Curs 5
Serializarea obiectelor

Ce este serializarea ?

Definitie

Serializarea este o metoda ce permite transformarea unui obiect într-o secventa de octeti din care sa poata fi refacut ulterior obiectul original. Cu alte cuvinte, serializarea permite salvarea într-o maniera unitara a datelor împreuna cu signatura unui obiect pe un mediu de stocare a informatiei extern programului. Procesul invers de citirea a unui obiect serializat pentru a-i reface starea originala se numeste *deserializare*. Intr-un cadru mai larg, prin serializare se întelege procesul de scriere/citire a obiectelor.

Utilitatea serializarii consta în urmatoarele aspecte:

- Compensarea diferentelor între sisteme de operare, adica putem crea un obiect pe o masina Windows, îl serializam, apoi îl trimitem prin retea catre o masina UNIX unde va fi corect reconstruit. In acest fel comunicarea între sisteme diferite se realizeaza unitar, independent de reprezentarea datelor, ordinea octetilor sau alte detalii specifice sistemelor repective.
- Permite *persistenta* obiectelor, ceea ce înseamna ca durata de viata a unui obiect nu este determinata de executia unui program în care acesta este definit - obiectul poate exista si între apelurile programelor care îl folosesc. Acest lucru se realizeaza prin serializarea obiectului si scrierea lui pe disc înainte de terminarea unui program, apoi, la relansarea programului, obiectul va fi citit de pe disc si starea lui refacuta. Acest tip de persistenta a obiectelor se numeste *persistenta usoara*, întrucât ea trebuie efectuata explicit de catre programator si nu este realizeazata automat de catre sistem.
- *RMI (Remote Method Invocation)* - comunicarea obiectelor prin socket-uri: este o modalitate prin care obiectele de pe o alta masina se comporta ca si când ar exista pe masina pe care ruleaza programul nostru. Atunci când este trimis un mesaj catre un obiect "remote" (de pe alta masina), serializarea este necesara pentru transportul argumentelor prin retea si pentru returnarea valorilor.
- *Java Beans* - sunt componente grafice definite de utilizator si care pot fi folosite la fel ca si componentele grafice standard. Orice componenta Bean are o stare initiala a informatiilor sale, stare care este specificata la definirea sa. Atunci când ea este folosita într-un program aceasta stare trebuie încarcata de undeva, ceea ce înseamna ca aceste componente trebuie serializate si salvate pe disc.

Curs 5
Un aspect important al serializarii este ca nu salveaza doar imaginea unui obiect ci si toate referintele la alte obiecte pe care acesta le contine. Acesta este un proces recusiv de salvare a datelor, întrucât celelalet obiectele referite de obiectul care se serializeaza pot referi la rândul lor alte obiecte, s.a.md. Asadar obiectele care construiesc starea altui obiect formeaza o întreaga retea de obiecte, ceea ce înseamna ca un algoritm de salvare a starii unui obiect nu este facil.
In cazul în care starea unui obiect este formata doar din valori ale unor variabile de tipuri primitive, atunci salvarea datelor înapsulate în acel obiect se poate face si prin salvarea pe rând a datelor, folosind clasa `DataOutputStream`, pentru ca apoi sa fie restaurate prin metode ale clasei `DataInputStream`, dar, asa cum am vazut, o asemenea abordare nu este în general suficienta, deoarece pot aparea probleme cum ar fi : datele obiectului pot fi instante ale altor obiecte, unele câmpuri fac referinta la acelasi obiect, etc.
Serializarea obiectelor se realizeaza prin intermediul fluxurilor definite de clasele `ObjectOutputStream` (pentru salvare) si `ObjectInputStream` (pentru restaurare).

Serializarea obiectelor

Serializarea obiectelor se realizeaza prin intermediul fluxurilor definite de clasele **`ObjectOutputStream`** (pentru salvare) si **`ObjectInputStream`** (pentru restaurare). Acestea sunt fluxuri de procesare ceea ce înseamna ca ele vor fi folosite împreuna cu alte fluxuri pentru citirea/scrierea efectiva a datelor pe mediul extern pe care va fi salvat sau de pe care va fi restaurat un obiect serializat.
Mecanismul implicit de serializare a unui obiect va salva numele clasei obiectului, signatura clasei obicetului, valorile tuturor câmpurile serializabile ale obiectului (vezi "Controlul serializarii").
Referintele la alte obiecte serializabile din cadrul obiectului curent vor duce automat la serializarea acestora iar referintele multiple catre un acelasi obiect sunt codificate utilizând un algoritm care sa poata reface "reteaua de obiecte" la aceeasi stare ca atunci când obiectul original a fost salvat.
Clasele `ObjectInputStream` si `ObjectOutputStream` implementeaza indirect interfetele `DataInput`, respectiv `DataOutput`, interfete ce declara metode de tipul `readXXX`, respectiv `writeXXX` pentru scrierea/citirea datelor primitive. Pe lânga aceste metode vor exista si metode pentru scrierea/citirea obiectelor.
Metodele pentru serializarea obiectelor sunt:

```
private void readObject(ObjectInputStream stream)     //salvare obiect
        throws IOException,ClassNotFoundException;

private void writeObject(ObjectOutputStream stream)   //refacere obiect
        throws IOException;
```

Aceste metode contin apeluri catre metodele implicite de seializare a obiectelor:

```
final void defaultWriteObject()
        throws IOException
final void defaultReadObject()
        throws IOException,ClassNotFoundException,NotActiveException
```

Clasele care necesita o serializare speciala trebuie sa supradefineasca metodele `writeObject` si `readObject` (obligatoriu pe amândoua!) pentru a implementa metode specifice de serializare. (vezi "Personalizarea serializarii obiectelor").

Clasa ObjectOutputStream

Scrierea obiectelor pe un flux este un proces extrem de simplu. Exemplul de mai jos, afla timpul curent în milisecunde construind un obiect de tip `Date` si îl salveaza într-un fisier `theTime`:

```
FileOutputStream out = new FileOutputStream("theTime");
ObjectOutputStream s = new ObjectOutputStream(out);
s.writeObject("Today");
s.writeObject(new Date());
s.flush();
s.close();
```

Asadar metoda pentru scrierea unui obiect este **`writeObject`**, responsabila cu serializarea completa a obiectului. Deoarece implementeaza interfata `DataOutput`, pe lânga metoda de scriere a obiectelor, clasa pune la dispozitie si metode de tipul `writeXXX` pentru scrierea tipurilor de date primitive, astfel încât apeluri ca cele de mai jos sunt permise :

```
FileOutputStream out = new FileOutputStream("t.tmp");
ObjectOutputStream s = new ObjectOutputStream(out);
s.writeInt(12345);
s.writeDouble(12.345);
s.writeUTF("Sir de caractere");
s.flush();
s.close();
```

Metoda `writeObject` arunca exceptii de tipul `NotSerializableException` daca obiectul primit ca argument nu este serializabil. Vom vedea în continuare ca un obiect este serializabil daca este instanta a unei clase ce implementeaza interfata `Serializable`.

Clasa ObjectInputStream

Odata ce au fost scrise obiecte si tipuri primitive de date pe un flux, citirea acestora si reconstruirea obiectelor salvate se va face printr-un flux de intrare de tip `ObjectInputStream`. Acesta este de asemenea un flux de procesare si va trebui asociat cu un flux pentru citirea efectiva a datelor, de exemplu `FileInputStream` (pentru date salvate într-un fisier).

```
FileInputStream in = new FileInputStream("theTime");
ObjectInputStream s = new ObjectInputStream(in);
String today = (String)s.readObject();
Date date = (Date)s.readObject();
```

Asadar, metoda pentru citirea unui obiect serializat si refacerea starii lui este **`readObject`**. Clasa `ObjectInputStream` implementeaza interfata `DataInput`, deci, pe lânga metoda de citire a obiectelor clasa pune la dispozitie si metode de tipul `readXXX` pentru citirea tipurilor de date primitive.

```
FileInputStream in = new FileInputStream("t.tmp");
ObjectInputStream s = new ObjectInputStream(in);
int n = s.readInt();
double d = s.readDouble(12.345);
String sir = s.readUTF();
```

Atentie

Ca si la celelate fluxuri de date (care implemeteaza interfata `DataInput`) citirea dintr-un flux de obiecte trebuie sa se faca exact în ordinea în carea acestea au fost scrise.

Trebuie observat ca metoda `readObject` returneaza un obiect de tipul `Object` si nu de tipul corespunzator obiectului citit, conversia la acest tip trebuind sa se faca explicit:

```
Date date = s.readObject();           // ilegal
Date date = (Date)s.readObject();     // corect
```

Obiecte serializabile

Un obiect este serializabil daca si numai daca clasa din care face parte implementeaza interfata **`Serializable`**. Asadar, daca dorim ca instantele unei clase sa poata fi serializate, clasa respectiva trebuie sa implementeze interfata `Serializable`. Aceasts interfata este mai deosebita, în sensul ca nu contine nici o declaratie de metoda, singurul ei scop fiind de a identifica clasele ale caror obiecte sunt serializabile.

Implementarea interfetei Serializable

Definitia completa a interfetei `Serializable` este:

```
package java.io;
```

```
public interface Serializable {
        // nimic!
}
```

Crearea claselor ale caror instante sunt serializabile este extrem de facila: la clasa respectiva trebuie sa adaugam în declaratia ei ca implementeze interfata `Serializable` si nimic mai mult:

```
public class ClasaSerializabila implements Serializable {
        //putem sa nu scriem nici o metoda deoarece
        //interfata nu declara nici o metoda!
}
```

Asadar, clasa poate sa nu contina nici o metoda, ea va contine totusi metode altfel nu ar avea nici un rost, dar metodele vor fi specifice scopului pentru care ea a fost creata si nu vor avea legatura cu serializarea.

Asa cum am vazut, serializarea implicita a obiectelor oricarei clase este definita în metoda `defaultWriteObject` a clasei `ObjectOutputStream` care va salva toate datele necesare reconstruirii obiectului : numele clasei, signatura, valorile variabilelor membre si obiectele referite de acestea. In majoritatea cazurilor aceasta metoda de serializare este suficienta, însa o clasa poate avea nevoie de mai mult control asupra serializarii.

Personalizarea serializarii obiectelor

Personalizarea serializarii se realizeaza prin supradefinirea (într-o clasa serializabila!) a metodelor `writeObject` si `readObject`, modificând astfel actiunea lor implicita.
Metoda `writeObject` controleaza ce date sunt salvate si este uzual folosita pentru a adauga informatii suplimentare la cele scrise implicit de metoda `defaultWriteObject`.
Metoda `readObject` controleaza modul în care sunt restaurate obiectele, citind informatiile salvate si, eventual, modifcând starea obiectelor citite astfel încât ele sa corespunda anumitor cerinte.
Aceste metode trebuie obligatoriu sa aiba urmatorul format:

```
private void writeObject(ObjectOutputStream stream)
        throws IOException
private void readObject(ObjectInputStream stream)
        throws IOException, ClassNotFoundException
```

De asemenea, uzual, primul lucru pe care trebuie sa îl faca aceste metode este apelul la metodele standard de serializare a obiectelor `defaultWriteObject`, respectiv `defaultReadObject` si abia apoi sa execute diverse operatiuni suplimentare. Forma lor generala este:

```
private void writeObject(ObjectOutputStream s)
                throws IOException {
        s.defaultWriteObject();
        // personalizarea serializarii
}

private void readObject(ObjectInputStream s)
                throws IOException,ClassNotFoundException {
        s.defaultReadObject();
        // personalizarea deserializarii
        . . .
        // actualizarea starii obiectului (daca e necesar)
}
```

Metodele `writeObject` si `readObject` sunt responsabile cu serializarea clasei în care sunt definite, serializarea superclasei sale fiind facuta automat (si implicit). Daca însa o clasa trebuie sa-si coordoneze serializarea proprie cu serializarea superclasei sale, atunci trebuie sa implementeze interfata `Externalizable`.

Implementarea interfetei Externalizable

Pentru un control complet, explicit, al procesului de serializare, o clasa trebuie sa implementeze interfata `Externalizable`. Pentru instante ale acestor clase doar numele clasei este salvat automat pe un flux de

obiecte, clasa fiind responsabila cu scrierea si citirea membrilor sai si trebuie sa se coordoneze cu superclasele ei.
Definitia interfetei `Externalizable` este:

```
package java.io;
public interface Externalizable extends Serializable {
        public void writeExternal(ObjectOutput out)
                throws IOException;
        public void readExternal(ObjectInput in)
                throws IOException, ClassNotFoundException;
}
```

Asadar, aceste clase trebuie sa implementeze obligatoriu metodele `writeExternal` si `readExternal` în care se va face serializarea completa a obiectelor si coordonarea cu superclasa ei.

Controlul serializarii

Exista cazuri când dorim ca unele variabile membre sau sub-obiecte ale unui obiect sa nu fie salvate automat în procesul de serializare. Acestea sunt cazuri comune atunci când respectivele câmpuri reprezinta informatii confidentiale, cum ar fi parole, sau variabile auxiliare pe care nu are rost sa le salvam. Chiar declarate ca private în cadrul clasei aceste câmpuri participa la serializare. O modalitate de a controla serializare este implementarea interfetei `Externalizable`, asa cum am vazut anterior. Aceasta metoda este însa incomoda atunci când clasele sunt greu de serializat iar multimea câmpurilor care nu trebuie salvate este redusa.
Pentru ca un câmp sa nu fie salvat în procesul de serializare atunci el trebuie declarat cu modificatorul **`transient`** si trebuie sa fie ne-static. De exemplu, declararea unei parole ar trebui facuta astfel:

```
transient private String parola;      //ignorat la serializare
```

<
Atentie

Modificatorul `static` anuleaza efectul modificatorului `transient`;

```
static transient private String parola;       //participa la serializare
```

