POWERING POTENTIAL

Inheritance

CONTENTS

- **Objectives**
  - To add functionality to existing classes using inheritance
- **Contents**
  - Basic concepts of inheritance
  - Extending a simple class
- **Hands-on labs**

Base and derived classes

- **A class can inherit the features of another class**
  - The original class is the ‘super/base’ class
  - The new class is the ‘sub/derived’ class
- **The ‘sub’ class can:**
  - Utilise all the features of the super class
  - Override certain behaviour of the super class
  - Add new features
- **Inheritance is a fundamental object-oriented concept**

Existing code in the super class can be reused by the subclass

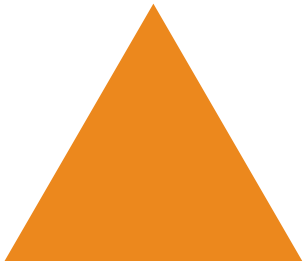
New classes can be defined simply in the terms of their differences from an existing class

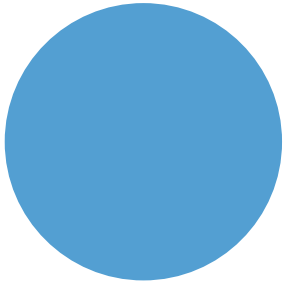
Inheritance in action

- **A vector graphics program**
  - Lots of commonality
    - `position` and `colour` fields
    - `draw` method
  - Want to benefit from re-use

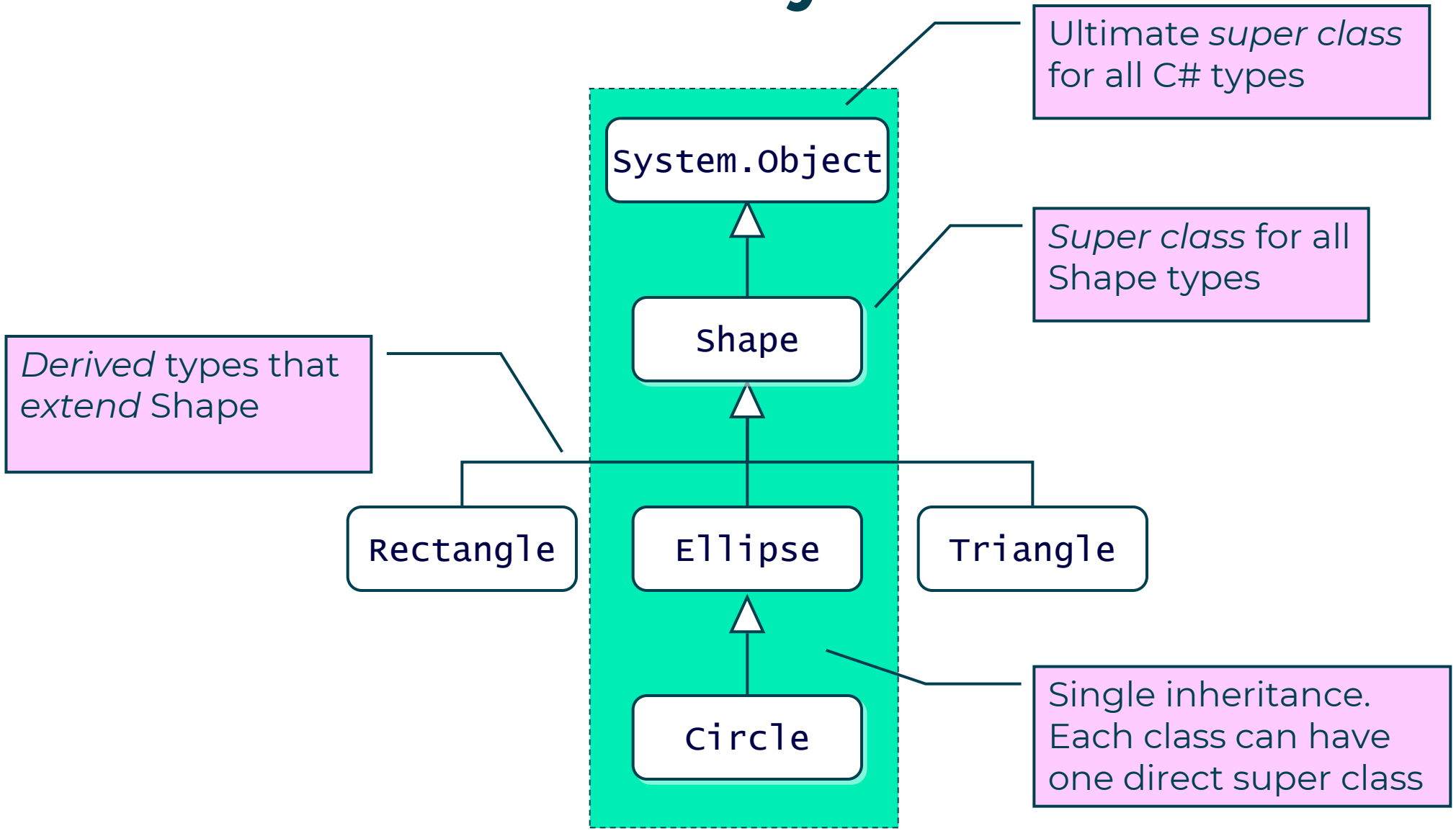
- **Create a base class called Shape**
  - Implement common code there

