Objektumorientált programozás C# nyelven III.

Kivételkezelés
Tulajdonságok
Feladatok

Készítette:
Miklós Árpád
Dr. Kotsis Domokos

Hallgatói tájékoztató

A jelen bemutatóban található adatok, tudnivalók és információk a számonkérendő anyag vázlatát képezik. Ismeretük szükséges, de nem elégséges feltétele a sikeres zárthelyinek, illetve vizsgának.

Sikeres zárthelyihez, illetve vizsgához a jelen bemutató tartalmán felül a kötelező irodalomként megjelölt anyag, a gyakorlatokon szóban, illetve a táblán átadott tudnivalók ismerete, valamint a gyakorlatokon megoldott példák és az otthoni feldolgozás céljából kiadott feladatok önálló megoldásának képessége is szükséges.

Kivételkezelés (1)

- A kivételek egyrészt nem várt, rendkívüli események, másrészt a kezelésükre szolgáló speciális osztályok
  - Segítségükkel a hibakezelés elkülöníthető a program logikájától és egységes, strukturált módon kezelhetők a felmerülő események
- A kivételkezeléssel ellátott programrészt háromféle típusú blokkra osztjuk fel:
  - A `try` kulcsszóval jelölt blokk tartalmazza a „normál” programot
  - A `catch` kulcsszóval jelölt blokkok a kivételkezelő kódot tartalmazzák
    - Egy „try” blokkhoz egynél több „catch” blokk is megadható az egyes kivételtípusok különálló kezelésére (az ún. univerzális „catch” blokk viszont minden kivételt kezel)
    - Használata nem kötelező
  - A `finally` kulcsszóval jelölt blokkok olyan kódot tartalmaznak, amelynek mindenképpen le kell futnia, attól függetlenül, hogy a „normál” programrészben történt-e rendkívüli esemény vagy sem
    - Használata nem kötelező

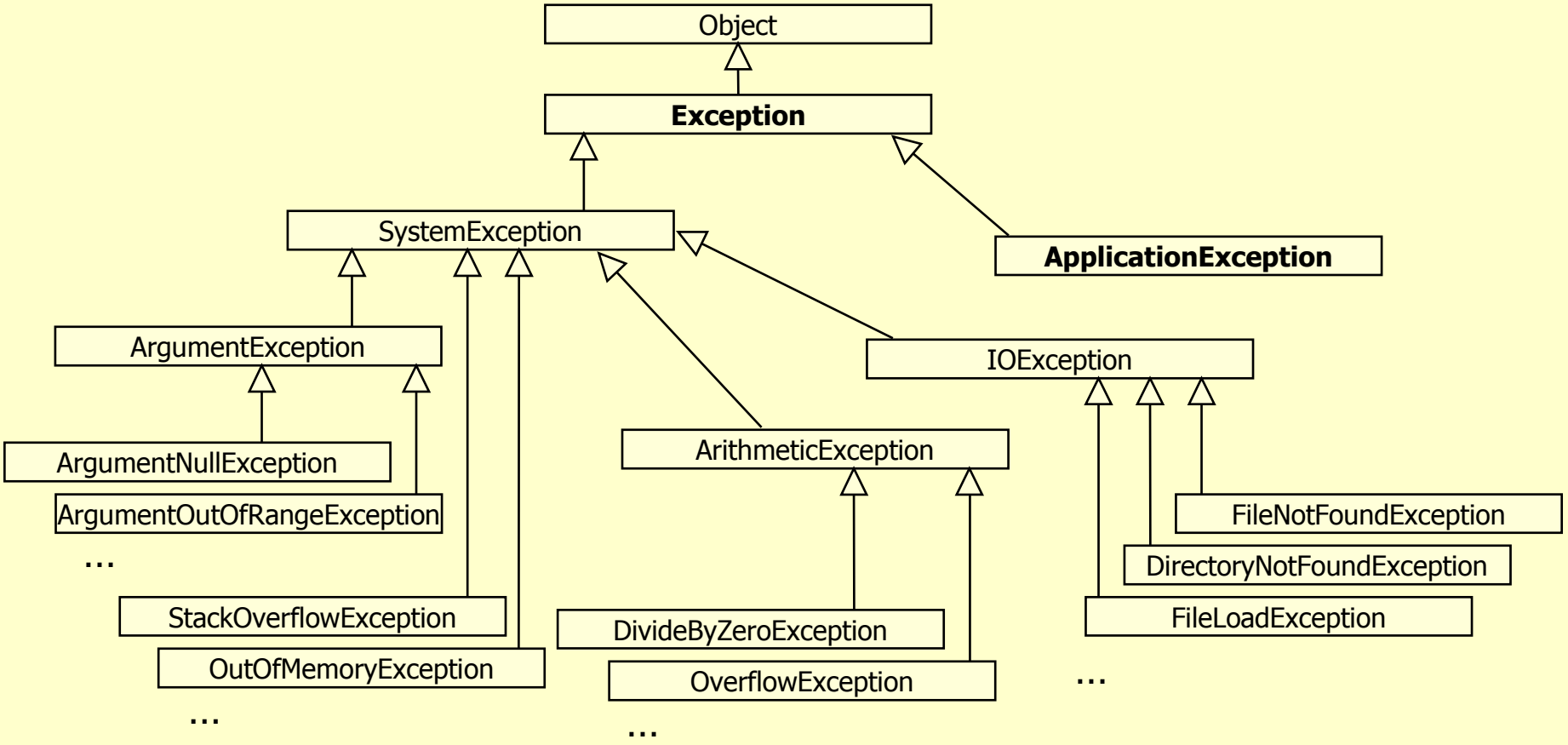
- Hiba esetén a throw kulcsszóval hozhatunk létre kivételt

