Unit 5: Using I/O in Java

5.1 Console and File I/O

Java supports input and output through console and file. Console I/O uses classes like `System.in` and `System.out`, while File I/O uses classes from `java.io` package like `FileReader`, `FileWriter`, etc.

Console I/O Example:

```
import java.util.Scanner;
public class ConsoleIOExample {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = sc.nextLine();
        System.out.println("Hello, " + name);
    }
}
```

File I/O Example:

```
import java.io.*;
public class FileIOExample {
    public static void main(String[] args) throws IOException {
        FileWriter writer = new FileWriter("output.txt");
        writer.write("Hello File");
        writer.close();
    }
}
```

5.2 Opening and Closing Files

To open a file, we create a stream object. Closing is done using the `close()` method.

Example:

```
FileWriter writer = new FileWriter("data.txt");
writer.write("Some data");
writer.close();
```

5.3 Scanner Class

The `Scanner` class is used to read input from various sources like console, files, etc.
Example: Reading from Console

```
Scanner sc = new Scanner(System.in);
int age = sc.nextInt();
```

Example: Reading from File

```
import java.io.*;
import java.util.*;
File file = new File("data.txt");
Scanner sc = new Scanner(file);
while(sc.hasNext()) {
    System.out.println(sc.nextLine());
}
```

5.4 Byte Streams and Character Streams

Byte Streams: Handle raw binary data(images, audio,video) using `InputStream` and `OutputStream`. Character Streams: Handle character data (Text) using `Reader` and `Writer`.

Byte Stream Example:

```
FileInputStream fin = new FileInputStream("data.bin");
int i;
while((i = fin.read()) != -1) {
    System.out.print((char)i);
}
fin.close();
```

Character Stream Example:

```
FileReader fr = new FileReader("data.txt");
int i;
while((i = fr.read()) != -1) {
    System.out.print((char)i);
}
fr.close();
```

5.5 Reading and Writing Byte Streams

Writing Byte Stream:

```
FileOutputStream fout = new FileOutputStream("bytefile.bin");
String s = "Binary data";
fout.write(s.getBytes());
fout.close();
```

Reading Byte Stream:

```
FileInputStream fin = new FileInputStream("bytefile.bin");
int i;
while((i = fin.read()) != -1) {
    System.out.print((char)i);
}
fin.close();
```

5.6 Reading and Writing Character Streams

Writing Character Stream:

```
FileWriter fw = new FileWriter("charfile.txt");
fw.write("Character data");
fw.close();
```

Writing an Image to a File using Byte Stream

```
import java.io.*;
public class ImageWriter {
    public static void main(String[] args) throws IOException {
        // Step 1: Create FileInputStream to read the input image
        // This stream reads the image file "input.jpg" in binary format.
        FileInputStream fis = new FileInputStream("input.jpg");

        // Step 2: Create FileOutputStream to write the image to a new file
        // This stream writes the image data to "output.jpg".
        FileOutputStream fos = new FileOutputStream("output.jpg");

        // Step 3: Variable to store each byte read from the input file
        int b;

        // Step 4: Read the image file byte by byte and write it to the
output file
        // The loop continues until the end of the input file is reached
(when fis.read() returns -1).
        while ((b = fis.read()) != -1) {
            fos.write(b); // Write each byte to the output file
        }

        // Step 5: Close the input and output file streams
        // It's important to close the streams to release system resources.
        fis.close();
        fos.close();

        // Step 6: Inform the user that the process was successful
        System.out.println("Image successfully copied from input.jpg to
output.jpg!");
    }
}
```

Reading Character Stream:

```
FileReader fr = new FileReader("charfile.txt");
int i;
while((i = fr.read()) != -1) {
    System.out.print((char)i);
}
fr.close();
```

5.7 Random Access Files: `RandomAccessFile` allows you to read and write at any position in the file.

Syntax:

```
RandomAccessFile file = new RandomAccessFile("random.txt", "rw");
```

Modes: "r" for read-only, "rw" for read and write.

- **`RandomAccessFile`**:
  A special Java class that lets you **read and write to a file at any position**, not just from beginning to end (like a normal stream). You can "randomly access" parts of the file.
- **`file`**:
  The reference (or object name) used to access the methods of the `RandomAccessFile`.
- **`new RandomAccessFile(...)`**:
  This creates a new `RandomAccessFile` object and opens the file `random.txt`.
- **`"random.txt"`**:
  This is the name of the file to open or create.
- **`"rw"`**:
  The mode for file access. `"rw"` means:
  - `"r"` → Read only.
  - `"rw"` → Read and write access.

What can you do with it?

- `writeUTF("Hello")`: Write data to the file.
- `seek(0)`: Jump to a specific position (byte) in the file.
- `readUTF()`: Read data from the file.

Example:

```
RandomAccessFile raf = new RandomAccessFile("random.txt", "rw");
raf.writeUTF("Hello");
raf.seek(0);
System.out.println(raf.readUTF());
raf.close();
```