PROGRAMOZÁS II
3. ÓRA: STATIKUS ADATOK ÉS ÖRÖKLÉS

STATIKUS ADATTAGOK ÉS METÓDUSOK

Változók hatóköre C#-ban

- Blokk
- Metódus
- Objektum
- Osztály

Ezeket ismerjük

Ez mi?

Statikus adattagok

- Az osztály hagyományos adattagjai objektum szintű változók
  - Létrejönnek, amikor az objektum
  - Megszűnnek, amikor az objektum

- Mi történik, ha olyan adatra van szükségünk, ami egy, a teljes programra érvényes érték
  - Ez lenne a globális változó
  - C#-ban nincs ilyen
  - Helyette vannak statikus adattagok

És általában is rossz megközelítés

Statikus adattagok

- Osztály szintű adatok
- Az egyes objektumok nem tartalmaznak saját példányt a statikus adatokból
- A programban pontosan 1 létezik belőlük
- A használatukhoz nincs szükség objektumokra

Statikus adattagok

```
class EventCounter
{
    public static int TriggerValue { get; set; } = 5;
    public int CurrentValue { get; private set; }
    public string Name { get; }

    public EventCounter(string name)
    {
        Name = name;
        CurrentValue = 0;
    }

    public void Trigger()
    {
        CurrentValue++;
        if (CurrentValue == TriggerValue)
        {
            Console.WriteLine($"Event \"{Name}\" triggered {CurrentValue} times");
        }
    }
}
```

A TriggerValue statikus, így 1 létezik belőle, azt éri ez az osztály összes példánya

Statikus adattagok

```
EventCounter newUserCounter = new EventCounter("New user");
EventCounter userLoginCounter = new EventCounter("User login");

EventCounter.TriggerValue = 2;
newUserCounter.Trigger();
newUserCounter.Trigger();
//Event "New user" triggered 2 times

newUserCounter.Trigger();
userLoginCounter.Trigger();
userLoginCounter.Trigger();
//Event "User login" triggered 2 times

EventCounter.TriggerValue = 4;
newUserCounter.Trigger();
//Event "New user" triggered 4 times
```

Elérhető pusztán az osztályon keresztül, nem szükséges hozzá objektum

Statikus metódusok

- Metódus is lehet statikus

- Nem tartozik objektumhoz, így
  - A meghívásához elég az osztály, nem kell objektum
  - Nem lehet benne objektum szintű változókra hivatkozni (nincs this)

Statikus metódusok

```
class EventCounter
{
    public static int TriggerValue { get; set; } = 5;
    public static int HighestValue { get; private set; } = 0;
    public int CurrentValue { get; private set; }
    …
    public void Trigger()
    {
        …
        if (CurrentValue > HighestValue)
        { HighestValue = CurrentValue; }
    }
    public static void UpdateTriggerValue()
    {
        if (TriggerValue < HighestValue)
        { TriggerValue = HighestValue + 5; }
    }
}
```

Statikus metódus, amely csak a statikus változókat képes kezelni

Statikus metódusok

```
EventCounter newUserCounter = new EventCounter("New user");
EventCounter.TriggerValue = 2;
newUserCounter.Trigger();
newUserCounter.Trigger();
//Event "New user" triggered 2 times

newUserCounter.Trigger();
newUserCounter.Trigger();
EventCounter.UpdateTriggerValue();

newUserCounter.Trigger();
newUserCounter.Trigger();
newUserCounter.Trigger();
newUserCounter.Trigger();
newUserCounter.Trigger();
//Event "New user" triggered 9 times
```

Statikus metódus megívásához sem kell objektum

ÖRÖKLÉS

Öröklés

- Öröklés során egy új osztályt definiálunk úgy, hogy egy már
  - Meglévő osztályt bővítünk
  - A meglévő osztályt ős-osztálynak hívjuk
  - Az új osztályt származtatott (gyerek) osztálynak hívjuk
- A származtatott osztály
  - Megörökli az ős minden adattagját és metódusát
  - Új adattagokat és metódusokat definiálhat
  - Felülírhatja az ős metódusait

Öröklés folyamata

- A származtatott osztály az ősnek egy specializált változata (az őshöz képest plusz tagokat tartalmaz)
  - Pl.: élőlény → állat → emlős → kutya
  - Több utód esetén öröklődési fák jelennek meg
  - Az ős osztály szerepét a leszármazottai is eljátszhatják: kompatibilitás
    - Ős helyett átadható leszármazott
    - Ős referenciája, hivatkozhat bármely leszármazottjára.
    - Pl.: készíthetünk olyan tömböt/listát, amelyben vegyesen szerepelnek különféle emlősök: kutyák és emberek is
- C#-ban egy osztálynak csak egy őse lehet

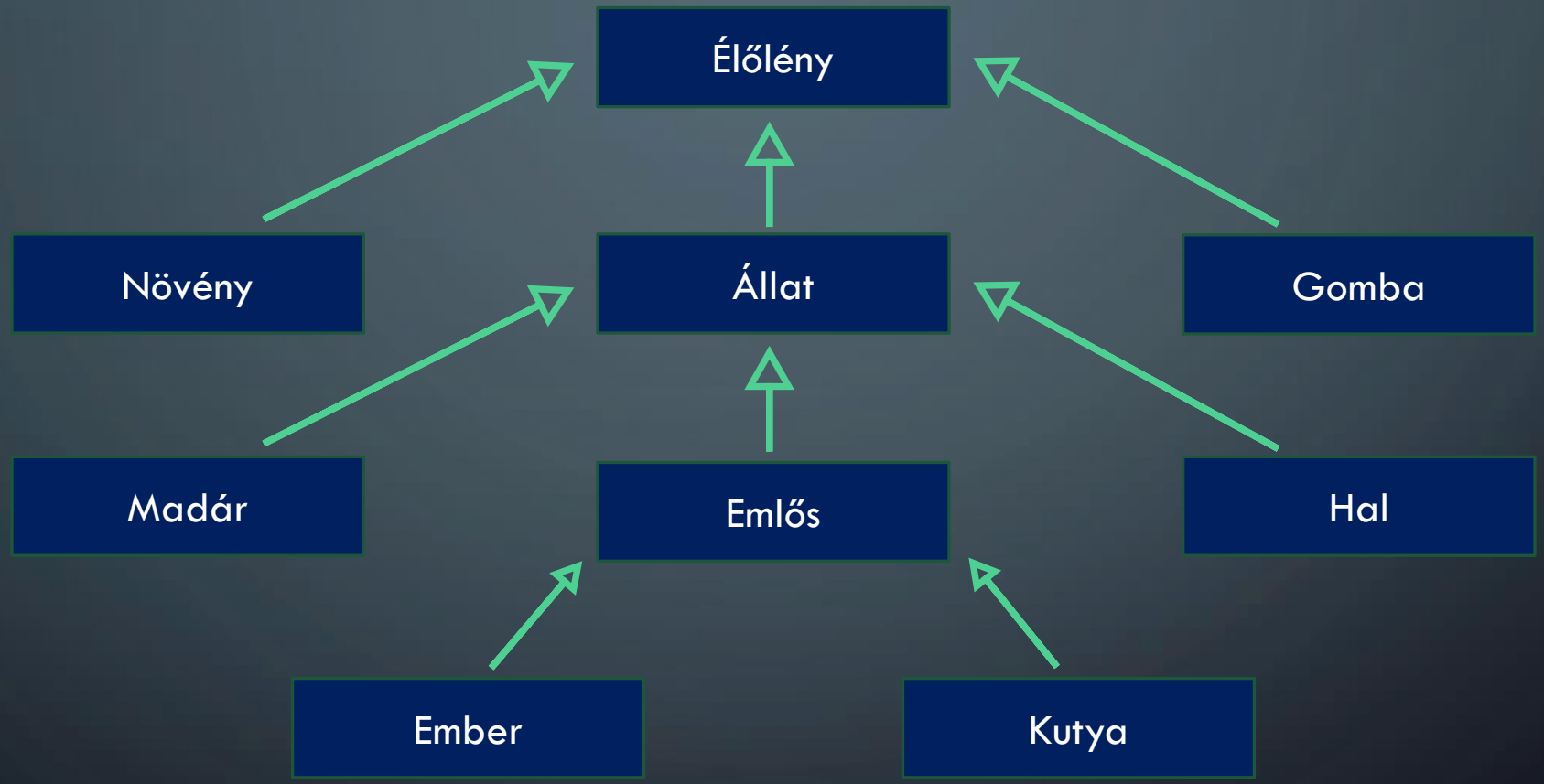
Ez az Interfészekkel kezelhető, de erről majd később

Öröklés előnyei

- Ösztönzi
  - A képességek hierarchiába rendezését
  - A kód újrahasznosítást

- Az ősben történt változás automatikusan megjelenik a leszármazottakban
  - Ha minden osztályt a nulláról építünk, akkor a változtatást mindenhol egyenként kell megvalósítanunk

Öröklési fa



Származtatott osztály létrehozása

- Csak a közvetlen őst kell megadni
- A származtatott osztálynak saját konstruktorra van szüksége
  - A konstruktor csak a saját adatok beállításával foglalkozik
  - Az ősosztály konstruktorát mindig meghívja
    - Ha az vár paramétert, azt át kell neki adni

- Származtatott osztályban lehetőség van olyan metódust (vagy akár adattagot) létrehozni, ami az ősben is szerepel
  - Ekkor az új tag elrejti az ősben lévőt a használat során

Származtatott osztály létrehozása

```
class Lakas
{
    public double Alapterulet { get; }
    public int LakosokSzama { get; set; }
    public Lakas(double terulet, int lakosok)
    {
        Alapterulet = terulet;
        LakosokSzama = lakosok;
    }
}
```

```
class PanelLakas : Lakas
{
    public int Emelet { get; }
    public int Ajto { get; }
    public PanelLakas(double terulet, int lakosok, int emelet, int ajto) :
        base(terulet, lakosok)
    {
        Emelet = emelet;
        Ajto = ajto;
    }
}
```

Az ős konstruktorát meg kell hívni

Ős adatok felülírása

```
class KertesHaz : Lakas
{
    public double KertTerulet { get; }
    public KertesHaz(double terulet, int lakosok, double kert):
        base(terulet, lakosok)
    { KertTerulet = kert; }

    public double Alapterulet
    {
        get { return base.Alapterulet + this.KertTerulet; }
    }
}
```

A base segítségével érhetjük el az ősosztály ugyanilyen nevű tagját

Ha a KertesHaz osztályon keresztül használjuk az Alapterulet property-t, akkor beleszámolja a kert területét is

```
KertesHaz k = new KertesHaz(83.5, 3, 245.3);
Console.WriteLine(k.Alapterulet);
```

NAGYOBB PÉLDA: ÜZENETKÜLDÉS

Üzenetküldés (Ál)hálózaton

- Rakjunk össze egy rendszert, ami hálózati üzenetküldést utánoz
- Legyen sima és titkosított üzenet
- Tényleges hálózati adatküldés helyett csak leegyszerűsítve utánozzuk

Osztályok

- Packet
  - Egy csomag, ami a hálózaton keresztül elküldünk
- PacketBuffer
  - A csomagok bufferelésére
- MockNetwork
  - A hálózati forgalom utánzására
- Socket
  - Alapvető kommunikáció mevalósítására
- SecureSocket
  - Titkosított kommunikáció megvalósítására

Packet

- Tárolja a csomagok adatait:
  - Küldő
  - Címzett
  - Tartalom

- Csak adattárolásra van

Packet

```
class Packet
{
    public string SourceAddress { get; }
    public string DestinationAddress { get; }
    public string Content { get; }

    public Packet()
    {
        SourceAddress = "";
        DestinationAddress = "";
        Content = "";
    }
    public Packet(string sourceAddress, string destinationAdress, string content)
    {
        SourceAddress = sourceAddress;
        DestinationAddress = destinationAdress;
        Content = content;
    }
```

Readonly property-k

PacketBuffer

- Hálózati adatküldés során fontos a beérkező adatok bufferelése
- Ennek egy egyszerű verzióját valósítja meg
- Tömb alapú sor (FIFO) implementáció

PacketBuffer

```
class PacketBuffer
{
    // Array-based implementation of a queue
    const int _maxBufferSize = 100;
    Packet[] _buffer = new Packet[_maxBufferSize];
    int _queueStart = 0;
    int _queueEnd = 0;
    int _queueSize = 0;

    public bool AddPacket(Packet packet)
    {}

    public Packet RemovePacket()
    {}
}
```

A const egy olyan statikus adatot jelöl, ami nem módosítható a futás során

Ennek a működése technikai, a részletek most nem fontosak

MockNetwork

- A hálózati kommunikációt utánozza, leegyszerűsítve
- Socket-et csatlakozhatnak rá
  - Ez lehet a Socket-ből származó osztály példánya is
- Üzenet küldés esetén megkeresi a címzettet, és továbbítja a csomagot
- A fizikai adatközeget utánozza, ami egy globális objektum
  - Az osztályban minden statikus, hiszen csak egy, globális közeg van

MockNetwork

```
class MockNetwork
{
    const int _defaultArraySize = 5;
    static Socket[] _sockets = new Socket[_defaultArraySize];
    static int _socketCount = 0;

    private static void ResizeArray()
    {}

    public static void RegisterSocket(Socket socket)
    {}

    public static void RemoveSocket(Socket socket)
    {}

    public static bool DeliverPacket(Packet packet)
    {}
```

Átméreteződő tömb, hogy minden Socket-nek legyen hely
Gyerek objektumot is tárolhat

Minden statikus

Socket

- Egy szereplő a hálózaton
- Rendelkezik:
  - Saját címmel
  - Bufferrel a bejövő csomagoknak
- A saját csomagok küldésével és fogadásával foglalkozik

Socket

```
class Socket
{
    public string MyAddress { get; }
    protected PacketBuffer _buffer = new PacketBuffer();
    public Socket(string myAdress)
    {}

    public void SendMessage(string destination, string message)
    {}

    public void ReceiveMessage()
    {}

    public bool AddIncomingMessage(Packet packet)
    {}

    protected Packet CreatePacket(string destination, string message)
    {}
}
```

Ezt használja a MockNetwork, hogy átadja a neki címzett csomagot

Segéd függvény

SecureSocket

- A Socket gyerekosztálya
- Titkosított üzenetek küldésére és fogadására
- A példában nincs benne extra adattag, de tárolhatna a titkosításhoz szükséges információkat

SecureSocket

```
class SecureSocket : Socket
{
    public SecureSocket(string myAddress):
        base(myAddress)
    {}

    protected string EncodeMessage(string message)
    {}
    protected string DecodeMessage(string message)
    {}

    public void SendMessage(string destination, string message)
    {}
    public void ReceiveMessage()
    {}
}
```

Titkosítás és visszafejtés

Ős metódusai módosítva, hogy alkalmazzák a titkosítást