ITE 365 Lab: Polymorphism in C#

Adam Lewis

June 11, 2018

Contents

1 Objectives

1. Understand polymorphism, particularly in C#.
2. Implement and override virtual methods in C#.
3. Use polymorphism in a program.
4. Implement our own package in C#.

2 Background

Polymorphism is often referred to as the third pillar of object-oriented programming after encapsulation and inheritance. The word is derived from Greek and means "many-shaped" in English. In object-oriented programming, polymorphism has two distinct aspects:

- At run time, objects of a derived class may be treated as objects of a base class in places such as method parameters and collections or arrays. This is known as the *Liskov Substitution Principle*.
- Base classes may define and implement *virtual* methods that derived classes can *override* to provide their own definition and implementation.

Let's consider an application that enables a user to create different shapes on a drawing surface. You do not know at compile time which specific types of shapes the user will create. The application has to keep track of all of the various types of shapes that have been created and it has to update them in response to user actions. Polymorphism can be used to solve this problem in two basic steps:

- Create a class hierarchy in which each specific shape class derives from a common base class.
- Use a virtual method to invoke the appropriate method on any derived class through a single call to the base class method.

3 Instructions

3.1 Setup

Start Visual Studio and create a new Console application named "PollyMoTest". Combine a copy of your source code and a screen-shot of your program into a PDF file (you can use Microsoft Word to do this) and attach it to the assignment on Blackboard.

3.2 Building the class library

- Use the *Project>Add New Item* dialog to add a new class named **Shape**... Select "Class" and rename your class and file to be "Shape":

```
public class Shape
   // A few example members
    public int X { get; private set; }
    public int Y { get; private set; }
    public int Height { get; set; }
    public int Width { get; set; }
```

```
    // Virtual method
    public virtual void Draw()
    {
        Console.WriteLine("Performing base class drawing tasks");
    }
}
```

- In the same file, add the following classes:
  - A derived class **Circle** that overrides the **Draw()** method.
  - A derived class **Rectangle** that overrides the **Draw()** method.
  - A derived class **Triangle** that overrides the **Draw()** method.
  - Make certain that each of these classes is defined as **public**.

3.3 Now back in the main function in the app

Add the following to the **main()** function in your application:

```
static void Main(string[] args)
    {
        // Polymorphism at work #1: a Rectangle, Triangle and Circle
        // can all be used whereever a Shape is expected. No cast is
        // required because an implicit conversion exists from a derived
        // class to its base class.
        System.Collections.Generic.List<Shape> shapes = new System.Collections.Generic
        shapes.Add(new Rectangle());
        shapes.Add(new Triangle());
        shapes.Add(new Circle());

        // Polymorphism at work #2: the virtual method Draw is
        // invoked on each of the derived classes, not the base class.
        foreach (Shape s in shapes)
        {
            s.Draw();
        }

        // Keep the console open in debug mode.
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
```

Build and run your application. Set a break point at call to the **Draw()** method in the **foreach** loop. What is the data type of the object in this case?

4 Submission instructions

Put your final code and screen shot of your application into a PDF file. Attach this PDF file to your submission on Blackboard.