Externalizable interface in Java

Externalization serves the purpose of custom Serialization, where we can decide what to store in stream.

Externalizable interface present in java.io, is used for Externalization which extends Serializable interface. It consist of two methods which we have to override to write/read object into/from stream which are-

```
// to read object from stream
void readExternal(ObjectInput in)

// to write object into stream
void writeExternal(ObjectOutput out)
```

Key differences between Serializable and Externalizable

- **Implementation :** Unlike Serializable interface which will serialize the variables in object with just by implementing interface, here we have to explicitly mention what fields or variables you want to serialize.
- **Methods :** Serializable is marker interface without any methods. Externalizable interface contains two methods: writeExternal() and readExternal().
- **Process:** Default Serialization process will take place for classes implementing Serializable interface. Programmer defined Serialization process for classes implementing Externalizable interface.
- **Backward Compatibility and Control:** If you have to support multiple versions, you can have full control with Externalizable interface. You can support different versions of your object. If you implement Externalizable, it's your responsibility to serialize super class.
- **public No-arg constructor:** Serializable uses reflection to construct object and does not require no arg constructor. But Externalizable requires public no-arg constructor.

Below is the example for Externalization-

```
// Java program to demonstrate working of Externalization interface
import java.io.*;
class Car implements Externalizable {
    static int age;
    String name;
    int year;

    public Car()
    {
        System.out.println("Default Constructor called");
    }

    Car(String n, int y)
    {
        name = n;
        year = y;
        age = 10;
    }
    public void writeExternal(ObjectOutput out)
                              throws IOException
    {
        out.writeObject(name);
        out.writeInt(age);
        out.writeInt(year);
    }
    public void readExternal(ObjectInput in)
       throws IOException, ClassNotFoundException
    {
        name = (String)in.readObject();
        year = in.readInt();
        age = in.readInt();
    }
    public String toString()
    {
        return ("Name: " + name + "\n" +
                "Year: " + year + "\n" +
                "Age: " + age);
    }
}

public class ExternExample {
    public static void main(String[] args)
    {
        Car car = new Car("Shubham", 1995);
        Car newcar = null;

        // Serialize the car
        try {
            FileOutputStream fo = new FileOutputStream("gfg.txt");
            ObjectOutputStream so = new ObjectOutputStream(fo);
            so.writeObject(car);
            so.flush();
        }
        catch (Exception e) {
            System.out.println(e);
        }

        // Deserializa the car
        try {
            FileInputStream fi = new FileInputStream("gfg.txt");
            ObjectInputStream si = new ObjectInputStream(fi);
            newcar = (Car)si.readObject();
        }
        catch (Exception e) {
            System.out.println(e);
        }

        System.out.println("The original car is:\n" + car);
        System.out.println("The new car is:\n" + newcar);
    }
}
```

Output:


```
Default Constructor called
The original car is:
Name: Shubham
Year: 1995
Age: 1995
The new car is:
Name: Shubham
Year: 10
Age: 1995
```

In the example, class Car has two methods- writeExternal and readExternal. So, when we write "Car" object to OutputStream, writeExternal method is called to persist the data. The same applies for readExternal method.

When an Externalizable object is reconstructed, an instance is created first using the public no-argument constructor, then readExternal method is called.So, it is manadatory to provide no-argument constructor.

When an object implements Serializable interface, is serialized or deserialized, no constructor of object is called and hence any initialization which is implemented in constructor can't be done.