Faculty of Science, Engineering and Technology

Object Oriented Programming

Distinction Task 5.3: Drawing Program — Flexible Saving

Overview

Drawing programs have a natural affinity with object oriented design and programming, with easy to see roles and functionality. In this task you will start to create an object oriented drawing program.

Purpose:	See how to use inheritance to create families of related objects and interact with them as a group (polymorphism)
Task:	Extend the shape drawing program to provide multiple different kinds of shapes.
Time:	Aim to complete this task by the start of week 6
Resources:	

Submission Details

You must submit the following files to Doubtfire:

- Program source code
- Screenshot of program execution

Instructions

One problem with our Loading of files is that the Drawing must know about the subclasses of Shape so that it can create the right version. Ideally this should not need to know these details, and we can solve this by adding some features to our Shape class and make use of C# reflection classes.

> **Note**: Reflection APIs provide you details about classes and objects in memory. This is a powerful API once you understand what it can do.

1. At the top of the Shape class, add the following:

```
public abstract class Shape
{
    private static Dictionary<string, Type> _ShapeClassRegistry = new Dictionary<string, Type>();

    public static void RegisterShape(string name, Type t)
    {
        _ShapeClassRegistry[name] = t;
    }

    public static Shape CreateShape(string name)
    {
        return (Shape)Activator.CreateInstance (_ShapeClassRegistry [name]);
    }
```

> **Note**: A Dictionary is a kind of collection. With a Dictionary you can specify what indexes (keys) you will use, and what values it can store. In this case the Dictionary uses a string index to access a **Type** object.

This code uses a **Dictionary** to allow callers to register a string to match a certain type. This allows the caller to register that "Rectangle" is the Rectangle class, and that "Circle" is the Circle class for example.

Create Shape is then able to fetch the Type from the Dictionary and use it to create an object (CreateInstance) using the **Activator** class from the reflections library.

This means that callers can now create objects using s = Shape.CreateShape("Rectangle"); or better yet, s = Shape.CreateShape(kind);

2. Switch to **Game Main** and register "Rectangle" as referring to the Rectangle class. Place this as the first instruction in Main.

```
Shape.RegisterShape ("Rectangle", typeof(Rectangle));
```

3. Also register the other classes with the strings they save in the file.
4. Switch to **Drawing** and locate the **Load** method. Remove the switch statement and replace it with the code to call Shape's CreateShape method. Notice it no longer needs to know about the different shape classes.
5. Compile and run the program and make sure it works correctly.

Currently the child classes of Shape write out the "key" that is used to locate their type in the dictionary. This could get messy if things get out of sync, instead we can get Shape to write this out.

6. Create a static method in Shape that allows you to get the key of a Type. This will need to search through the Dictionary's **Keys** property. The Keys property returns a list of key values you can use to access the values. Search through this list to find the matching value, you can then return its associated key value.
7. In **Shape**'s **SaveTo** method write out the key of the object's type.

> **Hint**: You can get the type of an object by asking it to **GetType()** (i.e. obj.GetType())

8. Remove the code from Circle, Rectangle, and Line that writes out the type. For example Rectangle should now be:

```
        public override void SaveTo (System.IO.StreamWriter writer)
        {
//          writer.WriteLine ("Rectangle");
            base.SaveTo (writer);
            writer.WriteLine (Width);
            writer.WriteLine (Height);
        }
```

9. Compile and run the program. Make sure that it works as it did before.

Upload the code from this tasks to Doubtfire to include it in your portfolio.