Objektumorientált programozás C# nyelven II.

Öröklés és többalakúság
Nemvirtuális metódusok, elrejtés
Virtuális metódusok, elrejtés
Típuskényszerítés, az „is” és „as” operátorok
Absztrakt osztályok, absztrakt metódusok
Lezárt osztályok, lezárt metódusok

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.

Öröklés (1)

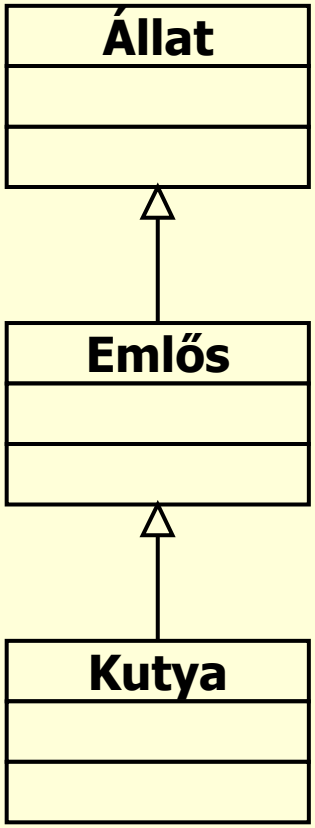
- Az osztályokból létrehozhatunk leszármazott osztályokat, amelyek öröklik az ősosztály összes tagját
  - Az örökölt metódusok a leszármazottakban módosíthatók
  - A leszármazottak új tagokkal (mezőkkel, metódusokkal) bővíthetik az ősosztálytól örökölt tagok halmazát
- Minden leszármazott osztálynak csak egy őse lehet
- Minden osztály közös őse a System.Object osztály

```
class Object
{
    public Object() {...}
    public Type GetType() {...}
    protected object MemberwiseClone() {...}
    public static bool ReferenceEquals(object objA, object objB) {...}
    public static bool Equals(object objA, object objB) {...}
    public virtual bool Equals(object obj) {...}
    public virtual int GetHashCode() {...}
    public virtual string ToString() {...}
}
```

Öröklés (2)

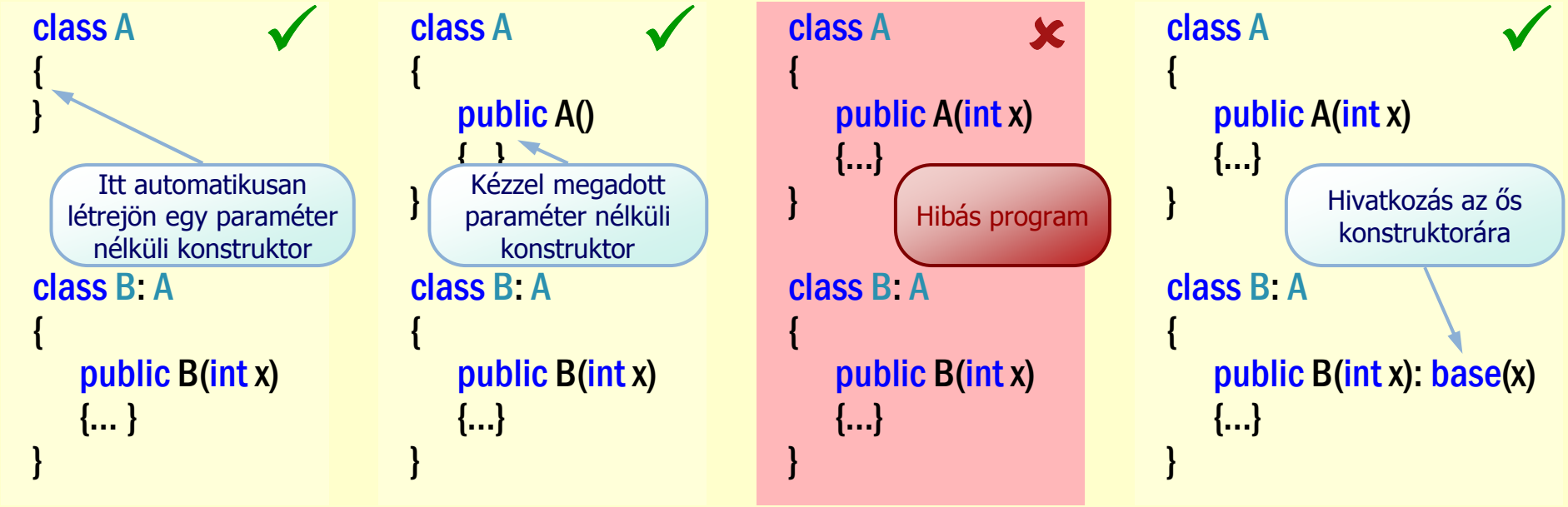
- A leszármazott osztályok deklarációjánál „ : ” karakterrel elválasztva meg kell adnunk az ősosztály nevét is
  - Ez nem kötelező abban az esetben, ha az ősosztály a System.Object

```
class Állat
{
    int lábszám;
    public Állat() {...}
    public void Fut() {...}
}

class Emlős: Állat
{
    public bool KicsinyétEteti() {...}
}

class Kutya: Emlős
{
    public void FarkátCsóválja() {...}
    public void Ugat() {...}
}
```



Öröklés (3)

- A konstruktorok nem öröklődnek
  - Ha az ősosztályban van paraméter nélküli konstruktor, az a leszármazott osztály konstruktorában automatikusan meghívódik
    - Ez egyaránt igaz az automatikusan generált alapértelmezett konstruktorra és a saját, paraméter nélküli konstruktorra
  - Ha az ősosztályban nincs paraméter nélküli konstruktor, az ősosztály konstruktorát meg kell hívni a leszármazott osztály konstruktorából
    - Erre a célra a **base** kulcsszó áll rendelkezésre



Nemvirtuális metódusok

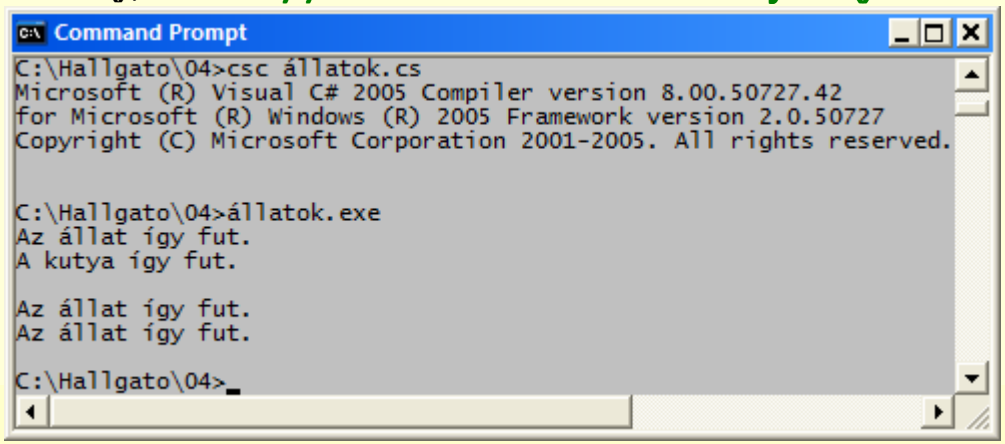
- A nemvirtuális metódusok változatlanul örökölhetők vagy a leszármazottakban elrejthetők
  - Statikus (fordítási idejű vagy „korai”) kötés jellemzi őket
    - A saját osztályuknak megfelelő típusú változókon keresztül hívhatók
    - Fordítási időben dől el, hogy az őket (adott típusú változókon keresztül) felhasználó programkód melyik osztályhoz tartozó nemvirtuális metódust hívja
  - Alapértelmezésben minden metódus nemvirtuális
- Nem igényelnek semmilyen külön szintaktikai megjelölést
- Elrejtés: a leszármazott osztályban azonos néven létrehozunk egy másik metódust
  - A leszármazott osztályban célszerű az újonnan bevezetett metódust a `new` kulcsszóval megjelölni
    - Bár ez nem kötelező, ha nem tesszük meg, a C# fordító figyelmeztet rá
  - Az ősosztály azonos nevű metódusa a leszármazottban is elérhető a `base` kulcsszó segítségével

Nemvirtuális metódusok (példa)

```
using System;
class Állat
{
    public void Fut()
    {
        Console.WriteLine("Az állat így fut.");
    }
}
class Kutya: Állat
{
    new public void Fut()
    {
        Console.WriteLine("A kutya így fut.");
    }
}
class Macska: Állat
{
    new public void Fut()
    {
        Console.WriteLine("A macska így fut.");
    }
}
```

Nemvirtuális metódusok (példa)

```
class NemvirtuálisMetódusok
{
    static void Main()
    {
        Állat egyikállat = new Állat();
        Kutya másikállat = new Kutya();
        egyikállat.Fut();           // Az Állat osztály Fut() metódusa hívódik meg
        másikállat.Fut();           // A Kutya osztály Fut() metódusa hívódik meg
        Console.ReadLine();

        Állat házikedvenc;
        házikedvenc = new Macska();
        házikedvenc.Fut();          // Az Állat osztály Fut() metódusa hívódik meg (!)
        házikedvenc = new Kutya();
        házikedvenc.Fut();          // ...és ismét az Állat osztály Fut() metódusa hívódik meg ☹
    }
}
```



Virtuális metódusok

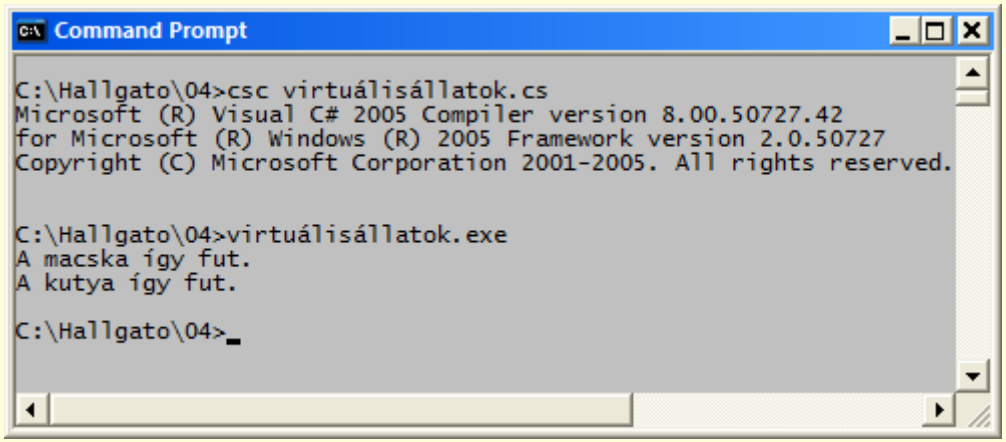
- A virtuális metódusok a leszármazottakban módosíthatók („felülbírálhatók”) vagy elrejthetők
  - Dinamikus (futási idejű vagy más szóval „késői”) kötés jellemzi őket
    - Segítségükkel valódi többalakúság valósítható meg
    - Saját vagy bármely ősosztályuknak megfelelő típusú változókon keresztül hívhatók
    - Futási időben, az adott változó típusától függően dől el, hogy az őket felhasználó programkód melyik osztályhoz tartozó virtuális metódust hívja
      - Hívási szabály: egy adott virtuális metódusból mindig a változó által ténylegesen hivatkozott osztályhoz legközelebb álló változatot hívja meg a program; a legtöbb esetben ez az osztály saját metódusváltozata
- Külön szintaktikai megjelölések tartoznak hozzájuk
  - A virtuális metódusokat az ősosztályban a **virtual** kulcsszóval kell megjelölnünk
  - A leszármazottakban felülbírált virtuális metódusokat az **override** kulcsszóval kell megjelölnünk
- Elrejtés: mint a nemvirtuális metódusok esetén
  - Ha a leszármazottban azonos néven létrehozott új metódus szintén virtuális, akkor ezzel új virtuális hívási láncot hozhatunk létre

Virtuális metódusok (példa)

```
using System;
class Állat
{
    public virtual void Fut()
    {
        Console.WriteLine("Az állat így fut.");
    }
}
class Kutya: Állat
{
    public override void Fut()
    {
        Console.WriteLine("A kutya így fut.");
    }
}
class Macska: Állat
{
    public override void Fut()
    {
        Console.WriteLine("A macska így fut.");
    }
} ...
```

Virtuális metódusok (példa)

```
...
class VirtuálisMetódusok
{
    static void Main()
    {
        Állat házikedvenc;
        házikedvenc = new Macska();
        házikedvenc.Fut();
        házikedvenc = new Kutya();
        házikedvenc.Fut();
    }
}
```



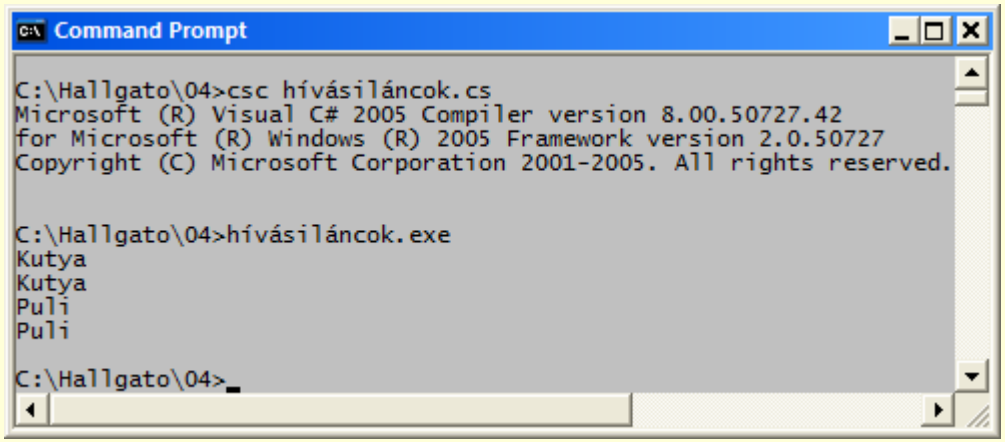
Virtuális hívási láncok (példa)

```
using System;

class Állat
{
    public virtual void MiVagyokÉn()      { Console.WriteLine("Állat"); }
}

class Kutya: Állat
{
    public override void MiVagyokÉn()   { Console.WriteLine("Kutya"); }
}

class Terelőkutya: Kutya
{
    public new virtual void MiVagyokÉn()        { Console.WriteLine("Terelőkutya"); }
}

class Puli: Terelőkutya
{
    public override void MiVagyokÉn()   { Console.WriteLine("Puli"); }
}
```

...

Virtuális hívási láncok (példa)

```
...
class VirtuálisHívásiLáncok
{
    static void Main()
    {
        Puli loncsoska = new Puli();
        Terelőkutya t = loncsoska;
        Kutya k = loncsoska;
        Állat á = loncsoska;

        á.MiVagyokÉn();
        k.MiVagyokÉn();
        t.MiVagyokÉn();
        loncsoska.MiVagyokÉn();
    }
}
```



Típuskényszerítés

- Típuskényszerítésnél („casting”) egy adott típusú objektumot úgy kezelünk, mintha egy másik típusba tartozna
  - Implicit típuskényszerítés: a típusok átalakítása automatikus
    - Példa: egész számok átalakítása valós számmá
  - Explicit típuskényszerítés: átalakítás a programozó kérésére
    - Módja: az átalakítandó típus elé, „ () ” karakterek közé kiírjuk a kívánt céltípust
  - Később részletesebben tárgyaljuk

```
Állat Cirmos = new Macska();
Állat amőba = new Állat();
Macska Lukrécia;
Kutya Bodri;

Lukrécia = (Macska) Cirmos;

Bodri = (Kutya) Lukrécia;

Lukrécia = (Macska) amőba;
```

Implicit típusátalakítás („Állat” helyén mindig szerepelhet „Macska”)

Explicit típusátalakítás a programozó kérésére (helyesen, mert erről az „Állat”-ról biztosan tudjuk, hogy „Macska”)

Fordítási hiba: a „Macska” típus nem alakítható át a „Kutya” típusra

Futási idejű hiba: „Macska” helyén nem szerepelhet „Állat”

Az is és as operátorok

- Az is operátor segítségével ellenőrizhető, hogy egy objektum egy adott osztályhoz vagy leszármazottjához tartozik-e
  - Ez az ellenőrző kifejezés logikai típusú értéket ad vissza
- Az as operátor segítségével explicit típusátalakítást hajthatunk végre futási idejű hiba veszélye nélkül
  - Ha az átalakítás nem sikerül, a kifejezés értéke **null** lesz

```
class Állatfarm
{
    Állat Cirmos = new Macska();
    Állat amőba = new Állat();
    Macska Lukrécia;
    Kutya Bodri;
    Lukrécia = Cirmos as Macska;
    if (amőba is Kutya)
        Bodri = amőba as Kutya;
    Lukrécia = amőba as Macska;
}
```

A típusátalakítás sikerülni fog (Cirmos értéke „Macska” típusú)

A típusátalakításra nem kerül sor, mert amőba értéke nem „Kutya” típusú, így már a feltétel sem teljesül

A típusátalakítás nem fog sikerülni („Macska” helyén nem szerepelhet „Állat”)

Absztrakt osztályok és metódusok

- Az absztrakt metódusok nem tartalmaznak megvalósítást
- Egy osztály akkor absztrakt, ha tartalmaz legalább egy absztrakt metódust
- Az absztrakt osztályok nem példányosíthatók
  - Absztrakt metódusaikat leszármazottaik kötelesek felülbírálni, azaz megvalósítást készíteni hozzájuk
  - Az absztrakt metódusok mindig virtuálisak (ezt nem kell külön jelölnünk)
- Az absztrakt osztályok garantálják, hogy leszármazottaik tartalmazni fognak bizonyos funkciókat
- Az absztrakt metódusokat és osztályok az **abstract** kulcsszóval kell megjelölnünk

Absztrakt osztályok (példa)

```
abstract class Alakzat
{
    public abstract void Kirajzol();
}

class Ellipszis: Alakzat
{
    public override void Kirajzol()
    {
        // Kirajzol() metódus az Ellipszis osztály megvalósításában
    }
}

class Kör: Ellipszis
{
    public override void Kirajzol();
    {
        // Kirajzol() metódus a Kör osztály megvalósításában
    }
}
```

Ebből az osztályból példányt nem hozhatunk létre, leszármazottai viszont biztosan tartalmaznak egy megvalósított Kirajzol() nevű metódust

Lezárt osztályok és metódusok

- Lezárt osztályból nem származtatható másik osztály
- Lezárt metódus leszármazottakban nem bírálható felül
  - Metódusok lezárásának csak felülbírált, eredetileg valamelyik ős által definiált virtuális metódusoknál van értelme
- A lezárt osztályok és lezárt metódusok célja az öröklés megakadályozása
  - Lehetséges indokai:
    - Előre nem látható célú felhasználás (és a vele járó karbantartási, illetve támogatási problémák) elkerülése
    - Teljesítmény optimalizálása
      - Csak osztályszintű tagokat tartalmazó osztályok
      - Biztosra vehető, hogy nem lesznek leszármazottak, így a virtuális metódusok nemvirtuálisra cserélhetők
    - Szerzői jogok védelme
- A lezárt osztályokat és metódusokat a **sealed** kulcsszóval kell megjelölnünk
  - Erős korlátozást jelentenek a fejlesztés során

Feladat: nemvirtuális metódusok

Készítsen Oktató osztályt, melynek az örökösök által is elérhető adattételei: „kernév” és „veznév” stringek, melyeket a kostruktor tölt fel! Az osztály tartalmazzon egy „névki” nevű nyilvános eljárást, mely a teljes nevet a képernyőre írja!

Készítse el az Oktató osztály örökösét a Főállású osztályt! Ez tartalmazzon egy nyilvános előjel nélküli short adattételt „alkév” néven! Az újra definiált „névki metódus ennek értékét is írja ki a képernyőre!

Oktató osztály

```
class Oktató
   {
      protected string veznév, kernév;
      public Oktató(string vn, string kn)
      {
      veznév = vn;
      kernév = kn;
      }
      public void névki()
      {
         Console.WriteLine(veznév+" "+kernév);
      }
   }
```

Főállású

```
class Főállású:Oktató
   {
      public ushort alkév;
      public Főállású(string vn, string kn): base(vn,kn)
      {}
      public Főállású(string vn, string kn, ushort aé): base(vn,kn)
      {
         alkév = aé;
      }
      public new void névki()
      {
         Console.WriteLine(veznév + " " + kernév+" "+alkév);
      }
   }
```

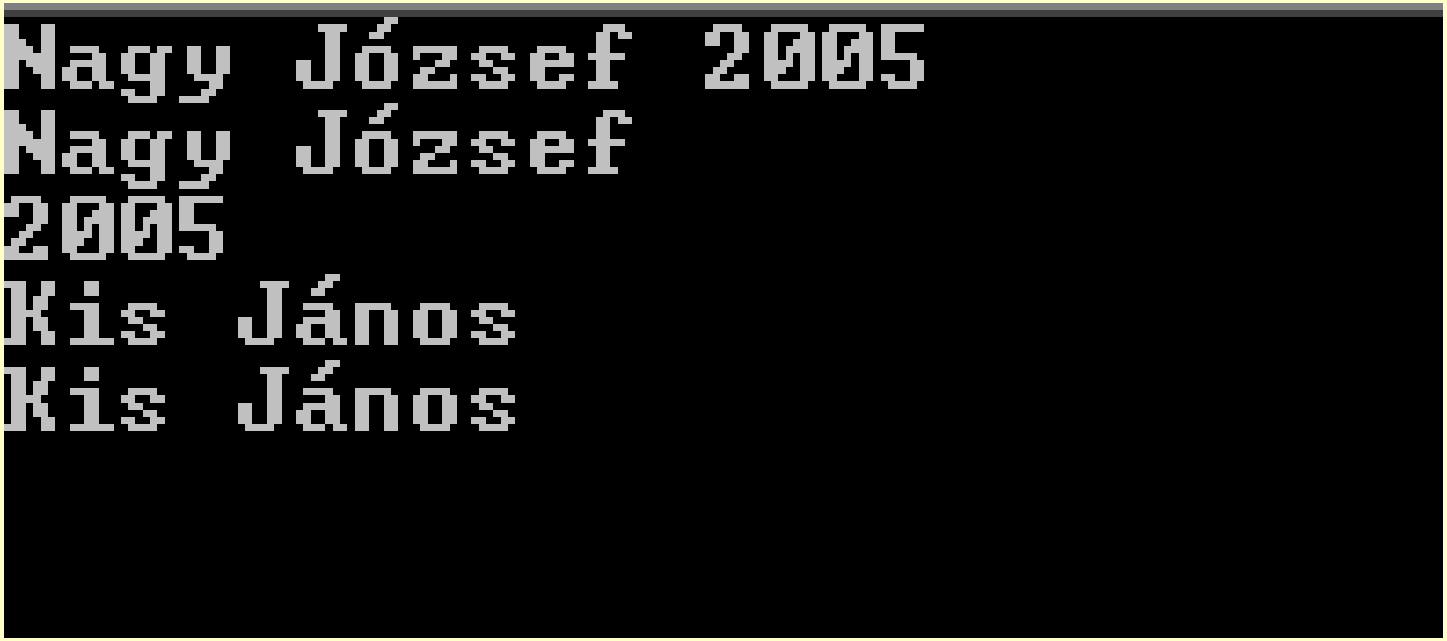
Főprogram: melyik sor hibás?

```
class Próba
   {
      public static void Main()
      {
         Főállású Józsi = new Főállású("Nagy", "József", 2005);
         Oktató Nagy = Józsi;
         Józsi.névki();
         Nagy.névki();
         Console.WriteLine(Józsi.alkév);
         Console.WriteLine(Nagy.alkév);
         Oktató Kis = new Főállású("Kis", "János");
         Kis.névki();
         Console.WriteLine(Kis.alkév);
         Oktató János = new Főállású("Kis", "János", 2007);
         János.névki();
         Console.WriteLine(János.alkév);
         Console.ReadLine();
      }
   }
```

Főprogram: mit ír ki?

```
class Próba
   {
      public static void Main()
      {
         Főállású Józsi = new Főállású("Nagy", "József", 2005);
         Oktató Nagy = Józsi;
         Józsi.névki();
         Nagy.névki();
         Console.WriteLine(Józsi.alkév);
         //Console.WriteLine(Nagy.alkév);
         Oktató Kis = new Főállású("Kis", "János");
        Kis.névki();
         //Console.WriteLine(Kis.alkév);
        Oktató János = new Főállású("Kis", "János", 2007);
        János.névki();
         //Console.WriteLine(János.alkév);
         Console.ReadLine();
      }
   }
```

Eredmény



Feladat: virtuális metódusok

Az Oktató osztály „névki” metódusát tegye virtuálissá!

A Főállású osztály „névki” metódusa legyen „override”!

Készítse el az Óraadó osztályt, mely ugyancsak az Oktató osztály örököse, új adattétele a nyilvános string típusú „cég”.

A „névki” metódus a teljes néven kívül írja ki a „cég” tartalmát is, de ne legyen „override” (azaz „new” legyen)!

Oktató osztály

```
class Oktató
   {
      protected string veznév, kernév;
      public Oktató(string vn, string kn)
      {
      veznév = vn;
      kernév = kn;
      }
      public virtual void névki()
      {
         Console.WriteLine(veznév+" "+kernév);
      }
   }
```

Főállású

```
class Főállású:Oktató
   {
      public ushort alkév;
      public Főállású(string vn, string kn): base(vn,kn)
      {}
      public Főállású(string vn, string kn, ushort aé): base(vn,kn)
      {
         alkév = aé;
      }
      public override void névki()
      {
         Console.WriteLine(veznév + " " + kernév+" "+alkév);
      }
   }
```

Óraadó

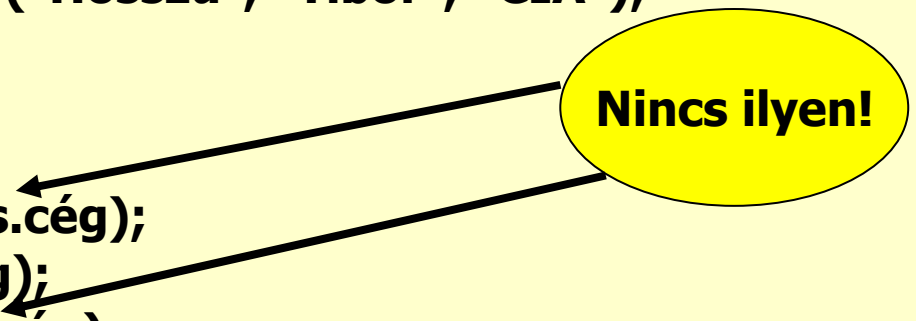
```
class Óraadó: Oktató
   {
      public string cég;
      public Óraadó(string vn, string kn)
         : base(vn, kn)
      { }
      public Óraadó(string vn, string kn, string cg)
         : base(vn, kn)
      {
         cég = cg;
      }
      public new void névki()
      {
         Console.WriteLine(veznév + " " + kernév + " " + cég);
      }
   }
```

Főprogram: melyik sor hibás?

```
public static void Main()
      {  Főállású Józsi = new Főállású("Nagy", "József", 2005);
         Oktató Nagy = Józsi;
         Józsi.névki();
         Nagy.névki();
         Console.WriteLine(Józsi.alkév);
         Oktató Kis = new Főállású("Kis", "János");
         Kis.névki();
         Oktató János = new Főállású("Kis", "János", 2007);
         János.névki();
         Óraadó Marcsi = new Óraadó("Közepes", "Marcsi", "IBM");
         Oktató Közepes = Marcsi;
         Oktató Hosszú = new Óraadó("Hosszú", "Tibor", "CIA");
         Marcsi.névki();
         Közepes.névki();
         Hosszú.névki();
         Console.WriteLine(Közepes.cég);
         Console.WriteLine(Marcsi.cég);
         Console.WriteLine(Hosszú.cég);
         Console.ReadLine();       }
```

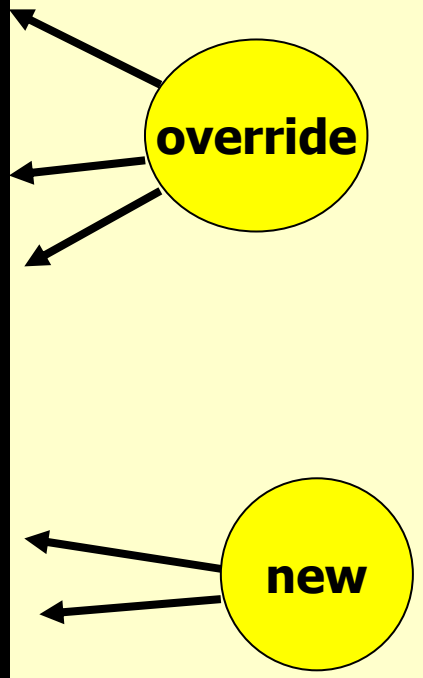
Főprogram: mit ír ki?

```
public static void Main()
{  Főállású Józsi = new Főállású("Nagy", "József", 2005);
   Oktató Nagy = Józsi;
   Józsi.névki();
   Nagy.névki();
   Console.WriteLine(Józsi.alkév);
   Oktató Kis = new Főállású("Kis", "János");
   Kis.névki();
   Oktató János = new Főállású("Kis", "János", 2007);
   János.névki();
   Óraadó Marcsi = new Óraadó("Közepes", "Marcsi", "IBM");
   Oktató Közepes = Marcsi;
   Oktató Hosszú = new Óraadó("Hosszú", "Tibor", "CIA");
   Marcsi.névki();
   Közepes.névki();
   Hosszú.névki();
   //Console.WriteLine(Közepes.cég);
   Console.WriteLine(Marcsi.cég);
   //Console.WriteLine(Hosszú.cég);
   Console.ReadLine();      }
```



Eredmény

```
Nagy József 2005
Nagy József 2005
2005
Kis János 0
Kis János 2007
Közepes Marcsi IBM
Közepes Marcsi
Hosszú Tibor
IBM
```

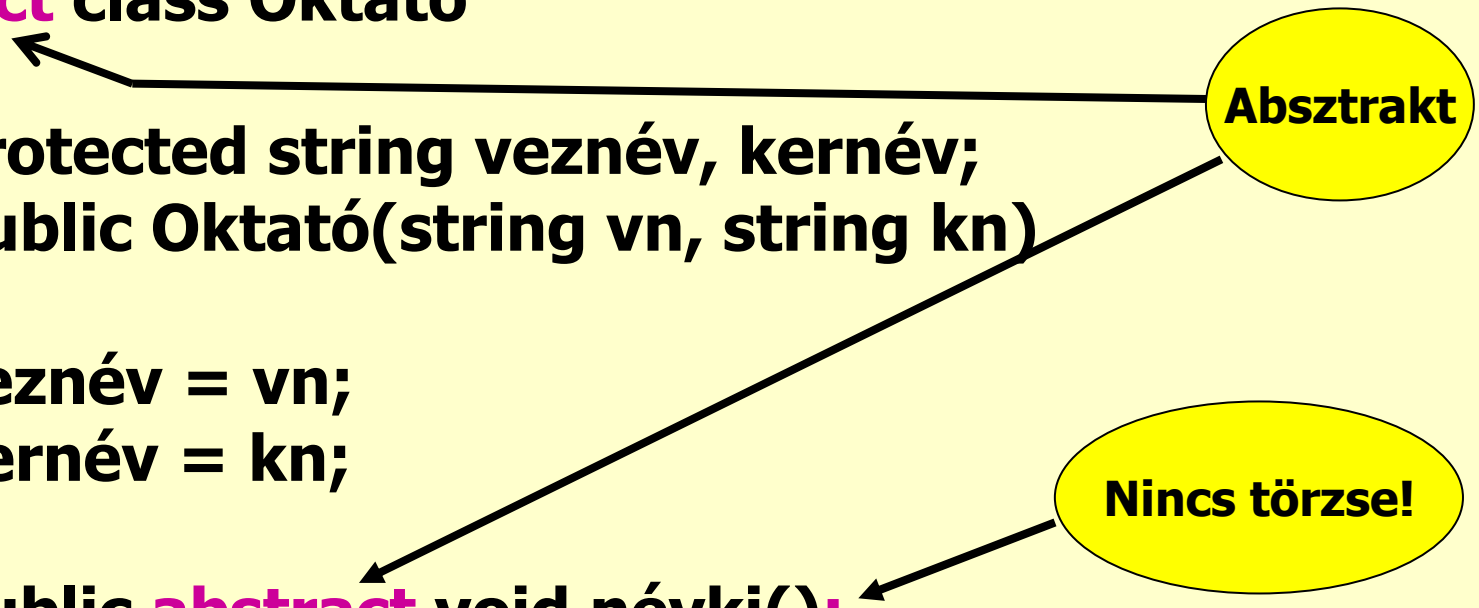


Feladat: absztrakt metódus és osztály

Legyen az előbbi feladatban a „névki” metódus és így az Oktató osztály absztrakt!

Oktató osztály

```
abstract class Oktató
   {
      protected string veznév, kernév;
      public Oktató(string vn, string kn)
      {
      veznév = vn;
      kernév = kn;
      }
      public abstract void névki();
    }
```



Főállású

```
class Főállású:Oktató
   {
      public ushort alkév;
      public Főállású(string vn, string kn): base(vn,kn)
      {}
      public Főállású(string vn, string kn, ushort aé): base(vn,kn)
      {
         alkév = aé;
      }
      public override void névki()
      {
         Console.WriteLine(veznév + " " + kernév+" "+alkév);
      }
   }
```

Óraadó

```
class Óraadó: Oktató
   {
      public string cég;
      public Óraadó(string vn, string kn)
         : base(vn, kn)
      { }
      public Óraadó(string vn, string kn, string cg)
         : base(vn, kn)
      {
         cég = cg;
      }
      public override void névki()
      {
         Console.WriteLine(veznév + " " + kernév + " " + cég);
      }
   }
```

Főprogram: melyik sor hibás?

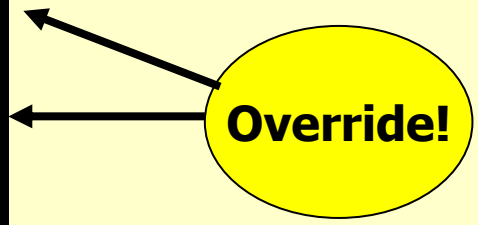
```
public static void Main()
    {  Főállású Józsi = new Főállású("Nagy", "József", 2005);
       Oktató Nagy = Józsi;
       Józsi.névki();
       Nagy.névki();
       Console.WriteLine(Józsi.alkév);
       Oktató Kis = new Főállású("Kis", "János");
       Kis.névki();
       Oktató János = new Főállású("Kis", "János", 2007);
       János.névki();
       Óraadó Marcsi = new Óraadó("Közepes", "Marcsi", "IBM");
       Oktató Közepes = Marcsi;
       Oktató Hosszú = new Óraadó("Hosszú", "Tibor", "CIA");
       Marcsi.névki();
       Közepes.névki();
       Console.WriteLine(Marcsi.cég);
       Oktató Pici = new Oktató("Pici", "Juliska");
       Pici.névki();
       Console.ReadLine();      }
```

Főprogram: mit ír ki?

```
public static void Main()
{  Főállású Józsi = new Főállású("Nagy", "József", 2005);
   Oktató Nagy = Józsi;
   Józsi.névki();
   Nagy.névki();
   Console.WriteLine(Józsi.alkév);
   Oktató Kis = new Főállású("Kis", "János");
   Kis.névki();
   Oktató János = new Főállású("Kis", "János", 2007);
   János.névki();
   Óraadó Marcsi = new Óraadó("Közepes", "Marcsi", "IBM");
   Oktató Közepes = Marcsi;
   Oktató Hosszú = new Óraadó("Hosszú", "Tibor", "CIA");
   Marcsi.névki();
   Közepes.névki();
   Hosszú.névki();
   Console.WriteLine(Marcsi.cég);
   //Oktató Pici = new Oktató("Pici", "Juliska")
   //Pici.névki();
   Console.ReadLine();       }
```

Eredmény

```
Nagy József 2005
Nagy József 2005
2005
Kis János 0
Kis János 2007
Közepes Marcsi IBM
Közepes Marcsi IBM
Hosszú Tibor CIA
IBM
```



Feladat

Készítsen számológépet: a „Számoló” osztály tartalmazzon egy „Kalk” metódust, mely adott műveleti kódra elvégzi két operandus között a megfelelő műveletet.

(A Kalk metódus az adatokat az osztály adattagjaiból vegye, ezeket a konstruktor állítsa be.)

A „Számol” osztály készítsen ebből egy „Kiszámol” nevű példányt, kérjen be két operandust és egy műveleti kódot, végeztesse el a műveletet, majd írja ki az eredményt.

Számoló osztály: a Konstruktor

```
class Számoló
{
 private float op1,op2;
 private char művelet;
 private float eredmény;
 public Számoló(float op10, float op20, char művelet0)
 {
  op1 = op10;
  op2 = op20;
  művelet = művelet0;
  eredmény = 0;
 }
```

Számoló osztály: a Kalk metódus

```
public float Kalk()
{
 switch (művelet)
 {
 case '+': eredmény = op1 + op2;
 break;
 case '-': eredmény = op1 - op2;
 break;
 case '*': eredmény = op1 * op2;
 break;
 case '/': eredmény = op1 / op2;
 break;
 }
 return eredmény;
 }
}
```

A Számol osztály

```
class Számol
{
 static void Main()
 {
 float x,y;
 char z;
  System.Console.Write("Első operandus: ");
 x=float.Parse(System.Console.ReadLine());
 System.Console.Write("Műveleti jel: ");
 z=char.Parse(System.Console.ReadLine());
 System.Console.Write("Második operandus: ");
 y=float.Parse(System.Console.ReadLine());
 Számoló Kiszámol= new Számoló(x,y,z);
 System.Console.WriteLine(x+" "+z+" "+y+" = "+Kiszámol.Kalk());
 System.Console.ReadLine();
 }
}
```

Egységbe zárás

A „private” (alapértelmezés) vagy „protected” láthatóságú adattételek kívűlről nem érhetőek el.

Próbáljuk közvetlenül kiírni az eredményt:

-
-
System.Console.WriteLine(x+" "+z+" "+y+" = "+Kiszámol.eredmény);
-
-

Egyszerű öröklés

Készítsük el a „Számoló” osztály örökösét, a „Számolgató” osztályt, mely pontos mása az ősnek.

A „Számolgató” osztály

```
class Számolgató:Számoló
{
   public Számolgató(float op10, float op20, char művelet0):
                                                base(op10,op20,művelet0)
   {
   }
}
```

A Számolgató és a Számoló ugyanazokkal az adattételekkel és metódusokkal rendelkezik, de a Számoló konstruktora paraméteres, így automatikusan nem hívódik meg. A „base” tétel használata után már ugyanúgy hívhatjuk:

```
.
.
Számolgató Kiszámol= new Számolgató(x,y,z);
.
.
```

Többalakúság I.

Bővítsük a „Számolgató” osztály „Kalk” metódusát egy új művelettel: a „<„ művelet eredménye legyen a két operandus közül a kisebb.

Az adattételek

```
protected float op1,op2;
protected char művelet;
protected float eredmény;
```

A „private” (ez az alapértelmezés) adattételek az örökös nem örökölt metódusai számára sem láthatók!

A módosított Kalk metódus

```
public new float Kalk()
{
 switch (művelet)
 {
   case '+': eredmény = op1 + op2;
   break;
   case '-': eredmény = op1 - op2;
   break;
   case '*': eredmény = op1 * op2;
   break;
   case '/': eredmény = op1 / op2;
   break;
   case '<': if (op2 < op1)
              { eredmény = op2; }
             else
              { eredmény = op1; }
             break;
  }
 return eredmény;
}
```

Hívás

```
class Számol
{
 static void Main()
 {
 float x,y;
 char z;
 System.Console.Write("Első operandus: ");
 x=float.Parse(System.Console.ReadLine());
 System.Console.Write("Műveleti jel: ");
 z=char.Parse(System.Console.ReadLine());
 System.Console.Write("Második operandus: ");
 y=float.Parse(System.Console.ReadLine());
 Számolgató Kiszámol= new Számolgató(x,y,z);
 System.Console.WriteLine(x+" "+z+" "+y+" = "+Kiszámol.Kalk());
 System.Console.ReadLine();
 }
}
```

Többalakúság II.: korai kötés

Bonyolítsuk a ”Számoló” és a „Számolgató” osztályokat: ne a „Kalk” metódus legyen kívülről elérhető, hanem a „Kalkuláló” metódus, mely meghívja a „Kalk”-ot.

Változások a „Számoló” osztályban

```
.
protected float Kalk()
 {
  switch (művelet)
.
.
  return eredmény;
 }


  public float Kalkuláló()
  {
    return Kalk();
   }
```

Változások a „Számolgató” osztályban

```
.
protected new float Kalk()
 {
  switch (művelet)
.
.
  return eredmény;
 }
```

Kérdés

A „Kalk” metódust újra definiáltuk, de a „Kalkuláló” metódust nem. A „Számolgató” osztályban a „Kalkuláló” melyik „Kalk”-ot hívja meg?

Válasz

Az eredetit. Ez a „korai kötés”. (Ha a „<„ műveletet adjuk meg, az eredmény 0 lesz.)

A késői kötés

Tegyük a „Kalk” metódust virtuálissá!

A Kalk metódus (Számoló)

```
protected virtual float Kalk()
{
 switch (művelet)
 {
   case '+': eredmény = op1 + op2;
   break;
   case '-': eredmény = op1 - op2;
   break;
   case '*': eredmény = op1 * op2;
   break;
   case '/': eredmény = op1 / op2;
   break;
  }
 return eredmény;
}
```

A Kalk metódus (Számolgató)

```
protected override float Kalk()
{
 switch (művelet)
 {
   case '+': eredmény = op1 + op2;
   break;
   case '-': eredmény = op1 - op2;
   break;
   case '*': eredmény = op1 * op2;
   break;
   case '/': eredmény = op1 / op2;
   break;
   case '<': if (op2 < op1)
              { eredmény = op2; }
             else
              { eredmény = op1; }
             break;
  }
 return eredmény;
}
```

Kérdés

A „Kalk” metódust újra definiáltuk, de a „Kalkuláló” metódust nem. A „Számolgató” osztályban a „Kalkuláló” melyik „Kalk”-ot hívja meg?

Válasz

Az újat. Ez a „késői kötés”. (Ha a „<„ műveletet adjuk meg, az eredmény jó lesz.)

Mentsük el ezt a programot, mert később még szükség lesz rá!

Feladat

Egy korábbi feladat módosításaképpen készítsen absztrakt „Vonatdef” osztályt, melyben a „beolvas” az állomások neve, az indulások ideje (max 100 db), és ezek az osztály adattételeiben tárolódnak.

Az üres állomásnév jelentse a beolvasás végét.

Legyen adattétel a „típus” string változó is.

A „kiír” metódus kiírja a vonat adatait a képernyőre.

A „betípus” absztrakt metódus legyen.

Készítse el a „Vonatdef” osztály örököseit a „Gyorsvonat” és a „Személyvonat” osztályokat, melyekben a „betípus” metódus beírja a vonat típusát a „típus” adattételbe.

Készítsen „Vonatkez” osztályt, melyben a fenti osztály példányait egyetlen tömbben helyezi el, beírja az adatokat, majd kiírja azokat a képernyőre.

Feladat

Készítsen a „Vonatdef” osztályban olyan metódust, mely a vonat nevét képes megváltoztatni.

Feladat

Végezze el ugyanezt a „beolvas” metódus átdefiniálásával.

Feladat

Készítsünk egy absztrakt Árú osztályt. Tartalmazzon név, készlet, maxDb, és nettóÁr mezőket az egységbezárás adatrejtés elvét betartva. Készítsünk utódosztályokat: TejTermék utódosztály minden tagjának típusa tejtermék legyen és tartalmazzon egy zsírtartalom nevű értéket. A pékárú minden tagjának pékárú legyen a típusa és adjuk meg, hogy friss, vagy tartós termék. Az édességeknél határozzuk meg a tartalmaz-e cukrot mezőt. Minden osztálynak legyen Kiír tagfüggvénye, mely a konzolra kiírja az adott osztály tagjait és azok értékét.

Egy tömbben tároljuk az árúkat. Töltsük fel a tömböt, különböző típusú árúkkal!

Írjuk ki a tömb elemeit a konzolra!

Számítsuk ki a raktárkészlet értékét!