Kivételkezelés (2)

- A kivételkezelés menete:
  - Egy „try” blokkon belül hiba következik be, melynek hatására létrejön a megfelelő kivételosztály egy példánya
  - A program nem hajtja végre a „try” blokk hátralévő utasításait, hanem megkeresi az első olyan „catch” blokkot, amely képes kezelni a bekövetkezett kivétel típusát
    - Ha nem talál ilyet az adott metódusban, akkor először végrehajtja a „try” blokkhoz tartozó „finally” blokk utasításait, majd folytatja a keresést a metódust eredetileg hívó metódusban, és így tovább
    - Ha megtalálta a megfelelő „catch” blokkot, akkor végrehajtja
    - Ha a programban sehol sem talál megfelelő „catch” blokkot, akkor a program leáll
      - A futtatókörnyezet minden programot egy univerzális „catch” blokkal ellátott „try” blokkban hajt végre, így akkor is lekezeli a bekövetkező kivételt, ha a program nem (de a program így sem folytatódhat)
  - A megfelelő „catch” blokk végrehajtása után folytatódik a végrehajtás
    - Ha létezik a megtalált „catch” blokkhoz tartozó „finally” blokk, akkor ennek az elején
    - Ha nem létezik, akkor a megtalált „catch” blokkhoz tartozó „try” blokk vége után
  - Destruktor végrehajtása közben keletkező kezeletlen kivételek
    - Ha az ősosztálynak van destruktora, akkor meghívódik; ha nincs, akkor a kivétel nyom nélkül megszűnik

Kivételkezelés (3)

- A futtatókörnyezet sok kivételosztályt előre definiál
  - Ezekből származtathatunk saját kivételeket
    - Származtatáshoz az ajánlott ősosztály a System.ApplicationException



Kivételkezelés (példa)

```
class Matek
{
    public static string Maradékképzés(int x, int y)
    {
        try
        {
                int z = x % y;
                return z.ToString();
        }
        catch (System.DivideByZeroException e)
        {
                return e.TargetSite + " Nullával sajnos nem lehet osztani.";
        }
        catch (System.Exception e)
        {
                return e.Source + " Ismeretlen hiba történt.";
        }
        finally
        {
        }
    }
}
```

