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

Java provides mechanism called serialization to persists java objects in a form of ordered or sequence of bytes that includes the object's data as well as information about the object's type and the types of data stored in the object.  
  
So if we need to serialize any object then it can be read and deserialize it using object's type and other information so we can retrieve original object.  
  
Classes ObjectInputStream and ObjectOutputStream are high-level streams that contain the methods for serializing and deserializing an object.   
ObjectOutputStream has many method for serializing object but commonly used method is

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

1. **private** **void** writeObject(ObjectOutputStream os) **throws** IOException
2. {
4. }

Similarly ObjectInputStream has

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

1. **private** **void** readObject(ObjectInputStream is) **throws** IOException, ClassNotFoundException
2. {
4. }

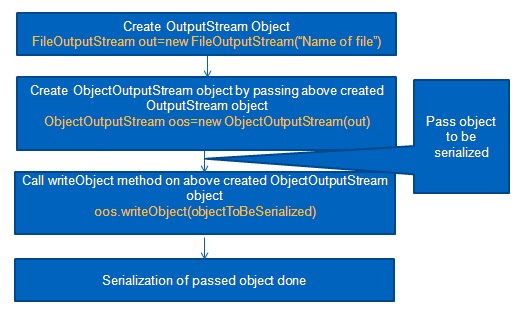
**Need of Serialization?**

Serialization is usually used when there is need to send your data over network or to store in files. By data I mean objects and not text.  
  
Now the problem is your Network infrastructure and your Hard disk are hardware components that understand bits and bytes but not Java objects.  
  
Serialization is the translation of Java object's values/states to bytes to send it over network or to save it.On other hand,Deserialization is conversion of byte code to corresponding java objects.

**Concept of serialVersionUID :**

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.You can read more at [serialVersionUID in java serialization](http://javapostsforlearning.blogspot.in/2013/03/serialversionuid-in-java-serialization.html)

**For Serialization:**  
steps are :

[[](http://4.bp.blogspot.com/-KSZjl6ugooE/UTuG7bBQp0I/AAAAAAAAA3U/zUvJUoDN0VU/s1600/SerializationMechanism.bmp)](http://4.bp.blogspot.com/-KSZjl6ugooE/UTuG7bBQp0I/AAAAAAAAA3U/zUvJUoDN0VU/s1600/SerializationMechanism.bmp)

Lets take an example:  
Create Employee.java in src->org.arpit.javapostsforlearning  
  
**1.Employee.java**

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

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

As you can see above,if you want to serialize any class then **it must implement Serializable interface which is marker interface.**  
Marker interface in Java is interfaces with no field or methods or in simple word empty interface in java is called marker interface  
Create SerializeMain.java in src->org.arpit.javapostsforlearning

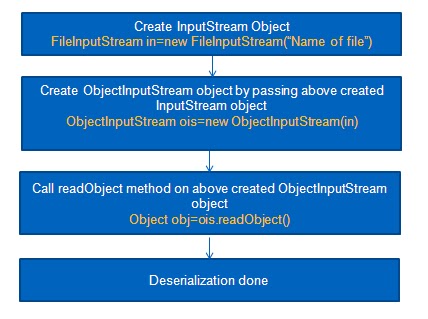
**2.SerializeMain.java**

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

1. **package** org.arpit.javapostsforlearning;
2. **import** java.io.FileOutputStream;
3. **import** java.io.IOException;
4. **import** java.io.ObjectOutputStream;
5. **public** **class** SerializeMain {
7. /\*\*
8. \* @author Arpit Mandliya
9. \*/
10. **public** **static** **void** main(String[] args) {
12. Employee emp = **new** Employee();
13. emp.setEmployeeId(101);
14. emp.setEmployeeName("Arpit");
15. emp.setDepartment("CS");
16. **try**
17. {
18. FileOutputStream fileOut = **new** FileOutputStream("employee.ser");
19. ObjectOutputStream outStream = **new** ObjectOutputStream(fileOut);
20. outStream.writeObject(emp);
21. outStream.close();
22. fileOut.close();
23. }**catch**(IOException i)
24. {
25. i.printStackTrace();
26. }
27. }
28. }

**For Deserialization:**

Steps are:

[[](http://2.bp.blogspot.com/-D_AXOy2reSk/UTuHu7fZmeI/AAAAAAAAA3c/FShmcMaROFM/s1600/DeserializationMechanism.bmp)](http://2.bp.blogspot.com/-D_AXOy2reSk/UTuHu7fZmeI/AAAAAAAAA3c/FShmcMaROFM/s1600/DeserializationMechanism.bmp)

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

**3.DeserializeMain.java**

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

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

So we have serialize an employee object and then deserialized it.It seems very simple but it can be very complex when reference object,inheritance come into the picture.So we will see different cases one by one and how we can apply serialization in different scenarios.

