Programozás III

KIVÉTEL

KIVÉTELKEZELÉS – PÉLDA HIBÁS PROGRAMRA

Nullával való osztás:

```
public class Kivetel{

    public static void main(String[] args){

        int szam = 20;
        System.out.println("A hányados: " + szam/0);
    }
}
```

KIVÉTELKEZELÉS

Hibátlan program nincs!!!

hiba esetén leállhat a program.

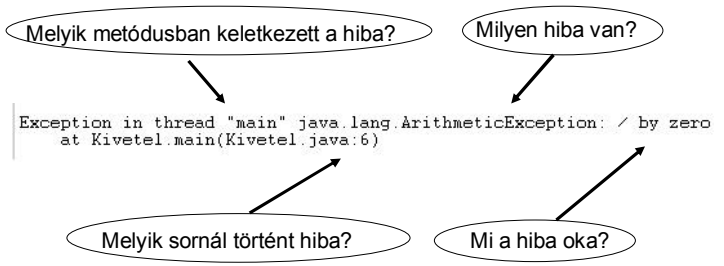
Példa ilyen hibákra:

- ArrayBoundsOfException (tömb túlindexelése)
- ArithmeticException (pl. osztás nullával)

A Java támogatja ezen hibák kezelését, ezt nevezzük ***kivételkezelésnek.***

KIVÉTELKEZELÉS – PÉLDA HIBÁS PROGRAMRA

A fordító minden további nélkül lefordította a programot!
DE! Futási időben hibát kapunk:



KIVÉTELKEZELÉS

A kivételek jelenítik meg a hibákat.

A kivételek (exceptions) objektumok, a kivételosztályok példányai

Amikor hiba történik, létrejön egy kivétel objektum, ami a hibáról tartalmaz értékes információkat.
(Ilyen kivétel objektum pl. az ArithmeticException vagy az ArrayBoundsOfException osztály egy példánya is)

Szóhasználat:

kivétel keletkezik (occurs), dobódik (is thrown) vagy explicite dobjuk (throw).

A kivétel kezelése (handling): a dobott kivételt elkapjuk (catch).

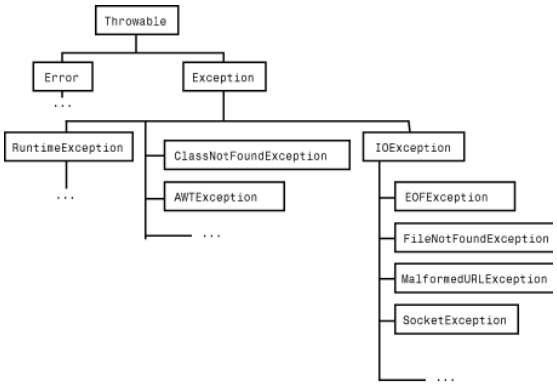
KIVÉTELKEZELÉS

Kivételek típusai:

Ellenőrzött kivétel (checked exception)
Az ilyen kivétel kezelése **kötelező**
Ezek az Exception osztály leszármazottai
Pl.: fájlkezeléssel kapcsolatos kivételek

Ellenőrizetlen kivétel (unchecked exception)
A hiba kezelése **nem kötelező**, mert nagyon kényelmetlenné tenné a programozást, átláthatatlanná a programkódot.
Ezek a RuntimeException, vagy az Error osztály leszármazottai.
Pl.: nullpointer használata, nullával való osztás.

KIVÉTELKEZELÉS - KIVÉTELOSZTÁLYOK



Error: rendszerhiba → nem kezelhető

Exception: kivétel → kezelhető!!!!!!! ☺

KIVÉTELKEZELÉS

Kivétel dobása:

Kivételt a **throw** kulcsszóval tudunk dobni.

Formátuma:
throw *Kivételobjektum;*

A kivételobjektum a Throwable osztálynak vagy leszármazottainak egy példánya lehet.

KIVÉTELKEZELÉS

(szándékos) Kivétel dobása – példa:

```
class Egyik{
   static void m1(){
       throw new RuntimeException("Próbálkozás");
   }
}

public class KivetelUtja{

   static void m2(){
       Egyik.m1();
   }

   public static void main(String args[]){
       m2();
   }
}
```

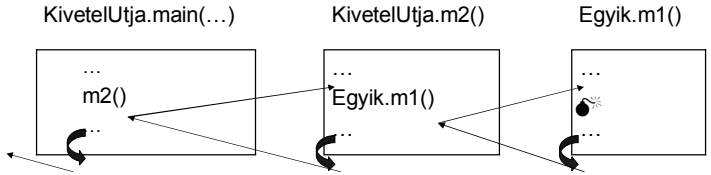
KIVÉTELKEZELÉS

A kivételeket el tudjuk kapni (catch) és kezelni tudjuk őket, tehát nem feltétlenül kell a programunknak hibával „elszállnia”. Erre jó a **try – catch** szerkezet.

```
try{            //try blokk

    //...utasítások ...
}
catch (KivételOsztály1 k) {     // 1. catch blokk

    //...1. típusú kivételek kezelése ...
}

catch (KivételOsztály2 k) {     // 2. catch blokk

    //...2. típusú kivételek kezelése ...
}
```

KIVÉTELKEZELÉS

Kivétel dobása – példa:



Exception in thread "main" java.lang.RuntimeException: Próbálkozás
at Egyik.m1(KivetelUtja.java:3)
at KivetelUtja.m2(KivetelUtja.java:10)
at KivetelUtja.main(KivetelUtja.java:14)

Process completed.

KIVÉTELKEZELÉS

A 7-es Java óta már az is megengedett, hogy a catch blokkban egyszerre több kivételtípust kezeljünk, pl.:

```
try{
    ...
}
catch( IOException | NumberFormatException ex ) {
    // Mindig kötelező kezelni a hibát!!
    ...
}
```

KIVÉTELKEZELÉS – TRY-CATCH SZERKEZET

Felépítése:

- **try blokk:**
A try blokk tartalmazza a program normális logikáját tükröző utasításokat. Általában a try blokk futása során keletkeznek azok a kivételek, amelyeket el kell fognunk

- **catch blokk**:
Minden catch blokk egy-egy kivételkezelőt definiál. A blokk fejében paraméterként egy (vagy a 7-es Java óta akár több) formális kivétel-objektum van megadva. A catch blokk fogja kezelni az érkező kivételobjektumot.

A TRY- CATCH SZERKEZET MŰKÖDÉSE

Általános hibakezelés:

Abban az esetben, ha nem tudjuk, hogy milyen hibák keletkezhetnek a try blokkon belül, akkor lehetőségünk van olyan hibakezelésre, amellyel bármilyen hibát elkapunk.

Ezt úgy valósíthatjuk meg, ha a catch ágban az általános **Exception** osztályt használjuk. Erre a szabályra minden kivétel illeszkedik.

A TRY- CATCH SZERKEZET MŰKÖDÉSE

A **try – catch** blokk végrehajtása:

• Ha a try blokk utasításainak végrehajtása során **nem dobódott kivétel**, a try-catch blokk végrehajtása **normálisan befejeződik**.

• Ha **kivétel dobódott**, sorban megpróbáljuk **illeszteni** a keletkezett kivétel objektumot **a catch blokkok fejében szereplő osztályokra**. Illeszkedés: a kivétel osztály megegyezik a catch blokk fejében adott osztállyal, vagy annak leszármazottja.

• Ha **volt illeszkedés**, az adott **catch blokkot végrehajtjuk**, és a try-catch blokk normálisan befejeződik, illetve ha a catch blokkban újabb kivétel dobódik, kivétellel fejeződik be. (⇒ fontos a sorrend)

• Ha **nincs illeszkedés**, a try-catch blokk **kivétellel fejeződik be**.

KIVÉTELKEZELÉS – A PÉLDA JAVÍTÁSA

```
class Egyik{
   static void m1(){
       throw new RuntimeException("Próbálkozás");
   }
}

public class KivetelUtja{

   static void m2(){
       Egyik.m1();
   }

   public static void main(String args[]){
       try{
           m2();
       }
       catch(Exception e){
           System.out.println(e.getMessage());
       }
   }
}
```

Próbálkozás
Process completed.

KIVÉTELKEZELÉS – TRY-CATCH-FINALLY SZERKEZET

Szükségünk lehet arra, hogy a try-catch lefutásának eredményétől függetlenül végrehajtsunk egy kódrészletet. Erre szolgál a finally blokk. Itt szokás elvégezni a mindenképpen szükséges műveleteket.

A finally blokk mindenképpen végrehajtódik!!!

```
try{              //try blokk

    //...utasítások ...
}
catch (KivételOsztály k) {      // catch blokk

    //...adott típusú kivételek kezelése ...
}
finally {                       // finally blokk

    //...utasítások ...
}
```

KIVÉTELKEZELÉS – SAJÁT KIVÉTEL

```
class SajatKivetel extends Exception{

   SajatKivetel(String uzenet){
      super(uzenet);
   }
}

public class Kivetel{

   static int szamOlvas(int tol, int ig){
      if(tol>ig){
         int temp = tol; tol = ig; ig = temp;
      }

      int szam;

      try{
         szam = Input.readInt();
         if(szam<tol || szam>ig)
            throw new SajatKivetel("A szám "+ tol + " és " + ig + " között lehet!");
         return szam;
      }
      catch(SajatKivetel e){
         System.out.println(e.getMessage());
      }
   }

   public static void main(String args[]){
      System.out.print("kérek egy 1 és 10 közötti számot: ");
      int szam = szamOlvas(1,10);
      System.out.println("Vége");
   }
}
```

```
kérek egy 1 és 10 közötti számot: 14
A szám 1 és 10 között lehet!
-2
A szám 1 és 10 között lehet!
4
Vége
```

KIVÉTELKEZELÉS – SAJÁT KIVÉTEL

```
class SajatKivetel extends Exception{

   SajatKivetel(String uzenet){
      super(uzenet);
   }
}
```

```
try{
    szam = Input.readInt();
    if(szam<tol || szam>ig)
       throw new SajatKivetel("A szám "+ tol + " és " + ig + " között lehet!");
    return szam;
}
catch(SajatKivetel e){
    System.out.println(e.getMessage());
}
```

KÖTELEZŐ KIVÉTELKEZELÉS

Bizonyos esetekben a kivételkezelés kötelező!!!! (ellenőrzött kivételkezelés)

(pl. bufferelt beolvasásnál, fájlkezelésnél)

```
BufferedReader bufr = new BufferedReader(insr);
 try
  { str=bufr.readLine(); }
 catch (IOException ioe)
  { System.out.println("Hiba a bemeneti csatornán [reason: "+ioe.getMessage()+"]"); }
```

```
try {
 if (input.equals("int")) Integer.parseInt(ret);
 else if (input.equals("float")) Float.parseFloat(ret);
 else if (input.equals("double")) Double.parseDouble(ret);
 else if (input.equals("char") && ret.length()<2) ret.charAt(0);
 else throw new StringIndexOutOfBoundsException("String must containt one character");
 ok=true;
}
catch (NumberFormatException nfe)
 { printError(nfe); }
catch (StringIndexOutOfBoundsException siob)
 { printError(siob); }
```

KIVÉTELKEZELÉS – PÉLDA

Az indító példa javítása:

```
System.out.print("Adjon meg egy számot: ");
szam = Input.readInt();
System.out.print("Adja meg az osztót: ");
oszto = Input.readInt();
try{
    eredmeny = szam/oszto;
    System.out.println("Az eredmény: " + eredmeny);
}
catch(Exception e){
    System.out.println("Elkaptuk a hibát, a hiba:" + e);
}
```

KIVÉTELKEZELÉS – PÉLDA: MI AZ EREDMÉNY?

```
public class Kivetel{
   public static void main(String[] args){

       int n=3, m=2;
       int tomb[] = {2,3,4};

       try{
           for(int i=0; i<=n; i++)
               tomb[i]++;
           m++;
       }
       catch(Exception e1){
           m = n+m;
       }
       catch(ArithmeticException e2){
           m ++;
       }
       catch(IndexOutOfBoundsException e3){
           n ++;
       }

       System.out.println("n= " + n + "\nm= " + m);
   } }
```

KIVÉTELKEZELÉS – PÉLDA: MI AZ EREDMÉNY?

```
public class Kivetel{
   public static void main(String[] args){

       int b = 3, c =5;

       try{
           b = b/c;
           c = c/b;
       }
       catch(NumberFormatException e1){
           c ++;
       }
       catch(ArithmeticException e2){
           c=c+b;
       }
       catch(Exception e3){
           b ++;
       }
       finally{
           c = c+2;
       }

       System.out.println("c= " + c);
   } }
```

KIVÉTELKEZELÉS – PÉLDA: MI AZ EREDMÉNY?

```
try (InputStream ins = this.getClass().getResourceAsStream(adatUt);
        Scanner fajlScanner = new Scanner(ins, CHAR_SET)) {
    String sor;
    String[] adatok;
    Ital ital;
    while (fajlScanner.hasNextLine()) {
        sor = fajlScanner.nextLine();
        adatok = sor.split(";");
        ital = new Ital(adatok[0], adatok[1],
                Integer.valueOf(adatok[2]));

        italok.add(ital);
    }
} catch (Exception ex) {
}
```

Mi a hiba?

TILOS üres catch ágat hagyni!!!

Ha nem generáljuk a catch ágat, akkor ezt mindenképpen bele kellene írni:

```
catch (Exception ex) {
    ex.printStackTrace();
}
```

Hf: Unit-teszt átismétlése.