GENERICS IN C#

contents

Introduction to Generics

- Generics introduced in C# 2.0. Generics allow you to define a class with placeholders for the type of its fields, methods, parameters, etc. Generics replace these placeholders with some specific type at compile time.
- A generic class can be defined using angular brackets <>. For example, the following is a simple generic class with a generic member variable, generic method and property.

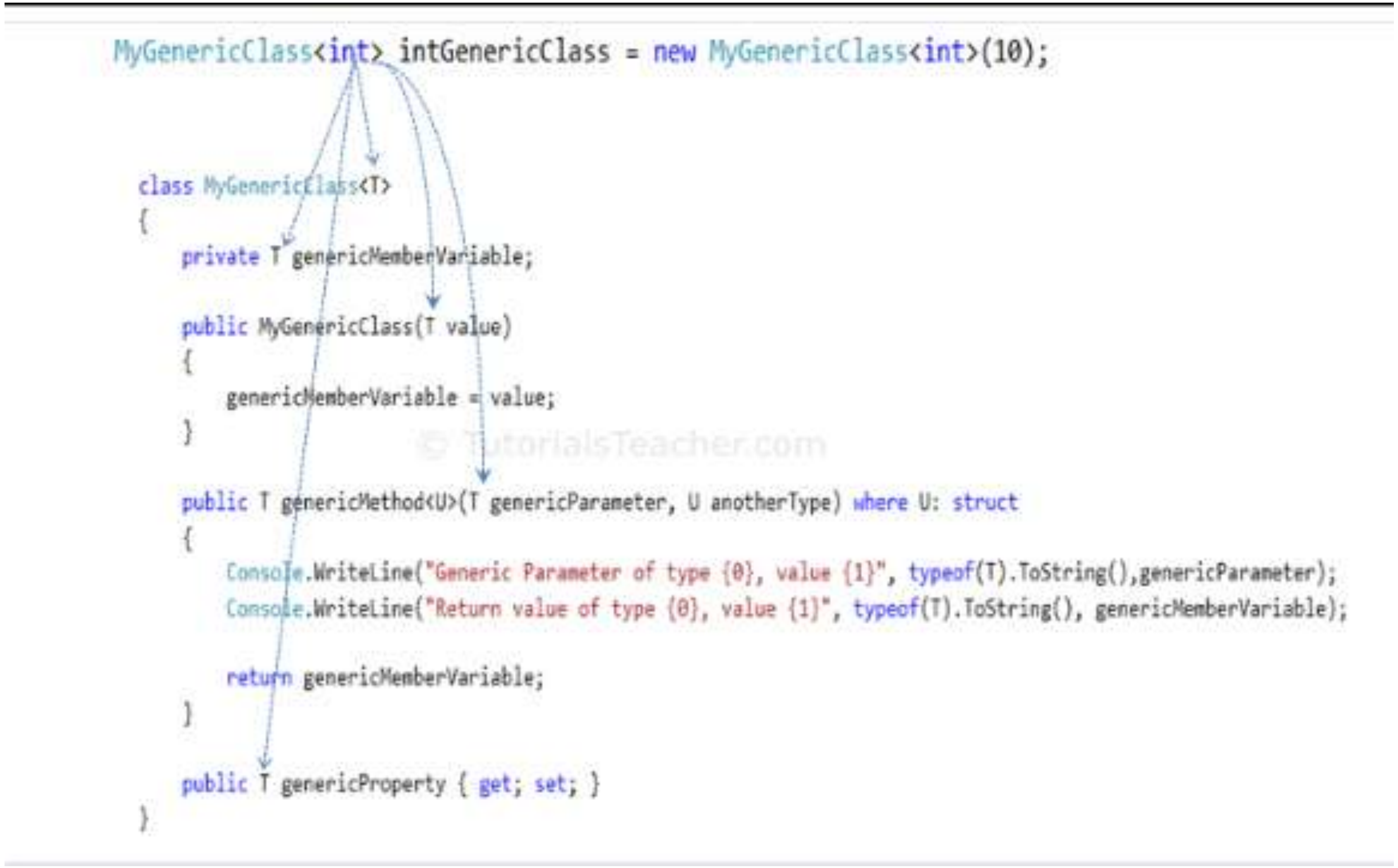
Benefits

- Increases the reusability of the code.
- Generic are type safe. You get compile time errors if you try to use a different type of data than the one specified in the definition.
- Generic has a performance advantage because it removes the possibilities of boxing and unboxing.

```
LinkedList<string> llist = new LinkedList<string>();
```

Generic Class

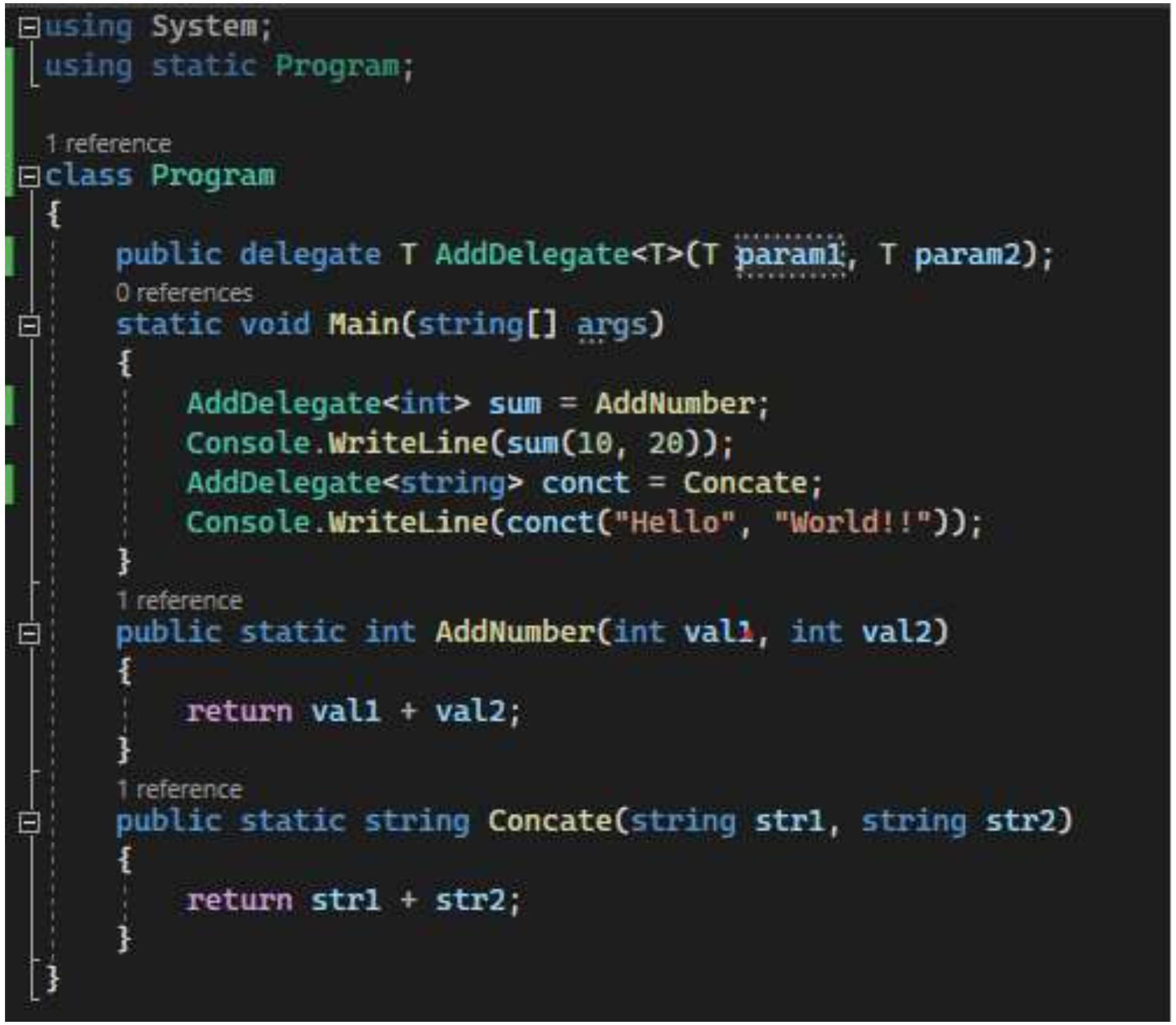
```
class MyGenericClass<T>
{
            private T genericMemberVariable;
            public MyGenericClass(T value)
            {
                        genericMemberVariable = value;
            }
            public T genericMethod(T genericParameter)
            {
                        Console.WriteLine("Parameter type: {0}, value: {1}", typeof(T).ToString(),genericParameter);
                        Console.WriteLine("Return type: {0}, value: {1}", typeof(T).ToString(), genericMemberVariable);
                        Return genericMemberVariable;
            }
            public T genericProperty
            {
                        get; set;
            }
}
```



```
MyGenericClass<int> intGenericClass = new MyGenericClass<int>(10);


class MyGenericClass<T>
{
    private T genericMemberVariable;

    public MyGenericClass(T value)
    {
        genericMemberVariable = value;
    }

    public T genericMethod<U>(T genericParameter, U anotherType) where U: struct
    {
        Console.WriteLine("Generic Parameter of type {0}, value {1}", typeof(T).ToString(),genericParameter);
        Console.WriteLine("Return value of type {0}, value {1}", typeof(T).ToString(), genericMemberVariable);

        return genericMemberVariable;
    }

    public T genericProperty { get; set; }
}
```

