1. Clasele File şi RandomAccessFile.

Clasele File şi RandomAccessFile furnizează funcționalitate pentru navigare pe sistemul local, descriind fişierele, directoarele şi accesul direct (nesecvential) la fişiere. Accesul secvential al fişierelor se face cu fluxuri (streams), cititori (readers) şi scriitori (writers).

Reprezentarea textului şi codificarea caracterelor

JAVA utilizează două tipuri de reprezentări text:
- Unicode - pentru reprezentarea internă a caracterelor şi string-urilor
- UTF - pentru intrări şi ieşiri

Unicode foloseşte 16 biți pentru reprezentarea unui caracter. Dacă cei mai semnificativi 9 biți sunt toți 0, atunci codificarea este standard ASCII (ultimii 7 biți fiind reprezentarea caracterelor). Codificarea Unicode este suficientă pentru majoritatea limbilor, dar limbile asiatice grafice reprezintă o problemă (sunt prea multe caractere grafice). Soluția constă in utilizarea UTF (Universal Character Set Transformation Format). Codificarea UTF foloseşte exact numărul dorit de biți: mai puțini pentru alfabete mici, mai multe pentru alfabete mari.

O codificare a caracterelor este o corespondenta bijectivă dintre multimea caracterelor şi o multime de numere binare. Fiecare platforma JAVA are o codificare a caracterelor implicită, care este folosită pentru interpretare dintre Unicode intern şi octeți externi. Codificarea implicită a caracterelor reflectă limba şi cultura locala. Fiecare codificare are un nume. De exemplu, "8859_1" inseamnă ASCII, "8859_8" este ISO Latin/Hebrew şi "CP1258" este vietnameză.

Când are loc o operație de I/O in JAVA, sistemul trebuie să cunoască codificarea caracterelor. Clasele I/O utilizează de obicei codificarea implicită locală, in caz că nu se precizează una explicită. În aplicatiile de retea, când se comunică intr-o rețea cu un alt computer, acestea trebuie să folosească aceeaşi codificare.

1.1. Clasa File

Clasa File pune la dispoziție facilități pentru manipularea fişierelor şi directoarelor. Programele Java ce folosesc această clasa trebuie să conțină:

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

Un constructor pentru această clasă este

```
File(String numeCale);
```

Construcția unei instanțe a lui *File* nu crează un fişier pe sistemul local, ci doar o instanță ce incapsulează şirul specificat.

Exemplu:
```
File fis = new File("c:\\jdk121.1\\bin\\javac.exe");
```

Mai există două variante de constuctori din clasa File:

a) File(String dir, String subCale);

Exemplu: Furnizăm un director şi o cale relativă:

```
File f = new File("/tmp", "xyz");
```

b) File(File dir, String subCale);

Exemplu: Furnizam o instanță a lui File şi o cale relativă:

```
File f1 = new File("C:\\tmp");
File f2 = new File(f1, "xyz.java");
```

Funcții utile pentru lucrul cu fişiere:

1.public String getPath() - furnizează calea relativă;

Exemplu: String caleRelativa = fis.getPath();
---
2. public String getAbsolutePath() - furnizează calea absolută;

Exemplu: String caleAbsoluta = fis.getAbsolutePath();
---
3. public String getCanonicalPath() - furnizează calea canonica a unui fişier sau director. Este similar cu getAbsolutePath(), dar simbolii . şi .. sunt rezolvati;

4. public boolean exists() - returnează true dacă fişierul sau directorul specificat există;

Exemplu: boolean există = fis.exists();
---
5. public boolean isDirectory() - returnează true dacă fişierul există şi este director;

Exemplu: boolean esteDirector = fis.isDirectory();
---
6. public boolean isFile() - returnează true dacă fişierul există pe sistemul de fişiere;

Exemplu: boolean esteFişier = fis.isFile();
---
7. public long length() - returnează lungimea fişierului in octeți sau 0 dacă nu există;

Exemplu: long lungime = fis.length();
---
8. public String[] list() - returnează un şir de nume de fişiere din directorul specificat de obiectul fişier;

Exemplu: String [] continutDirector = fis.list();

Dacă fişierul fis nu este director atunci funcția va intoarce null.

9. boolean canRead() - returnează true dacă fişierul sau directorul poate fi citit;

10. boolean canWrite() - returnează true dacă fişierul sau directorul poate fi modificat;

11. boolean delete() - returnează true dacă fişierul sau directorul poate fi şters;

12. boolean mkdir() - returnează true dacă s-a reusit crearea unui director a cărui cale este descrisă in File;

13. boolean renameTo(File noulNume) - returnează true dacă s-a putut redenumi fişierul sau directorul, altfel returnează false;

Exemplu:

```
import java.io.*;
public class Fişiere
 {
  public static void main(String args[])
   {
    int i;
    try
    {          boolean reusita;
               File dirNou = new File("C:\\dirProba");
               if (dirNou.isDirectory())
               {
                 System.out.println("Directorul C:\\dirProba există deja !");
               }
               reusita = dirNou.mkdir();
               if (reusita)
                 System.out.println("Am reusit sa cream directorul C:\\dirProba");
               else
                 System.out.println("Nu am creat directorul C:\\dirProba !");
               }
               catch (IOException e)
               {
                 System.out.println("Eroare creare director " + e);
               }
  }
}
```

1.2. Clasa RandomAccessFile

Această clasă se ocupă de studiul fişierelor binare (cu acces aleator direct), şi nu a fluxurilor fişiere de caractere (cu acces secvential). Intr-un fişier cu acces aleator, putem căuta o dată de la o anumită pozitie, putem scrie sau citi date la o poziție precizată. Clasa java.io.RandomAccessFile are doi constructori:

- RandomAccessFile(String fişier, String mod);
- Random AccessFile(File fişier, String mod);

String-ul mod poate fi "r" (dacă se deschide in acces de citire) sau "rw" pentru citire şi scriere. A doua formă de constructor este utilă când există deja o instanță a clasei File.

Exemplu: Inainte de a deschide un fişier in acces de citire/scriere, facem verificări:

```
 File fişier = new File(cale);
 if (! Fişier.isFile() || ! fişier.canRead() || ! fişier.canWrite())
  {
   throw new IOException();
  }
 RandomAccessFile raf = new RandomAccessFile(fişier, "rw");
```

Metodele de căutare din această clasă sunt:

- long getFilePointer() throws IOException; - returnează pozitia curentă a pointerului din fişier (in octeți).

- long length() throws IOException; - returnează lungimea fişierului, in octeți;
- void seek(long pozitie) throws IOException; - setează poziția curentă din fişier (in octeți). Fişierele incep cu pozțtia 0.

Metodele care suportă citirea şi scrierea de octeți sunt:

- int read() throws IOException; - returnează următorul octet din fişier (memorat in octetul cel mai nesemnificativ al unui int) sau -1 dacă pointerul este la sfârşitul fişierului;
- int read(byte destinatie[]) throws IOException; - incearcă să citească suficienți octeți pentru a umple şirul destinatie[]. Returnează numărul de octeți citiți din fişier sau -1 dacă pointerul este la sfârşitul fişierului;
- int read(byte destinatie[], int deplasament, int lungime) throws IOException; - incearcă să citească lungime octeți in şirul destinatie[], incepând de la deplasament. Returnează numărul de octeți cititi din fişier sau -1 dacă pointerul este la sfârşitul fişierului;
- int write(int b) throws IOException; - scrie octetul cel mai nesemnificativ a lui b;
- int write(byte b[]) throws IOException; - scrie toți octeții din şirul b[];
- int write(byte destinatie[], int deplasament, int lungime) throws IOException; - scrie lungime octeți din şirul destinatie[], incepând de la poziția deplasament.

Clasa RandomAccessFile are metode pentru citirea şi scrierea tuturor tipurilor primitive de date. Acestea sunt:

- boolean readBoolean();
- void writeBoolean(boolean b);
- byte readByte();
- void writeByte(int b);
- short readShort();
- void writeShort(int s);
- char readChar();
- void writeChar(int c);
- int readInt();
- void writeInt(int i);
- long readLong();
- void writeLong(long l);
- float readFloat();
- void writeFloat(float f);
- double readDouble();
- void writeDouble(double d);
- int readUnsignedByte();
- int readUnsignedShort();
- String readLine();
- String readUTF();
- void writeUTF(String s);

Când un fişier cu acces aleator nu mai este folosit, atunci acesta trebuie inchis:
- void close() throws IOException;
Apelul acestei metode eliberează resursele sistemului de memorie asociate fişierului.