Példa:

A korábbi, rendezett láncolt lista feladatunkat egészítsük ki úgy, hogy a lista tartalmát elmentjük szöveges fájlba. A program billentyűzetről feltölti az alábbi összetett adatszerkezetet (struktúrát), majd *azon*-ra rendezetten eltárolja egy láncolt listában. Az adatokat addig olvassa be a program, amig az *azon*-ra nullát nem adunk.

int azon;
string név;
float fizetés;

Megbeszélés:

1. karakterlánc részekre (tokenekre) bontása: *StringTokenizer* osztály
2. A *StringTokenizer* típus és a *StringTokenizer()* metódus.

Megoldás:

```
package file_ob;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import static java.lang.System.out;
import java.util.LinkedList;
import java.util.Scanner;
import java.util.StringTokenizer;
public class File_OB {
    public static void main(String[] args) throws IOException{
        LinkedList<OB_def> adatok = new LinkedList<>();
        feltölt(adatok);
        fájlírás(adatok);
        képernyő(adatok);
    }
    static void fájlírás(LinkedList<OB_def> adatok) throws IOException{
        PrintWriter file = new PrintWriter(new FileWriter("Adatok.txt"));
        for(int i = 0; i < adatok.size(); i++){
            file.print(adatok.get(i).azon+";");
            file.print(adatok.get(i).nev+";");
            file.println(adatok.get(i).fizetes);
        }
        file.close();
    }
    static void  feltölt(LinkedList<OB_def> adatok){        //Rendezett feltöltés
        Scanner bill = new Scanner (System.in, "ISO8859_2");
        OB_def OB = new OB_def();
        int i;
        out.printf("Kérem az azonosítót: ");
        OB.azon = bill.nextInt();
```

```
                while(OB.azon!=0){
                        out.printf("Kérem a nevet: ");
                        OB.nev = bill.next();
                        out.printf("Kérem a fizetést: ");
                        OB.fizetes = bill.nextFloat();
                        for(i=0; i<adatok.size(); i++){
                                if(adatok.get(i).azon > OB.azon){
                                        adatok.add(i, OB);
                                        break;
                                }
                        }
                        if( i == adatok.size())adatok.add(OB);
                        OB = new OB_def();
                        out.printf("Kérem az azonosítót: ");
                        OB.azon = bill.nextInt();
                }
        }
        static void képernyő(LinkedList<OB_def> adatok)throws IOException{     //Kiírás
                BufferedReader file = new BufferedReader(new FileReader("Adatok.txt"));
                String sor;
                StringTokenizer token;
                while((sor = file.readLine()) != null){
                        token = new StringTokenizer(sor, ";");
                        out.print(token.nextElement() + " - ");
                        out.print(token.nextToken() + " - ");
                        out.println(token.nextElement());
                }
                file.close();
        }
}
class OB_def{
        public Integer azon;
        public String nev;
        public Float fizetes;
}
```

1. Bináris fájl kezelése

Ekkor - ellentétben a szöveges állományokkal – itt nem történik konverzió, hanem a változók bájtjait egy az egyben kiírjuk az állományba.
A bináris állományok kezelésére a *FileInputStream* és a *FileOutputStream* osztályok *read(), write()* és *close()* metódusait használjuk, hasonlóan a szöveges állományokhoz.
Itt is kötelező az *IOException* osztály használata.

Példa:

Írjuk ki egy bináris fájlba 48 és 57 közötti egész számokat (0-9 karakter kódjai), majd olvassuk vissza szövegként.

Megbeszélés:

1. Fájlkezelés, bináris fájlok, fájlok megnyitása, lezárása.
2. A *write()* és A *read()* metódusok.

Megoldás: 1

```
package file_binary_1;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import static java.lang.System.err;
import static java.lang.System.out;
public class File_binary_1 {
	public static void main(String[] args) {
		try{
			FileOutputStream out = new FileOutputStream("Teszt.bin");
			for(int i = 48; i <= 57; i++) out.write(i);
			out.close();
		}
		catch (IOException error){
			err.println("Írási hiba - " + error.getMessage());
		}
		try {
			FileReader in = new FileReader("Teszt.bin");
			int c;
			while ((c = in.read()) != -1){
				out.print((char)c + " ");
			}
			out.println();
			in.close();
		}
		catch (IOException error){
			err.println("Olvasási hiba - " + error.getMessage());
		}
	}
}
```

Bájtnál hosszabb értékeket nem lehet kiírni vagy beolvasni *FileInputStream.read()* illetve a *FileOutputStream.write(()* metódusokkal. Abban az esetben, ha például egész értékeket (4 bájtos) akarunk használni, akkor ezt a *DataOutputStream.writeInt()* illetve a *DataInputStream.readInt()* metódusokkal tehetjük meg.

Példa:

Írjuk ki tetszőleges mennyiségű egész számot egy bináris fájlba, majd olvassuk azokat vissza.

Megbeszélés:

1. Fájlvég lekezelése kivétellel.

Megoldás: 1

```
package file_binary_2;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import static java.lang.System.err;
import static java.lang.System.out;
import java.util.InputMismatchException;
import java.util.Scanner;
public class File_binary_2 {
	public static void main(String[] args) {
		Scanner bill = new Scanner (System.in);
		int szam;
		try{
			FileOutputStream bki = new FileOutputStream("Teszt.bin");
			DataOutputStream ki = new DataOutputStream(bki);
			do{
				out.print("Kérek egy számot: ");
				try{
					szam = bill.nextInt();
					out.println(szam);
					ki.writeInt(szam);
				}
				catch (InputMismatchException error){
					break;
				}
			}while(true);
			ki.close();
		}
		catch (IOException error){
			err.println("Írási hiba - " + error.getMessage());
		}
```

```
                try{
                        FileInputStream bbe = new FileInputStream("Teszt.bin");
                        DataInputStream be = new DataInputStream(bbe);
                        boolean noteof = true;
                        while (noteof){
                                try{
                                        szam = be.readInt();
                                        out.println(" " + szam);
                                }
                                catch (EOFException e){                 //Fájl vég
                                        noteof = false;
                                }
                        }
                }
                catch (IOException error){
                        err.println("Olvasási hiba - " + error.getMessage());
                }
        }
}
```

2. Közvetlen elérésű (random) fájl kezelése

Az eddig megismert módszerek segítségével a fájlhoz sorosan férhettünk hozzá, vagyis az adatok írása vagy olvasása sorban, egymás után történik. Abban az esetben, ha egy fájl tartalmát közvetlenül szeretnénk elérni, akkor ehhez a *java.io* csomag *RandomAccessFil*e osztályát kell használnunk. Ez az osztály írásra és olvasásra egyaránt alkalmas.
Fájl megnyitása:

- csak olvasásra: *RandomAccessFile("fájlnév", "r"),*
- olvasásra és írásra: *RandomAccessFile("fájlnév", "rw").*

Fájlkezelő metódusok:

- írás, olvasás *read()* és *write()* metódusokkal, használatukkor a fájlmutató automatikusan egy adategység bájtméretével megnő,
- a fájlmutató mozgatása megadott pozícióra: *void seek(long pos)*, *pos* a fájl elejétől, bájtokban megadott pozíció,
- a fájlmutató mozgatása az aktuális pozícióhoz képest *n* bájttal előre: *int skipBytes(int n),*
- a fájlmutató aktuális pozíciójának lekérdezése *long getFilePointer()*.

Ebben az esetben is kötelező az *IOException* kivételkezelő használata.

Példa:

Írjuk ki egy random fájlba 48 és 57 közötti egész számokat (0-9 karakter kód), majd próbáljuk ki a fájlmutató pozicionáló metódusokat.

Megbeszélés:

1. Fájlkezelés, random fájlok, fájlok megnyitása, lezárása.
2. A *seek()*, a *skipBytes()* és a *getFilePointer()* metódusok.

Megoldás: 1

```
package file_random_1;
import java.io.IOException;
import java.io.RandomAccessFile;
import static java.lang.System.err;
import static java.lang.System.out;
public class File_Random_1 {
    public static void main(String[] args) {
        int szam;
        try{
            RandomAccessFile file = new RandomAccessFile("Teszt.txt", "rw");
            for(int i = 48; i <= 57; i++) file.write(i);
            file.seek(0);
            out.println(file.read());
            file.skipBytes(2);
            out.println(file.read());
            out.println("Aktuális pozició: " + file.getFilePointer());
            file.close();
        }
        catch (IOException error){
            err.println("Fájl hiba - " + error.getMessage());
        }
    }
}
```

Példa:

Írjuk ki egy random fájlba az angol ABC nagybetűit. Cseréljük ki minden harmadik betűt kisbetűre. Használjuk a *finally* blokkot.

Megoldás:

```
package file_random_2;
import java.io.IOException;
import java.io.RandomAccessFile;
import static java.lang.System.err;
import static java.lang.System.out;
public class File_Random_2 {
	public static void main(String[] args) {
		int kód;
		RandomAccessFile file = null;
		try{
			file = new RandomAccessFile("Teszt.txt", "rw");
			for(char c = 'A'; c <= 'Z'; c++) file.write(c);
			file.seek(0);			//Vissza a fájl elejére
			while((kód=file.read())!= -1)out.print((char) kód + " ");
			out.println();
			long poz = 2;
			while (poz < file.length()){
				file.seek(poz);
				kód = file.read();
				file.seek(file.getFilePointer()-1);
				file.write(kód + 32);
				poz += 3;
			}
			file.seek(0);
			while((kód=file.read())!= -1)out.print((char) kód + " ");
			out.println();
			file.close();
		}
		catch (IOException error){
			err.println("Fájl hiba - " + error.getMessage());
		}
		finally {
			try {
				if (file != null) file.close();
			}
			catch (IOException error) {
				err.println(error.getMessage());
			}
		}
	}
}
```

Feladatok:

1. Írjon programot, amely beolvassa e program forrásállományát (.java fájl), meghatározza soronként a szám karakterek számát, amit egy bináris állományban elment. A fájl tartalmát írja ki a képernyőre.

2. Írjon programot, amely metódus segítségével beolvassa e program forrásállományát (.java fájl) meghatározza soronként a karakterek számát A metódus visszaadott értéke a karakterek száma, amit a main egy bináris állományba ír. A fájlt írja ki a képernyőre.

3. Írjon programot, amely metódus segítségével beolvassa e program forrásállományát (.java fájl) és meghatározza soronként a szám karakterek számát. A metódus visszaadott értéke a megszámolt karakterek száma legyen, amit a main egy random állományba ír ki. A fájl tartalmát írja ki a képernyőre.

4. Írjon programot, amelyben a main metódus soronként beolvassa e program forrásállományát (.java fájl), egy metódus segítségével megszámolja az egy sorban lévő karakterek számát. A metódus paramétere a sort tartalmazó string, és a fájlobjektum. A visszaadott érték a karakterek száma, amit a main egy random állományba kiír. A fájl tartalmát visszafelé olvassa ki és írja azt a képernyőre.

5. Készítsen programot, amelyben a main metódus – üres string végjelig – sorokat olvas be a billentyűzetről, majd egy metódus segítségével kiírja azokat sorszámmal ellátva egy bináris állományba (a metódus paraméterei: fájlobjektum, sorszám és a sort tartalmazó string). A fájl tartalmát olvassa vissza és írja ki a képernyőre.

6. Készítsen programot, amely egy metódus segítségével – üres string végjelig – sorokat olvas be a billentyűzetről (a metódus paramétere a sort tartalmazó string), majd egy másik metódus segítségével kiírja azokat egy bináris állományba (a függvény paraméterei: fájlobjektum és a sort tartalmazó string). A fájl tartalmát írja ki a képernyőre.

7. Írjon programot, amelyben a main metódus beolvas billentyűzetről valós számokat (tetszőleges végjelig), és azokat egy bináris fájlban tárolja. Majd egy metódus segítségével visszaolvassa a fájl tartalmát és kiszámítja a tárolt értékek átlagát. A metódus paramétere a fájlobjektum, a visszaadott érték az átlag, amit a main ír ki.

8. Írjon programot, amelyben a main metódus beolvassa e program forrásállományát soronként, egy metódus segítségével megszámolja a sorban lévő számkarakterek számát. A metódus paramétere a sort tartalmazó string, a visszaadott érték a karakterek száma, amit sorszámmal együtt a main egy random állományba ír. A fájl tartalmát írja ki a képernyőre.

9. Készítsen programot, amely a main metódus segítségével sorokat olvas be a billentyűzetről (tetszőleges végjelig), majd meghatározza a sorok hosszát (a sorokban lévő karakterek számát), egy metódus segítségével kiírja a sorokat a sorok hosszával együtt egy random állományba (a metódus paraméterei: fájlobjektum, a sort tartalmazó sting és a sorhossz). A fájl tartalmát visszafelé olvassa ki és írja azt a képernyőre.

10. Készítsen programot, amely a main metódusban, egymás után bekéri több téglatest három méretét (tetszőleges végjelig). Metódus segítségével számítsa ki a téglatestek térfogatát (a metódus paraméterei a három méret, a visszaadott érték a térfogat). A téglatestek méreteit és a térfogatát a main kiírja egy random állományba. A fájlból minden harmadik sort olvasson ki és írja azt a képernyőre.