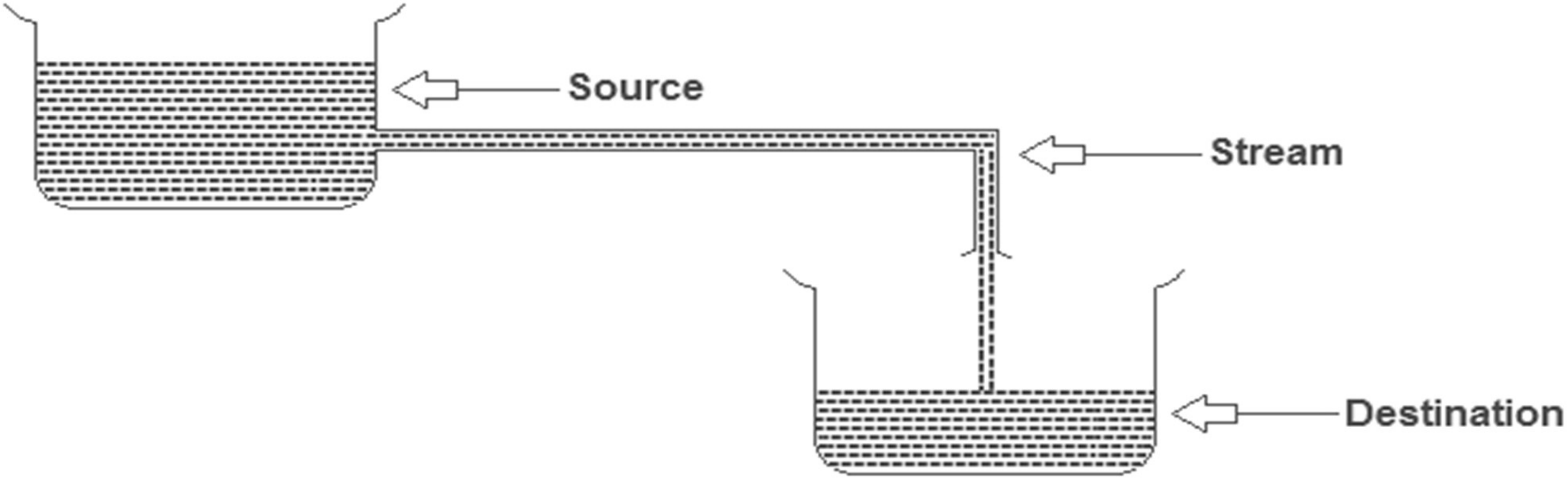
I/O Stream

AUGUST 2023

Introduction

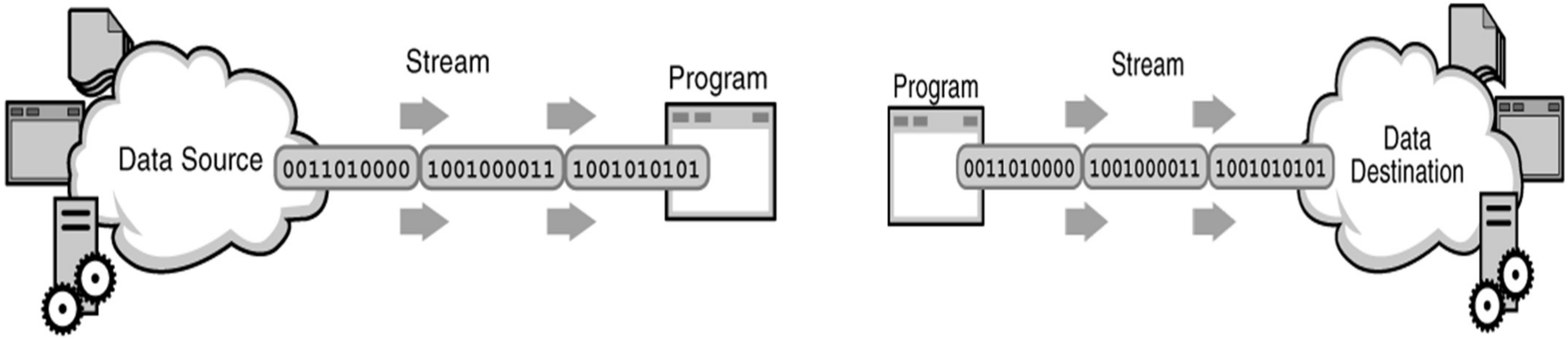
- **I/O Stream** concept is used in **Java** to make **faster I/O operations.**
- A stream is a **sequence of data**. It's called a stream because it is like a **stream of water** that continues to flow.



