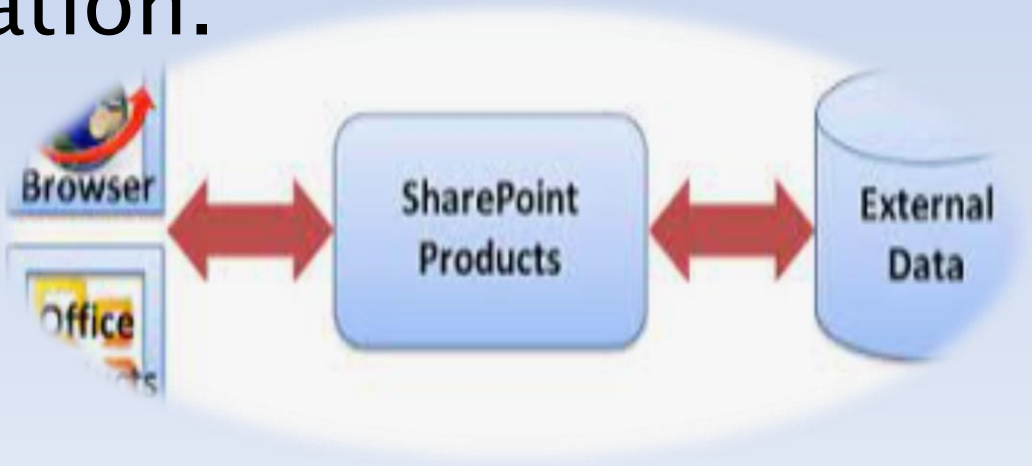
# Network Programming

## Input-output in Java

## Introduction

- Often a program needs to bring in information from an external source or to send out information to an external destination.



## Introduction

- The **information** <u>can be anywhere</u>: in a file, on disk, somewhere on the network, in memory, or in another program, or stored in a database.
- Also, the **information** <u>can be of any type</u>: objects, characters, images, or sounds.

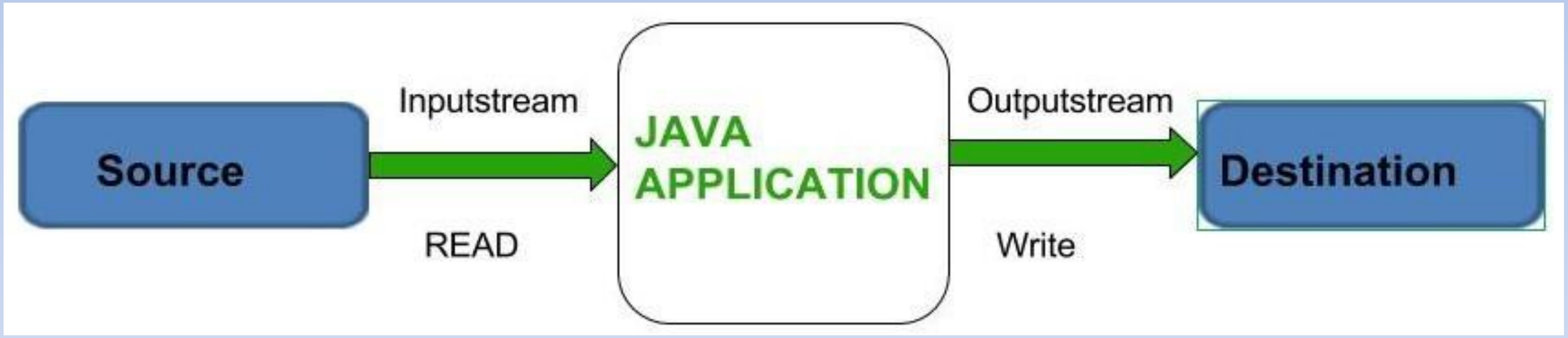
## Introduction

- This chapter covers the Java platform classes that your programs can use to read and to write data.

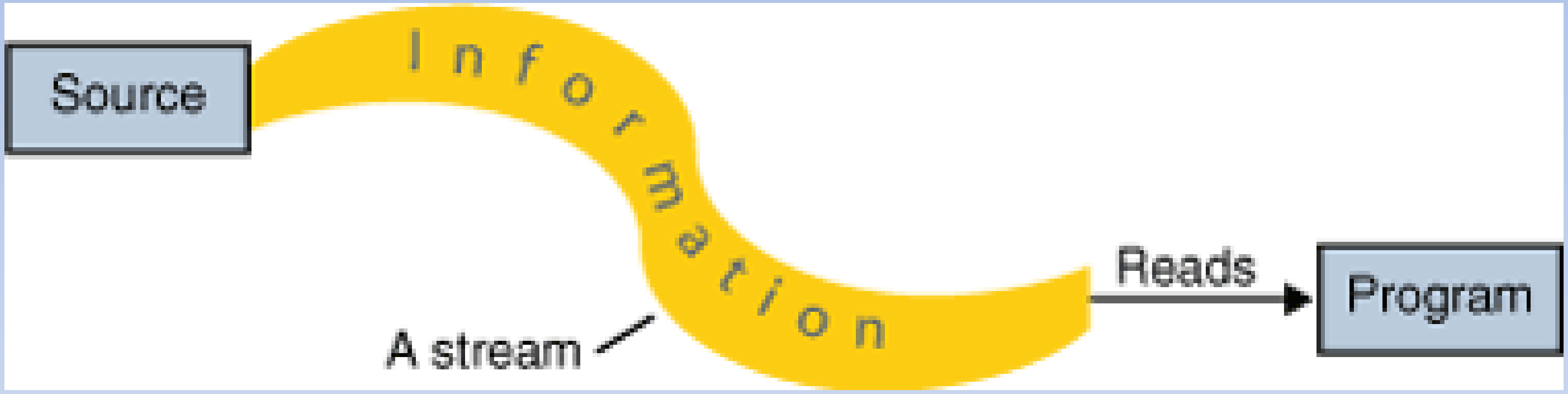
## Stream

- A **stream** can be defined as **a sequence of data**.
- **Depending on the type of operations**, streams can be divided into two primary classes:
  - **InPutStream** – The InputStream is used to read data from a source.
  - **OutPutStream** – The OutputStream is used for writing data to a destination.

## Stream

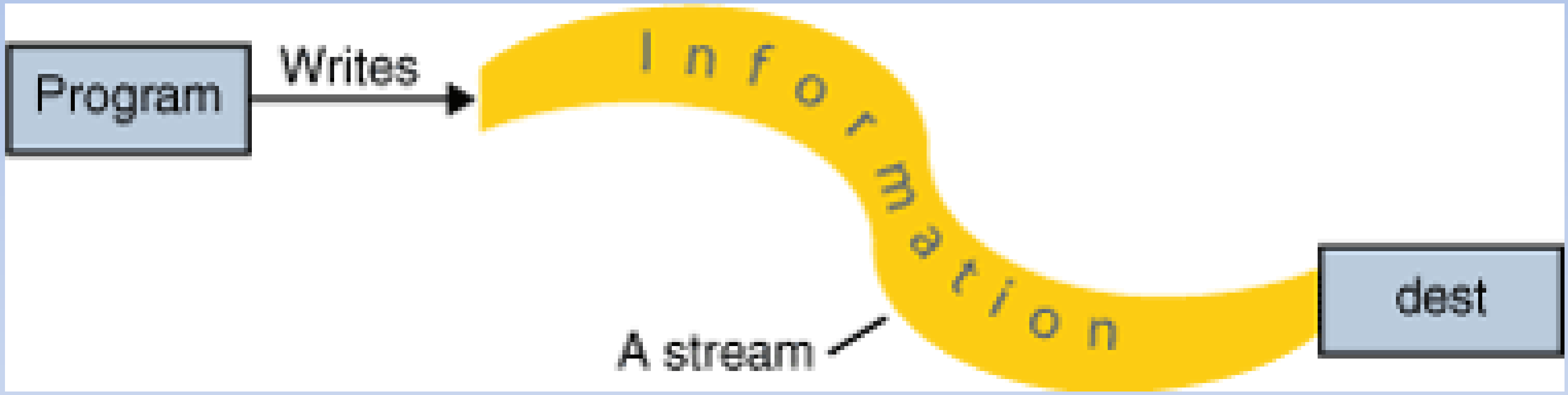

Source
Inputstream
READ
JAVA APPLICATION
Outputstream
Write
Destination

# Reading information into a program



- To bring in information, a program opens a *stream* on an information source (a file, memory, a socket) and reads the information sequentially.

# Writing information out of a program



- Similarly, a program can send information to an external destination by opening a stream to a destination and writing the information out sequentially, as shown in the following figure.

## Reading and Writing Algorithm for Data

- No matter where the data is coming from or going to and no matter what its type, the algorithms for sequentially reading and writing data are basically the same.

## Algorithms

**Reading**

```
open a stream
while more information
read information
close the stream
```

**Writing**

```
open a stream
while more information
write information
close the stream
```

# Java IO : Input-output in Java

- The java.io package contains nearly every class you might ever need to perform input and output (I/O) in Java.
- All these streams represent an input source and an output destination.

## Java IO : Input-output in Java

- The stream in the java.io package supports many data such as primitives, object, localized characters, etc.

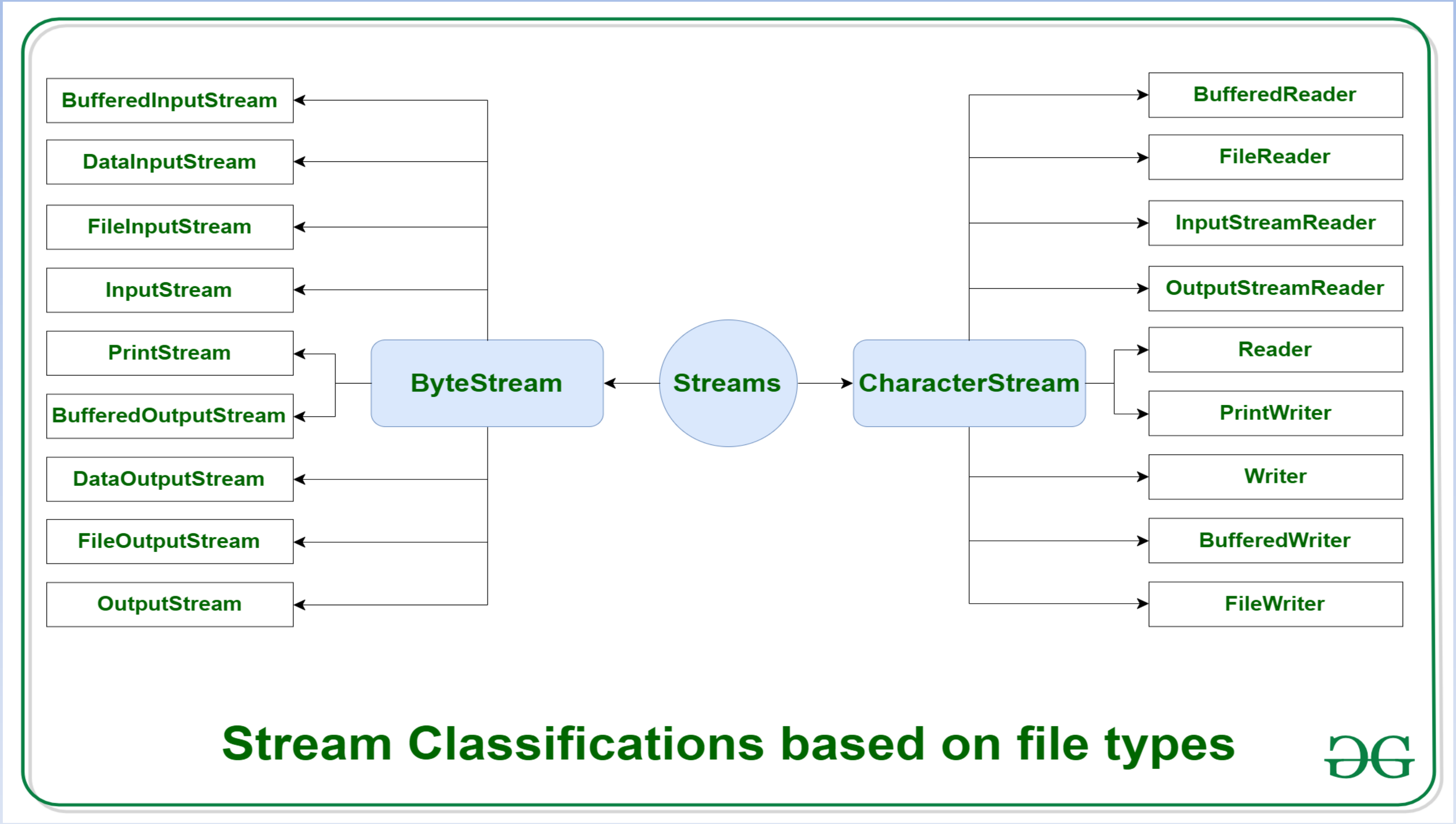
## I/O Stream in Java

- Java provides strong but flexible support for I/O related to files and networks.
  - but this tutorial covers very basic functionality related to streams and I/O.

## I/O Stream types

- Depending on the types of file, Streams can be divided into two primary classes:
  - Byte Streams.
  - Character Streams.

# I/O Stream types



## Byte Streams

- **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.
  - The FileOutputStream is used to write to the destination.

# Byte Streams

- 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 |

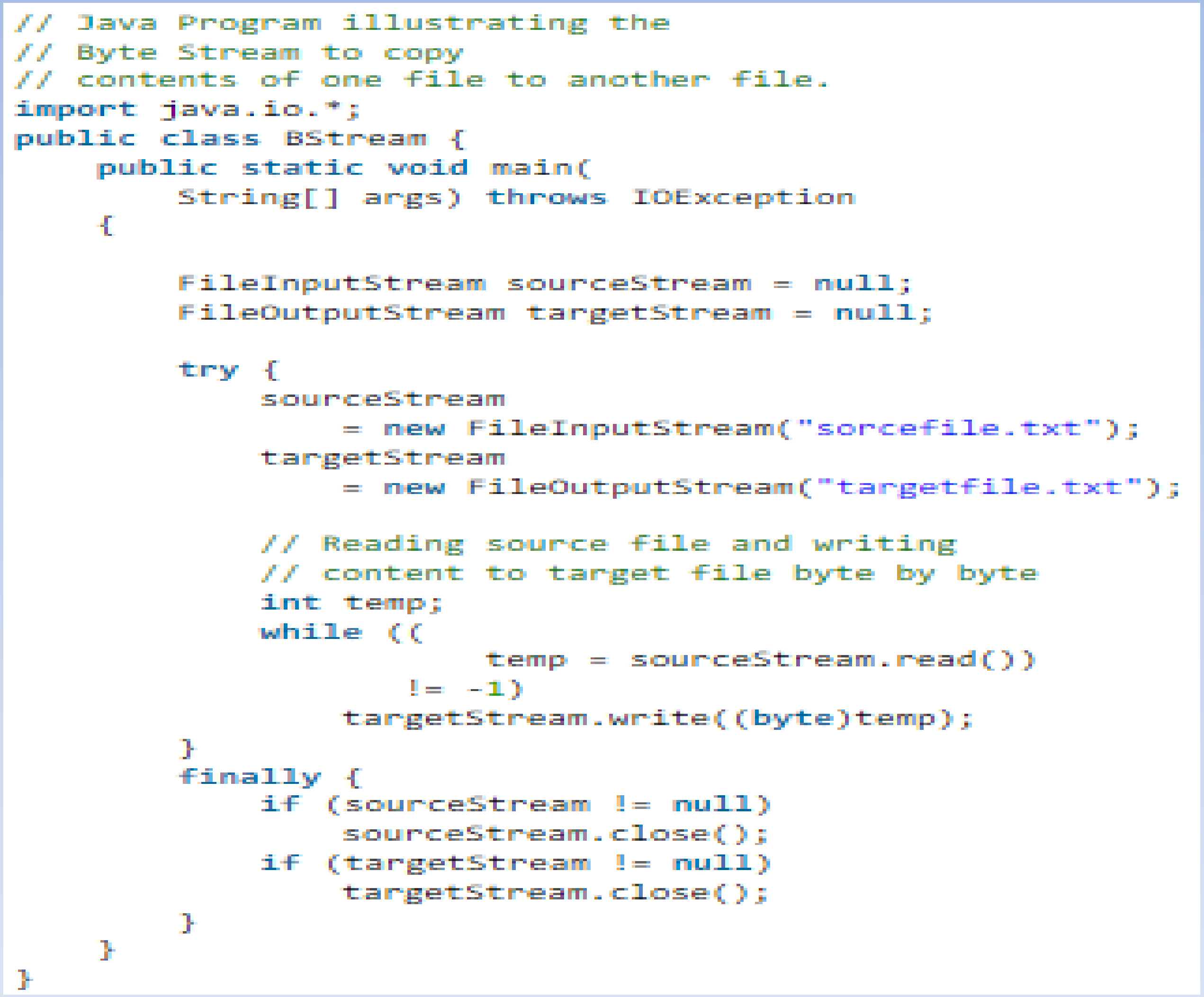
## Byte Streams

- Here is the list of various ByteStream Classes:

| Stream class | Description |
|---|---|
| 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. |

# Byte Streams Example

➢



```java
// Java Program illustrating the
// Byte Stream to copy
// contents of one file to another file.
import java.io.*;
public class BStream {
    public static void main(
        String[] args) throws IOException
    {

        FileInputStream sourceStream = null;
        FileOutputStream targetStream = null;

        try {
            sourceStream
                = new FileInputStream("sorcefile.txt");
            targetStream
                = new FileOutputStream("targetfile.txt");

            // Reading source file and writing
            // content to target file byte by byte
            int temp;
            while ((
                      temp = sourceStream.read())
                   != -1)
                targetStream.write((byte)temp);
        }
        finally {
            if (sourceStream != null)
                sourceStream.close();
            if (targetStream != null)
                targetStream.close();
        }
    }
}
```

## Character Streams

- In Java, characters are stored using Unicode conventions.
- Character stream automatically allows us to read/write data character by character.
  - i.e. Java **Character** streams are used to perform input and output for 16-bit Unicode.

