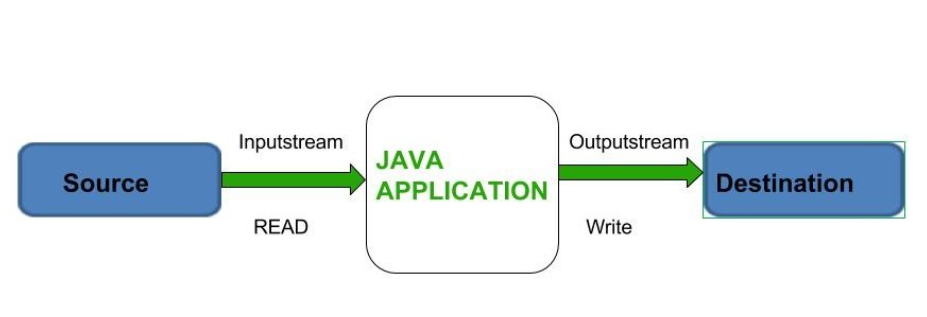
**Unit-5**

**(IO Streams)**

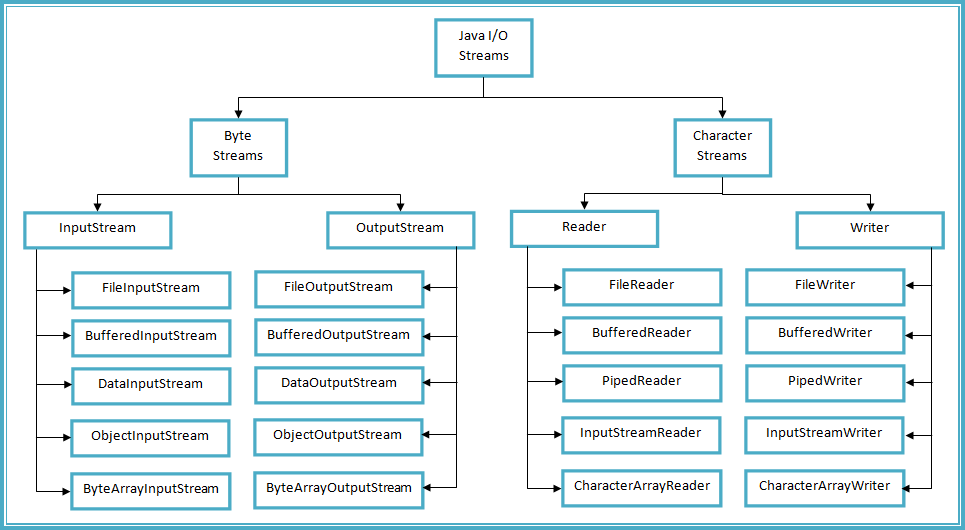
**Stream:**

**Sequence of data** is called **stream**. There are two types of Streams:

* **InPutStream**: The InputStream is used to **read data** from a source.
* **OutPutStream**: The OutputStream is used for **writing data** to a destination.

**Input** and **output** streams in Java provide a way to read from and write to various data sources and destinations, such as files, network connections, and in-memory buffers. The streams are a fundamental part of Java I/O (Input/Output) and are based on the abstract classes **InputStream** and **OutputStream**.

**IO Class Hierarchy**



**ByteStreams:**

Byte streams in Java handle I/O operations at the **byte level**. They are primarily used for reading and writing binary data, such as images, audio files, or any other non-textual data. Byte streams work with the **InputStream** and **OutputStream** classes

**Types of ByteStream:**

**ByteStreams** can be divided into two primary classes:

1. [**Input Stream**:](https://www.geeksforgeeks.org/java-io-inputstream-class-in-java/) 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.
2. [**Output Stream**:](https://www.geeksforgeeks.org/java-io-outputstream-class-java/) These streams are used to **write data** as outputs into an array or file or any output peripheral device. For eg., FileOutputStream, BufferedOutputStream, ByteArrayOutputStream etc.

