IOSTREAM

INTRODUCTION

When we use variables and array for storing data inside the programs. We face two Problems:

1) The data is lost either when a variable goes out of scope or when the program is terminated. The storage is temporary.

2) It is difficult to handle large volumes of data using variables and arrays.

- We can overcome these problems by storing data into secondary storage devices. We can store data using concept of files. Data stored in file is often called persistent data.

- A file is collection of related records placed area on the disk. A record is composed of several fields. Field is a group of characters.

- Storing and managing data using file is known as file processing which includes tasks such as creating files, updating files and manipulation of data.

- Reading and writing of data in a file can be done at the level of bytes or characters or fields depending on the requirement of application.java provides capabilities to read and write class object directly.

- The process of reading and writing objects is called **serialization.**

CONCEPT OF STREAM

- In file processing, input refers to the flow of data into a program and **output** means the flow of data out of a program.

- Input to a program may come from the keyboard, mouse, memory, disk a network or another program.

- Output from a program may go to the screen, printer, memory disk, network or another program.

- Input and output share certain common characteristics like unidirectional movement of data, treating data as a sequence of bytes or characters and support to sequential access to data.

- Java uses concept of **stream** to represent ordered sequence of data, a common characteristics shared by all input/output devices.

- A **stream** presents uniform, easy to use, object oriented interface between the program and the input/output devices.

- A stream in java is a path along which data flows (like pipe along which water flows). It has source (of data) and destination (for that data).

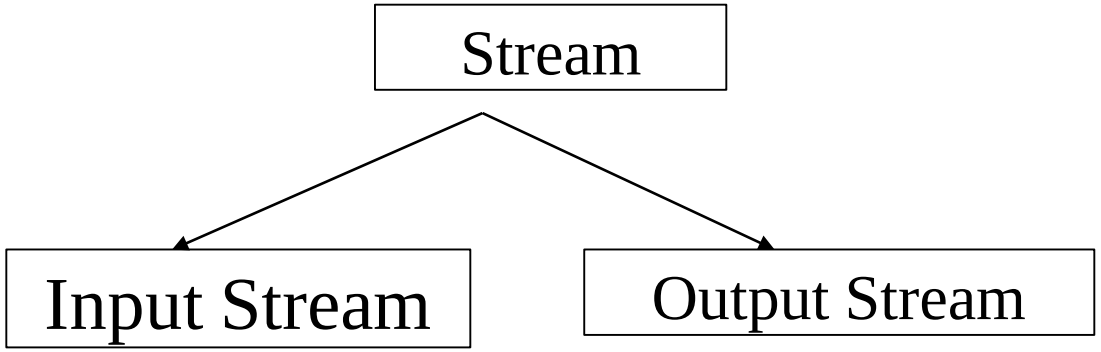
- Both the source and destination may be physical devices or programs or other streams in same program

- The concept of sending data from one stream to another has made streams in java a powerful tool for file processing.

- We can build a complex file processing sequence using a series of simple stream operation.

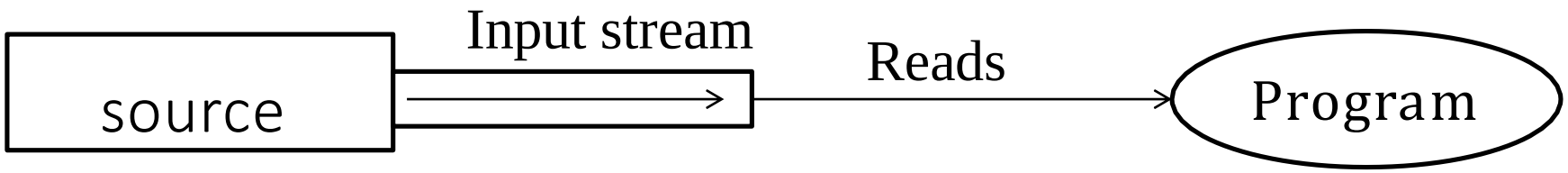
- This feature is used to filter data along the pipeline of streams so that we obtain data in desired format.

- Java stream classified into basic type as follow:

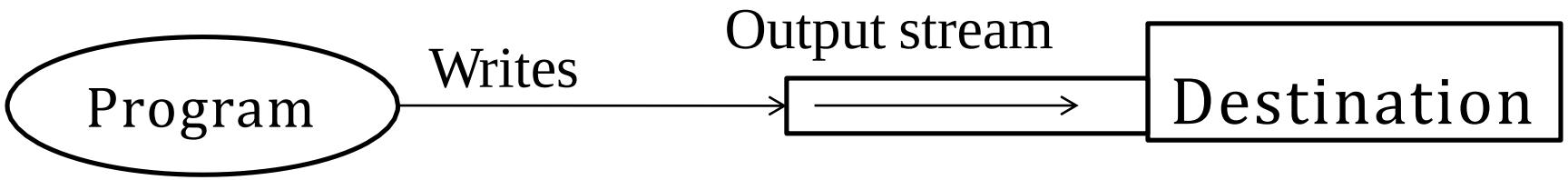


Input stream: it extracts (i.e. reads) data from the source (file) and sends it to the program.

