.out.println("------------------------
");
           System.out.println("Employee Number   :
"+eno);
           System.out.println("Employee Name     :
"+ename);
```

```
            System.out.println("Employee Salary   : "+esal);
            System.out.println("Employee Address  : "+eaddr);

        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

```
Employee Number     : 111
Employee Name       : Durga
Employee Salary     : 5000
Employee Address    : Hyd
Employee Details
------------------------
Employee Number   : 111
Employee Name     : Durga
Employee Salary   : 5000.0
Employee Address  : Hyd
```

3. Console:

In Java applications, if we want to read dynamic input by using BufferedReader and Scanner then we are able to get the following problems.

1. For every dynamic input we must consume two instructions.
    a. Display Request message by using System.out.println()
    b. Reading Dynamic input by using readLine()/nextXxx()

2. No security for the dynamic input data like password, pin numbers,...

EX:
```
import java.io.*;
import java.util.Scanner;

public class Main {
   public static void main(String[] args){
       try(
               Scanner scanner = new Scanner(System.in);
               ){

           System.out.print("User Name     : ");
           String uname = scanner.next();
           System.out.print("Password      : ");
           String upwd = scanner.next();

           if(uname.equals("durga") &&
upwd.equals("durga")){
               System.out.println("User Login Success");
           }else{
               System.out.println("User Login Failure");
           }
       }catch(Exception e){
           e.printStackTrace();
       }
   }
}
```

```
User Name     : durga
Password      : durga -----> Password data is visible , no security.
User Login Success
```

To overcome the above problems we have to use Console.

Console is a class, it is able to provide a very good environment to read dynamic input with the following advantages.

1. We can display request messages and we can read dynamic input by using a single method.
2. We can read the data like password, pin numbers,... with security.

In Java applications, if we want to use Console then we have to use the following steps.

1. Create COnsole object:

```
      Console console = System.console();
```

2. Read Dynamic Input:
   a. To display request message and to read dynamic input as a string:

```
      public String readLine(String message)
```

   b. To display request messages and to read password data as a char[].

```
      public char[] readPassword(String message)
```

Note: If we want to read primitive data as dynamic input by using Console then we must use wrapper classes.

EX:
```
import java.io.*;
import java.util.Scanner;

public class Main {
    public static void main(String[] args)throws Exception{
        Console console = System.console();
        String uname = console.readLine("User Name    : ");
        char[] pwd = console.readPassword("Password     : ");
        String upwd = new String(pwd);

        if(uname.equals("durga") && upwd.equals("durga")){
            System.out.println("User Login Success");
        }else{
            System.out.println("User Login Failure");
        }

    }
}
```

```
D:\java6>javac Main.java
```

```
D:\java6>java Main
User Name    : durga
Password     :
User Login Success

D:\java6>java Main
User Name    : durga
Password     :
User Login Failure

D:\java6>
```

EX:

```
import java.io.*;
import java.util.Scanner;

public class Main {
    public static void main(String[] args)throws Exception{
        Console console = System.console();

        int eno = Integer.parseInt(console.readLine("Employee Number   : "));
        String ename = console.readLine("Employee Name     : ");
        float esal = Float.parseFloat(console.readLine("Employee Salary   :
"));
        String eaddr = console.readLine("Employee Address    : ");

        System.out.println("Employee Details");
        System.out.println("----------------------");
        System.out.println("Employee Number     : "+eno);
        System.out.println("Employee Name       : "+ename);
        System.out.println("Employee Salary     : "+esal);
        System.out.println("Employee Address    : "+eaddr);

    }
}
```

```
D:\java6>javac Main.java

D:\java6>java Main
Employee Number   : 111
Employee Name     : Durga
Employee Salary   : 5000
Employee Address    : Hyd
Employee Details
-----------------------
Employee Number     : 111
Employee Name       : Durga
Employee Salary     : 5000.0
Employee Address    : Hyd

D:\java6>
```

Serialization And Deserialization:
———————————————————————————————————

If we design and execute any application on the basis of client-server arch or by distributing application logic over multiple machines then that application is called Distributed applications.

In Distributed applications, we will use local machine and Remote Machine, where Remote machine will have remote services that are consumed by the local machine applications.

In general, in distributed applications we will transfer messages from local machine to Remote machine and from Remote machine to local machine.

In Distributed applications , it is the minimum convention to carry an object from local machine to Remote machine and from Remote Machine to Local Machine.

In Distributed applications, to carry an Object from Local machine to Remote Machine we have to perform the following actions.

