Presented By:
Shakeel Ahmad
Roll No: #F24-720
Section: 'E'

Date: 07-MAY-2025

Serializable

`Serializable` is a **marker interface** in Java (it has no methods) that allows objects of a class to be converted into a **byte stream** (serialization) and later reconstructed back into objects (deserialization).

Key Points:

1. **Purpose**:
   - Enables **saving objects to files** (persistence).
   - Allows **sending objects over a network** (e.g., RMI, sockets).
   - Used in **caching** and **deep copying** objects.
2. **How It Works**:
   - When you implement `Serializable`, Java's serialization mechanism handles the conversion of objects to bytes (and vice versa) automatically.

Overriding.java | Inheritance_Class.java | Association.java | Aggregation.java | Composition.java | Seri.java

```java
import java.io.*;

public class Seri implements Serializable{
    int rno;
    String name;

    public static void main(String[] args) {

        Seri s1 = new Seri();
        s1.rno = 23;
        s1.name = "Shakeel is an intelligent boy";
        System.out.println(s1.rno + s1.name);

        try (
                ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("RemoteStub.txt"))
            ){
             oos.writeObject(s1); // Write the object to file
                System.out.println("Object serialized successfully!");
            } catch (IOException e) {
                e.printStackTrace();
            }
```

Explanation

1. `try`

- **Meaning:** Starts a block of code where Java will watch for errors (exceptions).
- **Why?** If something goes wrong (e.g., file not found), Java jumps to the `catch` block.
- **With `( ... )`:** This is `try-with-resources`—Java will auto-close anything inside `( )` when done.

2. `ObjectOutputStream oos`

- **`ObjectOutputStream`:** A Java class that converts objects into bytes (serialization).
- **`oos`:** A variable name (short for "Object Output Stream"). You could name it anything, like `outputStream`.

Explanation

5. `new FileOutputStream("person.ser")`

- **`new`:** Creates a new `FileOutputStream` object.
- **`FileOutputStream`:** A class that writes raw bytes to a file (here, `person.ser`).
- **`"person.ser"`:** The filename where the serialized object will be saved (`.ser` is a common extension for serialized files).

6. `))`

- Closes the nested parentheses for `FileOutputStream` and `ObjectOutputStream`.

7. `{`

- Starts the `try` block where you write code to serialize objects (e.g., `oos.writeObject(myObject);`).

Deserialization

Deserialization is the reverse process of **serialization**. It converts a **byte stream** (from a file, network, etc.) back into a **Java object** in memory

- **Deserialization** reconstructs an object from its serialized (binary) form.
- It restores the object's state (fields, data) exactly as it was when serialized.

2. How It Works

- Uses `ObjectInputStream` to read bytes from a file/network.
- Reconstructs the object using metadata stored during serialization.

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- Abstraction
- Acess_Modifier
- College_wallah
- HellowDemo
- inheritance
- Mudassir
- Pen
- Polymorphism
- practice
- Pre_java_oop
- Pre_java_OOP
- Relationships
- Student

Overriding.java | Inheritance_Class.java | Association.java | Aggregation.java | Composition.java | Seri.java

```java
import java.io.*;
public class Seri implements Serializable{
    int rno;
    String name;

    public static void main(String[] args) {

        Seri s1 = new Seri();
        s1.rno = 23;
        s1.name = "Shakeel is an intelligent boy";
        System.out.println(s1.rno + s1.name);

        try (
                ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("RemoteStub.txt"))
            ){
             oos.writeObject(s1); // Write the object to file
                System.out.println("Object serialized successfully!");
            } catch (IOException e) {
                e.printStackTrace();
            }

          try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("RemoteStub.txt"))) {
                // Step 1: Read the object from file
                Seri deserializedObj = (Seri) ois.readObject(); // Explicit type casting

                // Step 2: Print the deserialized data
                System.out.println("After Deserialization: " + deserializedObj.rno + " " + deserializedObj.name);
            } catch (IOException | ClassNotFoundException e) {
                e.printStackTrace();
            }
        }
    }
```

Writable | Smart Insert | 17 : 62 : 451 | 270M of 308M

Explanation Of Deserializable

Key Improvements:

1. **Added `Serializable`**: Your class correctly implements `Serializable`.
2. **Completed Serialization**:
   - Used `oos.writeObject(s1)` to save the object.
3. **Added Deserialization**:
   - Reads the object back using `ObjectInputStream`.
4. **Error Handling**:
   - Added proper exception handling for both operations.
5. **Clear Output**:
   - Added descriptive print statements to track the process.

How It Works:

1. **Serialization** converts `s1` into bytes and saves to "RemoteStub.txt".
2. **Deserialization** reconstructs the object from the file.
3. The `try-with-resources` blocks automatically close the streams.

Output

Overriding.java | Inheritance_Class.java | Association.java | Aggregation.java | Composition.java | Seri

```java
import java.io.*;
public class Seri implements Serializable{
    int rno;
    String name;

    public static void main(String[] args) {

        Seri s1 = new Seri();
        s1.rno = 23;
        s1.name = "Shakeel is an intelligent boy";
        System.out.println(s1.rno + s1.name);

        try (
                ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("RemoteStub.txt"))
            ){
             oos.writeObject(s1); // Write the object to file
                System.out.println("Object serialized successfully!");
            } catch (IOException e) {
                e.printStackTrace();
            }
```

Console

```
<terminated> Seri [Java Application] /home/shakeel345sh5/eclipse/plugins/org.eclipse.justj.openjdk.h
23Shakeel is an intelligent boy
Object serialized successfully!
After Deserialization: 23 Shakeel is an intelligent boy
```

File Location

Using Linux we can find the file in which our java object is present.

shakeel345sh5@penguin: ~/eclipse | shakeel345sh5@penguin: ~/eclipse

```
shakeel345sh5@penguin:~/eclipse$ ls
artifacts.xml  configuration  dropins  eclipse  eclipse.ini  features  icon.xpm  p2  plugins  readme
shakeel345sh5@penguin:~/eclipse$ cd
shakeel345sh5@penguin:~$ ls
 archive-key.asc         c++                 CV                                                                                  eclipse-dsl-2024-12-R-linux-gtk-x86_64.tar.gz  'Git Demo'     mariadb.png   Telegram_User_Number_Finder
 backup.sql              calculator          'download.php?file=%2Foomph%2Fepp%2F2023-09%2FR%2Feclipse-inst-jre-linux64.tar.gz'   eclipse-workspace                              hellow.js      msa           vs
 backup_with_users.sql  'C++ college wallah'  eclipse                                                                            'file handling'                                'Html + Css'   python
shakeel345sh5@penguin:~$ cd eclipse-workspace
shakeel345sh5@penguin:~/eclipse-workspace$ ls
Abstraction  Acess_Modifier  College_wallah  College_Wallah  Firstjavaproject  Hellow  HellowDemo  inheritance  Mudassir  Pen  Polymorphism  practice  Pre_java_oop  Pre_java_OOP  Relationships  Student
shakeel345sh5@penguin:~/eclipse-workspace$ cd Relationships
shakeel345sh5@penguin:~/eclipse-workspace/Relationships$ ls
bin  RemoteStub.txt  src
shakeel345sh5@penguin:~/eclipse-workspace/Relationships$ cat RemoteStub
cat: RemoteStub: No such file or directory
shakeel345sh5@penguin:~/eclipse-workspace/Relationships$ xxd RemoteStub.txt
00000000: aced 0005 7372 0004 5365 7269 1371 33e7  ....sr..Seri.q3.
00000010: 3391 baf5 0200 0249 0003 726e 6f4c 0004  3......I..rnoL..
00000020: 6e61 6d65 7400 124c 6a61 7661 2f6c 616e  namet..Ljava/lan
00000030: 672f 5374 7269 6e67 3b78 7000 0000 1774  g/String;xp....t
00000040: 001d 5368 616b 6565 6c20 6973 2061 6e20  ..Shakeel is an
00000050: 696e 7465 6c6c 6967 656e 7420 626f 79    intelligent boy
shakeel345sh5@penguin:~/eclipse-workspace/Relationships$
```

Explanation

1. **Default Location**: Project folder in `eclipse-workspace`.
2. **Find It**: Use `ls` or Eclipse's refresh.
3. **Debug**: Check exceptions, use absolute paths, and verify case sensitivity.

View Raw Binary Data (Hex/Text Mix)

Use the `xxd` (hex dump) or `od` (octal dump) command to inspect the binary content:

Deserialize & Read It Back in Java

The best way to "open" the file is to **deserialize it** using Java (since it's not human-readable).
Run this code in the same directory: