Faculty of Science, Engineering and Technology

Object Oriented Programming

Topic 2: Unit Testing

Resources

The following resources can help you with this topic:

- NUnit Documentation

Topic Tasks

Before starting to work on the tasks, first read through this entire document to get a sense of the direction in which you are heading. Complete the following tasks and submit your work to Doubtfire for feedback before the deadline (check Doubtfire for the submission deadline).

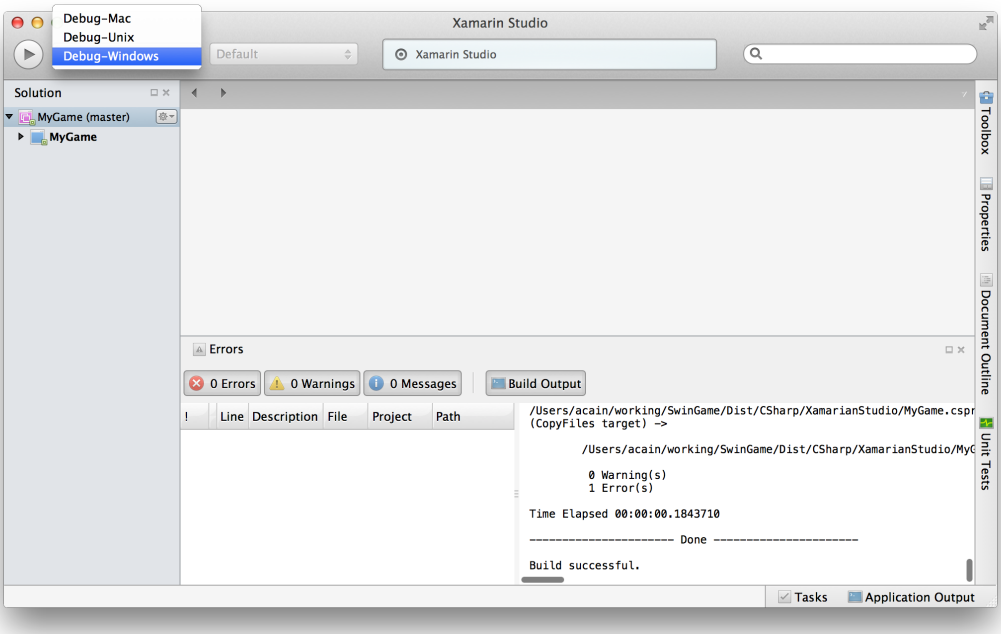
Remember to submit your progress, even if you haven't finished everything.

After you have **discussed** your work with your tutor and **corrected any issues**, it will be signed off as complete.

Pass Task 5 - Shape Drawer

Over the next few weeks you will develop a simple shape drawing program. Users will be able to select between different shapes and draw them to the canvas. In this stage you will start by creating a Shape class and drawing it to the screen.

1. Download the **SwinGame C# Template** from Blackboard. This contains the code to use SwinGame using C# code. This file contains a project that you can open with Xamarin Studio.
2. Open the **MyGame.sln** using Xamarin Studio.
3. Choose your operating system from the configurations drop down.

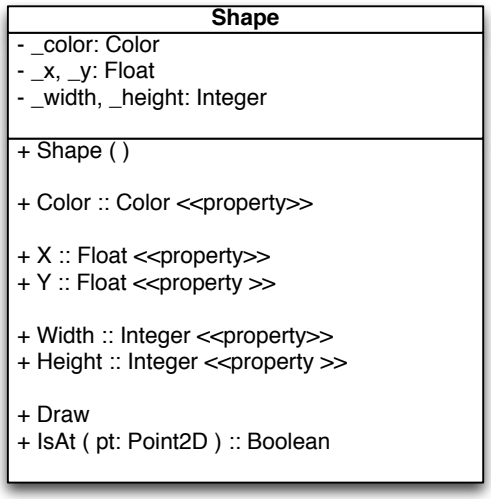


4. Review the code in the **GameMain** class.
5. Run the program and make sure you see the window and can hear the startup sound from the splash screen. The first time you run the program it may take some time to load as the .NET framework is loaded (typically after the splash screen shows). After the first time it should run much faster.

Note: If you have issues at this point let us know. Error messages can help identify problems. The text from the **Build Output** should help identify the issue.

Note: On Linux you may need to run from the Terminal after compiling from the IDE.

6. Now it is time to add your Shape class. Use the following UML class diagram as a guide.



7. To get the Color type you will need to add the following code, but the other fields and properties should be straight forward.

```
using Color = System.Drawing.Color;

namespace MyGame
{
    public class Shape
    {
        private Color _color;
        ...
```

8. In the constructor initialise the color to Color.Green, x, y to 0,0 and the size to 100 by 100.

9. The Draw method will draw the shape as a filled rectangle using the shape's Color, position, and size.

```
namespace MyGame
{
    public class Shape
    {
        ...
        public void Draw()
        {
            SwinGame.FillRectangle ( _clr,
                                     _x, _y,
                                     _width, _height);
        }
```

Tip: Type **SwinGame.** to get the IDE to list *all* of the SwinGame functions for you!

10. Add a **IsAt** method that takes in a **Point2D** (a struct that contains an X and Y value representing a point in 2d space - like a point on the screen) and returns a boolean to check if the shape is at that point. You need to return true if the point (pt) is within the shape's area (as defined by the shape's fields).

Tip: SwinGame has a PointInRect that can do the hard lifting for you here. When the function is selected in the editor press down to switch between the different **overloads**.

```
▲ 3 of 4 ▼
public static bool PointInRect (
  Point2D pt,
  float x,
  float y,
  float w,
  float h
)
```

Note: Method names can be **overloaded**, this means that multiple methods can use the one method name. This gives the caller options on how the method is called.

11. Switch back to **GameMain** so that you can test out your new shape class.
12. Add a **myShape** local variable of the Shape type.
13. Assign myShape, a **new** Shape object outside the loop.
14. Inside the loop:
    14.1. Tell **myShape** to **Draw** itself - after clearing the screen
    14.2. If the user clicks the LeftButton on their mouse, set the shapes x, y to be at the mouse's position.

Hint: Check out **MouseClicked**, **MouseX**, and **MouseY** functions.

    14.1. If the mouse is over the shape (i.e. it **is at** the same point as the **mouse position**) and the user presses the spacebar, then change the Color of the shape to a random color.

Hint: Check out **KeyTyped, MousePosition** and **RandomRGBColor** - set the Alpha to 255 so it is opaque.

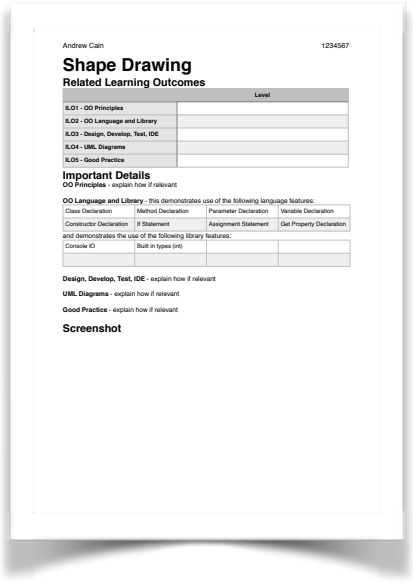
15. Compile and run your program.

Tip: It is probably best to make these changes one at a time, and compile and run as you go. It is easier to build a little at a time.

Once your program is working correctly you can prepare it for your portfolio.

Add a few sentences for the learning outcomes you feel that you have demonstrated some aspect of in this work. Include screenshots of the program in action.

Tip: Think about what you are demonstrating with this. Showing understanding and thought at this step will make it easier to summarise when you get to the end of the unit. Reflect on any aspects you found interesting or challenging in this process.



Andrew Cain 1234567

Shape Drawing

Related Learning Outcomes

	Level
ILO1 - OO Principles	
ILO2 - OO Language and Library	
ILO3 - Design, Develop, Test, IDE	
ILO4 - UML Diagrams	
ILO5 - Good Practice	

Important Details

OO Principles - explain how if relevant

OO Language and Library - this demonstrates use of the following language features:

Class Declaration	Method Declaration	Parameter Declaration	Variable Declaration
Constructor Declaration	If Statement	Assignment Statement	Get Property Declaration

and demonstrates the use of the following library features:

Console IO	Built in types (int)		

Design, Develop, Test, IDE - explain how if relevant

UML Diagrams - explain how if relevant

Good Practice - explain how if relevant

Screenshot

Pass Task 5 - Assessment Criteria

Make sure that your task has the following in your submission:

- The program is implemented correctly with a Shape class as indicated
- Code must follow the C# coding convention used in the unit (layout, and use of case).
- The code must compile and the screenshot show it outputting the correct details.