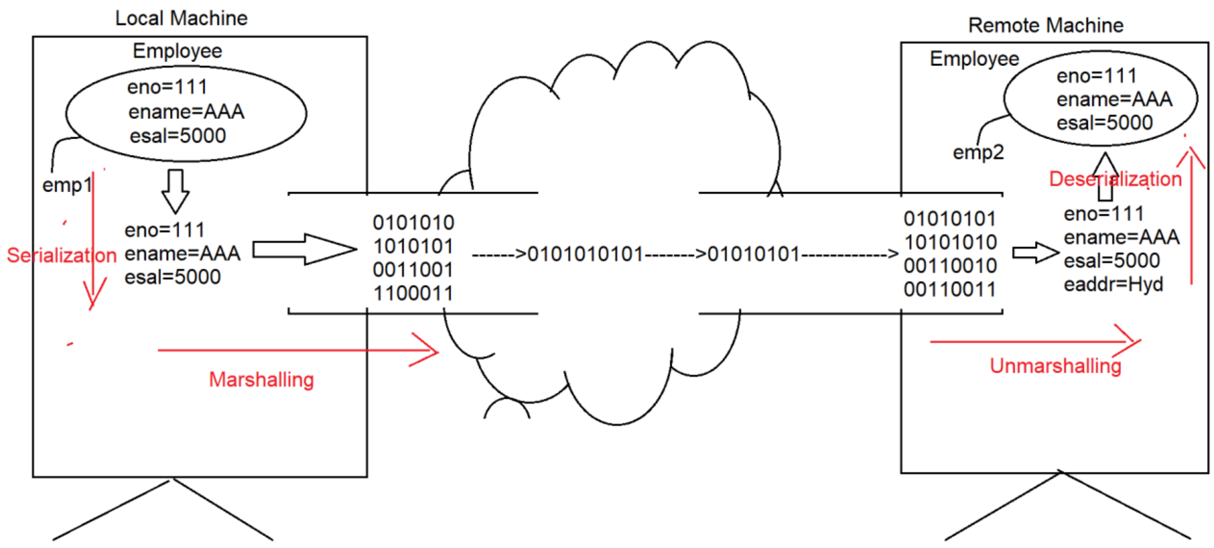
1. Create an Object which we want to send to a Remote Machine.
2. Separate data from the Object.
3. Converting data from System representation to network representations.
4. Send data to the network

In Distributed applications, when we send a n object to the Remote machine , at the Remote machine we have to perform the following actions.

1. Converting data from Network representation to System representations.
2. Re-constructing an Object on the basis of the data.

In the above Context,

1. The process of separating data from an object is called “Serialization”.
2. The process of converting data from System representation to the network representation is called “Marshalling”.
3. The process of re-constructing an object on the basis of the data is called “Deserialization”.
4. The process of converting data from network representation to system representation is called “Unmarshalling”.



In Java applications, to perform Serialization and Deserialization JAVA has provided the following two predefined byte oriented streams.

1. ObjectOutputStream for Serialization
2. ObjectInputStream for Deserialization

If we want to perform Serialization and Deserialization in Standalone Applications then we have to use a file to store Serialized data.

To perform Serialization in java applications we have to perform the following actions.

1. Prepare a Serializable class and its Object:
In Java applications, by default all the objects are not eligible for Serialization and Deserialization, only the objects whose classes are implementing java.io.Serializable marker interface are eligible for Serialization and Deserialization.

```
public class Employee implements Serializable{
}

Employee emp = new Employee();
```

2. Prepare FileOutputStream to make ready target file:

```
      FileOutputSTream fos = new FileOutputStream(“abc.txt”);
```

3. Create ObjectOutputStream with FileOutputSteam:

In Java applications, developers are not performing the real serialization , whereas in java applications ObjectOutputStream is performing the real Serialization, where in the Java applications the role of the developer is to supply the Serializable object to the ObjectOutputStream.

```
      ObjectOutputStream oos = new ObjectOutputStream(fos);
```

4. Write the Serializable Object to ObjectOutputStream:

```
      public void writeObject(Object obj)throws NotSerializableException
      EX: oos.writeObject(emp);
```

Note: When we send the Serializable object to ObjectOutputStream, ObjectOutputStream will perform the Serialization over the Serializable object, ObjectOutputStream will send the serializable object data to the FileOutputSTream, where FileOutputStream will send the serialized object data to the respective file.

To perform Deserialization in Java applications we have to use the following steps.