**Case 1-What if an object has a reference to other objects**

We have seen very simple case of serialization,now what if it also a reference to other objects.How will it serialized then? will reference object will also get serialized?.Yes,You don't have to explicitly serialize reference objects.When you serialize any object and if it contain any other object reference then Java serialization serialize that object's entire object graph.  
For example:Lets say,Employee now has reference to address object and Address can have reference to some other object(e.g.Home) then when you serialize Employee object all other reference objects such as address and home will be automatically serialized. Lets create Address class and add object of Address as a reference to above employee class.  
  
**Employee.java:**

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

1. **package** org.arpit.javapostsforlearning;
2. **import** java.io.Serializable;
4. **public** **class** Employee **implements** Serializable{
6. **private** **static** **final** **long** serialVersionUID = 1L;
7. **int** employeeId;
8. String employeeName;
9. String department;
10. Address address;
12. **public** **int** getEmployeeId() {
13. **return** employeeId;
14. }
15. **public** **void** setEmployeeId(**int** employeeId) {
16. **this**.employeeId = employeeId;
17. }
18. **public** String getEmployeeName() {
19. **return** employeeName;
20. }
21. **public** **void** setEmployeeName(String employeeName) {
22. **this**.employeeName = employeeName;
23. }
24. **public** String getDepartment() {
25. **return** department;
26. }
27. **public** **void** setDepartment(String department) {
28. **this**.department = department;
29. }
30. **public** Address getAddress() {
31. **return** address;
32. }
33. **public** **void** setAddress(Address address) {
34. **this**.address = address;
35. }
36. }

Create Address.java in org.arpit.javapostsforlearning  
**Address.java:**

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

1. **package** org.arpit.javapostsforlearning;
2. **public** **class** Address {
4. **int** homeNo;
5. String street;
6. String city;
7. **public** Address(**int** homeNo, String street, String city) {
8. **super**();
9. **this**.homeNo = homeNo;
10. **this**.street = street;
11. **this**.city = city;
12. }
13. **public** **int** getHomeNo() {
14. **return** homeNo;
15. }
16. **public** **void** setHomeNo(**int** homeNo) {
17. **this**.homeNo = homeNo;
18. }
19. **public** String getStreet() {
20. **return** street;
21. }
22. **public** **void** setStreet(String street) {
23. **this**.street = street;
24. }
25. **public** String getCity() {
26. **return** city;
27. }
28. **public** **void** setCity(String city) {
29. **this**.city = city;
30. }
31. }

Create SerializeDeserializeMain.java in org.arpit.javapostsforlearning

**SerializeDeserializeMain.java:**

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

1. **package** org.arpit.javapostsforlearning;
2. **import** java.io.FileInputStream;
3. **import** java.io.FileOutputStream;
4. **import** java.io.IOException;
5. **import** java.io.ObjectInputStream;
6. **import** java.io.ObjectOutputStream;
8. **public** **class** SerializeDeserializeMain {
9. /\*\*
10. \* @author Arpit Mandliya
11. \*/
12. **public** **static** **void** main(String[] args) {
14. Employee emp = **new** Employee();
15. emp.setEmployeeId(101);
16. emp.setEmployeeName("Arpit");
17. emp.setDepartment("CS");
18. Address address=**new** Address(88,"MG road","Pune");
19. emp.setAddress(address);
20. //Serialize
21. **try**
22. {
23. FileOutputStream fileOut = **new** FileOutputStream("employee.ser");
24. ObjectOutputStream outStream = **new** ObjectOutputStream(fileOut);
25. outStream.writeObject(emp);
26. outStream.close();
27. fileOut.close();
28. }**catch**(IOException i)
29. {
30. i.printStackTrace();
31. }
33. //Deserialize
34. emp = **null**;
35. **try**
36. {
37. FileInputStream fileIn =**new** FileInputStream("employee.ser");
38. ObjectInputStream in = **new** ObjectInputStream(fileIn);
39. emp = (Employee) in.readObject();
40. in.close();
41. fileIn.close();
42. }**catch**(IOException i)
43. {
44. i.printStackTrace();
45. **return**;
46. }**catch**(ClassNotFoundException c)
47. {
48. System.out.println("Employee class not found");
49. c.printStackTrace();
50. **return**;
51. }
52. System.out.println("Deserialized Employee...");
53. System.out.println("Emp id: " + emp.getEmployeeId());
54. System.out.println("Name: " + emp.getEmployeeName());
55. System.out.println("Department: " + emp.getDepartment());
56. address=emp.getAddress();
57. System.out.println("City :"+address.getCity());
58. }
59. }

