SERIALIZATION

📌 1️⃣ What is Serialization?

🔹 Basic Definition

Serialization is the process of converting a Java object into a byte stream.

Object → Byte Stream → File / Network / Database

🔹 Why Serialization is Needed?

Objects live in **heap memory (RAM)**.

When:

- Program stops ❌
- JVM shuts down ❌
- System crashes ❌

Object is destroyed.

So we serialize to:

- Save object permanently
- Transfer object over network
- Store session objects
- Send object in distributed systems (RMI)

📌 2️⃣ What is Deserialization?

Deserialization is the reverse process.

Byte Stream → Object

We reconstruct the object from stored bytes.

📌 3️⃣ Memory Level Understanding

When object is created:

Student s = new Student(101, "Kalyan");

Memory structure:

Heap:

----

id = 101

name = "Kalyan"

After Serialization:

File contains:

[Binary representation of id + name]

After Deserialization:

Heap:

----

id = 101

name = "Kalyan"

(New object created)

⚠ Important:
Deserialization creates a **new object**, not the old one.

📌 4️⃣ How Java Supports Serialization?

Java provides:

1️⃣ **Serializable Interface**

2️⃣ **ObjectOutputStream**

3️⃣ **ObjectInputStream**

📌 5️⃣ Serializable Interface

import java.io.Serializable;

It is a **Marker Interface**.

What is Marker Interface?

An interface with **no methods**.

Example:

interface Serializable { }

Purpose:
✔ Just to inform JVM that object is eligible for serialization.

If class does not implement Serializable →
You get:

NotSerializableException

📌 6️⃣ ObjectOutputStream

Used for writing object into file.

ObjectOutputStream oos = new ObjectOutputStream(fos);

oos.writeObject(obj);

📌 7️⃣ ObjectInputStream

Used for reading object from file.

ObjectInputStream ois = new ObjectInputStream(fis);

obj = (ClassName) ois.readObject();

📌 8️⃣ Complete Flow of Serialization

1. Object created in heap
2. writeObject() called
3. JVM checks:
    - Is class Serializable?
4. Converts object state into byte stream
5. Writes bytes into file

📌 9️⃣ serialVersionUID (VERY IMPORTANT)

Every Serializable class has a version number.

If not declared manually:

JVM generates automatically.

Problem:
If class structure changes →
Deserialization fails with:

InvalidClassException

Solution:

Manually declare:

private static final long serialVersionUID = 1L;

This ensures compatibility.

📌 🔟 What Gets Serialized?

✓ Instance variables
✓ Non-static variables
✓ Non-transient variables

❌ What Does NOT Get Serialized?

❌ static variables
❌ transient variables
❌ Constructor
❌ Methods

📌 1️⃣ 1️⃣ transient Keyword

Used to prevent variable from being serialized.

Syntax:

transient String password;

Why use it?

- Security data
- Temporary data
- Derived values
- Cache values

After Deserialization

transient variables get:

Type	Default Value
int	0
String	null
boolean	false

📌 1️⃣ 2️⃣ static Variable in Serialization

static belongs to class, not object.

So it is NOT saved.

During deserialization:
It takes current class value.

📌 1️⃣ 3️⃣ final Variable

final variables are serialized normally.

Because they belong to object.

📌 1️⃣ 4️⃣ Inheritance & Serialization

Case 1:
Parent NOT Serializable
Child Serializable

👉 Parent constructor runs during deserialization.

Case 2:
Parent Serializable
👉 Parent constructor NOT called.

📌 1️⃣ 5️⃣ Custom Serialization

You can override default mechanism using:

private void writeObject(ObjectOutputStream oos)

private void readObject(ObjectInputStream ois)

Used when:

- Want to encrypt data
- Want to modify data before saving
- Want custom control

📌 1️⃣ 6️⃣ Externalization

Advanced form of Serialization.

Instead of Serializable → use:

Externalizable

Key Differences

Serializable	Externalizable
Automatic	Manual
JVM controls	Programmer controls
Easy	Complex
Slower	Faster

Methods Required

writeExternal()

readExternal()

Important Rule

Externalizable class MUST have:

public no-arg constructor

Otherwise fails.

📌 1️⃣ 7️⃣ Internal Working Difference

Serializable

JVM internally saves:

- Class metadata
- Object state
- Variable values

Externalizable

JVM does:

1. Call no-arg constructor
2. Call readExternal()

You manually rebuild object.

📌 1️⃣8️⃣ Constructor Behavior

During Deserialization:

Serializable:
❌ Constructor does NOT run

Externalizable:
✓ Constructor runs

📌 1️⃣9️⃣ Common Exceptions

1️⃣ NotSerializableException

If class does not implement Serializable

2️⃣ InvalidClassException

serialVersionUID mismatch

3️⃣ EOFException

Reading beyond file

4️⃣ ClassNotFoundException

Class not found during deserialization

📌 2️⃣0️⃣ Real Time Usage

✓ RMI
✓ HTTP Session
✓ Distributed systems
✓ Caching
✓ File storage
✓ Saving game state

📌 2️⃣1️⃣ Interview Important Questions

Q1: Why Serializable is marker interface?

Because no methods needed, just tagging mechanism.

Q2: Why static not serialized?

Because it belongs to class, not object.

Q3: Why transient used?

To prevent sensitive data from being saved.

Q4: Why Externalizable needs no-arg constructor?

Because JVM first creates object using no-arg constructor before calling readExternal().

Q5: Is serialization secure?

No. Data is stored in binary but not encrypted.

📌 2️⃣ 2️⃣ Serialization vs Externalization Final Comparison

Feature	Serializable	Externalizable
Control	JVM	Developer
Performance	Moderate	Better
Ease	Easy	Hard
Constructor	Not required	Required
Speed	Slower	Faster
Flexibility	Low	High

📌 2️⃣ 3️⃣ Complete Concept Map

```
Serialization
  |
  |-- Serializable
  |      |-- serialVersionUID
  |      |-- transient
  |      |-- static behavior
  |
  |-- Custom Serialization
  |
  |-- Externalizable
```

🎯 Final Master Summary

Serialization:
Convert object → byte stream

Deserialization:
Convert byte stream → object

transient:
Prevent variable from saving

static:
Never serialized

final:
Serialized

serialVersionUID:
Version control for class

Externalizable:
Full manual control