Serializarea obiectelor

- **Ce este serializarea ?**
- **Serializarea tipurilor primitive**
- **Serializarea obiectelor**
- **ObjectInputStream**
  ObjectOutputStream
- **Interfaţa Serializable**
- **Controlul serializării**
- **Personalizarea serializării**
- **Clonarea obiectelor**

Ce este serializarea ?

Serializare= transformarea unui obiect într-o secvenţă de octeţi, din care să poată fi refăcut ulterior obiectul original.

Procesul invers, de citire a unui obiect serializat pentru a-i reface starea originală, se numeşte **deserializare**.

Referinţele care construiesc starea unui obiect formează o întreagă **reţea** de obiecte.

- `DataOutputStream`, `DataInputStream`
- `ObjectOutputStream`, `ObjectInputStream`

Utilitatea serializării

- Mecanism simplu de utilizat pentru **salvarea şi restaurarea** datelor.
- **Persistenţa** obiectelor
- **Compensarea diferenţelor** între sisteme de operare
- **Transmiterea datelor în reţea**
- **RMI** (Remote Method Invocation)
- **Java Beans** - asigurarea persistenţei componentelor.

Serializarea tipurilor primitive

DataOutputStream, DataInputStream

```
//Scriere
FileOutputStream fos =
  new FileOutputStream("test.dat");
DataOutputStream out = new DataOutputStream(fos);
out.writeInt(12345);
out.writeDouble(12.345);
out.writeBoolean(true);
out.writeUTF("Sir de caractere");
out.flush();
fos.close();
...
//Citire
FileInputStream fis =
  new FileInputStream("test.dat");
DataInputStream in = new DataInputStream(fis);
int i = in.readInt();
double d = in.readDouble();
boolean b = in.readBoolean();
String s = in.readUTF();
fis.close();
```

Serializarea obiectelor

- `ObjectOutputStream`
- `ObjectInputStream`

Mecanismul implicit va salva:

- Numele clasei obiectului.
- Signatura clasei.
- Valorile tuturor câmpurile serializabile. Acesta este un proces recursiv, de serializare a obiectelor referite ("reţeaua de obiecte");

Metode:

- **writeObject**, pentru scriere şi
- **readObject**, pentru restaurare.

Clasa ObjectOutputStream

```
ObjectOutputStream out =
  new ObjectOutputStream(fluxPrimitiv);
out.writeObject(referintaObiect);
out.flush();
fluxPrimitiv.close();

FileOutputStream fos =
  new FileOutputStream("test.ser");
ObjectOutputStream out = new ObjectOutputStream(fos);
out.writeObject("Ora curenta:");
out.writeObject(new Date());
out.writeInt(12345);
out.writeDouble(12.345);
out.writeBoolean(true);
out.writeUTF("Sir de caractere");
out.flush();
fos.close();
```

Excepţii: `IOException`, `NotSerializableException`, `InvalidClassException`.

Clasa ObjectInputStream

```
ObjectInputStream in =
  new ObjectInputStream(fluxPrimitiv);
Object obj = in.readObject();
//sau
TipReferinta ref = (TipReferinta)in.readObject();
fluxPrimitiv.close();

FileInputStream fis = new FileInputStream("test.ser");
ObjectInputStream in = new ObjectInputStream(fis);
String mesaj = (String)in.readObject();
Date data = (Date)in.readObject();

int i = in.readInt();
double d = in.readDouble();
boolean b = in.readBoolean();
String s = in.readUTF();

fis.close();
```

Interfaţa Serializable

Un obiect este serializabil $\Leftrightarrow$ clasa din care face parte implementează interfaţa **Serializable**.

Interfaţa este **declarativă**, definiţia ei fiind:

```
package java.io;
public interface Serializable {
  // Nimic !
}


public class ClasaSerializabila
  implements Serializable {

  // Corpul clasei
}
```

Controlul serializării

Modificatorul **transient**

```
transient private double temp;
// Ignorata la serializare
```

Modificatorul **static** anulează efectul modificatorului `transient`

```
static transient int N;
// Participa la serializare
```

Listing 1: Modificatorii static şi transient

```
import java.io.*;

public class Test1 implements Serializable {

  int x=1;                    //DA
  transient int y=2;          //NU
  transient static int z=3;   //DA
  static int t=4;             //DA

  public String toString() {
    return x + ", " + y + ", " + z + ", " + t;
  }
}
```

Membrii neserializabili

In procesul serializării, dacă este întâlnit un obiect care nu implementează interfaţa `Serializable` atunci va fi generată o excepţie de tipul `NotSerializableException` ce va identifica respectiva clasă neserializabilă.

Listing 2: Membrii neserializabili

```
import java.io.*;

class A {
  int x=1;
}

class B implements Serializable {
  int y=2;
}

public class Test2 implements Serializable{
  A a = new A();  //Exceptie
  B b = new B();  //DA

  public String toString() {
    return a.x + ", " + b.y;
  }
}
```

Serializarea câmpurilor moştenite

Listing 3: Serializarea câmpurilor moştenite

```
import java.io.*;
class C {
  int x=0;
  // Obligatoriu constructor fara argumente
}

class D extends C implements Serializable {
  int y=0;
}

public class Test3 extends D {
  public Test3() {
    x = 1; //NU
    y = 2; //DA
  }
  public String toString() {
    return x + ", " + y;
  }
}
```

Listing 4: Testarea serializării

```
import java.io.*;

public class Exemplu {

  public static void test(Object obj) throws IOException {
    // Salvam
    FileOutputStream fos = new FileOutputStream("fisier.ser")
        ;
    ObjectOutputStream out = new ObjectOutputStream(fos);

    out.writeObject(obj);
    out.flush();

    fos.close();
    System.out.println("A fost salvat obiectul: " + obj);

    // Restauram
    FileInputStream fis = new FileInputStream("fisier.ser");
    ObjectInputStream in = new ObjectInputStream(fis);
    try {
      obj = in.readObject();
    } catch(ClassNotFoundException e) {
      e.printStackTrace();
    }

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

  public static void main(String args[]) throws IOException {
    test(new Test1());
    try {
      test(new Test2());
    } catch(NotSerializableException e) {
        System.out.println("Obiect neserializabil: " + e);
    }

    test(new Test3());
  }
}
```

Personalizarea serializării

Dezavantajele mecanismului implicit:

- lent
- reprezentarea voluminoasă

Personalizarea serializarii:

```
private void writeObject(ObjectOutputStream stream)
    throws IOException {
  // Procesarea campurilor clasei (criptare, etc.)
  // Scrierea obiectului curent
  stream.defaultWriteObject();
  // Adaugarea altor informatii suplimentare
}
private void readObject(ObjectInputStream stream)
    throws IOException,ClassNotFoundException {
  // Restaurarea obiectului curent
  stream.defaultReadObject();
  // Actualizarea starii obiectului (decriptare, etc.)
  // si extragerea informatiilor suplimentare
}
```

Controlul versiunilor claselor

Listing 5: Prima variantă a clasei Angajat

```
import java.io.*;

class Angajat implements Serializable {

  public String nume;
  public int salariu;
  private String parola;

  public Angajat(String nume, int salariu, String parola) {
    this.nume = nume;
    this.salariu = salariu;
    this.parola = parola;
  }

  public String toString() {
    return nume + " (" + salariu + ")";
  }

}
```

Listing 6: Aplicaţia de gestionare a angajaţilor

```
import java.io.*;
import java.util.*;

public class GestiuneAngajati {

  //Lista angajatilor
  ArrayList ang = new ArrayList();

  public void citire() throws IOException {
    FileInputStream fis = null;
    try {
      fis = new FileInputStream("angajati.ser");
      ObjectInputStream in = new ObjectInputStream(fis);
      ang = (ArrayList) in.readObject();
    } catch(FileNotFoundException e) {
      System.out.println("Fisierul nou...");
    } catch(Exception e) {
      System.out.println("Eroare la citirea datelor...");
      e.printStackTrace();
    }finally {
      if (fis != null)
        fis.close();
    }
    System.out.println("Lista angajatilor:\n" + ang);
  }

  public void salvare() throws IOException {
    FileOutputStream fos =
      new FileOutputStream("angajati.ser");
    ObjectOutputStream out = new ObjectOutputStream(fos);
    out.writeObject(ang);
  }

  public void adaugare() throws IOException {
     BufferedReader stdin = new BufferedReader(
       new InputStreamReader(System.in));

     while (true) {
       System.out.print("\nNume:");
       String nume = stdin.readLine();

       System.out.print("Salariu:");
       int salariu = Integer.parseInt(stdin.readLine());
```

```
        System.out.print("Parola:");
        String parola = stdin.readLine();

        ang.add(new Angajat(nume, salariu, parola));

        System.out.print("Mai adaugati ? (D/N)");
        String raspuns = stdin.readLine().toUpperCase();
        if (raspuns.startsWith("N"))
          break;
      }
    }

    public static void main(String args[]) throws IOException {
      GestiuneAngajati app = new GestiuneAngajati();

      //Incarcam angajatii din fisier
      app.citire();

      //Adaugam noi angajati
      app.adaugare();

      //Salvam angajatii inapoi fisier
      app.salvare();
    }
}
```

