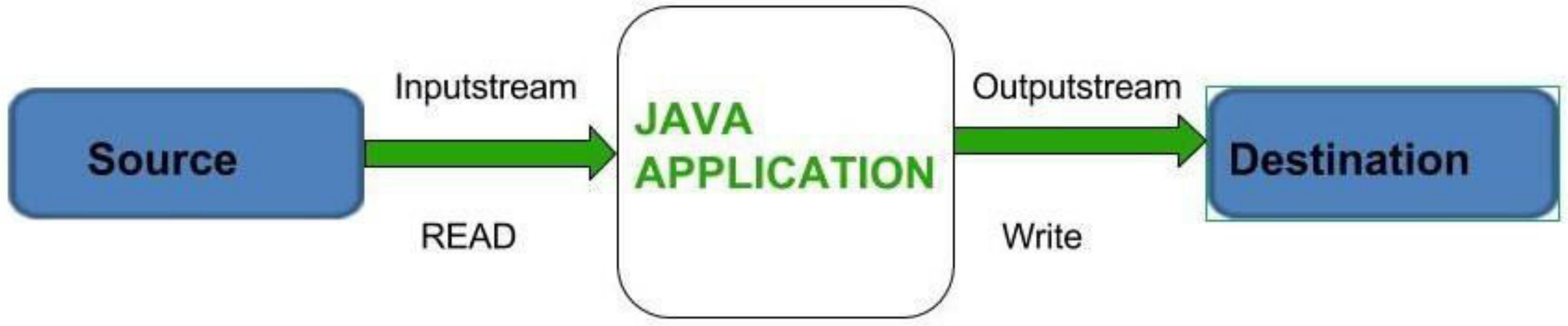
I/O Programming

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 brings various Streams with its I/O package that helps the user to perform all the input-output operations. These streams support all the types of objects, data-types, characters, files etc to fully execute the I/O operations.

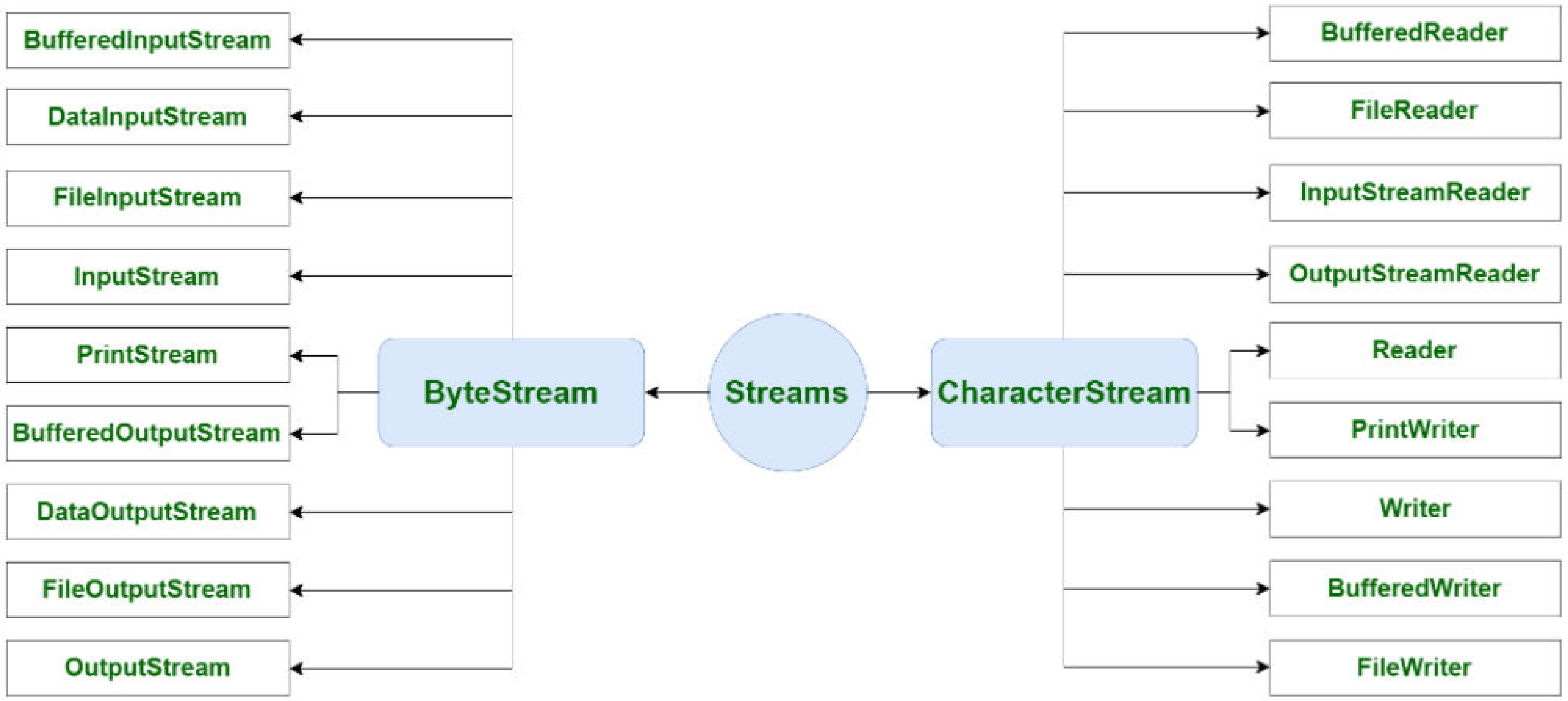
Stream
A stream can be defined as a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow. There are two kinds of Streams −
•**InPutStream** − The InputStream is used to read data from a source.
•**OutPutStream** − The OutputStream is used for writing data to a destination.



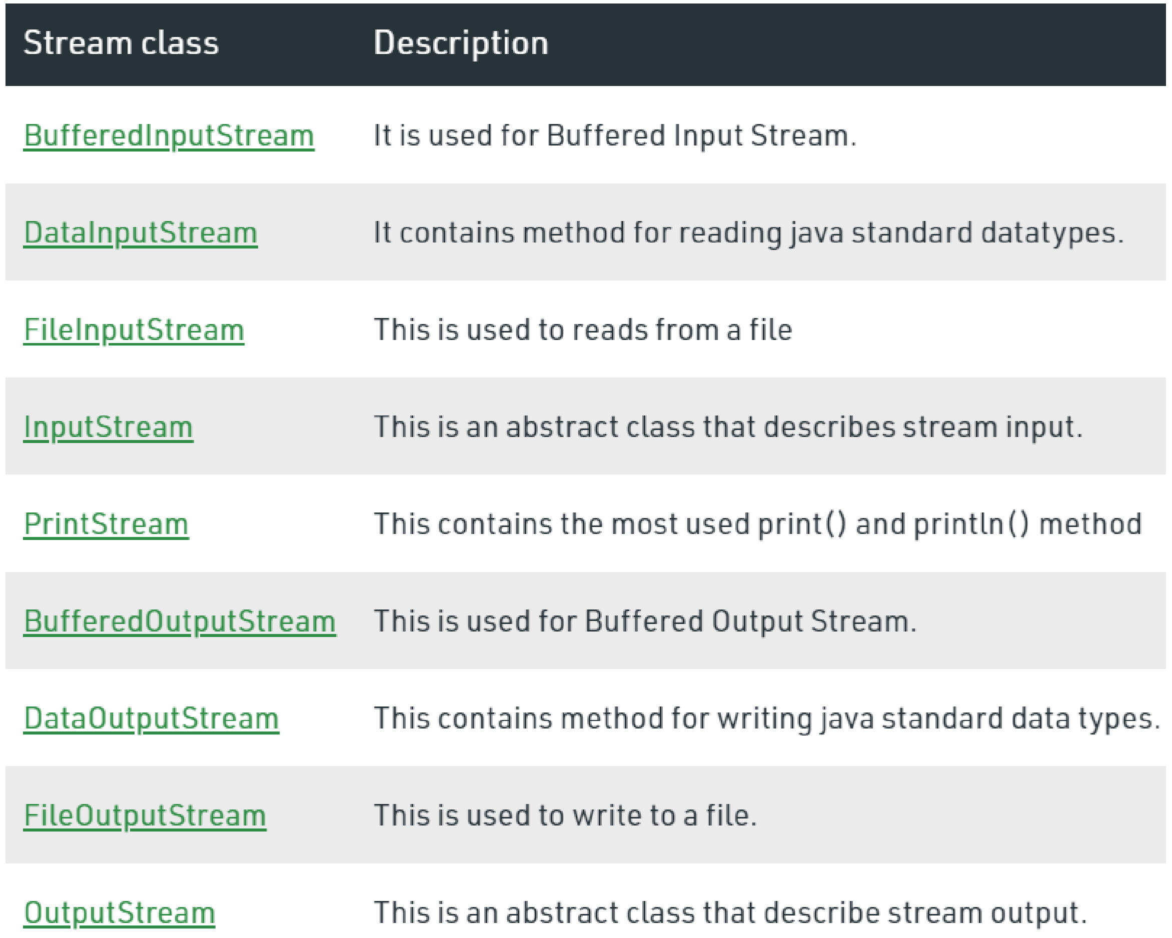
Types of Streams:

