Bevezetés

A Java jó cucc. Én már évek óta használom. Szeretem, mert nagyon szabályos. Szinte minden egyszerű szabályokat követ benne. Ezek a szabályok időnként félelmetesnek tűnhetnek, de ez a könyv segít megoldani a nehézségeket. Szóval,ha használni szeretnéd a Javát, és a szokásos szakembereknek szóló könyvektől eltérő olvasmányt szeretnél, akkor már el is kezdheted olvasni a *Tantusz könyvek* Java kötetét.

Hogyan használd ezt a könyvet?

Bárcsak azt mondhatnám, hogy „Nyisd ki a könyvet bármelyik oldalon, és kezdj hozzá Java-kódot írni. Csak töltsd ki az üresen hagyott részeket, és ne is lapozz vissza." Bizonyos értelemben ez igaz is. Nem ronthatsz el semmit azzal, hogy Java-kódot írsz, tehát szabadon kísérletezgethetsz.

De engedd meg, hogy őszinte legyek. Nagyon nehéz úgy programot írni, ha nem látod át az összefüggéseket. Ez az összes programozási nyelvre igaz, nemcsak a Javára. Ha csak találomra elkezdesz kódot gépelni, anélkül hogy tudnád, miről is van szó, akkor a kód nem úgy fog működni, ahogy szeretnéd, és el fogsz akadni.

De szerencsédre ebben a könyvben a Java könnyen emészthető falatokra van bontva. Minden falat (többé-kevésbé) egy fejezet. Ahhoz lapozhatsz, amelyikhez csak szeretnél. Az 5. a 10. fejezetre, vagy ahová csak akarsz. Még akár azt is megteheted, hogy a közepén kezded el olvasni az egyik fejezetet. Megpróbáltam a példákat elég érdekessé tenni, anélkül hogy az egyik fejezet megértése egy másiktól függne. Amikor egy fejezetben egy másik fejezetben szereplő fontos fogalom szerepel, azt az eligazodást segítő megjegyzés jelzi.

Általánosságban a tanácsom a következő:

- Ha valamit már tudsz, azért ne legyél rest utánaolvasni.
- Ha kíváncsi vagy, bátran lapozz előre. Ha úgy érzed belezavarodtál, egy korábbi fejezetbe bepillantva megtalálhatod a választ a kérdésedre.

Magyarázatok a könyv használatához

Szinte minden szakkönyv egy kis betűformázási magyarázattal kezdődik. Itt, a *Tantusz könyvek* Java kötetében sincs ez másképp. A következőkben a könyvben használt betűtípusok rövid magyarázatát olvashatod:

- Az új szakkifejezések (vagy angol megfelelőik) *dőlt* betűvel vannak szedve.
- Ha a rendes szövegben olyasmi szerepel, amit be kell gépelned, ezek a karakterek félkövérrel lesznek szedve. Például: „Gépeld be a szövegmezőbe azt, hogy **ÚjProjekt**".
- Találkozhatsz majd `ezzel a betűtípussal` is. Ezt a Java-kódokhoz, fájlnevekhez, weboldalak címeihez (URL-ekhez), a képernyőn megjelenő üzenetekhez és más hasonló dolgokhoz használjuk. Emellett ha az, amit be kell gépelned, nagyon hosszú, külön sorban (vagy sorokban) fog szerepelni ezzel a betűtípussal.
- Néhány dolgot meg kell majd változtatnod, amikor a saját számítógéped billentyűzetén pötyögöd be őket. Például, ha arra kérlek, hogy gépeld be a következőt:

  ```
  public class Valamilyennév
  ```

  akkor ez azt jelenti, hogy a **public class** szavak után azt a nevet kell begépelned, amelyet te magad találtál ki. Azokat a szavakat, amelyeket a sajátjaiddal kell helyettesítened, *`eltérő betűtípussal`* vannak szedve.

