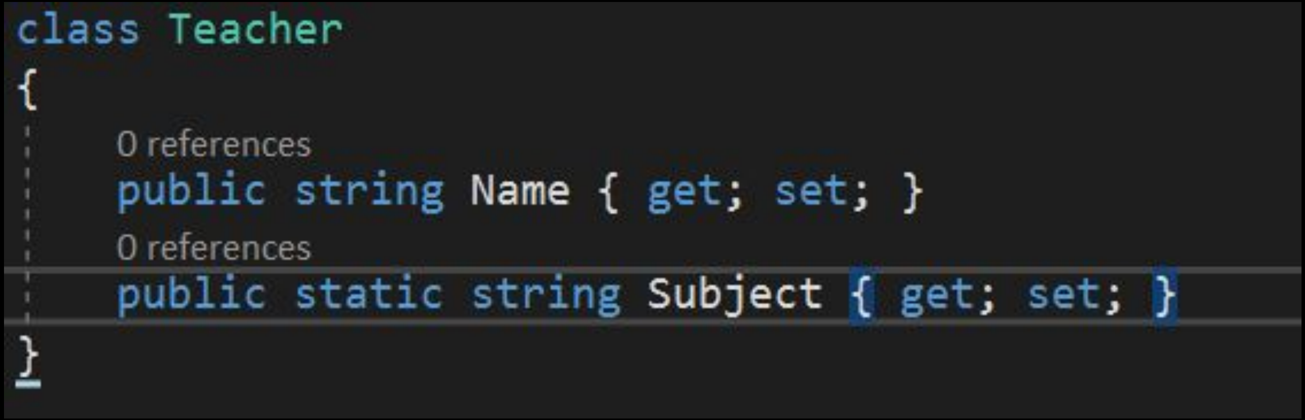
Static

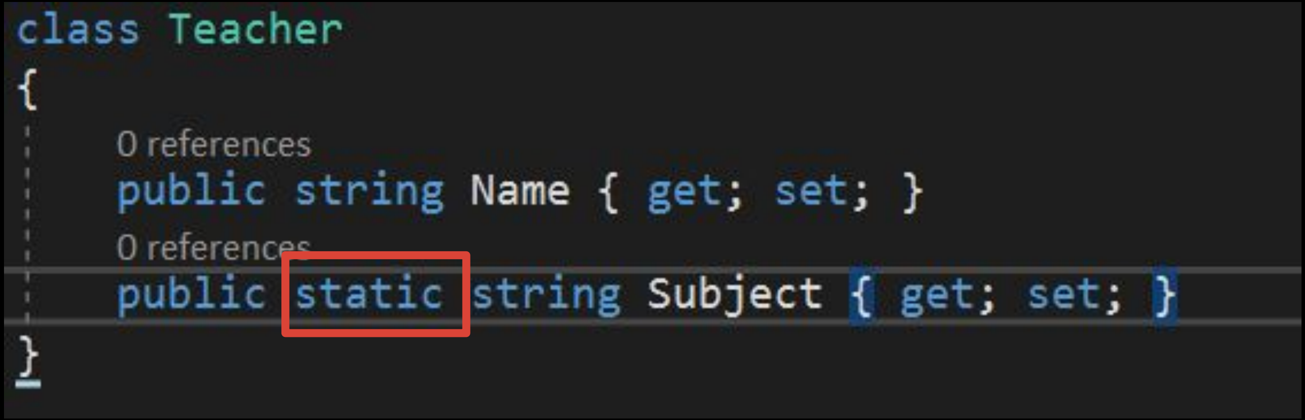
Static means it belongs to the class itself and not to the object/ instance

