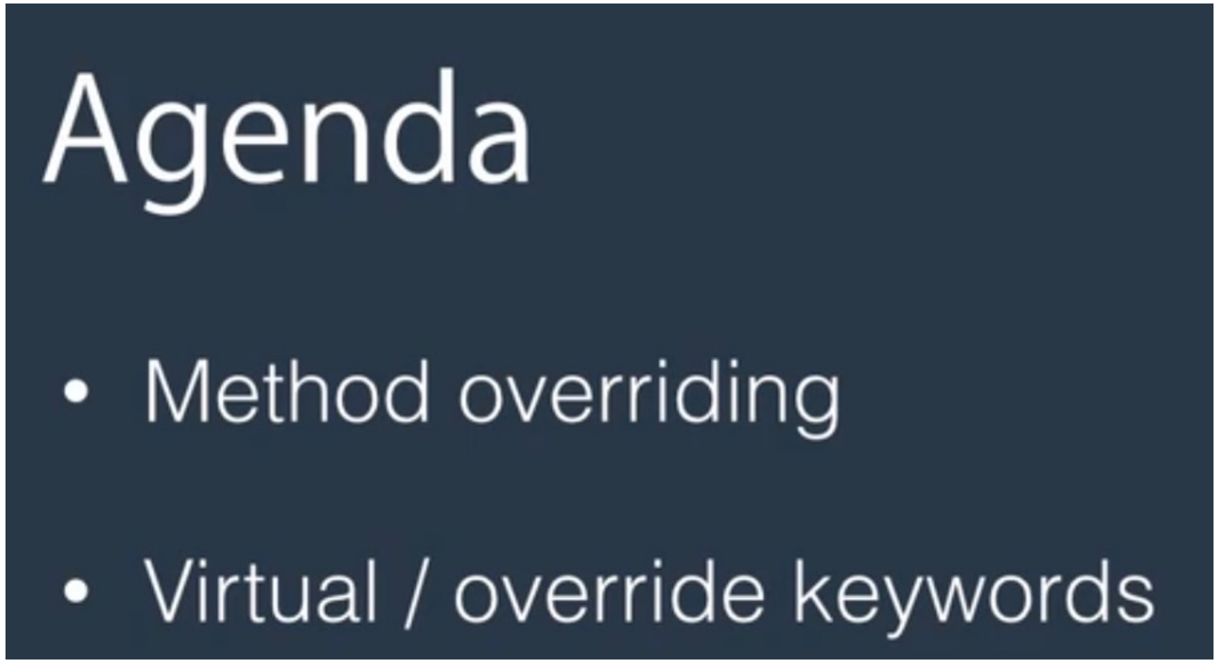
