CS202 Lab Report 10: Object-Oriented Concepts in C#

Md Sibtain Raza (22110148) | Github Link

November 23, 2025

Contents

1 Objective

This lab extends the object-oriented programming concepts introduced in lectures through hands-on tasks using classes, inheritance, and event-driven programming in C#. It focuses on building modular console applications that simulate real-world behaviours, combining delegates, events, and inheritance hierarchy control.

2 Lab Activities

The lab was divided into two main programming tasks followed by a series of output reasoning challenges.

2.1 Constructors and Data Control

2.1.1 Task

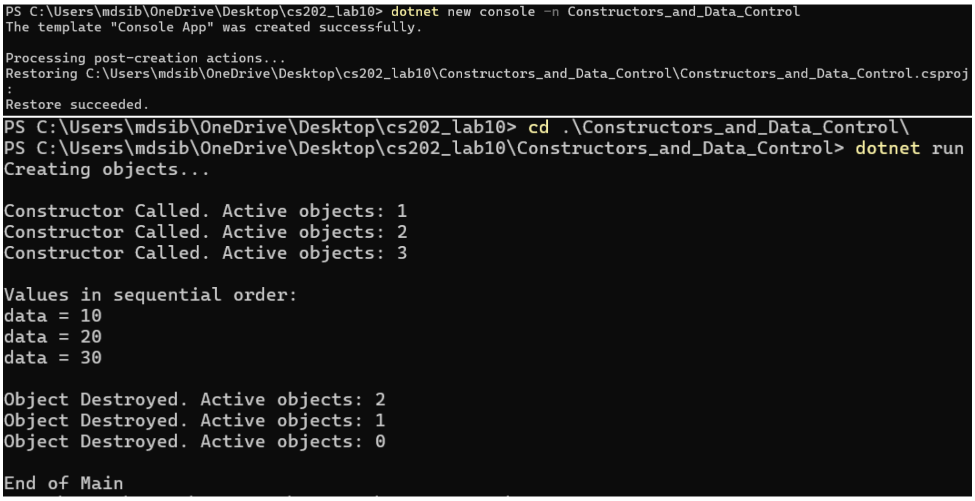
This task involved creating a class with a private field, a constructor, a destructor, and methods to set and show data. A static integer was used to track the number of active objects. The `Main` method demonstrated the object lifecycle by creating, using, and implicitly destroying three instances of the class.

2.1.2 Code

Link to code

2.1.3 Output

The following console output was captured from the execution of this task.



```
PS C:\Users\mdsib\OneDrive\Desktop\cs202_lab10> dotnet new console -n Constructors_and_Data_Control
The template "Console App" was created successfully.

Processing post-creation actions...
Restoring C:\Users\mdsib\OneDrive\Desktop\cs202_lab10\Constructors_and_Data_Control\Constructors_and_Data_Control.csproj
:
Restore succeeded.
```

```
PS C:\Users\mdsib\OneDrive\Desktop\cs202_lab10> cd .\Constructors_and_Data_Control\
PS C:\Users\mdsib\OneDrive\Desktop\cs202_lab10\Constructors_and_Data_Control> dotnet run
Creating objects...

Constructor Called. Active objects: 1
Constructor Called. Active objects: 2
Constructor Called. Active objects: 3

Values in sequential order:
data = 10
data = 20
data = 30

Object Destroyed. Active objects: 2
Object Destroyed. Active objects: 1
Object Destroyed. Active objects: 0

End of Main
```