- **Depending on the type of operations**, streams can be divided into two primary classes:
  - Input Stream: These streams are used to read data that must be taken as an input from a source array or file or any peripheral device. For eg., FileInputStream, BufferedInputStream, ByteArrayInputStream etc.
- Output Stream: These streams are used to write data as outputs into an array or file or monitor or any output peripheral device. For eg., FileOutputStream, BufferedOutputStream, ByteArrayOutputStream etc.

Depending on the types of file(the data a stream holds), Streams can be divided into two primary classes: ByteStream,CharacterStream.

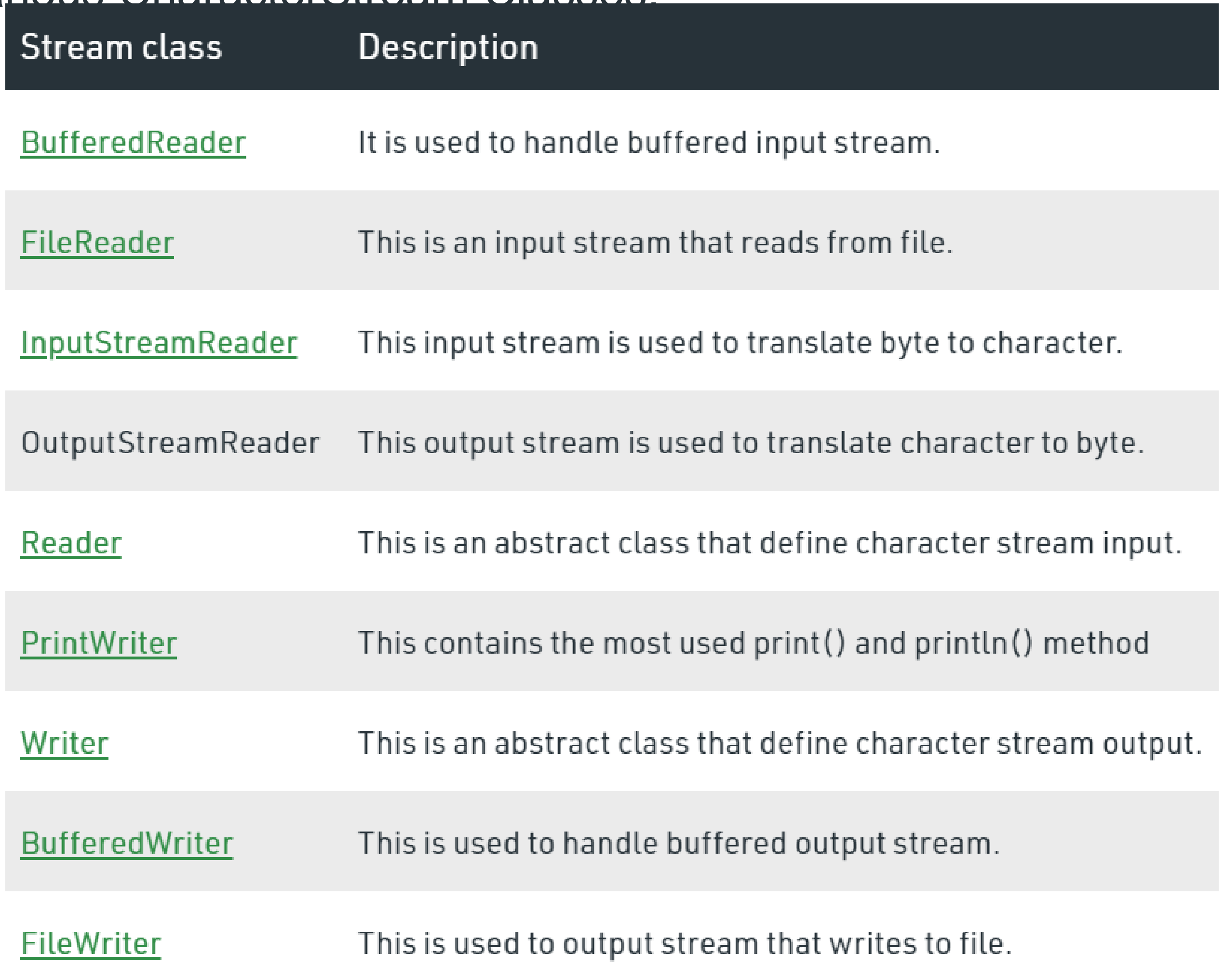


ByteStream: This is used to process data byte by byte (8 bits). Though it has many classes, the FileInputStream and the FileOutputStream are the most popular ones. The FileInputStream is used to read from the source and FileOutputStream is used to write to the destination.
Byte stream is used to read and write a single byte (8 bits) of data.
All byte stream classes are derived from base abstract classes called InputStream and OutputStream.
Here is the list of various ByteStream Classes:



Stream class	Description
BufferedInputStream	It is used for Buffered Input Stream.
DataInputStream	It contains method for reading java standard datatypes.
FileInputStream	This is used to reads from a file
InputStream	This is an abstract class that describes stream input.
PrintStream	This contains the most used print() and println() method
BufferedOutputStream	This is used for Buffered Output Stream.
DataOutputStream	This contains method for writing java standard data types.
FileOutputStream	This is used to write to a file.
OutputStream	This is an abstract class that describe stream output.

CharacterStream: In Java, characters are stored using Unicode conventions . Character stream automatically allows us to read/write data character by character. Though it has many classes, the FileReader and the FileWriter are the most popular ones. FileReader and FileWriter are character streams used to read from the source and write to the destination respectively.
Character stream is used to read and write a single character of data.
All the character stream classes are derived from base abstract classes **Reader and Writer.**
Here is the list of various CharacterStream Classes:



Stream class	Description
BufferedReader	It is used to handle buffered input stream.
FileReader	This is an input stream that reads from file.
InputStreamReader	This input stream is used to translate byte to character.
OutputStreamReader	This output stream is used to translate character to byte.
Reader	This is an abstract class that define character stream input.
PrintWriter	This contains the most used print() and println() method
Writer	This is an abstract class that define character stream output.
BufferedWriter	This is used to handle buffered output stream.
FileWriter	This is used to output stream that writes to file.