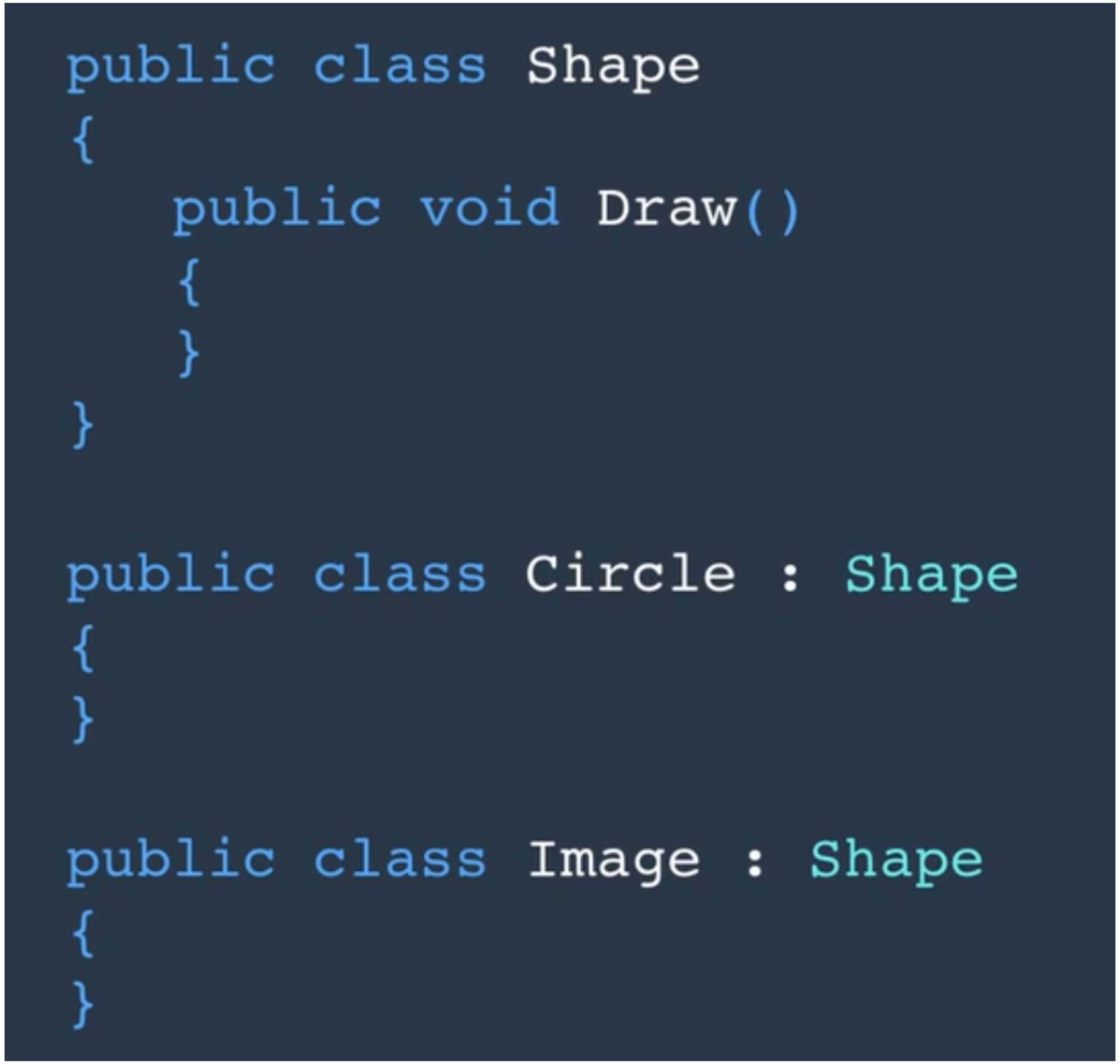


