Managing Input/Output Files in Java

Dr. Firoz Ahmed

Professor

Department of ICE, RU

July 1st, 2021

Outlines of Presentation

- Introduction
- Concept of streams
- Stream classes
- Byte stream class
- Character stream class
- Other useful I/O classes
- Using the file class

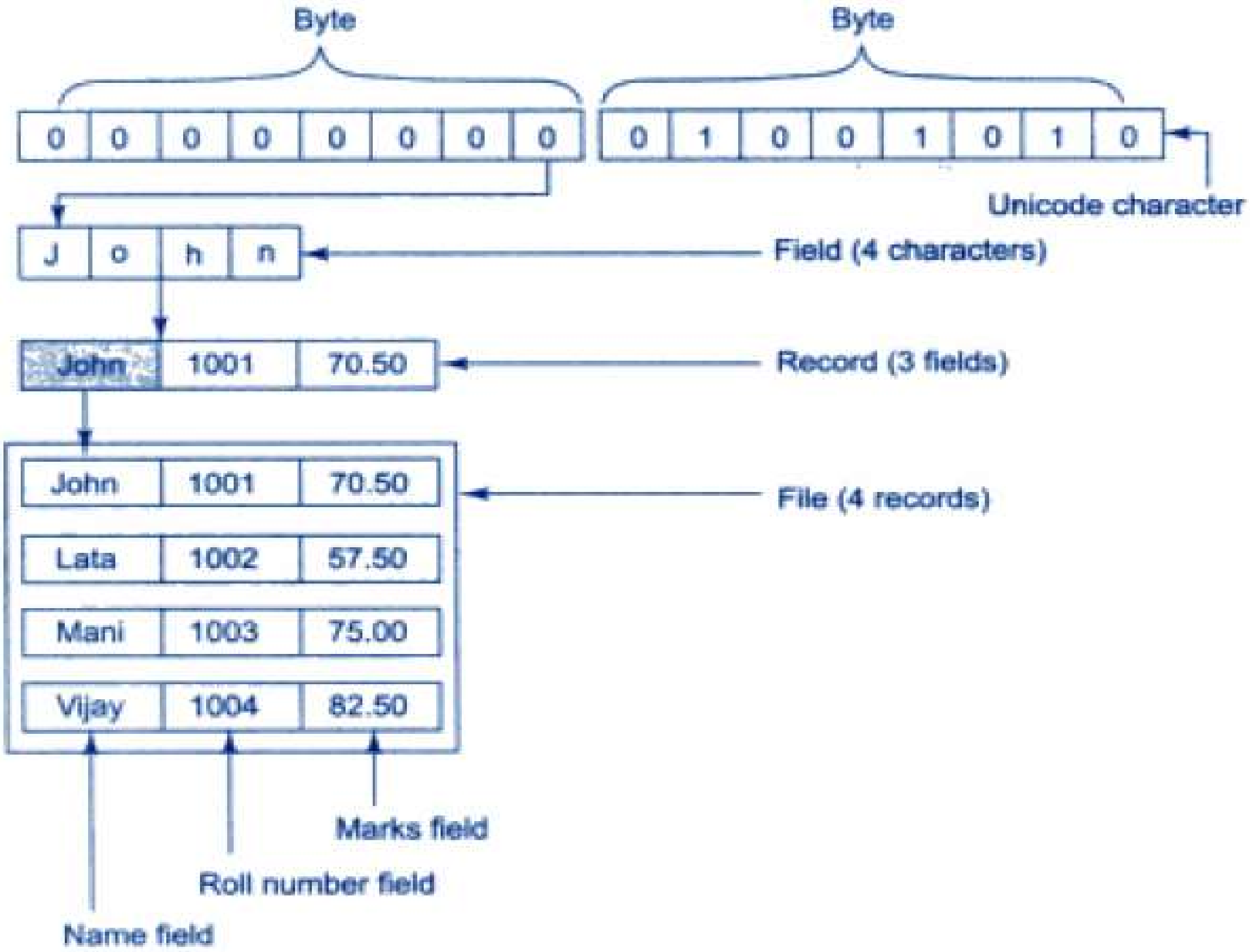
Introduction

- So far we have used variables and arrays for storing data inside the programs
- This approach poses the following problems
  - The data is lost either when a variable goes out of scope or when the programs terminated, that is the storage is temporary
  - It is difficult to handle large volume of data using variables and arrays
- We can overcome these problem by storing data on secondary storage device such as floppy or hard disk
- The data is stored in these devise using the concept of files

Introduction

- A file is a collection of related records places in a particular area on a disk
- A record is composed of several fields and a field is a group of characters
- Characters in Java are Unicode characters composed of two bytes, each byte containing eight binary digits, 1 or 0

Introduction



Byte

0 0 0 0 0 0 0 0

Byte

0 1 0 0 1 0 1 0

Unicode character

J o h n

Field (4 characters)

John	1001	70.50

Record (3 fields)

John	1001	70.50
Lata	1002	57.50
Mani	1003	75.00
Vijay	1004	82.50

File (4 records)

Marks field

Roll number field

Name field

Data representation in Java files

Introduction

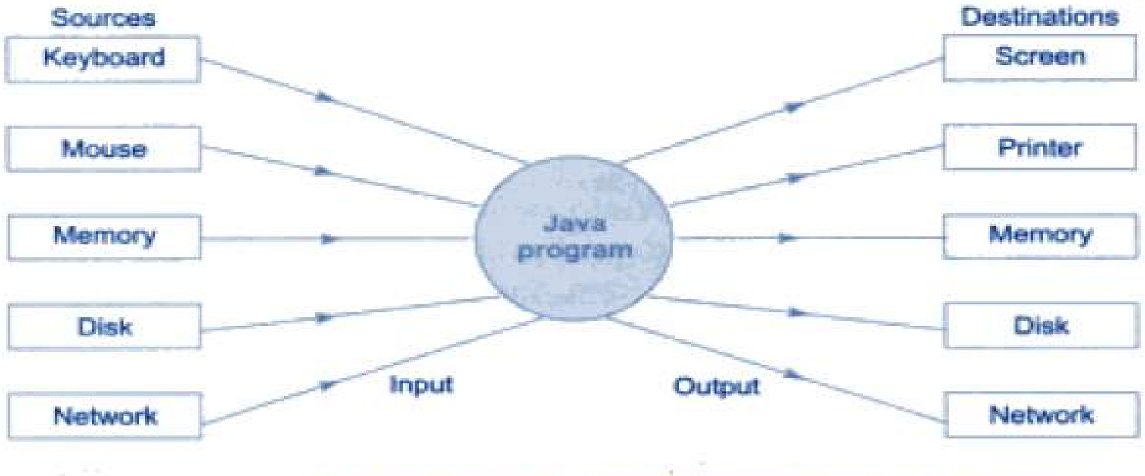
- Storing and managing data using files is known as file processing
- File processing includes the following task
  - Creating files
  - Updating files and
  - Manipulation of data
- Java supports many powerful features for managing input and output of data using files

Introduction

- 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 a particular applications
- Java also provides capabilities to read and write class object directly
- The process of reading and writing object is called object serialization

