IDHAYA COLLEGE FOR WOMEN
KUMBAKONAM – 612 001

DEPARTMENT OF INFORMATION TECHNOLOGY

SEMESTER	**:**	**IV**
CLASS	**:**	**II IT**
SUBJECT- INCHARGE	**:**	**G.ELAKKIYA**
SUBJECT NAME	**:**	**PROGRAMMING IN JAVA**
SUBJECT CODE	**:**	**16SCCIT 4**
TOPIC	**:**	**UNIT 5**

Java I/O

Java I/O (Input and Output) is used *to process the input* and *produce the output*.

Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.

We can perform **file handling in Java** by Java I/O API.

Stream

A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

In Java, 3 streams are created for us automatically. All these streams are attached with the console.

1) System.out: standard output stream

2) System.in: standard input stream

3) System.err: standard error stream

Let's see the code to print **output and an error** message to the console.

```
1. System.out.println("simple message");
2. System.err.println("error message");
```

Let's see the code to get **input** from console.

```
1. int i=System.in.read();//returns ASCII code of 1st character
2. System.out.println((char)i);//will print the character
```

OutputStreamvsInputStream

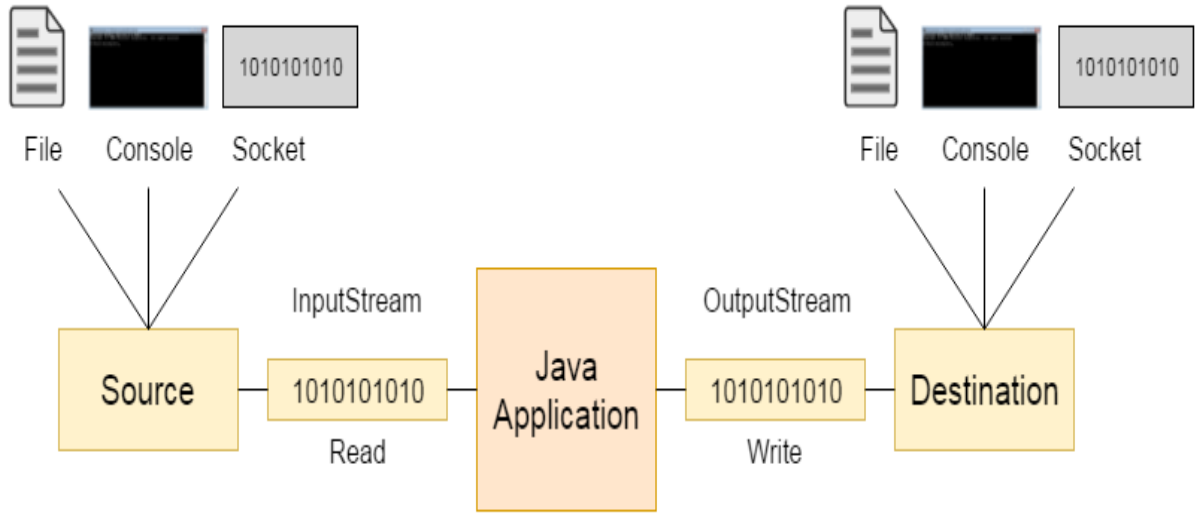
The explanation of OutputStream and InputStream classes are given below:

OutputStream

Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.

InputStream

Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.



