**Mutable vs Immutable Objects in C#**

In C#, objects can be **mutable** or **immutable** depending on whether their state can be changed after creation.

**1 Mutable Objects**

A **mutable** object is one whose state (fields/properties) can be modified after it is created.

**Example of a Mutable Class**

csharp

CopyEdit

class Person

{

public string Name { get; set; } // This property can be modified

public int Age { get; set; } // This property can be modified

public Person(string name, int age)

{

Name = name;

Age = age;

}

}

class Program

{

static void Main()

{

Person p = new Person("Alice", 30);

Console.WriteLine($"Before: {p.Name}, {p.Age}");

// Modifying the object properties

p.Name = "Bob";

p.Age = 35;

Console.WriteLine($"After: {p.Name}, {p.Age}");

}

}

**🔹 Key Points about Mutable Objects**

* Properties or fields can be modified after the object is created.
* Useful when you need to change object state frequently.
* Can lead to unintended side effects in multi-threaded environments.

**2️ Immutable Objects**

An **immutable** object is one whose state cannot be modified after it is created.

**Example of an Immutable Class**

csharp

CopyEdit

class ImmutablePerson

{

public string Name { get; }

public int Age { get; }

public ImmutablePerson(string name, int age)

{

Name = name;

Age = age;

}

}

class Program

{

static void Main()

{

ImmutablePerson p = new ImmutablePerson("Alice", 30);

Console.WriteLine($"{p.Name}, {p.Age}");

// The following line would cause a compilation error because properties are read-only

// p.Name = "Bob";

}

}

**🔹 Key Points about Immutable Objects**

* **Properties are read-only (get;) and set only via constructor.**
* **No modification after creation** – a new object must be created for any changes.
* **Thread-safe** – multiple threads can safely access immutable objects.
* **Better for functional programming and value objects (e.g., string, DateTime).**

**3️ Real-World Examples**

**Immutable Types in C#**

* string

string str = "Hello";

str += " World"; // Creates a new string object, does not modify the original

* DateTime

DateTime dt = DateTime.Now;

DateTime newDt = dt.AddDays(5); // Returns a new DateTime object

* Record (Introduced in C# 9)

record PersonRecord(string Name, int Age); // Immutable by default

**When to Use Each?**

| **Feature** | **Mutable Objects** | **Immutable Objects** |
| --- | --- | --- |
| **Performance** | Faster when modifying | New objects created on change |
| **Thread-Safety** | Risk of race conditions | Safe for multi-threading |
| **Memory Usage** | Can be lower | May allocate more memory |
| **Use Case** | Data structures, UI models | Value types, DTOs, caching |

**Conclusion**

* **Use mutable objects** when frequent modifications are needed.
* **Use immutable objects** for safety, simplicity, and functional programming.