|  |
| --- |
| **Serialization Interview Questions** |
| **Q1) What is Serialization?**  Ans) Serializable is a marker interface. When an object has to be transferred over a network ( typically through rmi or EJB) or persist the state of an object to a file, the object Class needs to implement Serializable interface. Implementing this interface will allow the object converted into bytestream and transfer over a network. |
| **Q2) What is use of serialVersionUID?**  Ans) During object serialization, the default Java serialization mechanism writes the metadata about the object, which includes the class name, field names and types, and superclass. This class definition is stored as a part of the serialized object. This stored metadata enables the deserialization process to reconstitute the objects and map the stream data into the class attributes with the appropriate type Everytime an object is serialized the java serialization mechanism automatically computes a hash value. ObjectStreamClass's*computeSerialVersionUID*() method passes the class name, sorted member names, modifiers, and interfaces to the secure hash algorithm (SHA), which returns a hash value.The serialVersionUID is also called *suid*. So when the serilaize object is retrieved , the JVM first evaluates the *suid* of the serialized class and compares the *suid* value with the one of the object. If the suid values match then the object is said to be compatible with the class and hence it is de-serialized. If not *InvalidClassException*exception is thrown.  Changes to a serializable class can be compatible or incompatible. Following is the list of changes which are compatible:   * Add fields * Change a field from static to non-static * Change a field from transient to non-transient * Add classes to the object tree   List of incompatible changes:   * Delete fields * Change class hierarchy * Change non-static to static * Change non-transient to transient * Change type of a primitive field   So, if no suid is present , inspite of making compatible changes, jvm generates new *suid* thus resulting in an exception if prior release version object is used .  The only way to get rid of the exception is to recompile and deploy the application again.  If we explicitly metion the suid using the statement:  private final static long serialVersionUID = <integer value>   then if any of the metioned compatible changes are made the class need not to be recompiled. But for incompatible changes there is no other way than to compile again. |
| **Q3) What is the need of Serialization?**  Ans) The serialization is used :-   * To send state of one or more object’s state over the network through a socket. * To save the state of an object in a file. * An object’s state needs to be manipulated as a stream of bytes. |
| **Q4) Other than Serialization what are the different approach to make object Serializable?**  Ans) Besides the Serializable interface, at least three alternate approaches can serialize Java objects:   1)For object serialization, instead of implementing the Serializable interface, a developer can implement the Externalizable interface, which extends Serializable. By implementing Externalizable, a developer is responsible for implementing the writeExternal() and readExternal() methods. As a result, a developer has sole control over reading and writing the serialized objects. 2)XML serialization is an often-used approach for data interchange. This approach lags runtime performance when compared with Java serialization, both in terms of the size of the object and the processing time. With a speedier XML parser, the performance gap with respect to the processing time narrows. Nonetheless, XML serialization provides a more malleable solution when faced with changes in the serializable object. 3)Finally, consider a "roll-your-own" serialization approach. You can write an object's content directly via either the ObjectOutputStream or the DataOutputStream. While this approach is more involved in its initial implementation, it offers the greatest flexibility and extensibility. In addition, this approach provides a performance advantage over Java serialization. |
| **Q5) Do we need to implement any method of Serializable interface to make an object serializable?**  Ans) No. Serializable is a Marker Interface. It does not have any methods. |
| **Q6) What happens if the object to be serialized includes the references to other serializable objects?**  Ans) If the object to be serialized includes the references to other objects whose class implements serializable then all those object’s state also will be saved as the part of the serialized state of the object in question. The whole object graph of the object to be serialized will be saved during serialization automatically provided all the objects included in the object’s graph are serializable. |
| **Q7) What happens if an object is serializable but it includes a reference to a non-serializable object?**  Ans- If you try to serialize an object of a class which implements serializable, but the object includes a reference to an non-serializable class then a ‘NotSerializableException’ will be thrown at runtime.  e.g.  public class NonSerial {      //This is a non-serializable class  }  public class MyClass implements Serializable{      private static final long *serialVersionUID* = 1L;      private NonSerial nonSerial;      MyClass(NonSerial nonSerial){          this.nonSerial = nonSerial;      }      public static void main(String [] args) {          NonSerial nonSer = new NonSerial();          MyClass c = new MyClass(nonSer);          try {          FileOutputStream fs = new FileOutputStream("test1.ser");          ObjectOutputStream os = new ObjectOutputStream(fs);          os.writeObject(c);          os.close();          } catch (Exception e) { e.printStackTrace(); }          try {          FileInputStream fis = new FileInputStream("test1.ser");          ObjectInputStream ois = new ObjectInputStream(fis);          c = (MyClass) ois.readObject();          ois.close();              } catch (Exception e) {              e.printStackTrace();            }      }  }  On execution of above code following exception will be thrown  – java.io.NotSerializableException: NonSerial              at java.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java) |