1. Create FileInputStream from the file where the Serialized data exist:

```
FileinputStream fis = new FileInputStream(“abc.txt”);
```

When we create a FileInputStream object , automatically the complete Serialized data will come to the FIleInputStream.

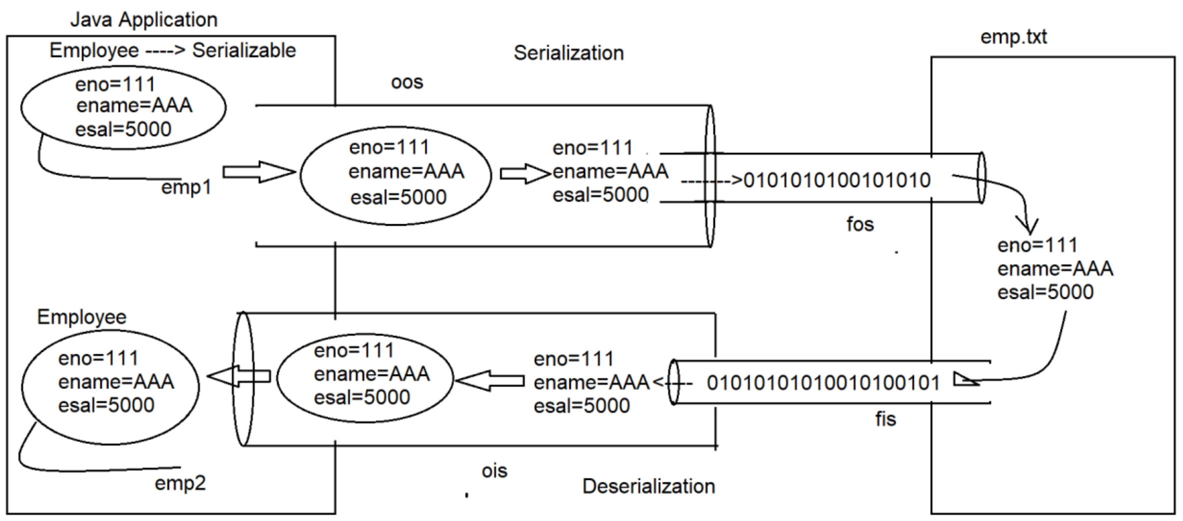
2. Create ObjectInputStream to perform Deserialization:

```
ObjectInputStream ois = new ObjectInputStream(fis);
```

When we create ObjectInputStream, ObjectInputStream will take serialized data from FileinputStream , ObjectinputStream will perform the required Deserialization over the serialized data and it will generate Deserialized Objects .

3. Read the Deserialize Object from the ObjectInputSTream:

```
public Object readObject()throws IOException

Object obj = ois.readObject();
Employee emp = (Employee)obj;
```



EX:
Employee.java
```
package com.durgasoft.entities;

import java.io.Serializable;

public class Employee implements Serializable {
   private int eno;
   private String ename;
   private float esal;
   private String eaddr;

   public Employee(int eno, String ename, float esal, String eaddr) {
       this.eno = eno;
       this.ename = ename;
       this.esal = esal;
       this.eaddr = eaddr;
   }

   public void getEmployeeDetails(){
       System.out.println("Employee Details");
       System.out.println("-----------------------");
       System.out.println("Employee Number      : "+eno);
       System.out.println("Employee Name        : "+ename);
       System.out.println("Employee Salary      : "+esal);
       System.out.println("Employee Address     : "+eaddr);
   }
}
```

Main.java
```
import com.durgasoft.entities.Employee;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class Main {
   public static void main(String[] args)throws Exception {
       FileOutputStream fileOutputStream = new
FileOutputStream("E:/abc/xyz/emp.txt");
       ObjectOutputStream objectOutputStream = new
ObjectOutputStream(fileOutputStream);
       Employee employee1 = new Employee(111, "Durga", 5000, "Hyd");
       System.out.println("Employee Details before Serialization");
       employee1.getEmployeeDetails();
```

```
        objectOutputStream.writeObject(employee1);
        System.out.println();

        FileInputStream fileInputStream = new FileInputStream("E:/abc/xyz/emp.txt");
        ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);
        Object obj = objectInputStream.readObject();
        Employee employee2 = (Employee) obj;

        System.out.println("Employee Details After the Deserialization");
        employee2.getEmployeeDetails();

        fileOutputStream.close();
        fileInputStream.close();
        objectOutputStream.close();
        objectInputStream.close();

    }
}
```

```
Employee Details before Serialization
Employee Details
-----------------------
Employee Number      : 111
Employee Name        : Durga
Employee Salary      : 5000.0
Employee Address     : Hyd

Employee Details After the Deserialization
Employee Details
-----------------------
Employee Number      : 111
Employee Name        : Durga
Employee Salary      : 5000.0
Employee Address     : Hyd
```

In Java applications, if we perform Serialization over an object without implementing java.io.Serializable interface then JVM will raise an exception like java.io.NotSerializableException .

EX:
Employee.java

```
package com.durgasoft.entities;

import java.io.Serializable;
```

```java
public class Employee  {
   private int eno;
   private String ename;
   private float esal;
   private String eaddr;

   public Employee(int eno, String ename, float esal, String eaddr) {
       this.eno = eno;
       this.ename = ename;
       this.esal = esal;
       this.eaddr = eaddr;
   }

   public void getEmployeeDetails(){
       System.out.println("Employee Details");
       System.out.println("-----------------------");
       System.out.println("Employee Number      : "+eno);
       System.out.println("Employee Name        : "+ename);
       System.out.println("Employee Salary      : "+esal);
       System.out.println("Employee Address     : "+eaddr);
   }
}
```

Main.java

```java
import com.durgasoft.entities.Employee;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class Main {
   public static void main(String[] args)throws Exception {
       FileOutputStream fileOutputStream = new
FileOutputStream("E:/abc/xyz/emp.txt");
       ObjectOutputStream objectOutputStream = new
ObjectOutputStream(fileOutputStream);
       Employee employee1 = new Employee(111, "Durga", 5000, "Hyd");
       System.out.println("Employee Details before Serialization");
       employee1.getEmployeeDetails();
```

```
        objectOutputStream.writeObject(employee1);
        System.out.println();

        FileInputStream fileInputStream = new
FileInputStream("E:/abc/xyz/emp.txt");
        ObjectInputStream objectInputStream = new
ObjectInputStream(fileInputStream);
        Object obj = objectInputStream.readObject();
        Employee employee2 = (Employee) obj;

        System.out.println("Employee Details After the Deserialization");
        employee2.getEmployeeDetails();

        fileOutputStream.close();
        fileInputStream.close();
        objectOutputStream.close();
        objectInputStream.close();

    }
}
```

In Java applications, Object Serialization allows only instance variables of the respective class, not static variables , because static variables data will not be stored in the Objects whereas instance variables data will be stored inside the objects.

EX:

```
import com.durgasoft.entities.Employee;

import java.io.*;

class User implements Serializable {
    int userId = 111;
    String userName = "Durga";
    static int MIN_AGE = 18;
```

```
   static int MAX_AGE = 25;
}
public class Main {
   public static void main(String[] args)throws
Exception {
       FileOutputStream fileOutputStream = new
FileOutputStream("E:/abc/xyz/user.txt");
       ObjectOutputStream objectOutputStream = new
ObjectOutputStream(fileOutputStream);
       User user = new User();
       objectOutputStream.writeObject(user);
       fileOutputStream.close();
       objectOutputStream.close();


   }
}
```

In java applications, if we have any data like password data, pin numbers,... in an object and if we perform Serialization over that object then the protected data will come to the file which we have used to store serialized data , here the file which we have used to store Serialized data is a open file and it will be accessed by all the users of the respective system, so Serialization and Deserialization are not providing security for the data.

In the above context, if we want to stop a property and its data in the Object Serialization and Deserialization explicitly then we have to declare that variable as a “transient” variable.

EX:

```
import com.durgasoft.entities.Employee;

import java.io.*;

class User implements Serializable {
    int userId = 111;
    String userName = "Durga";
    transient String userPassword = "durga123";
    static int MIN_AGE = 18;
    static int MAX_AGE = 25;
}
public class Main {
    public static void main(String[] args)throws Exception
{
        FileOutputStream fileOutputStream = new
FileOutputStream("E:/abc/xyz/user.txt");
        ObjectOutputStream objectOutputStream = new
ObjectOutputStream(fileOutputStream);
        User user = new User();
        objectOutputStream.writeObject(user);
        fileOutputStream.close();
        objectOutputStream.close();



    }
}
```

In Java applications, if we implement java.io.Serializable interface at super class and if we perform Serialization over Sub class objects then JVM will not raise any exception, where all superclass members and subclass members are participating in the Serialization.

