Java IX.
Kivételkezelés a Java-ban

Ficsor Lajos

Krizsán Zoltán

Miskolci Egyetem

Általános Informatikai Tanszék

Utolsó módosítás: 2005. 11. 17.

A kivételkezelés célja

- Kivételes helyzetek (hibák) a jól megírt programokban is előfordulnak. Például:
  - erőforrás hiány (lemez megtelt)
  - valamilyen funkció hiányos vagy hibás adatok miatt nem hajtható végre.
- Ezeket a szituációkat kezelni kell. A szükséges teendők elvégzése után általában a program folytatni tudja a munkát.

Hagyományos hibakezelés

- Függvény visszatérési értéke (paramétere) hátrányai:
  - azonosítás
  - hibaérték / valódi érték megkülönböztetése nehézkes
  - hívási hierarchia!
- Java: hibastátusz adattag és visszaadó függvény
- Ellenőrzés nehézkes, sok helyre kell beiktatni.
- Kód áttekinthetőségét rontja!
- A funkcionális és a hibakezelő kódrészletek keverednek.

Kivétel - exception

- Futás idejű hiba / nem normális eset
- Általában valamilyen hibához kötődik.
- Számos beépített kivétel.
- Saját kivétel definiálható.
- A Java szemléletében a kivétel egy objektum, ami a kivétel bekövetkezésekor jön létre.
- Saját (programozói) kivétel kiváltása:

  `throw` objektum

Kivétel - hiba

A kivétel keletkezésekor szokásos teendők :

- értesíteni a felhasználót, vagy naplózni,
- majd valamilyen plusz műveletek után
  - folytatni az alkalmazást,
  - esetleg kilépni, súlyos hiba esetén.
- Ellenőrzött kilépés lehetséges. (Például előtte minden értékes adat elmenthető.)
- A kivétel objektum tartalmazhat információkat a kivétel keletkezéséről, amelyet felhasználhatunk.

Kivételkezelés utasításai

- **`try`** – védett kód (blokk) kijelölése
- **`throw`** – kivétel dobása, generálása
- **`catch`** – kivétel elkapása, a blokkban definiált utasítások végrehajtása
- **`finally`** – végül, akár volt kivétel, akár nem volt, lefut

Try blokk

- A védett kódot **`try`** blokkban helyezzük el

`try { utasítások }`

- A blokkban keletkezett kivételt mi kezelhetjük le.
- A blokkok egymásba ágyazhatóak.
- Érdemes minél kisebb blokkokat definiálni.
- Kivétel keletkezik, ha a **`try`** blokkban egy **`throw`** utasításra fut a vezérlés. Formája:

`throw new` kivételTípus(konstruktor paraméterek)

- A **`throw`** paramétere nem maradhat el!

Kivétel keletkezése

Kivétel keletkezhet az alábbi módokon:

- Implicit módon: a JAVA rendszerből indul ki, azaz valamely utasítás vagy API elem végrehajtása során keletkezik.
- A programozó kódjában keletkezik, közvetlenül egy **`throw`** utasítás végrehajtásával.
- Aszinkron kivétel, amely a program egy másik szálán lépett fel.

Kivétel keletkezése (folyt.)

A kivétel keletkezése esetén:

- Az első kivételt kiváltó utasítással befejeződik a **`try`** blokk végrehajtása.
- Kilép a blokkból a vezérlés, rendcsinálással
  - verem visszaállítása,
  - lokális objektumok megszüntetése,
  - védett változók lezárásának megszüntetése.
- Létrejön a **`throw`** utasításban megjelölt objektum egy példánya.

A kivétel elkapása

- A kivétel objektumot minden esetben a virtuális gép hozza létre.
- A kivétel lekezelését szolgáló utasítások **`catch`** blok(kok)ban helyezkednek el. Formája:

```
catch (típus paraméter)
  {utasítások}
```

- A **`catch`** minden esetben a **`try`** blokkot követi, nem lehet köztük más utasítás.
- Egy **`try`** blokkhoz tartozhat több **`catch`** is.

A kivétel elkapása (folyt.)

- A virtuális gép megkeresi a sorrendben első "illeszkedő" blokkot, és annak végrehajtásával folytatódik a program.
- **Az illeszkedés feltétele**: a kivétel objektum típusa megegyezik a **`catch`** blokk fejében megadott típussal, vagy annak leszármazottja.
- A kiválasztott blokk végrehajtása során a paramétere úgy használható, mint a függvények esetén a formális paraméter.
  - Így lehet felhasználni a kivétel objektumban tárolt információkat.

