0

05

0

Java SE 8 Documentation (http://docs.oracle.com/javase/8/docs/)

## Gyakorlás

1. Készítsük a `utils.IntList` osztály implementációját! Ezt az osztályt `int` értékek tömbösített, dinamikus tárolására kell tudnunk használni és nagyban hasonlítani fog a `java.util.ArrayList` (http://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html) osztályhoz. Legyenek ilyen konstruktorai:
   - nincs semmilyen paramétere (ilyenkor a tömb még nem tartalmaz semmit),
   - meg lehet adni egy `int` értékeket tartalmazó tömbbel, hogy mit tároljon kezdetben,

   valamint legyenek a következő műveletei:
   - `add()` : egy új elem hozzáadása a tömb végére,
   - `add()` : elem beszúrása az adott indexre (túlterhelt változat),
   - `concat()` : egy másik `IntList` tartalmának hozzáfűzése az aktuálishoz,
   - `get()` : az első elem lekérdezése
   - `get()` : adott indexű elem lekérdezése (túlterhelt változat),
   - `set()` : adott indexű elem beállítása, amennyiben van olyan indexű elem,
   - `remove()` : az első elem törlése,
   - `remove()` : adott indexű elem törlése (túlterhelt változat),

- `indexOf()` : elem indexének megkeresése, ha nem található, akkor `-1` ,
- `size()` : a jelenleg tárolt elemek száma,
- `clear()` : az összes elem törlése,
- `toArray()` : az osztályban tárolt értékeket egyetlen tömbként történő visszaadása,
- `show()` : szöveggé alakítás.



és legyen egy olyan, `read()` nevű osztályszintű metódusa, amellyel szövegből tudjuk beolvasni a tárolni kívánt elemeket. Az értékeket ebben a beolvasni kívánt szöveges reprezentációban egyszerűen szóközökkel választjuk el. Ellenőrizzük azt is, hogy a szövegben valóban számok szerepelnek! Ha ez nem teljesül és nem tudjuk beolvasni az összes elemet, akkor adjuk vissza `null` referenciát!

Ügyeljünk arra, hogy az osztály semmilyen más egyéb eleme ne legyen a külvilág számára elérhető!

2. Valósítsunk meg `arrays.DoubleMatrix` osztályt, amely `double` típussal valós számokat tárol egy mátrixban! Az osztálynak legyenek a következő műveletei:
   - konstruktor, amely megadja, hogy az egyes dimenziókban mekkora a mátrix kiterjedése,
   - `set()` , amellyel be tudunk állítani értéket az adott pozícióban,
   - `get()` , amellyel le tudjuk kérdezni az adott pozíción található értéket,

   Érvénytelen indexek (vagyis amikor olyan értéket adunk meg, amelyek már nem a mátrix egy elemét címzik) esetén ne változtassuk meg a mátrixot! A `get()` esetén pedig `Double` (csomagoló) objektumokat adjunk vissza, így a `null` referenciával tudjuk jelezni, ha érvénytelen pozícióról nem kérdezhető le elem.
3. Általánosítsuk az előbbi `arrays.DoubleMatrix` osztályt `arrays.generic.DoubleMArray` néven, amely tetszőleges dimenziójú tömböt képes ábrázolni egyetlen egydimenziós tömb segítségével! Az interfésze legyen lényegében az előzővel megegyező, csupán egészítsünk ki egy `getDimensions()` metódussal, amely visszaadja, hogy mekkorák a tömb egyes dimenziói!
4. Módosítsuk az előbbi `arrays.DoubleMatrix` osztályunkat úgy, hogy előbbi, általánosított `arrays.generic.DoubleMArray` osztályunk segítségével valósuljon meg! Természetesen kifelé ennek semmi különbséget nem szabad jelentenie.
5. Írjunk egy `IntTree` osztályt, amely egy egész számokat rendezetten tároló bináris fát ábrázol! A bináris fa értékek olyan láncolata, ahol minden értéknek nulla, egy vagy két rákövetkezője lehet, és csak nulla vagy egy megelőzője. A láncolást `IntTree` objektumokra vonatkozó referenciákkal oldjuk meg, ezek fognak mutatni az adott fabeli csomópont bal- és jobb részfáira. Az csomópontokban ezenkívül még egy `int` értéket is eltárolunk. Ha nincs megelőző, akkor a fa gyökeréről beszélünk.

   Legyenek a következő műveleteink:
   - `insert()` , amely megkap egy `int` értéket és beilleszti a fába, annak értékétől függően. Ha a beszúrandó elem kisebb, a fa gyökerében található elemnél, akkor illesszük be a bal részfába (rekurzió). Ha nincs még bal részfánk, akkor hozzuk létre azzal az elemmel a gyökerében! Ugyanezt végezzük