Amit átugorhatsz, ha…

Keresd meg az első olyan fejezetet vagy alfejezetet, amelyben olyan dolgok vannak, melyeket még nem ismersz, és ott kezd az olvasást. Persze előfordulhat, hogy ugyanannyira utálsz ilyen döntéseket hozni, mint én. Ha ez a helyzet, a következő útmutatást követheted:

- Ha már tudod, hogy miféle szerzet a Java, és azt is tudod, hogyan szeretnéd használni, nyugodtan hagyd ki az 1. fejezetet, és lapozz rögtön a másodikhoz. Hidd el, nem fogok érte megsértődni.
- Ha már tudod, hogyan kell elindítani egy Java-programot, és nem foglalkoztat, mi történik a színfalak mögött, miközben fut, akkor hagyd ki a 2. fejezetet, és kezdd a harmadikkal.

- Ha rendszeresen írsz programokat, de nem a C vagy a C++ nyelvet használod, akkor kezdd a 2. vagy a 3. fejezettel. Amikor elérsz az 5. és a 6. fejezethez, bizonyára könnyen boldogulsz majd velük. Amikor eléred a 7. fejezetet, ideje lesz jobban belemerülnöd a dologba.
- Ha rendszeresen írsz programokat C (nem C++) nyelven, akkor kezdd az olvasást a 2. a 3. és a 4. fejezettel, de az 5. és a 6. fejezetből csak szemezgess.
- Ha rendszeresen írsz programokat C++ nyelven, akkor pillants rá a 2. és a 3. fejezetre, fusd át a 4. az 5. és a 6. fejezetet, és a 7. fejezetet kezdd el komolyabban tanulmányozni. (A Javában egy kicsit más az osztályok és az objektumok kezelése, mint a C++-ban.)
- Ha rendszeresen írsz programokat Javában, akkor gyere át hozzám, és segíts nekem megírni a *Java* könyv következő kiadását.

Nyugodtan hagyd ki a szürke szövegdobozban szereplő vagy az Elmélet ikonnál jelölt részeket, ha szeretnéd. Ami azt illeti, bármelyik részt teljesen nyugodtan átugorhatod.

Néhány feltételezés

Ebben a könyvben néhány dolgot feltételezek rólad, az Olvasóról. Ha ezek közül a feltételezések közül van olyan, amelyik nem igaz rád, az még aligha jelent gondot. Ha egyik feltételezés sem igaz, akkor… hát, akkor is érdemes megvenned a könyvet.

- **Feltételezem, hogy hozzáférsz egy számítógéphez.** Van számodra egy jó hírem: a könyvben szereplő kódokat szinte minden számítógépen futtathatod. Azok a számítógépek, amelyeken nem, már ősrégiek. Ez azt jelenti, hogy tíz évnél régebbiek (plusz-mínusz pár év).
- **Feltételezem, hogy eligazodsz a számítógéped gyakran használt menüiben és párbeszédablakaiban.** Nem kell a Windows, UNIX vagy Macintosh hozzáértő felhasználójának lenned, de tudnod kell, hogyan kell elindítani egy programot, megtalálni vagy berakni egy fájlt a megfelelő könyvtárba… meg hasonló dolgokat csinálni. Amikor a könyvben lévő anyagot gyakorlod, leginkább kódokat fogsz begépelni a billentyűzeteden, nem pedig az egérrel fogsz mutogatni és kattintgatni.

  Azon ritka alkalmakkor, amikor át kell húznod, vagy ki kell vágnod és be kell illesztened valamit valahova, vagy esetleg új hardvert kell konfigurálnod, gondosan végigvezetlek a lépéseken. A számítógéped viszont több milliárd különböző módon lehet konfigurálva, és előfordulhat, hogy az instrukciók a te különleges helyzetedhez

nem teljesen megfelelőek. Amikor elérsz egy ilyen platformspecifikus feladathoz, próbáld meg követni a könyv lépéseit. Ha a leírt lépések nálad nem alkalmazhatók, akkor nézz utána a dolognak egy olyan könyvben, amely a te rendszereddel foglalkozik.

» **Feltételezem, hogy tudsz logikusan gondolkozni.** A Java-programozáshoz csak ennyi kell – logikus gondolkozás. Ha tudsz logikusan gondolkodni, sínen vagy. Ha úgy gondolod, hogy nem tudsz, olvass csak tovább. Lehet, hogy kellemes csalódásban lesz részed.

» **Csak kevés feltételezésem van a korábbi számítógépes programozói ismereteidről (vagy az ismereteid teljes hiányáról).** Amikor a könyvet írtam, megpróbáltam a lehetetlent. Megpróbáltam a könyvet a tapasztalt programozók számára érdekessé, ugyanakkor azok számára is követhetővé tenni, akiknek programozói tapasztalatuk alig van vagy egyáltalán nincs. Ez azt jelenti, hogy nem feltételezek semmiféle programozói hátteret a részedről. Ha még sosem írtál ciklust, vagy indexeltél tömböt, akkor sincs semmi baj.

  Másrészről, ha már csináltál ilyeneket (talán Visual Basic-ben, Pythonban vagy C++-ban), akkor érdekes különbségeket fogsz felfedezni a Javában. A Java kifejlesztői fogták az objektumorientált programozás legjobb ötleteit, egyszerűsítették, átdolgozták, majd újraszervezték őket, és így egy sima és hatékony gondolkodásmódot hoztak létre a problémák megoldásához. A Javában több új, gondolatébresztő funkciót is találhatsz. Ahogy megismered ezeket a funkciókat, rá fogsz jönni, hogy sok közülük természetesnek hat. Így vagy úgy, de jól fogod érezni magad, miközben a Javát használod.

A könyv felépítése

A könyv bekezdései alfejezeteket alkotnak, az alfejezetek fejezetekké állnak össze, amelyek végül öt nagy részt alkotnak. (Amikor könyvet írsz, nagyon jól át fogod látni a felépítését. Ha hónapok óta folyamatosan írsz, akkor egyszer csak elkezdesz alfejezetekben és fejezetekben álmodni.) A következőkben a könyv részeiről olvashatsz.

I. rész: Ismerkedjünk meg a Javával

Ebben a részben egy teljes körű összefoglalót olvashatsz a Javáról. Találhatsz itt néhány fejezetet a „Mi a Java?” témakörben, és egy igazi „motort berúgó” fejezetet is – a harmadikat. A 3. fejezetben megismerheted a főbb technikai fogalmakat, és egy egyszerű programot is górcső alá veszünk.

Definiáljunk egy osztályt (avagy mitől lesz valaki focista)

Ami engem illet, abban biztos vagyok, hogy egy focistának van neve, és a meccsenként átlagosan rúgott góljainak száma. A 10.1. listában ezt a megérzésemet öntöm Java-kód formájába.

10.1. lista. A Játékos osztály

```
import java.text.DecimalFormat;

public class Játékos {
  private String név;
  private double gólÁtlag;

  public Játékos(String név, double gólÁtlag) {
    this.név = név;
    this.gólÁtlag = gólÁtlag;
  }

  public String getNév() {
    return név;
  }

  public double getGólÁtlag() {
    return gólÁtlag;
  }

  public String getGólÁtlagString() {
    DecimalFormat decimálisForámtum = new DecimalFormat();
    decimálisForámtum.setMaximumIntegerDigits(1);
    decimálisForámtum.setMaximumFractionDigits(3);
    decimálisForámtum.setMinimumFractionDigits(3);
    return decimálisForámtum.format(gólÁtlag);
  }
}
```

Most pedig lássunk neki a 10.1. listában szereplő kód ízekre szedésének. Szerencsére a korábbi fejezetekben már sok olyan dologról szó esett, melyek megjelennek ebben a kódban. A kód azt határozza meg, hogy mitől lesz valami a `Játékos` osztály példánya. A kódban a következők szerepelnek:

- **A `név` és a `gólÁtlag` mező deklarációja:** A mezők deklarációjáról szóló esti mesét a 7. fejezetben olvashatod.

rendszert, és egy átlagos program lefutása 2-4 óráig tart.) A szokásos várakozási idő után Bright kísérlete a FORTRAN-program lefordítására egyetlen hibát ad eredményül – az egyik utasításból hiányzik egy vessző. Bright kijavítja a hibát, és a program kitűnően működik.

