**Serlization & Deserlization**

* **Explain Serlization, deserlization and externalization what method used in that?**
* **What is the use of serialVersionId?**
* **Why do we need serlization?**
* **How to control EOFExceptin during deserlization?**
* **Different b/w serlization and Externalization?**
* **Compatible and incompatible changes in Serlization?**
* **Static and transient variable can be serialized or not?**
* **What is significance of transient variables?**
* [**Can you Serialize Singleton class such that object returned by Deserialization process is in same state as it was during Serialization time in java**](http://www.javamadesoeasy.com/2015/02/can-you-serialize-singleton-class-such.html) **?**
* [**Can subclass avoid Serialization if its super Class has implemented Serialization interface in java**](http://www.javamadesoeasy.com/2015/02/can-subclass-avoid-serialization-if-its.html)
* [**What values will int and Integer will be initialized to during DeSerialization process if they were not part of Serialization in java**](http://www.javamadesoeasy.com/2015/02/what-values-will-int-and-integer-will.html)
* [**Is constructor of super class called during DeSerialization process of sub class in java**](http://www.javamadesoeasy.com/2015/02/is-constructor-of-super-class-called.html)
* **Serializable with aggregation, collection, array?**
* **How to avoid NotSerializableException?**
* **When we get InvalidClassException?**
* **How to serialized super and sub class if super class not implementing sterilization interface?**
* **Is sub class serializable if super class implements serializable interface?**
* **How serrilizableVersionUID generated?**

[**http://www.javamadesoeasy.com/2015/02/serialization-top-25-interview.html**](http://www.javamadesoeasy.com/2015/02/serialization-top-25-interview.html)

**----------------------------------------------------------------------------------------------------------------------**

* **Explain Serlization, deserlization and externalization? List down method used in that?**

**Serlization**:

* Serlizaton is the process to convert object to stream.
* Method used in this process is **writeObject**()

**Deserlization**:

* Deserlization is the process to convert stream to object.
* Method used in this process is **readObject()**

**Externization**:

* It is also used to convert object to stream and stream to object.
* In case of Externization the application having full control on serialization and deserlization process.
* It is used normally to maintain data integrity.
* WriteExternal() and readExernal()
* **What is the use of serialVersionId?**
* SerialVersionUID is used for version control of object. It is verity that the sender and receiver object class loaded for that object is compatible or not.
* **Why do we need serlization?**
* Serlization we need to pass data over network and store in to the file
* **How to control EOFExceptin during deserlization?**
* Create a class EofIndicatorClass, which implements Serializable, interface.
* Write instance of EofIndicatorClass at EOF during serialization to indicate EOF during deserialization process.
* If oin.readObject() returns instance of EofIndicatorClass that means it's EOF, exit while loop and EOFException will not be thrown
* **Different b/w serlization and Externalization?**

|  |  |  |
| --- | --- | --- |
|  | **SERIALIZABLE** | **EXTERNALIZABLE** |
| Methods | It is a **marker** interface it does not have any method. | It is not a marker interface.  It has method’s called **writeExternal()** and **readExternal()** |
| Default Serialization process | **YES**, Serializable provides its own **default serialization process**, we just need to implement Serializable interface. | **NO**, we need to [override](http://www.javamadesoeasy.com/2015/06/method-overriding-in-java-in-detail.html) **writeExternal ()** and **readExternal ()** for serialization process to happen. |
| Customize serialization process | We **can** customize **default serialization process** by **defining following** methods in our class >**readObject()** and **writeObject()**  **Note**: We are not overriding these methods; we are defining them in our class. | Serialization process is completely customized  We need to **override** Externalizable interface’s **writeExternal ()** and **readExternal ()** methods. |
| Control over Serialization | It provides **less control** over Serialization as it is not mandatory to define **readObject ()** and **writeObject ()** methods. | Externalizable provides you **great control** over serialization process as it is important to override writeExternal **()** and **readExternal ()** methods. |
| Constructor call during **deSerialization** | Constructor is **not** called during deserialization. | Constructor **is called** during deserialization. |

* **Compatible and incompatible changes in Serlization?**

**Compatible Changes:**

* Compatible changes are those changes which does not affect deSerialization process even if class was updated after being serialized (provided serialVersionUID has been declared)
* Adding new fields - We can add new member variables in class.
* Adding writeObject ()/readObject () methods - We may add these methods to customize serialization process.
* Removing writeObject ()/readObject () methods - We may remove these methods and then default customization process will be used.
* Changing access modifier of a field - The change to access modifiers i.e. public, default, protected, and private have no effect on the ability of serialization to assign values to the fields.
* Changing a field from static to non-static OR changing transient filed to non-transient field. - It is like addition of fields.

