# STREAM BASED I/O (java.io)

Stream classes – Byte Stream and Character streams

Stream: Flow of information (data) from source to destination. A stream is not active on its own. Some other must push data into stream or pull data from streams.

Java provides two types of streams: 1. Byte Stream
2. Character Stream.

1. Byte Stream:

Java byte streams are used to perform input and output of 1 byte (8 bits). Though there are many classes related to byte streams, most frequently used classes are FileInputStream and FileOutputStream.

Super classes for byte stream are: i) Input Stream.
ii) Output Stream.

i) InputStream – InputStream is used to read data from source.

ii) OutputStream – OutputStream is used for writing data to a destination.

Input Streams contain following classes:-

1. FileInputStream
2. FilterInputStream
3. ByteArrayInputStream

4. SequentialInputStream
5. StringBufferInputStream

FilterInputStream has following classes:-

1. DataInputStream
2. BufferInputStream
3. LineNumberInputStream
4. PushbackInputStream

OutputStream contain following classes:

1. FileOutputStream
2. FilterOutputStream
3. ByteArrayOutputStream
4. StringBufferOutputStream.

FileOutputStream has following classes:

1. DataOutputStream.
2. BufferOutputStream
3. PrintStreamOutputStream.

Character Streams:

Java CharacterStreams are used to perform input and output for 16-bit (2 bytes) unicode. Though there are many classes related to character streams, most frequently used classes are FileReader and FileWriter

Super classes for CharacterStream are i) Reader
ii) Writer

i) Reader: It is used to read the data.

ii) Writer: It is used to write the data.

Reader has following classes:-

1. FileReader
2. BufferedReader
3. DataReader.
4. LineNumberReader.

Writer has following classes:

1. FileWriter
2. BuffeeedWriter
3. DataWriter
4. LineNumberWriter

## Reading Console Input and Writing Console Output:

Java Console class is be used to get input from console. It provides methods to read texts and passwords.

If you read password using Console class, it will not be displayed to user.

The java.io.Console class is attached with system console internally.

Declaration:

```
public final class Console extends Object
                    implements Flushable.
```

Program to demonstrate Console class.

```
import java.io.*;
class Console
{
public static void main(String[] args)
{
Console c = System.Console();
System.out.println("Enter your name");
String s = c.readLine();
System.out.println("Your name is --" +s);
}
}
```

File class.

The File class is an abstract representation of file and directory pathname. A pathname can be either absolute or relative.

File class have several methods for working with directories and files such as creating new directories or files, deleting and renaming directories or files, listing the contents of a directory etc...

Reading and writing files:-

→ To read an ordinary text file in our system's default encoding, use FileReader and Wrap it in a BufferedReader.

→ To write a text file in java, use FileWriter instead of ~~BufferedOutputReader~~ FileReader and BufferedOutputWriter.

Write a java program to list all files in a directory including the files present in all Sub-directories:-

```
import java.io.*;
class RecursiveFileAndFolder
{
void listFolder(File dir)
{
File[] SubDirs = dir.listFiles(new FileFilter()
{
public boolean accept(File pathname)
{
return pathname.isDirectory();
}
});}
System.out.println("\nDirectory of" + dir.getAbsolutePath());
listFile(dir);
for (File Reader : SubDirs)
{
listFolder(folder);
}
}
void listFile(File dir)
{
File[] files = dir.listFiles();
for(File file : files)
{
System.out.println(file.getName());
}}
public static void main(String[] args)
{
new RecursiveFileAndFolder().listFolder(new File("./"));
}
}
```

## Random Access File Operations:

The java.io.RandomAccessFile class file behaves like a large array of bytes stored in the file system. Instances of this class support both reading and writing to a random access file.

**Declaration:**

public class RandomAccessFile extends Object
implements DataOutput, DataInput, Closeable

**Methods:-**

1. void close()
This method closes this random access file stream and releases any system resources associated with stream.
2. long length()
returns the length of the file.
3. int read()
reads a byte of data from file.
4. char readChar()
reads a character from file
5. double readDouble()
reads a double from file

Console class:-

→ If a console is available, then a reference to it is returned. Thus if null is returned, no console I/O is possible.

