**Öröklés (C # programozási útmutató)**

* 2020.07.02

Az öröklés a kapszulázással és a polimorfizmussal együtt az objektum-orientált programozás három elsődleges jellemzője. Az öröklés lehetővé teszi új osztályok létrehozását, amelyek újrafelhasználják, kiterjesztik és módosítják a többi osztályban meghatározott viselkedést. Az osztályt, amelynek tagjai örököltek, *alaposztálynak* , az osztályokat pedig *származtatott osztálynak* nevezzük . A származtatott osztálynak csak egy közvetlen alaposztálya lehet. Az öröklés azonban tranzitív. Ha ClassCszármazik ClassBés ClassBszármazik ClassA, ClassCörökli a ClassBés ben deklarált tagokat ClassA.

**jegyzet**

A struktúrák nem támogatják az öröklést, de interfészeket képesek megvalósítani. További információ: [**Interfészek**](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/interfaces/) .

Fogalmilag egy származtatott osztály az alaposztály specializációja. Például, ha van alaposztálya Animal, akkor lehet, hogy van egy származtatott osztálya, amelyet elneveznek, Mammalés egy másik származtatott osztálya, amelyet elneveznek Reptile. A Mammaljelentése an Animalés a Reptilejelentése an Animal, de mindegyik származtatott osztály az alaposztály különböző specializációit képviseli.

Az interfész deklarációk meghatározhatnak egy alapértelmezett megvalósítást tagjai számára. Ezeket a megvalósításokat származtatott interfészek és az interfészeket megvalósító osztályok öröklik. Az alapértelmezett interfész-módszerekről további információt a nyelvi hivatkozás szakasz [interfészekről](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/interface) szóló cikkében talál .

Ha meghatároz egy osztályt, hogy egy másik osztályból származzon, akkor a származtatott osztály implicit módon megszerzi az alaposztály összes tagját, kivéve a konstruktorokat és a véglegesítőket. A származtatott osztály újrafelhasználja a kódot az alaposztályban anélkül, hogy újra ki kellene töltenie. További tagokat vehet fel a levezetett osztályba. A levezetett osztály kiterjeszti az alaposztály funkcionalitását.

A következő ábra egy olyan osztályt mutat be WorkItem, amely valamely üzleti folyamat munkáját reprezentálja. Mint minden osztály, ez is a [System.Object-](https://docs.microsoft.com/en-us/dotnet/api/system.object) ből származik, és örökli az összes módszerét. WorkItemöt saját tagot ad hozzá. Ezek a tagok tartalmaznak egy konstruktort, mert a konstruktorok nem öröklődnek. Az osztály egy adott típusú ChangeRequestmunkaelemet örököl WorkItemés képvisel. ChangeRequesttovábbi két tagot ad hozzá WorkItemaz [Object-](https://docs.microsoft.com/en-us/dotnet/api/system.object) től örökölt tagokhoz . Hozzá kell adnia a saját konstruktorát, és hozzá is ad originalItemID. A tulajdonság originalItemIDlehetővé teszi a ChangeRequestpéldány társítását az eredetivel WorkItem, amelyre a módosítási kérelem vonatkozik.

A következő példa bemutatja, hogy az előző ábrán bemutatott osztályviszonyok hogyan fejeződnek ki C # -ben. A példa azt is bemutatja, hogy miként lehet WorkItemfelülbírálni az [Object.ToString](https://docs.microsoft.com/en-us/dotnet/api/system.object.tostring) virtuális módszert , és az ChangeRequestosztály hogyan örökli WorkItema módszer megvalósítását. Az első blokk meghatározza az osztályokat:

C #Másolat

// WorkItem implicitly inherits from the Object class.

public class WorkItem

{

// Static field currentID stores the job ID of the last WorkItem that

// has been created.

private static int currentID;

//Properties.

protected int ID { get; set; }

protected string Title { get; set; }

protected string Description { get; set; }

protected TimeSpan jobLength { get; set; }

// Default constructor. If a derived class does not invoke a base-

// class constructor explicitly, the default constructor is called

// implicitly.

public WorkItem()

{

ID = 0;

Title = "Default title";

Description = "Default description.";

jobLength = new TimeSpan();

}

// Instance constructor that has three parameters.

public WorkItem(string title, string desc, TimeSpan joblen)

{

this.ID = GetNextID();

this.Title = title;

this.Description = desc;

this.jobLength = joblen;

}

// Static constructor to initialize the static member, currentID. This

// constructor is called one time, automatically, before any instance

// of WorkItem or ChangeRequest is created, or currentID is referenced.

static WorkItem() => currentID = 0;

// currentID is a static field. It is incremented each time a new

// instance of WorkItem is created.

protected int GetNextID() => ++currentID;

// Method Update enables you to update the title and job length of an

// existing WorkItem object.

public void Update(string title, TimeSpan joblen)

{

this.Title = title;

this.jobLength = joblen;

}

// Virtual method override of the ToString method that is inherited

// from System.Object.

public override string ToString() =>

$"{this.ID} - {this.Title}";

}

// ChangeRequest derives from WorkItem and adds a property (originalItemID)

// and two constructors.

public class ChangeRequest : WorkItem

{

protected int originalItemID { get; set; }

// Constructors. Because neither constructor calls a base-class

// constructor explicitly, the default constructor in the base class

// is called implicitly. The base class must contain a default

// constructor.

// Default constructor for the derived class.

public ChangeRequest() { }

// Instance constructor that has four parameters.

public ChangeRequest(string title, string desc, TimeSpan jobLen,

int originalID)

{

// The following properties and the GetNexID method are inherited

// from WorkItem.

this.ID = GetNextID();

this.Title = title;

this.Description = desc;

this.jobLength = jobLen;

// Property originalItemId is a member of ChangeRequest, but not

// of WorkItem.

this.originalItemID = originalID;

}

}

Ez a következő blokk bemutatja az alap és a származtatott osztályok használatát:

C #Másolat

// Create an instance of WorkItem by using the constructor in the

// base class that takes three arguments.

WorkItem item = new WorkItem("Fix Bugs",

"Fix all bugs in my code branch",

new TimeSpan(3, 4, 0, 0));

// Create an instance of ChangeRequest by using the constructor in

// the derived class that takes four arguments.

ChangeRequest change = new ChangeRequest("Change Base Class Design",

"Add members to the class",

new TimeSpan(4, 0, 0),

1);

// Use the ToString method defined in WorkItem.

Console.WriteLine(item.ToString());

// Use the inherited Update method to change the title of the

// ChangeRequest object.

change.Update("Change the Design of the Base Class",

new TimeSpan(4, 0, 0));

// ChangeRequest inherits WorkItem's override of ToString.

Console.WriteLine(change.ToString());

/\* Output:

1 - Fix Bugs

2 - Change the Design of the Base Class

\*/

**Absztrakt és virtuális módszerek**

Amikor egy alaposztály deklarálja a metódust [virtual](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/virtual), a származtatott osztály [override](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/override)a saját implementációjával képes a módszert használni. Ha egy alaposztály deklarálja a tagot [abstract](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/abstract), akkor ezt a módszert felül kell írni minden olyan nem elvont osztályban, amely közvetlenül örökli az adott osztályt. Ha egy származtatott osztály maga is elvont, akkor absztrakt tagokat örökít, anélkül, hogy azokat megvalósítaná. Az absztrakt és a virtuális tagok képezik az alapját a polimorfizmusnak, amely az objektumorientált programozás elsődleges jellemzője. További információ: [Polimorfizmus](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/polymorphism) .

**Absztrakt alaposztályok**

Akkor kijelentik osztály [absztrakt](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/abstract) , ha azt szeretné, hogy közvetlenül ne példányosítás segítségével [új](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/new-operator) üzemben. Absztrakt osztály csak akkor használható, ha új osztály származik belőle. Az absztrakt osztály tartalmazhat egy vagy több metódus aláírást, amelyeket maguk is absztraktnak nyilvánítanak. Ezek az aláírások meghatározzák a paramétereket és a visszatérési értéket, de nincsenek megvalósításuk (method body). Az absztrakt osztálynak nem kell absztrakt tagokat tartalmaznia; ha azonban egy osztály tartalmaz elvont tagot, akkor magát az osztályt is elvontnak kell nyilvánítani. Azoknak a származtatott osztályoknak, amelyek nem elvontak, maguknak kell biztosítaniuk az elvont alaposztály bármely elvont módszerének megvalósítását. További információ: [Absztrakt és lezárt osztályok és tagok](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/abstract-and-sealed-classes-and-class-members) .

**Interfészek**

Az *interfész* referencia típus, amely meghatározza a tagok halmazát. Valamennyi osztálynak és struktúrának, amely az interfészt megvalósítja, be kell vezetnie a tagokat. Az interfész meghatározhat alapértelmezett megvalósítást e tagok bármelyikének vagy mindegyikének. Egy osztály több interfészt is megvalósíthat, annak ellenére, hogy csak egyetlen közvetlen alaposztályból származhat.

Az interfészeket olyan képességek meghatározására használják, amelyek nem feltétlenül rendelkeznek "egy" kapcsolattal. Például a [System.IEquatable <T>](https://docs.microsoft.com/en-us/dotnet/api/system.iequatable-1) felületet bármely osztály vagy struktúra megvalósíthatja annak megállapítására, hogy két típusú objektum egyenértékű-e (azonban a típus meghatározza az ekvivalenciát). [Az IEquatable <T>](https://docs.microsoft.com/en-us/dotnet/api/system.iequatable-1) nem jelenti azt, hogy ugyanaz a fajta "is" kapcsolat létezik egy alaposztály és egy származtatott osztály között (például az a Mammalegy an Animal). További információ: [Interfészek](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/interfaces/) .

**A további levezetés megakadályozása**

Egy osztály megakadályozhatja, hogy más osztályok örököljenek tőle vagy bármely tagjától, azzal, hogy önmagát vagy tagját annak nyilvánítja [sealed](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/sealed). További információ: [Absztrakt és lezárt osztályok és tagok](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/abstract-and-sealed-classes-and-class-members) .

**Az alaposztály tagjainak levezetett osztályának elrejtése**

A származtatott osztály elrejtheti az alaposztály tagjait azáltal, hogy ugyanazon névvel és aláírással tagokat deklarál. A [new](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/new-modifier)módosítóval kifejezetten jelezni lehet, hogy a tag nem az alaptag felülbírálása. A (z) használata [new](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/new-modifier)nem szükséges, de a fordító figyelmeztetést generál, ha [new](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/new-modifier)nem használjuk. További információ: [Verziós verzió felülírása és új kulcsszavakkal,](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/versioning-with-the-override-and-new-keywords) valamint [az Felülírás és új kulcsszavak](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/versioning-with-the-override-and-new-keywords)[használatának ideje](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/knowing-when-to-use-override-and-new-keywords) .