Lec 43 File IO P2:

This lesson explains how Java handles data input/output efficiently using streams, and how objects can be saved and restored through serialization and deserialization. It also covers practical file operations like copying and moving files, demonstrating how different types of data are managed in Java.

Buffered Streams and Performance

Buffered input and output streams in Java are used to read and write data in chunks, which reduces the number of times the hard disk is accessed, leading to better performance compared to unbuffered streams.

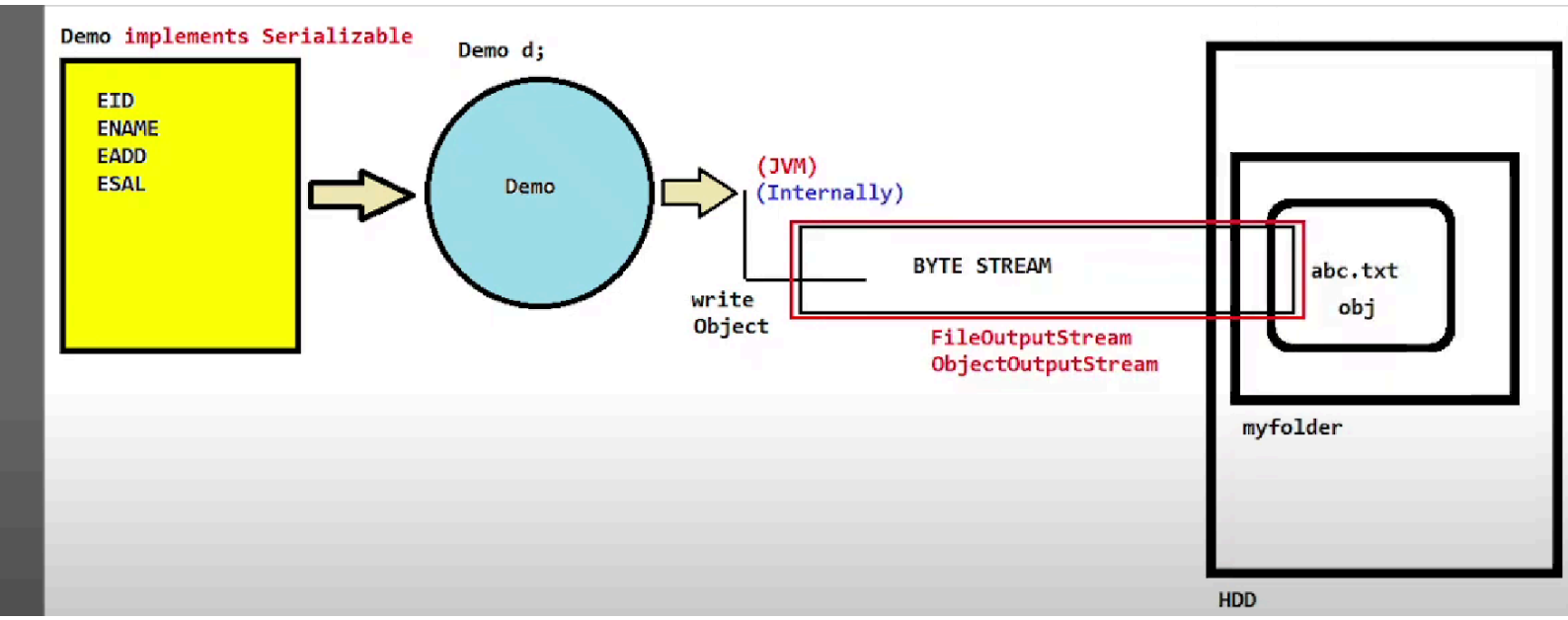
Introduction to Serialization and Deserialization

Serialization is converting an object into a byte stream so it can be stored or transferred, while deserialization is the reverse process—reconstructing the object from the byte stream.

How Serialization Works

To serialize, you convert an object to a byte stream and write it to a file; to deserialize, you read the byte stream

from the file and recreate the object. This is different from simply writing primitive data or strings, which are more straightforward.



Objects and Byte Streams in Java

Only objects that implement the Serializable marker interface (interface which have nothing!!!) can be converted into byte streams by the Java Virtual Machine (JVM). The JVM handles the conversion process when needed, such as when saving to a file or sending over a network.

Writing Objects to Files

To write an object to a file, you use an ObjectOutputStream wrapped around a

FileOutputStream. The method writeObject() is used instead of the usual write() for byte arrays or primitives.

Serialization in Practice

Attempting to serialize a class that does not implement Serializable will cause an exception. Once the class implements the interface, the object can be written to a file as a byte stream.

Reading Objects from Files (Deserialization)

To read an object back from a file, use an ObjectInputStream with a FileInputStream, and call readObject(). The returned object must be typecast to its original class.

Handling Multiple Objects in Files

Multiple objects can be written to the same file using writeObject() multiple times. When reading, each call to readObject() retrieves the next object in the file, but care must be taken with file structure and appending.

File Copy-Paste Operations

To copy a file, read its contents as a byte array using a FileInputStream, then write this array to the destination using a FileOutputStream. This approach works for all file types, including images and videos.

Java Version Compatibility Issues

Some methods, like readAllBytes(), are only available in newer Java versions (e.g., Java 17). If using an older version, alternative methods or manual byte reading must be used.

Completing File Copy and Cut-Paste

After copying the byte array to the destination, deleting the source file achieves a "cut-paste" effect. Proper closing of streams and handling file paths are important for reliable file operations.