Generic Delegates

- The delegate defines the signature of the method which it can invoke. A generic delegate can be defined the same way as delegate but with generic type.
- A generic delegate can point to methods with different parameter types. However, the number of parameters should be the same.



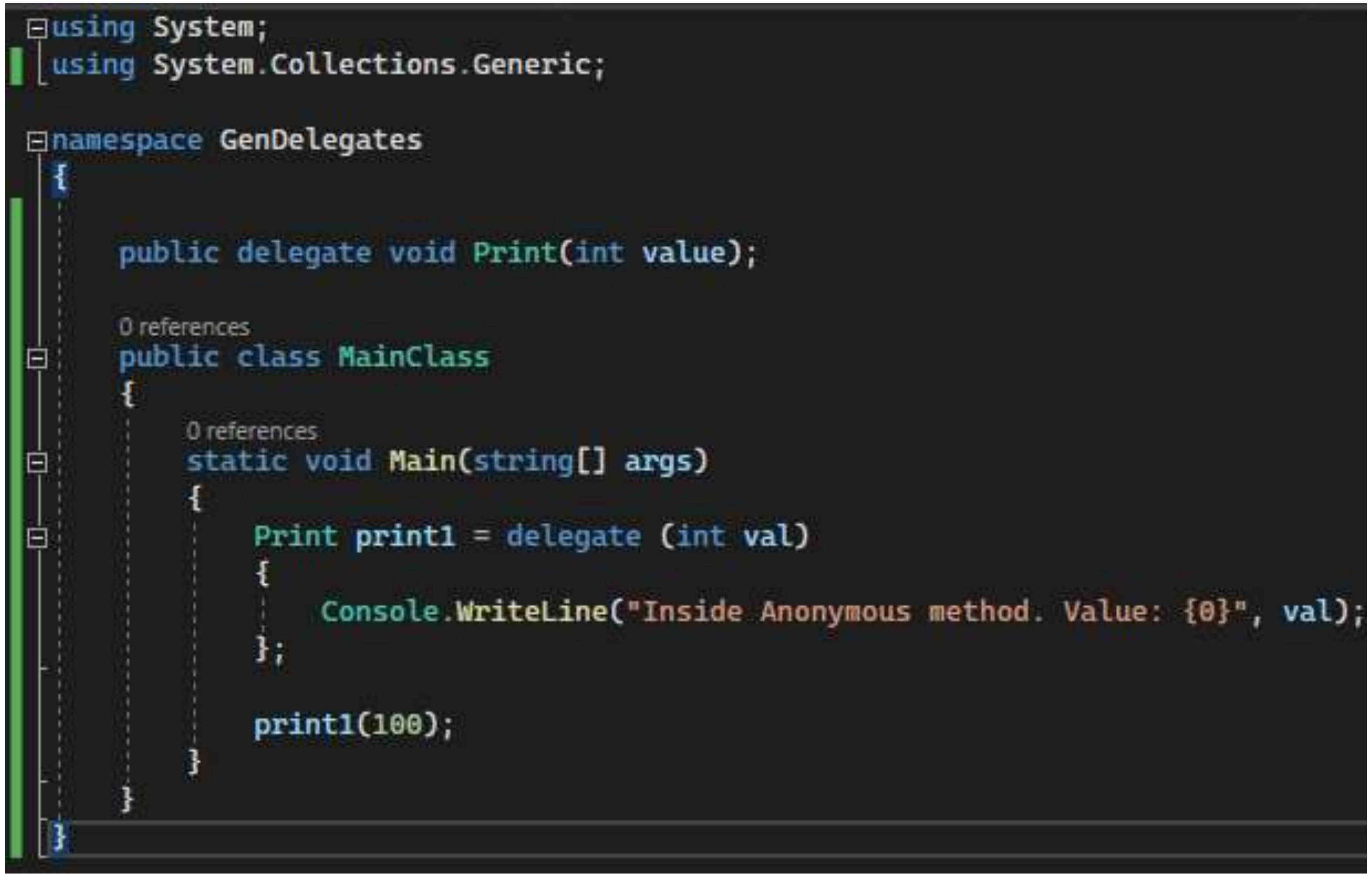
```
using System;
using static Program;

1 reference
class Program
{
    public delegate T AddDelegate<T>(T param1, T param2);
    0 references
    static void Main(string[] args)
    {
        AddDelegate<int> sum = AddNumber;
        Console.WriteLine(sum(10, 20));
        AddDelegate<string> conct = Concate;
        Console.WriteLine(conct("Hello", "World!!"));
    }
    1 reference
    public static int AddNumber(int val1, int val2)
    {
        return val1 + val2;
    }
    1 reference
    public static string Concate(string str1, string str2)
    {
        return str1 + str2;
    }
}
```

