

3.2P: Answer Sheet

Recall task 2.2P *Counter Class* and answer the following questions.

1. How many *Counter* objects were created?

A total of two Counter objects were created.

myCounters[0] and myCounters[1] each create a new Counter object, while myCounters[2] simply references the same object as myCounters[1], not a new one.

2. Variables declared without the **new** keyword are different to the objects created using **new**. In the **Main** function, what is the relationship between the variables initialized with and without the **new** keyword?

In the Main function, variables declared with the new keyword (myCounters[0] and myCounters[1]) create new Counter objects.

The variable assigned without new (myCounters[2] = myCounters[1]) **shares the reference** to an already existing object (myCounters[1]'s reference object) rather than creating a new one.

3. In the **Main** function, explain why the statement **myCounters[2].Reset()**; also changes the value of **myCounters[0]**.

The statement myCounters[2].Reset(); does not change the value of myCounters[0].

It only affects myCounters[1], because myCounters[2] and myCounters[1] point to the same Counter object, while myCounters[0] is a separate, independent object and only have one reference.

4. The difference between *heap* and *stack* is that heap holds “*dynamically allocated memory*.” What does this mean? In your answer, focus on the size and lifetime of the allocations.

Dynamically allocated memory refers to memory that is allocated on the heap at runtime, rather than at compile time.

The sizes of heap allocations are vary and are determined at runtime, with the lifetime of the allocated memory remaining until it is explicitly released or garbage collected.

Stack allocations, on the other hand, usually have a fixed size and are deallocated automatically when the function that created them finishes.

5. Are objects allocated on the heap or on the stack? What about local variables?

Objects (created with `new`) are allocated on the heap.

Local variables (such as references to objects) are stored on the stack, but they only hold the memory address (reference) pointing to the heap object.

6. What is the meaning of the expression `new ClassName()`, where *ClassName* refers a class in your application? What is the value of this expression?

The expression `new ClassName()` creates a new instance (object) of the specified class in the heap memory.

The value of this expression is a reference (memory address) pointing to the newly created object.

7. Consider the statement “`Counter myCounter;`”. What is the value of *myCounter* after this statement? Why?

The value of `myCounter` after the statement `Counter myCounter;` is `null`. This is because the statement declares a variable of type `Counter` (a reference type) without initializing it. In C#, uninitialized reference type variables are automatically assigned `null`, indicating they do not yet reference any object in memory.

8. Based on the code you wrote in task 2.2P *Counter Class*, draw a diagram showing the locations of the variables and objects in function *Main* and their relationships to one another.



Main

myCounters (Counter[]) ->
[reference to array in heap]

Array: Counter[3]

[0] ----> Counter Obj 1

[1] ----> Counter Obj 2

[2] ----> Counter Obj 2 (same reference as [1])

Counter Object 1:

Counter

_name: "Counter 1"