EX:

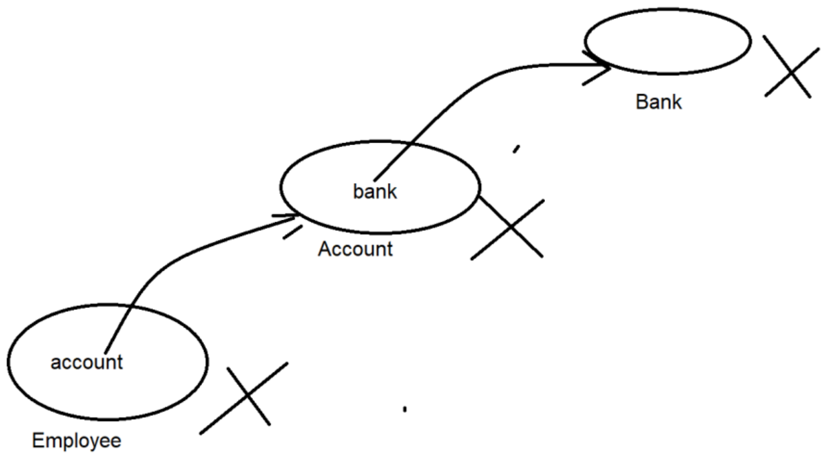
```
import java.io.*;

class Person implements Serializable {
   int personId = 111;
   String personName = "Durga";

}
class Employee extends Person{
   int eno = 12345;
   String equal = "MTech";
}
public class Main {
   public static void main(String[] args)throws
Exception {
       FileOutputStream fileOutputStream = new
FileOutputStream("E:/abc/xyz/employee.txt");
       ObjectOutputStream objectOutputStream = new
ObjectOutputStream(fileOutputStream);
       Employee emp = new Employee();
       objectOutputStream.writeObject(emp);
       fileOutputStream.close();
       objectOutputStream.close();
   }
}
```

In Java applications, when we perform Serialization over a Container object if any contained object is identified then JVM is trying to perform Serialization over Contained object also along with Container Object Serialization ,in this case, if the COntained object class is not implementing Serializable interface then JVM will raise an exception like java.io.NotSerializableException.

```
import java.io.*;
class Employee implements Serializable{
   int eno = 111;
   Account account = new Account();
}
class Account implements Serializable{
   String accNo = "abc123";
   Bank bank = new Bank();
}
class Bank implements Serializable{
   int bankId = 123;
   String bankName = "ICICI Bank";
}
public class Main {
   public static void main(String[] args)throws
Exception {
       FileOutputStream fileOutputStream = new
FileOutputStream("E:/abc/xyz/employee.txt");
```

```
        ObjectOutputStream objectOutputStream = new
ObjectOutputStream(fileOutputStream);
        Employee employee = new Employee();
        objectOutputStream.writeObject(employee);
        fileOutputStream.close();
        objectOutputStream.close();


    }
}
```

In the above associations, if we prepare a graph representation to represent Serialization over the objects then that Graph is called “Object Graph”.



Externalization:

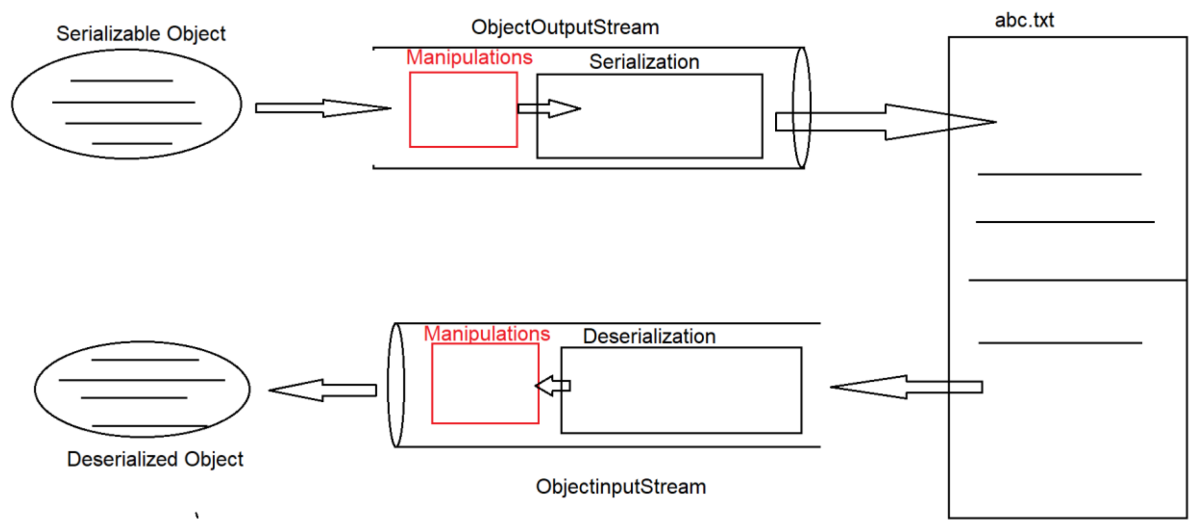
—----------------

In Java applications, to perform Serialization java has provided ObjectOutputStream, where developers have to provide Serializable object to the ObjectOutputStream , where ObjectOutputStream is able to perform Serialization over the object internally, where developers are not having any control over the Serialization.

In Java applications, to perform Deserialization Java has provided ObjectInputStream, where Developers have to read deserialized object from ObjectInputStream , here ObjectInputStream is performing the required Deserialization , where Developers are not having control over the deserialization process.

As per the requirement, we want to perform manipulations over the data just before performing Serialization and we want to perform manipulations over the data immediately after performing Deserialization.

In the above context, to perform manipulations over the data just before performing Serialization and immediately after performing Deserialization we have to use “Externalization”.



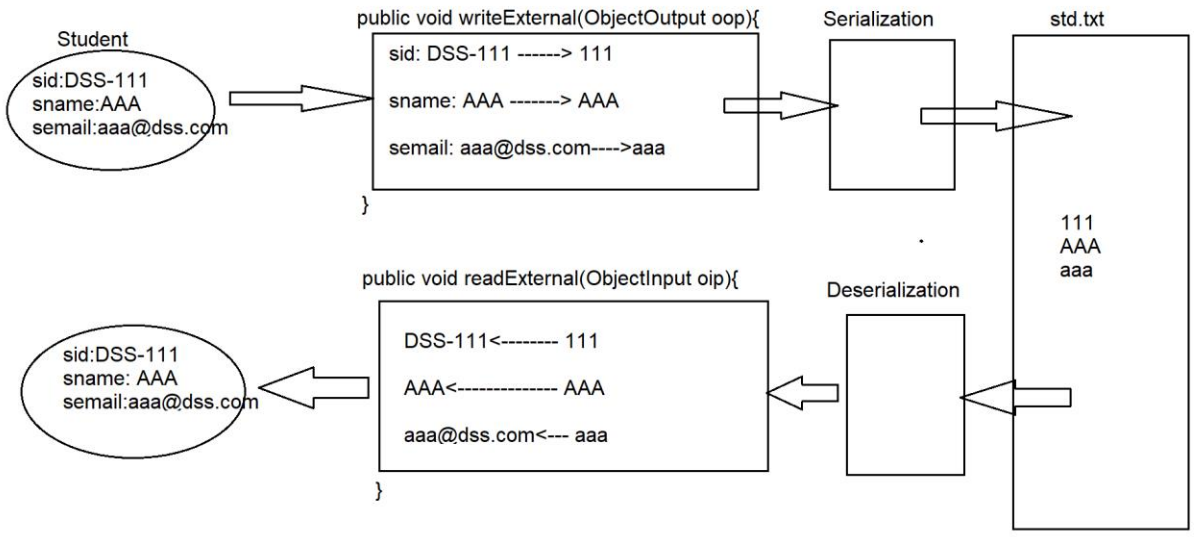
To perform Externalization we have to use the following steps.

```
1.  Create an Externalizable class and its object.

2.  Perform Serialization and Deserialization over the Externalizable
    Object.

Creating Externalizable class and its Object:
----------------------------------------------
a. Declare an user defined class.
b. Implement java.io.Externalizable interface in the user defined class.
c. Implement Externalizable interface methods in the user defined class.

   public void writeExternal(ObjectOutput oop)throws IOException
   i. This method will be executed just before performing Serialization.
   ii.This method will include the logic which we want to apply
      manipulations over the data just before Serialization.
   iii.After the data manipulations we must send the manipulated data to
       the Serialization  , fir this we have to use the following methods
       from ObjectOutput.

       public void writeXxx(xxx value)
      Where xxx may be byte, short, int, UTF[String]

   public void readExternal(ObjectInput oip)throws IOException,
   ClassNotFoundException
    i. This method will be executed immediately after performing
       Deserialization.
    ii.To perform manipulation over the Deserialized data first we have to
       Get Deserialized data , for this we have to use the following
       methods from ObjectInput.

       public xxx readXxx()
      Where xxx may be byte, short, int, UTF[String]

    iii. After the manipulations over the deserialized data we must send
         that data to the java applications.
d. Create a 0-arg constructor in the User defined class.



Perform Serialization and Deserialization over the Externalizable
```

Object:

Same steps as what we have used in Serialization and Deserialization.