| **Q8) Are the static variables saved as the part of serialization?**  Ans) No. The static variables belong to the class and not to an object they are not the part of the state of the object so they are not saved as the part of serialized object. |
| **Q9) What is a transient variable?**  Ans) These variables are not included in the process of serialization and are not the part of the object’s serialized state. |
| **Q10) What will be the value of transient variable after de-serialization?**  Ans) It’s default value. e.g. if the transient variable in question is an int, it’s value after deserialization will be zero.  public class TestTransientVal implements Serializable{           private static final long *serialVersionUID* = -22L;      private String name;      transient private int age;      TestTransientVal(int age, String name) {          this.age = age;          this.name = name;      }       public static void main(String [] args) {          TestTransientVal c = new TestTransientVal(1,"ONE");          System.*out*.println("Before serialization: - " + c.name + " "+ c.age);          try {          FileOutputStream fs = new FileOutputStream("testTransients.ser");          ObjectOutputStream os = new ObjectOutputStream(fs);          os.writeObject(c);          os.close();          } catch (Exception e) { e.printStackTrace(); }            try {          FileInputStream fis = new FileInputStream("testTransients.ser");          ObjectInputStream ois = new ObjectInputStream(fis);          c = (TestTransientVal) ois.readObject();          ois.close();          } catch (Exception e) { e.printStackTrace(); }          System.*out*.println("After de-serialization:- " + c.name + " "+ c.age);          }  }  Result of executing above piece of code – Before serialization: - Value of non-transient variable ONE Value of transient variable 1  After de-serialization:- Value of non-transient variable ONE Value of transient variable 0  Explanation –  The transient variable is not saved as the part of the state of the serailized variable, it’s value after de-serialization is it’s default value. |
| **Q11) Does the order in which the value of the transient variables and the state of the object using the defaultWriteObject() method are saved during serialization matter?**  Ans) Yes.  As while restoring the object’s state the transient variables and the serializable variables that are stored must be restored in the same order in which they were saved. |
| **Q12) How can one customize the Serialization process? or What is the purpose of implementing the writeObject() and readObject() method?**  Ans) When you want to store the transient variables state as a part of the serialized object at the time of serialization the class must implement the following methods –  private void wrtiteObject(ObjectOutputStream outStream) { //code to save the transient variables state as a part of serialized object }  private void readObject(ObjectInputStream inStream) { //code to read the transient variables state and assign it to the de-serialized object }  e.g.  public class TestCustomizedSerialization implements Serializable{       private static final long *serialVersionUID* =-22L;      private String noOfSerVar;      transient private int noOfTranVar;       TestCustomizedSerialization(int noOfTranVar, String noOfSerVar) {          this.noOfTranVar = noOfTranVar;          this.noOfSerVar = noOfSerVar;      }       private void writeObject(ObjectOutputStream os) {        try {       os.defaultWriteObject();       os.writeInt(noOfTranVar);       } catch (Exception e) { e.printStackTrace(); }       }        private void readObject(ObjectInputStream is) {       try {       is.defaultReadObject();       int noOfTransients = (is.readInt());       } catch (Exception e) {           e.printStackTrace(); }       }        public int getNoOfTranVar() {          return noOfTranVar;      }       }  The value of transient variable ‘noOfTranVar’ is saved as part of the serialized object manually by implementing writeObject() and restored by implementing readObject(). The normal serializable variables are saved and restored by calling defaultWriteObject() and defaultReadObject()respectively. These methods perform the normal serialization and de-sirialization process for the object to be saved or restored respectively. |
| **Q13) If a class is serializable but its superclass in not , what will be the state of the instance variables inherited  from super class after deserialization?**  Ans) The values of the instance variables inherited from superclass will be reset to the values they were given during the original construction of the object as the non-serializable super-class constructor will run.  E.g.  public class ParentNonSerializable {      int noOfWheels;           ParentNonSerializable(){          this.noOfWheels = 4;      }       }  public class ChildSerializable extends ParentNonSerializable implements Serializable {          private static final long *serialVersionUID* = 1L;      String color;      ChildSerializable() {          this.noOfWheels = 8;          this.color = "blue";      }  }  public class SubSerialSuperNotSerial {       public static void main(String [] args) {           ChildSerializable c = new ChildSerializable();          System.*out*.println("Before : - " + c.noOfWheels + " "+ c.color);          try {          FileOutputStream fs = new FileOutputStream("superNotSerail.ser");          ObjectOutputStream os = new ObjectOutputStream(fs);          os.writeObject(c);          os.close();          } catch (Exception e) { e.printStackTrace(); }           try {          FileInputStream fis = new FileInputStream("superNotSerail.ser");          ObjectInputStream ois = new ObjectInputStream(fis);          c = (ChildSerializable) ois.readObject();          ois.close();          } catch (Exception e) { e.printStackTrace(); }          System.*out*.println("After :- " + c.noOfWheels + " "+ c.color);          }  }  Result on executing above code – Before : - 8 blue  After :- 4 blue  The instance variable ‘noOfWheels’ is inherited from superclass which is not serializable. Therefore while restoring it the non-serializable superclass constructor runs and its value is set to 8 and is not same as the value saved during serialization which is 4. |
| **Q14) To serialize an array or a collection all the members of it must be serializable. True /False?**  Ans) true. |