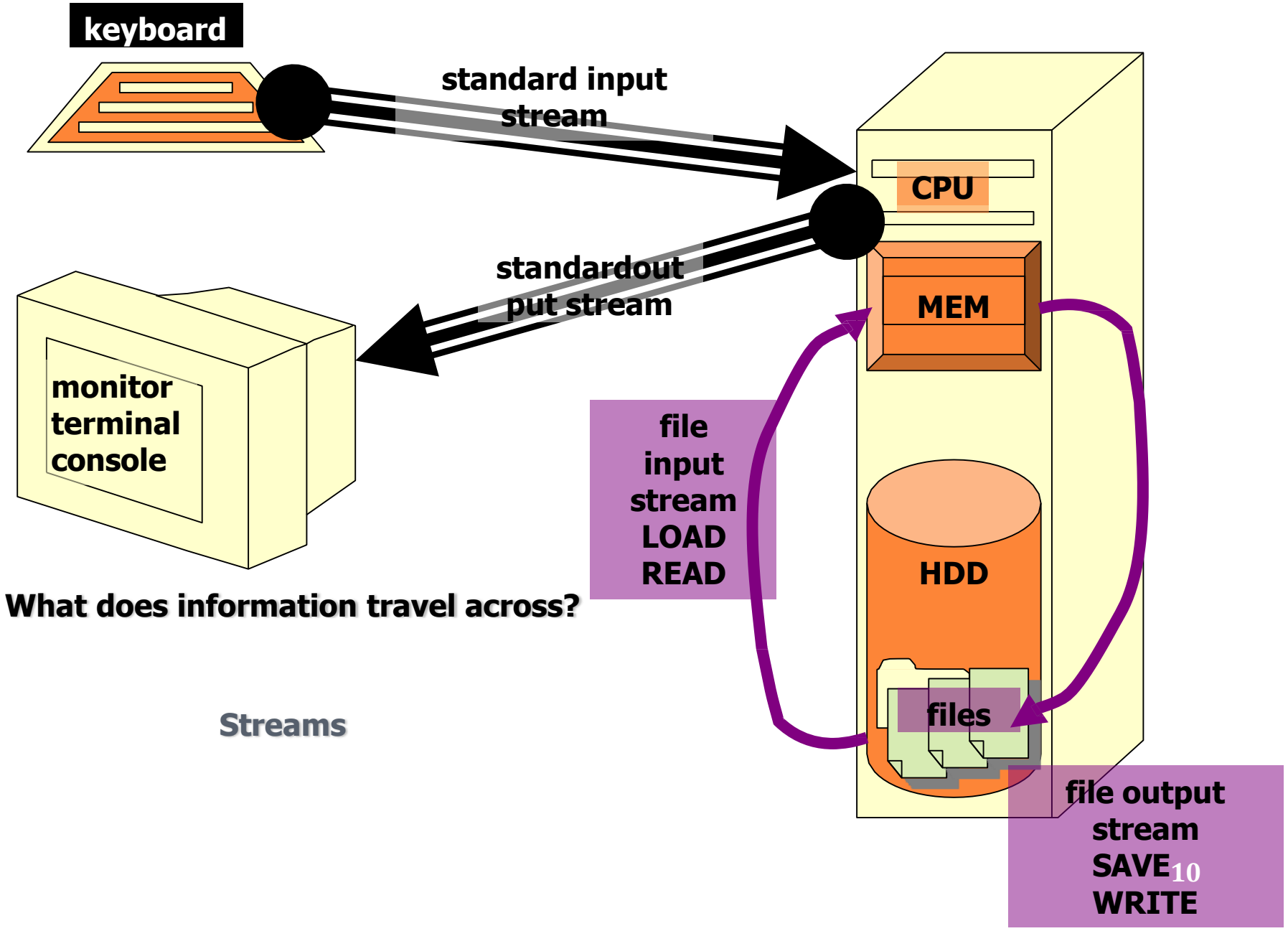
Output stream: it takes data from the program and sends (i.e. writes) it to the destination (file).



(a) Reading data into a program



(b) Writing data to a destination



What does information travel across?

Streams

STREAM CLASSES

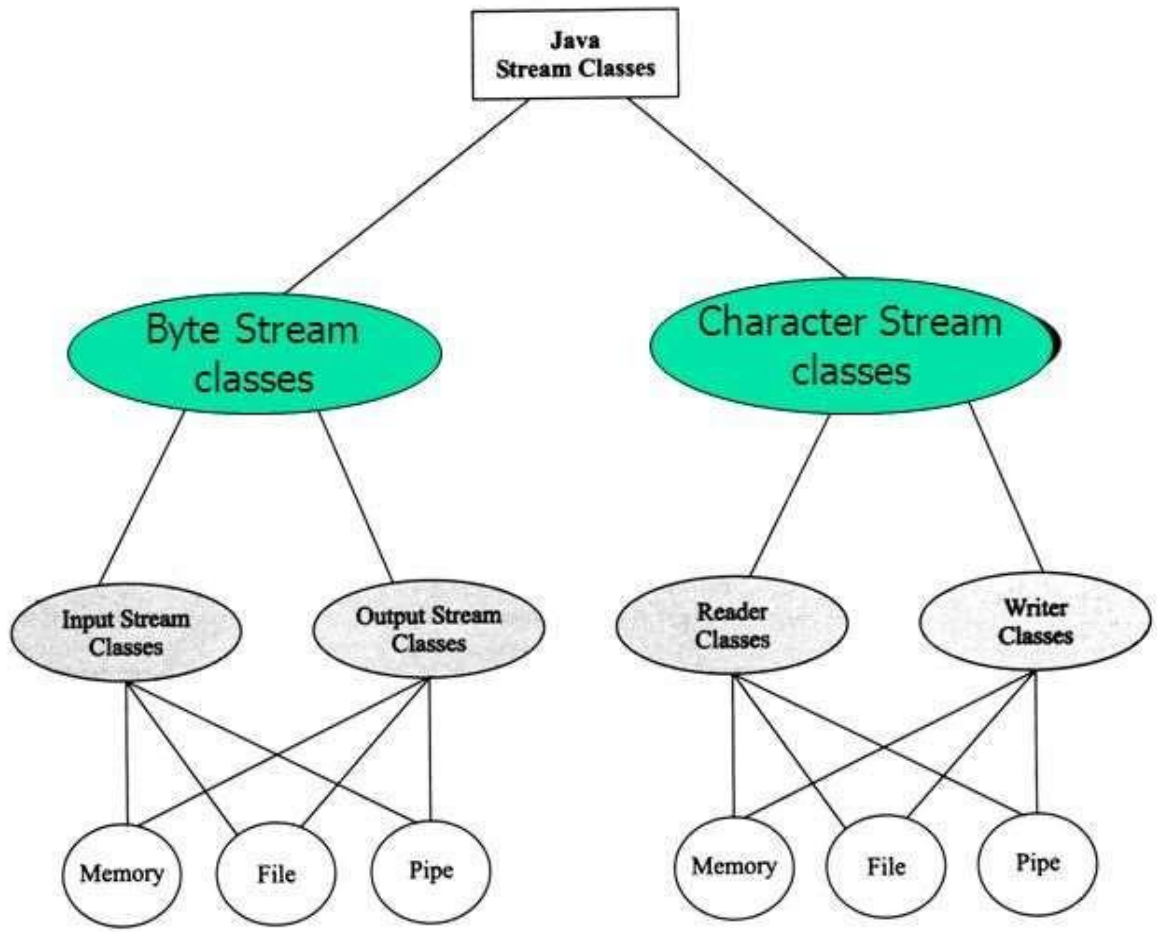
The java.io package contains a large number of stream classes that provide capabilities for processing all types of data.

The classes may be categorized into two groups based on the data type on which they operate.

1. **Byte stream classes** that provides support for handling I/O operations on bytes.

2. **Character stream** classes that provide support for managing I/O operation on characters

Classification of Java Stream Classes



Classification of Java stream classes

- These groups may further be classified based on their functions.
- Byte stream and character stream classes contain specialized classes to deal with input and output operations independently on various type of devices.
- We can also cross-group the streams based on the type of source or destination they from or write to.
- The source (or destination) may be memory, a file or a pipe.

1) Byte stream classes

- Byte stream classes have been designed to provide functional features for creating and manipulating streams and files reading and writing bytes.

- Since the steams are unidirectional, they can transmit bytes in only one direction and, therefore, Java provides two kinds of byte stream classes:

1) input stream classes

2) output stream classes.

Input Stream Classes

- Input stream classes that are used to read 8-bit bytes include super class known as **InputStream** and a number of subclasses for supporting various input-related functions.

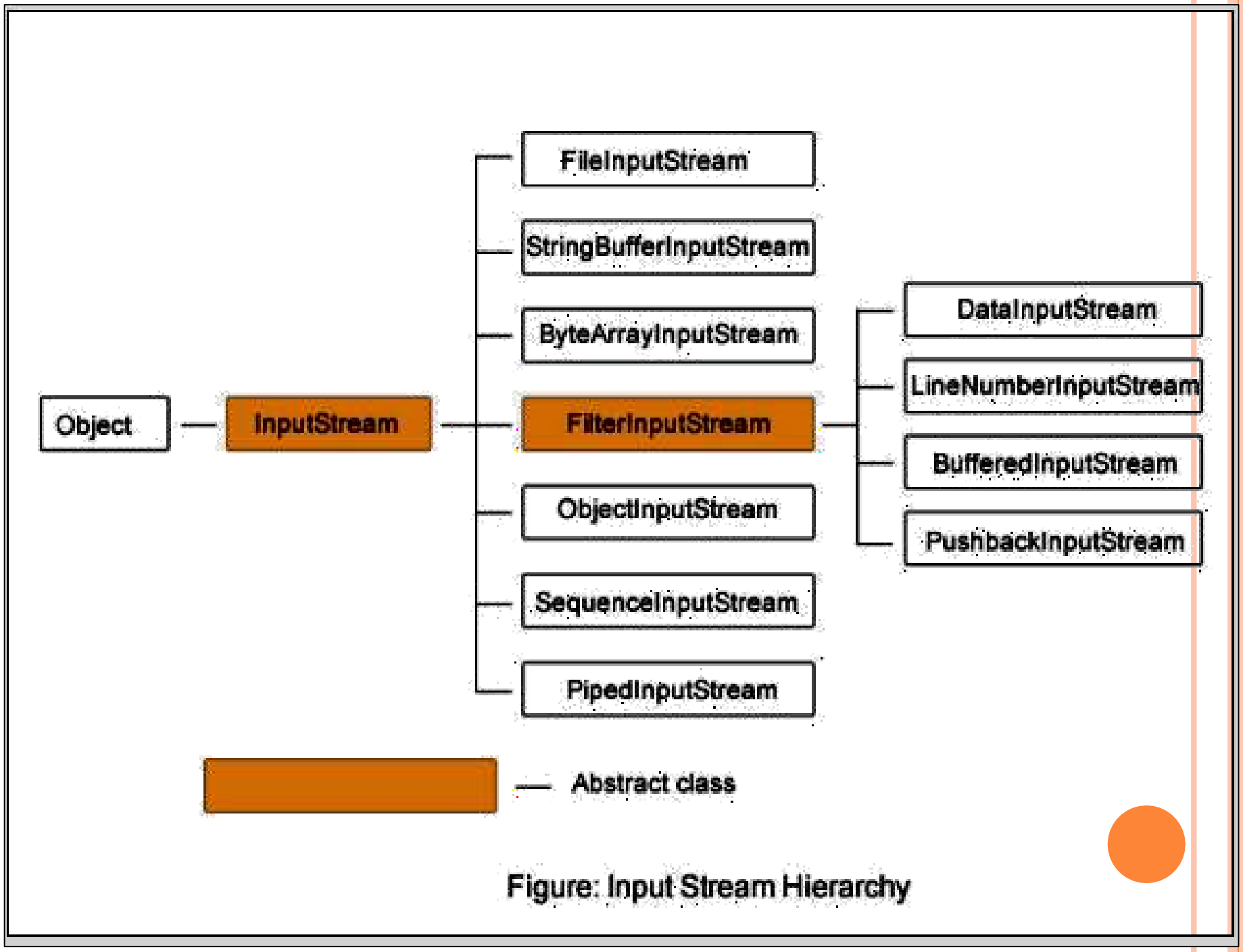


Figure: Input Stream Hierarchy

- The super class InputStream is an abstract class, and, therefore, we cannot create instances of this class.

- Rahter,we must use the subclasses that inherit from this class.

- The InputStream class defines method for performing input functions such as

- **Reading bytes**
- **Closing streams**
- **Marking position in streams**
- **Skipping ahead in a stream**
- **Finding the number of bytes in a stream**

	Method	Description
1	read()	Reads a byte from the input stream
2	read(byte b[])	Reads an array of bytes into b
3	read(byte b[],int n,int m)	Reads m bytes into b starting from nth byte
4	available()	Gives number of bytes available in the input
5	skip(n)	Skips over n bytes from the input stream
6	reset()	Goes back to the beginning of the stream
7	close()	Closes the input stream.

- Note that the class **DataInputStream** extends **FilterInputStream** and implements the interface **DataInput.**

- Therefore, the DataInputStream class implements the methods described in DataInput in addition to using the method of InputStream class.

- The DataInput interface contains the following method.
  - readShort()
  - readInt()
  - readLong()
  - readFloat()

  - readDouble()
  - readLine()
  - readChar()
  - readBoolean()

Output Stream Classes

- Data Output stream classes are derived from the base class Output Stream as shown in figure.

- Like InputStream, the OutputStream is an abstract class and therefore we cannot instantiate it. The several subclasses of the OutputStream can be used for performing the output operations.

- The OutputStream includes methods that are designed to perform the following task:
  - Writing bytes
  - Closing stream
  - Flushing stream

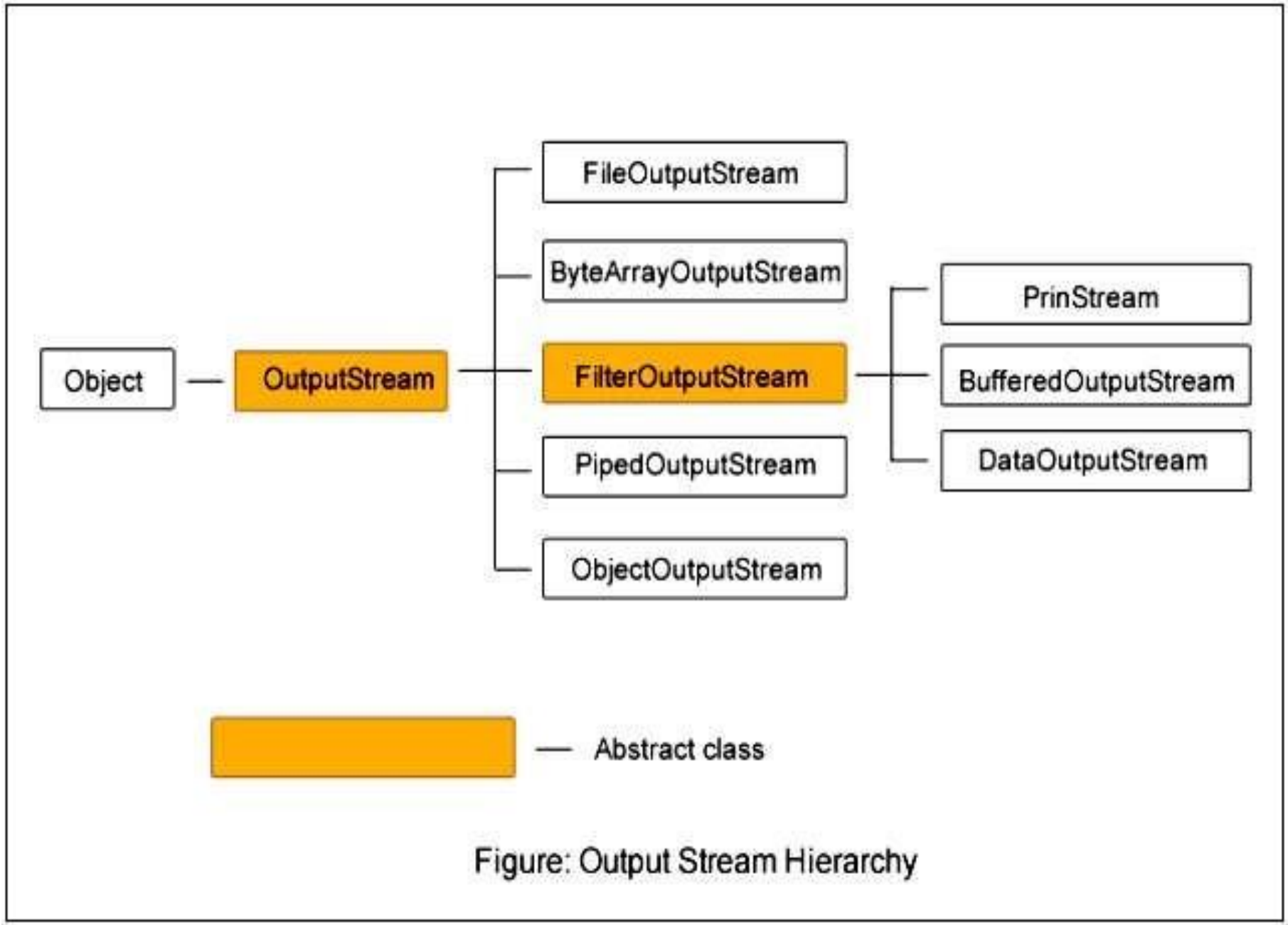


Figure: Output Stream Hierarchy

	Method	Description
1	write()	Writes a byte to the output stream
2	write(byte b[])	Writes all bytes in the array b to the output stream
3	write(byte b[],int n,int m)	Writes m bytes from array b starting from nth byte
4	close()	Close the output stream
5	flush()	Flushes the output stream

- The DataOutputStream, a counterpart of DataOutputStream, implements the interface DataOutputStream .

- Therefore ,implements the following method contained in **DataOutput** interface.
  - writeShort()
  - writeInt()
  - writeLong()
  - writeFloat()

  - writeDouble()
  - writeBytes()
  - writeChar()
  - writeBoolean()

Character stream classes

CHARACTER STREAM CLASSES

- Character stream classes were not a part of the language when it was released in 1995.

- They were added later when the version 1.1 was announced.

- Character streams can be used to read and write 16-bit Unicode characters.

- Like byte streams, there are two kinds of character stream classes

    1) **reader stream classes**
    2) **writer stream classes.**

READER STREAM CLASSES

- **Reader stream** classes are designed to read character from the files. Reader class is the base class for all other classes in this group as shown in figure.

- These classes are functionally very similar to the input stream classes, except input stream use bytes as their fundamental unit of information, while reader stream use characters.

- The Reader class contains method they are identical to those available in the InputStream class, except Reader is designed to handle characters. Therefore, reader classes can perform all the functions implemented by the input stream classes.