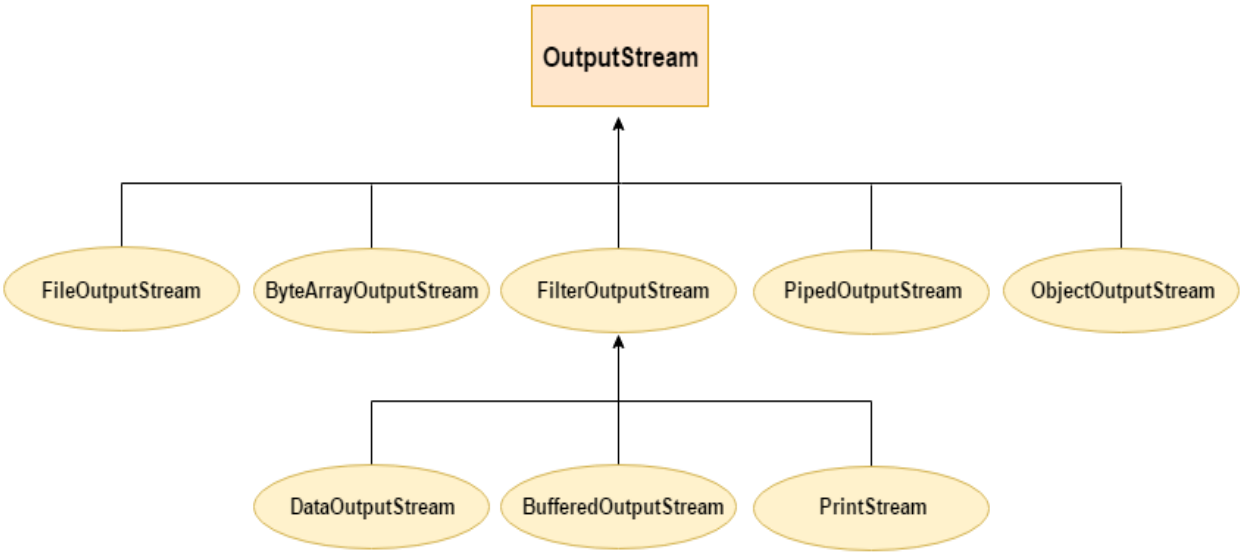
OutputStream class

OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Useful methods of OutputStream

Method	Description
1) public void write(int)throws IOException	is used to write a byte to the current output stream.
2) public void write(byte[])throws IOException	is used to write an array of byte to the current output stream.
3) public void flush()throws IOException	flushes the current output stream.
4) public void close()throws IOException	is used to close the current output stream.

OutputStream Hierarchy



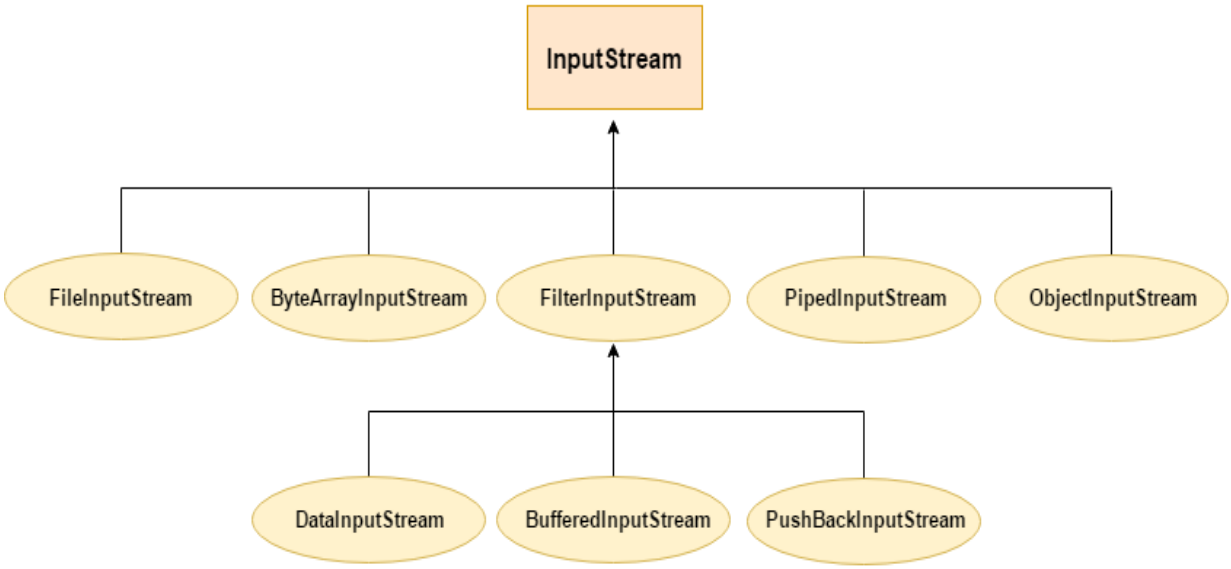
InputStream class

InputStream class is an abstract class. It is the superclass of all classes representing an input stream of bytes.

