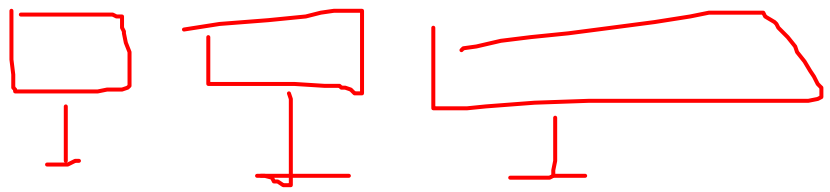
Java Exception Handling and Java I/O

Exception Handling

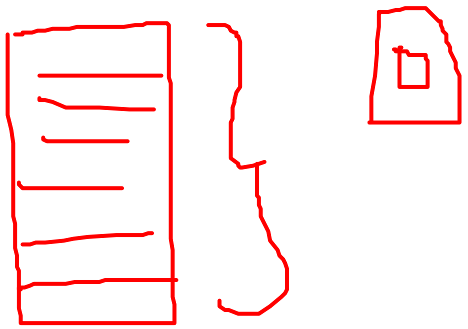
- Exception:
- Exception is an abnormal condition.

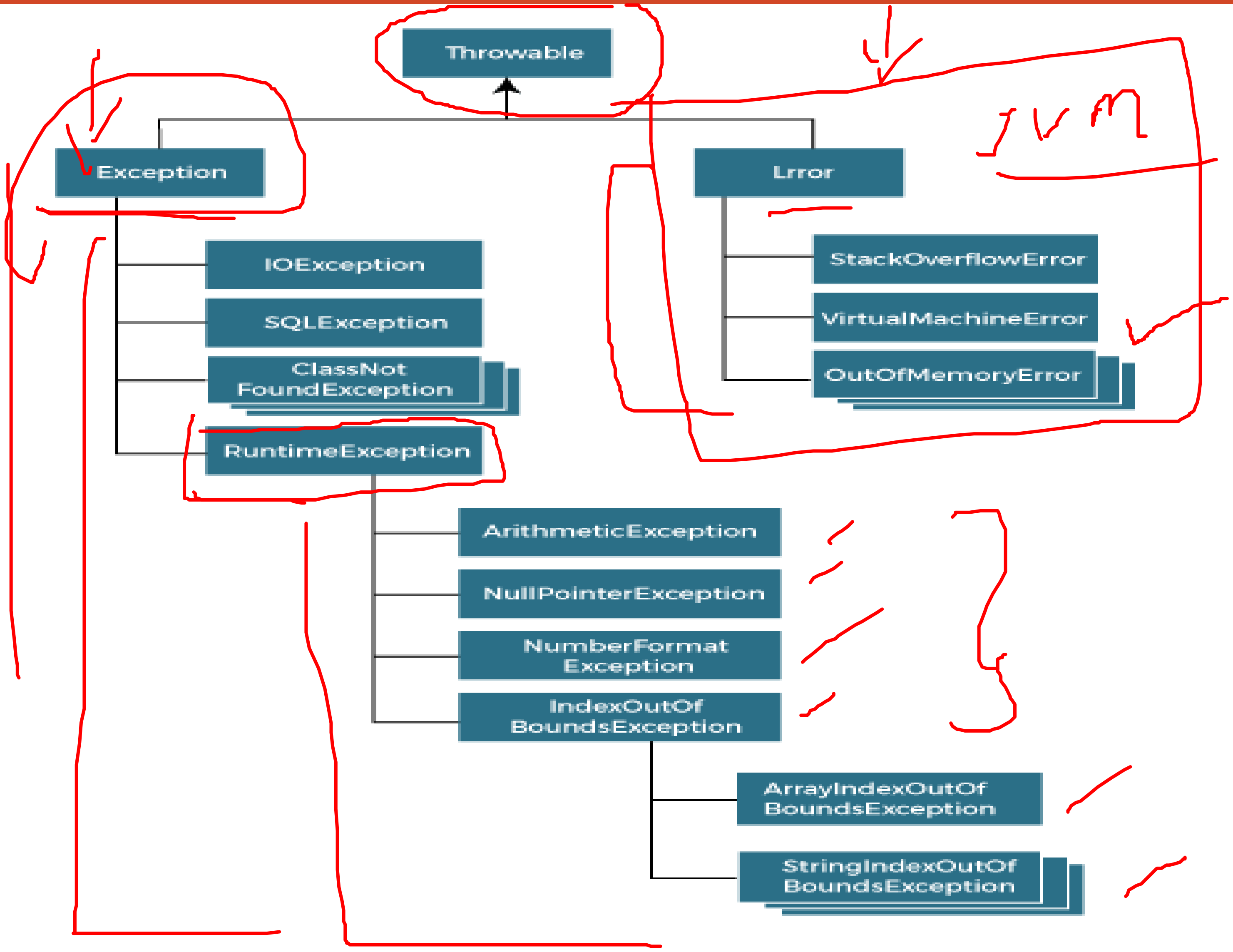


- In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.
- What is Exception Handling?
- Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.
- Advantage of Exception Handling
- The core advantage of exception handling is to maintain the normal flow of the application.
- An exception normally disrupts the normal flow of the application; that is why we need to handle exceptions.

Exception Handling

- [Hierarchy of Java Exception classes]
- The java.lang.Throwable class is the root class of Java Exception hierarchy inherited by two subclasses: Exception and Error.





Exception Handling

- Types of Java Exceptions
- There are mainly two types of exceptions: checked and unchecked.
- An error is considered as the unchecked exception.
- However, according to Oracle, there are three types of exceptions namely:

1. Checked Exception
2. Unchecked Exception
3. Error

Exception Handling

1) Checked Exception

The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException, etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) Error

Error is irrecoverable. Some example of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

Exception Handling

Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

Exception Handling

Java try block

- Java try block is used to enclose the code that might throw an exception. It must be used within the method.
- If an exception occurs at the particular statement in the try block, the rest of the block code will not execute. So, it is recommended not to keep the code in try block that will not throw an exception.
- Java try block must be followed by either catch or finally block.

```
try{
//code that may throw an exception

}catch(Exception_class_Name ref){}
```

```
try{
//code that may throw an exception

}finally{}
```

