I/O Data

Streams :

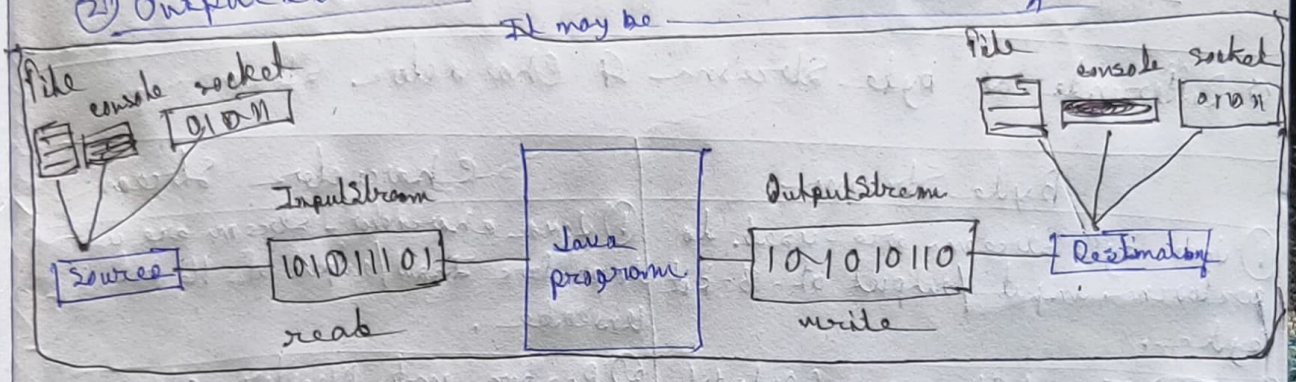
- A stream is a sequence of data. ~~In Java, a stream is composed of bytes or chars.~~
- The java.io package contains a larger number of stream class that helps the user to perform required I/O input/output operations.
- These streams support all the type of object, data type, characters, files to fully execute the I/O operation

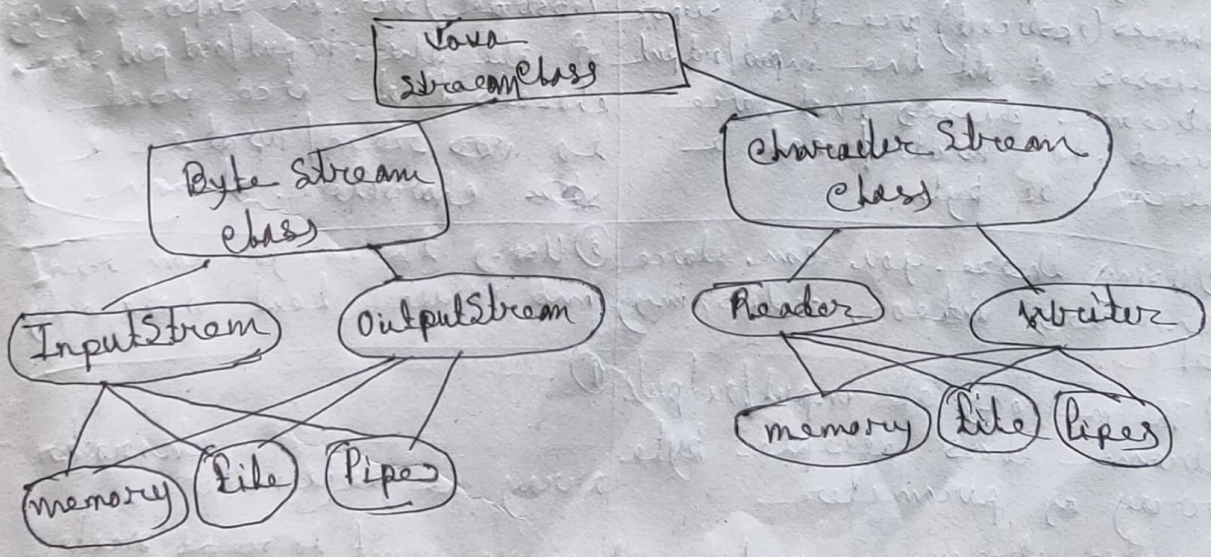
There are two kind of Streams —

(1) Input Stream : used to read data from a source, it may be a file, an array, peripheral device, socket etc

(2) ~~Input Stream~~

(2) Output Stream : used to write data ~~from~~ to a destination, It may be





2021

① File f = new File("myFile.text")

f.createNewFile() method is used to create "myFile.txt" file.

② What will be the output when execute with command line arguments?

Valid context how are you

```
class Context
{
    psvm(String[] args)
    {
        System.out.println(arg[7]);
    }
}
```

→ Any 8 are

○ Differents of byte stream & character stream.

Byte Stream	Character Stream
① Java byte streams are used to perform input/output of 8-bit byte.	① Java character stream are used to perform Input/output of 16-bit unicode.
② ~~Input~~ Input Stream and OutputStream classes (abstract) are the super classes of all the input/output stream classes that are used to write/read a stream of byte	② Reader and Writer classes (abstract) are the super classes of all the input/output stream classes that are used to read/write a character stream
③ Using these you can store characters, videos, audios, images etc.	③ Using these you can read and write text data only
④ The methods of input/output stream class accepts byte array as parameter	④ character array as parameter.

- Character stream is useful when we want to read/write text data.
- Byte stream is useful for processing raw data like binary files (video, audio, images) etc.

Most popular classes for performing Input/output.

The common classes for Byte streaming in Java are FileInputStream and FileOutputstream.

The common classes for character streaming in Java are FileReader and FileWriter.

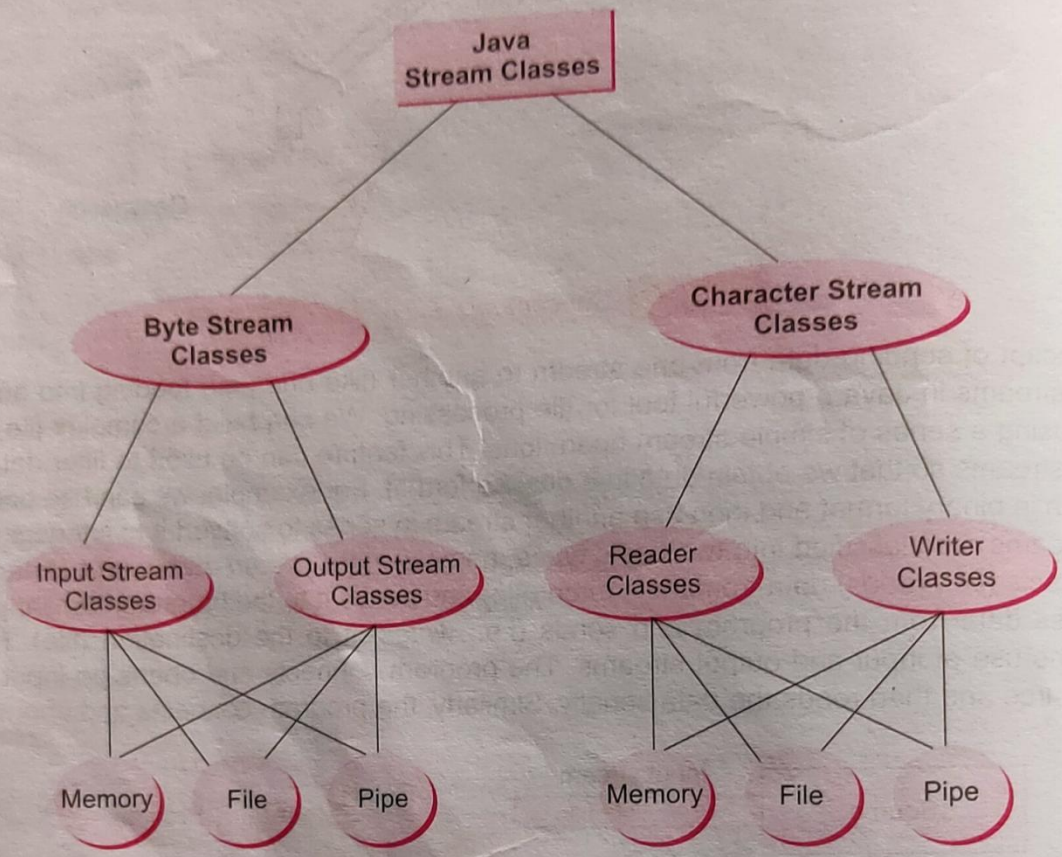


Fig. 16.5 *Classification of Java stream classes*

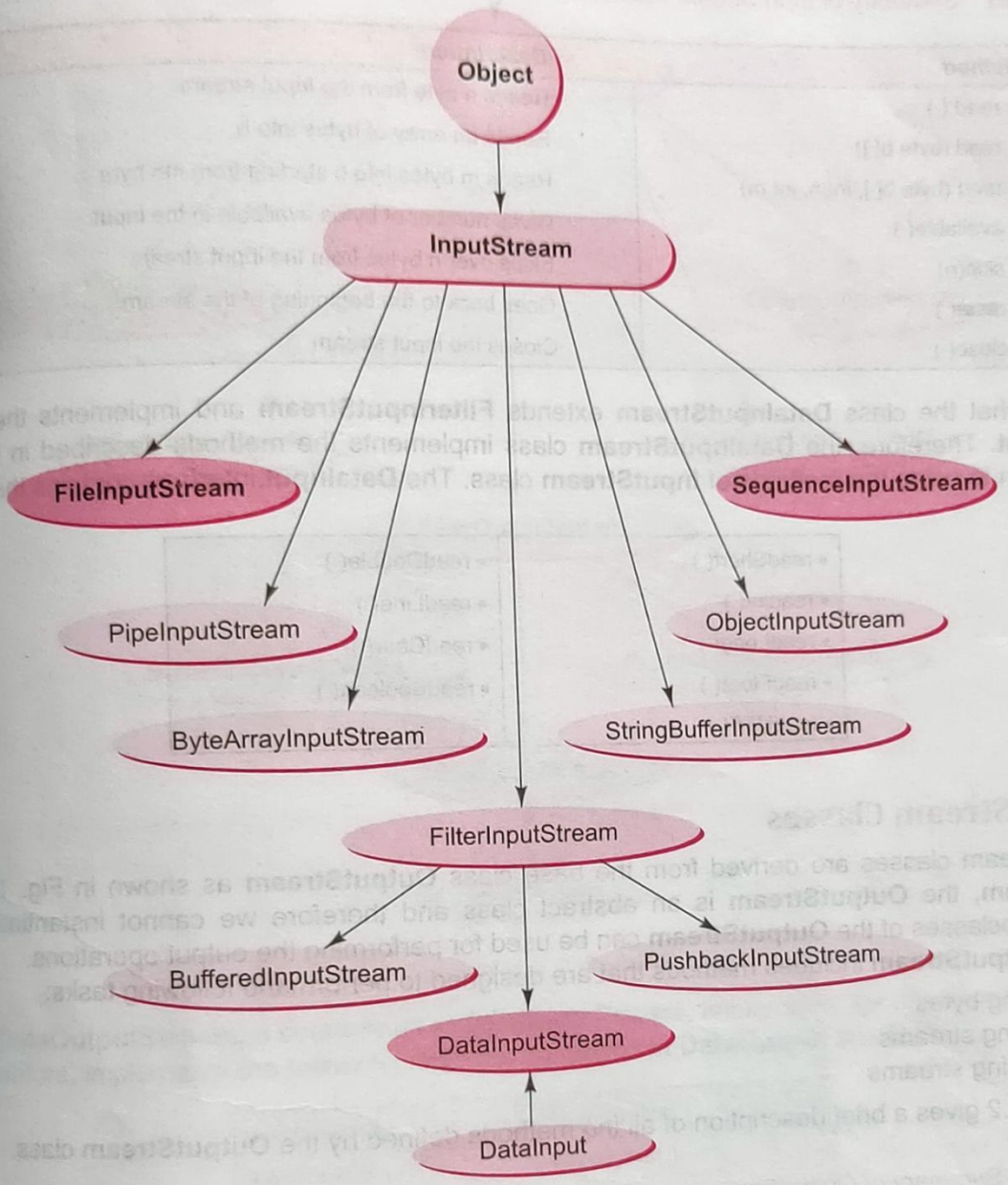


Fig. 16.6 *Hierarchy of input stream classes*

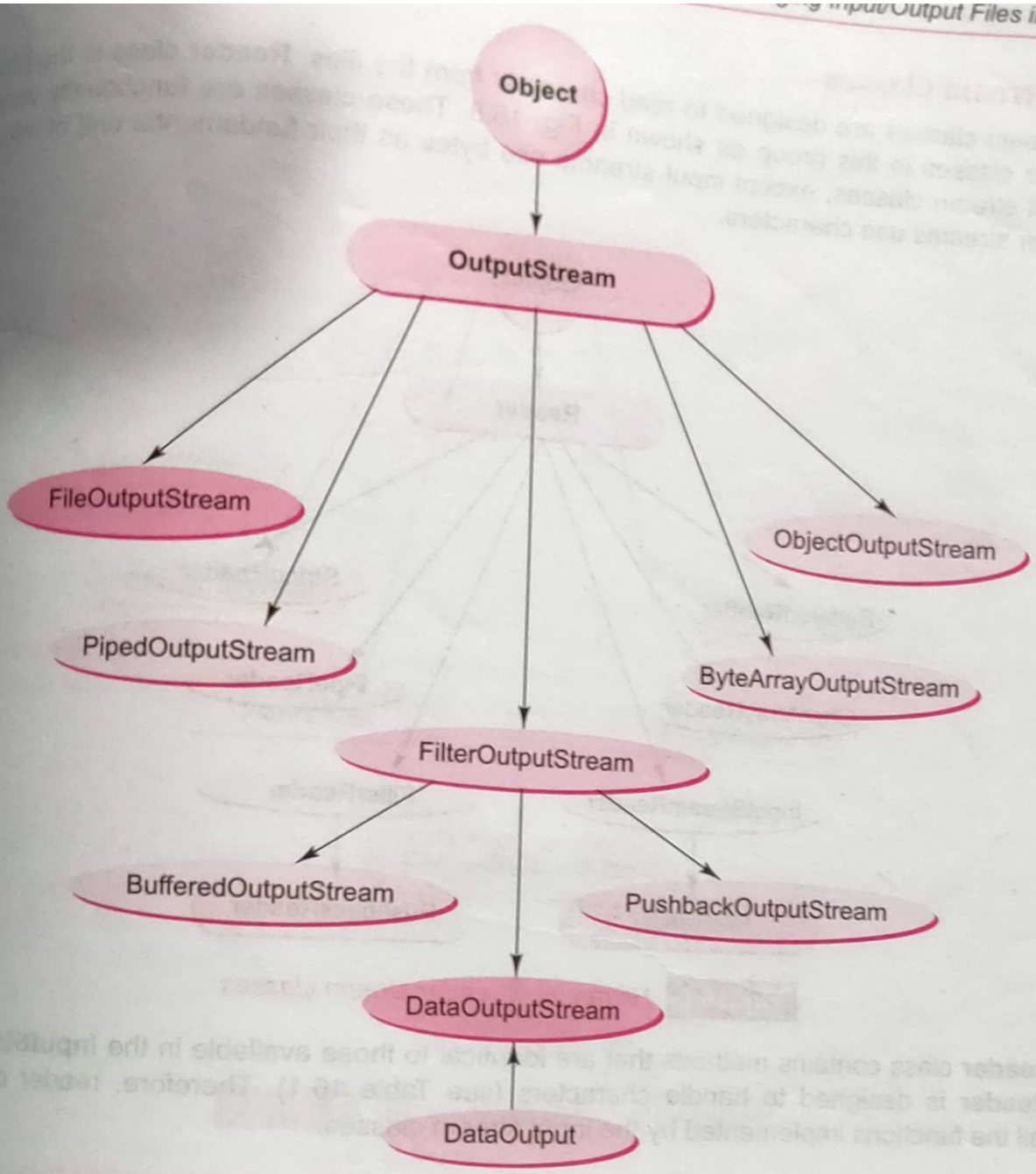


Fig. 16.7 *Hierarchy of output stream classes*

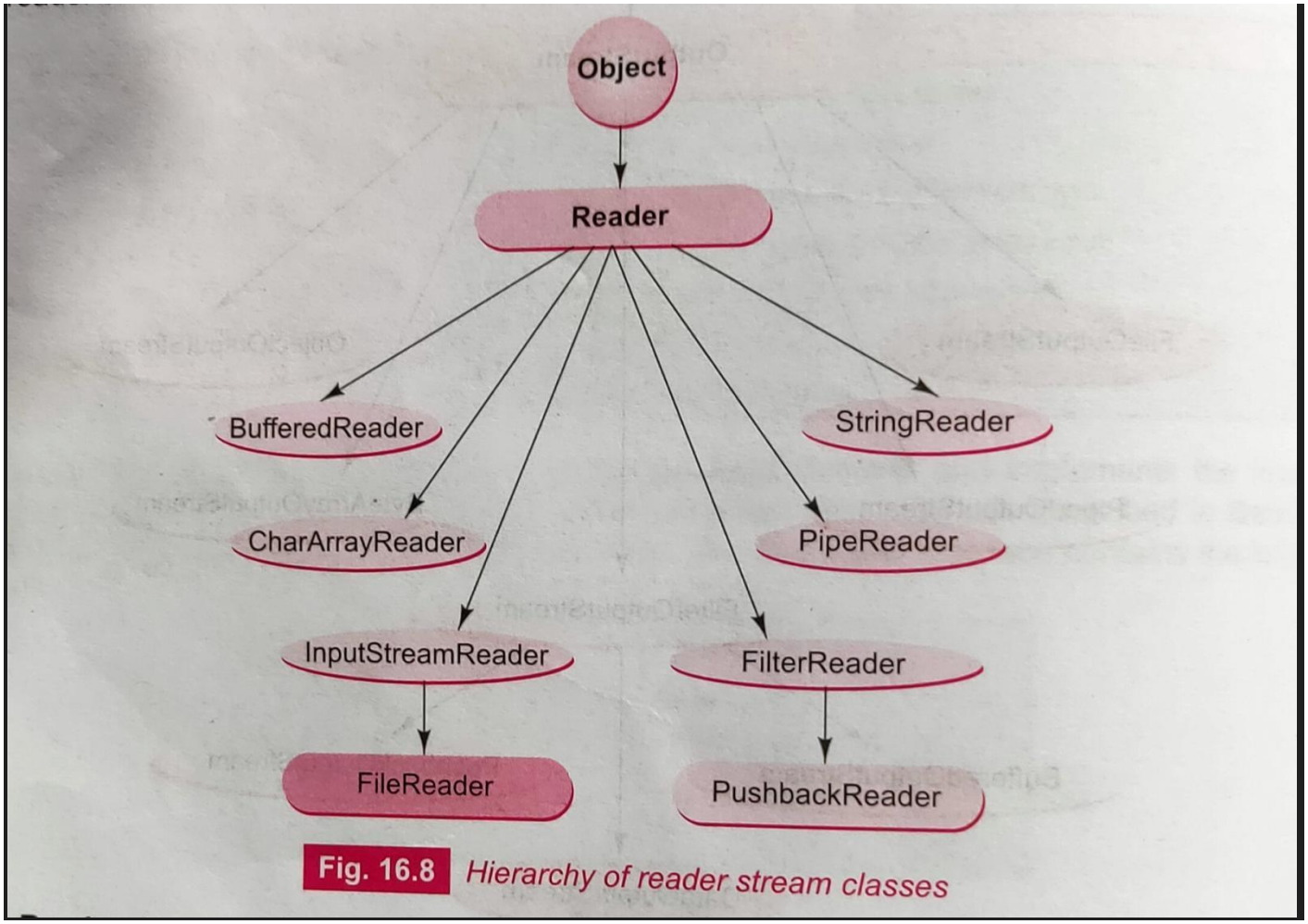


Fig. 16.8 *Hierarchy of reader stream classes*

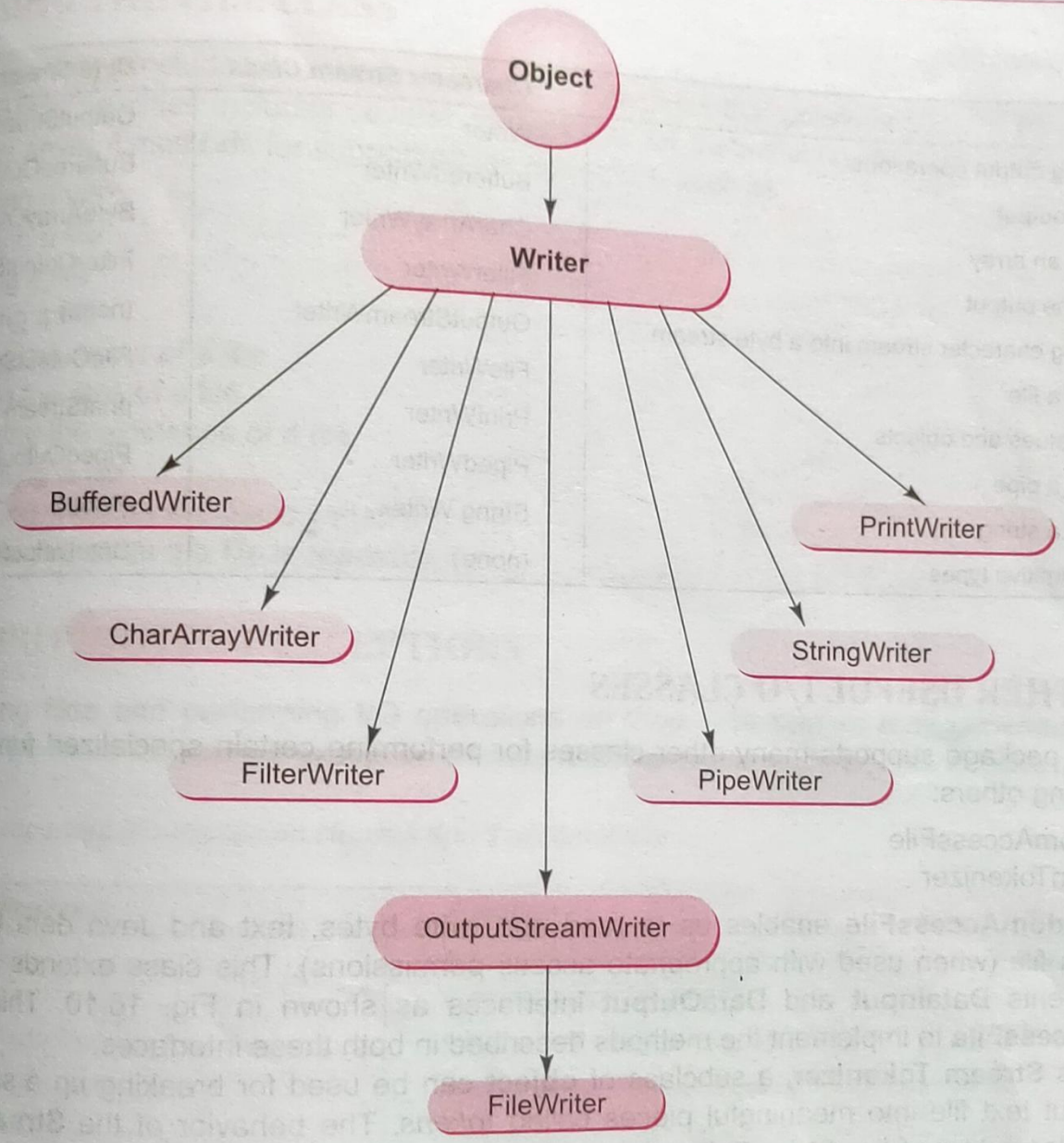


Fig. 16.9 *Hierarchy of writer stream classes*

Table 16.3 *List of tasks and classes implementing them*

Task	Character Stream Class	Byte Stream Class
Performing input operations	Reader	InputStream
Buffering input	BufFeredReader	BufferdInputStream
Keeping track of line numbers	LineNumberReader	LineNumberInputStream
Reading from an array	CharArrayReader	ByteArrayInputStream
Translating byte stream into a character stream	InputStreamReader	(none)
Reading from files	FileReader	FileInputStream
Filtering the input	FilterReader	FilterInputStream
Pushing back characters/bytes	PushbackReader	PushbackInputStream
Reading from a pipe	PipedReader	PipedInputStream
Reading from a string	StringReader	StringBufferInputStream
Reading primitive types	(none)	DataInputStream

(Contd.)

Table 16.3 *(Contd.)*

Task	Character Stream Class	Byte Stream Class
Performing output operations	Writer	OutputStream
Buffering output	BufferedWriter	BufferedOutputStream
Writing to an array	CharArrayWriter	ByteArrayOutputStream
Filtering the output	FilterWriter	FilterOutputStream
Translating character stream into a byte stream	OutputStreamWriter	(none)
Writing to a file	FileWriter	FileOutputStream
Printing values and objects	PrintWriter	printStream
Writing to a pipe	PipedWriter	PipedOutputStream
Writing to a string	String Writer	(none)
Writing primitive types	(none)	DataOutputStream

```
//Create File
import java.io.*;
class CreateFile
{
    public static void main(String[] args)
    {
        try
        {
            File file = new File("myFile.txt");
            if (file.createNewFile())
```

```
                System.out.println(file.getName()+ "
is created");
            else
                System.out.println("File is already
exists");
        }
        catch(IOException e)
        {
            System.out.println(e);
        }
```

```
    }
}


//Create Directory
import java.io.File;
//Create Directory
class MakeDirectory
{
    public static void main(String[] args)
    {
        File dir = new File("MyDir");
        // File dir = new File("C:/Users/MyDir");
        dir.mkdir();
        // dir.delete();
        String abs_path=dir.getAbsolutePath();
        String name = dir.getName();
        System.out.println("abs_path: "+abs_path);
        System.out.println("name: "+name);
    }
}
```

```
//Read Character from File
import java.io.*;
```

```
public class ReadCharToFile
{
    public static void main(String[] args)
    {
        FileReader fr=null;
        String s; //String a sequence of
characters  char [] ch
        int i;
        try
        {
            fr = new FileReader("myFile1.txt");
            while ((i=fr.read()) != -1)
            {
                System.out.print((char)i);
            }
        }
        catch(IOException e)
        {
            System.err.println(e);
        }
        finally
        {
            try
            {
                fr.close();
            }
            catch(IOException e)
            {
                System.err.println(e);
            }
        }
```

```
    }
}
```

