13. OBJECT SERIALIZATION

1. Serialization Concept:

It is the process of preserving the state of object in to file in terms of bytes. It is achieved via implementing Serializable interface.
Serializable interface is a marker interface which has no any method signature. It adds the speciality to the classes which are implemening it. If a class implements Serializable interface then its state of object can be written into file and later it can be read. Such file can be transfered in the network.

2. Need of Object Serialization:

Serialization is usually used When the need arises to send your data over network or stored in files. The problem is Network infrastructure and Hard disk are hardware components that understand bits and bytes but not JAVA objects. Serialization is the translation of your Java object's values/states to bytes to send it over network or save it.

3. Advantages of Java Object Serialization:

- Maintain the Java object type and safety properties in the serialized form.
- Be extensible to support marshaling and unmarshaling as needed for remote objects.
- Be extensible to support simple persistence of Java objects.
- Require per class implementation only for customization.
- Allow the object to define its external format.

For example:

```
public class Employee implements Serializable {
        private int eID;
        private String ePassword;
        private String eName;

}
```

The value of eID, ePassoword and eName at one state of and object, can be stored into the file. If we do not want to perssist the particular variable value for security reasons then declare the variable as transient. The value of variable declared as transient will not be persisted while saving the state of object. It is non-access modifier can be given to variable only.

```
public class Employee implements Serializable {
        private int eID;
        private transient  String ePassword;
        private String eName;
```

```
}
```

4. Serializing the object:

```
public interface ObjectOutput extends DataOutput {
   public void writeObject(Object obj) throws IOException;
   public void write(int b) throws IOException;
   public void write(byte b[]) throws IOException;
   public void write(byte b[], int off, int len) throws IOException;
   public void flush() throws IOException;
   public void close() throws IOException;
}
```

To write the state of an object to a file we use following method of **ObjectOutputStream** class.

```
public final void writeObject(Object obj) throws IOException
```

This method throws following exception.
InvalidClassException - Something is wrong with a class used by serialization.
NotSerializableException-Some object to be serialized does not implement the java.io.Serializable interface.
IOException - Any exception thrown by the underlying OutputStream. (Checked Exception)

5. Deserialing the object:

```
public interface ObjectInput extends DataInput {
   public Object readObject() throws ClassNotFoundException, IOException;
   public int read() throws IOException;
   public int read(byte b[]) throws IOException;
   public int read(byte b[], int off, int len) throws IOException;
   public long skip(long n) throws IOException;
   public int available() throws IOException;
   public void close() throws IOException;
}
```

To read the state of an object from file we use following method of **ObjectInputStream** class.

```
public final Object readObject()throws IOException, ClassNotFoundException
```

This method throws following exceptions
ClassNotFoundException - Class of a serialized object cannot be found. (Checked Exception)
InvalidClassException - Something is wrong with a class used by serialization.
StreamCorruptedException - Control information in the stream is inconsistent.
OptionalDataException - Primitive data was found in the stream instead of objects.
IOException - Any of the usual Input/Output related exceptions. (Checked Exception)

13. OBJECT SERIALIZATION

Example:

File 1: Employee.java : The object of Employee class to be serialized

```
package serialization;
import java.io.Serializable;
public class Employee implements Serializable{
	private int eID;
	private String eName;
	private transient String ePassword;
	private double eSalary;

	public int geteID() {
		return eID;
	}
	public void seteID(int eID) {
		this.eID = eID;

	}
	public String geteName() {
		return eName;
	}
	@Override
	public String toString() {
		return "Employee [eID=" + eID + ", eName=" + eName + ", ePassword="
				+ ePassword + ", eSalary=" + eSalary + "]";
	}
	public void seteName(String eName) {
		this.eName = eName;
	}
	public String getePassword() {
		return ePassword;
	}
	public void setePassword(String ePassword) {
		this.ePassword = ePassword;
	}
	public double geteSalary() {
		return eSalary;
	}
	public void seteSalary(double eSalary) {
		this.eSalary = eSalary;
	}
```

```
}
```

File 2: ObjectSerializationDemo.java : Logic for Serialization and Deserialization

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.OutputStream;

public class ObjectSerializationDemo {
	public static void main(String[] args) {
		try
		{
		Employee emp=new Employee();
		emp.seteID(101);
		emp.seteName("Jyoti Prakash Ulape");
		emp.seteSalary(54000);

		//write state of an object into file
		OutputStream write=new FileOutputStream("serialize.ser");
		ObjectOutputStream writefile=new ObjectOutputStream(write);
		writefile.writeObject(emp);

		//reading state of the object from file
		InputStream read=new FileInputStream("serialize.ser");
		ObjectInputStream readfile=new ObjectInputStream(read);
		Employee empread=(Employee)readfile.readObject();
		System.out.println("Employee information read from the file\n: "+empread);

		//ePassword field is transient so it is not saved...
		}catch(Exception e)
		{
			e.printStackTrace();
		}
	}
}
```

Output:

```
Employee information read from the file
ID: Employee [eID=101, eName=Jyoti Prakash Ulape, ePassword=null, eSalary=54000.0]
```

6. Important Ponits to Remember:

1. transient variable will be set to its default value while serializing the object.
2. If we try to write the state of non-serialized object then it gives **NotSerializableException** type exception.
3. If a serialized object contains reference to non serialized object, then while writing the state is gives **NotSerializableException** type exception.

ASSIGNMENTS

1. Serialize the accounts of 10 users and deserialize and display the accounts at destination.
2. Serialize the same above object but do not serialize the password of the account in above program.
3. Implement a program of HAS-A relationship in between Employee and Address (Employee HAS-A Address). Try to serialize the Employee object.
4. Serialize the same above object but do not serialize the Address of employee.