Concept of Streams

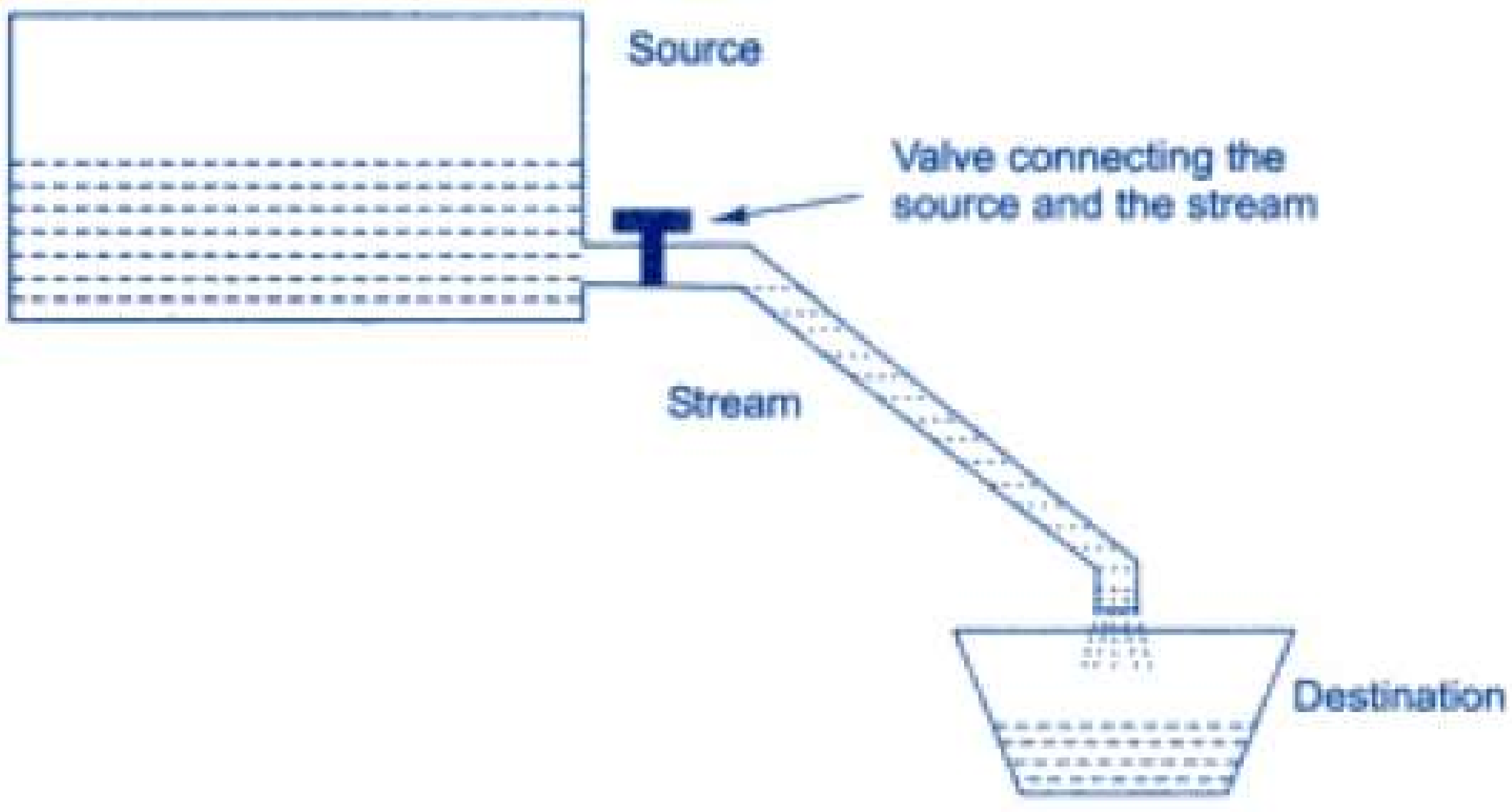
- In file processing..
  - Input refers to the flow of data into a program and
  - Output means the flow of data out of a program
- Input to a program may come form the key board, the mouse, the memory, the disk, a network or another program
- Similarly output from a program may go to the screen, the printer, the memory, the disk, a network, or another program



Relationship of Java program with I/O devices

Concept of Streams

- Java uses the concept of steams to represent the ordered sequence of data



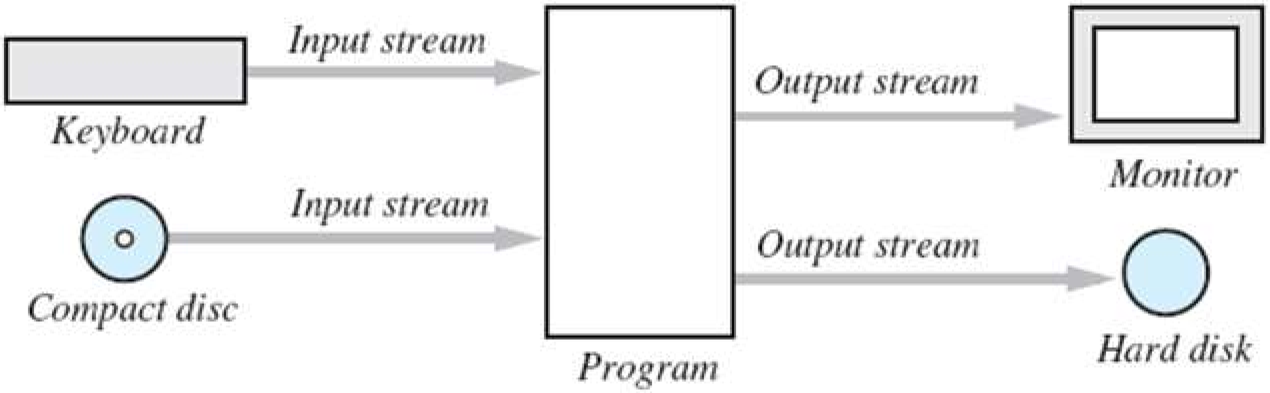
Conceptual view of a stream

Concept of Streams

- A streams presents a uniform, easy-to-use, object oriented interface between the program and the input/output device
- A stream in Java is a path along which data flows (like a river or a pipe along which water flows)
- It has a source (of data) and destination (for that data)
- Both the source and destination may be physical devices or programs or other steams in the same program

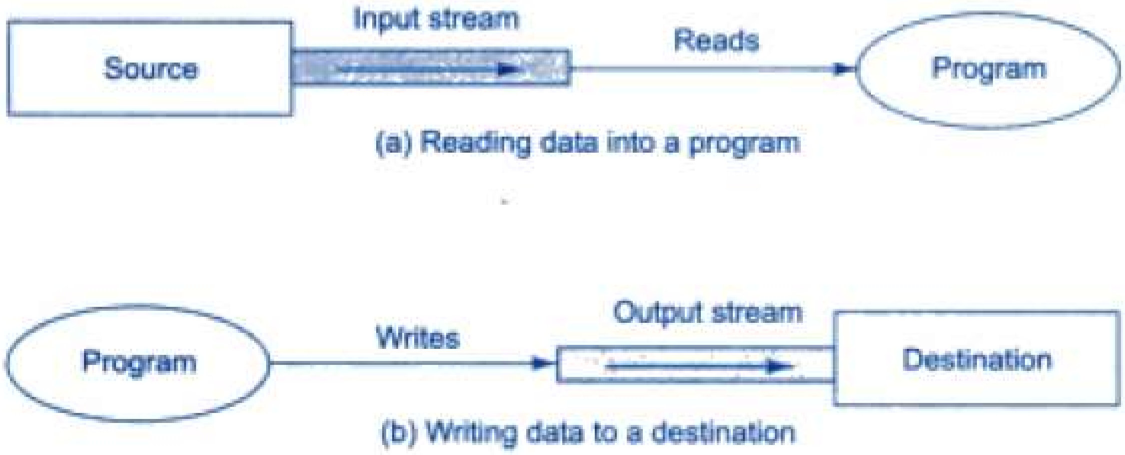
Concept of Streams

- ***Stream*** is an object that either delivers data to its destination (ex: screen, file) or that takes data from a source (ex: keyboard, file)
  - It acts as a buffer between the data source and destination
- Streams are channel for sending and receiving information in java program
  - When sending a stream of data, it is said that it is writing a stream
  - When receiving a stream of data, it is said that it is reading a stream



