# 17. Fájlok 4.

## Caaesar kódolás

Julius Caesarnak tulajdonítják a következő titkosírási módszert. Lényege, hogy minden karaktert az utána következő valahányadik (például harmadik) karakterrel helyettesítünk.

Ha az eltolás túlmutat az ábécé végén, az ábécé elejéről vesszük a karaktereket. Ehhez a kapott kódból le kell vonni az ABC hosszát (26).

Például a +3-as eltolás:

ABCDEFGHIJKLMNOPQRSTUVWXYZ  
ABCDEFGHIJKLMNOPQRSTUVWXYZABC

A kódolásban a fenti táblázat segít: megkeressük a karaktert a felső sorban, és helyettesítjük az alatta lévővel.

Példa: ABLAK 🡪 DEODN

Mi lesz a kódja a POGANY szónak? Írd ide:

Az üzenet olvasásakor ellenkező irányba kell eltolni a karaktereket ugyanannyival (vagyis -3-as eltolás). Ha az ABC elé kerülünk, a kódhoz hozzá kell adni az ABC hosszát.

ABCDEFGHIJKLMNOPQRSTUVWXYZ  
XYZABCDEFGHIJKLMNOPQRSTUVWXYZ

Mit jelent a kódolt VLPRQBL szó? Írd ide:

Ez a kódolás még könnyebb, mint az ATBAS módszer. Miért jobb mégis? Azért, mert nem csak 3-mal lehet eltolni a karaktereket, hanem tetszőleges 1 és 25 közötti számmal. Így a megfejtőnek több lehetőséget kell végigpróbálnia.

A mai leckében egy olyan programot készítünk, amely a *vers.txt* fájl tartalmát kódolja Caesar kódolással.

## Előkészületek

Kezdj egy új projektet *kodolas* néven!

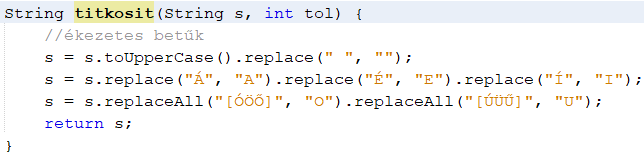
A Kodolas osztályban lesz a fájlkezelés, de magát a Caesar kódolást egy külön osztályban készítjük el. Ennek az az előnye, hogy a kódolást végző osztály másik programban is felhasználható.

Készíts egy új, Caesar nevú osztályt a kodolas csomagban! (jobb kattintás a csomag nevére, New, Java Class, Class name: Caesar, Finish)

## Titkosítás

Először a Caesar osztályt fogjuk elkészíteni. Egy titkosit nevű metódust hozunk létre benne, amely a fentiek szerint titkosítja a kapott szöveget. Paraméterei: s – a titkosítandó szöveg, tol – az eltolás mértéke.

A kódoláshoz először csupa nagybetűsre alakítjuk a szöveget, majd helyettesítjük az ékezetes betűket ékezet nélküli változatukkal. Korábban ezt egy elágazással oldottuk meg, de most egy másik módszert próbálunk ki. Nézzük a titkosit() metódus első változatát:



A korábban már tanult replace() metódust alkalmazzuk az ékezetes karakterek lecserélésére. Mivel több ilyen csere is kell, egymás után többször hívjuk meg a replace() metódust. Ezeket egymás után is fűzhetjük.

Van olyan eset, ahol többféle karaktert is ugyanazzal a karakterrel kell helyettesíteni. Ilyenkor a replaceAll() metódust alkalmazhatjuk. Ez úgy működik, hogy a []-ek közötti bármelyik karaktert lecseréli a megadottal.

Most ki kellene próbálni, hogy a metódus jól működik-e. Ezt megtehetnénk úgy, hogy a Kodolas osztály main metódusából meghívjuk valamilyen tesztadattal. Ezt az utasítást később törölnünk kellene.

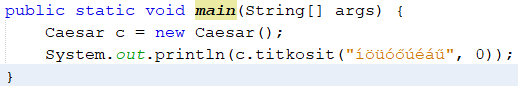
Ennél jobb megoldás az, hogy a Caesar osztályban is készítünk egy main metódust, és ebben helyezzük el a titkosit metódus teszteléséhez szükséges utasításokat.

A Caesar osztályt külön is futtathatjuk, és akkor a tesztet végezzük el a saját main metódusának végrehajtásával.

Ha viszont egy példányt hozunk létre a Caesar osztályból, akkor nem fogja végrehajtani a Caesar osztályban lévő main metódust.

A teszteléshez először létrehozunk egy példányt a Caesar osztályból, majd meghívjuk a titkosit metódusát egy teszt szöveggel, és kiíratjuk az eredményt.

