A JAVA NYELV KIVÉTEL- ÉS ESEMÉNYKEZELÉSÉNEK BEMUTATÁSA AZ OKTATÁSBAN

DEMONSTRATING OF EXCEPTION HANDLING AND EVENT HANDLING IN EDUCATION BY JAVA

Seres Iván, Kaczur Sándor
Gábor Dénes Főiskola Informatikai Intézet

Összefoglaló

A hallgatók rendszerint nincsenek tisztában a kivétel- és eseménykezelés részleteivel (milyen objektumok keletkeznek, hol keletkeznek, hogyan jutnak el az őket később kezelő metódusokhoz vagy blokkokhoz). Cikkünkben kísérletet teszünk arra, hogy a kivétel- és eseménykezelés folyamatát szemléletessé, mintegy kézzelfoghatóvá tegyük – rövid, de célra törő forráskódok, a BlueJ oktatószoftver, valamint a JBuilder fejlesztőeszköz segítségével. Bemutatjuk, hogyan keletkeznek a kivételek, hogyan lehet szándékosan kiváltani, továbbadni, lekezelni azokat, hogyan értelmezhető élettartamuk.

Kulcsszavak

objektumorientált programozás, kivételkezelés, eseménykezelés, Java

Abstract

Usually students do not know details of exception handling and event handling. (What sort of objects can be rise? Where do objects rise? How do objects be thrown to the handling methods or block?) In this article we make an attempt at make clear the process of the exception handling and event handling. We use short source codes, tutorial program BlueJ, and development tool JBuilder. We demonstrate how exceptions can be risen, how to give rise to exceptions, how exceptions can be thrown, how exceptions can be handled, what the lifetime of exception is.

Keywords

object-oriented programing, exception handling, event handling, Java

1. Bevezetés

A korszerű objektumorientált nyelveken fejlesztett programok működése eseményorientált. A programot objektumok kommunikációja, üzenetküldése működteti. A program futása felhasználói (belső) vagy más programok, erőforrások (külső) beavatkozásaitól, azaz eseményektől függ.

Az események szintek alapján is csoportosíthatók: az alacsony szintűek a hardver felől érkező megszakítások, magas szintűek a felhasználói tevékenységek (pl.: kattintás, görgetés, adatok megváltozása, űrlapok megnyitása). Az esemény hatással van a program futására, bekövetkezése megváltoztathatja valamely objektum állapotát. Az eseményorientált programozás eseménybegyűjtő és szétosztó mechanizmuson alapszik. Ennek működése egy olyan elöltesztelő ciklusként értelmezhető, amelynek ciklusmagjában kiolvassuk az események várakozó sorába került eseményt, majd továbbítjuk az eseménykezelőnek. Egy-egy eseményhez hozzárendelhető egy-egy eseménykezelő metódus, amely reagál arra.

A kivétel lehet esemény, feltétel, vagy hibás állapot, amely bekövetkezése esetén a program normális működése megszakad. A kivétel egy hibaobjektum, amely a program futása során létrejön, feldolgozásra kerül, végül megszűnik. A kivételre a kivételkezelő reagál. A korszerű nyelvek esetén egy-egy eljárást, függvényt az általánosan kedvező esetre írunk meg, majd a metódus végén felsoroljuk azon hibalehetőségeket, amelyekre figyelni kell. Az így készített forráskód áttekinthetőbb, mintha közben vizsgálnánk az esetleges előforduló hibákat.

Ha kódolás közben elkövethetünk szintaktikai hibát, akkor a forráskód nem fordítható le, mivel nem felel meg a programozási nyelv szabályrendszerének. Ha a tervezéskor vagy a kódoláskor szemantikai hibát vétünk, akkor a forráskód lefordítható, a program elindul, csupán nem a várt, kívánt rész- vagy végeredményt adja. Futási hiba esetén kezelni kell a keletkező hibát, és fel kell szabadítani a hiba bekövetkezését megelőzően lefoglalt erőforrásokat. A kivételkezelés további védelmet is nyújthat, védhet az előre nem várt, a felhasználótól, erőforrás rendelkezésre állásától függően keletkező problémáktól. Segítségével elérhető, hogy a felhasználónak a szoftver a beállított nyelven üzenje meg a problémát, adjon beavatkozási lehetőséget, a program futása folytatódhat is.