Figure 1: Output for Constructors and Data Control

2.2 Inheritance and Method Overriding

2.2.1 Task

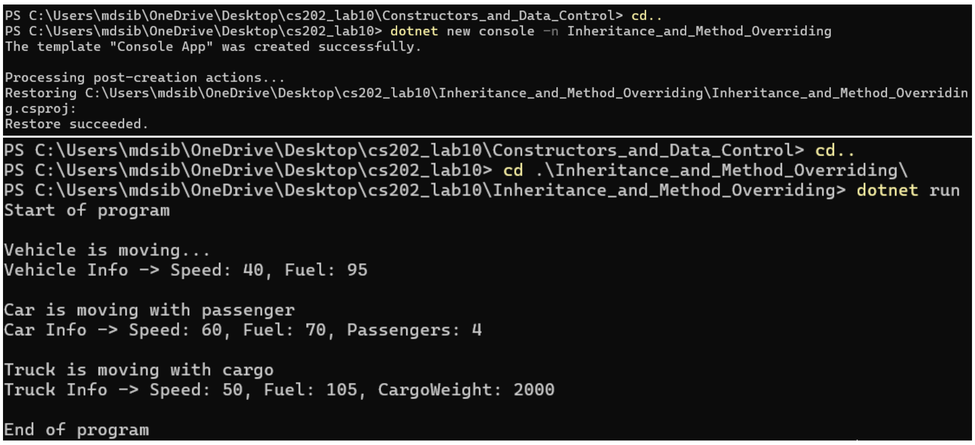
This task required creating a base class `Vehicle` and two derived classes, `Car` and `Truck`. The derived classes overrode the virtual methods `Drive()` and `ShowInfo()` from the base class to provide specialized implementations. The `Main` method demonstrated polymorphism by treating all objects as `Vehicle` types in an array and invoking their specific overridden methods.

2.2.2 Code

Link to code

2.2.3 Output

The following console output was captured, demonstrating the polymorphic behavior.



```
PS C:\Users\mdsib\OneDrive\Desktop\cs202_lab10\Constructors_and_Data_Control> cd..
PS C:\Users\mdsib\OneDrive\Desktop\cs202_lab10> dotnet new console -n Inheritance_and_Method_Overriding
The template "Console App" was created successfully.

Processing post-creation actions...
Restoring C:\Users\mdsib\OneDrive\Desktop\cs202_lab10\Inheritance_and_Method_Overriding\Inheritance_and_Method_Overridin
g.csproj:
Restore succeeded.
```

```
PS C:\Users\mdsib\OneDrive\Desktop\cs202_lab10\Constructors_and_Data_Control> cd..
PS C:\Users\mdsib\OneDrive\Desktop\cs202_lab10> cd .\Inheritance_and_Method_Overriding\
PS C:\Users\mdsib\OneDrive\Desktop\cs202_lab10\Inheritance_and_Method_Overriding> dotnet run
Start of program

Vehicle is moving...
Vehicle Info -> Speed: 40, Fuel: 95

Car is moving with passenger
Car Info -> Speed: 60, Fuel: 70, Passengers: 4

Truck is moving with cargo
Truck Info -> Speed: 50, Fuel: 105, CargoWeight: 2000

End of program
```

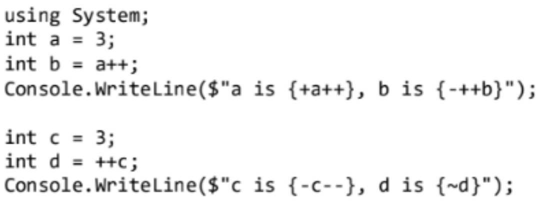
Figure 2: Output for Inheritance and Method Overriding

3 Output Reasoning

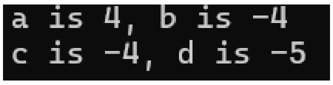
This section involved predicting and explaining the output of several C# code snippets.

3.1 Level 0

3.1.1 Snippet 1



```
using System;
int a = 3;
int b = a++;
Console.WriteLine($"a is {+a++}, b is {-++b}");

int c = 3;
int d = ++c;
Console.WriteLine($"c is {-c--}, d is {~d}");
```

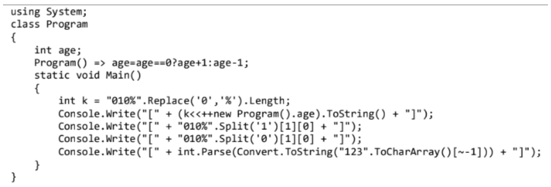


```
a is 4, b is -4
c is -4, d is -5
```

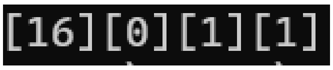
Figure 3: Output

Explanation: b gets 3 then a becomes 4 from $a++$, so $\{+a++\}$ prints 4 then a becomes 5, and $\{-++b\}$ pre-increments b to 4 then negates to -4; for c and d, d is 4 from $++c$ and printing $\{-c--\}$ uses 4 before c decrements to 3 while $\{\sim d\}$ is -5 since $x = -x - 1$ for two's complement integers.

3.1.2 Snippet 2



```
using System;
class Program
{
    int age;
    Program() => age=age==0?age+1:age-1;
    static void Main()
    {
        int k = "010%".Replace('0','%').Length;
        Console.Write("[" + (k<<++new Program().age).ToString() + "]");
        Console.Write("[" + "010%".Split('1')[1][0] + "]");
        Console.Write("[" + "010%".Split('0')[1][0] + "]");
        Console.Write("[" + int.Parse(Convert.ToString("123".ToCharArray()[~-1])) + "]");
    }
}
```



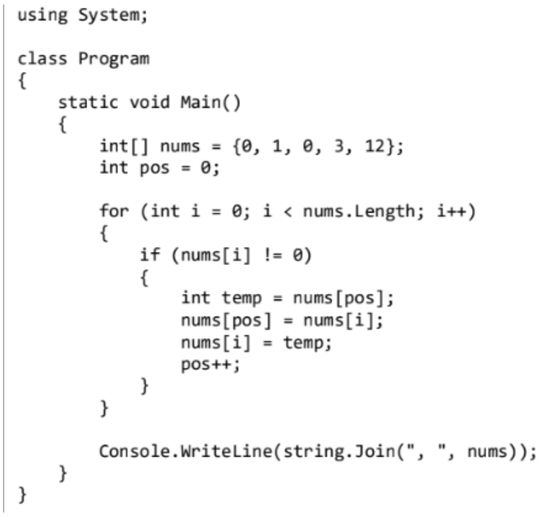
```
[16][0][1][1]
```

Figure 4: Output

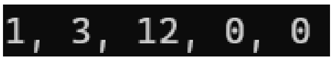
Explanation: Why: `"010%".Replace(’0’,’%’)` gives `"%1%%"` of length 4 so $k = 4$; a new `Program()` sets `age` to 1 then `++age` makes it 2 so $4 \ll 2 = 16$; splitting `"010%"` on `’1’` gives

["0","0%"] so index 1 then is '0'; splitting on '0' gives ["","1","%"] so index 1 then is '1', and ~-1 is 0 so "123" is '1' which parses to 1.

3.1.3 Snippet 3



```
using System;

class Program
{
    static void Main()
    {
        int[] nums = {0, 1, 0, 3, 12};
        int pos = 0;

        for (int i = 0; i < nums.Length; i++)
        {
            if (nums[i] != 0)
            {
                int temp = nums[pos];
                nums[pos] = nums[i];
                nums[i] = temp;
                pos++;
            }
        }

        Console.WriteLine(string.Join(", ", nums));
    }
}
```



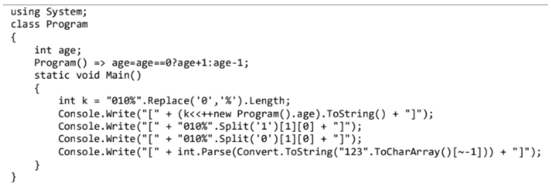
```
1, 3, 12, 0, 0
```

Figure 5: Output

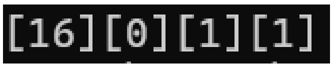
Explanation: the loop compacts nonzeros forward by swapping `nums[i]` with `nums[pos]` when nonzero and incrementing `pos`, preserving the order of `1, 3, 12` and leaving two zeros at the end.

3.2 Level 1

3.2.1 Snippet 1



```
using System;
class Program
{
    int age;
    Program() => age=age==0?age+1:age-1;
    static void Main()
    {
        int k = "010%".Replace('0','%').Length;
        Console.Write("[" + (k<<++new Program().age).ToString() + "]");
        Console.Write("[" + "010%".Split('1')[1][0] + "]");
        Console.Write("[" + "010%".Split('0')[1][0] + "]");
        Console.Write("[" + int.Parse(Convert.ToString("123".ToCharArray()[~-1])) + "]");
    }
}
```

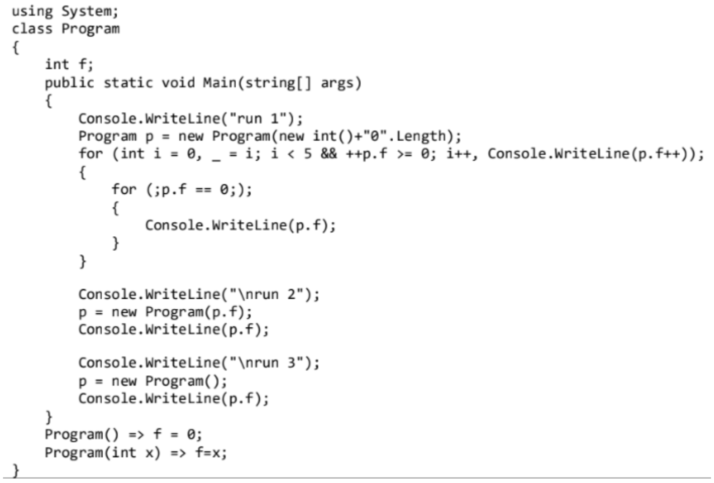


```
[16][0][1][1]
```

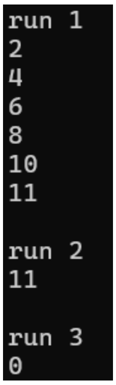
Figure 6: Output

Explanation: This is identical to Level 0, Snippet 2. Identical expression-bodied constructor initializes `age` to `1`, then `++age` is `2`, giving $4 \ll 2 = 16$, and the same string `Split` and bitwise steps yield `0, 1, 1` respectively.

3.2.2 Snippet 2



```
using System;
class Program
{
    int f;
    public static void Main(string[] args)
    {
        Console.WriteLine("run 1");
        Program p = new Program(new int()+"0".Length);
        for (int i = 0, _ = i; i < 5 && ++p.f >= 0; i++, Console.WriteLine(p.f++));
        {
            for (;p.f == 0;);
            {
                Console.WriteLine(p.f);
            }
        }

        Console.WriteLine("\nrun 2");
        p = new Program(p.f);
        Console.WriteLine(p.f);

        Console.WriteLine("\nrun 3");
        p = new Program();
        Console.WriteLine(p.f);
    }
    Program() => f = 0;
    Program(int x) => f=x;
}
```



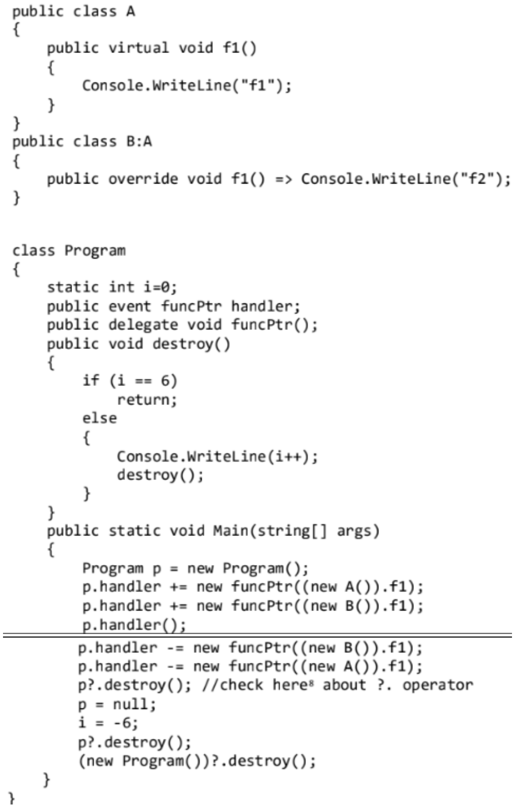
```
run 1
2
4
6
8
10
11

run 2
11

run 3
0
```

Figure 7: Output

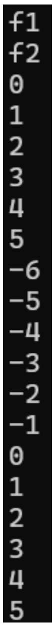
Explanation: `i` is a `static` variable, so it retains its value between calls.

- **run 1:** The `for` loop increments `j` by 2. Inside the loop, `i` (which is static) is incremented by 2 and printed five times (2, 4, 6, 8, 10). After the loop, `i++` makes `i` = 11, which is printed.
- **run 2:** while `f` ends at `11`. The following empty `for(; p.f == 0;)` does nothing and then prints `11`.
- **run 3:** Reinitializing with `Program(p.f)` preserves `11`, and the default `Program()` sets `f` back to `0`.

3.2.3 Snippet 3



```
public class A
{
    public virtual void f1()
    {
        Console.WriteLine("f1");
    }
}
public class B:A
{
    public override void f1() => Console.WriteLine("f2");
}


class Program
{
    static int i=0;
    public event funcPtr handler;
    public delegate void funcPtr();
    public void destroy()
    {
        if (i == 6)
            return;
        else
        {
            Console.WriteLine(i++);
            destroy();
        }
    }
    public static void Main(string[] args)
    {
        Program p = new Program();
        p.handler += new funcPtr((new A()).f1);
        p.handler += new funcPtr((new B()).f1);
        p.handler();
        p.handler -= new funcPtr((new B()).f1);
        p.handler -= new funcPtr((new A()).f1);
        p?.destroy(); //check here[8] about ?. operator
        p = null;
        i = -6;
        p?.destroy();
        (new Program())?.destroy();
    }
}
```



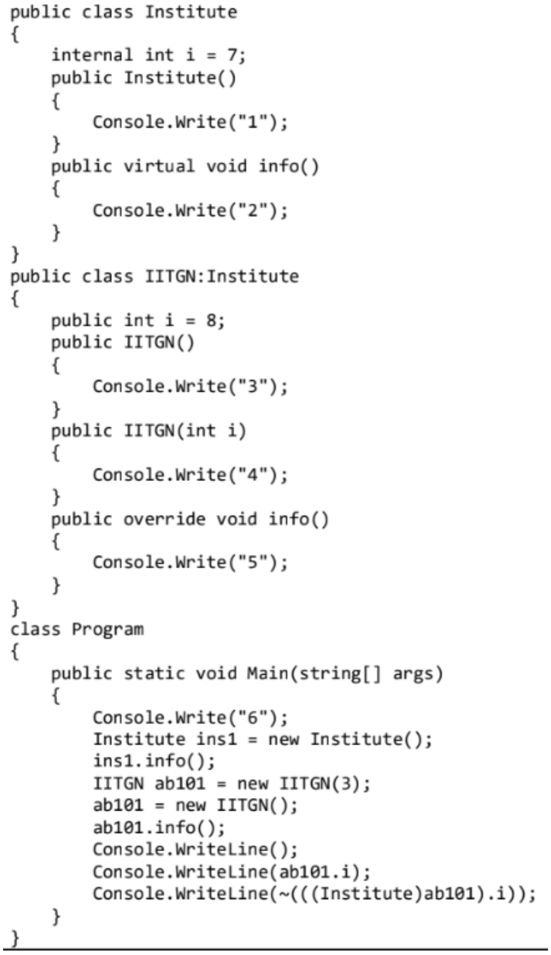
```
f1
f2
0
1
2
3
4
5
-6
-5
-4
-3
-2
-1
0
1
2
3
4
5
```

Figure 8: Output

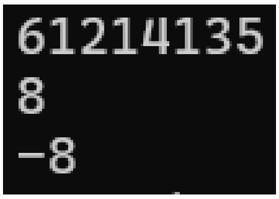
Explanation: The multicast event calls `A.f1` then `B.f1`. Delegate removals against new `A()`/`B()` instances do not match earlier subscriptions, so the handler remains unaffected. `p?.destroy` prints `0..5` then returns at `i == 6`, `p` becomes `null` and `i` is set to `-6`. `p?.destroy` does nothing, and a fresh `Program().destroy` prints from `-6` up to `5`.

3.3 Level 2

3.3.1 Snippet 1



```
public class Institute
{
    internal int i = 7;
    public Institute()
    {
        Console.Write("1");
    }
    public virtual void info()
    {
        Console.Write("2");
    }
}
public class IITGN:Institute
{
    public int i = 8;
    public IITGN()
    {
        Console.Write("3");
    }
    public IITGN(int i)
    {
        Console.Write("4");
    }
    public override void info()
    {
        Console.Write("5");
    }
}
class Program
{
    public static void Main(string[] args)
    {
        Console.Write("6");
        Institute ins1 = new Institute();
        ins1.info();
        IITGN ab101 = new IITGN(3);
        ab101 = new IITGN();
        ab101.info();
        Console.WriteLine();
        Console.WriteLine(ab101.i);
        Console.WriteLine(~(((Institute)ab101).i));
    }
}
```

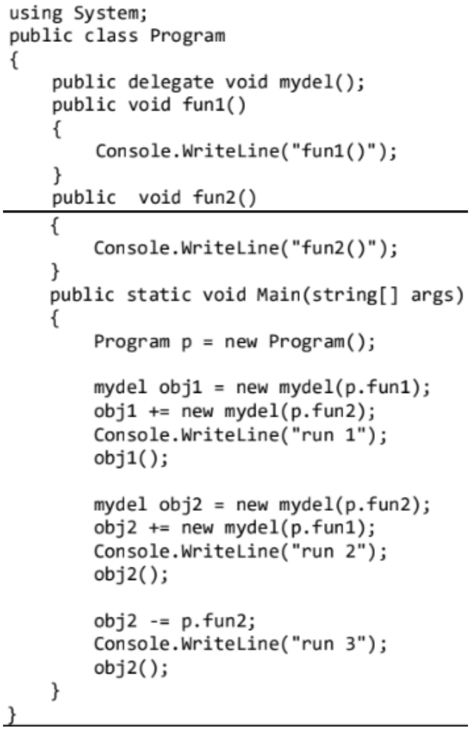


```
61214135
8
-8
```

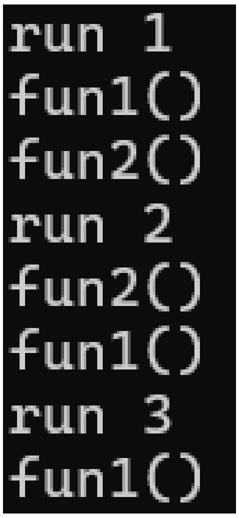
Figure 9: Output

Explanation: Printing starts with `"6"`. `Institute()` prints `"1"` and `info()` prints `"2"`. `IITGN(3)` implicitly calls the base constructor first and then the derived one, producing `"14"`. `IITGN()` writes `"13"`, and `ab101.info()` writes `"5"`. `IITGN.i` is 8, and casting to `Institute` accesses the base field where `i = 7`, hence the printed relation is `7 = -8`.

3.3.2 Snippet 2



```
using System;
public class Program
{
    public delegate void mydel();
    public void fun1()
    {
        Console.WriteLine("fun1()");
    }
    public  void fun2()
    {
        Console.WriteLine("fun2()");
    }
    public static void Main(string[] args)
    {
        Program p = new Program();

        mydel obj1 = new mydel(p.fun1);
        obj1 += new mydel(p.fun2);
        Console.WriteLine("run 1");
        obj1();

        mydel obj2 = new mydel(p.fun2);
        obj2 += new mydel(p.fun1);
        Console.WriteLine("run 2");
        obj2();

        obj2 -= p.fun2;
        Console.WriteLine("run 3");
        obj2();
    }
}
```

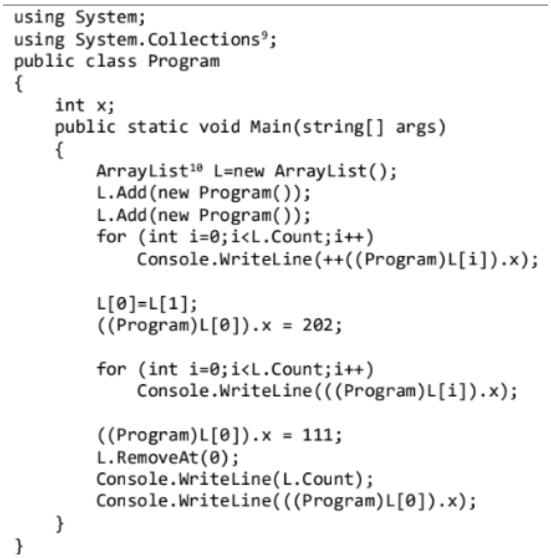


```
run 1
fun1()
fun2()
run 2
fun2()
fun1()
run 3
fun1()
```

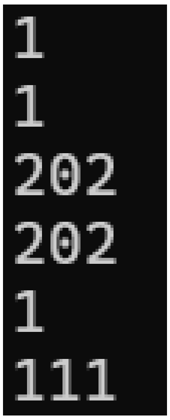
Output

Explanation: `obj1` invokes `fun1()` then `fun2()` in that order. `obj2` invokes `fun2()` followed by textttfun1(). Removing `fun2` from `obj2` leaves only `fun1()`, which executes in the final call.

3.3.3 Snippet 3



```
using System;
using System.Collections[9];
public class Program
{
    int x;
    public static void Main(string[] args)
    {
        ArrayList[10] L=new ArrayList();
        L.Add(new Program());
        L.Add(new Program());
        for (int i=0;i<L.Count;i++)
            Console.WriteLine(++((Program)L[i]).x);

        L[0]=L[1];
        ((Program)L[0]).x = 202;

        for (int i=0;i<L.Count;i++)
            Console.WriteLine(((Program)L[i]).x);

        ((Program)L[0]).x = 111;
        L.RemoveAt(0);
        Console.WriteLine(L.Count);
        Console.WriteLine(((Program)L[0]).x);
    }
}
```



```
1
1
202
202
1
111
```

Figure 10: Output

Explanation: Two fresh `Program` objects print `++x` as `1` and `1`. Aliasing with `L = L` makes both entries reference the same object, so setting `x = 202` echoes twice. After setting `x = 111` and removing index `0`, the list size becomes `1`, and the remaining object's `x` is `111`.

4 Conclusion

This lab provided a comprehensive practical exercise in core C# and object-oriented programming principles. Through the first task, the object lifecycle was clearly observed, demonstrating how constructors initialize objects and how destructors are called upon program termination, with static variables effectively tracking the count of active instances.

The second task offered a clear demonstration of inheritance and polymorphism. By overriding the `Drive()` and `ShowInfo()` methods in derived `Car` and `Truck` classes, we successfully invoked specialized behavior from a base `Vehicle` array, confirming the power of runtime polymorphism.

Finally, the output reasoning challenges reinforced a deeper understanding of C# specifics, including operator precedence, static variable behavior across instances, multicast delegates, and the critical difference between reference and value types, as seen in the `ArrayList` example. The lab objectives were successfully met, solidifying both theoretical concepts and their practical application in C#.