1962. július 22. Az első másik amerikai űrhajó, mely egy másik égitestet vett célba, a Mariner I megsemmisül, amikor a működésében hiba lép fel négy perccel a kilövés után. A hibát az okozta, hogy egy vonal (olyan, mint egy kötőjel) hiányzott a rakéta sebességének képletéből.

Nagyjából ugyanebben az időben fedezik fel, hogy a NASA keringési pályát számító szoftverében a hibás `DO 10 I=1.10` utasítás szerepel (a helyes `DO 10 I=1,10` helyett). Ez a modern jelölésben olyan, mintha a `(int i=1; i<=10; i++)` helyett azt írnánk, hogy `do10 i = 1.10`. Azzal, hogy a vessző helyére pont került, a ciklusból értékadó utasítás lett.

2000. január 1. A 2000-es év problémája feldúlja a modern világot.

A következő forrásokból emeltem át a történelmileg pontos tények, melyek a fentebbi jegyzetekben szerepelnek: a Computer Folklore hírcsoport (`https://groups.google.com/forum/#!forum/alt.folklore.computers`), a Free On-line Dictionary of Computing (vagyis az ingyenes informatikai online szótár) (`http://foldoc.org`), a *Computer* magazin (`www.computer.org/computer-magazine/`), és az IEEE más weblapjai (`www.computer.org`).

Kivételkezelés

Leltározol. Ez azt jelenti, hogy egymás után veszed számba a tételeket, a dobozokat, és ezeknek a dolgoknak a számát jegyzetlapokon, kis kézi kütyükön és a billentyűzet segítségével számítógépes űrlapokon jegyzed fel. A projekt egyik konkrét része, hogy felírd a Nagy Poros Soha Az Életben Ki Nem Nyitott Dobozok feliratú polcon heverő dobozok számát. Mivel nem akarod megtörni a cég évtizedes hagyományát, úgy döntesz, hogy egyik dobozt sem nyitod ki. Önkényesen mindhez a 700 Ft értéket rendeled.

A 13.1. listában láthatod a szoftvert, amellyel a leltárnak ezt a kis részét kezelheted. A programnak azonban van egy hibája, amelyet a 13.1. ábrán láthatsz. Amikor a felhasználó egész értéket ad meg, minden a legnagyobb rendben megy. Ha viszont valami mást ír be (például a 3,5 értéket), a program teljesen összeomlik. Biztosan lehet ezzel kezdeni valamit. A számítógépek buták, de nem annyira, hogy rögtön magukba roskadjanak, ha egy felhasználó nem megfelelő értéket ad meg.

13.1. lista. Számoljunk dobozokat

```
import static java.lang.System.out;
import java.util.Scanner;
import java.text.NumberFormat;

public class LeltárA {

   public static void main(String args[]) {
      final double dobozÁr = 700;
      Scanner billentyűzet = new Scanner(System.in);
      NumberFormat pénznem = NumberFormat.getCurrencyInstance();

      out.print("Mennyi dobozunk van? ");
      String dobozokSzámaBe = billentyűzet.next();
      int dobozokSzáma = Integer.parseInt(dobozokSzámaBe);

      out.print("Az érték ");
      out.println(pénznem.format(dobozokSzáma * dobozÁr));
      billentyűzet.close();
   }
}
```

```
Mennyi dobozunk van? 3
Az érték 2 100 Ft

Mennyi dobozunk van? 3,5
Exception in thread "main" java.lang.NumberFormatException: For input string: "3,5"
        at java.lang.NumberFormatException.forInputString(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at LeltárA.main(LeltárA.java:14)

Mennyi dobozunk van? három
Exception in thread "main" java.lang.NumberFormatException: For input string: "három"
        at java.lang.NumberFormatException.forInputString(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at LeltárA.main(LeltárA.java:14)
```

13.1. ábra. A 13.1. listában szereplő kód három különálló lefutása

Egy programhiba megoldásának kulcsa abban rejlik, hogy megvizsgáld a program összeomlásakor megjelenő hibaüzenetet. Az üzenetben, melyet a leltározó program megjelenít, az áll, hogy `java.lang.NumberFormatException`. Ez azt jelenti, hogy a *NumberFormatException* nevű osztály az API `java.lang` csomagjába tartozik. Az `Integer.parseInt` meghívása hívta elő valahogy ezt a `NumberFormatException` osztályt a rejtekéből.