|  |
| --- |
| **Q30) When an obj is passed through a function , one can set the properties but cannot set a new memory location?**  Ans) It is because when u pass an object the address value is passed and stored in some new address . like if address 1234 is passed , it is stored in 4567 location. So if u change in the value of an object it will take the address from 4567 and do 1234.setXXX(). If u set the object to null it will set 4567=null.  **Default and generated serial version UID:**  **DEFAULT:      private static final long serialVersionUID = 1L;  Use this option to add a user-defined ID in combination with custom serialization code if the type did undergo structural changes since its first release.**  **GENERATED: private static final long serialVersionUID = -5596856758368970189L;  Use this option to add a compiler-generated ID if the type did not undergo structural changes since its first release.** |

**Custom Serialization With Serializable:**

**package inputOutput;**

**import java.io.FileNotFoundException;**

**import java.io.FileOutputStream;**

**import java.io.IOException;**

**import java.io.ObjectInputStream;**

**import java.io.ObjectOutputStream;**

**import java.io.Serializable;**

**import java.io.FileInputStream;**

**/\*\***

**\* The Class CustomSerializationWithSerializable.**

**\*/**

**public class CustomSerializationWithSerializable{**

**/\*\***

**\* Instantiates a new custom serialization with serializable.**

**\*/**

**public CustomSerializationWithSerializable(){**

**super();**

**}**

**/\*\***

**\* The main method.**

**\* @param args the arguments**

**\*/**

**public static void main(String[] args){**

**Person p=new Person("abhinav kumar mishra", 25,"noida");**

**Person p1=new Person("abhinav mishra", 26,"deoria");**

**ObjectInputStream objIn=null;**

**ObjectOutputStream objOut=null;**

**try{**

**objOut=new ObjectOutputStream(new FileOutputStream("serializedObjectPerson.txt"));**

**objOut.writeObject(p);**

**objOut.writeObject(p1);**

**System.out.println("Object serialized..");**

**}**

**catch(FileNotFoundException fne){**

**fne.printStackTrace();**

**}**

**catch(IOException ioe){**

**ioe.printStackTrace();**

**}**

**finally{**

**try {**

**objOut.close();**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

**try{**

**objIn=new ObjectInputStream(new FileInputStream ("serializedObjectPerson.txt"));**

**try {**

**System.out.println("Deserialized..");**

**System.out.println("Person data: "+(Person)objIn.readObject());**

**System.out.println("Person data: "+(Person)objIn.readObject());**

**} catch (ClassNotFoundException e) {**

**e.printStackTrace();**

**}**

**}**

**catch(FileNotFoundException fne){**

**fne.printStackTrace();**

**}**

**catch(IOException ioe){**

**ioe.printStackTrace();**

**}**

**finally{**

**try {**

**objIn.close();**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**}**

**/\*\***

**\* The Class Person.**

**\*/**

**class Person implements Serializable{**

**/\*\* The Constant serialVersionUID. \*/**

**private static final long serialVersionUID = 3125336987429015959L;**

**/\*\* The name. \*/**

**private String name;**

**/\*\* The address. \*/**

**private String address;**

**/\*\* The age. \*/**

**transient private int age;**

**/\*\***

**\* Gets the name.**

**\* @return the name**

**\*/**

**public String getName() {**

**return name;**

**}**

**/\*\***

**\* Sets the name.**

**\* @param name the new name**

**\*/**

**public void setName(String name) {**

**this.name = name;**

**}**

**/\*\***

**\* Gets the address.