אורט בראודה
ORT Braude College

שיטות הנדסיות לפיתוח מערכות תוכנה

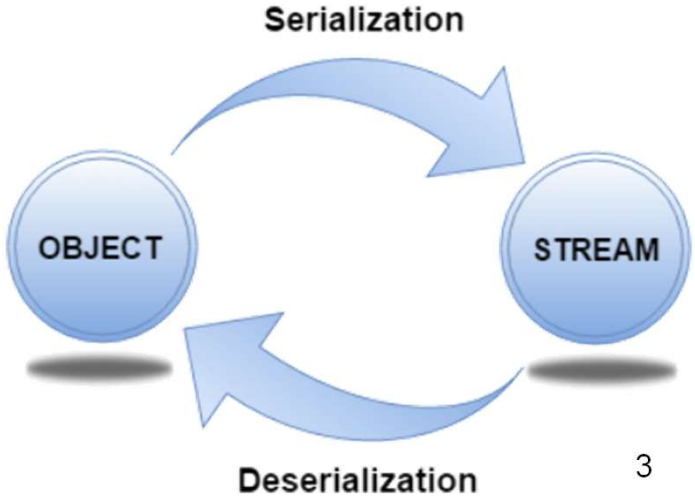
Serializable and Using OCSF

Object Serialization

- ✓ Data stored in variables and arrays is temporary
  - ✓ It's lost when a local variable goes out of scope or when the program terminates
- ✓ For long-term retention of data, computers use files
- ✓ Computers store files on secondary storage devices
  - ✓ hard disks, optical disks, flash drives and magnetic tapes.
- ✓ Data maintained in files is persistent data because it exists beyond the duration of program execution

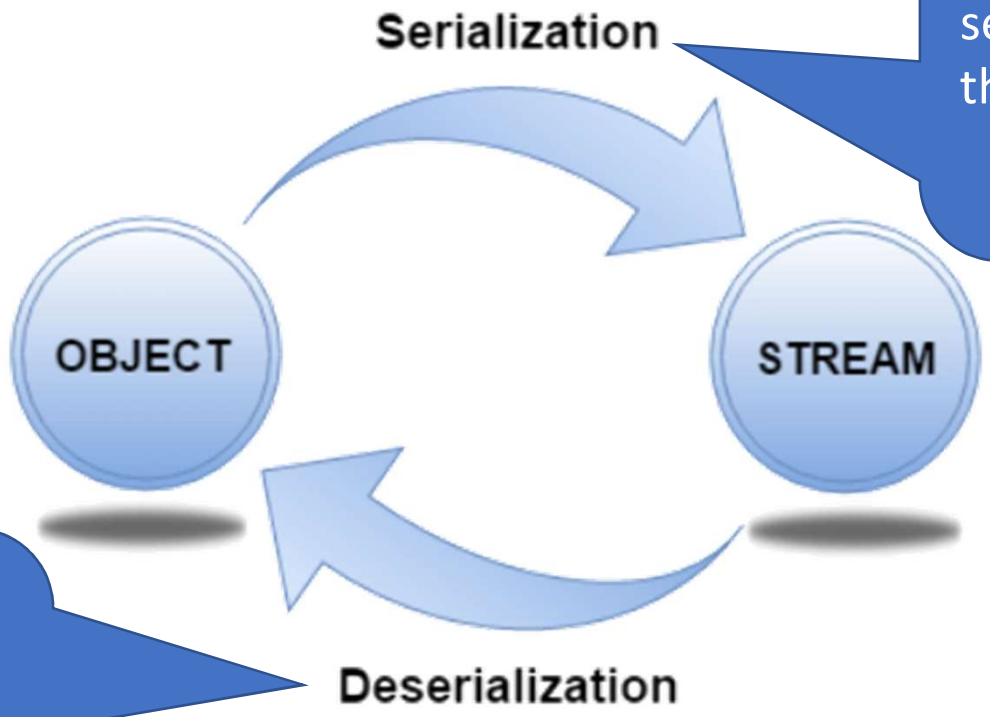
Object Serialization

- ✓ To read an entire object from or write an entire object to a file, Java provides **object serialization**.
- ✓ **A serialized object** is represented as a sequence of bytes that includes the object's data and its type information.
- ✓ After a serialized object has been written into a file, it can be read from the file and **deserialized** to recreate the object in memory.



Object Serialization

The process of converting an object into a sequence of bytes that can be stored or sent through steams



The reverse process of creating an object from a sequence of bytes is called deserialization

Object Serialization

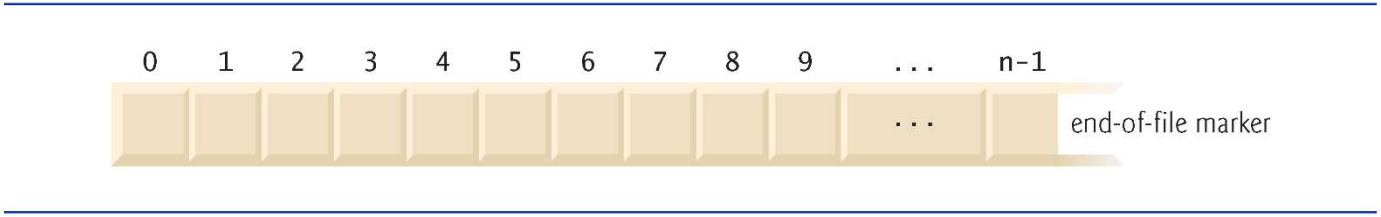
- ✓ Objects to be serialized
  - ✓ Must **implement Serializable interface:**

```
public class MyFile implements Serializable {
```

- ✓ The following is written and read during serialization
  - ✓ Class of the object
  - ✓ Class signature
  - ✓ Values of all non-transient and non-static members

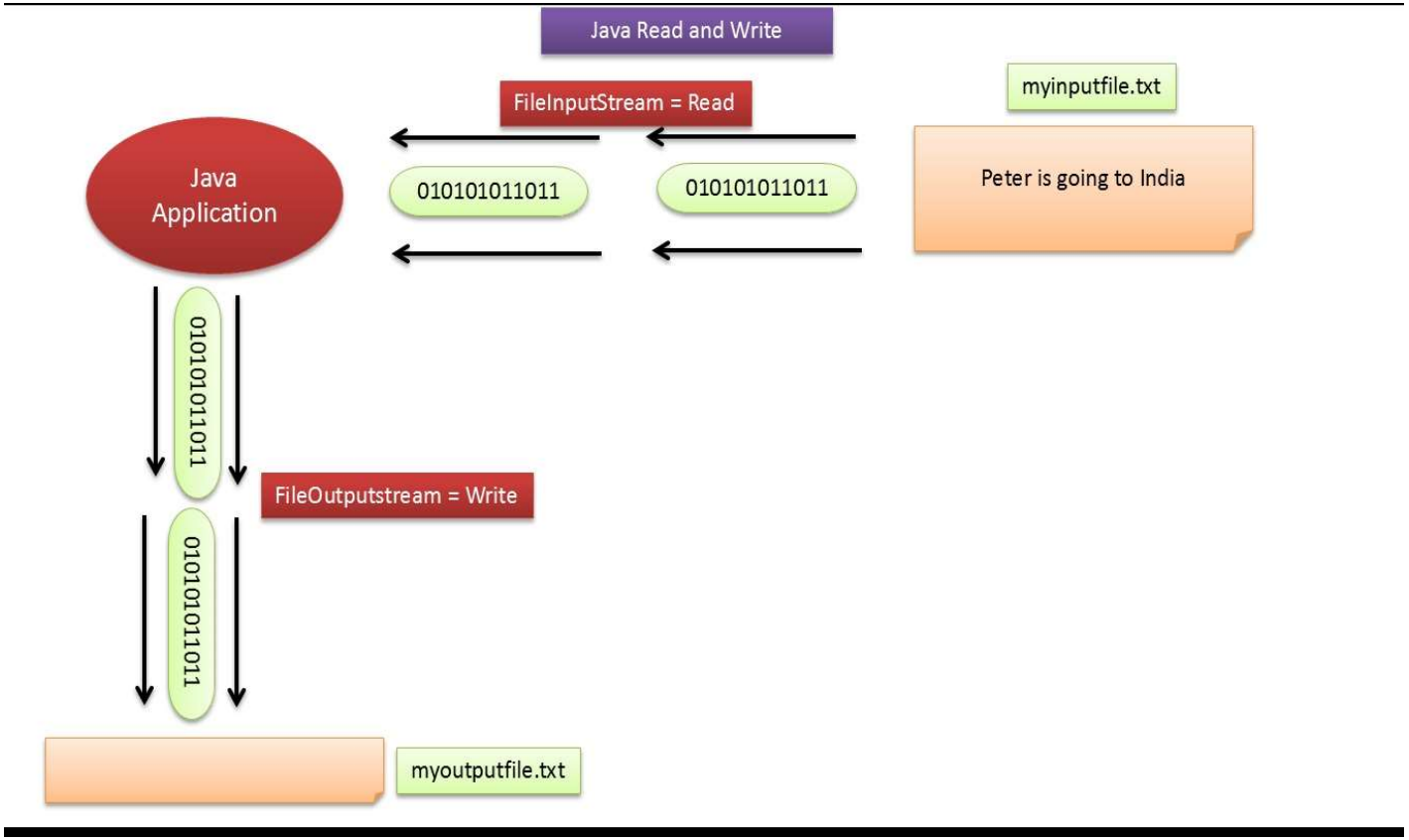
Object Serialization

- ✓ Classes **ObjectInputStream** and **ObjectOutputStream**, which respectively implement the **ObjectInput** and **ObjectOutput** interfaces, enable entire objects to be read from or written to a stream.
- ✓ An **ObjectOutputStream** will not output an object unless it is a **Serializable** object.
- ✓ Java views each file as a sequential **stream of bytes:**



Object Serialization

✓ To use serialization with files, initialize **ObjectInputStream** and **ObjectOutputStream** objects with **FileInputStream** and **FileOutputStream** objects.