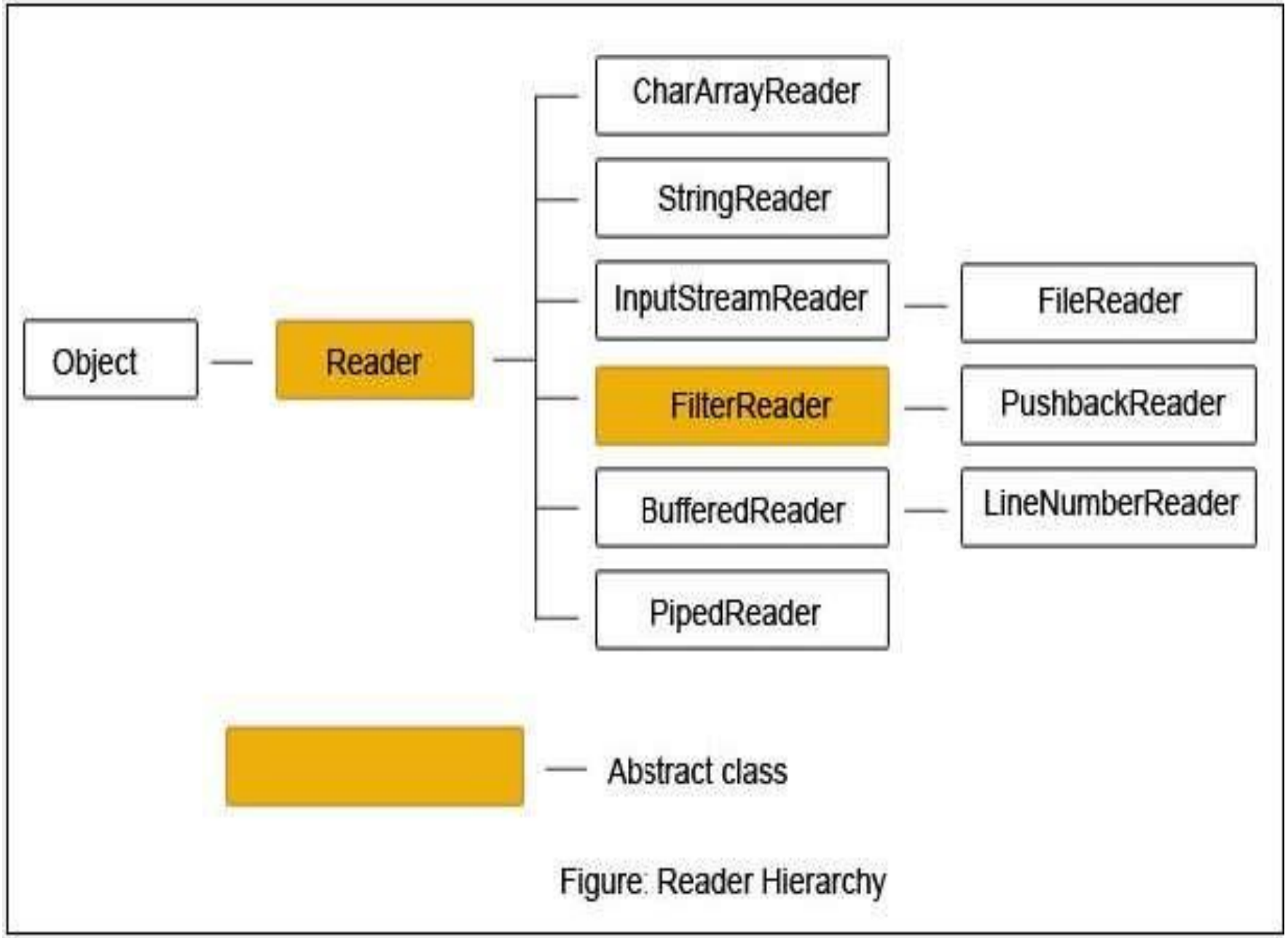


Figure: Reader Hierarchy

WRITER STREAM CLASSES

- Like output stream classes, the writer stream classes are designed to perform all output operations on files.

- Only difference is that while output stream classes are designed to write bytes, the writer stream classes are designed to write characters.

- The **Writer** class is an abstract class which acts as a base class for all other writer stream classes .

- This base class provides support for all output operations by defining method that are identical to those in **OutputStream class.**