## Character Streams

- 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 Streams

- 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. |

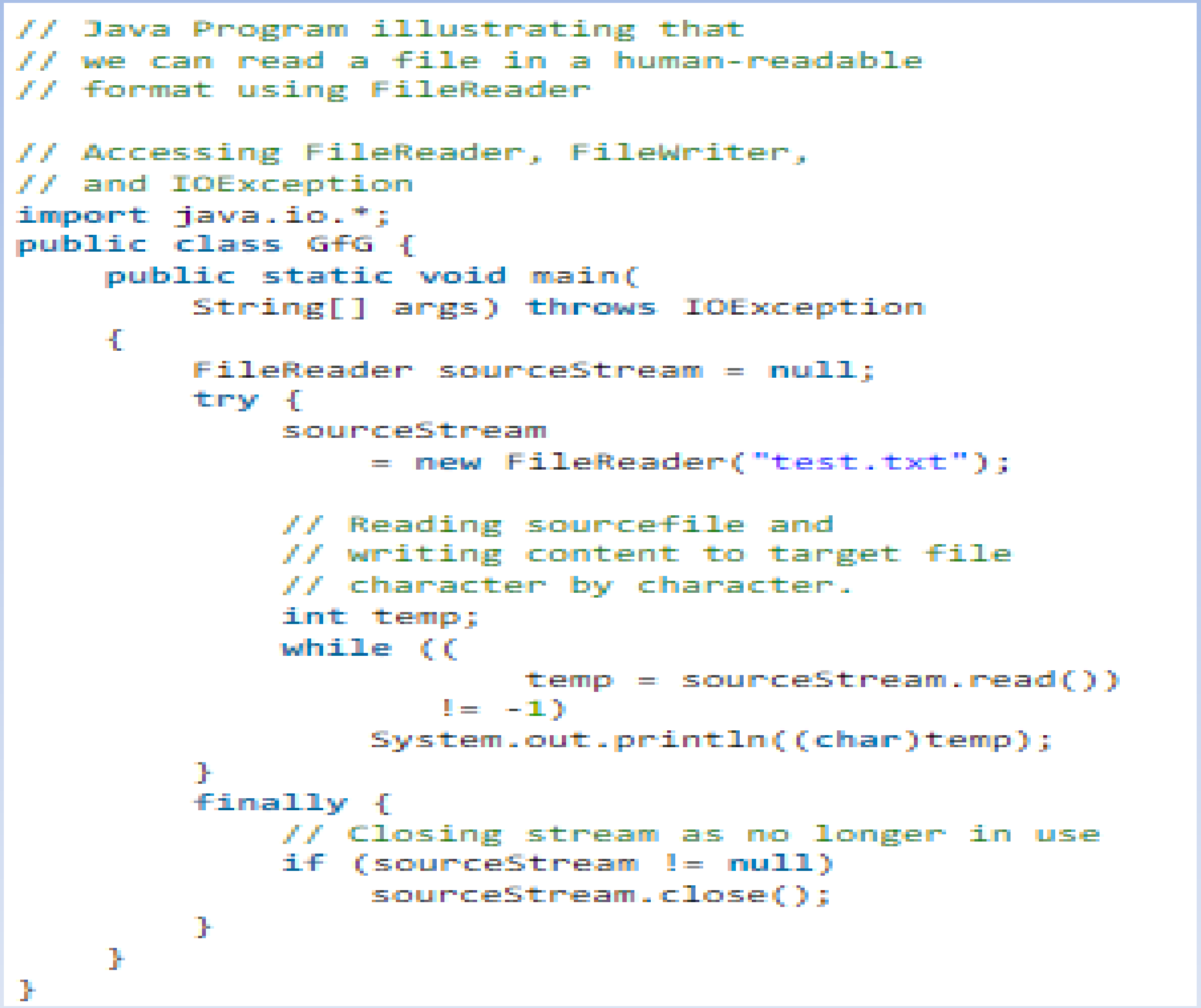
## Character Streams

- Here is the list of various CharacterStream Classes:

| Stream class | Description |
|---|---|
| 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. |

## Character Streams Example

➢



```
// Java Program illustrating that
// we can read a file in a human-readable
// format using FileReader

// Accessing FileReader, FileWriter,
// and IOException
import java.io.*;
public class GfG {
    public static void main(
        String[] args) throws IOException
    {
        FileReader sourceStream = null;
        try {
            sourceStream
                = new FileReader("test.txt");

            // Reading sourcefile and
            // writing content to target file
            // character by character.
            int temp;
            while ((
                    temp = sourceStream.read())
                  != -1)
                System.out.println((char)temp);
        }
        finally {
            // Closing stream as no longer in use
            if (sourceStream != null)
                sourceStream.close();
        }
    }
}
```

## Standard Streams

- All the programming languages provide support for standard I/O where the user's program can take input from a keyboard and then produce an output on the computer screen.

## Standard Streams

- Java provides the following three standard streams:
  - Standard Input stream.
  - Standard Output stream.
  - Standard Error stream.

## Standard Streams

- ➢ Standard Input stream:
  - – This is used to feed the data to user's program and usually a keyboard is used as standard input stream and represented as **System.in**.

## Standard Streams

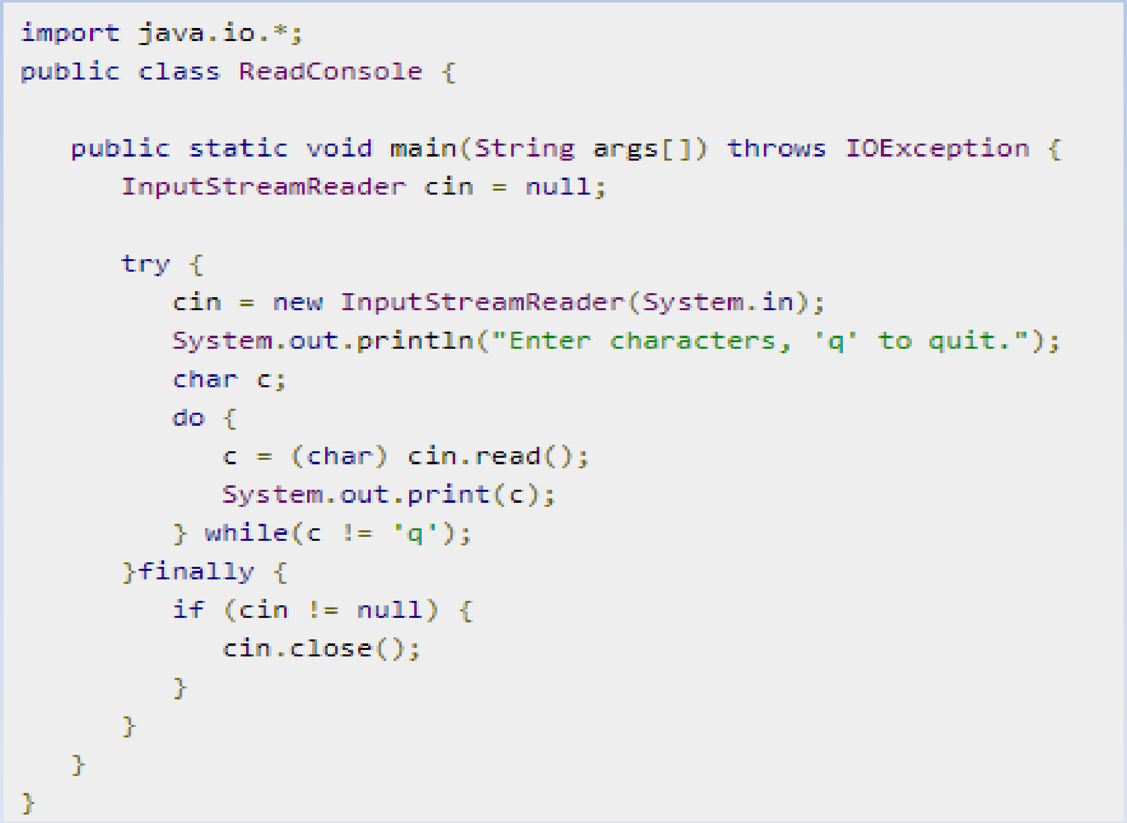
- Standard Output stream:
  - This is used to output the data produced by the user's program and usually a computer screen is used for standard output stream and represented as **System.out.**

## Standard Streams

- Standard Error stream:
  - This is used to output the error data produced by the user's program and usually a computer screen is used for standard error stream and represented as **System.err.**

# Standard Streams

- Following is a simple program, which creates **InputStreamReader** to read standard input stream until the user types a "q“.



