C12

Operatii de Intrare/iEsire cu fisiere

- Fisiere
- Fluxuri
  - pe octet
  - pe caracter
- Serializare
- Tokenizer (analiza lexicala)

Obiective

După parcurgerea acestui curs ar trebuie sa puteţi:

- înțelege principiile de lucru fisiere si fluxuri in Java;
- manipula date date prin fluxuri pe caracter si octer;
- serializa obiecte prin fluxuri;
- realiza analiza lexicala a datelor stocate in fisiere.

Fisiere si Fluxuri

Variabilele sunt utilizate în aplicaţii pentru stocarea de valori temporare respectiv, valorile stocate în acestea se pierd atunci când aplicaţia se termină. Fişierele sunt folosite pentru a păstra datele după terminarea aplicaţiei. Păstrarea valorile stocate în variabile prin salvarea lor în fişiere se numeşte persistenţa datelor. Stocarea datelor în fişiere se poate face în două moduri distincte: text şi binar. În modul text datele sunt stocate ca o secvenţă de caractere ASCII ce pot fi citite cu ajutorul unui editor de text. De exemplu, dacă considerăm întregul 2807, în modul text, acesta v-a fi stocat ca o secvenţă de patru caractere, '2', '8', '0', '7'. În modul binar datele sunt reprezentatea prin octeţi (opt biţi). Octetul poate stoca 256 de valori distincte în domeniul [0, 255]. Valorile întregi sunt stocate pe patru octeţi, deci reprezentarea în modul binar a lui 2807 este 000 000 010 247 (2807 = 10*256 + 247). Pentru datele stocate în modul text se utilizeaza clasele `Reader` şi `Writer`. Modul text se foloseşte pentru prelucrarea şirurilor de carctere. Dacă datele sunt stocate în modul binar prelucrările se fac folosind clasele `InputStream` şi `OutputStream`. Modul binar se utilizează la prelucrarea sunetelor, imaginilor sau a obiectelor.

Aplicaţiile Java pot citi date de la mai multe tipuri de dispozitive externe, generic acestea se numesc surse de date. La fel, scrierea se poate face către mai multe destinaţii, generic acestea fiind denumite ieşiri de date. Deoarece sursele împreună cu ieşirile de date acoperă un domeniu foarte larg de dispozitive (conexiune de reţea, buffere, disc, imprimată, ...), care deseori, fizic, sunt foarte diferite, s-a realiza abstractizarea tuturora sub conceptul de flux (stream). Fluxul este o entitate logică care produce sau consumă date. Fluxul este conectat întotdeauna la un dispozitiv fizic de intrare/ieşire de către Java. Toate fluxurile au acelaşi comportament deşi dispozitivele fizice prin care operează sunt foarte diferite.

Pachetul Java `java.io` are clasele grupate pe baza funcţionalităţii, adică toate clasele care se ocupă de operaţii de intrare sunt moştenite de la aceeaşi clasa la fel şi pentru ieşire.

Procedura de lucru cu orice flux este descrisă în continuare:

```
deschide fluxul
while (există_date)
    citeşte / scrie date
închide fluxul
```

Fisiere externe

Denumirea de fişiere extene este dată acelor fişiere ce sunt parte a sistemului de fişiere din sistemul de operare. În Java aceste fişiere nu sunt fluxuri şi se manipulează cu clasa `File`. Clasa nu specifica modul de stocare şi citire a datelor din fişier ci descrie doar proprietăţile fişierului respectiv în contextul sistemului de fişiere. Un obiect `File` stochează informaţiile pentru accesul, data, timp, cale şi ierarhii de subdirectori legate de fişier. Fişierele reprezintă sursa primară şi destinaţia finală a multor programe Java prin care se asigură persitenta datelor în aplicaţie şi partajarea loc cu alte aplicaţii. Un director, în Java, este doar un fişier ce are o proprietate adiţională de listă a numelor de fişiere ce se poate examina cu metoda `list()`.

Metodele uzuale pentru manipularea de fişiere se prezintă în aplicaţia ce urmează:

```
import java.io.File;
import java.io.IOException;

class TestFisiere {
  static void afisare(String s) {
    System.out.println(s);
  }

  public static void main(String args[]) throws IOException {
    File f1 = new File("TestFisiere.dat");
    afisare(f1.createNewFile()?"Fisier creat cu succes":"Nu se poate crea");
    afisare("Nume fisier: " + f1.getName());
    afisare("Cale: " + f1.getPath());
    afisare("Cale absoluta: " + f1.getAbsolutePath());
    afisare("Parinte: " + f1.getParent());
    afisare(f1.exists() ? "exista" : "nu exista");
    afisare(f1.canWrite() ? "se poate scrie " : "nu se poate scrie");
    afisare(f1.canRead() ? "se poate citi" : "nu se poate citi");
    afisare((f1.isDirectory() ? "este ditrector" : "nu este director"));
    afisare("Ultima modificare: " + f1.lastModified());
    afisare("Dimensiune (in octeti): " + f1.length());
  }
}
```

Rezultatele afişate sunt:

```
Fisier creat cu succes
Nume fisier: TestFisiere.dat
Cale: TestFisiere.dat
Cale absoluta: D:\Work\DidaTec\laboratoare\Aplicatii\AppFisiere\PrFisiere\TestFisiere.dat
Parinte: null
exista
se poate scrie
se poate citi
nu este director
Ultima modificare: 1367743357677
Dimensiune (in octeti): 0
```

Pentru manipularea de directori o aplicaţie tipică este:

```
import java.io.File;

class TestDirectori {
  public static void main(String args[]) {
    String dirname = ".";
    File f1 = new File(dirname);

    if (f1.isDirectory()) {
      System.out.println("Continutul directorului " + dirname);
      String s[] = f1.list();

      for (int i=0; i < s.length; i++) {
        File f = new File(dirname + "/" + s[i]);
        if (f.isDirectory()) {
          System.out.println(s[i] + " > DIRector");
        } else {
          System.out.println(s[i] + " > FISier");
        }
      }
    } else {
      System.out.println(dirname + " nu este un DIRECTOR!");
    }
  }
}
```

Rezultatele afişate de aplicaţie sunt:

```
Continutul directorului .
classes > DIRector
PrFisiere.jpr > FISier
src > DIRector
TestFisiere.dat > FISier
```

Clasele Stream

Fluxuri binare (pe octet)
Clasele pentru fluxurile pe octet asigură un mediu pentru lucrul eficient în contextul operaţiilor de intrare/ieşire orientate pe octet. Fluxul pe octeţi poate fi utilizat pentru orice tip de obiect, inclusiv pentru date binare.

InputStream
InputStream este o clasă abstractă care define te modelul Java pentru fluxurile de intrare binare (pe octet). Toate metodele acestei clase generează excepţii IOException în caz de eroare.

OutputStream
OutputStream este o clasă abstractă ce defineşte modelul Java pentru lucrul cu fluxuri de ieşire binare. Toate metodele acestei clase întorc o valoare void şi generează excepţia IOException în caz de eroare.

Fluxuri pe caracter
Fluxurile pe caracter stiu manipula direct orice categorie de caractere (cele binare nu lucrează cu Unicode).

Reader
Reader este o clasă abstractă care defineşte modelul Java pentru fluxurile de intrare pe caractere. Toate metodele acestei clase generează excepţia IOException în caz de eroare.

Writer
Writer este o clasă abstractă ce defineşte modelul Java pentru lucrul cu fluxuri de ieşire pe caractere. Toate metodele acestei clase întorc o valoare void şi generează excepţsia IOException în caz de eroare.

Serializare

Serializarea este procesul de scriere a stării unui obiect într-un flux de octeţi. Acţiunea este utilă dacă se doreste salvarea stării unei aplicaţii într-o zonă de stocare persistentă cu sunt fişierele. La o rulare ulterioară obiectele pot fi refăcute la starea respectivă prin procesul de deserializare.

Serializarea este necesară şi pentru implementarea RMI (Remote Method Invocation). RMI permite ca un obiect Java de pe o maşină să apeleze o metodă Java a unui obiect de pe o altă maşină. Metodele apelate pot avea ca argumente obiecte. Maşina transmiţătoare serializează obiectul de transmis, iar maşina receptoare îl deserializează.

Serializable
Obiectele serializabile trebuie să implementeze obligatoriu interfaţa `Serializable`. Interfaţa `Serializable` nu are metode, scopul ei esta să marcheze faptul că se doreşste serializarea clasei. Dacă o clasă este serializabilă atunci toate subclasele ei sunt şi ele serializabile. Variabilele declarate static nu beneficiază de salvare prin serializare (la fel şi cele transient).

Externalizable
Procedurile Java de serializare şi deserializare au fost create pentru salvarea şi refacerea automtă a stării unui obiect. Există însă şi cazuri în care programatorul doreşte să controleze procesul. Pentru această situaţie se foloseşte interfaţa `Externalizable` care defineşte două metode:

```
void readExternal(ObjectInput inStream)
```

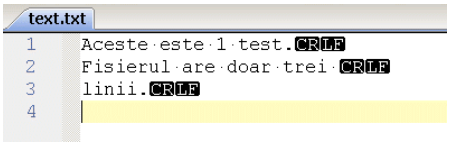
şi

```
void writeExternal(ObjectOutput outStream)
```

În aceste metode `inStream` este flux de octeţi de pe care obiectul se citeşte iar `outStream` un flux de octeţi pe care obiectul se scrie.

StreamTokenizer

Numeroase aplicaţii caută atomi lexicali pe fluxul de intrare. Şirul de caractere din flux conţine caractere speciale ce au rol de separare sau delimitare. Atomii lexicali sunt cuprinşi între aceste caractere speciale iar Java are o clasă specializată în acest sens `StreamTokenizer` ce se grupează carcterele de pe `InputStream` în atomi lexicali (token). Aplicaţia următoare face analiza lexicală a textului introdus în fişierul text.txt si numără (linii, cuvinte şi caractere).




```
import java.io.*;

class TestTokenizer {
    public static int w = 0;
    public static int l = 0;
    public static int c = 0;

    public static void wc(Reader r) throws IOException {
        StreamTokenizer tok = new StreamTokenizer(r);

        tok.resetSyntax();
        tok.wordChars(33, 255);
        tok.whitespaceChars(0, ' ');
        tok.eolIsSignificant(true);

        while (tok.nextToken() != StreamTokenizer.TT_EOF) {
            switch (tok.ttype) {
            case StreamTokenizer.TT_EOL:
                l++;
                c++;
                break;
            case StreamTokenizer.TT_WORD:
                w++;
            default: // FALLSTHROUGH
                c += tok.sval.length();
                break;
            }
        }
    }

    public static void main(String args[]) throws FileNotFoundException {
        File f1 = new File("text.txt");
        FileReader citestetxt = new FileReader(f1);
        try {
            wc(citestetxt);
           System.out.println(f1.getName() + " > linii:" + l + " cuvinte:" + w
+ " caractere:" + c);
        } catch (IOException e) {
            System.out.println("Eroare...");
        }
    }
}
```

Aplicaţia foloseşte variabilele w, l şi c pentru a contoriza cuvintele, liniile şi caracterele din fişier. Metoda `resetSyntax()` se folseşte pentru resetarea delimitatorilor. Atomii lexicali sunt cuvinte (şiruri de caractere vizibile pe ecran) separate prin sau încadrate între delimitatorul spaţiu (alb) şi se definesc folosind metoda `whitespacesChars()`. Metoda `eolIsSignificant()` face ca şi terminatorul de linie (newline) să fie considerat atom lexical pentru a putea număra şi liniile. Metoda `wordChars()` se foloseşte pentru a specifica domeniul de coduri ce se pot folosi în text pentru codificarea de caractere. Atomul lexical următor se obţine cu `nextToken()` de pe fluxul de intrare iar tipul acestuia poate fi unul dintre constantele TT_EOF, TT_EOL, TT_NUMBER şi TT_WORD.

C12- Întrebari

1. Ce este persistenta datelor, cum se ajunge la aceasta prin fisiere?
2. Cate clase abstracte pentru lucrul cu fluxuri sunt in Java si cand se folosesc acestea?
3. Cu ce clasa se manipuleaza sistemul de fisiere (al sistemului de operare), ce operatii sunt posibile?
4. Ce presupune procesul de serializare al unei clase si ce effort de implementare necesita?
5. Ce este o clasa Tokenizer?