A kivétel elkapása (folyt.)

- A futás az utolsó **`catch`** utáni sorral folytatódik.
- Minden ellenőrzött kivételt kezelni kell (fordítási hiba, ha van lekezeletlen kivétel)!
- Ha ha a **`catch`** blokk végrehajtása során újabb kivétel keletkezik, az eredeti kivétel kezelése megszakad, és az új kivétel lekezelése kezdődik el.

A kivétel elkapása (folyt.)

- Ha egyetlen **`catch`** blokk sem illeszkedett a kivételre, a keresés a beágyazó **`try`** blokk **`catch`** blokkjaival folytatódik, amíg sikeres nem lesz.
- Ha a keresés belülről kifelé minden **`try`** blokkot megvizsgált, és nem talált egyezést, a program terminálódik, és kiíródik a kivétel stack. (Lekezeletlen kivétel.)
- **Megjegyzés:** a **`try`** blokk kivételt kiváltó utasítása utáni utasítások tehát mindig kimaradnak!

A `finally` blokk

- Nem kötelező.
- A cacth(ek) után szerepelhet. Az utolsó **`catch`** blokk és a **`finally`** blokk között nem lehet más utasítás.
- Nem lehet paramétere.
- Minden esetben lefut.
  - Ha kivétel keletkezett a **`try`** blokkban, egy **`catch`** blokk végrehajtása után.
  - Ha nem volt kivétel, a **`try`** blokk utolsó utasítása után.
- Alkalmas például fájlok, adatbázis kapcsolatok lezárására.

Egymásba ágyazott `try` blokkok

A **`try`** blokkok egymásba ágyazhatók

- közvetlenül, vagy
- közvetve, amikor a **`try`** blokkban egy olyan metódus hívása szerepel, amely tartalmaz **`try`** blokkot.
- Úgy tekinthetjük, hogy a **`main`** metódus egy implicit **`try`** blokkban fut, és itt kezelődik le minden olyan kivétel, amelyet egyetlen **`catch`** blokk sem kapott el előtte.

Példa: a vezérlés menete

- Kivétel nélkül:

```
try {
// utasítások
throw new ...
// további utasítások
}
catch (T1 p1) {}
catch (T1 p1) {}
catch (T1 p1) {}
// további utasítások
```

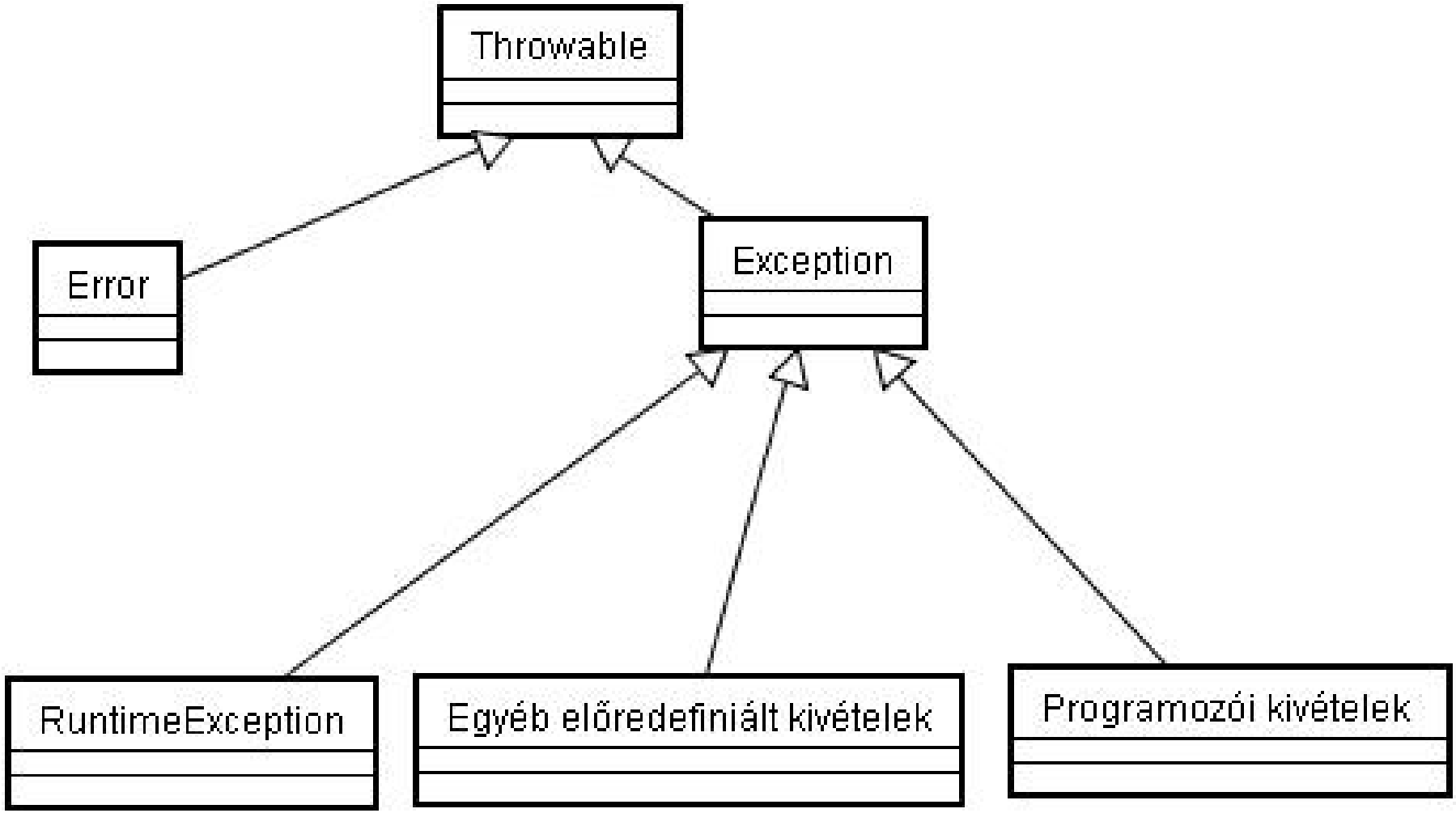
- Kivétellel:

```
try {
// utasítások
throw new ...
// további utasítások
}
catch (T1 p1) {}
catch (T1 p1) {}
catch (T1 p1) {}
// további utasítások
```

Kivétel objektum

- Mivel objektum, valamely definiált osztály példánya kell legyen (kivétel osztály).
- Minden kivétel osztály a **`Java.lang.Throwable`** osztály leszármazottja kell legyen, de ez általában közvetve valósul meg.
- A **`Java.lang.Throwable`** osztálynak van számos előre definiált (a Java API részét képező) leszármazottja.

Kivétel osztályok



Kivétel osztályok (folyt.)

- Az **`Error`** és a **`RuntimeException`** osztályú kivételek nem ellenőrzött kivételek.
- Az összes többi ellenőrzött kivétel.
- Ajánlott: minden programozó által definiált kivétel az **`Exception`** osztályból származzon.
  - Valamennyi ilyen kivétel ellenőrzött kivétel, kötelező lekezelni.
- Konvenció: minden kivételosztály neve

`ValamiException`

legyen, ahol a **`Valami`** a kivétel jellegére utal.

Nem ellenőrzött kivételek

- Az **`Error`** és a **`RuntimeException`** osztályból származó kivételek
  - **`Error`**: rendszerszintű hiba a JVM működésében. (Pl. **`OutOfMemoryError`**)
  - **`RuntimeException`**: a program számos pontján keletkezhetnek, ezért nem célszerű kötelezővé tenni a lekezelésüket. (Pl. **`ArrayIndexOutOfBounds`**).
- A programozó, ha akarja, lekezelheti.
- Lekezeletlen kivétel esetén a program terminálódik, és kiíródik a kivétel stack.

Ellenőrzött kivételek

A fordítóprogram hibát jelez, ha nincs lekezelve!

A kivétel lekezelésének módjai:

- A **`try`** blokk utáni valamelyik **`catch`** blokk elkapja, és teljes mértékben lekezeli. A metódus futása folytatódik a **`catch`** blokkok utáni első utasítással.
- A **`try`** blokk utáni valamelyik **`catch`** blokk elkapja, és részben lekezeli.
  - A lekezelés egy olyan **`throw`** utasítással fejeződik be, amely a paraméterül kapott kivétel objektumot tartalmazza. A metódus futása befejeződik.
  - A metódus fejlécében ezt a kivételt specifikálni kell.

Ellenőrzött kivételek (folyt.)