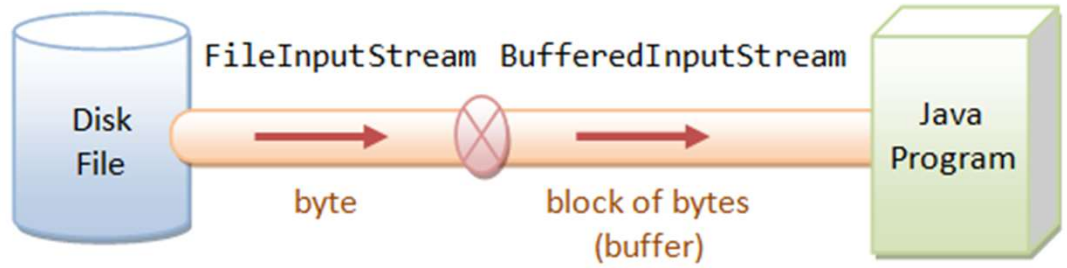
Object Serialization

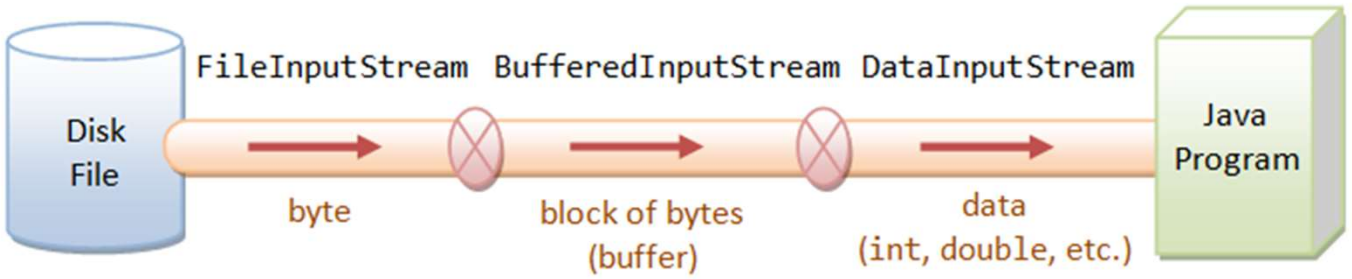
Layered (or Chained) I/O Streams

- ✓ The I/O streams are often layered or chained with other I/O streams, for purposes such as buffering, filtering, or data-format conversion (between raw bytes and primitive types)

Layered (or Chained) I/O Streams

✓ For example, we can layer a **BufferedInputStream** to a **FileInputStream** for buffered input and stack a **DataInputStream** in front for formatted data input (using primitives such as int, double), as illustrated in the following diagrams.





Object Serialization

Buffered I/O Byte-Streams - BufferedInputStream & BufferedOutputStream

- ✓ The read()/write() method in InputStream/OutputStream are designed to read/write a **single** byte of data on each call.
- ✓ **Buffering**, which reads/writes a **block** of bytes from the external device into/from a memory buffer in a single I/O operation, is commonly applied to speed up the I/O.

Object Serialization

Buffered I/O Byte-Streams - BufferedInputStream & BufferedOutputStream

- ✓ **FileInputStream/FileOutputStream** is not buffered. It is often chained to a BufferedInputStream or BufferedOutputStream, which provides the buffering.
- ✓ To chain the streams together, simply pass an instance of one stream into the constructor of another stream.

Object Serialization

For example, the following codes chain a FileInputStream to a BufferedInputStream, and finally, a DataInputStream:

```
FileInputStream fileIn = new FileInputStream("in.dat");
BufferedInputStream bufferIn = new
BufferedInputStream(fileIn);
DataInputStream dataIn = new DataInputStream(bufferIn);
// or
DataInputStream in = new DataInputStream(
                    new BufferedInputStream(
                      new FileInputStream("in.dat")));
```

Object Serialization

✓ In the ChatClient we create a new type of variable MyFile:

```
public void handleMessageFromClientUI(String message)
{
    MyFile msg= new MyFile("diagnosisE.jpg");
    String LocalfilePath="diagnosisE.jpg";

    try{

            File newFile = new File (LocalfilePath);

            byte [] mybytearray  = new byte [(int)newFile.length()];
            FileInputStream fis = new FileInputStream(newFile);
            BufferedInputStream bis = new BufferedInputStream(fis);

            msg.initArray(mybytearray.length);
            msg.setSize(mybytearray.length);

            bis.read(msg.getMybytearray(),0,mybytearray.length);
            sendToServer(msg);
        }
      catch (Exception e) {
          System.out.println("Error send (Files)msg) to Server");
      }
}
```

✓ It stores the data as a bytearray and additional data.

✓ This is the object that we send from client to server.

Object Serialization

✓This is how our server handles the received object:

```
public void handleMessageFromClient(Object msg, ConnectionToClient client)
{
    int fileSize =((MyFile)msg).getSize();
    System.out.println("Message received: " + msg + " from " + client);
    System.out.println("length "+ fileSize);
}
```

✓ As a result, we see:

```
Server listening for connections on port 5555
Message received: common.MyFile@2e2e9b32 from 127.0.0.1 (127.0.0.1)
length 18984
```

✓ Provide the following change:

```
System.out.println("Message received: " + ((MyFile)msg).getFileName() + " from " + client);
```

✓ The result will be:

```
Server listening for connections on port 5555
Message received: diagnosisE.jpg from 127.0.0.1 (127.0.0.1)
length 18984
```

מטלה

1. Import projects *OCSF & simpleSend_File (cln_to_srv)* to Eclipse (the .ZIP file is attached).
2. **Understand a code!**
3. Run the program and, if it's needed, fix the problem/s.
4. What the code does?

