Serializarea obiectelor

- **De ce serializare ?** Sunt multe situatii in care datele cu care lucreaza un program trebuie sa aiba o durata de viata mai mare decat a programului care le-a creat – trebuie sa supravietuiasca in afara spatiului de adrese a unei masini virtuale Java
- Simpla parcurgere a variabilelor unui obiect si salvarea lor una dupa alta nu este de ajuns (pentru ca se doreste si salvarea unei informatii privind tipul obiectului de care apartin); algoritmul e inca si mai complicat daca se doreste salvarea campurilor clasei parinte sau in cazul in care unele campuri sunt referinte ale unor alte obiecte.
- Obiectele serializabile implementeaza **Java.io.Serializable** sau **Java.io.Externalizable**
- Transferul obiectelor se realizeaza prin intermediul unor fluxuri de obiecte; clasele care reprezinta aceste fluxuri sunt **ObjectInputStream** respectiv **ObjectOutputStream**
- **ObjectInputStream** respectiv **ObjectOutputStream** implementeaza indirect interfetele DataInput respectiv DataOutput; de asemenea implementeaza interfetele ObjectInput respectiv ObjectOutput, oferind metode pentru citirea respectiv scrierea obiectelor.

Serializarea obiectelor

SimpleObject.java

```
import java.io.Serializable;

public class SimpleObject implements Serializable {
    // Nota: variabilele cu atributul transient ca si
    // variabilele statice nu se pot salva la serializare
    public static int n1;
    public String name = "implicit";
    private transient int n2;
    private transient int n3;
    public int n4;

    public SimpleObject(String name, int n1, int n2, int n3, int n4) {
        this.name = name;
        SimpleObject.n1 = n1;
        this.n2 = n2;
        this.n3 = n3;
        this.n4 = n4;
    }
    public String toString() {
        return new String("Name = "
                + name + " n1 = " + n1  + " n2 = " + n2 + " n3 = " + n3 + " n4 = " + n4);
    }
    public void print(){
        System.out.println("My Simple object");
    }
}
```

Serializarea obiectelor

SaveObject.java

```
import java.io.*;

public class SaveObject {

    static SimpleObject o = new SimpleObject("my object",10,20,30,40);

    public static void main (String args[]) {
        try {
            FileOutputStream fout = new FileOutputStream("test");
            ObjectOutputStream sout = new ObjectOutputStream (fout);
            sout.writeObject(o);
            System.out.println("Object wrote: " + o);
            o.print();
            sout.flush();
            sout.close();
            fout.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

}
```

Object wrote: Name = my object n1 = 10 n2 = 20 n3 = 30 n4 = 40
My Simple object

Serializarea obiectelor

RestoreObject.java

```
import java.io.*;

public class RestoreObject {

    static SimpleObject o;

    public static void main (String args[]) {
        try {
            FileInputStream fin = new FileInputStream("test");
            ObjectInputStream sin = new ObjectInputStream(fin);
            try {
                o = (SimpleObject)sin.readObject();
            } catch (ClassNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            System.out.println("Object read: " + o);
            o.print();
            sin.close();
            fin.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

Object read: Name = my object **n1 = 0 n2 = 0 n3 = 0** n4 = 40
My Simple object

Variabilele cu atributul transient si cele de tip static nu se salveaza la serializare

La serializare si deserializare nu trebuie sa folosim aceeasi clasa, ci doar o clasa compatibila
In acest exemplu se salveaza si se restaureaza un arbore de obiecte

In urmatorul exemplu vom salva si restaura un graf de obiecte;

Serializarea obiectelor: salvarea unui graf de obiecte

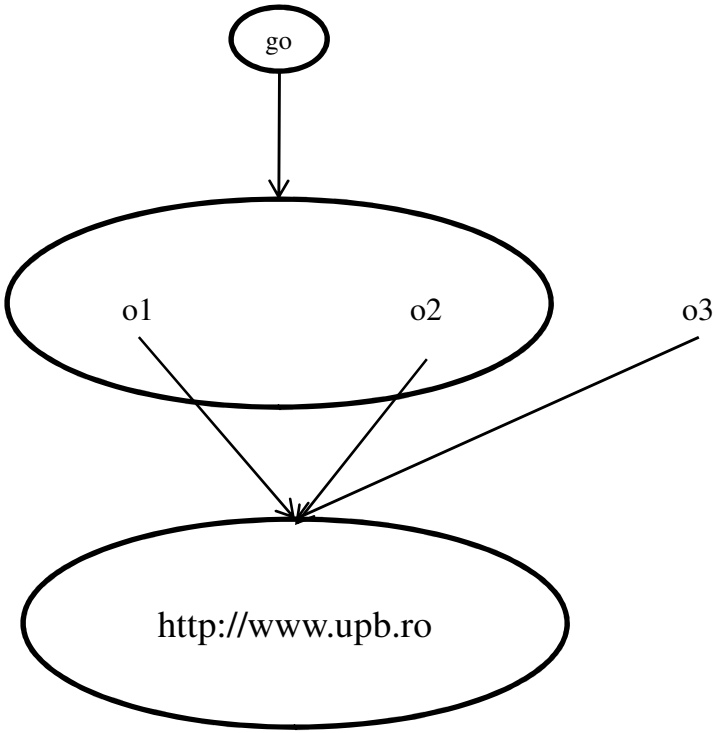
GraphObject.java

```
import java.io.*;

public class GraphObject  implements Serializable {

    public URL o1;
    public URL o2;

    public GraphObject(URL o1, URL o2) {
        this.o1 = o1;
        this.o2 = o2;
    }
    public String toString() {
        return new String("o1 = " + o1 + " o2 = " + o2);
    }
}
```



Deserializarea trebuie sa refaca structura initiala a grafului de obiecte

Serializarea obiectelor: salvarea unui graf de obiecte

SaveObject.java

```
import java.io.*;
import java.net.*;

public class SaveObject {
    static GraphObject go;
    public static void main (String args[]) {
        try {
            FileOutputStream fout = new FileOutputStream("test");
            ObjectOutputStream sout = new ObjectOutputStream (fout);
            URL o1 = new URL("http://www.upb.ro");
            URL o2 = o1;
            URL o3 = o1;
            go = new GraphObject(o1,o2);
            sout.writeObject(go);
            sout.writeObject(o3);
            sout.flush();
            System.out.println("Object wrote: " + go);
            System.out.println("Object wrote: " + o3);
            System.out.println("(go.o1 == go.o2)&&(go.o1 == o3) is: " + ((go.o1 == go.o2)&&(go.o1 == o3)));
            sout.close();
            fout.close();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

Object wrote: o1 = http://www.upb.ro o2 = http://www.upb.ro
Object wrote: http://www.upb.ro
(go.o1 == go.o2)&&(go.o1 == o3) is: true

Serializarea obiectelor: salvarea unui graf de obiecte

RestoreObject.java

```
import java.io.*;

public class RestoreObject {

    static GraphObject go;

    public static void main (String args[]) {
        try {
            FileInputStream fin = new FileInputStream("test");
            ObjectInputStream sin = new ObjectInputStream(fin);
            go = (GraphObject)sin.readObject();
            System.out.println("Object read: " + go);
            URL o3 = (URL)sin.readObject();
            System.out.println("Object read: " + o3);
            System.out.println("(go.o1 == go.o2)&&(go.o1 == o3) is: " + ((go.o1 == go.o2)&&(go.o1 == o3)));

            sin.close();
            fin.close();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

Object read: o1 = http://www.upb.ro o2 = http://www.upb.ro
Object read: http://www.upb.ro
(go.o1 == go.o2)&&(go.o1 == o3) is: true

Serializarea obiectelor

- O clasa care doreste sa implementeze actiuni speciale la serializare / deserializare trebuie sa implementeze metodele **writeObject()** si **readObject()**
- Daca se doreste sa se salveze si starea obiectului la fel ca atunci cand nu se implementeaza **writeObject()**, se apeleaza din **writeObject()** **defaultWriteObject()** a clasei **ObjectOutputStream()**;
- Daca se implementeaza **writeObject()**, trebuie implementata si **readObject()** (care poate apela **defaultReadObject()** a clasei **ObjectInputStream()**)
- Ordinea operatiilor de refacere trebuie sa fie acceasi cu a operatiilor de salvare
- La deserializare se creeaza un nou obiect, folosind constructorul fara parametri apoi sunt initializate campurile pe baza informatiilor citite
- Daca un obiect al unei clase derivate c2 este serializat, si clasa de baza c1 este la randul ei serializabila, atunci se serializeaza si variabilele clasei de baza;
- Daca in schimb clasa de baza nu este serializabila, clasa derivata care e serializata trebuie sa se ocupe si de salvarea campurilor din clasa de baza.
- Daca o clasa implementeaza interfata **Externalizable**, atunci se apeleaza **writeExternal()**; la deserializare, se apeleaza **readExternal()**;
- Serializarea poate fi utilizata si pentru a realiza copia unui obiect.

Serializarea obiectelor: clonare

Clonare fara duplicarea obiectelor referite

Something.java

