PrintStream

PrintWriter

OutputStreamWriter

InputStreamReader

PushbackInputStream

← prev next →

Java I/O Tutorial

Java I/O (Input and Output) is used to process the input and produce the output.

Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations.

We can perform **file handling in Java** by Java I/O API.

Core Concepts of Java I/O

Java I/O revolves around two primary concepts: **streams** and **readers/writers**.

Streams: Streams represent a sequence of data. In Java, there are two types of streams: input streams and output streams. Input streams are used to read data from a source, while output streams are used to write data to a destination. Streams can be categorized into byte streams (InputStream and OutputStream) and character streams (Reader and Writer).

Readers/Writers: Readers and writers are specialized stream classes designed for handling character data. They provide a convenient way to read from and write to character-based data sources. Readers read character data from input streams, while writers write character data to output streams.

Stream

A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.

In Java, 3 streams are created for us automatically. All 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

Let's see the code to print **output and an error** message to the console.

```
System.out.println("simple message");
System.err.println("error message");
```

Let's see the code to get **input** from console.

```
int i=System.in.read();//returns ASCII code of 1st character
System.out.println((char)i);//will print the character
```

OutputStream Vs. InputStream

The explanation of OutputStream and InputStream classes are given below:

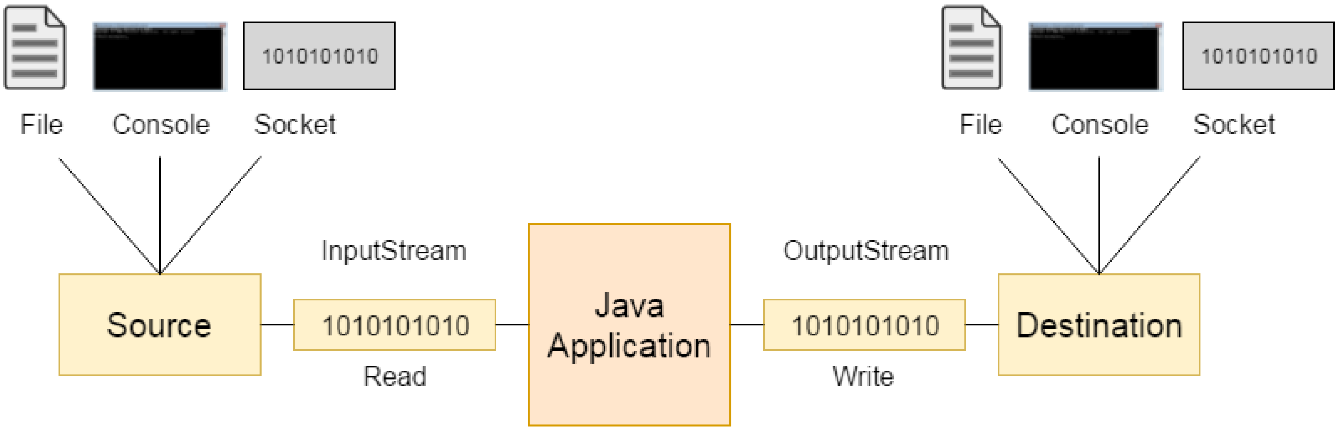
OutputStream

Java application uses an output stream to write data to a destination; it may be a file, an array, peripheral device or socket.

InputStream

Java application uses an input stream to read data from a source; it may be a file, an array, peripheral device or socket.

Let's understand the working of Java OutputStream and InputStream class by the figure given below.



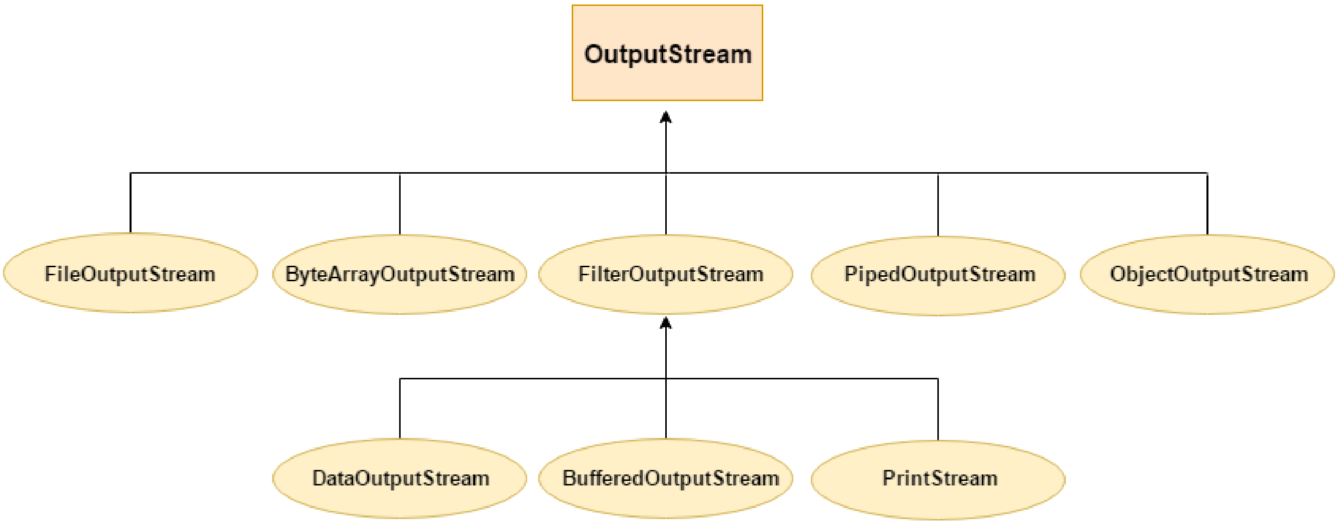
OutputStream Class

OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. An output stream accepts output bytes and sends them to some sink.

Useful Methods of OutputStream Class

Method	Description
public void write(int) throws IOException	It is used to write a byte to the current output stream.
public void write(byte[])throws IOException	It is used to write an array of byte to the current output stream.
public void flush() throws IOException	It flushes the current output stream.
public void close() throws IOException	It is used to close the current output stream.

OutputStream Hierarchy



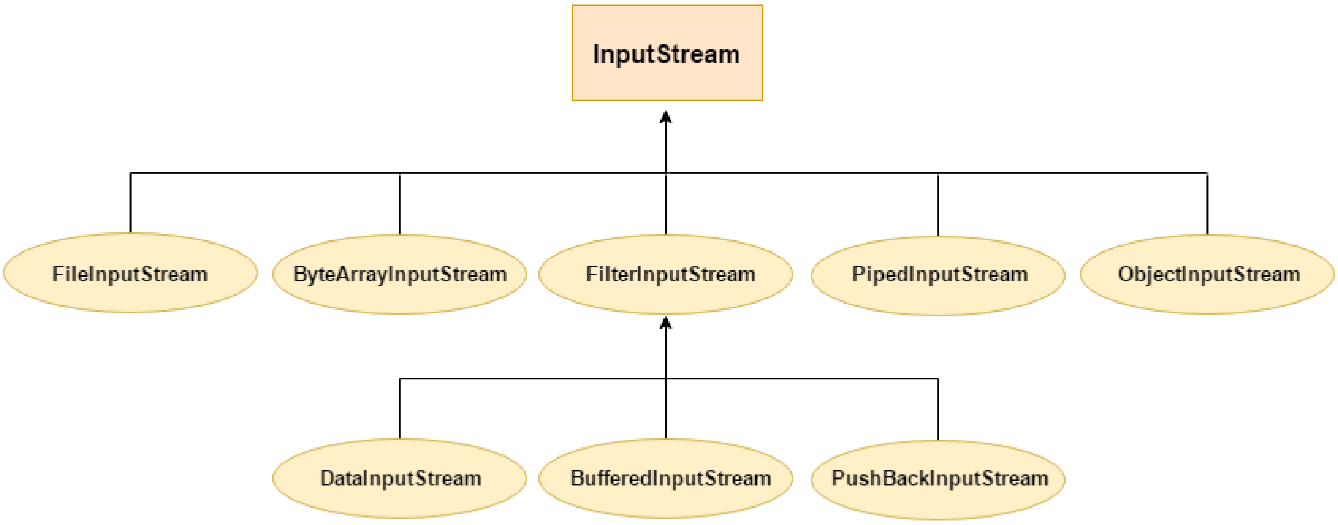
InputStream Class

InputStream class is an abstract class. It is the superclass of all classes representing an input stream of bytes.

Useful Methods of InputStream Class

Method	Description
public abstract int read() throws IOException	It reads the next byte of data from the input stream. It returns -1 at the end of the file.
public int available() throws IOException	It returns an estimate of the number of bytes that can be read from the current input stream.
public void close() throws IOException	It is used to close the current input stream.

InputStream Hierarchy



Java I/O Classes

1. Java provides a rich set of classes for performing I/O operations. Some of the key classes include:
2. **InputStream and OutputStream:** These abstract classes form the foundation for byte-oriented I/O operations. They provide methods for reading and writing bytes from/to various sources and destinations.
3. **Reader and Writer:** These abstract classes are used for character-based I/O operations. They provide methods for reading and writing characters from/to character-based streams.
4. **FileInputStream and FileOutputStream:** These classes allow reading from and writing to files in a byte-oriented manner.
5. **FileReader and FileWriter:** These classes enable reading from and writing to files using character-oriented operations.
6. **BufferedInputStream and BufferedOutputStream:** These classes provide buffering capabilities, which can significantly improve I/O performance by reducing the number of system calls.
7. **BufferedReader and BufferedWriter:** These classes offer buffered reading and writing of character data, enhancing I/O efficiency when working with character-based streams.

Practical Applications of Java I/O

Java I/O is employed in various real-world scenarios, including:

1. **File Handling:** Java I/O is extensively used for reading from and writing to files. Developers can manipulate files, create directories, and perform file-related operations using Java's file I/O classes.
2. **Network Communication:** Java's socket classes (Socket and ServerSocket) facilitate network communication by enabling data exchange between client and server applications over TCP/IP.

3. **Serialization:** Java's serialization mechanism allows objects to be converted into a stream of bytes for storage or transmission. It is particularly useful for storing object state or transferring objects between applications.
4. **Data Processing:** Java I/O is integral to data processing tasks such as parsing text files, processing CSV data, and interacting with databases through JDBC (Java Database Connectivity).

Best Practices for Java I/O

When working with Java I/O, consider the following best practices:

1. **Use Try-with-Resources:** Always use the try-with-resources statement when working with streams to ensure proper resource management. This automatically closes the streams when they are no longer needed, preventing resource leaks.
2. **Use Buffered I/O:** Whenever possible, use buffered I/O classes to minimize the number of system calls and improve performance.
3. **Handle Exceptions Gracefully:** Handle I/O exceptions gracefully by implementing error handling mechanisms such as logging or error propagation.
4. **Use NIO for Performance:** For high-performance I/O operations, consider using Java's NIO (New I/O) package, which provides non-blocking I/O features and enhanced performance.

Conclusion

Java I/O is a fundamental aspect of Java programming, offering powerful capabilities for handling input and output operations. By understanding the core concepts, classes, and best practices of Java I/O, developers can build robust and efficient applications that interact with external resources seamlessly. Whether it's reading from files, communicating over networks, or processing data streams, Java I/O provides the tools necessary to tackle a wide range of I/O tasks effectively.

Java IO MCQ

1. Which of the following classes is used for reading character streams in Java?

1. FileInputStream
2. FileReader
3. BufferedReader
4. InputStreamReader

Show Answer Workspace

2. What is the purpose of the BufferedWriter class in Java I/O?

1. It is used for reading binary data from a file.
2. It is used for writing text to a character-output stream, buffering characters to provide efficient writing.
3. It is used for reading text from a character-input stream, buffering characters to provide efficient reading.
4. It is used for writing binary data to a file.

Show Answer Workspace

3. Which of the following methods is used to read a line of text from a file using BufferedReader in Java?

1. readLine()
2. read()
3. readChar()
4. readText()

Show Answer Workspace

4. What is the purpose of the ObjectInputStream class in Java?

1. It is used to serialize objects into byte streams.
2. It is used to deserialize objects from byte streams.
3. It is used to read primitive data types from a binary stream.
4. It is used to read objects from a text file.

Show Answer Workspace

5. Which of the following statements is true about the FileWriter class in Java?

1. It is used to write characters to a file using binary streams.
2. It is used to write characters to a file using character streams.
3. It is used to read characters from a file using binary streams.
4. It is used to read characters from a file using character streams.

Show Answer Workspace