Agenda

- Method overriding
- Virtual / override keywords

Method Overriding

- Modifying the implementation of an inherited method.



```
public class Shape
{
    public void Draw()
    {
    }
}

public class Circle : Shape
{
}

public class Image : Shape
{
}
```

```
public class Shape
{
    public virtual void Draw()
    {
        // Default implementation
    }
}

public class Circle : Shape
{
    public override void Draw()
    {
        // New implementation
    }
}
```

C# INTERMEDIATE

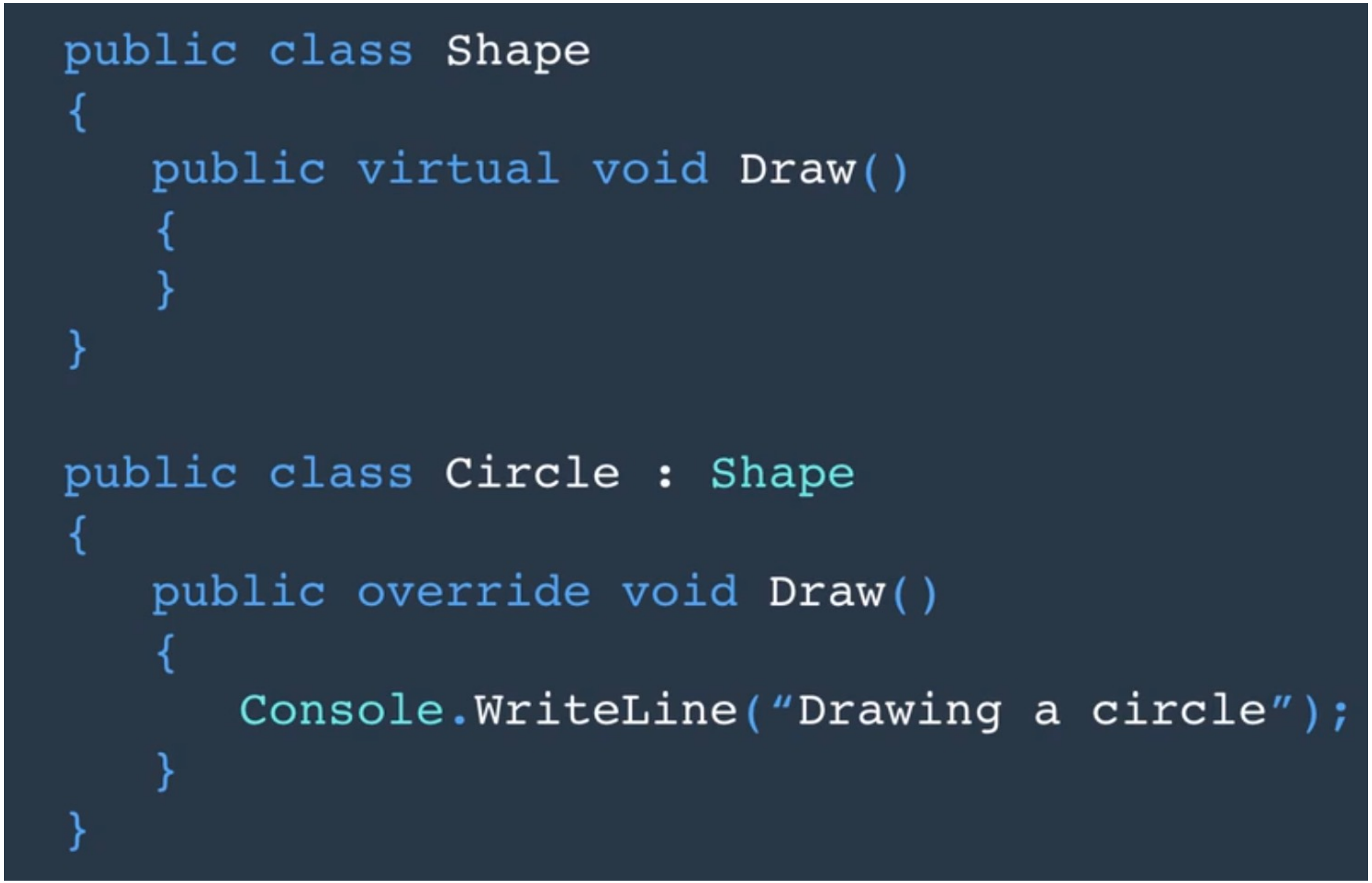
Abstract Classes and Members

Agenda

- Abstract modifier
- Rules about abstract classes and members

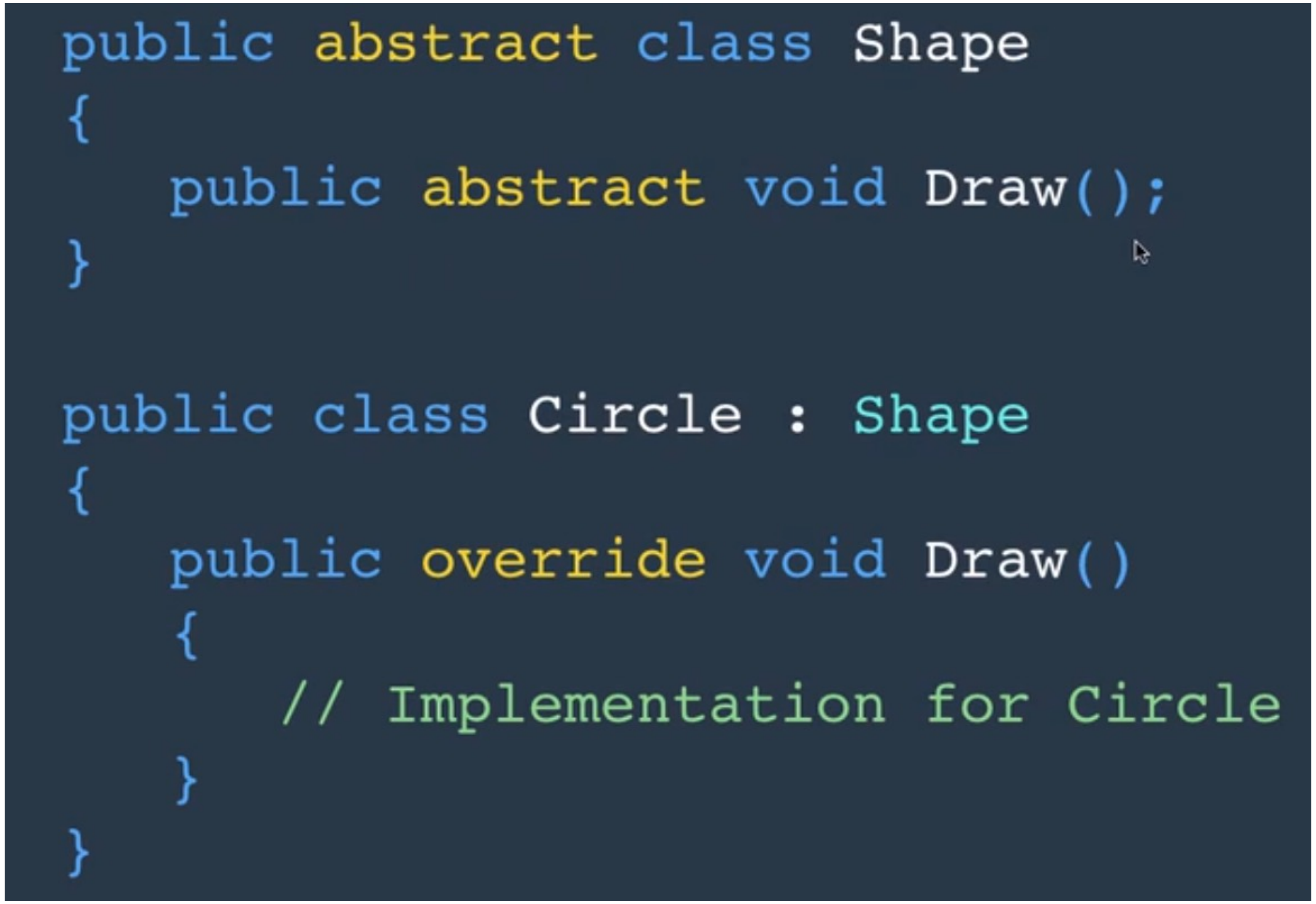
Abstract Modifier

- Indicates that a class or a member is missing implementation.



```
public class Shape
{
   public virtual void Draw()
   {
   }
}

public class Circle : Shape
{
   public override void Draw()
   {
      Console.WriteLine(“Drawing a circle”);
   }
}
```



```
public abstract class Shape
{
    public abstract void Draw();
}

public class Circle : Shape
{
    public override void Draw()
    {
        // Implementation for Circle
    }
}
```

Abstract Members

- Do not include implementation.

```
public abstract void Draw();
```

Abstract Members

- If a member is declared as abstract, the containing class needs to be declared as abstract too.

```
public abstract class Shape
{
    public abstract void Draw();
}
```

Derived Classes

- Must implement all abstract members in the base abstract class.

```
public class Circle : Shape
{
    public override void Draw()
    {
        // Implementation for Circle
    }
}
```

Abstract Classes

- Cannot be instantiated.

```
var shape = new Shape(); // Won't compile
```

Why to use Abstract?

- When you want to provide some common behaviour, while forcing other developers to follow your design.