Two members

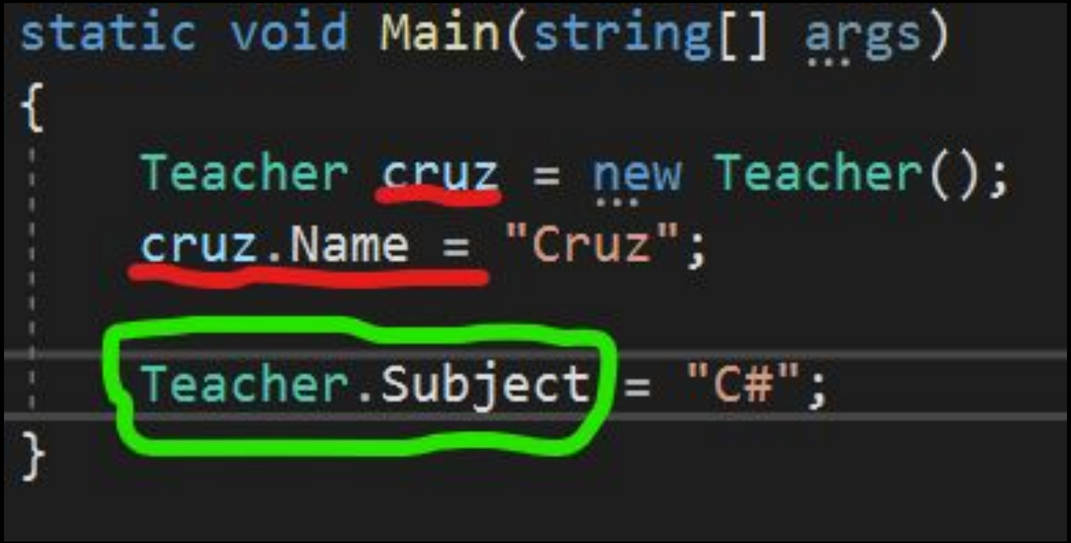


```
class Teacher
{
    0 references
    public string Name { get; set; }
    0 references
    public static string Subject { get; set; }
}
```

One is marked static



```
class Teacher
{
    0 references
    public string Name { get; set; }
    0 references
    public static string Subject { get; set; }
}
```



```
static void Main(string[] args)
{
    Teacher cruz = new Teacher();
    cruz.Name = "Cruz";

    Teacher.Subject = "C#";
}
```

Can be made static

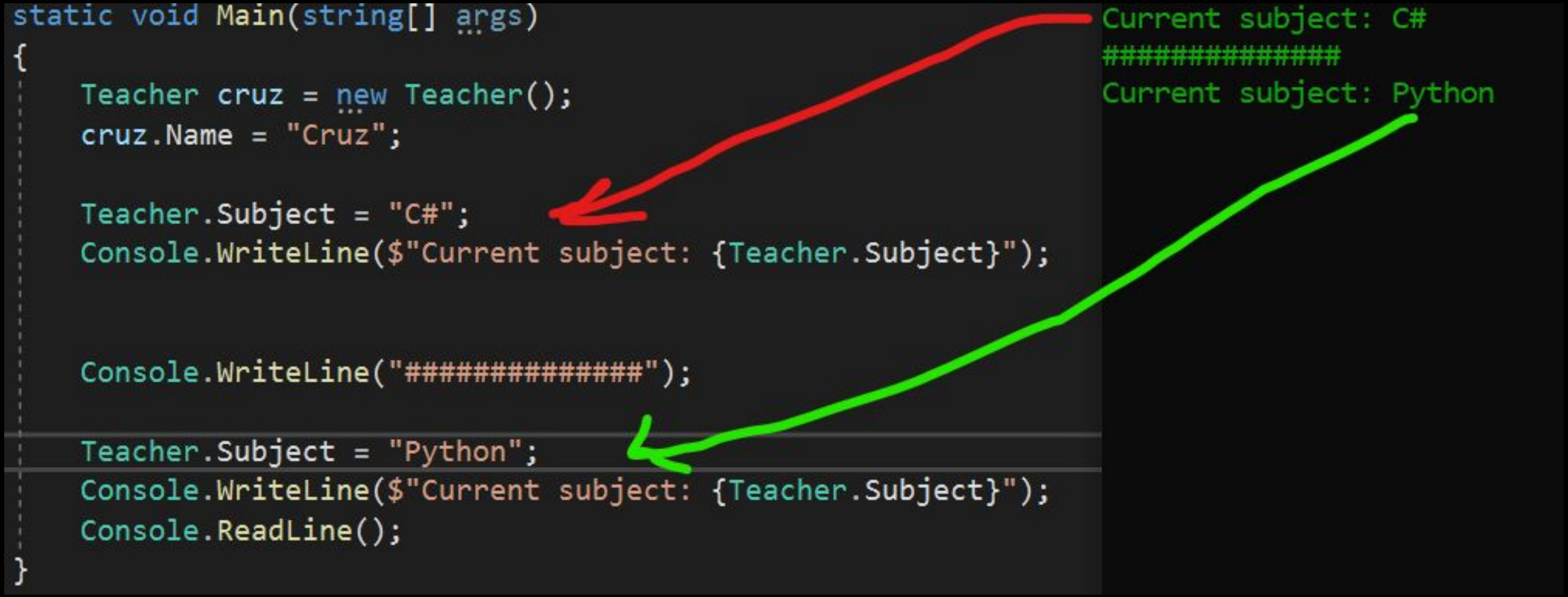
Properties are not the only thing that can be made static. Here is a list of items that can be static:

- **Classes**
- **Fields or Variables**
- **Properties**
- **Methods**
- **Constructors**

When you declare a member static there is only ONE copy of that member.

Static

Teacher.Subject is assigned the value “C#” and it will always be that value until it is reassigned. See the following example:



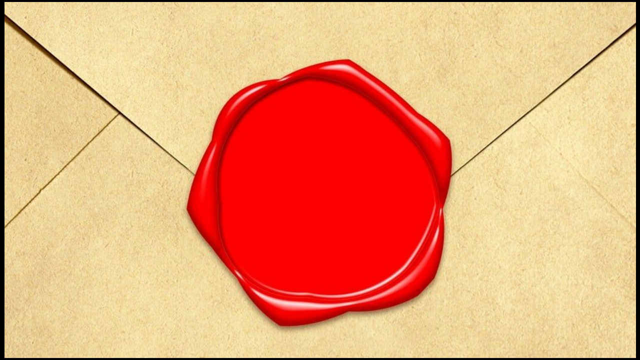
```
static void Main(string[] args)
{
    Teacher cruz = new Teacher();
    cruz.Name = "Cruz";

    Teacher.Subject = "C#";
    Console.WriteLine($"Current subject: {Teacher.Subject}");


    Console.WriteLine("##############");

    Teacher.Subject = "Python";
    Console.WriteLine($"Current subject: {Teacher.Subject}");
    Console.ReadLine();
}
```

```
Current subject: C#
##############
Current subject: Python
```

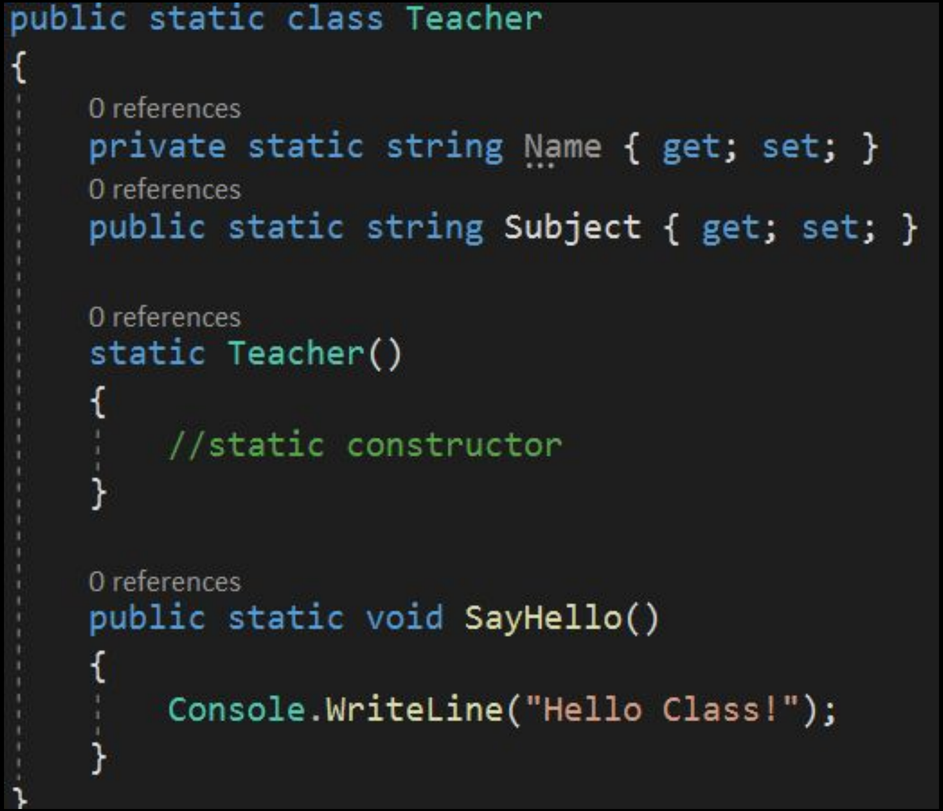


Static classes are sealed.

Meaning you cannot have a child class that inherits from your static class.

When we declare a class as static, **ALL MEMBERS MUST BE STATIC AS WELL!**

Static Class - All members must be static

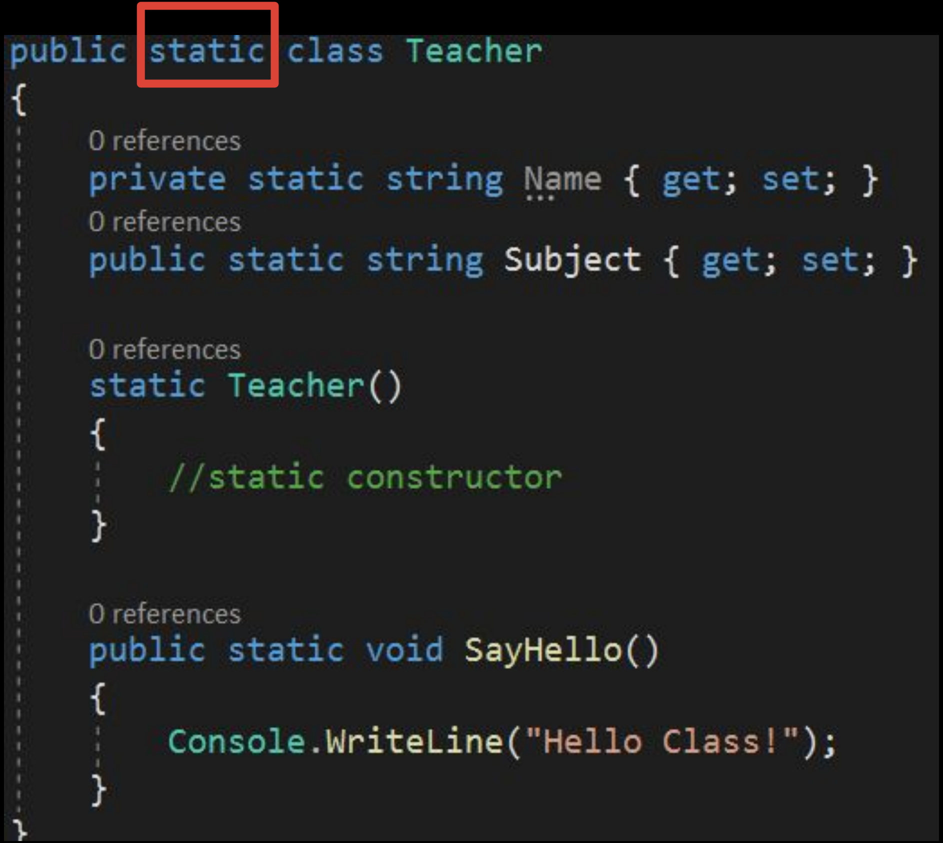


```
public static class Teacher
{
    0 references
    private static string Name { get; set; }
    0 references
    public static string Subject { get; set; }

    0 references
    static Teacher()
    {
        //static constructor
    }

    0 references
    public static void SayHello()
    {
        Console.WriteLine("Hello Class!");
    }
}
```