De asemenea, nu participa la serializare sub-obiectele neserializabile ale unui obiect, adica cele ale caror clase nu au fost declarate ca implementând interfata `Serializable` (sau `Externalizable`).

```
Exemplu: (câmpurile marcate 'DA' participa la serializare, cele marcate 'NU',
nu participa)
  class A { ... }
  class B implements Serializable { ... }
  public class Test implements Serializable {
          private int x;                  // DA
          transient public int y;                 // NU
          static int var1;                // DA
          transient static var2;          // DA
          A a;                            // NU
          B b1;                           // DA
          transient B b2;                 // NU
  }
```

Atunci când o clasa serializabila deriva dintr-o alta clasa, salvarea câmpurilor clasei parinte se va face doar daca si aceasta este serializabila. In caz contrar, subclasa trebuie sa salveze explicit si câmpurile mostenite.

```
Ex1: class Parinte implements Serializable {
          int x;
  }
  class Fiu extends Parinte implements Serializable {
          int y;
  }//La serializarea obiectelor de tip Fiu se salveaza atât x cât si y.

Ex2: class Parinte {
```

```
        int x;
  }
  class Fiu extends Parinte implements Serializable {
        int y;
  }//Serializarea nu decurge normal.
```

Exemplu de folosire a serializarii

```
import java.io.*;
public class TestSerializare {
  public static void main(String args[]) throws IOException {
        MyObject obj = new MyObject(10, 20, 30);

        //salvam obiectul in fisierul "fisier.tmp"
        FileOutputStream fout = new FileOutputStream("fisier.tmp");
        ObjectOutputStream sout = new ObjectOutputStream(fout);

        sout.writeObject(obj);
        sout.flush();

        sout.close();
        fout.close();
        System.out.println("A fost salvat obiectul " + obj);

        System.out.println("Restauram...");
        FileInputStream fin = new FileInputStream("fisier.tmp");
        ObjectInputStream sin = new ObjectInputStream(fin);
        try {
                obj = (MyObject) sin.readObject();
        } catch (ClassNotFoundException e) {}

        sin.close();
        fin.close();
        System.out.println("A fost restaurat obiectul " + obj);
  }
}

class MyObject implements Serializable {
  int x;                        //este salvat
  transient private int y;      //nu este salvat
  transient static int z;       //nu este salvat

  public MyObject(int x, int y, int z) {
        this.x = x;
        this.y = y;
        this.z = z;
  }
  public String toString() {
        return new String("x=" + x + ", y=" + y + ", z=" + z);
  }
}
```

Rezultatul acestui program va fi :

```
  A fost salvat obiectul x=10, y=20, z=30
  Restauram...
  A fost restaurat obiectul x=10, y=0, z=30
```

Folosirea serializarii pentru copierea obiectelor

Curs 5

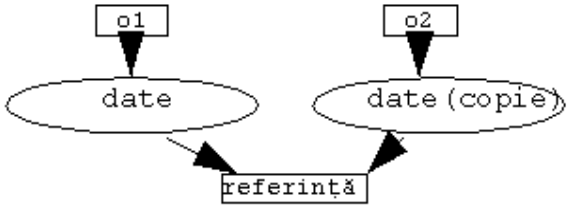
Se stie ca nu putem copia un obiect prin instructiunea de atribuire. O secventa de forma:

```
MyObject o1 = new MyObject(10, 20, 30);
MyObject o2 = o1;
```

nu face decât sa declare obiectul o2 ca fiind o referinta la obiectul o1 si prin urmarea orice schimbare într-unul din cele doua obiecte se va reflecta si în celalalt.

O posibilitate de a face o copie unui obiect este folosirea metodei `clone()` a clasei Object.

```
MyObject o1 = new MyObject(10, 20, 30);
MyObject o2 = (MyObject) o1.clone();
```



Conversia la clasa `MyObject` este necesara deoarece metoda `clone()` returneaza un obiect de tip `Object`. Deficienta acestei metode este ca nu functioneaza corect decât atunci când clasa clonata nu are câmpuri referinta ca alte obiecte, obiectele referite nemaifiind copiate la rândul lor.
O metoda clone() care sa realizeze o copie efectiva a unui obiect, împreuna cu copierea tuturor obiectelor referite de câmpurile acelui obiect poate fi implementata prin mecanismul serializarii astfel:

```
public Object clone() {
  try {
          ByteArrayOutputStream bout = new ByteArrayOutputStream();
          ObjectOutputStream out = new ObjectOutputStream(bout);
          out.writeObject(this);
          out.close();

          ByteArrayInputStream bin = new ByteArrayInputStream();
          ObjectInputStream in = new ObjectInputStream(bin);

          Object ret = in.readObject();
          in.close();
          return ret;

  } catch (Exception e) {
          System.out.println(e);
          return null;
  }
}
```