EX:



EX:

```
import java.io.*;

class Student implements Externalizable{
   private String sid;
   private String sname;
   private String semailId;
   private String smobileNo;

   public Student(String sid, String sname, String semailId, String
smobileNo) {
        this.sid = sid;
        this.sname = sname;
        this.semailId = semailId;
        this.smobileNo = smobileNo;
   }

   public Student() {
   }

   @Override
   public void writeExternal(ObjectOutput out) throws IOException {
```

```java
        try{
            // sid: DSS-111 -----> 111
            String[] str1 = sid.split("-");
            int stdRollNo = Integer.parseInt(str1[1]);

            // semailId: aaa@durgasoft.com
            String[] str2 = semailId.split("@");
            String emailId = str2[0];

            // smobileNo: 91-9988776655 -----> 9988776655
            String[] str3 = smobileNo.split("-");
            long mobileNo = Long.parseLong(str3[1]);

            out.writeInt(stdRollNo);
            out.writeUTF(sname);
            out.writeUTF(emailId);
            out.writeLong(mobileNo);

        }catch(Exception e){
            e.printStackTrace();
        }
    }

    @Override
    public void readExternal(ObjectInput in) throws IOException,
ClassNotFoundException {
        sid = "DSS-"+in.readInt();
        sname = in.readUTF();
        semailId = in.readUTF()+"@durgasoft.com";
        smobileNo = "91-"+in.readLong();
    }

    public void getStudentDetails(){
        System.out.println("Student Details");
        System.out.println("----------------------");
        System.out.println("Student Id            : "+sid);
        System.out.println("Student Name          : "+sname);
        System.out.println("Student Email Id      : "+semailId);
        System.out.println("Student Mobile Number : "+smobileNo);
    }
}
public class Main {
    public static void main(String[] args)throws Exception {
        FileOutputStream fileOutputStream = new
 FileOutputStream("E:/abc/xyz/student.txt");
        ObjectOutputStream objectOutputStream = new
 ObjectOutputStream(fileOutputStream);
        Student student1 = new Student("DSS-111", "Durga",
 "durga@durgasoft.com", "91-9988776655");
        System.out.println("Student Details before Serialization");
        student1.getStudentDetails();
        objectOutputStream.writeObject(student1);
        System.out.println();

        FileInputStream fileInputStream = new
 FileInputStream("E:/abc/xyz/student.txt");
        ObjectInputStream objectInputStream = new
 ObjectInputStream(fileInputStream);
        Student student2 = (Student) objectInputStream.readObject();
        System.out.println("Student Details After Deserialization");
        student2.getStudentDetails();

        fileOutputStream.close();
        fileOutputStream.close();
        objectOutputStream.close();
        objectInputStream.close();
    }
 }
```

File System In Java:

File is a storage Area to store data.

There are two types of Files in Java.

1. Sequential Files
2. Random Access Files

Sequential Files: These files are able to read data in sequential manner.

To represent Sequential files in Java applications, JAVA has provided a predefined class in the form of java.io.File .

To create a File class object we have to use the following constructor.

```
      public File(String fileName)

      EX: File file = new File(“E:/abc/xyz/std.txt”);
```

If we execute the above instruction, JVM will create the File class object at heap memory only, it will not create a file really.

In the above context, if we want to create a file really at the specified location then we have to use the following method.

```
      public File createNewFile()
```

To check whether a file is created or not we have to use the following method.

```
      public boolean isFile()
```

To get File name:
`public String getName()`

To get File Parent :
`public String getParent()`

To get absolute path:
`public String getAbsolutePath()`

EX:

```
import java.io.*;


public class Main {
   public static void main(String[] args)throws Exception {
        File file = new File("E:/abc/xyz/welcome.txt");
        System.out.println(file.isFile());
        file.createNewFile();
        System.out.println(file.isFile());

        System.out.println(file.getName());
        System.out.println(file.getParent());
        System.out.println(file.getAbsolutePath());

        FileOutputStream fileOutputStream = new FileOutputStream(file);
        fileOutputStream.write("Welcome To Durgasoft".getBytes());
        FileInputStream fileInputStream = new FileInputStream(file);
        byte[] btArray = new byte[fileInputStream.available()];
        fileInputStream.read(btArray);
        System.out.println(new String(btArray));


    }
}
```

To create a directory or folder at a particular location we have to create a File class object with the folder name and location.

```
File file = new File("D:/abc/xyz/employee");
```

If we execute the above instruction, only the File class object is created in Heap memory , it will not create the directory at the specified location.

