1. The "Photocopy" Problem (Value Types)

Goal: Predict how simple integers behave when passed between variables.

The Code:

```C#
int a = 10;
int b = a;
b = 20;

Console.WriteLine($"a: {a}, b: {b}");
```

The Challenge:

- What will be the output?
- Explain why changing b does or does not affect a.

Answer:

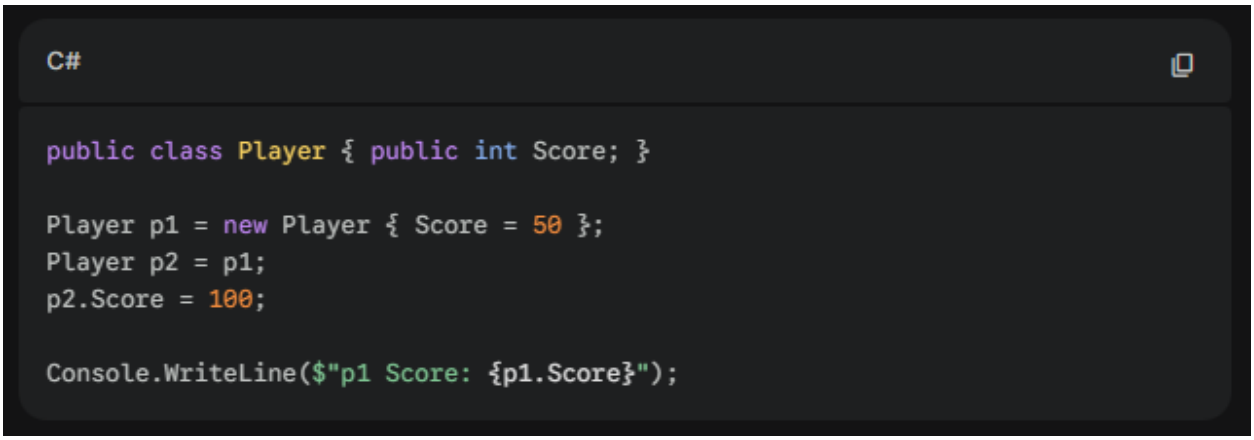
a=10 and b=20

This is due to integer is pass by value instead of reference.

2. The Shared Reference (Reference Types)

Goal: Observe how classes behave when multiple variables point to the same object.

The Code:



The Challenge:

- What is the final score of p1?
- What happens in memory when p2 = p1 is executed?

Answer:

The final score of p1 is 100.
p2 and p1 are sharing the same memory location. This means that both objects (p1 and p2) are sharing the same value at the memory location. Thus, changing value (score) on one of the objects will impact other.

3. Structs vs. Classes

Goal: Differentiate behavior between struct (Value Type) and class (Reference Type) using the exact same logic.

The Code:

```C#
public struct PointStruct { public int X; }
public class PointClass { public int X; }

static void UpdatePoint(PointStruct s, PointClass c) {
    s.X = 100;
    c.X = 100;
}

// Execution
PointStruct myStruct = new PointStruct { X = 10 };
PointClass myClass = new PointClass { X = 10 };

UpdatePoint(myStruct, myClass);

Console.WriteLine($"Struct: {myStruct.X}, Class: {myClass.X}");
```

The Challenge:

- Predict the output.
- Why does the method fail to update the struct but succeeds with the class?

Answer:

Struct: 10 , Class:100

Struct is passed by values to the function whereas class is passed by reference to the function.

4. The "Pure" String Mystery

Goal: Understand the "Reference Type Exception." Strings are reference types, but they are **immutable**.

The Code:

```C#
string name1 = "Alice";
string name2 = name1;
name2 = "Bob";

Console.WriteLine($"name1: {name1}, name2: {name2}");
```

The Challenge:

- If strings are reference types, why doesn't name1 change to "Bob"?
- **Bonus:** Research the term "String Interning" and how it relates to this.

Answer:

String got a property called immutability.

.NET tries to save memory through **String Interning.**

- **The Concept**: If you create ten variables that all equal "Alice", .NET doesn't want to waste heap space with ten identical copies.
- **The Intern Pool:** It maintains a table (the "Intern Pool") where it stores one unique copy of each literal string. All your variables will simply point to that same single memory address.
- **Safety:** This is only possible *because* strings are immutable. If you could change "Alice" to "Alex", and all ten variables pointed to it, you'd accidentally change the data for every part of your program!

5. Passing by Reference (ref keyword)

Goal: Use the ref keyword to make a Value Type behave like a Reference Type.

The Code:

```
C#

static void Increment(ref int val) {
    val++;
}

int myNumber = 5;
Increment(ref myNumber);
Console.WriteLine(myNumber);
```

The Challenge:

- What is the output?
- Modify the code: What happens if you remove the ref keyword from both the method signature and the call?

Answer:

myNumber become 6 because it is passed by reference to the function.

myNumber remain as 5 if both ref keyword been removed. Means the variable is passed by value to the function.