Adăugăm un nou câmp: *adresa*.

Problema: `InvalidClassException`

serialVersionUID

- număr pe 64 de biţi
- generat în procesul de serializare/deserializare
- salvat împreună cu starea obiectului
- codifică signatura unei clase
- "sensibil" la orice modificare a clasei

Setarea explicită:

```
static final long
  serialVersionUID = /* numar_serial_clasa */;
```

Utilitarul **serialVer** permite generarea numărului `serialVersionUID`

Listing 7: Variantă compatibilă a clasei Angajat

```
import java.io.*;

class Angajat implements Serializable {

  static final long serialVersionUID = 5653493248680665297L;

  public String nume, adresa;
  public int salariu;
  private String parola;

  public Angajat(String nume, int salariu, String parola) {
    this.nume = nume;
    this.adresa = "Iasi";
    this.salariu = salariu;
    this.parola = parola;
  }

  public String toString() {
    return nume + " (" + salariu + ")";
  }

}
```

Listing 8: Varianta securizată a clasei Angajat

```
import java.io.*;

class Angajat implements Serializable {

  static final long serialVersionUID = 5653493248680665297L;

  public String nume, adresa;
  public int salariu;
  private String parola;

  public Angajat(String nume, int salariu, String parola) {
    this.nume = nume;
    this.adresa = "Iasi";
    this.salariu = salariu;
    this.parola = parola;
  }

  public String toString() {
    return nume + " (" + salariu + ")";
  }

  static String criptare(String input, int offset) {
    StringBuffer sb = new StringBuffer();
    for (int n=0; n<input.length(); n++)
      sb.append((char)(offset+input.charAt(n)));
    return sb.toString();
  }

  private void writeObject(ObjectOutputStream stream)
      throws IOException {
    parola = criptare(parola, 3);
    stream.defaultWriteObject();
    parola = criptare(parola, -3);
  }

  private void readObject(ObjectInputStream stream)
      throws IOException, ClassNotFoundException {
    stream.defaultReadObject();
    parola = criptare(parola, -3);
  }
}
```

Interfaţa Externalizable

Control **complet, explicit** al procesului de serializare.

Definiţia interfeţei `Externalizable`:

```
package java.io;
public interface Externalizable
       extends Serializable {

  public void writeExternal(ObjectOutput out)
    throws IOException;
  public void readExternal(ObjectInput in)
    throws IOException, ClassNotFoundException;
}
```

Scop: creşterea vitezei procesului de serializare.

Listing 9: Serializare implicită

```
import java.io.*;

class Persoana implements Serializable {
  int cod;
  String nume;

  public Persoana(String nume, int cod) {
    this.nume = nume;
    this.cod  = cod;
  }
}
```

Listing 10: Serializare proprie

```
import java.io.*;

class Persoana implements Externalizable {
  int cod;
  String nume;

  public Persoana(String nume, int cod) {
    this.nume = nume;
    this.cod  = cod;
  }

  public void writeExternal(ObjectOutput s)
      throws IOException {
    s.writeUTF(nume);
    s.writeInt(cod);
  }

  public void readExternal(ObjectInput s)
      throws ClassNotFoundException, IOException {
    nume = s.readUTF();
    cod = s.readInt();
  }

}
```

Clonarea obiectelor

```
  TipReferinta o1 = new TipReferinta();
  TipReferinta o2 = (TipReferinta) o1.clone();

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

    byte[] buffer = baos.toByteArray();
    ByteArrayInputStream bais =
      new ByteArrayInputStream(buffer);
    ObjectInputStream in = new ObjectInputStream(bais);
    Object ret = in.readObject();
    in.close();
    return ret;
  } catch (Exception e) { return null; }
}
```