

Java tutorial

Copyright © 2000-2001, Kozsik Tamás

Kivételek

- A program végrehajtása során ritkán bekövetkező események
- Nem a „fő végrehajtási ág”; logikailag alacsonyabbrendű feladat jelzése
- Hiba
- Felhasználó butaságot csinál
- Speciális/abnormális számítási eredmény

Hibák

- Más nyelvekben elképzelhető, hogy egy hiba fejreállítja a programot
 - Jó esetben csak elszáll
 - Esetleg nagy butaságot csinál, pl. elrontja az adatbázist
- Java-ban a futtató rendszer ellenőrzi a hibákat

Milyen hibák vannak?

- Nullával való osztás
- Tömb túlindexelés
- Hivatkozás „null” mutatón keresztül
- Érték túl/alulcsordulás
- Nincs meg egy fájl
- Hálózati kapcsolat megszakad

Kivétel != Hiba

- A kivételek nem mindig hibát jeleznek
- Lehet, hogy csak egy ritkán bekövetkező, vagy a feladat szempontjából kevésbé fontos eseményt

Kivétel kezelése

- Egy jól megírt, megbízható program jelentős része a kivételes eseményekkel foglalkozik
- Jó, ha van programnyelvi támogatás erre

Ha nincs kivételkezelésre támogatás

- Pl. C-ben, Pascal-ban nincs speciális eszköz a kivételek kezelésére
- Megoldás: visszatérési hibakódok, plussz paraméterek, esetvizsgálatok (elágazások)
- ... vagy egyszerűen semmi ...
 - lustaság
 - olvashatóság, elegancia

Kivételkezelést támogató nyelvi elemek

- Kis erőfeszítéssel, az olvashatóságot és az eleganciát megtartva lehessen kivételeket kezelni
- Általában a kivételeket kezelő kódot elválasztják a többitől, a „lényegtől”
- Az elkészült programok megbízhatóságát, olvashatóságát növelik

Tartalom

- Kivételek fellépése
- Kivételek terjedése
- Kivételek lekezelése
- Kivételek továbbterjedésének specifikálása
- Kivételek definiálása
- Kivételek kiváltása
- Különböző kivételfajták

Java tutorial

Copyright © 2000-2001, Kozsik Tamás

Kivétel fellépése

- A program egy pontján, egy utasítás végrehajtása közben
- Jelezhet hibát, vagy speciális eseményt
- NullPointerException,
ArrayIndexOutOfBoundsException
ClassCastException
IOException

Példa

```
class A {  
    public static void main(String args[]) {  
        int[] t;  
        t[0] = 12;           // fordítási hiba  
        t = new int[3];  
        t[3] = 21;  
    }  
}
```

Példa

```
class A {  
    public static void main(String args[]){  
        int[] t = null;  
        t[0] = 12;    // NullPointerException  
        t = new int[3];  
        t[3] = 21;  
    }  
}
```

Példa

```
class A {  
    public static void main(String args[]){  
        int[] t;  
        // t[0] = 12;  
        t = new int[3];  
        t[3] = 21; // ArrayIndexOutOfBoundsException.  
    }  
}
```

Java tutorial

Copyright © 2000-2001, Kozsik Tamás

Kivételek terjedése

- A hívási lánc mentén
 - A végrehajtási verem mentén
- Ha egy **m** metódusban kivétel lép fel, akkor az azt meghívó metódusban is fellép, azon a ponton, ahol meghívtuk az **m** metódust
 - hacsak persze le nem kezeljük...
- Egészen addig, amíg a main-ben is fel nem lép: ekkor leáll a program, és kiírja a kivételt
 - stack trace

Példa

```
class A {  
    public static void main(String args[]){  
        m1(3);  
        m1(80);  
    }  
    static void m1( int i ) {  
        int[] t = new int[i];  
        m2(t);  
    }  
    static void m2( int[] t ){ t[7] = 12; }  
}
```