**Run it :**

When you run SerializeDeserializeMain.java.You will get following output

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

1. java.io.NotSerializableException: org.arpit.javapostsforlearning.Address
2. at java.io.ObjectOutputStream.writeObject0(Unknown Source)
3. at java.io.ObjectOutputStream.defaultWriteFields(Unknown Source)
4. at java.io.ObjectOutputStream.writeSerialData(Unknown Source)
5. at java.io.ObjectOutputStream.writeOrdinaryObject(Unknown Source)
6. at java.io.ObjectOutputStream.writeObject0(Unknown Source)
7. at java.io.ObjectOutputStream.writeObject(Unknown Source)

We got exception what went wrong.I forgot to mention,Address class must also be serializable.So you have to make Address serializable by implement serialzable interface.

**Address.java:**

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

1. **import** java.io.Serializable;
3. **public** **class** Address **implements** Serializable{
5. **private** **static** **final** **long** serialVersionUID = 1L;
6. **int** homeNo;
7. String street;
8. String city;
9. **public** Address(**int** homeNo, String street, String city) {
10. **super**();
11. **this**.homeNo = homeNo;
12. **this**.street = street;
13. **this**.city = city;
14. }
15. **public** **int** getHomeNo() {
16. **return** homeNo;
17. }
18. **public** **void** setHomeNo(**int** homeNo) {
19. **this**.homeNo = homeNo;
20. }
21. **public** String getStreet() {
22. **return** street;
23. }
24. **public** **void** setStreet(String street) {
25. **this**.street = street;
26. }
27. **public** String getCity() {
28. **return** city;
29. }
30. **public** **void** setCity(String city) {
31. **this**.city = city;
32. }
33. }

**Run again:**

When you run again SerializeDeserializeMain.java.You will get following output

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

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

**Case 2:What if you don't have access to reference object's source code(e.g you don't have access to above Address class)**  
If you don't have access to address class then how will you implement serializable interface in Address class.Is there any alternative to that? yes there is,You can create another class which extends address and make it serialzable but It can fails in many cases:

* What if class is declared as final
* What if class have reference to other non serializable object.

So then how will  you serialize Employee object? so solution is you can make it transient.If you don't want to serialize any field then make it transient.

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

1. **transient** Address address

So after making address transient in Employee class when you run program.You will get nullPointerException because during deserialization address reference will be null

**Case 3:What if you still want to save state of reference object(e.g above address object):**

If you make address transient then during deserialization it will return null.But what if you still want to have same state as when you have serialized address object.Java serialization provides a mechnism such that if you have private methods with particular signature then they will get called during serialization and deserialization so we will override writeObject and readObject method of employee class and they will be called during serialization and deserialization of Employee object.  
  