FileInputStream

This class reads the data from a specific file (byte by byte). It is usually used to read the contents of a file with raw bytes, such as images.

To read the contents of a file using this class −
•First of all, you need to instantiate this class by passing a String variable(path of file) or a File object, representing the path of the file to be read.

```
FileInputStream inputStream = new FileInputStream("file_path");
or,
File file = new File("file_path");
FileInputStream inputStream = new FileInputStream(file);
```

•Then read the contents of the specified file using either of the variants of read() method −

- **int read()** − This simply reads data from the current InputStream and returns the read data byte by byte (in integer format).
- This method returns -1 if the end of the file is reached.
- **int read(byte[] b)** − This method accepts a byte array as parameter and reads the contents of the current InputStream, to the given array
- This method returns an integer representing the total number of bytes or, -1 if the end of the file is reached.
- **int read(byte[] b, int off, int len)** − This method accepts a byte array, its offset (int) and, its length (int) as parameters and reads the contents of the current InputStream, to the given array.
- This method returns an integer representing the total number of bytes or, -1 if the end of the file is reached.

read() Method:

- read() - reads a single byte from the file
- read(byte[] array) - reads the bytes from the file and stores in the specified array
- read(byte[] array, int start, int length) - reads the number of bytes equal to length from the file and stores in the specified array starting from the position start.

```java
import java.io.FileInputStream;

public class Main {

  public static void main(String args[]) {

     try {
        FileInputStream input = new FileInputStream("input.txt");

        System.out.println("Data in the file: ");

        // Reads the first byte
        int i = input.read();

       while(i != -1) {
           System.out.print((char)i);

           // Reads next byte from the file
           i = input.read();
        }
        input.close();
     }

     catch(Exception e) {
        e.getStackTrace();
     }
  }
}
```

```
Data in the file:
This is a line of text inside the file.
```

close() Method

To close the file input stream, we can use the close() method. Once the close() method is called, we cannot use the input stream to read data.

available() Method

To get the number of available bytes, we can use the available() method.

```java
FileInputStream input = new FileInputStream("input.txt");

// Returns the number of available bytes
System.out.println("Available bytes at the beginning: " + input.available());

// Reads 3 bytes from the file
input.read();
input.read();
input.read();

// Returns the number of available bytes
System.out.println("Available bytes at the end: " + input.available());
```

```
Available bytes at the beginning: 39
Available bytes at the end: 36
```

skip() Method

To discard and skip the specified number of bytes, we can use the skip() method.

```java
FileInputStream input = new FileInputStream("input.txt");

// Skips the 5 bytes
input.skip(5);
System.out.println("Input stream after skipping 5 bytes:");

// Reads the first byte
int i = input.read();
while (i != -1) {
   System.out.print((char) i);

   // Reads next byte from the file
   i = input.read();
}
```

```
Input Stream after skipping 5 bytes:
is a line of text inside the file.
```

FileOutputStream class

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.

```
import java.io.FileOutputStream;
public class FileOutputStreamExample {
   public static void main(String args[]){
          try{
            FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
            String s="Welcome to javaTpoint.";
            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);}
     }
}
```

FileReader

FileReader is useful to read data in the form of characters from a ‘text’ file.

•This class inherit from the Reader Class.

•FileReader is meant for reading streams of characters. For reading streams of raw bytes, consider using a FileInputStream.

Constructors:

•**FileReader(File file) –** Creates a FileReader , given the File to read from

•**FileReader(String fileName) –** Creates a new FileReader , given the name of the file to read from

Methods:

•**public int read () throws IOException –** Reads a single character. This method will block until a character is available, an I/O error occurs, or the end of the stream is reached.

•**public int read(char[] cbuff) throws IOException –** Reads characters into an array. This method will block until some input is available, an I/O error occurs, or the end of the stream is reached.

•**public abstract int read(char[] buff, int off, int len) throws IOException –**Reads characters into a portion of an array. This method will block until some input is available, an I/O error occurs, or the end of the stream is reached.

Parameters:

cbuf – Destination buffer

off – Offset at which to start storing characters

len – Maximum number of characters to read

•**public void close() throws IOException** closes the reader.

•**public long skip(long n) throws IOException –**Skips characters. This method will block until some characters are available, an I/O error occurs, or the end of the stream is reached.

Parameters:

n – The number of characters to skip

```java
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
class ReadFile
{
    public static void main(String[] args) throws IOException
    {
        // variable declaration
        int ch;

        // check if File exists or not
        FileReader fr=null;
        try
        {
            fr = new FileReader("text");
        }
        catch (FileNotFoundException fe)
        {
            System.out.println("File not found");
        }

        // read from FileReader till the end of file
        while ((ch=fr.read())!=-1)
            System.out.print((char)ch);

        // close the file
        fr.close();
    }
}
```

FileWriter

FileWriter is useful to create a file writing characters into it.
•This class inherits from the Writer class.
•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.

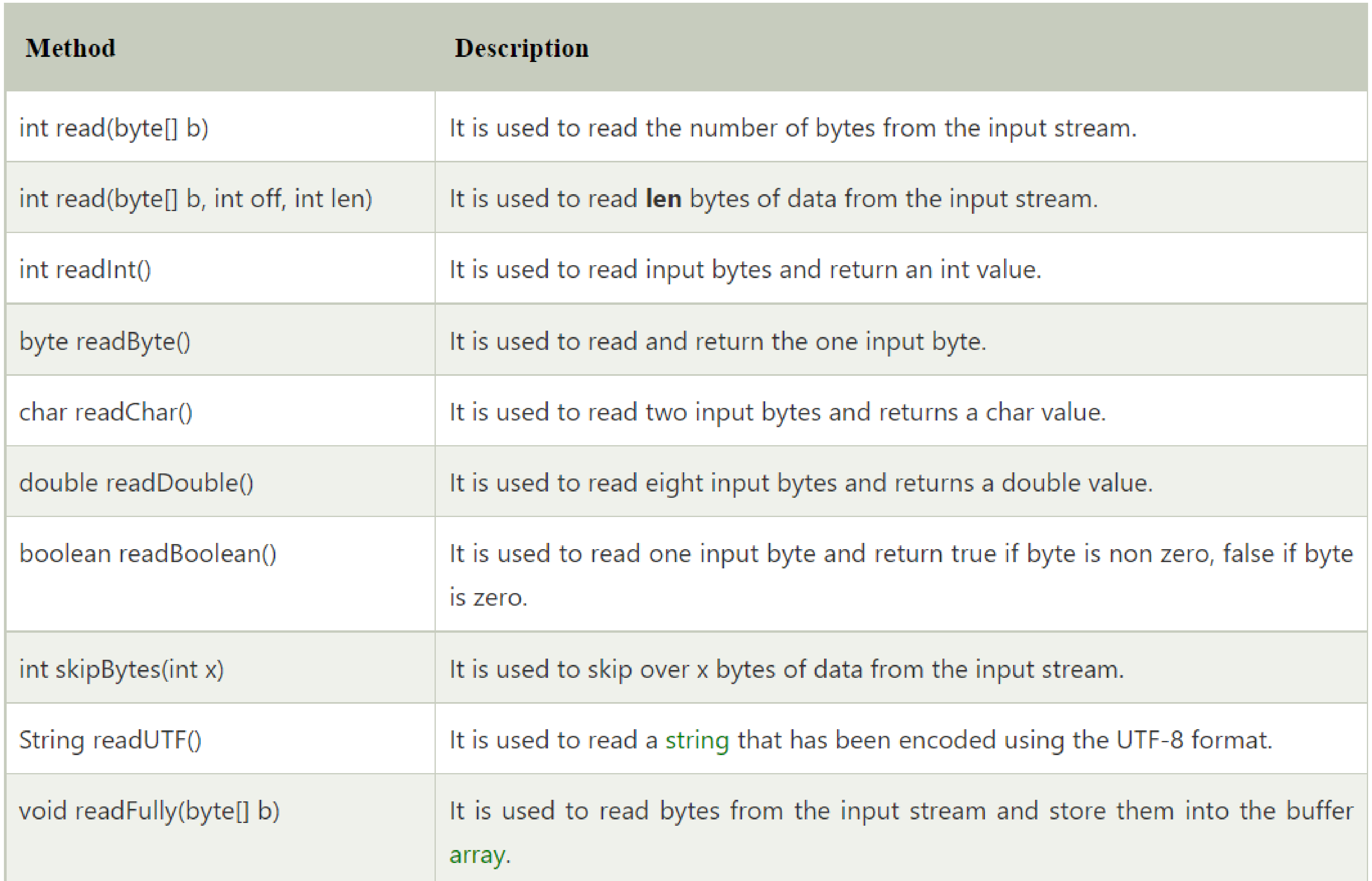
```java
import java.io.FileWriter;
public class FileWriterExample {
    public static void main(String args[]){
        try{
          FileWriter fw=new FileWriter("D:\\testout.txt");
          fw.write("Welcome to javaTpoint.");
          fw.close();
        }catch(Exception e){System.out.println(e);}
        System.out.println("Success...");
    }
}
```

```java
try (
    FileReader reader = new FileReader("Notes1.txt");
    FileWriter writer = new FileWriter("Notes2.txt");
) {

    int charRead = -1;

    while ((charRead = reader.read()) != -1) {
        writer.write(charRead);
    }

} catch (IOException ex) {
    System.err.println(ex);
}
```