**

**\* @return the address**

**\*/**

**public String getAddress() {**

**return address;**

**}**

**/\*\***

**\* Sets the address.**

**\* @param address the new address**

**\*/**

**public void setAddress(String address) {**

**this.address = address;**

**}**

**/\*\***

**\* Gets the age.**

**\* @return the age**

**\*/**

**public int getAge() {**

**return age;**

**}**

**/\*\***

**\* Sets the age.**

**\* @param age the new age**

**\*/**

**public void setAge(int age) {**

**this.age = age;**

**}**

**/\*\***

**\* Instantiates a new person.**

**\*/**

**Person(){**

**super();**

**}**

**/\*\***

**\* Instantiates a new person.**

**\* @param name the name**

**\* @param age the age**

**\* @param address the address**

**\*/**

**Person(String name,int age,String address){**

**super();**

**this.name=name;**

**this.age=age;**

**this.address=address;**

**}**

**/\* (non-Javadoc)**

**\* @see java.lang.Object#toString()**

**\*/**

**public String toString(){**

**return getName()+" | "+getAge()+" | "+getAddress();**

**}**

**/\* (non-Javadoc)**

**\* @see java.lang.Object#equals(java.lang.Object)**

**\*/**

**public boolean equals(Object o){**

**Person p=(Person)o;**

**return this.age==p.age && this.name.equals(p.name)&& this.address.equals(p.address);**

**}**

**/\*\***

**\* Read object.**

**\* @param in the in**

**\* @throws IOException Signals that an I/O exception has occurred.**

**\* @throws ClassNotFoundException the class not found exception**

**\*/**

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

**in.defaultReadObject();**

**setAge(in.readInt());**

**}**

**/\*\***

**\* Write object.**

**\* @param out the out**

**\* @throws IOException**

**\*/**

**private void writeObject(ObjectOutputStream out) throws IOException{**

**out.defaultWriteObject();**

**out.writeInt(age); //Writing a transient variable to the state of object**

**}**

**}**

**Output>>**

Object serialized..

Deserialized..

Person data: abhinav kumar mishra | 25 | noida

Person data: abhinav mishra | 26 | deoria

**Singleton object serializable must implement readResolve() method:**

**If the Singleton class implements the java.io.Serializable interface, when a singleton is serialized and then deserialized more than once, there will be multiple instances of Singleton created. In order to avoid this readResolve method should be implemented.**

**package** designPattern;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**import** java.io.Serializable;

/\*\*

\* The Class SingletonObjectSerializing.

\*/

**public** **class** SingletonObjectSerializing {

/\*\*

\* The main method.

\* **@param** args the arguments

\*/

**public** **static** **void** main(String[] args) {

System.*out*.println("Singleton object: "+SingletonObject.*getInstance*());

**try** {

ObjectOutputStream objOut=**new** ObjectOutputStream(**new** FileOutputStream("SingletonObject.txt"));

objOut.writeObject(SingletonObject.*getInstance*());

System.*out*.println("Singleton object serialized: "+SingletonObject.*getInstance*());

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

**try** {

ObjectInputStream objIn=**new** ObjectInputStream(**new** FileInputStream("SingletonObject.txt"));

SingletonObject sinObj=**null**;

**try** {

sinObj = (SingletonObject) objIn.readObject();

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

}

System.*out*.println("Singleton object deserialized: "+sinObj);

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

/\*\*

\* The Class SingletonObject.

