3.2P: Answer Sheet

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

1. How many *Counter* objects were created?
   2

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?
   Without using new, we only get references that can point to objects, but no actual objects are created

3. In the ***Main*** function, explain why the statement ***myCounters***[2].***Reset***(); also changes the value of ***myCounters***[0].
   This happens because myCounters[2] and myCounters[0] reference the same Counter object. This is shown in the code where myCounters[2] = myCounters[0] assigns the same reference. When we call Reset() on either reference, it affects the same underlying object.

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.

- Heap memory is dynamically allocated, meaning:
- Size is determined at runtime
- Memory persists until explicitly freed or garbage collected
- Can grow and shrink as needed
- Objects can outlive the function that created them
- In our Counter example, the Counter objects are allocated on the heap and can exist beyond the Main function's scope

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

Heap Memory:

● Size: You can ask for different amounts of memory as needed.

● Lifetime: It can last beyond the function that created it.

Stack Memory:

● Size: It's fixed and determined when the program is written.

● Lifetime: It's automatically freed when the function using it ends.

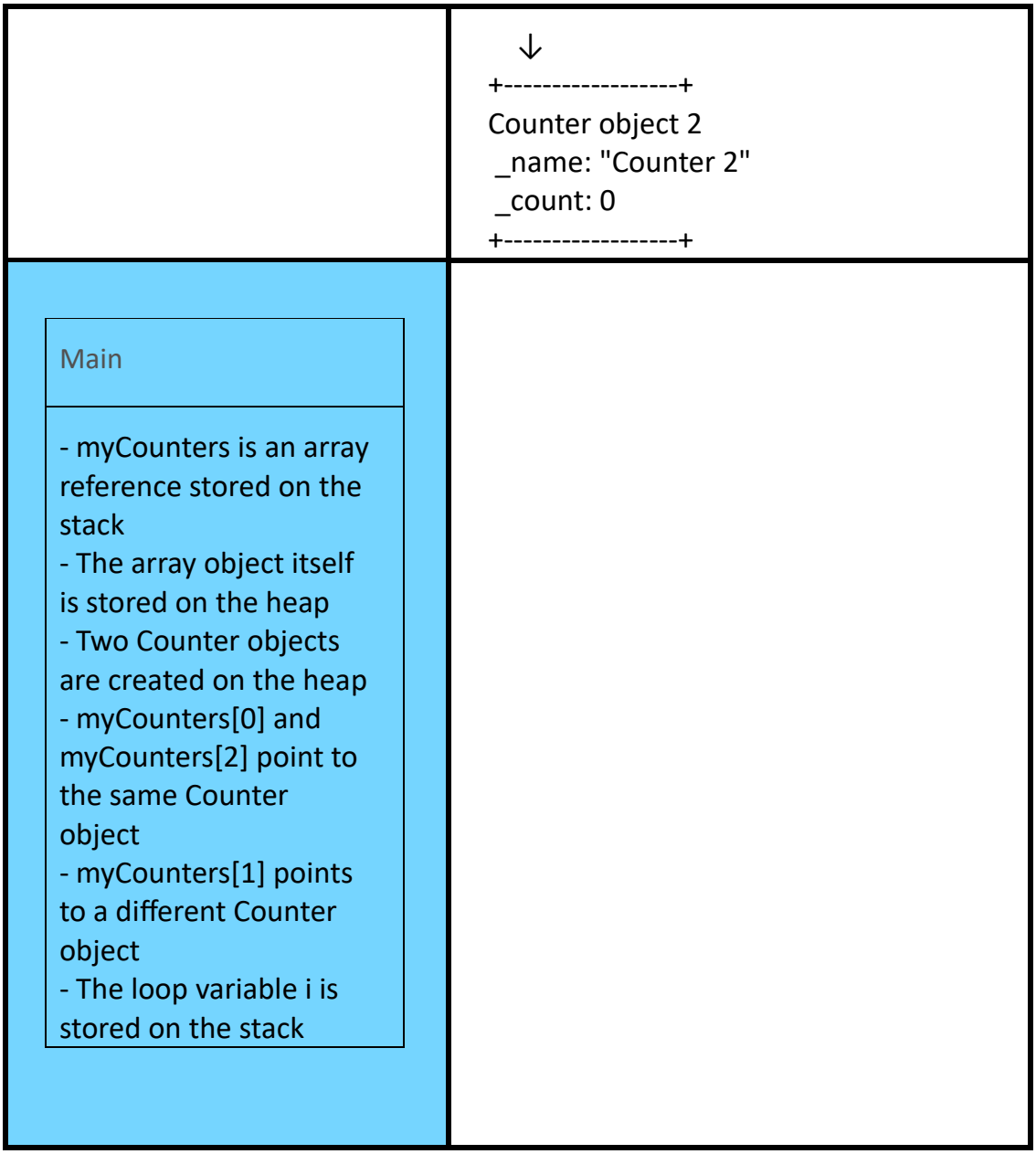
6. What is the meaning of the expression ***new*** *ClassName*(), where *ClassName* refers a class in your application? What is the value of this expression?
   The new keyword is used to create a new instance of a class. When using the new keyword with a class, it allocates memory for a new object of that class and returns a reference to that object.

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

- After Counter myCounter;, the value of myCounter is null
- This is because we've only declared a reference variable but haven't initialized it
- The variable exists on the stack but doesn't point to any object
- We would need to use new Counter(...) to create an actual object for it to reference

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.

Stack MyCounter i	Heap Counter[3] [0] → Counter 1 [1] → Counter 2 [2] → Counter 1 +------------------+ ↓ Counter object 1 _name: "Counter 1" _count: 0 +------------------+



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 explaination.**

If myCounters is set to null and we try to access myCounters[X], we would get a NullReferenceException because:

- A null reference means the variable doesn't point to any object
- Trying to access an element of a null array is like trying to access a non-existent object

- This is a common runtime error in C# when trying to use a null reference
- The program would crash with a NullReferenceException at the point of access

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/crowstrike-nullpointer-blamed-rca/
- CrowdStrike Blog, https://www.crowdstrike.com/blog/tech-analysis-channel-file-maycontain-null-bytes/