Készíts egy main metódust a Caesar osztályba (psvm, majd Tab) az alábbi utasításokkal:

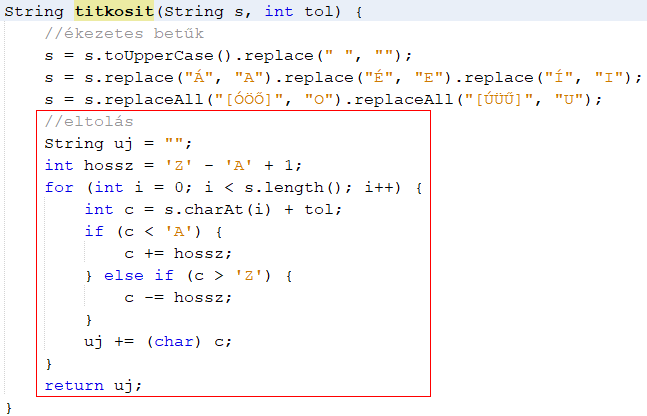


Figyeld meg, hogy a Caesar osztályban hoztunk létre egy új példányt saját magából! Ebben a példányban már nem hajtja végre a main metódust, és a titkosit metódusnak nem kell statikusnak lennie!

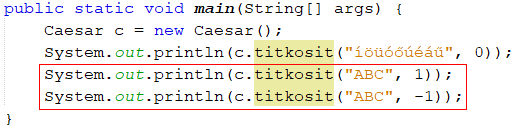
A titkosit metódus következő változatában már az eltolást is elvégezzük. Ennél kihasználjuk azt, hogy a Java szükség esetén automatikusan átalakítja a char típusú értéket int-té, és ez vissza is alakítható a (char) típuskonverzióval.

Egy uj nevű stringben állítjuk össze a kódolt szöveget. Karakterenként végigmegyünk az eredeti szövegen, és minden karakter kódjához hozzáadjuk az eltolást. Utána szükség esetén eltoljuk még az ABC hosszával is.

Egészítsd ki a titkosit metódust így:



Ezután egészítsd ki a main metódust is az eltolás tesztelésével:



Próbáld ki!

Ha jól működik, akkor a Caesar osztályt már használhatjuk a programban.

## Fájlkezelés

Az előző, robotos programban szükség volt arra, hogy a szöveges fájl sorait beolvassuk egy tömbbe, mert az egyes sorokkal többször is végeztünk műveleteket. A tömbben történő tárolást megkönnyítette, hogy a fájl elején meg volt adva a sorok száma. Így tudtuk a tömb méretét.

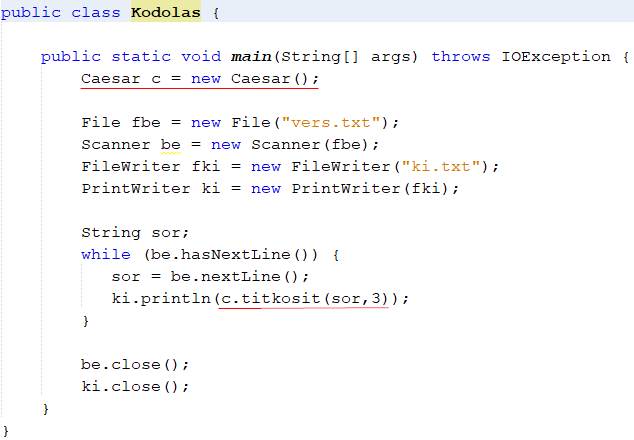
A gyakorlatban viszont sokszor nem tudjuk előre, hogy hány sor lesz. Ilyenkor három lehetőségünk van:

* Az egyik, hogy akkora tömböt hozunk létre, ami biztosan elég lesz, és egy változóban számláljuk az elemeket. Ennek a megoldásnak az a hátránya, hogy sok memóriát foglal.
* A második lehetőség, hogy a beolvasott adatokat egy listában tároljuk. Ennek nem kell előre megadni a méretét.
* A harmadik lehetőség, hogy amikor egy sort beolvasunk, azt rögtön feldolgozzuk, és utána nem tároljuk tovább. Így sokkal kevesebb memóriára van szükség, de ez csak akkor alkalmazható, ha a feldolgozást egyszerre el lehet végezni.

Mi most a harmadik megoldást fogjuk alkalmazni.

A programban először létrehozunk egy példányt a Caesar osztályból. Utána beolvassuk a sorokat a *vers.txt* fájlból, majd kiírjuk a *ki.txt* fájlba. Mindkét fájl UTF-8 kódolású.

Lássuk a programot:



A program elején megnyitjuk a bemeneti és a kimeneti fájlt is, a végén pedig bezárjuk őket. Ez különösen a kimeneti fájlnál fontos, mert e nélkül lehet, hogy üres lesz!

A középső részben beolvassuk a sorokat, átalakítjuk őket, majd kiírjuk a kimeneti fájlba.

Másold a *vers.txt* fájlt a projekt mappájába, majd futtasd a programot! Ellenőrizd a *ki.txt* fájlt!

Változtasd az eltolást más értékre, és futtasd le így is!

## Kérdés

Hogyan lehetne visszaállítani az eredeti verset? Próbáld ki!

## Önálló feladat

Készítsd el a hibakezelést try – catch szerkezettel!