# Destructor

## Problem Scenario

Suppose we have a Point Class.

```
class Point
{

}
```

## Problem Scenario 01

We need to count the number of active objects in the memory. When object is created, we want to increment the counter.

## Problem Scenario 01

We need to count the number of active objects in the memory. When object is created, we want to increment the counter.

How can we do that?

## Problem Scenario 01

We need to count the number of active objects in the memory. When object is created, we want to increment the counter.

```
class Point
{
    private static int counter = 0;
    public Point()
    {
        counter = counter + 1;
        Console.WriteLine("Point is Created");
    }
}
```

## Problem Scenario 02

We need to count the number of active objects in the memory. When object is created, we want to increment the counter. Also when the object is removed we want to decrement the counter.

## Problem

We know the constructor, but now we need a function where we write the code when any object is removed from the memory.

## Destructor

When object is removed from the memory a function (Destructor) is called.

## Destructor

The destructor function is same like constructor means it has no return type and same name as that of the class.

## Destructor

Also it has 2 additional properties.

1. Destructor can not have access modifier (i.e., public, private).
2. It is prefixed with the tilda sign ( ~ ).

## Destructor

Following is destructor function of the class Point

```
class Point
{
    ~Point()
    {
        counter = counter - 1;
        Console.WriteLine("Point is Removed");
    }
}
```

## Problem

The point class keeps count of all its objects created and when the object is freed from the memory, it updates the counter. Both constructor and destructor are used to keep that count.

```
class Point
{
  private static int counter = 0;
  private int x;
  private int y;

  public Point(int x, int y)
  {
     counter = counter + 1;
      Console.WriteLine("Point is Created");
     this.x = x;
     this.y = y;
  }

  ~Point()
  {
      counter = counter - 1;
      Console.WriteLine("Point is Removed");
  }
}
```

## Destructor

However, to see the destructor in action, there should be two condition fulfilled.

1. Object Reference should not be accessible within the code.
2. Garbage Collector (A program that automatically runs and frees the memory of the object that cannot be referenced)

## Destructor

For seeing the working of the destructor, we can create an object reference in any function call because when functions end the object reference will not be accessible.

## Destructor

When newPoint() function completes its execution, the point variable will go out of scope.

```
static void Main(string[] args)
{
    newPoint();
    //Other Code
}
```

```
public static void newPoint()
{
   Point point = new Point(30,20);
}
```

## Destructor

However, the object reference is getting out of scope does not make sure the destructor is called because it is marked to remove from memory but yet it has not been removed.

```
static void Main(string[] args)
{
    newPoint();
    //Other Code
}
```

```
public static void newPoint()
{
    Point point = new Point(30,20);
}
```

## Garbage Collector

We can guide garbage collector that the time to collect some memory. It initiates the garbage memory collection and hence the call of the destructor.

```
static void Main(string[] args)
{
    newPoint();
    GC.collect();
    //Other Code
}
```

```
public static void newPoint()
{
    Point point = new Point(30,20);
}
```

## Learning Objective

Explain Garbage Collector and its role in calling the Destructor and clearing the memory.

## Conclusion

| Constructor | Destructor |
|---|---|
| The constructor allocates the memory to an object. | Destructor de-allocates the memory of an object. |
| Constructor accepts arguments. | Destructor does not accept any argument. |
| Constructor is called automatically when the object is created. | Destructor is called automatically by the garbage collector, when the object goes out of scope. |
| Constructors are called in successive order. | Destructors are called in reverse order of constructor. |
| There can be multiple constructors of a single class, and this concept is known as constructor overloading. | There is always only a single destructor for one class. |
| Access modifiers are used with constructors. | Access modifier are not allowed on destructor. |