A "stack trace"

```
$ java A  
Exception in thread "main"  
java.lang.ArrayIndexOutOfBoundsException  
    at A.m2(A.java:10)  
    at A.m1(A.java:8)  
    at A.main(A.java:3)
```

- Rengeteg hasznos információ
 - Melyik végrehajtási szál
 - Milyen kivétel (hiba) lépett fel
 - Melyik fájlban, melyik sorban, melyik metódusban
 - Milyen hívási lánc mentén terjedt

Feladat

- Váltunk ki egy hibát: osszunk le egy egész számot nullával
- Először a főprogramban
- Próbáljuk ki egy, a főprogramból meghívott metódusban
- Az osztandó és az osztó legyen parancssori argumentum

Java tutorial

Copyright © 2000-2001, Kozsik Tamás

Kivétel lekezelése

- A kivétel terjedése közben egy ponton a hívási láncon lekezelhetjük
- Egy speciális vezérlési szerkezet:
try catch finally

Példa

```
class A {  
    public static void main(String args[]){  
        m1(3);  
        m1(80);  
    }  
    static void m1( int i ) {  
        int[] t = new int[i];  
        try { m2(t); } catch (Exception e){}  
    }  
    static void m2( int[] t ){ t[7] = 12; }  
}
```

Példa

```
class A {  
    public static void main(String args[]){  
        m1(3);  
        m1(80);  
    }  
    static void m1( int i ) {  
        int[] t = new int[i];  
        try { m2(t); } catch (Exception e){}  
    }  
    static void m2( int[] t ){ t[7] = 12; }  
}
```

Példa

```
class A {  
    public static void main(String args[]){  
        m1(3);  
        m1(80);  
    }  
    static void m1( int i ) {  
        int[] t = new int[i];  
        try { m2(t); }  
            catch (Exception e){...}  
    }  
    static void m2( int[] t ){ t[7] = 12; }  
}
```

Példa

```
static void m1( int i ) {  
    int[] t = new int[i];  
    try { m2(t); }  
    catch (Exception e){  
        System.out.println(e);  
    }  
}
```

Hogyan kezeljük le egy kivételt

- Valami értelmes dolgot csináljunk
- Próbáljuk folytatni a működést a kivétel (pl. hiba) ellenére
- Hárítsuk el a hibát, és próbálkozzunk újra
- Mentsük, ami menthető
 - Zárjuk le a fájlokat, adatbázist...

A hiba kiírása

- Sok programozó kiírja, hogy hiba történt, és kilép a programból
- Értelmetlen módja a hiba „lekezelésének”
- Amúgy is kiírta volna a virtuális gép...

Java tutorial

Copyright © 2000-2001, Kozsik Tamás

try - catch

- Védett blokk: try
- Kivételkezelő ágak: catch
- Egy védett blokkhoz több kivételkezelő ág
 - Különböző kivételekhez...
- A kivétel fajtájától függ, melyik kivételkezelő ág aktivizálódik

Példa

```
try { ... }  
catch (NullPointerException e){ ... }  
catch (IOException e){ ... }  
catch (InterruptedException e){ ... }
```

Példa

```
try {  
    ...  
} catch (NullPointerException e) {  
    ...  
} catch (IOException e) {  
    ...  
} catch (InterruptedException e) {  
    ...  
}
```

A kivételkezelő keresése

- Ha a try blokkban kivétel lép fel, akkor a hozzá tartozó catch ágakban keres a JVM kivételkezelőt
- Sorba nézi a catch ágakat, az első megfelelő törzset végrehajtja
- Megfelelő: ha a kivétel fajtája beletartozik a specifikált kivételosztályba

Példa

```
try {  
    ... IOException fellép  
} catch (NullPointerException e) {  
    ...  
} catch (IOException e) {  
    ...  
} catch (InterruptedException e) {  
    ...  
}
```

Feladat

- Az előbbi feladat folytatása: kezeljük le a kivételt az osztást végző metódusban.