- A metódus nem kezeli le a kivételt, de specifikálja azt a fejlécében. A metódus futása befejeződik.
  - Ez akkor szükséges, amikor az adott metódus nem, csak a hívó tudja értelmesen lekezelni a kivételt.
  - A hívó is tovább adhatja a kivétel lekezelését, így a kivétel tetszőleges hívási mélységből is eljuthat oda, ahol lekezelhető.

Kivételek specifikálása

- Ha egy metóduson belül kivétel keletkezhet, de nem tartalmaz **`catch`** utasításokat annak kezelésére.
- Ilyenkor a metódus hívójának kell azokat lekezelni. Erre számítania kell, ezért az ilyen kivételeket a metódus fejlécében specifikálni kell.
- Formája:
  metódusnév`([parameterlista])` **`[throws`**
  kivételosztály1 `[,` kivételosztály2`, ...] ]`

Az `Exception` osztály

- Van egy **`String`** paraméterű konstruktora, amellyel egy leírás definiálható a kivételhez.
- Számos hasznos metódust definiál, amely örökölődik, és szükség esetén felüldefiniálható.
  - **`String toString()`**: visszaad egy sztringet, amely az osztály azonosítójából és a konstruktorban megadott sztringből áll
  - **`String getMessage()`**: visszaadja a konstruktorban megadott sztringet.

Az `Exception` osztály (folyt.)

- **`printStack()`**: kiírja a keletkezett kivételeket, a keletkezésük sorrendjében és megadja a keletkezésük helyét.
  - Hasznos lehet a hibakereséshez.
  - Paraméter nélkül a standard hibacsatornára ír, de paraméterben megadható, hogy hová írjon.

Saját kivétel osztály

Ajánlások:

- Az **`Exception`** leszármazotja legyen.
- Név konvenció! (**`ValamiException`**)
- A konstuktorának a paraméterei között legyen egy String, amivel az ősosztály konstruktorát hívja meg.
- A további paraméterek a hibára jellemző adatokat fogadhatnak.
- A metódusai a **`catch`** blokkban használhatók a kivétel lekezelése során.

Saját kivétel osztály (folyt.)

- A kivételek csoportosítására célszerű leszármazási hierarciát létrehozni a saját kivételosztályok között is.

Egyszerű példa

- Saját kivétel osztály:

```
package kivetel2;
public class PeldaException extends
  Exception {
  public PeldaException(String a) {
     super(a);
  }
}
```

Egyszerű példa (folyt.)

- Kivételt kiváltó kód:

```
package kivetel2;
public class proba2 {
  public static void main(String[] args) {
    try {
     throw new PeldaException("Leiras");
    }
```

Egyszerű példa (folyt.)

- Kivételt lekezelő kód:

```
catch (PeldaException e){
      System.out.println(e); // e.toString()
      System.out.println(e.getMessage());
      e.printStackTrace(System.out);
      }
finally {
      System.out.println("Kész");
      }
  } // main vége
} // class proba2 vege
```

Egyszerű példa (folyt.)

- Az output:

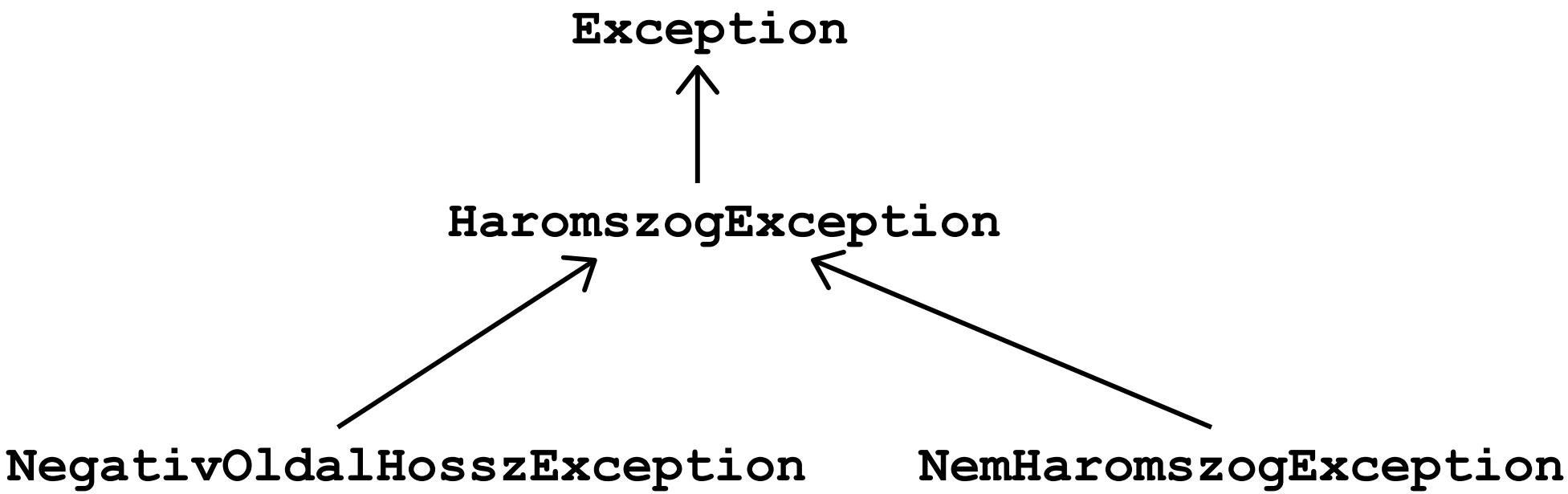
```
// e.toString() eredménye:
kivetel2.PeldaException: Leiras
// e.getMessage eredménye:
Leiras
// printStack eredménye:
kivetel2.PeldaException: Leiras
at kivetel2.proba2.main(proba2.java:9)
// finally blokk
Kész
```