Java DataInputStream Class

Java DataInputStream class allows an application to read primitive data from the input stream in a machine-independent way.

Java DataInputStream class Methods

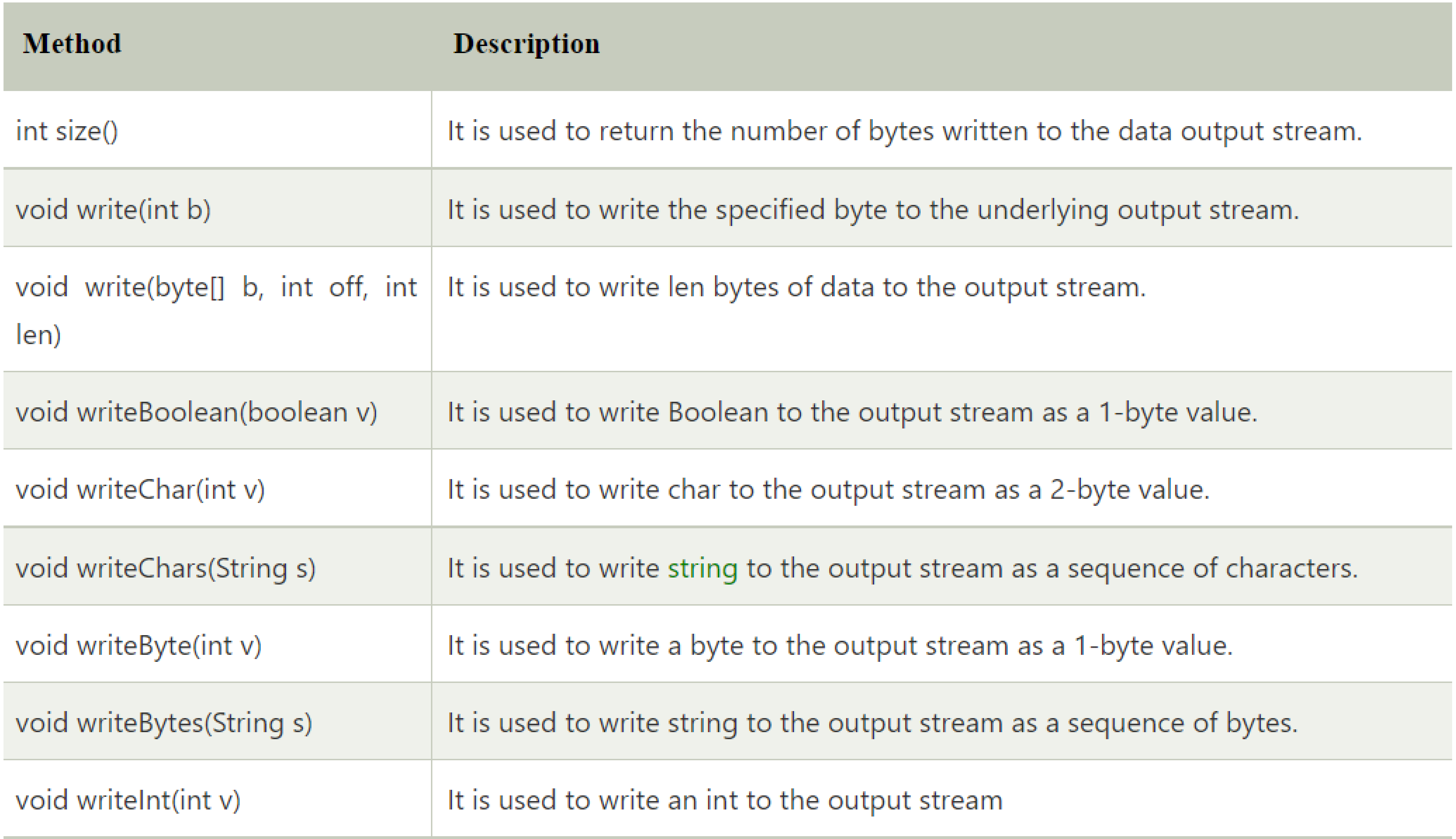


Method	**Description**
int read(byte[] b)	It is used to read the number of bytes from the input stream.
int read(byte[] b, int off, int len)	It is used to read **len** bytes of data from the input stream.
int readInt()	It is used to read input bytes and return an int value.
byte readByte()	It is used to read and return the one input byte.
char readChar()	It is used to read two input bytes and returns a char value.
double readDouble()	It is used to read eight input bytes and returns a double value.
boolean readBoolean()	It is used to read one input byte and return true if byte is non zero, false if byte is zero.
int skipBytes(int x)	It is used to skip over x bytes of data from the input stream.
String readUTF()	It is used to read a string that has been encoded using the UTF-8 format.
void readFully(byte[] b)	It is used to read bytes from the input stream and store them into the buffer array.

```
class DataOutputStreamDemo
{
    public static void main(String args[]) throws IOException
    {
        //writing the data using DataOutputStream
        try ( DataOutputStream dout =
                    new DataOutputStream(new FileOutputStream("file.dat")) )
        {
            dout.writeDouble(1.1);
            dout.writeInt(55);
            dout.writeBoolean(true);
            dout.writeChar('4');
        }
        catch(FileNotFoundException ex)
        {
            System.out.println("Cannot Open the Output File");
            return;
        }

        // reading the data back using DataInputStream
        try ( DataInputStream din =
                    new DataInputStream(new FileInputStream("file.dat")) )
        {
            double a = din.readDouble();
            int b = din.readInt();
            boolean c = din.readBoolean();
            char d=din.readChar();
            System.out.println("Values: " + a + "  " + b + "  " + c+"  " + d);
        }
        catch(FileNotFoundException e)
        {
            System.out.println("Cannot Open the Input File");
            return;
        }
    }
}
Output:1.1 55 true 4
```

Java DataOutputStream Class

Java DataOutputStream class allows an application to write primitive Java data types to the output stream in a machine-independent way.



