C#

STRUKTURY, KLASY I KONSTRUKTORY

Struktura pamięci w komputerze

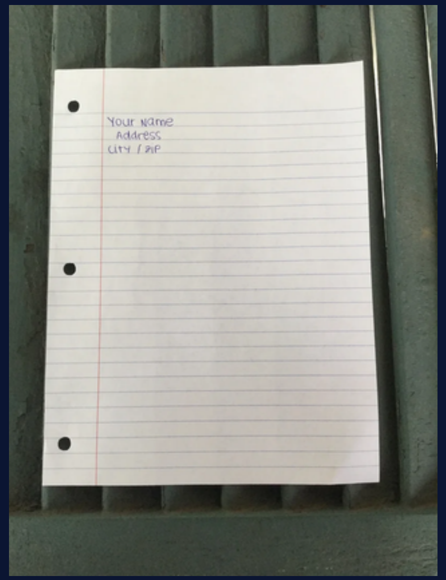
Stack (stos)

Heap (sterta)

Typy w C#

Value type

Reference type



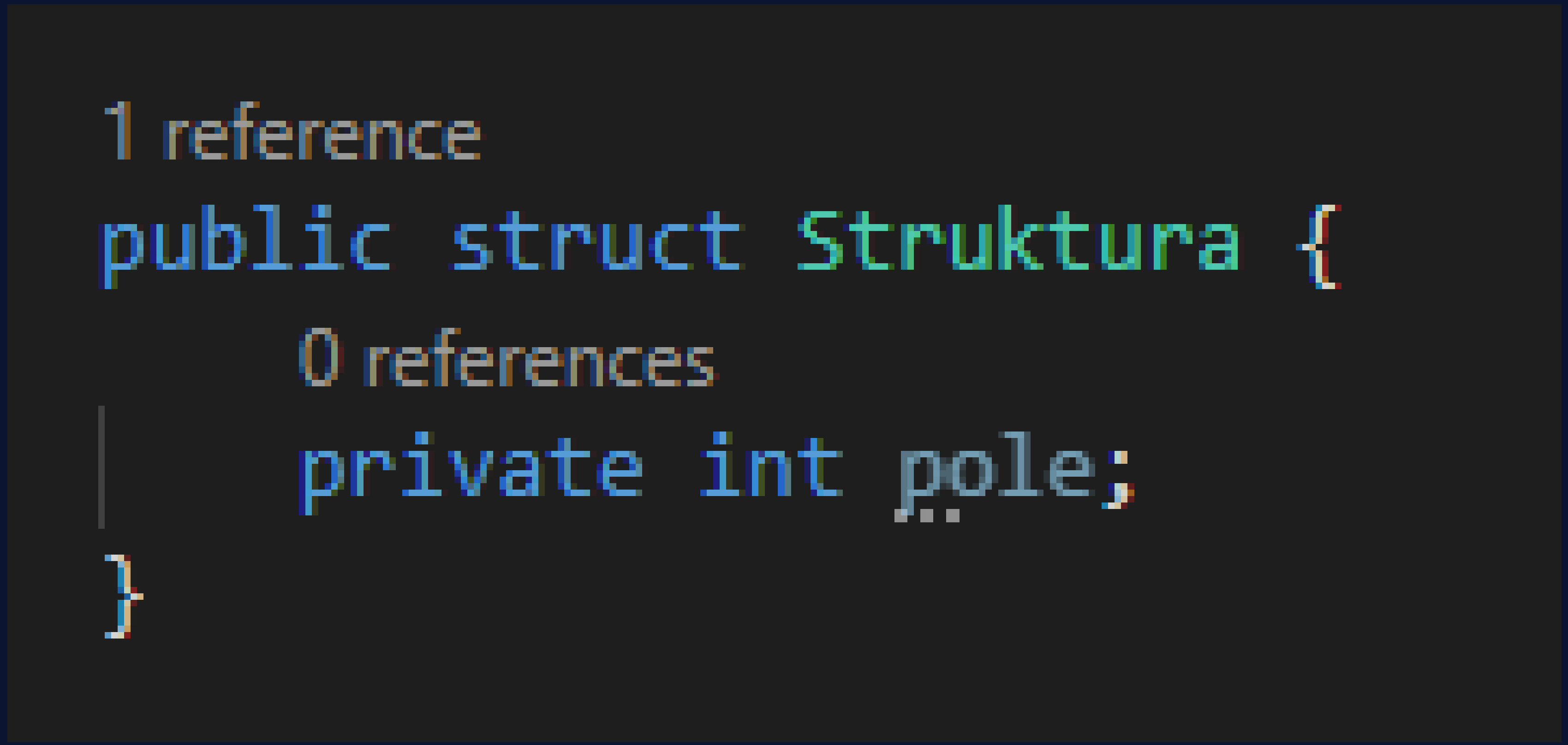
Gdzie mieszkają zmienne?

Stack (stos)

Heap (sterta)

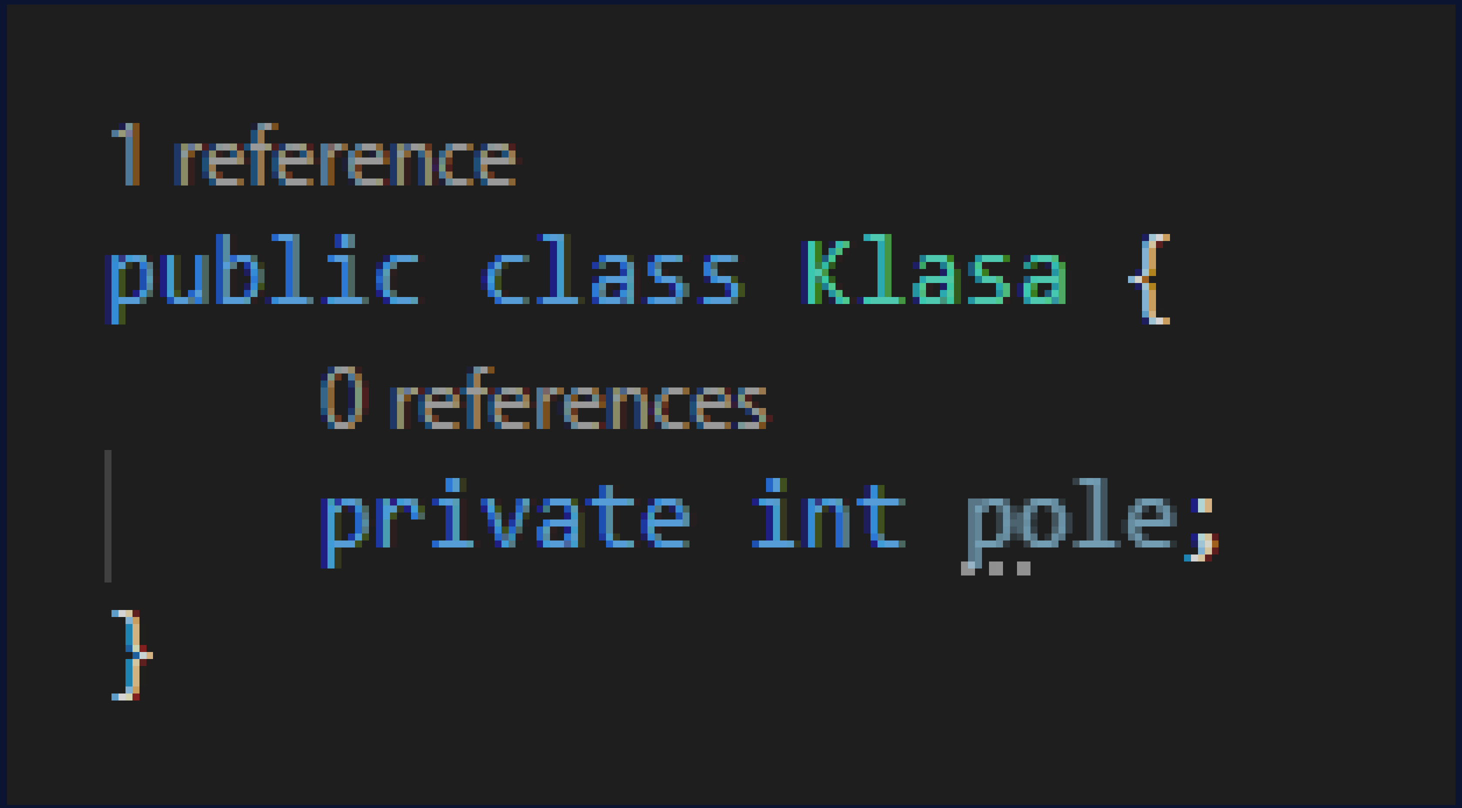
Jak się to ma do struktur i klas?