Useful methods of InputStream

Method	Description
1) public abstract int read()throws IOException	reads the next byte of data from the input stream. It returns -1 at the end of the file.
2) public int available()throws IOException	returns an estimate of the number of bytes that can be read from the current input stream.
3) public void close()throws IOException	is used to close the current input stream.

InputStream Hierarchy



RandomAccessFile

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.

Class declaration

Following is the declaration for Java.io.RandomAccessFile class −

public class RandomAccessFile

extends Object

implementsDataOutput, DataInput, Closeable

Class RandomAccessFile

java.lang.Object

java.io.RandomAccessFile

All Implemented Interfaces:

Closeable, DataInput, DataOutput, AutoCloseable

public class RandomAccessFile

extendsObject

implementsDataOutput, DataInput, Closeable

- Instances of this class support both reading and writing to a random access file.
- A random access file behaves like a large array of bytes stored in the file system.
- There is a kind of cursor, or index into the implied array, called the file pointer; input operations read bytes starting at the file pointer and advance the file pointer past the bytes read.
- If the random access file is created in read/write mode, then output operations are also available; output operations write bytes starting at the file pointer and advance the file pointer past the bytes written.
- Output operations that write past the current end of the implied array cause the array to be extended.
- The file pointer can be read by the getFilePointer method and set by the seek method.
- It is generally true of all the reading routines in this class that if end-of-file is reached before the desired number of bytes has been read, an EOFException (which is a kind of IOException) is thrown.
- If any byte cannot be read for any reason other than end-of-file, an IOException other than EOFException is thrown.
- In particular, an IOException may be thrown if the stream has been closed.

SERIALIZATION

- Java provides a mechanism, called object serialization where an object can be represented as a sequence of bytes that includes the object's data as well as information about the object's type and the types of data stored in the object.
- After a serialized object has been written into a file, it can be read from the file and deserialized that is, the type information and bytes that represent the object and its data can be used to recreate the object in memory.
- Most impressive is that the entire process is JVM independent, meaning an object can be serialized on one platform and deserialized on an entirely different platform.
- Classes **ObjectInputStream** and **ObjectOutputStream** are high-level streams that contain the methods for serializing and deserializing an object.

- The ObjectOutputStream class contains many write methods for writing various data types, but one method in particular stands out −
- public final void writeObject(Object x) throws IOException
- The above method serializes an Object and sends it to the output stream. Similarly, the ObjectInputStream class contains the following method for deserializing an object −public final Object readObject() throws IOException, ClassNotFoundException
- This method retrieves the next Object out of the stream and deserializes it. The return value is Object, so you will need to cast it to its appropriate data type.
- To demonstrate how serialization works in Java, I am going to use the Employee class that we discussed early on in the book.
- Suppose that we have the following Employee class, which implements the Serializable interface −

Example

```
Public class Employeeimplementsjava.io.Serializable{
public String name;
public String address;
public transientint SSN;
public int number;

public void mailCheck(){
System.out.println("Mailing a check to "+ name +" "+ address);
}
}
```

Notice that for a class to be serialized successfully, two conditions must be met −

- The class must implement the java.io.Serializable interface.
- All of the fields in the class must be serializable. If a field is not serializable, it must be marked **transient**.

- If you are curious to know if a Java Standard Class is serializable or not, check the documentation for the class. The test is simple: If the class implements java.io.Serializable, then it is serializable; otherwise, it's not.

Serializing an Object

- The ObjectOutputStream class is used to serialize an Object. The following SerializeDemo program instantiates an Employee object and serializes it to a file.
- When the program is done executing, a file named employee.ser is created. The program does not generate any output, but study the code and try to determine what the program is doing.