To create a folder really in the specified location we have to use the following method from File class.

```
public File mkdir()
```

EX:

```
File file = new File("D:/abc/xyz/employee");
file.mkdir();
```

The above method is able to create only the deepest folder, that is parent folder structure must exist, but if we want to create a Parent folder structure and the exact folder then we have to use the following method.

```
public boolean mkdirs()
```

To check whether the directory is created or not we have to use the following method.

```
public boolean isDirectory()
```

To get the details of the directory we have to use the following methods.

```
public String getName()
public String getParent()
public String getAbsolutePath()
```

EX:

```
import java.io.*;
public class Main {
   public static void main(String[] args)throws Exception {
        File file = new File("E:/abc/xyz/student");
        System.out.println(file.isDirectory());
        file.mkdir();
        System.out.println(file.isDirectory());

        System.out.println(file.getName());
        System.out.println(file.getParent());
        System.out.println(file.getAbsolutePath());

   }
}
```

Random Access Files:

These Files are able to read data from random positions.

To represent RandomAccess Files , JAVA has provided a predefined class in the form of java.io.RandomAccessFile.

To create an Object for the RandomAccessFile class we have to use the following constructor.

Public RandomAccessFile(String fileName, String accessPrivileges)

Where access Privileges may be a “r” [Read Only] “rw” [Read And Write]

EX:

RandomAccessFile raf = new RandomAccessFile(“E:/abc/xyz/welcome.txt”, “r”);

To write Data into the Random Access FIle we have to use the following method.
public void writeXXX(xxx value)
Where xxx may be byte, short, int , long,...... UTF[String],....

To read data from the RandomAccessFIle we have to use the following method
public XXX readXXX()
Where XXX may be byte, short, int, …… UTF[String],.....

If we create a file in Java , automatically a pointer will be created to read and to write data in the file .

To move a File pointer to a particular location we have to use the following method.

public void seek(int position)

EX:
```
import java.io.*;
public class Main {
   public static void main(String[] args)throws Exception {
       RandomAccessFile randomAccessFile = new
RandomAccessFile("E:/abc/xyz/welcome.txt", "rw");
       randomAccessFile.writeInt(111);
       randomAccessFile.writeUTF("Durga");
       randomAccessFile.writeFloat(50000.0f);
       randomAccessFile.writeUTF("Hyderabad");
       randomAccessFile.seek(0);

       System.out.println("Employee Details");
       System.out.println("----------------------");
       System.out.println("Employee Number    : "+randomAccessFile.readInt());
       System.out.println("Employee Name      : "+randomAccessFile.readUTF());
       System.out.println("Employee Salary    : "+randomAccessFile.readFloat());
       System.out.println("Employee Address   : "+randomAccessFile.readUTF());


   }
}
```

EX:
```
import java.io.*;
public class Main {
   public static void main(String[] args)throws Exception {
       RandomAccessFile randomAccessFile = new
RandomAccessFile("E:/abc/xyz/welcome.txt", "r");
```

```
        randomAccessFile.writeInt(111);
        randomAccessFile.writeUTF("Durga");
        randomAccessFile.writeFloat(50000.0f);
        randomAccessFile.writeUTF("Hyderabad");
        randomAccessFile.seek(0);

        System.out.println("Employee Details");
        System.out.println("-----------------------");
        System.out.println("Employee Number    :
"+randomAccessFile.readInt());
        System.out.println("Employee Name      :
"+randomAccessFile.readUTF());
        System.out.println("Employee Salary    :
"+randomAccessFile.readFloat());
        System.out.println("Employee Address   :
"+randomAccessFile.readUTF());


   }
}

Exception in thread "main" java.io.IOException: Access is denied
      at java.io.RandomAccessFile.write0(Native Method)
      at java.io.RandomAccessFile.write(RandomAccessFile.java:489)
      at java.io.RandomAccessFile.writeInt(RandomAccessFile.java:1030)
      at Main.main(Main.java:5)
```
—--