I/O and File Handling

By – Mohammad Imran

I/O Operations **Introduction**

The **java.io** package is used to handle input and output operations. Java IO has various classes that handle input and output sources. **A Stream is also a sequence of data.** It is neither a data structure nor a store of data.

Java input stream classes can be used to read data from input sources such as **keyboard** or a **file**. Similarly output stream classes can be used to write data on a **display** or a **file** again.

These streams support all the types of objects, data-types, characters, files, etc., to fully execute the **I/O operations**.

MyAnatomy

I/O Operations

Standard or Default Stream

Before exploring various input and output streams, There are **3 category of streams classes in IO package:**

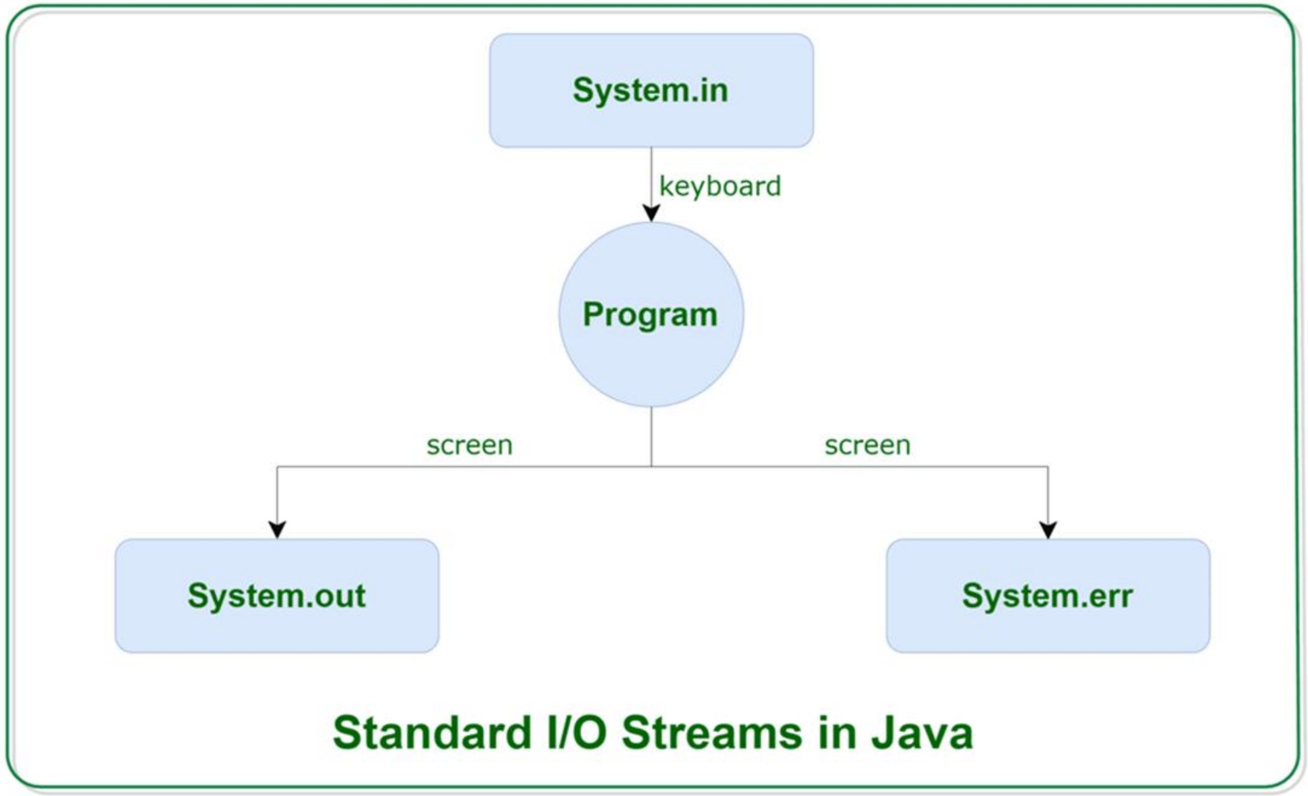
- ✓ Input Streams.
- ✓ Output Streams.
- ✓ Error Streams.

