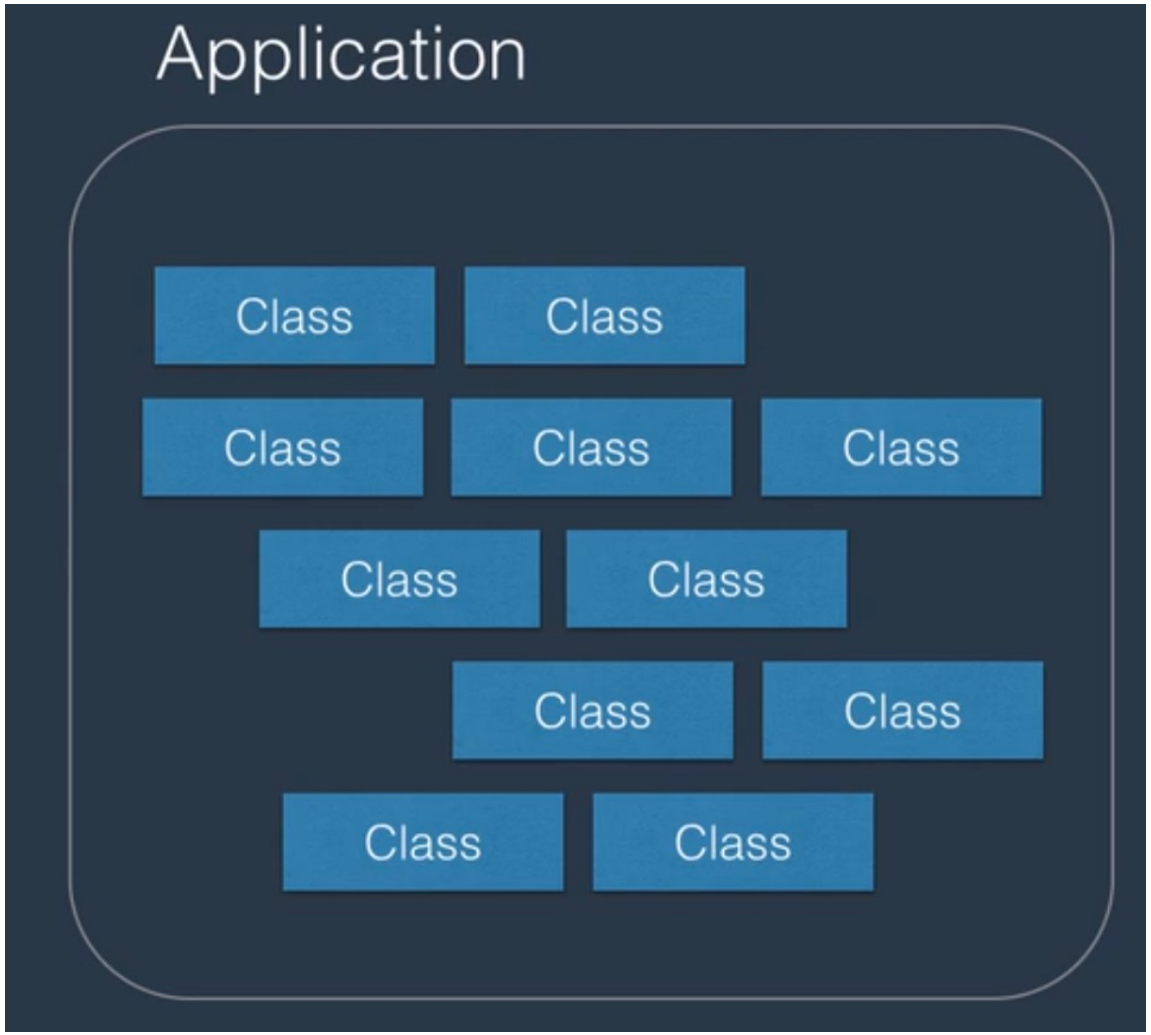
Classes

- What is a class
- Real-world example of classes
- What is an object
- Static members

Class – A building block of an appliation



A real world application often consist of hundreds or even thousands of classes collaborating with each other.