Example

```
import java.io.*;
publicclassSerializeDemo{

publicstaticvoid main(String[]args){
Employee e =newEmployee();
    e.name ="Reyan Ali";
e.address="PhokkaKuan, Ambehta Peer";
e.SSN=11122333;
e.number=101;

try{
FileOutputStreamfileOut=
newFileOutputStream("/tmp/employee.ser");
ObjectOutputStreamout=newObjectOutputStream(fileOut);
out.writeObject(e);
out.close();
fileOut.close();
System.out.printf("Serialized data is saved in /tmp/employee.ser");
```

```
}catch(IOExceptioni){
i.printStackTrace();
}
}
}
```

Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantage of Applet

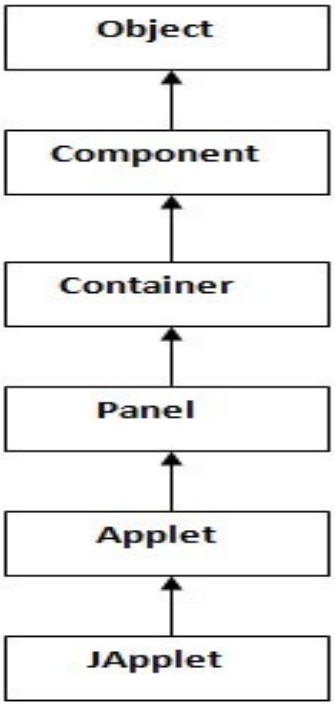
There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, MacOs etc.

Drawback of Applet

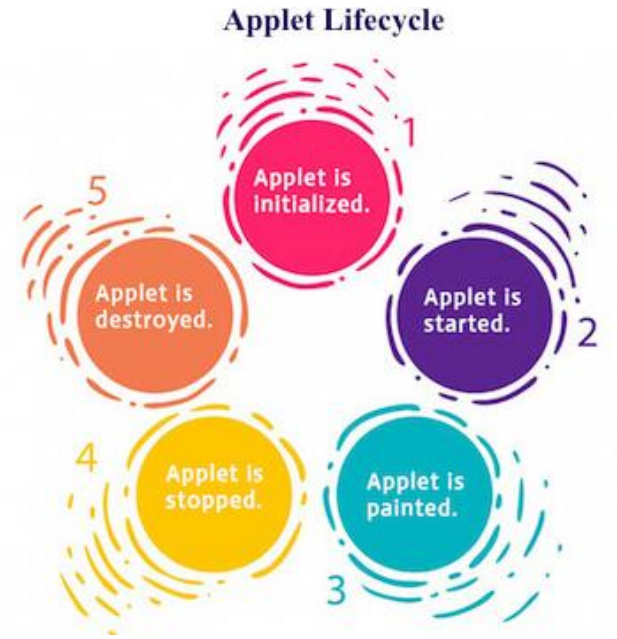
- Plugin is required at client browser to execute applet.

Hierarchy of Applet



Lifecycle of Java Applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.



java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1. **public void init():** is used to initialized the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

java.awt.Component class

The Component class provides 1 life cycle method of applet.

1. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

How to run an Applet?

There are two ways to run an applet

1. By html file.
2. By appletViewer tool (for testing purpose).

Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
1. //First.java
2. import java.applet.Applet;
3. import java.awt.Graphics;
4. public class First extends Applet{
5.
6. public void paint(Graphics g){
7. g.drawString("welcome",150,150);
8. }
9.
10. }
```

myapplet.html

```
1. <html>
2. <body>
3. <applet code="First.class" width="300" height="300">
4. </applet>
5. </body>
6. </html>
```

Simple example of Applet by appletviewer tool:

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
1. //First.java
2. import java.applet.Applet;
3. import java.awt.Graphics;
4. public class First extends Applet{
5.
6. public void paint(Graphics g){
7. g.drawString("welcome to applet",150,150);
8. }
9.
10. }
11. /*
12. <applet code="First.class" width="300" height="300">
13. </applet>
14. */
```

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac First.java
c:\>appletviewer First.java
```