Method	Description
int size()	It is used to return the number of bytes written to the data output stream.
void write(int b)	It is used to write the specified byte to the underlying output stream.
void write(byte[] b, int off, int len)	It is used to write len bytes of data to the output stream.
void writeBoolean(boolean v)	It is used to write Boolean to the output stream as a 1-byte value.
void writeChar(int v)	It is used to write char to the output stream as a 2-byte value.
void writeChars(String s)	It is used to write string to the output stream as a sequence of characters.
void writeByte(int v)	It is used to write a byte to the output stream as a 1-byte value.
void writeBytes(String s)	It is used to write string to the output stream as a sequence of bytes.
void writeInt(int v)	It is used to write an int to the output stream

```
import java.io.*;
public class OutputExample {
    public static void main(String[] args) throws IOException {
        FileOutputStream file = new FileOutputStream(D:\\testout.txt);
        DataOutputStream data = new DataOutputStream(file);
        data.writeInt(65);
        data.flush();
        data.close();
        System.out.println("Succcess...");
    }
}
```

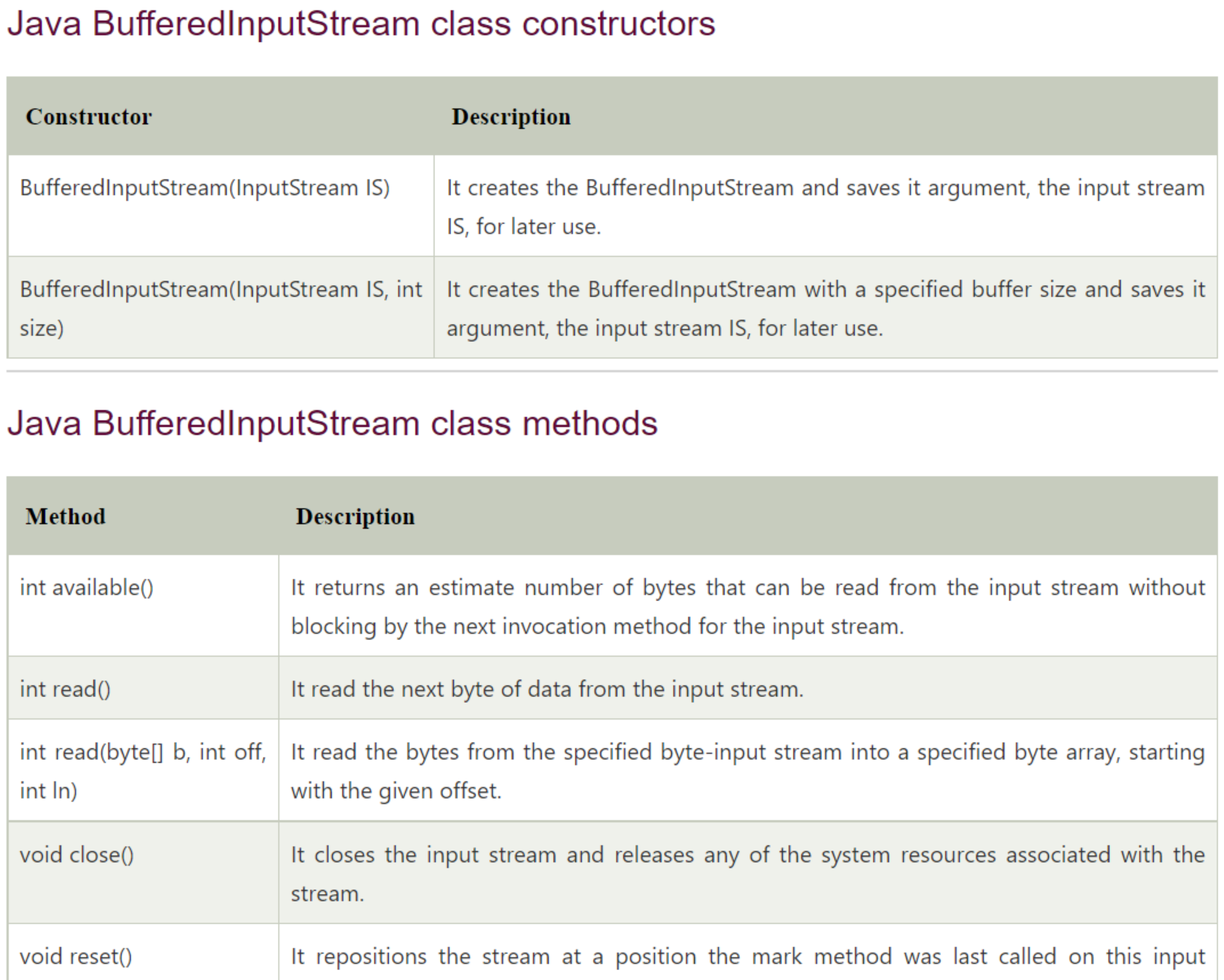
Java BufferedInputStream Class

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 arrav 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.

