# Unit 2 - Programming Projects

Create a new project/solution for each of the following projects. Name the solution Unit?Project? and the project Project?.

Example:

* Solution: Unit1Project4
* Project: Project4

Some projects will require files from previous projects. Copy files from previous projects to the new projects using File Explorer in Windows. The files are then added to the project in Visual Studio.

## Properties

### Project 1

Create a new console application. Update the Rectangle class created in a previous project such that it include properties.

Make the following changes:

1. Add two properties name Width and Height. Each property has a get and set accessor.
2. Replace the GetArea() and GetPerimeter() with the properties named Area and Perimeter.

Update the Program class to demonstrate new Rectangle functionality.

### Project 2

Create a new console application. Update the Stock class created in a previous project such that it include properties.

Make the following changes:

1. Add properties for Stock’s symbol, name and previous closing price. These properties are read only.
2. Add a property for the Stock’s current price. This property has a get and set accessor.
3. Replace the GetChangePercent() method with a property.

Update the Program class to demonstrate new Stock functionality.

## Inheritance

### Project 3

Create a new console application. Code a class named Person and its two subclasses named Student and Employee. Make Faculty and Staff subclasses of Employee.

A Person has the following attributes:

* name
* address
* phone number
* email address

A student has the following attributes:

* class status (Enumeration: Freshman, Sophomore, Junior, or Senior)

An employee has the following attributes:

* office (string)
* salary (decimal)
* date hired (DateTime)

A faculty member has the following attributes:

* office hours (string)
* rank (Enumeration: Assistant, Teaching, Associate, Senior, Distinguished)

A staff member has the following attributes:

* title (string)

Create properties for each attribute (get and set). Create at least one constructor for each class that accepts data for all attributes. Override the ToString method in each class to display the class name and the person’s name.

Write a test program that creates a Person, Student, Employee, Faculty, and Staff, and invokes their ToString() methods.

### Project 4

Create a new console application. Create a class called GeometricShape. GeometricShapes have:

* A string field called color that specifies the GeometricShape’s color.
* A boolean field called isFilled that specifies whether the GeometricShape is filled.
* The default GeometricShape is white and not filled.
* A no-argument constructor that creates a default GeometricShape.
* A constructor that creates a GeometricShape with a specified color and whether it is filled or not.
* A property for each attribute with both get and set accessors.
* A ToString() method that returns a string that includes the GeometricShape’s color and isFilled.

Create a class called Circle that is a GeometricShape. A Circle has:

* A decimal field called radius that specifies that Circle’s radius.
* A no-argument constructor that creates a Circle with a radius of 1.
* A constructor that creates a Circle with a specified radius.
* A constructor that creates a Circle with a specified radius, color, and whether it is filled or not.
* A property for the radius with both get and set accessors.
* A read-only property called Area that gets the area of the Circle.
* A read-only property called Perimeter that gets the perimeter of the Circle.
* A read-only property called Diameter that gets the diameter of the Circle.
* A ToString() method that returns a string that includes the Circle’s color, isFilled, radius, area, perimeter and diameter.

Create a class called Rectangle that is a GeometricShape. A Rectangle has:

* A decimal field called height that specifies that Rectangle’s height.
* A decimal field called width that specifies that Rectangle’s width.
* A no-argument constructor that creates a Rectangle with a height and width of 1.
* A constructor that creates a Rectangle with a specified height and width.
* A constructor that creates a Rectangle with a specified height, width, color, and whether it is filled or not.
* A property for height and width with both get and set accessors.
* A read-only property called Area that gets the area of the Rectangle.
* A read-only property called Perimeter that gets the perimeter of the Rectangle.
* A ToString() method that returns a string that includes the Rectangle’s color, isFilled, height, width, area and perimeter.

Write a test program that creates a Circle and Rectangle and invokes all methods.

## Abstract Classes

Modify the GeometricShape class to be abstract. Ensure that all GeometricShapes implement a read-only property for Area and Perimeter.

Code a class named Triangle that extends GeometricObject. The class contains these unique attributes and behaviours:

* Three decimal data fields named side1, side2, and side3 to denote length of each side of the Triangle.
* A no-arg constructor that creates a default Triangle where its sides are 1.0.
* A constructor that creates a Triangle with the specified side1, side2, and side3.
* Properties for each of the fields with both get and set accessors.

Create a test program that creates a Triangle object and tests all of its behaviours.

## Exceptions

### Modify project 2

Update the Stock class such that it throws the following exceptions:

Constructor

* throw ArgumentException when the symbol contains no characters after whitespace is removed.
* throw ArgumentException when the name contains no characters after whitespace is removed.
* throw ArgumentOutOfRangeException when the current price is less than zero.

CurrentPrice Property

* throw ArgumentOutOfRangeException when the property is set to a value less than zero.

Create a test program to ensure that the exceptions are thrown (and not thrown) as stated above.

### Modify project 4

Update the following classes such that the methods throw the following exceptions:

#### **GeometricShape Class**

Constructor

* throw the appropriate exception when the color has no characters after removing white space.

Color Property

* throw the appropriate exception when the property is set to a value that has no characters after removing white space.

#### **Circle Class**

Constructor

* throw the appropriate exception when the radius is less than or equal to zero.

Radius Property

* throw the appropriate exception when the property is set to a value that is less than or equal to zero.

#### **Rectangle Class**

Constructor

* throw the appropriate exception when the height is less than or equal to zero.
* throw the appropriate exception when the width is less than or equal to zero.

Height Property

* throw the appropriate exception when the property is set to a value less than or equal to zero.

Width Property

* throw the appropriate exception when the property is set to a value less than or equal to zero.

#### **Triangle Class**

Constructor

* throw the appropriate exception when either of the sides is less than or equal to zero.

Side Properties

* throw the appropriate exception when the properties are set to a value less than or equal to zero.

Modify the Program to ensure that the exceptions are thrown (and not thrown) as stated above.

### Project 5

Create a new console application. Update the GradeItem class created in a previous project such that it throws the following exceptions:

Constructor

* throw ArgumentOutOfRangeException when the score is less than or equal to zero.
* throw ArgumentOutOfRangeException when the max score is less than score.

Create a test program to ensure that the exceptions are thrown (and not thrown) as stated above.

## Static Classes

### Project 6

Create a new console application. Create a static class named Number with the following static members:

* A constant field named MY\_FAVORITE\_NUMBER. The value of which would be your favorite number.
* A method named IsNumeric(number : string) : bool that returns true when the specified string contains a decimal equivalent value. It will return false when it does not.
* A method named IsDivisibleBy(number : int, divisor : int) : bool that returns true when the specified number is divided evenly (no remainder) by the divisor. It will return false when it does not.
* A method named GetRandomNumber(low : int, high : int) : int that returns a random number between low and high. When the low value is greater than or equal to high, return the high.

Create a program that tests the functionality of the Number class created in Exercise 1. Ensure that you test all outcomes of each method. No user input is required.

Update the Number class such that it throws the following exceptions:

IsDivisibleBy method

* throw the appropriate exception when number <= 0.
* throw the appropriate exception when divisor <= 0.

GetRandomNumber method

* throw the appropriate exception when high <= low.

Update the program to ensure that the exceptions are thrown (and not thrown) as stated above.