Serializare

De la POO

Salt la: Navigare, căutare

Introducere

In laboratorul de ***I/O*** am studiat mecanisme de transmisie a valorilor avand tipuri predefinite (`int`, `float`, `String` etc.), utilizand ***stream***-uri. Astfel, le puteam:

- stoca in **fisiere**, din care sa le citim ulterior
- propaga pe **retea**/***pipe***-uri
- etc.

Intrebarea fireasca este: *putem extinde acest mecanism pentru a opera si cu **instantele** claselor definite de utilizator*? Raspunsul este *da*.

Java pune la dispozitie conceptul de **serializare**. Acesta poate fi definit astfel: reprezentarea unei entitati sub forma unei **secvente** de octeti, cu **ascunderea** diverselor detalii, precum:

- **reprezentarile** particulare ale datelor pe diferitele platforme (*byte ordering* etc.). Acest aspect se dovedeste util mai ales in cazul transmisiei pe retea, cand cele doua capete ale canalului de comunicatie ruleaza pe sisteme de operare sau arhitecturi diferite.
- **structurile** arbitrar de complexe ale obiectelor: acestea pot contine variabile membru de tip referinta, vectori de orice adancime etc. Cu exceptia cazului in care programatorul doreste sa defineasca **explicit** o schema particulara de serializare, sistemul poate asigura serializarea **automata** a intregii componenţe a obiectelor, explorand recursiv structura acestuia. Graful de referinte obtinut poarta numele de ***web of objects***.

Serializarea este utilizata pentru implementarea conceptului de **persistenta**, vazuta drept capacitate a unui obiect de a supravietui **dupa** incheierea executiei programului. Acest deziderat se obtine prin **depozitarea** informatiilor acestuia si **recuperarea** lor la urmatoarea executie. Desigur, acelasi efect se poate reproduce prin stocarea explicita a componentelor unui obiect in fisiere sau baze de date, dar serializarea reprezinta un mecanism **uniform**, aplicabil asupra **oricarui** tip de obiecte.

Interfata `Serializable`

In exemplul de mai jos, clasa `Group` tine un vector de referinte la clasa `Student`:

```
public class Student implements Serializable {

        private String name;

        public Student(String name) {
                System.out.println(name);
                this.name = name;
        }

        @Override
        public String toString() {
                return name;
        }
}

public class Group implements Serializable {

        private Student[] students;

        /**
         * Construieste o grupa pe baza unor studenti.
         *
         * @param students
         *                      parametri cu numar variabil. Acestia pot fi pasati in 2 moduri:
         *                      * separandu-i prin virgula:     new Group(s1, s2, s3);
         *                      * sub forma unui vector:        new Group(new Student[] {s1, s2, s3});
         *                      In cadrul constructorului, parametrul este vazut ca un vector.
         */
        public Group(Student... students) {
                /* Realizam o copie a vectorului, pentru ca modificarile externe
                 * asupra acestuia sa nu se reflecte in interiorul clasei.
                 */
                this.students = Arrays.copyOf(students, students.length);
        }
```

```
        @Override
        public String toString() {
                // Necesar pentru a accesa reprezentarile elementelor vectorului.
                return Arrays.deepToString(students);
        }

}

public class Test {

        public static void main(String[] args) {
                Group g = new Group(new Student("Gigel"), new Student("Costel"));
                System.out.println("Initial: " + g);

                // Serializare.
                ObjectOutputStream os = null;
                try {
                        os = new ObjectOutputStream(new FileOutputStream("out.bin"));
                        os.writeObject(g);
                }
                catch (IOException e) {
                        e.printStackTrace();
                }
                finally {
                        if (os != null)
                                try {
                                        os.close();
                                }
                                catch (IOException e) {}
                }

                // Deserializare.
                ObjectInputStream is = null;
                try {
                        is = new ObjectInputStream(new FileInputStream("out.bin"));
                        g = (Group)is.readObject();
                        System.out.println("Deserialized: " + g);
                }
                catch (IOException e) {
                        e.printStackTrace();
                }
                catch (ClassNotFoundException e) {
                        e.printStackTrace();
                }
                finally {
                        if (is != null)
                                try {
                                        is.close();
                                }
                                catch (IOException e) {}
                }
        }

}
```

Observatii:

- **declaram** faptul ca instantele unei clase sunt serializabile, implementand interfata `Serializable`. Aceasta interfata are unicul rol de semnala sistemului intentia noastra, necontinand efectiv **nicio** metoda. In astfel de cazuri, utilizam numele de *tagging interface*
- utilizam clasele **decorator** `ObjectInputStream` si `ObjectOutputStream`, cu metodele lor `readObject`, respectiv `writeObject`.
  - daca, in procesul de serializare, obiectul insusi sau un membru al sau (la orice adancime) apartine unei clase ce **nu** a fost declarata serializabila, se va arunca `NotSerializableException`, ce va contine informatie despre numele clasei cu pricina
  - `readObject` intoarce tipul `Object`, care trebuie, ulterior, transformat prin ***downcast*** la tipul concret. In cazul in care clasa obiectului **nu** este disponibila in momentul deserializarii, se arunca `ClassNotFoundException` (care este prinsa intr-un bloc `catch`)
- serializarea acopera **in intregime** (tranzitiv) informatia referita de obiect. In cazul nostru, in cursul serializarii instantei `Group`, este serializat vectorul membru, alaturi de toate elementele sale (instante `Student`). Mai departe, sunt serializate campurile membru de tipul `String` din acestea s.a.m.d.
- desi constructorul clasei `Student` contine o instructiune de afisare, deserializarea **nu** conduce la aparitia textului la consola. Informatia este recuperata **direct** din *stream*, **fara** antrenarea constructorilor in acest demers!

Exercitii

1. Demonstrati ca serializarea/deserializarea functioneaza corect si in cazul referintelor **ciclice**:
   - Definiti **clasa** `CircularList`, pentru implementarea unei **liste simplu inlantuite, circulara**

- **Constructorul** clasei va primi drept parametru un **numar** nenegativ, `n`, si va contrui o lista cu numerele de la `n` la `0`, in ordine descrescatoare. Ultimul element al listei va retine o referinta catre primul.
  - De exemplu, instructiunea `new CircularList(4)` va produce lista `[4, 3, 2, 1, 0]`, unde `0` este legat la `4`
- Metoda **`toString`** va intoarce o reprezentare a listei, similara celei de mai sus.
- **Serializati**, apoi **deserializiazati** o instanta a acestei clase.

Adus de la "http://cursuri.cs.pub.ro/~poo/wiki/index.php/Serializare"