POWERING POTENTIAL

# Polymorphism

# CONTENTS

- **Objectives**
  - To understand and use polymorphism

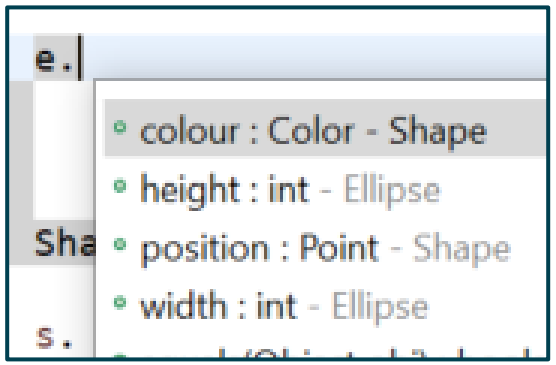
- **Contents**
  - Constructors – how they are affected
  - Overriding of methods
  - Substitutability
  - Runtime method version look up - polymorphism

- **Hands-on labs**

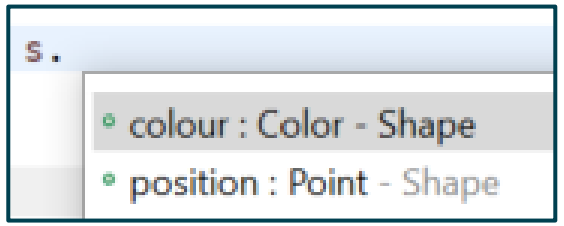
# The principle of substitutability

- **Object of derived type exhibits all behavior of base type**
  - A derived object is a ‘kind of’ base object

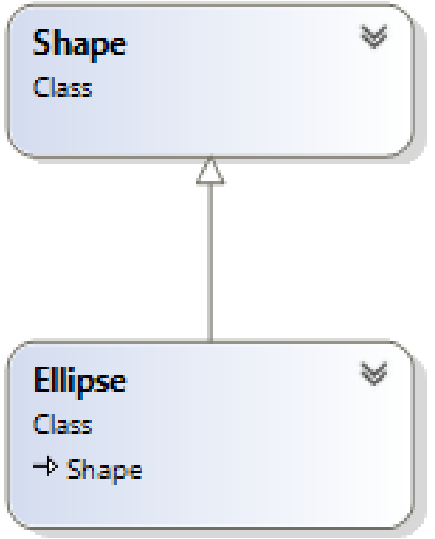
```
Ellipse e = new Ellipse();
Shape s = e;
```




But why would you do it?




Missing all the Ellipse's stuff



# Why use substitution of references?

```
public static Shape MakeShape(int picNo)
{
  if (picNo == 1)
        return new Ellipse(10,5, 100,20);

  if (picNo == 2)
        return new Rectangle(3,10, 20,50);
}
```

Returns an instance of base class type (Shape)

```
public static void Main(string[] args)
{
        Shape s = MakeShape(1);
        Ellipse e = (Ellipse) MakeShape(1);
}
```

# Why use substitution of references?

```
Shape[] shapes = {
        myEllipse,
        yourTriangle,
        ourCircle
};

foreach (Shape s in shapes) {
        DrawShape(s);
}
```

Collections and arrays

```
public void DrawShape(Shape shape)
{
    // code for drawing a shape
}
```

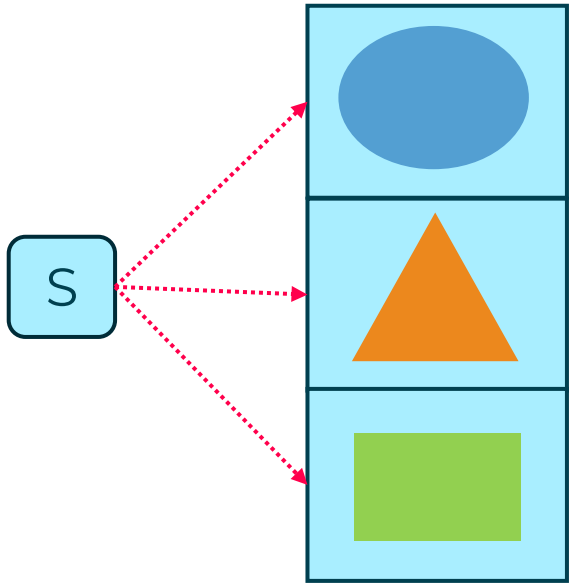
Is passed a parameter of base class type (Shape)

# Towards polymorphism…

- Shape can take many (poly) forms (morphism).

```
Shape[] shapes = {
        myEllipse,
        yourTriangle,
        ourRectangle
};


foreach (Shape s in shapes)
{
        DrawShape(s);
}
```




```
public void DrawShape(Shape shape)
{
    // code for drawing a shape
}
```

# Overriding base class methods

```
Rectangle rect = new Rectangle();

Console.WriteLine(rect.GetArea());
```

```
class Shape
{
    public virtual int GetArea()
    {
        return 0;
    }
}

class Rectangle : Shape
{
    public override int GetArea()
    {
        return 100;
    }
}
```

**Which method is invoked?**

**Shape getArea() or Rectangle getArea()?**

**100**

# Overriding base class methods...

```
Shape rect = new Rectangle();

Console.WriteLine(rect.GetArea());
```

```
class Shape
{
    public virtual int GetArea()
    {
        return 0;
    }
}

class Rectangle : Shape
{
    public override int GetArea()
    {
        return 100;
    }
}
```