Concept of Streams

- Java streams are classified into two basic types
  - Input stream
    - An input stream extracts (i.e reads) data from the source (file) and sends it to the program
  - Output stream
    - Similarly, an output stream takes data from the program and sends (i.e write) it to the destination file

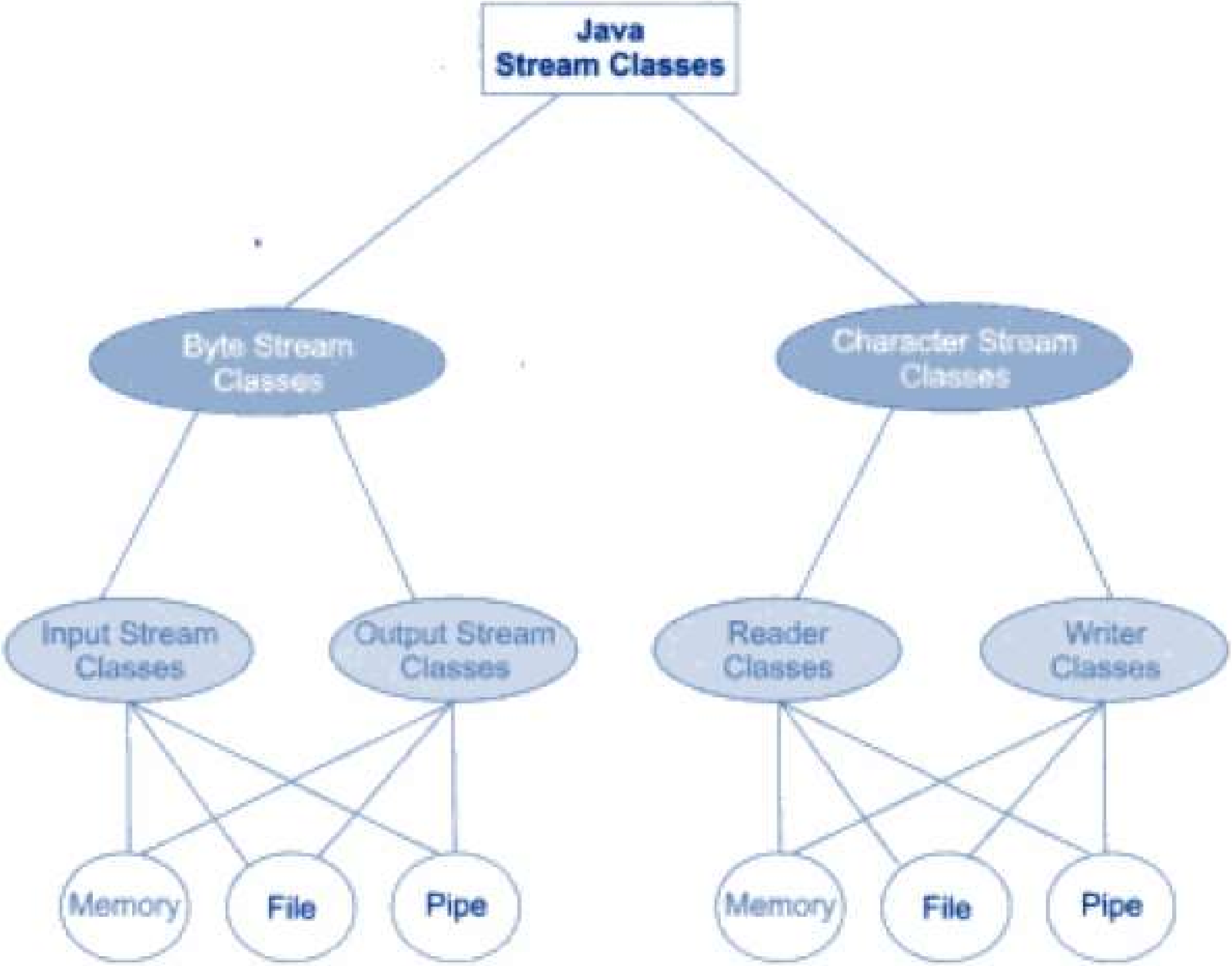


Using input and output streams

Stream Classes

- The **Java.io** package contains a large number of stream classes
  - It provides capabilities for processing all types of data
- These classes may be categorized into two groups based on the data types on which they operate
  - Byte stream class that provide support for handling I/O operation on bytes
  - Character stream classes that provide support for managing I/O operation on character
- These two groups may further be classified based on their purpose

Stream Classes



Classification of Java stream classes

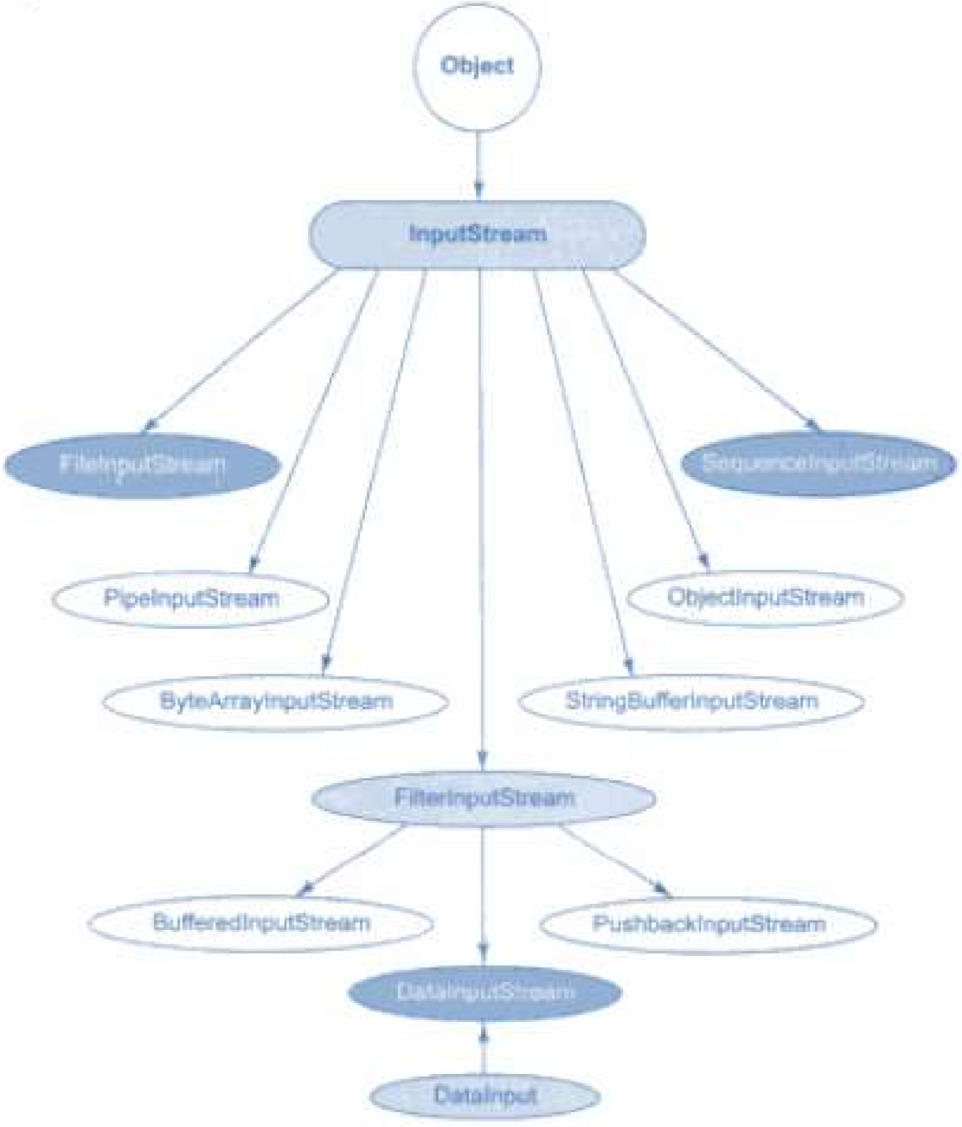
Byte Stream Classes

- Byte stream classes have been designed..
  - To provide functional features for creating and manipulating streams and files for reading and writing bytes
- Since the streams are unidirectional, they can transmit bytes in only one direction
- Therefore Java provides two kind of byte stream classes
  - Input stream classes and
  - Output stream classes