_count: 9

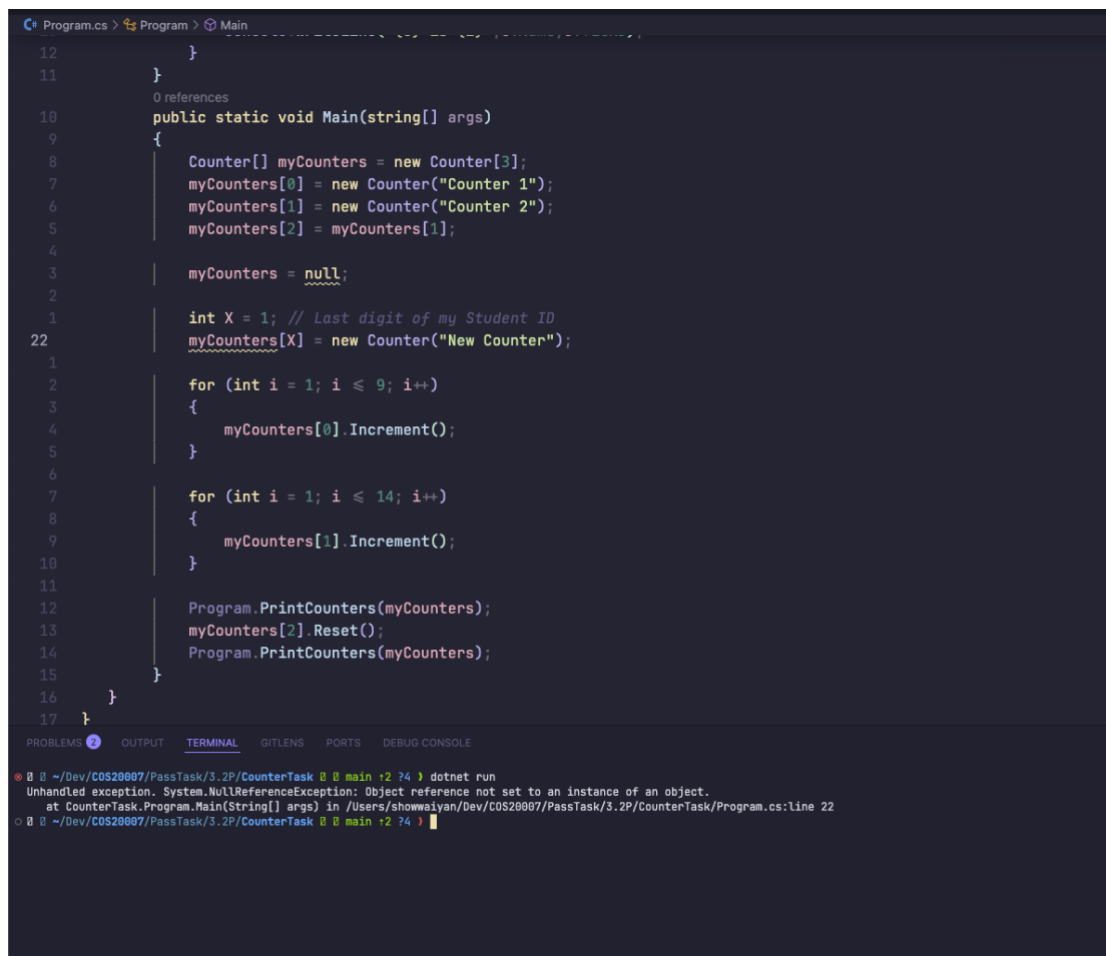
Counter Object 2:

Counter

_name: "Counter 2"

_count: 0 (reset via myCounters[2].Reset())

9. If the variable `myCounters` is assigned to `null`, then you want to change the value of `myCounters[X]`, where `X` is the last digit of your student ID, what will happen? Please provide your observation with screenshots and explanation.



```
12     }
11     }
10     public static void Main(string[] args)
9     {
8         Counter[] myCounters = new Counter[3];
7         myCounters[0] = new Counter("Counter 1");
6         myCounters[1] = new Counter("Counter 2");
5         myCounters[2] = myCounters[1];
4
3         myCounters = null;
2
1         int X = 1; // Last digit of my Student ID
22        myCounters[X] = new Counter("New Counter");
1
2        for (int i = 1; i <= 9; i++)
3        {
4            myCounters[0].Increment();
5        }
6
7        for (int i = 1; i <= 14; i++)
8        {
9            myCounters[1].Increment();
10        }
11
12        Program.PrintCounters(myCounters);
13        myCounters[2].Reset();
14        Program.PrintCounters(myCounters);
15    }
16 }
17 }
```

PROBLEMS OUTPUT TERMINAL GITLENS PORTS DEBUG CONSOLE

~/Dev/COS20007/PassTask/3.2P/CounterTask 0 0 main +2 ?4) dotnet run
Unhandled exception. System.NullReferenceException: Object reference not set to an instance of an object.
at CounterTask.Program.Main(String[] args) in /Users/showmalyan/Dev/COS20007/PassTask/3.2P/CounterTask/Program.cs:line 22
~/Dev/COS20007/PassTask/3.2P/CounterTask 0 0 main +2 ?4)

When `myCounters` is `null`, it no longer points to the 3 members of `Counter` array on the heap. Attempting to access `myCounters[1]` (for instance, to assign a new object or call a method such as `Increment()`) will result in a `NullReferenceException` because `myCounters` is `null`, and we can't index into a `null` reference.

In C#, arrays are reference types on the heap. The variable `myCounters` has a reference to the array. Setting `myCounters = null` eliminates this reference, so the array cannot be accessed using `myCounters`. Any dereference of `myCounters` (for example, `myCounters[1]`) will result in a `NullReferenceException` because there is no object to operate on.

Hint. You may want to read this material for this task <https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/null> For further reading at your own.

- Null pointer CrowdStrike Bug, <https://www.thestack.technology/crowdstrike-nullpointer-blamed-rca/>
- CrowdStrike Blog, <https://www.crowdstrike.com/blog/tech-analysis-channel-file-maycontain-null-bytes/>