A Java nyelvben a kivételek a `Throwable` osztály leszármazottai. A `RuntimeException` közvetlen leszármazottból futási hiba keletkezésekor (pl.: tömb túlindexelése), az `IOException`-ből pedig beviteli/kiviteli problémák esetén (pl.: nem létező fájl megnyitása) jön létre objektum. A kivételek szándékosan is kiválthatók a `throw` utasítással, amely után csak a `java.lang.Throwable` osztály leszármazottja írható.

Az esemény is objektum, a kivételhez hasonlóan. A forrásobjektumon keletkezik; másképp a forrásobjektum kapja meg feldolgozásra, majd továbbítja az eseményfigyelőnek. Eseményfigyelő egyszerre több is lehet. Az alacsonyszintű eseményeket az operációs rendszer egy eseményekhez rendelt várakozó sorban helyezi el (pl.: billentyűzet, egér, ablak események), ezek forrása vizuális komponens. A várakozó sor meghatározza sorrendiségüket. Esemény csak olyan látható komponensen keletkezhet, amely része az alkalmazás komponenshierarchiájának. Minden mást magasszintű eseménynek tekintünk (pl.: akció-, igazítási esemény).

A Java nyelvben az eseményobjektumok az `EventObject` osztályból származnak. A Swing komponensek eseményeit a `javax.swing.event` és a `java.awt.event` csomag definiálja.

2. Példák

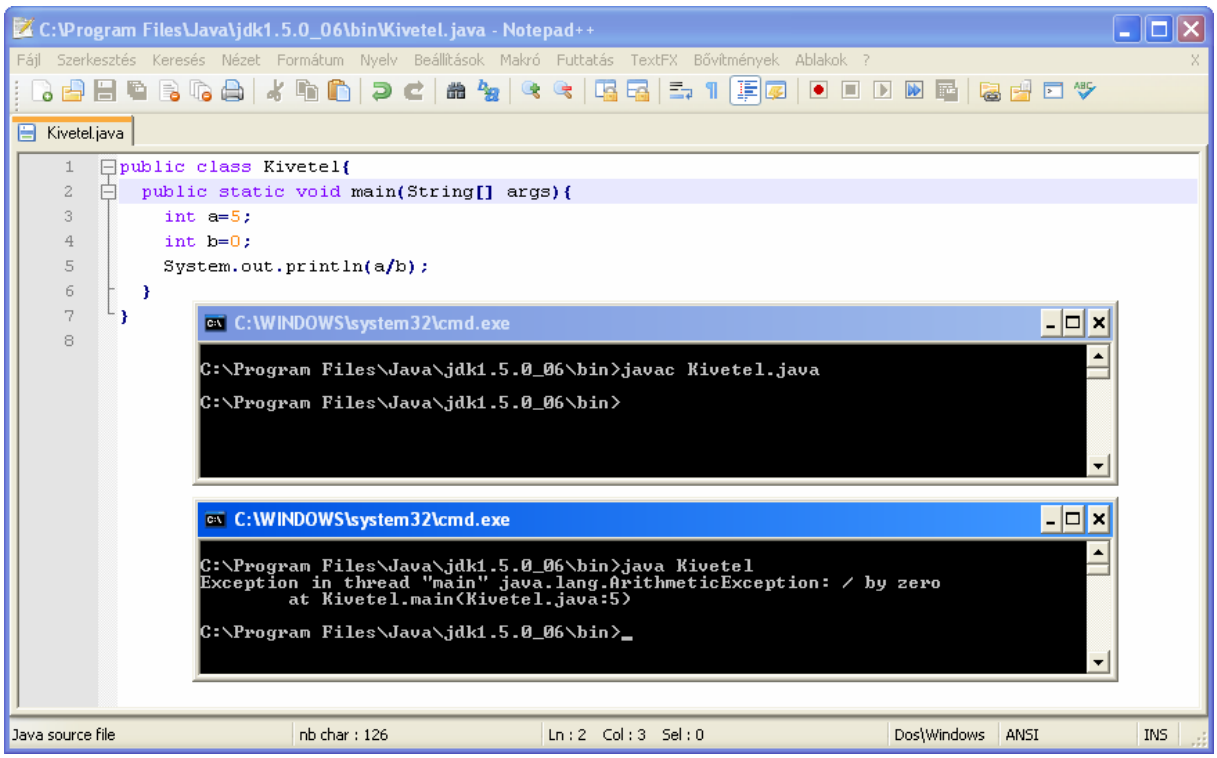
Az alábbiakban a Notepad++, a BlueJ eszközt, valamint a JBuilder fejlesztőeszközt felhasználva egyszerű mintapéldákkal mutatjuk be a kivétel- és eseménykezelés lehetőségeit. Az első két eszköz szabadon letölthető: http://notepad-plus.sourceforge.net/hu/site.htm, http://www.bluej.org, a harmadik telepítőanyagát pedig főiskolánk hallgatói a félévenként kiosztásra kerülő oktatócsomagban megkapják. A példák megoldásához az 1.4.2_01-es, illetve az 1.6.0_05-ös Java virtuális gépet használtuk.

