# 19. Bináris fájlok

## Szöveges és bináris fájlok

Az adatokat kétféleképpen tárolhatjuk fájlokban:

* A szöveges fájlok karakterekből állnak, ezért olvashatók.   
  (pl. 12345 -> ’1’,’2’,’3’,’4’,’5’)
* A bináris fájlok bájtokból állnak, és nem olvashatók.  
  (pl. 12345 -> 0 0 48 57, mert 12345 = 48x256 + 57)

Melyik foglal kevesebb helyet?   
Melyiket kell átalakítani beolvasás után vagy kiírás előtt?  
Ezek alapján melyik tárolás hatékonyabb?

Megjegyzés: a szöveges fájlok kezelhetők binárisan, bájtonként is, de fordítva nem!

A szöveges fájlokat a Scanner és a PrintWriter osztályok segítségével olvassuk. illetve írjuk. Ezek szükség esetén kezelik a különböző kódolásokat (UTF-8, Cp1250, …)

Ebben a leckében a bináris fájlok kezelésével foglalkozunk.

## Bináris fájlok írása, olvasása

A bináris fájlok bájtonkénti olvasásához egy FileInputStream-et kell készíteni:

FileInputStream be = new FileInputStream("input.bin");

Ennek van egy read metódusa, amely vagy a következő bájtot adja vissza (0‑255), vagy -1-et, ha nincs több bájt (vagyis a fájl végére értünk). A visszaadott érték int típusú.

Például:

FileInputStream be = new FileInputStream("input.bin");  
int b = be.read();  
if (b != -1) {  
 // bájt feldolgozása…  
}

A bináris fájlok írásához egy FileOutputStream-et kell készíteni:

FileOutputStream ki = new FileOutputStream("input.bin");  
int b = 123; // 0-255  
ki.write(b);

Ezeket a fájlokat is be kell zárni. Ha azt szeretnénk, hogy automatikusan zárja be a megnyitott fájlokat, akkor a try-catch szerkezetet kell használni, és zárójelben megnyitni a fájlokat. Például:

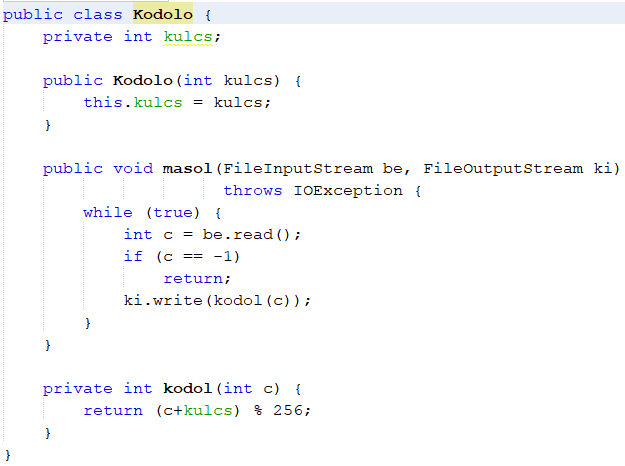
try (FileInputStream be = …, FileOutputStream ki = …) {  
 // beolvasás, kiírás  
} catch (IOException e) {  
 …  
}

## Titkosítás

Példaképpen egy olyan programot készítünk, amely bájtonként kódol egy szöveget.

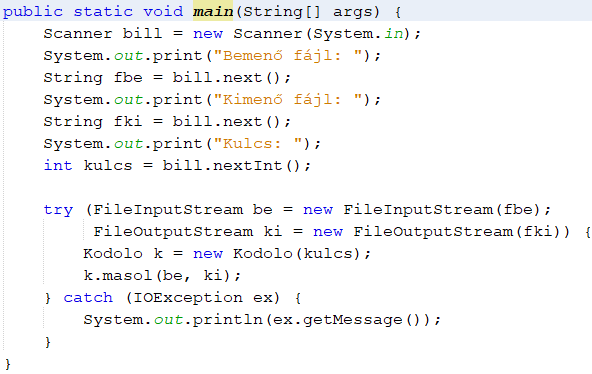
Nyisd meg a *caesar2* projektet a NetBeans-ben!

Nyisd meg a Kodolo osztályt, és gondold végig a működését!



Hogyan végzi az eltolást a kodol metódus?

Nézd meg a Caesar2 osztályt is, és gondold végig a működését!



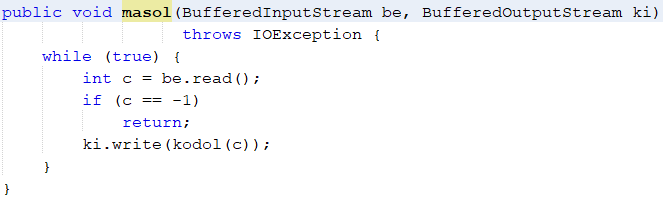
Próbáld ki mindkét kapott szöveges fájllal oda-vissza a kódolást! Milyen karakterkódolásnál használható a program?

Lehet-e más fájlt, például egy képet kódolni vele? Próbáld ki a kapott képpel! Figyeld meg, mennyi ideig tart az átalalakítás!