**Employee.java:**

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

1. **package** org.arpit.javapostsforlearning;
2. **import** java.io.IOException;
3. **import** java.io.ObjectInputStream;
4. **import** java.io.ObjectOutputStream;
5. **import** java.io.Serializable;
7. **public** **class** Employee **implements** Serializable{
8. **private** **static** **final** **long** serialVersionUID = 1L;
10. **int** employeeId;
11. String employeeName;
12. String department;
13. **transient** Address address;
15. **public** **int** getEmployeeId() {
16. **return** employeeId;
17. }
18. **public** **void** setEmployeeId(**int** employeeId) {
19. **this**.employeeId = employeeId;
20. }
21. **public** String getEmployeeName() {
22. **return** employeeName;
23. }
24. **public** **void** setEmployeeName(String employeeName) {
25. **this**.employeeName = employeeName;
26. }
27. **public** String getDepartment() {
28. **return** department;
29. }
30. **public** **void** setDepartment(String department) {
31. **this**.department = department;
32. }
33. **public** Address getAddress() {
34. **return** address;
35. }
36. **public** **void** setAddress(Address address) {
37. **this**.address = address;
38. }
40. **private** **void** writeObject(ObjectOutputStream os) **throws** IOException, ClassNotFoundException
41. {
42. **try** {
43. os.defaultWriteObject();
44. os.writeInt(address.getHomeNo());
45. os.writeObject(address.getStreet());
46. os.writeObject(address.getCity());
47. }
48. **catch** (Exception e)
49. { e.printStackTrace(); }
50. }
52. **private** **void** readObject(ObjectInputStream is) **throws** IOException, ClassNotFoundException
53. {
54. **try** {
55. is.defaultReadObject();
56. **int** homeNo=is.readInt();
57. String street=(String) is.readObject();
58. String city=(String) is.readObject();
59. address=**new** Address(homeNo,street,city);
61. } **catch** (Exception e) { e.printStackTrace(); }
62. }
63. }

One thing should be kept in mind that ObjectInputStream should read data in same sequence in which we have written data to ObjectOutputStream.  
  
Create Address.java in org.arpit.javapostsforlearning  
**Address.java:**

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

1. **package** org.arpit.javapostsforlearning;
2. **import** java.io.Serializable;
4. **public** **class** Address {
6. **int** homeNo;
7. String street;
8. String city;

11. **public** Address(**int** homeNo, String street, String city) {
12. **super**();
13. **this**.homeNo = homeNo;
14. **this**.street = street;
15. **this**.city = city;
16. }
17. **public** **int** getHomeNo() {
18. **return** homeNo;
19. }
20. **public** **void** setHomeNo(**int** homeNo) {
21. **this**.homeNo = homeNo;
22. }
23. **public** String getStreet() {
24. **return** street;
25. }
26. **public** **void** setStreet(String street) {
27. **this**.street = street;
28. }
29. **public** String getCity() {
30. **return** city;
31. }
32. **public** **void** setCity(String city) {
33. **this**.city = city;
34. }
35. }

Create SerializeDeserializeMain.java in org.arpit.javapostsforlearning

**SerializeDeserializeMain.java:**

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

1. **package** org.arpit.javapostsforlearning;
2. **import** java.io.FileInputStream;
3. **import** java.io.FileOutputStream;
4. **import** java.io.IOException;
5. **import** java.io.ObjectInputStream;
6. **import** java.io.ObjectOutputStream;
8. **public** **class** SerializeDeserializeMain {
9. /\*\*
10. \* @author Arpit Mandliya
11. \*/
12. **public** **static** **void** main(String[] args) {
14. Employee emp = **new** Employee();
15. emp.setEmployeeId(101);
16. emp.setEmployeeName("Arpit");
17. emp.setDepartment("CS");
18. Address address=**new** Address(88,"MG road","Pune");
19. emp.setAddress(address);
20. //Serialize
21. **try**
22. {
23. FileOutputStream fileOut = **new** FileOutputStream("employee.ser");
24. ObjectOutputStream outStream = **new** ObjectOutputStream(fileOut);
25. outStream.writeObject(emp);
26. outStream.close();
27. fileOut.close();
28. }**catch**(IOException i)
29. {
30. i.printStackTrace();
31. }
33. //Deserialize
34. emp = **null**;
35. **try**
36. {
37. FileInputStream fileIn =**new** FileInputStream("employee.ser");
38. ObjectInputStream in = **new** ObjectInputStream(fileIn);
39. emp = (Employee) in.readObject();
40. in.close();
41. fileIn.close();
42. }**catch**(IOException i)
43. {
44. i.printStackTrace();
45. **return**;
46. }**catch**(ClassNotFoundException c)
47. {
48. System.out.println("Employee class not found");
49. c.printStackTrace();
50. **return**;
51. }
52. System.out.println("Deserialized Employee...");
53. System.out.println("Emp id: " + emp.getEmployeeId());
54. System.out.println("Name: " + emp.getEmployeeName());
55. System.out.println("Department: " + emp.getDepartment());
56. address=emp.getAddress();
57. System.out.println("City :"+address.getCity());
58. }
59. }

**Run it :**

When you run SerializeDeserializeMain.java.You will get following output

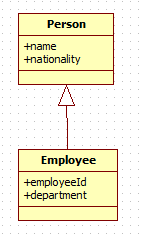
[view plainprint?](http://www.java2blog.com/2013/03/serialization-in-java.html)

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

so now we got same state of address object as it was before serialization.

