IO Operations

Assignment Questions

Assignment Questions:

1. What is input and output stream in Java?

Ans → In Java, streams are used for input and output operations. An input stream is used to read data from a source, while an output stream is used to write data to a destination.

An input stream can be associated with various data sources such as files, network sockets, or in-memory data structures. The InputStream class is the base class for all input streams in Java. Some common subclasses of InputStream include FileInputStream, ByteArrayInputStream, and BufferedInputStream.

An output stream can be associated with various data destinations such as files, network sockets, or in-memory data structures. The OutputStream class is the base class for all output streams in Java. Some common subclasses of OutputStream include FileOutputStream, ByteArrayOutputStream, and BufferedOutputStream.

1. What are the methods of output streams in Java?

Ans →The OutputStream class in Java is the base class for all output streams, and it provides several methods that can be used to write data to an output stream. Here are some of the most commonly used methods of the OutputStream class:

void write(int b): Writes a single byte of data to the output stream.

void write(byte[] b): Writes an array of bytes to the output stream.

void write(byte[] b, int off, int len): Writes a portion of an array of bytes to the output stream.

void flush(): Flushes the output stream, ensuring that all buffered data is written to the destination.

void close(): Closes the output stream, releasing any resources associated with the stream.

void writeBoolean(boolean v): Writes a boolean value to the output stream.

void writeByte(int v): Writes a byte value to the output stream.

void writeShort(int v): Writes a short value to the output stream.

void writeChar(int v): Writes a character value to the output stream.

void writeInt(int v): Writes an integer value to the output stream.

void writeLong(long v): Writes a long value to the output stream.

void writeFloat(float v): Writes a float value to the output stream.

void writeDouble(double v): Writes a double value to the output stream.

1. What is serialization in Java?

Ans →Serialization in Java is the process of converting an object into a stream of bytes so that it can be easily stored in a file, database, or sent over a network. The reverse process of converting a stream of bytes back into an object is called deserialization.

Java provides built-in support for serialization through the java.io.Serializable interface. Any class that implements this interface can be serialized and deserialized using Java's serialization framework.

1. What is a serializable interface in Java?

Ans →The Serializable interface in Java is a marker interface that indicates to the Java runtime that a class can be serialized. When a class implements the Serializable interface, it can be converted into a stream of bytes and stored or transmitted in a variety of formats, such as a file or over a network.

The Serializable interface itself does not contain any methods that need to be implemented. Instead, it is used to indicate that the class can be serialized and that the Java runtime can use its default serialization mechanism to serialize and deserialize objects of that class.

1. What is deserialization in Java?

Ans → Deserialization in Java is the process of converting a stream of bytes back into an object. This is typically done after the object has been serialized and stored in a file or transmitted over a network.

Java provides built-in support for deserialization through the java.io.Serializable interface. Any class that implements this interface can be deserialized using Java's deserialization framework.

1. How is serialization achieved in Java?

Ans →Serialization in Java is achieved using the java.io.Serializable interface and the built-in Java serialization framework. When a class implements the Serializable interface, it tells the Java runtime that its objects can be converted into a stream of bytes and then restored back into objects.

1. How is deserialization achieved in Java?

Ans →Deserialization in Java is achieved using the java.io.Serializable interface and the built-in Java serialization framework. When an object is serialized, its state is converted into a stream of bytes, which can then be sent over a network or stored in a file. When the object is deserialized, the bytes are read back into memory and used to reconstruct the object's state.

1. How can you avoid certain member variables of class from getting serialized?

Ans →In Java, you can prevent certain member variables of a class from getting serialized by marking them as transient. When an object is serialized, any variables marked as transient will not be written to the output stream and will be initialized to their default values when the object is deserialized.

public class Person implements Serializable {

private String name;

private transient int age; // this variable will not be serialized

public Person(String name, int age) {

this.name = name;

this.age = age;

}

}

1. What classes are available in the Java IO File classes API?

Ans → The Java IO File classes API provides a variety of classes for working with files and directories in Java. Here are some of the most commonly used classes in this API:

File - represents a file or directory on the file system and provides methods for manipulating its attributes, such as its name, path, and size.

FileInputStream - provides a stream for reading bytes from a file.

FileOutputStream - provides a stream for writing bytes to a file.

FileReader - provides a stream for reading characters from a file.

FileWriter - provides a stream for writing characters to a file.

BufferedReader - provides buffering for a FileReader or any other Reader.

BufferedWriter - provides buffering for a FileWriter or any other Writer.

PrintWriter - provides a convenient way to write formatted text to a file.

RandomAccessFile - provides methods for reading and writing data to a file at a specific location.

10. What is the difference between Externalizable and serializable interface?

Ans → The Serializable interface and the Externalizable interface are both used in Java for object serialization, but they have some important differences.

Here are the main differences between Serializable and Externalizable interfaces:

1. Control over serialization process: The Serializable interface is a marker interface, meaning it doesn't have any methods. When an object of a Serializable class is serialized, the JVM takes care of the entire serialization process. On the other hand, the Externalizable interface provides more control over the serialization process, as it requires the implementing class to define its own writeExternal() and readExternal() methods for serialization and deserialization.
2. Serialization size: When an object is serialized using Serializable interface, it includes all its fields in the serialized data, regardless of whether they are null or not. This can result in larger serialized data. However, with Externalizable interface, you have the ability to control the serialized data size by choosing which fields to serialize or not.
3. Custom serialization logic: Externalizable interface allows you to define your own custom serialization and deserialization logic, while Serializable interface does not provide this level of control.