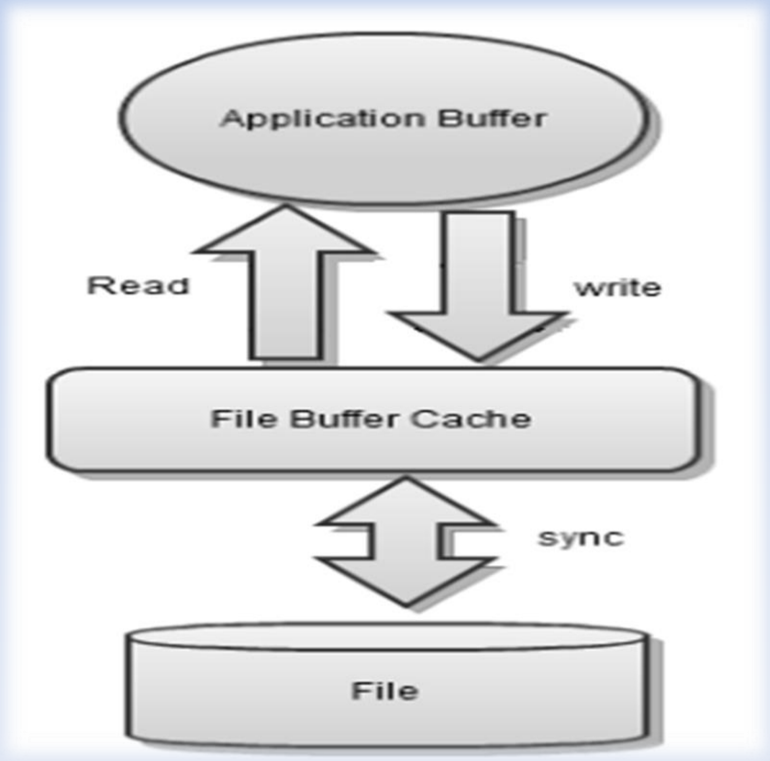
```
import java.io.*;
public class ReadConsole {

   public static void main(String args[]) throws IOException {
      InputStreamReader cin = null;

      try {
         cin = new InputStreamReader(System.in);
         System.out.println("Enter characters, 'q' to quit.");
         char c;
         do {
            c = (char) cin.read();
            System.out.print(c);
         } while(c != 'q');
      }finally {
         if (cin != null) {
            cin.close();
         }
      }
   }
}
```

## Buffered Streams

- In unbuffered I/O, each read or writes request is handled directly by the underlying OS.
  - This can make a program much less efficient.
    - since each such request often triggers disk access, network activity, or some other operation that is relatively expensive.

## Buffered Streams

- To reduce this kind of overhead, the Java platform implements buffered I/O streams.



## Buffered Streams

- Buffered input streams read data from a memory area known as a **buffer.**
  - The native input API is called only when the buffer is empty.

## Buffered Streams

- Similarly, buffered output streams write data to a buffer
  - and the native output API is called only when the buffer is full.

## Buffered Streams classes

- There are four buffered stream classes used to wrap unbuffered streams:
  - BufferedInputStream and BufferedOutputStream
    - create buffered byte streams
  - BufferedReader and BufferedWriter
    - create buffered character streams

# BufferedOutputStream Class

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

## BufferedInputStream Class

- Java **BufferedInputStream** class is used to read information from the stream.
- It internally uses the buffer mechanism to make the performance fast.

## BufferedInputStream Class

- 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.

## BufferedWriter Class

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

## BufferedReader Class

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

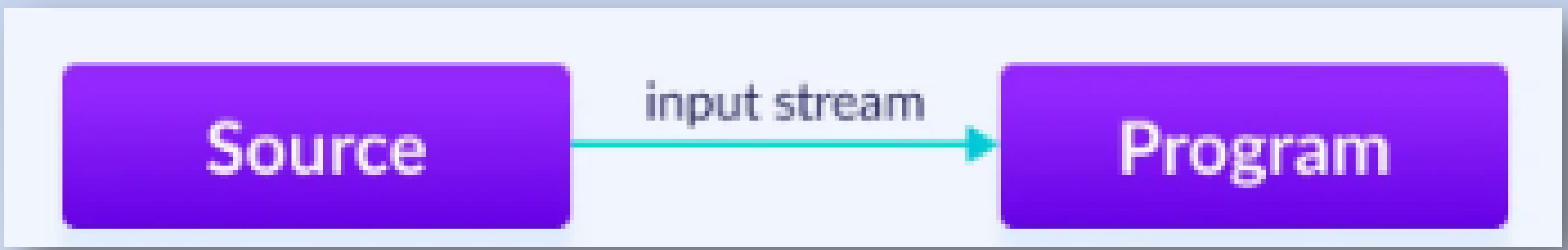
## File Streams

- **In real life**, we'll more commonly attach streams to data sources like files and network connections.

# File Streams

- The **java.io.FileInputStream** and **java.io.FileOutputStream** classes
  - Which are concrete subclasses of **java.io.InputStream** and **java.io.OutputStream**
  - Which provide methods for reading and writing data in files.