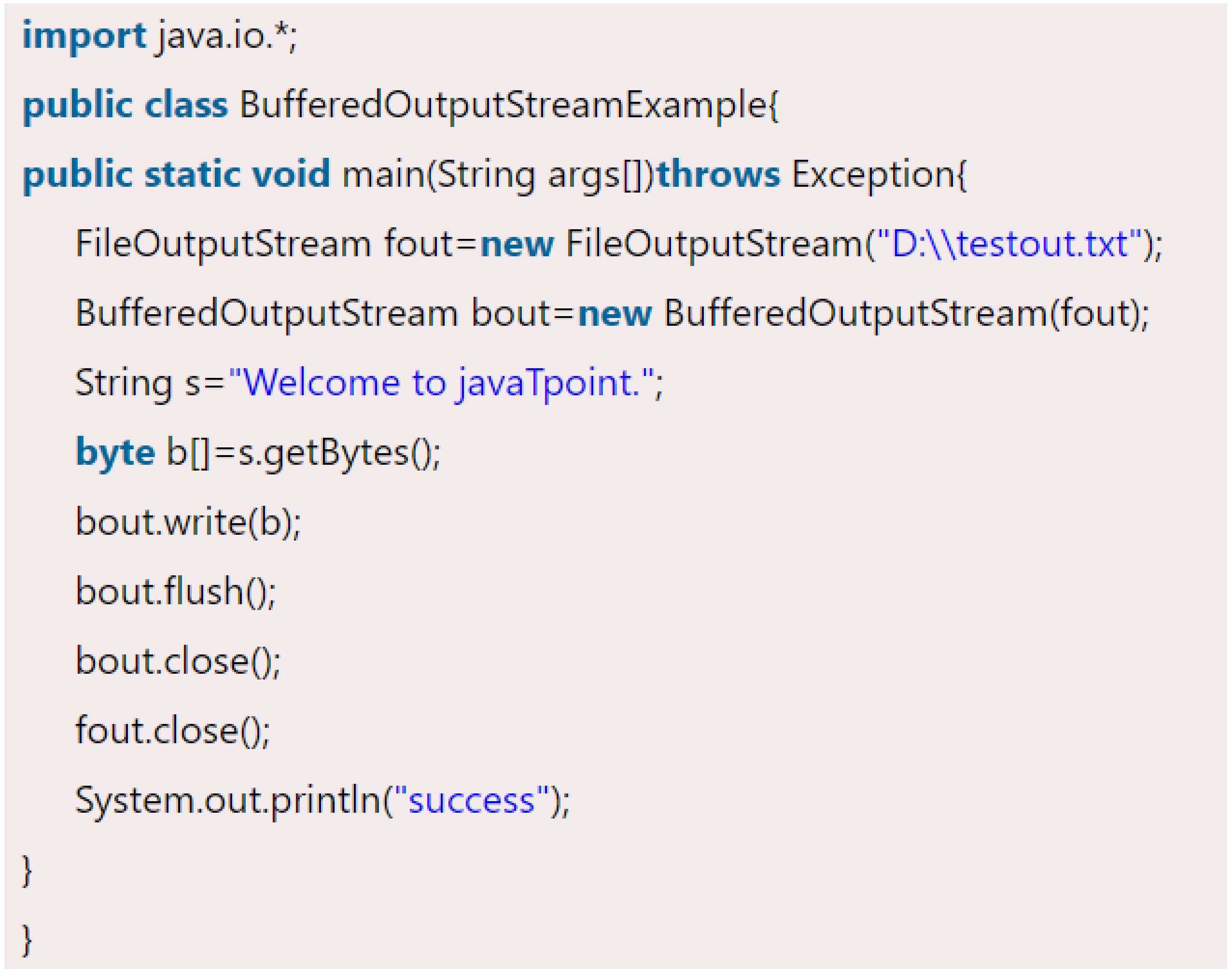
Java BufferedInputStream class methods

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

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

Java BufferedOutputStream Class

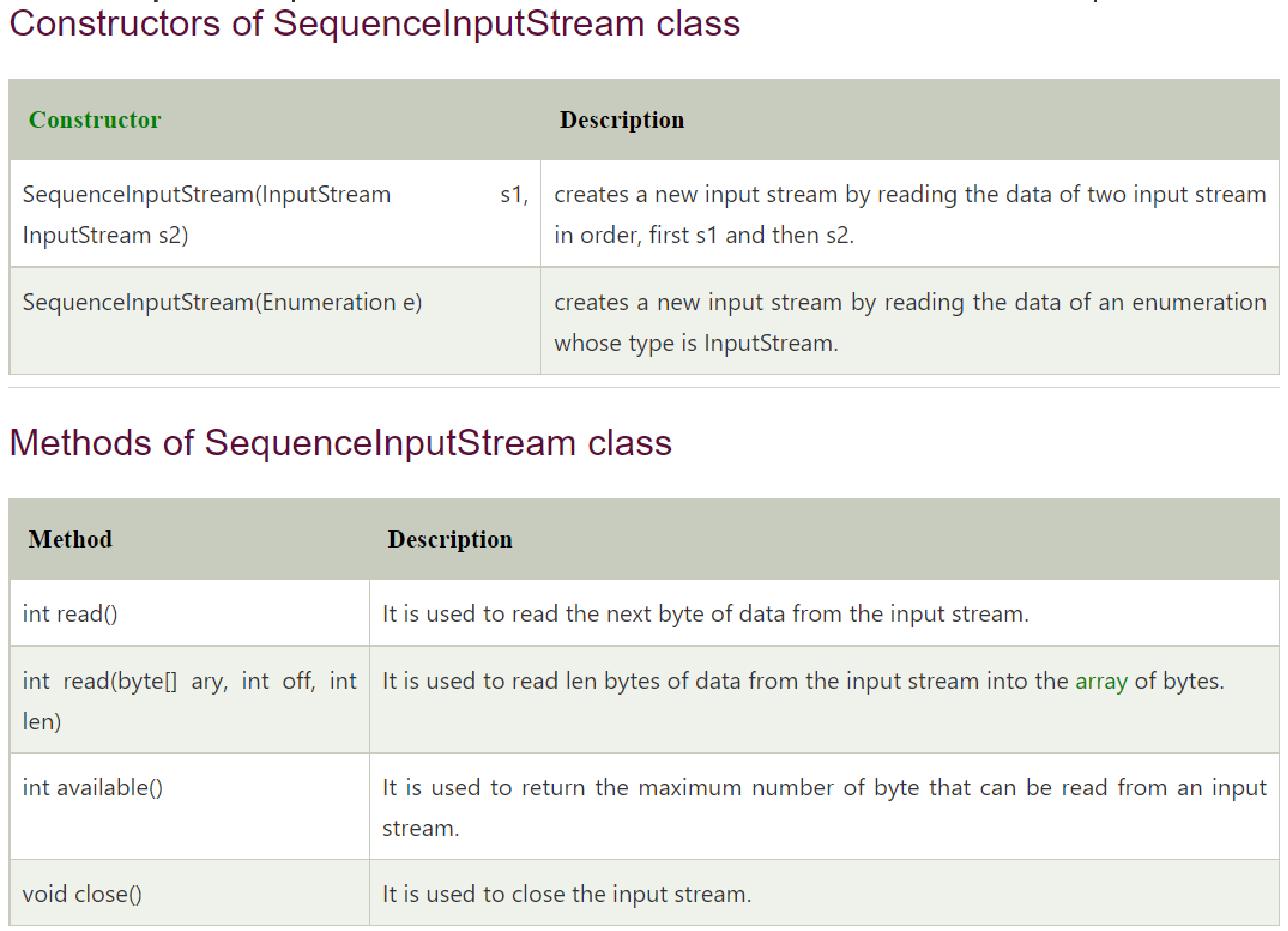
Java BufferedOutputStream class is used for buffering an output stream. It internally uses buffer to store data. It adds more efficiency than to write data directly into a stream. So, it makes the performance fast. The flush() flushes the data of one stream and send it into another. It is required if you have connected the one stream with another.



```
import java.io.*;
public class BufferedOutputStreamExample{
public static void main(String args[])throws Exception{
    FileOutputStream fout=new FileOutputStream("D:\\testout.txt");
    BufferedOutputStream bout=new BufferedOutputStream(fout);
    String s="Welcome to javaTpoint.";
    byte b[]=s.getBytes();
    bout.write(b);
    bout.flush();
    bout.close();
    fout.close();
    System.out.println("success");
}
}
```

Java SequenceInputStream Class

Java SequenceInputStream class is used to read data from multiple streams. It reads data sequentially (one by



Constructors of SequenceInputStream class

Constructor	Description
SequenceInputStream(InputStream s1, InputStream s2)	creates a new input stream by reading the data of two input stream in order, first s1 and then s2.
SequenceInputStream(Enumeration e)	creates a new input stream by reading the data of an enumeration whose type is InputStream.

Methods of SequenceInputStream class

Method	Description
int read()	It is used to read the next byte of data from the input stream.
int read(byte[] ary, int off, int len)	It is used to read len bytes of data from the input stream into the array of bytes.
int available()	It is used to return the maximum number of byte that can be read from an input stream.
void close()	It is used to close the input stream.

```java
import java.io.*;
class InputStreamExample {
 public static void main(String args[])throws Exception{
  FileInputStream input1=new FileInputStream("D:\\testin.txt");
  FileInputStream input2=new FileInputStream("D:\\testout.txt");
  SequenceInputStream inst=new SequenceInputStream(input1, input2);
  int j;
  while((j=inst.read())!=-1){
   System.out.print((char)j);
  }
  inst.close();
  input1.close();
  input2.close();
 }
}
```

testin.txt:

```
Welcome to Java IO Programming.
```

testout.txt:

```
It is the example of Java SequenceInputStream class.
```

After executing the program, you will get following output:

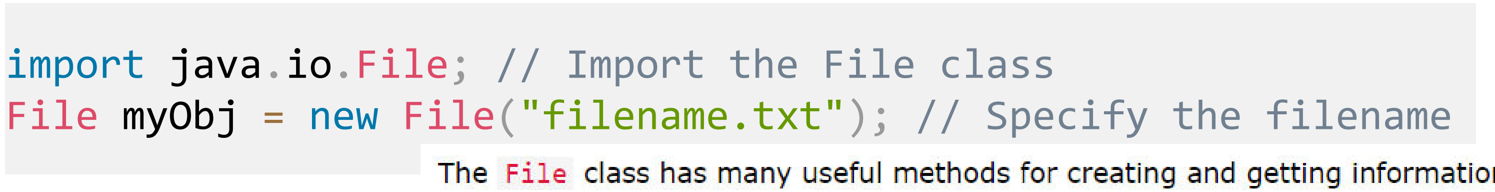
Output:

```
Welcome to Java IO Programming. It is the example of Java SequenceInputStream class.
```

