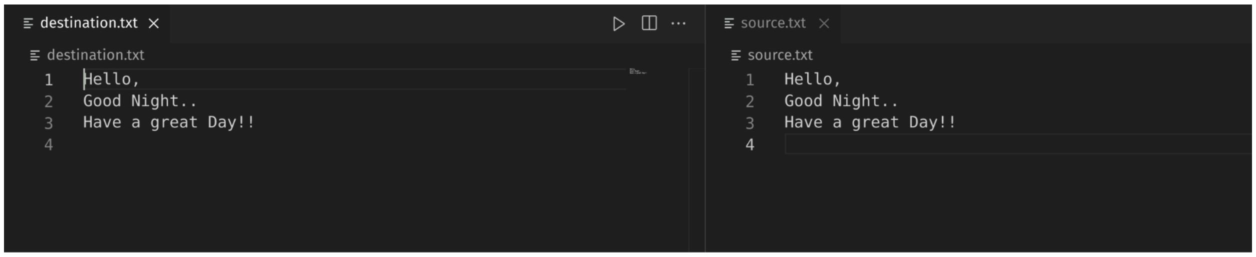
Assignment- 10
22610011 - Anuja Suntnur

1. Write a program to copy contents of one file to another file.
Ans:

```
import java.io.*;

public class FileCopyExample {
   public static void main(String[] args) {
       String sourceFilePath = "source.txt"; // Path to the source file
       String destinationFilePath = "destination.txt"; // Path to the
destination file

       try (
           FileReader fileReader = new FileReader(sourceFilePath);
           FileWriter fileWriter = new FileWriter(destinationFilePath);
           BufferedReader bufferedReader = new BufferedReader(fileReader);
           BufferedWriter bufferedWriter = new BufferedWriter(fileWriter)
       ) {
           String line;
           while ((line = bufferedReader.readLine()) != null) {
               bufferedWriter.write(line);
               bufferedWriter.newLine(); // Add new line after each line
copied
           }
           System.out.println("File copied successfully!");
       } catch (IOException e) {
           System.out.println("An error occurred while copying the file: "
+ e.getMessage());
       }
   }
}
```

Output :


destination.txt

```
Hello,
Good Night..
Have a great Day!!
```

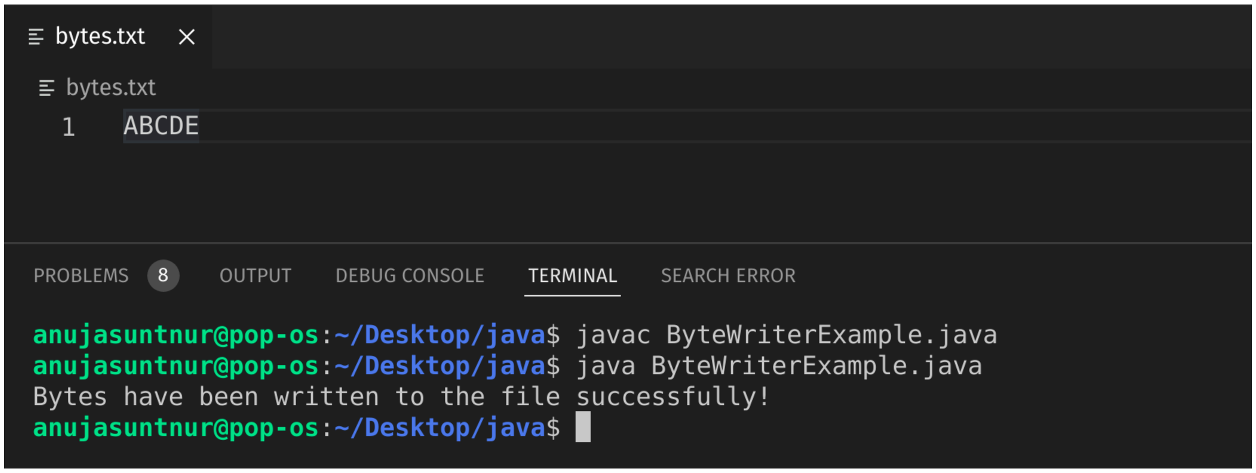
source.txt

```
Hello,
Good Night..
Have a great Day!!
```

2. Write a program to write bytes to a file.

```java
import java.io.*;

public class ByteWriterExample {
   public static void main(String[] args) {
       String filePath = "bytes.txt"; // Path to the file

       try (FileOutputStream outputStream = new
FileOutputStream(filePath)) {
           // Bytes to write to the file
           byte[] bytesToWrite = {65, 66, 67, 68, 69}; // Corresponding to
ASCII values of 'A', 'B', 'C', 'D', 'E'

           // Writing bytes to the file
           outputStream.write(bytesToWrite);

           System.out.println("Bytes have been written to the file
successfully!");
       } catch (IOException e) {
           System.out.println("An error occurred while writing bytes to
the file: " + e.getMessage());
       }
   }
}
```

Output:


bytes.txt

```
1   ABCDE
```

PROBLEMS 8   OUTPUT   DEBUG CONSOLE   TERMINAL   SEARCH ERROR

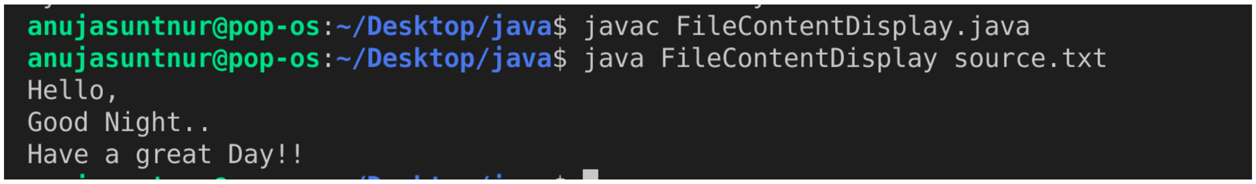
```
anujasuntnur@pop-os:~/Desktop/java$ javac ByteWriterExample.java
anujasuntnur@pop-os:~/Desktop/java$ java ByteWriterExample.java
Bytes have been written to the file successfully!
anujasuntnur@pop-os:~/Desktop/java$
```

3. Develop a program to display contents of file supplied as command line argument.
Ans:

```
import java.io.*;

public class FileContentDisplay {
   public static void main(String[] args) {
       // Check if a file path is provided as command-line argument
       if (args.length == 0) {
           System.out.println("Usage: java FileContentDisplay
<file_path>");
           return;
       }

       String filePath = args[0]; // Get the file path from command-line
arguments

       try (BufferedReader reader = new BufferedReader(new
FileReader(filePath))) {
           String line;
           // Read and display each line from the file
           while ((line = reader.readLine()) != null) {
               System.out.println(line);
           }
       } catch (IOException e) {
           System.out.println("An error occurred while reading the file: "
+ e.getMessage());
```

```
        }
    }
}
```

Output:



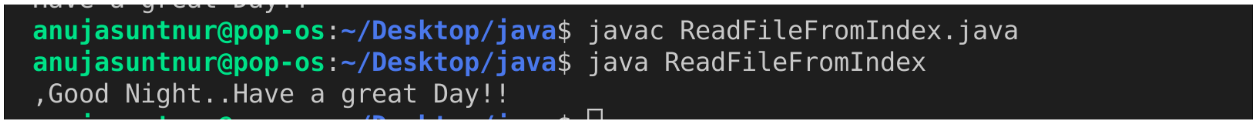
```
anujasuntnur@pop-os:~/Desktop/java$ javac FileContentDisplay.java
anujasuntnur@pop-os:~/Desktop/java$ java FileContentDisplay source.txt
Hello,
Good Night..
Have a great Day!!
```

4. Write Java program to read text from file from a specified index or skipping byte using FileInputStream.

Ans:

```java
import java.io.FileInputStream;
import java.io.IOException;

public class ReadFileFromIndex {
    public static void main(String[] args) {
        String filename = "source.txt"; // Replace with your file name
        int startingIndex = 5; // Starting index to read from

        try (FileInputStream fis = new FileInputStream(filename)) {
            // Skip bytes to reach the starting index
            fis.skip(startingIndex);

            // Read and display the remaining content
            int byteData;
            while ((byteData = fis.read()) != -1) {
                System.out.print((char) byteData);
            }
        } catch (IOException e) {
            System.err.println("Error reading file: " + e.getMessage());
        }
    }
}
```

Output:

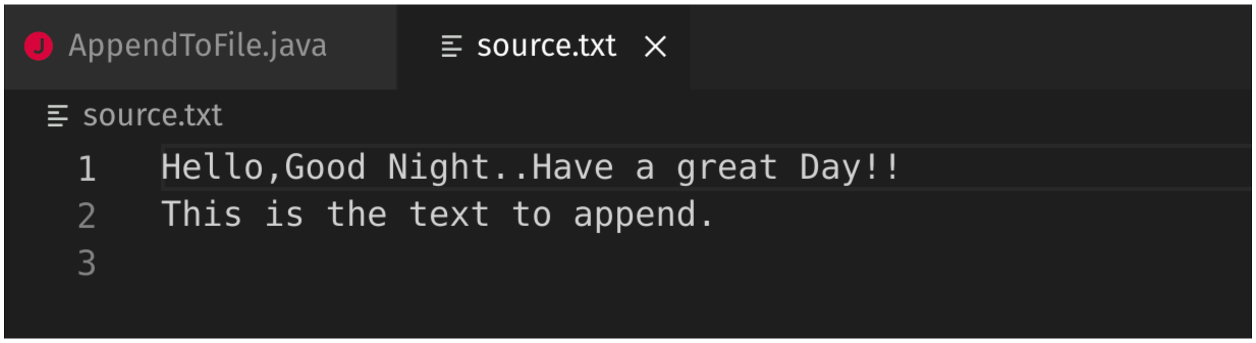


```
anujasuntnur@pop-os:~/Desktop/java$ javac ReadFileFromIndex.java
anujasuntnur@pop-os:~/Desktop/java$ java ReadFileFromIndex
,Good Night..Have a great Day!!
```

5. Write Java program to append text/string in a file.
Ans:

```java
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class AppendToFile {
   public static void main(String[] args) {
       String filename = "source.txt"; // Specify your file name here
       String textToAppend = "This is the text to append.";

       try (BufferedWriter writer = new BufferedWriter(new
FileWriter(filename, true))) {
           writer.write(textToAppend);
           writer.newLine(); // Add a new line after appending text
           System.out.println("Text appended successfully.");
       } catch (IOException e) {
           System.err.println("Error appending text to file: " +
e.getMessage());
       }
   }
}
```

Output:

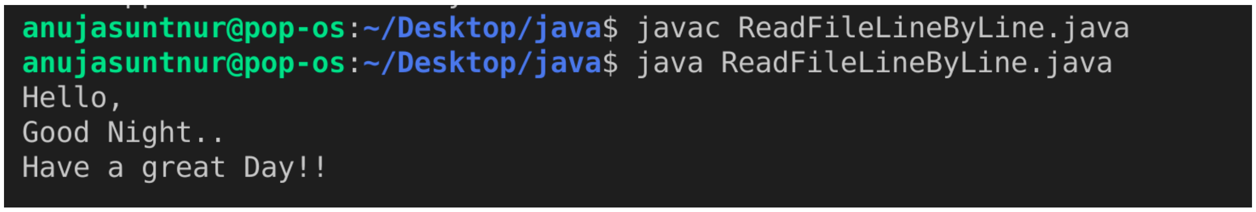


```
Hello,Good Night..Have a great Day!!
This is the text to append.

```

6. Write Java program to read a file line by line.
Ans:

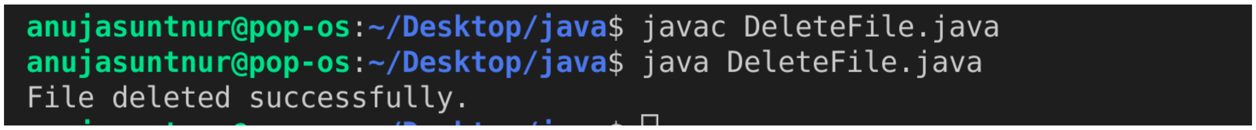
```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadFileLineByLine {
   public static void main(String[] args) {
       String filename = "source.txt"; // Specify your file name here

       try (BufferedReader br = new BufferedReader(new
FileReader(filename))) {
           String line;
           while ((line = br.readLine()) != null) {
               System.out.println(line);
           }
       } catch (IOException e) {
           System.err.println("Error reading file: " + e.getMessage());
       }
   }
}
```

Output:



```
anujasuntnur@pop-os:~/Desktop/java$ javac ReadFileLineByLine.java
anujasuntnur@pop-os:~/Desktop/java$ java ReadFileLineByLine.java
Hello,
Good Night..
Have a great Day!!
```

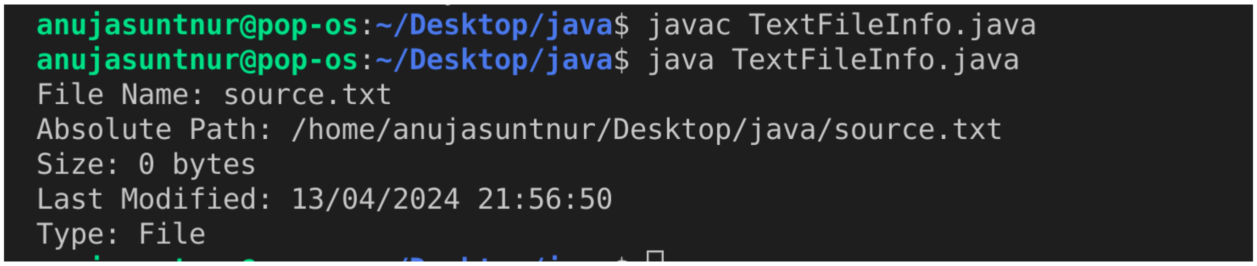
7. Write Java program to delete a file.
Ans:
```java
import java.io.File;

public class DeleteFile {
   public static void main(String[] args) {
       String filename = "source.txt"; // Specify the file name to delete

       File fileToDelete = new File(filename);

       if (fileToDelete.delete()) {
           System.out.println("File deleted successfully.");
       } else {
           System.err.println("Failed to delete the file.");
       }
   }
}
```

Output:

```
anujasuntnur@pop-os:~/Desktop/java$ javac DeleteFile.java
anujasuntnur@pop-os:~/Desktop/java$ java DeleteFile.java
File deleted successfully.
```

8. Write Java Program to display text file information.
Ans:
```java
import java.io.File;
import java.text.SimpleDateFormat;

public class TextFileInfo {
   public static void main(String[] args) {
       String filename = "source.txt"; // Specify your file name here

       File file = new File(filename);

```

```java
        if (file.exists()) {
            System.out.println("File Name: " + file.getName());
            System.out.println("Absolute Path: " + file.getAbsolutePath());
            System.out.println("Size: " + file.length() + " bytes");
            System.out.println("Last Modified: " + new
SimpleDateFormat("dd/MM/yyyy HH:mm:ss").format(file.lastModified()));

            if (file.isDirectory()) {
                System.out.println("Type: Directory");
            } else {
                System.out.println("Type: File");
            }
        } else {
            System.out.println("File does not exist.");
        }
    }
}
```

Output:



```
anujasuntnur@pop-os:~/Desktop/java$ javac TextFileInfo.java
anujasuntnur@pop-os:~/Desktop/java$ java TextFileInfo.java
File Name: source.txt
Absolute Path: /home/anujasuntnur/Desktop/java/source.txt
Size: 0 bytes
Last Modified: 13/04/2024 21:56:50
Type: File
```

9. Explain java stream classes.
Ans:

Java provides two main types of streams: byte streams and character streams.

1. Byte Streams: Byte streams are used for handling raw binary data. They work with bytes, making them suitable for reading and writing any type of data, including text and binary data. The basic classes for byte streams are `InputStream` and `OutputStream`. Examples of byte stream classes include `FileInputStream`,

`FileOutputStream`, `ByteArrayInputStream`, and `ByteArrayOutputStream`.

2. Character Streams: Character streams are used specifically for handling character data. They handle data in terms of characters, making them more suitable and efficient for working with text data compared to byte streams. Java uses Unicode for character encoding, allowing character streams to handle characters from different languages and character sets. The basic classes for character streams are `Reader` and `Writer`. Examples of character stream classes include `FileReader`, `FileWriter`, `BufferedReader`, and `BufferedWriter`.

Java stream classes provide various methods for reading from or writing to the underlying data source efficiently. They often include methods for reading or writing data in bulk, buffering data for improved performance, and handling various exceptions that may occur during I/O operations.

10. Explain file operations.
Ans:

1. **Creating Files**: This operation involves creating a new file in the file system. The file can be empty or initialized with some initial data.
2. **Reading Files**: Reading files involves accessing the contents of a file stored in the file system. It can be done sequentially, line by line, or by reading the entire file at once.
3. **Writing to Files**: Writing to files involves adding or appending data to an existing file or creating a new file and writing data into it.
4. **Updating Files**: Updating files involves modifying the contents of an existing file. It can include appending new data, overwriting existing data, or modifying specific parts of the file.
5. **Deleting Files**: Deleting files involves removing files from the file system permanently.

6. **Moving and Renaming Files**: Moving files involves transferring files from one location to another within the file system. Renaming files involves changing the name of a file without changing its contents.
7. **File Metadata Operations**: File metadata operations involve retrieving information about files, such as file size, creation date, last modified date, permissions, and file type.
8. **File Searching and Traversal**: File searching and traversal operations involve searching for specific files or directories within a directory structure and traversing through directories to access files.
9. **File Locking**: File locking operations involve preventing other processes or threads from accessing or modifying a file while it is being read from or written to by another process or thread.
10. **Error Handling:** Error handling in file operations involves handling various exceptions that may occur during file operations, such as file not found, permission denied, or disk full errors.