Laborator Nr. 8:
Fluxuri Java

Întocmit de: Dobrinaş Alexandra

Îndrumător: Asist. Drd. Danciu Gabriel

November 15, 2011

I. NOŢIUNI TEORETICE

Fluxurile Java pun la dispoziţie modalitatea prin care o aplicaţie permite citirea unor informaţii care se găsesc pe o sursă externă, respectiv trimiterea unor informaţii către o destinaţie externă. Informaţia se poate găsi oriunde: într-un fişier pe disc, în reţea, în memorie sau în alt program şi poate fi de orice tip: date primitive, obiecte, imagini, sunete, etc. Mai mult, prin fluxuri este posibilă comunicarea între două sau mai multe fire de execuţie ale aceleiaşi aplicaţii. Fluxurile sunt secvenţe de octeţi. Indiferent de tipul informaţiilor, citirea/scrierea lor de pe un mediu extern, respectiv pe un mediu extern respectă următorii algoritmi:

- Citirea:

```
deschide canal comunicatie;
cat timp (mai sunt informatii) {
citeste informatie;
}
inchide canal comunicatie;
```

- Scrierea:

```
deschide canal comunicatie
cat timp (mai sunt informatii) {
scrie informatie;
}
inchide canal comunicatie;
```

A. Fluxuri pentru lucrul cu fişiere

Fluxurile pentru lucrul cu fişiere sunt cele mai uşor de înteles. Clasele care implementează aceste fluxuri sunt urmatoarele:

FileReader	caractere
FileWriter	caractere
FileInputStream	octeti
FileOutputStream	octeti

Constructorii acestor clase acceptă ca argument un obiect prin care se specifică fişierul folosit. Acesta poate fi un şir de caractere, un obiect de tip *File* sau un obiect de tip *FileDescriptor*.

Constructorii clasei FileReader (vezi şi API):

```
public FileReader( String fileName ) throws FileNotFoundException
public FileReader( File file ) throws FileNotFoundException
public FileReader( FileDescriptor fd )
```

Constructorii clasei FileWriter (vezi şi API):

```
public FileWriter( String fileName ) throws IOException
public FileWriter( File file ) throws IOException
public FileWriter( FileDescriptor fd )
public FileWriter( String fileName, boolean append ) throws IOException
```

Constructorii clasei FileOutputStream (vezi şi API):

```
FileOutputStream(File file) throws FileNotFoundException
FileOutputStream(File file, boolean append)  throws FileNotFoundException
FileOutputStream(FileDescriptor fd)
FileOutputStream(String name) throws FileNotFoundException
FileOutputStream(String name, boolean append) throws FileNotFoundException
```

Constructorii clasei FileInputStream (vezi şi API):

```
InputStreamReader(InputStream in)
    InputStreamReader(InputStream in, Charset cs)
    InputStreamReader(InputStream in, CharsetDecoder dec)
    InputStreamReader(InputStream in, String charsetName)  throws UnsupportedEncodingException
```

Cei mai uzuali constructori sunt cei care primesc ca argument numele fişierului. Aceştia pot provoca excepţii de tipul *FileNotFoundException* în cazul în care fişierul cu numele specificat nu există. Din acest motiv orice creare a unui flux de acest tip trebuie facută ţntr-un bloc *try....catch* sau metoda în care sunt create fluxurile respective trebuie să arunce excepţii de tipul *FileNotFoundException* sau de tipul superclasei *IOException*.

Următoarele programe copiază conţinutul unui fişier într-un alt fişier.

```
1  import java.io.*;
2  public class Copy1 {
3     public static void main(String[] args) throws IOException {
4
5        FileReader in = new FileReader("in.txt");
6        FileWriter out = new FileWriter("out.txt");
7
8        int c;
9        while ((c = in.read()) != -1)
10          out.write(c);
11
12       in.close();
13       out.close();
14    }
15 }
```

Se observă ca metoda main aruncă excepţii de tipul *IOException*, deci nu este necesară folosirea blocurilor *try....catch*. Pentru citirea şi scrierea din\în fişiere s-au folosit clasele *FileReader* şi textitFileWriter

În următorul exemplu este exemplificat modul de lucru cu clasele *FileInputStream* şi *FileOutputStream*, precum şi folosirea blocurilor *try...catch*.

```
1  import java.io.*;
2  public class Copy2 {
3     public static void main(String args[]) {
4
5        FileInputStream in;
6        FileOutputStream out;
7
8        int ok;
9        try {
10          in = new FileInputStream(args[0]);
11          try {
12             out = new
13             FileOutputStream(args[1]);
14             ok = 0;
15
16             while(ok != -1) {
17                try {
18                   ok = in.read();
19                   out.write(ok);
20                   System.out.print((char)ok);
21                }
22                catch (IOException e) {
23                      System.out.println(e.getMessage());
24                      System.exit(1);
25                }
26
27             }
28          }
29          catch (IOException e) {
30             System.out.println(e.getMessage());
31             System.exit(1);
32          }
33       }
34       catch (FileNotFoundException e) {
35          System.out.println(e.getMessage());
36          System.exit(1);
37       }
38    }
39 }
```

Se recomandă folosirea claselor *FileReader* şi *FileWriter* atunci când se citesc sau se scriu şiruri de caractere. Pentru citirea sirurilor de octeţi(de exemplu pentru imagini) se vor folosii clasele *FileOutputStream* şi *FileInputStream.*

B. Citirea şi scrierea cu ajutorul buffer-elor

Sunt folosite pentru a introduce o zonă tampon ţn procesul de scriere/citire a informaţiilor, reducând astfel numarul de accese la dispozitivul ce reprezintă sursa originală de date. Sunt mult mai eficiente decât fluxurile fără buffer şi din acest motiv se recomandă folosirea lor ori de câte ori este posibil.

Clasele pentru citirea/scrierea cu zonă tampon sunt:

BufferedReader	caractere
BufferedWriter	caractere
BufferedInputStream	octeti
BufferedOutputStream	octeti

Clasele *BufferedReader* şi *BufferedInputStream* citesc ţn avans date şi le memorează şntr-o zonă tampon. Atunci când se execută o operaţie *read()*, octetul citit va fi preluat din buffer. În cazul în care buffer-ul este gol citirea se face direct din flux şi, odata cu citirea octetului, vor fi memorati în buffer şi octeţii care îi urmează. Similar, se lucrează şi cu clasele *BufferedWriter* şi *BufferedOutputStream.*

Fluxurile de citire/scriere cu buffer sunt fluxuri de procesare şi sunt folosite prin suprapunere cu alte fluxuri.

Exemplu de folosire a zonei tampon pentru realizarea citirii din fişier:

```
BufferedInputStream out = new BufferedInputStream(new FileInputStream("out.dat"), 1024)
```

Constructorii clasei *BufferedReader* sunt:

```
BufferedReader( Reader in )
BufferedReader( Reader in, int dim_buffer )
```

Constructorii clasei *BufferedWriter* sunt:

```
BufferedWriter( Writer out )
BufferedWriter( Writer out, int dim_buffer )
```

Constructorii clasei *BufferedInputStream* sunt:

```
BufferedInputStream( InputStream in )
BufferedInputStream( InputStream in, int dim_buffer )
```

Constructorii clasei *BufferedOutputStream* sunt:

```
BufferedOutputStream( OutputStream out )
BufferedOutputStream( OutputStream out, int  dim_buffer )
```

În cazul constructorilor în care dimensiunea buffer-ului nu este specificată, aceasta primeâte valoarea implicită de 512 octeţi.

Metodele acestor clase sunt cele uzuale de tipul *read* şi *write* (vezi API Reader, API Writer). Pe lângă acestea, clasele pentru scriere prin buffer mai au şi metoda *flush* care goleşte explicit zona tampon chiar dacă aceasta nu este plină.

Exemplu pentru folosirea zonelor tampon:

```
1  BufferedWriter out = new BufferedWriter(new FileWriter("out.dat"), 1024); //am creat un flux cu buffer de 1024
       octeti
2  for(int i=0; i<1024; i++)
3     out.write(i); //bufferul nu este plin -> in fisier nu s-a scris nimic
4  out.flush(); //bufferul este golit -> datele se scriu in fisier
```

C. Intrări/Ieşiri formatate

Orice program Java are:

- o intrare standard;
- o ieşire standard;
- o ieşire standard pentru erori;

În general intrarea standard este tastatura, iar ieşirea standard este ecranul. Intrarea şi ieşirea standard sunt de fapt, obiecte pre-create ce descriu fluxuri de date pentru citirea respectiv scrierea la dispozitivele standard ale sistemului. Aceste obiecte sunt definite publice în clasa *System* şi sunt:

Variabila	Semnificaţie	Tip flux
System.in	flux standard de intrare	InputStream
System.out	flux standard de ieşire	PrintStream
System.err	flux standard pentru afişarea erorilor	PrintStream

Fluxul standard de iesire se foloseşte pentru afişarea datelor pe ecran, în modul consola: *System.out.println("mesaj")*. Fluxul standard pentru afişarea erorilor se foloseşte similar:

```
catch (IOException e) {
System.err.println("Eroare de intrare/iesire!")
}
```

Fluxurile de ieşire pot fi folosite, aşadar fără probleme deoarece tipul lor este *PrintStream*, clasa primitivă pentru scrierea efectivă a datelor. În schimb, fluxul standard de intrare *System.out*, de tip *InputStream* care este o clasa abstractă, deci pentru a-l putea utiliza va trebui să-l folosim împreuna cu un flux de procesare a datelor sau cu orice alt flux ce permite citirea efectiva a datelor.

Uzual, se foloseşte metoda *readLine* pentru citirea datelor de la tastatură şi din acest motiv vom folosi intrarea standard împreună cu o clasa de procesare care implementează metoda *readLine*. Exemplul tipic este:

```
BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
System.out.print("Introduceti o linie:");
String linie = stdin.readLine()
System.out.println(linie);
```

Exemplu: un program care afiseaza liniile introduse de la tastatura

```
1  import java.io.*;
2  public class Afisare {
3
4      public static void main(String[] args) {
5
6          BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
7          String s;
8          try {
9              while((s = stdin.readLine()).length() != 0)
10                 System.out.println(s); //Programul se opreste cu o linie vida
11         }
12         catch(IOException e) {
13             e.printStackTrace();
14         }
15     }
16 }
```

Se observă că metoda *readLine* poate provoca exceptii de tipul *IOException*.

II. TEME DE LABORATOR

1. Într-un fişier se regăsesc pe prima linie dimensiunile *nxm* ale unei matrice, iar pe următoarele linii elementele matricei corespunzatoare dimensiunilor date. Să se citească de la tastatură încă o matrice de dimensiune *mxn* şi să se scrie într-un alt fişier produsul celor doua matrici.(Se vor folosi clasele *FileWriter*, *FileReader*, iar citirea de la tastatură se va face folosind clasa *BufferedReader*)

2. Să se citească de la tastatură un user şi o parolă. Acestea se vor compara cu înregistrările existente în fişierul *parole.txt*. Dacă user-ul şi parola se regăsesc printre acestea (pe aceeaşi linie), se va afişa mesajul "acces permis", dacă se regăseşte doar user-ul, iar parola este greşită se va afişa "parola gresita" şi se va mai cere introducerea parolei încă o dată, dar nu mai mult de 3 ori, dacă se atinge acest prag se va afişa mesajul "cont blocat". În caz contrar se reia procesul de introduce a datelor, dar nu mai mult de 5 ori. Dacă se atinge limita de 5 intrări se va afişa mesajul "Nu ai cont. Inregistreaza-te." (Se vor folosi clasele *FileInputStream* şi *FileOutputStream*.)
   Exemplu de date in fisierul parole.txt:
   user user
   gigi parolaMea
   user1 parola1

3. Într-un fişier numit *clienti.txt* sunt memorate date despre clienţii unui magazin virtual. Pe fiecare linie se reţine numele, prenumele şi vârsta clienţilor. Se cere să se afişeze numărul şi lista clienţilor majori şi numărul clienţilor minori.

!!! Pentru toate problemele se vor folosii mecanisme de aruncare a excepţilor.