- **Derive classes from Shape**
  - `Rectangle, Ellipse, Triangle`





The inheritance hierarchy



Specifying the base class

Super class the sub class extends

```
public class Shape
{
  private int x, y;
  private string colour;
  ...
}
```

Sub classes extending the super class

```
public class Rectangle : Shape {
   ...
}

public class Ellipse   : Shape {
   ...
}

public class Circle    : Ellipse {
   ...
}
```

Rectangle specific members

Ellipse specific members

Circle specific members

Subclass inherits all the base class fields

Can be exposed via public properties

```
public class Shape
{
  private int x, y;
  private string colour;
   ...
}
```

```
public class Ellipse : Shape
{
  private int width;
  private int height;
   ...
}
```

Shape object

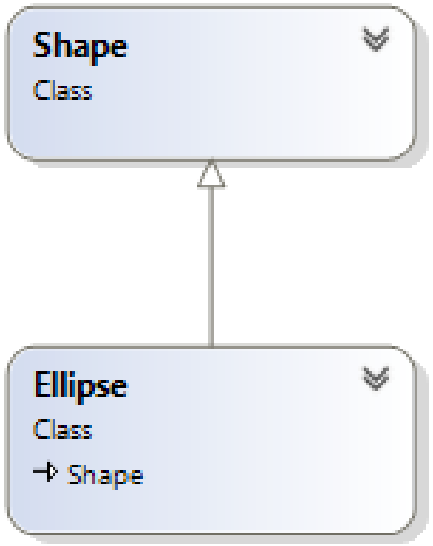
x & y:	10, 10
colour:	Green

Ellipse object

x & y:	10, 10
colour:	Green
width:	20
height:	10

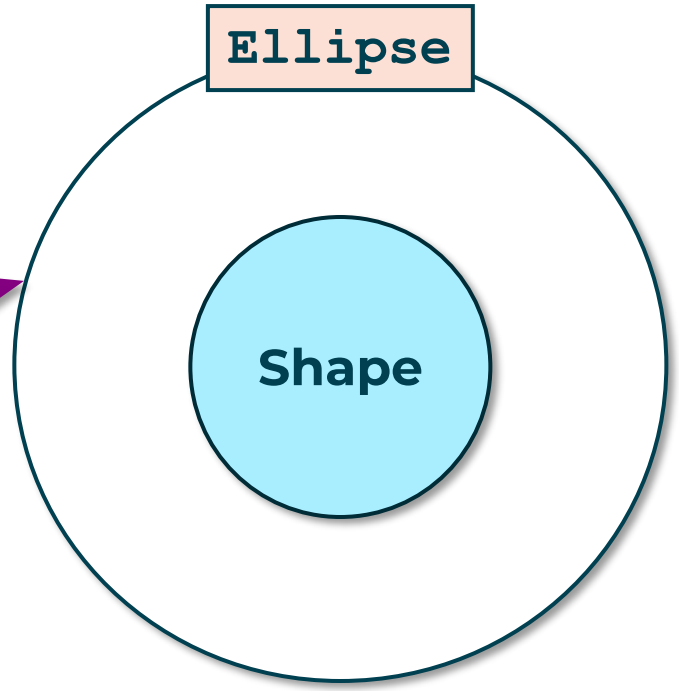
Constructing the derived objects

- **Base class constructors are not inherited**
  - But, default constructor of the base class is called
- **You can invoke the base class constructor**
  - **Mandatory** if there is no default (no argument) constructor in the base class