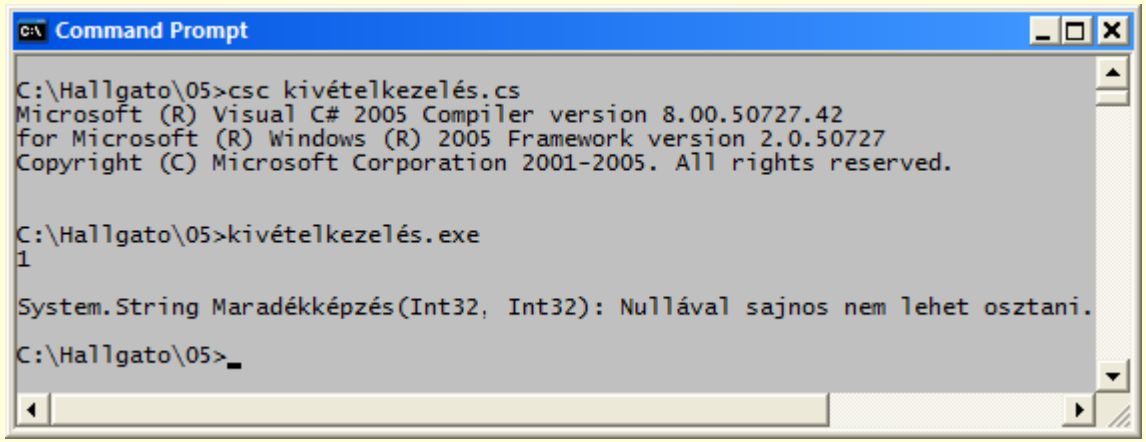
...

Itt olyan kódot helyezhetnénk el, amelynek még hiba esetén is mindenképpen le kell futnia

Kivételkezelés (példa)

```
...
class Kivételkezelő
{
    public static void Main()
    {
        System.Console.WriteLine(Matek.Maradékképzés(5, 1));
        System.Console.ReadLine();

        System.Console.WriteLine(Matek.Maradékképzés(9, 0));
    }
}
```



Kivételkezelés I.

Módosítsuk az 5. témakörben készített „számológép”-e „Main” metódusát úgy, hogy a program csak szám értékű operandust fogadjon el. Hiba esetén kérjen be másik operandust.

A „Main” módosítása I.

```
.
.
elso:System.Console.Write("Első operandus: ");
 try
{x=float.Parse(System.Console.ReadLine());}
 catch (System.Exception)
 {System.Console.WriteLine("Hibás operandus!");
  goto elso;}
.
.
```

Értelemszerűen a másik operandus bekérése hasonlóan történhet.

A „Main” módosítása II.

```
bool ujra=true;
...
while (ujra)
 {System.Console.Write("Első operandus: ");
 try
  {x=float.Parse(System.Console.ReadLine());
   ujra=false;
   }
  catch (System.Exception)
  {System.Console.WriteLine("Hibás operandus");
  ujra=true;
 }
}
...
```

Kivételkezelés II.

Módosítsuk a programot úgy, hogy a „Kalk” metódus észlelje a nem megfelelő műveleti kódot, ilyenkor adjon hibajelzést, és a „Main” metódus se írjon ki eredményt.

A Számoló, Számolgató módosítása

```
public bool hiba;
...
protected virtual float Kalk()
 {try{
  switch (művelet)
  {...
  break;
  default: throw new System.Exception();
  }}
  catch(System.Exception)
  {System.Console.WriteLine("Hibás kód!");
   hiba =true;}
  return eredmény;
 }
```

A Számoló, Számolgató módosítása

```
public bool hiba;
...
protected virtual float Kalk()
switch (művelet)
  {...
  break;
  default: hiba =true;}
  return eredmény;
 }
```

Ugyanez throw használatával

```
public bool hiba;
...
protected virtual float Kalk()
 {try{
  switch (művelet)
  {...
  break;
  default: throw new System.Exception();
  }}
  catch(System.Exception)
  {System.Console.WriteLine("Hibás kód!");
   hiba =true;}
  return eredmény;
 }
```

A „Main” módosítása

```
float e;
...
Számolgató Kiszámol= new Számolgató(x,y,z);
Kiszámol.hiba=false;
 e = Kiszámol.Kalkuláló();
 if (Kiszámol.hiba)
   {System.Console.WriteLine("Nincs eredmény!");}
 else
   {System.Console.WriteLine(x+" "+z+" "+y+" = "+ e);}
System.Console.ReadLine();
...
```

Kivételkezelés Ia.

Módosítsuk a „Main” metódust úgy, hogy a program csak szám értékű operandust fogadjon el. Hiba esetén kérjen be másik operandust, de az ötödik alkalommal lépjen ki, és ekkor már egyéb adatot (műveleti kód, másik operandus) se kérjen be.

Deklaráció

```
...
float e=0;
 bool ujra=true;
 char z='?';
 int ismétlés=0;
...
```

Az első operandus bekérése

```
...
while (ujra && ismétlés<5)
 {System.Console.Write("Első operandus: ");
 try
  {x=float.Parse(System.Console.ReadLine());
   ujra=false;
   ismétlés=0;}
  catch (System.Exception)
  {System.Console.WriteLine("Hibás operandus");
   ujra=true;
   ismétlés++;}}
...
```

A műveleti jel bekérése

```
...
if (ismétlés<5)
   {System.Console.Write("Műveleti jel: ");
    z=char.Parse(System.Console.ReadLine());}
...
```

A második operandus bekérése

```
...
 ujra=true;
  if (ismétlés<5) ismétlés=0;
  while (ujra && ismétlés<5)
 {System.Console.Write("Második operandus: ");
  try
  {y=float.Parse(System.Console.ReadLine());
   ujra=false;}
  catch (System.Exception)
  {System.Console.WriteLine("Hibás operandus");
   ujra=true;
   ismétlés++;}}
...
```

Az eredmény kiszámítása

```
...
Számolgató Kiszámol= new Számolgató(x,y,z);
 if (ismétlés<5)
  {Kiszámol.hiba=false;
  e = Kiszámol.Kalkuláló();}
 if (Kiszámol.hiba || ismétlés>=5)
{System.Console.WriteLine("Nincs eredmény!");}
  else
{System.Console.WriteLine(x+" "+z+" "+y+" =
"+ e);}
...
```

Tulajdonságok (1)

- A tulajdonságok olyan „intelligens mezők”, amelyek olvasását és írását a programozó által megadott ún. hozzáférési metódusok végzik
  - Az osztályt felhasználó kód számára mezőnek tűnnek
  - Az osztály fejlesztője számára metóduspárként jelennek meg
- A hozzáférési metódusok bármilyen műveletet végrehajthatnak (pl. beírandó adat ellenőrzése, kiolvasandó adat kiszámítása…)
  - Nem célszerű hosszan tartó műveletekkel lelassítani a tulajdonság elérését
  - Kerülni kell a felhasználó kód számára váratlan mellékhatásokat

Tulajdonságok (2)

- Létrehozhatók csak olvasható vagy csak írható tulajdonságok is
  - Ha nem adjuk meg az írási, illetve olvasási metódust, akkor csak olvasható, illetve csak írható tulajdonság jön létre
- Az olvasási és írási metódushoz hozzáférési szint adható meg
  - Ezzel a módszerrel például létrehozhatók nyilvánosan olvasható, de csak a saját vagy a leszármazott osztályok által írható tulajdonságok
- Az olvasási metódust a `get`, az írási metódust a `set` kulcsszó segítségével kell megadnunk
  - A hozzáférési metódusokat a tulajdonság definícióján belül kell megadnunk

Tulajdonságok (példa)

```
using System;

class Személy
{
    private string vezetéknév = "<üres>";
    private string keresztnév  = "<üres>";

    public string Vezetéknév
    {
        get     { return vezetéknév; }
        set     { if (value != " ") vezetéknév = value; }
    }
    public string Keresztnév
    {
        get     { return keresztnév; }
        set     { if (value != " ") keresztnév = value; }
    }
    public string Név
    {
        get     { return vezetéknév + " "+ keresztnév; }
    }
}
```

Gyakori megoldás, hogy a nyilvános tulajdonság neve nagybetűs, a hozzá tartozó privát mező neve pedig ugyanaz kisbetűs változatban

A tulajdonság értékét a return kulcsszóval adjuk vissza a hívónak

A tulajdonság megadott új értékét mindig egy value nevű rejtett paraméter tartalmazza

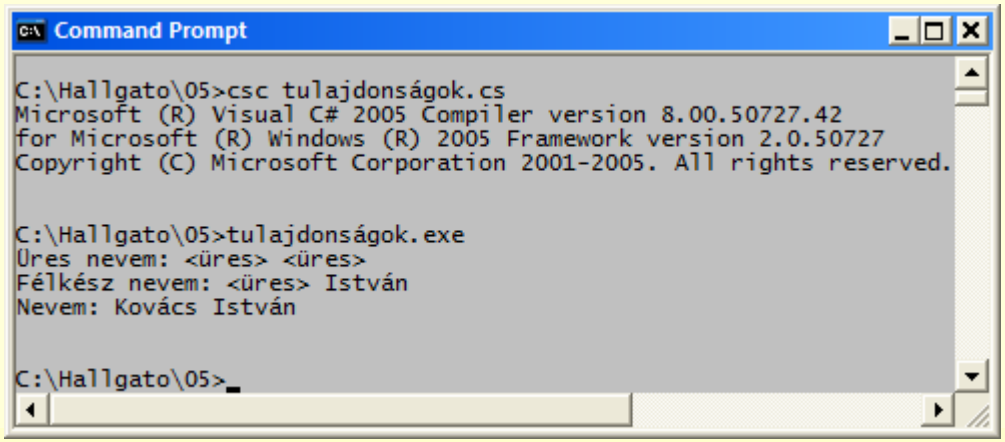
Nem adtunk meg set metódust, így ez a tulajdonság csak olvasható

Tulajdonságok (példa)

```
class Személykezelő
{
    static void Main()
    {
        Személy Pistike = new Személy();
        Console.WriteLine("Üres nevem: " + Pistike.Név);

        Pistike.Keresztnév = "István";
        Console.WriteLine("Félkész nevem: " + Pistike.Név);

        Pistike.Vezetéknév = "Kovács";
        Pistike.Keresztnév = "";
        Console.WriteLine("Nevem: " + Pistike.Név);
        Console.ReadLine();
    }
}
```



Feladat

Módosítsa a számológép programot úgy, hogy a „Számoló” osztálybeli operandusok és a műveleti kód tulajdonságok legyenek!

Számoló tulajdonsággal

```
class Számolo
{
   protected float op1, op2;
   protected char művelet;
   protected float eredmény;

   public float Op1
     { set { op1 = value; } }
   public float Op2
      {set { op2 = value; } }
   public char Művelet
      { set { művelet = value; }}
...
```

Feladat

Módosítsa a számológép programot úgy, hogy az operandusok szám mivoltának ellenőrzése a megfelelő tulajdonsághoz tartozó set metódusban történjék!

Feladat

Készítsen hallgatói nyilvántartást.

A „Hallgató” osztályban legyenek tulajdonságok:

Vnév, Knév, Szülidő, Szülhely.

Valamennyi string lehet, de a Szülidő formája 4+2+2 számjegy, tetszőleges karakterrel elválasztva.

Ellenőrizze a megfelelő „set” metódusban!

Feladat II.

Készítse el „Kollégista” osztályt, amely a „Hallgató” osztály örököse.

További tulajdonság:

Szobaszám.

Ehhez készítsen saját kivétel osztályt „Rosszoba” néven!

Saját kivétel osztály

```
class Rosszoba : ApplicationException
{ }
```

Feladat III.

Készítse el az „Kiskolesz” és a „Nagykolesz” osztályokat, amelyek a „Kollégista” osztály örökösei.

Módosított tulajdonság:

Szobaszám.

Ez „Kiskolesz” –nél 0-nál nagyobb és 100-nél kisebb szám, „Nagykolesz”-nél 0-nál nagyobb és 1000-nél kisebb szám lehet.

Ellenőrizze a megfelelő „set” metódusban!

Feladat IV.

Készítse el a „Kezelés” osztályt, amelyben egy menü van három menüponttal:

1. Felvitel
2. Kiírás
3. Kilépés

A felvitel előtt kérdezze meg, melyik kollégiumhoz tartozik az illető!

A példányokat (max 25) egy tömbben tárolja!

Házi feladat 1

Készítsünk egy háttértároló osztályt, amely fájlok listáját kezeli az alábbi módon:
-A háttértárolónak van egy maximális tárolókapacitása, melyet csak
-konstruktorban lehet beállítani
- Van egy Format() metódusa, mely üríti a fájlok listáját
-Van egy MaximálisKapacitás tulajdonsága, amellyel le lehet kérdezni a maximális kapacitás értékét
-Van egy SzabadKapacitás tulajdonsága, amellyel le lehet kérdezni a meghajtó szabad kapacitását
-Van egy FoglaltKapacitás tulajdonsága, amellyel le lehet kérdezni a tárolt
-fájlok összméretét
-Van egy Hozzáad() metódusa, amellyel új fájlt lehet hozzáadni, ha ugyanilyen nevű fájl nincs még a háttértárolón, illetve az új fájl elfér a háttértárolón
-Van egy Keres() metódusa, amely egy megadott fájlnév alapján megkeresi és visszaadja a fájlt
-Van egy Töröl() metódusa, amely letörli a megadott fájlt, amennyiben létezik
A fájlok jellemzői a következők:
- Van nevük és méretük
- Van egy-egy CsakOlvasható, Rendszer és Rejtett attribútumuk

Házi feladat 2

Fejlesszük tovább az alap háttértároló osztályt Floppy osztállyá az alábbi módosítások szerint:
- A floppy mérete 1440KB
-A floppy-nak van írásvédő tolókája, amely ha „írásvédett'”állapotba kerül, akkor a floppyn sem a Format(), sem a Hozzáad(), sem a Törlés() nem működik

Fejlesszük tovább az alap háttértároló osztályt DVD osztállyá az alábbi módosítások szerint:
- A DVD mérete 4700MB
-A DVD alapból még írható, törölhető, de miután meghívjuk a Zárolás() metódusát (DVD felírása), akkor utána már nem használható rajta sem a Format(), sem a Hozzáad(), sem a Töröl() („egyszer írható DVD”)
- A zárolt DVD szabad kapacitása mindig 0 legyen
Fejlesszük tovább a DVD osztályt DVD-RW osztállyá az alábbi módosítások szerint:
-A DVD-RW többször is írható DVD, ezért van egy Megnyitás() metódusa is.
-Ekkor a DVD lemez visszaáll egy üres alapállapotba.

Házi feladat 3

Fejlesszük tovább az alap Háttértároló osztályt HDD osztállyá a célnak megfelelő módosítások szerint.

Fejlesszünk ki egy Számítógép osztályt az alábbiak szerint:
-Több háttértárolója is lehet, melyek egy Felcsatol() metódus segítségével lehet csatolhatók a géphez
-Van egy Összkapacitás tulajdonsága, amely megadja az összes háttértároló teljes kapacitásának összegét
-Van egy SzabadKapacitás tulajdonsága, amely megadja az összes háttértároló teljes szabad kapacitásának összegét
-Van egy FoglaltKapacitás tulajdonsága, amely megadja az összes háttértároló teljes foglalt kapacitásának összegét
-Van egy Archivál() metódusa, mely a megadott fájlt megkeresi a háttértárolók valamelyikén, majd a megadott másik háttértárolóra a megfelelő módon archiválja a fájlt. Amennyiben nem adunk meg másik háttértárolót, a metódus keres egy másik használható háttértárolót és oda végzi el az archiválást.