I/O Stream

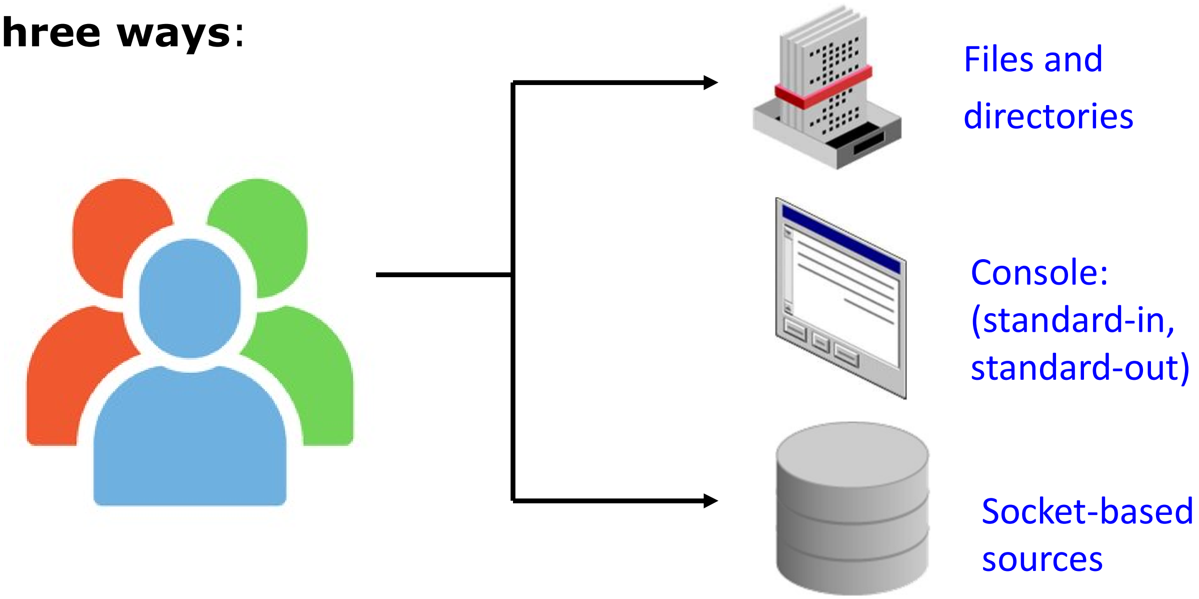
Introduction

- Streams represent **input source** and **output destination.**
- A program uses an **input stream** to **read data from a source**, one item at a time.
- A program uses an **output stream** to **write data to a destination** (sink), one item at time.



Introduction

- I/O streams support **different kinds of data**, including simple **bytes, primitive data types, localized characters, and objects.** Typically, a **developer** uses input and output in **three ways**:



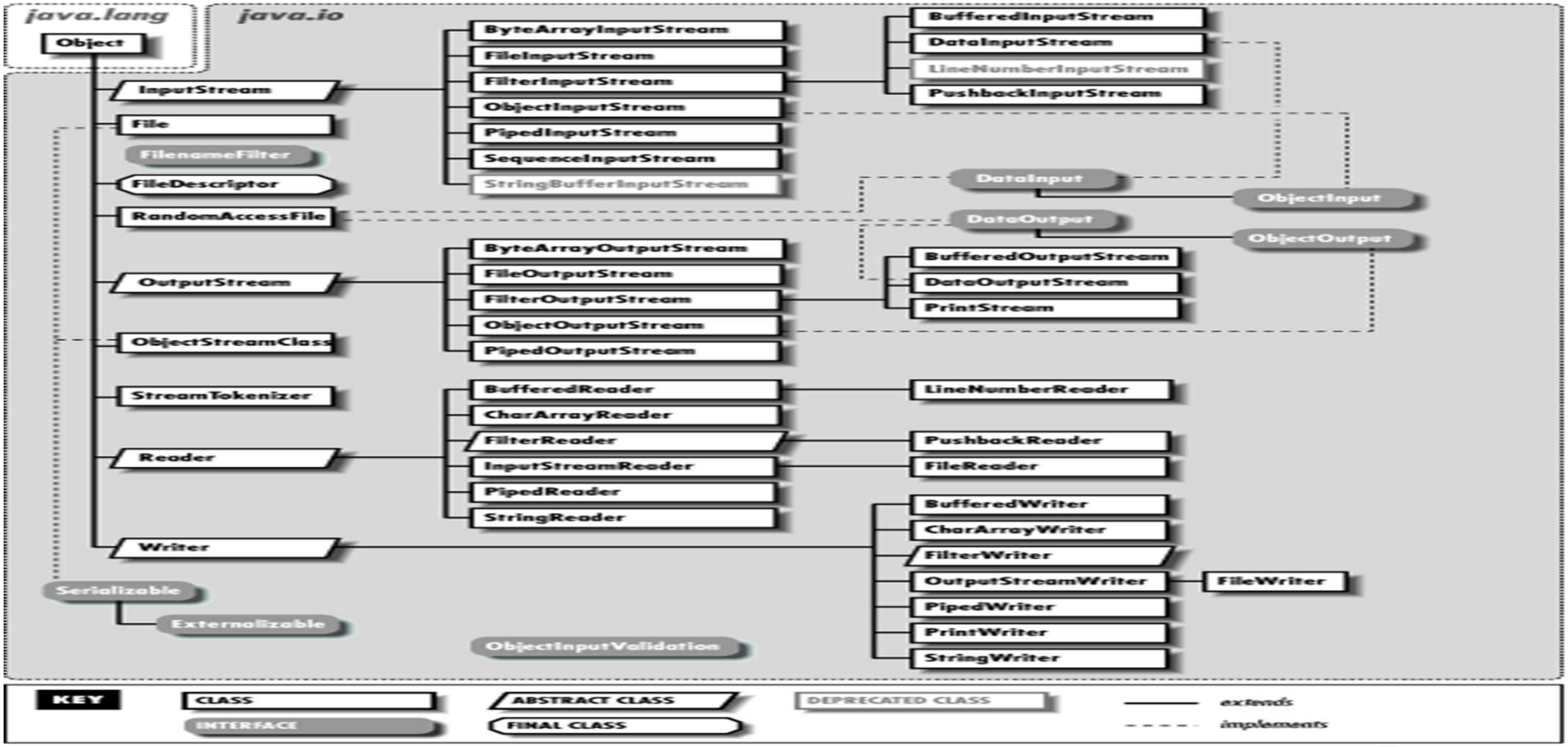
Files and directories

Console: (standard-in, standard-out)

Socket-based sources

- In Java**, java.io package** contains all the **classes and methods** need to support Input and Output operations.

Java.io package Hierarchy

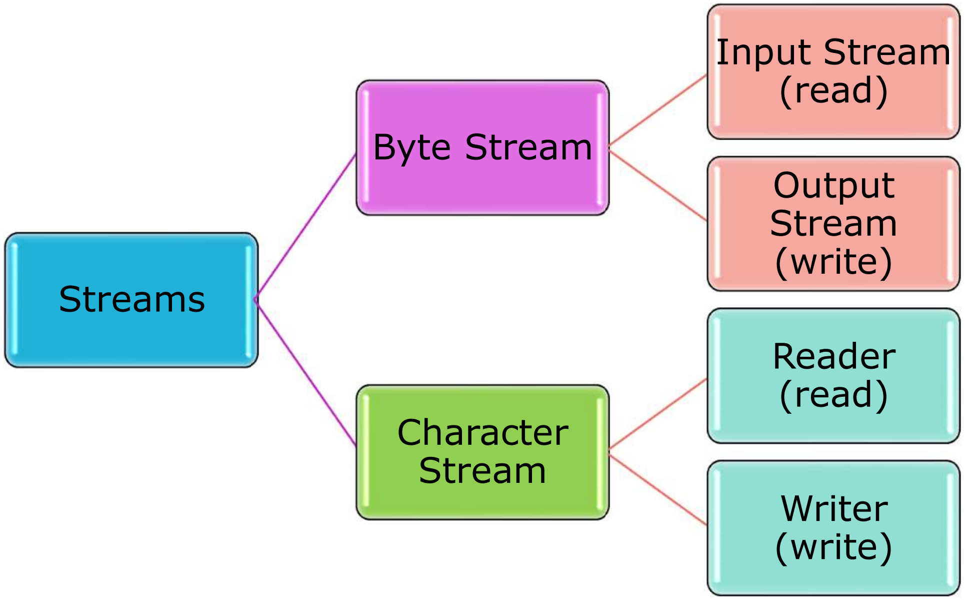


I/O Stream

Data within Streams

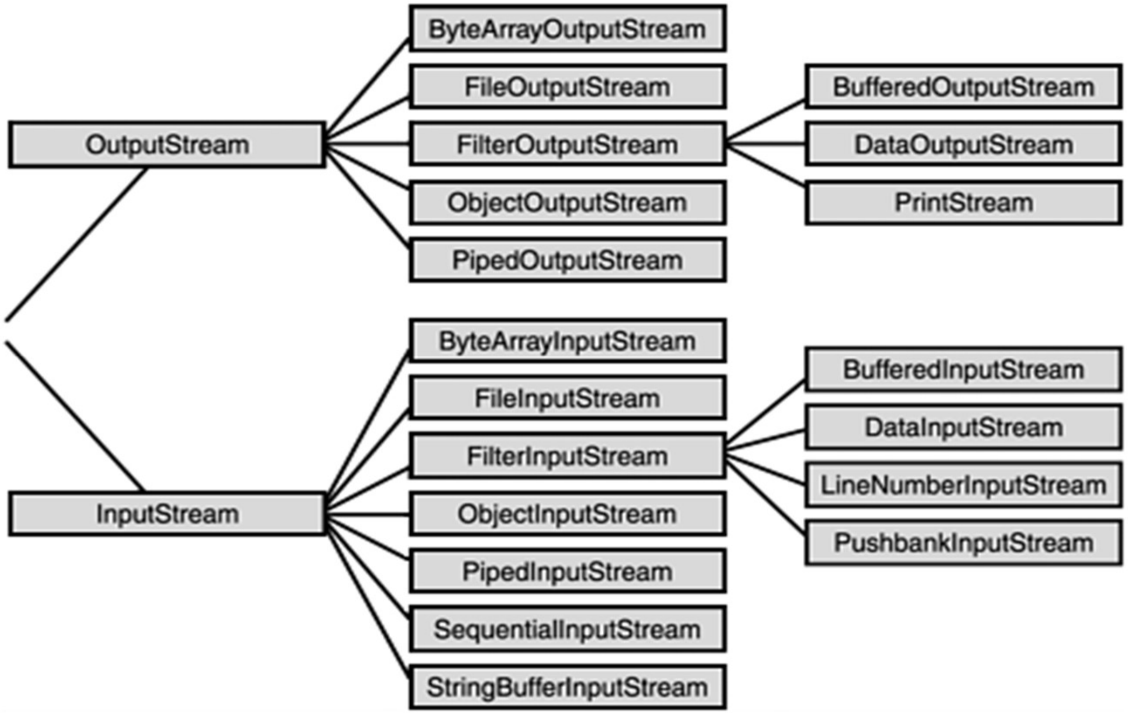
- Java technology supports two types of **streaming** data: raw **bytes** and **Unicode**

  characters. The term streams usually ref ers to **byte** and **character streams**.
- **Byte Stream**: It is used to process **data as bytes.** The bytes of data such as image files, audio files, and objects.
- **Character Stream**: It is used to process **data as characters**. It includes files and other character-based streams.



Byte Stream Classes

- Specific subclasses provide methods **for providing a** specific support for each of these **types of streams**.



Byte Stream Classes

- Programs use **byte streams** to perform input and output of **8-bit bytes**.
- Byte oriented streams **do not use** any **encoding scheme.**

Classes	Description
FileInputStream / FileOutputStream	Read data from or write data to a **file** on the native file system.
ByteArrayInputStream / ByteArrayOutputStream	Read data from or write data to a **byte array** in memory.
PipedInputStream / PipedOutputStream	Implement the input and output components of a **pipe**. Pipes are used to channel the output from one program (or thread) into the input of another.
FilterInputStream / FilterOutputStream	**Filtered streams** which process data as it's being read or written.

Byte Stream Classes

Classes	Description
DataInputStream / DataOutputStream	Read or write **primitive Java data types** in a machine-independent format.
BufferedInputStream / BufferedOutputStream	**Buffer data** while reading or writing, thereby reducing the number of accesses required on the original data source. Buffer data while reading and writing to speed it up.
SequenceInputStream	Concatenate **multiple input streams** into one input stream.
StringBufferInputStream	Allow programs to read from a **StringBuffer** as if it were an input stream.
PushbackInputStream	An input stream with a **one-byte pushback** buffer.
LineNumberInputStream	**Deprecated**

InputStream and OutputStream Methods

InputStream:

- The three basic **read methods** are,

```
int read()
nt read(byte[] buffer)
int read(byte[] buffer, int offset, int length)
```

- **Other methods** are,

```
void close();
int available();
long skip(long n);
```

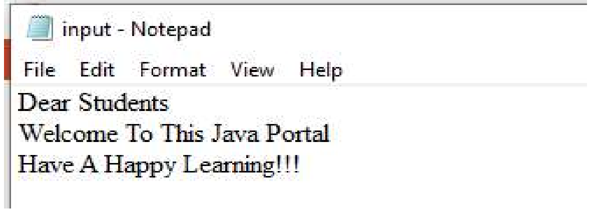
- **More Details:**

  https://docs.oracle.com/javase/7/docs/api/java/io/InputStream.html

FileInputStream: Example

```
/*** This example demonstrate the FileInputStream */
Import java.io.FileInputStream;
public class FileInputStream01 {
  public static void main(String args[]) {
     try {
        FileInputStream input = new FileInputStream("input.txt");
        System.out.println("Data in the file: ");
        int i = input.read(); // Reads the first byte
        while(i != -1) {
               System.out.print((char)i);
               i = input.read(); // Reads next byte from the file
        }
        input.close();  //close the file stream
     }catch(Exception e) {
           System.out.println(e); //Exception details
     }
} }
```

input.txt



```
Dear Students
Welcome To This Java Portal
Have A Happy Learning!!!
```

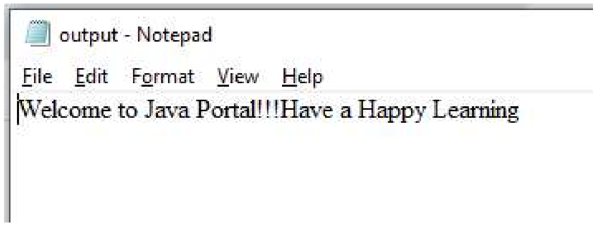
Output:

```
Data in the file:
Dear Students
Welcome To This Java Portal
Have A Happy Learning!!!
```

FileInputStream: Example

```
/*** This example demonstrate the FileOutputStream */
import java.io.FileOutputStream;
public class FileOutputStream {
    public static void main(String[] args) {
        String data = "Welcome to Java Portal!!!Have a Happy Learning";
        try {
            FileOutputStream output = new FileOutputStream("output.txt");
            byte[] array = data.getBytes();
            // Writes byte to the file
            output.write(array);
            output.close();
        }
        catch(Exception e) {
               System.out.println(e); //Exception details
        }
    }
}
```

output.txt



```
Welcome to Java Portal!!!Have a Happy Learning
```

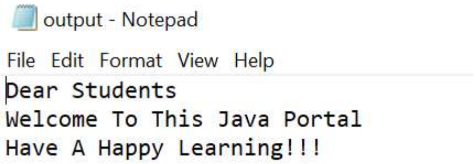
FileInputStream and FileOutputStream : Example

```
/***This example demonstrate the FileInputStream and FileOutputStream Classes*/
Import java.io.*;
class ByteIOStream{
        public static void main(String[] args) {
         byte[] b = new byte[128];
         try { FileInputStream fis = new FileInputStream (“input.txt”);
              FileOutputStream fos = new FileOutputStream (“output.txt”) )
                   System.out.println ("Bytes available: " + fis.available());
                   int count = 0; int read = 0;
                   while ((read = fis.read(b)) != -1) {
                             fos.write(b);
                             count += read;
                   }
                   System.out.println (“Total Count: " + count);
                   fis.close();
                   fos.close();
         }
```

FileInputStream and FileOutputStream : Example

```
            catch (FileNotFoundException f){
                        System.out.println ("File not found: " + f);
            }
            catch (IOException e) {
                        System.out.println ("IOException: " + e);
            }
        }
}
```

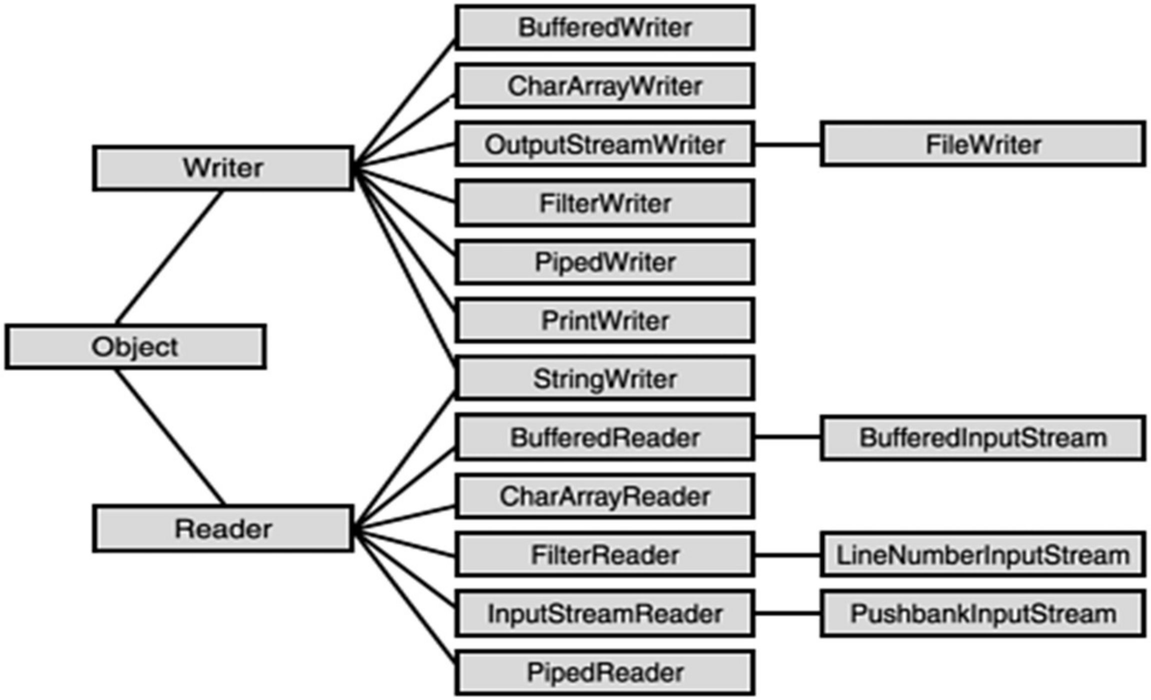
output.txt



```
Dear Students
Welcome To This Java Portal
Have A Happy Learning!!!
```

Character Stream Classes

- Specific subclasses provide methods **for providing a** specific support for each of these **types of streams**.



Character Stream Classes

- A character stream access the file **character by character.**
- Character stream can **support** all types of **character sets** ASCII, Unicode, UTF-8, UTF-16 etc.

Classes	Description
FileReader / FileWriter	Read data from or write data to a **file** on the native file system.
CharacterArrayReader / CharacterArrayWriter	Read data from or write data in to a **array**
PipedReader / PipedWriter	Implement the input and output components of a **pipe**.
FilterReader / FilterWriter	**Filtered streams** which process data as it's being read or written.

Character Stream Classes

Classes	Description
InputStreamReader / OutputStreamWriter	Used for translates **bytes to character** / Character to bytes
BufferedReader / BufferedWriter	**Buffer data** while reading and writing to speed it up.
StringReader / StringWriter	Allow programs to read and write as a **string**.
PrintStream	An **output stream** with convenient printing methods.
LineNumberReader	Buffered character stream that **keeps track of line number**

Reader and Writer Methods

Reader:

- The three basic **read methods** are,

```
int read()
nt read(char[] buffer)
int read(char[] buffer, int offset, int length)
```

- **Other methods** are,

```
void close()                    boolean ready()
long skip(long n)               boolean markSupported()
void mark(int readAheadLimit)
void reset()
```

More Details: https://docs.oracle.com/javase/7/docs/api/java/io/Reader.html

Reader and Writer Methods

Writer:

- The basic **write methods** are,

```
void write(int c)
void write(char[] cbuf)
void write(char[] cbuf, int offset, int length)
void write(String string)
void write(String string, int offset, int length)
```

- **Other methods** are,

```
void close();
void flush();
```

More Details: https://docs.oracle.com/javase/7/docs/api/java/io/Reader.html

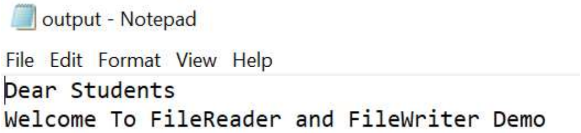
FileReader and FileWriter : Example

```
/***This example demonstrate the FileReader and FileWriter Classes using command line arguments*/
Import java.io.*;
class ByteIOStream{
    public static void main(String[] args) {
           char[] b = new char[128];
           try{
                      FileReader fr = new FileReader(args[0]);
                      FileWriter fw = new FileWriter(args[1]);
                      int count = 0; int read = 0;
                      while ((read = fr.read(b)) != -1) {
                                 fw.write(b);
                                 count += read;
                      }
                      System.out.println (“Total Count: " + count+ " characters.");
           }
```

FileReader and FileWriter : Example

```
        catch (FileNotFoundException f){
                System.out.println("File " + args[0] + " not found.");
        }
        catch (IOException e) {
                System.out.println ("IOException: " + e);
        }
    }
}
```

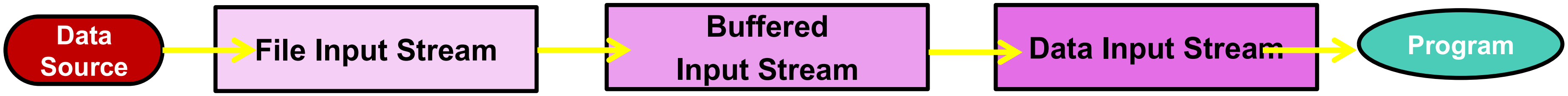
output.txt



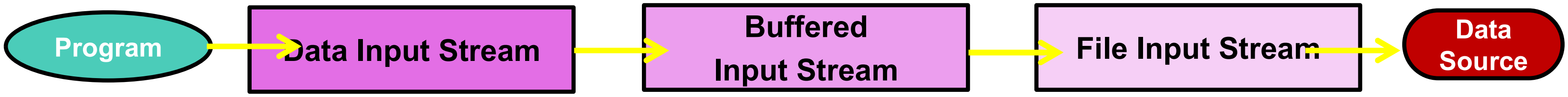
```
Dear Students
Welcome To FileReader and FileWriter Demo
```

I/O Stream Chaining

- A program **rarely** uses a **single stream** object. Instead, it **links** a series of streams to process the data. A file stream is **buffered for efficiency** and then converted into **data (Java primitives) items**.



- Data is written, then **buffered**, and finally **written** to a file.



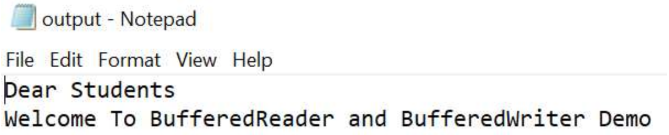
I/O Stream Chaining: Example

```
/***This example demonstrate I/O Stream Chaining using command line arguments*/
Import java.io.*;
class BufferedStream{
        public static void main(String[] args) {
            try{
                 BufferedReader bufInput = new BufferedReader(new FileReader(args[0]));
                 BufferedWriter bufOutput = new BufferedWriter(new FileWriter(args[1]));
                 String line = "";
                 while ((line = bufInput.readLine()) != null) {
                         bufOutput.write(line);
                         bufOutput.newLine();
                 }
                 bufInput.close();
                 bufOutput.close();
            }
```

I/O Stream Chaining: Example

```
                catch (FileNotFoundException f) {
                        System.out.println("File not found: " + f);
                }
                catch (IOException e) {
                         System.out.println("Exception: " + e);
                }
        }
}
```

output.txt



```
Dear Students
Welcome To BufferedReader and BufferedWriter Demo
```

Console I/O

- The **System class** in the **java.lang package** has **three static instance fields**: **out, in, and err.**
- In Java, **3 streams** are created a**utomatically**. All these streams are attached with the **console**.

1) **System.out**: standard output stream

- It is a static instance of a **PrintStream object** that enables you to write to standard output.

2) **System.in**: standard input stream

- It is a static instance of an **InputStream object** that enables to read from standard input.

3) **System.err**: standard error stream

- It is a static instance of a **PrintStream object** that enables you to write to standard error.

Console I/O

Writing to Standard Output

- The **println** and **print** methods are part of the **java.io.PrintStream** class.
- The **println** methods print the argument **with newline character** (‘\n’).
- The **print** methods print the argument **without a newline character**.
- The **print** and **println** methods are **overloaded** for **most primitive types** (boolean, char, int, long, float, and double) and for char[], Object, and String.
- The **print(Object)** and **println(Object)** methods call the **toString method** on the argument.

Console I/O

Read User Input from console

- **InputStreamReader** class can be used to **read data from console / keyboard**.
- It performs **two tasks**:

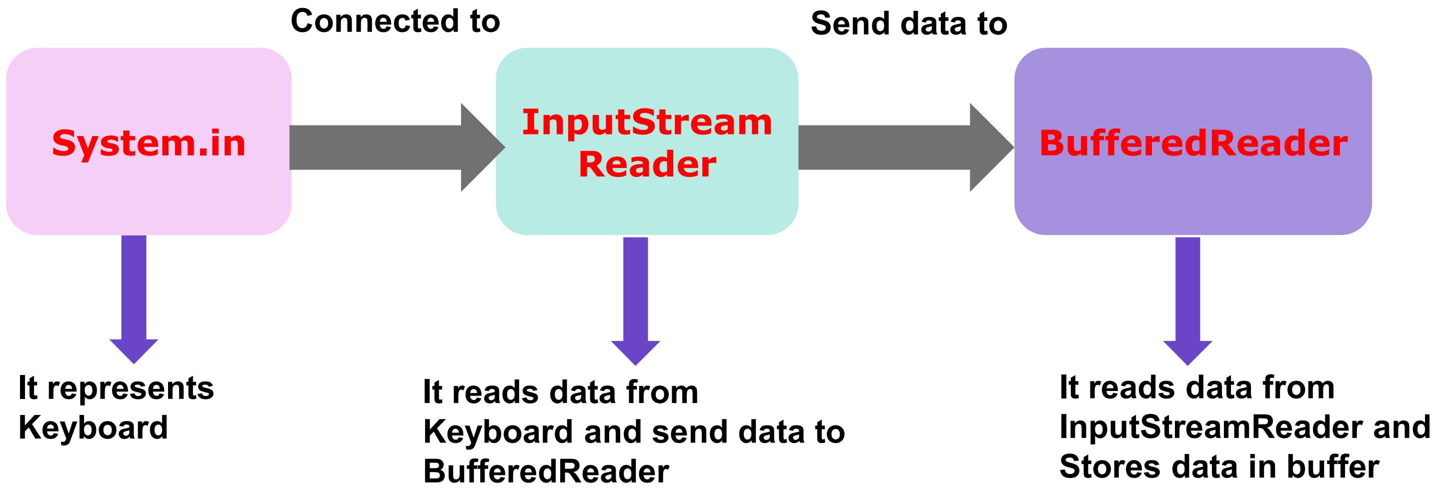
1) connects to input stream of **keyboard**

2) converts the **byte-oriented stream** into **character-oriented stream**

- **BufferedReader** Class is used to read the text from a character based input stream. It can be used to read data **line by line** using **readLine()** method.
- It makes the **performance fast.**

Console I/O

Read User Input from console



It represents Keyboard

It reads data from Keyboard and send data to BufferedReader

It reads data from InputStreamReader and Stores data in buffer

I/O Stream Chaining: Example

```
/***This example demonstrate read user input from console using InputStreamReader and BufferedReader
class*/
Import java.io.*;
class BufferedStream{
        public static void main(String[] args) throws IOException {
                 /*Accepting Different type of Input(integer,float,double,short,
                long, byte, char, string, boolean) values from Keyboard*/
                Boolean bul=false;
                BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
                System.out.println("Enter a string:");
                String str=br.readLine();
                System.out.println("Enter a integer:");
                int n=Integer.parseInt(br.readLine());
                System.out.println("Enter float value:");
                Float ft=Float.parseFloat(br.readLine());
                System.out.println("Enter short value:");
                Short sht=Short.parseShort(br.readLine());
```

I/O Stream Chaining: Example

```
 System.out.println("Enter a Double value:");
Double dub=Double.parseDouble(br.readLine());
System.out.println("Enter a long value:");
long log=Long.parseLong(br.readLine());
System.out.println("Enter a Byte value:");
Byte bit=Byte.parseByte(br.readLine());
System.out.println("Enter a character:");
char ch=(char)br.read();
//Displaying values on console
System.out.println("Entered Values are :");
System.out.println("Integer value is: "+n);
System.out.println("Float value is: "+ft);
System.out.println("Short value is: "+sht);
System.out.println("Double value is: "+dub);
System.out.println("Long value is: "+log);
System.out.println("Byte value is: "+bit);
```

I/O Stream Chaining: Example

```
                System.out.println("Byte value is: "+bit);
                System.out.println("Character value is: "+ch);
                System.out.println("String value is: "+str);
                System.out.println("Boolean value is: "+bul);
      }
}
```

Output:

```
Enter a string: hello
Enter a integer: 5
Enter float value: 5.7
Enter short value: 5
Enter a Double value: 4.342
Enter a long value: 567
Enter a Byte value: 6
Enter a character: e
```

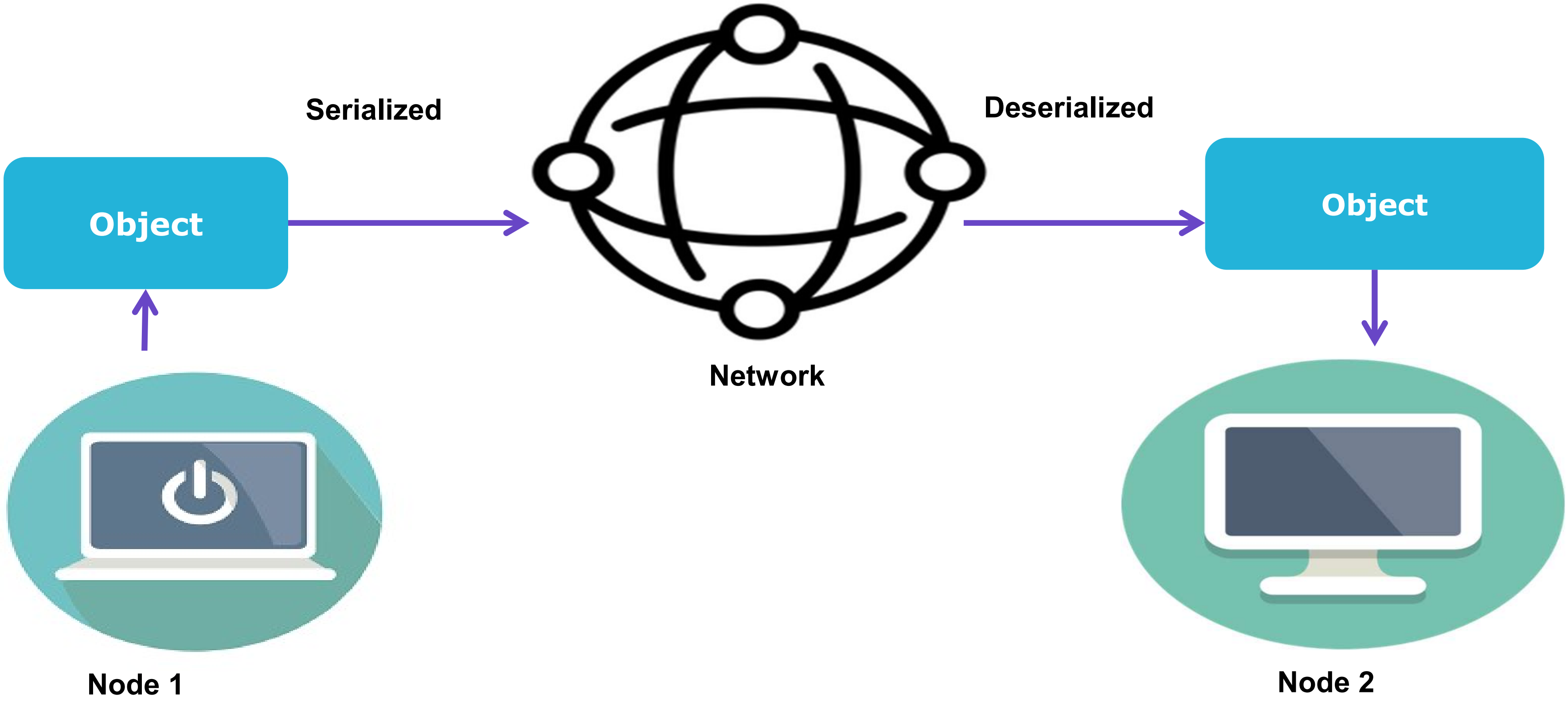
```
Entered Values are :

Integer value is: 5
Float value is: 5.7
Short value is: 5
Double value is: 4.342
Long value is: 567
Byte value is: 6
Character value is: e
String value is: hello
Boolean value is: false
```

Serialization and Deserialization

- **Persistence:** Saving data to some type of **permanent storage.**
- In general, a **non-persisted object** exists only as long as the **Java Virtual Machine is running.** An **object** that is persistent-capable can be stored on disk (or any other storage device), or sent to another machine to be stored there.
- **In Java** to achieve the **persistence** with the help of **serialization and deserialization** concepts.
- **Serialization** is a process to **convert objects** into a **writable byte stream**. Once converted into a byte-stream, these objects can be written to a **file or any other storage device** or **sent to another machine** to be stored there.
- **Deserialization** is the process of converting the **serialized** form of an **object back into a copy of the object.**

Serialization and Deserialization



Serialization and Deserialization

- A Java object is **serializable** if its **class** or any of its **superclasses** implements either the **java.io.Serializable** interface or its subinterface, **java.io.Externalizable**.
- The **ObjectOutputStream** class contains **writeObject()** method for **serializing an Object**.

  public final void writeObject(Object obj) throws IOException

- The **ObjectInputStream** class contains **readObject()** method **deserializing an object**.

  public final Object readObject() throws IOException, ClassNotFoundException

Serialization and Deserialization

Serial Version UID

- During **serialization**, a version number, **serialVersionUID**, is used to **associate the serialized output** with the class used in the serialization process.
- After **deserialization**, the **serialVersionUID** is checked to **verify** that the classes loaded are **compatible** with the **object** being **deserialized**.
- A serializable class can declare its **own serialVersionUID** by **explicitly** declaring a field named serialVersionUID as a **static final and of type long**:

```
private static long serialVersionUID = 42L;
```

Serialization and Deserialization

```
/***This example demonstrate the serialization and deserialization  concepts*/
Import java.io.*;

class Employee implements java.io.Serializable {
            public int empId;
            public String empName;

            // Parameterized Constructor
            public Employee(int id, String name) {
                        this.empId = id;
                        this.empName = name;
            }

}
```

