1. **What are inputstream and outputstream?**

Ans:

Inputstream is used to read from a source and outputstream is used to write to the destination.

1. **What is the use of File class in java?**

Ans:

File class is used to get the information about a particular file or folder such as last modified date, length, whether it exists or not etc.

1. **What are Byte streams and Unicode character streams?**

Ans:

Byte Streams

Purpose: Byte streams are used for handling raw binary data like images, audio files, videos, and other non-text data. They read and write data byte by byte.

Classes:

InputStream (for reading) → e.g., FileInputStream

OutputStream (for writing) → e.g., FileOutputStream

Unicode (Character) Streams

Purpose: Unicode streams (character streams) are used for handling text data, ensuring proper encoding and decoding (e.g., UTF-8, UTF-16). They read and write data character by character, making them suitable for processing text files.

Classes:

Reader (for reading) → e.g., FileReader

Writer (for writing) → e.g., FileWriter

1. **What is the use of RandomAccessFile class?**

Ans:

It is used to write and read both inside the file.

1. **What are the steps need to be performed in order write and read primitives inside the file?**

Ans:

In order to write primitives :

FileOutputStream fos=new FileOutputStream(“abc.txt”)

DataOutputStream dos=new DataOutputStream(fos);

dos.writeInt

dos.writeDouble

etc.

in order to read primitives:

FileInputStream fis=new FileInputStream(“abc.txt”);

ObjectInputStream ois=new ObjectInputStream(fis);

ois.readInt

ois.readDouble

etc.

1. **What do you mean by serialization and deserialization?**

Ans:

Serialization means writing java object inside the file or sent over the network and deserialization means reading java object from the file or read from the network.

1. **What are rules for serialization in java?**

Ans:

There are two rules for serialization in java:

1. A class whose object we are going to write inside the file, must implement either Serializable or Externalizable interface.
2. All the instance members of the class must be of serialized type.
3. **What do you mean by an instance member should be of serialized type in java?**

Ans:

Serialized type means one should be able to be converted into sequence of bytes which can be later stored inside the file or sent over the network.

In java Primitives are by default serialized but reference types we have to make serializable either by implementing Serializable or Externalizable interface. Some of the built in classes in java already implement Serializable e.g. String,wrapper classes etc.

1. **What if we don’t follow serialization rules and try to serialize the object ?**

Ans:

In that case “java.io.NotSerializableException” is raised.

1. **What are the steps required to serialize and deserialize object to and from the filesystem?**

Ans:

In order to serialize an object inside the filesystem:

FileOutputStream fos=new FileOutputStream(“abc.txt”);

ObjectOutputStream oos=new ObjectOutputStream(fos);

oos.writeObject(object to be serialized)

In order to deserialize an object from the filesystem:

FileInputStream fis=new FileInputStream(“abc.txt”);

ObjectInputStream ois=new ObjectInputStream(fis);

ois.readObject();

1. **What is the use of transient keyword in java serialization?**

Ans:

The transient keyword in Java is used to **prevent a field from being serialized** when an object is converted into a byte stream. This means that when the object is deserialized, the transient field will be **reset to its default value** instead of restoring its original value.

1. **What happens when we deserialize the object in case of Serializable interface?**

Ans:

When we invoke readObject() in order to deserialize the object:

1. A new object is created on the heap
2. All the instance members are allocated memory inside the object
3. Constructor does not get called

[ if the constructor gets called we will not get those values i.e. latest values which were given to a variable/s just before serialization. We will get old values ]

1. All the non-transient members get the value from the file system ( previously stored)
2. Transient members get the default value as per java rule. E.g.primitives get 0 and reference types get null.
3. **Given:**

**class Parent {}**

**class Child extends Parent implements Serializable{}**

**what will happen if we try to serialize an instance of Child?**

Ans:

Only child class members will be serialized and not parent class members.

1. **Why java.lang.Object does not implement Serializable?**

Ans:

if Object class were Serializable,

then each and every class in java would be serialized by default. In case of some classes like Thread, Socket, Database connection etc. serialization does not make any sense.

Threads, Sockets, Database Connections etc. are tightly linked to the JVM’s runtime environment, which means:

They depend on the OS and JVM state (e.g., a thread’s execution stack, a socket’s network session, a database connection’s session).

If you serialize them in one JVM and deserialize in another JVM, they won’t work because the new JVM has a completely different runtime state.

1. **What gets stored inside the file in serialization?**

Ans:

a) metadata (only information including "SerialVersionUID")

from sub class to base class recursively till "Object" is not encountered

b) actual data (instance members)

from base class to sub class

c) metadata of "has a" till u don't get "reference type" as "has a" member

actual data of "has a".

1. **What is serialversionuid in serialization and How does it work?**

Ans:

serialVersionUID is a unique identifier used during Java object **serialization** and **deserialization** to verify that a serialized object's class definition matches the class definition present at the time of deserialization.

When an object is serialized, Java stores the serialVersionUID of the class along with the serialized data. During deserialization:

* If the serialVersionUID of the serialized object **matches** the serialVersionUID of the current class, deserialization succeeds.
* If it **does not match**, Java throws an InvalidClassException, preventing incompatible class versions from being used.

