Exploring Java I/O, Directories

- ➢ The data is lost when a variable goes out of scope, or the program is terminated; this storage is temporary.
- ➢ It is difficult to handle large volumes of data using variables and arrays.

We can overcome these problems by storing data on secondary devices such as floppy or hard disks. The data is stored in these devices using the concept of files.

Concept of File:

A file is a data structure where a considerable amount of data can be stored. Typically, any programming language supports two types of files-

Text Files

- Definition: Data is stored in a human-readable format (plain text).
- **Characteristics:**
  - Easily readable and editable using any text editor.
  - Slightly slower to process compared to binary files.
  - Ideal for source code, configuration files, logs, etc.
- **Examples:**
  - .txt (Text Document)
  - .java (Java source file)
  - .csv (Comma Separated Values)
  - .xml, .html, .json (structured text formats)

Binary Files

- Definition: Data is stored in a machine-readable binary format (not human-readable).
- **Characteristics:**
  - Faster and more efficient to process.
  - Not readable using a plain text editor without decoding.
  - Suitable for compiled code, images, audio, videos, etc.
- **Examples:**
  - .class (Compiled Java class files)

    - .exe (Executable file)
    - .jpg, .png (Images)
    - .mp3, .mp4 (Audio/Video)
    - .dat (Raw binary data)

File Handling in Java

File handling is a core concept in Java that allows programs to interact with files stored on a computer. Java provides the File class, part of the java.io package, to perform various file-related operations such as creating, reading, writing, and deleting files.

Why is File Handling Important?

File handling is essential in programming because it allows you to:

- **Store data permanently:** Unlike variables that lose their data when a program ends, files retain data even after the program stops.
- **Read and write data:** You can save program outputs, configuration settings, logs, or user data to files.
- **Process large data sets:** Files are useful when working with extensive data that cannot be handled entirely in memory.

In Simple Terms

File handling means:

- Reading data from a file
- Writing data to a file

What is the File Class in Java?

- The File class in Java represents a file or directory in the file system.
- The File class is used to access file information (such as name, path, size, and permissions) but cannot directly read or write the file content.
- It works with the attributes or properties of a file, not with its actual data.
- It helps check file existence, type (file or directory), and other metadata before performing file operations.
- The File class belongs to the java.io package.

- While Java is platform-independent, file naming conventions depend on the underlying Operating System.
- In Java, even directories are treated as File objects, often called "directory files".
- To work with a file, you first create an object of the File class and pass the name or path of the file as a parameter. This object refers to the file or directory you want to work with.

Constructors of the File Class

- File(String dirPath);   // Creates a file object with full path
- File(String dirPath, String fileName);   // Creates a file object by combining directory path and file name
- File(File parent, String fileName);   // Uses another File object as the directory

Parameter Meanings:

- dirPath → The directory path
- fileName → The name of the file
- parent → A File object referring to the parent directory

Examples:

```
File f1 = new File("D:\\Java Code\\IO\\First.java");
File f2 = new File("D:\\Java Code\\IO", "First.java");
File f3 = new File(f2, "First.java");
```

Note: Use double backslashes (\\) in Windows paths inside Java strings.

This works (no escape sequences involved):

```
File f1 = new File("D:\JavaCode\IO\First.java");
```

This will cause an error or unexpected behavior:

```
File f1 = new File("D:\JavaCode\IO\notes.java");  // '\n' is a newline!
```

File class methods

1.String getName() : Return the name of the directory or file of the invoking object.

2.String getPath(): Return the path of the invoking object.

3.String getAbsolutePath(): Return the absolute path of the invoking object.

4.String getParent(): return the name of the parent directory of the invoking object

5.boolean exists(): Return true if the dir/file object exists; otherwise, it is false.

6.boolean canWrite(): Return true if the file object is writable, otherwise false.

7.boolean canRead(): Return true if the file object is readable otherwise, false.

8.boolean isDirectory() :Return true if invoking object is a directory,otherwise false

9.boolean isFile() : Return true if invoking object is a file ,otherwise false

10.boolean isHidden(): Return true if invoking file is a object, otherwise false

11.long length(): Return a long value representing the size of the file in bytes.

12.long lastModified(): return a long value representing the last modified date in milliseconds.

13.boolean renameTo(): Renames the file object with the new name; return true upon successful renaming otherwise false.

14.boolean delete(): deletes the file of the invoking object; returns true on successful deletion, otherwise false.

15.boolean setReadonly(): Set the file of the invoking object to read only mode; return true on successful setting otherwise false.

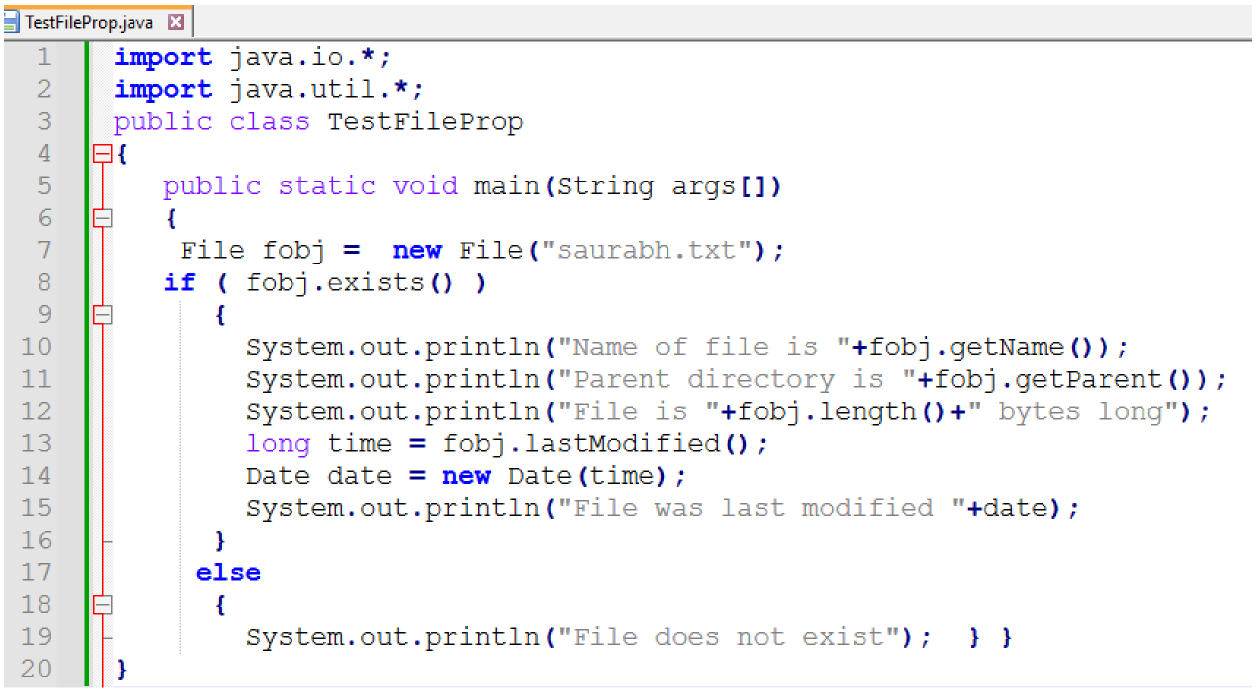
16. boolean deleteOnExit(): Deletes the file of invoking object upon the termination of Java virtual machine.

17.**String[]list** : This methods returns all the content of directory file.

EXAMPLES

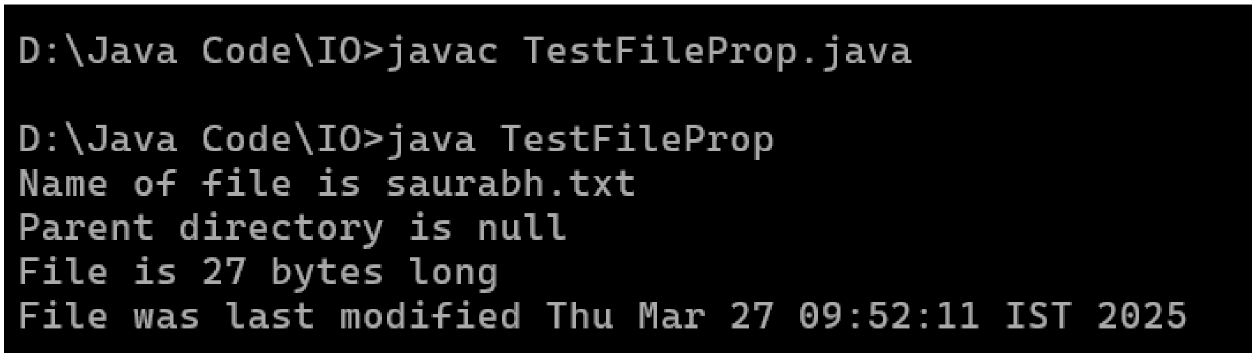
Using functions exists(),getName(),getParent(),length() &   lastModified().

Program Code:


TestFileProp.java

```java
import java.io.*;
import java.util.*;
public class TestFileProp
{
    public static void main(String args[])
    {
     File fobj =  new File("saurabh.txt");
    if ( fobj.exists() )
       {
         System.out.println("Name of file is "+fobj.getName());
         System.out.println("Parent directory is "+fobj.getParent());
         System.out.println("File is "+fobj.length()+" bytes long");
         long time = fobj.lastModified();
         Date date = new Date(time);
         System.out.println("File was last modified "+date);
       }
      else
       {
         System.out.println("File does not exist");  } }
}
```

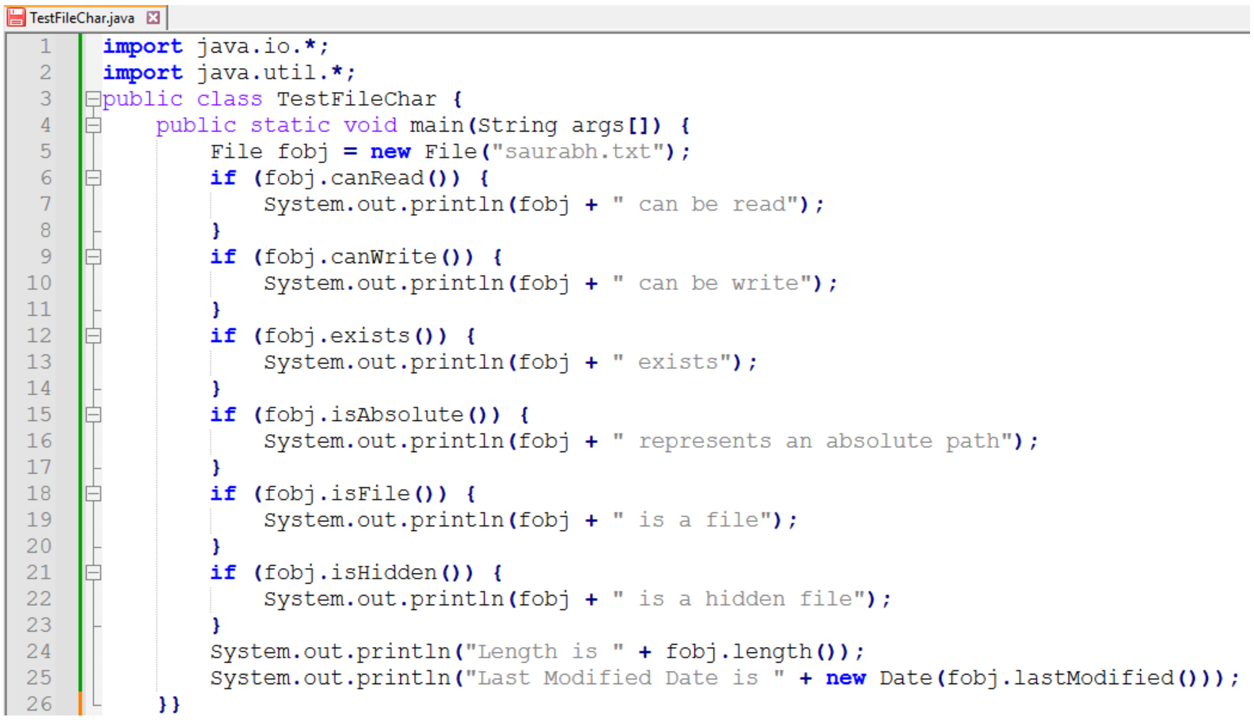
Output:



```
D:\Java Code\IO>javac TestFileProp.java

D:\Java Code\IO>java TestFileProp
Name of file is saurabh.txt
Parent directory is null
File is 27 bytes long
File was last modified Thu Mar 27 09:52:11 IST 2025
```

2.Using canRead(), canWrite(), exists(), isAbsolute(), isFile() & isHidden().

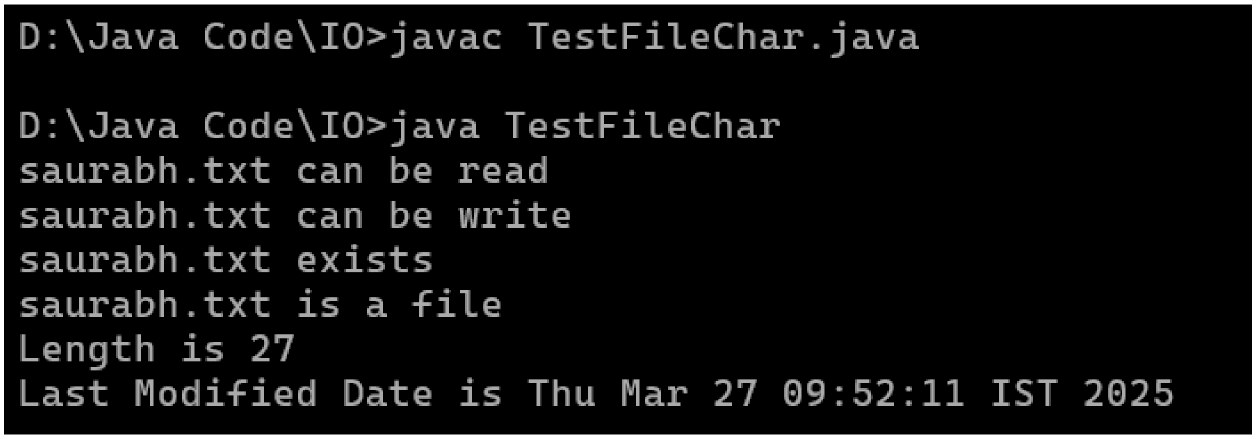
Program Code



TestFileChar.java

```java
import java.io.*;
import java.util.*;
public class TestFileChar {
    public static void main(String args[]) {
        File fobj = new File("saurabh.txt");
        if (fobj.canRead()) {
            System.out.println(fobj + " can be read");
        }
        if (fobj.canWrite()) {
            System.out.println(fobj + " can be write");
        }
        if (fobj.exists()) {
            System.out.println(fobj + " exists");
        }
        if (fobj.isAbsolute()) {
            System.out.println(fobj + " represents an absolute path");
        }
        if (fobj.isFile()) {
            System.out.println(fobj + " is a file");
        }
        if (fobj.isHidden()) {
            System.out.println(fobj + " is a hidden file");
        }
        System.out.println("Length is " + fobj.length());
        System.out.println("Last Modified Date is " + new Date(fobj.lastModified()));
    }}
```

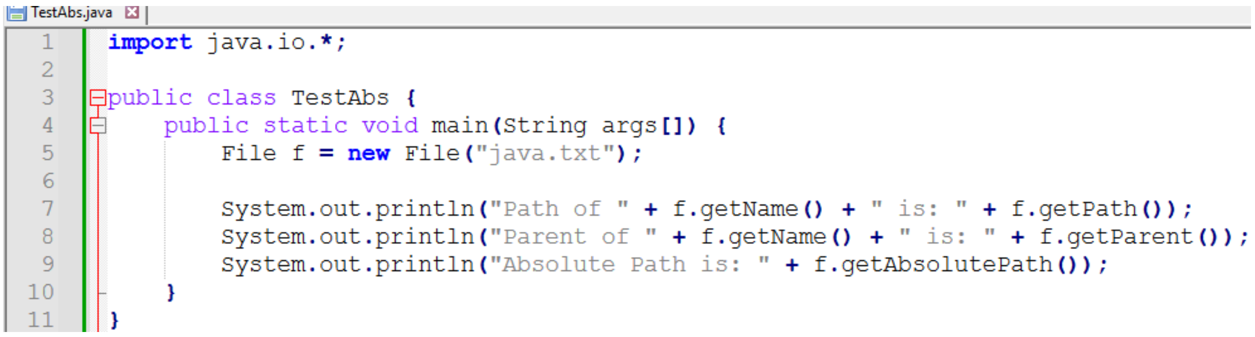
Output:



```
D:\Java Code\IO>javac TestFileChar.java

D:\Java Code\IO>java TestFileChar
saurabh.txt can be read
saurabh.txt can be write
saurabh.txt exists
saurabh.txt is a file
Length is 27
Last Modified Date is Thu Mar 27 09:52:11 IST 2025
```

3.getAbsolutePath() absolute path.

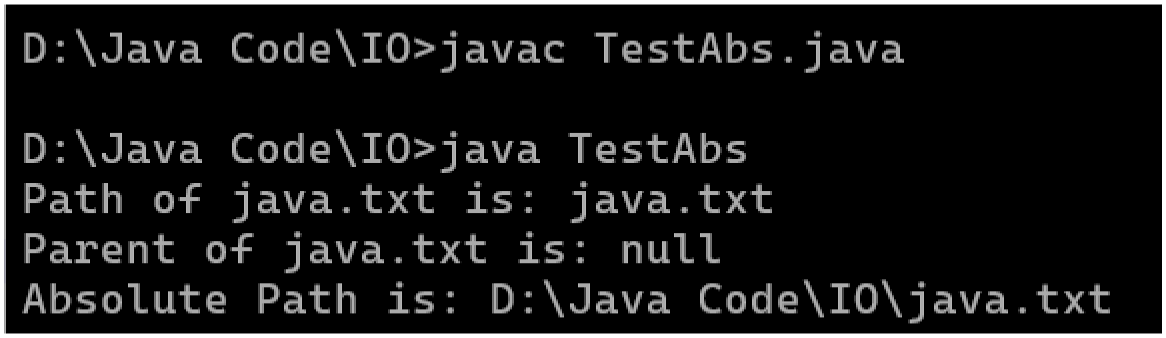
Program Code



TestAbs.java

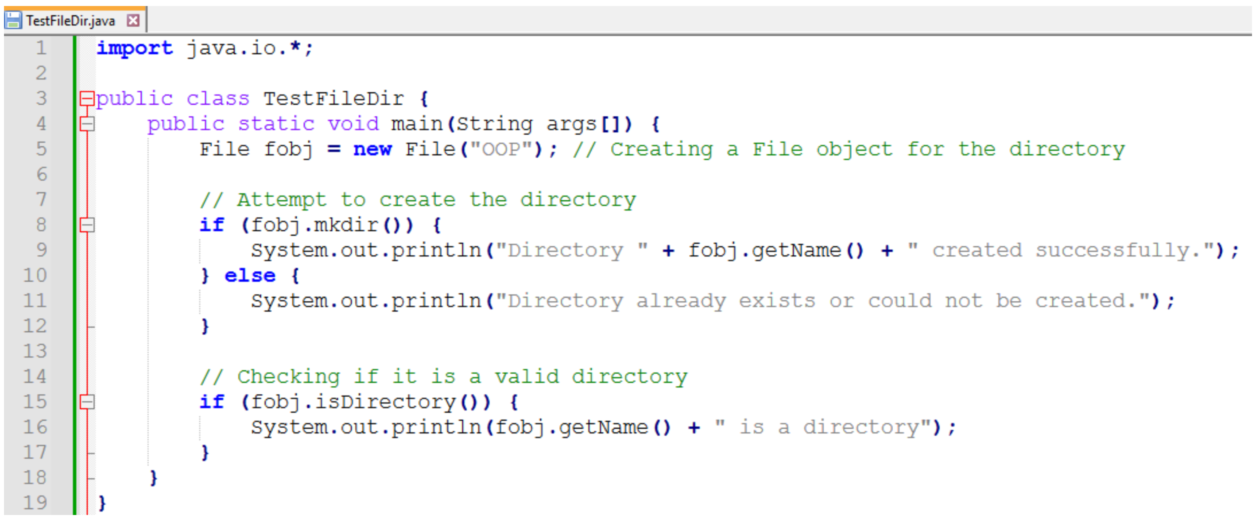
```java
import java.io.*;

public class TestAbs {
    public static void main(String args[]) {
        File f = new File("java.txt");

        System.out.println("Path of " + f.getName() + " is: " + f.getPath());
        System.out.println("Parent of " + f.getName() + " is: " + f.getParent());
        System.out.println("Absolute Path is: " + f.getAbsolutePath());
    }
}
```

Output:



```
D:\Java Code\IO>javac TestAbs.java

D:\Java Code\IO>java TestAbs
Path of java.txt is: java.txt
Parent of java.txt is: null
Absolute Path is: D:\Java Code\IO\java.txt
```

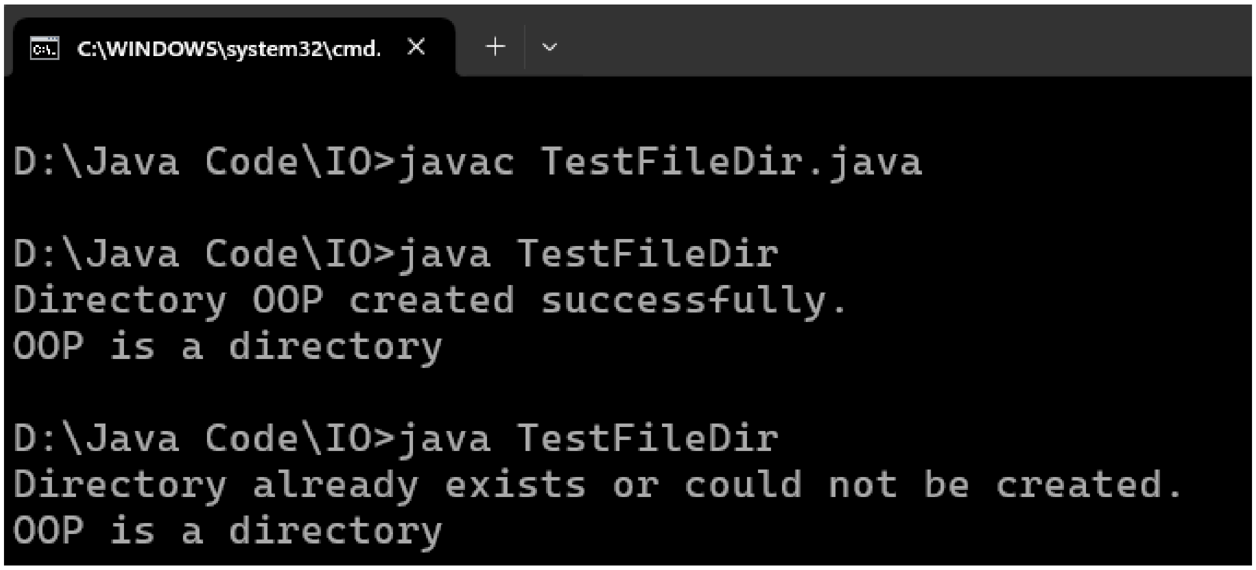
4. mkdir() & isDirectory()



TestFileDir.java

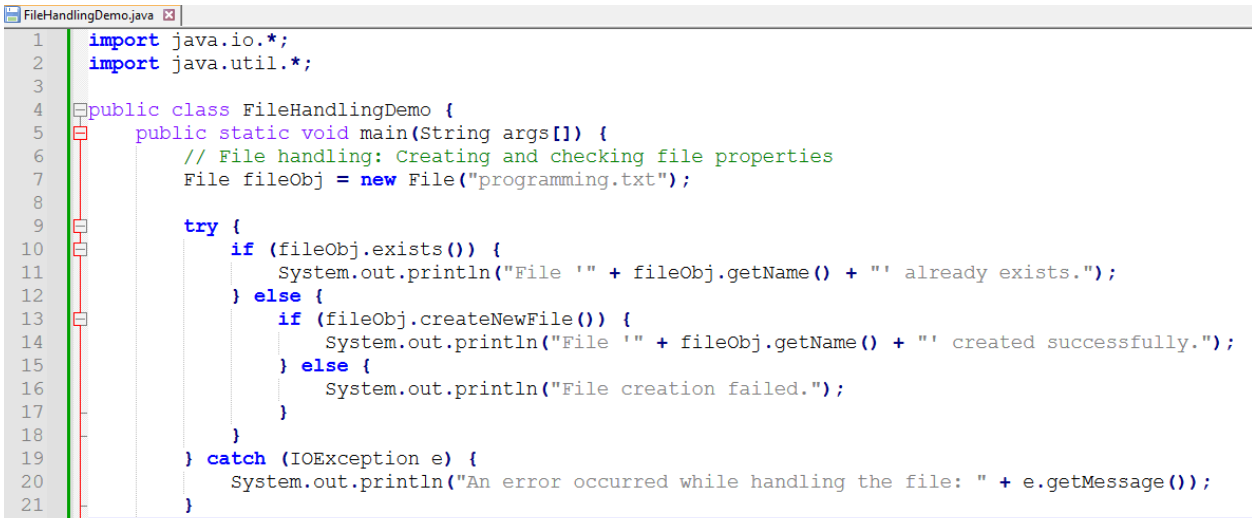
```java
import java.io.*;

public class TestFileDir {
    public static void main(String args[]) {
        File fobj = new File("OOP"); // Creating a File object for the directory

        // Attempt to create the directory
        if (fobj.mkdir()) {
            System.out.println("Directory " + fobj.getName() + " created successfully.");
        } else {
            System.out.println("Directory already exists or could not be created.");
        }

        // Checking if it is a valid directory
        if (fobj.isDirectory()) {
            System.out.println(fobj.getName() + " is a directory");
        }
    }
}
```

Output:



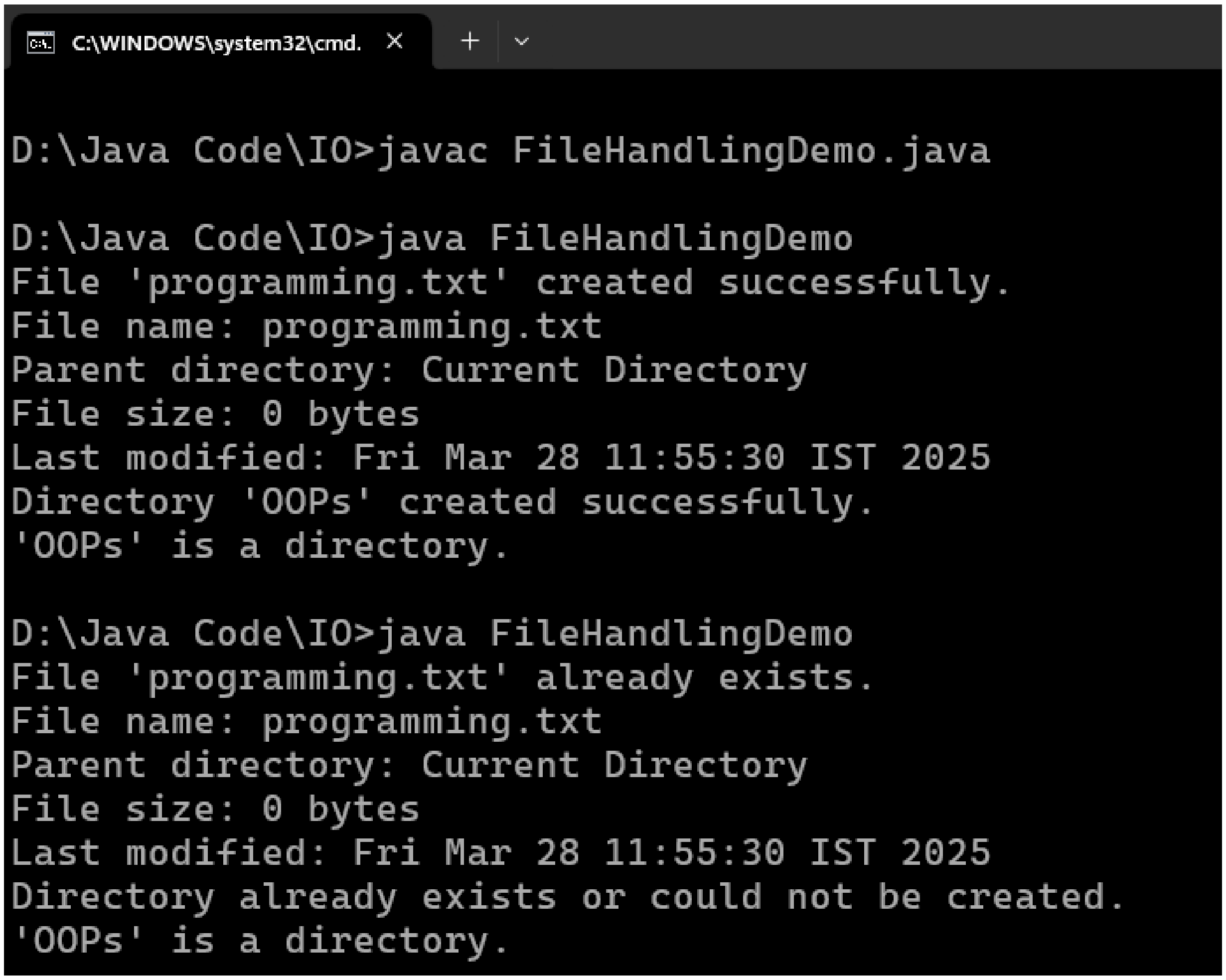
```
D:\Java Code\IO>javac TestFileDir.java

D:\Java Code\IO>java TestFileDir
Directory OOP created successfully.
OOP is a directory

D:\Java Code\IO>java TestFileDir
Directory already exists or could not be created.
OOP is a directory
```

All Methods Example



```java
import java.io.*;
import java.util.*;

public class FileHandlingDemo {
    public static void main(String args[]) {
        // File handling: Creating and checking file properties
        File fileObj = new File("programming.txt");

        try {
            if (fileObj.exists()) {
                System.out.println("File '" + fileObj.getName() + "' already exists.");
            } else {
                if (fileObj.createNewFile()) {
                    System.out.println("File '" + fileObj.getName() + "' created successfully.");
                } else {
                    System.out.println("File creation failed.");
                }
            }
        } catch (IOException e) {
            System.out.println("An error occurred while handling the file: " + e.getMessage());
        }
        // Checking file properties if it exists
        if (fileObj.exists()) {
            System.out.println("File name: " + fileObj.getName());
            System.out.print("Parent directory: " );
            System.out.println((fileObj.getParent() != null ? fileObj.getParent() : "Current Directory"));
            System.out.println("File size: " + fileObj.length() + " bytes");
            System.out.println("Last modified: " + new Date(fileObj.lastModified()));
        } else {
            System.out.println("File does not exist.");
        }

        // Directory handling: Creating and checking directories
        File dirObj = new File("OOPs");

        if (dirObj.mkdir()) {
            System.out.println("Directory '" + dirObj.getName() + "' created successfully.");
        } else {
            System.out.println("Directory already exists or could not be created.");
        }

        // Checking if the directory exists
        if (dirObj.isDirectory()) {
            System.out.println("'" + dirObj.getName() + "' is a directory.");
        }
    }
}
```

Output:



```
D:\Java Code\IO>javac FileHandlingDemo.java

D:\Java Code\IO>java FileHandlingDemo
File 'programming.txt' created successfully.
File name: programming.txt
Parent directory: Current Directory
File size: 0 bytes
Last modified: Fri Mar 28 11:55:30 IST 2025
Directory 'OOPs' created successfully.
'OOPs' is a directory.

D:\Java Code\IO>java FileHandlingDemo
File 'programming.txt' already exists.
File name: programming.txt
Parent directory: Current Directory
File size: 0 bytes
Last modified: Fri Mar 28 11:55:30 IST 2025
Directory already exists or could not be created.
'OOPs' is a directory.
```

Streams in Java

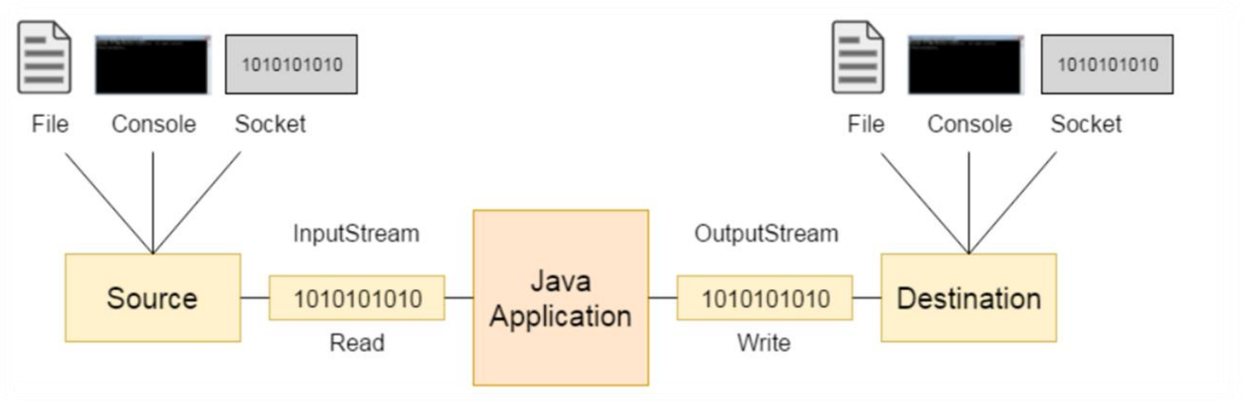
- A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.
- It is used for input and output (I/O) operations such as reading data from a file or writing data to a file, network socket, or console.
- Java uses the concept of a stream to make I/O operations fast. The java.io package contains all the classes required for input and output operations.

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

1. **System.in:** Standard input stream that reads characters from the keyboard or any other standard input device.
2. **System.out:** A standard output stream produces the result of a program on an output device like a computer screen.
3. **System.err:** A standard error stream outputs all the error data a program might throw on a computer screen or any standard output device.

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

- ➢ InputStream
- ➢ OutputStream



In Java, there are two main types of streams based on the type of data they handle:

Type	**Handles**	**Examples**
Byte Streams	Raw bytes (binary data like images, videos)	InputStream, OutputStream
Character Streams	Characters (text data like .txt, .csv)	FileReader, FileWriter

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

- An InputStream reads data from a source (such as a file, keyboard, or network).
- It is an abstract class in the java.io package.
- As an abstract class, InputStream cannot be used directly. Instead, we use its subclasses, which provide concrete implementations.
- Common Subclasses of `InputStream`:

Subclass	**Description**
`FileInputStream`	Reads data from files
`ByteArrayInputStream`	Reads data from byte arrays
`AudioInputStream`	Reads audio data
`FilterInputStream`	Provides filtered input streams
`StringBufferInputStream`	Deprecated – used to read characters from a `StringBuffer`
`ObjectInputStream`	Reads objects previously written by `ObjectOutputStream`

Example: Creating an InputStream

InputStream input = new FileInputStream("example.txt");

Common Methods of InputStream

Method	**Description**
int read()	Reads a single byte of data
int read(byte[] array)	Reads multiple bytes into an array
int available()	Returns the number of available bytes
void close()	Closes the input stream
boolean markSupported()	Checks if the stream supports the mark() and reset() methods

Method	**Description**
void mark(int readlimit)	Marks the current position in the stream
void reset()	Resets the stream to the marked position
long skip(long n)	Skips and discards n bytes of data

Output Stream

- An **OutputStream** writes **data** to a destination (such as a file, monitor, or network).
- Like InputStream, OutputStream is also an **abstract class** and must be subclassed to be used.

Common Subclasses of OutputStream:

Subclass	**Description**
FileOutputStream	Writes data to files
ByteArrayOutputStream	Writes data to byte arrays
DataOutputStream	Writes primitive data types
PrintStream	Prints formatted representations of objects
ObjectOutputStream	Writes objects to the output stream
StringBufferOutputStream	Deprecated – used to write characters to a StringBuffer

Example: Creating an OutputStream

```
OutputStream output = new FileOutputStream("example.txt");
```

Common Methods of OutputStream

Method	**Description**
void write(int b)	Writes a single byte
void write(byte[] array)	Writes a sequence of bytes from an array
void flush()	Forces all buffered data to be written immediately

Method	**Description**
void close()	Closes the output stream

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

What is FileReader?

- FileReader is a character-based stream in Java.
- It is used to read data from a file — character by character.
- It is mainly used for reading text files (like .txt, .csv).
- It automatically handles character encoding (like UTF-8).

How to Create a FileReader?

FileReader fr = new FileReader("filename.txt");

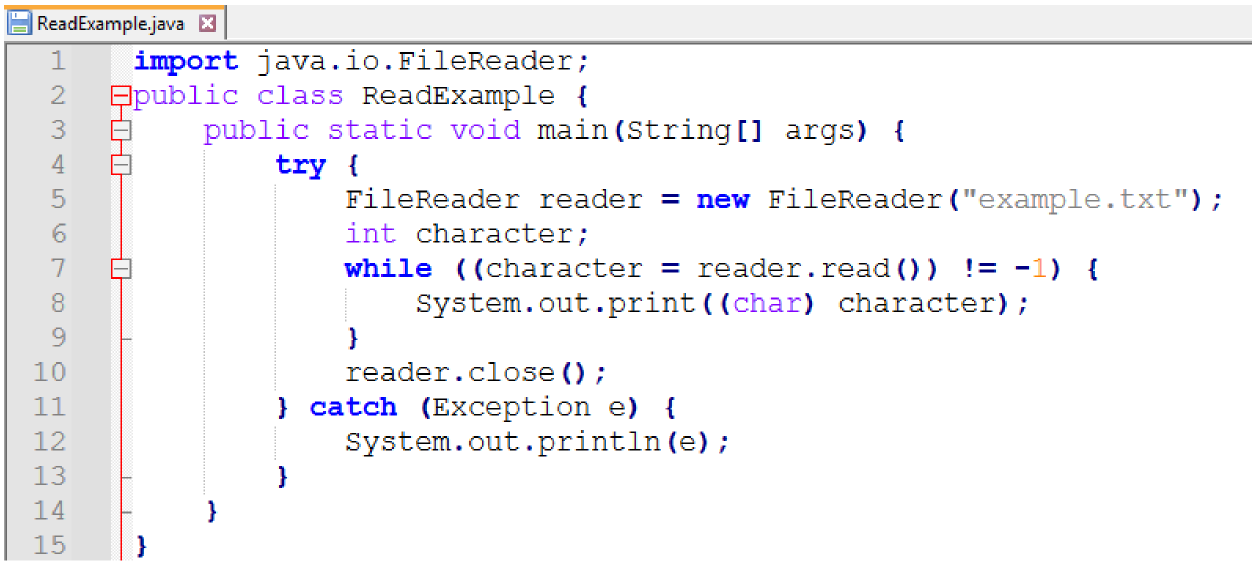
- This opens the file filename.txt for reading.

If the file doesn't exist, it throws a FileNotFoundException.

Common Methods of FileReader

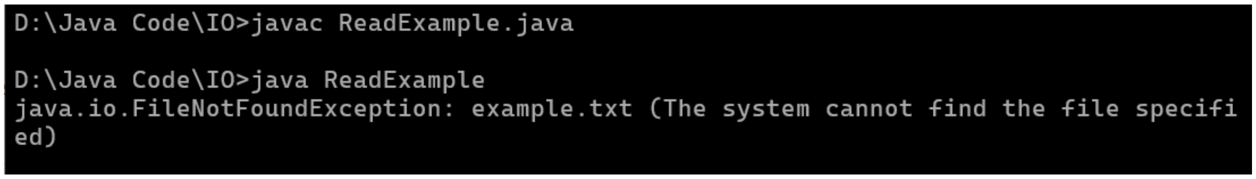
Method	**Description**
read()	Reads a single character and returns it as an integer. Returns -1 when the end of file is reached.
read(char[] cbuf)	Reads characters into a char array.
close()	Closes the FileReader to free resources.

Example of FileReader



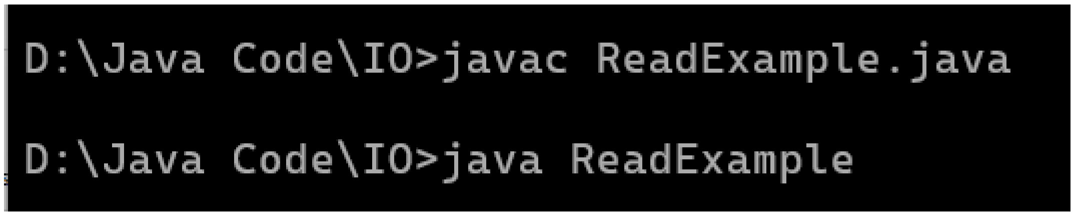
```java
import java.io.FileReader;
public class ReadExample {
    public static void main(String[] args) {
        try {
            FileReader reader = new FileReader("example.txt");
            int character;
            while ((character = reader.read()) != -1) {
                System.out.print((char) character);
            }
            reader.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Note: if The File example.txt is not created by the user in the current working directory.



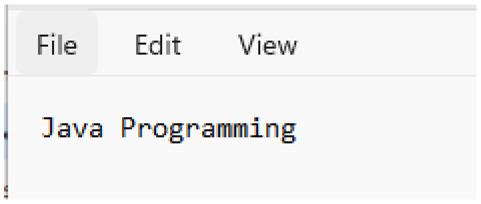
```
D:\Java Code\IO>javac ReadExample.java

D:\Java Code\IO>java ReadExample
java.io.FileNotFoundException: example.txt (The system cannot find the file specifi
ed)
```

When file example.txt is created, and it is blank

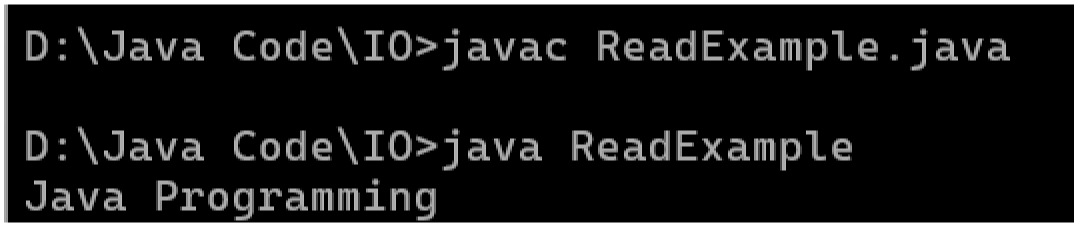


```
D:\Java Code\IO>javac ReadExample.java

D:\Java Code\IO>java ReadExample
```

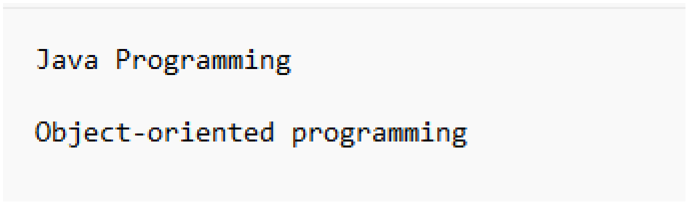
When file example.txt has some data



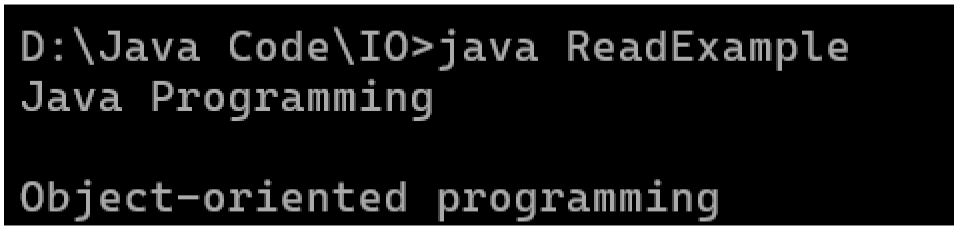
```
Java Programming
```



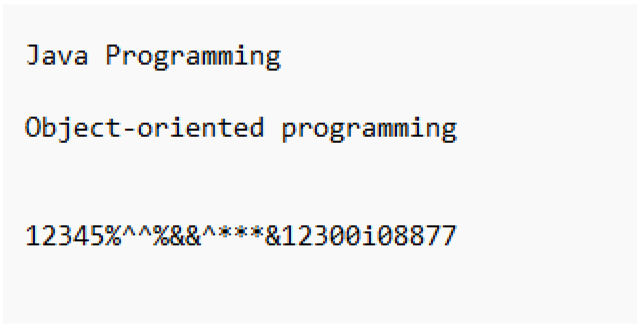
```
D:\Java Code\IO>javac ReadExample.java

D:\Java Code\IO>java ReadExample
Java Programming
```



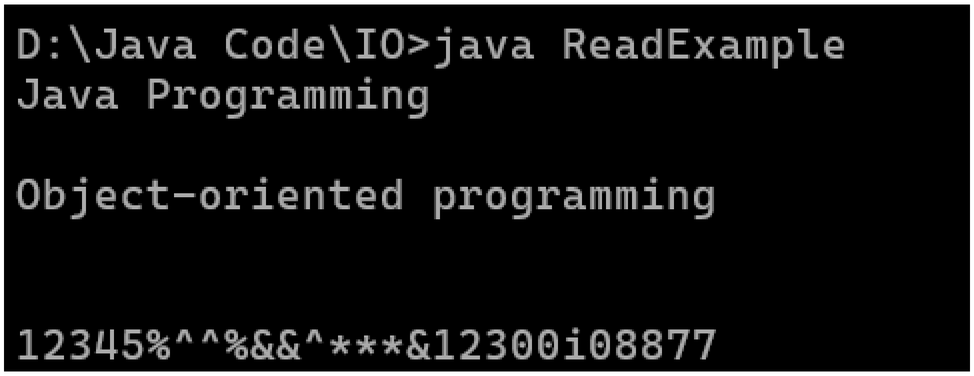
```
Java Programming

Object-oriented programming
```



```
D:\Java Code\IO>java ReadExample
Java Programming

Object-oriented programming
```



```
Java Programming

Object-oriented programming


12345%^^%&&^***&12300i08877
```



```
D:\Java Code\IO>java ReadExample
Java Programming

Object-oriented programming


12345%^^%&&^***&12300i08877
```

Explanation:

```
import java.io.FileReader; // Importing the FileReader class from java.io package
public class ReadExample {
    public static void main(String[] args) {
        try { // Using try-catch to handle exceptions (like file not found)
            // Creating a FileReader object and opening "example.txt" for reading
            FileReader reader = new FileReader("example.txt");
            int character;  // Variable to store the character read (as an integer)
            // Reading characters one by one until end of file (EOF) is reached
            while ((character = reader.read()) != -1) {
                // reader.read() reads one character at a time and returns its ASCII value
                // We cast it to (char) to print the actual character instead of ASCII number
                System.out.print((char) character);
            }
            // Closing the FileReader after reading is complete
            reader.close();
        }
        // Catching any exception that occurs during reading (like file not found)
        catch (Exception e) {
            // Printing the stack trace for debugging if an error happens
            System.out.println(e);
        }
    }
}
```

What is FileWriter?

- FileWriter is a character-based output stream in Java.
- It is used to write data to a file — character by character.
- It is used for creating new files or writing to existing text files.

How to Create a FileWriter?

FileWriter fw = new FileWriter("filename.txt");

- This creates a new file or opens the existing file for writing.
- If the file exists, its content will be overwritten unless you specify append mode.

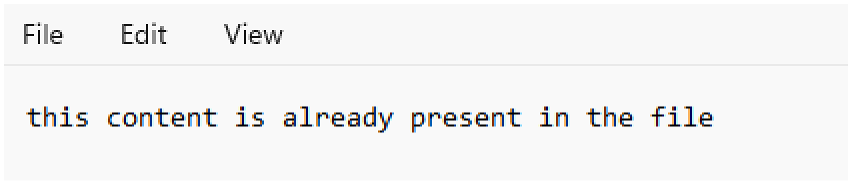
To append data instead of overwriting:

FileWriter fw = new FileWriter("filename.txt", true);

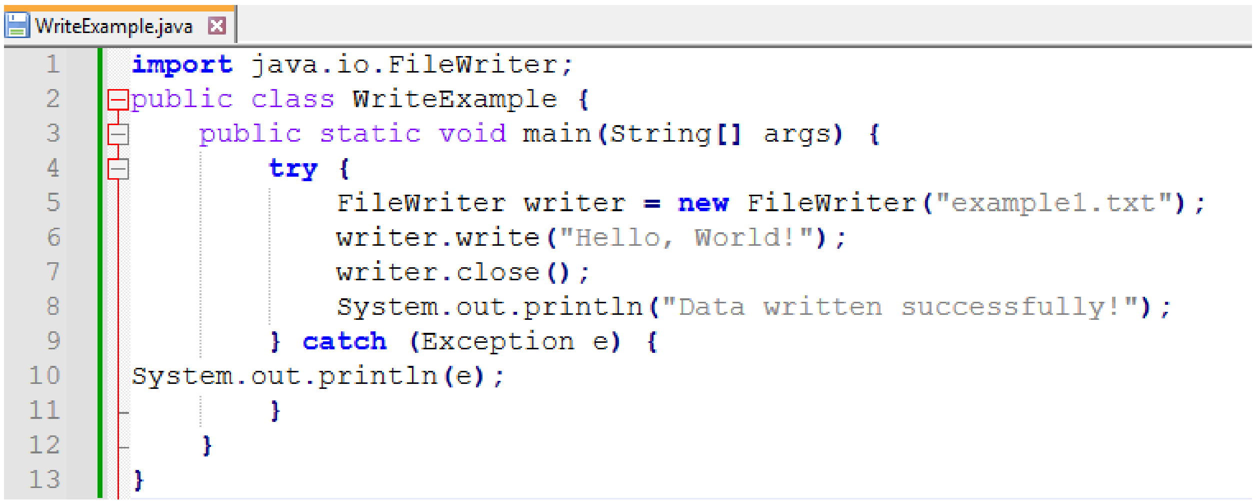
Common Methods of FileWriter

Method	**Description**
write(int c)	Writes a single character.
write(char[] cbuf)	Writes an array of characters.
write(String str)	Writes a string.
close()	Closes the FileWriter.

Example of FileWriter

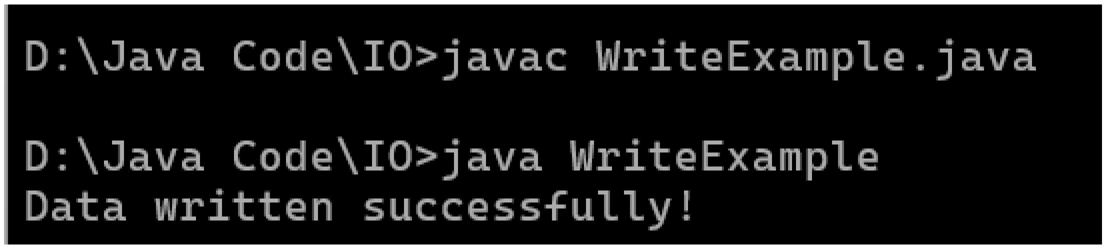


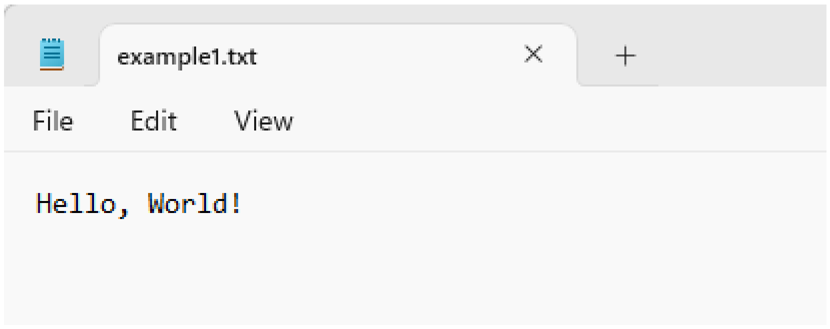
```
this content is already present in the file
```



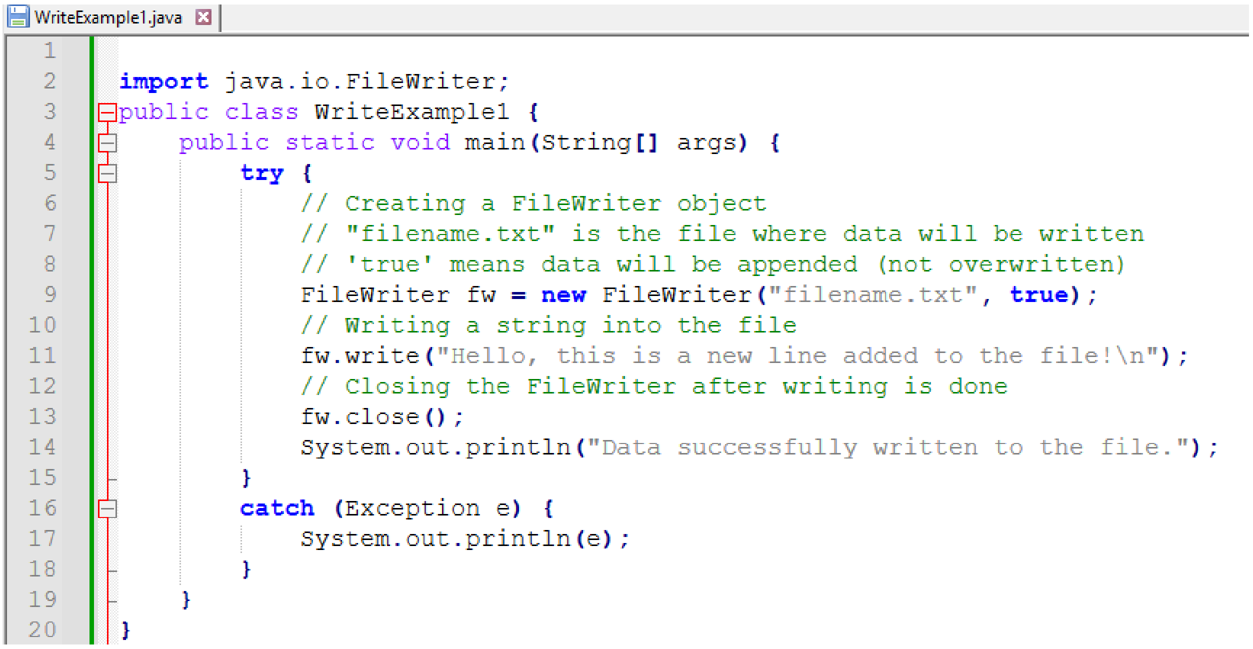
```
import java.io.FileWriter;
public class WriteExample {
    public static void main(String[] args) {
        try {
            FileWriter writer = new FileWriter("example1.txt");
            writer.write("Hello, World!");
            writer.close();
            System.out.println("Data written successfully!");
        } catch (Exception e) {
System.out.println(e);
        }
    }
}
```

Output:

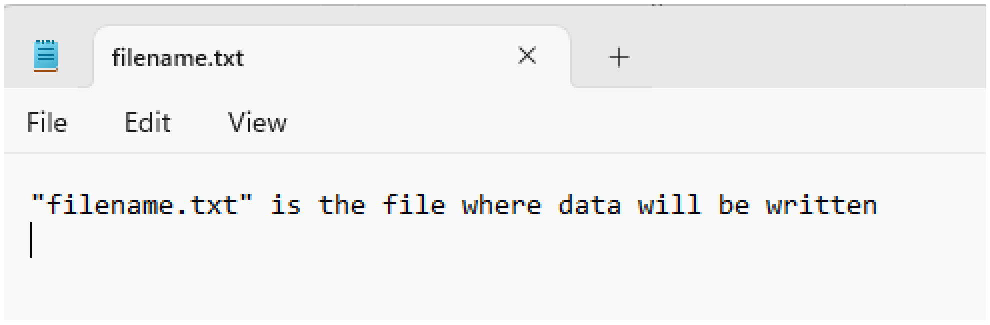

```
D:\Java Code\IO>javac WriteExample.java

D:\Java Code\IO>java WriteExample
Data written successfully!
```



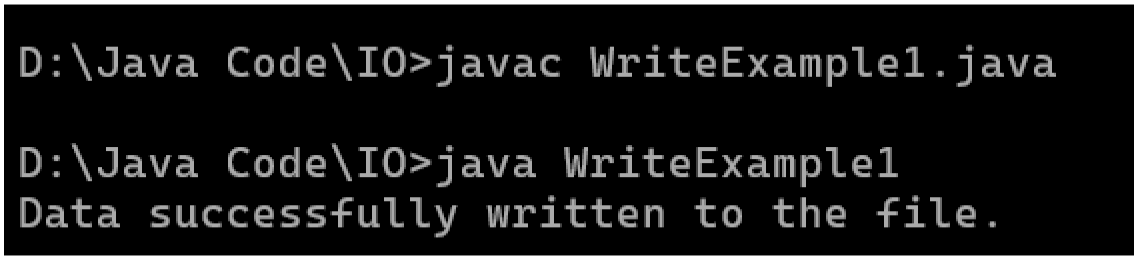
```
Hello, World!
```



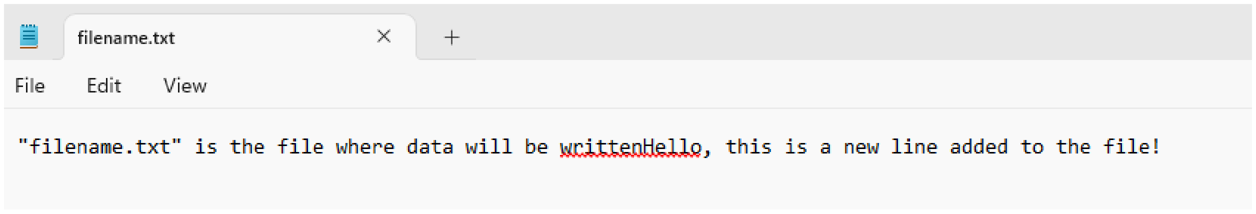
```java

import java.io.FileWriter;
public class WriteExample1 {
    public static void main(String[] args) {
        try {
            // Creating a FileWriter object
            // "filename.txt" is the file where data will be written
            // 'true' means data will be appended (not overwritten)
            FileWriter fw = new FileWriter("filename.txt", true);
            // Writing a string into the file
            fw.write("Hello, this is a new line added to the file!\n");
            // Closing the FileWriter after writing is done
            fw.close();
            System.out.println("Data successfully written to the file.");
        }
        catch (Exception e) {
            System.out.println(e);
        }
    }
}
```



```
"filename.txt" is the file where data will be written
```



```
D:\Java Code\IO>javac WriteExample1.java

D:\Java Code\IO>java WriteExample1
Data successfully written to the file.
```



```
"filename.txt" is the file where data will be writtenHello, this is a new line added to the file!
```

Practice Programs:

1. Write a Java program that reads a file name from the user and attempts to open and read the file. Define a method readFile() that throws a FileNotFoundException using the throws keyword. In the main method, call this method and handle the exception using a try-catch block. Display an appropriate message if the file is not found. Use a finally block to ensure a message like "File operation attempted" is printed.
2. Write a Java program that takes user input for a student's name, roll number, and grade, and writes this information to a file named student.txt using FileWriter. Ensure the program appends the data to the file if it already exists. Handle any exceptions using try-catch and display an appropriate message if an error occurs.

   Sample File Content:

   Name: Aman, Roll Number: 120112, Grade: A

   Name: Parul, Roll Number: 120131, Grade: B
3. Write a Java program that reads the contents of a file named student.txt using FileReader and displays the data on the console. Handle FileNotFoundException if the file does not exist and display an appropriate error message. Use a try-catch block for exception handling.