**Which method is invoked?**

**Shape getArea() or Rectangle getArea()?**

**100**

# Enabling overriding

- A derived class might want to alter implementation
- Required to use the `virtual` keyword

```
public class Shape
{
  public Point position;
  public Color colour;

  public virtual int GetArea()
  {
        return 0;
  }
}
```

```
public class Rectangle : Shape
{
  public int width, height;

  public override int GetArea()
  {
        return width * height;
  }

  public int GetArea(int k)
  {
        return width * height * k;
  }
}
```

Can only override virtual methods

Method overloading (different from overriding)

# Polymorphism – Lists and Arrays

```
public class Shape
{
  public Point position;
  public Color colour;

  public virtual int GetArea()
  {
        return 0;
  }
}
```

```
public class Rectangle : Shape
{
  public int width, height;

  public ộŵêssîđê int GetArea()
  {
        return width * height;
  }
}
```

```
Shape myShape = new Shape();
Rectangle myRectangle = new Rectangle();
Shape[] shapes = { myShape, myRectangle };

foreach (Shape shape in shapes)
        Console.WriteLine(shape.GetArea());
```

**Which of the GetArea() methods are invoked?  Shape or Rectangle?**

# **Basics of casting – downcasting** (type refinement)

- The data type of a reference controls what is 'visible'

```
class Person
{
  private string name;
  public string GetName()
  {
     return name;
  }
}
```

```
class Student : Person
{
  private string subject;
  public string GetSubject()
  {
    return subject;
  }
}
```

```
Person[] people = { new Person(), new Student() };
foreach(Person p in people)
{
        Console.WriteLine(p.GetName());          // Every Person has a name
        Console.WriteLine(p.GetSubject());       // Person -no subject

        Student s = (Student)p;                  // new reference has new type
        Console.WriteLine(s.GetSubject());       // Student has 'Subject'
}
```

**Will this compile? Will it crash at runtime?**

# Safe downcasting

**A downcast could fail at runtime with '`InvalidCastException`'**

- Test whether cast is safe via the `is` keyword

```
Person[] people = { new Person(), new Student() };

foreach(Person p in people)
{
    Console.WriteLine(p.GetName());      // Every Person has a name

☑   if (p is Student)
    {
        Student s = (Student) p;              // cast to Student type
        Console.WriteLine(s.GetSubject());  // only a Student has a subject
    }
}
```

# Invoking base class functionality

- **A derived class can access base class member**

**Calls first method with matching signature up the inheritance hierarchy**

```
public class Student : Person
{
  private string subject;
  public string GetDetails()
  {
    string data = base.GetDetails(); // call base class
    return data + "\t" + subject;
  }
}
```

# Non-extensible classes and methods

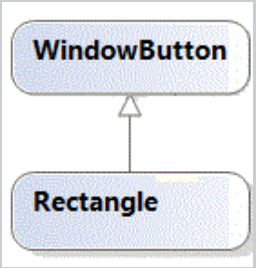
- A class can be written with the `sealed` modifier to prevent extension

```
public sealed class String
{
   ...
}
```

# Best practice

**Use inheritance only for genuine "is a" relationships**

Logical to substitute object of derived class for object of base class. Methods in base class should make sense in derived class




**Ad-hoc inheritance for short-term convenience tends to lead to future problems and surprises!**

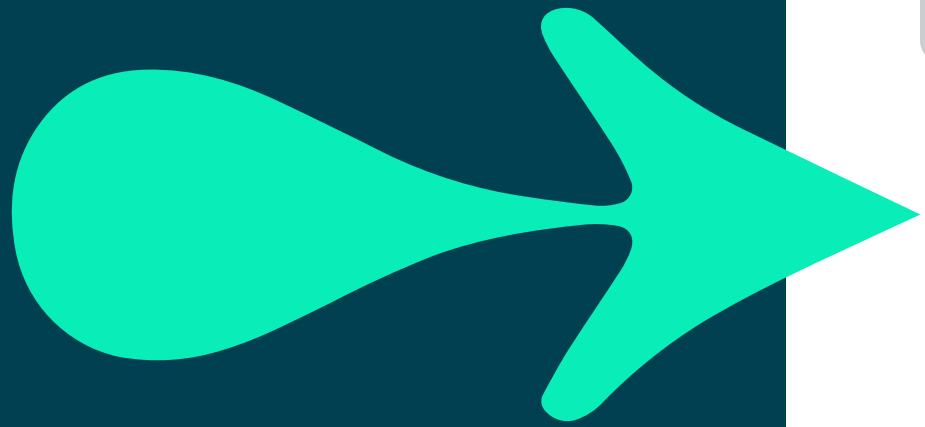
**C# only supports single inheritance** Choosing a base class is thus significant


- **What is polymorphism?**
  - Poly (many) morphism (forms)
  - We can treat an object as a generic type (e.g. Shape), but when we try to access it, the program will determine the specific type of the object (e.g. Rectangle, Ellipse) and run the associated code.

- **A derived class inherits from the base class, and can override methods**
  - Method calls are automatically polymorphic

- **Looked at downcasting**
  - Refining a reference of a base class down to a more specific subclass

# LABS



**Polymorphism**

**Duration 60 minutes**