Part01

```
using System;

struct Point
{
    public int X;
    public int Y;

    public Point()
    {
        X = 0;
        Y = 0;
    }

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

    public override string ToString()
    {
        return $"({X}, {Y})";
    }
}
```

```
class Program
{
   static void Main()
   {
      Point p1 = new Point();
      Point p2 = new Point(5, 10);

      Console.WriteLine(p1); // (0, 0)
      Console.WriteLine(p2); // (5, 10)
   }
}
```

الـ **struct** في C# نوع **Value Type**، وكل الـ Value Types بتورث بشكل مباشر من **System.ValueType** فقط، وده معناه إنك ماينفعش تخلي struct يرث من struct أو class تاني، لأن الميراث في الـ Value Types غير مسموح، لكن ممكن يعمل **Implement** للـ Interfaces

---

```
using System;

public class TypeA
{
   private int F;
   internal int G;
   public int H;

   public TypeA(int f, int g, int h)
   {
      F = f;
```

```
        G = g;
        H = h;
    }

    public void ShowF()
    {
        Console.WriteLine($"F = {F}");
    }
}

class Program
{
    static void Main()
    {
        TypeA obj = new TypeA(1, 2, 3);
        obj.ShowF();
        Console.WriteLine($"G = {obj.G}");
        Console.WriteLine($"H = {obj.H}");
    }
}
```

- **private** → العضو بيكون متاح جوه نفس الكلاس فقط.
- **internal** → العضو بيكون متاح داخل نفس الـ project (أو الـ assembly).
- **public** → العضو بيكون متاح في أي مكان حتى خارج المشروع.

```
using System;

struct Employee
{
   private int EmpId;
   private string Name;
   private double Salary;

   public Employee(int id, string name, double salary)
   {
      EmpId = id;
      Name = name;
      Salary = salary;
   }

   public string GetName()
   {
      return Name;
   }

   public void SetName(string name)
   {
      Name = name;
   }

   public double SalaryProperty
```

```
    {
        get { return Salary; }
        set { Salary = value; }
    }

    public int EmpIdProperty
    {
        get { return EmpId; }
        set { EmpId = value; }
    }
}

class Program
{
    static void Main()
    {
        Employee emp = new Employee(1, "Ali", 5000);

        Console.WriteLine(emp.GetName());
        emp.SetName("Ahmed");
        Console.WriteLine(emp.GetName());

        emp.SalaryProperty = 6000;
        Console.WriteLine(emp.SalaryProperty);
    }
}
```

الـ **Encapsulation** مهمة لأنها:

1. بتحافظ على البيانات من التغيير العشوائي أو غير المصرح به.
2. بتديك تحكم كامل في طريقة الوصول أو التعديل على البيانات.
3. بتخلي الكود أسهل في الصيانة والتطوير.
4. بتحافظ على مبدأ الـ **Data Hiding** وبالتالي تقلل الأخطاء.

```
using System;


struct Point
{
   public int X;
   public int Y;


   public Point(int x)
   {
      X = x;
      Y = 0;
   }


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

```
class Program
{
  static void Main()
  {
    Point p1 = new Point(5);
    Point p2 = new Point(10, 20);

    Console.WriteLine($"P1: ({p1.X}, {p1.Y})");
    Console.WriteLine($"P2: ({p2.X}, {p2.Y})");
  }
}
```

الـ **Constructors** في الـ struct هي دوال خاصة بيتم استدعاؤها تلقائي وقت إنشاء الكائن (object) من الـstruct ، وظيفتها تهيئة القيم الابتدائية للمتغيرات.(fields)

- لازم كل الـ fields تتحدد قيمها في الـconstructor.
- في الـ struct ينفع تعمل **parameterized constructor** لكن ماينفعش تكتب **default constructor** مخصص #C) بيعمله تلقائيًا.(

```
using System;

struct Point
{
  public int X;
  public int Y;

  public Point(int x, int y)
  {
```

```
    X = x;
    Y = y;
  }
  public override string ToString()
  {
    return $"Point Coordinates => X: {X}, Y: {Y}";
  }
}
class Program
{
  static void Main()
  {
    Point p1 = new Point(5, 10);
    Point p2 = new Point(20, 30);
    Point p3 = new Point(-5, 15);

    Console.WriteLine(p1);
    Console.WriteLine(p2);
    Console.WriteLine(p3);
  }
}
```

عمل **Override**للـ ()ToStringبيخلي عرض البيانات أوضح وأسهل قراءة بدل ما يظهر اسم النوع فقط.

- بيقدم **تنسيق مخصص** يوضح المعلومات المهمة مباشرة.
- بيخلي الكود أو النتائج في الكونسول مفهومة حتى بدون شرح إضافي.
- بيساعد في تتبع الأخطاء وفهم القيم أثناء الـdebugging بسرعة.

