SerialVersionUID in Java

Difficulty Level : Medium • Last Updated : 31 Oct, 2020

Prerequisite : Serialization & Deserialization

The serialization at runtime associates with each serializable class a version number, called a serialVersionUID, which is used during deserialization to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to serialization.

Why so we use SerialVersionUID : SerialVersionUID is used to ensure that during deserialization the same class (that was used during serialize process) is loaded.

Example: Suppose a person who is in UK and another person who is in India, both are going to perform serialization and deserialization respectively. In this case to authenticate that the receiver who is in India is the authenticated person, JVM creates an Unique ID which is known as **SerialVersionUID**.

In most of the cases, serialization and deserialization both activities are done by a single person with the same system and same location. But in serialization, sender and receiver are not the same person i.e. the persons may be different, machine or system may be different and location must be different then SerialVersionUID comes in the picture. In serialization, both sender and receiver should have .class file at the time of beginning only i.e. the person who is going to do serialization and the person who is ready for deserialization should contain same .class file at the beginning time only.

Serialization : At the time of serialization, with every object sender side JVM will save a **Unique Identifier**. JVM is responsible to generate that unique ID based on the corresponding .class file which is present in the sender system.

Deserialization: At the time of deserialization, receiver side JVM will compare the unique ID associated with the Object with local class Unique ID i.e. JVM will also create a Unique ID based on the corresponding .class file which is present in the receiver system. If both unique ID matched then only deserialization will be performed. Otherwise we will get Runtime Exception saying **InvalidClassException**. This unique Identifier is nothing but **SerialVersionUID**.

Problem of depending on default SerialVersionUID generated by JVM :

- Both sender and receiver should use the same JVM with respect to platform and version also. Otherwise receiver unable to deserialize because of different SerialVersionUID.
- Both sender and receiver should use same .class file version. After serialization if there is any change in .class file at receiver side then receiver unable to deserialize.
- To generate SerialVersionUID internally JVM may use complex algorithm which may create performance problem.

We can solve the above problem by configuring our own SerialVersionUID. We can configure our own SerialVersionUID as follows:

```
private static final long SerialVersionUID=10l;
```

A class Geeks which contains two variable which are going to Serialize

```
    // User-defined SerialVersionUID
    private static final long SerialVersionUID = 10l;
    int i = 10;
    int j = 20;
}
```

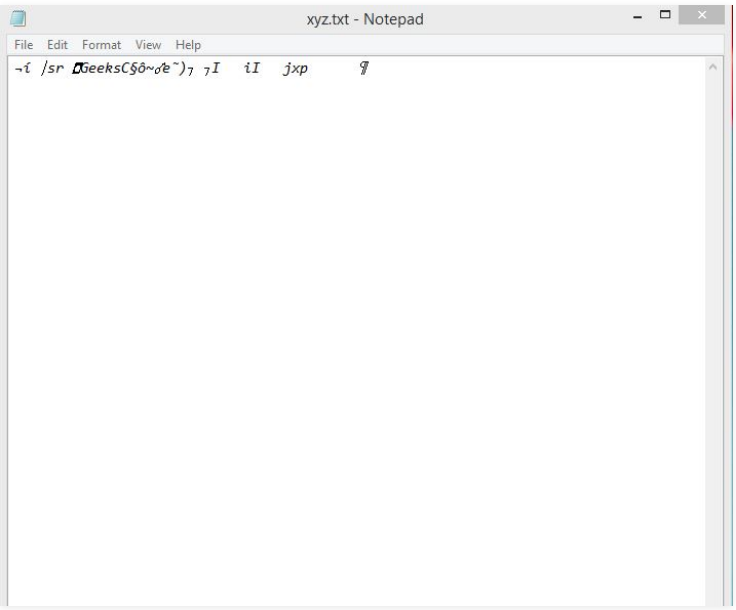
Program of Sender side which is going to Serialize object:

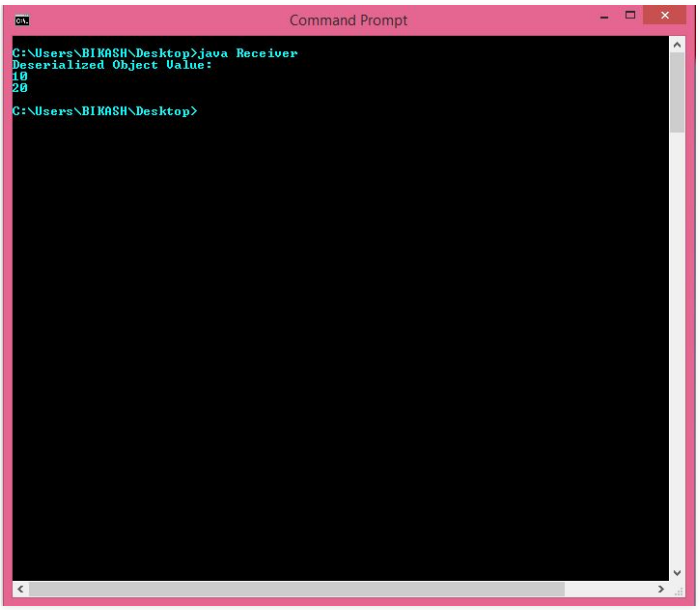
```
// Java program to illustrate
// implementation of User-defined
// SerialVersionUID
import java.io.*;
class Sender {
    public static void main(String[] args)
    {
        Geeks g = new Geeks();
        // Here xyz.txt is the file name where the object is
        // going to serialize
        FileOutputStream fos = new FileOutputStream("xyz.txt");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(g);
    }
}
```

Program of Receiver side which is going to deserialize

```
// Java program to illustrate
// implementation of User-defined
// SerialVersionUID
import java.io.*;
class Receiver {
    public static void main(String[] args)
    {
        // Here xyz.txt is the file name where the object is
        // going to Deserialized
        FileInputStream fis = new FileInputStream("xyz.txt");
        ObjectInputStream ois = new ObjectInputStream(fis);
        Geeks g1 = (Geeks)ois.readObject();
        System.out.println("Deserialized Object Value:" + g1.i + "..."+g1.j);
    }
}
```

we deserialize the Object:





In the above program, if we perform any change to the Geeks .class file at the receiver end . We dont get any problem at the time of deserialization. In this case sender and receiver not required to maintain same JVM versions.

Attention reader! Don't stop learning now. Get hold of all the important **Java Foundation** and Collections concepts with the **Fundamentals of Java and Java Collections Course** at a student-friendly price and become industry ready. To complete your preparation from learning a language to DS Algo and many more, please refer