Note: For more help see for example the following links:

- http://www.java2s.com/Code/Java/Network-Protocol/TransferafileviaSocket.htm

מטלה

5. Add additional functionality:
   - Uploading a content of file from Client to a file on Server (simpleSend_File(cln_to_srv))

Note: This functionality should be presented as a method/s and has to be as general as is possible. For example, you need to type only a file name and its path (d:/test.mov) and all rest job has to be performed automatically.

מטלה (5)

On the server side we should:

1. Create a new file on the server file system
2. Write to this file:
   - ✓ Create file output stream:

```
FileOutputStream fos = new FileOutputStream(newFile);
```

   - ✓ Create BufferedFileOutputStream:

```
BufferedOutputStream bos = new BufferedOutputStream(fos);
```

   - ✓ Write byte array to output stream:

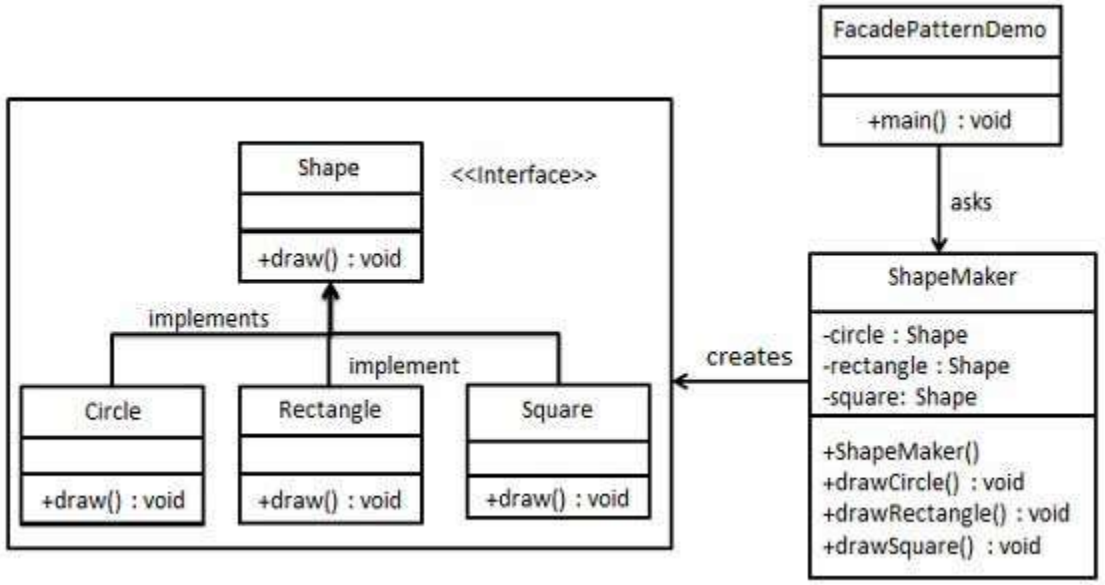
```
bos.write(mybytearray, 0 , current);
bos.flush();
fos.flush();
```

מטלה

6. Create big file ~20Mb and present to a teacher this functionality (upload this file from the Client to the Server).

שבוע הבא נדבר על...

Design Patterns



Object Serialization

When working with big files (~20Mb), upload them from the Client to the Server, we use BLOB tables.

- ✓ A BLOB (binary large object) is a varying-length binary string that can be up to 2,147,483,647 characters long

The steps are:

1. To create BLOB use Connection.createBlob
2. To write BLOB to DB use PreparedStatement.setBlob
3. To read BLOB from DB use ResultSet.getBlob