```
using System;

struct Point
{
    public int X;
    public int Y;
}

class Employee
{
    public string Name;
    public double Salary;
}

class Program
{
    static void ChangePoint(Point p)
    {
        p.X = 100;
        p.Y = 200;
    }

    static void ChangeEmployee(Employee e)
    {
        e.Name = "Updated";
        e.Salary = 9999;
```

```
    }

    static void Main()
    {
        Point pt = new Point { X = 10, Y = 20 };
        Employee emp = new Employee { Name = "Ali", Salary = 5000 };

        ChangePoint(pt);
        ChangeEmployee(emp);

        Console.WriteLine($"Point: X={pt.X}, Y={pt.Y}");
        Console.WriteLine($"Employee: Name={emp.Name}, Salary={emp.Salary}");
    }
}
```

→ **Struct (Value Type)** بيتخزن في **Stack**، ولما تبعته لميثود بيتنسخ نسخة مستقلة، وأي تعديل عليها ما بيأثرش على النسخة الأصلية.

→ **Class (Reference Type)** المؤشر بيتخزن في **Stack** لكن البيانات نفسها في **Heap**، ولما تبعته لميثود بيتبعت المرجع(reference) ، فأي تعديل بيأثر على الأصل.

Part 02

تخيّلي إنتِ عندك دفتر فيه كل معلوماتك: اسمك، سنك، وهواياتك.
الـ **Copy Constructor** هو زي لما تدي الدفتر ده لصحابك وتقولهم:
"انسخوا كل حاجة فيه في دفتر جديد بنفس التفاصيل، نفس الخط، نفس الألوان."

الفكرة إنه بدل ما نكتب البيانات من أول وجديد (ونقعد نعيد نفس الشغل)، بنعمل نسخة طبق الأصل من كائن موجود أصلاً، ونشتغل عليها براحتنا.
الميزة بقى إن النسخة دي مستقلة، يعني لو عدلتي في واحدة مش هتبوظ التانية.

الـ **Copy Constructor** = "اعمل لي كائن جديد، يكون نسخة طبق الأصل من كائن تاني موجود."

Wafaa Mohammed • أنت
–
الآن •

عمرك سألت نفسك: إزاي أول ما تنشئ Object في البرمجة تلاقيه جالك جاهز بالقيم اللي انت عاوزها؟ 🤔
السر هنا اسمه Constructor... ده مش أي دالة، ده زي "الممر السحري" اللي بيدخل منه الكائن عشان يتكوّن ويتظبط من أول لحظة.

💡 يعني إيه Constructor؟
هو دالة خاصة في الكلاس أو الستركت بتشتغل أوتوماتيك أول ما تعمل Object، هدفها تهيأ القيم الأولية وتجهز الكائن قبل ما تبدأ تستخدمه.
مفيش داعي تنادي عليها بنفسك... هي بتشتغل لوحدها أول ما تولد الكائن.

🔍 أنواع الـ Constructor:

Default Constructor
بيولّد الكائن بالقيم الافتراضية، حتى لو انت ما كتبتوش، الكومبايلر بيعمله لوحده.

Parameterized Constructor 🎯
هنا بقى انت اللي بتحدد القيم أثناء إنشاء الكائن.
مثال: "اعمل لي عربية لونها أحمر وسرعتها ٢٠٠" بدل ما يجيبهالك عشوائي.

Copy Constructor 📋
ده زي ما قولنا قبل كده... "انسخلي الكائن ده وحطهولي في واحد جديد بالظبط".

Static Constructor ⚡
بيشتغل مرة واحدة بس طول عمر البرنامج، وبيستخدم لتهيئة البيانات المشتركة بين كل الكائنات.

✨ ليه الـ Constructor مهم أوي؟
بيخليك تبدأ بالكائن جاهز بدل ما تفضل تهيأه خطوة خطوة.

بيمنع الغلطات اللي بتحصل لما تشتغل على كائن ناقص بيانات.

بيخلي الكود شكله أنضف وأكتر تنظيم.

#CSharp #DotNet #Constructor #ProgrammingTips
#OOP #CleanCode #CodeSmart #LearnToCode
#SoftwareDevelopment #TechInsights

إعجاب تعليق إعادة نشر إرسال

إضافة تعليق...

🔍يعني إيهIndexer ؟

في#C ، الـ **Indexer**هو طريقة تقدر تخلي الكائن (Object) يتعامل كأنه **Array**أو **Collection**، بحيث تقدر توصل للبيانات اللي جواه باستخدام الأقواس .[]

بدل ما تستدعي ميثود عشان ترجع عنصر، بتكتب حاجة زي:

// obj[0]تجيب أول عنصر

وده بيكون مريح أكتر ويخلي الكود أنضف وأسهل للقراءة.

🛠بيستخدم إمتى؟

- لما الكائن بتاعك بيخزن **مجموعة عناصر** وعايز توصل لها بسهولة زي الـArray.
- لما تحب تدي إحساس إن الكائن "قابل للفهرسة.(Indexable) "
- لما تكون البيانات ليها **مفتاح (Key)** أو **رقم فهرس.(Index)**

💼أمثلة من البيزنس:(Business Use Cases)

1. **إدارة موظفين في شركة** 🏢
عندك كائن Employeesفيه كل الموظفين، وعايز توصل لموظف معين برقم الـ ID أو ترتيبه:

// var emp = employees[102];يرجع الموظف اللي ID بتاعه 102

2. **مخزن منتجات**📦 **(Inventory System)**
عندك Productsوعايز توصل لمنتج برقم أو كود المنتج:(SKU)

```
var product = products["A123"];
```

3. **تطبيق ترجمة**📚 **(Dictionary App)**
عندك قاموس للكلمات، والـ Indexer يرجع الترجمة مباشرة:

```
var meaning = dictionary["Hello"];
```

4. **نظام حجوزات**🏨 **(Booking System)**
تحجز غرفة بفهرس أو رقم الحجز:

```
var booking = bookings[555];
```