Java tutorial

Copyright © 2000-2001, Kozsik Tamás

A kivételek is objektumok

- A kivétel fajtája - az objektum osztálya
- A kivételek hierarchiába vannak szervezve: az osztályhierarchia által
- Beletartozik egy kategóriába: altípusosság

Példa

```
try {
    ... EOFException fellép
} catch (NullPointerException e) {
    ...
} catch (IOException e) {
    ...
} catch (InterruptedException e) {
    ...
}
```

Ha nincs megfelelő catch ág

- Ha nem találunk megfelelő kivételkezelőt, akkor a kivétel továbbterjed
 - Mintha nem is lett volna kivételkezelő rész
- A hívóban újból lehetőségünk van a kivétel lekezelésére

Példa

```
try {
    ... IndexOutOfBoundsException fellép
} catch (NullPointerException e) {
    ...
} catch (IOException e) {
    ...
} catch (InterruptedException e) {
    ...
}
```

Java tutorial

Copyright © 2000-2001, Kozsik Tamás

Hol kezeljük le a kivételt

- Ott, ahol ez értelmesen megtehető
 - ne előbb
 - ne később
- Ha nem tehető meg értelmesen, inkább engedjük, hogy a program elszálljon

Feladat

- Az előző feladat folytatása: a metódus legyen függvény, ami visszaadja az osztás eredményét. A főprogram kezeli le a kivételt. Írja ki, hogy az osztás eredménye pozitív vagy negatív végtelen, vagy esetleg definiálatlan. (0/0)

Java tutorial

Copyright © 2000-2001, Kozsik Tamás

Ha sikerül lekezelni a kivételt

- A futás a kivételkezelő rész után folytatódik
 - Nem „megy vissza” a try-ba, ahol fellépett

```
<ut 1>
try {
    <ut 2>
    <ut 3>
    <ut 4>
} catch ( <exc 1> ) { <ut 5> <ut 6> <ut 7> }
  catch ( <exc 2> ) { <ut 8> <ut 9> }
  catch ( <exc 3> ) { <ut 10> }
<ut 11>
```

Ha **nem** sikerül lekezelni a kivételt

- A kivétel a hívás helyén fellép
 - Az adott metódus végrehajtása megszakad

```
<ut 1>
try {
    <ut 2>
    <ut 3>
    <ut 4>
} catch ( <exc 1> ) { <ut 5> <ut 6> <ut 7> }
  catch ( <exc 2> ) { <ut 8> <ut 9> }
  catch ( <exc 3> ) { <ut 10> }
<ut 11>
```

A kivételkezelő ágak sorrendje

- A szűkebb meg kell, hogy előzze a bővebbet

```
try { ... }
catch( EOFException e1 ){ ... }
catch( IOException e2 ){ ... }
```

- Ez így jó.

A kivételkezelő ágak sorrendje

- A szűkebb meg kell, hogy előzze a bővebbet

```
try { ... }
catch( IOException e1 ){ ... }
catch( EOFException e2 ){ ... }
```

- Ez **nem** jó. Fordítási hiba.
 - A második sohasem választható ki.

finally

- A try blokk és a catch ágak után írható egy finally blokk
- Azokat az utasításokat tartalmazza, amelyeket **mindenfélképpen** végre kell hajtani.

Példa

```
try { ... }  
catch( ... ){ ... }  
catch( ... ){ ... }  
finally { ... }
```

Példa

```
try {  
    ...  
} catch( ... ){  
    ...  
} catch( ... ){  
    ...  
} finally {  
    ...  
}
```

finally: mindenképpen

- Ha nem lépett fel kivétel...
- Ha fellépett, de nem találunk megfelelő kivételkezelő ágat...
- Ha találunk...: akkor utána

Java tutorial

Copyright © 2000-2001, Kozsik Tamás

Továbbterjedés specifikálása

- Ha egy kivétel fellép egy metódusban, akkor: vagy le kell kezelni vagy jelezni kell, hogy továbbadhatjuk
- A metódusok specifikációja tartalmazza a metódusban fellépő lehetséges kivételeket
- A paraméterlista és a törzs között
- throws utasítás

Példa

```
public static void main(String args[]){  
    try {  
        InputStream in =  
            new FileInputStream("input.txt");  
        ...  
    } catch (IOException e) { ... }  
}
```

Példa

```
public static void main(String args[])
throws IOException {
    InputStream in =
        new FileInputStream("input.txt");
    ...
}
```

Szabályozott terjedés

- Ha egy művelet kiválthat egy kivételt, akkor a művelet használója tudni fog róla
- Pl. lekezelheti
- Ha nem, neki is specifikálnia kell, így az őt használó is tudomást szerez a kivételről
- A kivétel olyan, mint egy speciális visszatérési érték

RuntimeException

- Vannak olyan kivételek, amelyeket nem kell lekezelni vagy a továbbterjedését specifikálni
- Túl sok helyen felléphetnek
 - Lényegében a program minden pontján
- Elbonyolítaná a programot, ha ...
- Általában programozói hibát jelentenek, nem „kivételes eseményt”

Mik ezek

- NullPointerException
 - Bármelyik objektumhivatkozásnál
- ArrayIndexOutOfBoundsException
 - Bármelyik tömbindexelésnél
- ArithmeticException
 - Bármelyik egész osztáskor
- stb.

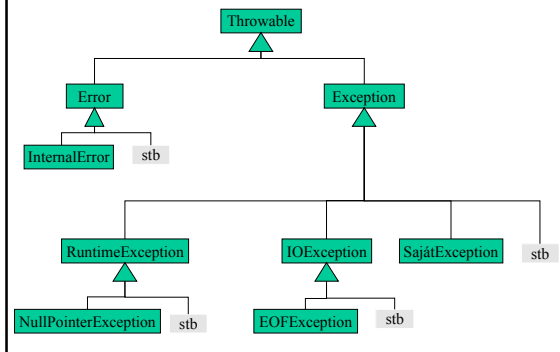
Programozói hibák

- A programozó általában úgy írja meg a programját, hogy vigyáz arra, hogy ne legyenek programozói hibák
- Sokszor felesleges hibakezelést betenni, vagy specifikálni a továbbterjedést
- Persze megengedett mind a lekezelés, mind a továbbterjedés specifikációja

Java tutorial

Copyright © 2000-2001, Kozsik Tamás

A kivételosztályok hierarchiája



Az Error leszármazottjai

- Fatális hibák: már nincs mit tenni...
- Nem kötelező lekezelni vagy a terjedést specifikálni
- Például:
 - OutOfMemoryError ClassFormatError
 - InstantiationError LinkageError
 - NoClassDefFoundError VirtualMachineError
 - StackOverflowError

A RuntimeException leszármazottai

- Az előbb már beszéltünk róluk
- Programozói hibát jeleznek
- NullPointerException
ArrayIndexOutOfBoundsException
ArithmeticException

Az Exception egyéb leszármazottai

- Ezekből van a legtöbb
- Kivételes esemény
- Vagy lekezeljük, vagy specifikáljuk a terjedésüket
- Például
 - IOException, FileNotFoundException, InterruptedException, SQLException

Java tutorial

Copyright © 2000-2001, Kozsik Tamás

Saját kivételosztályok

- Saját kivételes események jelzése
- Nagy divat...
- Célszerű az Exception osztályból leszármaztatni
 - ne a RuntimeException osztályból
- Sima osztálydefiníció
- Eltávolíthatunk egy kivételben információt a fellépés okáról

Példa

```
public class VeremMegteltException
    extends Exception {
    public VeremMegteltException(){ super(); }
    public VeremMegteltException( String s ){
        super(s);
    }
    public Object nemFértBele;
    public VeremMegteltException( Object o ){
        nemFértBele = o;
    }
}
```

Kivétel kiváltása

- A saját kivételeinket mi magunk válthatjuk ki, jelezve a kivételes esemény bekövetkezését
- A predefinit kivételeket is kiválthatjuk, sőt, akár még Error-okat is
- A throw kulcsszót kell használni, és utána megadni egy kivétel példányt

Példa

```
public void push( Object o )
    throws VeremMegteltException
{
    if( tele() )
        throw new VeremMegteltException(o);
    else
        ...
}
```

Példa (tömbös ábrázolás)

```
public void push( Object o )
    throws VeremMegteltException
{
    try {
        elemek[veremtető] = o;
        veremtető ++;
    } catch( ArrayIndexOutOfBoundsException e ){
        throw new VeremMegteltException(o);
    }
}
```

Feladat

- A Sor osztály kiegészítése saját kivételosztályokkal és kivételkezeléssel
- A mátrixösszeadásos program kiegészítése saját kivételosztállyal és kivételkezeléssel

Java tutorial

Copyright © 2000-2001, Kozsik Tamás

Kivétel újrakiváltása

- Lehet, hogy egy ponton még nem tudunk teljesen lekezelni egy kivételt
- Tovább is adjuk a hívónak
- Nem hozunk létre új kivételpéldányt
- és a fillInStackTrace

Példa

```
try {  
    ...  
} catch ( Exception e ){  
    log.println(e);  
    throw e;  
}
```

- Ilyenkor látszik, hogy a kivétel nem itt keletkezett
- A printStackTrace() kimutatja

Példa

```
class A {  
  
    public static void main( String args[] ){  
        első();  
    }  
  
    static void első(){ második(); }  
  
    static void második() {  
        throw new NullPointerException();  
    }  
}  
Exception in thread "main" java.lang.NullPointerException  
    at A.második(A.java:10)  
    at A.első(A.java:7)  
    at A.main(A.java:4)
```

Példa

```
class A {  
  
    public static void main( String args[] ){  
        try{ első(); }  
        catch( Exception e ){ System.err.println(e); }  
    }  
  
    static void első(){ második(); }  
  
    static void második() {  
        throw new NullPointerException();  
    }  
}  
  
java.lang.NullPointerException
```

Java tutorial

Copyright © 2000-2001, Kozsik Tamás

Példa

```
class A {  
  
    public static void main( String args[] ){  
        try{ első(); }  
        catch( Exception e ){ e.printStackTrace(); }  
    }  
  
    static void első(){ második(); }  
  
    static void második() {  
        throw new NullPointerException();  
    }  
}  
Exception in thread "main" java.lang.NullPointerException  
    at A.második(A.java:15)  
    at A.első(A.java:10)  
    at A.main(A.java:4)
```

Példa

```
class A {
    public static void main( String args[] ){ első(); }

    static void első(){
        try { második(); }
        catch( NullPointerException e ){ throw e; }
    }

    static void második() {
        throw new NullPointerException();
    }
}
Exception in thread "main" java.lang.NullPointerException
    at A.második(A.java:10)
    at A.első(A.java:5)
    at A.main(A.java:2)
```

Példa

```
class A {
    public static void main( String args[] ){ első(); }

    static void első(){
        try { második(); }
        catch( NullPointerException e )
            { e.fillInStackTrace(); throw e; }
    }

    static void második() {
        throw new NullPointerException();
    }
}
Exception in thread "main" java.lang.NullPointerException
    at A.első(A.java:7)
    at A.main(A.java:2)
```

Feladat

- Az osztásos példában dobjuk el újra a kivételt, ha a 0/0 eset van.

Polimorfizmus és kivételek

- Metódus felüldefiniálásakor az új metódus által kiváltható kivételek csak specifikusabbak lehetnek
- Azaz a leszármazott metódusa nem válthat ki több kivételt, mint az őse metódusa
- Kivétel elmaradhat, vagy lehet helyette specifikusabb (leszármazott kivétel)

Példa

```
class A {
    public void m(int i)
        throws IOException, InterruptedException { ... }
}

class B extends A {
    public void m(int i)
        throws EOFException { ... }
}
```

Java tutorial

Copyright © 2000-2001, Kozsik Tamás