Java Input Output

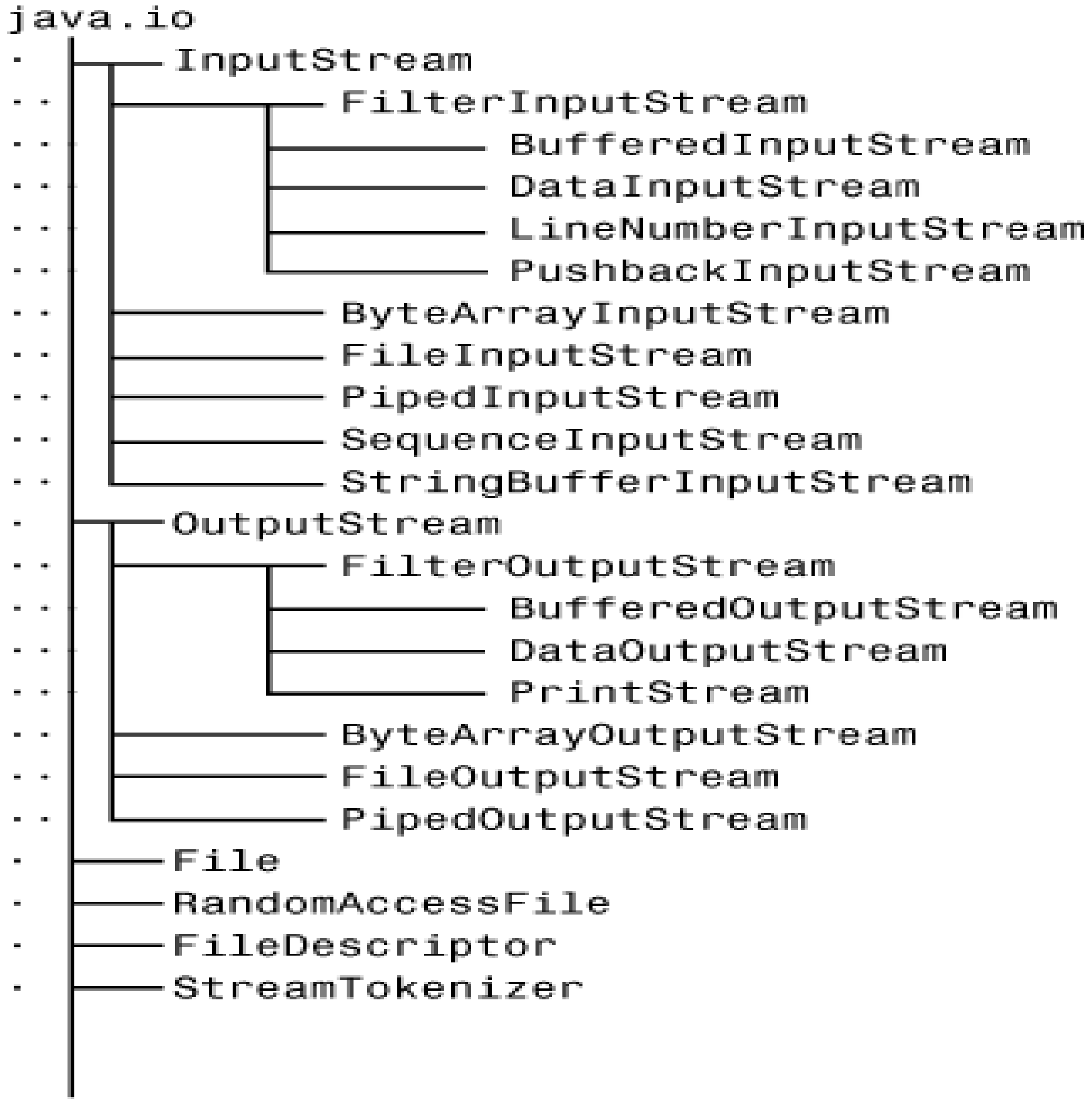
Java I/O

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

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

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

java.io



java.io
InputStream
FilterInputStream
BufferedInputStream
DataInputStream
LineNumberInputStream
PushbackInputStream
ByteArrayInputStream
FileInputStream
PipedInputStream
SequenceInputStream
StringBufferInputStream
OutputStream
FilterOutputStream
BufferedOutputStream
DataOutputStream
PrintStream
ByteArrayOutputStream
FileOutputStream
PipedOutputStream
File
RandomAccessFile
FileDescriptor
StreamTokenizer

Stream

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

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

1) System.out: standard output stream

2) System.in: standard input stream

3) System.err: standard error stream

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

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

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

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

OutputStream vs InputStream

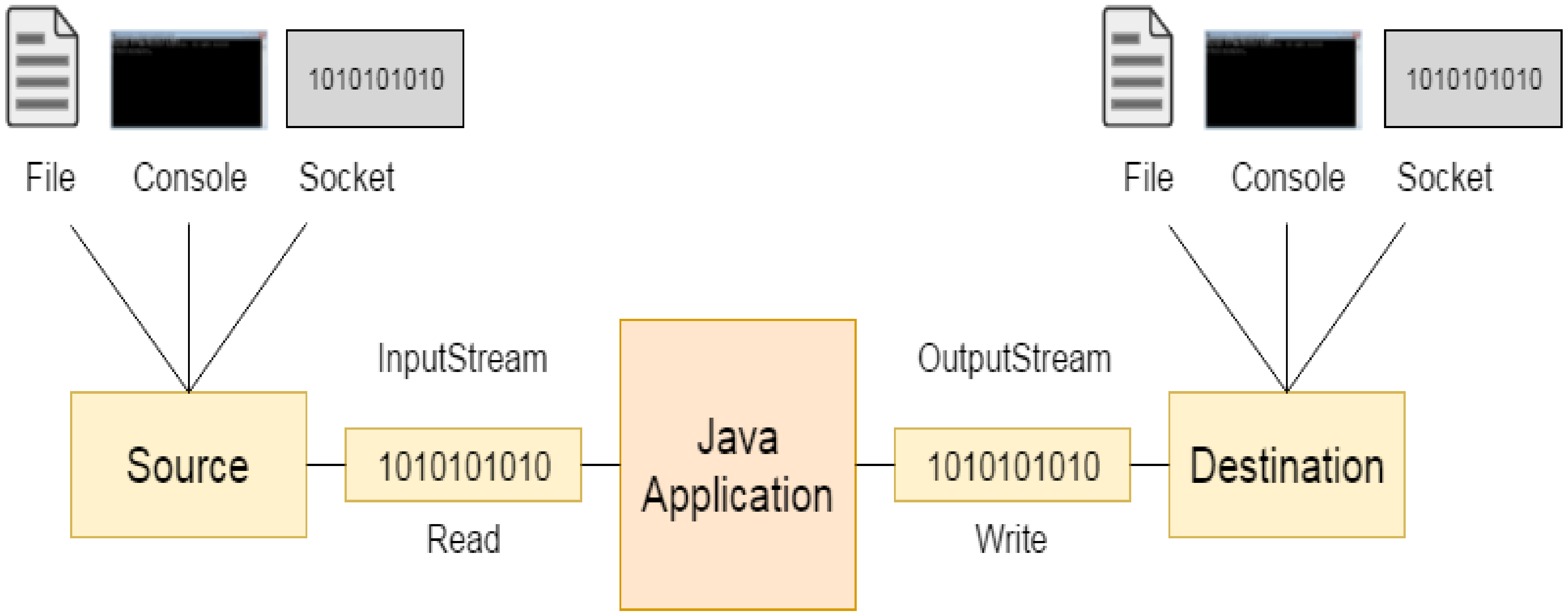
OutputStream

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

InputStream

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

Let's understand the working of Java OutputStream and InputStream by the figure given below.