→ It provides methods to read texts & passwords.

→ Few of important methods in Console class are:- writer, reader, format, printf, readLine, readPassword, flush etc..

→ Console is primarily a convenience class because most of its functionality is available through System.in and System.out.

→ However, its use can simplify some types of console interactions, especially when reading Strings from Console.

→ Console supplies no constructors.

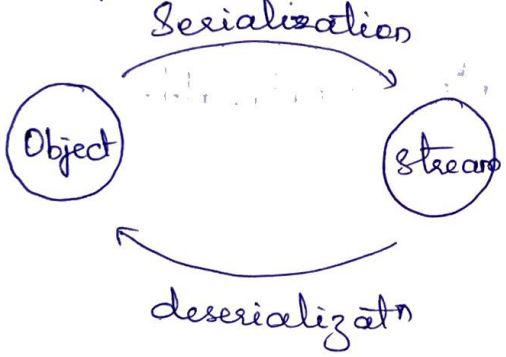
Instead, a Console object is obtained by calling System.Console().

# Serialization:

Serialization in java is a mechanism of writing the state of an object into a byte stream.

→ The reverse operation of serialization is deserialization

→ It is mainly used to travel object's state on the network




Program to demonstrate serialization.

```
import java.io.*;
class Persist
{
public static void main(String[] args) throws
                                    IOException
{
Student s1 = new Student (211, "Ravi");
FileOutputStream fout = new FileOutputStream
                                ("f.txt");
```

```
ObjectOutputStream out = new ObjectOutputStream(fout);
out.writeObject(s1);
out.flush();
System.out.println("Success");
}
}
class Student implements Serializable
{
int id;
String name;
public Student(int id, String name)
{
this.id = id;
this.name = name;
}
}
```

# Enumeration in java:-

→ The enumeration designated by keyword "enum" in java represents special type of class.

→ It basically represents a list of named constants.

→ Apart from java, almost all other prominent programming languages have feature of Enumeration.

→ Although, Java has the 'final' keyword to represent constants, enumeration was included as a convenience to meet many of streamlined needs of programmers.

→ Enum improves type safety.

→ enum can have fields, constructors and methods.

→ enum can be traversed.

→ enum can be easily used in switch.

→ enum may implement many interfaces but cannot extend any class because it internally extends @ Enum class.

## Autoboxing:-

→ Automatic conversion of primitive data type into equivalent wrapper type is known as autoboxing

→ It's opposite operation is known as Unboxing.

→ Less coding is required by autoboxing.

Program to demonstrate concept of Boxing:-

```
class BoxingDemo
{
public static void main (String[] args)
{
int p=20;
Integer c = new Integer (p);
Integer d = 5;
int e= d;
System.out.println (" primitive data type=" +p);
System.out.println ("Boxing=" +c);
System.out.println ("AutoBoxing=" +d);
System.out.println ("Unboxing=" +e);
}
}
```

```
20
20
5
5
```

Generics:-

Generic is a mechanism for creating a general model in go which generic methods and generic classes enables programmer to specify a single method and single class for performing a desired task.

Program to demonstrate Generic method.

```
import java.io.*;
public class GenericMethod
{
public static <T> void display(T[] a)
{
int i;
for(i=0; i<5; i++)
{
System.out.println(a[i]);
}
}
public static void main(String[] args)
{
Integer[] i = {10, 11, 12, 13, 14};
Float[] f = {10.5f, 11.5f, 6.5f};
```

```
Character[] c = {'A','B','C','D','E'};
Double[] d = {20.5, 30.5, 67.7};
System.out.println("integer list");
GenericMethod.display(i);
System.out.println("Float list");
GenericMethod.display(f);
System.out.println("Character list");
GenericMethod.display(c);
System.out.println("Double list");
GenericMethod.display(d);
}
}
```

Program to demonstrate Generic Class;

```
class MyGen<T>
{
T ob;
void add(T ob)
{
this.ob=ob;
}
T get()
{
return ob;
}
}
class GenericClass
{
public static void main(String[] args)
{
MyGen<Integer> m = new MyGen<Integer>();
m.add(20);
System.out.println(m.get());
}
}
```