📘 Դաս 7.3. `System.Object` խորացված

Ինչու է կարևոր

.NET-ում **բոլորը** (class, record, array, delegate, enum, struct*) վերջում աշխատում են որպես `object`— այսինքն՝ ժառանգում կամ հատուկ վերաբերմունք ունեն `System.Object` արմատային տիպի հանդեպ: Դրա վերագրված (override) 6 մեթոդները ձևավորում են.

- **ինքնության/հավասարության սեմանտիկա** (`Equals`, `GetHashCode`),
- **ներկայացում** (`ToString`),
- **ռանթայմ տիպաբանություն** (`GetType`),
- **կյանքի ցիկլի մաքուր ավարտ** (`Finalize`/destructor),
- **կրկնօրինակման (copy) սեմանտիկա** (`MemberwiseClone`՝ shallow copy):

> ✱ structs-ը չեն “inherits” անում, բայց ունեն `ValueType` հիմք, որն էլ իր հերթին Object-ից է գալիս, ու ստանում են այս մեթոդների իմպլեմենտացիաներ/վերահաստատումներ:

---

1) `GetType()` — ռանթայմ տիպը (sealed)

Ի՞նչ է անում. Վերադարձնում է օբյեկտի **իրական** ռանթայմ տիպը (`System.Type`): Սա `virtual` չէ (sealed):
Տարբերություն `typeof(T)` և `x.GetType()` միջև.

- `typeof(T)` → compile-time տիպը.
- `x.GetType()` → runtime իրական տիպը (կարող է լինել ենթատիպ):

```
object x = new StringBuilder();
Console.WriteLine(x.GetType().Name); // "StringBuilder"
```

```
Type t1 = typeof(object);          // compile-time` object
Type t2 = x.GetType();             // runtime` StringBuilder
Console.WriteLine(t1 == t2);       // False
```

Օգտագործում.

- Reflection, ակտիվ պոլիմորֆիզմ, logging/debugging (“what type did I get?”),
- Pattern matching-ի decision-making (փոխարենը նախընտրիր `is` patterns, եթե կոնկրետ տիպի վրա տարբեր վարք է պետք):

---

2) `Equals(object?)` — հավասարության սեմանտիկա

Լռելյայն (classes). `Object.Equals`-ը reference types-ի համար → **reference equality** (նույն հղում = հավասար):
Structs. `ValueType.Equals`-ը լռելյայն կատարում է **field-by-field value equality**:

Equality contract (պարտադիր պահպանել, եթե override անում ես).

- **Reflexive**: `x.Equals(x)` → true
- **Symmetric**: `x.Equals(y)` == `y.Equals(x)`
- **Transitive**: եթե `x==y` և `y==z`, ապա `x==z`
- **Consistent**: նույն արժեքների դեպքում միշտ նույն արդյունքը
- **Null-safe**: `x.Equals(null)` → false

Լավ պրակտիկա. Եթե value equality ես ուզում **class**-ում՝

- **Սահմանիր immutable դաշտեր**,
- Իմպլեմենտիր `IEquatable<T>` + override `Equals(object)` և **պարտադիր** `GetHashCode()`:

```
public sealed class Money : IEquatable<Money>
{
    public string Currency { get; }
```

```
    public decimal Amount  { get; }

    public Money(string currency, decimal amount)
    {
        Currency = currency ?? throw new ArgumentNullException(nameof(currency));
        Amount   = amount;
    }

    public bool Equals(Money? other)
        => other is not null
        && Amount == other.Amount
        && string.Equals(Currency, other.Currency, StringComparison.Ordinal);

    public override bool Equals(object? obj) => Equals(obj as Money);

    public override int GetHashCode() => HashCode.Combine(Amount, Currency);
}
```

Նշումներ.

- Եթե override անում ես `Equals`, միշտ override արա նաև `GetHashCode` (տե՛ս հաջորդ բաժինը):
- Եթե նաև գերբեռնում ես `operator ==`/`!=`, պահպանի՛ր նույն սեմանտիկան, որ `==` համահունչ լինի `Equals`-ի հետ:
- Ժառանգական հիերարխիաներում value equality-ն բարդանում է (symmetry կոտրվելու ռիսկ). sealed տիպերում ավելի հեշտ է; մյուս դեպքում նախընտրելի է record կամ equality-ի սեմանտիկան թողնել derived-ին:

> 💡 `string`-ը override արած ունի value equality (`"ab" == "a" + "b"` → true):

---

3) `GetHashCode()` — հեշ՝ dictionary/set կառուցվածքների համար

Պայմանագիր. Եթե `x.Equals(y) == true`, ապա **պարտադիր** `x.GetHashCode() == y.GetHashCode()`: Հակադարձը պարտադիր չէ:
Կանոններ.

- Հեշը **կայուն** պետք է լինի օբյեկտի կյանքի ընթացքում (մասնավորապես՝ եթե օգտագործվում է `Dictionary`/`HashSet`-ում key/մեմբեր լինելով):

- **Մի՛ օգտագործիր mutable դաշտեր**, որոնք կարող են փոխվել այն բանից հետո, երբ օբյեկտը դարձել է key:

- Օգտագործիր `HashCode.Combine(...)` (NET ≥ Core 2.1) կամ լավ կոմբինացիոն մեթոդ:

```
public sealed class Point2D : IEquatable<Point2D>
{
    public int X { get; }
    public int Y { get; }

    public Point2D(int x, int y) { X = x; Y = y; }

    public bool Equals(Point2D? other) => other is not null && X == other.X && Y == other.Y;
    public override bool Equals(object? obj) => Equals(obj as Point2D);

    public override int GetHashCode() => HashCode.Combine(X, Y); // բալանսավորված հեշ
}
```

Հակաորինակ.

```
public sealed class BadKey
{
    public int A { get; set; } // mutable!
    public override bool Equals(object? o) => (o as BadKey)?.A == A;
    public override int GetHashCode() => A; // փոփոխվում է՝ պղտորում է Dictionary-ն
}
```

Այսպիսի key-ով `Dictionary`-ում արժեք փնտրելը կարող է “կորցնել” տարրը, քանի որ bucket-ը փոխվել է:

---

4) `ToString()` — ներկայացում (debug/log/UI)

Լռելյայն. Տիպի լրիվ անունը (`Namespace.TypeName`):
Լավ պրակտիկա.

- Վերագրիր մարդկային/մեքենայական ընթեռնելի ձևաչափով,

- Կարող ես տրամադրել **մի քանի ձևաչափ**՝ `IFormattable`-ով կամ overload-ներով:

```
public sealed class Person : IFormattable
{
    public string Name { get; }
    public DateTime BirthDate { get; }

    public Person(string name, DateTime birthDate) { Name = name; BirthDate = birthDate; }

    public override string ToString() => $"{Name} ({BirthDate:yyyy-MM-dd})";

    // Custom formats: "F" full, "S" short
    public string ToString(string? format, IFormatProvider? provider)
    {
        return (format?.ToUpperInvariant()) switch
        {
            "F" => $"{Name}, born on {BirthDate:D}",
            "S" => $"{Name} ({BirthDate:yyyy-MM})",
            _   => ToString()
        };
    }
}
```

Գործնական հուշում. Թեթև telemetry/logging-ում `ToString()`-ը շատ օգնում է՝ meaningful context տալու համար:

---

5) `Finalize()` (C# սինթաքս՝ `~ClassName()`) — վերջնականացումը

Ինչ է. Գործարկվում է GC-ի կողմից, երբ օբյեկտը հասանելի չէ ու գտնվում է finalization queue-ում:
Երբ օգտագործել. Գրեթե **երբեք** չես override անում ուղղակիորեն — փոխարենը կիրառիր **IDisposable pattern + SafeHandle** unmanaged ռեսուրսների համար:

Կանոններ.

- Finalizer-ը **կատարողականության** վրա վատ ազդեցություն ունի (երկկայան GC անցում, երկարաձգում):

- Եթե ունես unmanaged handle/ֆայլ/սոկետ, օգտագործիր `SafeHandle` և Dispose pattern:

- Կանչիր `GC.SuppressFinalize(this)` Dispose-ի վերջում, եթե finalizer ունես:

Ճիշտ Dispose pattern (հակիրճ).

```
using Microsoft.Win32.SafeHandles;

public class NativeResourceUser : IDisposable
{
    private bool _disposed;
    private readonly SafeFileHandle _handle;

    public NativeResourceUser(string path)
    {
        _handle = File.OpenHandle(path); // օրինակ
    }

    protected virtual void Dispose(bool disposing)
    {
        if (_disposed) return;

        if (disposing)
        {
            // managed cleanup (դարձյալ IDisposable-ներ)
        }

        // unmanaged cleanup՝ միշտ
        _handle?.Dispose();

        _disposed = true;
    }

    public void Dispose()
    {
        Dispose(true);
        GC.SuppressFinalize(this); // finalizer այլևս պետք չէ
    }

    ~NativeResourceUser() => Dispose(false); // safety net, եթե забудешь Dispose-ը
}
```

> 💡 Երբ հնարավոր է, finalizer **չէ** պետք; SafeHandle + Dispose՝ “gold standard”:

---

6) `MemberwiseClone()` — protected shallow copy

Ինչ է. Սահմանված է Object-ում, հասանելի է **միայն class-ի ներսից**: Ստեղծում է **մակերեսային** պատճեն (միևնույն հղումները copy են լինում reference դաշտերի համար):

```
public sealed class Address { public string City { get; set; } = ""; }

public class PersonCloneable
{
    public string Name { get; set; } = "";
    public Address Address { get; set; } = new Address();

    public PersonCloneable ShallowClone()
        => (PersonCloneable)this.MemberwiseClone();
}

// Դեմո
var p1 = new PersonCloneable { Name = "Anna", Address = new Address { City = "Yerevan" } };
var p2 = p1.ShallowClone();

p2.Name = "Armen";
p2.Address.City = "Gyumri";

Console.WriteLine(p1.Name);          // Anna  (քոքի պատճեն՝ անկախ)
Console.WriteLine(p1.Address.City);  // Gyumri  (նույն հղում՝ փոփոխվեց նաև p1-ում)
```

Երբ պետք է deep copy. Հղումային դաշտերը դարձնում ես **նոր instance** (manual clone), կամ օգտագործում ես serialize/deserialize (performance trade-off):
`ICloneable`-ից խուսափի՛ր public API-ում, որովհետև դրա semantics-ը .NET-ում չպարզված է (shallow թե deep):

Լրացուցիչ՝ `ReferenceEquals(a,b)`

Չի վերագրվում, static է, ստուգում է **հղումների նույնականությունը** (արդյոք նույն object instance-ն են), անտեսելով `Equals` override-ները:

```
var a = new StringBuilder("x");
var b = a;
Console.WriteLine(object.ReferenceEquals(a, b)); // True
```

💻 Օրինակների հավաքածու (կիրառական)

A) Հավասարություն + հեշ (sealed value object)

```
public sealed class Email : IEquatable<Email>
{
    public string Value { get; }
    public Email(string value)
    {
        if (string.IsNullOrWhiteSpace(value) || !value.Contains("@"))
            throw new ArgumentException("Invalid email");
        Value = value.Trim().ToLowerInvariant();
    }

    public bool Equals(Email? other) => other is not null && Value == other.Value;
    public override bool Equals(object? obj) => Equals(obj as Email);
    public override int GetHashCode() => Value.GetHashCode(StringComparison.Ordinal);
    public override string ToString() => Value;
}
```

B) `ToString` + `IFormattable`

```
public sealed class Temperature : IFormattable
{
    public double Celsius { get; }
    public Temperature(double c) => Celsius = c;

    public override string ToString() => $"{Celsius:0.0} °C";

    public string ToString(string? format, IFormatProvider? provider)
        => (format?.ToUpperInvariant()) switch
        {
            "F" => $"{Celsius * 9/5 + 32:0.0} °F",
            "K" => $"{Celsius + 273.15:0.00} K",
            _   => ToString()
        };
}
```

C) Downcast անվտանգ օգտագործում՝ `is`/`as`

```
Animal a = GetAnimal();
if (a is Dog dog) dog.Bark();

Cat? cat = a as Cat;
```

```
if (cat is not null) cat.Meow();
```

D) Shallow vs Deep clone

```
public class Node
{
    public int Value { get; set; }
    public Node? Next { get; set; }

    public Node Shallow() => (Node)MemberwiseClone();

    public Node Deep()
        => new Node { Value = Value, Next = Next?.Deep() };
}
```

🧪 Լաբորատոր անձարկումներ

1. **Value equality & hashing (Dictionary key)**

- Կազմիր immutable `ProductCode` (`string Sku`, `string? Country`) value object `IEquatable<T>`-ով:
- Օգտագործիր որպես `Dictionary<ProductCode, Product>` բանալի:
- Ցույց տուր, որ նույն Sku/Country զույգով բանալիները վերագրեն մեկ տարրը (no duplicates):

2. **`ToString` ձևաչափեր**

- Սահմանիր `Money` (`decimal Amount`, `string Currency`)՝ `IFormattable` support (“F”→`1,234.56 USD`, “S”→`USD 1234.56`):

3. **Dispose pattern + SafeHandle**

- Նմուշային class, որը բացում է ֆայլ handle և ապահով փակվում է Dispose-ում:
- Կանչիր `GC.Collect()` demo only (լաբում տեսնելու համար, թե finalizer-ը ե՞րբ է խաղում), բայց learners-ին բացատրի՛ր՝ production-ում **մի՛ օգտագործեք** ձեռքով GC:

4. **Shallow vs Deep Clone demo**

- Օգտագործիր reference դաշտով class և ցույց տուր shallow copy-ի ազդեցությունը:
- Ապա գրիր deep clone մեթոդ՝ ձեռքով ստեղծելով ներքին օբյեկտների նոր նմուշներ:

❓ Քննարկման հարցեր

- Ինչու է **Equals/GetHashCode** զույգը պարտադիր միասին override անել:
- Ի՞նչ ռիսկեր ունի mutable դաշտերով hash key ունենալը:
- Ինչու է `Finalize`-ը performance-heavy, և ինչո՞ւ է **IDisposable + SafeHandle** նախընտրելի:
- Ինչ տարբերություն կա `x.GetType()` և `typeof(T)` միջև:
- Ինչու **`MemberwiseClone` = shallow** և երբ է պետք **deep clone**:

📎 Ամփոփում (cheat-sheet)

- **GetType()** → runtime տիպ, sealed.
- **Equals()** → value/reference equality semantics՝ պահիր contract-ը, sealed + `IEquatable<T>` լավ կոմբո:
- **GetHashCode()** → համահունչ `Equals`-ին, immutable բաղադրիչներ, `HashCode.Combine(...)`:
- **ToString()** → meaningful ներկայացում, ըստ կարիքի՝ `IFormattable`:
- **Finalize()** → հազվադեպ, միայն unmanaged safety net, Dispose + `GC.SuppressFinalize`:
- **MemberwiseClone()** → protected shallow copy, deep clone՝ ձեռքով: