📘 Դաս 5. Կլասներ և Օբյեկտներ (Classes & Objects)

🎯 Ո՞ւմ համար է այս դասը

Ուսանողը պետք է կարողանա.

- բացատրել **class / object** հասկացությունները և **stack vs heap** գաղափարը,
- ստեղծել class՝ **դաշտերով** (fields), **հատկանիշներով** (properties), **կոնստրուկտորներով** (constructors),
- տարբերել **instance** ու **static** անդամները,
- կիրառել **encapsulation**՝ private դաշտ + public property՝ վավերացմամբ,
- օգտագործել **object initializer**, **constructor chaining**, **this** բանալի բառը,
- աշխատել **nullable reference types**-ի հետ (`string?`),
- հասկանալ **reference equality vs value equality**, և երբ ընտրել **record:**

🧠 Արտակարգ հակիրճ պատկերացում

- **Class** → սեղմագրություն/կաղապար (τύπος), որը նկարագրում է **ինչ հատկություններ և վարք** ունի օբյեկտը:
- **Object** → class-ից ստեղծված **կոնկրետ նմուշ** (instance) հիշողության մեջ:
- **Value types** (օր. `int`) պահվում են stack-ում արժեքով, **Reference types** (օր. `class`, `string`)՝ heap-ում, որտեղ փոփոխականը պահում է հղում:

ASCII պատկերացում.

Stack: Heap:
studentRef ——————————► [Student instance { Name="Anna", Age=19 }]

📖 Տեսական մաս՝ մանրամասն

1) Ամենապարզ class + object

```
public class Student
{
    // Fields (սովորաբար private)
    private string _name = "Unknown";
    private int _age;

    // Property (Auto-property)
    public string Name
    {
        get => _name;
        set => _name = string.IsNullOrWhiteSpace(value) ? "Unknown" : value.Trim();
    }

    // Property + validation
    public int Age
    {
        get => _age;
        set
        {
            if (value < 0 || value > 120)
                throw new ArgumentOutOfRangeException(nameof(Age), "Age must be between
0 and 120.");
            _age = value;
        }
    }

    // Method (վարք)
    public void Introduce()
    {
        Console.WriteLine($"Hi, I'm {Name}, {Age} years old.");
    }
}

// Օգտագործում
var s = new Student();
s.Name = "  Anna  ";   // setter → Trim
s.Age = 19;           // setter → validation
s.Introduce();        // "Hi, I'm Anna, 19 years old."
```

> Լավ պրակտիկա. դաշտերը **private**, տվյալների մատչումը՝ **properties**-ով:
> Setter-ում կատարիր validation/normalize:

---

2) Կոնստրուկտորներ, object initializer, this

```
public class Course
{
    public string Title { get; set; }
    public int Credits { get; set; }

    // Default ctor
    public Course()
    {
        Title = "Untitled";
        Credits = 1;
    }

    // Parametrized ctor
    public Course(string title, int credits)
    {
        Title = string.IsNullOrWhiteSpace(title) ? "Untitled" : title.Trim();
        Credits = credits is < 1 or > 15 ? 1 : credits;
    }
}

// Օբյեկտների ստեղծում՝ տարբեր եղանակներով
var c1 = new Course();                          // default ctor
var c2 = new Course("Algorithms", 6);           // parametrized ctor
var c3 = new Course { Title = "Databases", Credits = 5 }; // object initializer

// this՝ նույն անունով պարամետրերի դեպքում
public class Person
{
    private string _name;
    public Person(string name) => this._name = name; // this տարբերակում է դաշտը
պարամետրից
}
```

---

3) Constructor chaining (կոնստրուկտորների շղթայավորում)

```
public class Book
{
    public string Title { get; }
    public string Author { get; }
    public int Year { get; }

    public Book(string title) : this(title, "Unknown") { }

    public Book(string title, string author) : this(title, author, DateTime.Now.Year) { }
```

```
    public Book(string title, string author, int year)
    {
        Title = string.IsNullOrWhiteSpace(title) ? "Untitled" : title.Trim();
        Author = string.IsNullOrWhiteSpace(author) ? "Unknown" : author.Trim();
        Year = year;
    }
}
```

---

4) Auto-properties, backing field, expression-bodied, computed property

```
public class Rectangle
{
    // Auto-properties
    public double Width  { get; set; }
    public double Height { get; set; }

    // Computed read-only property
    public double Area => Width * Height;

    // Expression-bodied method
    public double Perimeter() => 2 * (Width + Height);
}

// Օգտագործում
var r = new Rectangle { Width = 3, Height = 4 };
Console.WriteLine(r.Area);        // 12
Console.WriteLine(r.Perimeter()); // 14
```

---

5) const vs readonly, init-only setters (C# 9+), immutable pattern

```
public class AppConfig
{
    public const string AppName = "MyApp"; // compile-time
    public static readonly DateTime BootAt = DateTime.UtcNow; // runtime set once

    // init-only → կարելի է նշանակել միայն initialization-ի պահին
    public string Environment { get; init; } = "Dev";
}

// Օգտագործում
var cfg = new AppConfig { Environment = "Prod" };
// cfg.Environment = "Test"; // ❌ compile error (init-only)
```

Երբ ուզում ես “քիչ շարժվող” արժեքներ՝ **immutable** մոտեցումը պահպանում է կանխատեսելիությունը:

---

6) Instance vs Static անդամներ

```
public class MathUtils
{
   public static double Pi => Math.PI; // class-ային անդամ

   public static int Add(int a, int b) => a + b;
}

int sum = MathUtils.Add(2, 3); // օբյեկտ պետք չէ
```

Static անդամները կապված են տիպի, ոչ թե կոնկրետ օբյեկտի հետ: Լոռկա, որը **state** չի պահում, հաճախ static է լինում:

---

7) Encapsulation՝ վավերացում setter-ում

```
public class BankAccount
{
   public string Owner { get; }
   public string Number { get; }
   private decimal _balance;

   public decimal Balance => _balance; // read-only արտաքինից

   public BankAccount(string owner, string number, decimal initial = 0)
   {
      Owner = string.IsNullOrWhiteSpace(owner) ? throw new
ArgumentNullException(nameof(owner)) : owner.Trim();
      Number = string.IsNullOrWhiteSpace(number) ? throw new
ArgumentNullException(nameof(number)) : number.Trim();
      if (initial < 0) throw new ArgumentOutOfRangeException(nameof(initial));
      _balance = initial;
   }

   public void Deposit(decimal amount)
   {
      if (amount <= 0) throw new ArgumentOutOfRangeException(nameof(amount));
      _balance += amount;
   }

   public void Withdraw(decimal amount)
```

```
    {
        if (amount <= 0) throw new ArgumentOutOfRangeException(nameof(amount));
        if (amount > _balance) throw new InvalidOperationException("Insufficient funds");
        _balance -= amount;
    }
}
```

> Այստեղ balance-ը **դրսից չի փոխվում**, միայն methods-ով՝ ապահով և վերահսկելի:

---

8) Nullability (nullable reference types) և պաշտպանություն null-երից

- .NET 6 նախագծերում **nullable**-ը on է լռելյայն:
- `string?` նշանակում է արժեքը կարող է լինել null:
- Օգտագործիր `ArgumentNullException.ThrowIfNull`.

```
public class Email
{
    public string Address { get; }

    public Email(string? address)
    {
        ArgumentNullException.ThrowIfNull(address);
        Address = address.Trim();
    }
}
```

---

9) Equals / GetHashCode / ToString, reference vs value equality

- `class`-երի default՝ **reference equality** (նույն հղում ≙ հավասար):
- Value semantics պետք լինի՝ override/implement կամ օգտագործիր **record**:

```
public class Point
{
    public int X { get; }
    public int Y { get; }
    public Point(int x, int y) { X = x; Y = y; }
```

```
    public override bool Equals(object? obj)
        => obj is Point p && p.X == X && p.Y == Y;

    public override int GetHashCode() => HashCode.Combine(X, Y);

    public override string ToString() => $"({X},{Y})";
}

var p1 = new Point(1,2);
var p2 = new Point(1,2);
Console.WriteLine(p1 == p2);         // False (reference)
Console.WriteLine(p1.Equals(p2));    // True  (value equality)
Console.WriteLine(p1);               // "(1,2)"
```

> Եթե հիմնականում “պետք է համեմատես ըստ արժեքների”, **record**-ը հեշտացնում է կյանքը (կքննարկենք առանձին դասում)։

---

10) Composition vs Inheritance (**սպոյլեր**)

Այս դասում կենտրոնը **composition**-ն է (մեկ օբյեկտը ունի երկրորդը)։
Inheritance-ը կանցնենք հաջորդ դասին։

```
public class Enrollment
{
    public Student Student { get; }
    public Course  Course  { get; }
    public DateTime EnrolledAt { get; } = DateTime.UtcNow;

    public Enrollment(Student student, Course course)
    {
        Student = student ?? throw new ArgumentNullException(nameof(student));
        Course  = course  ?? throw new ArgumentNullException(nameof(course));
    }
}
```

🧪 Հաճախակի սխալներ և հակաօրինակներ

- ❌ Public fields՝ `public int age;` → խախտում է encapsulation-ը:
  ✅ Օգտագործիր property + validation:
- ❌ Setter-ում լուռ սխալների անտեսում:
  ✅ Քաշե՛ քուր եղիր՝ `throw` համապատասխան exception:
- ❌ Static-ի չարաշահում՝ shared mutable state:
  ✅ Static օգտագործիր pure utility-ների համար:
- ❌ Null-երի չստուգում:
  ✅ `ArgumentNullException.ThrowIfNull`, nullable annotations (`?`), guard clauses:

💻 Լաբորատոր աշխատանք (30՛)

Լաբ 1: Rectangle Pro

Պահանջներ

- Class `Rectangle`՝ `Width`, `Height` (double, >0)՝ validation setter-ներում,
- Read-only `Area`, `Perimeter()` մեթոդ,
- `ToString()` → `"Rectangle 3x4 (Area=12, Perimeter=14)"`.

Ստուգում

- Սխալ արժեքների դեպքում `ArgumentOutOfRangeException`:
- Object initializer-ով ստեղծում ու տպում:

Լաբ 2: BankAccount Mini

Պահանջներ

- Class `BankAccount`՝ Owner, Number (read-only), Balance (read-only),

- `Deposit(amount)` և `Withdraw(amount)`՝ վավերացմամբ,
- Console demo՝ 3 գործողություն, վերջում տպել Balance:

Ստուգում

- Deposit/Withdraw՝ <=0 → սխալ,
- Withdraw > Balance → արգելել:

Լաբ 3: Student + Course + Enrollment (composition)

Պահանջներ

- `Student(Name, Age)` վավերացումներով,
- `Course(Title, Credits)` վավերացումներով,
- `Enrollment(Student, Course)` read-only `EnrolledAt`,
- Console demo՝ 2 ուսանող, 2 կուրս, 3 գրանցում, foreach-ով տպել:

Ստուգում

- Null student/course → throw,
- Credits միջակայք (1–15) enforced:

🏠 Տնային աշխատանք

1. **Point 2D (վարժ. value semantics)**

- Class `Point(int x, int y)`՝ read-only properties,
- Override `Equals`, `GetHashCode`, `ToString()`,
- Console demo՝ `p1.Equals(p2)` տարբեր դեպքեր:

2. **Email (nullability & normalize)**

- `Email(string? address)` - null → throw,
- Normalize՝ `Trim`, lower-case,
- `IsValid()` (պարզ regex կամ `Contains('@')` այս փուլում),
- `ToString()` վերադարձնում է հասցեն:

3. **LibraryBook (constructor chaining)**

- 3 կոնստրուկտոր՝ մեկ միմյանց կանչող,
- Վերնագիր/հեղինակ՝ trims + default, Year՝ >=1450 և ≤ ընթացիկ,
- Demo՝ երեք տարբեր ստեղծում methods:

4. **AppSettings (immutable/init-only)**

- `Environment` (Dev/Stage/Prod), `Version` (semantic string),
- Բոլորը `init;`, `ToString()`՝ `"Environment=Prod, Version=1.2.0"`: