# Serialization

1.How to make a Java class serializable ?

* Class should implement java.io.serializable interface.
* java.io.serializable interface has no methods, so there will be no impact on the class which implements this interface.
* We can also add special static field "serialVersionUID".

Example -

public class Person implements Serializable {

private static final long serialVersionUID = 20987689172797908L;  
private String name;  
private int age;

//Some more methods..  
}

Now this Person class can be passed from one application to the another via network or through a disk.

2.How to compute serialVersionUID?  
 All Java IDE's support mechanism to generate serialVersionUID. Below are its details,

* Its basically a hash code
* It is computed from the class name
* Interface implemented
* Methods and fields
* Using a SHA algorithim
* Please note that you can have this field (serialVersionUID) present at the compile time, i.e let say in this case you have computed it and then put it inside the class. In such case JVM will use it as it is , with no validation.

3. Tell the difference between private static final and private final ?  
Ans - "private final" should be considered as constant per object / instance. While "private static final" should be considered as constant on the class level. so "private static final" is a constaint which is available to all instances of a class.

4.Tell something about serialization and de-serialization.  
Ans - Object Serialization is a process used to convert the state of an object into a byte stream, which can be persisted into disk/file or sent over the network to any other running Java virtual machine.

The reverse process of creating an object from the byte stream is called deserialization. The byte stream created is platform independent. So, the object serialized on one platform can be deserialized on a different platform.  
  
Serializability can be enabled in your Java class by implementing the java.io.Serializable interface. It is a marker interface that means it contains no methods or fields and only serves to identify the semantics of being serializable

5. What is java.io.NotSerializableException ?  
Ans - if We Are Trying to Serialize a Non-Serializable Object, i.e if class has not implemented java.io.Serializable interface, and we are trying to serialize it .

6. What Is the serialVersionUID?  
Ans - SerialVersionUID is used for version control of an object. The consequence of not specifying serialVersionUID is that when you add or modify any field in the class, then the already-serialized class will not be able to recover because the serialVersionUID was generated for the new class and the old serialized object will be different. The Java serialization process relies on correct serialVersionUID for recovering the state of the serialized object and throws java.io.InvalidClassException in case of serialVersionUID mismatch.

7. What is transient keyword?  
Ans - The transient modifier/keyword is applicable only for variables but not for methods and classes.

At the time of serialization, if we don't want to serialize the value of a particular variable to meet security constraints, then we should declare that variable as transient.

8. Transient Vs. Static ?  
Ans - A static variable is not part of an object state, and hence, it won't participate in serialization. Due to this declaring static variable as transient, there is no use.

9. Final Vs. Transient ?  
Ans - Final variables will be participated in serialization directly by the value. Hence, declaring a final variable as transient causes no impact.

java.io.NotSerializableException: com.pluralsight.corejava8.io.Person

Design Pattern Questions :-

* Behavioral Pattern

1. Template pattern - Actually you define skelton of a function / or method. You typically define method as final, which means that method cannot be over-ridden in its sub-classes. But note that typically you have some steps in the method which is re-defined or being overridden in the sub-classes.

Advantage - Code resue, You basically fix high-level algorithim of your class.

2. Strategy Pattern / or policy pattern - Strategy pattern is also known as policy pattern. Advantages of policy pattern are, - It defines , different behaviour in different classes and so because of that conditional statements are not required. Incorporating new behaviour is easy without changing the application.

3. Command pattern - Below are the features of command pattern. There should be a command interface and also there should be Concrete Command implementing the command interface. We want to de-couple Invoker from the receiver, increasing the decoupling in the code which increases flexibility of code. We also want to store command logs and also want to implement undo. Basically we will have to maintain stack of commands.

4. Chain of responsibility design pattern - Aim of this design pattern is also to achieve loose coupling. So what happens is that Request initiator is not aware of exact request handler and this way system becomes more flexible.

5. Observer Pattern - Observer pattern says that - Just define one-to-many dependency so that when one object changes state, all its dependents are notified and updated automatically. This pattern is also known as "Publish - subscribe" model or "Subject - Observer"

* Structural Pattern

1. Adapter design pattern - With the help of this design pattern, two incompatible interface work together. However please note that inner functionality / or say core functionality of two interfaces should be the same. Adapter pattern is also known as wrapper pattern, and decorator pattern is also known as wrapper pattern.

* + - Two patterns are known as wrapper pattern. Those two patterns are Adapter design pattern and Decorator design pattern.
    - Adapter design pattern lets two incompatible interface work together. But note that we only want two incompatible interface to work together because actually both interfaces are doing the same thing.
  + 2. Facade design pattern - Just provide a unified and simple interface to the client, hiding the complexity. With this definition we can say that Factory and Abstract factory is also kind of facade example.
  + 3. Proxy design pattern - This design pattern provides the control for accessing original object. There are several benefits of this design pattern.