File Class

- The File class provides the methods for obtaining the properties of a file/directory and for renaming and deleting a file/directory.

- An absolute file name (or full name) contains a file name with its complete path and drive letter.
- For example, c:\book\Welcome.java

- A relative file name is in relation to the current working directory.
- The complete directory path for a relative file name is omitted.
- For example, Welcome.java

CODERINDEED

Method and constructor for File class

Constructor:

File(String path_name)

Creates a File object for the specified path name. The path name may be a directory or a file.

Methods of File class

Methods:

```
boolean isFile()
boolean isDirectory()
boolean isHidden()
boolean exists()
boolean canRead()
boolean canWrite()
String getName()
String getPath()
String getAbsolutePath()
long lastModified()
long length()
boolean delete()
boolean renameTo(File f)
File[] listFiles()
```

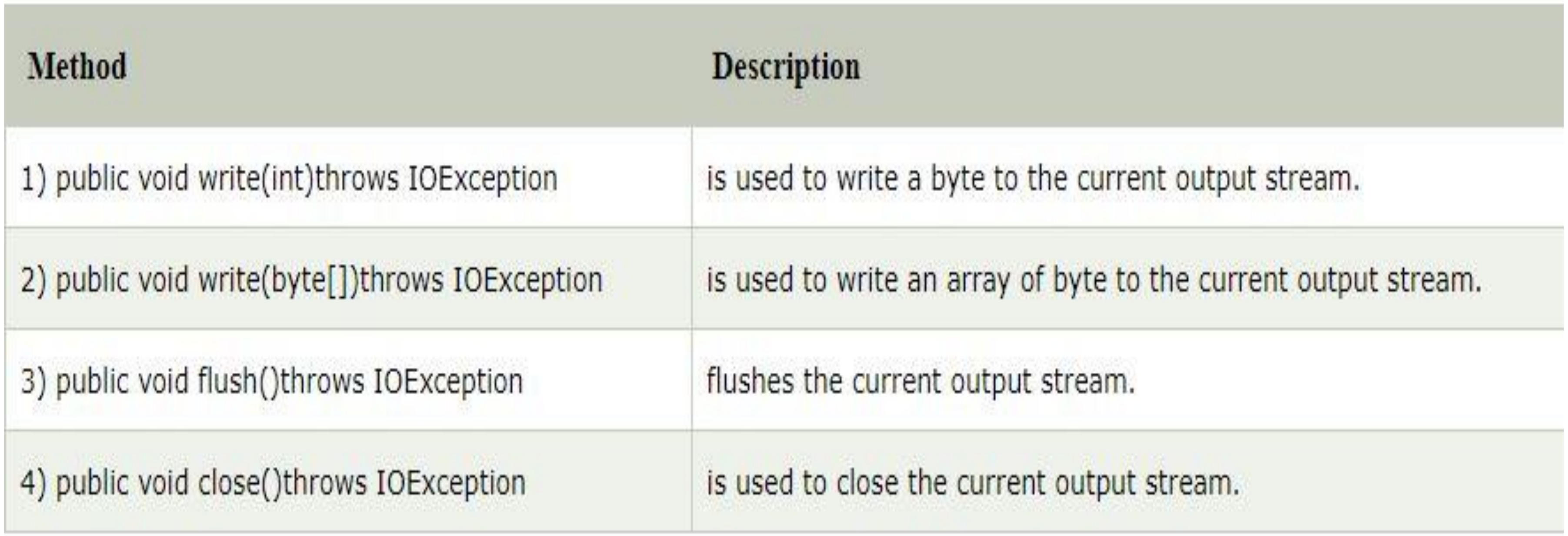
Example

```
import java.io.*;
public class FileClass
{
		public static void main(String[] args) {
		File ref1=new
File("C:\\Users\\Salil\\Desktop\\Java\\UNIT-
5\\Codes\\JavaIO");
		System.out.println(ref1.isFile());//It will give
false as JavaIO is not a file
	System.out.println(ref1.isDirectory());//It will give true
as JavaIO is a folder/or directory
	File ref2=new
File("C:\\Users\\Salil\\Desktop\\Java\\UNIT-
5\\Codes\\JavaIO\\abc.txt");
	System.out.println(ref2.exists());
	System.out.println(ref2.getName());
	System.out.println(ref2.getPath());
	File ref3=new
File("C:\\Users\\Salil\\Desktop\\Java\\UNIT-
5\\Codes\\JavaIO\\abc2.txt");
	File ref4=new
File("C:\\Users\\Salil\\Desktop\\Java\\UNIT-
5\\Codes\\JavaIO\\newname.txt");
	System.out.println(ref3.isHidden());
	System.out.println(ref3.canRead());
	System.out.println(ref3.canWrite());
 File ref5=new File("abc.txt");
	System.out.println(ref5.getPath());
	System.out.println(ref5.getAbsolutePath());
	System.out.println("Last modified on " + new
java.util.Date(ref5.lastModified()));
	System.out.println("Length:"+ref5.length());
	/*File ref6=new File("abc3.txt");
	if(ref6.delete())
	System.out.println("File deleted successfully");
	else
	System.out.println("File does not exists");*/
	File x[]=ref1.listFiles();
	for(File var:x)
	System.out.println(var);
	boolean flag = ref5.renameTo(ref4);
	if (flag == true) {
	   System.out.println("File Successfully Renamed");
	}
	else {
	   System.out.println("Operation Failed");
	}
		}
}
```