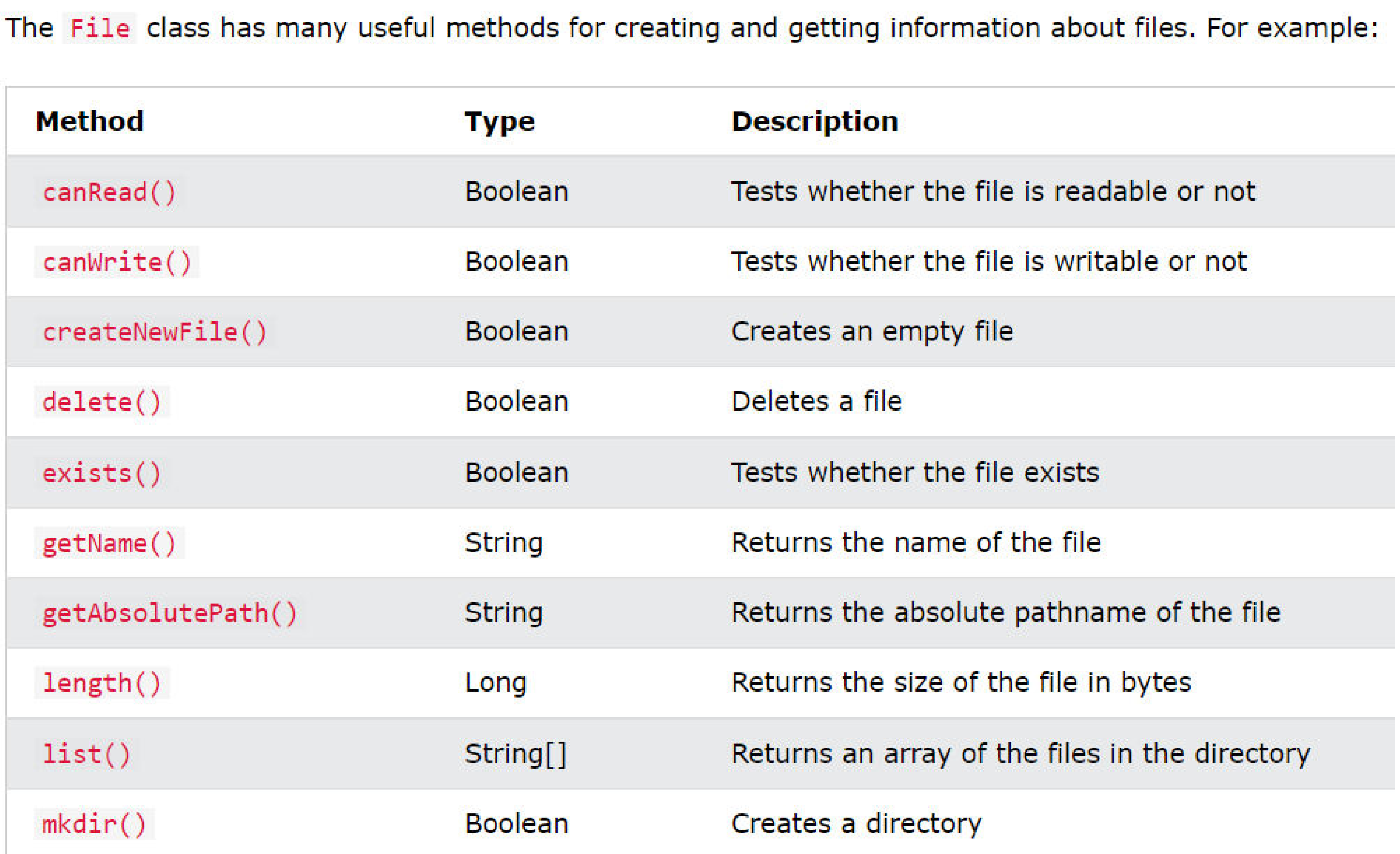
File class:

File handling is an important part of any application.
Java has several methods for creating, reading, updating, and deleting files.

The `File` class from the `java.io` package, allows us to work with files.
To use the `File` class, create an object of the class, and specify the filename or directory name



```
import java.io.File; // Import the File class
File myObj = new File("filename.txt"); // Specify the filename
```



The `File` class has many useful methods for creating and getting information about files. For example:

Method	Type	Description
`canRead()`	Boolean	Tests whether the file is readable or not
`canWrite()`	Boolean	Tests whether the file is writable or not
`createNewFile()`	Boolean	Creates an empty file
`delete()`	Boolean	Deletes a file
`exists()`	Boolean	Tests whether the file exists
`getName()`	String	Returns the name of the file
`getAbsolutePath()`	String	Returns the absolute pathname of the file
`length()`	Long	Returns the size of the file in bytes
`list()`	String[]	Returns an array of the files in the directory
`mkdir()`	Boolean	Creates a directory

Create a File

To create a file in Java, you can use the `createNewFile()` method. This method returns a boolean value: `true` if the file was successfully created, and `false` if the file already exists. Note that the method is enclosed in a `try...catch` block. This is necessary because it throws an `IOException` if an error occurs (if the file cannot be created for some reason):

```
import java.io.File;  // Import the File class
import java.io.IOException;  // Import the IOException class to handle errors

public class CreateFile {
  public static void main(String[] args) {
    try {
      File myObj = new File("filename.txt");
      if (myObj.createNewFile()) {
        System.out.println("File created: " + myObj.getName());
      } else {
        System.out.println("File already exists.");
      }
    } catch (IOException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

```
File created: filename.txt
```

To create a file in a specific directory (requires permission), specify the path of the file and use double backslashes to escape the "\" character (for Windows). On Mac and Linux you can just write the path, like: /Users/name/filename.txt

```java
File myObj = new File("C:\\Users\\MyName\\filename.txt");
```

```java
import java.io.File;  // Import the File class

public class GetFileInfo {
  public static void main(String[] args) {
    File myObj = new File("filename.txt");
    if (myObj.exists()) {
      System.out.println("File name: " + myObj.getName());
      System.out.println("Absolute path: " + myObj.getAbsolutePath());
      System.out.println("Writeable: " + myObj.canWrite());
      System.out.println("Readable " + myObj.canRead());
      System.out.println("File size in bytes " + myObj.length());
    } else {
      System.out.println("The file does not exist.");
    }
  }
}
```

```
File name: filename.txt
Absolute path: C:\Users\MyName\filename.txt
Writeable: true
Readable: true
File size in bytes: 0
```

There are many available classes in the Java API that can be used to read and write files in Java: `FileReader`, `BufferedReader`, `Files`, `Scanner`, `FileInputStream`, `FileWriter`, `BufferedWriter`, `FileOutputStream`, etc.
Which one to use depends on the Java version you're working with and whether you need to read bytes or characters, and the size of the file/lines etc.
Following is the example for reading data from the file using Scanner class:

```java
import java.io.File;  // Import the File class
import java.io.FileNotFoundException;  // Import this class to handle errors
import java.util.Scanner; // Import the Scanner class to read text files

public class ReadFile {
  public static void main(String[] args) {
    try {
      File myObj = new File("filename.txt");
      Scanner myReader = new Scanner(myObj);
      while (myReader.hasNextLine()) {
        String data = myReader.nextLine();
        System.out.println(data);
      }
      myReader.close();
    } catch (FileNotFoundException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

Delete a File

To delete a file in Java, use the `delete()` method:

```java
import java.io.File;  // Import the File class

public class DeleteFile {
  public static void main(String[] args) {
    File myObj = new File("filename.txt");
    if (myObj.delete()) {
      System.out.println("Deleted the file: " + myObj.getName());
    } else {
      System.out.println("Failed to delete the file.");
    }
  }
}
```

```
Deleted the file: filename.txt
```

Delete a Folder

```java
import java.io.File;

public class DeleteFolder {
  public static void main(String[] args) {
    File myObj = new File("C:\\Users\\MyName\\Test");
    if (myObj.delete()) {
      System.out.println("Deleted the folder: " + myObj.getName());
    } else {
      System.out.println("Failed to delete the folder.");
    }
  }
}
```

```
Deleted the folder: Test
```

Java - RandomAccessFile

This class is used for reading and writing to random access file. In java, the **java.io** package has a built-in class **RandomAccessFile** that enables a file to be accessed randomly. The RandomAccessFile class has several methods used to move the cursor position in a file.If end-of-file is reached before the desired number of byte has been read than EOFException is thrown. It is a type of IOException.

