JAVA gyorsjegyzet

gyakori eljárások első ZH-hoz és beadandóhoz  
frissítve (leginkább a ZH tanulságai alapján): 2015.11.07.

https://www.facebook.com/groups/ELTE.IK.TKK/permalink/1066521386705380/

Programozási nyelvek: JAVA

Programozási technológa 1.

Szabó Norbert

YECVA4

ELTE IK 2015/ősz

## Kósza gondolatok

* Amikor megadsz egy String-et, két idézőjel közé rakd és ne aposztrófok közé, mert az mást jelent! Ezzel az aprósággal könnyen megszívhatod, pl ekkor:  
  f = in.split(":");
* A lenti példákban \*-os importokat használtam, de ezek veszélyesek egy kicsit. Pl. én  
  import java.util.\*;  
  -ot írtam, holott csak egy osztályra lett volna szükségem (Scanner). Ez kényelmesebb, de gondot okoz, amikor egy feladat megoldásához olyan saját osztályt írsz, ami amúgy már létezik a standard libraryben. Ilyen pl a Calendar. Ekkor a fordító panaszkodni fog, hogy példányosításnál nem tudja melyikre gondolsz pontosan. A megoldás, hogy vagy konkrétabban megmondod neki mit akarsz, pl:  
  calendar.Calendar c = new calendar.Calendar();  
  ...vagy nem \*-os importot használsz:  
  import java.util.Scanner;
* Scannernél nem mindegy, hogy next() vagy nextLine(), ezzel is könnyen mellényúlhatsz. Pl ilyen inputot vársz:  
  2015-11-02: Java határidő  
  2015-12-18: Analízis vizsga  
  A feldolgozást valahogy így akarod:  
    
  String[] f;  
  Scanner sc = new Scanner(System.in);  
  String in;  
  while (!(in = sc.nextLine()).equals("x")){  
   f = in.split(":");  
   [...feldolgozod...]  
  }  
    
  Ha sc.nextLine() helyett sc.next()-et írsz, furcsaságokat fogsz tapasztalni. Úgy fog viselkedni a program, mintha először ezt írnád:  
  2015-11-02: Java  
  majd ezt:  
  határidő  
  két különböző körben. A next() az (be nem állított delimiter esetén) csak space-ig megy, a nextLine() sortörésig.
* For ciklus végigiterálás (~foreach) előnye a gyors futás és rövid kód. Működik tömbbel, LinkedList-tel és ArrayList-tel is.  
    
  String[] arr = new String[3];  
  arr[0] = "elso";  
  arr[1] = "masodik";  
  arr[2] = "harmadik";  
  for (String s : arr){  
   System.out.println(s);  
  }
* Ne felejtsd el, hogy ha for ciklussal mész végig pl. egy listán és közben törölsz belőle elemet, akkor a lista mérete időközben csökkenni fog; ez anomáliákat okozhat.
* Létrehozásnál a LinkedList és ArrayList a két relációs jel között osztályt vár, nem rakhatsz bele primitív típust (pl. int). Vagyis ha egy integer LinkedListet akarsz létrehozni, akkor:  
  LinkedList<Integer> a = new LinkedList<Integer>();
* **Tudd használni a dokumentációt és figyelj a hibaüzenetekre!**  
  A ZH egyik feladata az volt, hogy egy függvény adjon vissza List formátumban adatokat. Oké, List-tel nem foglalkoztál még - gondolod magadban -, semmi gond: ránézel a dokumentációra. Rögtön látod a metódusok alapján, hogy ugyanazokat tudja, mint a LinkedList és ArrayList.  
    
  Nagyszerű, máris implementálod a kódot, éééés… Fordítás error.  
  Hibaüzenetet: a List az egy absztrakt osztály.  
    
  De akkor miért List-et kér vissza a feladat?!  
    
  Ekkor leesik, hogy valószínűleg az ArrayList és a LinkedList is valamilyen List, vagyis a List-ből származtatták őket. Tehát egyszerűen kiválasztod mondjuk a LinkedList-et, és átírod a kódban néhány másodperc alatt a List-et LinkedList-re. Ennyi volt a megoldás.  
    
  Egy másik példát a jó dokumentációforgatásra lejjebb találsz: **Objektum létrehozása static függvénnyel (+ hibakezelés)**.

