**TP Persistencia**

---

**¿Cómo se convierte una clase en un flujo de bytes?**
Una clase se convierte en un flujo de bytes implementando la interfaz Serializable. Luego, se puede usar un ObjectOutputStream junto con un FileOutputStream para escribir el objeto en un archivo o transmitirlo
**¿Cuáles son los miembros de la interfaz Serializable?**
La interfaz Serializable no tiene métodos ni miembros; es una interfaz marcadora (marker interface). Su única función es indicar que una clase puede ser serializada.
**¿Cuáles son las clases que permiten la persistencia de objetos?**
**ObjectOutputStream:** Para escribir objetos en un flujo.
**ObjectInputStream:** Para leer objetos desde un flujo.
También se utilizan FileOutputStream y FileInputStream para manejar archivos donde se guardan/leen los objetos.
**¿Cuál es la diferencia en el uso de FileReader y FileInputStream?**
**FileReader** se usa para leer archivos de texto (caracteres), trabaja con char y utiliza codificación.
**FileInputStream** se usa para leer archivos binarios (bytes), trabaja con byte.
**¿Cuáles son las excepciones que pueden lanzarse en un fallo de entrada/salida?**
IOException (genérica para errores de E/S)
FileNotFoundException (archivo no encontrado)
EOFException (fin de archivo inesperado al leer)
NotSerializableException (cuando se intenta serializar un objeto que no implementa Serializable)
StreamCorruptedException, entre otras.
**¿Cuál es el paquete a importar para obtener las clases de Entrada/Salida?**
import java.io.*;

**Complete el siguiente cuadro:**

| | Random Access File | Almacenar Bytes | Almacenar caracteres |
|---|---|---|---|
| Declaración de archivo para escribir | RandomAccessFile raf = new RandomAccessFile("archivo.dat", "rw"); | FileOutputStream fos = new FileOutputStream ("archivo.dat"); | FileWriter fw = new FileWriter("archivo.txt"); |
| Declaración de archivo para leer | RandomAccessFile raf = new RandomAccessFile("archivo.dat", "r"); | FileInputStream fis = new FileInputStream(" archivo.dat"); | FileReader fr = new FileReader("archivo.txt"(; |
| Escribir en archivo | raf.writeInt(123); | fos.write(65); // escribe 'A' | fw.write("Hola"); |
| Leer en archivo | int x = raf.readInt(); | int b = fis.read(); | int c = fr.read(); |
| Cerrar Archivo | raf.close(); | fos.close(); / fis.close(); | fw.close(); / fr.close(); |

4a. NO, no son equivalentes.
La primera forma usa un try-with-resources, que cierra automáticamente el DataOutputStream al terminar.
La segunda forma NO lo cierra automáticamente, y además el try está mal porque la variable dos fue declarada afuera y no se asegura que se cierre.}

b. No es posible, Arrays.asList(new int[]{...}) devuelve una lista con un único elemento que es un array de int, NO una lista de Integer.
Además, ArrayList guarda un array de tipos primitivos (int[]), que no es serializable directamente en la forma que esperás.

c. ObjectInputStream entrada = new ObjectInputStream(new FileInputStream("figura.obj"));
Figura figuraLeida = (Figura) entrada.readObject();
entrada.close();

```
d) public class Escribir {
    public static void main(String[] args) {
        String saludo = "Hola";
        try {
            File archivo = new File("texto.txt");
            // → Crear objeto de archivo de escritura de bytes
            FileOutputStream escribir = new FileOutputStream(archivo);
            escribir.write(saludo.getBytes()); // convierte el String a bytes y los escribe
            escribir.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```