KEEPING IT CLASSY WITH C#

Level 2

Classes

Expanding Our Bands

Our Band needs to store their musicians to be able to announce them.

What our application will do:

- Store information about a band and its musicians
- Announce the band
- Announce the musicians

Band

```
Name:  "The C Sharps"
Musicians: 4
```

In this level:

- Create our Band class
- Share methods between Band and Program

What Makes up a Band

Our band will have several properties including a Name and Musicians.

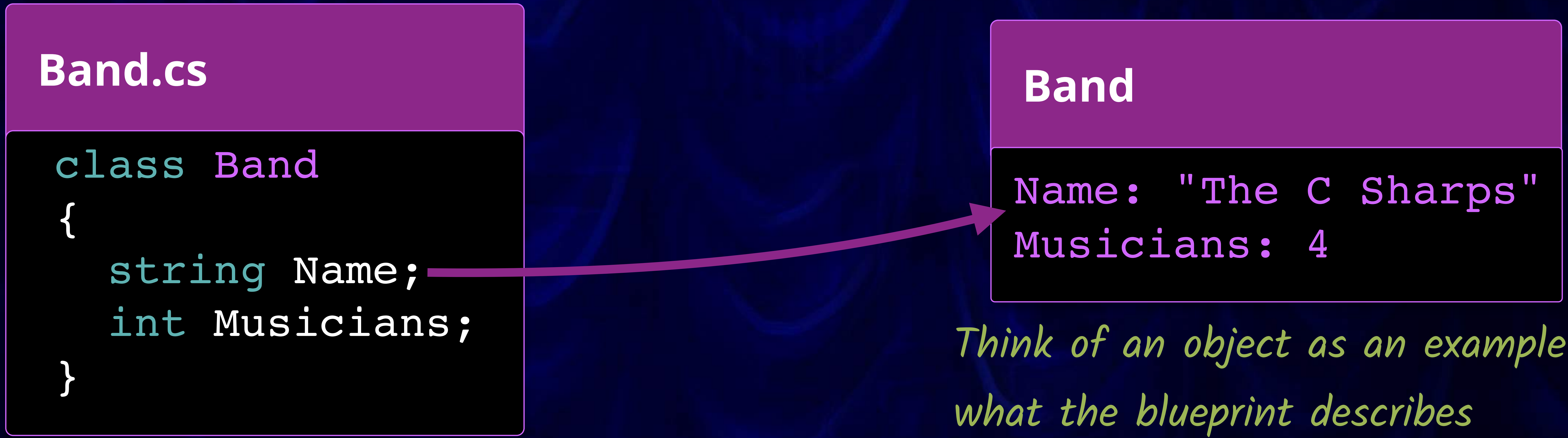
Band

```
Name: "The C Sharps"
Musicians: 4
```

But how do we create this in code?

Classes Define Our Objects

The Class defines the structure of our Band, while an object is an actual instance of a Band.



The Band Class

Band.cs

```
class Band
{
  string Name;
  int Musicians;
}
```

Think of a class like a blueprint

Instance of the Band Class (object)

Band

```
Name: "The C Sharps"
Musicians: 4
```

Think of an object as an example of what the blueprint describes

Declaring Our Band Class

To declare a class, we use the class keyword followed by the name we want for our class.

Band.cs

```
class Band
{

}
```

File names typically match whatever class is contained in that file

Band (End Goal)

```
Name: "The C Sharps"
Musicians: 4
```

Instance Variables

Instance Variables define the information we store in each instance of our class.

Band.cs

```
class Band
{
    string Name;
    int Musicians;
}
```

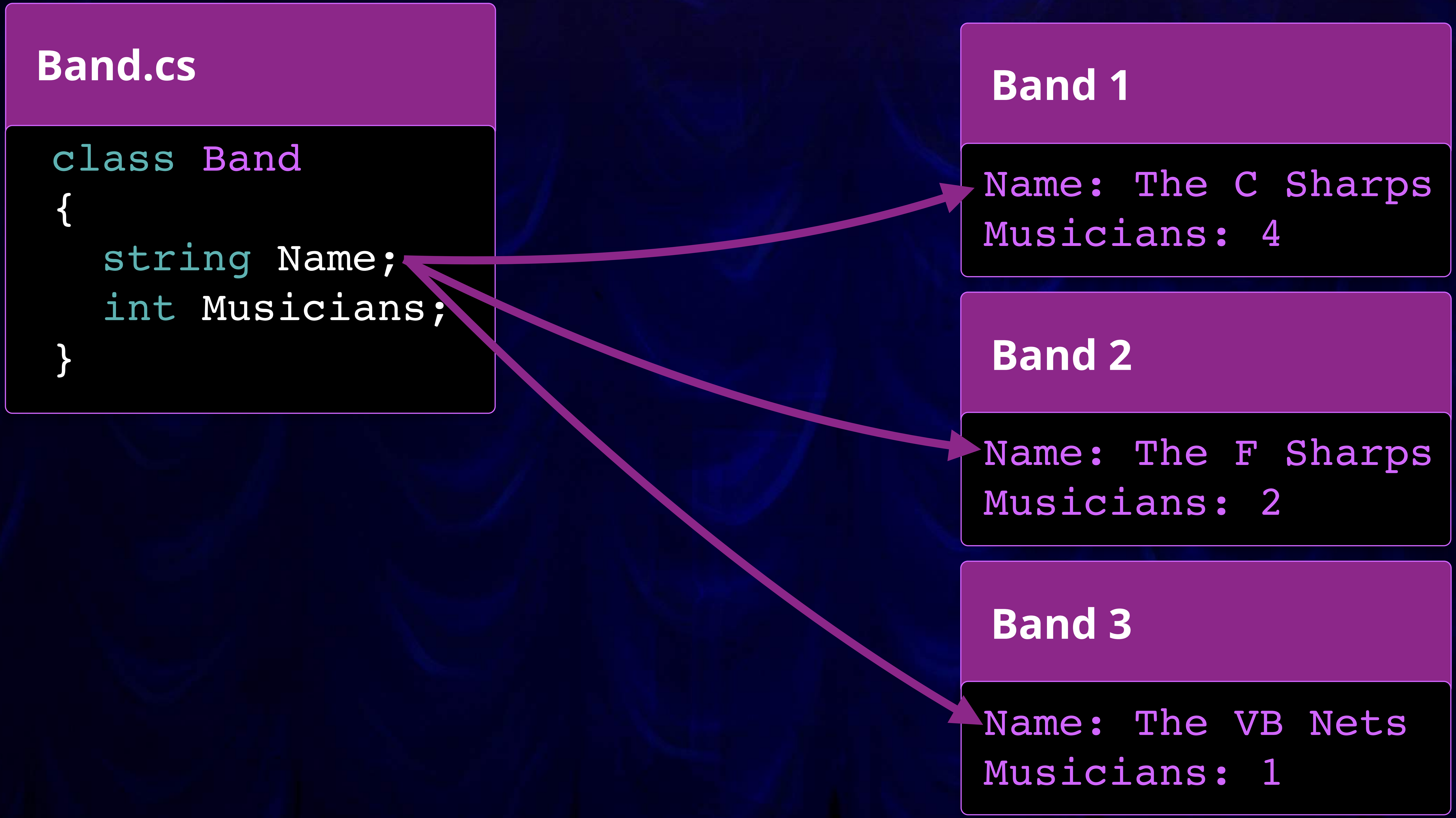
Band (End Goal)

```
Name: "The C Sharps"
Musicians: 4
```

*Okay, so where exactly does **Object** come into play with a* `Class`

Instances of an Object

We can have several instances of our Band class, each with their own values.



