**C# Concepts – Value Type And Reference Type**

This seems to be a basic but very important part of C# programming.

Value type

Value types are generally (not always) stored on the stack and are passed by copying.

The way in which a variable assignment works differs between reference and value types.

If we have something like:

class Program

{

static void Main(string[] args)

{

A obj1 = new A(12);

int v1 = 12;

int v2 = 22;

v2 = v1;

Console.WriteLine(v2);

Console.ReadLine();

}

}

C#

Copy

**Implementation**

Here, both v1 and v2 will be on the stack and are different entities.

Reference Type

A value type is basically stored on the heap and passed by creating a reference.

using System;

class A {

public int value

{

get;

set;

}

public A(int passbyref)

{

this.value = passbyref;

}

}

class Program

{

static void Main(string[] args)

{

A v1 = new A(12);

A v2 = new A(22); //Breakpoint

v2 = v1;

Console.WriteLine(v1.value);

Console.WriteLine(v2.value);

Console.ReadLine();

}

}

C#

Copy

**Implementation**

v1 and v2 will be on the heap as two entities until a breakpoint.

And after the breakpoint, they both point to one entity.

A diagram of a diagram

Description automatically generated

**Figure 1: BreakPoin1**

A diagram of a diagram

Description automatically generated

**Figure 2: BreakPoint2**

So, a change in one will affect the other.

Conclusion

Once you pass a value type, you pass a copy to the other method.

But what if we want to change it? Use the “ref” keyword for that.

**Suggestion**

The difference between ref and out should be studied (I will try to write another article for that topic).

Use of ref in value types

The ref keyword passes the value by reference (details to be explained later).

class Program

{

static void Main(string[] args)

{

int v1 = 12;

methodtoshowref(ref v1);

Console.WriteLine(v1);

Console.ReadLine();

}

public static void methodtoshowref(ref int v2)

{

v2 = 100;

}

}

C#

Copy

Now, v1 becomes 100 because both share the same reference (one entity).

Passing Arguments

We have the following four possibilities:

1. Pass value type by value.
2. pass value type by reference
3. pass reference type by value.
4. pass reference type by reference.

Pass value type by value

struct A

{

public int val

{

get;

set;

}

}

class Program

{

static void Main(string[] args)

{

A v1 = new A();

v1.val = 10;

methodtoshowref(v1);

Console.WriteLine(v1.val);

Console.ReadLine();

}

public static void methodtoshowref(A obj)

{

obj = new A();;

}

}

CSS

Copy

**Output**

10 (because one more copy is created and thus the original is not affected).

Pass value type by reference

using System;

struct A

{

public int val

{

get;

set;

}

}

class Program

{

static void Main(string[] args)

{

A v1 = new A();

v1.val = 10;

methodtoshowref(ref v1);

Console.WriteLine(v1.val);

Console.ReadLine();

}

public static void methodtoshowref(ref A obj)

{

obj = new A();;

}

}

C#

Copy

**Output**

0 (since now one copy is shared by both methods).

Pass reference type by value (by default)

using System;

class A

{

public int val

{

get;

set;

}

}

class Program

{

static void Main(string[] args)

{

A v1 = new A();

v1.val = 10;

methodtoshowref(v1);

Console.WriteLine(v1.val);

Console.ReadLine();

}

public static void methodtoshowref(A obj)

{

obj = null;

}

}

C#

Copy

**Output**

10 (this happens because we are passing it by value).

Now here if we do, obj.val=100 then it will print 100 (this is because passing a variable to a function by value is equivalent to instantiating a new variable and assigning it to the first).

More to be discussed in the section on shallow copy vs deep copy (in a later article).

Pass reference type by reference

using System;

class A

{

public int val

{

get;

set;

}

}

class Program

{

static void Main(string[] args)

{

A v1 = new A();

v1.val = 10;

methodtoshowref(ref v1);

Console.WriteLine(v1.val);

Console.ReadLine();

}

public static void methodtoshowref(ref A obj)

{

obj = null;

}

}

C#

Copy

**Output**

The error we have now is obj=null so it will give a nullobject error.

**Disclaimer -** I got inspiration from JonSkeet and various other talented guys.