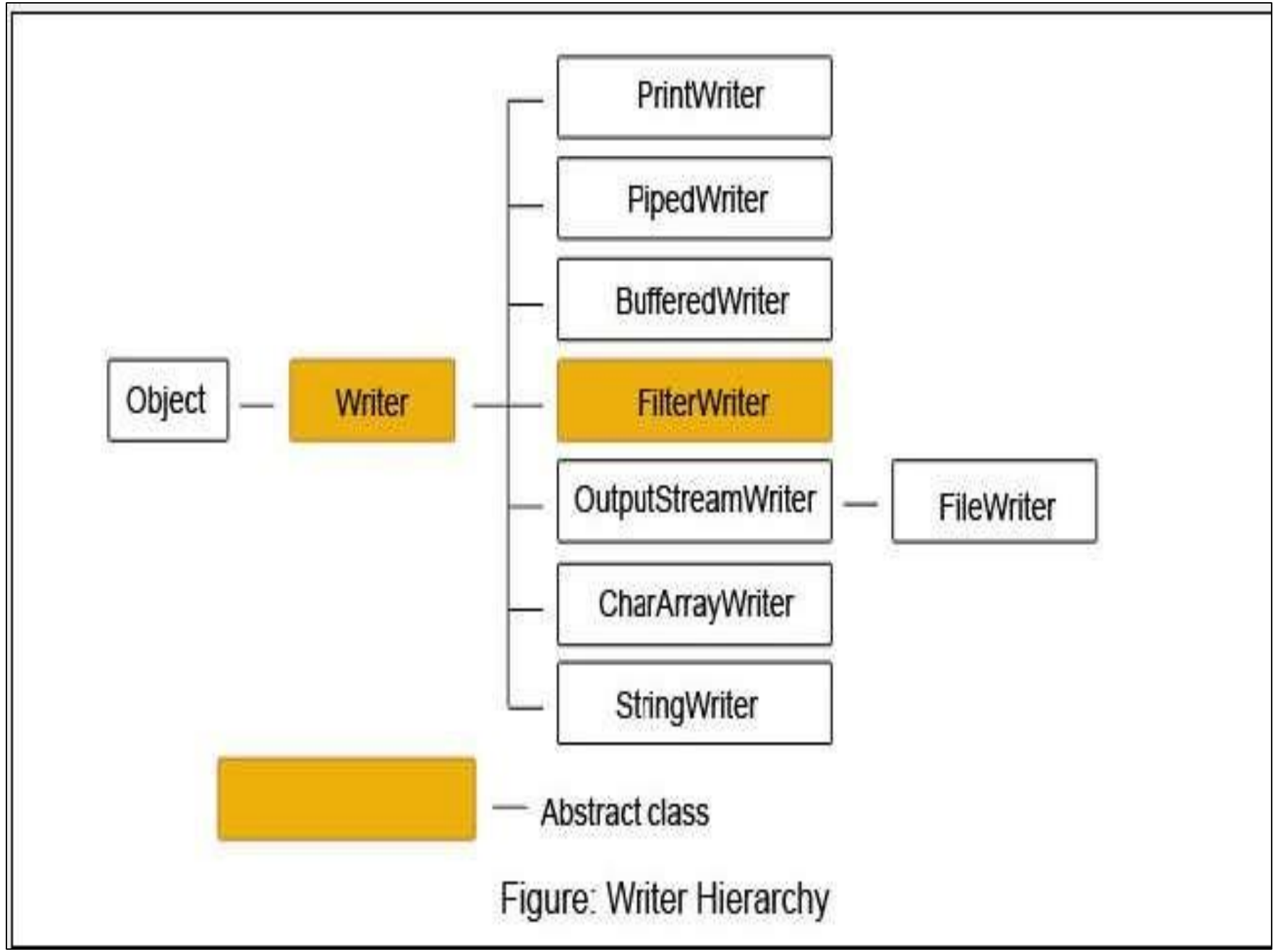


Figure: Writer Hierarchy

USING STREAMS

- We have types of input and output stream classes used for handling both the 16-bit characters and 8-bit bytes. Although all the classes are known as i/o classes, not all of them are used for reading and writing operations only.

- Some perform operations such as buffering, filtering, data conversion, counting and concatenation while carrying out I/O tasks.

- As pointed out earlier, both the character stream group and the byte stream group contain parallel pairs of classes that perform the same of operations but for the different data type.

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

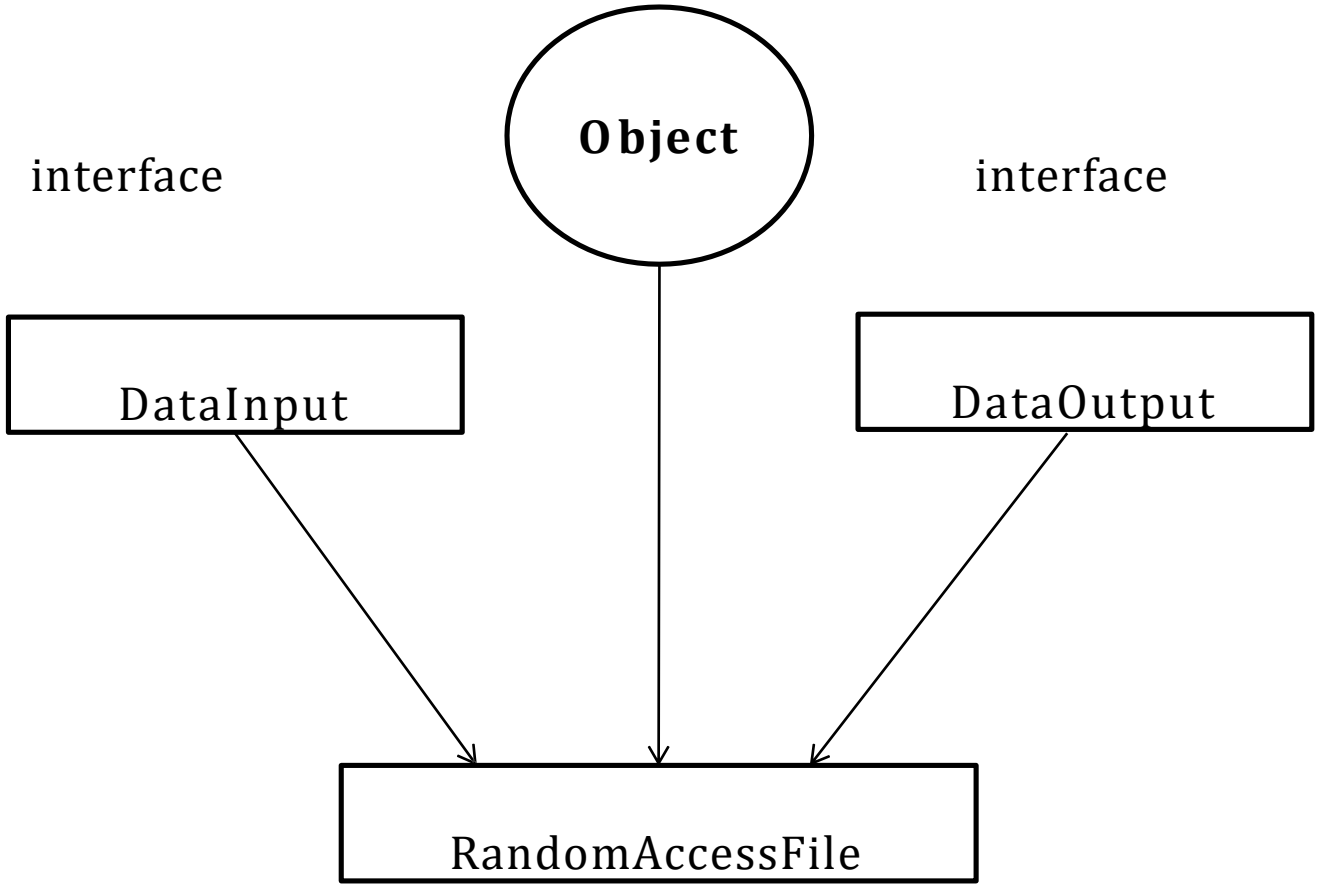
Task	Character Stream Class	Byte Stream Class
Pushing back characters/bytes	PushbackReader	PushbackInputStream
Reading from a pipe	PipedReader	PipedInputStream
Reading from a string	StringReader	StringBufferInputStream
Reading primitive types	(none)	DataInputStream
Performing output operations	Writer	OutputStream
Buffering output	BufferedWriter	BufferedOutputStream
Writing to an array	CharArrayWriter	ByteArrayOutputStream

Task	Character Stream Class	Byte Stream Class
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

OTHER USEFUL CLASS

- The java.io package supports many other classes for performing certain specialized functions. They include among others:
- **RandomAccessFile :**
- The RandomAccessFile enables us to read and write bytes, text and java data types to any location in a file. This class extends **object** class and implements **DataInput** and **DataOutput** interface

- **StreamTokenizer :**
- The class **StreamTokenizer,** a subclass of **object** can be used for breaking up a stream of text from an input text file into meaningful pieces called **tokens.** The behaviour of the **StreamTokenizer** class is similar to that of **StringTokenizer** class (of java.util package) that breaks string into its component tokens.