Static Class - All members must be static

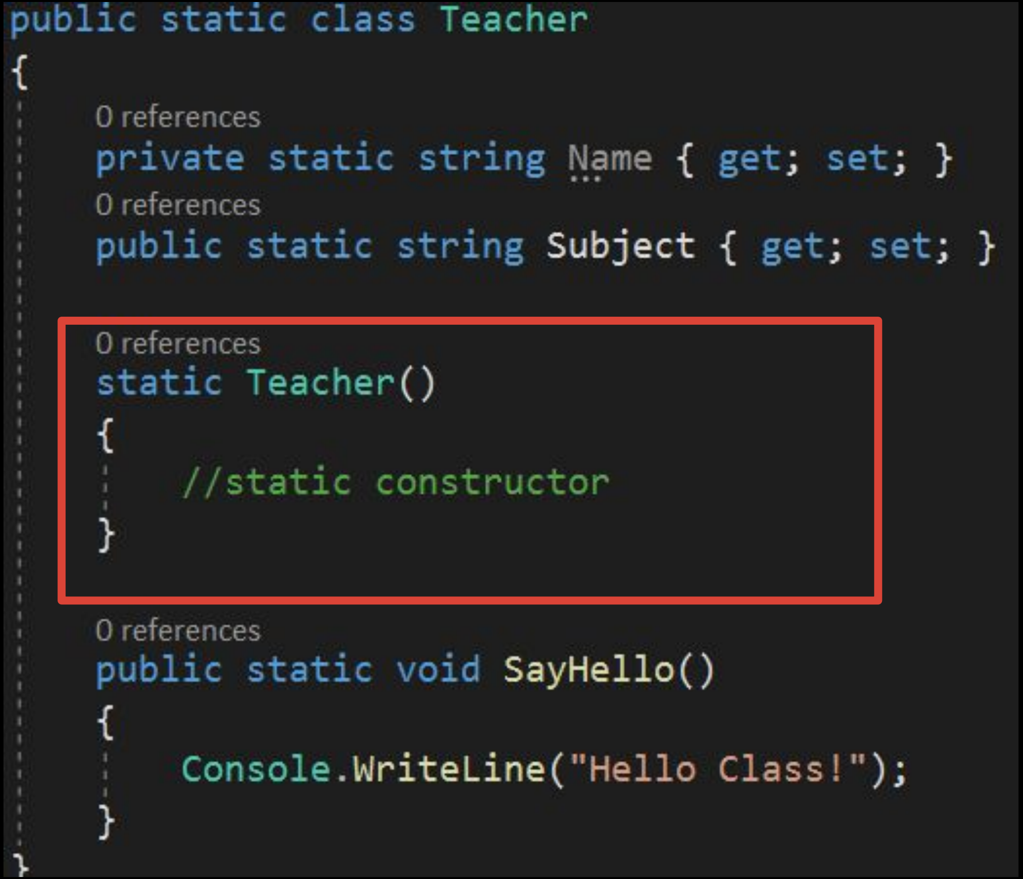


```
public static class Teacher
{
    0 references
    private static string Name { get; set; }
    0 references
    public static string Subject { get; set; }

    0 references
    static Teacher()
    {
        //static constructor
    }

    0 references
    public static void SayHello()
    {
        Console.WriteLine("Hello Class!");
    }
}
```

Static Classes use a Parameterless or Default static constructor.



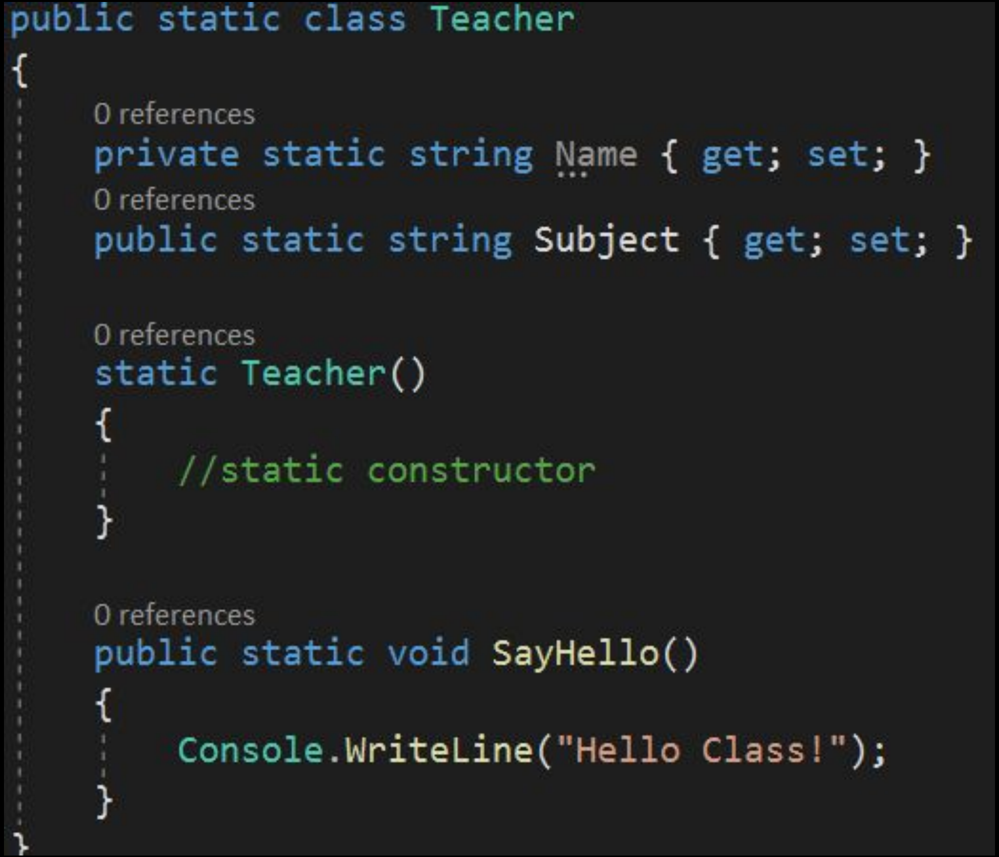
```
public static class Teacher
{
    0 references
    private static string Name { get; set; }
    0 references
    public static string Subject { get; set; }

    0 references
    static Teacher()
    {
        //static constructor
    }

    0 references
    public static void SayHello()
    {
        Console.WriteLine("Hello Class!");
    }
}
```

A static class is a class that cannot be instantiated.

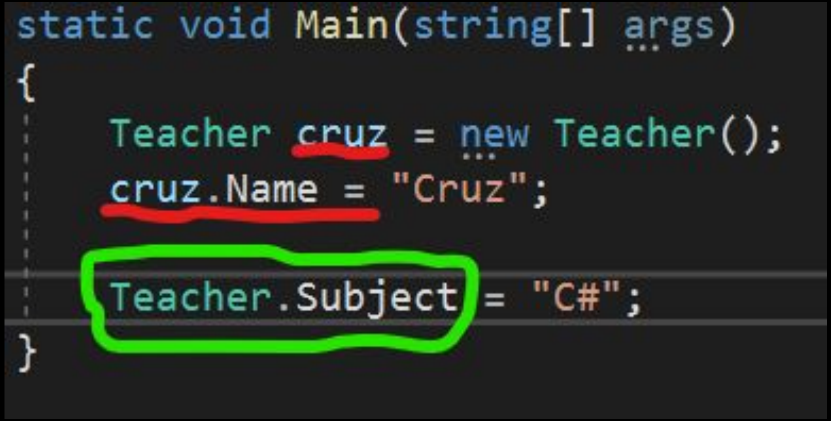
You access the members of a static class by using the class name itself.



```
public static class Teacher
{
    0 references
    private static string Name { get; set; }
    0 references
    public static string Subject { get; set; }

    0 references
    static Teacher()
    {
        //static constructor
    }

    0 references
    public static void SayHello()
    {
        Console.WriteLine("Hello Class!");
    }
}
```



```
static void Main(string[] args)
{
    Teacher cruz = new Teacher();
    cruz.Name = "Cruz";

    Teacher.Subject = "C#";
}
```

What’s the point?

What’s the point?

- In code, **when we create an instance of a class, we are allocating a certain amount of memory for that object to reference.**

```
Car myCar = new Car();
```

Efficiency

- We may have a class that performs a particular function or method in multiple parts of our code. As a programmer, you should always strive to have the **most efficient and readable code possible.**

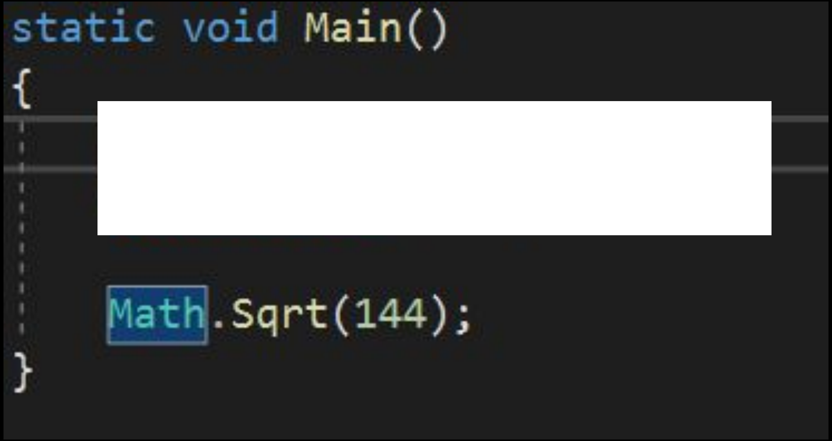
Static Keyword

- Instead of instantiating the object and allocating the memory every time in our code just to invoke a couple of methods, **we can use the static keyword.**

When to make it static

When to make it static

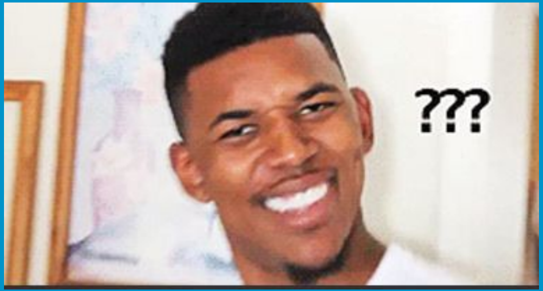
If somewhere in our code we need to do more advanced math operations like getting the square root of something or sin, cos, and tan, we can call on the Math class to do those operations for us.



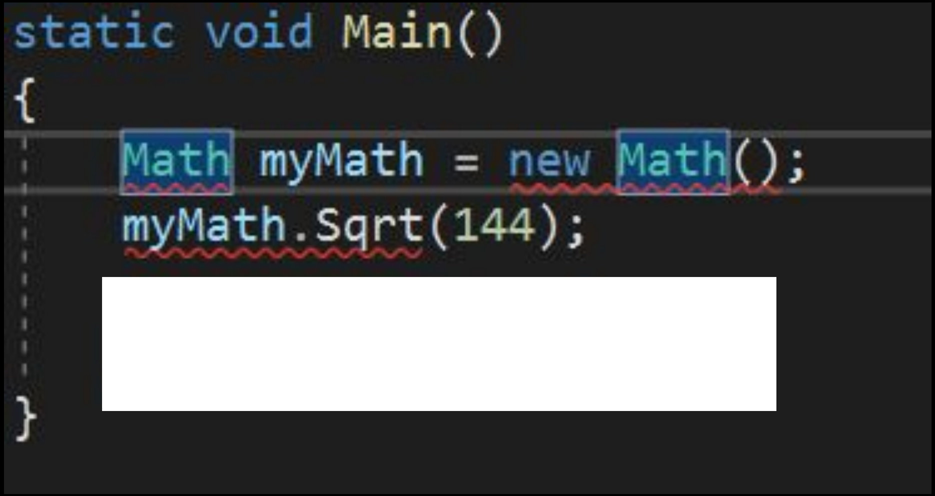
```
static void Main()
{
    [redacted]

    Math.Sqrt(144);
}
```

When to make it static



From an efficiency standpoint; it doesn't make sense to create a new instance of the Math class every time

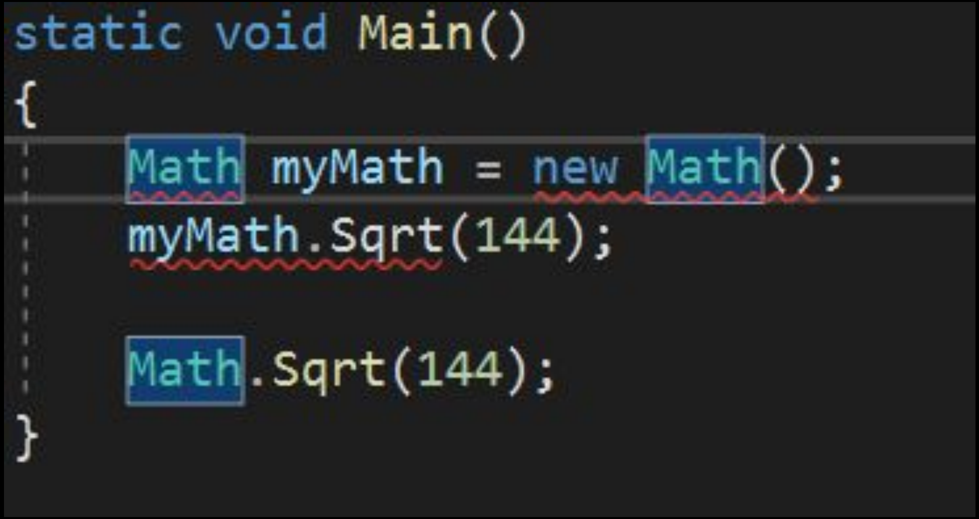


```
static void Main()
{
    Math myMath = new Math();
    myMath.Sqrt(144);


}
```

It’s cleaner!

Since there is only one copy of the Math class, we aren’t creating a bunch of unnecessary objects all throughout ou
code anytime we want to use those methods. Obviously, we have red squiggles under the top example and that is because we tried to create an instance of a static class that we know isn’t allowed.



```
static void Main()
{
    Math myMath = new Math();
    myMath.Sqrt(144);

    Math.Sqrt(144);
}
```

Static

Static: `Console.WriteLine();`

Instead of:

```
Console c = new Console();

c.WriteLine();
```

Syntax

Static:

```
ClassName.MethodName();
```

Instance:

```
ClassName instance = new ClassName();

instance.MethodName();
```

STATIC METHOD

```
ClassName.MethodName();
```

INSTANCE METHOD

```
var instance = new ClassName();
instance.MethodName();
```

Instance

Static