[**serialVersionUID in java Serialization**](http://www.java2blog.com/2013/03/serialversionuid-in-java-serialization.html)

**serialVersionUID** is used to ensure that same class(That was used during Serialization) is loaded during Deserialization. **serialVersionUID** is used for version control of object.  
If you have used Serialization then You might have seen  **serialVersionUID** because whenever you implement **Serializable** interface your IDE will give you warning.

[[http://2.bp.blogspot.com/-X70ijuuGXsE/UTx5E6DwjTI/AAAAAAAAA4M/teKF_b5yI4w/s1600/SerialVersionUIDWarning.gif](http://2.bp.blogspot.com/-X70ijuuGXsE/UTx5E6DwjTI/AAAAAAAAA4M/teKF_b5yI4w/s1600/SerialVersionUIDWarning.gif)](http://2.bp.blogspot.com/-X70ijuuGXsE/UTx5E6DwjTI/AAAAAAAAA4M/teKF_b5yI4w/s1600/SerialVersionUIDWarning.gif)

**Serialversionuid Syntax:**

As per java docs

[view plainprint?](http://www.java2blog.com/2013/03/serialversionuid-in-java-serialization.html)

1. ANY-ACCESS-MODIFIER **static** **final** **long** serialVersionUID = 42L;

serialVersionUID must be Static and final.You can assign any number to it.  
Lets see an example:  
Create Employee.java in src->org.arpit.javapostsforlearning  
  
**1.Employee.java**

[view plainprint?](http://www.java2blog.com/2013/03/serialversionuid-in-java-serialization.html)

1. **package** org.arpit.javapostsforlearning;
2. **import** java.io.Serializable;
3. **public** **class** Employee **implements** Serializable{
5. **private** **static** **final** **long** serialVersionUID = 1L;
7. **int** employeeId;
8. String employeeName;
9. String department;
11. **public** **int** getEmployeeId() {
12. **return** employeeId;
13. }
14. **public** **void** setEmployeeId(**int** employeeId) {
15. **this**.employeeId = employeeId;
16. }
17. **public** String getEmployeeName() {
18. **return** employeeName;
19. }
20. **public** **void** setEmployeeName(String employeeName) {
21. **this**.employeeName = employeeName;
22. }
23. **public** String getDepartment() {
24. **return** department;
25. }
26. **public** **void** setDepartment(String department) {
27. **this**.department = department;
28. }
29. }

Create SerializeMain.java in src->org.arpit.javapostsforlearning

**2.SerializeMain.java**

[view plain](http://www.java2blog.com/2013/03/serialversionuid-in-java-serialization.html)

**package** org.arpit.javapostsforlearning;

1. **import** java.io.FileOutputStream;
2. **import** java.io.IOException;
3. **import** java.io.ObjectOutputStream;
4. **public** **class** SerializeMain {
6. /\*\*
7. \* @author Arpit Mandliya
8. \*/
9. **public** **static** **void** main(String[] args) {
11. Employee emp = **new** Employee();
12. emp.setEmployeeId(101);
13. emp.setEmployeeName("Arpit");
14. emp.setDepartment("CS");
15. **try**
16. {
17. FileOutputStream fileOut = **new** FileOutputStream("employee.ser");
18. ObjectOutputStream outStream = **new** ObjectOutputStream(fileOut);
19. outStream.writeObject(emp);
20. outStream.close();
21. fileOut.close();
22. }**catch**(IOException i)
23. {
24. i.printStackTrace();
25. }
26. }
27. }

Create DeserializeMain.java in src->org.arpit.javapostsforlearning

**3.DeserializeMain.java**

[view plainprint?](http://www.java2blog.com/2013/03/serialversionuid-in-java-serialization.html)

1. **package** org.arpit.javapostsforlearning;
2. **import** java.io.IOException;
3. **import** java.io.ObjectInputStream;
5. **public** **class** DeserializeMain {
6. /\*\*
7. \* @author Arpit Mandliya
8. \*/
9. **public** **static** **void** main(String[] args) {
11. Employee emp = **null**;
12. **try**
13. {
14. FileInputStream fileIn =**new** FileInputStream("employee.ser");
15. ObjectInputStream in = **new** ObjectInputStream(fileIn);
16. emp = (Employee) in.readObject();
17. in.close();
18. fileIn.close();
19. }**catch**(IOException i)
20. {
21. i.printStackTrace();
22. **return**;
23. }**catch**(ClassNotFoundException c)
24. {
25. System.out.println("Employee class not found");
26. c.printStackTrace();
27. **return**;
28. }
29. System.out.println("Deserialized Employee...");
30. System.out.println("Emp id: " + emp.getEmployeeId());
31. System.out.println("Name: " + emp.getEmployeeName());
32. System.out.println("Department: " + emp.getDepartment());
33. }
34. }

**4.Run it:**

First run SerializeMain.java then DeserializeMain.java and you will get following output:

[view plainprint?](http://www.java2blog.com/2013/03/serialversionuid-in-java-serialization.html)

1. Deserialized Employee...
2. Emp id: 101
3. Name: Arpit
4. Department: CS

So when you run program,it was completed successfully and employee.ser has been created on disk.If you again run DeserializeMain.java,it will again run successfully. Now change value of variable serial to

[view plainprint?](http://www.java2blog.com/2013/03/serialversionuid-in-java-serialization.html)

1. **private** **static** **final** **long** serialVersionUID = 2L;

and if you now run DeserializeMain.java it will give you following error.

[view plainprint?](http://www.java2blog.com/2013/03/serialversionuid-in-java-serialization.html)

1. java.io.InvalidClassException: org.arpit.javapostsforlearning.Employee; local **class** incompatible: stream classdesc serialVersionUID = 1, local **class** serialVersionUID = 2

So here during deserialization,we got error.It complained about Serialvesionuid being changed.But how does it know? because serialversionuid is a static variable and we know that "We can not serialize static variables".How does it store  serialversionuid? yes ,there is exception.Inspite of serialversionuid being static,it get serialized.So ObjectOutputStream writes every time to output stream and ObjectInputStream reads it back and if it does not have same values as in current version of class then it throw InvalidClassException.

**Why serialversionuid is required?**

In real time,It is possible that you have serialized a object in a file and you deserialized it after few months on different JVM.In between serialization and deserialization class declaration has been changed.So it is a good idea to maintain version system and serialversionid does exactly same thing.It checks if you are deserializing same object which you have serialized.

**Best Practices:**

Java docs says:

"the default serialVersionUID computation is highly sensitive to class details that may vary depending on compiler implementations, and can thus result in unexpected InvalidClassExceptions during deserialization".

So it says you must declare serialVersionUID because it give us more control.for e.g. Default rules for generating serialVersionUID can be too strict in some cases. For example when the visibility of a field changes, the serialVersionUID changes too. or sometimes you just want to forbid deserialization of old serialized object then you can just change serialVersionUID.

Is this enough?.No,you must not only declare it but also maintain it.So most important part is maintaining  serialVersionUID otherwise every thing will run without any exceptions.You should change serialVersionUID when there is some change in the definition of data stored in the class for example data type of field is changed.