Pass Task 6 - Unit Testing Shape

Unit Testing is a practice of writing small tests to check functional aspects of your program. In object oriented programming this can be used to create tests for the objects you create. In this case we can test the **IsAt** function of the **Shape** to ensure it is working correctly.

> **Note**: You run these tests frequently during development. That way if changes break existing features you find out as soon as possible.

1. In the **Shape Drawing** program, add a new File and choose **NUnit, Test Fixture**. Name the file **Shape Tests**.
2. Add a **using** statement to give you access to the Color and SwinGame classes.

```
using Color = System.Drawing.Color;
using SwinGameSDK;
```

3. Implement the following test to make sure that the IsAt method works correctly.

```
[TestFixture ()]
public class ShapeTest
{
    [Test ()]
    public void TestShapeAt ()
    {
        Shape s = new Shape ();

        s.X = 25;
        s.Y = 25;
        s.Width = 50;
        s.Height = 50;

        Assert.IsTrue (s.IsAt(SwinGame.PointAt(50, 50)) );
        Assert.IsTrue (s.IsAt(SwinGame.PointAt(25, 25)) );
        Assert.IsFalse (s.IsAt(SwinGame.PointAt(10, 50)) );
        Assert.IsFalse (s.IsAt(SwinGame.PointAt(50, 10)) );
    }
}
```

> **Note**: The code in the [ ] before the class and method is called an **attribute**. This is an *annotation* to the class or method and is used by NUnit when it reads the project to run the tests.

4. Once you have the code as shown above, run the unit tests. There is a tab titled **Unit Tests** where you can run the tests and see the results. Otherwise use the Run, Run Unit Tests from the menu.
5. Make sure that the TestShapeAt test is giving you the green light. If it isn't check your code both in the test and in the class.

In general each Unit Test follows a fairly simple pattern: you create an object, initialise it with data and then ask it to do something or give you some information. You then **assert** that the information you got is correct, or that the object is now in the right state. The NUnit framework helps support this by providing tools to make this easier to visualise the results.

Note: An assertion (to assert something) means that the statement must be true, otherwise the test **fails**. If there are no assertions, then the test is **inconclusive**.

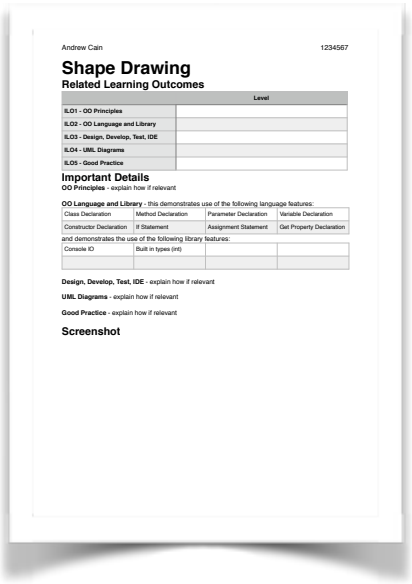
NUnit provides a number of classes that can help you make assertions in your unit tests.

- The **Assert** class provides basic abilities to test if things are true, false, the same, etc.
- **StringAssert** provides methods that make it easier to make assertions about strings.
- **CollectionAssert** makes it easier to test arrays and other collections of objects.

6. Add a second test to check that moving the shape changes the result of IsAt.
   - Name the test **TestShapeAtWhenMoved**
   - Position the shape, and test one point of it at that location.
   - Change the shape's position so that it is no longer at that point.
   - Retest to see if the shape isAt its old location - that should be false.
7. Get two green lights before continuing.
8. Add a third test to check that resizing the shape changes the result of IsAt.
   - Name the test **TestShapeAtWhenResized**
   - Position the shape, and test one point of it at that location.
   - Change the shape's size so that it is no longer at that point.
   - Retest to see if the shape isAt its old location - that should be false.
9. Get three green lights before continuing.

Tip: When things don't work you should be able to view the results in the **Test Results** pad. Double click the error to locate the line of code where the test failed.

Once your tests are working correctly you can prepare this piece for your portfolio. Do not create a new cover page, instead add to the existing Shape Drawing page. You will only submit the final version in your portfolio.



Andrew Cain 1234567

Shape Drawing

Related Learning Outcomes

	Level
ILO1 - OO Principles	
ILO2 - OO Language and Library	
ILO3 - Design, Develop, Test, IDE	
ILO4 - UML Diagrams	
ILO5 - Good Practice	

Important Details

OO Principles - explain how if relevant

OO Language and Library - this demonstrates use of the following language features:

Class Declaration	Method Declaration	Parameter Declaration	Variable Declaration
Constructor Declaration	If Statement	Assignment Statement	Get Property Declaration

and demonstrates the use of the following library features:

Console IO	Built in types (int)		

Design, Develop, Test, IDE - explain how if relevant

UML Diagrams - explain how if relevant

Good Practice - explain how if relevant

Screenshot

Pass Task 6 - Assessment Criteria

Make sure that your task has the following in your submission:

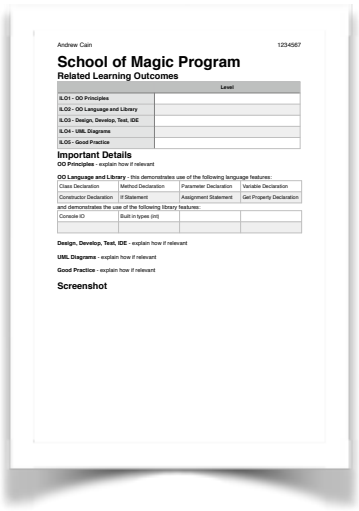
- The unit tests correctly check the position of the shape.
- Code must follow the C# coding convention used in the unit (layout, and use of case).
- The code must compile and the screenshot show the tests passing.

Pass Task 7 - Unit Testing the Spells

Return to your School of Magic program. Add Unit Tests to check the functionality of the Spell class.

- Create one test for each spell kind to check that casting it returns the correct data (3 tests)
  - For example: check that a Heal spell returns "Ahhh… you feel better".
- Create one test to check that you can change the name of a spell

Once your tests are working correctly you can prepare this piece for your portfolio. Do not create a new cover page, instead add to the existing School of Magic page.



Andrew Cain 1234567

School of Magic Program

Related Learning Outcomes

	Level
ILO1 - OO Principles	
ILO2 - OO Language and Library	
ILO3 - Design, Develop, Test, IDE	
ILO4 - UML Diagrams	
ILO5 - Good Practice	

Important Details

OO Principles - explain how if relevant

OO Language and Library - this demonstrates use of the following language features:

Class Declaration	Method Declaration	Parameter Declaration	Variable Declaration
Constructor Declaration	If Statement	Assignment Statement	Get Property Declaration

and demonstrates the use of the following library features:

Console IO	Built in types (int)		

Design, Develop, Test, IDE - explain how if relevant

UML Diagrams - explain how if relevant

Good Practice - explain how if relevant

Screenshot

Pass Task 7 - Assessment Criteria

Make sure that your task has the following in your submission:

- The unit tests correctly check the output from the different kinds of spell, and the ability to change the name of the spell
- Code must follow the C# coding convention used in the unit (layout, and use of case).
- The code must compile and the screenshot show the tests passing.