Struct to value type



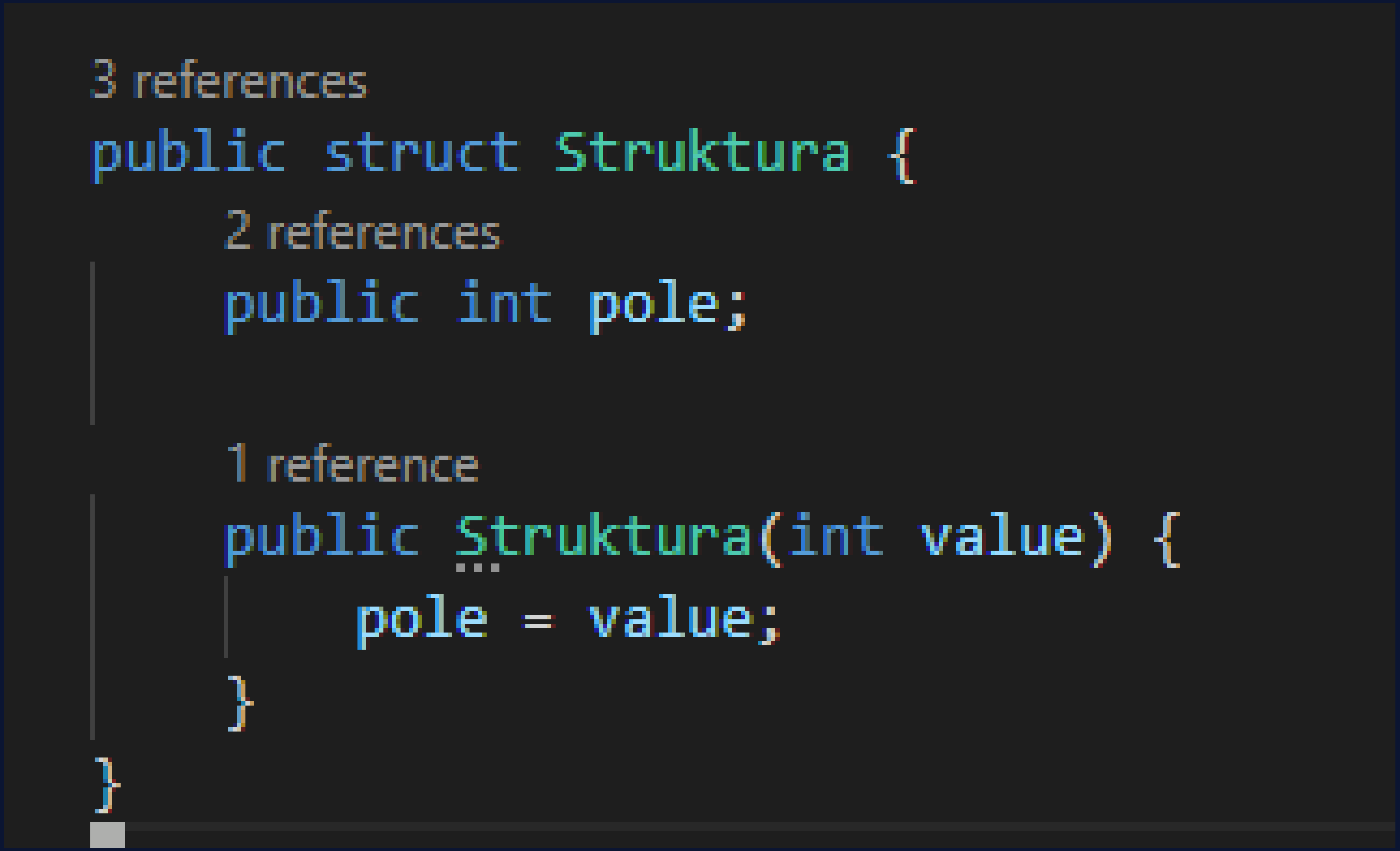
```
1 reference
public struct Struktura {
    0 references
    private int pole;
}
```