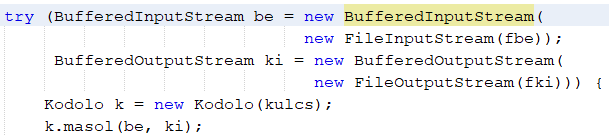
## Írás és olvasás gyorsítása

A program bájtonként olvas és ír. Nagyobb fájloknál ez lassú lehet. A műveletek felgyorsíthatók, ha a FileInputStream helyett a BufferedInputStream, a FileOutputStream helyett pedig a BufferedOutputStream osztályt használjuk. Ezek nagyobb blokkokban végzik az olvasást, illetve az írást, ami sokkal gyorsabb.

Írd át a Kodolo osztály masol metódusát így:



A Caesar2 osztály main metódusában a try ágat kell módosítanod:



Próbáld ki így is a kép kódolását! hasonlítsd össze a sebességet az előző változat sebességével! Mit tapasztalsz?

## Objektumok írása és olvasása

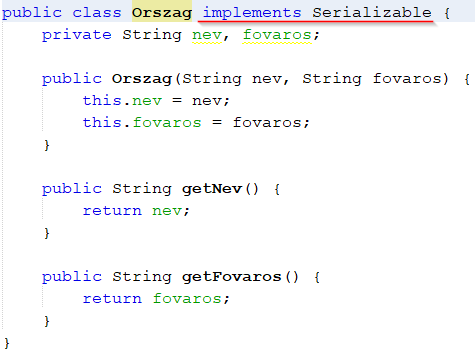
Az ObjectInputStream, illetve az ObjectOutputStream osztályok segítségével egy teljes objektumot (pl. egy listát) tudunk egyszerre olvasni, illetve írni. Lista esetén azokat az objektumokat is menti, amelyekre a listaelemek hivatkoznak.

Ehhez a mentett objektumoknak meg kell valósítaniuk a Serializable interface-t. Ebben nincsenek metódusok, csak az osztály fejlécében kell jelezni az implementációt. A listákban ez automatikusan megvan, a listák elemeinél meg kell adnunk.

Az orszagok programban ezt próbáljuk ki. Ország-főváros párokat fogunk egy listába tenni, majd elmenteni, illetve a következő indításkor megnyitni.

Nyisd meg a projektet a NetBeans-ben!

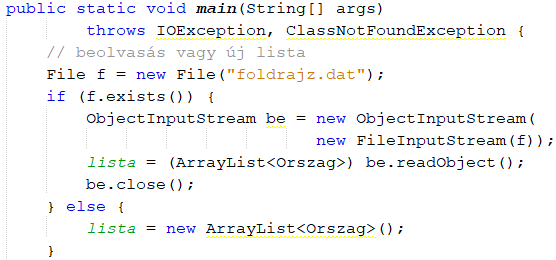
Nézd meg először az Orszag osztályt, amelyben egy országot és a hozzá tartozó fővárost kezeljük. Figyeld meg, hogy jeleztük a Serializable interface megvalósítását, de nem kell egyetlen absztrakt metódust sem felülírnunk!



Ezután nyisd meg az Orszagok osztályt! Ebben először deklarálunk egy új listát az országoknak:



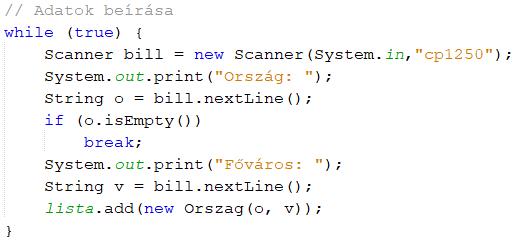
A main metódusban először egy File objektumot. Ha létezik a megadott nevű fájl a háttértáron, akkor beolvassuk a listába, egyébként pedig létrehozunk egy új, üres listát:



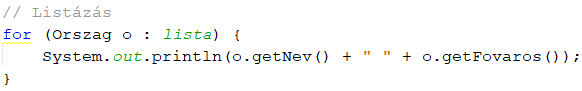
Figyeld meg, hogyan vizsgáljuk a fájl létezését és hogyan nyitjuk meg az ObjectInputStream-et! A readObject metódus egy Object típusú objektumot ad vissza, ezt át kell alakítanunk listává.

Futtasd le a programot kétszer! Figyeld meg, hogy az első után létrehoz egy *foldrajz.dat* fájlt! A későbbiekben ebbe fog menteni.

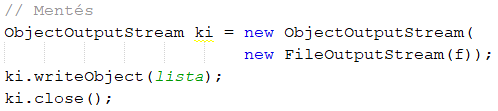
A main metódus folytatásában lehetőség van ország-főváros párok bevitelére. Az üres országnévvel jelezzük a bevitel végét:



A bevitel után a program kiírja a listában lévő adatokat a konzolra:



Végül jön a mentés:



Próbáld ki ismét a programot! Írj be néhány adatot! Lépj ki, majd indítsd el újra! Ellenőrizd, hogy betöltötte-e az adatokat! Nézd meg a *foldrajz.dat* fájl tartalmát!