MyAnatomy

I/O Operations

Standard or Default Stream

Java supports three streams that are automatically attached with the console. let's look at **3 standard or default streams** that Java has to provide, which are also most commonly used:



Standard I/O Streams in Java

By – Mohammad Imran

MyAnatomy

I/O Operations

Standard or Default Stream

Java supports three streams that are automatically attached with the console. let's look at **3 standard or default streams** that Java has to provide, which are also most commonly used:

1. **System.out:** This is the standard output stream **(`System.out`)** that is used to produce the result of a program on an output device like the computer screen. Here is a list of the various print functions that we use to output statements:

   ✓ `print()`
   ✓ `println()`
   ✓ `printf()`

MyAnatomy

I/O Operations

Standard or Default Stream

2. **System.in:** This is the standard input stream **(`System.in`)** that is used to read characters from the keyboard or any other standard input device.

3. **System.err:** It is used to display the error messages. It works similarly to **`System.out`** with **`print()`**, **`println()`**, and **`printf()`** methods.

Stream

An **I/O Stream** is a sequence of data that flows from a **source** to a **destination**.

Types of Stream

Depending on the type of operations, streams can be divided into two primary classes:

- ✓ Input Stream
- ✓ Output Stream

Stream

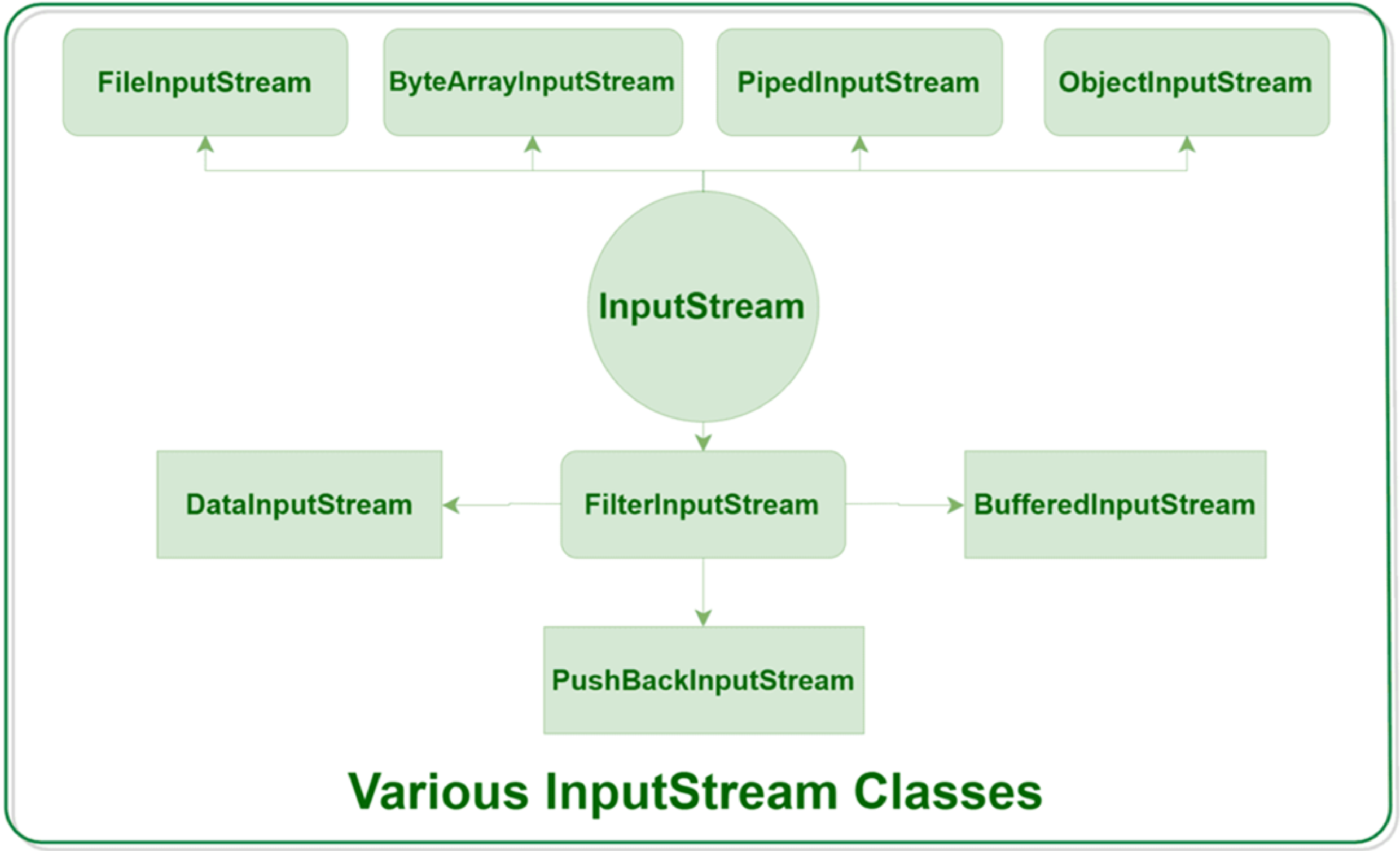
Input Stream

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.

It is an abstract superclass of the **java.io** package and is used to read the data from an input source. In other words, it means reading data from files, using a keyboard, etc. We can create an object of the input stream class using the new keyword. The input stream class has several types of constructors.

MyAnatomy

Input Stream Hierarchy



Various InputStream Classes

By – Mohammad Imran

MyAnatomy

Input Stream Methods

1. `public abstract int read() throws IOException`

   The method above returns the data of the next byte in the input stream. The value returned is between 0 and 255. If no byte is read, the code returns -1, indicating the file's end.

2. `public int available() throws IOException`

   The method above returns the number of bytes that can be read from the input stream.

By – Mohammad Imran

Input Stream Methods

3. `public void close() throws IOException`

The method above closes the current input stream and releases any associated system resources.

4. `public void mark(int readlimit)`

It marks the current position in the input stream. The readlimit argument tells the input stream to read that many bytes before the mark position becomes invalid.

Input Stream Methods

5. `public boolean markSupported()`

It tells whether a particular input stream supports the mark() and reset() method. It returns true if the particular input stream supports the mark and reset methods or returns false.

6. `public int read(byte[ ] b) throws IOException`

The method above reads the bytes from the input stream and stores every byte in the buffer array. It returns the total number of bytes stored in the buffer array. If there is no byte in the input stream, it returns -1 as the stream is at the end of the file.

MyAnatomy

Input Stream Methods

7. `public int read(byte[ ] b , int off , len) throws IOException`

It reads up to len bytes of data from the input stream and returns the total number of bytes stored in the buffer. Here, the "off" is the start offset in buffer array b where the data is written, and the "len" represents the maximum number of bytes to read.

8. `public void reset() throws IOException`

It repositions the stream to the last called mark position. The reset method does nothing for input stream class except throwing an exception.

9. `public long skip(long n) throws IOException`

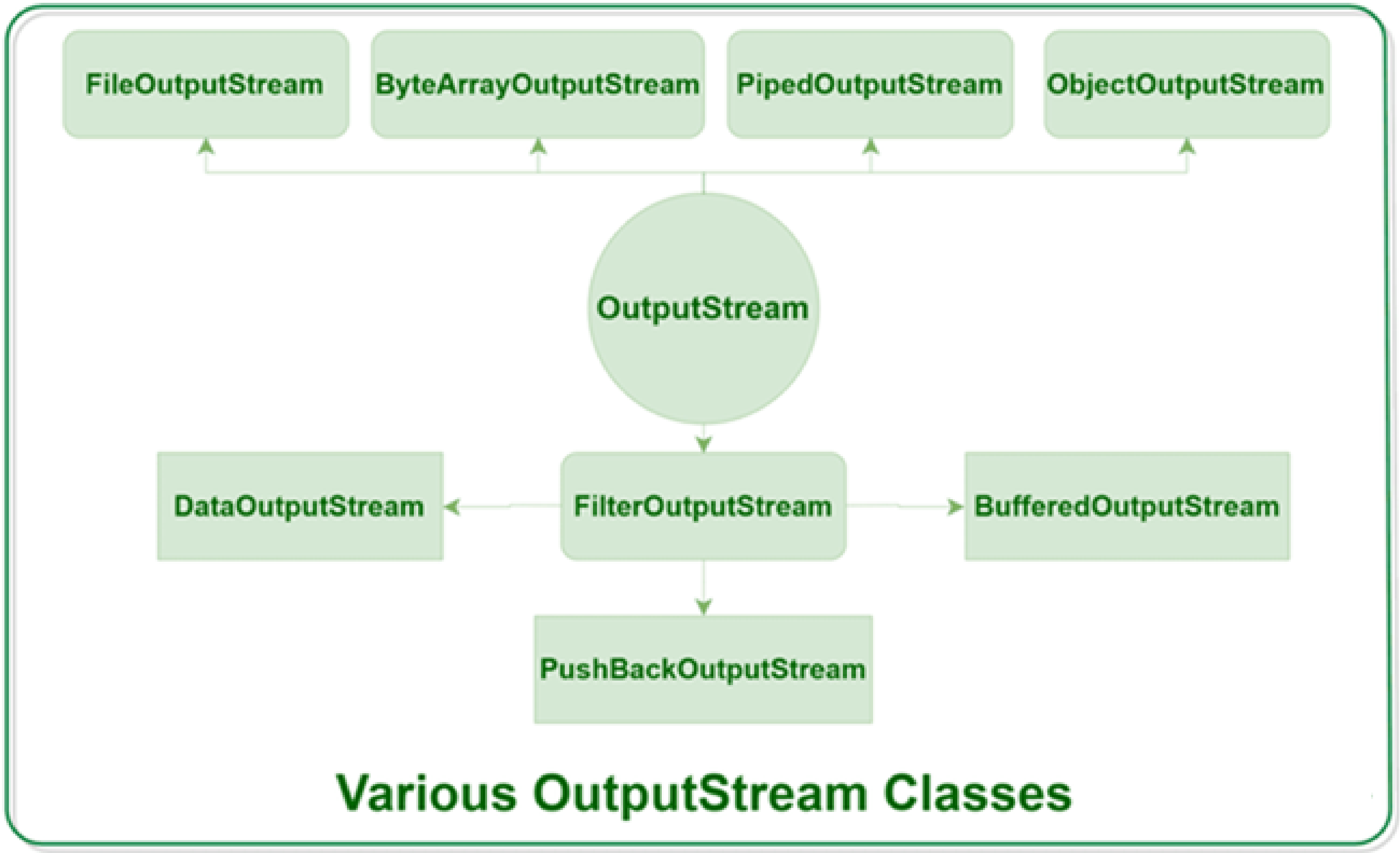
This method discards n bytes of data from the input stream.

Stream

Output Stream

The output stream is used to write data to numerous output devices like the monitor, file, etc. OutputStream is an abstract superclass that represents an output stream. OutputStream is an abstract class and because of this, it is not useful by itself. However, its subclasses are used to write data.

Output Stream Hierarchy



Various OutputStream Classes

By – Mohammad Imran

MyAnatomy

Output Stream Methods

1. `public abstract void write(int b) throws IOException`

The method above writes the specific bytes to the output stream. It does not return a value.

2. `public void write(byte[ ] b) throws IOException`

This method writes the b.length bytes from the specified byte array to the output stream.

3. `public void flush() throws IOException`

It flushes the current output stream and forces any buffered output to be written out.

MyAnatomy

Output Stream

Methods

4. `public void close() throws IOException`

This method closes the current output stream and releases any associated system resources. The closed stream cannot be reopened, and operations cannot be performed on it.

5. `public void write(byte[ ] b ,int off ,int len) throws IOException`

It writes up to len bytes of data for the output stream. Here, the “off” is the start offset in buffer array b, and the “len” represents the maximum number of bytes to be written in the output stream.

By – Mohammad Imran

MyAnatomy

File Handling

A file is a designated location where associated information is stored. Java provides Files as an abstract data type. Java provides multiple file operations to create new files, read, update, and delete. This makes file handling in Java easy.

Java File Handling is a way to carry out read and write operations on a file. Java's package "java.io" contains a File class, enabling us to work with different file formats.

File Operations

The following are the several operations that can be performed on a file in Java:

- ✓ Create a File
- ✓ Read from a File
- ✓ Write to a File
- ✓ Delete a File

MyAnatomy

File Operations

Creating File

The **File** class in Java represents the pathname of a file or directory, not the file content.

It is used to create, delete, inspect, and manipulate files and directories.

Package:

```
    import java.io.File;
```

File Operations

Creating File

How to Create a **File** Object

Syntax:

```
File file = new File("Data.txt");
```

This creates a **File** object pointing to **`Data.txt`**, but does not create the file physically.

