WHAT IS SERIALIZATION IN JAVA

Serialization in Java is a process to persist any Java Object's state into File System or convert them into byte stream to transfer over the network to other JVM or program. Serialization in Java is done by JVM by employing default Serialization process which persist all of Object's state except [transient variable](http://java67.blogspot.sg/2012/08/what-is-transient-variable-in-java.html) and a [static variable](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html). class needs to implements java.io.Serializable interface and JVM will automatically serialize its instance when passed over to java.io.ObjectOutputStream using writeObject(). Serializable interface is also called [marker interface or tag interface](http://javarevisited.blogspot.sg/2012/01/what-is-marker-interfaces-in-java-and.html) because it does not contain any method or behavior and just used to tag a class, by seeing a class implementing Serializable JVM knows that it needs to serialize an instance of this class.

[**SERIALIZATION**](http://www.javamadesoeasy.com/2015/02/serialize-and-deserialize-object.html)

Create object of ObjectOutput and give it’s reference variable name oout and call writeObject() method and pass our employee object as parameter [**oout.writeObject(object1) ]**

|  |
| --- |
| OutputStream fout = **new** FileOutputStream("ser.txt");  ObjectOutput oout = **new** ObjectOutputStream(fout);  System.*out*.println("Serialization process has started, serializing employee objects...");  **oout.writeObject(object1);** |

[**DESERIALIZATION**](http://www.javamadesoeasy.com/2015/02/serialize-and-deserialize-object.html)

Create object of ObjectInput and give it’s reference variable name oin and call readObject() method [**oin.readObject() ]**

|  |
| --- |
| InputStream fin=**new** FileInputStream("ser.txt");  ObjectInput oin=**new** ObjectInputStream(fin);  System.*out*.println("DeSerialization process has started, displaying employee objects...");  Employee emp;  emp=(Employee)**oin.readObject();** |

**What is Volatile variable in Java:**

Volatile variable in Java is a special variable which is used to signal threads, compiler that this particular variables values is going to be updated by multiple thread inside Java application. By making a variable volatile using volatile keyword in Java, application programmer ensures that its value should always been read from [main memory](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html) and thread should not use cached value of that variable from their own stack.

Read more: <http://java67.blogspot.com/2012/08/what-is-volatile-variable-in-java-when.html#ixzz458pThxhG>

## DIFFERENCE BETWEEN SERIALIZABLE AND EXTERNALIZABLE IN JAVA

1. One of the obvious difference between Serializable and Externalizable is that Serializable is a [marker interface](http://javarevisited.blogspot.in/2012/01/what-is-marker-interfaces-in-java-and.html) i.e. does not contain any method but Externalizable interface contains two methods writeExternal() and readExternal().
2. Second difference between Serializable vs Externalizable is responsibility of Serialization. When a class implements Serializable interface, default Serialization process gets kicked off and that takes responsibility of serializing super class state. When any [class in Java](http://javarevisited.blogspot.sg/2011/10/class-in-java-programming-general.html) implement java.io.Externalizable than it’s your responsibility to implement Serialization process i.e. preserving all important information.
3. This difference between Serializable and Externalizable is **performance**. You cannot do much to improve performance of default serialization process except reducing number of fields to be serialized by using [transient](http://java67.blogspot.sg/2012/08/what-is-transient-variable-in-java.html) and [static keyword](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html) but with Externalizable interface you have full control over Serialization process.
4. Another important difference between Serializable and Externalizable [interface](http://javarevisited.blogspot.sg/2012/04/10-points-on-interface-in-java-with.html) is **maintenance**. When your Java class implements Serializable interface it’s tied with default representation which is fragile and easily breakable if structure of class changes e.g. adding or removing field. By using java.io.Externalizable interface you can create your own custom binary format for your object.

### **What will happen if one of the members in the class doesn't implement Serializable interface?**

One of the easy question about Serialization process in Java. 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 and this is why I always put a *SerializableAlert* (comment section in my code) , one of the [code comment best practices](http://javarevisited.blogspot.sg/2011/08/code-comments-java-best-practices.html), to instruct developer to remember this fact while adding a new field in a Serializable class.

## [How can a sub-class of Serializable super class avoid serialization? If serializable interface is implemented by the super class of a class, how can the serialization of the class be avoided?](http://www.fromdev.com/2012/06/15-java-serialization-interview.html" \l "how-can-a-sub-class-of-serializable-super-class-avoid-serialization--if-serializable-interface-is-implemented-by-the-super-class-of-a-class--how-can-the-serialization-of-the-class-be-avoided-" \o "How can a sub-class of Serializable super class avoid serialization? If serializable interface is implemented by the super class of a class, how can the serialization of the class be avoided?)

In Java, if the super class of a class is implementing Serializable interface, it means that it is already serializable. Since, an interface cannot be unimplemented, it is not possible to make a class non-serializable. However, the serialization of a new class can be avoided. For this, writeObject () and readObject() methods should be implemented in your class so that a Not Serializable Exception can be thrown by these methods. And, this can be done by customizing the Java Serialization process. Below the code that demonstrates it

class MySubClass extends SomeSerializableSuperClass {

private void writeObject(java.io.ObjectOutputStream out)

throws IOException {

throw new NotSerializableException(“Can not serialize this class”);

}

private void readObject(java.io.ObjectInputStream in)

throws IOException, ClassNotFoundException {

throw new NotSerializableException(“Can not serialize this class”);

}

private void readObjectNoData()

throws ObjectStreamException; {

throw new NotSerializableException(“Can not serialize this class”);

}

}

### **Suppose you have a class which you serialized it and stored in persistence and later modified that class to add a new field. What will happen if you deserialize the object already serialized?**

It depends on whether class has its own serialVersionUID or not. As we know from above question that if we don't provide serialVersionUID in our code java compiler will generate it and normally it’s [equal to hashCode of object](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html). by adding any new field there is chance that new serialVersionUID generated for that class version is not the same of already serialized object and in this case Java Serialization API will [throw](http://javarevisited.blogspot.sg/2012/02/difference-between-throw-and-throws-in.html) java.io.InvalidClassException and this is the reason its recommended to have your own serialVersionUID in code and make sure to keep it same always for a single class.

**Question 1. What is Serialization in java?**

**Answer**. Let’s start by understanding what is Serialization, it’s most basic question which **you will have to answer almost in each and every java interview**. Serialization is process of converting **object into byte stream**.

Serialized object (byte stream) can be:

>Transferred over network.

>Persisted/saved into file.

>Persisted/saved into database.

Once, object have have been transferred over network or persisted in file or in database, we could deserialize the object and retain its state as it is in which it was serialized.

**Question 2. How do we Serialize object, write a program to serialize and deSerialize object and persist it in file (Important)?**

**Answer**. **You must be able to write Serialization code** to impress interviewer. In order to serialize object our class needs to implement **java.io.Serializable** interface. Serializable interface is **Marker interface** i.e. it **does not have any methods** of its own, **but** it **tells Jvm that object has to converted into byte stream**.

[**SERIALIZATION**](http://www.javamadesoeasy.com/2015/02/serialize-and-deserialize-object.html)

Create object of ObjectOutput and give it’s reference variable name oout and call writeObject() method and pass our employee object as parameter [**oout.writeObject(object1) ]**

|  |
| --- |
| OutputStream fout = **new** FileOutputStream("ser.txt");  ObjectOutput oout = **new** ObjectOutputStream(fout);  System.*out*.println("Serialization process has started, serializing employee objects...");  **oout.writeObject(object1);** |

[**DESERIALIZATION**](http://www.javamadesoeasy.com/2015/02/serialize-and-deserialize-object.html)

Create object of ObjectInput and give it’s reference variable name oin and call readObject() method [**oin.readObject() ]**

|  |
| --- |
| InputStream fin=**new** FileInputStream("ser.txt");  ObjectInput oin=**new** ObjectInputStream(fin);  System.*out*.println("DeSerialization process has started, displaying employee objects...");  Employee emp;  emp=(Employee)**oin.readObject();** |

**Question 3 . Difference between Externalizable and Serialization interface (Important)?**

**Answer**. Here comes the time to **impress interviewer** by differentiating Serializable and Externalizable use.

|  |  |  |
| --- | --- | --- |
|  | [**SERIALIZABLE**](http://www.javamadesoeasy.com/2015/02/serialize-and-deserialize-object.html) | [**EXTERNALIZABLE**](http://www.javamadesoeasy.com/2015/02/serialize-and-deserialize-object-by.html) |
| Methods | It is a **marker** interface it doesn’t have any method. | It’s 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 **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’s 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. |

**Question 4. How can you customize Serialization and DeSerialization process when you have implemented Serializable interface (Important)?**

**Answer**.  Here comes the quite **challenging question**, where you could prove how strong your Serialization concepts are.We can [customize **Serialization** process by defining **writeObject()**  method & **DeSerialization** process by defining **readObject()** method](http://www.javamadesoeasy.com/2015/02/customize-serialization-process-by.html).

Let’s customize **Serialization** process by defining **writeObject()**  method :

|  |
| --- |
| **private void writeObject(ObjectOutputStream os) {**            System.*out*.println("In, writeObject() method.");  **try** {                   os.writeInt(**this**.id);                   os.writeObject(**this**.name);            } **catch** (Exception e) {                   e.printStackTrace();            }     } |

We have serialized id and name manually by writing them in file.

  Let’s customize **DeSerialization** process by defining **readObject()**  method :

|  |
| --- |
| **private void readObject(ObjectInputStream ois) {**            System.*out*.println("In, readObject() method.");  **try** {                   id=ois.readInt();                   name=(String)ois.readObject();            } **catch** (Exception e) {                   e.printStackTrace();            }     } |

We have DeSerialized id and name manually by reading them from file.

**Question 5. Wap to explain how can we Serialize and DeSerialize object by implementing Externalizable interface (Important)?**

**Answer**. For [serializing object by implementing Externalizable interface](http://www.javamadesoeasy.com/2015/02/serialize-and-deserialize-object-by.html), we need to override writeExternal() and readExternal() for serialization process to happen.

For **Serialization** process override **writeExternal()**  method & for **DeSerialization** process by override **readExternal()** method.

Let’s customize **Serialization** process by overriding [**writeExternal()**](http://www.javamadesoeasy.com/2015/02/serialize-and-deserialize-object-by.html)method :

|  |
| --- |
| **public** **void** **writeExternal**(ObjectOutput oo) **throws** IOException {            System.*out*.println("in writeExternal()");            oo.writeInt(id);            oo.writeObject(name);    } |

We have serialized id and name manually by writing them in file.

  Let’s customize **DeSerialization** process by overriding [**readExternal()**](http://www.javamadesoeasy.com/2015/02/serialize-and-deserialize-object-by.html)  method :

|  |
| --- |
| **public** **void** **readExternal**(ObjectInput in) **throws** IOException, ClassNotFoundException {            System.*out*.println("in readExternal()");  **this**.id=in.readInt();  **this**.name=(String)in.readObject();    } |

We have DeSerialized id and name manually by reading them from file.

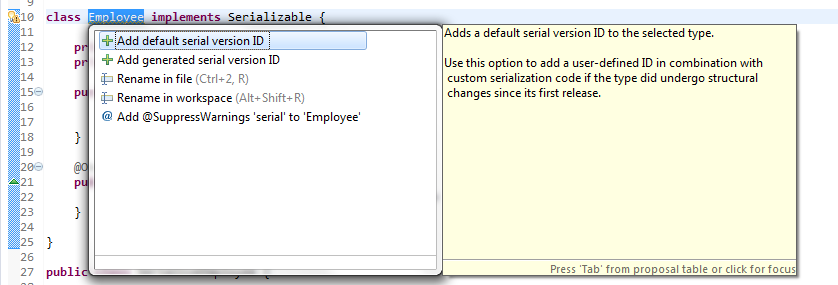
**Question 6. How can you avoid certain member variables of class from getting Serialized?**

**Answer**. Mark member variables as [**static**](http://www.javamadesoeasy.com/2015/05/static-keyword-in-java-variable-method.html)or **transient**, and those member variables will no more be a part of Serialization.

**Question 7. What is serialVersionUID?**

**Answer**. The serialization at runtime associates with each serializable class a version number, called a serialVersionUID, which is used during deserialization to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to serialization.

We can use eclipse to generate serialVersionUID for our class (as done in below snapshot)



How to avoid **warning** ‘The serializable class Employee does not declare a static final serialVersionUID field of type long’ ?

Again answer is we can use eclipse to generate serialVersionUID for our class (as mentioned in above screenshot, click on warning button on left in line 10).

**Question 8. What will be** [**impact of not defining serialVersionUID**](http://www.javamadesoeasy.com/2015/02/impact-of-not-defining-serialversionuid.html) **in class (Important)?**

**Answer**.  This is one my favourite question, i am going to discuss it in a very detailed manner. serialVersionUID is used for **version control of object**.

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)

Let’s frame another question by twisting few words in it.

*If you have serialized a class & then added few fields in it and then deserialize already serialized version of class, how can you ensure that you don’t end up throwing* ***InvalidClassException****?*

**>**Simply we need to define **serialVersionUID** in class.

When we Deserialize class ( class which has been modified after Serialization and also class **doesn’t declare SerialVersionUID**) **InvalidClassException** is thrown.

When we Deserialize class ( class which has been modified after Serialization and also class **declare SerialVersionUID**) its gets DeSerialized **successfully**.

[Let’s discuss this interesting topic in detail - Impact of not defining serialVersionUID in class and  avoiding **InvalidClassException**](http://www.javamadesoeasy.com/2015/02/impact-of-not-defining-serialversionuid.html)

**Question 9. What are compatible and incompatible changes in Serialization process?**

**Answer**.

**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’s 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 nonstatic field to static or  non transient field to transient field. -** it’s 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.

**Question 10. What if Serialization is not available, is any any other alternative way to transfer object over network?**

**Answer**.

>We can can convert **JSON** to transfer the object. JSON is helpful in stringifying and de stringifying object.

>**Hibernate** (ORM tool) helps in persisting object as it in database and later we can read persisted object.

>We can convert object into **XML** (as done in web services) and transfer object over network.

**Question 11. Why static member variables are not part of java serialization process (Important)?**

**Answer**. 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 object’s of same class have same value for static member**.

So, serializing static member will consume unnecessary space and time.

Also, if modification is made in static member by any of the object, it won’t be in sync with other serialized object’s value.

**Question 12. What is significance of transient variables?**

**Answer**. 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.

**Question 13. What will happen if one the member of class does not implement Serializable interface (Important)?**

**Answer**. This is classy question which will check your in depth knowledge of Serialization concepts. If any of the member does not implement Serializable than  NotSerializableException is thrown. [Now, let’s see a program.](http://www.javamadesoeasy.com/2015/02/if-member-of-class-does-not-implement.html)

**Question 14. What will happen if we have used List, Set and Map as member of class?**

**Answer**. This question which will check your in depth knowledge of Serialization and Java Api’s. ArrayList, HashSet and HashMap implements Serializable interface, so if we will use them as member of class they will get Serialized and DeSerialized as well.  [Now, let’s see a program.](http://www.javamadesoeasy.com/2015/02/can-list-set-and-maps-be-serialized-and.html)

**Question 15. Is constructor of class called during DeSerialization process?**

**Answer**. This question which will check your in depth knowledge of Serialization and constructor chaining concepts. It depends on whether our object has implemented Serializable or Externalizable.

If **Serializable** has been implemented - constructor is **not called** during DeSerialization process.

But, if **Externalizable** has been implemented - constructor **is called** during DeSerialization process.

[DETAILED DESCRIPTION : Is constructor of class called during DeSerialization process](http://www.javamadesoeasy.com/2015/02/is-constructor-of-class-called-during.html)

**Question 16 . Are primitive types part of serialization process?**

**Answer**. **Yes**, [primitive types are part of serialization process](http://www.javamadesoeasy.com/2015/02/are-primitive-types-part-of.html). Interviewer tends to check your basic java concepts over here.

**Question 17. Is constructor of super class called during DeSerialization process of subclass (Important)?**

**Answer**. Again your basic java concepts will be tested over here. It is depends on whether our superclass has implemented Serializable or not.

If superclass **has implemented Serializable** - constructor **is not called** during DeSerialization process.

If superclass has **not implemented Serializable** - constructor **is called** during DeSerialization process.

[DETAILED DESCRIPTION : Is constructor of super class called during DeSerialization process of sub class](http://www.javamadesoeasy.com/2015/02/is-constructor-of-super-class-called.html)

**Question 18. What values will int and Integer will be initialized to during DeSerialization process if they were not part of Serialization?**

**Answer**.  [int will be initialized to 0 and Integer will be initialized to null during DeSerialization](http://www.javamadesoeasy.com/2015/02/what-values-will-int-and-integer-will.html) (if they were not part of Serialization process).

**Question 19. How you can avoid Deserialization process creating another instance of Singleton class (Important)?**

**Answer**.This is another classy and very important question which will check your in depth knowledge of Serialization and Singleton concepts. I’ll prefer you must understand this concept in detail. We can simply use **readResove()** method to return same instance of class, rather than creating a new one.

Defining readResolve() method ensures that we don't break singleton pattern during DeSerialization process.

|  |
| --- |
| **private** Object readResolve() **throws** ObjectStreamException {  **return** *INSTANCE*;   } |

Also define readObject() method, rather than creating new instance, assign current object to INSTANCE like done below :

|  |
| --- |
| **private** **void** **readObject**(ObjectInputStream ois) **throws** IOException,ClassNotFoundException{         ois.defaultReadObject();  **synchronized** (SingletonClass.**class**) {  **if** (*INSTANCE* == **null**) {  ***INSTANCE* = this;**          }         }   } |

[DETAILED DESCRIPTION : Avoid Deserialization process creating another instance of Singleton class](http://www.javamadesoeasy.com/2015/02/avoid-deserialization-process-creating.html)

**Question 20. Can you Serialize Singleton class such that object returned by Deserialization process  is in same state as it was during Serialization time (regardless of any change made to it after Serialization)  (Important)?**

**Answer**. It’s another very important question which will be important in testing your Serialization and Singleton related concepts, you must try to understand the concept and question in detail.

**YES**, we can Serialize Singleton class such that object returned by Deserialization process is in same state as it was during Serialization time (regardless of any change made to it after Serialization)

Defining readResolve() method ensures that we don't break singleton pattern during DeSerialization process.

|  |
| --- |
| **private** Object readResolve() **throws** ObjectStreamException {  **return** *INSTANCE*;   } |

Also define readObject() method, rather than creating new instance, assign current object to INSTANCE like done below :

|  |
| --- |
| **private** **void** **readObject**(ObjectInputStream ois) **throws** IOException,ClassNotFoundException{         ois.defaultReadObject();  **synchronized** (SingletonClass.**class**) {  **if** (*INSTANCE* == **null**) {  ***INSTANCE* = this;**          }         }   } |

[DETAILED DESCRIPTION : Can you Serialize Singleton class such that object returned by Deserialization process  is in same state as it was during Serialization time](http://www.javamadesoeasy.com/2015/02/can-you-serialize-singleton-class-such.html)

**Question 21. Purpose of serializing Singleton class OR  purpose of saving singleton state?**

**Answer**.Let’s take example of our laptop, daily eod we need to shut it down, but rather than shutting it down hibernate (save state of  laptop) is better option because it enables us to resume at same point where we leaved it, like wise serializing singleton OR saving state of Singleton can be very handy.

**Question 22. How can subclass avoid Serialization if its superClass has implemented Serialization interface (Important)?**

**Answer**. 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()** from there as done below.

|  |
| --- |
| **private void writeObject(ObjectOutputStream os) throws NotSerializableException {**  **throw new NotSerializableException("This class cannot be Serialized");**  **}** |

[DETAILED DESCRIPTION : Can subclass avoid Serialization if its superClass has implemented Serialization interface](http://www.javamadesoeasy.com/2015/02/can-subclass-avoid-serialization-if-its.html)