Input Stream Classes

- Input stream classes are used to read 8-bit bytes include a super class known as *InputStream*
- It is also used to read 8-bit bytes include a number of subclasses for supporting various input related functions
- The super class *InputStram* is abstract class, and therefore, we cannot create instances of this class
- Rather, we must use the subclasses that inherit from this class

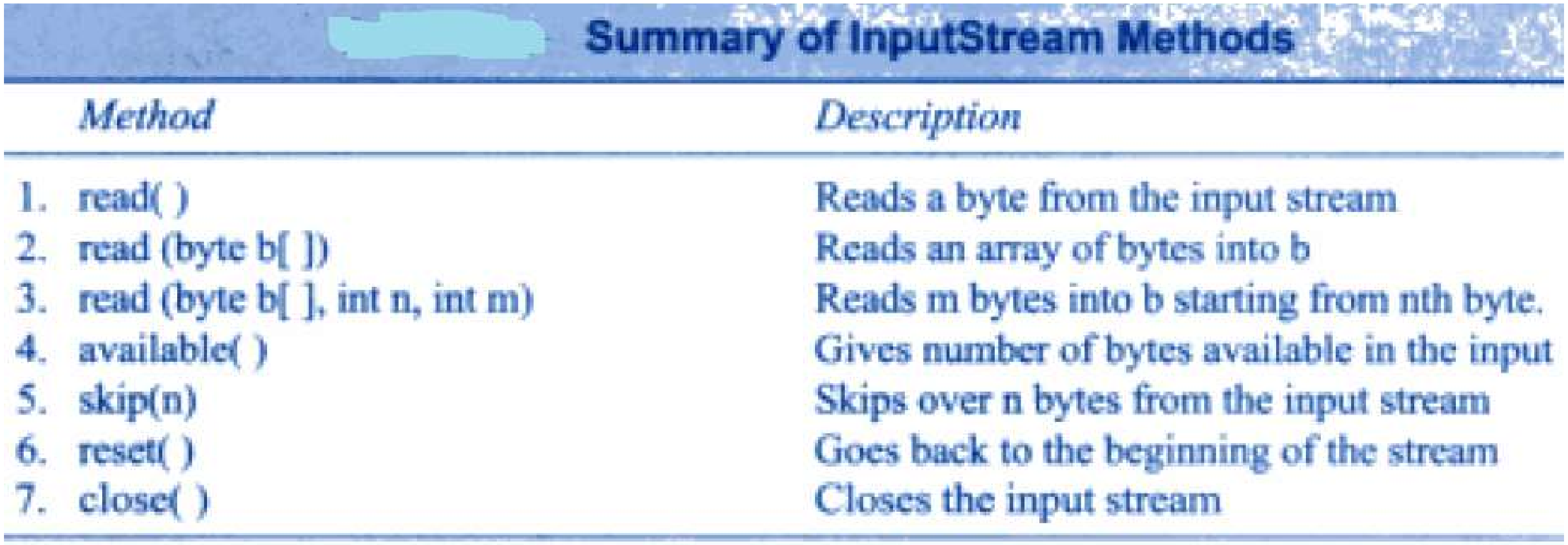
Input Stream Classes



Hierarchy of input stream classes

Input Stream Classes

- The *InputStream* defines methods 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

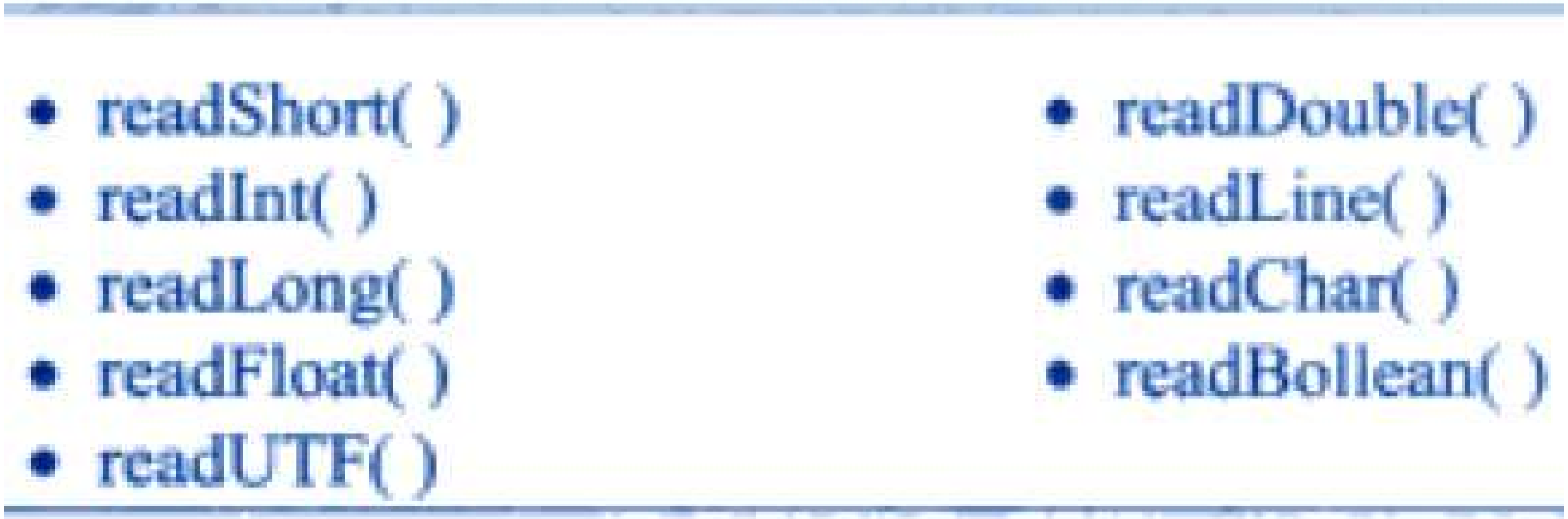


Summary of InputStream Methods

	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

Input Stream Classes

- The class *DataInputStream* extends *FilterInputStream* and implements the interface *DataInput*
- Therefor, the *DataInputStream* class implements the methods describe in *DataInput* in addition to using the methods of *InputStream* Class
- The *DataInput* interface contains the following methods

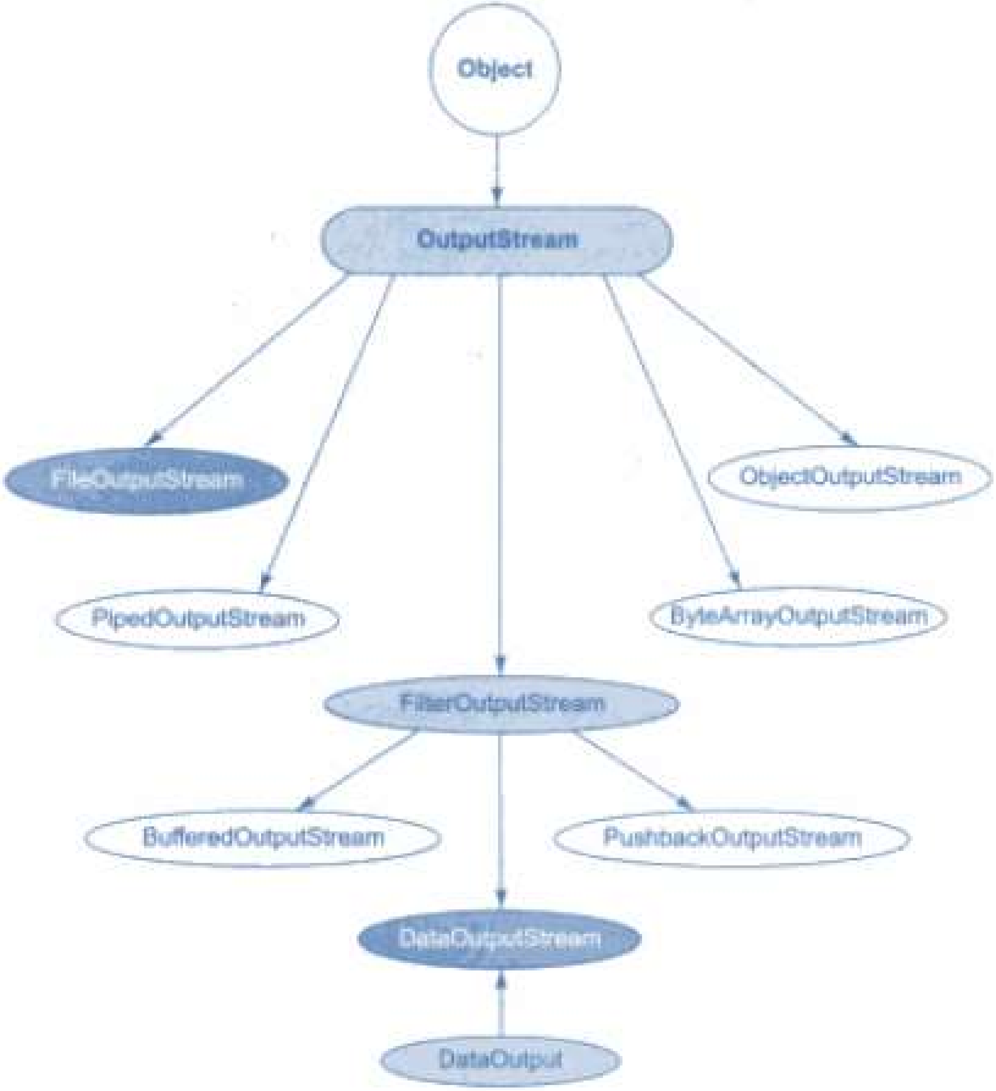


- readShort( )
- readInt( )
- readLong( )
- readFloat( )
- readUTF( )
- readDouble( )
- readLine( )
- readChar( )
- readBollean( )

Output Stream Classes

- Output stream classes are derived from the base class *OutputStream*
- 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

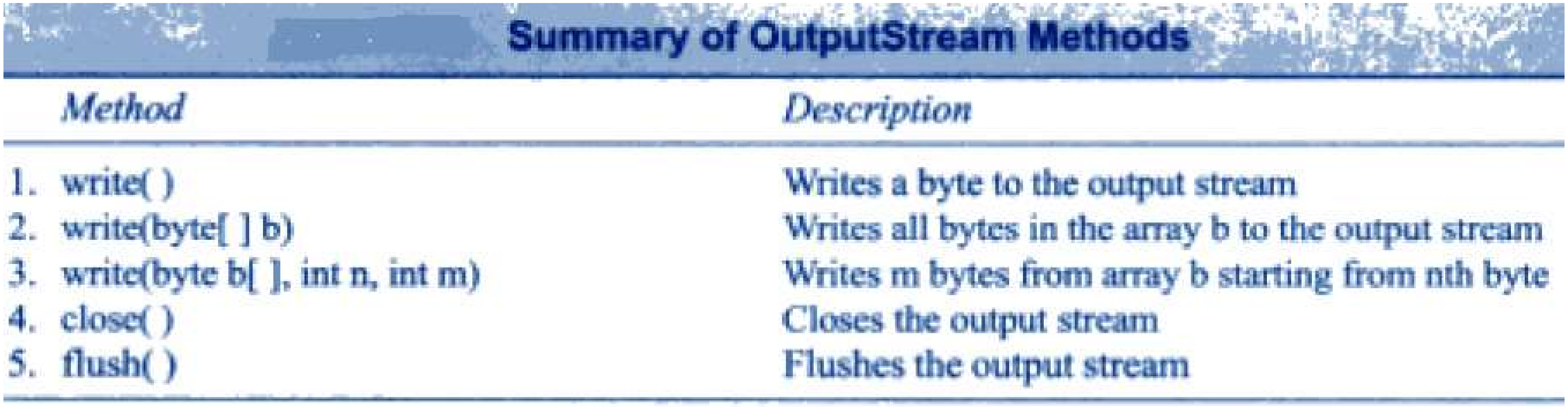
Output Stream Classes



Hierarchy of output stream classes

Output Stream Classes

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



Summary of OutputStream Methods

	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( )	Closes the output stream
5.	flush( )	Flushes the output stream

Output Stream Classes

- The *DataOutputStream*, a counterpart of *DataInputStream*
- Implements the interface *DataOutput*, therefore implements the following methods contained in *DataOutput* interface
  - writeShort()
  - writeInt()
  - writeLong()
  - writeFloat()
  - writeUTF()
  - writeDouble()
  - writeBytes()
  - writeChar()
  - writeBoolean()

FileInput/FileOutput Stream

- The *FileInputStream* class creates an *InputStream* that use to read bytes from a file

  Two most common constructors

  - FileInputStream(String *filepath*) //*filepath* is the full name of a file
  - FileInputStream(File *fileobj*) //*fileobj* is a File object that describes the file

- *FileOutputStream* creates an *OutputStream* that use to write bytes to a file

  Most common constructors

  - FileOutputStream(String *filepath*)
  - FileOutputStream(File *fileobj*)
  - FileOutputStream(String *filepath, boolean append*)// if append is true, the file is opend in append mode

ByteArrayInput/ByteArrayOutput Stream

- *ByteArrayInputStream* is an implementation of an input stream that uses a byte array as the source

  Two construcor

  - *ByteArrayInputStream (byte array[]) //array is the input*
  - *ByteArrayInputStream (byte array[], int start, int numBytes)* //it begins with the character at the index specified by *start* and is *numbBytes* long

- *ByteArrayOutputStream* is an implementation of an output stream that uses a byte array as the destination

  Two construcor

  - *ByteArrayOutputStream()*
  - *ByteArrayOutputStream(int numBytes)*

Demonstration

```
import java.io.*;
public class ByteArrayInputStreamDemo {

public static void main(String args[]) throws IOException {

String tmp = "abcdefghijklmnoprstuvwxyz";
byte b[] = tmp.getBytes();

ByteArrayInputStream input1 = new ByteArrayInputStream(b);

ByteArrayInputStream input2 = new ByteArrayInputStream(b);
```

Example

```
import java.io.*;
public class wc {

public static void main(String args[])
throws IOException {

String tmp = "abc";
byte b[] = tmp.getBytes();

ByteArrayInputStream in = new
ByteArrayInputStream(b);

for (int i=0;i<2;i++){
int c;
```

```
while ((c = in.read())!=-1){
                    if (i==0){
System.out.print((char)c);
}
else {
System.out.print(Character.toUpper
Case((char)c));
}
}
System.out.println();
in.reset();
}
}
}
```

PipedInput/Output Stream

- Pipes are typically used when two different processes need to communicate large amounts of data in a synchronized fashion
- Class *PipedInputStream* requires to connect it with another pipe, an instance of *PipedOutputStream*
- A *PipedOutputStream* must be connected to a *PipedInputStream* and *vice versa*

Their respective constructors are as follows

public PipedInputStream (PipedOutputStream src) throws IOException

public PipedOutputStream (PipedInputStream snk) throws IOException

ObjectInput/OutputStream

- Object input and output streams support object serialization

SequenceInputStream

- Class *SequenceInputStream* allows to concatenate multiple InputStreams
- A *SequenceInputStream* constructor uses either a pair of *InputStream* or an Enumeration of *InputStream* as its arguments

Constructors

public SequenceInputStream(InputStream s1, InputStream s2)

public SequenceInputStream (Enumeration e)

FilterInput/OutputStream

- *Filter streams* are simply wrappers around underlying input or output streams that transparency provide some extended of functionality
- These streams are typically accessed by methods that are expecting a generic stream, which is a super class of the filtered stream
- Filter byte streams are *FilterInputStream* and *FilterOutputStream*

Constructor

- FilterOutputStream(OutputStream *os*)
- FilterInputStream(OutputStream *is*)

BufferedInput/OutputStream

- Class *BufferedInputStream* enhances the bare-bones *InputStream* by adding to it a buffer of bytes, which usually improves reading performance significantly
  - *BufferedInputStream(*InputStream *InputStream*)
  - *BufferedInputStream(*InputStream *InputStream, int bufSize*) //the size of the buffer is passed in *bufSize*

- A *BufferedOutputStream* is similar to any OutputStream with the exception of an added flush() method that is used to ensure that data buffers as physically written to the actual output device
  - *BufferedOutputStream*(OutputStream *OutputStream*)
  - *BufferedOutputStream*(OutputStream *OutputStream, int bufSize*)

PushbackInputStream

- **Classes** ***PushbackInputStream*** - adds ability to "push back" or "unread" one byte from stream

Constructor

PushbackInputStream(InputStream *InputStream*)

PushbackInputStream(InputStream *InputStream, int numBytes*)

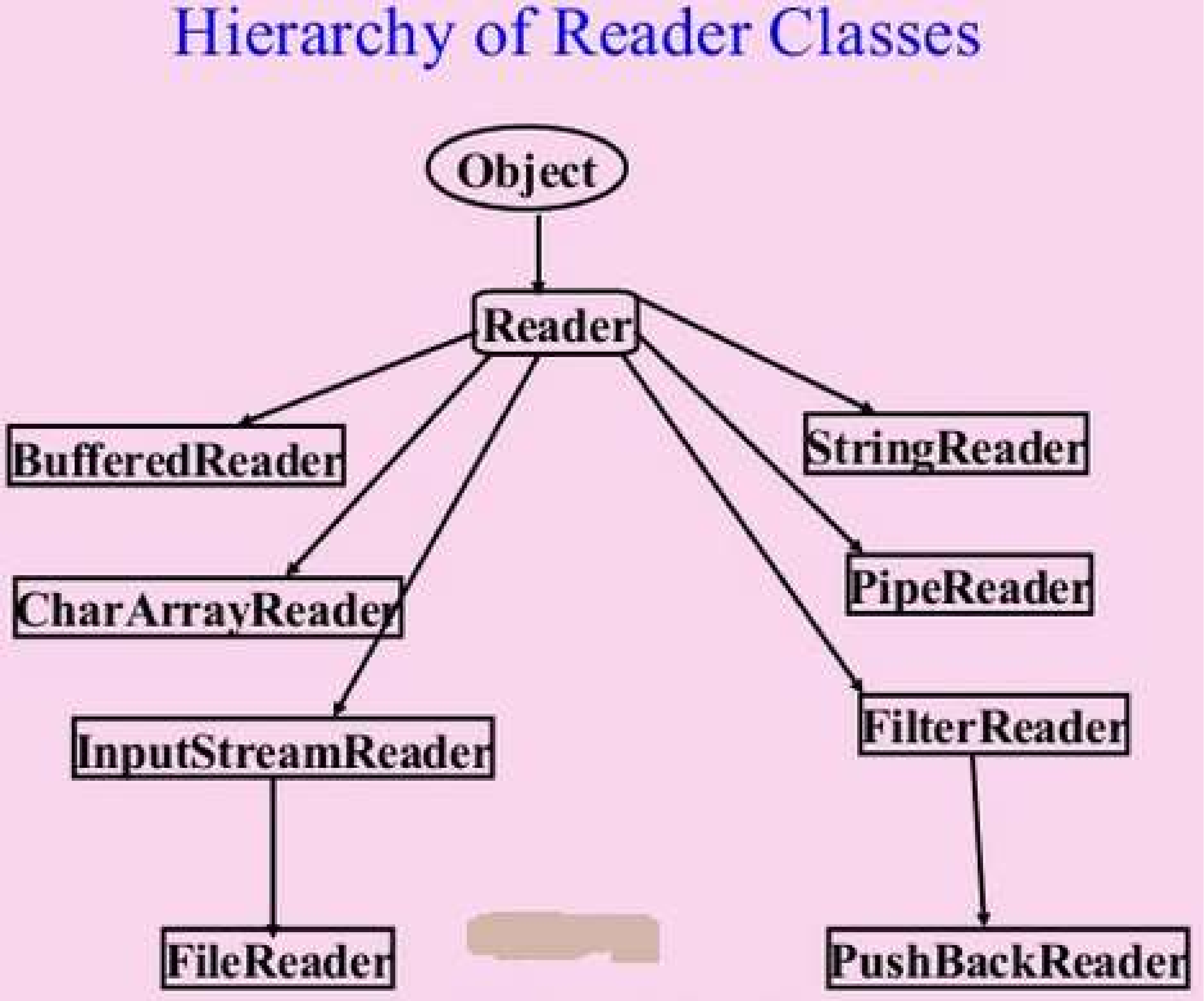
Character Stream Classes

- Character steam 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 kind of character stream classes
  - Reader stream classes
  - Writer stream classes

Reader Stream Classes

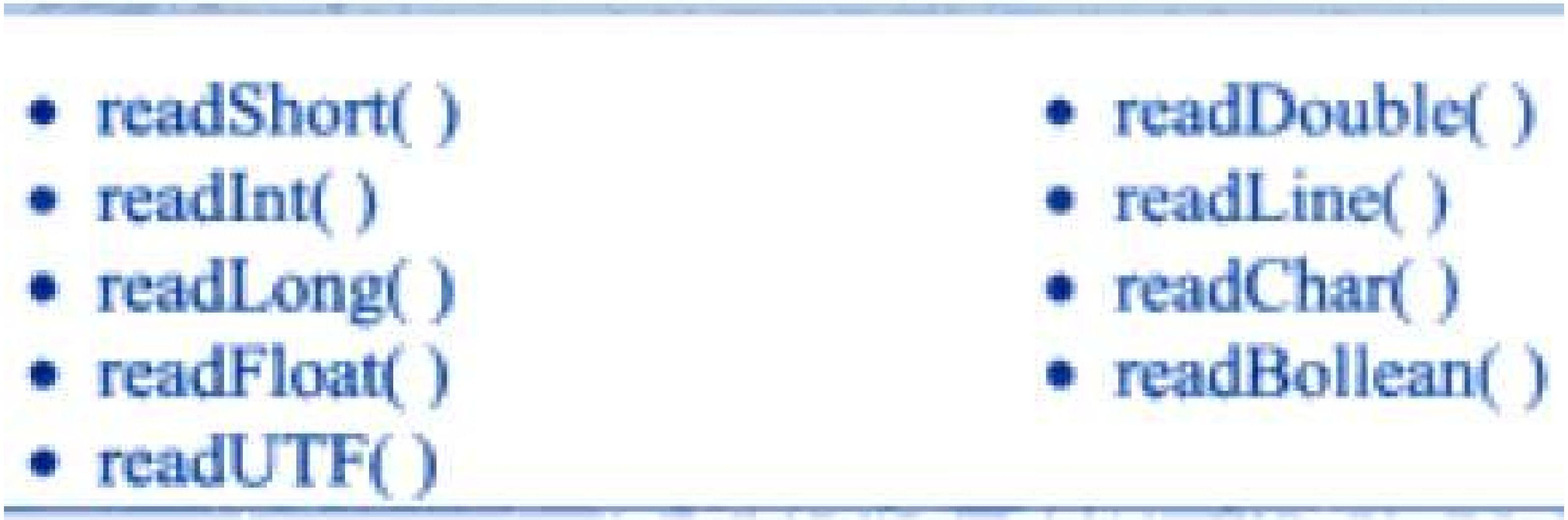
- Reader stream classes are designed to read character from the files
- Reader class is the base class for all the classes in the group
- These classes are functionally very similar to the input stream classes
- Except input streams use bytes as their fundamental unit of information, while reader streams use character

Reader Stream Classes



Reader Stream Classes

- The reader class contains methods that are identical to those available in the *InputStream* class
- Except reader is designed to handle characters
- Therefore reader classes can perform al the function implemnted by the input stream classes

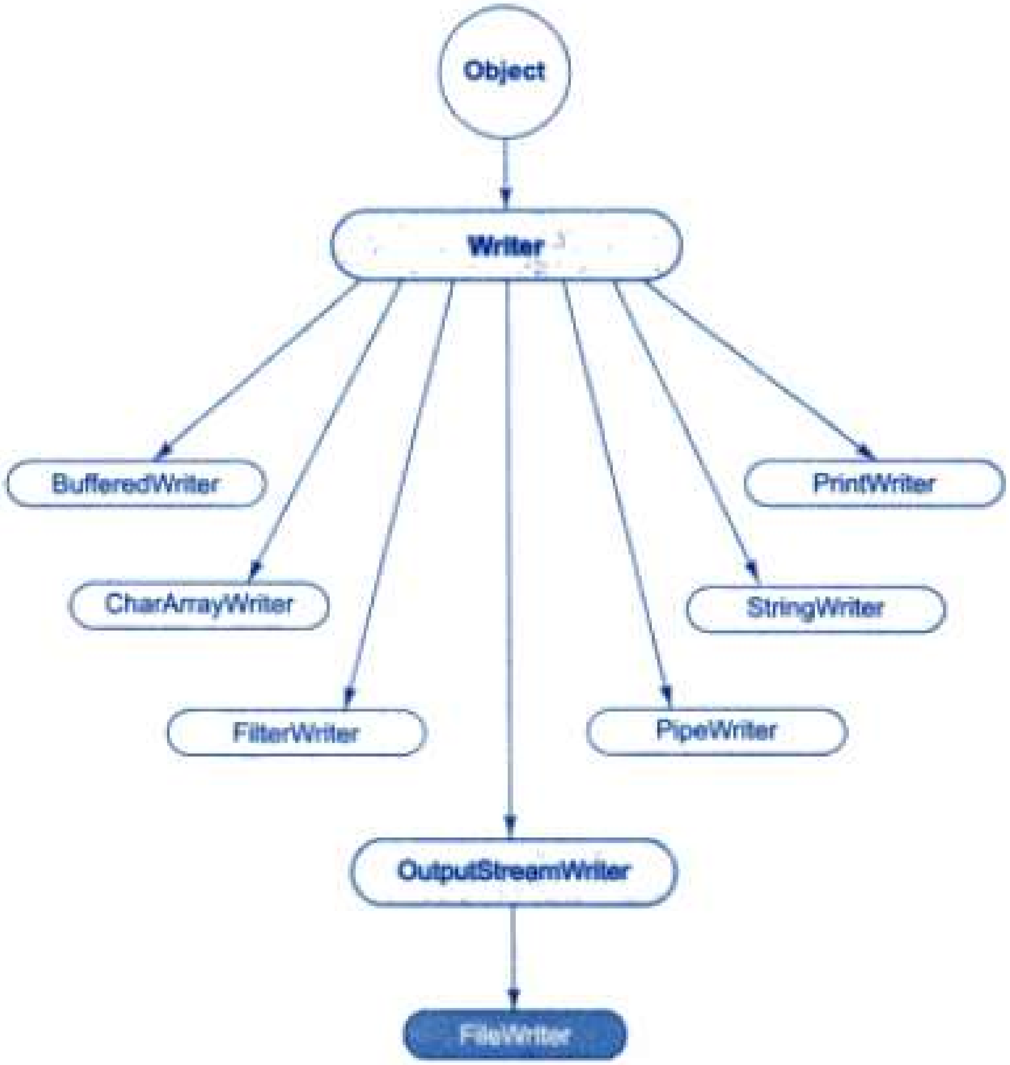


- readShort( )
- readInt( )
- readLong( )
- readFloat( )
- readUTF( )
- readDouble( )
- readLine( )
- readChar( )
- readBollean( )

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 character
- The write class is an abstract class which acts as a base class for all the other writer stream classes

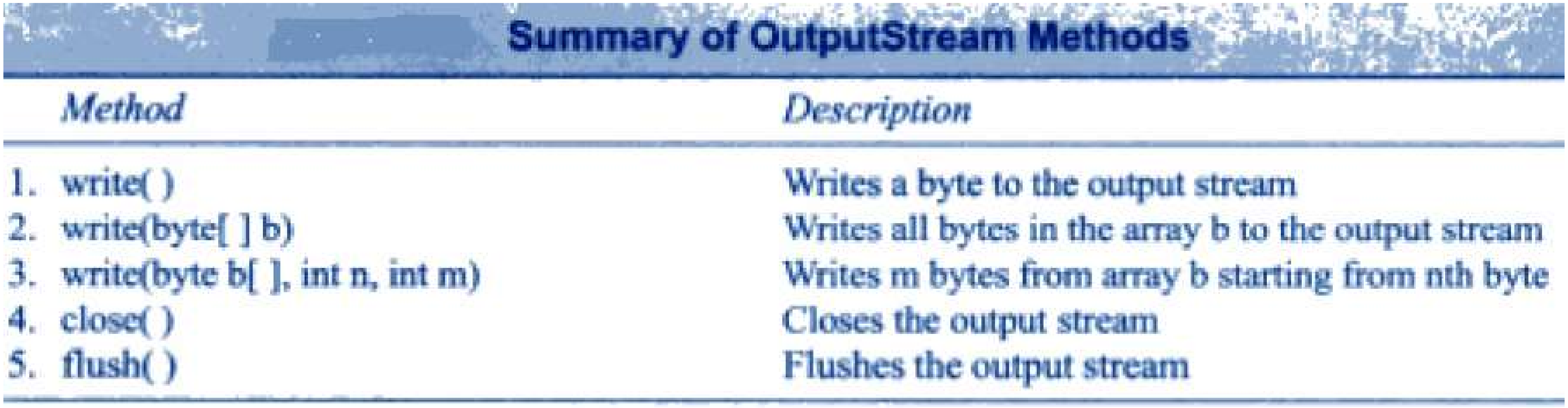
Writer Stream Classes



Hierarchy of writer stream classes

Writer Stream Classes

- The base class provides for all output operations by defining methods that are identical to those in *OutputStream* class



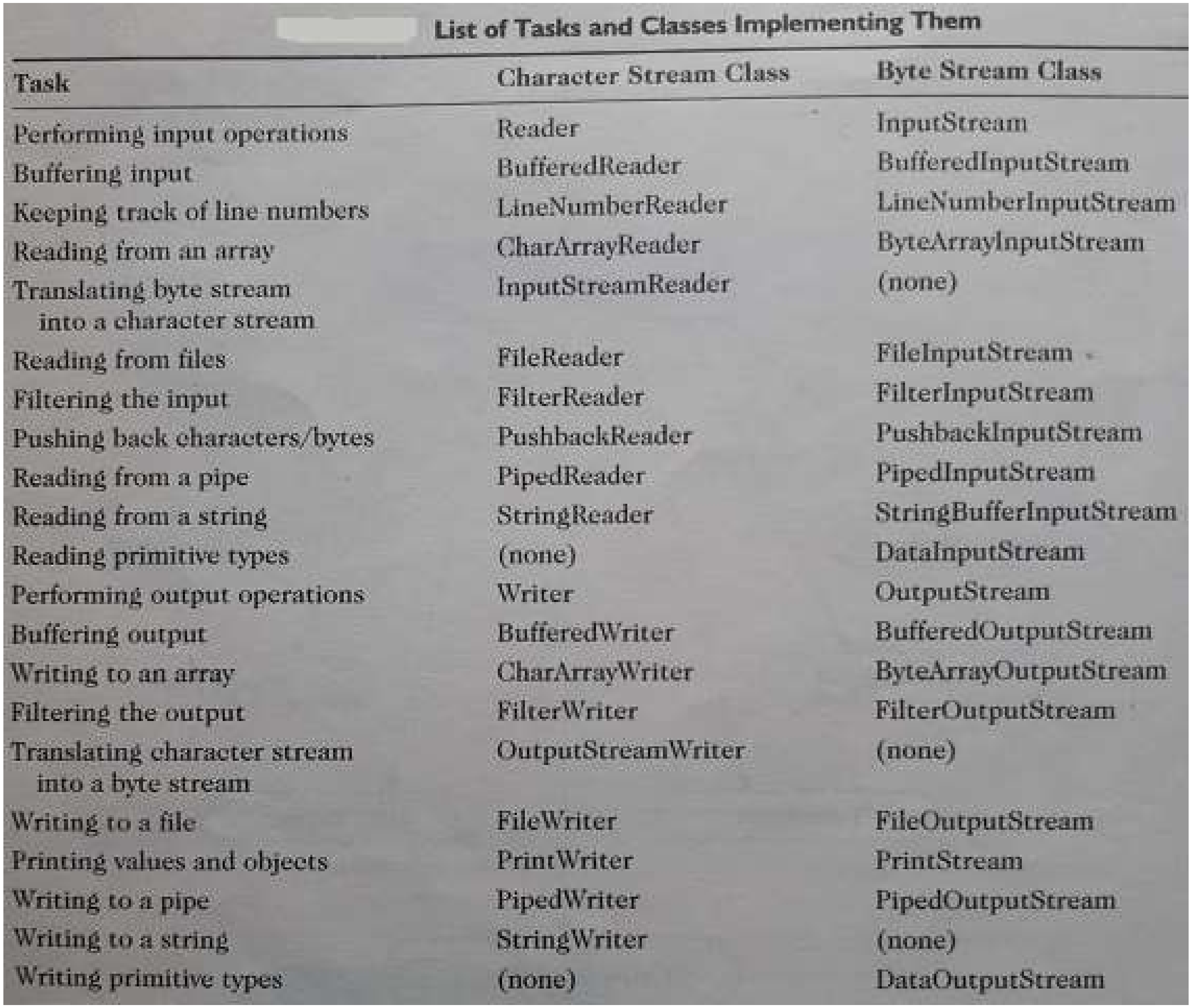
Summary of OutputStream Methods

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( )	Closes the output stream
5. flush( )	Flushes the output stream

Using Streams

- We have seen various 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 operations such as buffering, filtering, data conversion, counting and concatenation while carrying out i/o tasks
- Both of the character stream group and the bytes stream group contain parallel pairs of classes
  - It performs the same kind of operation but for the different data types

Using Streams



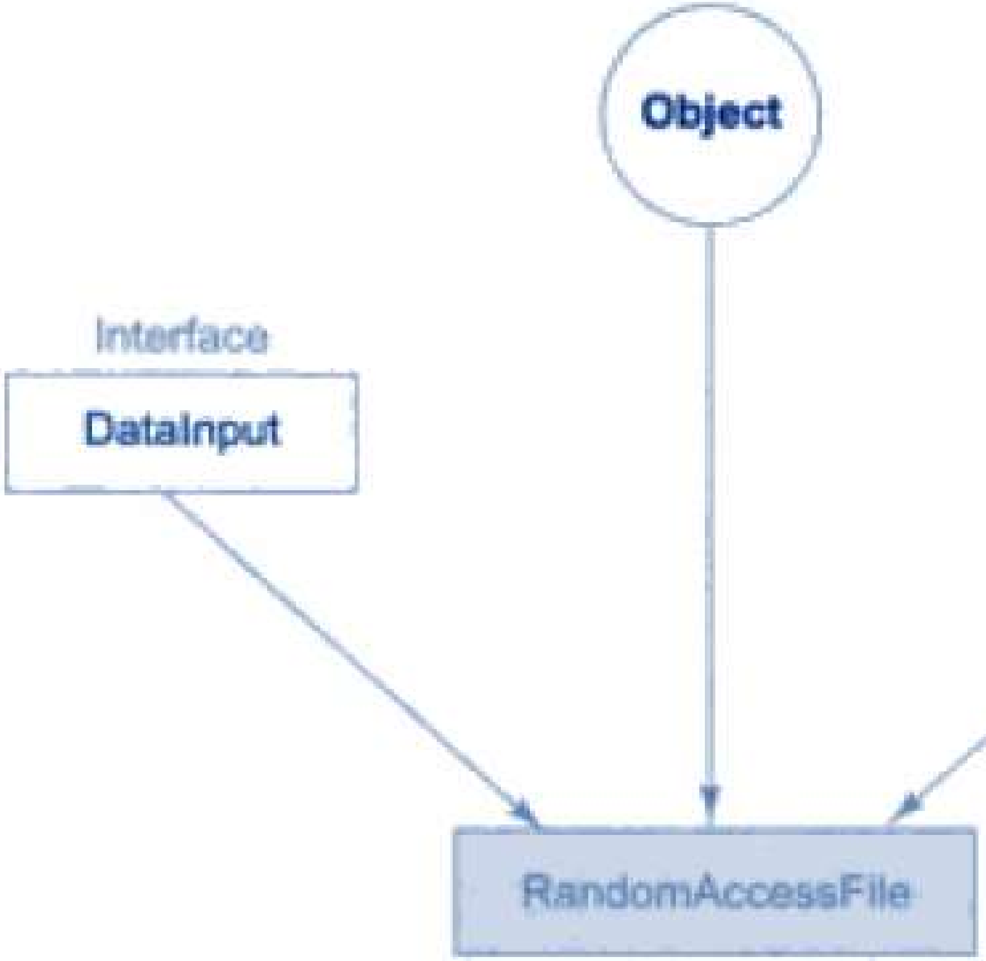
List of Tasks and Classes Implementing Them

Task	Character Stream Class	Byte Stream Class
Performing input operations	Reader	InputStream
Buffering input	BufferedReader	BufferedInputStream
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
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	StringWriter	(none)
Writing primitive types	(none)	DataOutputStream

Other Useful I/O Classes

- The **Java.io** package supports many other classes for performing certain specialize functions
  - RandomAccessFile
  - StreamToKenizer
- RandomAccessFile
  - The *RandomAccessFile* enables us to read and write byte, text and Java types to any location in a file
  - This class extend object class and implements *DataInput* and *DataOutput* interface
  - This forces the *RandomAccessFile* to implement the methods describe in both these interface

Other Useful I/O Classes



Implementation of the RandomAccessFile

Other Useful I/O Classes

- StreamToKenizer
  - The class Stream Tokenizer, a subclass of object can be used for breaking up a stream of text from an input text file into meaningful piece called *tokens*
  - The behavior of *StreamTokenizer* class is similar to that of the *StringTokenizer* class
    - That breaks a string into its component *tokens*

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
- This class also contains several methods for supporting the operations such as

• Creating a file	• Getting the size of the file
• Opening a file	• Checking the existence of a file
• Closing a file	• Renaming a file
• Deleting a file	• Checking whether the file is writable
• Getting the name of the file	• Checking whether the file is writable