UML notation

Before constructing an Ellipse, should construct the Shape object within



Derived class constructor

```
class Shape
{
  private int x, y;
  private string colour;

  public Shape(int x, int y, string colour)
  {
     this.x = x;
     this.y = y;
     this.colour = colour;
  }
}
```

No default constructor
So all derived classes must invoke this constructor

```
class Ellipse : Shape
{
  private int width, height;

  public Ellipse(int x, int y, int width, int height, string colour) : base(x, y, colour)
  {
    this.width = width;
    this.height = height;
  }
}
```

Calling base constructor to initialise base fields

```
Ellipse e1 = new Ellipse(4,7, 23, 24, "RED");
```

Constructor chaining

```
class Ellipse : Shape
{
  private int width, height;

  public Ellipse(int x, int y, int width, int height, String colour) : base(x, y, colour)
  {
        this.width = width;
        this.height = height;
  }

  public Ellipse(int x, int y) : this(x, y, 10, 10, Color.BLUE)
  {
  }
}
```

Calling base constructor initialises base fields

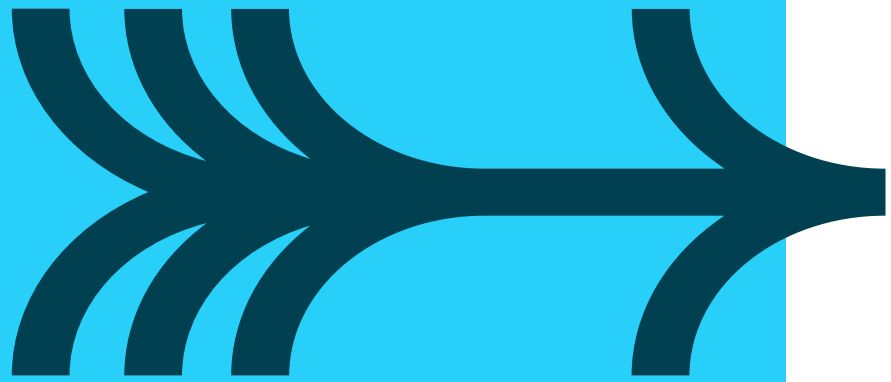
Calling another Ellipse constructor

```
Ellipse e1 = new Ellipse(4,7);
```

PROTECTED

- **Modifier that allows access to deriving types only**
  - Used to restrict access to methods
    - Remember, fields should always be private

- **Let's view an example...**



Protected access modifier

```
namespace barclays;

public class CreditCard
{
   protected int pin;

   public CreditCard(int pin)
   {
      this.pin = pin;
   }
}
```

```
namespace tesco;
using barclays;

public class Bank
{
    public static void Main(string[] args)
    {
         CreditCard card = new CreditCard(333);
         Console.WriteLine(cc1.pin);
    }
}
```

No access by a class outside of the project/assembly

```
namespace barclays;

public class Program
{
   public static void Main(string[] args)
   {
       CreditCard cc1 = new CreditCard(111);
       Console.WriteLine(cc1.pin);
   }
}
```

Can only be accessed by extended classes

Protected example with inheritance

```
namespace barclays;

public class CreditCard
{
   protected int pin;

   public CreditCard(int pin)
   {
      this.pin = pin;
   }
}
```

```
namespace tesco;
using barclays;

public class TescoCreditCard : CreditCard
{
   public TescoCreditCard(int pin) : base(pin)
   {
   }

   public void ChangePin(int newPin)
   {
         this.pin = newPin;
   }
}
```

Can be accessed by a class (in any project) which extends the base class

USING INHERITANCE FOR CREATING CUSTOM EXCEPTIONS

- **Custom exception class must derive from `Exception`**
  - Add your own constructors
    - Pass `'string Message'` up to base class
      (the only time you can write to the inherited Message field)
    - Can add additional methods

- **View code example on the next slide …**

A C# example

```
public class QAException : Exception
{
    public QAException(string message) : base(message)
    {
    }

    public QAException() : this("General error")
    {
    }
}
```

```
void MethodY()
{
    throw new QAException();
}
```

```
void MethodX()
{
    try
    {
        MethodY();
    }
    catch (QAException e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Review

- **Why do we do inheritance?**
  - Code reuse
  - Perhaps there will be other reasons soon!

- **Subclass inherits and can add additional functionality**

LAB

Working with inheritance

Duration 90 minutes