Klasa to reference type



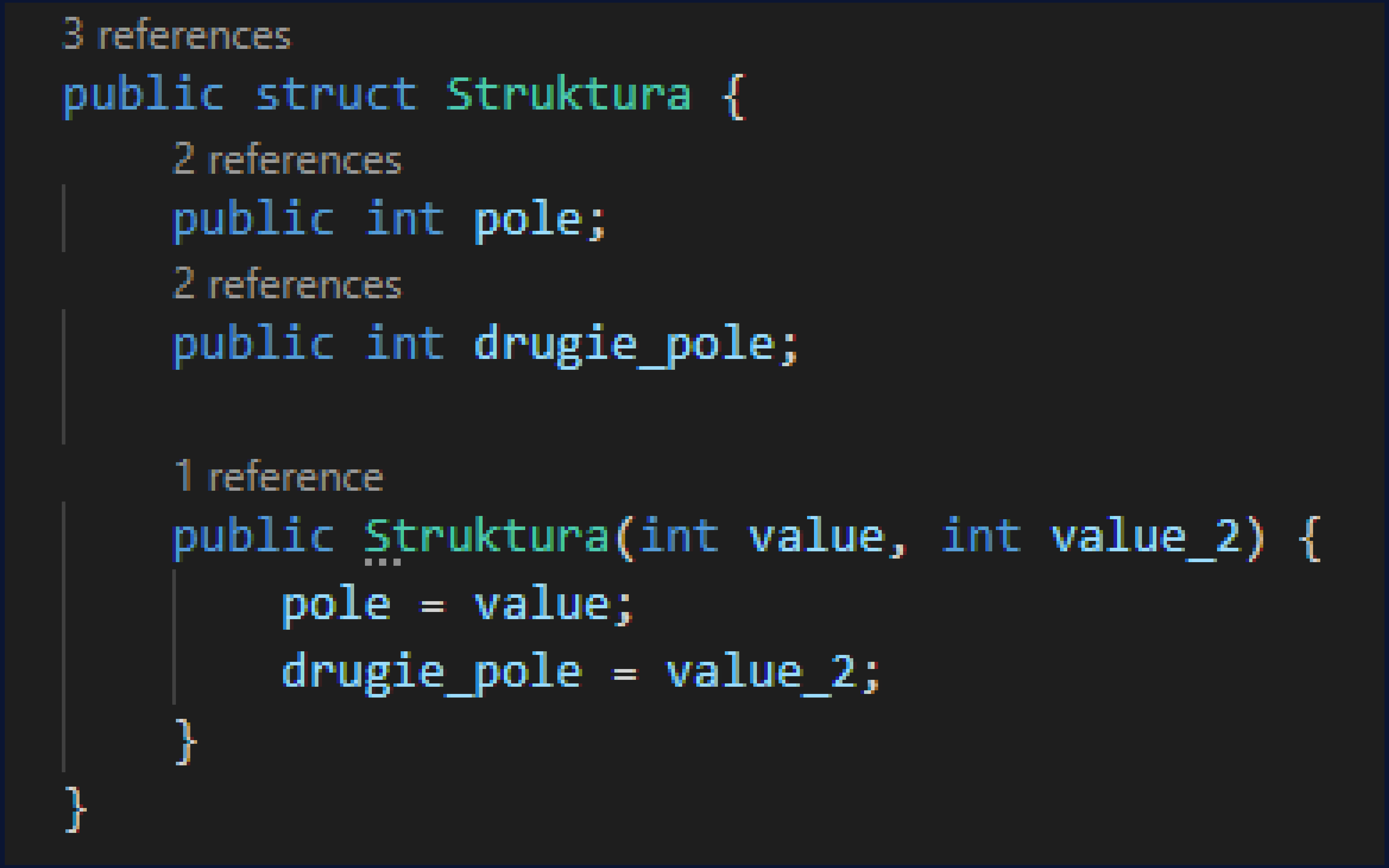
```
1 reference
public class Klasa {
    0 references
    private int pole;
}
```

Co to są struktury?

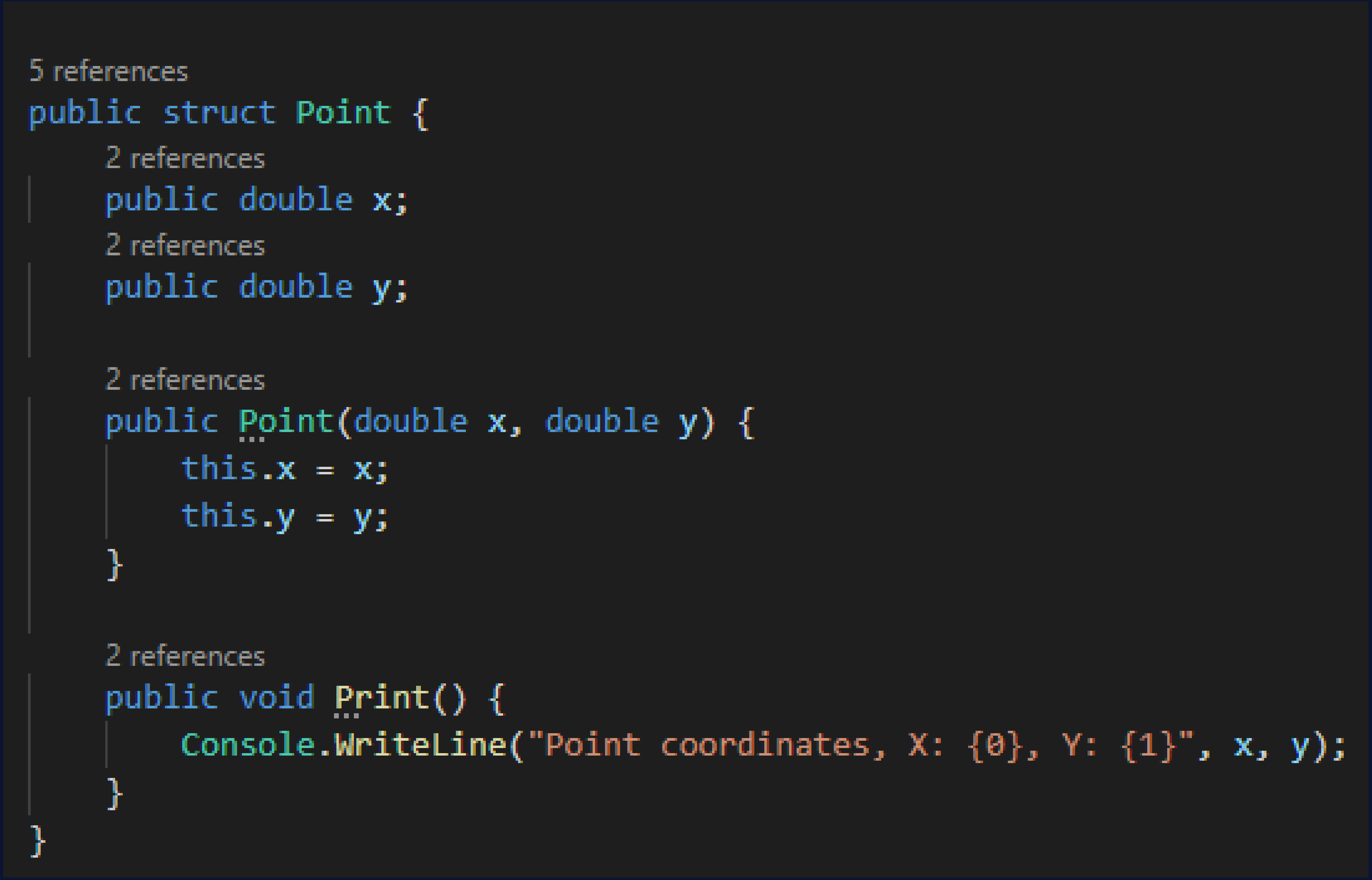


```
3 references
public struct Struktura {
    2 references
    public int pole;

    1 reference
    public Struktura(int value) {
        pole = value;
    }
}
```