MyAnatomy

File Operations

Creating File

Commonly used constructor for **File** class

- ✓ `File(String pathname)` – Creates a file object with a pathname
- ✓ `File(String parent, String child)` – Combines parent and child path
- ✓ `File(File parent, String child)` – Uses another File object as parent

MyAnatomy

File Operations

File Methods

The following table depicts several **File** Class methods:

Method Name	Description	Return Type
canRead()	It tests whether the file is readable or not.	Boolean
canWrite()	It tests whether the file is writable or not.	Boolean
createNewFile()	It creates an empty file.	Boolean
delete()	It deletes a file.	Boolean
exists()	It tests whether the file exists or not.	Boolean
length()	Returns the size of the file in bytes.	Long
getName()	Returns the name of the file.	String
list()	Returns an array of the files in the directory.	String[]
mkdir()	Creates a new directory.	Boolean
getAbsolutePath()	Returns the absolute pathname of the file.	String

MyAnatomy

File Operations | Creating File | Example – 2

```java
import java.io.File;
import java.io.IOException;
public class CreateFile2
{
    public static void main(String[] args)
    {
        File file = new File("Data.txt");
        try
        {
            if (file.createNewFile())
            {
                System.out.println("File created successfully: " + file.getName());
            }
            else
            {
                System.out.println(file.getName() + " -> File already exists.");
            }
        }
```

MyAnatomy

File Operations | Creating File | Example – 2

```
        catch (IOException e)
        {
            System.out.println("An error occurred while creating the file.");
            e.printStackTrace();
        }
    }
}
```

MyAnatomy

File Operations | exists() and length() | Example – 3

```java
import java.io.File;
public class ExistLengthMethod
{
    public static void main(String[] args)
    {
        File file = new File("Data.txt");
        if (file.exists())
        {
            System.out.println("File name: " + file.getName());
            System.out.println("File size: " + file.length() + " bytes");
        }
        else
        {
            System.out.println("File does not exist.");
        }
    }
}
```

MyAnatomy

File Operations | getAbsolutePath() | Example – 4

```
import java.io.File;
public class AbsolutePathExample
{
   public static void main(String[] args)
   {
      File file = new File("Data.txt");
      if(file.exists())
      {
         System.out.println("File Name       : " + file.getName());
         System.out.println("Absolute Path   : " + file.getAbsolutePath());
      }
      else
      {
         System.out.println("File doesnot exist.");
      }
   }
}
```

MyAnatomy

File Operations | directory() and list() | Example – 4

```
import java.io.File;
public class DirectoryListMethods
{
   public static void main(String[] args)
   {
      File directory = new File("MyDirectory");   // Create a new directory
      if (!directory.exists())
      {
         if (directory.mkdir())
         {
            System.out.println("Directory created:" + directory.getName());
         }
         else
         {
            System.out.println("Failed to create directory.");
         }
      }
```

MyAnatomy

File Operations | directory() and list() | Example – 4

```
        else {
            System.out.println("Directory already exists.");
        }
        File currentDir = new File(".");
        String[] filesList = currentDir.list();
        if (filesList != null) {
            System.out.println("\nContents of current directory:");
            for (String fileName : filesList) {
                System.out.println(fileName);
            }
        }
        else {
            System.out.println("Directory is empty or does not exist.");
        }
    }
}
```

File Operations

Writing to a File

Writing to a file in Java means storing data into a file on the disk using classes from the **`java.io`** or **`java.nio`** package. The most common classes for writing are:

- ✓ **FileWriter** – Writes character data to a file.
- ✓ **BufferedWriter** – Adds buffering for efficient writing.
- ✓ **PrintWriter** – Supports formatted text writing.

MyAnatomy

FileWriter Class

The **`FileWriter`** class in Java is a part of the **`java.io`** package and is used to write character data to files. It is a convenient class for writing streams of characters, as opposed to binary data.

It is a subclass of **`OutputStreamWriter`**, which means it writes characters to an output stream by converting them into bytes using a specified charset.

MyAnatomy

FileWriter Class

Constructors

Constructor	Description
FileWriter(String fileName)	Creates a writer to the file with the given name (overwrites if exists).
FileWriter(String fileName, boolean append)	If append is true, it appends to the file instead of overwriting.
FileWriter(File file)	Accepts a File object to write into.
FileWriter(FileDescriptor fd)	Writes to the file represented by the given file descriptor.