```
import java.io.*;

class SomeObject implements Serializable {
    private static final long serialVersionUID = 1L;
    String name;
    SomeObject(String name){
        this.name = name;
    }
    String getName(){
        return name;
    }
    void modifyName(String name) {
        this.name = name;
    }
}
public class Something implements Cloneable {
    SomeObject o;
    Something(String s){
        o = new SomeObject(s);
    }
    void modify(String s){
        o.modifyName(s);
    }
    void print(){
        System.out.println(o.getName());
    }
    public Object clone() {
        try {
            return super.clone();
        }catch(CloneNotSupportedException e){
            e.printStackTrace();
            return null;
        }
    }
}
```

SomethingTest.java

```
/* testare Something: clonare fara duplicarea obiectelor referite */
public class SomethingTest {
    public static void main(String[] argv){
        Something s1 = new Something("aaaa");
        Something s2 = (Something)s1.clone();
        System.out.println("Info about s1: ");
        s1.print();
        s1.modify("cccccccc");
        System.out.println("Info about s1: ");
        s1.print();
        System.out.println("Info about s2: ");
        s2.print();
    }
}
```

Info about s1:
aaaa
Info about s1:
cccccccc
Info about s2:
cccccccc

Daca se realizeaza clonarea fara duplicarea obiectelor referite, la modificarea obiectului clonat se va modifica si clona; pentru a se evita aceasta, trebuie facuta o clonare bazata pe copiere (vezi urmatorul exemplu)

Serializarea obiectelor: clonare

Clonare bazata pe copiere

Cloning.java

```
import java.io.*;

public class Cloning implements Cloneable {
    public Object clone() {
        try {
            ByteArrayOutputStream bout = new ByteArrayOutputStream();
            ObjectOutputStream out = new ObjectOutputStream(bout);
            out.writeObject(this);
            out.close();
            ByteArrayInputStream bin = new ByteArrayInputStream(bout.toByteArray());
            ObjectInputStream in = new ObjectInputStream(bin);
            Object ret = in.readObject();
            in.close();
            return ret;
        }catch(Exception e){
            e.printStackTrace();
            return null;
        }
    }
}
```

SomethingCC.java

```
import java.io.*;

public class SomethingCC extends Cloning implements Serializable {
    private static final long serialVersionUID = 2L;
    SomeObject o;
    SomethingCC(String s){
        o = new SomeObject(s);
    }
    void modify(String s){
        o.modifyName(s);
    }
    void print(){
        System.out.println(o.getName());
    }
}
```

Serializarea obiectelor: clonare

Clonare bazata pe copiere

SomethingCCTest.java

```
/* testare SomethingCC: clonare cu duplicarea obiectelor referite */
public class SomethingCCTest {
    public static void main(String[] argv){
        SomethingCC scc1 = new SomethingCC("aaaa");
        SomethingCC scc2 = (SomethingCC)scc1.clone();
        System.out.println("Info about scc1: ");
        scc1.print();
        scc1.modify("cccccccc");
        System.out.println("Info about scc1: ");
        scc1.print();
        System.out.println("Info about scc2: ");
        scc2.print();
    }
}
```

Clonarea bazata pe copiere se foloseste, in acest exemplu, de operatia de serializare; se creeaza un flux de obiecte care permite serializarea obiectului curent. Apoi, se deserializeaza

Info about scc1:
aaaa
Info about scc1:
cccccccc
Info about scc2:
aaaa