Output:

NAME: Tanmay Roy
Roll: 32

```
//Write Character to File
import java.io.*;
public class WriteCharToFile
{
    public static void main(String[] args)
    {
        FileWriter fw = null;
        // char ch;
        String s;
        try
        {
            fw = new FileWriter("myFile1.txt");
            s="NAME: Tanmay Roy\nRoll: 32";
            // ch = 'a';

            fw.write(s);
            // fw.write(ch);
        }
        catch(IOException e)
        {
            System.err.println(e.getStackTrace());
        }
        finally
        {
            try
```

```
            {
                fw.close();
            }
            catch(IOException e)
            {
                System.err.println(e);
            }
        }
    }
}
```

```
//Read byte from File
import java.io.*;
class ReadByteFromFile
{
    public static void main(String[] args) //throws
IOException
    {
        FileInputStream infile = null;
        int i;
        try
        {
            // File f = new File("myFile");
            infile = new
FileInputStream("myFile1.txt");
            while ((i=infile.read()) != -1)
            {
                System.out.print((char)i);
            }
        }
        catch(IOException e)
```

```
        {
            System.err.println(e);
        }
        finally
        {
            try
            {
                infile.close();
            }
            catch(IOException e)
            {
                System.out.println(e);
            }
        }
    }
}


//Write byte to File
import java.io.*;
class WriteByteToFile
{
    public static void main(String[] args) //throws
IOException
    {
        try
        {
            // File f = new File("myFile");
            FileOutputStream outfile = new
FileOutputStream("myFile.txt");
```

```
            String s = "Name: Tanmay Samanta\nAge:
21";

            byte [] b = s.getBytes();
            // byte [] b = {65,66,67};
            //            A  B  C
            outfile.write(b);
            outfile.close();
        }
        catch(IOException e)
        {
            System.err.println(e);
        }
    }
}
```

```
//Random access the contains of a file
import java.io.*;
public class RandomAccessFileTest {
    public static void main(String[] args)
    {
        RandomAccessFile raf = null;
        int i;
        try
        {
            raf = new
RandomAccessFile("myFile1.txt","rw");
            for (int a=0;a<=raf.length();a+=2)
            {
                raf.seek(a);
                i=raf.read();
```

```
                System.out.print((char)i);
            }
        }
        catch(IOException e)
        {
            System.err.println(e);
        }
        finally
        {
            try
            {
                raf.close();
            }
            catch(IOException e)
            {
                System.err.println(e.getStackTrace()
);
            }
        }

    }
}

Contain of myFile1.txt
```

NAME: Tanmay Roy

Roll: 32

Output:

NM:Tna o

ol 2

Write a program in java that takes a file name and a search string from the user. If the search string occurs in the file, then it counts the number of occurrences of the string. (Assume that search pattern can exist more than two times in a line).

```
/*
 2011-12
 Write a program in java that takes a file name and a search
 string from the user. If the search string occurs in the
 file, then it counts the number of occurrences of the string.
 (Assume that search pattern can exist more than two times in a line).
 */

 import java.io.BufferedReader;
 import java.io.File;
 import java.io.FileReader;
 import java.io.IOException;

 public class CountOccurrencesOfStringInFile {

     public static void main(String[] args) throws IOException {
         // Get the file name and search string from the user.
         System.out.print("Enter the file name: ");
         String fileName = System.console().readLine();
         System.out.print("Enter the search string: ");
         String searchString = System.console().readLine();

         // Create a File object for the file.
         File file = new File(fileName);

         // Create a BufferedReader object to read the file.
         BufferedReader bufferedReader = new BufferedReader(new
FileReader(file));

         // Initialize the count of occurrences to 0.
         int count = 0;

         // Read each line from the file.
         String line;
         while ((line = bufferedReader.readLine()) != null) {

             // Find the index of the search string in the line.
             int index = line.indexOf(searchString);
```

```
            // If the search string is found in the line, increment the
count.
            while (index != -1) {
                count++;
                index = line.indexOf(searchString, index +
searchString.length());
            }
        }

        // Close the BufferedReader object.
        bufferedReader.close();

        // Print the number of occurrences of the search string.
        System.out.println("The search string \"" + searchString + "\"
appears " + count + " times in the file.");
    }
 }
```

```
Contain of myFile1.txt
```

NAME: Tanmay Roy

Roll: 32 Roy

Output:

Enter the search string: Roy

The search string "Roy" appears 2 times in the file.

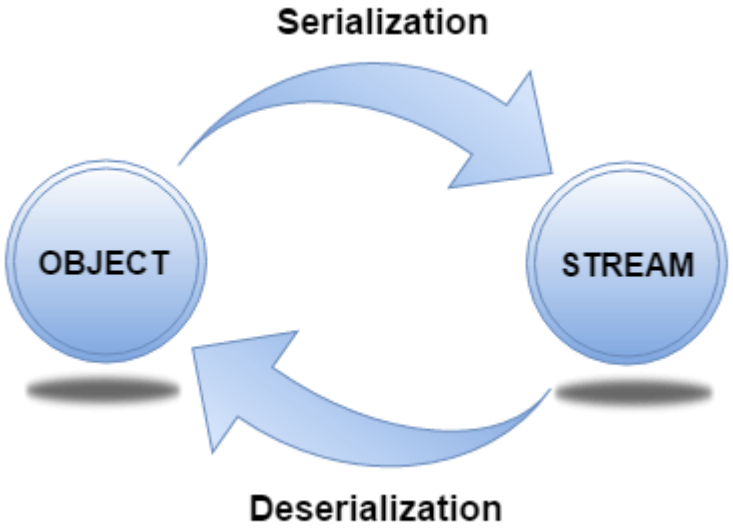
Serializtion & De-Serializtion:

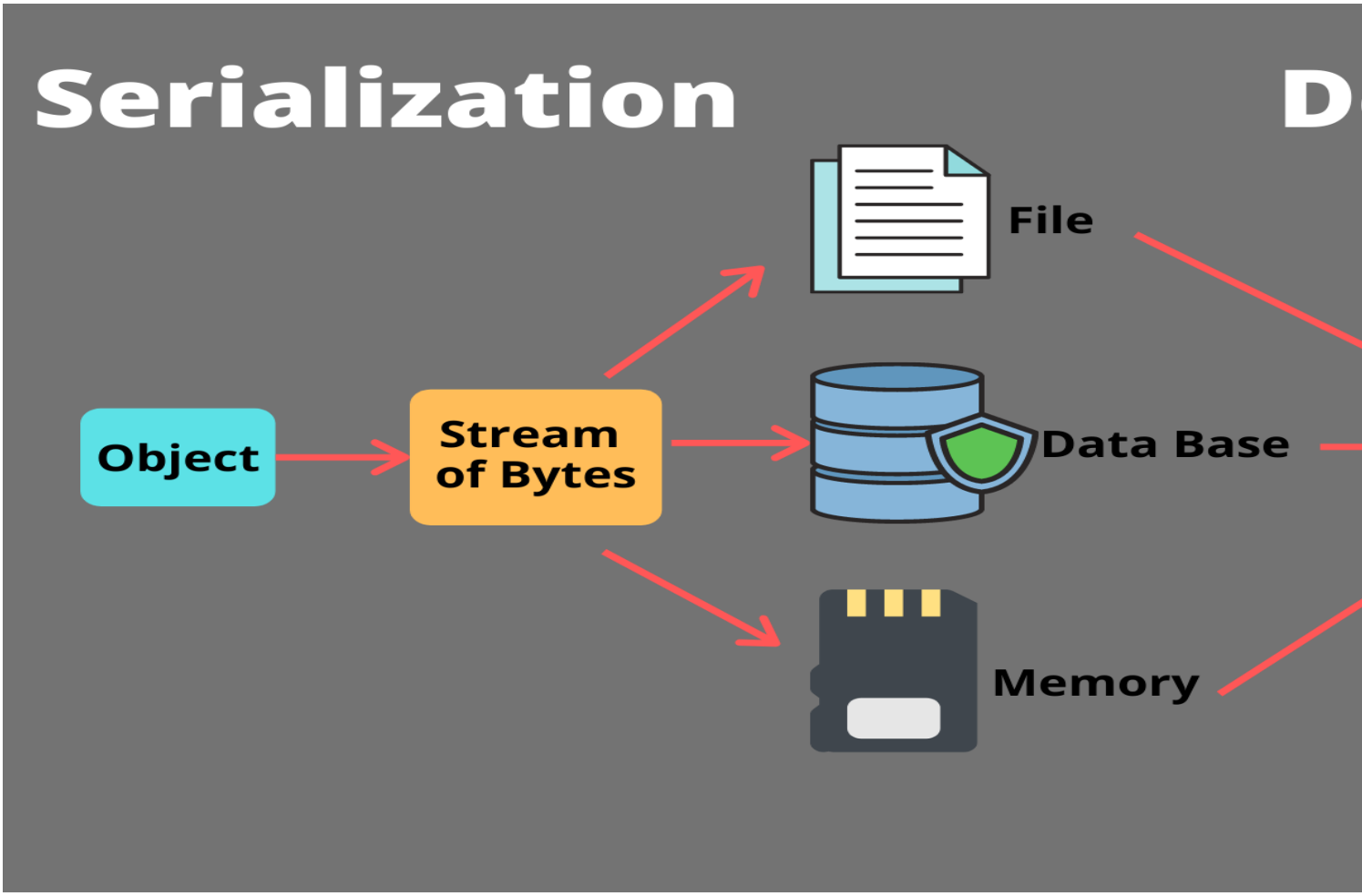
Serialization is a mechanism of converting the state of an object into a byte stream.

Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory.

This mechanism is used to persist the object.

It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies.





Advantages of Serialization

1. To save/persist state of an object.
2. To travel an object across a network.

Example of Serialization :

```
import java.io.Serializable;
public class Student implements Serializable
{
    int id;
    String name;
    public Student(int id, String name)
    {
        this.id = id;
        this.name = name;
    }
}
```

```
import java.io.*;
class Persist
{
    public static void main(String args[])throws
Exception
    {
        Student s1 =new Student(211,"ravi");

        FileOutputStream fout=new
FileOutputStream("f.txt");
        ObjectOutputStream out=new
ObjectOutputStream(fout);
```

```
        out.writeObject(s1);
        out.flush();

        out.close();
        System.out.println("success");
    }
}
```

Output:

```
success
```

Example of De-erialization:

```
import java.io.Serializable;
public class Student implements Serializable
{
    int id;
    String name;
```

```
    public Student(int id, String name)
    {
        this.id = id;
        this.name = name;
    }
}
```

```
import java.io.*;
class DePersist
{
    public static void main(String args[])throws
Exception
    {

        ObjectInputStream in=new
ObjectInputStream(new FileInputStream("f.txt"));
        Student s=(Student)in.readObject();
        System.out.println(s.id+" "+s.name);

        in.close();
 }
}
```

Output:

```
success
211 ravi Engineering 50000
```

Ways to read input from console in Java

In Java, there are four different ways for reading input from the user in the command line environment(console).