**Inheritance in Serialization:**

Now we will see how inheritance affects serialization.So there can be muliple cases whether super class is serializable or not.If not then how will you handle that and how it works.Lets see by example.  
We will create Person.java which will be superclass of Employee.

[[](http://3.bp.blogspot.com/-zRhAWcMIfiE/UTudV-43vaI/AAAAAAAAA30/HQOrCaEC0iM/s1600/PersonEmployeeSerialization.gif)](http://3.bp.blogspot.com/-zRhAWcMIfiE/UTudV-43vaI/AAAAAAAAA30/HQOrCaEC0iM/s1600/PersonEmployeeSerialization.gif)

**Case 4:** **What  if superclass is Serializable?**  
If superclass is serialzable then all its subclasses are automatically serializable.  
  
**Case 5:What if superclass is not Serializable?**  
If super class is not serializable then we have to handle it quite differently.

* If superclass is not serializable then it must have no argument constructor.

**Person.java**

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

1. **package** org.arpit.javapostsforlearning;
2. **public** **class** Person {
4. String name="default";
5. String nationality;
7. **public** Person()
8. {
9. System.out.println("Person:Constructor");
10. }
12. **public** Person(String name, String nationality) {
13. **super**();
14. **this**.name = name;
15. **this**.nationality = nationality;
16. }
18. **public** String getName() {
19. **return** name;
20. }
22. **public** **void** setName(String name) {
23. **this**.name = name;
24. }
26. **public** String getNationality() {
27. **return** nationality;
28. }
30. **public** **void** setNationality(String nationality) {
31. **this**.nationality = nationality;
32. }
34. }

Create Employee.java in org.arpit.javapostsforlearning  
**Employee.java:**

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

1. **package** org.arpit.javapostsforlearning;
2. **import** java.io.Serializable;
4. **public** **class** Employee **extends** Person **implements** Serializable{
5. **private** **static** **final** **long** serialVersionUID = 1L;
6. **int** employeeId;
7. String department;
9. **public** Employee(**int** employeeId,String name,String department,String nationality)
10. {
11. **super**(name,nationality);
12. **this**.employeeId=employeeId;
13. **this**.department=department;
14. System.out.println("Employee:Constructor");
15. }
17. **public** **int** getEmployeeId() {
18. **return** employeeId;
19. }
20. **public** **void** setEmployeeId(**int** employeeId) {
21. **this**.employeeId = employeeId;
22. }
24. **public** String getDepartment() {
25. **return** department;
26. }
27. **public** **void** setDepartment(String department) {
28. **this**.department = department;
29. }
30. }

Create SerializeDeserializeMain.java in org.arpit.javapostsforlearning

**SerializeDeserializeMain.java:**

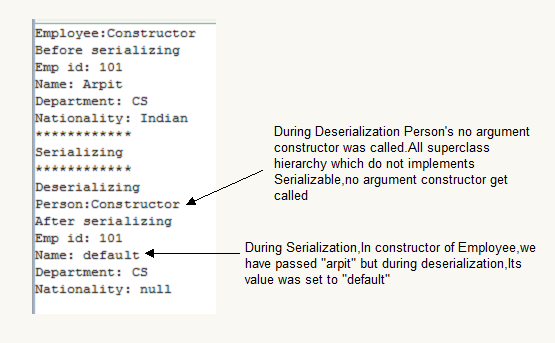
[view plainprint?](http://www.java2blog.com/2013/03/serialization-in-java.html)