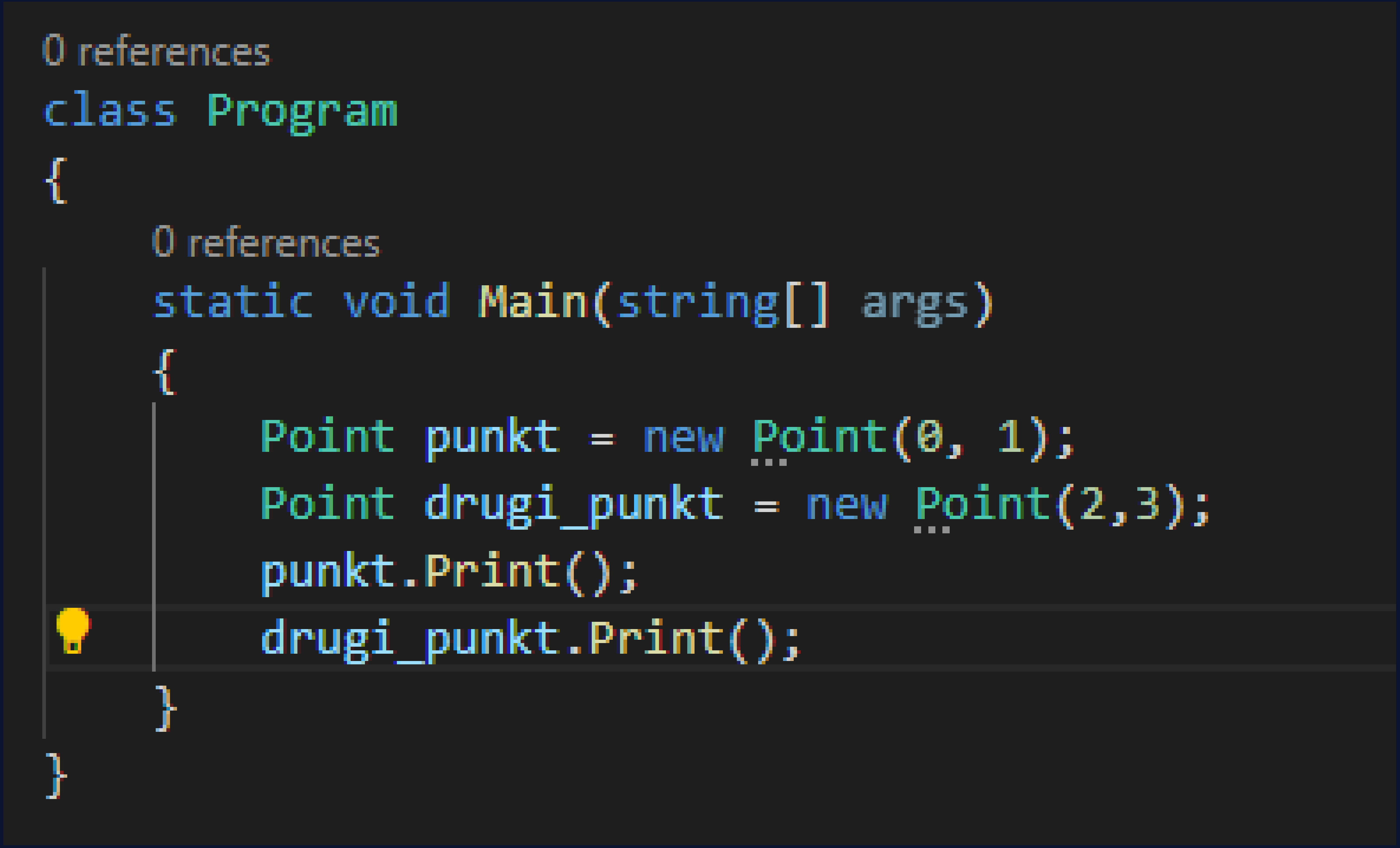
Więcej pól!



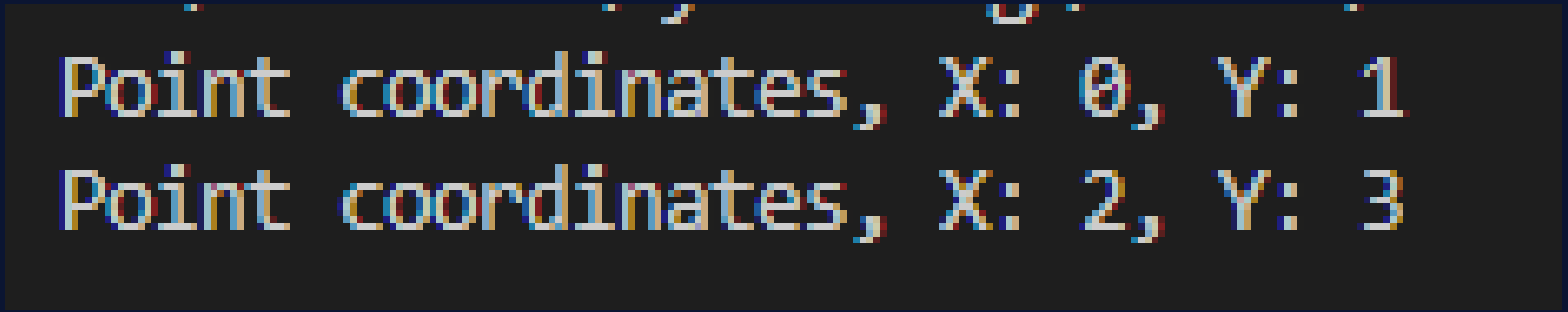
```
3 references
public struct Struktura {
    2 references
    public int pole;
    2 references
    public int drugie_pole;

    1 reference
    public Struktura(int value, int value_2) {
        pole = value;
        drugie_pole = value_2;
    }
}
```