Points to Remember

1. Generics denotes with angular bracket <>.
2. Compiler applies specified type for generics at compile time.
3. Generics can be applied to interface, abstract class, method, static method, property, event, delegate and operator.
4. Generics performs faster by not doing boxing & unboxing.

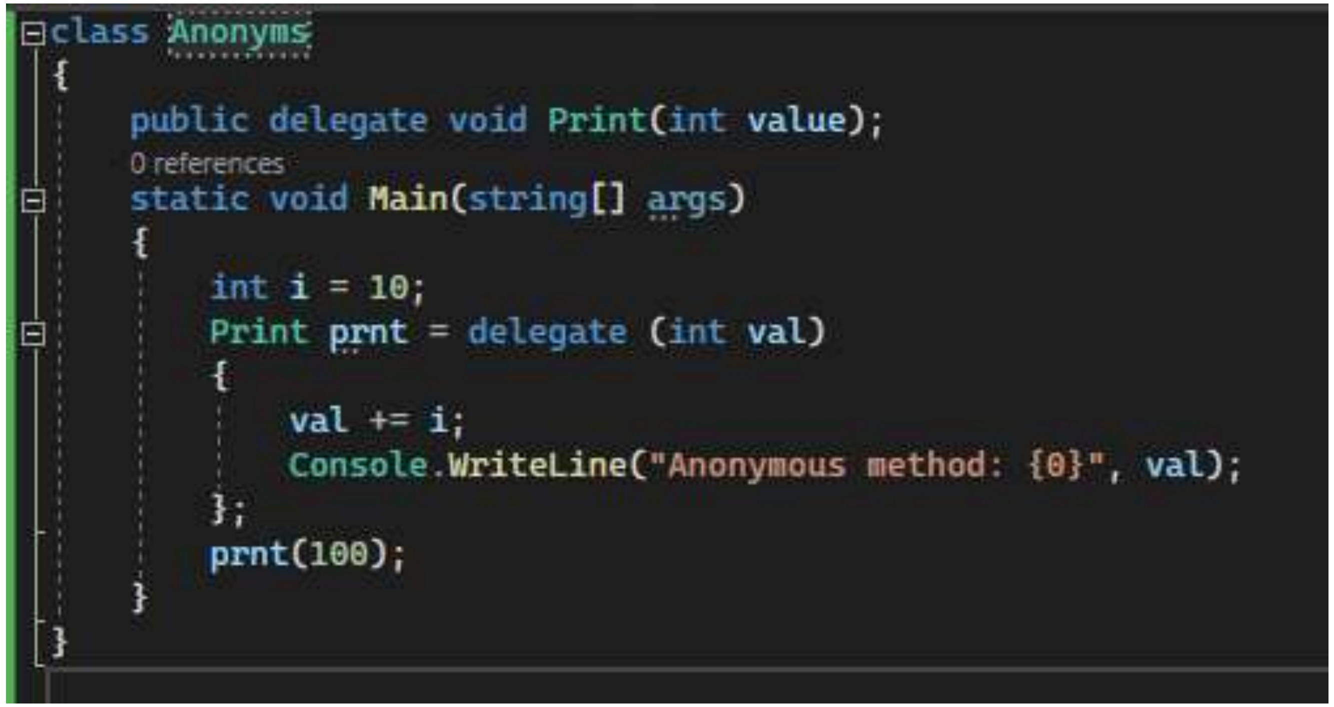
Anonymous methods
In C#

- An anonymous method is a method without a name.
- Anonymous methods in C# can be defined using the delegate keyword and can be assigned to a variable of delegate type.
- Example:



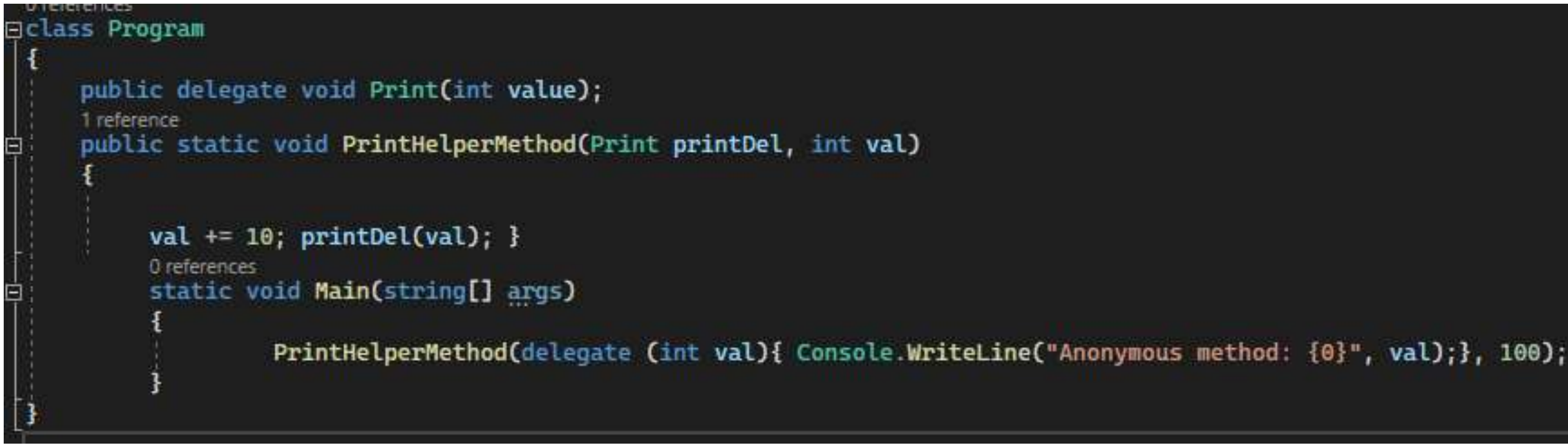
```
using System;
using System.Collections.Generic;

namespace GenDelegates
{

    public delegate void Print(int value);

    0 references
    public class MainClass
    {
        0 references
        static void Main(string[] args)
        {
            Print print1 = delegate (int val)
            {
                Console.WriteLine("Inside Anonymous method. Value: {0}", val);
            };

            print1(100);
        }
    }
}
```

- Anonymous methods can access variables defined in an outer function.



```
class Anonyms
{
    public delegate void Print(int value);
    0 references
    static void Main(string[] args)
    {
        int i = 10;
        Print prnt = delegate (int val)
        {
            val += i;
            Console.WriteLine("Anonymous method: {0}", val);
        };
        prnt(100);
    }
}
```

- Anonymous methods can also be passed to a method that accepts the delegate as a parameter.



```
0 references
class Program
{
    public delegate void Print(int value);
    1 reference
    public static void PrintHelperMethod(Print printDel, int val)
    {

         val += 10; printDel(val); }
         0 references
         static void Main(string[] args)
         {
                  PrintHelperMethod(delegate (int val){ Console.WriteLine("Anonymous method: {0}", val);}, 100);
         }
}
```

- Anonymous methods can be used as event handlers:

```
 saveButton.Click += delegate(Object o, EventArgs e)
{
   System.Windows.Forms.MessageBox.Show("Save
Successfully!");
};
```

 Typical usage of an anonymous method is to assign it to an event.

Limitations

- It cannot contain jump statement like goto, break or continue.
- It cannot access ref or out parameter of an outer method.
- It cannot have or access unsafe code.
- It cannot be used on the left side of the is operator.

Points to Remember

- Anonymous method can be defined using the delegate keyword
- Anonymous method must be assigned to a delegate.
- Anonymous method can access outer variables or functions.
- Anonymous method can be passed as a parameter.
- Anonymous method can be used as event handlers.