1. **package** org.arpit.javapostsforlearning;
2. **import** java.io.FileInputStream;
3. **import** java.io.FileOutputStream;
4. **import** java.io.IOException;
5. **import** java.io.ObjectInputStream;
6. **import** java.io.ObjectOutputStream;
8. **public** **class** SerializeDeserializeMain {
10. /\*\*
11. \* @author Arpit Mandliya
12. \*/
13. **public** **static** **void** main(String[] args) {
15. //Serialize
16. Employee emp = **new** Employee(101,"Arpit","CS","Indian");
17. System.out.println("Before serializing");
18. System.out.println("Emp id: " + emp.getEmployeeId());
19. System.out.println("Name: " + emp.getName());
20. System.out.println("Department: " + emp.getDepartment());
21. System.out.println("Nationality: " + emp.getNationality());
22. System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*");
23. System.out.println("Serializing");
24. **try**
25. {
26. FileOutputStream fileOut = **new** FileOutputStream("employee.ser");
27. ObjectOutputStream outStream = **new** ObjectOutputStream(fileOut);
28. outStream.writeObject(emp);
29. outStream.close();
30. fileOut.close();
31. }**catch**(IOException i)
32. {
33. i.printStackTrace();
34. }
36. //Deserialize
37. System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*");
38. System.out.println("Deserializing");
39. emp = **null**;
40. **try**
41. {
42. FileInputStream fileIn =**new** FileInputStream("employee.ser");
43. ObjectInputStream in = **new** ObjectInputStream(fileIn);
44. emp = (Employee) in.readObject();
45. in.close();
46. fileIn.close();
47. }**catch**(IOException i)
48. {
49. i.printStackTrace();
50. **return**;
51. }**catch**(ClassNotFoundException c)
52. {
53. System.out.println("Employee class not found");
54. c.printStackTrace();
55. **return**;
56. }
57. System.out.println("After serializing");
58. System.out.println("Emp id: " + emp.getEmployeeId());
59. System.out.println("Name: " + emp.getName());
60. System.out.println("Department: " + emp.getDepartment());
61. System.out.println("Nationality: " + emp.getNationality());
62. }
63. }

**Run it :**

When you run SerializeDeserializeMain.java.You will get following output

[image: http://1.bp.blogspot.com/-ToyEyFYsVYY/UTuPt\_DeILI/AAAAAAAAA3s/vWY6RsKq7jQ/s1600/OutputSerializationInnheritance.gif](http://1.bp.blogspot.com/-ToyEyFYsVYY/UTuPt_DeILI/AAAAAAAAA3s/vWY6RsKq7jQ/s1600/OutputSerializationInnheritance.gif)

[[](http://1.bp.blogspot.com/-ToyEyFYsVYY/UTuPt_DeILI/AAAAAAAAA3s/vWY6RsKq7jQ/s1600/OutputSerializationInnheritance.gif)](http://1.bp.blogspot.com/-ToyEyFYsVYY/UTuPt_DeILI/AAAAAAAAA3s/vWY6RsKq7jQ/s1600/OutputSerializationInnheritance.gif)

If superclass is not Serializable then all values of the instance variables inherited from super class will be initialized by calling constructor of Non-Serializable Super class during deserialization process.so here name is inherited from person so during deserialization,name is initialized to default.  
  
**Case 6-What if superclass is Serializable but you don't want subclass to be Serializable**  
If you don't want subclass to serializable then you need to implement writeObject() and readObject() method and need to throw NotSerializableException from this methods.  
 **Case 7-Can you Serialize static variables?**  
No,you can't.As you know static variable are at class level not at object level and you serialize a object so you can't serialize static variables.  
  
Externalizable interface can be used in place of serializable if you want more control over serialization.You can read more about it at [Externalizable in java](http://javapostsforlearning.blogspot.in/2014/02/externalizable-in-java.html)

**Summary:**

* Serialization is the translation of your Java object's values/states to bytes to send it over network or save it.On other hand,Deserialization is conversion of byte code to corresponding java objects.
* Good thing about Serialization is entire process is JVM independent, meaning an object can be serialized on one platform and deserialized on an entirely different platform.\
* If you want to serialize any class then it must implement Serializable interface which is marker interface.
* Marker interface in Java is interface with no field or methods or in simple word empty interface in java is called marker interface
* serialVersionUID is used to ensure that same object(That was used during Serialization) is loaded during Deserialization.serialVersionUID is used for version control of object.
* When you serialize any object and if it contain any other object reference then Java serialization serialize that object's entire object graph.
* If you don't want to serialize any field,then make it trasient.
* If superclass is Serializable then its subclasses are automatically Serializable.
* If superclass is not Serializable then all values of the instance variables inherited from super class will be initialized by calling constructor of Non-Serializable Super class during deserialization process.
* If you don't want subclass to serializable then you need to implement writeObject() and readObject() method and need to throw NotSerializableException from this methods.
* You can't serialize static variables.