## FileInputStream in Java



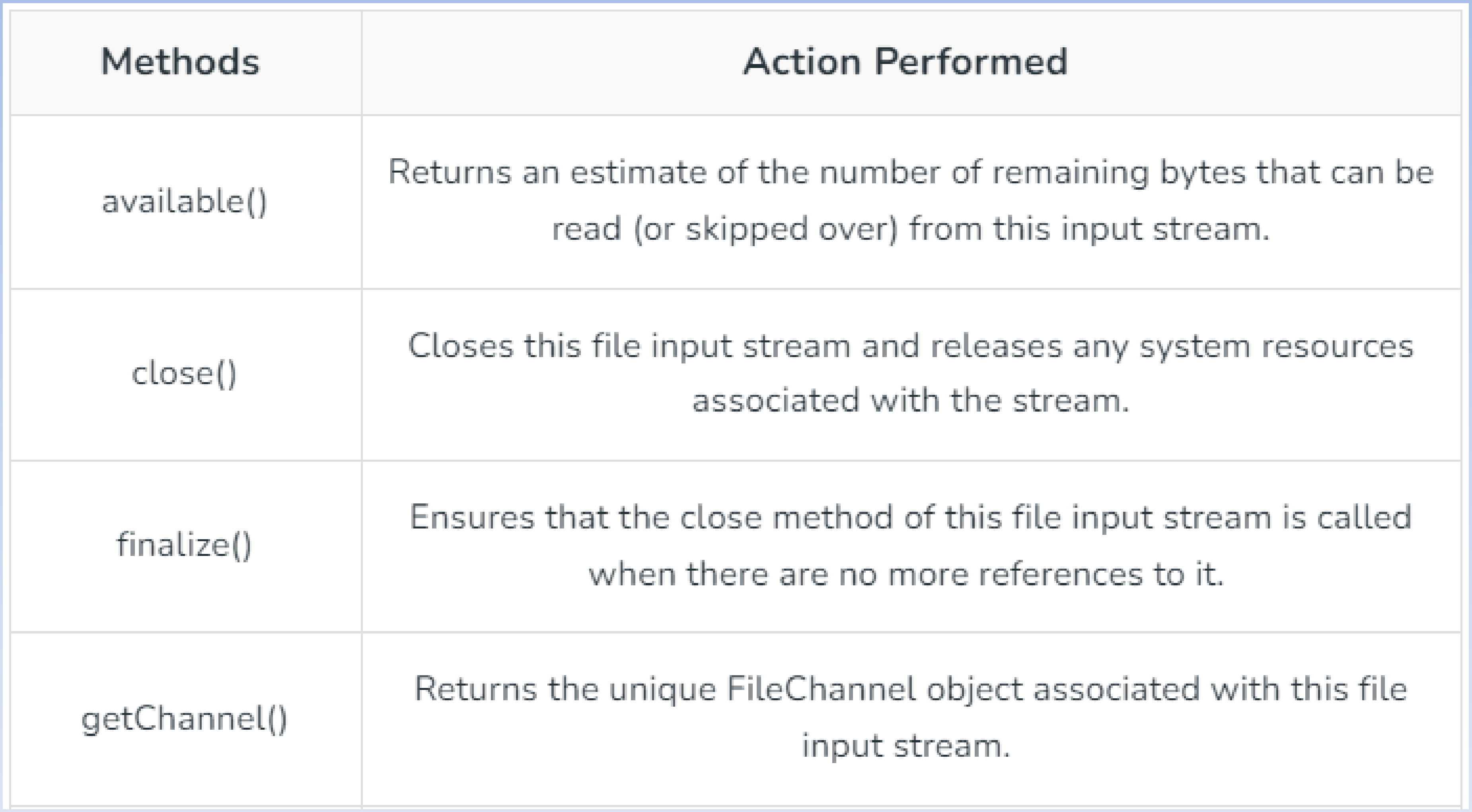
## FileInputStream class

- FileInputStream class is useful to read data from a file in the form of sequence of bytes.
- FileInputStream is meant for reading streams of raw bytes such as image data.

## Constructors of FileInputStream Class

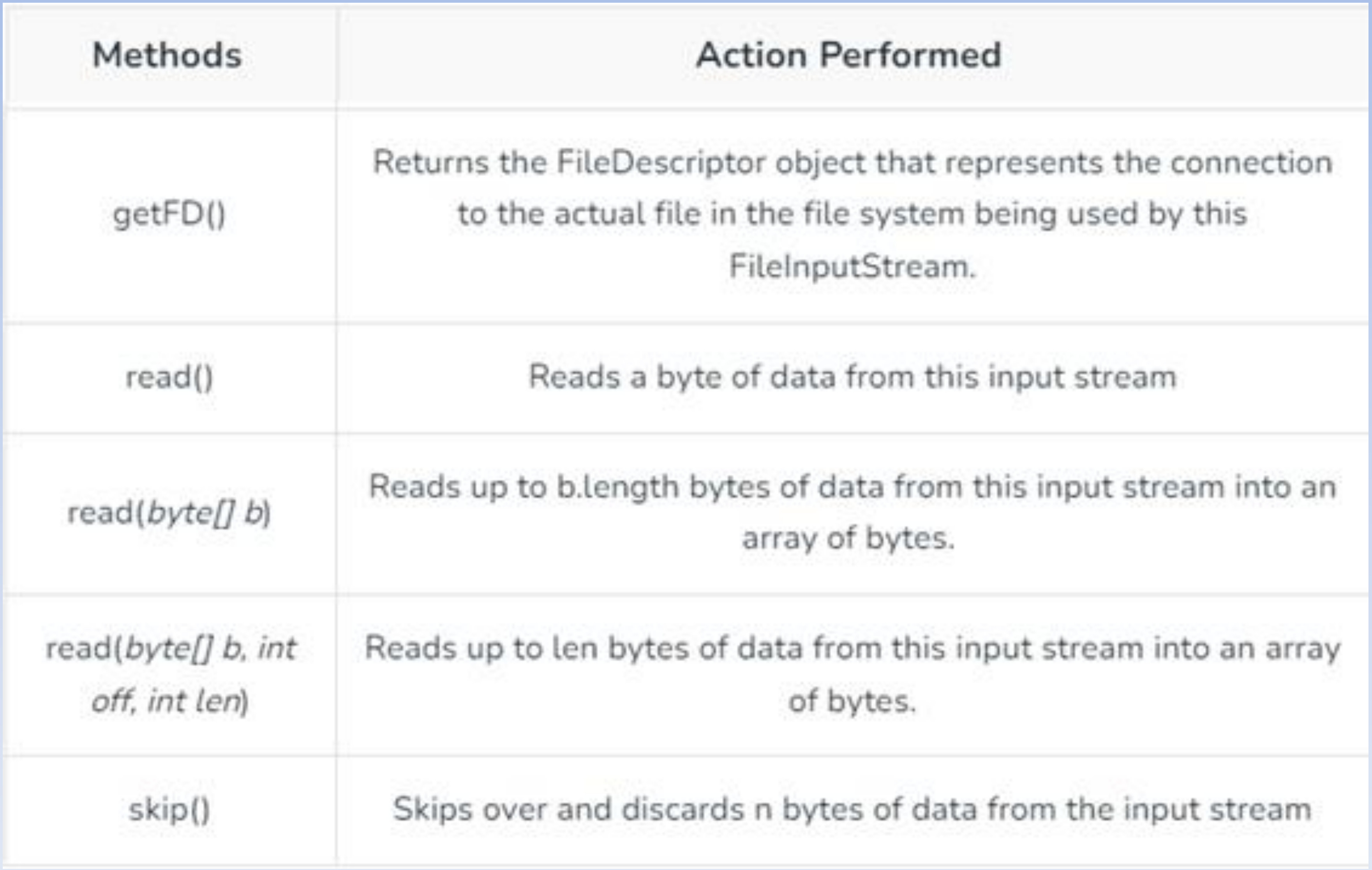
1. **FileInputStream(File file):**
   - ✓ Creates an input file stream to read from the specified File object.
2. **FileInputStream(FileDescriptor fdobj) :**
   - ✓Creates an input file stream to read from the specified file descriptor.
3. **FileInputStream(String name):**
   - ✓Creates an input file stream to read from a file with the specified name.

# Methods of FileInputStream Class



| Methods | Action Performed |
|---|---|
| available() | Returns an estimate of the number of remaining bytes that can be read (or skipped over) from this input stream. |
| close() | Closes this file input stream and releases any system resources associated with the stream. |
| finalize() | Ensures that the close method of this file input stream is called when there are no more references to it. |
| getChannel() | Returns the unique FileChannel object associated with this file input stream. |

# Methods of FileInputStream Class



| Methods | Action Performed |
|---|---|
| getFD() | Returns the FileDescriptor object that represents the connection to the actual file in the file system being used by this FileInputStream. |
| read() | Reads a byte of data from this input stream |
| read(*byte[] b*) | Reads up to b.length bytes of data from this input stream into an array of bytes. |
| read(*byte[] b, int off, int len*) | Reads up to len bytes of data from this input stream into an array of bytes. |
| skip() | Skips over and discards n bytes of data from the input stream |

## steps to read data from a file using FileInputStream

- ✓**Step 1:** Attach a file to a FileInputStream as this will enable us to read data from the file as shown below as follows:
  - ➢**FileInputStream fileInputStream =new FileInputStream("file.txt");**
- ✓**Step 2:** Now in order to read data from the file, we should read data from the FileInputStream as shown below:
  - ➢**ch=fileInputStream.read();**

## steps to read data from a file using FileInputStream

- ✓**Step 3-A:** When there is no more data available to read further, the read() method returns -1;
- ✓**Step 3-B:** Then we should attach the monitor to the output stream. For displaying the data, we can use System.out.print.
  - System.out.print(ch);

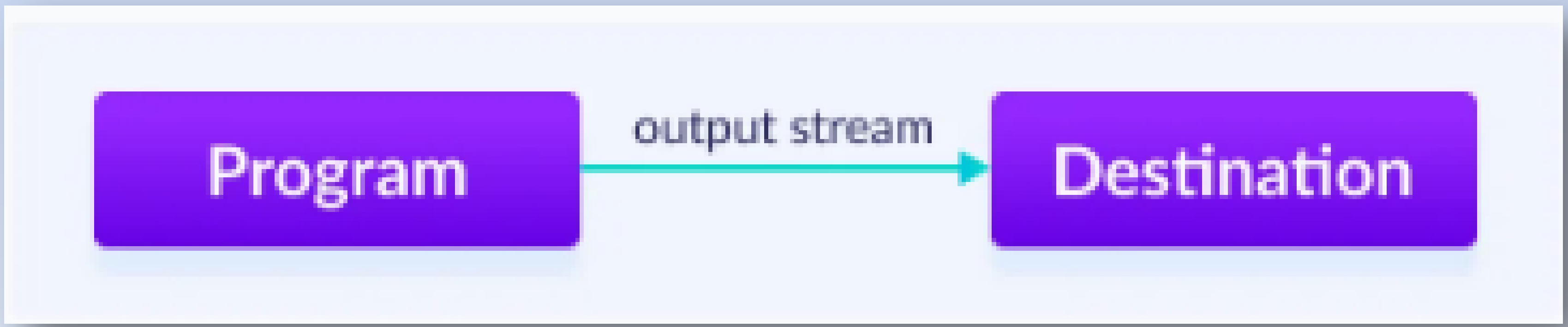
## Example

```
// Java Program to Demonstrate FileInputStream Class

// Importing I/O classes
import java.io.*;

// Main class
// ReadFile
class GFG {

    // Main driver method
    public static void main(String args[])
        throws IOException
    {

        // Attaching the file to FileInputStream
        FileInputStream fin
            = new FileInputStream("file1.txt");

        // Illustrating getChannel() method
        System.out.println(fin.getChannel());

        // Illustrating getFD() method
        System.out.println(fin.getFD());
```