Nem, kedves olvasó. Ez nincs így!

A publikus mezőket könnyű használni, de még ennél is könnyebb velük visszaélni. Úgy alakíthatod ki a legjobban a kódodat, ha az egyes mezőkhöz a lehető legjobban korlátozod a hozzáférést. Ha egy mezőnek nem szükséges mindenképpen publikusnak lennie, próbáld meg privátá tenni. Ha más osztályoknak le kell kérdezniük vagy be kell állítaniuk a mező értékeit, ehhez adj meg publikus beállító és lekérdező metódusokat. Ez pedig szépen le is vezet a következő bekezdéshez…

GYAKORLÁS

Az alfejezet egyik példájában nem használhatod a `valamiObjektum.énMezőm` hivatkozást, mert a `ValamiOsztály` osztályban az `énMezőm` mező privátként van deklarálva. Ezt javítsd úgy, hogy az osztályt beállító és lekérdező metódusokkal egészíted ki, és ennek megfelelően módosítod a `valamiObjektum.énMezőm` hivatkozást.

Tegyük keretbe a rajzot

Ahhoz, hogy ez a láthatósági módosítós dolog világos legyen a számodra, látnod kell egy-két példát. A fejezet első példájában szinte minden publikus. A publikus láthatóságnál nem tévedhetsz nagyot abban, hogy ki mit használhat.

Ennek az első példának a kódja több részből áll. A 14.1. listában található első részben egy `KépKeret` nevű osztályt láthatsz. A `KépKeret` egy `Rajz` objektumot tartalmaz. Ha minden darab a helyére kerül, a 14.1. lista kódjának lefutásakor egy ahhoz hasonló ablakot fogsz látni, amely a 14.4. ábrán is szerepel.

14.1. lista. Jelenítsünk meg egy keretet

```
import com.burdbrain.rajzok.Rajz;
import com.burdbrain.keretek.KépKeret;

class KeretMegjelenítő {

    public static void main(String args[]) {
        KépKeret képKeret = new KépKeret(new Rajz());

        képKeret.setSize(300, 100);
        képKeret.setVisible(true);
    }
}
```

A 14.1. listában szereplő kód egy új `KépKeret` példányt hoz létre. Talán sejted, hogy a `KépKeret` a Java `frame` osztályának alosztálya, és valóban, ez így is van. A 9. fejezetben viszont azt mondtuk, hogy a Java-keretek

14.4. ábra. Egy KépKeret

alapértelmezés szerint nem láthatók. Szóval a 14.1. listában ahhoz, hogy láthatóvá tedd a `KépKeret` példányt, meg kell hívnod a `setVisible` metódust.

Most figyeld meg, hogy a 14.1 lista két `import` deklarációval kezdődik. Az első lehetővé teszi, hogy rövidítsd a `com.burdbrain.rajzok` csomagba tartozó `Rajz` osztály nevét. A második segítségével pedig a `com.burdbrain.keretek` csomagba tartozó `KépKeret` osztály nevén rövidíthetsz.

KERESZT-HIVATKOZÁS

Az `import` deklarációk leírását a 4. fejezetben találod.

A benned rejlő nyomozó nyilván arra gondol, hogy „Biztosan több kódot is írt (olyan kódot, amelyet itt nem látok), és azt tette a *com.burdbrain.rajzok* és a *com.burdbrain.keretek* csomagba.” És valóban, igazad van. Ahhoz, hogy működésre bírjam a 14.1. listát, létrehozok egy `Rajz` nevű valamit, és az összes rajzomat a `com.burdbrain.rajzok` csomagban gyűjtöm össze. Szükség van továbbá egy `KépKeret` osztályra, és az összes ilyesfajta osztályt a `com.burdbrain.keretek` csomagba teszem.

Na de valójában mi is egy `Rajz`? Ha mindenképpen tudni szeretnéd, nézd meg a 14.2. listát.

14.2. lista. A Rajz osztály

```
package com.burdbrain.rajzok;

import java.awt.Graphics;

public class Rajz {
    public int x = 40, y = 40, szélesség = 40, magasság = 40;

    public void paint(Graphics g) {
        g.drawOval(x, y, szélesség, magasság);
    }
}
```

A `Rajz` osztály kódja nem valami terjedelmes – néhány `int` mező és egy `paint` (rajzol) metódus van benne. Ennyi az egész. Nos, amikor osztályokat hozok létre, próbálom rövidre fogni a kódjukat. Akárhogy is, itt van néhány megjegyzés a `Rajz` osztályommal kapcsolatban:

Gyerünk… nyomd meg a gombot

A korábbi fejezetekben olyan ablakokat hoztunk létre, amelyek nem sok mindent csináltak. Az ablakokban jellemzően volt némi információ, de nem voltak interaktív elemei. Most eljött az ideje, hogy változtassunk ezen. Ennek a fejezetnek az első példája egy olyan ablak lesz, melyben van egy gomb. Amikor a felhasználó megnyomja azt a fránya gombot, történik valami. A kódot a 16.1. listában, a 16.1. lista kódját meghívó `main` metódust pedig a 16.2. listában láthatod.

16.1. lista. Egy kitalálós játék

```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Random;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

class JátékKeret extends JFrame implements ActionListener {
   private static final long serialVersionUID = 1L;

   int véletlenSzám = new Random().nextInt(10) + 1;
   int tippekSzáma = 0;

   JTextField szövegMező = new JTextField(5);
   JButton gomb = new JButton("Tippelek");
   JLabel címke = new JLabel(tippekSzáma + " próbálkozás");

   public JátékKeret() {
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setLayout(new FlowLayout());
      add(szövegMező);
      add(gomb);
      add(címke);
      gomb.addActionListener(this);
      pack();
      setVisible(true);
   }

   @Override
   public void actionPerformed(ActionEvent e) {
      String szövegMezőSzövege = szövegMező.getText();
```

Előzzük meg a null értékű mutatók előfordulását

A könyv példái nem hajlamosak `NullPointerException` kivételt okozni, a valódi Java-programozásban azonban minden pillanatban találkozhatsz vele. A `NullPointerException` akkor következik be, amikor egy olyan metódust hívsz meg, melynek egy objektumot kellene visszaadnia, de az nem ad vissza semmit. Itt van egy primitív példa:

```
import static java.lang.System.out;
import java.io.File;

class FájlListázó {

    public static void main(String args[]) {
        File énFájlom = new File("/Users");
        String tartalom[] = énFájlom.list();

        for (String fájlNév : tartalom) {
            out.println(fájlNév);
        }
    }
}
```

Ez a program kilistázza a `Users` könyvtár összes fájlját.

Mi történik azonban akkor, ha a `Users` könyvtárat valami olyasmire módosítod, ami nem egy könyvtár nevét jelöli?

```
File énFájlom = new File("&*%$!");
```

Ekkor a `new File` hívás egy `null` értéket ad vissza (a `null` a Java egy speciális kifejezése, mely azt jelenti, hogy *semmi*), így az `énFájlom` változóban nincs semmi. A kód későbbi részében a `tartalom` változó nem hivatkozik semmire, és a kísérlet a `tartalom` összes értékének bejárására csúfos kudarcot fog vallani. Kapsz egy nagy `NullPointerException` kivételt, a programod pedig összeomlik.

Az ilyen szerencsétlenségek elkerülése érdekében ellenőrizd a Java API dokumentációját. Ha olyan metódust hívsz meg, amely `null` értéket adhat vissza, lásd el a programodat kivételkezelő kóddal.

A kivételkezelésről a 13. fejezetben tudhatsz meg többet. A 3. fejezetben, illetve a könyv weboldalán (`www.allmycode.com/JavaForDummies`) találhatsz néhány tanácsot az API-dokumentáció olvasásához.