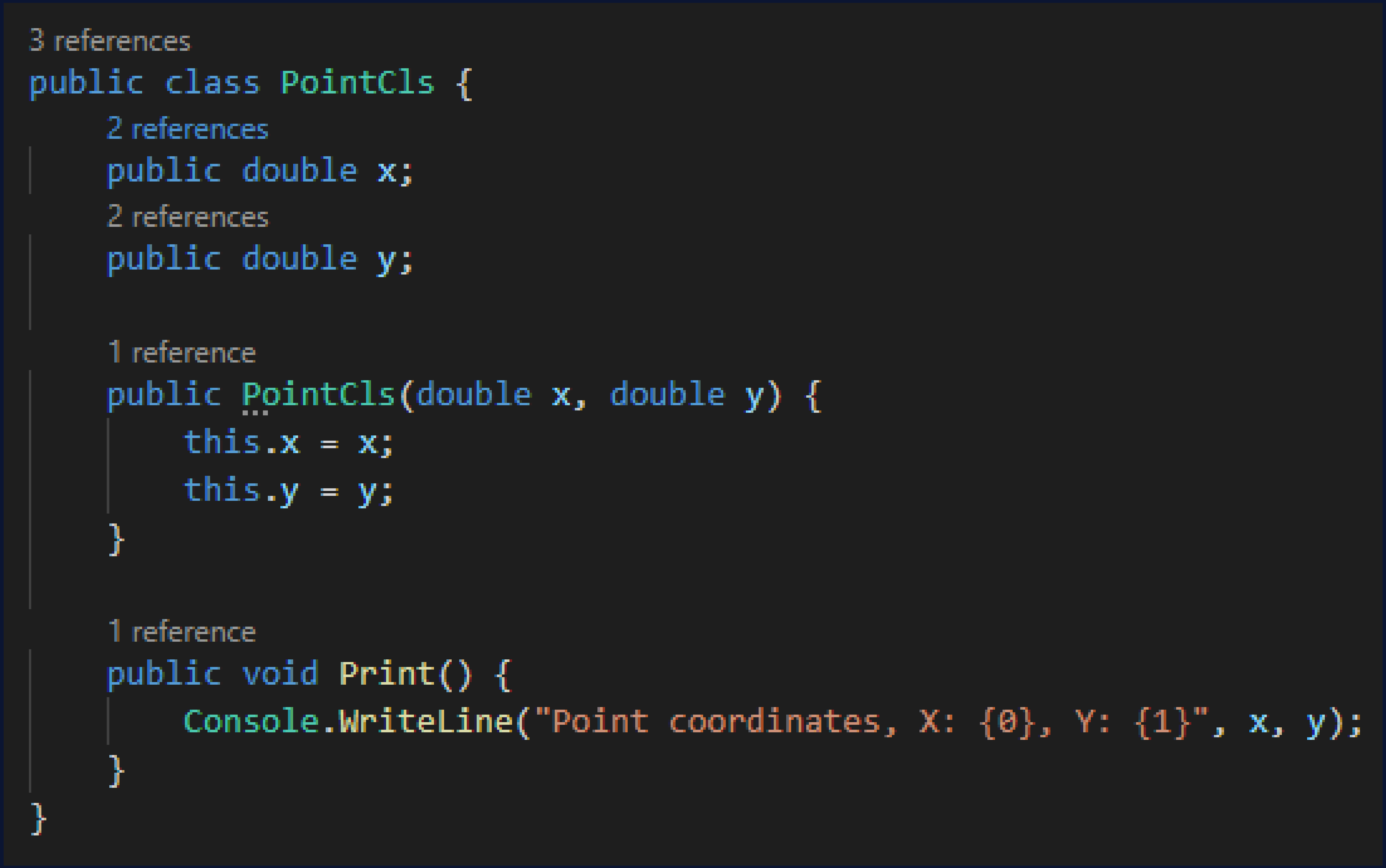
```
5 references
public struct Point {
    2 references
    public double x;
    2 references
    public double y;

    2 references
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    2 references
    public void Print() {
        Console.WriteLine("Point coordinates, X: {0}, Y: {1}", x, y);
    }
}
```



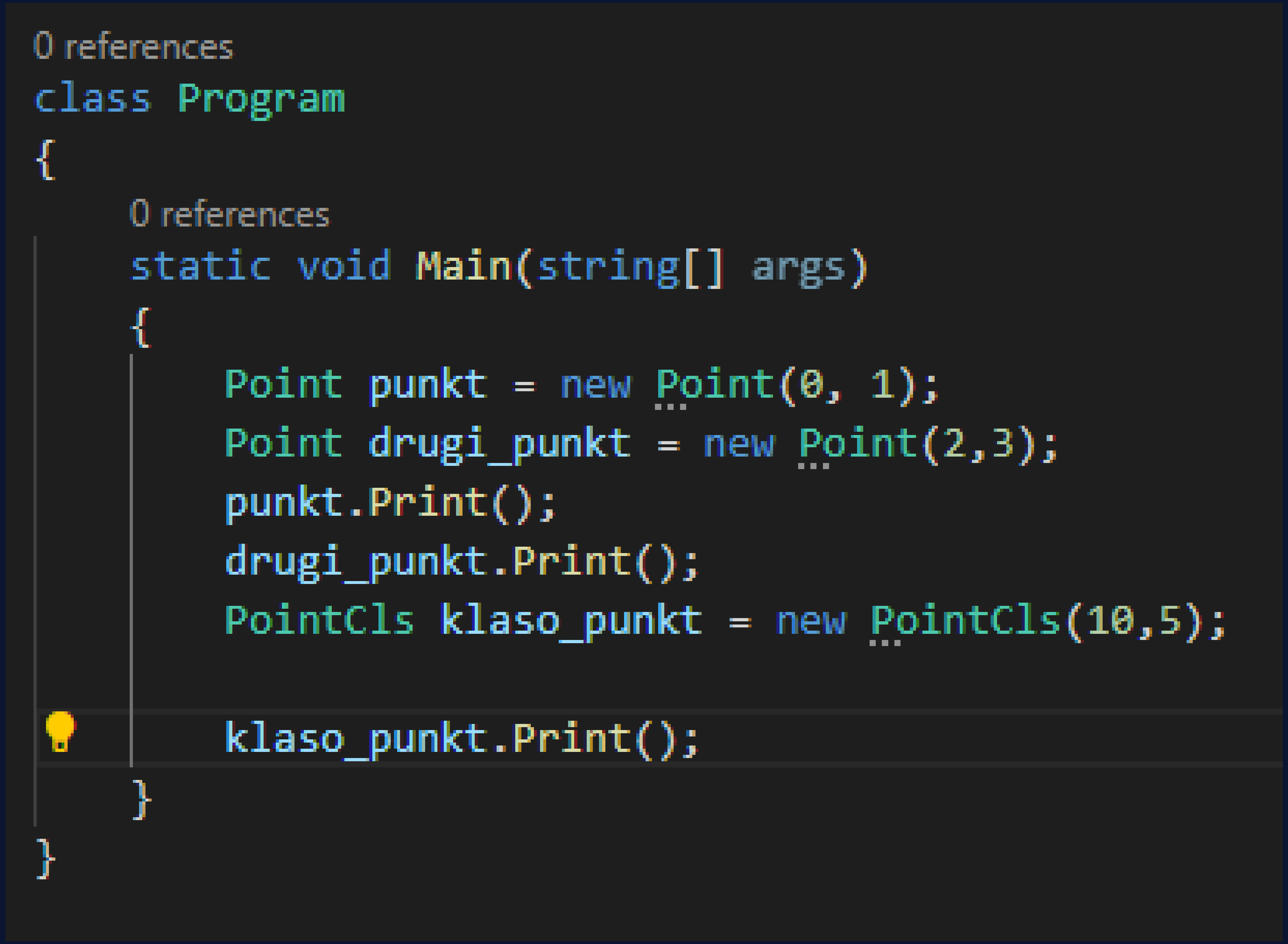
```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Point punkt = new Point(0, 1);
        Point drugi_punkt = new Point(2,3);
        punkt.Print();
        drugi_punkt.Print();
    }
}
```