Band.cs

```
class Band
{
  string Name;
  int Musicians;
}
```

Band 1

```
Name: The C Sharps
Musicians: 4
```

Band 2

```
Name: The F Sharps
Musicians: 2
```

Band 3

```
Name: The VB Nets
Musicians: 1
```

Refactor AnnounceBand **Method**

Our AnnounceBand **Method should live in our** Band **class.**

Band.cs

```
class Band
{
   string Name;
   int Musicians;
}
```

Currently `AnnounceBand` *lives in our* Program *class even though it is designed to announce our* `Band`*, so we should move it into our* `Band` *class*

Program.cs

```
void AnnounceBand(string bandName)
{
   Console.WriteLine("Welcome " + bandName + " to the stage!");
}
```

KEEPING IT CLASSY WITH C#

Refactor AnnounceBand Method

Our AnnounceBand **Method should live in our** Band **class.**

Band.cs

```
class Band
{
  string Name;                    Now AnnounceBand lives in our Band class
  int Musicians;

  void AnnounceBand(string bandName)
  {
    Console.WriteLine("Welcome " + bandName + " to the stage!");
  }
}
```

Some Issues With AnnounceBand

Our AnnounceBand **Method has a few smells we need to clean up.**

Band.cs

```
class Band
{
   string Name;
   int Musicians;

   void AnnounceBand(string bandName)
   {
      Console.WriteLine("Welcome " + bandName + " to the stage!");
   }
}
```

The name AnnounceBand is redundant

We can now use our Name variable instead of using a parameter

KEEPING IT CLASSY WITH C#

Refactor AnnounceBand Method

Renaming the method and using the Name variable Announce better fits in the Band class.

Band.cs

```
class Band
{
    string Name;
    int Musicians;

    void Announce()
    {
        Console.WriteLine("Welcome " + Name + " to the stage!");
    }
}
```

Rename AnnounceBand *to just* Announce

Remove the parameter and use our Name *variable instead*

What happens when we attempt to compile this?

Reference Error

We're getting an error because Console isn't part of our Band class.

Band.cs

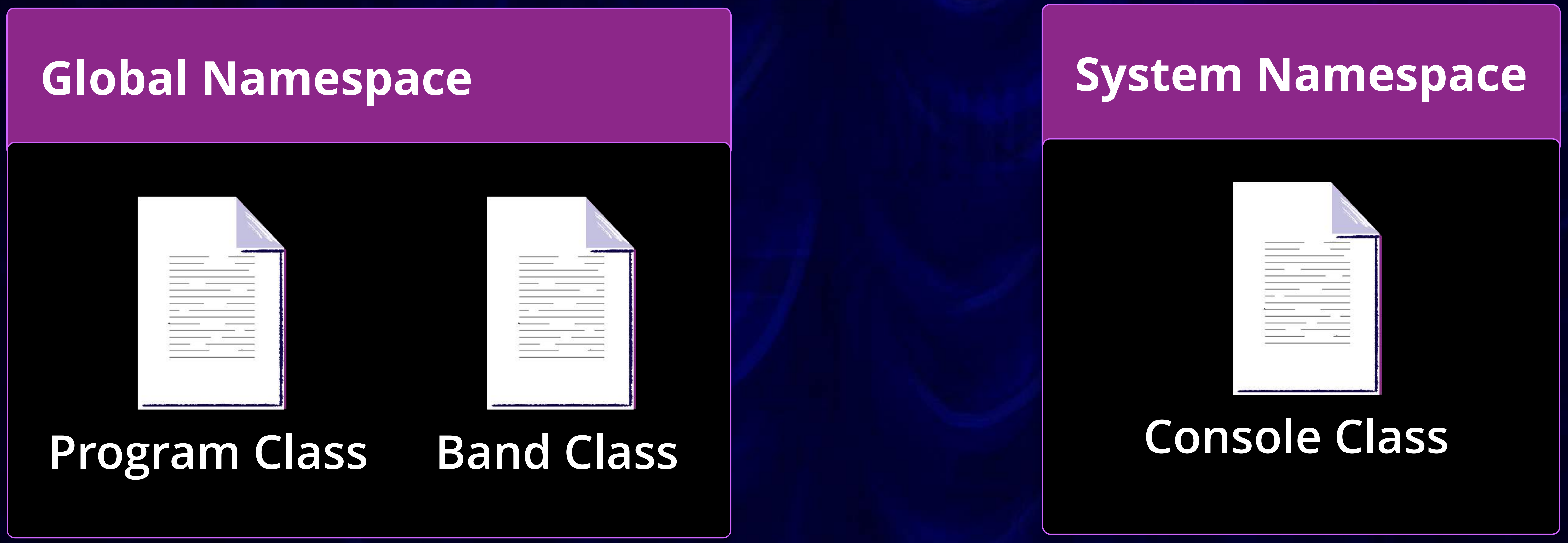
```
class Band
{
   string Name;
   int Musicians;

   void Announce()
   {
     Console.WriteLine("Welcome " + Name + " to the stage!");
   }
}
```

