Part1

Problem1:

```
using System;


class Car
{
   public int Id { get; set; }
   public string Brand { get; set; }
   public double Price { get; set; }


   public Car()
   {
      Id = 0;
      Brand = "Unknown";
      Price = 0.0;
      Console.WriteLine("Default constructor called");
   }


   public Car(int id)
   {
      Id = id;
      Brand = "Unknown";
      Price = 0.0;
      Console.WriteLine("Constructor with Id called");
   }
```

```
    public Car(int id, string brand)
    {
        Id = id;
        Brand = brand;
        Price = 0.0;
        Console.WriteLine("Constructor with Id and Brand called");
    }

    public Car(int id, string brand, double price)
    {
        Id = id;
        Brand = brand;
        Price = price;
        Console.WriteLine("Constructor with Id, Brand, and Price called");
    }

    public void ShowInfo()
    {
        Console.WriteLine($"Id: {Id}, Brand: {Brand}, Price: {Price}");
    }
}

class Program
{
    static void Main()
    {
```

```
        Car car1 = new Car();
        car1.ShowInfo();

        Car car2 = new Car(101);
        car2.ShowInfo();

        Car car3 = new Car(102, "Toyota");
        car3.ShowInfo();

        Car car4 = new Car(103, "BMW", 50000);
        car4.ShowInfo();
    }
}
```

Question1:

في:C#

- لو ما كتبتش أيConstructor ، الكومبايلر بيعمل **Default Constructor**تلقائيًا (من غير ما تكتبه).
- لكن لو أنت عرّفت Constructor بنفسك (زي اللي بياخد بارامترات)، الكومبايلر بيفترض إنك **مش محتاج** الـ Default Constructor، فـ مش بيولّده تلقائيًا.
- الحل لو محتاج الاتنين ← لازم تكتب الـ Default Constructor بنفسك زي ما عملنا فوق.

Problem2:

```
using System;


class Calculator
{
   public int Sum(int a, int b)
   {
      return a + b;
   }


   public int Sum(int a, int b, int c)
   {
      return a + b + c;
   }


   public double Sum(double a, double b)
   {
      return a + b;
   }
}


class Program
{
   static void Main()
   {
      Calculator calc = new Calculator();
```

```
        Console.WriteLine(calc.Sum(5, 10));
        Console.WriteLine(calc.Sum(1, 2, 3));
        Console.WriteLine(calc.Sum(2.5, 3.7));
    }
}
```

Question2:

دل ما تكتب دوال بأسماء مختلفة زي:

- SumTwoInt
- SumThreeInt
- SumDouble

ممكن تستخدم **نفس الاسم (Sum)** مع تغيير **عدد أو نوع البارامترات**.
ده بيخلي الكود:

- **أوضح** :لأن اسم الدالة واحد وواضح الغرض منها.
- **قابل لإعادة الاستخدام** :تقدر تستدعي نفس الميثود بطرق مختلفة بدل ما تحفظ أسماء كتيرة.

Problm3:

```
using System;

class Parent
{
    public int X { get; set; }
    public int Y { get; set; }

    public Parent(int x, int y)
```

```
    {
        X = x;
        Y = y;
        Console.WriteLine("Parent constructor called");
    }
}

class Child : Parent
{
    public int Z { get; set; }

    public Child(int x, int y, int z) : base(x, y)
    {
        Z = z;
        Console.WriteLine("Child constructor called");
    }

    public void ShowInfo()
    {
        Console.WriteLine($"X: {X}, Y: {Y}, Z: {Z}");
    }
}

class Program
{
    static void Main()
```

```
    {
        Child obj = new Child(10, 20, 30);
        obj.ShowInfo();
    }
}
```

Question3:

ما الهدف من Constructor Chaining في الوراثة؟

الهدف إنك تضمن إن الـ **خصائص الأساسية (Parent)** يتم تهيئتها بشكل صحيح قبل ما تضيف وتجهز خصائص الكلاس الفرعي.(Child)

- ده بيخلي الكود **منظم**، وبيمنع التكرار.
- وبيضمن إن الكائن الجديد متكامل من أول الجد للأب للابن.

Problem4:

```
using System;

class Parent
{
    public int X { get; set; }
    public int Y { get; set; }

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

```
    public virtual int Product()
    {
        return X * Y;
    }
}

class Child : Parent
{
    public int Z { get; set; }

    public Child(int x, int y, int z) : base(x, y)
    {
        Z = z;
    }

    public new int Product()
    {
        return X * Y * Z;
    }

    public override int Product()
    {
        return (X + Y) * Z;
    }
}
```

```
class Program
{
   static void Main()
   {
      Parent p = new Parent(2, 3);
      Console.WriteLine("Parent Product: " + p.Product());

      Child c = new Child(2, 3, 4);
      Console.WriteLine("Child Product (new): " + ((Parent)c).Product());
      Console.WriteLine("Child Product (override): " + c.Product());
   }
}
```

Question4:

new:

- بتعمل *إخفاء (hiding)* للميثود الأصلية في الـ.Parent
- يعني لو استدعيت الميثود من خلال **مرجع Parent** هيتنفذ الكود بتاع الـ Parent مش.Child

override:

- بتعمل *إعادة تعريف (true overriding)* للميثود الأصلية.
- يعني لو استدعيت الميثود من خلال **مرجع Parent أو Child** هيتنفذ الكود بتاع الـ.Child

Problem5:

```
using System;

class Parent
{
   public int X { get; set; }
   public int Y { get; set; }

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

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

class Child : Parent
{
   public int Z { get; set; }

   public Child(int x, int y, int z) : base(x, y)
   {
```

```
        Z = z;
    }

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

class Program
{
    static void Main()
    {
        Parent p = new Parent(5, 10);
        Console.WriteLine("Parent: " + p);

        Child c = new Child(5, 10, 15);
        Console.WriteLine("Child: " + c);

        Parent poly = new Child(1, 2, 3);
        Console.WriteLine("Polymorphism (Parent ref to Child obj): " + poly);
    }
}
```

Question5:

أن ToString() الافتراضية (من الكلاس object) بترجع اسم الكلاس بس، زي:

Namespace.ClassName

وده مش مفيد للمبرمج أو المستخدم.
فلما نعمل **override**:

- نقدر نعرض القيم الداخلية للأوبجكت بشكل أوضح.
- يسهل الـ**Debugging**.
- يخلي الطباعة واللوج أوضح ومقروءة أكتر.

Problem6:

```
using System;

abstract class Shape
{
   public virtual void Draw()
   {
      Console.WriteLine("Drawing Shape");
   }

   public abstract double CalculateArea();
}

class Rectangle : Shape
{
   public double Width { get; set; }
   public double Height { get; set; }

   public Rectangle(double width, double height)
```

```
    {
        Width = width;
        Height = height;
    }

    public override void Draw()
    {
        Console.WriteLine("Drawing Rectangle");
    }

    public override double CalculateArea()
    {
        return Width * Height;
    }
}

class Program
{
    static void Main()
    {
        Rectangle rect = new Rectangle(5, 10);

        rect.Draw();
        Console.WriteLine("Area: " + rect.CalculateArea());

        Shape s = new Rectangle(3, 7);
```

```
        s.Draw();
        Console.WriteLine("Area: " + s.CalculateArea());
    }
}
```

Question6:

virtual method:

- بيكون ليها **كود افتراضي** في الكلاس الأب.
- ممكن تسيبها زي ما هي أو تعمل **override** في الكلاس الابن.

abstract method:

- ما بيكونش ليها أي **كود/جسم** في الكلاس الأب.
- لازم **إجباريًا** أي كلاس يورثها يعملها .**override**
- الكلاس اللي فيه ميثود abstract لازم يكون .**abstract**

Part02

-:1

الفرق بين class و struct في:C#

1. **مكان التخزين:**
   - class بتتخزن في (**Heap** يعني مرجع.(reference
   - struct بتتخزن في (**Stack** يعني قيمة.(value
2. **الوراثة:**
   - class تقدر تعمل منها **وراثة** .(inheritance)
   - struct ما ينفعش تعمل منها وراثة) لكن ممكن تطبّق.(Interfaces
3. **الافتراضي:**

- لو ما أنشأتش كائن من classبيكون قيمته .null
- لو ما أنشأتش كائن من structبيكون ليه قيم افتراضية للـ) fields زي 0 للأرقام.(

4. **الاستخدام المناسب:**

- :classللحاجات الكبيرة والمعقدة اللي فيها وراثة وسلوكيات كتيرة.
- :structللحاجات الصغيرة والبسيطة اللي بتمثل بيانات بس) زي نقطة Point أو تاريخ.(Date

---

مثال:

```
class Car
{
public string Brand;
}

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

- Car (class)مناسب ككائن معقد ليه سلوك.
- Point (struct)مناسب كبيانات بسيطة وصغيرة.