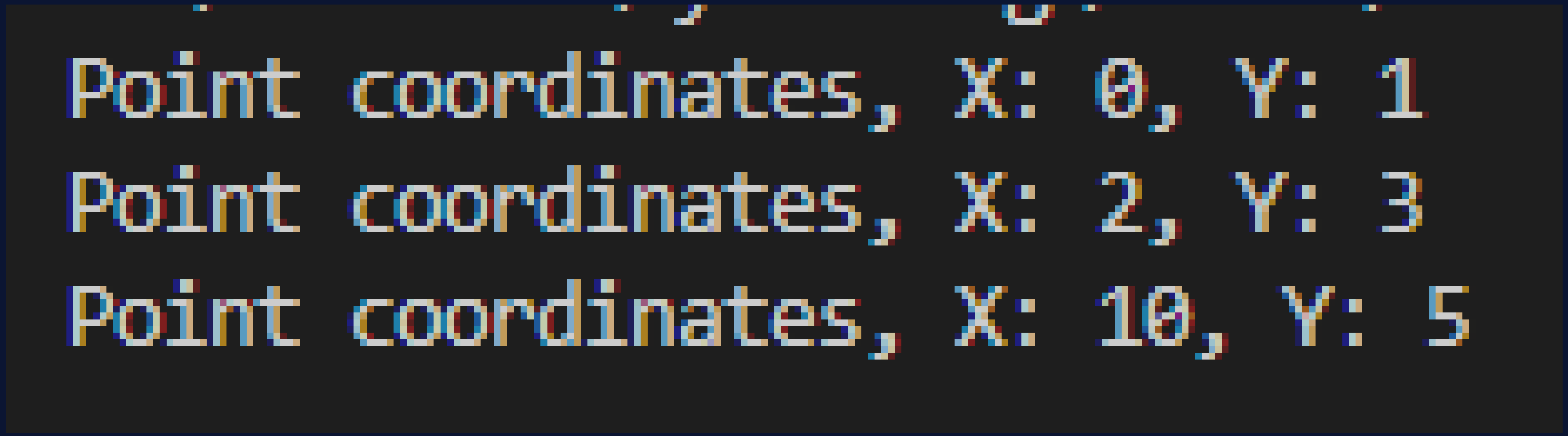
```
Point coordinates, X: 0, Y: 1
Point coordinates, X: 2, Y: 3
```



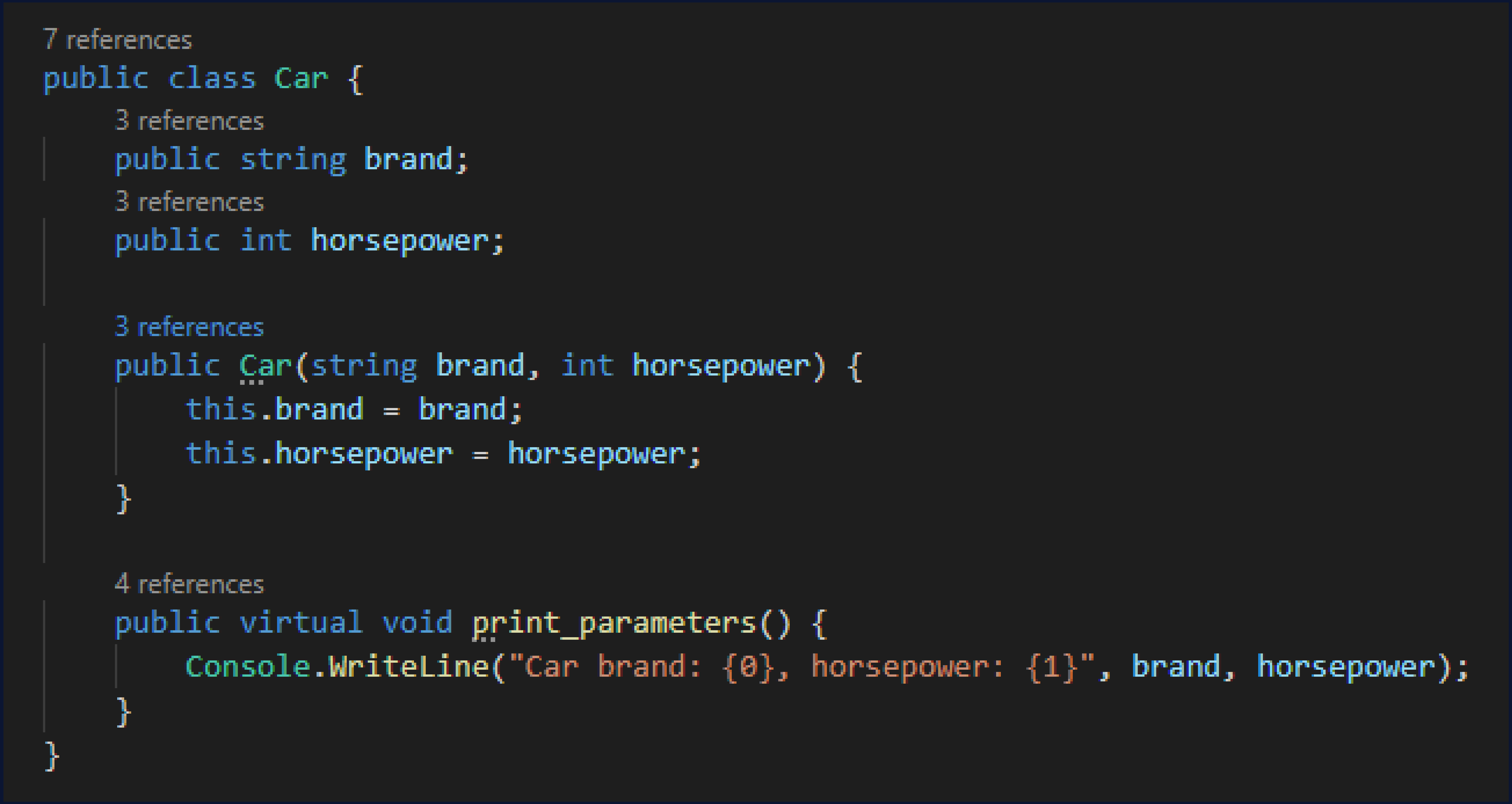
```
3 references
public class PointCls {
    2 references
    public double x;
    2 references
    public double y;

    1 reference
    public PointCls(double x, double y) {
        this.x = x;
        this.y = y;
    }

    1 reference
    public void Print() {
        Console.WriteLine("Point coordinates, X: {0}, Y: {1}", x, y);
    }
}
```