1. **Why is it highly recommended to declare serialversionuid in case of serialization?**

Ans:

serialVersionUID is Important especially for Cross-Platform Serialization.

When you **serialize an object on one OS** (e.g., Windows) and **deserialize it on another OS** (e.g., Linux or Mac), the **JVM versions might be different**. Since Java uses **default computation for serialVersionUID if not explicitly set**, it may generate different values on different JVMs. If the automatically generated serialVersionUID differs between JVMs, deserialization will fail with InvalidClassException.

*When we declare serialversionuid in the class which implement Serializable, the same serialVersionUID is used and JVM will not generate it* , no matter we serialize-deserialize on same JVMs or different.

Conclusion:

**Different JVMs may compute different serialVersionUIDs for the same class** if not explicitly set.  
✔ **If the generated IDs don’t match, deserialization fails with InvalidClassException.**  
✔ **Manually defining serialVersionUID ensures object compatibility across OS and JVM versions.**

1. **What is the difference between Serializable and Externalizable?**

Ans:

Both Serializable and Externalizable are used for **object serialization**, but they differ in terms of **control, performance, and flexibility**.

* 1. **Definition**
* Serializable: A **marker interface** (no methods) that enables **default serialization**.
* Externalizable: Extends Serializable and provides **custom serialization logic**.
  1. **Control**
* Serializable: **Automatic** (Java handles serialization internally).
* Externalizable: **Manual** (You must define how the object is serialized/deserialized).
  1. **Performance**
* Serializable: **Slower** (reflective serialization, more metadata).
* Externalizable: **Faster** (only required fields are written, no extra metadata).
  1. **Methods Required**
* Serializable: No methods to implement.
* Externalizable: Must implement writeExternal() and readExternal().
  1. **Flexibility**
* Serializable: **Limited** (cannot exclude fields unless marked transient).
* Externalizable: **Complete control** over how data is serialized and deserialized.
  1. **Use Case**
* Serializable: When **default serialization is sufficient**.
* Externalizable: When you need **custom, optimized serialization** (e.g., avoiding unnecessary data).

1. **What are private writeObject and readObject methods in java serialization?**

Ans:

In Java serialization, if a class implements Serializable, Java automatically serializes all non-transient fields. However, sometimes you need custom serialization logic (e.g., encrypting data, handling transient fields).

For this, Java allows defining private writeObject() and readObject() methods in a Serializable class.

These methods must be private because they are meant only for the class inside which they are defined. If we define them as non-private, they will simply be ignored by java serialization and deserialization process.

1. **What happens when you deserialize the object in case of Externalizable?**

Ans:

a) new object gets created

b) instance members are allocated memory

c) default constructor gets called

[ It is true that default constructor gives us default initial values, But Externalizable gives us one more chance to reinitialize our members with the help of file info ie. using "readExternal()" method ]

1. readExternal() method will get called which initializes non-static and non-transient members with the help of file info.
2. **What all things are checked when we serialize an object by invoking “writeObject(object to be serialized)” on ObjectOutputStream?**

Ans:

when u call "writeObject" , it will check whether the class (whose object you have passed ) implements "Serializable" or "Externalizable"

if it implements "Serializable"

it will check whether

u have defined "private writeObject"

if yes

it will invoke it

if no

it will go for def. serialization

else if it implements "Externalizable"

it will invoke "writeExternal()"

else

NotSerializedException

1. **What all things are checked when we deserialize an object by invoking “readObject()” on ObjectInputStream?**

Ans:

when u call "readObject",it will check whether class has implements "Serializable" or "Externalizable".

if it implements "Serializable"

it will check whether

u have defined "private readObject"

if yes

it will invoke it

if no

it will go for default deserialization

if it implements "Externalizable"

invoke default constructor

invoke "readExternal()" method

1. **Why is it necessary to implement Serializable in the outer class also when we serialize inner class?**

Ans:

It is true that when we want to serialize object of inner class, along with inner class, outer class also should implement Serializable interface. It’s because in case of inner class compiler secretly puts reference of outer class inside inner class object and as per java serialization rule, if not only the class (inner class) should implement Serializable but also all the instance members (reference of outer class) must be of serialized type.

1. **Why is it not necessary to implement Serializable in the outer class also when we serialize static nested class?**

Ans:

Object of static nested class can be created without the object of outer class. That’s why object of static nested class does not contain any reference of outer class and Hence it is not necessary to implement Serializable in the outer class , when we want to serialize object of static nested class.

1. **How many ways object can be created in Java?**

Ans:

In java there are total 5 ways in which object can be created:

a) Using new keyword

b) Using newInstance() method of class "Class"

c) Using newInstance() method of “Constructor” class

d) Using clone() method

e) Using deserialization

1. **In case of Serializable, entire object graph is serialized. What does that mean?**

Ans:

The **object graph** in Java serialization means that **when an object is serialized, all objects it references (directly or indirectly) are also serialized**.

This **also includes all parent classes** (considering multilevel inheritance) of the object **until java.lang.Object class is reached**. However, there are some important rules regarding **how parent classes are handled** during serialization.

[ if is-a is not of serialized type , no issues it is ignored during serialization. If has-a is not of serialized type, you get NotSerializationException.]