OutputStream class

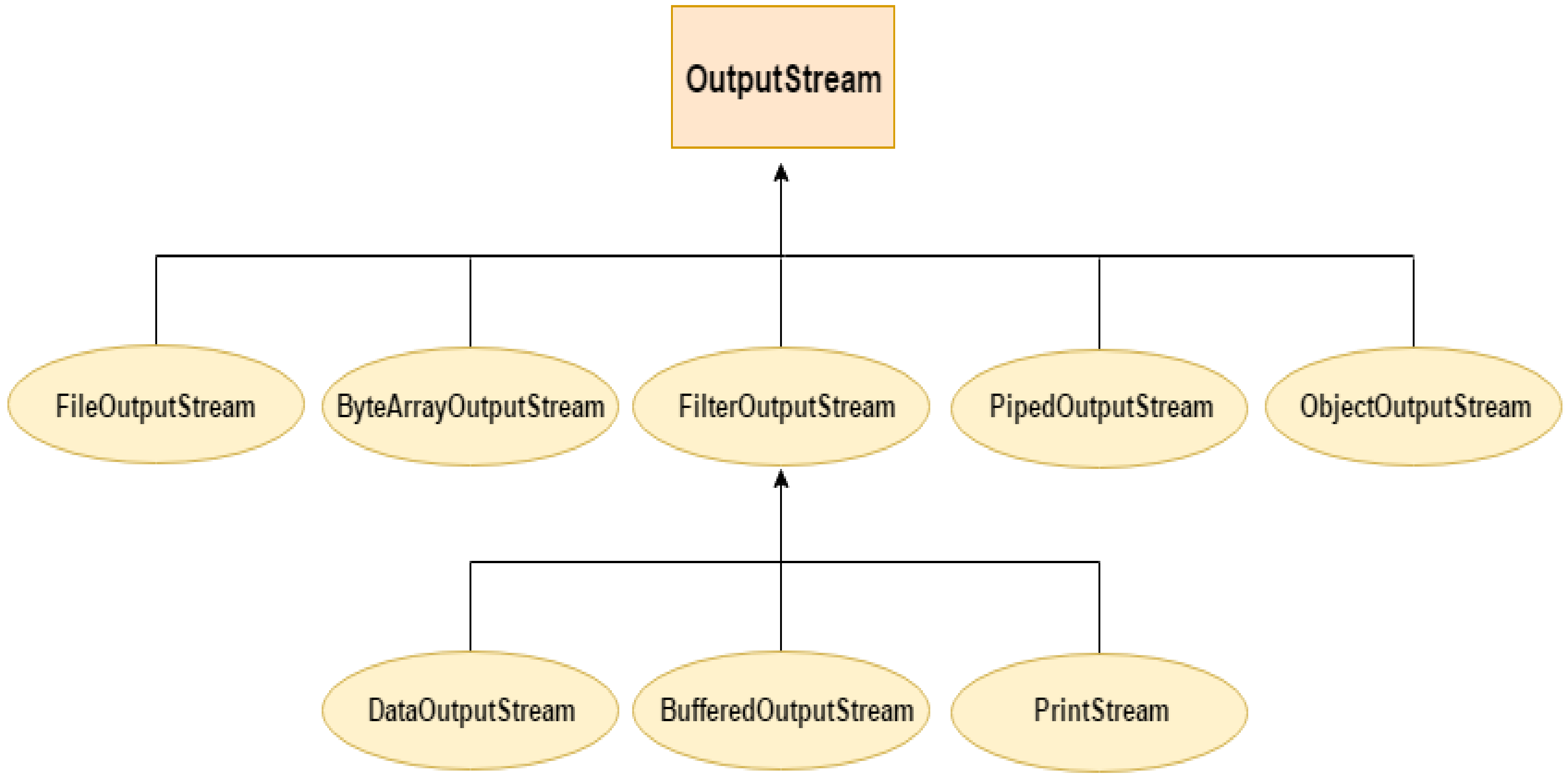
OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Useful methods of OutputStream



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

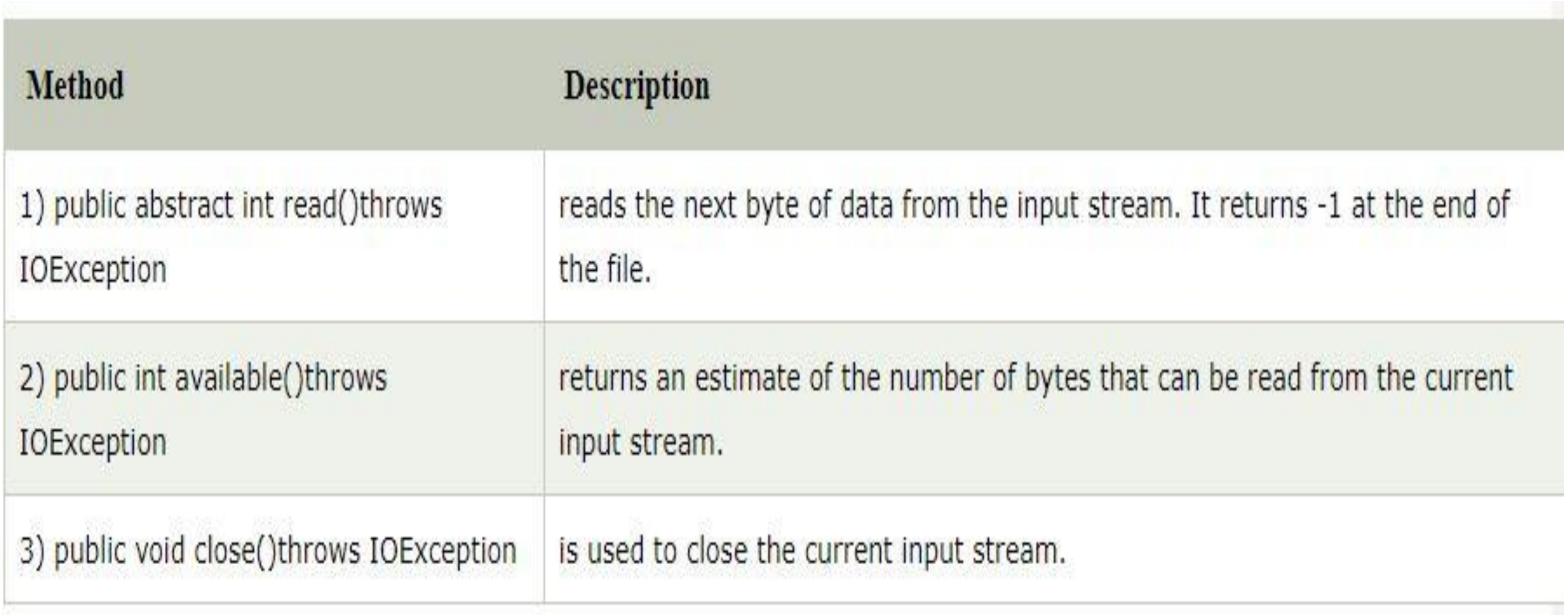
OutputStream Hierarchy



InputStream class

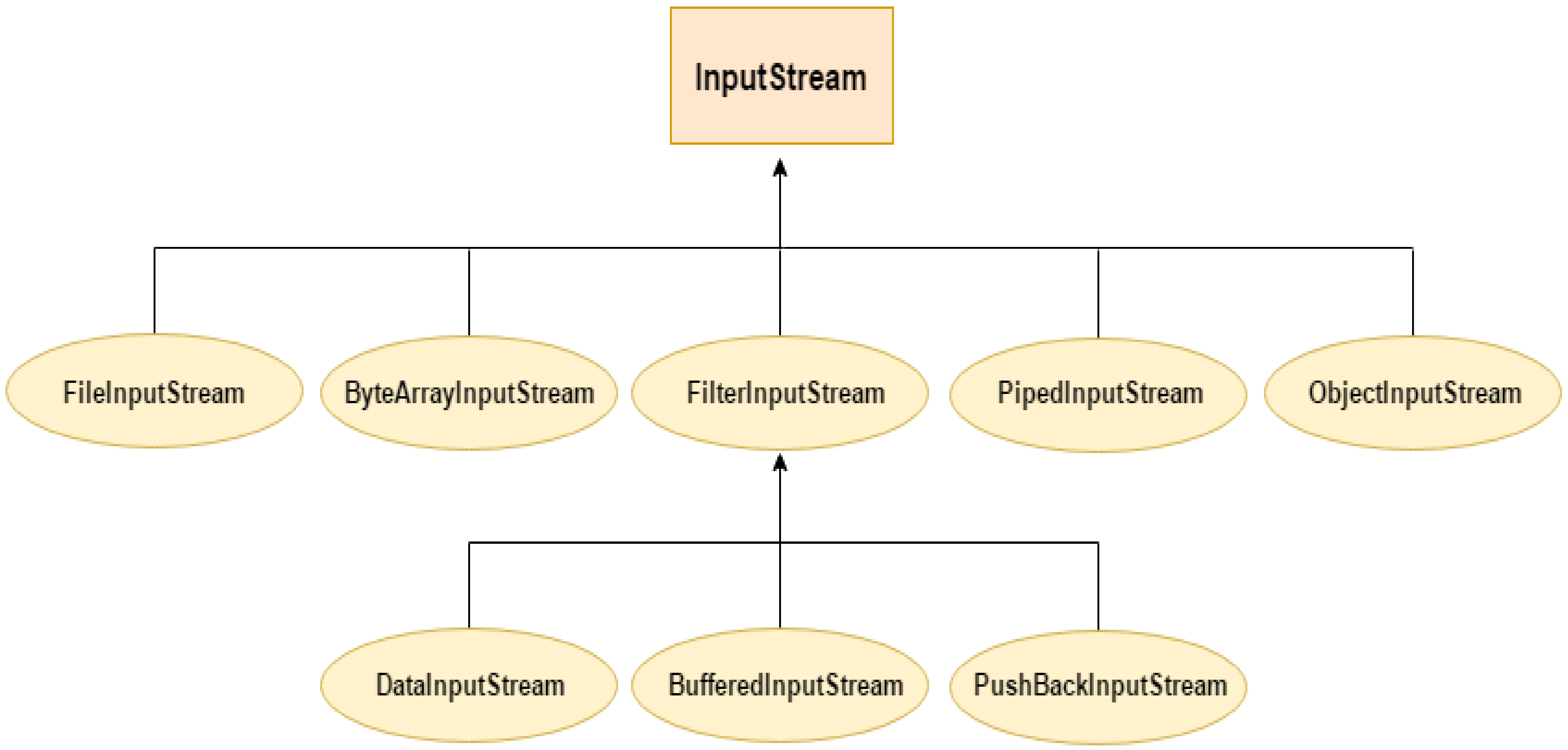
InputStream class is an abstract class. It is the superclass of all classes representing an input stream of bytes.