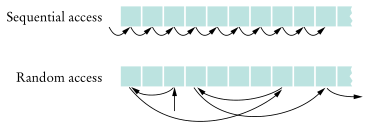
## Véletlen hozzáférés

Az eddigiekben sorban egymás után (szekvenciálisan) olvastuk a fájlokból vagy írtuk a fájlokba az adatokat. Módsítás után újra kiírtuk az adatokat a lemezre.

Nagy fájlok esetén ez sokáig tarthat, és lehetséges, hogy a memória nem elég az összes adat beolvasásához.

Ilyen esetekben jobb, ha a lemezen lévő fájl egy tetszőleges részét tudjuk módosítani vagy beolvasni. Ezt lehet megvalósítani a véletlen hozzáféréssel (random access).

Ennek a nevével ellentétben nincs köze a véletlenhez, csak azt jelenti, hogy a fájl tetszőleges pozíciójára ugorhatunk, és onnan olvashatunk vagy oda írhatunk.



A véleltlen elérésű fájlok kezeléséhez a RandomAccessFile osztályt használjuk. Például:

RandomAccessFile f = new RandomAccessFile("bank.dat", "rw");

A konstruktor első paramétere a fájl neve, második pedig a megnyitás módja. Ez lehet r vagy rw. Az r esetén csak olvasni lehet a fájlból, az rw esetén pedig írni is lehet bele.

A véletlen elérésű fájlokban van egy mutató, amely a következő olvasási/írási pozícióra mutat. Ez alapesetben a fájl vége.

A mutató mozgatásához a seek metódus használható:  
f.seek(pozíció);

A fájl első bájtja a 0-s pozíció, és utána bájtonként egyesével növekszik. Az aktuális pozíciót a getFilePointer metódussal lehet lekérdezni, amely long típusú eredményt ad:

long poz = f.getFilePointer();

A véletlen elérésű fájlokban az összetartozó adatokat fix hosszúságú rekordokban tároljuk. Így könnyű meghatározni egy adott sorszámű rekord pozícióját: a rekord méretét meg kell szorozni a sorszámmal. A fájl ebben az esetben egy tömbhöz hasonlóan használható.

A fájl hosszát bájtokban a length metódussal határozhatjuk meg:

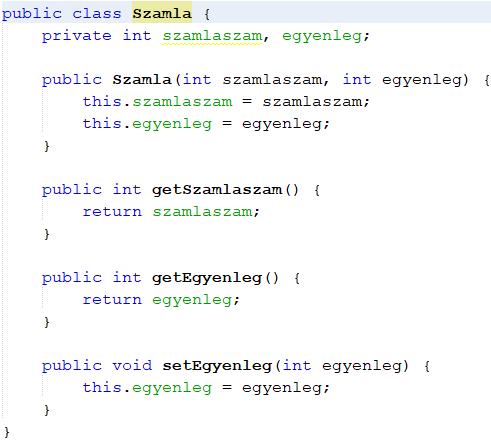
long hossz = f.length();

Ha azt szeretnénk megtudni, hogy rekord van a fájlban, a hosszt el kell osztani a rekordok méretével.

Egész típusú számok a readInt és a writeInt metódusokkal, a törtek a readDouble és a writeDouble metódusokkal olvashatók, illetve írhatók.

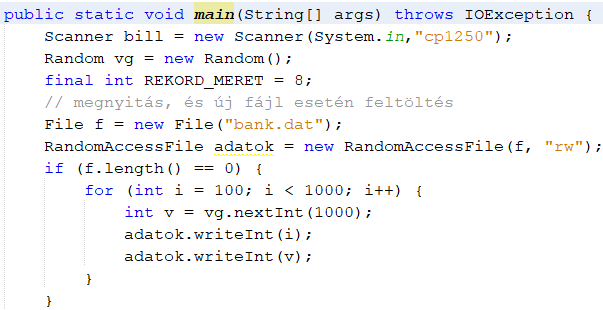
A következő példaprogramban egy bank bankszámláinak adataival dolgozunk. Nyisd meg a kapott *bank* projektet!

Először nézd meg a Szamla osztályt! Milyen adatokat tárolunk minden számlánál? Mekkora lesz a rekordok mérete?



Ezután válts a Bank osztályra!

A main metódusban a megnyitás után megviszgáljuk a fájl méretét. Ha nulla, vagyis üres a fájl, akkor feltöltjük adatokkal. Hány számla lesz, mi lesz az egyenlegek értéke az egyes számláknál?



Futtasd a programot, majd nézd meg a létrehozott fájlt! Mekkora a mérete, és miért?

A main metódus második felében bekérünk számlaszámokat. (100 alatti vagy 999 feletti számmal lehet kilépni.)

Ezután a fájlban az adott számlát tartalmazó rekordhoz ugrunk, és kiolvassuk az adatait.

A kiolvasás után visszaugrunk a rekord elé, hogy tudjuk módosítani, majd bekérünk egy betétet, vagy negatív szám esetén kivétet. Ezt hozzáadjuk az egyenleghez, majd kiírjuk a képernyőre és a fájlba is.

A program végén bezárjuk a fájlt.



Próbáld ki! Változik a fájl mérete a módosítások után?