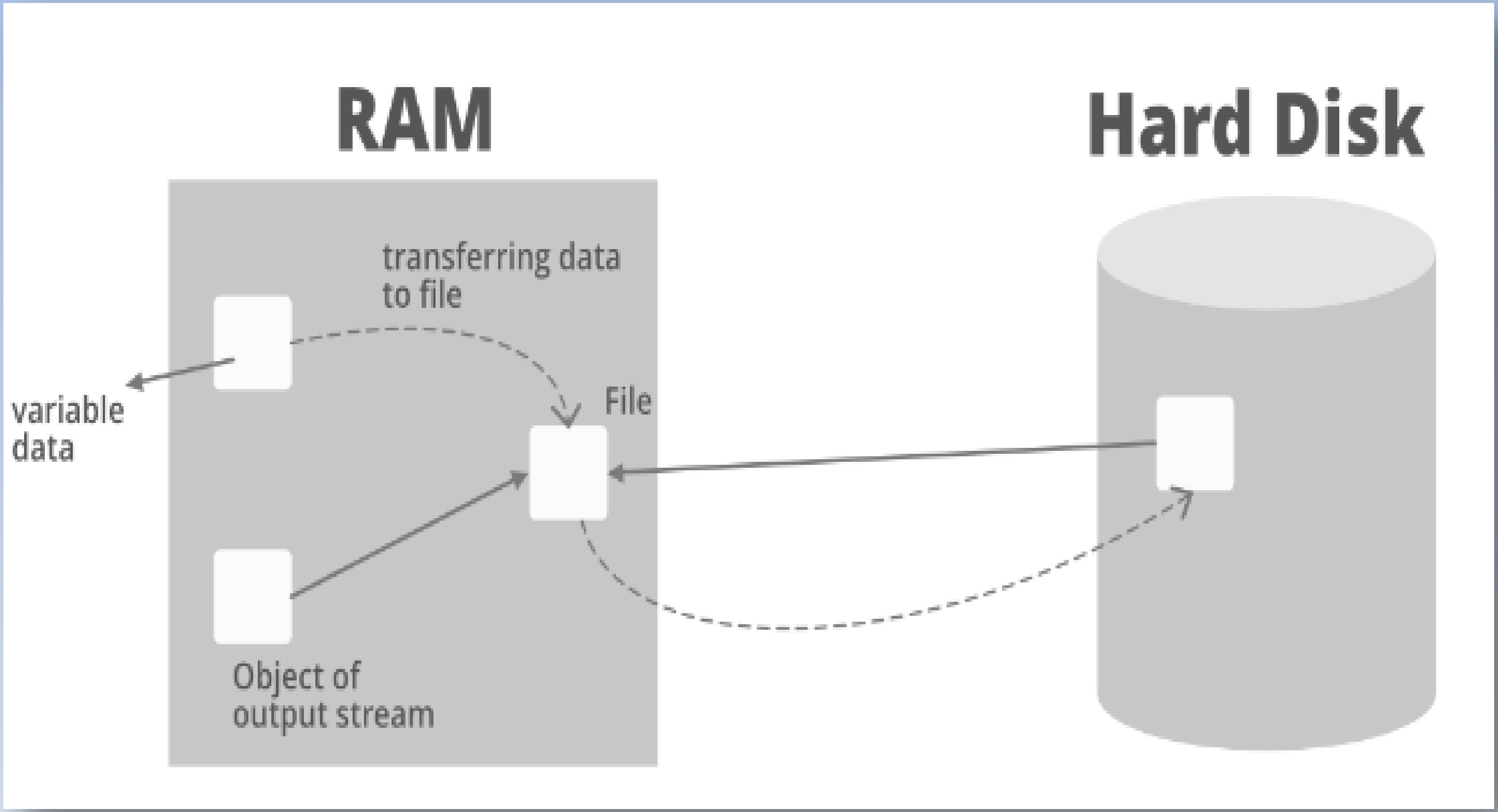
## Example

```
        // Illustrating skip() method
        fin.skip(4);

        // Display message for better readability
        System.out.println("FileContents :");

        // Reading characters from FileInputStream
        // and write them
        int ch;

        // Holds true till there is data inside file
        while ((ch = fin.read()) != -1)
            System.out.print((char)ch);

        // Close the file connections
        // using close() method
        fin.close();
    }
}
```

# FileOutputStream in Java



## What is meant by storing data to files?



## What is meant by storing data to files?

- When we run a Java program, data is stored in the RAM.
- To access the variable data stored in RAM and bring it to a file on the hard disk, an OutputStream object is created in RAM and this will point to a file referencing to hard disk.
- Then the data from the variable data file in the RAM will go to the referencing file (object of Output Stream) and from there will be transferred/stored in the file of the hard disk.

## Constructors of FileOutputStream

1. **FileOutputStream(File file):** Creates a file output stream to write to the file represented by the specified File object.

*FileOutputStream fout = new FileOutputStream(File file);*

## Constructors of FileOutputStream

2. **FileOutputStream( File file, boolean append):** Creates a file output stream object represented by specified file object.

*FileOutputStream fout = new FileOutputStream(File file, boolean append);*

# Constructors of FileOutputStream

3. **FileOutputStream(FileDescripter fdobj):** Creates a file output stream for writing to the specified file descriptor, which represents an existing connection with the actual file in the file system.

*FileOutputStream fout = new FileOutputStream(FileDescripter fdobj);*

## Constructors of FileOutputStream

4. **FileOutputStream( String name):** Creates an object of file output stream to write to the file with the particular name mentioned.

*FileOutputStream fout = new FileOutputStream( String name);*

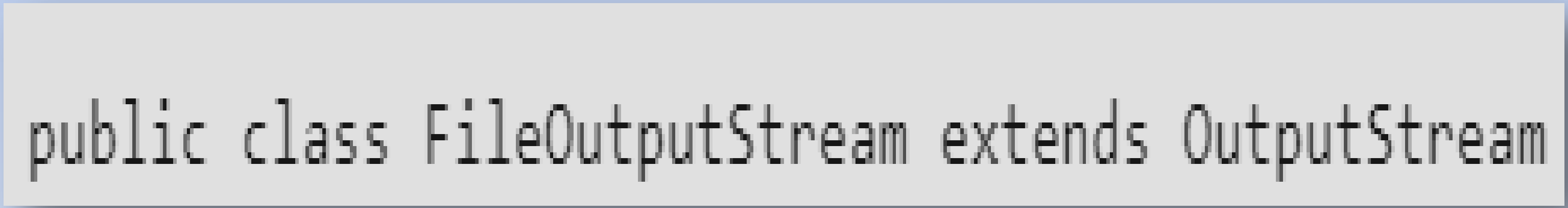
## Constructors of FileOutputStream

5. **FileOutputStream( String name, boolean append):** Creates an object of file output stream to write to the file with the specified name.