el a jobb részfára, ha a beszúrandó elem nagyobb, vagy egyenlő, mint a gyökérben levő!
    - `contains()`, amely eldönti, hogy a paramétereként megadott elem megtalálható-e a fában! Ezt egy logikai értékkel adja vissza: igaz, ha igen, hamis, ha nem! (Ez is rekurzív metódus lesz.)
    - egy olyan konstruktor, amely megkapja `int` értékek egy tömbjét és azokat beszúrja a fába.
    - egy olyan konstruktor, amely egy elemből létrehoz egy olyan fát, amelyben csak egyetlen gyökérelem van (és nincsenek rákövetkezői).
    - `toArray()`, amely egy `int` értékeket tartalmazó tömbben visszaadja azokat az értékeket, amelyeket a fában tárolunk! Ha szeretnénk kihasználni, hogy a beszúrást rendezetten végeztük, akkor érdemes ezt a tömböt úgy felépíteni, hogy először a bal részfa elemeit vesszük, aztán a gyökérben levő elemet, majd a jobb részfa elemeit. Az így összeállított tömb is rendezett lesz, ezt nevezik inorder bejárásnak. (Természetesen ez a metódus is rekurzív.)
    - `show()`, amely megadja egy fa szöveges alakját. Itt most elegendő, ha csak a tárolt elemeket jelenítjük meg egy felsorolásban.
    - `equalsTo()`, amely eldönti az objektumról és a paraméteréről, hogy a kettő megegyezik-e. Két bináris fát akkor tekintsünk egyenlőnek, ha ugyanazokat az elemeket tartalmazzák!
6. File-olvasás `BufferedReader` segítségével
    - Olvassuk be egy parancssori paraméterben megadott fájl tartalmát és írjuk ki a konzolra. A feladatot a `static void printFile(String filePath)` metódus végezze.

    Segítség:
    - A `new File(filePath)` létrehoz egy fájl objektumot, ami egy (nem feltétlenül létező) fájlt (vagy egy könyvtárat) reprezentál.
    - A `new FileReader(file)` egy fájl objektum tartalmát tudja kiolvasni.
    - A `new BufferedReader(reader)` segítségével hatékonyabban (pl. soronként) tudunk egy `Reader` -en keresztül olvasni.
    - A `readLine()` függvény kiolvas egy sort a fájlból, ha nincs több sor, akkor `null` -al tér vissza.
    - A `FileNotFoundException` -t és az IOException (http://docs.oracle.com/javase/8/docs/api/java/io/IOException.html)-t kötelező lekezelni: vagy elkapni vagy továbbdobni (az `IOException` az általánosabb). Ahhoz, hogy a nem létező fájl hibáját kezeljük a `try { } catch(FileNotFoundException e) { }` szerkezetet használjuk. Az első blokkba kerülnek a normális végrehajtás utasításai, a másikba a hibakezelő kód.
    - Példa a hiba továbbdobása: `static void printFile(String filePath) throws IOException { ... }`
    - `void close()`: lezárás
    - Figyeljünk arra, hogy az adott fájl létezik-e. Ha nem, akkor ezt közöljük a felhasználóval.
7. A felhasználótól konzolon keresztül kérjünk be egy szöveget és írjuk ki, ha valamelyik sor teljesen megegyezik vele. Ezt egy `static boolean findLine(String lineToFind, String filePath)` függvénnyel valósítsuk meg. Ez `true` -t adjon vissza, ha megtalálta a szöveget.
8. File-olvasás `Scanner` segítségével Valósítsd meg az 6. feladatot, de most `Scanner` segítségével!

Segítség:

- `Scanner` létrehozása fájlhoz: `new Scanner(file)`
- `boolean hasNext()` : van-e még olvasandó elem (változatok: `boolean hasNextInt()` , `boolean hasNextDouble()` , `boolean hasNextLine()` , stb.)
- `String next()` : következő token beolvasása (változatok: `int nextInt()` , `double nextDouble()` , `String nextLine()` , stb.)
- `useDelimiter(String pattern)` : tokenek közötti elválasztójel beállítása
- `void close()` : lezárás

9. Tegyük fel, hogy a fájl minden sora vesszővel elválasztott számokat tartalmaz. `Scanner` segítségével olvasd a fájlt, majd minden sorban add össze a benne található számokat. Az eredményeket tárold el egy `ArrayList` -ben. Feltehetjük, hogy a fájl formátuma helyes.



Szorgalmi:

1. Készítsd el a 7. feladatnak egy olyan változatát, amely akkor is megtalálja a keresett szöveget, ha az nem egy egész sort alkot a fájlban, csak valahol elhelyezkedik egy sorban. (keress megfelelő metódust a `String` osztályban)
2. Készítsd el a 9. feladat egy olyan változatát, ahol a fájl hibás adatokat is tartalmazhat. A program ezeket a hibás adatokat (a nem számokat) egyszerűen hagyja figyelmen kívül.
3. Írjuk egy programot, ami egy adott mappán belül keres egy olyan fájlt, amelyben megtalálható egy adott szövegrészlet (Find in files).
4. Hozzunk létre egy `Finder` osztályt a `findinfiles` csomagban.
5. Olvassuk be egy parancssori paraméterben megadott fájl tartalmát és írjuk ki a konzolba. Ez egy `static void printFile(String filePath)` függvényben legyen megvalósitva.
6. Figyeljünk arra, hogy az adott fájl létezik-e. Ha nem, akkor ezt közöljük a felhasználóval.
7. A felhasználótól konzolon keresztül kérjünk be egy szöveget és írjuk ki, ha valamelyik sor teljesen megegyezik vele. Ezt egy `static boolean findText(String lineToFind, String filePath)` függvénnyel valósítsuk meg. Ez adjon true-t vissza, ha megtalálta a szöveget.
8. Azt is figyeljük, ha a keresett szöveg nem egy egész sort alkot a fájlban, csak valahol elhelyezkedik egy sorban.
9. Szorgalmi: Legyen arra lehetőség, hogy nagybetű-érzéketlen legyen a keresés.
10. Legyen arra lehetőség, hogy egész szavakra keressünk.

## Linkek

Kapcsolódó forráskódok (http://oktnb127.inf.elte.hu/javagyak/05/tree/master)