# Serialization and Deserialization

Serialization is a mechanism of converting the state of an object into a byte stream. Deserialization is the reverse process where the byte stream is used to recreate the actual Java object in memory. This mechanism is used to persist the object. The byte stream created is platform independent. So, the object serialized on one platform can be deserialized on a different platform.

Only the objects of those classes can be serialized which are implementing java.io.Serializable interface.  
Serializable is a marker interface (has no data member and method). It is used to “mark” java classes so that objects of these classes may get certain capability. Other examples of marker interfaces are: Cloneable and Remote.

To make a Java object serializable we implement the java.io.Serializable interface.

Serialization is the process of converting Java objects into a stream of bytes. The stream of bytes can be transmitted through a network connection, stored in a database as a BLOB object or saved as a binary file. The stored or transmitted stream of bytes can be reconstructed to Java object later.

## Serialization

The ObjectOutputStream class contains writeObject () method for serializing an Object.



## Deserialization

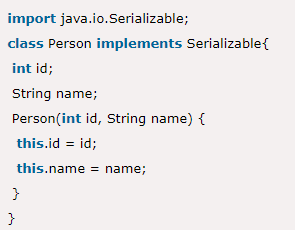
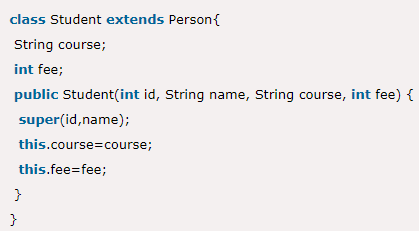
The ObjectInputStream class contains readObject () method for deserializing an object.



Points to remember  
1. If a parent class has implemented Serializable interface then child class doesn’t need to implement it but vice-versa is not true.  
2. Only non-static data members are saved via Serialization process.  
3. Static data members and transient data members are not saved via Serialization process. So, if you don’t want to save value of a non-static data member then make it transient.  
4. Associated objects must be implementing Serializable interface.

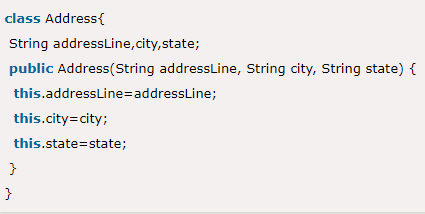
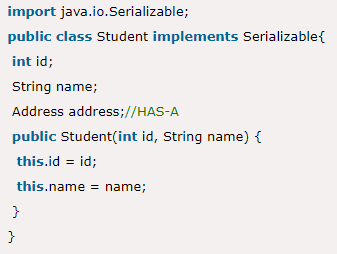
## Java Serialization with Inheritance (IS-A Relationship)

If a class implements serializable then all its sub classes will also be serializable.

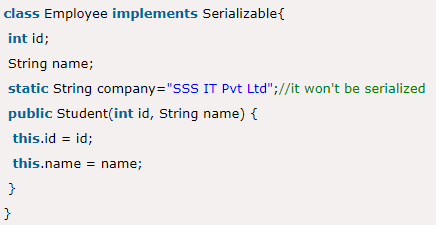
## Java Serialization with Aggregation (HAS-A Relationship)

If a class has a reference to another class, all the references must be Serializable otherwise serialization process will not be performed. In such case, NotSerializableException is thrown at runtime.

## Java Serialization with the static data member

If there is any static data member in a class, it will not be serialized because static is the part of class not object.



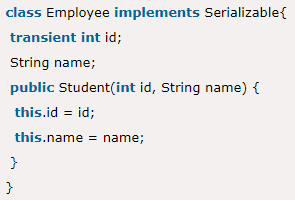
## Java Serialization with array or collection

## Rule: In case of array or collection, all the objects of array or collection must be serializable. If any object is not serializable, serialization will be failed. E.g. ArrayList is serializable by default but the objects used in collection should also be serializable.

<https://howtodoinjava.com/java/collections/arraylist/serialize-deserialize-arraylist/>

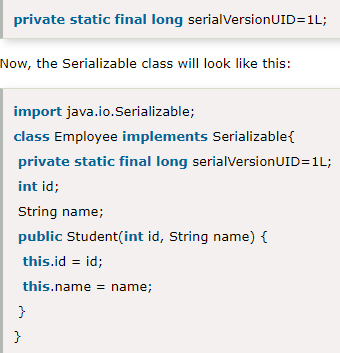
## Java Transient Keyword

If you don't want to serialize any data member of a class, you can mark it as transient.



## SerialVersionUID

The serialization process at runtime associates an id with each Serializable class which is known as SerialVersionUID. It is used to verify the sender and receiver of the serialized object. The sender and receiver must be the same. To verify it, SerialVersionUID is used. The sender and receiver must have the same SerialVersionUID, otherwise, InvalidClassException will be thrown when you deserialize the object. We can also declare our own SerialVersionUID in the Serializable class. To do so, you need to create a field SerialVersionUID and assign a value to it. It must be of the long type with static and final. It is suggested to explicitly declare the SerialVersionUID field in the class and have it private also.



## Different Cases for Serialization

<https://www.javacodegeeks.com/2013/03/serialization-in-java.html>