

Introduction

Serialization is a converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. The serialization and deserialization process is platform-independent, it means you can serialize an object on one platform and deserialize it on a different platform.

Benefits

- 1) **Data Persistence:** Serialized objects can be saved to files or databases, allowing the state of an application to be preserved between sessions.
- 2) **Communication:** Serialized data can be transmitted over networks, facilitating distributed systems and remote method invocation (RMI).
- 3) **Deep Cloning:** Serialization provides a mechanism for creating deep copies of objects without manually copying each field.
- 4) **Interoperability:** Serialization allows for the easy exchange of data between different parts of a system or between different systems.
- 5) **State Capture:** Serialization can capture the state of an object at a specific point in time, aiding in logging and debugging complex applications by preserving the exact state of objects for later analysis.

Mechanism

Serialization Process

Serialization in Java involves converting an object's state into a byte stream, which can then be saved to a file, sent over a network, or stored in memory. This process captures the object's state so it can be recreated later.

- 1) Implementing Serializable Interface: For a class to be serializable, it must implement the `java.io.Serializable` interface. This is a marker interface, meaning it does not contain any methods but indicates that the class can be serialized.

```
public class Person implements Serializable { 4 usages new *
    private static final long serialVersionUID = 1L; no usages
    private String code; 4 usages
    private int id; 4 usages
```

- 2) Using ObjectOutputStream: To serialize an object, an `ObjectOutputStream` is used in conjunction with a `FileOutputStream`. The `ObjectOutputStream` writes the object's state to a byte stream, and the `FileOutputStream` writes this byte stream to a file.
- 3) Writing Objects: The `writeObject` method of `ObjectOutputStream` is used to serialize the object.

```
Person person = new Person( code: "Dinelka", id: 12345);

// Serialize object
try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream( name: "person.ser"))) {
    oos.writeObject(person);
    System.out.println("Serialization done: " + person);
} catch (IOException e) {
    e.printStackTrace();
}
```

Deserialization Process

Deserialization is the process of reconstructing an object from a byte stream, essentially reversing the serialization process. It restores the object's state from the byte stream.

- 1) Using `ObjectInputStream`: To deserialize an object, an `ObjectInputStream` is used with a `FileInputStream`. The `ObjectInputStream` reads the byte stream from the file and reconstructs the object.
- 2) Reading Objects: The `readObject` method of `ObjectInputStream` is used to deserialize the object.

```
// Deserialize object
try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("person.ser"))) {
    Person deserializedEmployee = (Person) ois.readObject();
    System.out.println("Deserialization done: " + deserializedEmployee);
} catch (IOException | ClassNotFoundException e) {
    e.printStackTrace();
}
```

SerialVersionUID Explained:

The `serialVersionUID` is a unique identifier for each `Serializable` class. This identifier is used during the deserialization process to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to serialization.

Version Control: The `serialVersionUID` helps in version control of `Serializable` classes. When a class structure is changed (for example, adding new fields), the `serialVersionUID` ensures that the class is compatible with its previous versions.

Deserialization Verification: During deserialization, Java's serialization mechanism checks that the `serialVersionUID` of the serialized object matches that of the corresponding class.

```
private static final long serialVersionUID = 1L; no usages
```

`private static final`: These keywords ensure that the `serialVersionUID` field is a constant, belongs to the class (not instances of the class), and cannot be modified.

`long serialVersionUID = 1L`: sets the `serialVersionUID` to a specific value (1 in this case). This value can be changed manually if the class undergoes incompatible changes.

See code snippet:

<https://github.com/dinelkalakshan40/Serialization-and-Deserialization.git>

Serialization and Deserialization Diagram:

