C# NOTES

OOP, .NET Basics and Code Examples, explained simply

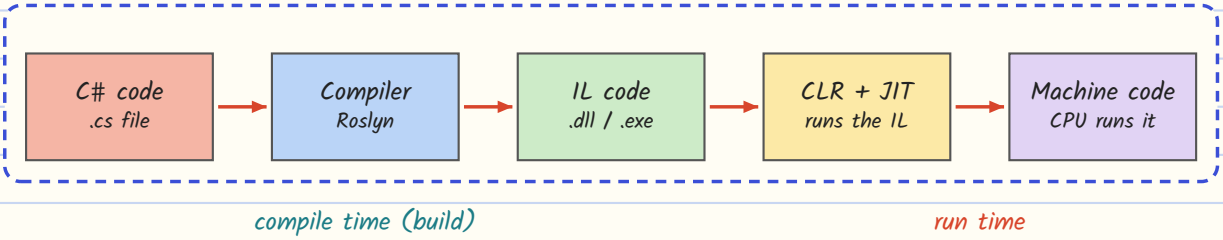
C# Notes · easy to understand · free for students

I. Start with C# and .NET

- C# is a modern, type-safe, object-oriented language made by Microsoft. It builds web apps, desktop apps, mobile apps and games.
- .NET is the platform that runs C# code. It gives the runtime and a big library of ready-made classes. C# is the language, .NET is the platform.

IDEA Think of a restaurant. C# is the recipe language, .NET is the kitchen, and the CLR is the chef who cooks your code.

Term	Job	Example
C#	The language you write code in	`class, method, loop`
.NET	Platform with runtime and libraries	`CLR, BCL, NuGet`
CLR	Runs your code and cleans memory (garbage collector)	`JIT, GC`
IL	Code made by the compiler, before machine code	`.dll, .exe`



- Code goes in an editor like Visual Studio or VS Code. Run it from the terminal with `dotnet run`.

EXAMPLE Hello World. Create it with dotnet new console, then dotnet run.

```
using System;

namespace HelloApp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, C#!");
        }
    }
}
// Newer .NET allows top-level statements. Just write:
// Console.WriteLine("Hello, C#!");
```

2. Variables, Data Types and Operators

- A variable is a named box that stores a value. C# is strongly typed: every variable has a type like `int` or `string`.

Type	Stores	Example
int	Whole number	`int age = 25;`
double	Decimal number	`double pi = 3.14;`
decimal	Exact decimal, good for money	`decimal price = 9.99m;`
char	One character	`char grade = 'A';`
string	Text	`string name = "Ravi";`
bool	true or false	`bool ok = true;`

EXAMPLE Variables, var, const and conversion.

```
int x = 10;
var y = 20.5;               // type found by the compiler: double
const double PI = 3.14159;  // never changes
int? maybe = null;          // nullable int
int result = maybe ?? 0;    // use 0 if maybe is null
string msg = $"x = {x}";    // string interpolation

double d = x;               // implicit conversion
int n = (int)3.9;           // explicit cast, n = 3
int a = int.Parse("42");
bool ok = int.TryParse("abc", out int z);   // ok = false
```

Value type and reference type

- Value types (`int`, `double`, `bool`, `struct`) hold the data directly. Assigning copies the value.
- Reference types (`class`, `string`, arrays) hold an address. Two variables can point to the same object.

REMEMBER `int` is a value. `Student s = new Student();` means `s` holds a reference. null means no object. Value types cannot be null unless you write `int?`.

Kind	Operators	Example
Arithmetic	`+ - * / %`	`a % b`
Compare	`== != < > <= >=`	`a >= b`
Logical	`&& || !`	`a > 1 && b < 5`
Assign	`= += -= *= /=`	`x += 2;`
Null	`?? ?. ??=`	`name ?? "none"`

3. Control Flow and Loops

- *if / else* chooses a path. *switch* picks one of many cases. Loops repeat code: *for*, `while`, `do while` and *foreach*.

EXAMPLE Decisions.

```
if (marks >= 90) Console.WriteLine("A");
else if (marks >= 60) Console.WriteLine("B");
else Console.WriteLine("C");

string grade = marks switch {     // switch expression
    >= 90 => "A",
    >= 60 => "B",
    _     => "C"
};
```

EXAMPLE Loops.

```
for (int i = 0; i < 5; i++) Console.WriteLine(i);

int j = 0;
while (j < 3) { Console.WriteLine(j); j++; }

foreach (var item in new[] { 1, 2, 3 })
    Console.WriteLine(item);
```

Keyword	What it does
`break`	Leave the loop completely
`continue`	Skip to the next turn of the loop
`return`	Leave the method (and give a value back)

4. Methods, Arrays and Strings

- A *method* is a block of code with a name. It can take parameters and give back a return value.
- *Overloading* means same method name with a different parameter list.

EXAMPLE Different kinds of methods.

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

static void Greet(string name = "Guest")        // optional parameter
{ Console.WriteLine($"Hello {name}"); }

static void Swap(ref int a, ref int b)           // ref: change the original
{ int t = a; a = b; b = t; }

static bool TryDivide(int a, int b, out int r)   // out: give an extra result
{ r = 0; if (b == 0) return false; r = a / b; return true; }
```

Arrays and strings

EXAMPLE Common operations.

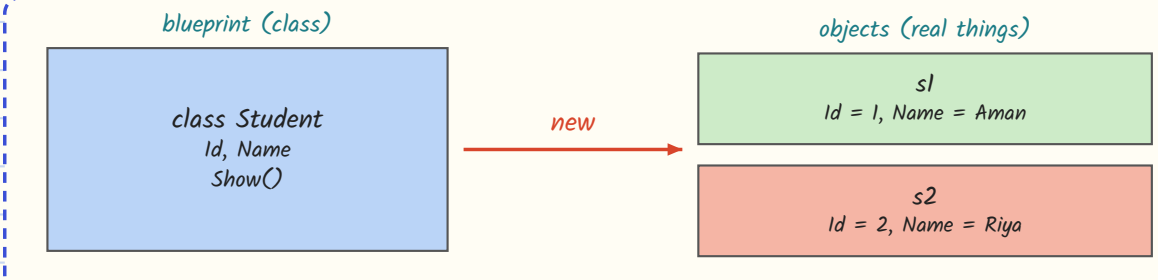
```
int[] nums = { 5, 3, 8, 1 };
Array.Sort(nums);                // 1 3 5 8
Console.WriteLine(nums.Length);  // 4

string s = "Hello World";
s.ToUpper();  s.Trim();  s.Contains("World");
s.Substring(0, 5);               // "Hello"
s.Replace("World", "C#");
s.Split(' ');
```

REMEMBER A string is immutable: every change makes a new string. For many changes use `StringBuilder`.

5. Classes and Objects

- A *class* is a blueprint. An *object* is a real thing made from it using new.
- OOP has four pillars: *Encapsulation*, *Inheritance*, *Polymorphism* and *Abstraction*.



Member	Meaning
Field	Variable inside a class
Property	A field with get and set rules
Method	An action the object can do
Constructor	Runs when the object is created. Same name as the class
static	Belongs to the class, not to one object

EXAMPLE A Student class with constructors.

```
public class Student
{
    public int Id { get; set; }              // auto property
    public string Name { get; set; }

    public Student() { }                      // default constructor
    public Student(int id, string name)       // parameterized constructor
    {
        this.Id = id;
        this.Name = name;
    }

    public void Show() => Console.WriteLine($"{Id} - {Name}");
}

var s1 = new Student(1, "Aman");
var s2 = new Student { Id = 2, Name = "Riya" };   // object initializer
s1.Show();
```

Modifier	Who can use it
public	Everywhere
private	Same class only (default for members)
protected	Same class and child classes
internal	Same project (assembly)

REMEMBER Keep fields private and open them through properties. this means the current object.

6. Encapsulation and Properties

- Encapsulation means hiding data inside the class and giving safe access with properties and methods.

EXAMPLE *A safe bank account.*

```
public class BankAccount
{
    private decimal balance;                // hidden

    public decimal Balance => balance;      // read only from outside

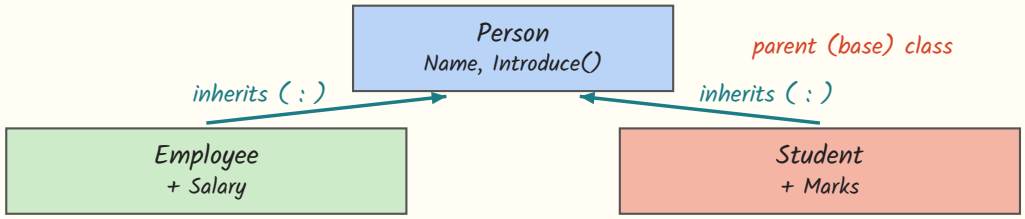
    public void Deposit(decimal amount)
    {
        if (amount <= 0) throw new ArgumentException("Invalid amount");
        balance += amount;
    }
}
```

EXAMPLE *Property styles.*

```
public string Name { get; set; }            // read and write
public int Id { get; private set; }         // only the class can set
public string City { get; init; }           // set once at creation
public double Area => Width * Height;       // calculated
```

7. Inheritance

- *Inheritance lets a child class reuse the members of a parent class. Use : to inherit.*
- *C# allows only one parent class. Multiple inheritance is done with interfaces.*



EXAMPLE *Employee inherits Person.*

```
public class Person
{
    public string Name { get; set; }
    public Person(string name) { Name = name; }
    public void Introduce() => Console.WriteLine($"I am {Name}");
}

public class Employee : Person
{
    public double Salary { get; set; }
    public Employee(string name, double salary) : base(name)   // parent constructor
    { Salary = salary; }
}

new Employee("Neha", 50000).Introduce();     // inherited method
```

Keyword	Use
`:`	Inherit from a parent class
`base`	Call the parent constructor or method
`protected`	Child classes can use it
`sealed`	Nobody can inherit from this class

IDEA Use the is-a test. An Employee is a Person, so inherit. A Car has an Engine, so use a field, not inheritance.

8. Polymorphism and Abstraction

- Polymorphism means one name, many forms. Overloading is decided at compile time. Overriding is decided at run time.

Compile time	Run time
Method overloading	Method overriding
Same name, different parameters	virtual in parent, override in child
Compiler decides	CLR decides from the real object

EXAMPLE Overriding with virtual and override.

```
public class Shape { public virtual double Area() => 0; }

public class Circle : Shape
{
    public double R { get; set; }
    public override double Area() => Math.PI * R * R;
}
public class Rectangle : Shape
{
    public double W { get; set; }
    public double H { get; set; }
    public override double Area() => W * H;
}

Shape s = new Circle { R = 5 };
Console.WriteLine(s.Area());     // Circle.Area() runs
```

➤ Abstraction

- Abstraction shows only what is needed and hides the details. An `abstract` class cannot be created with `new`.

EXAMPLE *Abstract class.*

```
public abstract class Animal
{
    public abstract void Sound();                          // no body
    public void Sleep() => Console.WriteLine("Zzz");       // normal method
}
public class Dog : Animal
{
    public override void Sound() => Console.WriteLine("Bark");
}
Animal a = new Dog();
a.Sound();                  // Bark
```

9. Interfaces, Enums and Records

- *An interface is a contract. A class that implements it must provide every member. A class can implement many interfaces.*

EXAMPLE *Interface in action.*

```
public interface IPayment { void Pay(decimal amount); }

public class UpiPayment : IPayment
{ public void Pay(decimal amount) => Console.WriteLine($"Paid {amount} by UPI"); }

public class CardPayment : IPayment
{ public void Pay(decimal amount) => Console.WriteLine($"Paid {amount} by card"); }

IPayment p = new UpiPayment();
p.Pay(500);
```

	Abstract class	Interface
Inherit from	Only one	Many
Fields, constructors	Yes	No
Best for	Shared code, is-a	A contract, can-do
Members	Any access level	Public by default

EXAMPLE *struct, enum and record.*

```
struct Point { public int X, Y; }               // value type

enum Day { Mon = 1, Tue, Wed }                  // named constants
Day d = Day.Tue;                                // (int)d is 2

public record Person(string Name, int Age);     // short, value equality
var p1 = new Person("Ali", 20);
var p2 = p1 with { Age = 21 };                  // copy with a change
```

10. Collections, Generics and LINQ

Collection	Use
`List<T>`	Growing list, access by index
`Dictionary<K,V>`	Key and value pairs, fast lookup
`HashSet<T>`	Only unique items
`Queue<T>`	First in, first out
`Stack<T>`	Last in, first out

EXAMPLE List and Dictionary.

```
using System.Collections.Generic;

var list = new List<int> { 5, 2, 9 };
list.Add(7);  list.Remove(2);  list.Sort();

var dict = new Dictionary<string, int> { ["a"] = 1, ["b"] = 2 };
if (dict.TryGetValue("a", out int val)) Console.WriteLine(val);

var set = new HashSet<int> { 1, 1, 2 };      // {1, 2}
```

Generics

- Generics let one class or method work with any type, safely. `T` is a placeholder for the type.

EXAMPLE A generic class and method.

```
public class Box<T> { public T Value { get; set; } }

static T Max<T>(T a, T b) where T : IComparable<T>
    => a.CompareTo(b) > 0 ? a : b;

var box = new Box<string> { Value = "Hi" };
Console.WriteLine(Max(3, 8));      // 8
```

LINQ

- LINQ lets you filter, sort and shape collections with short readable code.

EXAMPLE LINQ methods.

```
using System.Linq;
int[] nums = { 1, 2, 3, 4, 5, 6 };

var evens   = nums.Where(n => n % 2 == 0);       // 2, 4, 6
var squares = nums.Select(n => n * n);           // 1, 4, 9 ...
int total   = nums.Sum();                        // 21
var sorted  = nums.OrderByDescending(n => n);
bool any    = nums.Any(n => n > 5);              // true
```

11. Exceptions and File I/O

- An *exception* is a run time error. Handle it with `try`, `catch` and `finally` so the program does not crash.

EXAMPLE Handle errors.

```
try
{
    int x = int.Parse("abc");
}
catch (FormatException ex)
{
    Console.WriteLine("Bad format: " + ex.Message);
}
catch (Exception ex)               // general catch goes last
{
    Console.WriteLine(ex.Message);
}
finally
{
    Console.WriteLine("Always runs");
}
```

Keyword	Meaning
`try`	Code that may fail
`catch`	Handle the error
`finally`	Always runs, good for cleanup
`throw`	Raise your own error

EXAMPLE Read and write files.

```
using System.IO;

File.WriteAllText("a.txt", "Hello");
string text = File.ReadAllText("a.txt");
File.AppendAllText("a.txt", "\nMore");

using var sr = new StreamReader("a.txt");     // closes the file for you
string line;
while ((line = sr.ReadLine()) != null) Console.WriteLine(line);
```

REMEMBER `using` disposes files and connections automatically, even when an error happens.

12. Delegates, Lambda and Async

- A *delegate* is a type-safe pointer to a method. A *lambda* is a short method written as `x => x * 2`. An *event* tells other code that something happened.

Delegate	Returns
`Func<T, R>`	A value
`Action<T>`	Nothing
`Predicate<T>`	true or false

EXAMPLE Func, Action and event.

```
Func<int, int, int> add = (a, b) => a + b;       // returns a value
Action<string> print = s => Console.WriteLine(s);
print("Sum is " + add(2, 3));

public class Button
{
    public event EventHandler Clicked;
    public void Click() => Clicked?.Invoke(this, EventArgs.Empty);
}
var btn = new Button();
btn.Clicked += (s, e) => Console.WriteLine("Clicked!");
btn.Click();
```

➤ async and await

- Use **async** and **await** for slow work like web calls and files. The thread is not blocked while it waits.

EXAMPLE Download data without freezing the app.

```
static async Task<string> GetDataAsync(string url)
{
    using var client = new HttpClient();
    return await client.GetStringAsync(url);      // waits without blocking
}

string data = await GetDataAsync("https://example.com");
await Task.WhenAll(Task.Delay(1000), Task.Delay(1000));   // run together
```

13. Projects and Practice

Project	What you practice
1. Calculator console app	Variables, operators, switch
2. Student record manager	Classes, List<T>, properties
3. Bank account system	Encapsulation, exceptions
4. Shape area calculator	Inheritance, polymorphism
5. Library management	Interfaces, LINQ, collections
6. File based to-do app	File I/O, methods, menus
7. Weather or API reader	async await, HttpClient, JSON

Common mistakes

Mistake	Fix
Using == to compare objects	Override Equals, or compare the fields
Forgetting new before a class	Create the object: new Student()
Catching Exception everywhere	Catch the specific error first
Many string + in a loop	Use StringBuilder
Not closing files	Use a using statement

Practice questions

1. What is the difference between value type and reference type?
2. Explain the four pillars of OOP with a small example.
3. Abstract class vs interface. When do you use which?
4. What is the difference between overloading and overriding?
5. What is the difference between ref and out?
6. What is boxing and unboxing? Why is it slow?
7. String vs StringBuilder. When do you use each?
8. Explain garbage collection and the using statement.
9. What is LINQ? Write a query to get even numbers.
10. How does async and await avoid blocking the thread?

- Build one small console project every week. Rewrite old code with classes and LINQ to learn faster.