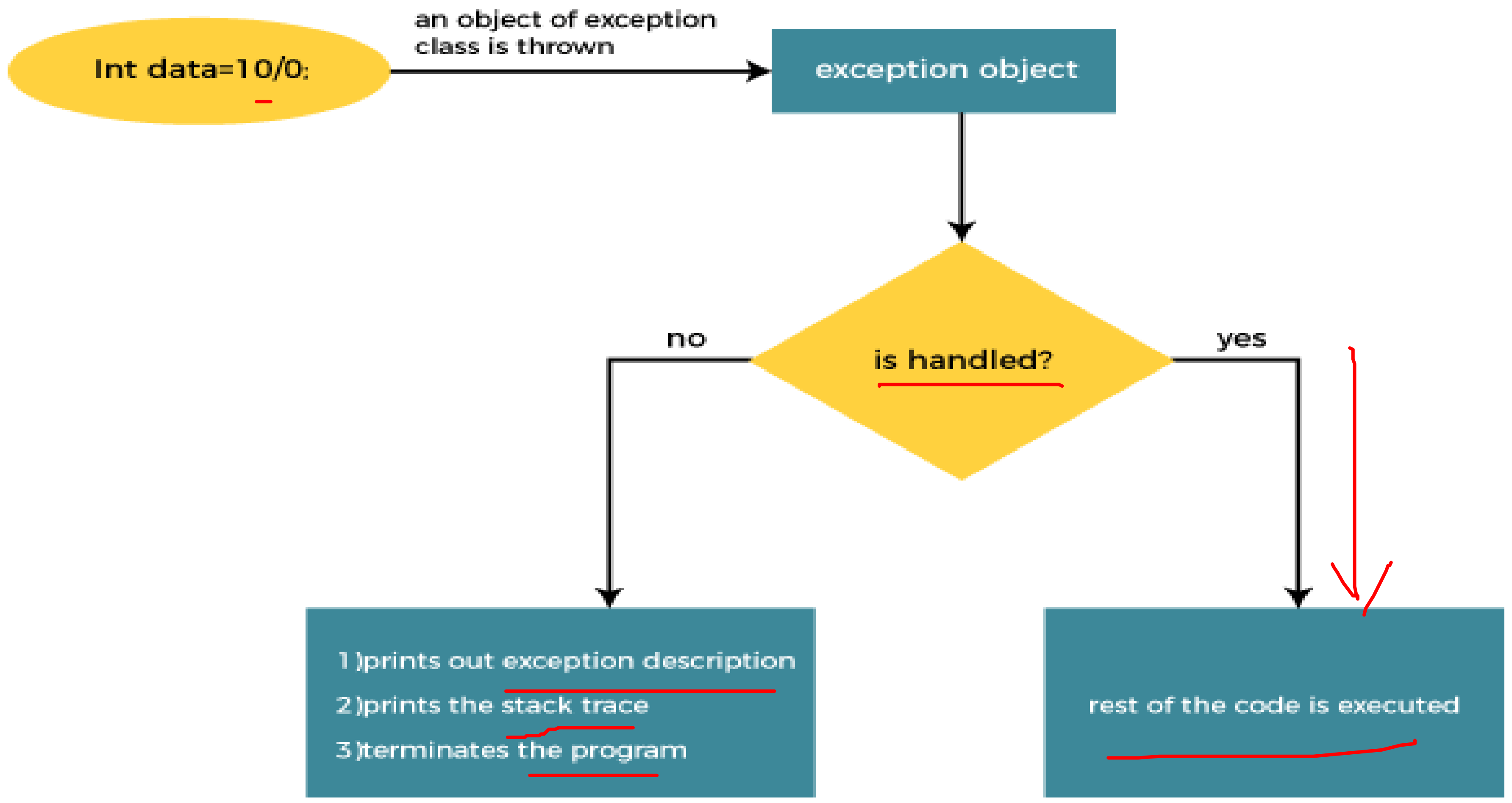
Exception Handling

Java catch block

- Java catch block is used to handle the Exception by declaring the type of exception within the parameter.
- The declared exception must be the parent class exception (i.e., Exception) or the generated exception type. However, the good approach is to declare the generated type of exception.
- The catch block must be used after the try block only. You can use multiple catch block with a single try block.

Exception Handling

Internal Working of Java try-catch block


Int data=10/0;
an object of exception class is thrown
exception object
is handled?
no
yes
1)prints out exception description
2)prints the stack trace
3)terminates the program
rest of the code is executed

Exception Handling

Java Multi-catch block

- A try block can be followed by one or more catch blocks.
- Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.
- At a time only one exception occurs and at a time only one catch block is executed.
- All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.
- MultipleCatchBlock1.java

Exception Handling

Java Nested try block

- In Java, using a try block inside another try block is permitted.
- It is called as nested try block. Every statement that we enter a statement in try block, context of that exception is pushed onto the stack.
- For example, the **inner try block** can be used to handle **ArrayIndexOutOfBoundsException** while the **outer try block** can handle the **ArithemeticException** (division by zero).
- Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.
- NestedTryBlock.java NestedTryBlock2.java

Exception Handling

Java finally block

- **Java finally block** is a block used to execute important code such as closing the connection, etc.
- Java finally block is always executed whether an exception is handled or not. Therefore, it contains all the necessary statements that need to be printed regardless of the exception occurs or not.
- The finally block follows the try-catch block.
- finally block in Java can be used to put "cleanup" code such as closing a file, closing connection, etc.
- The important statements to be printed can be placed in the finally block.
- TestFinallyBlock.java

Exception Handling

Java throw Exception

- The Java throw keyword is used to throw an exception explicitly.
- We specify the exception object which is to be thrown. The Exception has some message with it that provides the error description. These exceptions may be related to user inputs, server, etc.
- We can throw either checked or unchecked exceptions in Java by throw keyword. It is mainly used to throw a custom exception.
- We can also define our own set of conditions and throw an exception explicitly using throw keyword. For example, we can throw ArithmeticException if we divide a number by another number. Here, we just need to set the condition and throw exception using throw keyword.
- **throw new** exception_class("error message");
- **throw new** IOException("sorry device error");

Exception Handling

Java throw Keyword

- The Java throw keyword is used to throw an exception explicitly.
- We specify the exception object which is to be thrown. The Exception has some message with it that provides the error description. These exceptions may be related to user inputs, server, etc.
- We can throw either checked or unchecked exceptions in Java by throw keyword. It is mainly used to throw a custom exception. Instance must be of type Throwable or a subclass of Throwable.
- We can also define our own set of conditions and throw an exception explicitly using throw keyword. For example, we can throw ArithmeticException if we divide a number by another number. Here, we just need to set the condition and throw exception using throw keyword.
- **throw new** exception_class("error message"); **throw new** IOException("sorry device error");
- TestThrow.java

Exception Handling

Java throws Keyword

- throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions.
- The caller to these methods has to handle the exception using a try-catch block.
- **type method_name(parameters) throws exception_list**
- exception_list is a comma separated list of all the exceptions which a method might throw
- In a program, if there is a chance of raising an exception then compiler always warn us about it and compulsorily we should handle that checked exception, Otherwise we will get compile time error saying **unreported exception must be caught or declared to be thrown**. To prevent this compile time error we can handle the exception in two ways:

1. By using try catch
2. By using **throws** keyword TestThrows.java

Exception Handling

Java Userdefined Exception

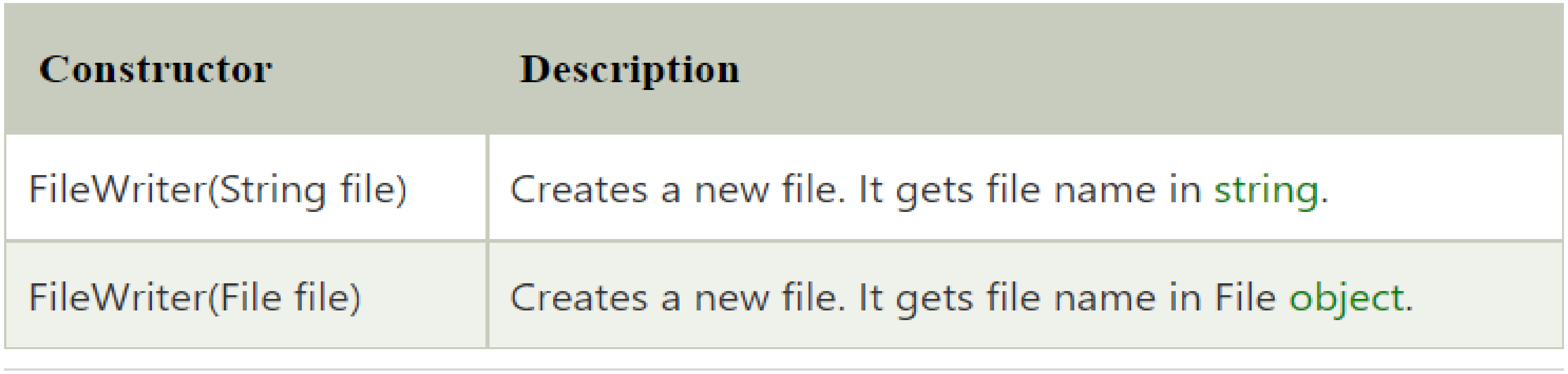
- In Java, we can create our own exceptions that are derived classes of the Exception class. Creating our own Exception is known as custom exception or user-defined exception. Basically, Java custom exceptions are used to customize the exception according to user need.
- Using the custom exception, we can have your own exception and message.
- TestUserDefinedException1.java

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 FileWriter Class

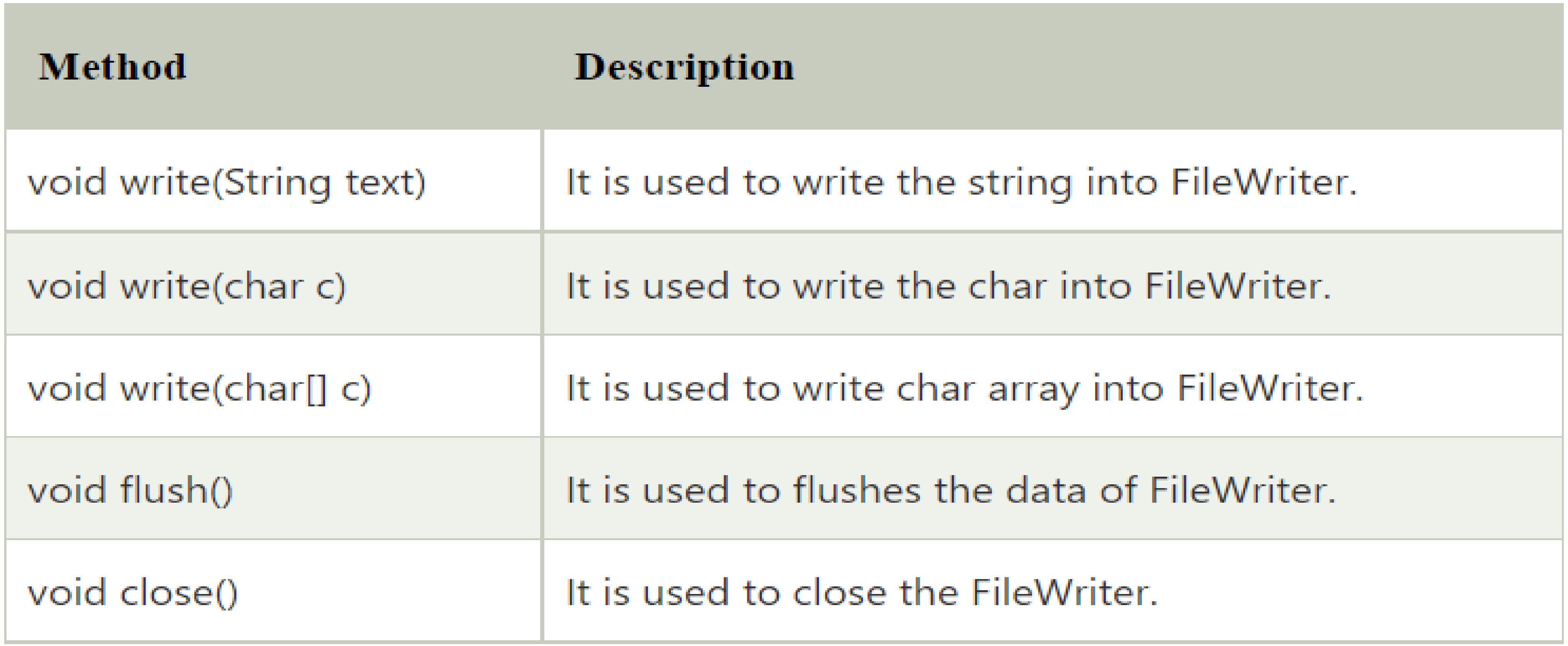
- Java FileWriter class is used to write character-oriented data to a file.
- It is character-oriented class which is used for file handling in java.
- Java has suggested not to use the FileInputStream and FileOutputStream classes if you have to read and write the textual information.
- Constructors of FileWriter class



Constructor	Description
FileWriter(String file)	Creates a new file. It gets file name in string.
FileWriter(File file)	Creates a new file. It gets file name in File object.

Java FileWriter Class

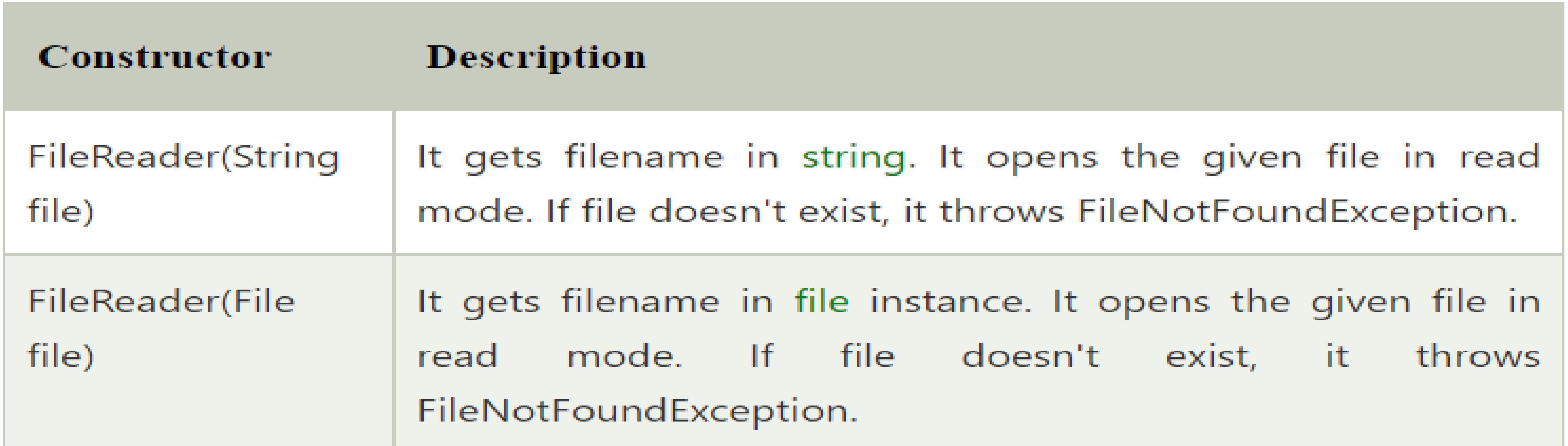
- Methods of FileWriter class
- FileWriterExample.java



Method	Description
void write(String text)	It is used to write the string into FileWriter.
void write(char c)	It is used to write the char into FileWriter.
void write(char[] c)	It is used to write char array into FileWriter.
void flush()	It is used to flushes the data of FileWriter.
void close()	It is used to close the FileWriter.

Java FileReader Class

- Java FileReader class is used to read data from the file.
- It returns data in byte format like FileInputStream class.
- It is character-oriented class which is used for file handling in java.
- Constructors of FileReader class

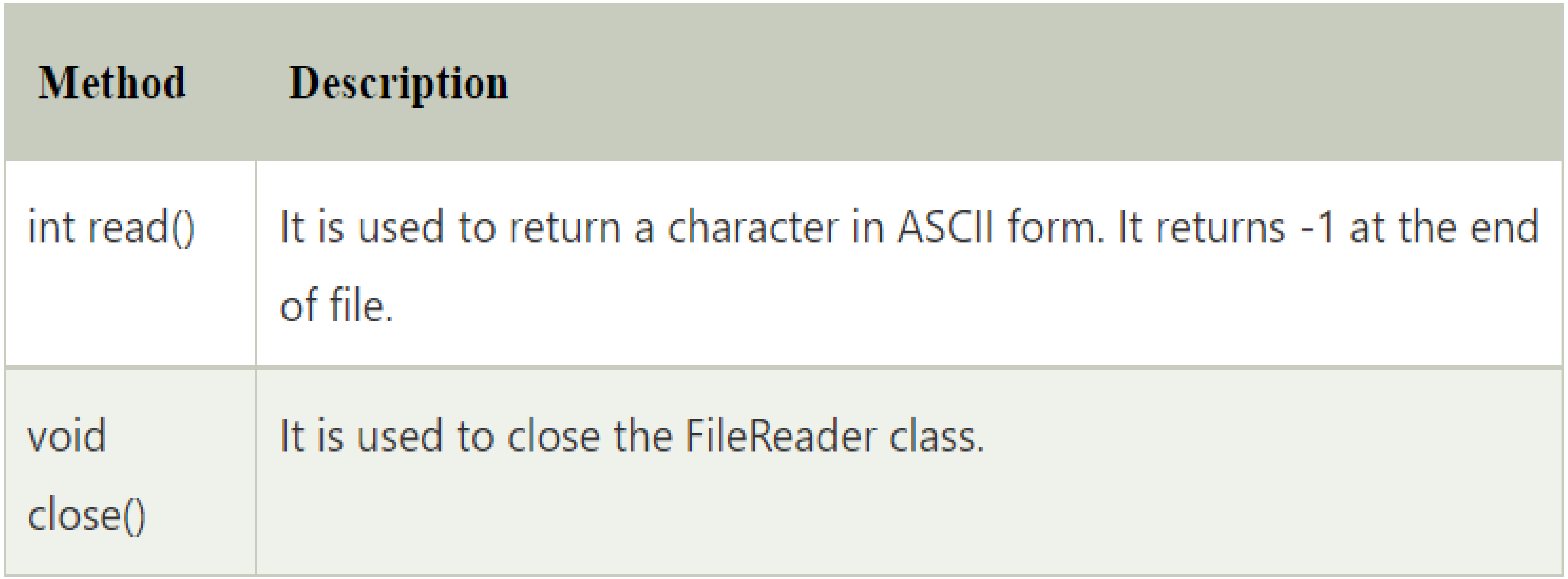


Constructor	Description
FileReader(String file)	It gets filename in string. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.
FileReader(File file)	It gets filename in file instance. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.

Java FileReader Class

Methods of FileReader class

- FileReaderExample.java



Method	Description
int read()	It is used to return a character in ASCII form. It returns -1 at the end of file.
void close()	It is used to close the FileReader class.

Java Stream

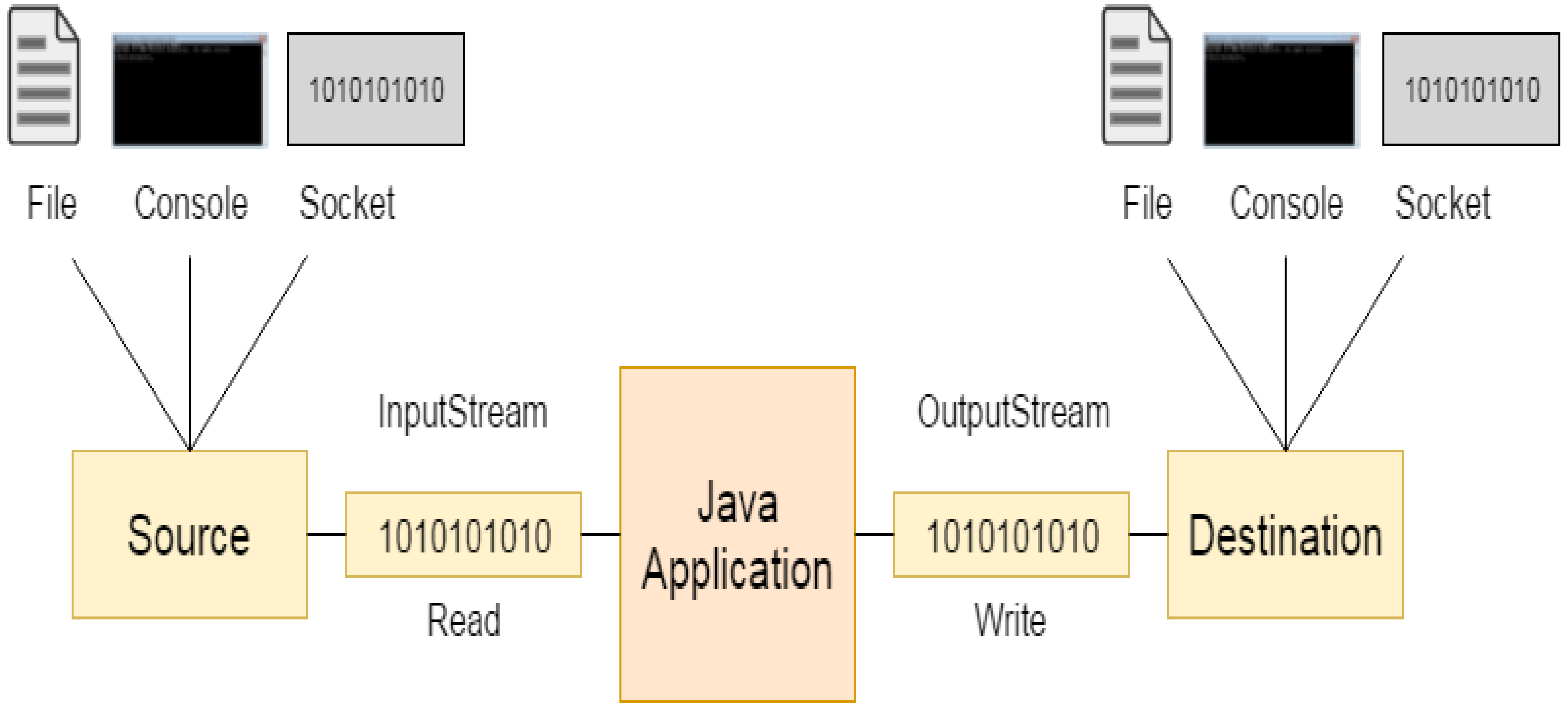
- A stream is a sequence of data.
- In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.
- In Java, 3 streams are created for us automatically. All these streams are attached with the console.

1. **System.out:** standard output stream
2. **System.in:** standard input stream
3. **System.err:** standard error stream

OutputStream vs InputStream

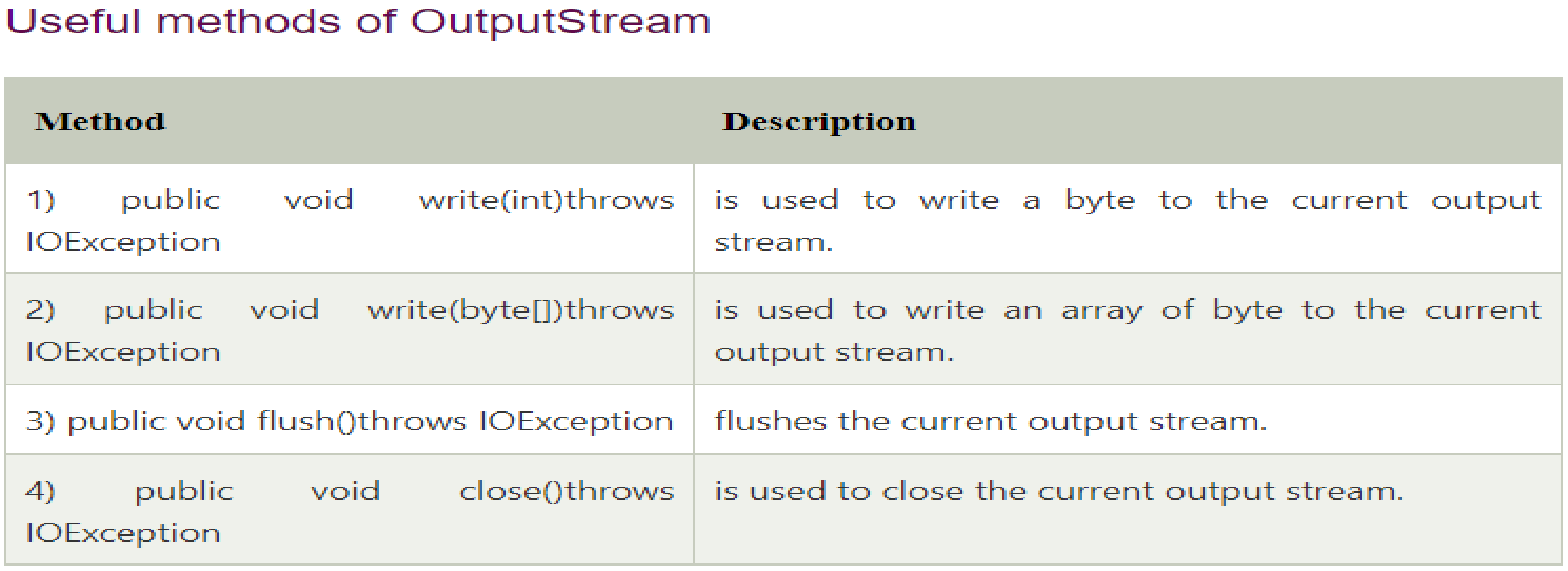
- OutputStream
- Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.
- InputStream
- Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.

OutputStream vs InputStream



OutputStream vs InputStream

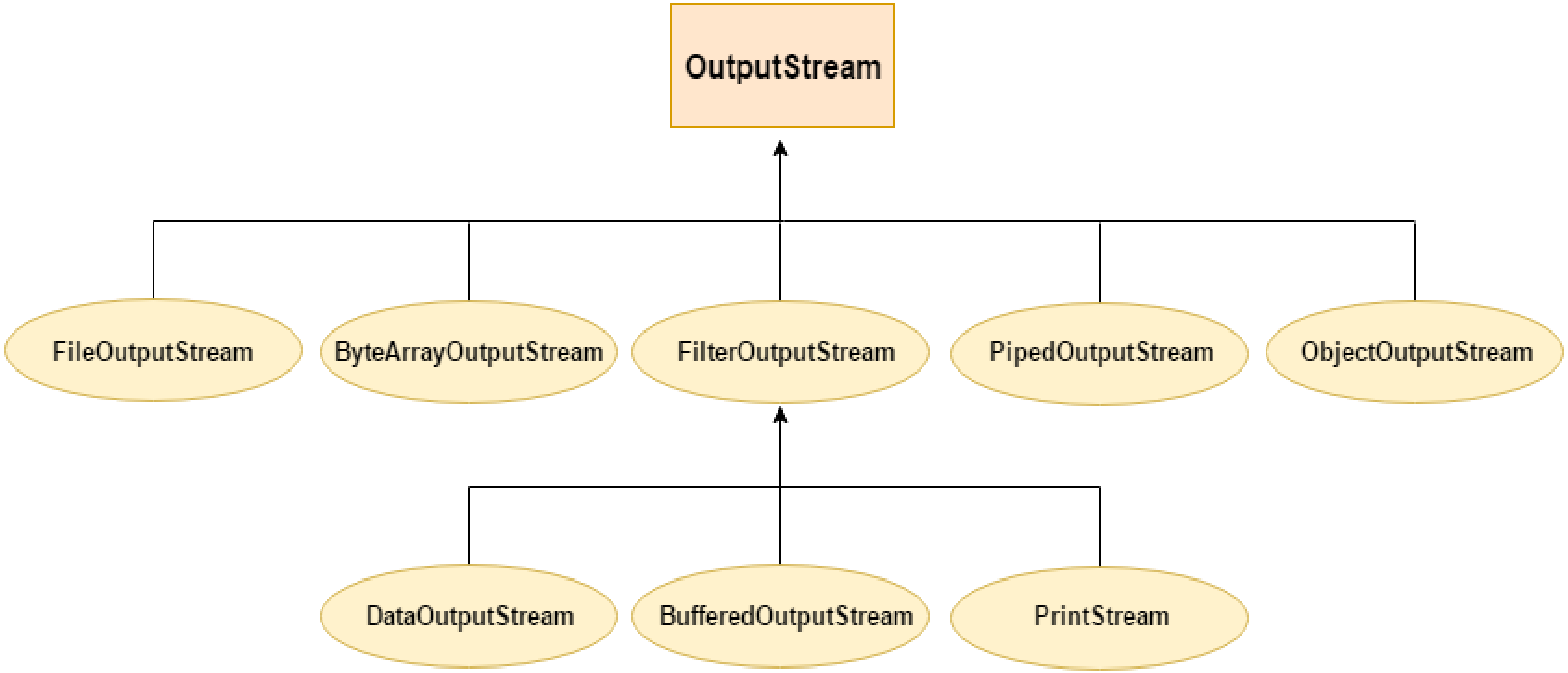
- OutputStream class
- OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.



Useful methods of OutputStream

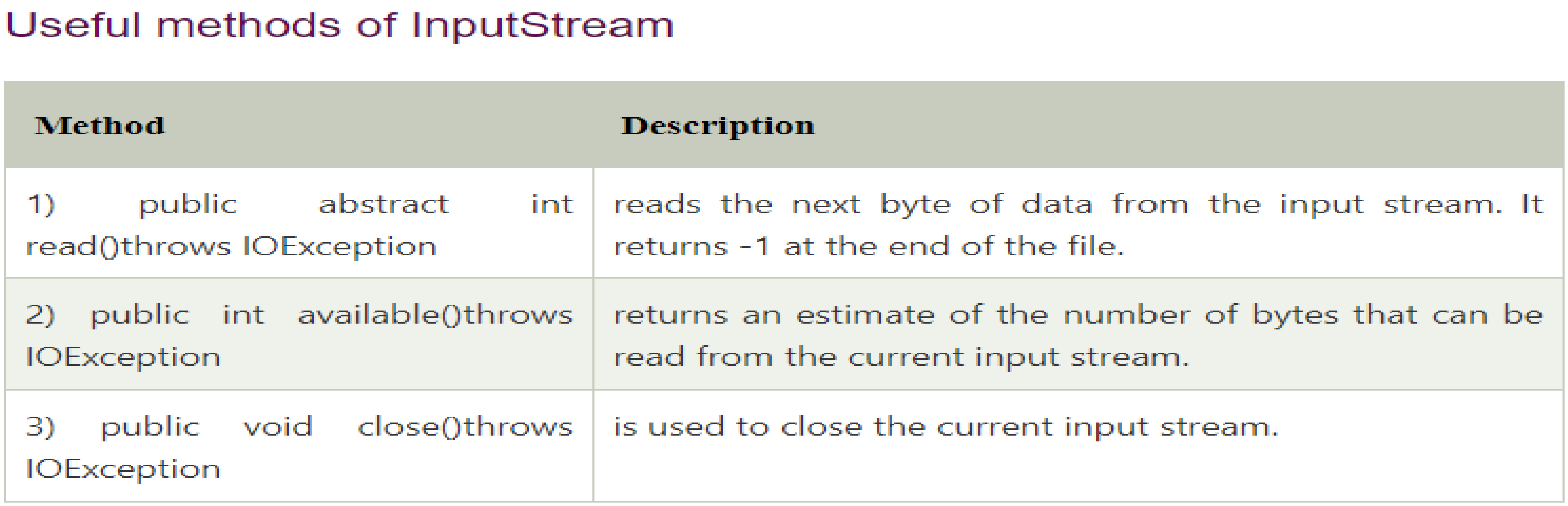
Method	Description
1) public void write(int)throws IOException	is used to write a byte to the current output stream.
2) public void write(byte[])throws IOException	is used to write an array of byte to the current output stream.
3) public void flush()throws IOException	flushes the current output stream.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