Useful methods of InputStream



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

InputStream Hierarchy



Java FileOutputStream Class

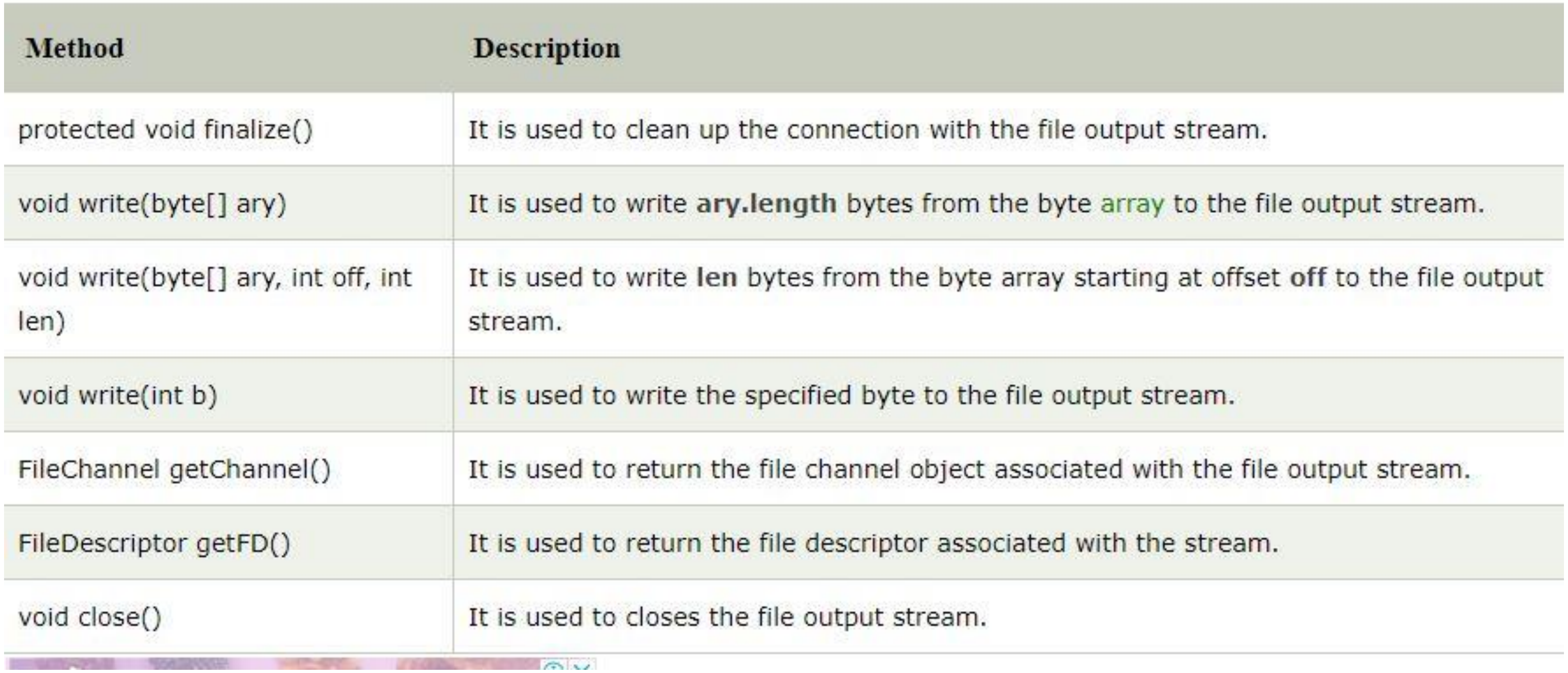
Java FileOutputStream is an output stream used for writing data to a file.

If you have to write primitive values into a file, use FileOutputStream class. You can write byte-oriented as well as character-oriented data through FileOutputStream class. But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.

Let's see the declaration for Java.io.FileOutputStream class:

public class FileOutputStream **extends** OutputStream

Methods of Java FileOutputStream Class



Method	Description
protected void finalize()	It is used to clean up the connection with the file output stream.
void write(byte[] ary)	It is used to write **ary.length** bytes from the byte array to the file output stream.
void write(byte[] ary, int off, int len)	It is used to write **len** bytes from the byte array starting at offset **off** to the file output stream.
void write(int b)	It is used to write the specified byte to the file output stream.
FileChannel getChannel()	It is used to return the file channel object associated with the file output stream.
FileDescriptor getFD()	It is used to return the file descriptor associated with the stream.
void close()	It is used to closes the file output stream.

