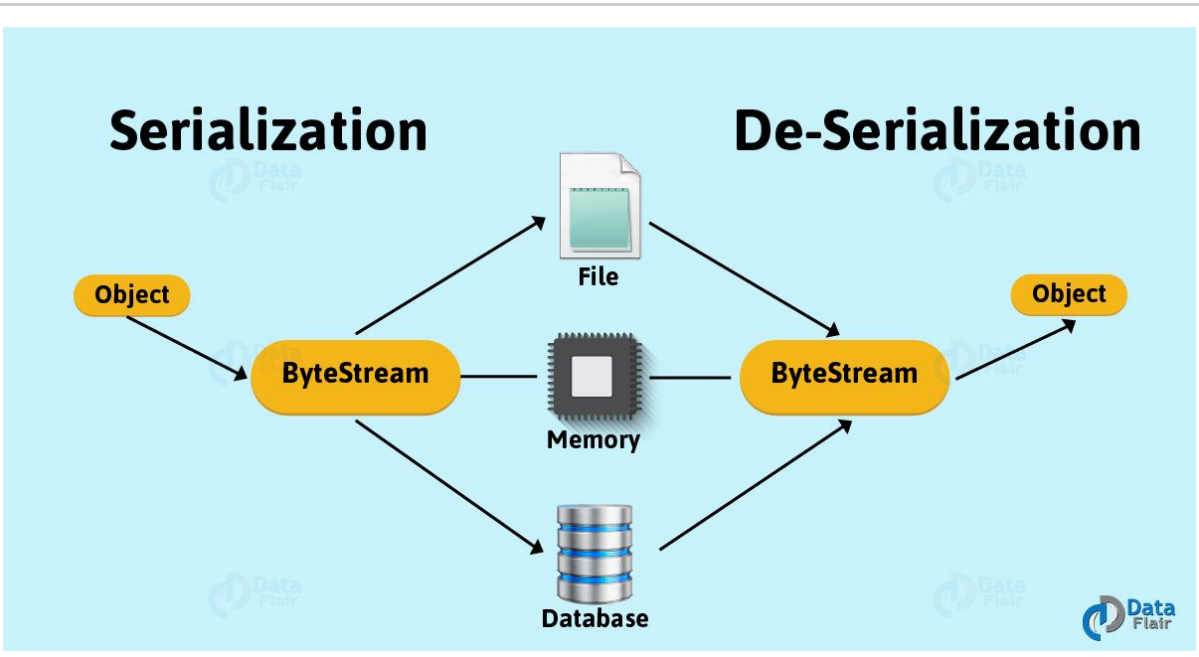
Serialization and Deserialization in Java

1. Serialization and Deserialization in Java

In this **tutorial for Java**, we are going to study the process of Java serialization and deserialization in Java, Serialization in java real-time examples, Deserialization in java with examples, and advantages and disadvantages of Serialization in Java and Deserialization in Java.

2. Introduction to Serialization and Deserialization in Java

The process of serialization in Java is a process in which the object's state is converted into a byte stream while deserialization is just the reverse of serialization in which we use the byte stream to convert into the original state of a **Java object**.



Serialization in Java and Deserialization in Java

The classes which can be serialized should have the implementation java.io.Serializable interface. Serializable in Java is used to mark classes, so that they may get the certain capability, it is basically marker **interface**.

Points to ponder

i. If serializable is implemented in a certain parent class then we don't need to implement it in the child class, but the reverse is not applicable.

ii. Nonstatic data members are stored via this method and not static and transient data members.

iii.A **constructor** for an object is never called while deserialization.

Read more about **Java Classes**

Example –

```
1. class A implements Serializable
2. {
3. // B also implements Serializable interface.
4. B ob=new B();
5. }
```

3. SerialVersionUID

A serializable class associates itself with a SerialVersionUID while the Java serialization runtime. This is used during Java Deserialization to verify that the object loaded by the sender and receiver of a serialized object is compatible with respect to Java serialization if the UID is different it will result in InvalidClassException. A class which can be serialized can declare its own UID by declaring a field name which should be of type long, static and final. If it doesn't explicitly declare a seialVersionUID then a default one will be automatically be created by runtime based on various aspects of class as described by the Java Object Serialization Specification. It is recommended to specify UID as any change in information can highly change the results and also to declare it as private so it is not inherited.

Read about **Inheritance in Java**

Serialver

This tool is to get serialVersionUID of Java classes, it comes with JDK.

4. Advantages and Disadvantages

Following are the Advantages and Disadvantages of Deserialization and Serialization in Java

i. Serialization is very easy to use and also a serialized stream supports secure Java computing as it can be authenticated, encrypted and compressed. It supports coherent versioning and also supports flexible to allow the gradual evolution of the program. It can also be used to support exchange in libraries between Java and C++.

ii. The disadvantages being, it cannot be used with large sized objects, also It offers overheads, this in whole delays the process of garbage collection as large objects contain large memory.

5. Examples of Serialization and Deserialization in Java

a. Example 1

```
1. // Java code for serialization and deserialization of a Java object
2. import java.io.*;
3. class Demo implements java.io.Serializable
4. {
5. public int a;
6. public String b;
7. // Default constructor
8. public Demo(int a, String b)
9. {
10. this.a = a;
11. this.b = b;
12. }
13. }
14. class Test
15. {
16. public static void main(String[] args)
17. {
18. Demo object = new Demo(1, "geeksforgeeks");
19. String filename = "file.ser";
20. // Serialization
21. try
22. {
23. //Saving of object in a file
24. FileOutputStream file = new FileOutputStream(filename);
25. ObjectOutputStream out = new ObjectOutputStream(file);
26. // Method for serialization of object
27. out.writeObject(object);
28. out.close();
29. file.close();
30. System.out.println("Object has been serialized");
31. }
32. catch(IOException ex)
33. {
34. System.out.println("IOException is caught");
35. }
36. Demo object1 = null;
37. // Deserialization
38. try
```

```
{
// Reading the object from a file
FileInputStream file = new FileInputStream(filename);
ObjectInputStream in = new ObjectInputStream(file);
// Method for deserialization of object
object1 = (Demo)in.readObject();
in.close();
file.close();
System.out.println("Object has been deserialized ");
System.out.println("a = " + object1.a);
System.out.println("b = " + object1.b);
}
catch(IOException ex)
{
System.out.println("IOException is caught");
}
catch(ClassNotFoundException ex)
{
System.out.println("ClassNotFoundException is caught");
}
}
}
```

b. Example 2

```
// Java code for serialization and deserialization of a Java object
import java.io.*;
class Emp implements Serializable
{
private static final long serialversionUID =
129348938L;
transient int a;
static int b;
String name;
int age;
// Default constructor
public Emp(String name, int age, int a, int b)
{
this.name = name;
this.age = age;
this.a = a;
this.b = b;
}
}
public class SerialExample
{
public static void printdata(Emp object1)
```

```
23. {
24. System.out.println("name = " + object1.name);
25. System.out.println("age = " + object1.age);
26. System.out.println("a = " + object1.a);
27. System.out.println("b = " + object1.b);
28. }
29. public static void main(String[] args)
30. {
31. Emp object = new Emp("ab", 20, 2, 1000);
32. String filename = "shubham.txt";
33. // Serialization
34. try
35. {
36. // Saving of object in a file
37. FileOutputStream file = new FileOutputStream
38. (filename);
39. ObjectOutputStream out = new ObjectOutputStream
40. (file);
41. // Method for serialization of object
42. out.writeObject(object);
43. out.close();
44. file.close();
45. System.out.println("Object has been serialized\n"
46. + "Data before Deserialization.");
47. printdata(object);
48. // value of static variable changed
49. object.b = 2000;
50. }
51. catch(IOException ex)
52. {
53. System.out.println("IOException is caught");
54. }
55. object = null;
56. // Deserialization
57. try
58. {
59. // Reading the object from a file
60. FileInputStream file = new FileInputStream
61. (filename);
62. ObjectInputStream in = new ObjectInputStream
63. (file);
64. // Method for deserialization of object
65. object = (Emp)in.readObject();
66. in.close();
67. file.close();
68. System.out.println("Object has been deserialized\n"
```

```
69.  + "Data after Deserialization.");
70.  printdata(object);
71.  // System.out.println("z = " + object1.z);
72.  }
73.  catch(IOException ex)
74.  {
75.  System.out.println("IOException is caught");
76.  }
77.  catch(ClassNotFoundException ex)
78.  {
79.  System.out.println("ClassNotFoundException" +
80.  " is caught");
81.  }
82.  }
83.  }
```