Serialization is converting an object in memory into an array of bytes, which can be used to transfer through streams or saved as a file.

Why do we need Serialization?
When we have large data set, let us suppose an object of a class, sometimes we need to save or transfer that data systematically. In that case we use Serialization.

How can we achieve Serialization in Java?
Java has got a Serializable interface that enables us to Serialize an Object of a class.

Requirement to use Serializable on a Class in Java
1. Make a POJO class as you do it normally.
2. That class has to implement an Interface called java.io.Serializable Interface.
3. Now this object can be converted to bytes/file by using *writeObject()* available in ObjectOutputStream class.
4. This Interface 'java.io.Serializable', hasn't got any method, it's a Marker Interface.

Marker interface
Marker Interface is a special interface in Java without any field and method. Marker interface is used to inform compiler that the class implementing it has some special behaviour or meaning. Some example of Marker interface are,

java.io.Serializable
java.lang.Cloneable
java.rmi.Remote
java.util.RandomAccess

All these interfaces does not have any method and field. They only add special behavior to the classes implementing them. However marker interfaces have been deprecated since Java 5, they were replaced by Annotations. Annotations are used in place of Marker Interface that play the exact same role as marker interfaces did before.

'transient' Keyword
While serializing an object, if we don't want certain data member of the object to be serialized we can mention it transient. transient keyword will prevent that data member from being serialized. This saves memory and resources.

Code examples for Serialization using Serializable Interface in Java

1. A function to serialize an Object that returns a byte array in Java -

// you pass the Object of a Class that implements Serializable as an argument here

```
public byte[] serializeObject(Object o) {
   ByteArrayOutputStream bos = new ByteArrayOutputStream();

   try {
     ObjectOutput out = new ObjectOutputStream(bos);
     out.writeObject(o);
     out.close();

     // Get the bytes of the serialized object
     byte[] buf = bos.toByteArray();

     return buf;
   } catch(IOException ioe) {
     Log.e("serializeObject", "error", ioe);

     return null;
   }
 }
```

2. Serializing an Object of a Java Class and storing it as a file -

// the Class '*Student*', implements Serializable:

```
Student obj = new Student(101, 25);
FileOutputStream fos = new FileOutputStream("Student.ser");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(obj);
oos.close();
fos.close();
System.out.println("Serialization Done!!");
```

Deserialization

is taking bytes or a file and converting them back into an Object in memory so that we can use that object in our code.

Requirement to use Deserialization in Java

1. Make the POJO class as you do normally.
2. That class has to implement an Interface called java.io.Serializable Interface.
3. Now any matching file can be converted to object of this class using *readObject()* method in ObjectOutputStream class.

Code Example for doing Deserialization in Java

```
FileInputStream fis = new FileInputStream("student.ser");
ObjectOutputStream ois = new ObjectOutputStream(fis);
si = (studentinfo)ois.readObject();
```

What are the use cases of doing Serialization using Serializable Interface on Android ?

1. For saving an Object to some file, same as you do in Java
2. For passing an Object in Intent -

   You make a class to Implement Serializable same as in Java and the object of that class can be passed via Intent to another activity.

```
// Student is a Model class that implements Serializable
Student stud = new Student();
Intent intent = (getBaseContext(), SomeActivity.class);
intent.putExtra("StudentObject", stud);
startActivity(intent);
```

Advantages of doing Serialization on android using Serializable Interface

- It is easier to implement.
- It is faster to implement

Disadvantages of doing Serialization on android using Serializable Interface

- The Serialization process is slow. It affects the performance when the fields are more or data set is more.
- Serializable interface creates a lot of temporary objects and cause quite a bit of garbage collection.

To solve these issues of Serializable Interface, Google has come up with Parcelable Interface to do Serialization on Android only. Parcelable is faster than serializable interface. Sometimes about 10 times faster.

Using Parcelable on Android

(Implementing Parcelable is little complicated, that's why I have given the boilerplate code here)

1. The Model class

```
public class MyModel implements Parcelable {

    /** 1.
     * These 'variables declaration' and the 'constructor' below are the only extra thing we need to add.
     * Everything else is auto generated by alt+enter
     */
    String name, age;
    // Instead of a Constructor to initialize the variables, you can even use Getters and Setters
    public MyModel(String namee, String agee) {
        this.name = namee;
        this.age = agee;
    }


    /** 2.
     * These things below are auto generated by alt+enter by hovering on the Class name and Parcelable keyword.
     */
    public MyModel(Parcel in) {
        name = in.readString();
        age = in.readString();
    }
    public static final Creator<MyModel> CREATOR = new Creator<MyModel>() {
        @Override
        public MyModel createFromParcel(Parcel in) {
            return new MyModel(in);
        }
        @Override
        public MyModel[] newArray(int size) {
            return new MyModel[size];
        }
    };
    @Override
    public int describeContents() {
        return 0;
    }
    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeString(name);
        dest.writeString(age);
    }
}
```

When you want to send data of the type ‘MyModel’ in Intent from *MainActivity* to *SecondActivity* -

2. Code in MainACtivity

```
MyModel dataToSend = new MyModel("namee1", "agee1");
Intent i = new Intent(this, SecondActivity.class);
i.putExtra("myData", dataToSend);
startActivity(i);
```

3. Code in SecondActivity

```
Intent i = getIntent();
MyModel model = i.getExtras().getParcelable("myData");
Log.i("data_received", model.name + " " + model.age);
```