By – Mohammad Imran

MyAnatomy

FileWriter Class

Methods

Method	Description
write(String str)	Writes a string to the file.
write(char[] cbuf)	Writes an array of characters.
write(int c)	Writes a single character.
flush()	Flushes the stream (forces writing of buffered data).
close()	Closes the stream and releases resources.

MyAnatomy

FileWriter Class

Example – 5

```
import java.io.*;
public class FileWriter1
{
    public static void main(String[] args)
    {
        try
        {
            FileWriter fw = new FileWriter("Data.txt");
            fw.write("Java FileWriter Example\nWriting to a file made easy.");
            fw.close();
            System.out.println("File written successfully.");
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}
```

By – Mohammad Imran

MyAnatomy

FileWriter Class

Behavior

✓ If the file does not exist:

➤ **`FileWriter`** will create a new file automatically.

✓ If the file already exists:

➤ It will overwrite the file by default (unless you set **`append = true`**).

MyAnatomy

FileWriter Class | User Input | Example – 6

```
import java.io.*;
import java.util.Scanner;
public class UserInputToFile
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        try
        {
            // Create FileWriter object
            FileWriter writer = new FileWriter("UserInput.txt");
            System.out.println("Enter text to write to the file (type 'exit' to
            finish):");
            while (true)
            {
                String line = scanner.nextLine();
                // Exit condition
                if (line.equalsIgnoreCase("exit"))
```

MyAnatomy

FileWriter Class | User Input | Example – 6

```
                {
                    break;
                }
                // Write line to the file
                writer.write(line + "\n");
            }
            // Close writer and scanner
            writer.close();
            scanner.close();
            System.out.println("User input written to user_input.txt");
        }
        catch (IOException e)
        {
            System.out.println("An error occurred while writing to the file.");
            e.printStackTrace();
        }
    }
}
```

By – Mohammad Imran

MyAnatomy

File Operations

Reading from File

Reading from a file in Java means retrieving data stored in a file using file-handling classes. The most common classes used for reading character data are **`FileReader`** and **`BufferedReader`**.

MyAnatomy

FileReader Class

The **`FileReader`** class in Java is a part of the **`java.io`** package and is used to read character data from a file. It is designed for reading text files only, not binary files.

MyAnatomy

FileReader Class

Reading from File

Constructor	Description
FileReader(String fileName)	Creates a new FileReader, given the name of the file as a string.
FileReader(File file)	Creates a new FileReader, given the File object.
FileReader(FileDescriptor fd)	Creates a new FileReader associated with a file descriptor.

MyAnatomy

FileReader Class

Methods

Method	Description
int read()	Reads a single character. Returns -1 if end of file is reached.
int read(char[] cbuf)	Reads characters into an array. Returns number of characters read or -1 if end of file.
int read(char[] cbuf, int offset, int length)	Reads up to length characters into an array starting at offset.
void close()	Closes the stream and releases system resources.

MyAnatomy

FileReader Class Example – 7

```
import java.io.*;
public class FileReaderExample
{
    public static void main(String[] args)
    {
        try
        {
            FileReader fr = new FileReader("UserInput.txt");
            int ch;
            while ((ch = fr.read()) != -1)
            {
                System.out.print((char) ch);  // Casting int to char
            }
            fr.close();
        }
        catch (IOException e)
        {
            e.printStackTrace(); } } }
```

By – Mohammad Imran

MyAnatomy

Optional

Lets understand the small program.

```
public class WithoutOptional
{
    public static void main(String[] args)
    {
        String[] words = new String[10];
        String word = words[5].toLowerCase();
        System.out.print(word);
    }
}
```

OUTPUT

```
Exception in thread "main"
java.lang.NullPointerException
```

MyAnatomy

Quiz

By – Mohammad Imran

MyAnatomy

Quiz – 1

What keyword is used to define a module in Java?

a) package
b) module
c) import
d) class

Answer

b

By – Mohammad Imran

MyAnatomy

Coding Questions

By – Mohammad Imran

MyAnatomy

Question – 1

Problem Statement: ()

Click here to see code

By – Mohammad Imran