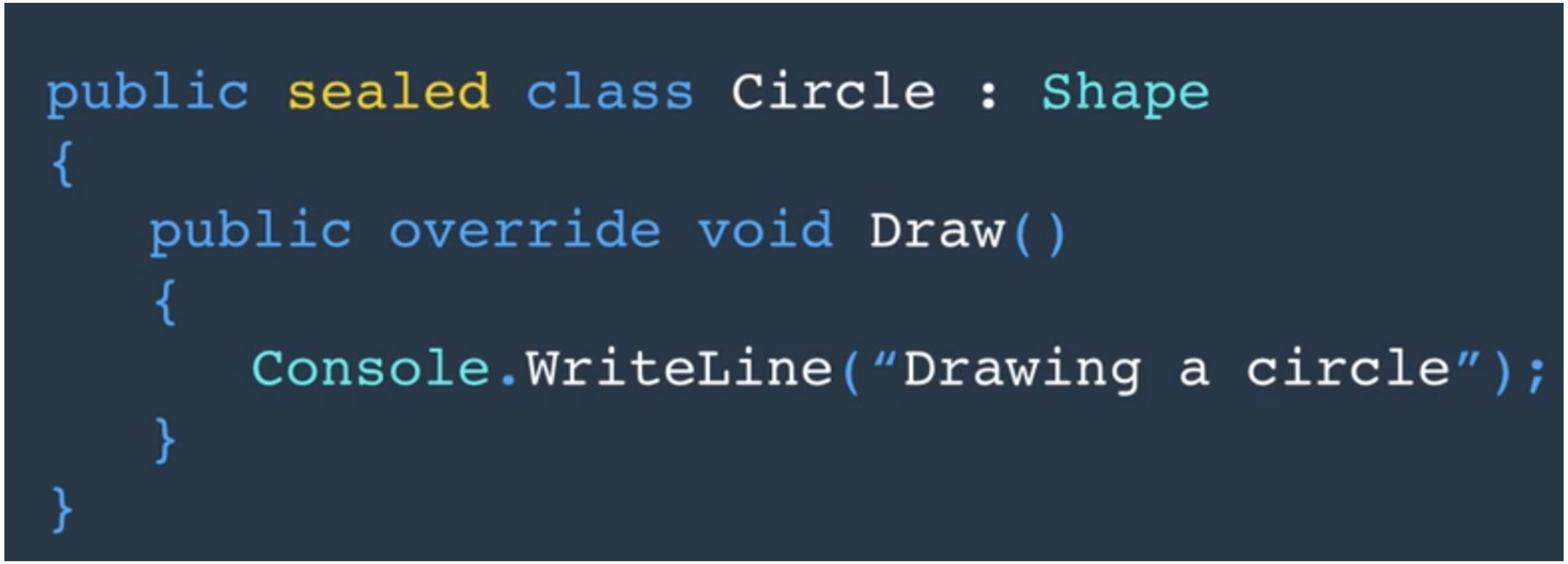
C# INTERMEDIATE

Sealed Classes and Members

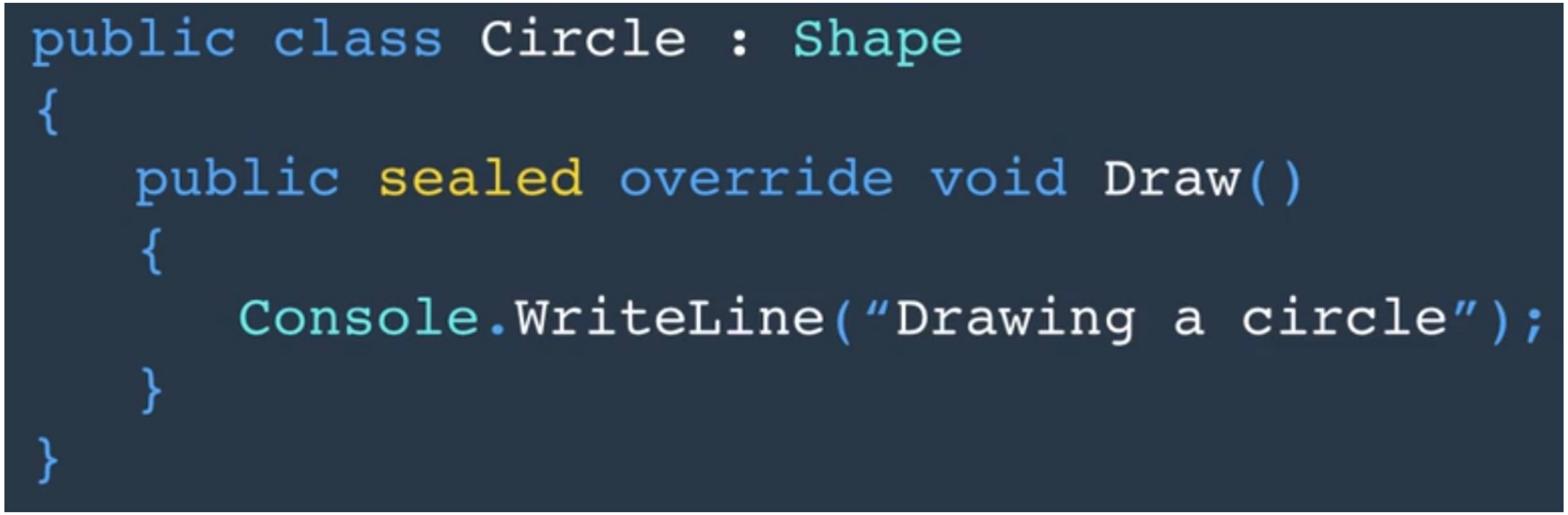
Sealed Modifier

- Prevents derivation of classes or overriding of methods.

```
public class Circle : Shape
{
    public override void Draw()
    {
        Console.WriteLine(“Drawing a circle”);
    }
}
```



```
public sealed class Circle : Shape
{
    public override void Draw()
    {
        Console.WriteLine(“Drawing a circle”);
    }
}
```



```
public class Circle : Shape
{
    public sealed override void Draw()
    {
        Console.WriteLine("Drawing a circle");
    }
}
```

Why?

- Sealed classes are slightly faster because of some run-time optimizations.

Why?

- Sealed classes are slightly faster because of some run-time optimizations.

Extension Methods