Java IO

Every now and then a Java Program needs to connect with an external source to read data from and write data to.

The Java IO library is designed to facilitate this reading and writing of data.

Java Streams

- Java uses the concept of streams to read and write data from and to a data source.
- A stream can be thought of as a sequence of data.
- In Java, a stream is composed of bytes.
- We can either read from a stream or write to a stream.
- A stream is connected to a data source or a data destination.

Please refer figures:
- 01_src_target.png
- 02_streams.jpg

Types of Streams

Streams in Java IO can be of following 2 types:
1. Byte based Stream (reading and writing bytes)
2. Character based Stream (reading and writing characters)

For classes in Java IO, refer following figures:
- 04_parent_classes_io.png
- 05_byte_classes.png
- 06_char_classes.png

There are different types of IO based on the type of source and destination.

Console IO

Console Input/Output (IO) in Java involves using the console (or terminal) to interact with the user by taking input from them and displaying output to them.

Default Java Streams

- To work with console IO, there are 3 default streams to work with.
- These streams are attached with the console.
    1. System.out: standard output stream
    2. System.in: standard input stream
    3. System.err: standard error stream

1. Basic Console Output

Java provides the System.out object for console output. The most common methods are:

- System.out.print(): Prints text to the console without adding a newline at the end.
- System.out.println(): Prints text to the console and adds a newline at the end.
- System.out.printf(): Formats the output string using format specifiers (like %d, %s, etc.).
- System.err.print(): Used for printing error messages or warnings useful when debugging or logging.

Example:

```
System.out.print("Hello, "); // prints: Hello,
System.out.println("World!"); // prints: World! (on the same line)
System.out.printf("Num: %d, String: %s", 10, "Java"); // prints: Number:
10, String: Java
System.err.println("This is an error message.");
```

2. Basic Console Input

To take input from the console, Java provides several ways, but the most common is using the Scanner class.

# Steps to use Scanner:
1. Import the Scanner class:

```
   import java.util.Scanner;
```

2. Create an instance of Scanner:

```
   Scanner scanner = new Scanner(System.in);
```

3. Use appropriate methods to read different data types:
   - nextInt() - Reads an integer
   - nextDouble() - Reads a double
   - next() - Reads a single word (up to a space)
   - nextLine() - Reads an entire line
   - nextBoolean() - Reads a boolean value

Example:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter your name: ");
        String name = scanner.nextLine();  // Reads a line of text

        System.out.print("Enter your age: ");
        int age = scanner.nextInt();  // Reads an integer

        System.out.println("Hello, " + name + "! You are " + age + "
years old.");
```

```
        scanner.close();  // Closing the scanner
    }
}
```

3. Handling Input Errors

- When using Scanner, be cautious of mismatched input types.
- For instance, if you expect an integer but the user inputs a string, an InputMismatchException will occur.

Example of Handling Exceptions:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter an integer: ");
        try {
            int number = scanner.nextInt();
            System.out.println("You entered: " + number);
        } catch (java.util.InputMismatchException e) {
            System.out.println("Invalid input. Please enter an 
integer.");
        }

        scanner.close();
    }
}
```

4. Closing the Scanner

Always close the Scanner object after use with scanner.close() to free up system resources.
-----X-----X-----X-----X-----
File IO
=======
Read Write Characters
=====================
- Following are frequently used classes to deal with reading/writing chracter data from/to files:
1.   FileReader
2.   FileWriter

FileReader

- FileReader can read all characters from a file.
- It reads characters one by one until the end of the file is reached.
- However, FileReader does not provide a built-in method to read the entire file in one go directly
- We typically use a loop to read all characters sequentially with read().

- read() reads a single character at a time and returns its integer representation (Unicode value).
- When the end of the file is reached, read() returns -1.
- So, the idea is to read 1 character at a time and loop until End of File (until read() returns -1)

FileWriter

- Used to write data to a file as characters
- Can append to an existing file or overwrite it, depending on how it is instantiated.
- Uses the write(String str) to write a String in the file.
- If the file is not located at the path, will create an empty one.

Read/Write Binary Data
======================
- Following are frequently used classes to deal with reading/writing binary data (like images, videos, and other non-text files) from/to files:
1. FileInputStream
2. FileOutputStream

1. FileInputStream
- Purpose: Reads data from a file in the form of bytes.
- Common Methods:
  - `read()`: Reads a single byte or an array of bytes.
  - `close()`: Closes the input stream.

2. FileOutputStream
- Purpose: Writes data to a file in the form of bytes.
- Common Methods:
  - `write(int b)`: Writes a single byte.
  - `write(byte[] b)`: Writes an array of bytes.
  - `close()`: Closes the output stream.

Binary File v/s Text File

- A binary file is any file that contains data in a format that isn't plain text.
- Examples include images, videos, music, or even software files.
- Instead of human-readable text, binary files are made up of bytes that represent different kinds of data.

- On the other hand A text file is readable in a text editor, while a binary file is not.
- Binary files store data in a format that needs specific software to interpret it, like an image viewer for .png files.

PDF, word, excel files

- These files contain data encoded in a format that is not plain text and require specific software to interpret the data.
- These file cannot be read by plain Text Editors

```
- These files require specialized software to interpret and display their
contents correctly.

- We need special Java libraries to read/write data from/to these files
such as:
      - Apache PDFBox
      - Apache POI
      - Apache Commons
      - etc
-----X-----X-----X-----X-----
```