\*/

**class** SingletonObject **implements** Serializable{

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

/\*\*

\* Instantiates a new singleton object.

\*/

**private** SingletonObject(){

**super**();

System.*out*.println("Instantiated..");

}

/\*\* The instance. \*/

**private** **static** SingletonObject *instance*=**new** SingletonObject();

/\*\*

\* Gets the single instance of SingletonObject.

\*

\* **@return** single instance of SingletonObject

\*/

**public** **static** SingletonObject getInstance(){

**return** *instance*;

}

/\*\*

\* Read resolve.

\* This method is called immediately after an object of this class is deserialized.

\* This method returns the singleton instance.

\* **@return** the object

\*/

**protected** Object readResolve() {

**return** *getInstance*();

}

}

**Output>>**

**Instantiated..**

**Singleton object: designPattern.SingletonObject@addbf1**

**Singleton object serialized: designPattern.SingletonObject@addbf1**

**Singleton object deserialized: designPattern.SingletonObject@addbf1**

**package** designPattern;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**import** java.io.Serializable;

/\*\*

\* The Class SingletonObjectSerializing.

\*/

**public** **class** SingletonObjectSerializing {

/\*\*

\* The main method.

\* **@param** args the arguments

\*/

**public** **static** **void** main(String[] args) {

System.*out*.println("Singleton object: "+SingletonObject.*getInstance*());

**try** {

ObjectOutputStream objOut=**new** ObjectOutputStream(**new** FileOutputStream("SingletonObject.txt"));

objOut.writeObject(SingletonObject.*getInstance*());

System.*out*.println("Singleton object serialized: "+SingletonObject.*getInstance*());

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

**try** {

ObjectInputStream objIn=**new** ObjectInputStream(**new** FileInputStream("SingletonObject.txt"));

SingletonObject sinObj=**null**;

**try** {

sinObj = (SingletonObject) objIn.readObject();

} **catch** (ClassNotFoundException e) {

e.printStackTrace();

}

System.*out*.println("Singleton object deserialized: "+sinObj);

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

/\*\*

\* The Class SingletonObject.

\*/

**class** SingletonObject **implements** Serializable{

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

/\*\*

\* Instantiates a new singleton object.

\*/

**private** SingletonObject(){

**super**();

System.*out*.println("Instantiated withoutreadResolve method..");

}

/\*\* The instance. \*/

**private** **static** SingletonObject *instance*=**new** SingletonObject();

/\*\*

\* Gets the single instance of SingletonObject.

\*

\* **@return** single instance of SingletonObject

\*/

**public** **static** SingletonObject getInstance(){

**return** *instance*;

}

/\*\*

\* Read resolve.

\* This method is called immediately after an object of this class is deserialized.

\* This method returns the singleton instance.

\* **@return** the object

\*/

/\*protected Object readResolve() {

return getInstance();

}\*/

}

**Output>>**

**Instantiated without readResolve method..**

**Singleton object: designPattern.SingletonObject@19821f**

**Singleton object serialized: designPattern.SingletonObject@19821f**

***Singleton object deserialized: designPattern.SingletonObject@de6ced***

### JAVA.IO.SERIALIZATION: Custom Serialization Process (writeObject, readObject, writeReplace, readResolve)

**JAVA.IO.SERIALIZATION:**  
**>>>> USE OF "private void writeObject(ObjectOutputStream objOut)throws IOException{}",  
  
>>>> USE OF "private void readObject(ObjectInputStream objIn)throws IOException{}",  
  
>>>> USE OF "ANY-ACCESS-MODIFIER Object writeReplace() throws ObjectStreamException{}";  
  
>>>> USE OF "ANY-ACCESS-MODIFIER Object readResolve() throws ObjectStreamException{}";**  
  
Note: when we use writeReplace(..) method to replace the serializing object and serialize some other object then we must implement the readResolve(..) method in other object.  
See example below, writeReplace() in Employee class, and readResolve() in Organization class.  
################################################################################################  
  
package io\_serialization;  
  
import java.io.Serializable;  
  
public class Employee implements Serializable{  
  
private static final long serialVersionUID = 1L;  
int empId;  
public Employee() {  
super();  
System.out.println("Employee instantialted..");  
}  
  
}  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
package io\_serialization;  
import java.io.IOException;  
import java.io.ObjectInputStream;  
import java.io.ObjectOutputStream;  
import java.io.ObjectStreamException;  
  
public class Engineer extends Employee{  
private static final long serialVersionUID = 1L;  
int age;  
String name;  
String address;  
transient int salary=0;  
//Organization org=null;  
  
public Engineer(int age, String name, String address,String orgName,int orgId) {  
super();   
this.age = age;  
this.name = name;  
this.address = address;  
this.salary=22000;  
empId=101;  
//org=new Organization(orgId,orgName);  
System.out.println("Engineer object instantiated\_param-constructor");  
}  
  
public Engineer(Organization o) {  
System.out.println("Engineer object instantiated\_param-constructor\_org");  
System.out.println("OrgData: "+o.orgId+" | "+o.orgName);  
}  
  
public String toString(){  
return "Emp: "+name+" | "+age+" | "+address+" |transientVar "+salary+" |empId "+empId;  
}  
  
Engineer() {  
super();  
System.out.println("Engineer object instantiated\_def-constructor");  
}  
  
/\*private void writeObject(ObjectOutputStream objOut)throws IOException{  
objOut.defaultWriteObject();  
objOut.writeInt(salary);  
}  
  
private void readObject (ObjectInputStream objIn)throws IOException, ClassNotFoundException{  
objIn.defaultReadObject();  
salary=objIn.readInt();  
}\*/  
  
public Object writeReplace()throws ObjectStreamException{  
Organization org=new Organization(102,"abcd");  
return org;  
}  
  
/\*public Object readResolve()throws ObjectStreamException{  
return new Employee();  
}\*/  
}  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
package io\_serialization;  
  
import java.io.Serializable;  
  
public class Organization implements Serializable {  
private static final long serialVersionUID = 1L;  
int orgId;  
String orgName="India pvt.ltd.";  
public Organization() {  
super();  
System.out.println("Org instantiated..\_default");  
}  
  
public Organization(int orgId, String orgName) {  
super();  
this.orgId = orgId;  
this.orgName = orgName;  
System.out.println("Org instantiated..\_param");  
  
}  
  
public String toString(){  
return "OrgData: "+orgId+" | "+orgName;  
}  
  
public Object readResolve(){  
return new Engineer(this);  
}  
}  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
package io\_serialization;  
  
import java.io.FileOutputStream;  
import java.io.IOException;  
import java.io.ObjectOutputStream;  
  
public class SerializeObject {  
public static void main(String[] args) {  
ObjectOutputStream objOut=null;  
try{  
Engineer e=new Engineer(25,"abhinav","noida","BCDS",102);  
System.out.println("Emp obj before serialization: "+e);  
objOut=new ObjectOutputStream(new FileOutputStream("emp.txt"));  
objOut.writeObject(e);  
objOut.flush();  
System.out.println("Emp object serialized..");  
}catch (IOException e) {  
e.printStackTrace();  
}finally{  
try {  
if(objOut!=null){  
objOut.close();  
}  
} catch (IOException e) {  
e.printStackTrace();  
}  
}  
}  
}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
package io\_serialization;  
  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.io.ObjectInputStream;  
  
public class DeserializeObject {  
public static void main(String[] args) {  
ObjectInputStream objIn=null;  
try{  
Engineer e=null;  
objIn=new ObjectInputStream(new FileInputStream("emp.txt"));  
e=(Engineer)objIn.readObject();  
System.out.println("Deserialized emp obj: "+e);  
}catch (IOException e) {  
e.printStackTrace();  
} catch (ClassNotFoundException e) {  
e.printStackTrace();  
}finally{  
if(objIn!=null){  
try {  
objIn.close();  
} catch (IOException e) {  
e.printStackTrace();  
}  
}  
}  
}  
}  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
**OUTPUT SERIALIZATION>>>**

Employee instantiated..  
Engineer object instantiated\_param-constructor  
Emp obj before serialization: Emp: abhinav | 25 | noida |transientVar 22000 |empId 101  
Org instantiated..\_param  
Emp object serialized..  
  
**OUTPUT DE-SERIALIZATION>>>**  
  
Employee instantiated..  
Engineer object instantiated\_param-constructor\_org  
OrgData: 102 | abcd  
Deserialized emp obj: Emp: null | 0 | null |transientVar 0 |empId 0