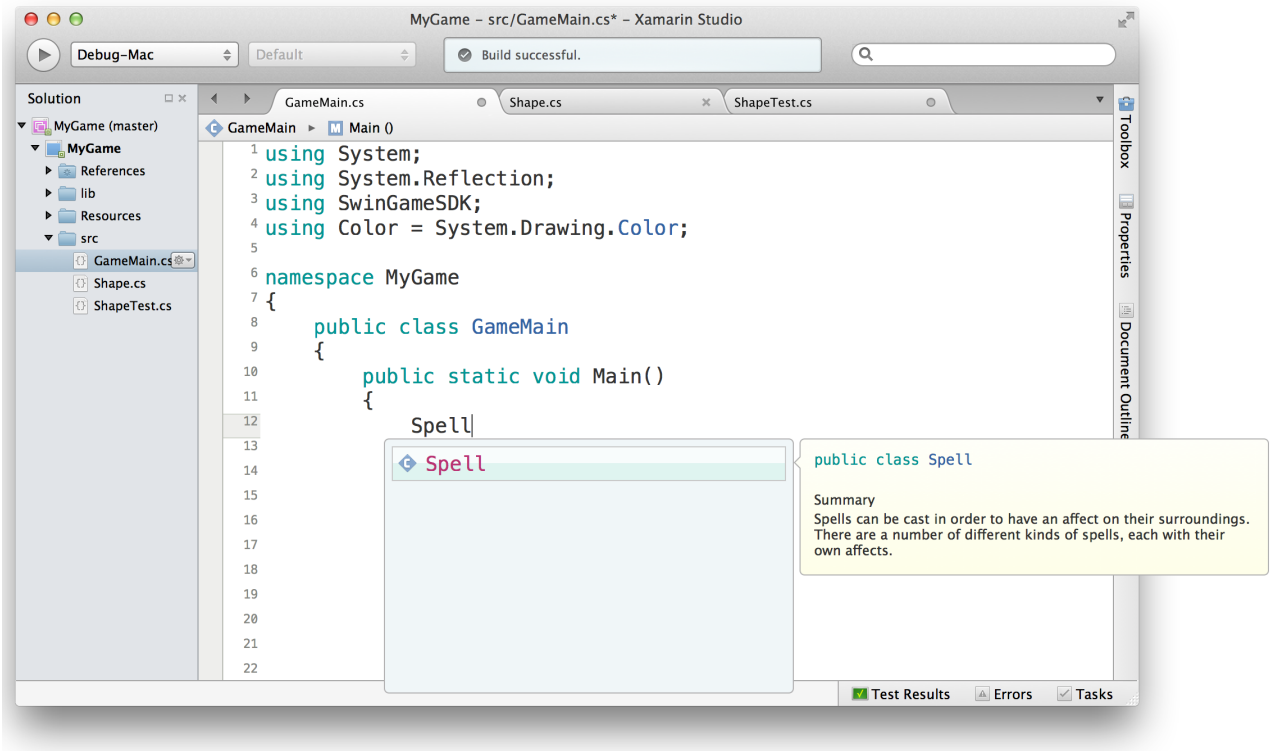
Pass Task 8 - Documenting the Spell Class

In a real-world project it is critically important to document you code. Software is developed in teams, and documentation can help everyone quickly understand what the classes, methods, and properties of your code do.

C# provides a code documentation standard called **XML documentation**. This provides a means of

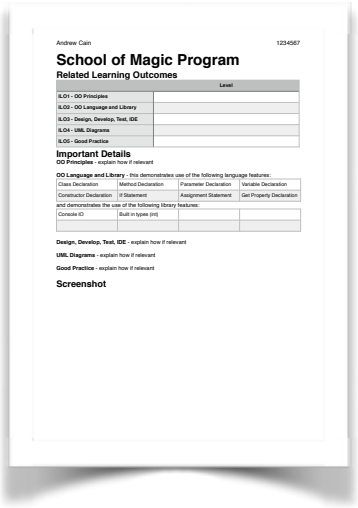
```
/// <summary>
/// Spells can be cast in order to have an affect
/// on their surroundings. There are a number of
/// different kinds of spells, each with their own
/// affects.
/// </summary>
public class Spell
{
    /// <summary>
    /// Cast this spell, causing it to have an
    /// effect on its surroundings.
    /// </summary>
    /// <returns>a description of the effect</returns>
    public string Cast()
    {
```

The great thing about this documentation is that it is read and understood by the IDE, so when you document the method and classes in your project that documentation is immediately useful.



1. Open your School of Magic program
2. Add XML documentation to the Spell class by adding *///* to the line before the class, constructor, method, or property header. This will start the documentation for you, though you can add extra tags in < … > after the summary.
   2.1. Write a short description for the class itself.
   2.2. Write descriptions for the constructor, methods and properties within the class
3. Check that you can see this documentation in the IDE

Once your code is correctly documented you can prepare this piece for your portfolio. Do not create a new cover page, instead add to the existing School of Magic page.



Andrew Cain 1234567

School of Magic Program

Related Learning Outcomes

	Level
ILO1 - OO Principles	
ILO2 - OO Language and Library	
ILO3 - Design, Develop, Test, IDE	
ILO4 - UML Diagrams	
ILO5 - Good Practice	

Important Details

OO Principles - explain how if relevant

OO Language and Library - this demonstrates use of the following language features:

Class Declaration	Method Declaration	Parameter Declaration	Variable Declaration
Constructor Declaration	If Statement	Assignment Statement	Get Property Declaration

and demonstrates the use of the following library features:

Console IO	Built in types (int)		

Design, Develop, Test, IDE - explain how if relevant

UML Diagrams - explain how if relevant

Good Practice - explain how if relevant

Screenshot

Pass Task 8 - Assessment Criteria

Make sure that your task has the following in your submission:

- All public features of the Spell class have XML documentation.
- Code must follow the C# coding convention used in the unit (layout, and use of case).
- The code must compile and the screenshot show the tests passing.