Object
interface
interface
DataInput
DataOutput
RandomAccessFile

USING THE FILE CLASS

- The java.io package includes a class known as the **File** class that provides support for creating files and directories. The class includes several constructors for instantiating the **File** objects.
- File class provides methods for operations like:
  - Creating a file
  - Opening a file
  - Closing a file
  - Deleting a file
  - Getting the name of a file, Getting the size of a file
  - Checking the existence of a file
  - Renaming a file
  - Checking whether the file is writable
  - Checking whether the file is readable

Input/output Exceptions

- When creating files and performing I/O operations on them, the system may generate I/O related exceptions. The basic I/O related exception classes and their functions:

I/O exception class	Function
EOFException	Signals that an end of file or end of stream has been reached unexpectedly during input
FileNotFoundException	Informs that a file could not be found
InteruuptedIOException	Warns that an I/O operations has been interrupted
IOException	Signals that an I/O exception of some sort has occurred

CREATION OF FILES

- If we want to create and use a disk file, we need to decide the following about the file and its intended purpose:
  - Suitable name for the file.
  - Data type to be stored.
  - Purpose (reading, writing, or updating).
  - Method of creating the file.

- A filename is a unique string of character that helps identify a file on the disk. The length of a filename and the characters allowed are dependent on the OS on which the java program is executed.
- A filename may contain two parts, a primary name and an optional period with extension.

- Data type is important to decide the type of file stream classes to be used for handling the data. We should decide whether the data to be handled is in the form of characters, bytes or primitive type.

- The purpose of using a file must also be decided before using it. For example, we should know whether the file is created for reading only, or both the operations.

- As we know, for using a file, it must be opened first. This is done by creating a file stream and then linking it to the filename.

- A file stream can be defined using the class of **Reader/InputStream** for reading data and **Writer/OutputStream** for writing data.

- The common stream classes used for various i/o operations given in table . The constructors of stream classes may be used to assign the desired filenames to the Stream objects.

Source/destination	Characters	
	Read	Write
Memory	CharArrayReader	CharArrayWriter
File	FileReader	FileWriter
Pipe	PipedReader	PipedWriter

Source/destination	Bytes	
	Read	Write
Memory	ByteArrayInputStream	ByteArrayOutputStream
File	FileInputStream	FileoutputStream
Pipe	PipedInputStream	PipedoutputStream

- There are two ways of initializing the file Stream objects.

- All of the constructors require that we provide the name of the file either **directly** or **indirectly** by giving a file object that has already assigned a file name

DIRECT CREATION

```
FileInputStream fis;
 try
{
//Assign the filename to the file stream object
 fis = new FileInputStream (“test.txt”);
……..
}
 catch (IOException e)
……….
……….
```

INDIRECT CREATION

```
.............
File inFile;
InFile = new file (“test.txt”);
FileInputStream fis;
try
{
//give the value of the file object
//tothe file stream object
Fis=new FileInputStream (inFile);
.............
}
catch (.....)
{
.................
 }
```

The code above includes five tasks:

- Select a filename.
- Declare a file object.
- Give a selected name to the file object declared.
- Declare a file stream object.
- Connect the file to the stream object.

Reading/writing characters

- As pointed out earlier, subclass of Reader and Writer implement streams that can handle characters.

- The two subclasses used for handling characters in files are **FileReader** and **FileWriter**.

- The concept of using file streams and file object for reading and writing characters in program is illustrated in fig:

READING/WRITING BYTES:

- We have used **FileReader** and **FileWriter** classes to read and write 16-bit characters.
- Most file systems use only 8-bit bytes.
- java i/o system provides a number of classes for handling bytes are FileInputStream and FileOutputStream classes.
- We can use them in place of FileReader and file writer

FileOutputStream

- FileOutputStream class is used to write data, byte by byte, to a file.
- The creation of a FileOutputStream class is not dependent on the file that already exists.

Constructors of FileOutputStream

- FileOutputStream(String name) throws FileNotFoundException
  - This creates an output file stream to write to the file with the specified name.
- FileOutputStream(File file) throws FileNotFoundException
  - This creates a file output stream to write to the file represented by the specified *File* object.
- FileOutputStream(FileDescriptor fd) throws FileNotFoundException
  - This creates an output file stream to write to the specified file descriptor, which represents an existing connection to an actual file in the file system.

Constructors of FileOutputStream

- FileOutputStream(String name, Boolean append) throws FileNotFoundException
  - This creates an output file stream to write to the file with the specified name.
- FileOutputStream(File file, Boolean append) throws FileNotFoundException
  - This creates a file output stream to write to the file represented by the specified File object.

FileInputStream

- FileInputStream class is specially designed to work with byte-orented input from files & is derived from InputStream.
- Constructors of FileInputStream are:
  1. FileInputStream(File file)
  2. FileInputStream(FileDescriptor fd)
  3. FileInputStream(String name)

Constructors of FileInputStream

- public FileInputStream(String name)
  - This creates a FileInputStream by opening a connection to an actual file, the file named by the path name *name* in the file system.
- public FileInputStream(File file)
  - This creates a FileInputStream by opening a connection to an actual file, the file named by the File object *file* in the file system.
- public FileInputStream(FileDescriptor fd)
  - This creates a FileInputStream by using the file descriptor *fdObj*, which represents an existing connection to an actual file in the file system.

```java
//program to read data from file
import java.io.*;
class FISDemo {
    public static void main(String args[]) throws IOException,
FileNotFoundException{
        FileInputStream fis=new FileInputStream("hello.txt");
        int data;
        while((data= fis.read())!=-1){
            System.out.print((char)data);
        }
        fis.close();
    }
}
```

OUTPUT:

```
PS C:\Users\HP\Desktop\java_example\Fileread> javac FISDemo.java
PS C:\Users\HP\Desktop\java_example\Fileread> java FISDemo
hello
PS C:\Users\HP\Desktop\java_example\Fileread>
```

```java
//program to write data in file
import java.io.*;
class FOSDemo {
    public static void main(String args[]) throws IOException,
FileNotFoundException{
        FileOutputStream fos=new FileOutputStream("hello.txt");
        String s= "This is file output stream";
        byte b[]=s.getBytes();
        fos.write(b);
        System.out.print("data is saved");
        fos.close();
    }
}
```

OUTPUT

```
PS C:\Users\HP\Desktop\java_example\Fileread> javac FOSDemo.java
PS C:\Users\HP\Desktop\java_example\Fileread> java FOSDemo
data is saved
```

TEXT IN hello.txt flie:

This is file output stream

```java
//program to write data from file to another
import java.io.*;
class Filecopy {
    public static void main(String args[]) throws IOException,
FileNotFoundException{
        FileInputStream fis=new FileInputStream("file1.txt");
        FileOutputStream fos=new FileOutputStream("file2.txt");
        int data;
        while((data= fis.read())!=-1){
            fos.write(data);;
        }
        System.out.print("File copied");
        fis.close();
        fos.close();
    }
}
```

OUTPUT

```
PS C:\Users\HP\Desktop\java_example\Fileread> java filecopy
Error: Could not find or load main class filecopy
Caused by: java.lang.NoClassDefFoundError: Filecopy (wrong name: filecopy)
PS C:\Users\HP\Desktop\java_example\Fileread> java Filecopy
File copied
PS C:\Users\HP\Desktop\java_example\Fileread>
```

FileReader class

- FileReader class can be used to create a character-based stream that reads from a file. This class only defines constructors in addition to the functionality it inherits from InputStreamReader class.
- All constructors of FileReader can throw a FileNotFoundException

Constructors of FileReader class

1. **FileReader(File file)**
   Creates a new FileReader with given File to read (using default charset)
2. **FileReader(FileDescriptor fd)**
   Creates a new FileReader with given FileDescriptor to read (using default charset)
3. **FileReader(File file, Charset charset)**
   Creates a new FileReader with given File to read (using given charset)
4. **FileReader(String filename)**
   Creates a new FileReader with given FileName to read (using default charset)
5. **FileReader(String filename, Charset charset)**
   Creates a new FileReader with given File to read (using given charset)

```
import java.io.*
class FileRead{
        public static void main(String[] args){
                int i=0;
                FileReader fr= new FileReader(“file.txt”);
                while((i=fr.read())!=-1){
                        System.out.print((char)i);
                }
                fr.close();
        }
}
```

FileWriter class

- **Java FileWriter** class of java.io package is used to write **data in character** form to file. Java FileWriter class is used to write character-oriented data to a file. It is a character-oriented class that is used for file handling in java.
- This class inherits from OutputStreamWriter class which in turn inherits from the Writer class.
- The constructors of this class assume that the default character encoding and the default byte-buffer size are acceptable. To specify these values yourself, construct an OutputStreamWriter on a FileOutputStream.
- FileWriter is meant for writing streams of characters. For writing streams of raw bytes, consider using a FileOutputStream.
- FileWriter creates the output file if it is not present already.

Constructors of FileWriter

1. **FileWriter(File file):**
   This constructor creates a FileWriter object given a File object.
2. **FileWriter(File file, boolean append)**
   This constructor creates a FileWriter object given a File object with a boolean indicating whether or not to append the data written.
3. **FileWriter(FileDescriptor fd)**
   This constructor creates a FileWriter object associated with the given file descriptor.
4. **FileWriter(String fileName)**
   This constructor creates a FileWriter object, given a file name.
5. **FileWriter(String fileName, Boolean append)**
   This constructor creates a FileWriter object given a file name with a boolean indicating whether or not to append the data written.

```
import java.io.*
class FileWrite{
        public static void main(String args[]){
        FileWriter fw= new FileWriter(“text.txt”);
        string s= “This is filewriter”;
        char ch[]=s.toCharArray();
        for(int i=0;i<ch.length;i++){ fw.write(ch[i]); }
        fw.close();
        }
}
```

BufferedReader Class

- Java BufferedReader class is used to read the text from a character-based input stream. It can be used to read data line by line by readLine() method. It makes the performance fast. It inherits Reader class
- Constructors:
  - BufferedReader(Reader rd)
  - BufferedReader(Reader rd, int size)

```java
//bufferedReader
import java.io.*;
class BufRead{
    public static void main(String args[]) throws
IOException, FileNotFoundException{
        InputStreamReader io= new
InputStreamReader(System.in);
        BufferedReader br=new BufferedReader(io);
        String name=br.readLine();
        System.out.print("Your name is: "+name);
    }
}
```

OUTPUT

```
PS C:\Users\HP\Desktop\java_example\Fileread> javac BufRead.java
PS C:\Users\HP\Desktop\java_example\Fileread> java BufRead
DIKSHA
Your name is: DIKSHA
PS C:\Users\HP\Desktop\java_example\Fileread>
```

Working with Buffers

- In the java.io package, a file or stream is a sequence of bytes or byte array. The DataInput and DataOutput interfaces can help out, but they work on streams and do not allow byte order to be specified.
- Buffer class allows a byte array to be managed as a buffer with methods to deal with the current read/write position, maximum position and array of bytes

BufferedReader class

- The BufferedReader class provides a buffered character reader class and the readLine() method.
- This class consists of two constructors:
  1. BufferedReader(Reader in)

     It constructs a buffering character input stream
  2. BufferedReader(Reader in, int sz)

     It constructs a buffering character input stream that uses an input buffer of the given size

```
import java.io.*
class Breader{
        public static void main(String[] args){
                FileReader fr= new
FileReader(“file.txt”);
                BufferedReader bread= new
BufferedReader(fr);
                String s;
                while(s= bread.readLine())!=null){
                        System.out.println(s)
                }
                fr.close();
        }
}
```

BufferedWriter class

- Java BufferedWriter class is used to provide buffering for Writer instances. It makes the performance fast. It inherits Writer class. The buffering characters are used for providing the efficient writing of single arrays, characters, and strings.
- Constructors:
  - BufferedWriter(Writer wrt)
  - BufferedWriter(Writer wrt,int size)

```java
//BufferedWriter
import java.io.*;
public class Bufwrite {
public static void main(String[] args) throws
IOException,FileNotFoundException {
    FileWriter writer = new
FileWriter("C:/Users/HP/Desktop/java_example/Fileread/file.txt");

    BufferedWriter buffer = new BufferedWriter(writer);
    buffer.write("Welcome to javaTpoint.");
    buffer.close();
    System.out.println("Success");
    }
}
```