

أنواع الـ Classes في #C

في المقالة دي، هنشرح كل أنواع الـ Classes في #C:

Concrete, Static, Abstract, Sealed, Partial, Nested, Generic, Record, Anonymous.

هنغطي كل التفاصيل: التعريف، الخصائص، الاستخدامات، المميزات، العيوب، أمثلة عملية، واستخدامات واقعية.

1. Concrete Class في #C

الـ **Concrete Class** هي الـ Class العادية اللي تقدر تنشئ منها كائنات (Objects) باستخدام كلمة **new**. هي فئة كاملة التنفيذ (Fully Implemented)، مش abstract أو sealed، وبحتوي على أعضاء زي Fields, Methods, Properties. تقدر تورثها أو ترث منها.

خصائصها:

- تقدر تنشئ كائنات منها مباشرة.
- تحتوي على تنفيذ كامل لكل الأعضاء.
- مش sealed، فتقدر تتورث.
- تقدر تحتوي على static و non-static members.

مثال:

```
public class Car {  
    public string Model { get; set; }  
    public void Drive() {  
        Console.WriteLine($"Driving {Model}...");  
    }  
}  
  
// استخدامها
```

```
Car myCar = new Car { Model = "Toyota" };  
myCar.Drive(); // يطبع: Driving Toyota...
```

مميزات:

- سهولة الاستخدام لتمثيل كائنات حقيقية (زي Car أو Employee).
- مرنة، تقدر تستخدمها كـ base class أو standalone.
- تدعم كل أنواع الأعضاء (static و non-static).

عيوب:

- لو محتاج قيود على التوريث، هتحتاج sealed class.
- لو محتاج تنفيذ جزئي، هتحتاج abstract class.

استخدامات واقعية:

- هتستخدمها بقي في كل حاجة طول ما مفيش Business Case تخليك تستخدم نوع ثاني.

الفرق عن غيرها:

- **عن Abstract:** ال Concrete مش بتحتوي على abstract methods وتقدر تنشئ منها كائنات.
- **عن Static:** تقدر تحتوي على instance members، بينما ال Static بس static.

2. Static Class في #C

الـ **Static Class** هي فئة متقدرش تنشئ منها Objects، وبتحتوي بس على أعضاء static (مثل methods و fields). بتُستخدم عادةً لتجميع الـ utility functions أو الثوابت، وهي تلقائيًا، فمتقدرش تورثها أو ترث منها.

خصائصها:

- مش تقدر تستخدم **new** عليها. بما ان مبيتعملش منها Instances.
- كل الـ Members لازم تكون static.
- مينفعش تحتوي على instance members.
- تلقائيًا sealed، فمش تقدر تتورث.
- تقدر تحتوي على static constructor للتهيئة.

مثال:

```
public static class MathHelper {
    public static string AppName = "CalculatorApp";
    static MathHelper() {
        AppName = "Advanced " + AppName;
    }
    public static int Add(int a, int b) => a + b;
}
// استخدامها
Console.WriteLine(MathHelper.AppName); // Advanced
CalculatorApp
Console.WriteLine(MathHelper.Add(5, 3)); // 8
```

مميزات:

- مثالية للـ utility methods زي فئات **Math** أو **DateTime**.
- الأداء عالي لأنها مش بتحتاج إنشاء كائنات.
- سهولة الوصول باستخدام اسم الكلاس مباشرة.

عيوب:

- مش مناسبة للكائنات اللي بتحتاج حالة (state).
- محدودة لأنها مش بتدعم التوريث أو instance members.

استخدامات واقعية:

- إنشاء أدوات مساعدة (زي تحويل التواريخ أو الحسابات الرياضية).
- تخزين ثوابت عامة للمشروع (زي إعدادات لحاجة معينة).

3. Abstract Class في C#

ال **Abstract Class** هي فئة غير مكتملة متقدرش تنشئ منها Objects، وبتستخدم كـ base class لفئات تانية. بتحتوي على abstract methods (بدون تنفيذ) و non-abstract methods (بتنفيذ). لازم تتورثها عشان تستخدمها.

خصائصها:

- متقدرش تستخدم **new** عليها.
- تقدر تحتوي على abstract و virtual methods.
- تقدر تحتوي على fields, properties, constructors.
- مش بتدعم multiple inheritance. يعني مفيش class يقدر يورث من اكر من abstract class.
- تقدر تورث من أي كلاس عادي لكن لازم تورث منها عشان تستفيد منها ومن اللي ورثته.

مثال:

```
public abstract class Animal {  
    public abstract void Speak();  
    public void Eat() => Console.WriteLine("Eating...");  
}
```

```
public class Dog : Animal {
    public override void Speak() =>
        Console.WriteLine("Woof!");
}
// استخدامها
Dog dog = new Dog();
dog.Speak(); // Woof!
dog.Eat(); // Eating...
```

مميزات:

- مثالية لتحديد قواعد للكلاسات التي هتورث منها.
- تدعم تنفيذ مشترك (shared implementation) من خلال non-abstract members.
- مرنة لأنها بتدعم الوراثة والتعديل (Polymorphism).

عيوب:

- متقدريش تنشئ كائنات مباشرة.
- لو التنفيذ معقد، ممكن يزيد تعقيد الكود.

استخدامات واقعية:

- تصميم base classes في ألعاب الفيديو (مثل Character يتورث لـ Player و Enemy).
- تصميم base controllers في ASP.NET MVC لتوحيد الـ logic (مثل BaseController لإدارة الـ CRUD).

4. Sealed Class في #C

ال **Sealed Class** هي فئة تمنع التوريث منها، عشان تحمي التنفيذ أو تحسن الأداء. تقدر تنشئ منها Objects، لكن متقدرش تعمل subclass.

خصائصها:

- تستخدم كلمة **sealed**.
- تقدر تنشئ Objects.
- متقدرش تتورث.
- تقدر تكون virtual methods أو properties مختومة.

مثال:

```
public sealed class Logger {  
    public void Log(string msg) => Console.WriteLine(msg);  
}  
// استخدامها  
Logger logger = new Logger();  
logger.Log("Error occurred"); // Error occurred
```

مميزات:

- بتحسن الأداء لأن ال CLR بيعرف إن مفيش توريث.
- بتحمي الكود من التعديل غير المرغوب.
- مثالية لل libraries اللي عايزة تحد من التوسع.

عيوب:

- تحد من المرونة لو احتجت توريث لاحقاً.
- مش مناسبة لو عايز تصميم قابل للتوسع.

استخدامات واقعية:

- إنشاء خدمات أمان (Authentication Services) في Web API.
- حماية الكود في تطبيقات حساسة (مثل الـ logging أو security).

5. Partial Class في #C

الـ **Partial Class** تسمح بتقسيم تعريف الـ Class عبر ملفات متعددة، عشان تسهل التعاون أو التعامل مع كود مولد تلقائيًا.

خصائصها:

- تستخدم كلمة **partial** في كل جزء.
- كل الأجزاء لازم تكون في نفس الـ namespace و assembly.
- كل الأعضاء متاحة في كل الأجزاء.
- لو جزء **sealed** أو **abstract**، الكل هيبقى كده.

مثال:

```
// File: Person1.cs
public partial class Person {
    public string FirstName { get; set; }
}

// File: Person2.cs
public partial class Person {
    public string LastName { get; set; }
    public string FullName => $"{FirstName} {LastName}";
}

// استخدامها
Person p = new Person { FirstName = "Ahmed", LastName = "Ali" };
Console.WriteLine(p.FullName); // Ahmed Ali
```

مميزات:

- تسهل فصل الكود (مثل UI من business logic).
- مثالية مع الكود المولد تلقائيًا (زي في LINQ to SQL).
- تتيح لأكثر من مطور يشتغلوا على نفس ال Class.

عيوب:

- ممكن تزيد التعقيد لو التقسيم مش منظم.
- كل الأجزاء لازم تكون في نفس ال assembly.

استخدامات واقعية:

- فصل الكود في ASP.NET MVC (مثل Controller.cs و Controller.designer.cs).
- مع LINQ to SQL لإضافة properties مخصصة.

مثال LINQ to SQL:

```
// Person.designer.cs (مولد تلقائيًا)
public partial class Person {
    public int Id { get; set; }
    public string Name { get; set; }
    public DateTime DateOfBirth { get; set; }
}

// PersonExtension.cs
public partial class Person {
    public int Age => DateTime.UtcNow.Year -
DateOfBirth.Year;
}
```


6. Nested Class في C#

ال Nested Class هي فئة معرفة داخل فئة ثانية (outer class)، عشان ترتبط بال outer class وتخفي التفاصيل.

خصائصها:

- تقدر تكون public, private, protected, إلخ.
- ال Inner Class تقدر توصل لأعضاء ال Outer Class (حتى private).
- مش تقدر تنشئ inner خارج ال outer إلا لو public.
- تقدر تكون static أو non-static.

مثال:

```
public class Outer {  
    private string secret = "Hidden";  
    public class Inner {  
        public void Show(Outer outer) =>  
        Console.WriteLine(outer.secret);  
    }  
}  
  
// استخدامها  
Outer outer = new Outer();  
Outer.Inner inner = new Outer.Inner();  
inner.Show(outer); // Hidden
```

مميزات:

- تساعد في encapsulation لل helper classes.
- تنظم الكود عن طريق تجميع ال logic المتعلق.
- تحسن readability لو استخدمت صح.

عيوب:

- الإفراط في استخدامها ممكن يعقد الكود.
- لو كانت public بشكل غير ضروري، بتكشف تفاصيل داخلية.

استخدامات واقعية:

- كلاس ال Node داخل LinkedList.
- Helper classes في أنظمة معقدة زي compilers.

7. Generic Class في #C

ال **Generic Class** تسمح بإنشاء فئات مرنة تعمل مع أنواع بيانات مختلفة بطريقة type-safe.

خصائصها:

- تستخدم **<T>** أو constraints زي **where T : class**.
- تمنع boxing/unboxing.

مثال:

```
public class Box<T> {
    public T Content { get; set; }
}
// استخدامها
Box<int> intBox = new Box<int> { Content = 10 };
Box<string> strBox = new Box<string> { Content = "Hello" };
Console.WriteLine(intBox.Content); // 10
Console.WriteLine(strBox.Content); // Hello
```

مميزات:

- **Type Safety**: تمنع إضافة types غير صحيحة.
- **Performance**: تقلل boxing/unboxing.
- **Code Reuse**: نفس الكود يشتغل مع أنواع مختلفة.

عيوب:

- ممكن تكون معقدة في التصميم لو فيه constraints كثير.
- مش مناسبة لو الكود مش محتاج مرونة ال types.

استخدامات واقعية:

- Collections زي `List<T>` و `Dictionary<TKey, TValue>`.
- أنظمة caching أو data structures مخصصة.

8. Record Class في C# (من 9 C#)

ال **Record Class** هي فئة immutable بطبيعتها، بتدعم مقارنة بالقيمة و `with` expressions. متاحة من 9 C# و 5 .NET+.

خصائصها:

- Immutable افتراضياً (init-only properties).
- Built-in equality comparison.
- تدعم `with` expression لإنشاء نسخ معدلة.
- مش متاحة في .NET Framework القديم.

مثال:

```
public record Person(int Id, string FirstName, string
LastName);
Person p1 = new Person(1, "Ahmed", "Ali");
Person p2 = p1 with { LastName = "Mohamed" };
Console.WriteLine(p2); // Person { Id = 1, FirstName =
Ahmed, LastName = Mohamed }
```

مميزات:

- مثالية لـ DTOs و data models.
- تقلل الكود المكرر بفضل `with` expression.
- تدعم equality comparison تلقائياً.

عيوب:

- غير مدعومة في .NET Framework القديم.
- محدودة لو محتاج mutability.

استخدامات واقعية:

- تمثيل بيانات في APIs (مثل User أو Order).
- أنظمة immutable data في تطبيقات مالية.

9. Anonymous Class في C#

الـ **Anonymous Class** هي فئة بدون اسم، تُستخدم للبيانات المؤقتة، خاصة في LINQ.

خصائصها:

- .read-only properties
- مش تقدر تورثها.
- النوع inferred تلقائيًا باستخدام **var**.
- لا تحتوي على methods أو static members.
- تحتوي بس على public fields.

مثال:

```
var person = new { Name = "John", Age = 30 };
Console.WriteLine($"Name: {person.Name}, Age: {person.Age}"); // Name: John, Age: 30
```

مميزات:

- مثالية للـ temporary data في LINQ queries.
- تقلل الكود لو محتاج هيكل بيانات سريع.
- تحسن readability لو استخدمت صح.



















عيوب:

- محدودة لأنها read-only وبدون methods.
- مش مناسبة للبيانات طويلة الأمد.

استخدامات واقعية:

- Grouping في LINQ queries.
- تمثيل بيانات مؤقتة في تقارير.

جدول ملخص

Class Type	Instantiable	Inheritable	Use Case
Concrete	Yes 	Yes 	Standard object with full definition
Static	No 	No 	Helper methods, constants
Abstract	No 	Yes 	Base for other classes
Sealed	Yes 	No 	Prevents inheritance
Partial	Yes 	Yes 	Split class across files
Nested	Yes 	Yes 	Encapsulated logic
Generic	Yes 	Yes 	Type-safe reusable components
Record	Yes 	Yes 	(+Immutable data models (C# 9
Anonymous	Yes 	No 	Temporary data object

Sources:

[Types of Classes in C# with Examples | Concrete, Static, Abstract, Sealed, and More](#)

[What is a Static Class in C#?](#)

[Abstract Class In C#](#)

[Sealed Class in C#](#)

[Partial class in C#](#)

[Generics in C#](#)

[Nested class in C#](#)

[Record class in C#](#)

[Anonymous Classes in C#](#)