Java FileOutputStream Example 1: write byte

```
import java.io.FileOutputStream;
public class Main{
    public static void main(String args[]){
            try{
              FileOutputStream fout=new
              FileOutputStream("D:\\testout.txt");
              fout.write(65);
              fout.close();
              System.out.println("success...");
             }catch(Exception e){System.out.println(e);}
       }
}
```

Java FileOutputStream example 2: write string

```
import java.io.FileOutputStream;
public class Main {
    public static void main(String args[]){
           try{
             FileOutputStream fout=new FileOutputStream("testout.txt");
             String s="Welcome to file handling.";
             byte b[]=s.getBytes();//converting string into byte array
             fout.write(b);
             fout.close();
             System.out.println("success...");
            }catch(Exception e){System.out.println(e);}
      }
}
```

Appending data through FileOutputStream

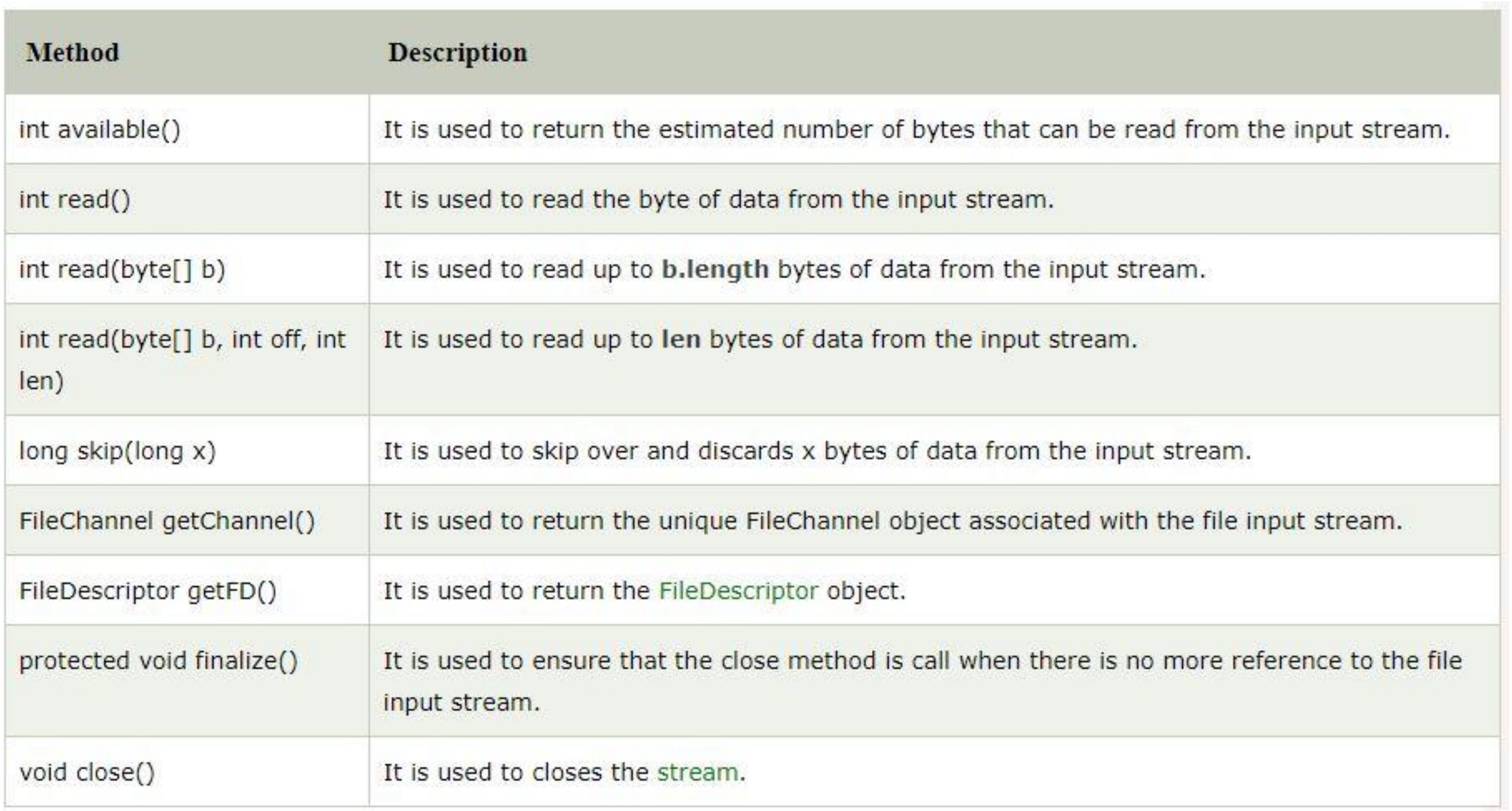
```
import java.io.FileOutputStream;
public class Main{
     public static void main(String args[]){
          try{
           String textToAppend = "Hello !!"; //new line in content
    FileOutputStream outputStream = new
FileOutputStream("testout.txt", true);//For appending the data here
we need to give true in second argument
    byte[] strToBytes = textToAppend.getBytes();
    outputStream.write(strToBytes);
    outputStream.close();
          }catch(Exception e){System.out.println(e);}
         }
        }
```

Java FileInputStream Class

Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use FileReader class.

public class FileInputStream **extends** InputStream

Java FileInputStream class methods



Method	Description
int available()	It is used to return the estimated number of bytes that can be read from the input stream.
int read()	It is used to read the byte of data from the input stream.
int read(byte[] b)	It is used to read up to **b.length** bytes of data from the input stream.
int read(byte[] b, int off, int len)	It is used to read up to **len** bytes of data from the input stream.
long skip(long x)	It is used to skip over and discards x bytes of data from the input stream.
FileChannel getChannel()	It is used to return the unique FileChannel object associated with the file input stream.
FileDescriptor getFD()	It is used to return the FileDescriptor object.
protected void finalize()	It is used to ensure that the close method is call when there is no more reference to the file input stream.
void close()	It is used to closes the stream.

Java FileInputStream example 1: read single character

```
import java.io.FileInputStream;
public class Main{
    public static void main(String args[]){
        try{
          FileInputStream fin=new FileInputStream("testout.txt");
          int i=fin.read();
          System.out.print((char)i);

          fin.close();
        }catch(Exception e){System.out.println(e);}
       }
      }
```

Java FileInputStream example 2: read all characters

```
import java.io.FileInputStream;
public class Main{
     public static void main(String args[]){
          try{
            FileInputStream fin=new FileInputStream("testout.txt");
            int i=0;
            while((i=fin.read())!=-1){
             System.out.print((char)i);
            }
            fin.close();
          }catch(Exception e){System.out.println(e);}
         }
        }
```

Practice Programs to do

WAP to copy the data of one file into another file

WAP to read all characters from a file and count total number of vowels from it

WAP to read the content from two files and write their merged content into third file

Reading and writing files

- In Java, all files are byte-oriented, and Java provides methods to read and write bytes from and to a file.
- Java allows us to wrap a byte-oriented file stream within a character-based object.
- We can use Scanner and PrintWriter class to read and write Files.

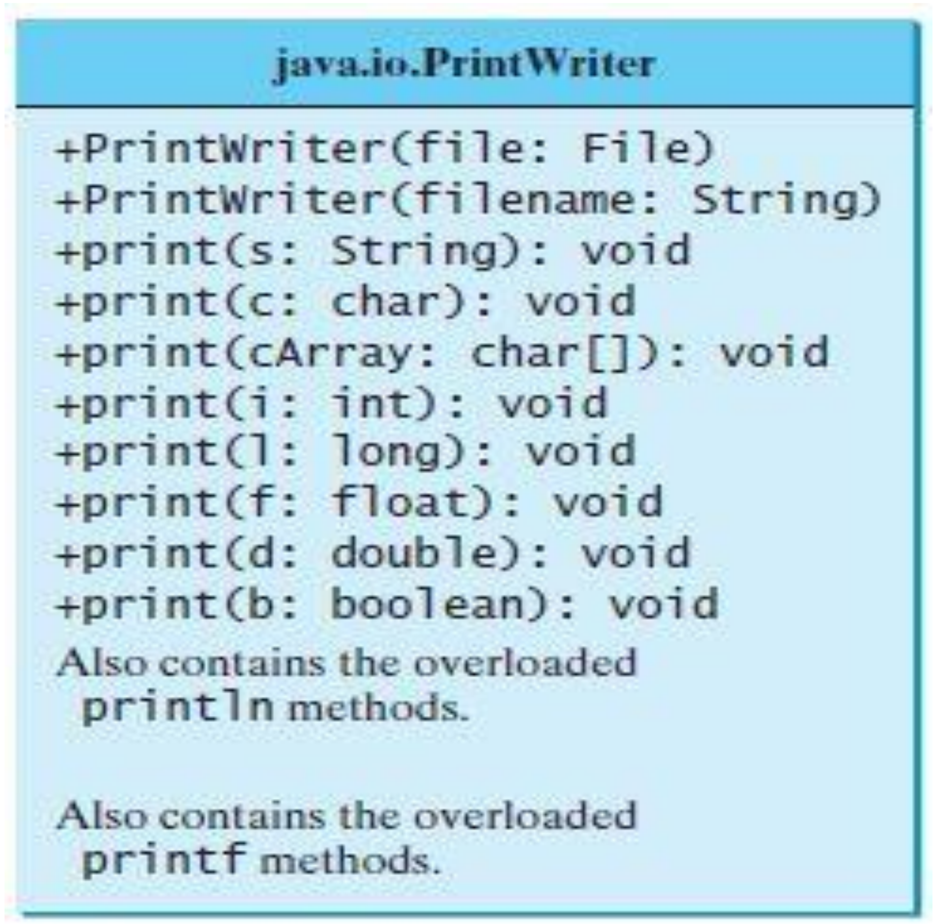
PrinterWriter Class

This class gives Prints formatted representations of objects to a text-output stream. It implements all of the print methods found in PrintStream.

PrintWriter supports the print( ) and println( ) methods for all types including Object. Thus, we can use these methods in the same way as they have been used with System.out.

CODERINDEED

PrintWriter Methods



java.io.PrintWriter

```
+PrintWriter(file: File)
+PrintWriter(filename: String)
+print(s: String): void
+print(c: char): void
+print(cArray: char[]): void
+print(i: int): void
+print(l: long): void
+print(f: float): void
+print(d: double): void
+print(b: boolean): void
```

Also contains the overloaded `println` methods.

Also contains the overloaded `printf` methods.

Example 1

```
import java.io.*;
public class Main
    {
        public static void main(String args[])
            {
                PrintWriter pw = new PrintWriter(System.out, true);
                pw.println(“Using PrintWriter Object");
                int i = -7;
                pw.println(i);
                double d = 4.5e-7;
                pw.println(d);
            }
    }
```

Example 2

```
import java.io.*;
public class printwriter{
    public static void main(String args[]){
            try{
        PrintWriter pw=new PrintWriter(new File("target.txt"));
        pw.print("Hi");
        pw.flush();
        pw.close();
        }catch(Exception e){System.out.println(e);}
     }
    }
```

Using Scanner

- The java.util.Scanner class is used to read strings and primitive values from the console.

- Scanner breaks the input into tokens delimited by whitespace characters.

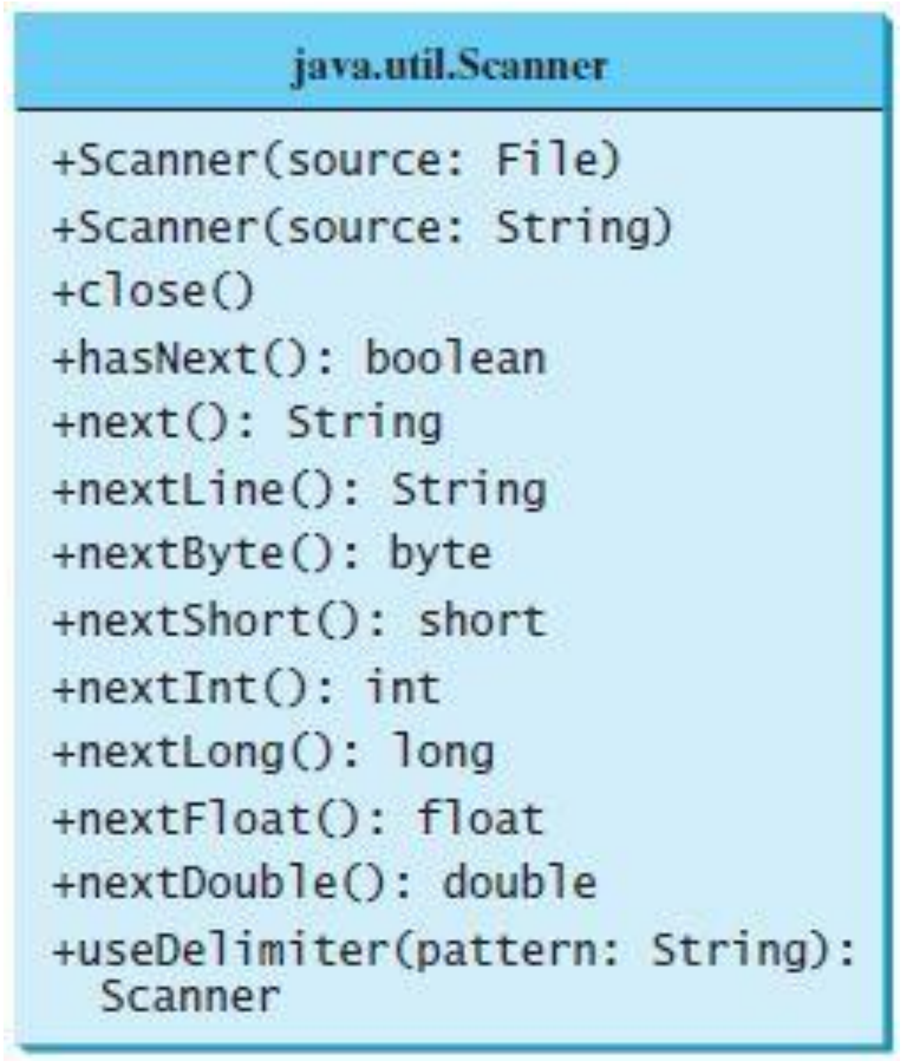
- To read from the keyboard, we create a Scanner as follows:

```
Scanner s = new Scanner(System.in);
```

- To read from a file, create a Scanner for a file, as follows:

```
Scanner s = new Scanner(new File(filename));
```

Scanner Methods



```
java.util.Scanner
+Scanner(source: File)
+Scanner(source: String)
+close()
+hasNext(): boolean
+next(): String
+nextLine(): String
+nextByte(): byte
+nextShort(): short
+nextInt(): int
+nextLong(): long
+nextFloat(): float
+nextDouble(): double
+useDelimiter(pattern: String):
   Scanner
```

Serialization and Deserialization in Java

Serialization in Java is a mechanism of writing the state of an object into a byte-stream.

The reverse operation of serialization is called deserialization where byte-stream is converted into an object. The serialization and deserialization process is platform-independent, it means you can serialize an object in a platform and deserialize in different platform.

For serializing the object, we call the **writeObject()** method ObjectOutputStream, and for deserialization we call the **readObject()** method of ObjectInputStream class.

- We must have to implement the *Serializable* interface for serializing the object.

Advantages of Java Serialization

- It is mainly used to travel object's state on the network (which is known as marshaling).

java.io.Serializable interface

Serializable is a marker interface (has no data member and method). It is used to "mark" Java classes so that the objects of these classes may get a certain capability. The Cloneable and Remote are also marker interfaces.

It must be implemented by the class whose object you want to persist.

The String class and all the wrapper classes implement the *java.io.Serializable* interface by default.

```
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;
        }
}
```

In the above example, Student class implements Serializable interface. Now its objects can be converted into stream.

ObjectOutputStream class

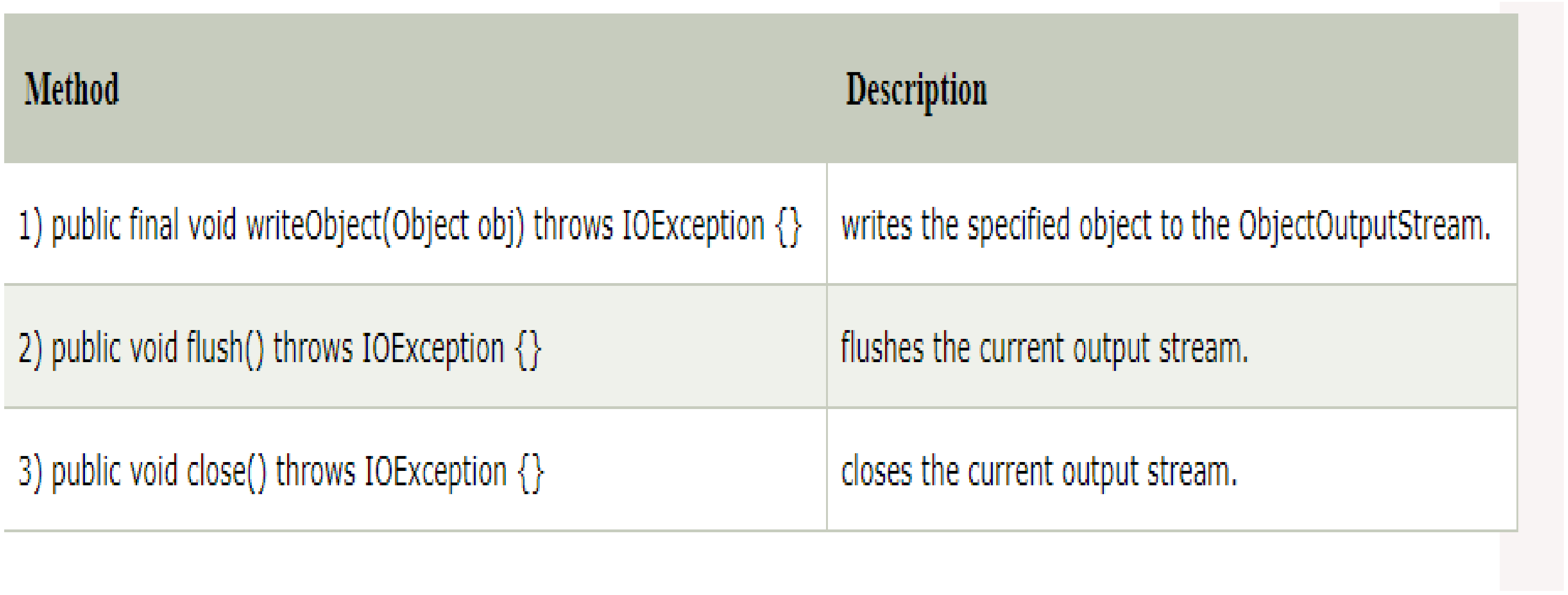
The ObjectOutputStream class is used to write primitive data types, and Java objects to an OutputStream. Only objects that support the java.io.Serializable interface can be written to streams.

Constructor

public ObjectOutputStream(OutputStream out) throws IOException {}

creates an ObjectOutputStream that writes to the specified OutputStream.

Important Methods



Method	Description
1) public final void writeObject(Object obj) throws IOException {}	writes the specified object to the ObjectOutputStream.
2) public void flush() throws IOException {}	flushes the current output stream.
3) public void close() throws IOException {}	closes the current output stream.

ObjectInputStream class

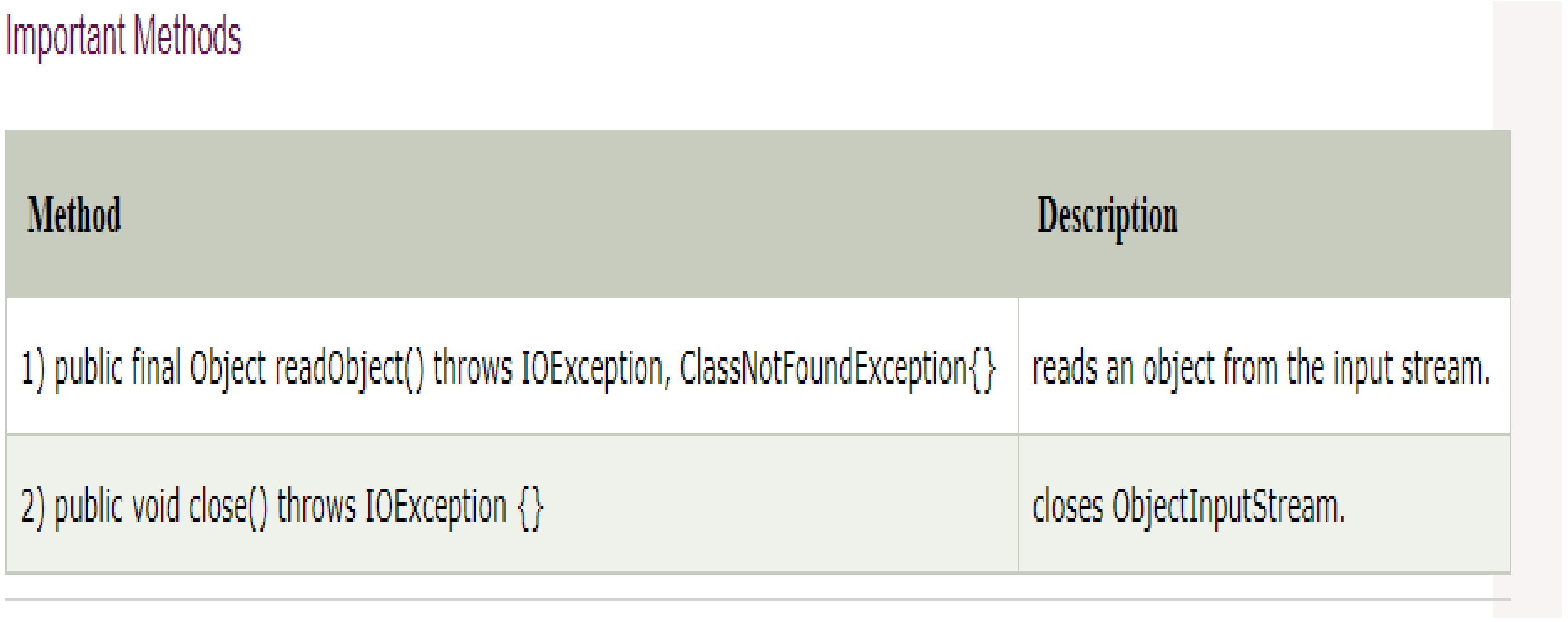
An ObjectInputStream deserializes objects and primitive data written using an ObjectOutputStream.

Constructor

public ObjectInputStream(InputStream in) throws IOException {}

creates an ObjectInputStream that reads from the specified InputStream.

Important Methods



Important Methods

Method	Description
1) public final Object readObject() throws IOException, ClassNotFoundException{}	reads an object from the input stream.
2) public void close() throws IOException {}	closes ObjectInputStream.

Example ObjectOutputStream

```
import java.io.*;
class student implements Serializable
{
    int roll_no;
    String name;
    student (int r,String s)
    {
        roll_no=r;
        name=s;
    }
}
public class Main
{
            public static void main(String[] args) throws Exception
            {
                        FileOutputStream fos = new FileOutputStream("t.txt");
                       ObjectOutputStream oos = new ObjectOutputStream(fos);
                        student s= new student(1,"kk");
                        oos.writeObject(s);

                         oos.close();
            }
}
```

Example ObjectInputStream

```
import java.io.*;
class student implements Serializable
{
   int roll_no;
   String name;
   student (int r,String s)
   {
      roll_no=r;
      name=s;
   }
}
public class Example
{
                public static void main(String[] args) throws Exception
                {
                                FileInputStream fos = new FileInputStream("E:\\t.txt");
       ObjectInputStream oos = new ObjectInputStream(fos);
       student s=(student)oos.readObject();
       System.out.println(s.roll_no+" "+s.name);
       oos.close();
                }
}
```

Java Serialization with Inheritance

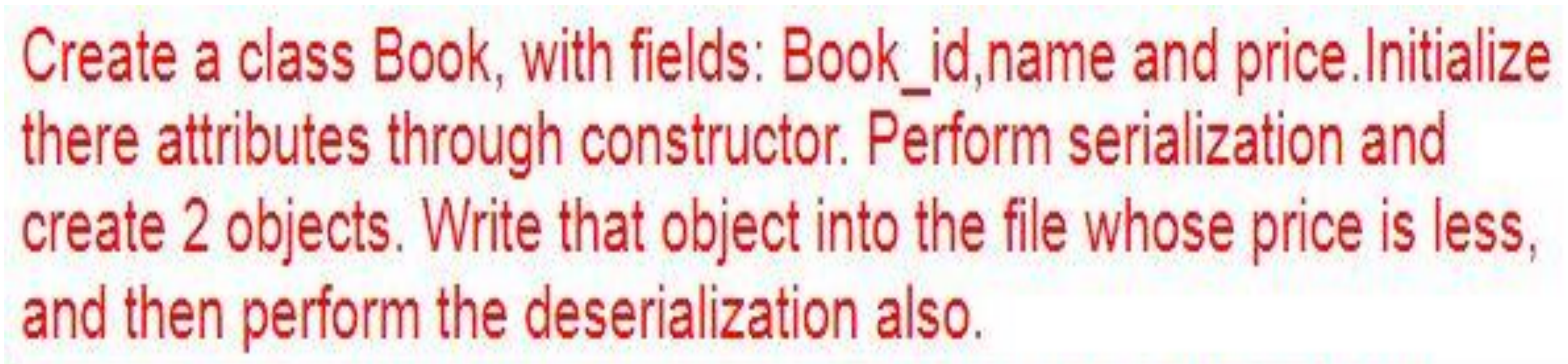
If a class implements serializable then all its sub classes will also be serializable. Let's see the example given below:

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

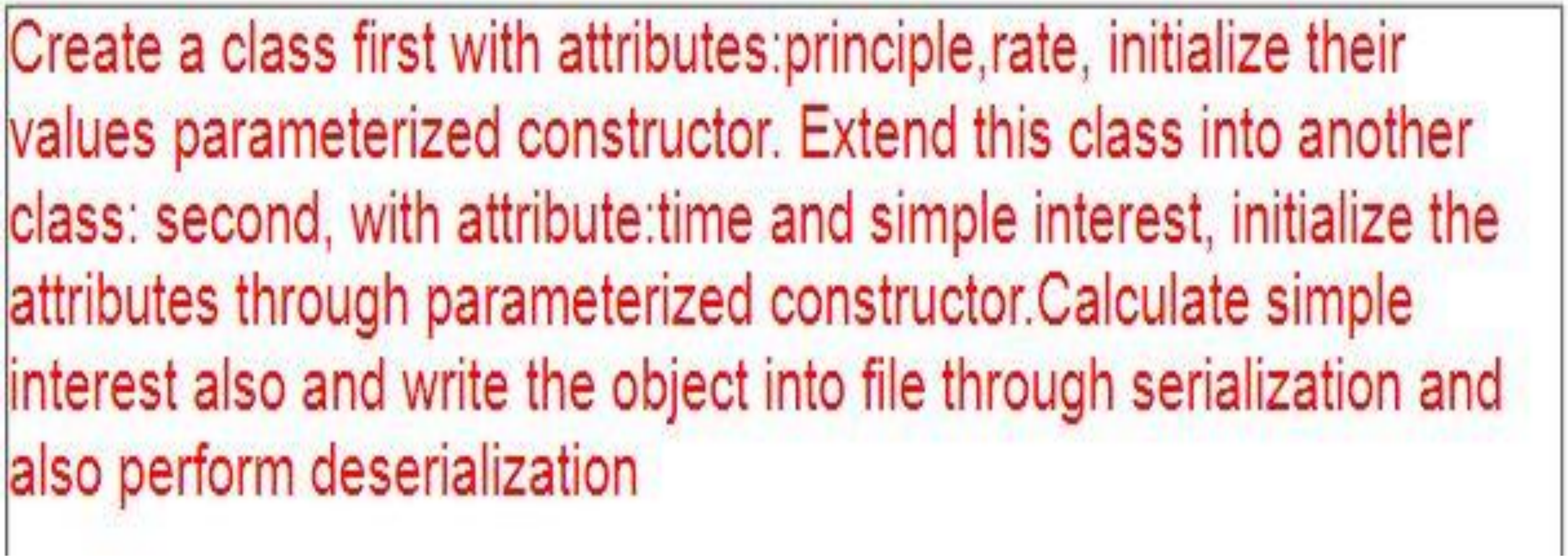
```
class Student extends Person{
 String course;
 int fee;
 public Student(int id, String name, String course, int fee) {
  super(id,name);
  this.course=course;
  this.fee=fee;
 }
}
```

Now you can serialize the Student class object that extends the Person class which is Serializable. Parent class properties are inherited to subclasses so if parent class is Serializable, subclass would also be.

Practice questions



Create a class Book, with fields: Book_id,name and price.Initialize there attributes through constructor. Perform serialization and create 2 objects. Write that object into the file whose price is less, and then perform the deserialization also.



Create a class first with attributes:principle,rate, initialize their values parameterized constructor. Extend this class into another class: second, with attribute:time and simple interest, initialize the attributes through parameterized constructor.Calculate simple interest also and write the object into file through serialization and also perform deserialization

Which of these methods are used to read in from file?

a) get()

b) read()

c) scan()

d) readFileInput()

Which of these values is returned by read() method is end of file (EOF) is encountered?

a) 0

b) 1

c) -1

d) Null

Which of these exception is thrown by close() and read() methods?

a) IOException

b) FileException

c) FileNotFoundException

d) FileInputOutputException

Which of these methods is used to write() into a file?

a) put()

b) putFile()

c) write()

d) writeFile()

What will be the output of the following Java program?

```
import java.io.*;
class filesinputoutput
{
    public static void main(String args[])
    {
        InputStream obj = new FileInputStream("inputoutput.java");
        System.out.print(obj.available());
    }
}
```

Note: inputoutput.java is stored in the disk.

a) true
b) false
c) prints number of bytes in file
d) prints number of characters in the file