## String eljárások

* String.length()  
  String hossza.
* String.substring(int begin, [int end])  
  String egy részét adja vissza, az első és utolsó indexet lehet megadni. Ha nem adsz utolsó indexet, akkor az a String végét jelenti.
* String.equals(String other)  
  Igaz/hamis: ugyanaz a két string?  
  Mivel a String nem primitív típus, hanem objektum, így az “==” nem működik, mert az referenciát hasonlít össze objektumok esetén. Vagyis az equals-t kell használnod.
* String.trim()  
  Visszaadja a String-et úgy, hogy leszedi az elejéről és végéről az üres karaktereket.
* String.startsWith(String prefix)  
  Igaz/hamis: ezzel kezdődik a String?
* String.endsWith(String suffix)  
  Igaz/hamis: ezzel ér véget a String?
* String.split(String pattern)  
  Feldarabolja a String-et a minta szerint és visszaadja tömb formájában az elemeket.  
    
  String input = "Budapest-Szolnok-Mezőtúr-Gyoma-Mezőberény";  
  String[] cities = input.split("-");  
    
  A feltételt regexp szerint adjuk meg.

## Integer eljárások

* Integer.parseInt(String a)  
    
  String-et konvertál integerré:  
    
  String text = "145";  
  int a = Integer.parseInt(text);

## Tömb eljárások

Ha lehet, akkor inkább LinkedList-et vagy ArrayList-et használj, mert a tömb nem bővíthető. Jobban jársz. Sajnos azonban előfordul, hogy direkt tömböt kérnek, mint a legutóbbi ZH-n.

* Tömb deklarálása  
  String[] a = new String[2];  
    
  Először megmondod a programnak, hogy milyen típusú adatokról van szó (String). A nyitó-csukó szögletes zárójellel tudatod a programmal, hogy ez egy tömb lesz. Majd megadod a változó nevét, amivel hivatkozni fogsz rá.  
    
  Amikor pedig valójában létrehozod a tömb objektumot (=) (mert ugye a tömb nem primitív típus, tehát new), már előre meg kell adnod, hogy a tömb hány elemet fog tartalmazni, ugyanis nem bővíthető.
* Hivatkozás a tömb egy elemére  
  System.out.println(a[0]);
* A tömb egy elemének értékadása  
  a[0] = "Elso";
* Deklarálás és értékadás egyszerre  
  String[] a = new String[]{"Elso","Masodik"};
* Array.length  
  Visszaadja a tömb méretét. Vedd észre, hogy (a Stringgel ellentétben) ez nem függvény, hanem konstans (mert úgysem bővíthető a tömb), tehát a végére nem kell a nyitó-csukó zárójel.

## Arrays eljárások

http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html  
  
import java.util.\*;

Az Arrays osztály - akármilyen meglepő - tömbökkel foglalkozik. Nincs konstruktora; ez lényegében egy olyan függvénygyűjtemény, amiket bár te is tudsz implementálni, egyszerűbb és gyorsabb ha ezeket használod ZH-n. Ebből most egyet emelek ki, de ha elakadsz egy tömbös teendővel, nem árthat belenézni a dokumentációba.

* Arrays.copyOf(forras, forras.length)  
  Első paraméter maga a forrás tömb, a második paraméter pedig a cél tömb hossza. Ez általában pont ugyanakkora, mint a forrás tömb.  
    
  A tömb ugye objektum, tehát az alábbi esetben ha átírod a forrás tömb egy elemét, akkor “átíródik” a cél tömb eleme is, mert lényegében a két tömb pontosan ugyanaz, hiszen a két különböző változó ugyanarra az objektumra mutat valójában (ezt hívják aliasingnak):  
    
  String[] forras = new String[]{"Elso","Masodik"};  
  String[] cel = forras;  
    
  Viszont ha szeretnél a forrás tömbről létrehozni egy másolatot, vagyis újra beleírni az egészet a memóriába, majd az új tömb referenciáját odaadni a cél változónak, akkor lehet szükséged a Arrays.copyOf-ra:  
    
  String[] forras = new String[]{"Elso","Masodik"};  
  String[] cel = Arrays.copyOf(forras,forras.length);  
    
  Az egészet persze egy ciklussal is meg tudod oldani ~10 másodperc alatt, ebben inkább az most a fontos, hogy jegyezd meg: objektumok esetén valójában referenciát másolsz a sima egyenlőségjellel.  
    
  És ha már itt tartunk: ha egy konstruktornak/függvénynek paraméterként objektumot adsz át, szintén a referenciát adod oda - tehát ekkor sem készül másolat az átadott objektumról.

## LinkedList eljárások

## http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html

import java.util.\*;

* Létrehozásánál meg kell adnod, hogy milyen típusú adatot akarsz benne tárolni úgy, hogy két relációs jel közé beírod az osztály nevét.  
    
  LinkedList<String> a = new LinkedList<String>();  
    
  A végén van egy nyitó-csukó zárójel, ne felejtsd el. Ez azért van, mert a LinkedList is objektum, aminek a konstruktorához lehetne paramétert írni, de erre nincs most szükségünk. Ha nem írsz semmit, üres LinkedList-et fogsz létrehozni.
* LinkedList.add([int i], T element)  
  Hozzáfűz a listához egy elemet. Az első paraméter nem kötelező; megadja, hogy hányadik elem helyére akarsz befűzni. Ha csak az elemet adod meg, akkor a lista végére rakja. (T itt azt jelenti, hogy bármilyen típus lehet. Pontosabban az lehet, amit létrehozáskor megadtál. Az előző pontban egy Stringekből álló LinkedListet hoztunk létre, tehát T element helyére valami String kerülhet.)
* LinkedList.get(int i)  
  Visszaadja az i. elemet.
* LinkedList.set(int i, T element)  
  Az i. elemet tudod átállítani.
* LinkedList.remove(int i)  
  Törli az i. elemet. Vigyázz: ha nem a legvégéről törölsz, a további elemek egyel hátracsúsznak: így ennyivel rövidebb lesz a listád és nem is marad “üres hely”. Ez ciklusban okozhat problémát.
* LinkedList.size()  
  Visszaadja, hogy hány elemből áll a listád.
* LinkedList.clear()  
  Törli az összes elemet, a végeredmény egy üres lista.

## ArrayList eljárások

## http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html

import java.util.\*;

Az ArrayList máshogy működik belül, mint a LinkedList, viszont mázlid van, mert kívülről pontosan ugyanannak tűnik - tehát ugyanazok az eljárások. Hogy mikor melyik az optimálisabb az nem témája e jegyzetnek; ZH-n azt használd amelyik szimpatikusabb. Előfordulhat, hogy a feladat konkrétan kéri valamelyiket, erre figyelj oda.

## Scanner eljárások http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html import java.util.\*;

* Konstruktorban meg tudod adni a forrást, amivel dolgozni akarsz. Ez a mi esetünkben a System.in lesz (ha beolvasásra akarod használni), vagy egy String (ha szöveg szétdarabolására).  
    
  Scanner sc = new Scanner(System.in);
* Scanner.useDelimiter("-")  
  A next() -szerű parancsok ezt a szövegrészletet figyelik eztán. A delimitert bármikor módosíthatod; erre találsz is példát lejjebb a **String feldarabolása, Scanner módszer** résznél. Az alap delimiter a (bármennyi) whitespace. De vigyázz: itt nem bármilyen String-et, hanem regexp kifejezést kell megadnod.  
    
  **Ha nem tudsz regexp-ül, életmentő tipp:**  
  Ha nem nyeli be a Java azt a karaktert amit szeretnél használni (azaz futás közben furcsaságokat csinál, nem jó helyen tördel), akkor próbáld ki, hogy a karakter elé raksz backslash-t (visszaper: \), és ha még így sem jó, rakj elé még egyet. Például ha pontot akarsz delimiternek használni az azért problémás, mert a pont önmagában egy regexp kifejezés. Viszont a backslash un. escape karakter, amit épp az ilyen esetekre találtak ki. Vagyis ha pontok szerint akarod tördelni az inputot, ez a helyes parancs:  
  Scanner.useDelimiter("\\.");
* Scanner.hasNext()  
  Igaz/hamis: jelenlegi delimiterrel értelmezhető-e a következő elem? Ha fel akarod dolgozni az összes adatot, akkor nagyon hálás ezt használni egy while ciklussal (pl szavakkal akarod feltölteni a bevásárlólista LinkedList-edet, de nem tudod még, hogy egy sorban hány adat lesz).  
    
  while (input.hasNext()){  
   list.add(input.next());  
  }
* Scanner.next()  
  Visszaadja a következő elemet String-ben és lépteti a Scannert.
* Scanner.nextInt()  
  Olyan mint a next(), csak egyből Integerré konvertálva kapod vissza az adatokat. Mással is működik, pl nextFloat(), nextDouble(), stb.
* Scanner.hasNextInt()  
  Olyan mint a next(), csak Integert vizsgál. Mással is működik, pl hasNextFloat(), hasNextDouble(), stb.
* Scanner.nextLine()  
  Visszaadja az egész sort String-ben és lépteti a Scannert. Ha beolvasáshoz használtad a Scanner-t (System.in), akkor “el is fogyott belőle” az adat nyilván.

