# Serialization & Deserialization

When we create a class, we may create the object for that particular class.

Once you close the class, or once you execute the program and terminate the program, the object is destroyed by itself.

What if we want to call the class without re-creating the object reference for that?

In those cases, we use the serialization concept where the data is converted to byte stream.

**Serialization:**

Process of converting the object into a sequence of bytes.

These sequence of bytes are then stored in disk or any other database and can be moved to any other database which is calling this byte streams.

**De-serialization:**

The reverse process of creating an object from sequence of bytes is called deserialization.

Any class that is trying to use this serialization or deserialization should implement the Serializable interface which is part of java.io package (java.io.Serializable).

So a java object is serializable if its class or any of its subclasses implements java.io.Serializable or its sub-interface java.io.Externizable interface.

The entire process is JVM independent, meaning an object can be serialized on one platform and de-serialize on an entirely different platform.

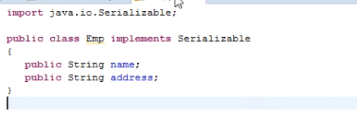
It’s not necessary to do the serialization and de-serialization at one place.

ObjectInputStream and ObjectOutputStreams are the classes that we use while working with serialization or deserialization.

These classes contains the methods used to serialization and deserialization.

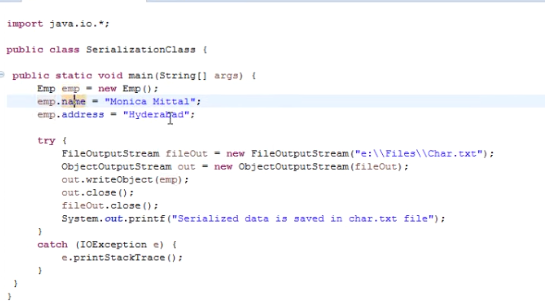
**Serialization – Example**

**Emp.java**



Here Emp class is implementing Serializable interface, so that we do serialization and de-serialization for this particular class.

**SerializationClass.java**



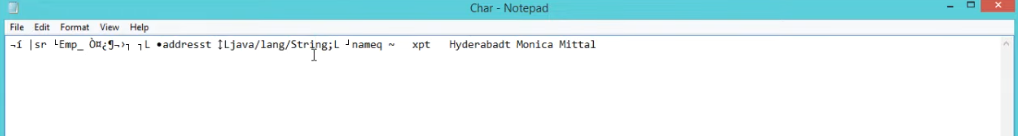
Here we want to implement the serialization concept, which is the process of converting object to byte streams.

So we have created an object for the class Emp, which is already implementing the Serializable interface (or else serialization or deserialization can’t be done).

Using ObjectOutputStream method “writeObject()”, we are pushing the content to the file specified.

So after the successful serialization process, Char.txt file will have the byte streams.

The file looks like below after serialization.

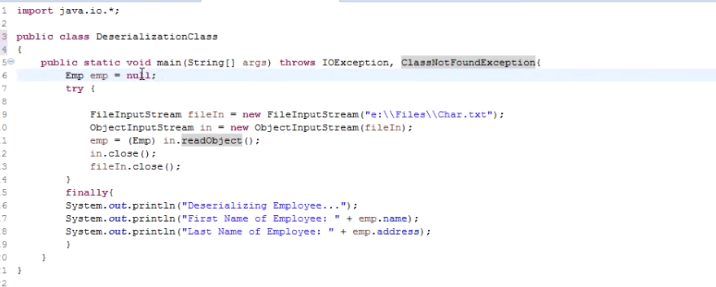


**Deserialization – Example**

Now without creating an object reference to the class Emp, using Deserialization concept we can retrieve the class data.

Here we use ObjectInputStream and method “readObject()” to read the byte streams and covert it back as object.

**DeserializationClass.java**



Output looks like below.