További példa

- Háromszög osztály.
- A konstruktornak vizsgálnia kell, hogy a kapott oldalhosszakból képezhető-e háromszög.
- Ha nem, azt két módon jelezheti (nincs visszatérési érték!):
  - hiba státusz adattag (nem elegáns megoldás)
  - kivételt dob.
- A keletkezett kivételt nem tudja értelmesen lekezelni, tehát specifikálnia kell a hívó részére.

További példa (folyt.)

- Saját kivétel osztályok:



További példa (folyt.)

- Az osztályok felüldefiniálják a **`toString`** metódust.
- Ezért mindhárom kivétel kiírható a **`System.out.println()`** metódussal.
- A kiíráshoz a megfelelő metódust a dinamikus típus alapján kell megkeresni (késői kötés).
- A példa a kivételkezelésen kívül néhány korábbi nyelvi szerkezetet is szemléltet.

További példa (folyt.)

- A kivételek jelentése:
  - **`NegativOldalHosszException:`** legalább egy oldalhossz nem pozitív.
  - **`NemHaromszogException:`** az oldalhosszak pozitívok, de a háromszög egyenlőtlenség nem teljesül.
  - **`HaromszogException:`** valami miatt nem képezhető háromszög az adatokból.
- A közös ős hasznos lehet a kivételek összefoglaló típusaként is.

További példa (folyt.)

```
package kivetel3;
public class HaromszogException extends
  Exception {
// Az adatok tarolasara
  protected double a;
  protected double b;
  protected double c;
```

További példa (folyt.)

```
// Konstruktor
  public HaromszogException(double a,
  double b, double c) {
     super("Nem haromszög!");
     this.a = a; // Eltárolja a hibás
     this.b = b; // adatokat
     this.c = c;
  }
  public String toString() {
     return "Nem alkot haromszöget: " + a
  + b + c;
   }
} // class vege
```

További példa (folyt.)

Megjegyzés:

- A konstruktor (vagy valamely metódus) formális paraméterének azonosítója egyezhet egy adattagéval.
- A **`this`** pszeudó változó használatával a kettő megkülönböztethető.
- A konstruktor paraméterének és a hozzá tartozó adattagnak az egyezése egy szokásos kódolási konvenció.

További példa (folyt.)

```
package kivetel3;
public class NegativOldalHosszException
  extends HaromszogException {

  public NegativOldalHosszException(double
  a, double b, double c) {
     super(a, b, c);
  }
  public String toString() {
     return "Legalább egy negatív oldal: "
  + a + b + c;
  }
}
```

További példa (folyt.)

Megjegyzés:

- Az `a, b, c` adattagokra vonatkozó hivatkozás helyes, annak ellenére, hogy nincsenek az osztálynak ilyen nevű adattagjai.
  - Az ősosztálytól örökli.
  - Mivel a minősítésük **`protected`**, közvetlenül el is érheti.

További példa (folyt.)

```
package kivetel3;
public class NemHaromszogException
  extends HaromszogException {
  public NemHaromszogException(double a,
  double b, double c) {
     super(a, b, c);
     }
  public String toString() {
     return "Nem teljesül a háromszög
  egyenlőtlenség: " + a + b + c;
  }
}
```

További példa (folyt.)

És akkor a **`Haromszog`** osztály:

```
package kivetel3;
public class Haromszog {
  private double a;
  private double b;
  private double c;
```

További példa (folyt.)

```
// Konstruktor
public Haromszog(double a, double b,
  double c) throws HaromszogException {
  try {
     if (a <= 0 || b <= 0 || c <= 0){
     throw new
  NegativOldalHosszException(a, b, c);
     }
```

További példa (folyt.)

```
if (((a + b) > c) && ((b + c) > a) &&
((c + a) > b)) { // objektum OK!
            this.a = a; // Inicializálás
            this.b = b;
            this.c = c;
  }
 else {
  throw new NemHaromszogException(a, b,
   c);
  }
} // try blokk vége
```

További példa (folyt.)

Megjegyzés:

- A **`try`** blokkban csak a két leszármazott típusú kivétel keletkezhet, de a konstruktornak elég a közös őst specifikálnia.

További példa (folyt.)

```
        catch (HaromszogException h) {
          throw h; // kivétel tovább dobása
        }
    } // konstruktor vége

    public double kerulet() { // Metódus
        return a + b + c;
    }

} // class Haromszog vege
```

További példa (folyt.)

Megjegyzés:

- Csak egy **`catch`** blokk van, amelyben az ős osztály típusa szerepel.
- Ez a blokk mindkét kivételre illeszkedik.
- A **`catch`** blokk nem tudja lekezelni a kivételt, ezért tovább dobja.
- Kivétel keletkezése esetén az adattagok explicite nem inicializálódnak.
  - Alapértelmezés szerint 0 lesz az értékük.

További példa (folyt.)

```
package kivetel3;
public class proba {
  public static void main(String[] args) {
     try {
          Haromszog h = new Haromszog (4.,
  5., 6.);
          System.out.println ("Kerület:" +
  h.kerulet());
     }
     catch (HaromszogException e) {
          System.out.println(e);
     }
```

További példa (folyt.)

Megjegyzés:

- Nem keletkezik kivétel. Az output:

```
Kerület:15.0
```

- A **`catch`** blokk mindkét kivételt elkapná, de most kimarad a végrehajtásból.

További példa (folyt.)

```
try {
  Haromszog h=new Haromszog (4.,-5.,
  6.);
  System.out.println ("Kerület:" +
  h.kerulet());
}
catch(NegativOldalHosszException e) {
    System.out.println(e);
}
catch (HaromszogException e) {
    System.out.println(e);
}
```

További példa (folyt.)

Megjegyzés:

- Kivétel keletkezik.
- A try blokk további részei (a kerület számítása és kiírása) elmarad.
- A kivételt az első **`catch`** blokk kapja el. Az output:

```
Legalább egy negatív oldal: 4.0-5.06.0
```

- A második **`catch`** blokk kimarad.

További példa (folyt.)

```
try {
  Haromszog h = new Haromszog (4., 5., 60.);
  System.out.println ("Kerület: " +
  h.kerulet());
}
catch (NegativOldalHosszException e) {
  System.out.println(e);
}
catch (NemHaromszogException e) {
  System.out.println(e);
}
catch (HaromszogException e) {
  System.out.println(e);
}
```

További példa (folyt.)

Megjegyzés:

- Kivétel keletkezik, amelyet a második **`catch`** blokk kap el. Az output:

```
Nem teljesül a haromszög egyenlőtlenség:
   4.05.060.0
```

- Az első és a harmadik **`catch`** blokk kimarad.

További példa (folyt.)

```
try {
  Haromszog h = new Haromszog (4., 5.,
  60.);
  System.out.println ("Kerület: " +
  h.kerulet());
}
catch (HaromszogException e) {
  System.out.println(e);
}
```

További példa (folyt.)

Megjegyzés:

- Most is **`NemHaromszogException`** kivétel keletkezik, amelyet a **`catch`** blokk elkap, mert leszármazottja a **`HaromszogException`** kivételnek. Az output:

```
Nem teljesül a háromszög egyenlőtlenség:
  4.05.060.0
```

További példa (folyt.)

- A kiírás a tényleges hibának megfelelő, annak ellenére, hogy **`HaromszogException`** típusú kivételt írunk ki.
- Magyarázat: a **`toString`** metódusok felül definiáltak, így hívásukat nem a statikus, hanem a dinamikus típus határozza meg.