The Basics Of Object-Lifetime

Classes, Objects and References.

Classes, objects, and references are fundamental concepts in object-oriented programming (OOP) languages like C#.

1. Classes:

- A class is a blueprint or template for creating objects in a program.
- It defines the properties (attributes) and behaviors (methods) that objects of the class will have.
- Classes encapsulate data and behavior, providing a way to organize and structure code.

Example:

```
// Class definition
public class Car
{
    // Class members
    public string Make;
    public string Model;
    public int Year;
    // Constructor
    public Car(string make, string model, int year)
    {
        Make = make;
        Model = model;
        Year = year;
    }
    // Method
    public void DisplayDetails()
    {
        Console.WriteLine($"Make: {Make}, Model: {Model}, Year: {Year}");
    }
}
```

2. Objects:

- An object is an instance of a class.
- When you create an object, you are instantiating a class and allocating memory for its data members.
- Each object has its own set of data members, but shares the methods defined in its class.

Example:

```
// Creating objects of the Car class
Car car1 = new Car("Toyota", "Camry", 2020);
Car car2 = new Car("Honda", "Accord", 2019);
```

3. References:

- In C#, objects are accessed through references.
- A reference is a variable that holds the memory address of an object.
- When you create an object, you are actually creating it on the heap memory, and the reference holds the address of this memory location.
- Multiple references can refer to the same object, allowing for sharing and passing of objects between different parts of the program.

Example:

```
// Creating references to car objects
Car reference1 = car1;
Car reference2 = car2;
```

Example to demonstrate Classes, Objects and References.

```
using System;

// Define a class
class MyClass
{
    public int MyProperty { get; set; }

    public void MyMethod()
    {
        Console.WriteLine("Hello from MyClass");
```

```
    }
}

class Program
{
    static void Main()
    {
        // Create an object of MyClass
        MyClass obj1 = new MyClass();

        // Set property and call method
        obj1.MyProperty = 10;
        obj1.MyMethod();

        // Create another reference to the same object
        MyClass obj2 = obj1;

        // Accessing the same object through both references
        Console.WriteLine(obj2.MyProperty); // Output: 10
        obj2.MyMethod(); // Output: Hello from MyClass
    }
}
```

The basics of Object-Lifetime

Object lifetime is the time when a block of memory is allocated to this object during some process of execution and that block of memory is released when the process ends. Once the object is allocated with memory, it is necessary to release that memory so that it is used for further processing, otherwise, it would result in memory leaks. We have a class in .Net that releases memory automatically for us when the object is no longer used. We will try to understand the entire scenario thoroughly of how objects are created and allocated memory and then deallocated when the object is out of scope.

The class is a blueprint that describes how an instance of this type will look and feel in memory. This instance is the object of that class type. A block of memory is allocated when the new keyword is used to instantiate the new object and the constructor is

called. This block of memory is big enough to hold the object. When we declare a class variable it is allocated on the stack and the time it hits a new keyword and then it is allocated on the heap. In other words, when an object of a class is created it is allocated on the heap with the C# new keyword operator. However, a new keyword returns a reference to the object on the heap, not the actual object itself. This reference variable is stored on the stack for further use in applications.

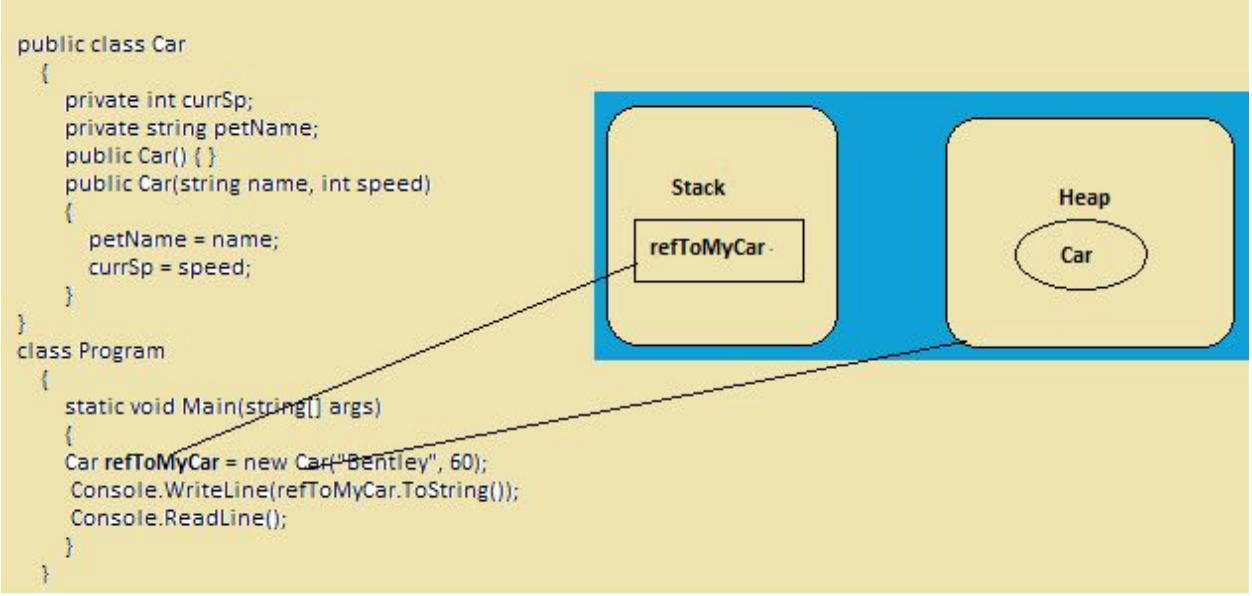

Diagram - new keyword returns a reference to the object on the heap and the actual reference variable is stored on stack.

When the new operator is used to create an object, memory is taken from the managed heap for this object and the managed heap is more than just a random chunk of memory accessed by the CLR. When the object is no longer used then it is de-allocated from the memory so that this memory can be reused.

The key pillar of the .NET Framework is the automatic garbage collection that manages memory for all .NET applications. When an object is instantiated the garbage collector will destroy the object when it is no longer needed. There is no explicit memory deallocation since the garbage collector monitors unused objects and does a collection to free up memory that is an automatic process. The Garbage Collector removes objects from the heap when they are unreachable by any part of your codebase. The .Net garbage collector will compact empty blocks of memory for the purpose of optimization.

The heap is categorized into three generations so it can handle long-lived and short-lived objects. Garbage collection primarily occurs with the reclamation of short-lived objects that typically occupy only a small part of the heap.

Generations ob Garbage collection

There are the following three generations of objects on the heap:

- **Generation 0:** Newly created objects are in Generation 0. These objects on Generation 0 are collected frequently to ensure that short-lived objects are quickly collected and the memory is released. Objects that survive Generation 0, the collections are promoted to Generation 1. Most objects are reclaimed for garbage collection in Generation 0 and do not survive to the next generation.

- **Generation 1:** Objects that are collected less frequently than Generation 0 and contain longer-lived objects that were promoted from Generation 0. Objects that survive Generation 1, collection are promoted to Generation 2.

- **Generation 2:** Objects promoted from Generation 1 that are the longest-lived objects and collected infrequently. The overall strategy of the garbage collector is to collect and move longer-lived objects less frequently.