Console doesn't exist within Band so our compiler doesn't know what to do here

```
ERROR: The name 'Console' doesn't exist in the current context
```

Namespaces

Namespaces are used to organize classes further allowing reuse of class names.



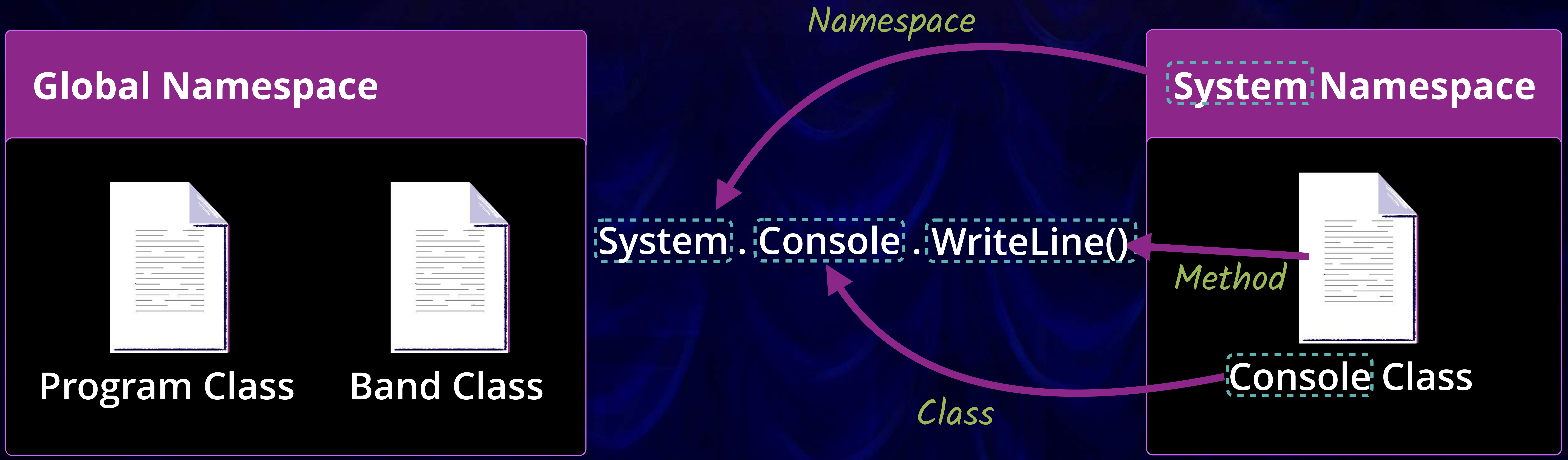
*All of our classes live in a **Global** Namespace available throughout the application*

***Console** lives in the built in **System** namespace*

KEEPING IT CLASSY WITH C#

Accessing Outside Namespaces

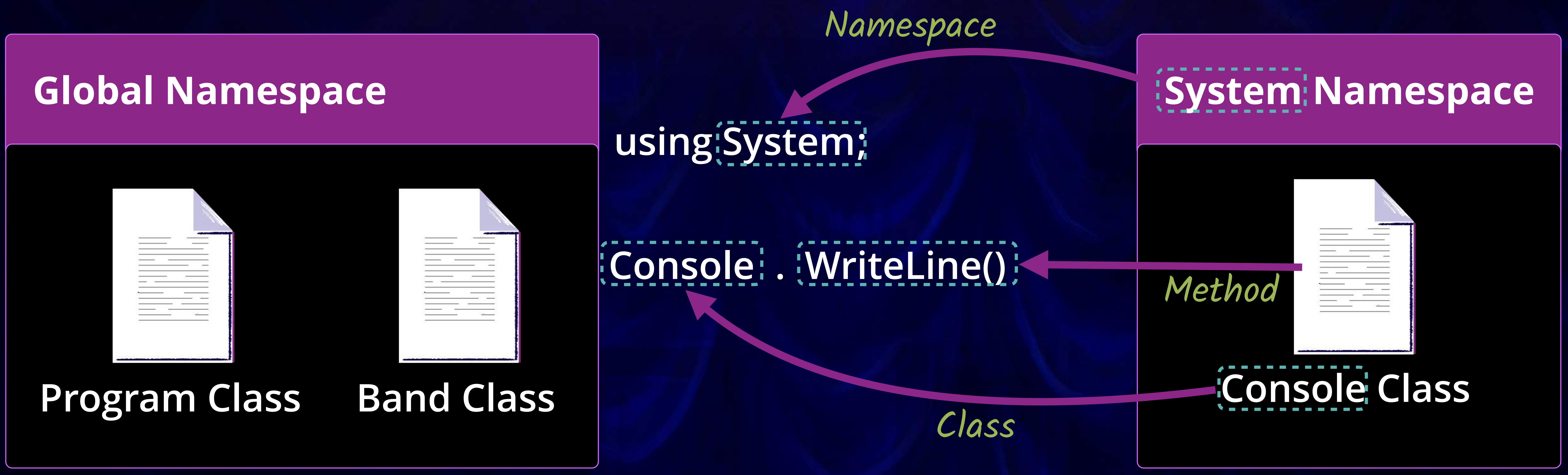
Namespaces can be accessed using the Namespace.Class.Method **format.**



Rewriting that every time can get annoying is there a shorter way of doing this?

Using Directives

You can reference a namespace throughout a file with a using directive.



We didn't get an error doing this in `Program.cs` *because it already had* `using System;` *in it*

This allows us to use our Console class throughout a file and only reference System once

KEEPING IT CLASSY WITH C#

Using Directive

Adding a using directive for System resolves our error.

Band.cs

```
using System;

class Band
{
  string Name;
  int Musicians;

  void Announce()
  {
    Console.WriteLine("Welcome " + Name + " to the stage!");
  }
}
```

This gives Band access to our System Namespace, letting our Compiler know where to find Console

Now how do we create a Band object using our Band class?

KEEPING IT CLASSY WITH C#

Create an Instance of Our Band Object

We can use new Band() **to instantiate a new** Band **object.**

Program.cs

```
…
static void Main(string[] args)
{
   Console.WriteLine("What is the name of your band?");
   Band band = new Band();
   band.Name = Console.ReadLine();
}
…
```

new Band() *will create our new band object*

Change Console.ReadLine *to set our band's* Name *variable*

Band **Is Inaccessible**

An error is being thrown, saying the Band class is not accessible.

Program.cs

```
…
static void Main(string[] args)
{
   Console.WriteLine("What is the name of your band?");
   Band band = new Band();
   band.Name = Console.ReadLine();
}
…
```

Band is throwing an error because we've not made it accessible to other classes

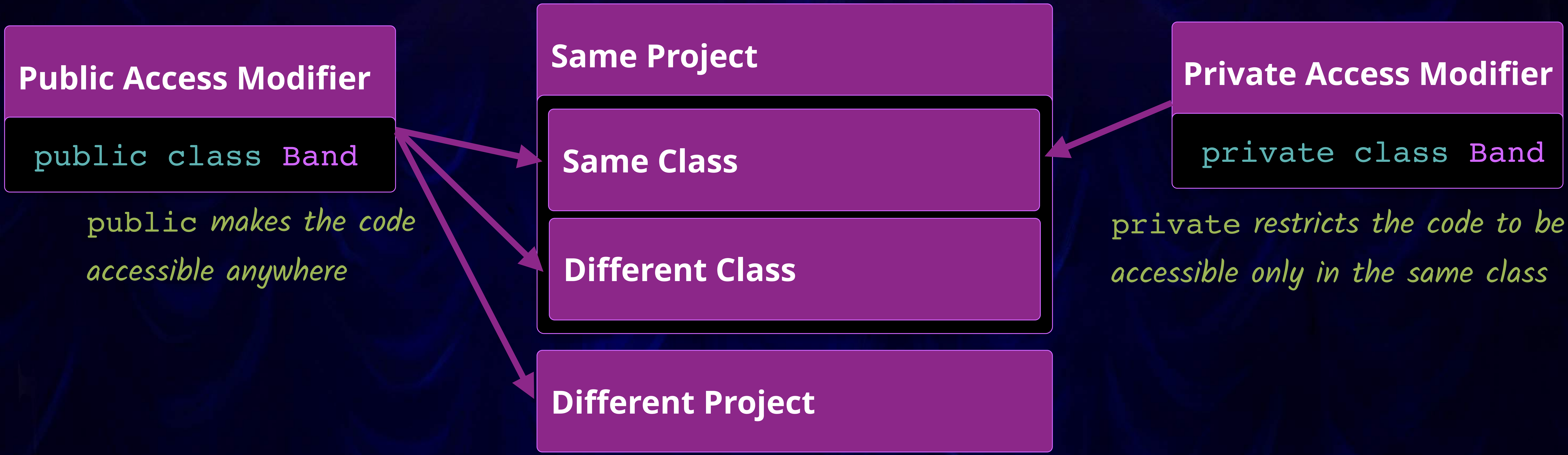
ERROR: 'Band' is inaccessible due to protection level

What is a protection level and why is it making Band inaccessible?

KEEPING IT CLASSY WITH C#

Understanding Protection Levels

Protection Levels are set by Access Modifiers **and** determine where classes, methods, etc can be accessed.



```
public class Band
```

```
private class Band
```

`public` *makes the code accessible anywhere*

`private` *restricts the code to be accessible only in the same class*

*Access Modifiers **default to** private **anytime one isn't provided***

KEEPING IT CLASSY WITH C#

The Public Access Modifier

Adding public to classes, methods, variables, etc makes them visible to other classes.

Band.cs

```
using System;
…
public class Band
{
  public string Name;
  public int Musicians;

  public void Announce()
  {
    Console.WriteLine("Welcome " + Name + " to the stage!");
  }
}
```

This will make Band *accessible from our* Program *class*

Now we can finish calling our Announce *method in our* Program *class's* Main *Method*

KEEPING IT CLASSY WITH C#

Call The Announce Method

band.Announce() **will run the Announce method and use the band's Name in the announcement.**

Program.cs

```
…
static void Main(string[] args)
{
  Console.WriteLine("What is the name of your band?");
  Band band = new Band();
  band.Name = Console.ReadLine();
  band.Announce();
}
…
```

>>>

```
What is the name of your band?
$ The C Sharps
Welcome The C Sharps to the stage!
```

KEEPING IT CLASSY WITH C#

A Quick Recap on Classes

Classes are used to further organize our code into collections.

- C# is an Object-Oriented programing language based around Classes
- Classes define what an object will look like
- An object is an actual instance of a class
- The default access-modifier is private, restricting code to only be used within the same class
- To use something from a different namespace you can use the full namespace.class.methodname or a using directive

KEEPING IT CLASSY WITH C#