**You might be given code snippets in interviews and asked to give output -**

**Question 23. Find output of following code :**

|  |
| --- |
| **class** MyClass **implements** Serializable {  **private** **static** **final** **long** *serialVersionUID* = 1L;  **private** List<Integer> list;  **private** Set<Integer> set;  **private** Map<Integer,Integer> map;  **public** MyClass(List<Integer> list, Set<Integer> set,                   Map<Integer, Integer> map) {  **super**();  **this**.list = list;  **this**.set = set;  **this**.map = map;     }     @Override  **public** String toString() {  **return** "MyClass [list=" + list + ", set=" + set + ", map=" + map + "]";     }    }  **public** **class** SerializeEmployee {  **public** **static** **void** main(String[] args) {            List<Integer> list=**new** ArrayList<Integer>();            list.add(2);            list.add(3);            Set<Integer> set=**new** HashSet<Integer>();            set.add(4);            set.add(5);            Map<Integer, Integer> map=**new** HashMap<Integer,Integer>();            map.put(6, 34);            map.put(7, 35);            MyClass object1 = **new** MyClass(list,set,map);  **try** {                   OutputStream fout = **new** FileOutputStream("ser.txt");                   ObjectOutput oout = **new** ObjectOutputStream(fout);                   System.*out*.println("Serialization process has started, serializing objects...");                   oout.writeObject(object1);                   fout.close();          oout.close();          System.*out*.println("Object Serialization completed.");                      //DeSerialization process >                       InputStream fin=**new** FileInputStream("ser.txt");                   ObjectInput oin=**new** ObjectInputStream(fin);                   System.*out*.println("\nDeSerialization process has started, displaying objects...");                   MyClass object=(MyClass)oin.readObject();                   System.*out*.println(object);                   fin.close();         oin.close();         System.*out*.println("Object DeSerialization completed.");              } **catch** (IOException | ClassNotFoundException  e) {                   e.printStackTrace();            }     }  } |

**Answer**. Here intention of interviewer will be to find out whether you know that list, set and map can be serialized or not.

/\*OUTPUT

Serialization process has started, serializing objects...

Object Serialization completed.

DeSerialization process has started, dispalying objects...

MyClass [list=[2, 3], set=[4, 5], map={6=34, 7=35}]

Object DeSerialization completed.

\*/

**Question 24.  Find output of following code  (Important):**

|  |
| --- |
| **package** SerDeser10memberNotSer;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **import** java.io.ObjectOutput;  **import** java.io.ObjectOutputStream;  **import** java.io.OutputStream;  **import** java.io.Serializable;  **class** MyClass {}  /\*Author : AnkitMittal  Copyright- contents must not be reproduced in any form\*/  **class** Employee **implements** Serializable {    **private** **static** **final** **long** *serialVersionUID* = 1L;  **private** Integer id;  **private** MyClass myClass ;    **public** Employee(Integer id) {  **this**.id = id;            myClass=**new** MyClass();     }     @Override  **public** String toString() {  **return** "Employee [id=" + id + "]";     }  }  **public** **class** SerializeDeser {  **public** **static** **void** main(String[] args) {            Employee object1 = **new** Employee(8);  **try** {                   OutputStream fout = **new** FileOutputStream("ser.txt");                   ObjectOutput oout = **new** ObjectOutputStream(fout);                   System.*out*.println("Serialization process has started, serializing objects...");                   oout.writeObject(object1);                   System.*out*.println("Object Serialization completed.");                   fout.close();          oout.close();              } **catch** (IOException  e) {                   e.printStackTrace();            }     }  } |

**Answer**. Here intention of interviewer will be to find out whether you know that if any of the member does not implement Serializable than  NotSerializableException is thrown.

/\*OUTPUT

Serialization process has started, serializing objects...

java.io.NotSerializableException: SerDeser10memberNotSer.MyClass

   at java.io.ObjectOutputStream.writeObject0(Unknown Source)

   at java.io.ObjectOutputStream.defaultWriteFields(Unknown Source)

   at java.io.ObjectOutputStream.writeSerialData(Unknown Source)

   at java.io.ObjectOutputStream.writeOrdinaryObject(Unknown Source)

   at java.io.ObjectOutputStream.writeObject0(Unknown Source)

   at java.io.ObjectOutputStream.writeObject(Unknown Source)

   at SerDeser10memberNotSer.SerializeConstructorCheck.main(SerializeConstructorCheck.java:42)

\*/