2.1. Nullával való osztás

A probléma tipikus. A kód szintaktikailag helyes, szemantikailag hibás, fordítása hiba nélkül megtörténik, futáskor `ArithmeticException` osztályú kivétel keletkezik.

```
public class Kivetel {
  public static void main(String[] args) {
    int a=5;
    int b=0;
    System.out.println(a/b);
  }
}
```

A legegyszerűbb kipróbálási lehetőség a Notepad++ eszközben adódik: az F5-re megjelenő Futtatás ablakába a `cmd` parancsot beírva parancssorból fordítható a forrásfájl és futtatható a bináris bájtkód.



1. ábra – A nullával való osztás mintaprogramja a Notepad++ eszközben

2.2. Előjeles végtelen eredmény

Ha két lebegőpontos formában tárolt valós számot osztunk egymással, és az osztó 0, akkor az `Infinity` (előjeles végtelen) eredményt kapjuk. Kivétel nem keletkezik.

```
public class Kivetel {
  public static void main(String[] args) {
    double a=1.0, b=0.0;
    System.out.println(a/b);
  }
}
```

2.3. Nem szám eredmény

Amikor negatív számot adunk a négyzetgyökvonó függvénynek paraméterként, akkor a `NaN` (Not A Number) üzenetet adja. Kivétel nem keletkezik.

```
public class Kivetel {
  public static void main(String[] args) {
    int x=-1;
    System.out.println(Math.sqrt(x));
  }
}
```

2.4. Tömb túlindexelése

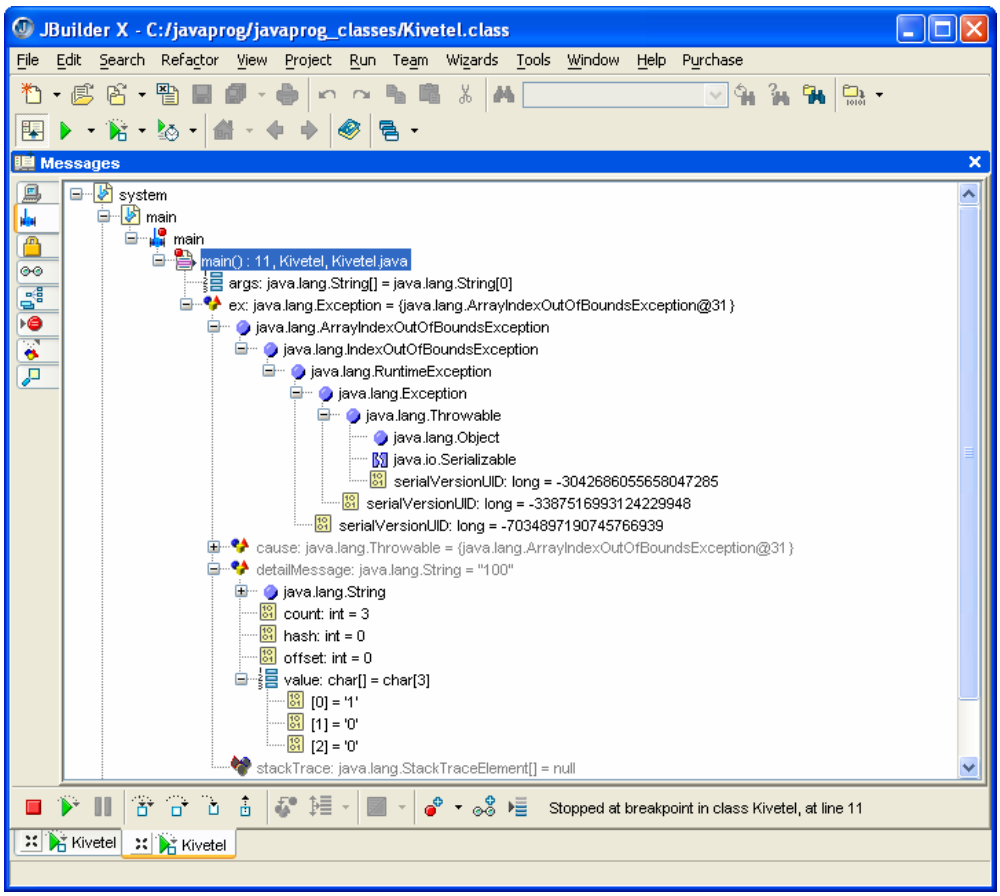
Az `n`(>0) elemű tömb érvényes tömbindexe `0` és `n-1` közötti egész szám. Ha `n=100`, és a `for` ciklus feltételében a ciklusváltozó megegyezhet a 100-zal, akkor a Java virtuális gép a kezdőértékként kapott nullákat követően az `ArrayIndexOutOfBoundsException` osztályú kivételt váltja ki.

```
public class Kivetel {
  static int[] tomb=new int[100];
  public static void main(String[] args) {
    for(int i=0; i<=100; i++)
      System.out.println(tomb[i]);
  }
}
```