Serialization and Deserialization

```
class Serialization {
        public static void main(String[] args) {
                Employee object = new Employee(1, "Ram");
                String filename = "file.ser";
                try{        // Serialization
                        //Saving of object in a file
                        FileOutputStream file = new FileOutputStream(filename);
                        ObjectOutputStream out = new ObjectOutputStream(file);
                        out.writeObject(object); // Method for serialization of object
                        out.close();
                        file.close();
                        System.out.println("Object has been serialized");
                }
                catch(IOException ex) {
                        System.out.println("IOException is caught");
                }
```

Serialization and Deserialization

```
Employee object1 = null;
        try{      // Deserialization
                FileInputStream file = new FileInputStream(filename);  // Reading the object from a file
                ObjectInputStream in = new ObjectInputStream(file);
                object1 = (Employee)in.readObject(); // Method for deserialization of object
                in.close();  file.close();
                System.out.println("Object has been deserialized ");
        System.out.println("Employee Id = " + object1.empId +"Employee Name = " + object1.empName);
        }
        catch(IOException ex){
                System.out.println("IOException is caught");
        }
        catch(ClassNotFoundException ex) {
                System.out.println("ClassNotFoundException is caught");
        }
    }
}
```

Output:

```
Object has been serialized
Object has been deserialized
Employee Id = 1
Employee Name = Ram
```

Serialization and Deserialization

Transient Keyword

- Java **transient** keyword is used in **serialization**.
- Fields that are marked as **transient** can not be part of the serialization and deserialization.

Example:

- In Employee class, it has three data members empId, empName and empAge. If you serialize the object, all the values will be serialized but I don't want to serialize one value, **e.g. empAge** then we can declare the **empAge data member as transient**.

Serialization and Deserialization

Points to Remember

- If a **parent class** has implemented **Serializable interface** then child class doesn’t need to implement it but vice-versa is not true.
- **Only non-static data members** are saved via Serialization process.
- **Static data members** and **transient** data members are **not saved** via Serialization process.
- **Constructor of object** is never called when an object is deserialized.

Serialization and Deserialization

```
/*This example demonstrate the serialization and deserialization  concepts with transient and static members*/
class Employee implements java.io.Serializable {
        private static final long serialversionUID =129348938L;
        transient int empId;
        static int deptId;
        String empName;
        int empAge;
        public Employee(String name, int age, int id, int deptid)  {          // Default constructor
                empName = name;
                empAge = age;
                empId = id;
                deptId = deptid;
        }
         public void printData()  {
                System.out.println(serialversionUID+" "+empId+" "+empName+" "+empAge+" "+deptId);
        }
}
```

Serialization and Deserialization

```
class Serialization {
    public static void main(String[] args) {
        Employee object = new Employee("Ram", 29, 2, 1000);
        String filename = "Ram.txt";
        try {// Serialization
            FileOutputStream file = new FileOutputStream (filename);
            ObjectOutputStream out = new ObjectOutputStream(file);
            out.writeObject(object); // Method for serialization of object
            out.close();
            file.close();
            System.out.println("Object has been serialized\n”+ "Data before Deserialization.");
            object.printData(object);
            object.deptId = 2000; // value of static variable changed
            object.empAge=100;          // value of non static vaiable changed
        } catch (IOException ex) {
            System.out.println("IOException is caught");
        } object = null;
```

Serialization and Deserialization

```
        try {       // Deserialization
                    // Reading the object from a file
                    FileInputStream file = new FileInputStream(filename);
                    ObjectInputStream in = new ObjectInputStream(file);
                    // Method for deserialization of object
                    object= (Employee)in.readObject();
                    in.close();
                    file.close();
                    System.out.println("Object has been deserialized\n”+ "Data after Deserialization.");
                    object.printData(object);
        }
        catch (IOException ex) {
                    System.out.println("IOException is caught");
        }
        catch (ClassNotFoundException ex) {
                    System.out.println("ClassNotFoundException" + " is caught");
        }
} }
```

Output:

```
Object has been serialized
Data before Deserialization.
129348938 2 Ram 29 1000
Object has been deserialized
Data after Deserialization.
129348938 0 Ram 29 2000
```

Quiz

1. In java, how many streams are created for us automatically?

a) 2

b) 4

c) 1

d) 3

e) None of the above

d) 3

Quiz

2. Which of these class is not a member class of java.io package?

a) File

b) StringReader

c) Writer

d) String

e) None of the above

d) String

I/O Stream

Quiz

3. Which of these class is used to read characters in a file?

a) FileReader

b) FileWriter

c) FileInputStream

d) InputStreamReader

e) File

a) FileReader

Quiz

4. Which of these is a method to clear all the data present in output buffers?

a) clear()

b) flush()

c) fflush()

d) close()

e) cls()

b) flush()

Quiz

5. Which of these class can be used to implement the input stream that uses a character array as the source?

a) BufferedReader

b) FileReader

c) CharArrayReader

d) FileArrayReader

e) ByteReader

c) CharArrayReader

I/O Stream

Quiz

6. Which streams are used to perform input and output of 8-bit bytes?

a) Character

b) Byte

c) Bit

d) Double

e) String

b) Byte

Quiz

7. Which one is used to output the error data produced by the user's program

a) System.in

b) System.err

c) System.out

d) System.io

b) System.err

I/O Stream

Quiz

8. Which of these classes defined in java.io and used for file-handling are abstract?

a) InputStream
b) PrintStream
c) Reader
d) FileInputStream

a) Only a

b) Only c

c) Both a & c

d) Both b & d

e) None of these Above

c) Both a & c

I/O Stream

Quiz

9. Which of these is a method of ObjectInput interface used to deserialize an object from a stream?

a) int read()

b) void close()

c) Object readObject()

d) None of these above

c) Object readObject()