**FILE STREAMS**

**byte streams** (for handling binary data)

**character streams** (for handling text data).

**Byte Streams**

Byte streams are used to handle raw binary data, such as images, audio files, and other non-textual data.

1. **FileInputStream**:
   * Reads raw data from a file.
   * Example: Reading bytes from an image file.
2. **FileOutputStream**:
   * Writes raw data to a file.
   * Example: Writing bytes to save an audio file.

**Character Streams**

Character streams are used to handle textual data efficiently by reading and writing characters.

1. **FileReader**:
   * Reads characters from a text file.
   * Example: Reading the contents of a .txt file.
2. **FileWriter**:
   * Writes characters to a text file.
   * Example: Writing text to a .txt file.

**Buffered Streams**

Buffered streams wrap around basic byte and character streams to enhance performance by reducing the number of direct I/O operations.

1. **BufferedInputStream**:
   * Buffers raw data for efficient reading.
   * Example: Reading large binary files.
2. **BufferedOutputStream**:
   * Buffers raw data for efficient writing.
   * Example: Writing large chunks of binary data.
3. **BufferedReader**:
   * Buffers text data for efficient reading line by line.
   * Example: Reading lines of text from a file.
4. **BufferedWriter**:
   * Buffers text data for efficient writing.
   * Example: Writing multiple lines of text to a file.

|  |  |  |  |
| --- | --- | --- | --- |
| **Stream Type** | **Usage** | **Related Classes** | **Example Use Case** |
| FileInputStream | Reads raw binary data | FileInputStream | Image file reading |
| FileOutputStream | Writes raw binary data | FileOutputStream | Audio file saving |
| BufferedInputStream | Buffers raw data for reading | BufferedInputStream | Reading large binary files |
| BufferedOutputStream | Buffers raw data for writing | BufferedOutputStream | Writing chunks of binary data |

**Character Streams**

These are used for reading and writing character data (text).

|  |  |  |  |
| --- | --- | --- | --- |
| **Stream Type** | **Usage** | **Related Classes** | **Example Use Case** |
| FileReader | Reads characters from text | FileReader | Text file reading |
| FileWriter | Writes characters to text | FileWriter | Text file writing |
| BufferedReader | Buffers text for reading | BufferedReader | Line-by-line text reading |
| BufferedWriter | Buffers text for writing | BufferedWriter | Writing multiple lines |
| PrintStream | Writes formatted text | PrintStream | Logging to file |

// PROGRAM TO READ FROM FILE

import java.io.FileInputStream;

import java.io.IOException;

public class FileInputExample {

public static void main(String[] args) {

try (FileInputStream fis = new FileInputStream("input.txt")) {

int content;

while ((content = fis.read()) != -1) {

System.out.print((char) content); // Print character by character

}

} catch (IOException e) {

System.out.println(“Read operation unsuccessful”);

//e.printStackTrace();

}

}

}

// PROGRAM TO WRITE INTO A FILE

import java.io.FileOutputStream;

import java.io.IOException;

public class FileOutputExample {

public static void main(String[] args) {

String data = "Hello, this is a test!";

try (FileOutputStream fos = new FileOutputStream("output.txt")) {

fos.write(data.getBytes()); // Convert string to bytes and write

System.out.println("Data written to file successfully.");

} catch (IOException e) {

System.out.println(“Write operation UNSUCCESSFUL”);

//e.printStackTrace();

}

}

}

In Java, the printStackTrace() method is used to print the stack trace of an exception to the console or log. A stack trace provides detailed information about the sequence of method calls that led to the exception, helping developers debug their code more effectively.

Buffered streams in Java are used to enhance the performance of input and output operations by reducing the number of direct interactions with the underlying data source or destination. Instead of reading or writing one byte or character at a time, buffered streams read chunks of data into an internal buffer and then process them, making the operations faster and more efficient.

**Types of Buffered Streams**

Java provides two main types of buffered streams:

1. **BufferedInputStream**
2. **BufferedOutputStream**

**How They Work**

* **BufferedInputStream**: It wraps around another input stream (like FileInputStream) and reads data into an internal buffer. Subsequent read operations are performed directly on this buffer until it's exhausted.
* **BufferedOutputStream**: It wraps around another output stream (like FileOutputStream) and stores data in a buffer before writing it to the underlying stream. The data is only written when the buffer is full or when you explicitly flush the stream.

// FOR READING

import java.io.BufferedInputStream;

import java.io.FileInputStream;

import java.io.IOException;

public class BufferedInputExample {

public static void main(String[] args) {

try (BufferedInputStream bis = new BufferedInputStream(new FileInputStream("input.txt"))) {

int content;

while ((content = bis.read()) != -1) {

System.out.print((char) content); // Print character by character

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

// FOR WRITING

import java.io.BufferedOutputStream;

import java.io.FileOutputStream;

import java.io.IOException;

public class BufferedOutputExample {

public static void main(String[] args) {

String data = "Buffered streams improve I/O efficiency!";

try

(BufferedOutputStream bos = new BufferedOutputStream(new FileOutputStream("output.txt"))) {

bos.write(data.getBytes());

System.out.println("Data written successfully.");

} catch (IOException e) {

//e.printStackTrace();

System.out.println(“Unable to perform Write operation”/);

}

}

}

// PROGRAM TO READ-WRITE IN FILE IN CHARACTER FORMAT

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

public class CharacterStreamExample {

public static void main(String[] args) {

try (FileReader fr = new FileReader("ip.txt");

FileWriter fw = new FileWriter("op.txt")) {

int data;

while ((data = fr.read()) != -1) { // Reads one character at a time

fw.write(data); // Writes the character to the output file

}

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

} catch (IOException e) {

System.out.println(“Copy Not done”); //e.printStackTrace();

}

}

}

**Using BufferedReader**

BufferedReader is ideal for reading larger chunks of text efficiently to **READ** from file

import java.io.\*;

public class BufferedReaderExample {

public static void main(String[] args) {

try (BufferedReader reader = new BufferedReader(new FileReader("input.txt"))) {

String line;

System.out.println("Reading from file:");

while ((line = reader.readLine()) != null) {

System.out.println(line);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Using BufferedWriter**

BufferedReader is ideal for writing larger chunks of text efficiently to **WRITE** into file

import java.io.\*;

public class BufferedWriterExample {

public static void main(String[] args) {

try (BufferedWriter writer = new BufferedWriter(new FileWriter("output.txt"))) {

writer.write("Hello, world!");

writer.newLine(); // Adds a new line

writer.write("This is written using BufferedWriter.");

System.out.println("Content written to file successfully.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

**READ WRITE with CONSOLE**

**Using System.out (for writing) and System.in (for reading)**

import java.io.IOException;

public class ConsInOut {

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

System.out.println("Enter a character:");

int input = System.in.read(); //data is read in byte form

System.out.println("You entered: " + (char) input); // data type casted to character form

}

}

**Using Scanner**

The Scanner class makes it easier to read user input in various formats, like strings, integers.

import java.util.Scanner;

public class ConsoleWithScanner {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter your name:");

String name = scanner.nextLine(); // Reads String

System.out.println("Welcome, " + name + "!");

System.out.println("Enter your age:");

int age = scanner.nextInt(); // Reads an integer

System.out.println("You are " + age + " years old.");

}

}

Read Bytes - System.in

Read Characters - InputStreamReader

Read Lines - BufferedReader

Write output - System.out

**Formatting**

The PrintWriter class is used to produce formatted output

Example 1

import java.io.PrintWriter;

import java.io.IOException;

public class FormattedOutputExample {

public static void main(String[] args) {

try (PrintWriter writer = new PrintWriter("Alice.txt")) {

String name = "Alice";

int age = 25;

double score = 95.5;

writer.printf("Name: %s | Age: %d | Score: %.2f%n", name, age, score);

writer.println("This is another line of text.");

writer.println("PrintWriter makes output elegant!");

System.out.println("Data written to file successfully.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

Example 2

import java.io.PrintWriter;

import java.io.IOException;

public class PrintWriterFormattingExample {

public static void main(String[] args) {

try (PrintWriter writer = new PrintWriter("employee.txt")) {

writer.println("Employee Salary Report");

writer.println("========================");

writer.printf("%-15s %-10s %10s%n", "Name", "Position", "Salary");

writer.println("---------------------------------------");

writer.printf("%-15s %-10s %10.2f%n", "Alice", "Manager", 75000.0);

writer.printf("%-15s %-10s %10.2f%n", "Bob", "Developer", 55000.5);

writer.printf("%-15s %-10s %10.2f%n", "Eve", "Analyst", 60000.25);

writer.println("---------------------------------------");

writer.printf("%-15s %-10s %10s%n", "", "Total:", "$190,000.75");

System.out.println("Formatted data written successfully.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

Example 3

import java.io.PrintWriter;

import java.io.IOException;

public class PrintWriterGeneralFormatting {

public static void main(String[] args) {

try (PrintWriter writer = new PrintWriter("general\_formatting.txt")) {

writer.println("Student Marks Report");

writer.println("======================");

writer.printf("%-15s %-10s %-10s%n", "Name", "Roll No", "Marks");

writer.println("-----------------------------------");

writer.printf("%-15s %-10d %-10.2f%n", "Alice", 101, 95.50);

writer.printf("%-15s %-10d %-10.2f%n", "Bob", 102, 87.25);

writer.printf("%-15s %-10d %-10.2f%n", "Eve", 103, 92.75);

writer.println("-----------------------------------");

writer.printf("%-15s %-10s %-10.2f%n", "Average", "-", (95.50 + 87.25 + 92.75) / 3);

System.out.println("Formatted data written successfully.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

**OBJECT SERIALIZABLE**

**To convert Object to bytes or bits stream**

import java.io.\*; // marker interface or empty interface

class Student implements Serializable {

private static final long serialVersionUID = 1L;

String name;

int age;

transient String password; // not serialized

public Student(String name, int age, String password) {

this.name = name;

this.age = age;

this.password = password;

}

void display() {

System.out.println("Name: " + name);

System.out.println("Age: " + age);

System.out.println("Password: " + password);

}

}

public class SerializationExample {

public static void main(String[] args) {

Student student = new Student("Alice", 20, "secret123");

try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("student.ser"))) {

out.writeObject(student);

System.out.println("Object serialized successfully.");

} catch (IOException e) {

e.printStackTrace();

}

student = null;

try (ObjectInputStream in = new ObjectInputStream(new FileInputStream("student.ser"))) {

Student ds = (Student) in.readObject();

System.out.println("\nDeserialized Object:");

ds.display(); // password will be null

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

}

}

}

This line is a **version identifier** used during the **serialization and deserialization** process in Java.