**Incompatible Changes:**

* Incompatible changes are those changes which affect deSerialization process if class was updated after being serialized (provided serialVersionUID has been declared)
* Deletion of fields.
* Changing a no static field to static or non-transient field to transient field. - It is equal to deletion of fields.
* Modifying the writeObject () / readObject () method - we must not modify these method, though adding or removing them completely is compatible change.
* **Static and transient variable can be serialized or not?**
* Serialization is applicable on objects or primitive data types only, but [static](http://www.javamadesoeasy.com/2015/05/static-keyword-in-java-variable-method.html) members are class level variables, therefore, different objects of same class have same value for static member.
* Serializing static member will consume unnecessary space and time.
* If modification is made in static member by any of the object, it will not be in sync with other serialized object’s value.
* **What is significance of transient variables?**
* Serialization is not applicable on transient variables (it helps in saving time and space during Serialization process), we must mark all rarely used variables as transient. We can initialize transient variables during deSerialization by customizing deSerialization process.
* [**Can you Serialize Singleton class such that object returned by Deserialization process is in same state as it was during Serialization time in java**](http://www.javamadesoeasy.com/2015/02/can-you-serialize-singleton-class-such.html) **?**

[**http://www.javamadesoeasy.com/2015/02/can-you-serialize-singleton-class-such.html**](http://www.javamadesoeasy.com/2015/02/can-you-serialize-singleton-class-such.html)

* Yes

**private** **void** readObject(ObjectInputStream ois) **throws** IOException,ClassNotFoundException {

          System.*out*.println("in readObject()");

          ois.defaultReadObject();

*INSTANCE* = **this**; //rather than creating new instance, assign current object to INSTANCE

   }

   /\*\*

   \* Method ensures that we don't impact state of object in which it was Serialized.

   \*/

**private** Object readResolve() {

          System.*out*.println("in readResolve()");

**return** *INSTANCE*; //return INSTANCE.

   }

* [**Can subclass avoid Serialization if its superClass has implemented Serialization interface in java**](http://www.javamadesoeasy.com/2015/02/can-subclass-avoid-serialization-if-its.html)
* Yes
* If superclass has implemented Serializable that means subclass is also Serializable (as subclass always inherits all features from its parent class), for avoiding Serialization in sub-class we can define writeObject() method and throw NotSerializableException()
* [**What values will int and Integer will be initialized to during DeSerialization process if they were not part of Serialization in java**](http://www.javamadesoeasy.com/2015/02/what-values-will-int-and-integer-will.html)
* Int by zero and Integer by null.
* [**Is constructor of super class called during DeSerialization process of sub class in java**](http://www.javamadesoeasy.com/2015/02/is-constructor-of-super-class-called.html) **?**
* If superclass has implemented Serializable - constructor is not called during DeSerialization process.
* If super class has implemented Externalizable interface then constructor is called during DeSerialization process.
* **Serializable with aggregation, collection, array?**
* All collection implements serializable interface, so if we have these as member of class class will be serializable.
* **How to avoid NotSerializableException?**
* We got to ensure that during Serialization all the members of class implements Serializable.
* **When we get InvalidClassException?**
* If we don’t define serialVersionUID in the class, and any modification is made in class, then we won’t be able to deSerialize our class because serialVersionUID generated by java compiler for modified class will be different from old serialized object. And deserialization process will end up throwing java.io.InvalidClassException  (because of serialVersionUID mismatch)
* **How to serialized super and sub class if super class not implementing sterilization interface?**
* We need to define readObject and writeOject method in super class.