```
// POJO class Exmaple
// Example POJO class
public class Person {
    private String name;
    private int age;

    // Constructor
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Getters
    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    // Setters
    public void setName(String name) {
        this.name = name;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}

public class Main {
    public static void main(String[] args) {
        // Create a Person object
        Person person = new Person("Alice", 30);

        // Access the data using getters
        System.out.println("Name: " + person.getName());
```

```
        System.out.println("Age: " + person.getAge());

        // Modify the data using setters
        person.setName("Bob");
        person.setAge(35);

        // Print the updated object
        System.out.println(person); // Output: Person{name='Bob', age=35}
    }
}
```

JAVA Bean

```
public class PersonBean implements Serializable {
    private String name;
    private int age;

    public PersonBean() {
        // Default no-argument constructor
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}


public class Main {
    public static void main(String[] args) {
        // Create a Person object
        Person person = new Person("Alice", 30);

        // Access the data using getters
        System.out.println("Name: " + person.getName());
        System.out.println("Age: " + person.getAge());

        // Modify the data using setters
        person.setName("Bob");
        person.setAge(35);

        // Print the updated object
        System.out.println(person); // Output: Person{name='Bob', age=35}
    }
}
```

Serialization and Deserialization

```
package com.journaldev.serialization;

import java.io.Serializable;

public class Employee implements Serializable {
//	private static final long serialVersionUID = -6470090944414208496L;
	private String name;
	private int id;
	transient private int salary;
//	private String password;

	@Override
	public String toString(){
		return "Employee{name="+name+",id="+id+",salary="+salary+"}";
	}

	//getter and setter methods
	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}

	public int getSalary() {
		return salary;
	}

	public void setSalary(int salary) {
		this.salary = salary;
	}

//	public String getPassword() {
```

```
//		return password;
//	}
//
//	public void setPassword(String password) {
//		this.password = password;
//	}

}
```

Notice that it's a simple java bean with some properties and getter-setter methods. If you want an object property to be not serialized to stream, you can use transient keyword like I have done with salary variable. Now suppose we want to write our objects to file and then deserialize it from the same file. So we need utility methods that will use ObjectInputStream and ObjectOutputStream for serialization purposes.

```
package com.journaldev.serialization;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
public class SerializationUtil {
	// deserialize to Object from given file
public static Object deserialize(String fileName) throws IOException, ClassNotFoundException
{
		FileInputStream fis = new FileInputStream(fileName);
		ObjectInputStream ois = new ObjectInputStream(fis);
		Object obj = ois.readObject();
		ois.close();
		return obj;
}

// serialize the given object and save it to file
public static void serialize(Object obj, String fileName) throws IOException {
		FileOutputStream fos = new FileOutputStream(fileName);
		ObjectOutputStream oos = new ObjectOutputStream(fos);
		oos.writeObject(obj);
		fos.close();
	}
}
```

Notice that the method arguments work with Object that is the base class of any java object. It's written in this way to be generic in nature. Now let's write a test program to see Java Serialization in action.

```
package com.journaldev.serialization;
```

```
import java.io.IOException;
public class SerializationTest {
	public static void main(String[] args) {
		String fileName="employee.ser";
		Employee emp = new Employee();
		emp.setId(100);
		emp.setName("Pankaj");
		emp.setSalary(5000);

		//serialize to file
		try {
			SerializationUtil.serialize(emp, fileName);
		} catch (IOException e) {
			e.printStackTrace();
			return;
		}

		Employee empNew = null;
		try {
			empNew = (Employee) SerializationUtil.deserialize(fileName);
		} catch (ClassNotFoundException | IOException e) {
			e.printStackTrace();
		}

		System.out.println("emp Object::"+emp);
		System.out.println("empNew Object::"+empNew);
	}
}
```

