**Serialization**

Saving state of an object to a file is **Serialization**. (Process of converting an object from Java supported form to file supported form or network supported form is **Serialization** in Heap memory).

Process of reading state of an object to a file is **Deserialization.** (Process of converting an object from file supported form or network supported form to from Java supported form is **Deserialization)**.

**How to Serialize**

1. Create **FileOutputStream** (Writes binary data to file).
2. Create an **ObjectOutputStream** to write the above **FileOutputStream** (Converts object to binary data).
3. Then the object can be written to a file.

**How to Deserialize**

1. Create **FileInputStream** (File data to binary data).
2. Create **ObjectInputStream** (Binary data to Object).
3. Then Object is obtained.

**Packages required**

1. Java.IO

**Ex**

class Dog **implements Serializable** {

int i = 10;

int j = 20;

}

class SerialiseDemo{

public static void main(String[] args) **throws Exception**{

//Serialisation

Dog d1 = new Dog();

FileOutputStream fos = new FileOutputStream("file.ser");

//Java can take any file extension. (anyFile.anyExtension)

ObjectOutputStream oos = new ObjectOutputStream(fos);

oos.writeObject(d1);

//DeSerialisation

FileInputStream fis = new FileInputStream("file.ser");

ObjectInputStream ois = new ObjectInputStream(fis);

Dog d2 = (Dog)ois.readObject();

System.out.println(d2.i +” ”+ d2.j); }

}

**Note:**

1. If Dog class does not implement Serializable, there wont be any compilation error. But it will throw NotSerializableException.
2. main() must throw or handle the required checked exception.
3. Serializable interface is a Marker interface (does not have ant methods).
4. An object can be Serializable if it implements Serializable interface.

**Transient**

1. This modifier is applicable only for **Variables.**
2. If any data that should not be stored to meet security constraints, that variable should be made transient.
3. At the time of Serialization JVM will check if any variable is **Transient**. If there are any, JVM will ignore the original value of transient variable and will save the default value.

Ex.

class Dog **implements Serializable** {

int i = 10;

transient int j = 20;

}

Stored value: i = 10, j = 0.

1. **Static vs Transient:**

If **transient** variables are made **static**, they belong to class and hence there will not be Serialization. It is useless to make it transient

Ex. class Dog implements Serializable {

int i = 10;

transient static int j = 20;

}

Stored value i = **10**; j = **20;**

1. **Final vs transient**

All final variables are replaced by its **value** in runtime. Since variables are replaced by values, there will be no effect of transient keyword on these variables.

Ex.

class Dog **implements Serializable** {

transient final int i = 10;

int j = 20;

}

Stored value: i = 10, j = 20.

|  |  |
| --- | --- |
| Input | Output |
| int i = 10;  int j = 20; | 10..20 |
| transient int i = 10;  int j = 20; | 0…20 |
| transient static int i = 10;  transient int j = 20; | 10…20 |
| transient int I = 10;  transient final int j = 20; | 0…20 |

**Serializing Multiple Objects**

1. Any number of objects can be Serialized into a file but order of De-Serialization should be same as Serialization
2. If order of Serialization is unknown then **instanceOf** is used.

**Ex**

FileInputStream fis = new FileInputStream(“file.ser”);

ObjectInputStream ois = new ObjectInputStream(fis);

Object obj = ois.readObject();

if (obj instanceOf Dog){

Dog d1 = (Dog) obj;

}

if (obj instanceOf Cat){

Cat c1 = (Cat) obj; }

**Object Graph**

When Serializing an object, set of all objects which are reachable from that object will be serialized automatically.

This is **Object Graph**.

To run successfully, all underlying objects should implement **Serializable**

class Dog **implements Serializable** {

Cat c = new Cat();

}

class Cat **implements Serializable** {

Rat r = new Rat();

}

class Rat **implements Serializable** {

int j = 20;

}

**Customized Serialization**

In default Serialization, there is chance of Data loss due to transient variable. To recover this Customized Serialization is required

class Account **implements Serializable** {

String username = “name”;

transient String *pwd* = “password”;

}

Class CustomSerializationDemo{

p s v main(String[] args){

Account acc = new Account();

//Serialize

//DeSerialize

System.out.println(acc.username + “ … “+ acc.*pwd*);

}

}

Output: name … null