*FileOutputStream fout = new FileOutputStream( String name, boolean append);*

## FileOutputStream

- ✓ **Declaration:**

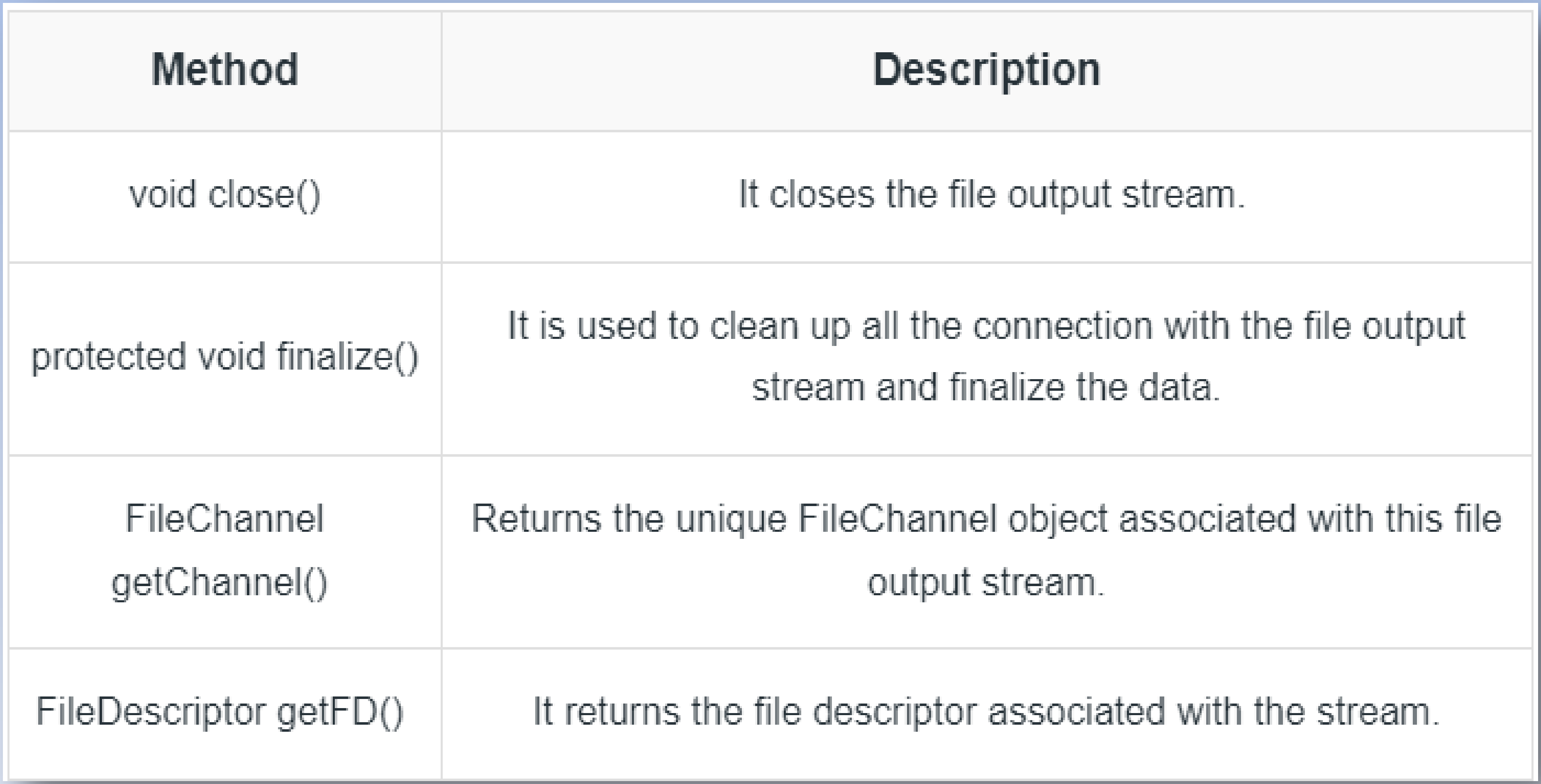


```
public class FileOutputStream extends OutputStream
```

## Steps to write data to a file using FileOutputStream:

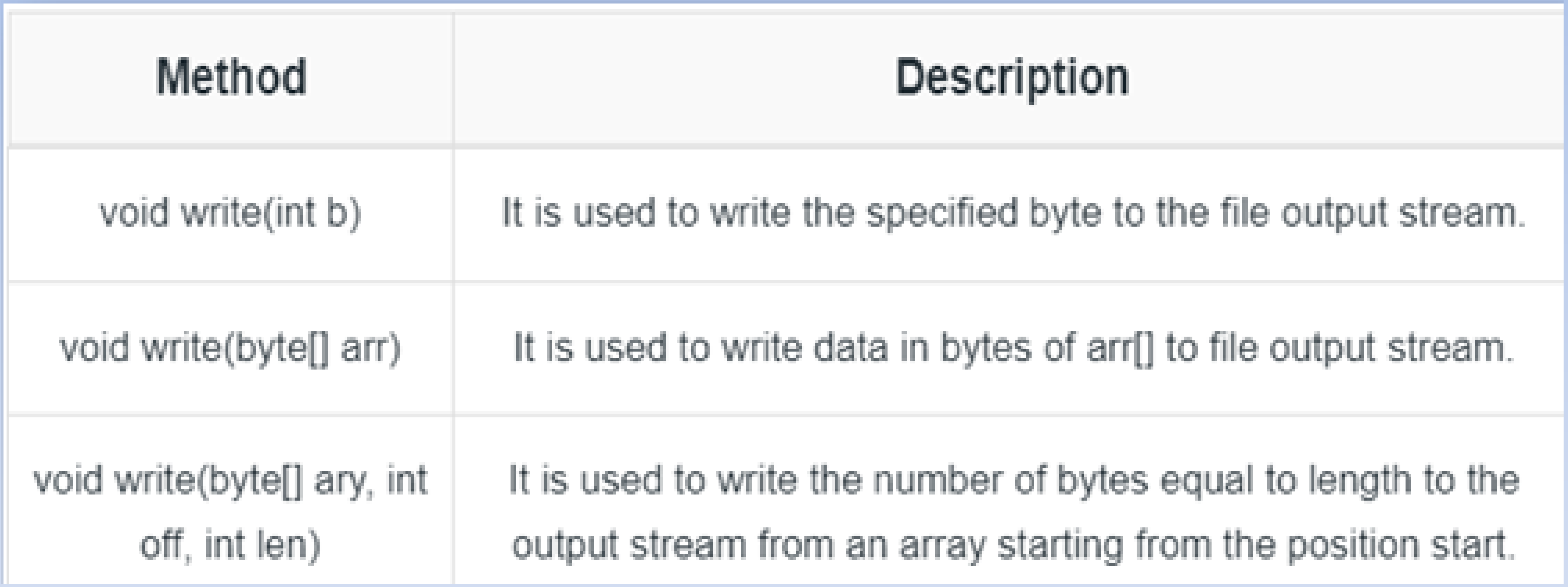
- First, attach a file path to a FileOutputStream as shown here:
  - FileOutputStream fout = new FileOutputStream("file1.txt");
- This will enable us to write data to the file. Then, to write data to the file, we should write data using the FileOutputStream as,
  - fout.write();
- Then we should call the close() method to close the fout file.
  - fout.close()

# Methods of fileOutputStream



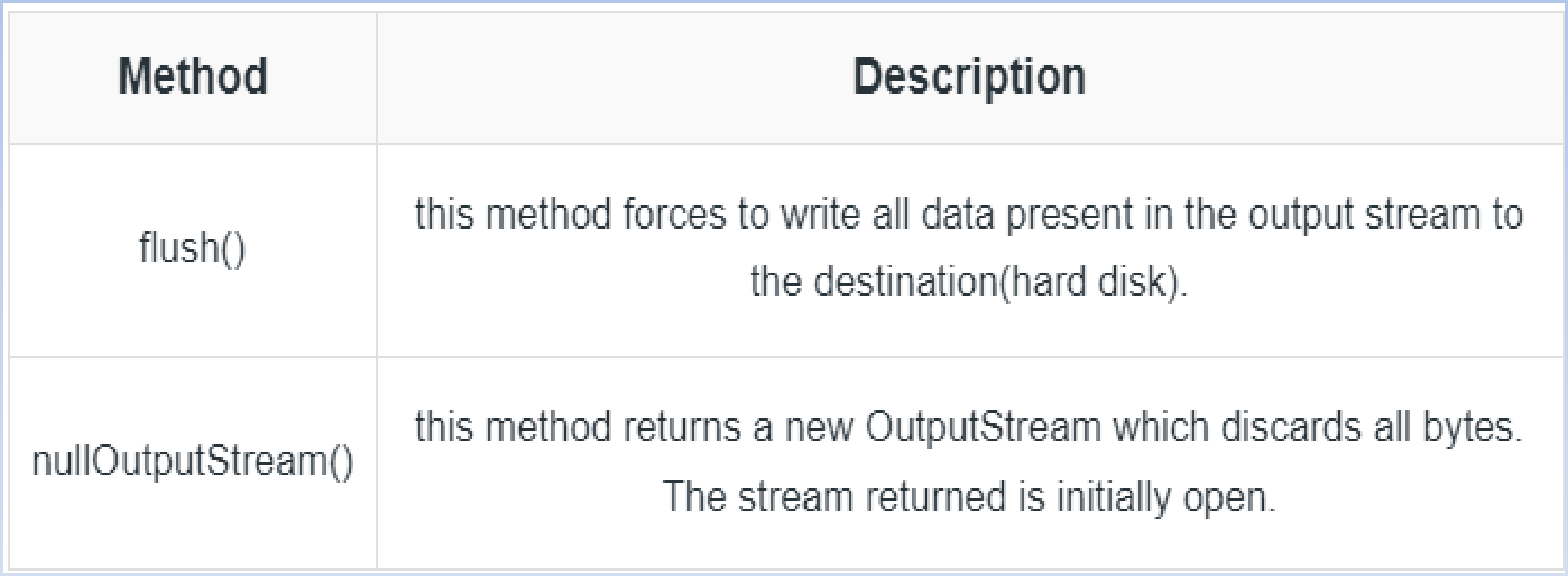
| Method | Description |
|---|---|
| void close() | It closes the file output stream. |
| protected void finalize() | It is used to clean up all the connection with the file output stream and finalize the data. |
| FileChannel getChannel() | Returns the unique FileChannel object associated with this file output stream. |
| FileDescriptor getFD() | It returns the file descriptor associated with the stream. |

# Methods of fileOutputStream



| Method | Description |
|---|---|
| void write(int b) | It is used to write the specified byte to the file output stream. |
| void write(byte[] arr) | It is used to write data in bytes of arr[] to file output stream. |
| void write(byte[] ary, int off, int len) | It is used to write the number of bytes equal to length to the output stream from an array starting from the position start. |

# Methods declared in OutputStream class

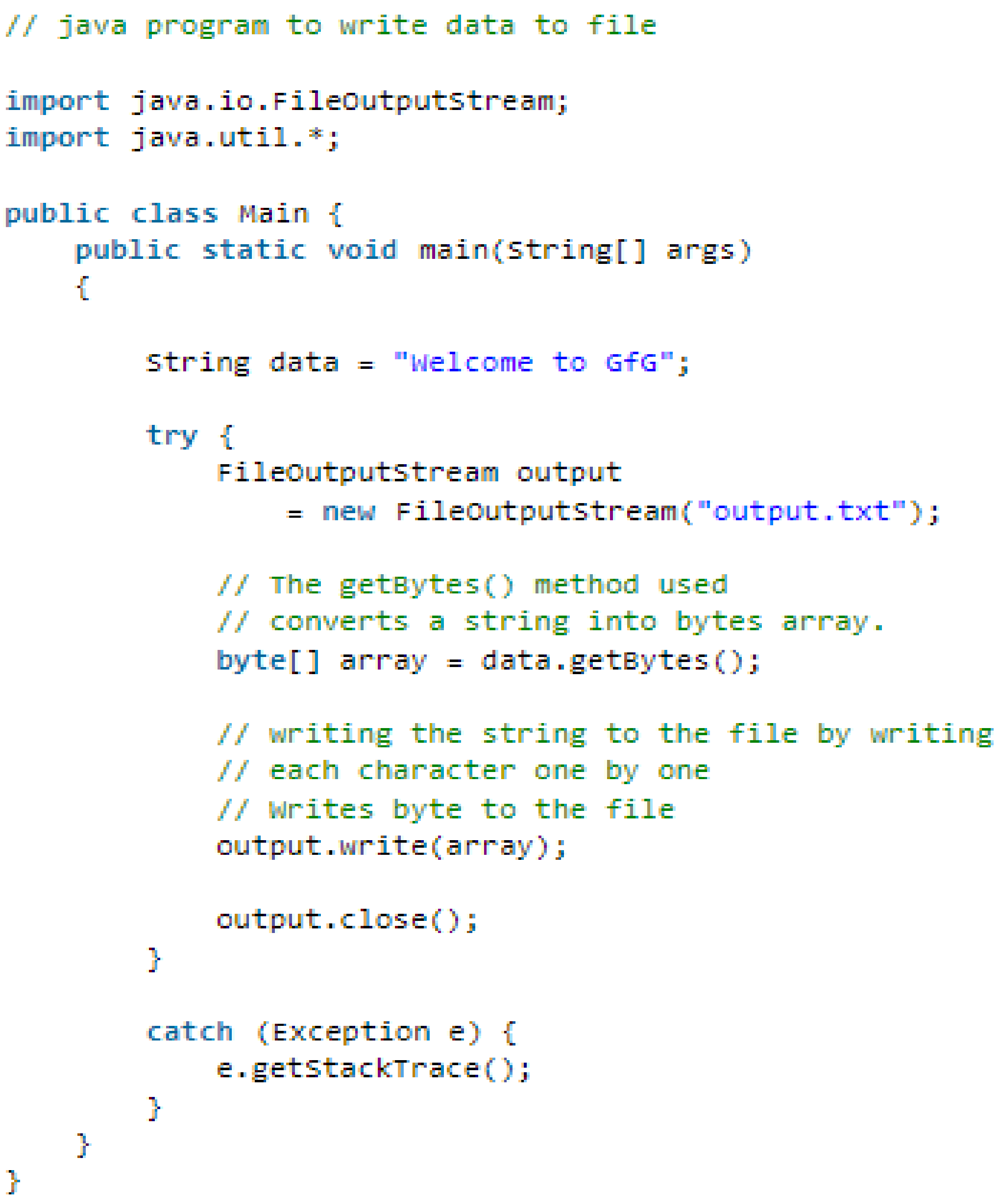


| Method | Description |
|---|---|
| flush() | this method forces to write all data present in the output stream to the destination(hard disk). |
| nullOutputStream() | this method returns a new OutputStream which discards all bytes. The stream returned is initially open. |

## Example

```java
// java program to use FileOutputStream object for writing
// data

import java.io.*;

class FileExample {
    public static void main(String[] args)
        throws IOException
    {
        int i;

          // create a fileoutputstream object
        FileOutputStream fout = new FileOutputStream("../files/name3.txt",
                                             true);

        // we need to transfer this string to files
        String st = "TATA";

        char ch[] = st.toCharArray();
        for (i = 0; i < st.length(); i++) {

            // we will write the string by writing each
            // character one by one to file
            fout.write(ch[i]);
        }

        // by doing fout.close() all the changes which have
        // been made till now in RAM had been now saved to
        // hard disk
        fout.close();
    }
}
```

## Example



```
// java program to write data to file

import java.io.FileOutputStream;
import java.util.*;

public class Main {
    public static void main(String[] args)
    {

        String data = "Welcome to GfG";

        try {
            FileOutputStream output
                = new FileOutputStream("output.txt");

            // The getBytes() method used
            // converts a string into bytes array.
            byte[] array = data.getBytes();

            // writing the string to the file by writing
            // each character one by one
            // Writes byte to the file
            output.write(array);

            output.close();
        }

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

## Example

```java
// java program to show the usage of flush() method
import java.io.FileOutputStream;
import java.io.IOException;

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

        FileOutputStream out = null;
        String data = "Welcome to GfG";

        try {
            out = new FileOutputStream(" flush.txt");

            // Using write() method
            out.write(data.getBytes());

            // Using the flush() method
            out.flush();
            out.close();
        }
        catch (Exception e) {
            e.getStackTrace();
        }
    }
}
```

## Java Stream Filter

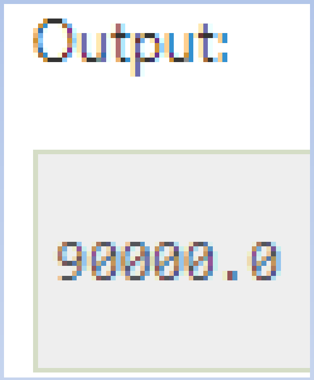
- Java stream provides a method filter() to filter stream elements on the basis of given predicate. Suppose you want to get only even elements of your list then you can do this easily with the help of filter method.
- This method takes predicate as an argument and returns a stream of consisting of resulted elements.

## Java Stream Filter

- ✓ Signature
  - – The signature of Stream filter() method is given below:
    - Stream<T> filter(Predicate<? **super** T> predicate)
- ✓ Parameter
  - – **predicate:** It takes Predicate reference as an argument.
    - Predicate is a functional interface. So, you can also pass lambda expression here.
- ✓ Return
  - – It returns a new stream

## Java Stream filter() example

```
import java.util.*;
class Product{
    int id;
    String name;
    float price;
    public Product(int id, String name, float price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
}
public class JavaStreamExample {
    public static void main(String[] args) {
        List<Product> productsList = new ArrayList<Product>();
        //Adding Products
        productsList.add(new Product(1,"HP Laptop",25000f));
        productsList.add(new Product(2,"Dell Laptop",30000f));
        productsList.add(new Product(3,"Lenevo Laptop",28000f));
        productsList.add(new Product(4,"Sony Laptop",28000f));
        productsList.add(new Product(5,"Apple Laptop",90000f));
        productsList.stream()
                    .filter(p ->p.price> 30000)   // filtering price
                    .map(pm ->pm.price)          // fetching price
                    .forEach(System.out::println);  // iterating price
    }
}
```



Output:

```
90000.0
```