Ha a tömbelemek felveszik az index értékét, és a keletkezett kivételobjektumnak lekérdezzük az osztályát és annak paraméterét (üzenetét), akkor az utolsó érvényes 99-es értéket követően a: `Kivétel: class java.lang.ArrayIndexOutOfBoundsException, Üzenet: 100` sort kapjuk a virtuális géptől.

```
public class Kivetel {
  static int[] tomb=new int[100];
  public static void main(String[] args) {
    try {
      for (int i=0; i<=100; i++)
        System.out.println(tomb[i]=i);
    }
    catch (Exception ex) {
      System.out.println("Kivétel: "+ex.getClass()+
        ", Üzenet: "+ex.getMessage());
    }
  }
}
```

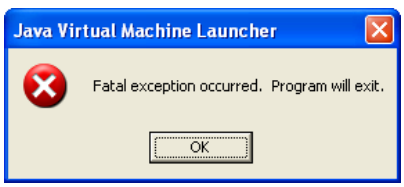
A `getClass()` függvény a kivételobjektum osztályát, a `getMessage()` függvény pedig annak konstruktorban külön beállítható `detailMessage` szöveges leírását adja vissza.



2. ábra – A tömb túlindexelés mintaprogramjának nyomkövetése a JBuilder eszközben

A JBuilder fejlesztőeszköz képes a dinamikusan létrejövő `ex` kivételobjektumot azonosítani, leszármazottságát bemutatni, a kivétel helyét a forráskódban megjelölni, üzenetének felépítését megmutatni. Erre az eszköz Debug projekt menüpontja használható.

Kivétel keletkezése esetén a JBuilder az alábbi hibaüzenet adja:



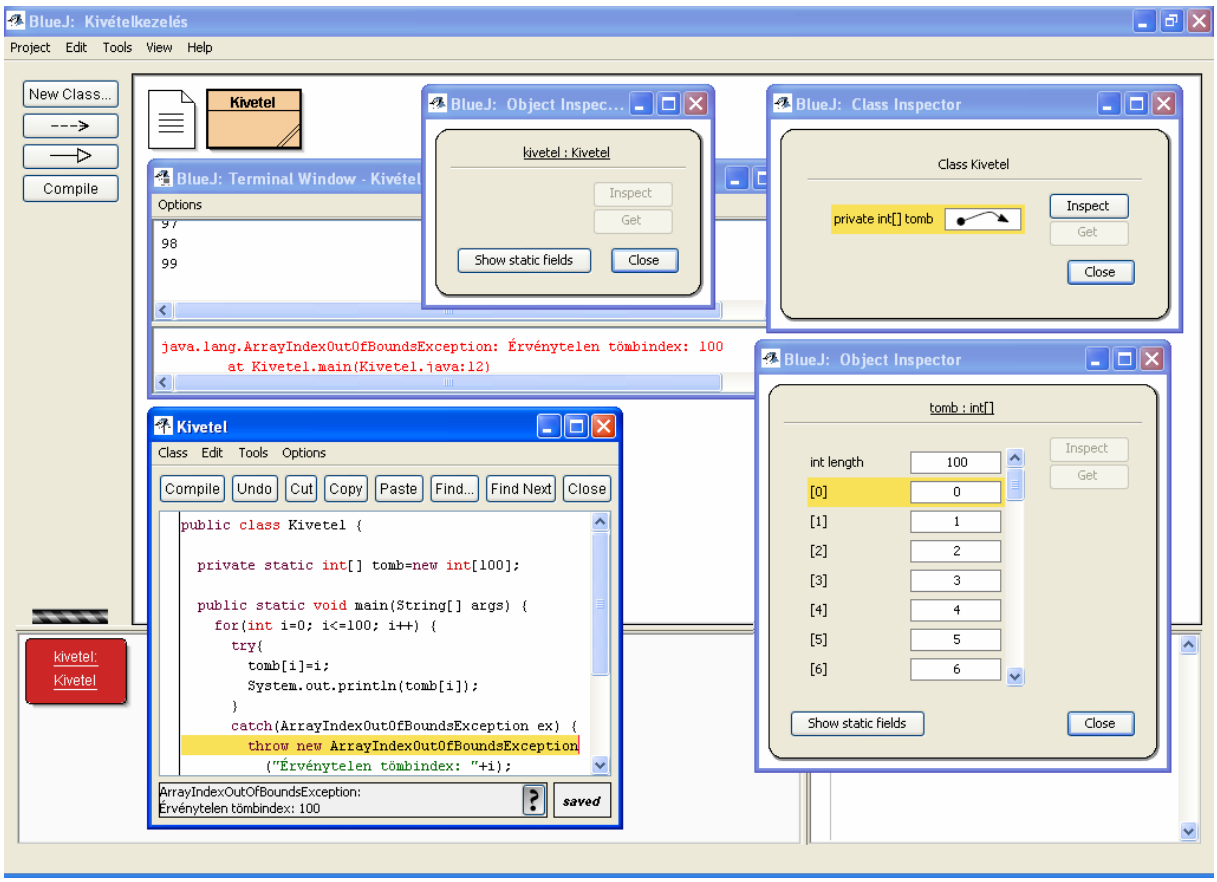
3. ábra – A JBuilder eszköz hibaüzenete kivétel esetén

Ha `for` ciklus belépési feltételét kicseréljük `i<=tomb.length`-re, akkor is az előző kivételobjektum keletkezik. A tömb `-1`-edik elemére hivatkozva is az előző üzenetet kapjuk. A tömb „másfeledik” elemére történő hivatkozás viszont már fordításkor hibát ad. Ha érvényes indexű tömbelemben nem egész valós számot szeretnénk tárolni, akkor konstans értékadáskor fordítási hibát, konzolról beolvasva futási hibát kapunk.

```
public class Kivetel {
  static int[] tomb=new int[100];
  public static void main(String[] args) {
    for(int i=0; i<=100; i++) {
      try{
        System.out.println(tomb[i]=i);
      }
      catch(ArrayIndexOutOfBoundsException ex) {
        throw new ArrayIndexOutOfBoundsException
          ("Érvénytelen tömbindex: "+i);
      }
    }
  }
}
```

Ha a kivételt elkapjuk, paramétere jelzi a hiba okát: `Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Érvénytelen tömbindex: 100 at Kivetel.main (Kivetel.java:11)`. Ha több kivételt kell lekezelni, akkor több `catch` is alkalmazható. Ekkor a sorrend számít: haladjunk a speciális eset lekezelésétől az általános eset felé.

A BlueJ eszközben projekt tartalmazza a `Kivetel` osztályt, a forráskódja szerkeszthető. Ha lefordítjuk, és létrejön belőle az objektum, akkor a forráskódot futtatva, a kivétel keletkezésekor megtekinthetők az objektum statikus adatai. Látható, hogy a tömb létrejön, feltöltődik, elemszáma 100, megjelenik a paraméterezett kivételüzenet is.



4. ábra – A tömb túlindexelés mintaprogramjának nyomkövetése a BlueJ eszközben

A keletkezett kivételobjektum élettartama a `catch` blokkra korlátozódik. Ha megpróbálunk pl. a `getClass()` metódussal üzenni neki a `try-catch-finally` blokkon kívül, akkor a fordító az `<identifier> expected` hibaüzenettel leáll. Sajnos a BlueJ eszköz nem képes megjeleníteni a dinamikusan létrejövő objektumok adatait, tulajdonságait.

2.5. Háromszög megszerkeszthetősége

Három megegyező mértékegységgel megadott szakaszból mikor szerkeszthető háromszög? Amikor teljesül a háromszög-egyenlőtlenség. Feltételezzük, hogy az `a`, `b`, `c` változók mértékegysége megegyezik. Számoljuk ki a háromszög területét Héron-képlettel, ha lehetséges! Hagyományos ellenőrzése történhet az alábbiak szerint.

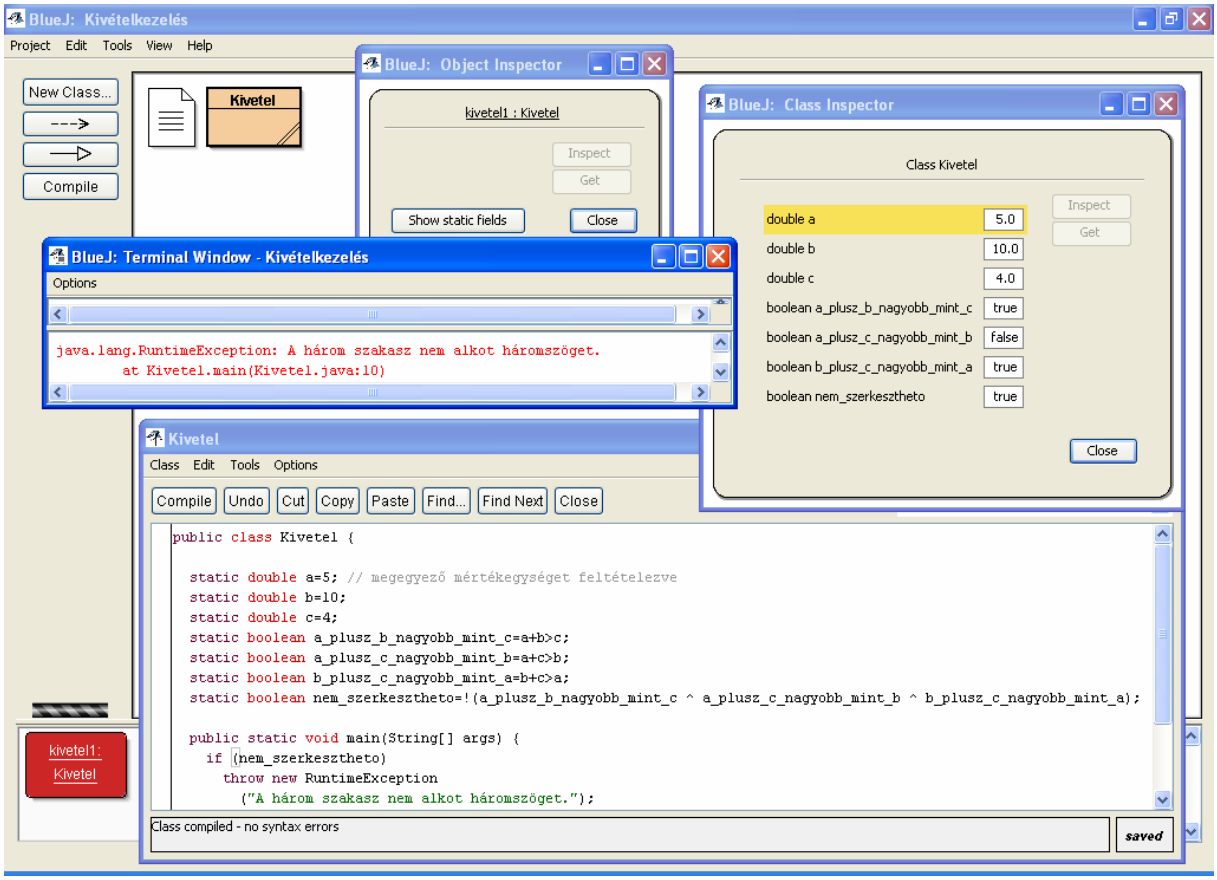
```
public class Kivetel {
  public static void main(String[] args) {
    double a=5, b=10, c=4;
    boolean haromszogE=(a+b>c)^(a+c>b)^(b+c>a);
    if (haromszogE) {
      System.out.println("A három szakasz háromszöget alkot.");
      double s=(double)(a+b+c)/2;
      double terulet=Math.sqrt(s*(s-a)*(s-b)*(s-c));
      System.out.println("A háromszög területe: "+terulet);
    }
    else
      System.out.println("A három szakasz nem alkot háromszöget.");
  }
}
```

Hozzunk létre saját kivételobjektumot, ha háromszög nem szerkeszthető: `Exception in thread "main" java.lang.RuntimeException: A három szakasz nem alkot háromszöget. at Kivetel.main(Kivetel.java:7)`

```
public class Kivetel {
  public static void main(String[] args) {
    double a=5, b=10, c=4;
    if (!((a+b>c)^(a+c>b)^(b+c>a)))
      throw new RuntimeException("A három szakasz nem alkot háromszöget.");
  }
}
```

Ha statikus osztályváltozóként hozzuk létre a valós oldalak hosszát, az oldalak vizsgálatának eredményét tartalmazó logikai, valamint a `nem_megszerkesztheto` logikai változókat, akkor a BlueJ eszköz meg tudja mutatni ezek értékét a kivétel keletkezésekor.

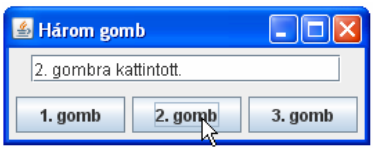
```
public class Kivetel {
  static double a=5, b=10, c=4;
  static boolean a_plusz_b_nagyobb_mint_c=a+b>c,
                 a_plusz_c_nagyobb_mint_b=a+c>b,
                 b_plusz_c_nagyobb_mint_a=b+c>a,
                 nem_szerkesztheto=!(a_plusz_b_nagyobb_mint_c ^
                                     a_plusz_c_nagyobb_mint_b ^
                                     b_plusz_c_nagyobb_mint_a);
  public static void main(String[] args) {
    if (nem_szerkesztheto)
      throw new RuntimeException("A három szakasz nem alkot háromszöget.");
  }
}
```



5. ábra – A háromszög megszerkeszthetőség mintaprogramjának nyomkövetése a BlueJ eszközben

2.6. Nyomógombok azonosítása

Az eseménykezelés alapja, hogy azonosítani tudjuk az esemény kiváltóját, forrását, vagyis azt, hol keletkezett, honnan származik az esemény. A mintapélda `szövegmező` komponensébe attól függően, hogy melyik nyomógombra kattintottunk, beíródik a nyomógomb feliratát is tartalmazó szöveg. A `figyelő` nevű figyelőobjektumot a láncra ciklus fűzi fel, és az esemény bekövetkezésekor az eseményt kiváltó objektumot nyomógombbá típuskényszerítve kapjuk meg annak feliratát, majd abból konkatenáljuk a megfelelő paraméterszöveget.



6. ábra – A Gombok program felhasználói felülete

A gombok lenyomásakor keletkezik az `ev` akcióesemény, amelynek forrása az aktuális nyomógomb. A keletkezett eseményt a `figyelő` objektum dolgozza fel. Az `addActionListener` metódussal rendeljük hozzá a forrásobjektumhoz; vagyis a `JButton` osztály deklarálja a metódust. Bármely gomb lenyomásakor meghívásra kerül az `actionPerformed` metódus, és paraméterként átveszi az akcióeseményt.

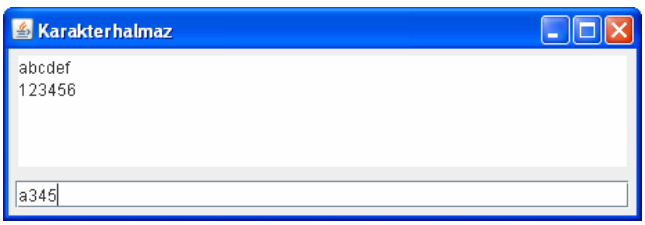
Így a lenyomás eseménye eljut a forrásobjektumból az eseményfigyelőhöz, amely lekezeli azt. Az `ActionListener` objektum itt dinamikusan jön létre, más elvet követve az osztály implementálhatná is az `ActionListener` interfészt.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Gombok {
  public static void main(String[] args) {
    JFrame fr=new JFrame("Három gomb");
    fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    fr.setBounds(100, 100, 200, 100);
    JPanel déliPanel=new JPanel();
    for (int i=1; i<=3; i++)
      déliPanel.add(new JButton(i+". gomb"));
    fr.add(déliPanel, BorderLayout.SOUTH);
    final JTextField szövegmező=new JTextField(20);
    szövegmező.setText("Még nem kattintott.");
    JPanel középsőPanel=new JPanel();
    középsőPanel.add(szövegmező);
    fr.add(középsőPanel, BorderLayout.CENTER);
    ActionListener figyelő=
      new ActionListener() {
        public void actionPerformed(ActionEvent ev) {
          szövegmező.setText(((JButton) ev.getSource()).getText()+
            "ra kattintott.");
        }
      };
    for (Component gomb: déliPanel.getComponents())
      if (gomb instanceof JButton)
        ((JButton) gomb).addActionListener(figyelő);
    fr.pack();
    fr.setVisible(true);
  }
}
```

2.7. Engedélyezett karakterek használata

A következő példát úgy alakítottuk ki, hogy az alsó szövegmezőbe csak olyan karaktereket lehessen beírni, amelyik megvan a felső szövegterületen. A módszer az, hogy a szövegmezőt billentyűzetesemény forrásának tekintjük; ezt a forrást megfigyeltetjük a `KeyListener` interfészt „röptében” implementáló osztály egy objektumával (erre mutat a figyelő hivatkozás). Ez az objektum a születése pillanatában a `szöveg` hivatkozáson át `setEditable (true)` üzenetet küld a szövegmezőnek. Ha a felhasználó lenyom egy billentyűt, és ahhoz a billentyűhöz nem karakter tartozik vagy olyan karakter tartozik, amelyik nincs rajta a szövegterületen (`halmaz`), akkor a szövegmezőt szerkeszthetetlenné tesszük (a `setEditable (false)` üzenettel), majd a billentyű felengedésekor – a `keyReleased` metódusban – visszaállítjuk szerkeszthetővé (a `szöveg` hivatkozáson át küldött `setEditable(true)` üzenettel).

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Karakterhalmaz {
  public static void main(String[] args) {
    JFrame fr=new JFrame("Karakterhalmaz");
    fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    fr.setBounds(100, 100, 200, 100);
    JPanel északiPanel=new JPanel();
    JPanel déliPanel=new JPanel();
    fr.add(északiPanel, BorderLayout.NORTH);
    fr.add(déliPanel, BorderLayout.SOUTH);
    final JTextArea halmaz=new JTextArea(5, 40);
    északiPanel.add(halmaz);
    final JTextField szöveg=new JTextField(40);
    déliPanel.add(szöveg);
    fr.pack();
    fr.setVisible(true);
    KeyListener figyelő=
      new KeyListener() {
        {
          szöveg.setEditable(true);
        }
        public void keyReleased(KeyEvent ev) {
          szöveg.setEditable(true);
        }
        public void keyPressed(KeyEvent ev) {
          if (halmaz.getText().indexOf(ev.getKeyChar())==-1)
            szöveg.setEditable(false);
        }
        public void keyTyped(KeyEvent ev) {
          ;
        }
      };
    szöveg.addKeyListener(figyelő);
  }
}
```



7. ábra – A Karakterhalmaz program felhasználói felülete

Irodalomjegyzék

[1] Angster Erzsébet (2002) Objektumorientált tervezés és programozás Java 2, ISBN 963 00 6264 X, 77-94, 263-314

[2] Kölling M. (2008) The BlueJ Tutorial (version 2.0.1)

[3] Nagy Tibor István (É.n.) Kivételkezelés Java-ban (prezentáció)

[4] Rogers Cadenhead (2006) Tanuljuk meg a Java programozási nyelvet 24 óra alatt, ISBN 963 9637 07 6, 217-275, 299-315, 447-497