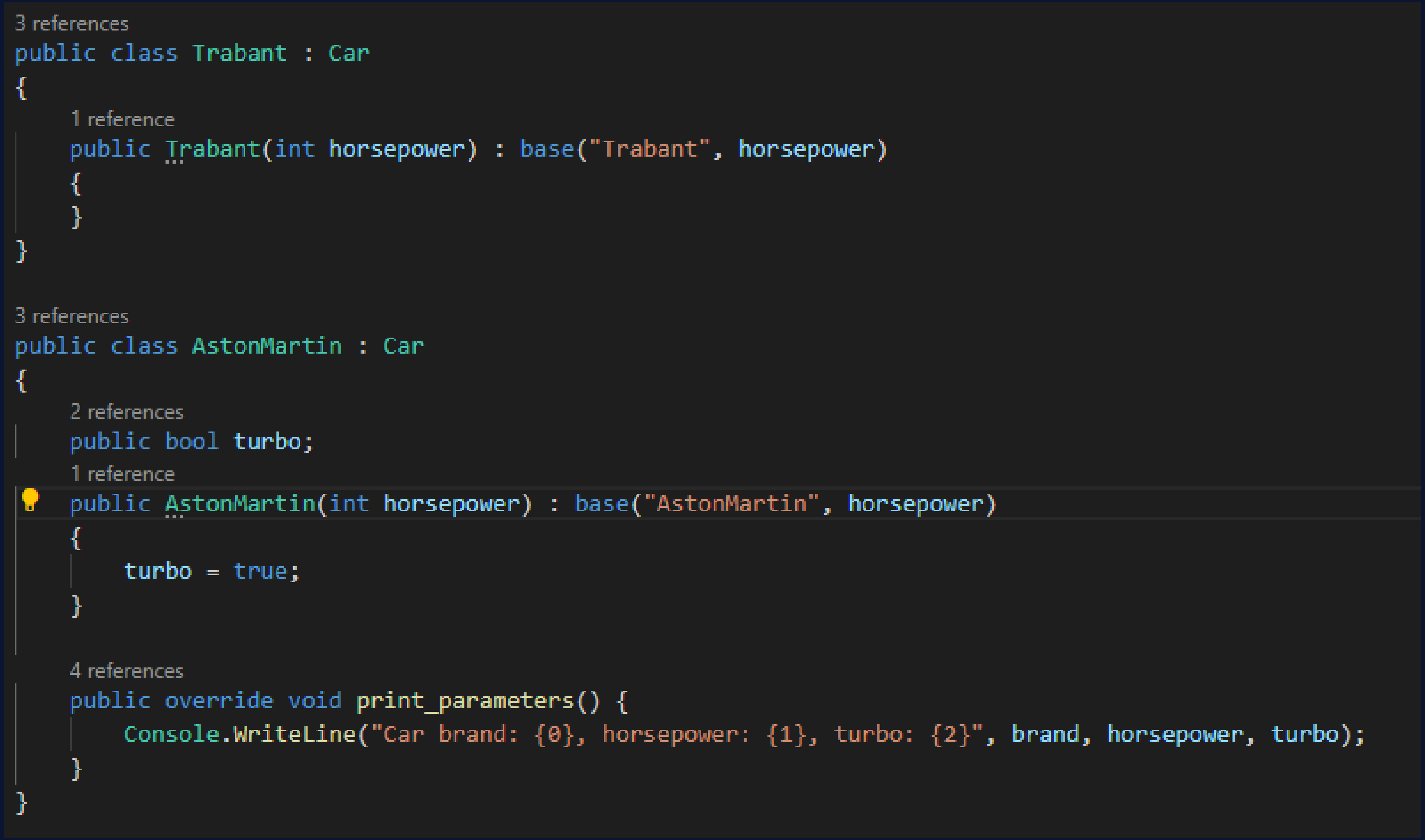
```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        Point punkt = new Point(0, 1);
        Point drugi_punkt = new Point(2,3);
        punkt.Print();
        drugi_punkt.Print();
        PointCls klaso_punkt = new PointCls(10,5);

        klaso_punkt.Print();
    }
}
```



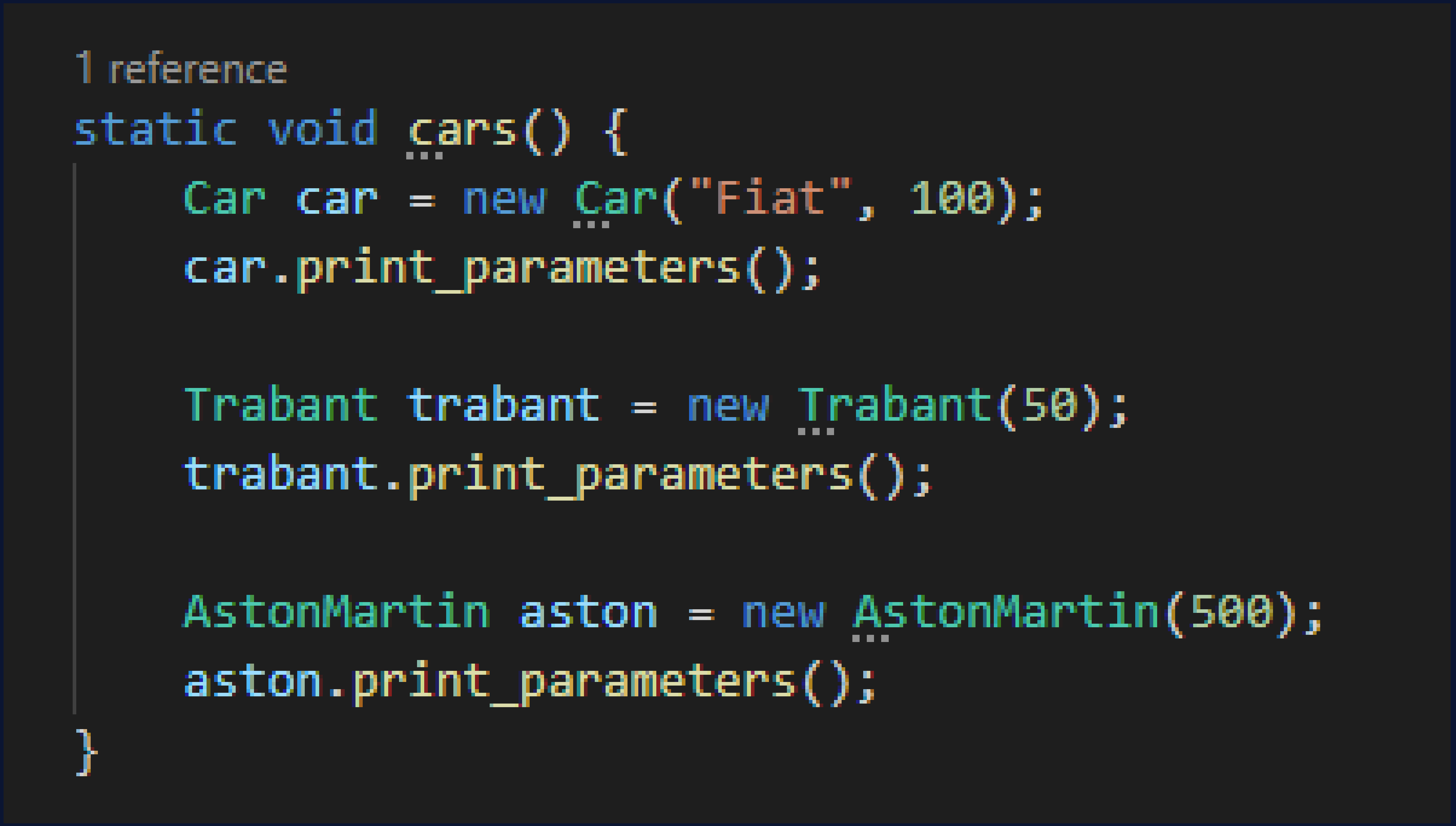
```
Point coordinates, X: 0, Y: 1
Point coordinates, X: 2, Y: 3
Point coordinates, X: 10, Y: 5
```



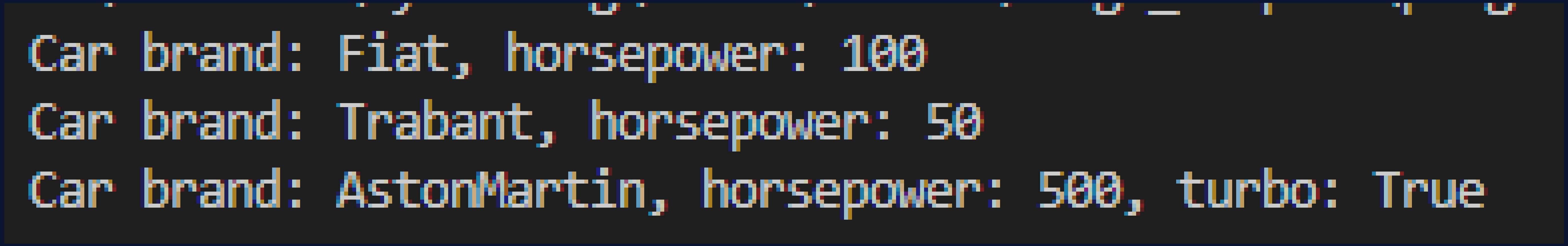
```
7 references
public class Car {
    3 references
    public string brand;
    3 references
    public int horsepower;

    3 references
    public Car(string brand, int horsepower) {
        this.brand = brand;
        this.horsepower = horsepower;
    }

    4 references
    public virtual void print_parameters() {
        Console.WriteLine("Car brand: {0}, horsepower: {1}", brand, horsepower);
    }
}
```



```
3 references
public class Trabant : Car
{
    1 reference
    public Trabant(int horsepower) : base("Trabant", horsepower)
    {
    }
}

3 references
public class AstonMartin : Car
{
    2 references
    public bool turbo;
    1 reference
    public AstonMartin(int horsepower) : base("AstonMartin", horsepower)
    {
        turbo = true;
    }

    4 references
    public override void print_parameters() {
        Console.WriteLine("Car brand: {0}, horsepower: {1}, turbo: {2}", brand, horsepower, turbo);
    }
}
```

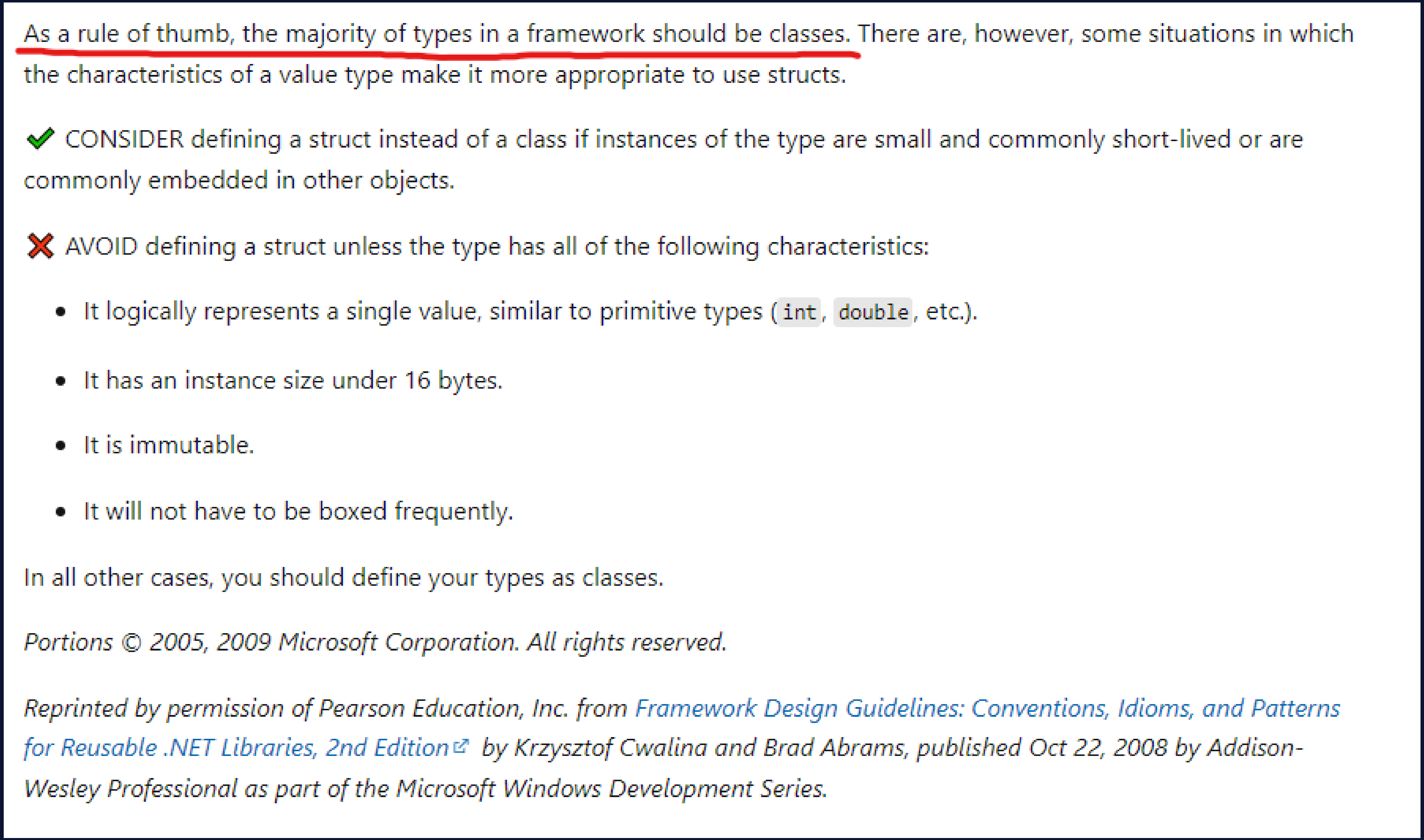


```
1 reference
static void cars() {
    Car car = new Car("Fiat", 100);
    car.print_parameters();

    Trabant trabant = new Trabant(50);
    trabant.print_parameters();

    AstonMartin aston = new AstonMartin(500);
    aston.print_parameters();
}
```



```
Car brand: Fiat, horsepower: 100
Car brand: Trabant, horsepower: 50
Car brand: AstonMartin, horsepower: 500, turbo: True
```

Kiedy czego używać?



As a rule of thumb, the majority of types in a framework should be classes. There are, however, some situations in which the characteristics of a value type make it more appropriate to use structs.

✓ CONSIDER defining a struct instead of a class if instances of the type are small and commonly short-lived or are commonly embedded in other objects.

✗ AVOID defining a struct unless the type has all of the following characteristics:

- It logically represents a single value, similar to primitive types (`int`, `double`, etc.).
- It has an instance size under 16 bytes.
- It is immutable.
- It will not have to be boxed frequently.

In all other cases, you should define your types as classes.

Portions © 2005, 2009 Microsoft Corporation. All rights reserved.

Reprinted by permission of Pearson Education, Inc. from Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries, 2nd Edition by Krzysztof Cwalina and Brad Abrams, published Oct 22, 2008 by Addison-Wesley Professional as part of the Microsoft Windows Development Series.

The end