## Beolvasás billentyűzetről

http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

Scanner objektummal a legegyszerűbb, ez működik NetBeans-ben és console-ban is.

import java.util.\*;

[...]

Scanner sc = new Scanner(System.in);

String input = sc.nextLine();

## A legegyszerűbb console menü Scanner-rel

http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

Valószínűleg mindenkinek teljesen evidens (van is ilyen példa a jegyzet elején), de azért itt van.

import java.util.\*

[...]

Scanner console = new Scanner(System.in);

String in;

while (!(in = console.nextLine()).equals("x")){

if (in.startsWith("add ")){

[...feldolgozod az add parancsot...]

} else if (in.startsWith("delete ")){

[...feldolgozod a delete parancsot...]

} else {

System.out.println("Nincs ilyen parancs.");

}

}

A ciklusfeltételben rögtön értéket is adtunk, így a leggyorsabb. De lehetne így is:

Scanner console = new Scanner(System.in);

String in = console.nextLine();

while (!in.equals("x")){

[...]

in = console.nextLine();

}

## Beolvasás fájlból

http://docs.oracle.com/javase/7/docs/api/java/io/File.html

http://docs.oracle.com/javase/7/docs/api/java/io/FileReader.html

http://docs.oracle.com/javase/7/docs/api/java/io/BufferedReader.html

BufferedReader(FileReader(File))

import java.io.\*

[...]

File f = new File("valami.txt");

try{

BufferedReader reader = new BufferedReader(new FileReader(f));

String in;

while ((in = reader.readLine()) != null){

System.out.println(in);

}

reader.close();

}

catch(FileNotFoundException e){

System.out.println("Nincs meg a file.");

}

catch(IOException e){

System.out.println("Valami hiba történt");

}

Kezelned kell az exception-t fileok esetén, hogy ne legyen fordítási hiba. Az IOException a legáltalánosabb, ez a minimum. Ellenben az előtte lévő (FileNotFoundException-t) kihagyhatod a kódból, ha akarod. A catch-eknek természetesen sorban kell történnie, a kisebbektől az átfogóbbak felé építkezve. IOException után hiába is raknál FileNotFoundException-t, az IOException már tartalmazza azt is, tehát az a hiba már “el lett kapva”. A végén illik lezárni a file-t.

## Kiírás fájlba

http://docs.oracle.com/javase/7/docs/api/java/io/File.html

http://docs.oracle.com/javase/7/docs/api/java/io/FileWriter.html

http://docs.oracle.com/javase/7/docs/api/java/io/BufferedWriter.html

BufferedWriter(FileWriter(File))

import java.io.\*

[...]

File f = new File("valami.txt");

try{

BufferedWriter writer = new BufferedWriter(new FileWriter(f,true));

writer.write("Teszt");

writer.flush();

writer.close();

}

catch(IOException ex){

System.out.println("Valami hiba történt.");

}

Hibakezelés ugyanúgy, mint beolvasásnál. Amikor mindent kiírtál, flush() és close(). (Ha nincs flush, lehet nem fog bekerülni minden adat a fileba; ha nincs close, lehet hibás lesz a file.)

A FileWriter példányosításánál a második attribútum nem kötelező, alapértéke false. Az a kérdés, hogy fájlkiegészítésről van-e szó.

false = nem, töröld a file tartalmát és kezdd el írni

true = igen, menj a file végére és folytasd onnan

## String feldarabolása, Scanner módszer

http://docs.oracle.com/javase/8/docs/api/java/lang/String.html

http://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html

http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

Eltérő delimiterek esetén érdemes.

import java.util.\*;

[...]

String input = "2015.10.12-JAVA ZH";

Scanner sc = new Scanner(input);

sc.useDelimiter("\\.");

int year = sc.nextInt();

int month = sc.nextInt();

sc.useDelimiter("-");

String d = sc.next(); //d = ".12"

d = d.substring(1); //d = "12"

int day = Integer.parseInt(d); // day = 12

String task = sc.next();

System.out.println(task);

Vegyünk észre, hogy delimiternek regexp kifejezést kell használnunk, azért kell a két backslash a pont elé.