**InputStream**:

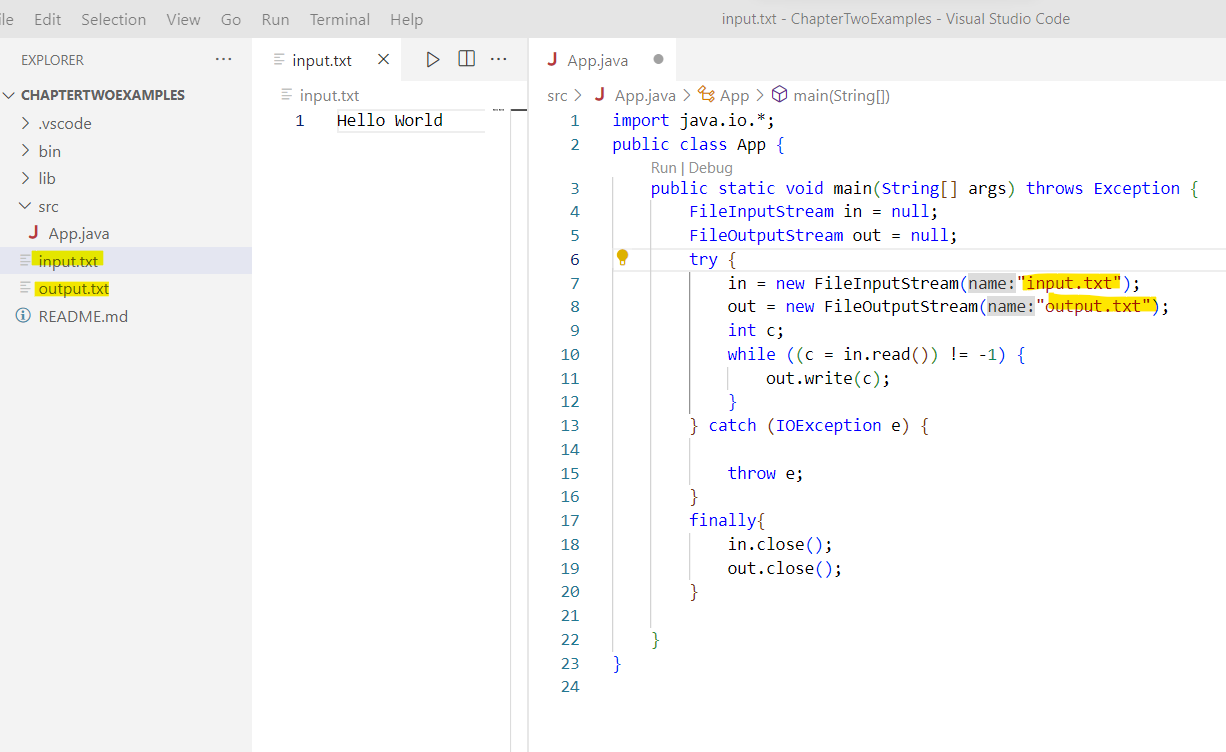
* FileInputStream: Reads data from a file as a sequence of bytes.
* ByteArrayInputStream: Reads data from a byte array.
* BufferedInputStream: Provides buffering capabilities to improve input performance.
* DataInputStream: Reads primitive Java data types from an underlying input stream.
* ObjectInputStream: Reads objects from an underlying input stream, including serialized objects.

**OutputStream**:

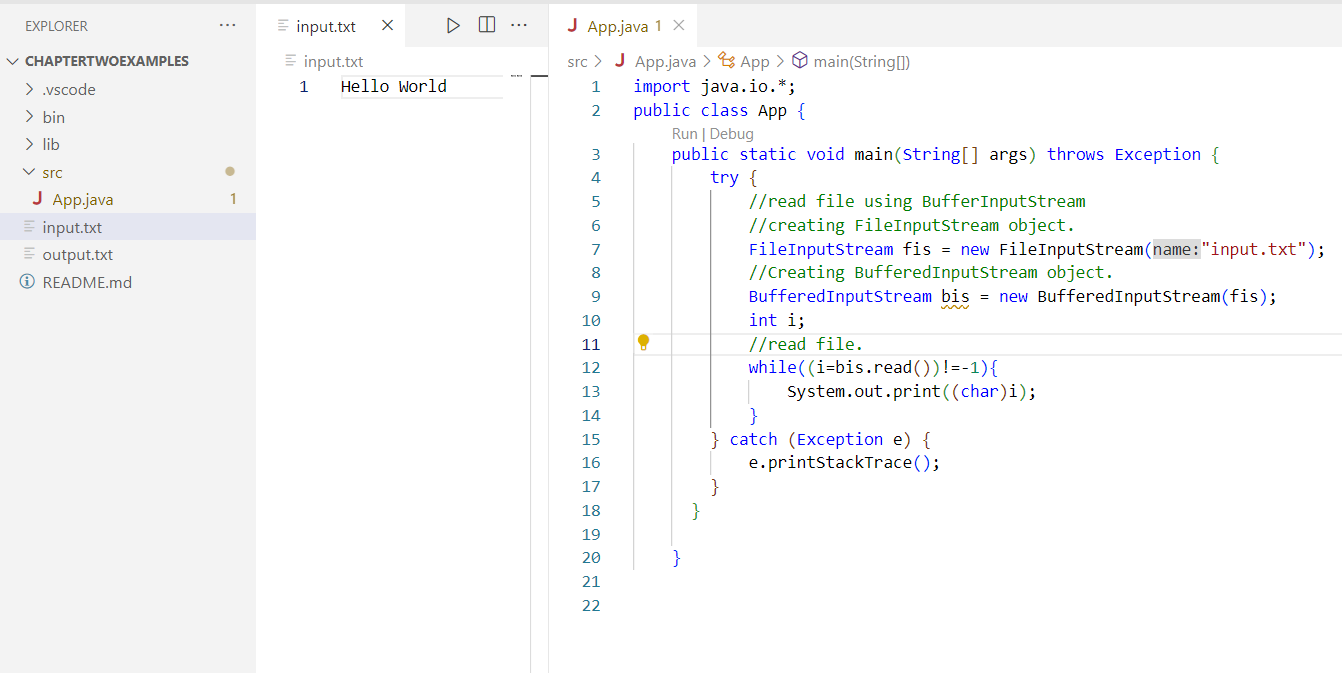
* FileOutputStream: Writes data to a file as a sequence of bytes.
* ByteArrayOutputStream: Writes data to a byte array.
* BufferedOutputStream: Provides buffering capabilities to improve output performance.
* DataOutputStream: Writes primitive Java data types to an underlying output stream.
* ObjectOutputStream: Writes objects to an underlying output stream, including serialized objects.

**Example ByteStream:**

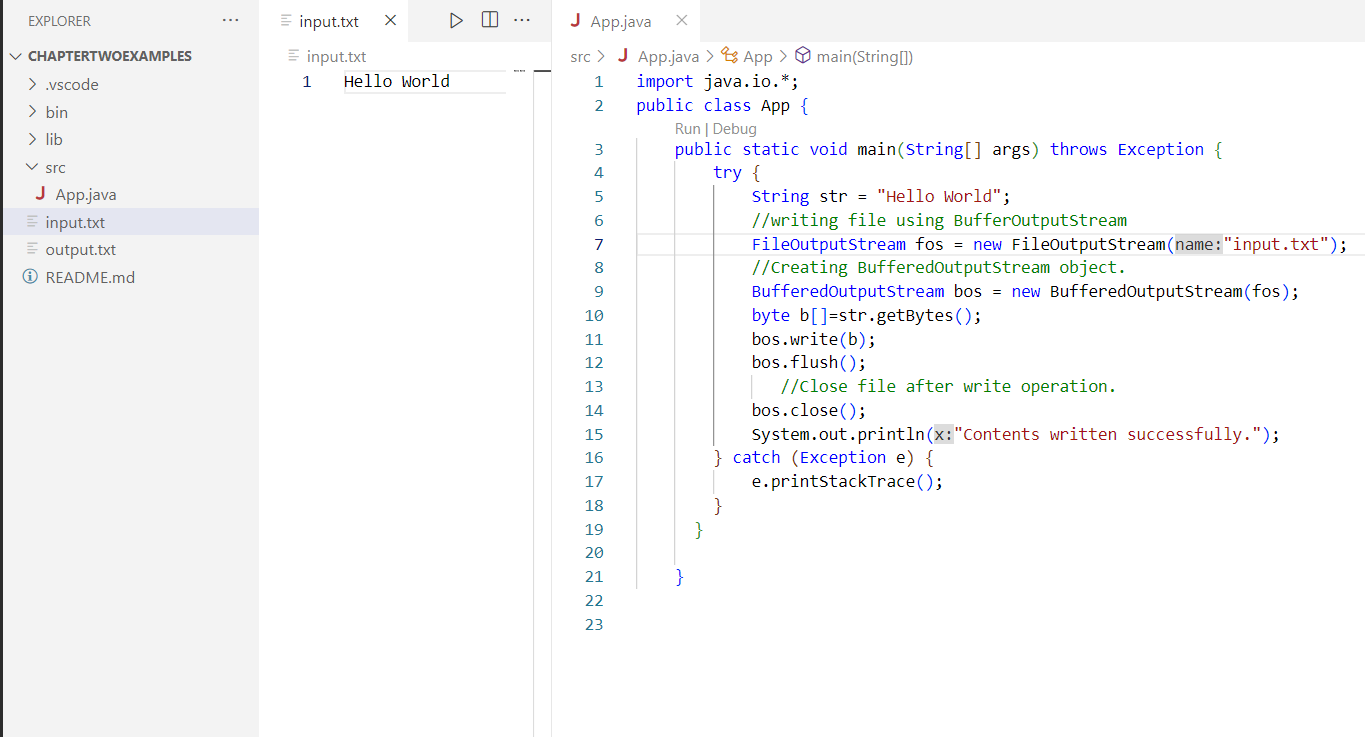
**Lab: Write a Java program to read from file input.txt and write its contents to output.txt.**



**Example BufferInput Stream(Reading File Content)**

****

**Example BufferOutput Stream(Writing to File)**

****

**CharacterStreams:**

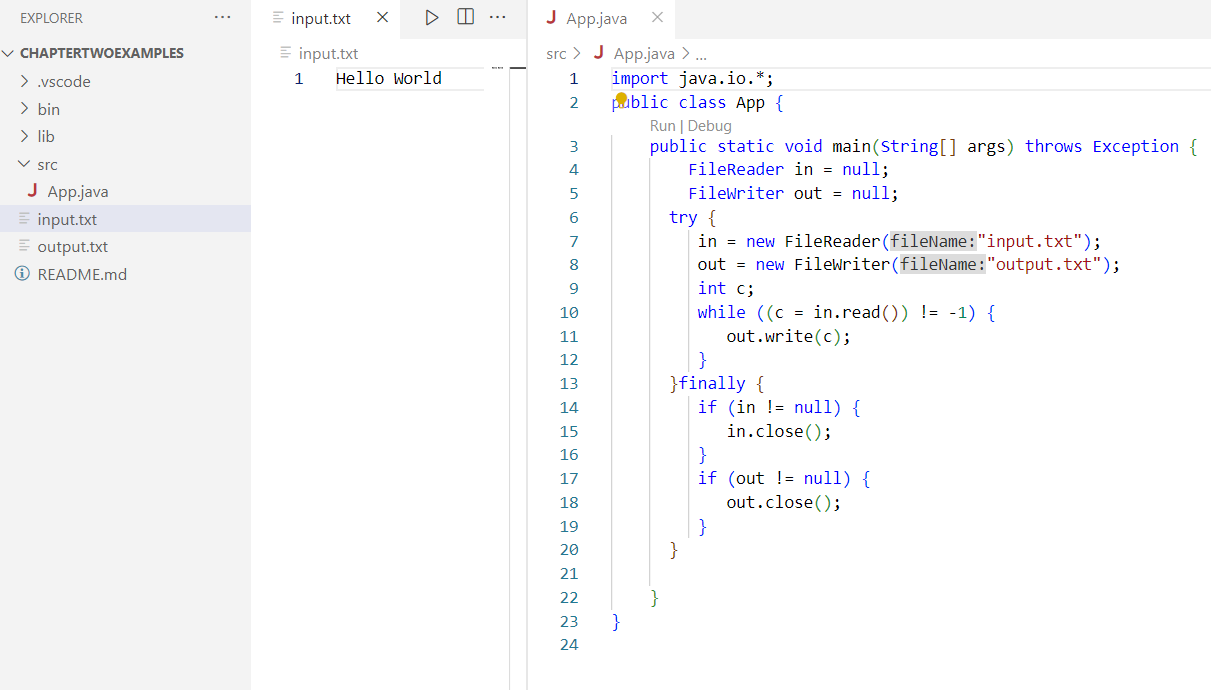
Character streams in Java handle I/O operations at the character level, making them suitable for reading and writing text-based data. They are designed to handle character encoding and decoding automatically, allowing for the reading and writing of characters. Character streams work with the **Reader** and **Writer** classes

**Types of CharacterStreams:**

1. **Reader**: The Reader class is an abstract base class for all character input streams. It defines methods for reading characters from a source and provides a common interface for working with character-based input streams. Some commonly used subclasses of Reader include **FileReader, BufferedReader, StringReader, and InputStreamReader**.
2. **Writer**: The Writer class is an abstract base class for all character output streams. It defines methods for writing characters to a destination and provides a common interface for working with character-based output streams. Some commonly used subclasses of Writer include **FileWriter, BufferedWriter, StringWriter, and OutputStreamWriter.**

**Character StreamsExample:**

**Lab: Write a Java program to read from file input.txt and write its contents to output.txt using FileReader and FileWriter(CharacterStreams).**



**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** **Class**

| **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. |

**File and Directories:**

**List all file from directories**

import java.io.\*;

public class App {

    public static void main(String[] args) throws Exception {

        String directoryPath = "E:\\OOP in Java";

        File directory = new File(directoryPath);

        File[] files = directory.listFiles();

                for (File file : files) {

                        System.out.println(file.getName());

                }

    }

}

**Create New File:**

import java.io.\*;

public class App {

    public static void main(String[] args) throws Exception {

        File file = new File("newfile.txt");

        try

        {

           file.createNewFile();

        } catch (IOException e)

        {

            System.out.println("An error occurred: ");

        }

}

}

**Copy File:**

import java.io.\*;

import java.nio.file.\*;

public class App {

    public static void main(String[] args) throws Exception {

       String sourceFilePath = "E://OOP in Java//Threads.docx";

        String destinationFilePath = "E://OOP in Java//Sample Questions//Threads.docx";

        Path sourcePath = Paths.get(sourceFilePath);

        Path destinationPath = Paths.get(destinationFilePath);

        try {

            Files.copy(sourcePath, destinationPath);

            System.out.println("File copied successfully!");

        } catch (IOException e) {

            System.out.println("An error occurred: " + e.getMessage());

        }

}

}

**Delete File:**

import java.io.\*;

import java.nio.file.\*;

public class App {

    public static void main(String[] args) throws Exception {

       String filePath = "E://OOP in Java//Threads.docx";

       Path path = Paths.get(filePath);

       try {

           Files.delete(path);

           System.out.println("File deleted successfully!");

       } catch (IOException e) {

           System.out.println("An error occurred: " + e.getMessage());

       }

}

}

**Object Serialization:**

Object serialization is the process of converting an **object into a byte stream**, which can be saved to a file, sent over a network, or stored in a database. It is mainly used in Hibernate, RMI (Remote Method Invocation) technologies.

The reverse operation of serialization is called *deserialization* **where byte-stream is converted into an object**. The serialization and deserialization process is **platform-independent**, it means you can serialize an object on one platform and deserialize it on a different platform.

For serializing the object, we call the **writeObject()** method of *ObjectOutputStream*class, and for deserialization we call the **readObject()** method of *ObjectInputStream* class.

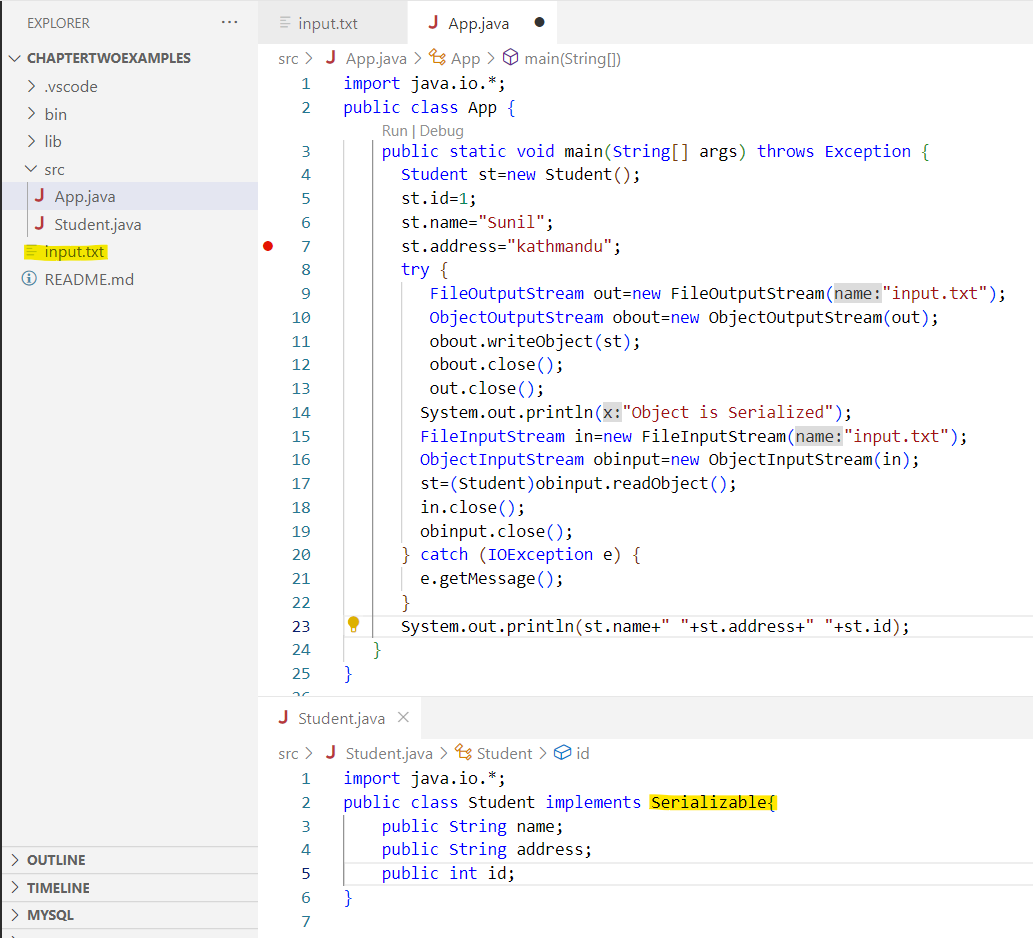
**Syntax for the writeObject() method:**

public final void writeObject(Object o) throws IO Exception

**Syntax for the readObject() method:**

public final Object readObject() throws IOException, ClassNotFoundException

Lab: Write a java program to demonstrate Object serialization and *deserialization*.



**RandomAccessFile**

The **RandomAccessFile** class is used to perform random access to a file. Unlike other file handling classes like **FileInputStream** or **FileOutputStream**, which provide **sequential access** to a file, **RandomAccessFile** allows you to read from or write to any position within the file.

To work with a file using **RandomAccessFile**, you first need to create an instance of the class. You can do this by providing the file path and the mode (read-only, write-only, or both) as parameters to the constructor.

There are two modes: **"r" for reading** and **"rw" for reading and writing.**

**Reading from a** RandomAccessFile**:**

  RandomAccessFile raf = new RandomAccessFile("example.txt", "r");

  byte[] buffer = new byte[1024];

  int bytesRead = raf.read(buffer);

**Writing To a** RandomAccessFile:

RandomAccessFile raf = new RandomAccessFile("example.txt", "rw");

byte[] data = "Hello, World!".getBytes();

raf.write(data);

**Moving** Pointer:

raf.seek(50);

**Closing** RandomAccessFile:

raf.close();

Lab: Write a java program to read, read write using RandomAccessFile

Example:

import java.io.\*;

public class App {

    public static void main(String[] args)

    {

        String FILEPATH ="input.txt";

        try {

            System.out.println(new String(readFromFile(FILEPATH, 0, 18)));

            writeToFile(FILEPATH, "Stay positive. Better days are on their way.", 31);

            System.out.println(new String(readFromFile(FILEPATH, 0,100)));

        } 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();

    }

}