When we run above test program for serialization in java, we get following output.

```
emp Object::Employee{name=Pankaj,id=100,salary=5000}
empNew Object::Employee{name=Pankaj,id=100,salary=0}
```

serialuid.java

```
import java.io.*;

class Emp implements Serializable {
private static final long serialversionUID =
                        129348938L;
   transient int a;
   static int b;
   String name;
   int age;

   // Default constructor
public Emp(String name, int age, int a, int b)
   {
      this.name = name;
      this.age = age;
      this.a = a;
      this.b = b;
   }

}

public class SerialExample {
public static void printdata(Emp object1)
   {

      System.out.println("name = " + object1.name);
      System.out.println("age = " + object1.age);
      System.out.println("a = " + object1.a);
      System.out.println("b = " + object1.b);
   }

public static void main(String[] args)
   {
      Emp object = new Emp("ab", 20, 2, 1000);
      String filename = "shubham.txt";

      // Serialization
      try {

         // Saving of object in a file
         FileOutputStream file = new FileOutputStream
                              (filename);
         ObjectOutputStream out = new ObjectOutputStream
                              (file);
```

```
    // Method for serialization of object
    out.writeObject(object);

    out.close();
    file.close();

    System.out.println("Object has been serialized\n"
                    + "Data before Deserialization.");
    printdata(object);

    // value of static variable changed
    object.b = 2000;
}

catch (IOException ex) {
    System.out.println("IOException is caught");
}

object = null;

// Deserialization
try {

    // Reading the object from a file
    FileInputStream file = new FileInputStream
                           (filename);
    ObjectInputStream in = new ObjectInputStream
                           (file);

    // Method for deserialization of object
    object = (Emp)in.readObject();

    in.close();
    file.close();
    System.out.println("Object has been deserialized\n"
                     + "Data after Deserialization.");
    printdata(object);

    // System.out.println("z = " + object1.z);
}

catch (IOException ex) {
    System.out.println("IOException is caught");
}

catch (ClassNotFoundException ex) {
```

```
            System.out.println("ClassNotFoundException" +
                               " is caught");
        }
    }
}
```

Run on IDE
Output:

```
Object has been serialized
Data before Deserialization.
name = ab
age = 20
a = 2
b = 1000
Object has been deserialized
Data after Deserialization.
name = ab
age = 20
a = 0
b = 2000
```

Java Serialization with Inheritance (IS-A Relationship)

Isaserialize.java

```
import java.io.Serializable;
class Person implements Serializable{
 int id;
 String name;
 Person(int id, String name) {
  this.id = id;
  this.name = name;
 }
}

class Student extends Person{
 String course;
 int fee;
 public Student(int id, String name, String course, int fee) {
  super(id,name);
  this.course=course;
  this.fee=fee;
 }
}
```

Java Serialization with Aggregation (HAS-A Relationship)

Hasaserialize.java

```
class Address{
 String addressLine,city,state;
 public Address(String addressLine, String city, String state) {
  this.addressLine=addressLine;
  this.city=city;
  this.state=state;
 }
}
import java.io.Serializable;
public class Student implements Serializable{
 int id;
 String name;
 Address address;//HAS-A
 public Student(int id, String name) {
  this.id = id;
  this.name = name;
 }
}
```

Since Address is not Serializable, you can not serialize the instance of Student class.

Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.

Transient.java

```
import java.io.Serializable;
public class Student implements Serializable{
 int id;
 String name;
 transient int age;//Now it will not be serialized
 public Student(int id, String name,int age) {
  this.id = id;
  this.name = name;
  this.age=age;
 }
}
```

Now write the code to serialize the object.

```
import java.io.*;
class PersistExample{
 public static void main(String args[])throws Exception{
  Student s1 =new Student(211,"ravi",22);//creating object
  //writing object into file
  FileOutputStream f=new FileOutputStream("f.txt");
  ObjectOutputStream out=new ObjectOutputStream(f);
  out.writeObject(s1);
  out.flush();

  out.close();
  f.close();
  System.out.println("success");
 }
}
```

Output:

```
success
```

Now write the code for deserialization.

```
import java.io.*;
class DePersist{
 public static void main(String args[])throws Exception{
  ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt"));
  Student s=(Student)in.readObject();
  System.out.println(s.id+" "+s.name+" "+s.age);
  in.close();
 }
}
```

211 ravi 0

As you can see, printing age of the student returns 0 because value of age was not serialized.