## Objektum létrehozása static függvénnyel (+ hibakezelés)

Létezik egy ilyen osztályunk:

public class Konyv{

private String szerzo;

private String cim;

private int oldalszam;

public Konyv(String szerzo, String cim, int oldalszam){

this.szerzo = szerzo;

this.cim = cim;

this.oldalszam = oldalszam;

}

[...getterek helye...]  
  
 [IDE AKAROK ÍRNI EGY ÚJ FÜGGVÉNYT]

}

A feladat, hogy írjunk egy make függvényt, aminek segítségével létrehozható egy String alapján is az objektum a következőképpen:  
“Szerző-Cím-oldalszám”. Ha hibás a String, pl: “Szerző-Cím”, vagy “Szerző-Cím-alma”, akkor térjen vissza null-al.  
  
A megoldás valahogy így néz ki:  
   
 public static Konyv make(String stuff){

String[] a = stuff.split("-");

if (a.length != 3) return null;

int t = 0;

try{

t = Integer.parseInt(a[2]);

}

catch (NumberFormatException ex){

return null;

}

return new Konyv(a[0],a[1],t);

}  
  
Értelmezzük:

* A függvény static, mert nincs szükség objektumra (példányosításra) ahhoz, hogy a működőképes legyen.
* Visszaad egy Konyv -et (vagy null-t).
* Ha nem három adattagból áll, akkor biztosan hibás, egyből return null; ezzel megszakad a függvény további futása.
* Ha három adattagból áll, akkor az első két String az a szerző és a cím, a harmadik pedig egy integer. Viszont ezt a harmadikat még muszáj átalakítani integerré, **de nem biztos, hogy sikerülni is fog**.  
  Bizonytalanság esetén az általános teendő, hogy megnyitod a dokumentációt és megnézed mit ír -ebben az esetben - az Integer.parseInt() függvény. Egyből látod, hogy NumberFormatException -t dob, ha nem jön össze neki az átalakítás. Ezt tehát rögtön fel is tudod használni: NumberFormatException esetén azonnal térjen vissza null-al. Ugyanúgy kell csinálnod ezt az egészet, mint a fájlkezelésnél láttad: try és catch.  
    
  Most azért fontos a hiba kezelése, mert rossz input esetén futás során (épp a NumberFormatException miatt) kilépne a program, a feladat pedig pont azt kérte, hogy ilyen ne történhessen meg.
* ...és ha eddig mindenen átment, akkor a szokásos konstruktorral létrehozod az objektumot, majd visszaadod azt.
* A főprogramban tehát könyvet készíteni kétféleképpen tudsz már:  
  Konyv k = new Konyv("Szerzo", "Cim", 259);  
  Konyv f = Konyv.make("Szerzo-Cim-325");

## 

## A ZH gyakorlati részéhez és a beadandók megoldásához szerintem ezek az eljárások tökéletesen elegendők, sok sikert!

Innentől gyakorlati ZH és beadandó szempontjából nem fontos dolgok jönnek.

## String összefűzés StringBuilderrel

http://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html

Az is jól működik, ha simán “+” jelekkel konkatenálod a Stringeket, de memóriakihasználás és sebesség szempontjából sok összefűzés esetén sokkal-sokkal optimálisabb, ha StringBuildert használsz. (Nem kell hozzá importálni semmilyen speciális osztályt.)

StringBuilder sb = new StringBuilder();

sb.append("Valamit");

sb.append(" akartam");

sb.append(" ");

sb.append("írni.");

System.out.println(sb.toString());

Mivel a String egy objektum, a + jeles konkatenálás valójában nem történik meg, csak szimulálja azt a Java. Amikor létrehozod a String-ed, és beállítasz neki egy értéket, az onnantól úgy is marad. Amikor "konkatenálsz", valójában egy teljesen új String jön létre, de a referenciát a program átállítja arra - majd ennek örömére (mivel már nem mutat rá referencia) a szemétgyűjtő eltakarítja az előzőt. A StringBuilder tehát azért optimálisabb, mert

* a végeredmény String minden elemét egyszer hozod létre,
* a String-eket nem kell folyton összemásolgatni,
* mindössze egyszer kell összefűzni a toString() során (de ekkor nem is lesz belőle új objektum)
* és még a szemétgyűjtőt sem dolgoztatod feleslegesen.