OutputStream vs InputStream



OutputStream vs InputStream

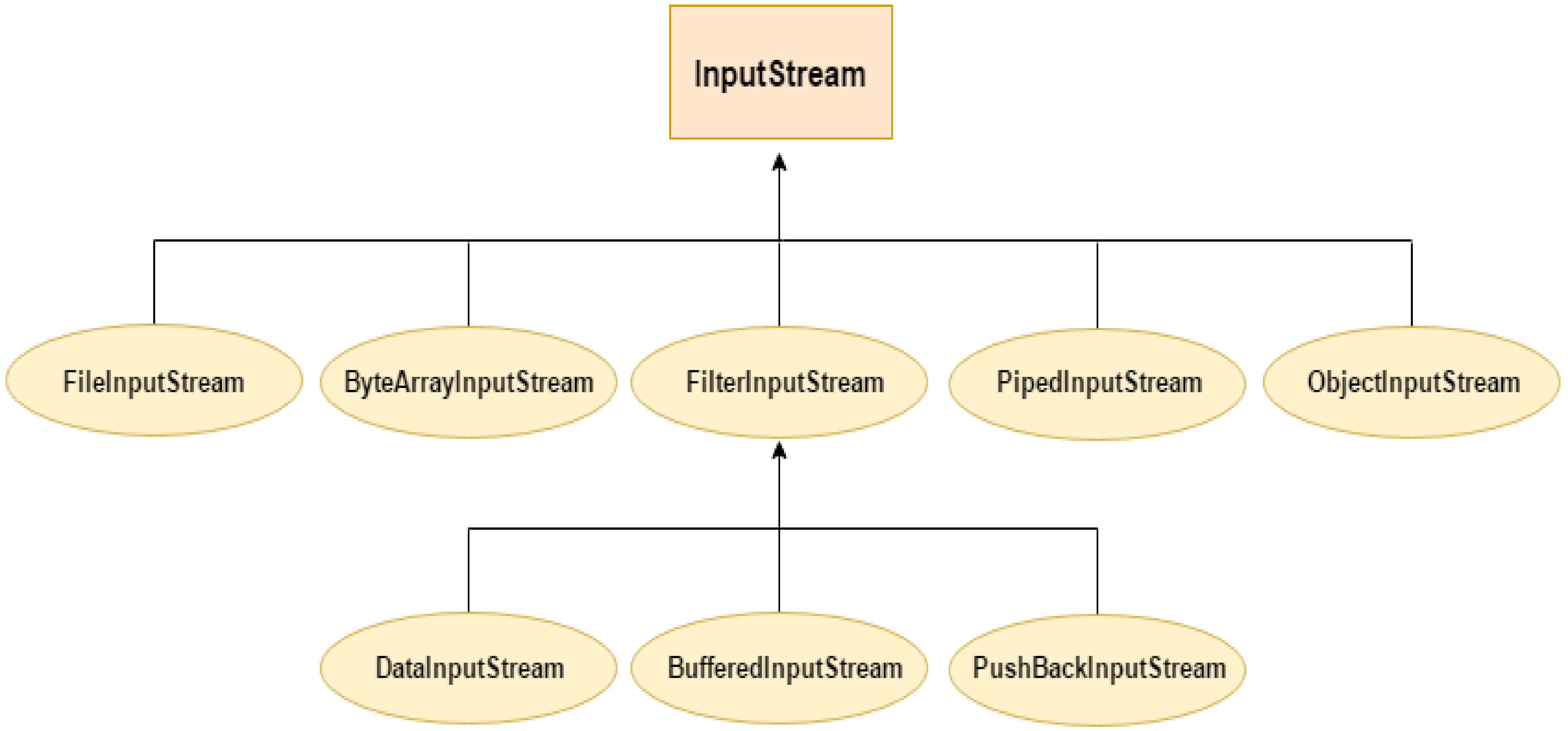
- InputStream class
- InputStream class is an abstract class. It is the superclass of all classes representing an input stream of bytes.



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.

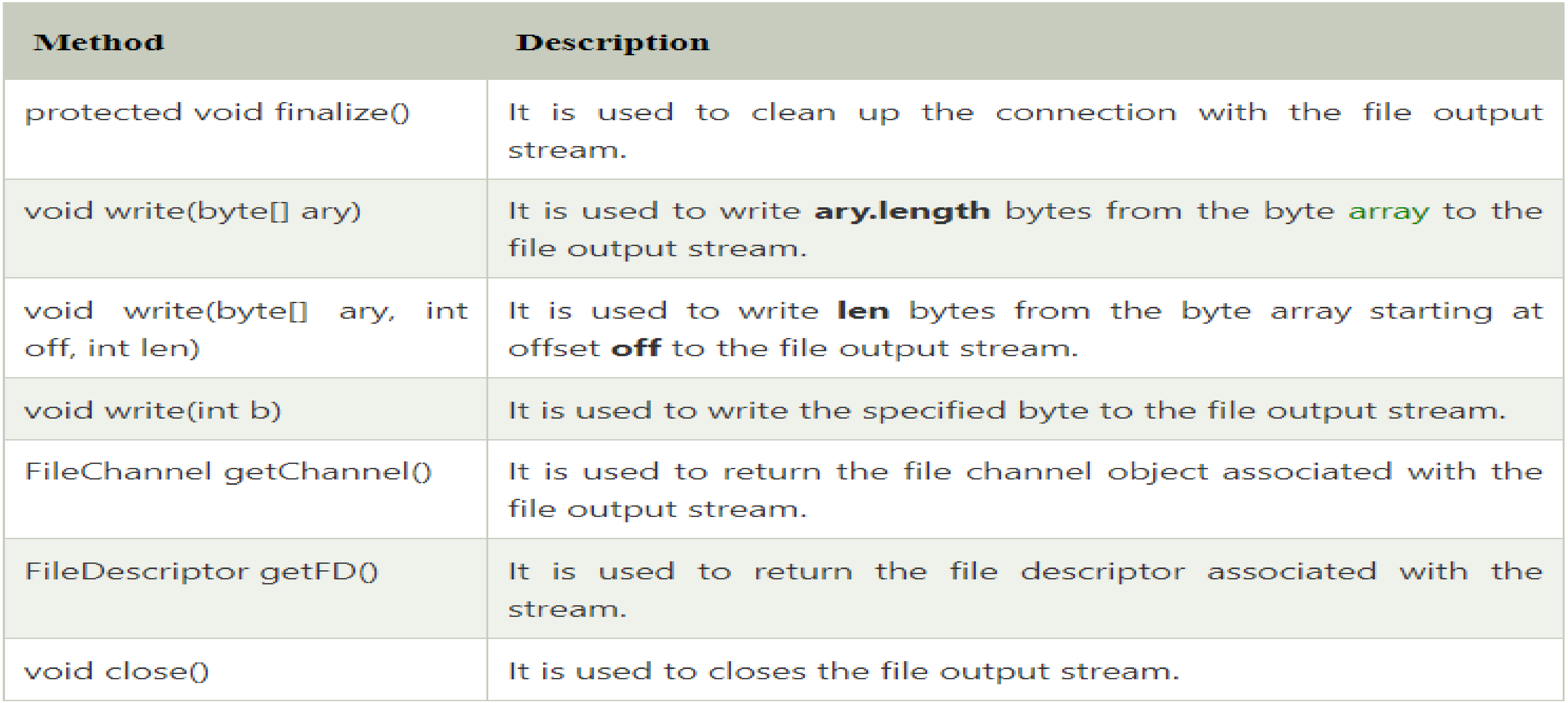
OutputStream vs InputStream



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.
- FileOutputStreamExample.java (Write Byte)
- FileOutputStreamExample1.java (Write string)

FileOutputStream class methods

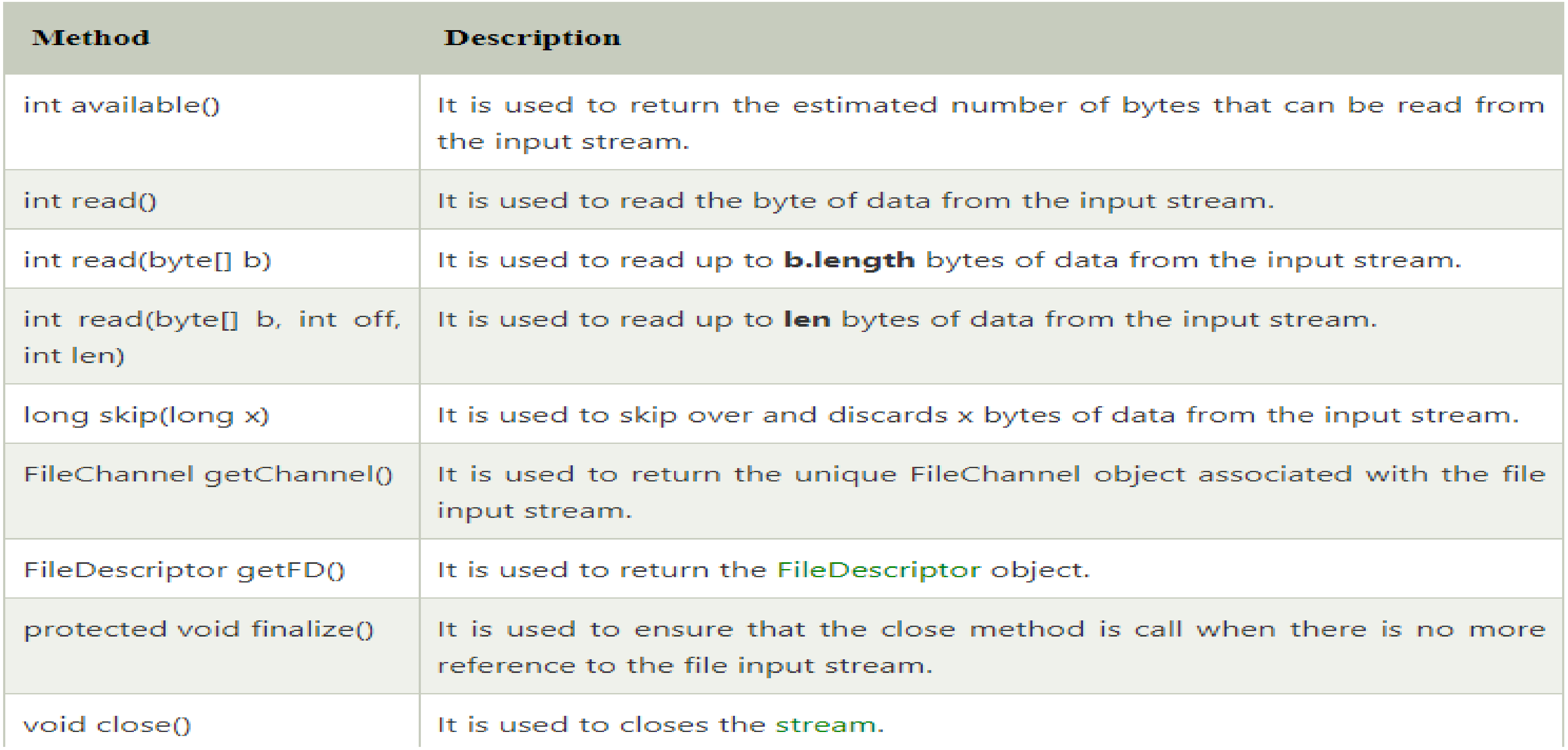


Method	Description
protected void finalize()	It is used to clean up the connection with the file output stream.
void write(byte[] ary)	It is used to write **ary.length** bytes from the byte array to the file output stream.
void write(byte[] ary, int off, int len)	It is used to write **len** bytes from 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 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.
- DataStreamExample.java (read single character)
- DataStreamExample1.java (read all characters)

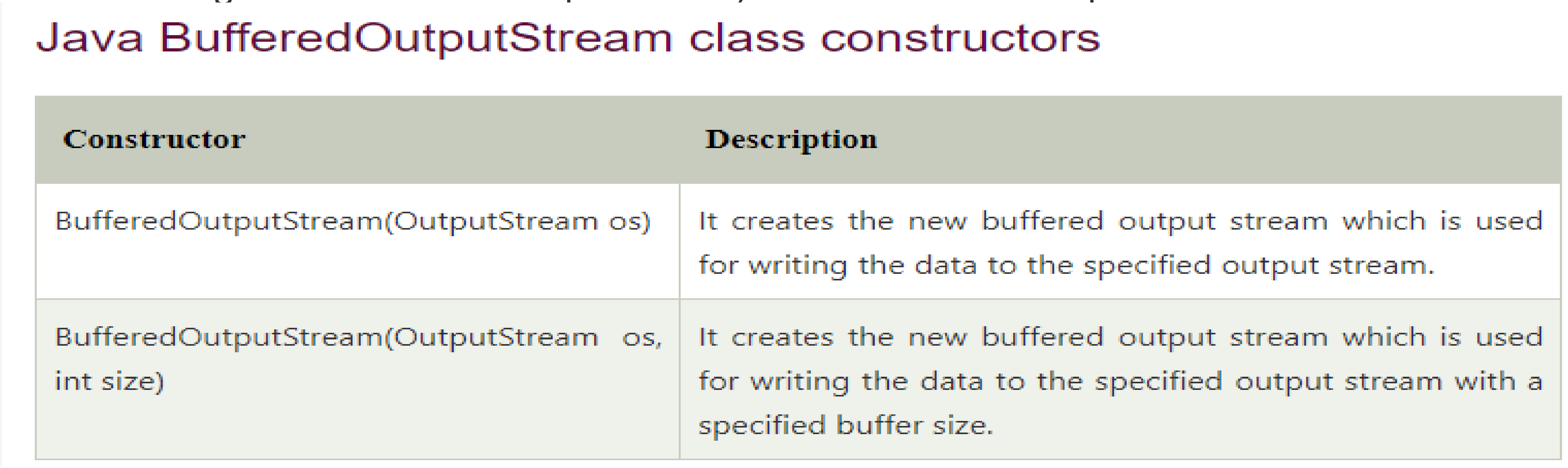
Java FileInputStream Class



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 BufferedOutputStream Class

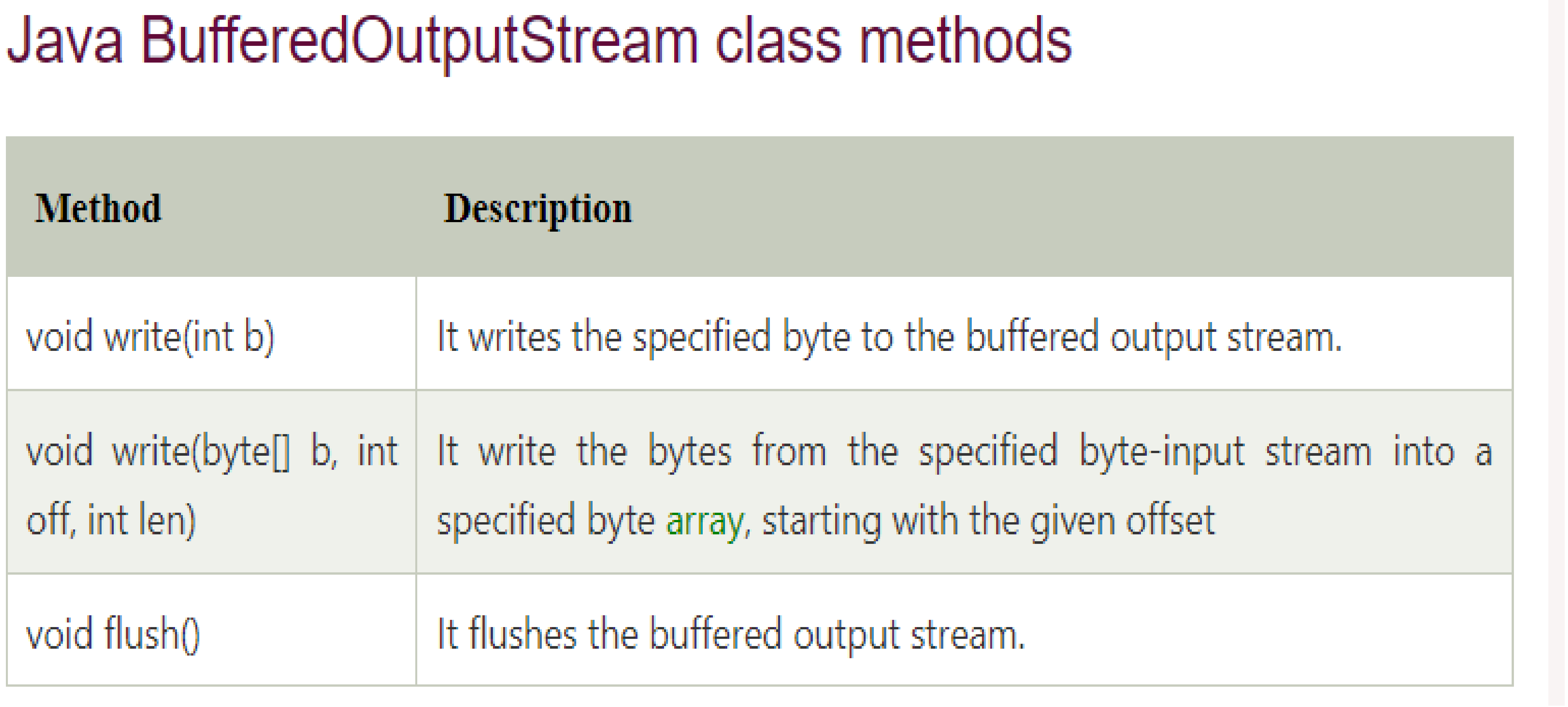
- Java BufferedOutputStream class is used for buffering an output stream.
- It internally uses buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast.
- For adding the buffer in an OutputStream, use the BufferedOutputStream class.



Java BufferedOutputStream class constructors

Constructor	Description
BufferedOutputStream(OutputStream os)	It creates the new buffered output stream which is used for writing the data to the specified output stream.
BufferedOutputStream(OutputStream os, int size)	It creates the new buffered output stream which is used for writing the data to the specified output stream with a specified buffer size.

Java BufferedOutputStream Class



Java BufferedOutputStream class methods

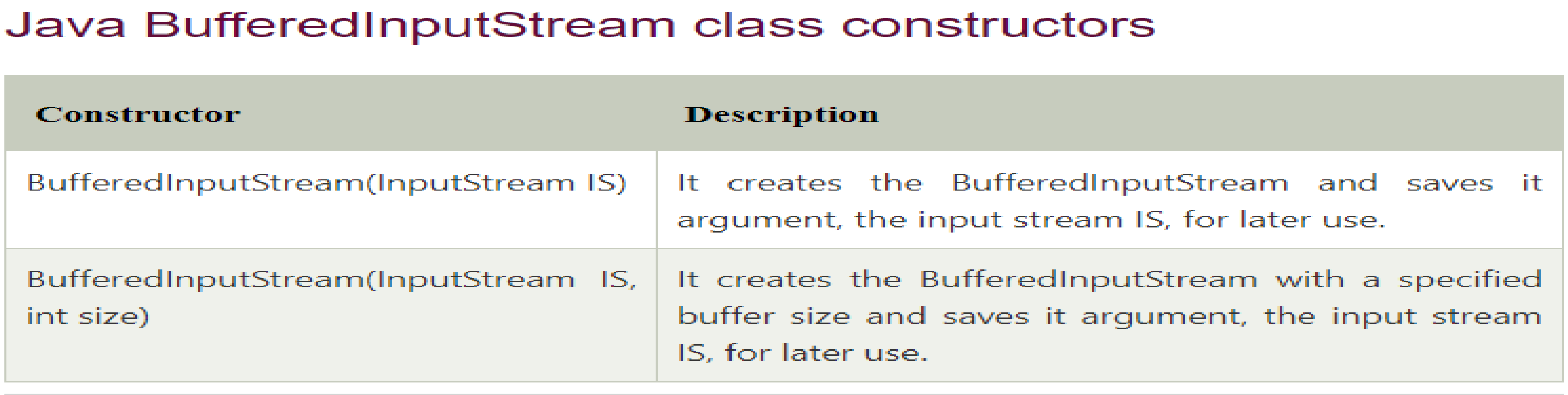
Method	Description
void write(int b)	It writes the specified byte to the buffered output stream.
void write(byte[] b, int off, int len)	It write the bytes from the specified byte-input stream into a specified byte array, starting with the given offset
void flush()	It flushes the buffered output stream.

Java BufferedOutputStream Class

- Example of BufferedOutputStream class:
- In this example, we are writing the textual information in the BufferedOutputStream object which is connected to the FileOutputStream object.
- The flush() flushes the data of one stream and send it into another. It is required if you have connected the one stream with another.
- BufferedOutputStreamExample.java

Java BufferedInputStream Class

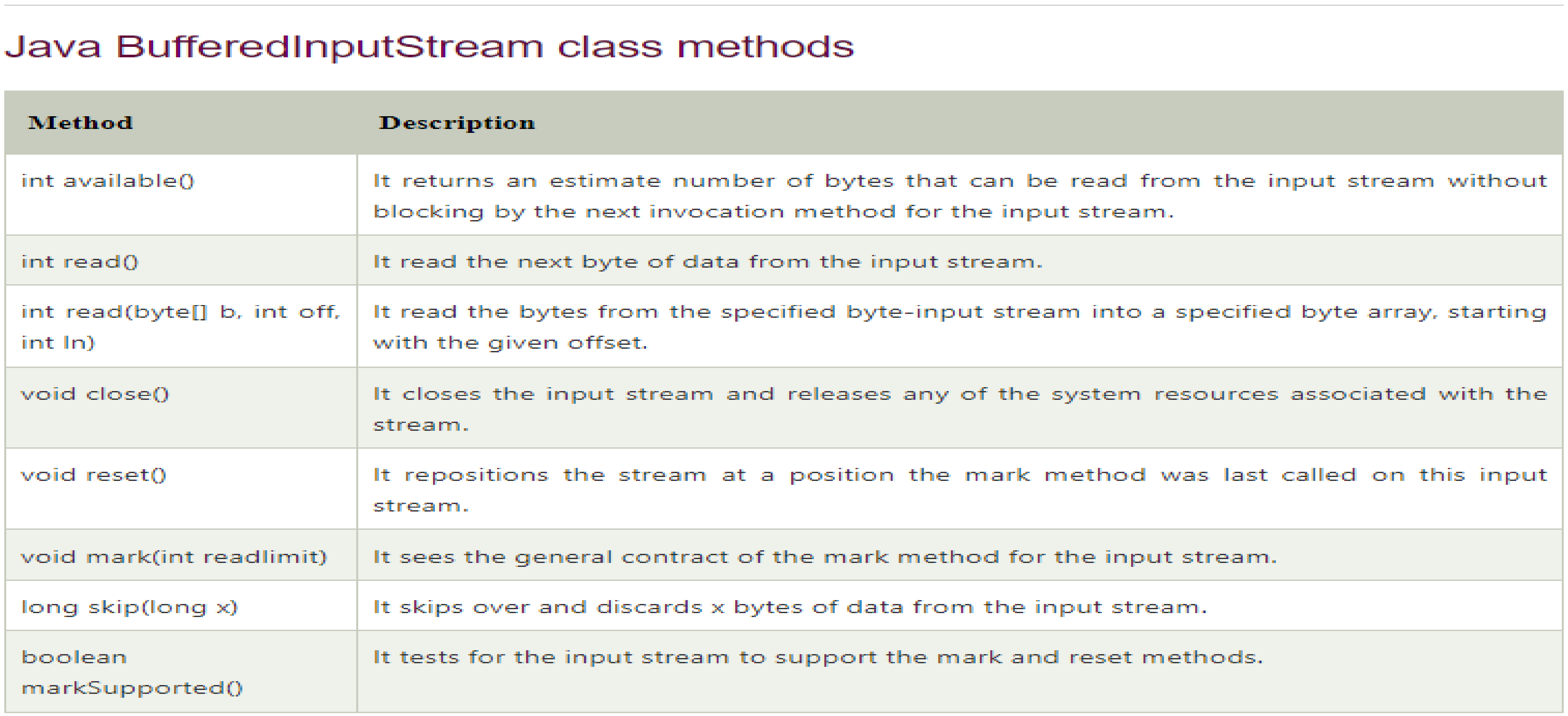
- Java BufferedInputStream class is used to read information from stream. It internally uses buffer mechanism to make the performance fast.
- The important points about BufferedInputStream are:
- When the bytes from the stream are skipped or read, the internal buffer automatically refilled from the contained input stream, many bytes at a time.
- When a BufferedInputStream is created, an internal buffer array is created.



Java BufferedInputStream class constructors

Constructor	Description
BufferedInputStream(InputStream IS)	It creates the BufferedInputStream and saves it argument, the input stream IS, for later use.
BufferedInputStream(InputStream IS, int size)	It creates the BufferedInputStream with a specified buffer size and saves it argument, the input stream IS, for later use.

Java BufferedInputStream Class



Java BufferedInputStream class methods

Method	Description
int available()	It returns an estimate number of bytes that can be read from the input stream without blocking by the next invocation method for the input stream.
int read()	It read the next byte of data from the input stream.
int read(byte[] b, int off, int ln)	It read the bytes from the specified byte-input stream into a specified byte array, starting with the given offset.
void close()	It closes the input stream and releases any of the system resources associated with the stream.
void reset()	It repositions the stream at a position the mark method was last called on this input stream.
void mark(int readlimit)	It sees the general contract of the mark method for the input stream.
long skip(long x)	It skips over and discards x bytes of data from the input stream.
boolean markSupported()	It tests for the input stream to support the mark and reset methods.