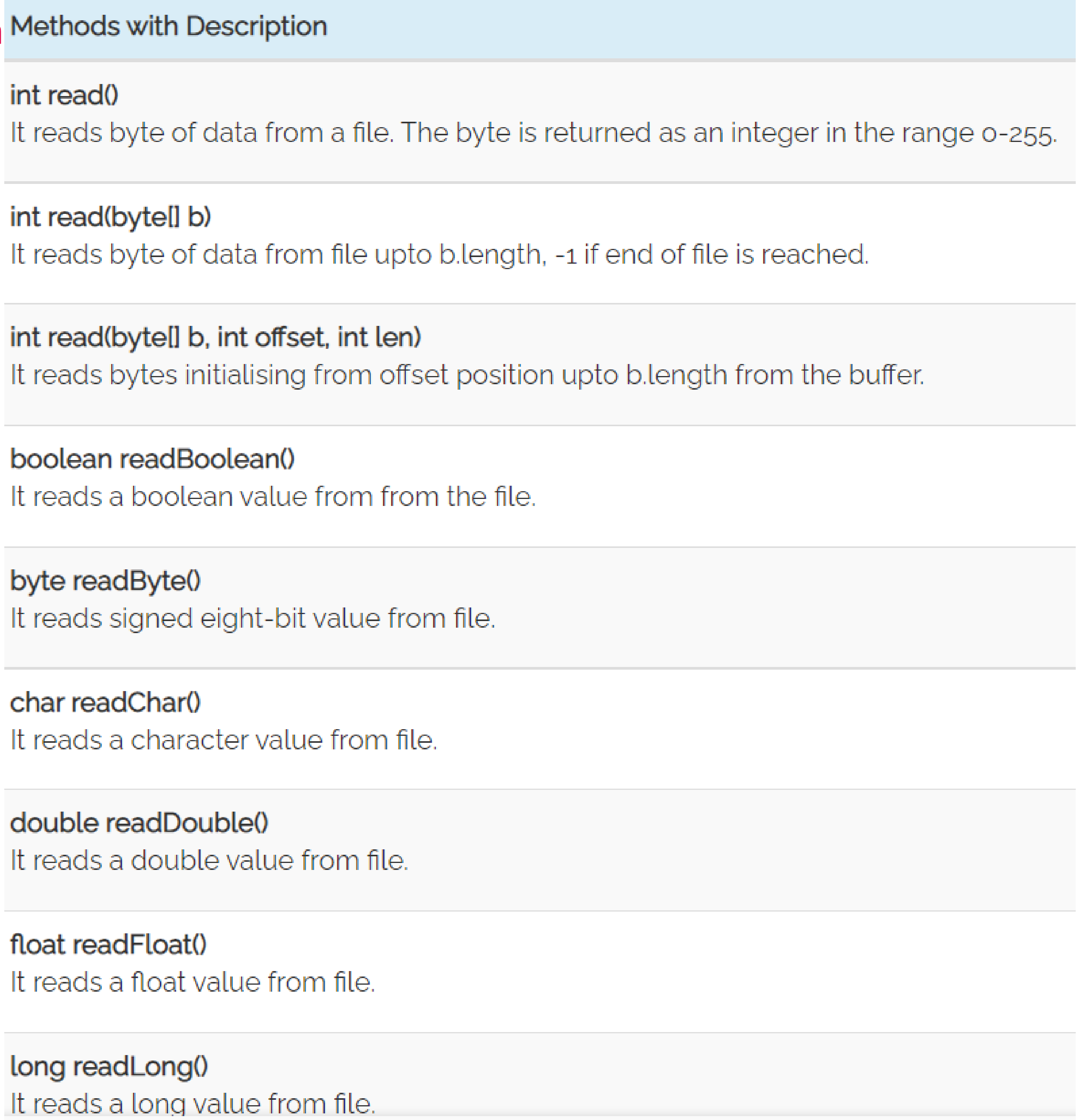
Constructor	**Description**
RandomAccessFile(File file, String mode)	Creates a random access file stream to read from, and optionally to write to, the file specified by the File argument.
RandomAccessFile(String name, String mode)	Creates a random access file stream to read from, and optionally to write to, a file with the specified name.

Access Modes

Using the RandomAccessFile, a file may created in th following modes.

- r - Creates the file with read mode; Calling write methods will result in an IOException.
- rw - Creates the file with read and write mode.
- rwd - Creates the file with read and write mode - synchronously. All updates to file content is written to the disk synchronously.
- rws - Creates the file with read and write mode - synchronously. All updates to file content or meta data is written to the disk synchronously.

RandomAccessFile meth



Methods with Description
int read() It reads byte of data from a file. The byte is returned as an integer in the range 0-255.
int read(byte[] b) It reads byte of data from file upto b.length, -1 if end of file is reached.
int read(byte[] b, int offset, int len) It reads bytes initialising from offset position upto b.length from the buffer.
boolean readBoolean() It reads a boolean value from from the file.
byte readByte() It reads signed eight-bit value from file.
char readChar() It reads a character value from file.
double readDouble() It reads a double value from file.
float readFloat() It reads a float value from file.
long readLong() It reads a long value from file.

10	**int readInt()** It reads a integer value from file.
11	**void readFully(byte[] b)** It reads bytes initialising from offset position upto b.length from the buffer.
12	**void readFully(byte[] b, int offset, int len)** It reads bytes initialising from offset position upto b.length from the buffer.
13	**String readUTF()** t reads in a string from the file.
14	**void seek(long pos)** It sets the file-pointer(cursor) measured from the beginning of the file, at which the next read or write occurs.
15	**long length()** It returns the length of the file.
16	**void write(int b)** It writes the specified byte to the file from the current cursor position.
17	**void writeFloat(float v)** It converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the file as a four-byte quantity, high byte first.
18	**void writeDouble(double v)** It converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the file as an eight-byte quantity, high byte first.

```java
import java.io.RandomAccessFile;
import java.io.IOException;

public class RandomAccessFileExample {
    static final String FILEPATH ="myFile.TXT";
    public static void main(String[] args) {
        try {
            System.out.println(new String(readFromFile(FILEPATH, 0, 18)));
            writeToFile(FILEPATH, "I love my country and my people", 31);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    private static byte[] readFromFile(String filePath, int position, int size)
            throws IOException {
        RandomAccessFile file = new RandomAccessFile(filePath, "r");
        file.seek(position);
        byte[] bytes = new byte[size];
        file.read(bytes);
        file.close();
        return bytes;
    }
    private static void writeToFile(String filePath, String data, int position)
            throws IOException {
        RandomAccessFile file = new RandomAccessFile(filePath, "rw");
        file.seek(position);
        file.write(data.getBytes());
        file.close();
```

The myFile.TXT contains text "This class is used for reading and writing to random access file."

after running the program it will contains

This class is used for reading I love my country and my peoplele.