1.Using Buffered Reader Class

This is the Java classical method to take input, Introduced in JDK1.0. This method is used by wrapping the System.in (standard input stream) in an InputStreamReader which is wrapped in a BufferedReader, we can read input from the user in the command line.

- The input is buffered for efficient reading.
- The wrapping code is hard to remember.

Implementation:

Java

```
// Java program to demonstrate BufferedReader
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class Test {
    public static void main(String[] args)
        throws IOException
```

```
    {
        // Enter data using BufferReader
        BufferedReader reader = new BufferedReader(
            new InputStreamReader(System.in));

        // Reading data using readLine
        String name = reader.readLine();

        // Printing the read line
        System.out.println(name);
    }
}
```

Input:

```
Geek
```

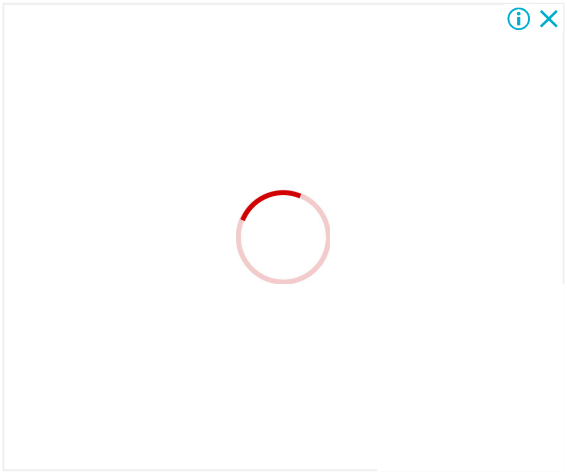
Output:

Auxiliary Space : O(1)

```
Geek
```

Note:

To read other types, we use functions like Integer.parseInt(), Double.parseDouble(). To read multiple values, we use split().



2. Using Scanner Class

This is probably the most preferred method to take input. The main purpose of the Scanner class is to parse primitive types and strings using regular expressions, however, it is also can be used to read input from the user in the command line.

- Convenient methods for parsing primitives (nextInt(), nextFloat(), ...) from the tokenized input.
- Regular expressions can be used to find tokens.
- The reading methods are not synchronized

To see more differences, please see this article.

Java

```java
// Java program to demonstrate working of Scanner in Java
import java.util.Scanner;

class GetInputFromUser {
    public static void main(String args[])
    {
        // Using Scanner for Getting Input from User
        Scanner in = new Scanner(System.in);

        String s = in.nextLine();
        System.out.println("You entered string " + s);

        int a = in.nextInt();
        System.out.println("You entered integer " + a);

        float b = in.nextFloat();
        System.out.println("You entered float " + b);
    }
}
```

Input:

```
GeeksforGeeks
12
3.4
```

Output:

```
You entered string GeeksforGeeks
You entered integer 12
You entered float 3.4
```

3. Using Console Class

It has been becoming a preferred way for reading user's input from the command line. In addition, it can be used for reading password-like input without echoing the characters entered by the user; the format string syntax can also be used (like System.out.printf()).

Advantages:

- Reading password without echoing the entered characters.
- Reading methods are synchronized.
- Format string syntax can be used.
- Does not work in non-interactive environment (such as in an IDE).

Java

```java
// Java program to demonstrate working of System.console()
// Note that this program does not work on IDEs as
// System.console() may require console
public class Sample {
    public static void main(String[] args)
    {
        // Using Console to input data from user
        String name = System.console().readLine();

        System.out.println("You entered string " + name);
    }
}
```

Input:

```
GeeksforGeeks
```

Output:

```
You entered string GeeksforGeeks
```

4. Using Command line argument

Most used user input for competitive coding. The command-line arguments are stored in the String format. The parseInt method of the Integer class converts string argument into Integer. Similarly, for float and others during execution. The usage of args[] comes into existence in this input form. The passing of information takes place during the program run. The command line is given to args[]. These programs have to be run on cmd.

Code:

Java

```java
// Program to check for command line arguments
class Hello {
    public static void main(String[] args)
    {
        // check if length of args array is
        // greater than 0
        if (args.length > 0) {
            System.out.println(
                "The command line arguments are:");

            // iterating the args array and printing
            // the command line arguments
            for (String val : args)
                System.out.println(val);
        }
        else
```

```
            System.out.println("No command line "
                               + "arguments found.");
    }
}
```

Command Line Arguments:

```
javac GFG1.java
java Main Hello World
```

Output:

```
The command line arguments are:
Hello
World
```

Please refer this for more faster ways of reading input.

This article is contributed by **D Raj Ranu**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Last Updated : 18 Mar, 2022

Similar Reads

1. Java Multithreading - One Thread to Take Input, Another to Print on Console
2. Taking input from console in Python
3. Java.io.Console class in Java

Java Console Example to read password

```
import java.io.Console;
class ReadPasswordTest{
public static void main(String args[]){
Console c=System.console();
System.out.println("Enter password: ");
char[] ch=c.readPassword();
String pass=String.valueOf(ch);//converting char array into string
System.out.println("Password is: "+pass);
}
}
```

Output

```
Enter password:
Password is: 123
```