**Key points to explain:**

1. **Enum Singleton nature**
   * An enum in Java guarantees only **one instance per constant** (like INSTANCE here).
   * The JVM itself ensures that deserialization does **not** create a new object, but instead returns the **same instance** that was created at class loading time.
2. **Serialization of Enums**
   * When we serialize an enum, **only the name of the constant** (e.g., "INSTANCE") is written to the stream.
   * During deserialization, the JVM looks up that name in the enum class and returns the **existing enum constant**.
   * That’s why s1 == s2 is always true.
3. **Why the value resets**
   * Non-transient fields (**value** in our case) are **not serialized** for enums in the usual sense.
   * The enum constant is reconstructed by name, **not by copying fields from the stream**.
   * Hence, after deserialization, we don’t get back the **value** we had set (99). Instead, we see the **current state of the already-loaded enum constant** in memory (which was reset to 0 before deserialization).

**Summary:**

*"When we serialize an enum, Java does not store its fields. It just writes the name of the constant. During deserialization, the JVM doesn’t make a new object — it simply gives you back the same enum constant that already exists. That’s why s1 == s2 is true. But since the fields are not restored from the file, we only see the value that is currently inside the enum instance at runtime."*