3.2P: Answer Sheet

Student's name: Duong Ha Tien Le

Student's ID: 104700948

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

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

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, myCounter[0] and myCounter[1] are initialized with the new keyword, each variable points to a separate object. These variables store references to where the objects are located in memory. However, myCounter[2], which is declared without the new keyword and is assigned the value of myCounter[0], references the same object as myCounter[0]. This means that myCounter[0] and myCounter[2] are linked to the same memory location, so any changes made to the object via myCounter[2] will also be reflected when accessing the object through myCounter[0].

3. In the ***Main*** function, explain why the statement ***myCounters***[2].***Reset***(); also changes the value of ***myCounters***[0].
   - As the given answer for question 2, the pointers of myCounter[2] and myCounter[0] point to the same Counter object in memory. Hence, when resetting the value of myCounter[2], the value of myCounter[0] will also be affected.
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.
   - This means that memory is allocated during the program's runtime as needed. The size of this memory can be accessible and determined when the whole application runs, so it can be larger and more flexible compared to stack memory. The lifetime of heap memory is also different: it remains allocated until destroyed by the program or when the program ends. In contrast, stack memory is typically used for static memory allocation and has a fixed size. In

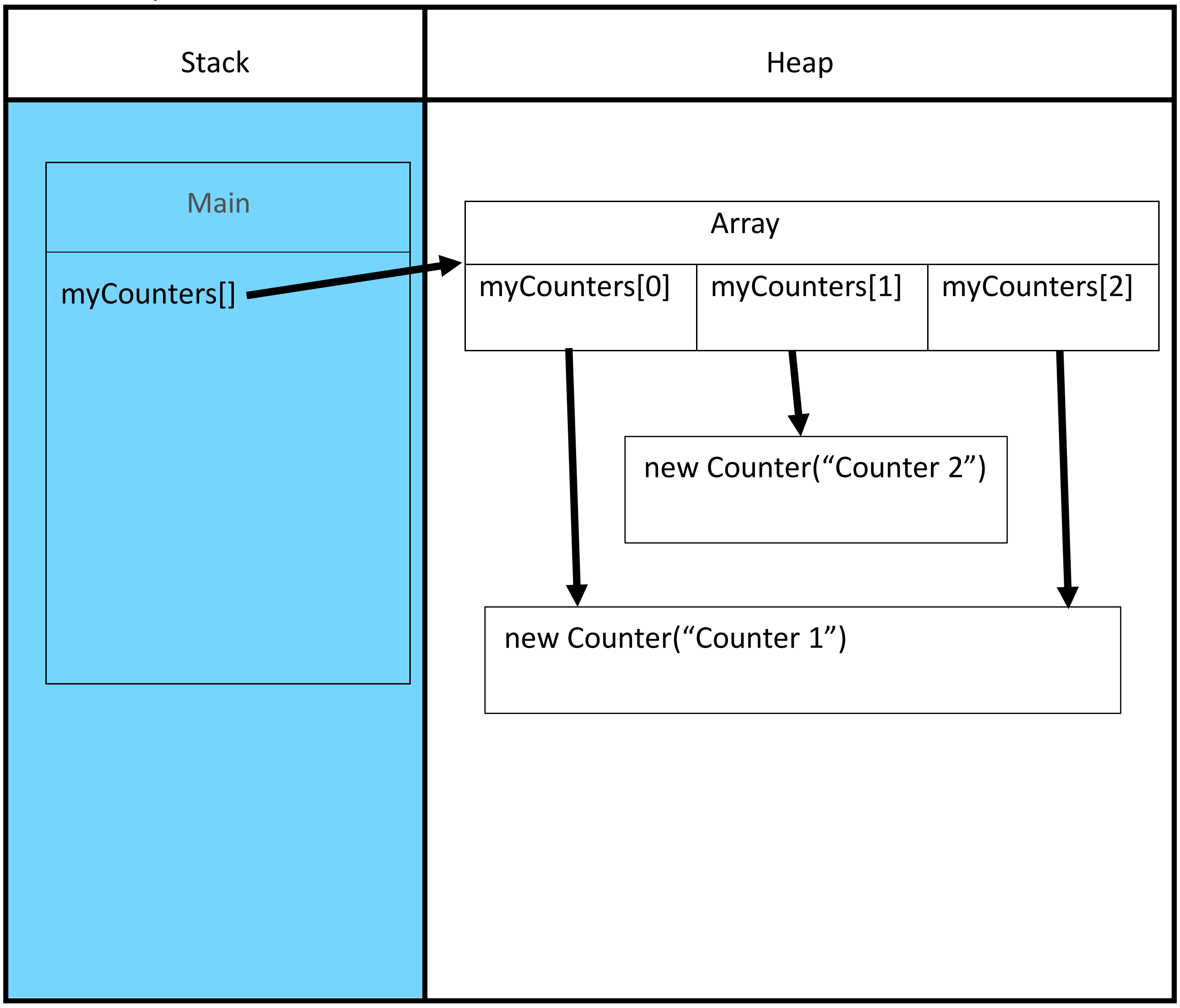
terms of the lifetime allocation, it is a temporary phase which means when the object's task is complete, it is automatically removed from the stack.

5. Are objects allocated on the heap or on the stack? What about local variables?
   - Objects (reference) are allocated on the heap. Meanwhile, local variables (value) are allocated on the stack.

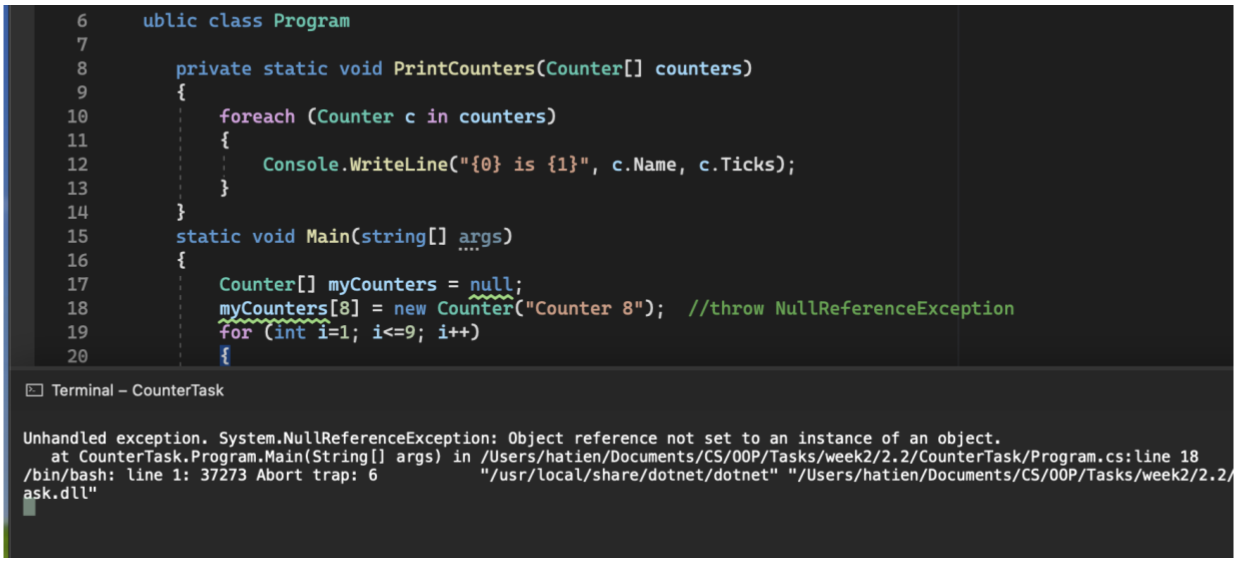
6. What is the meaning of the expression ***new*** *ClassName*(), where *ClassName* refers a class in your application? What is the value of this expression?
   - The expression is used to create a new instance of the class. The value of this expression is a reference to the newly created object, which can be assigned to a variable.

7. Consider the statement “*Counter* ***myCounter***;”. What is the value of ***myCounter*** after this statement? Why?
   - In this case, it can be seen that the Counter object hasn’t created any instances since there was no “=” after that statement. That means myCounter has no reference to Counter object or holds any data. Hence, the value will be null as a reference type.

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.



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.**



```
ublic class Program

    private static void PrintCounters(Counter[] counters)
    {
        foreach (Counter c in counters)
        {
            Console.WriteLine("{0} is {1}", c.Name, c.Ticks);
        }
    }
    static void Main(string[] args)
    {
        Counter[] myCounters = null;
        myCounters[8] = new Counter("Counter 8");  //throw NullReferenceException
        for (int i=1; i<=9; i++)
        {
```

Terminal – CounterTask

```
Unhandled exception. System.NullReferenceException: Object reference not set to an instance of an object.
   at CounterTask.Program.Main(String[] args) in /Users/hatien/Documents/CS/OOP/Tasks/week2/2.2/CounterTask/Program.cs:line 18
/bin/bash: line 1: 37273 Abort trap: 6           "/usr/local/share/dotnet/dotnet" "/Users/hatien/Documents/CS/OOP/Tasks/week2/2.2/
ask.dll"
```

- In the provided screenshot, I assigned myCounters as a null value, which means it doesn’t reference any object. Then, when changing the value of myCounters[8], the application will notify an error message “NullReferenceException”, that is because myCounters doesn’t point to any object or any arrays that can be accessed. In short, the null references can affect deeply the behavior of the whole program and somehow lead to runtime errors.

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/