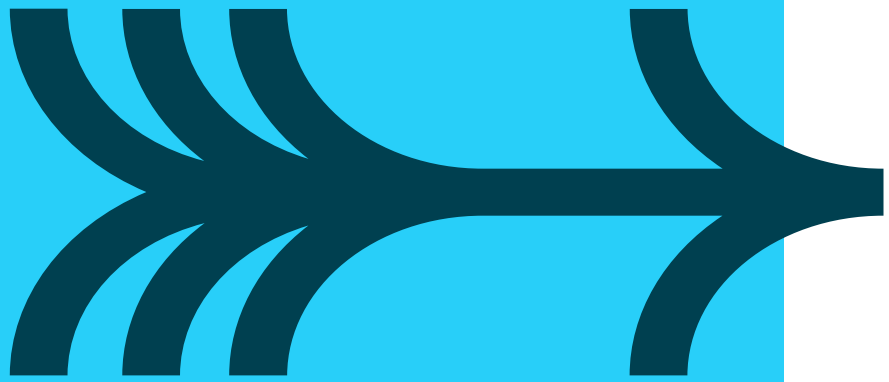
POWERING
POTENTIAL

Static fields and methods

CONTENTS



- **Objectives**
  - To understand what static types are

- **Contents**
  - `static` – what does it mean?
  - When to use a static field, property or method

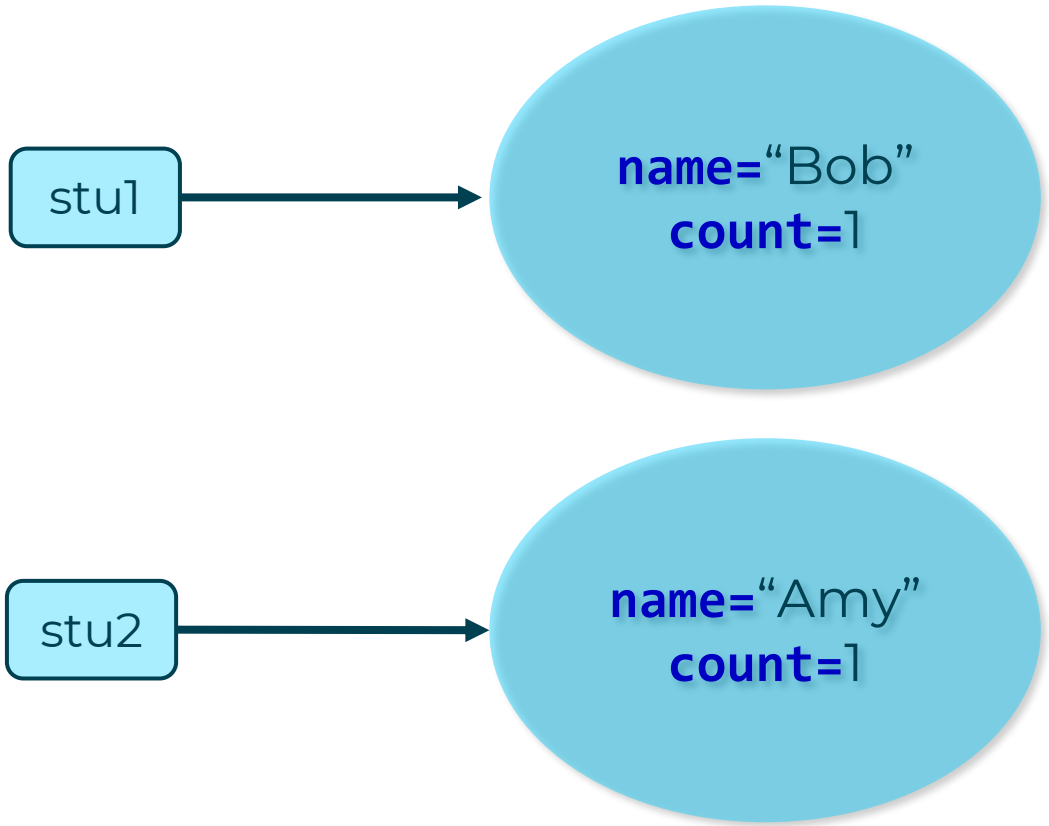
- **Hands-on labs**

Objects in memory

```
class Student
{
	public int count = 0;
	private string name;

	public Student(string name)
	{
		this.name = name;
		this.count++;
	}
}
```

```
Student stu1 = new Student("Bob");
Student stu2 = new Student("Amy");
```



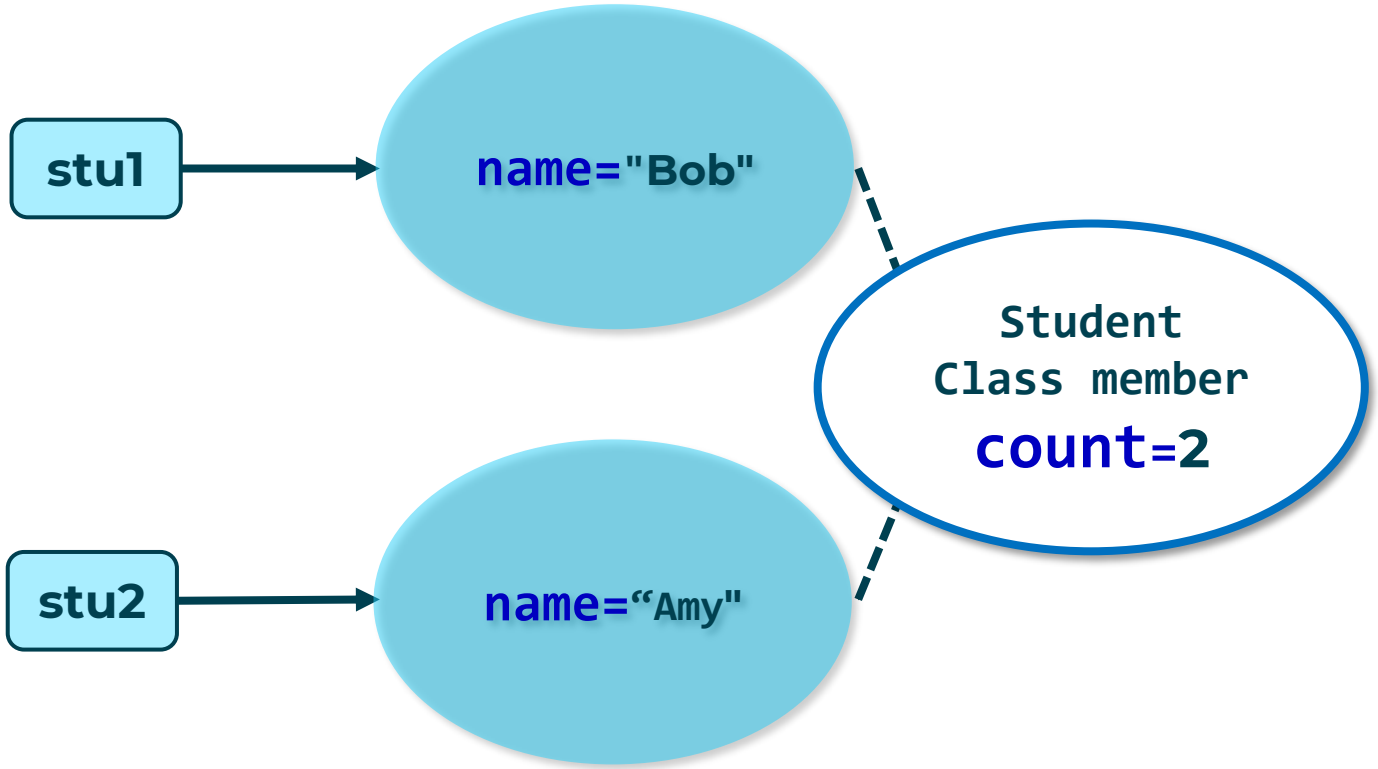
Static members

```
class Student
{
    public static int count = 0;
    private string name;

    public Student(string name)
    {
        this.name = name;
        count++;
    }
}
```

```
Student stu1 = new Student("Bob");
Student stu2 = new Student("Amy");
```

```
int total = Student.count;
```



Static – A few framework classes

Static means 'belongs to the class, not to an instance of the class'

Static members visible via the **class name**.
No need to create an instance

```
Console.WriteLine("Hi!");
```

```
Console.WriteLine(Math.PI);
```

```
System s = new System();

s.Console.WriteLine();
```

✘

```
Math m = new Math();

m.sqrt(25.6);
```

✘

this

- **`this` refers to the object on which method was invoked**
  - Could write Accelerate method as follows:

```
public class Car
{
   private int speed;

   public void Accelerate(int weightOfFoot)
   {
      this.speed += weightOfFoot;

   }
   ...
}
```

Using static - Simple factory pattern

Factory class

```
public class Bank
{
    public static Account CreateAccount(string owner)
    {
        // possible security code
        return new Account(owner);
    }
}
```

```
internal class Account {
    public Account() {
    }
}
```

Constructor is only visible from within the same library

```
public static void Main(string[] args)
{
    Account acc = new Account("Bob");

    Account acc = Bank.CreateAccount("Bob");
}
```

Client code using the Bank account factory

More on static Members

- **Can access other static members**
- **Cannot access instance member**
  - Can only access by using an object reference
- **Instance methods can call static methods and use static fields**

```
public class Car
{
  private string model, owner;

  public Car(string model)
  {
     this.model =  model;
  }

  public static Car MakeBMW()
  {
    Car car = new Car("BMW");
    car.SetOwner(“Tom“);
    return car;
  }

  public void SetOwner(string owner)
  {
    this.owner = owner;
  }
   ...
}
```

C# static initialiser block(s)

- **You can create a static constructor with no parameter or access modifiers**
  - Invoked on first access to any of the type's members
  - Runs before any methods are invoked or objects created
  - Used to do non-trivial initialisation of static fields

```
public class Car
{
  static licenseAgency DVLA;

  static Car()
  {
    if(...) DVLA = new LicenseAgency( ... );
  }

  public static void SetLicenseAgency (LicenseAgency dvla)
  {
    DVLA = dvla;
  }
}
```

Creating read-only values by property procedures

- `readonly` variables can be assigned to in declaration or in a constructor
  - Then they are read-only – can be instance or static members

```
public class Car
{
  public readonly int yearOfCreation;

  public Car()
  {
     yearOfCreation = DateTime.Now.Year;
  }

  public ŵộîđ ChangeYearOfCreation()
  {
     yearOfCreation++;   ✘
  }
}
```

Constructors can change sêắđộŋľỳ vars

Creating read-only values using properties

```
public class Car
{
   public int yearOfCreation   ĝêţ  řsîŵắţê şêţ   ;

   public Car()
   {
      yearOfCreation = DateTime.Now.Year;
   }

   public ŵộîđ ChangeYearOfCreation()
   {
      yearOfCreation++; ✓
   }
}
```

Constructors can change the value

Or any method in the class

Creating constant values

```
public class Car
{
   public const double VAT = 0.2;

   public const double RATE = GetRate();


   public Car()
   {
      VAT = 0.15;
   }

    ...
}
```

Value can only be set during declaration

Has to be a constant value

Not even a constructor can change the value

Type members are either:

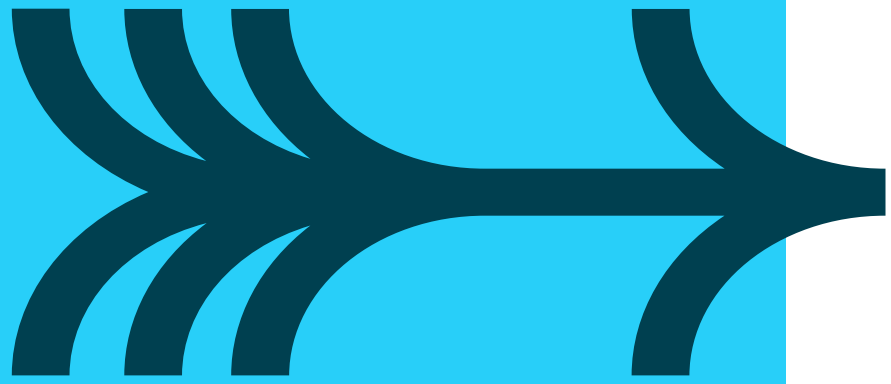
- static (belong to type itself)
  - Always accessed via (Uppercase) **T**ype name
  - No instance needed to use them

```
Math.Pow(2,3);
```

- instance (belong to an instance of the type)
  - Typically accessed via a (lowercase) **r**eference

```
myCar.Accelerate(10);
```

LAB



Use static members

We need a Vehicle Count

Each Vehicle needs a Registration Plate

Create a Registration plate Factory

to provide the next plate

Duration 1.5 hours

Part 1 1 hours
Part 2 30 minutes