As a C-sharp programmer it's absolutely crucial to understand classes and how to use them effectively to build a maintainable clean robust reliable application.

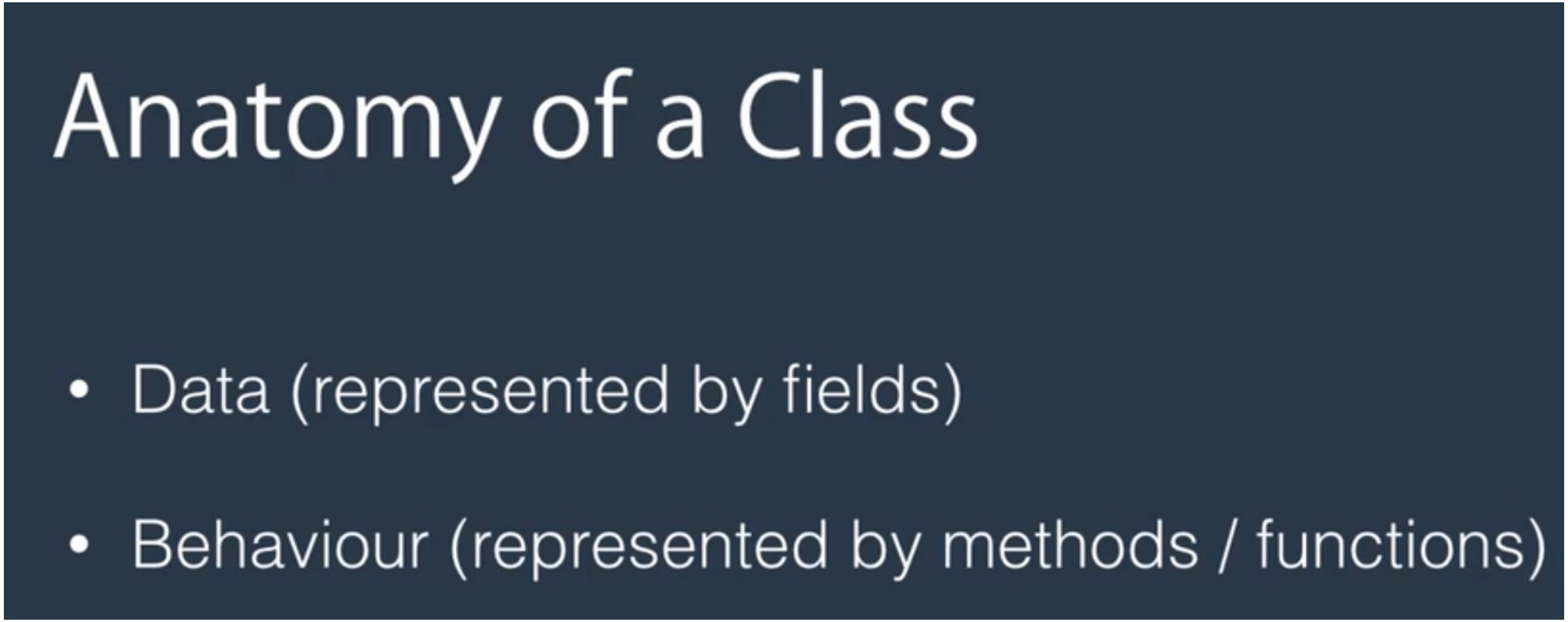
By the end of this course you're going to learn a lot of materials about how to use classes and interfaces to be loosely coupled extensible and maintainable applications.

We all are familiar with the concept of blogs and posts. Let's say we would like to implement a web application as a blog engine with a typical layered architecture.

Our application consists of at least three layers. Presentation business logic which is also referred to as domain layer and data access which is also referred to as persistence layer.

In each of these layers we can have classes each responsible for something particular. For example in the presentation layer we can have a class like post view which is purely responsible for displaying imposed to the user in the business logic or domain layer.

We can have the concept of post itself. And finally in the data access or persistence layer we can have a class like post repository which knows how to save opposed to the database or loading from a database.



It's a type that we developers define and give it a responsibility depending on the kind of application

Here is how we illustrate a class in UML or unified modeling language.

UML is a graphical language are representing classes under collaboration and we use that as a communication tool as part of building software.

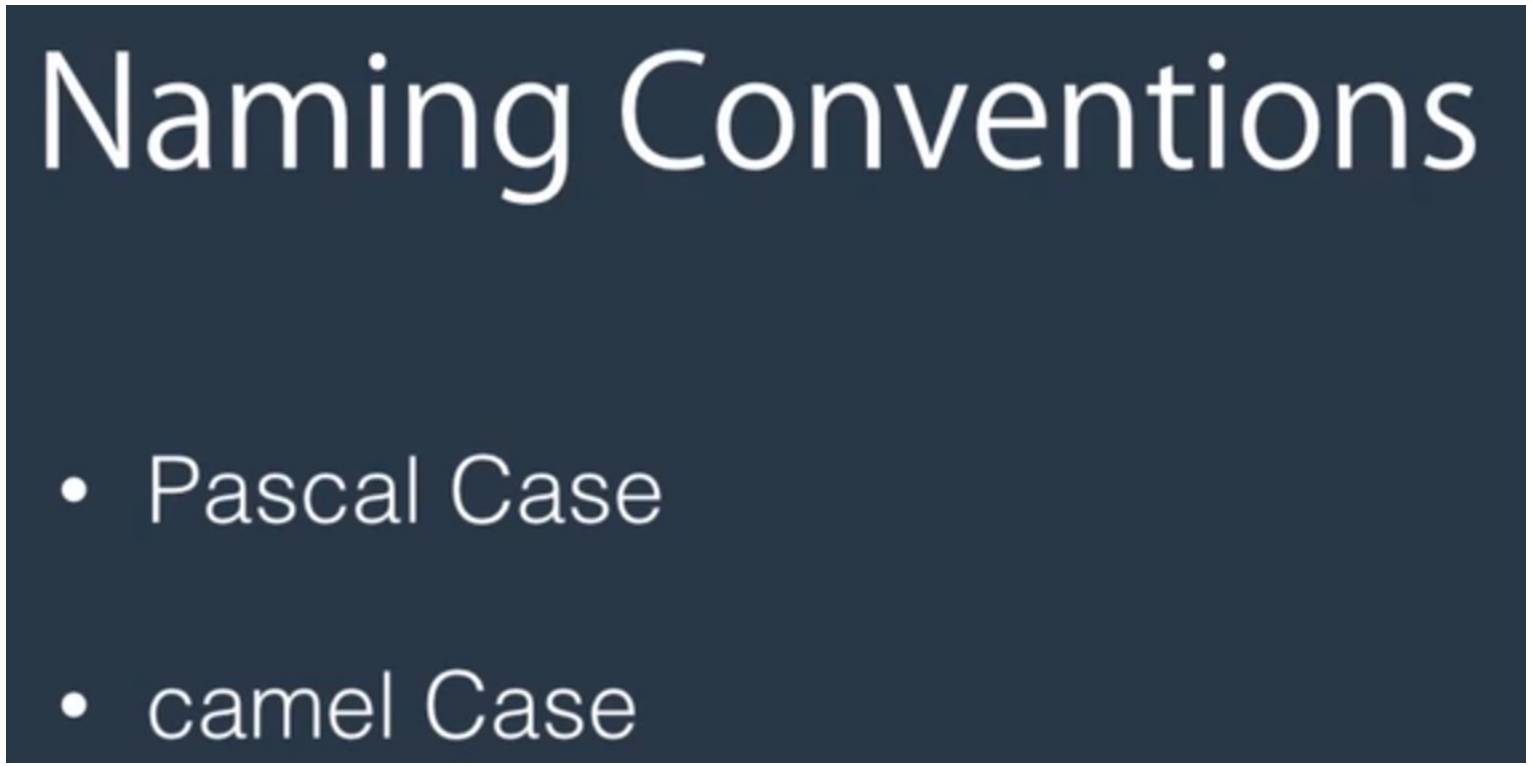
Its essentially an instance of a class that resides in the memory.

If person is a class we can have objects like John Mary and Scott. These are instances of that person. The person class or the person type defines the blueprint from which we can create objects at runtime.

These objects into memory talk to each other and as a result we get the behavior expected from the application.

Declaring Classes

```
public class Person
{
}
```



We use camel case when naming parameters of methods.

Declaring Classes

```
public class Person
{
    public string Name;
}
```

Declaring Classes

```
public class Person
{
    public string Name;

    public void Introduce()
    {
        Console.WriteLine("Hi, my name is " + Name);
    }
}
```

Creating Objects

```
Person person = new Person();

var person = new Person();
```

Using Objects

```
var person = new Person();
person.Name = "Mosh";
person.Introduce();
```

Class Members

- Instance: accessible from an object.

```
var person = new Person();
person.Introduce();
```

- Static: accessible from the class.

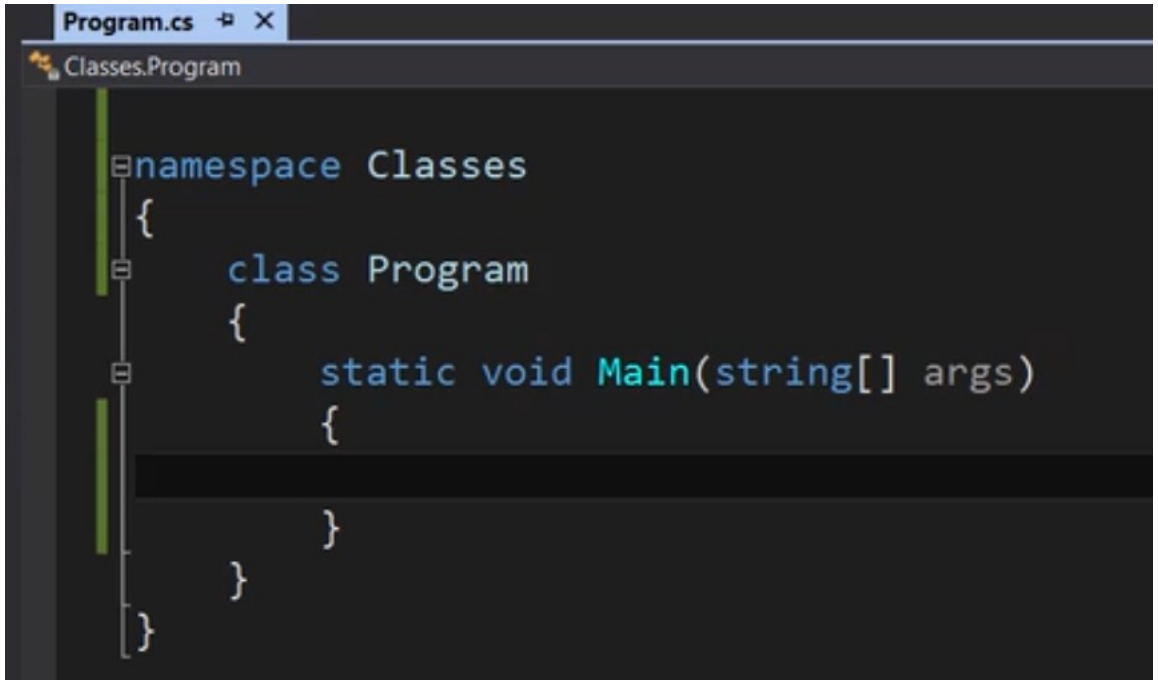
```
Console.WriteLine();
```

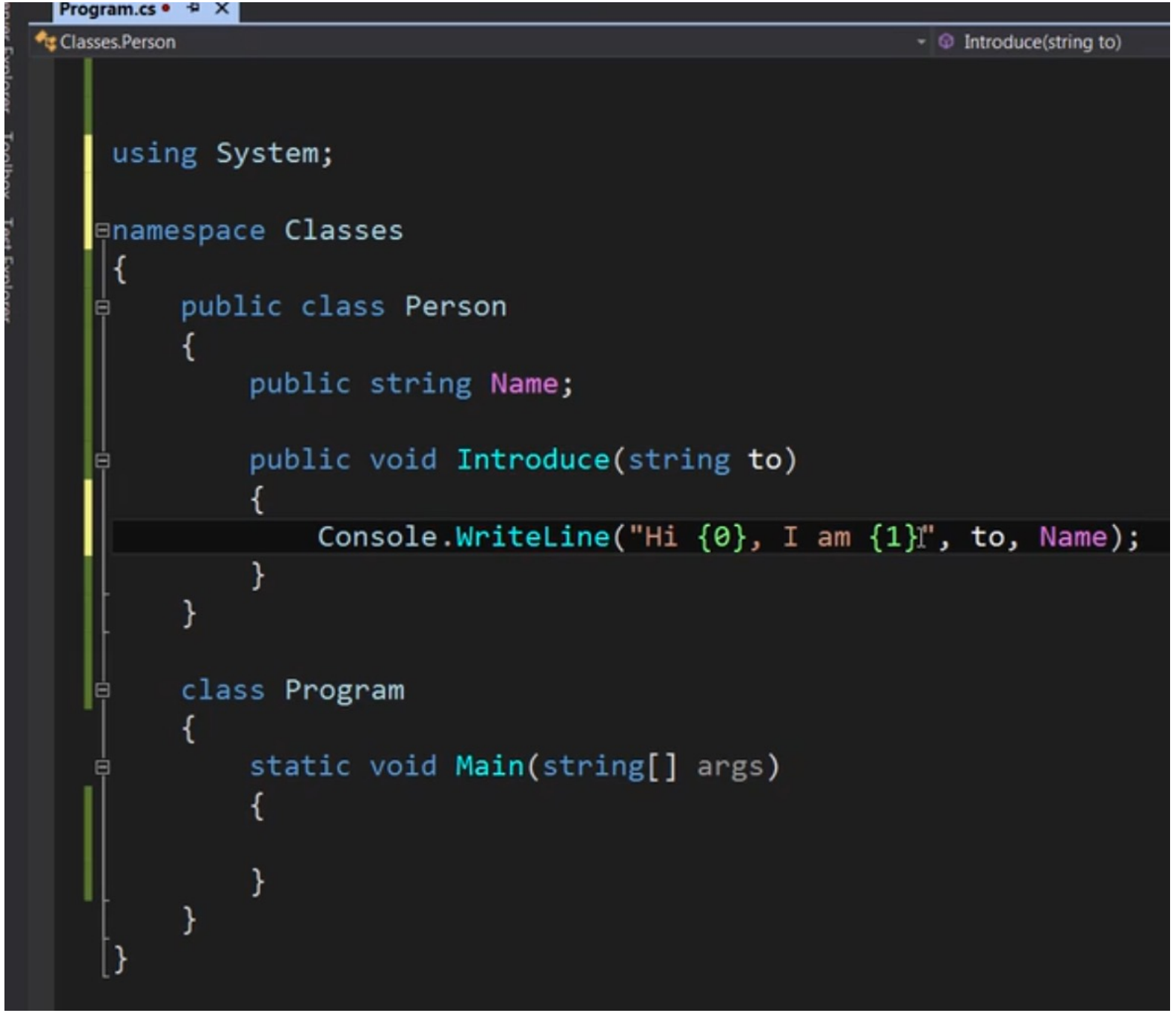
Why use static members?

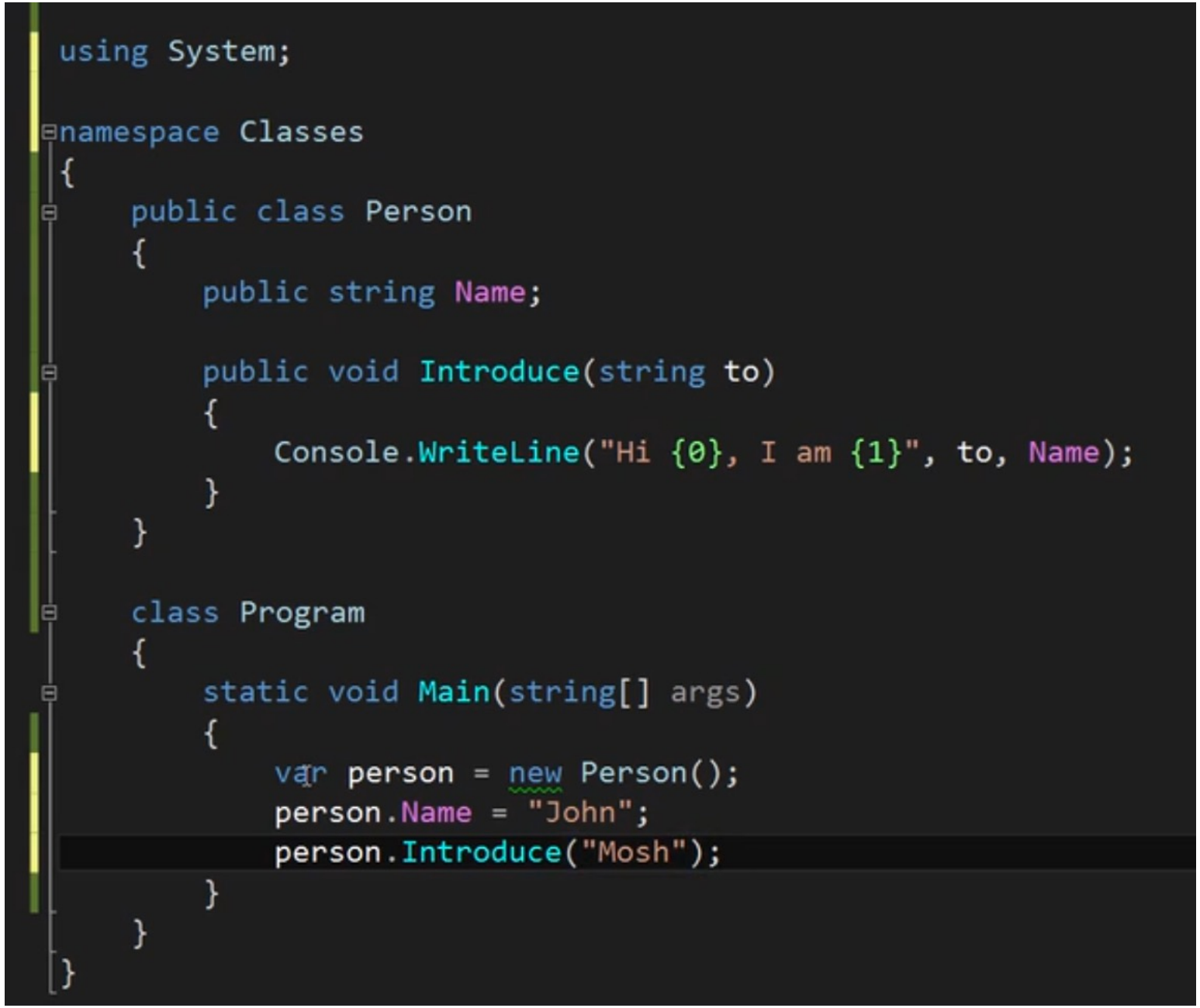
- To represent concepts that are singleton.
- DateTime.Now
- Console.WriteLine()

Declaring Static Members

```
public class Person
{
    public static int PeopleCount = 0;
}
```


Program.cs
Classes.Program
namespace Classes
{
class Program
{
static void Main(string[] args)
{
}
}
}


Program.cs
Classes.Person
Introduce(string to)
using System;
namespace Classes
{
public class Person
{
public string Name;
public void Introduce(string to)
{
Console.WriteLine("Hi {0}, I am {1}", to, Name);
}
}
class Program
{
static void Main(string[] args)
{
}
}
}


using System;
namespace Classes
{
public class Person
{
public string Name;
public void Introduce(string to)
{
Console.WriteLine("Hi {0}, I am {1}", to, Name);
}
}
class Program
{
static void Main(string[] args)
{
var person = new Person();
person.Name = "John";
person.Introduce("Mosh");
}
}
}