Overview

This project will provide practice with several concepts learned thus far this semester

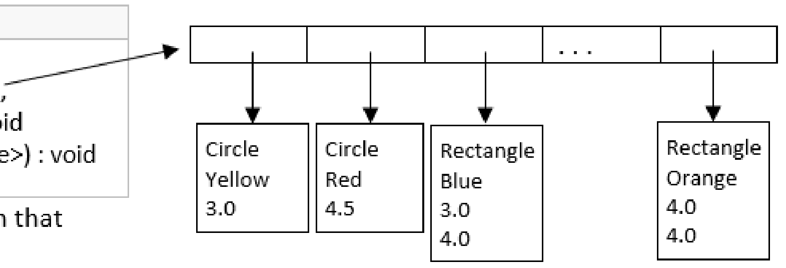
Sample Input File and Format (shapes.txt)

Sample data (shapes.txt)	Sample with Bad Data (highlighted)	Record format
Circle,Yellow,3.0	Circle,Yellow,-3.0	Circle,color,radius
Circle,Red,4.5	Circle,Red,2.5	or
Rectangle,Blue,3,4	Rectangle,Blue,3.5,4.5	Rectangle,color,length,width
Rectangle,Red,3.5,5.1	Rectangle,Red,-3.5,-5.1	
Circle,Light Green,2.5	Circle,Gray,2.1	
Rectangle,Orange,4.0,4,0	Rectangle,Orange,4.0,-4.0	
	Rectangle,Black,-2.0,4.0	
	Rectangle,Dark Purple,5.0,4.0	

The length, width, and radius could contain floating point values.
color will contain a string.

Visual

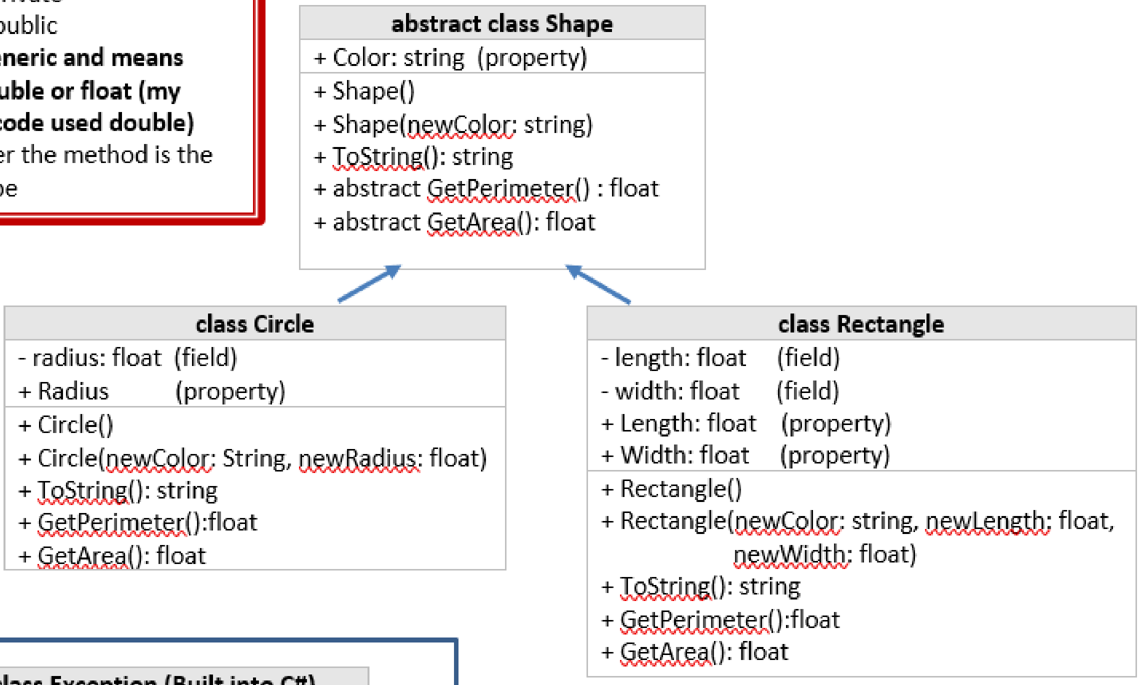
class Program
+ main(args: String[]): void
+ static getShapes(shapes: List<Shape>, filename: String) : void
+ static showShapes(shapes: List<Shape>) : void



FYI: **shapes** is a local variable in main that refers to a list of Shape objects

•

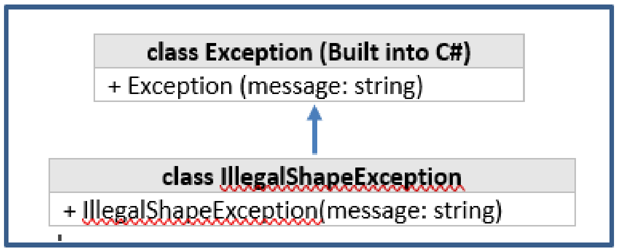
In these diagrams,
- means private
+ means public
float is generic and means either double or float (my solution code used double)
: type after the method is the return type



abstract class Shape
+ Color: string (property)
+ Shape()
+ Shape(newColor: string)
+ ToString(): string
+ abstract GetPerimeter() : float
+ abstract GetArea(): float

class Circle
- radius: float (field)
+ Radius (property)
+ Circle()
+ Circle(newColor: String, newRadius: float)
+ ToString(): string
+ GetPerimeter():float
+ GetArea(): float

class Rectangle
- length: float (field)
- width: float (field)
+ Length: float (property)
+ Width: float (property)
+ Rectangle()
+ Rectangle(newColor: string, newLength: float, newWidth: float)
+ ToString(): string
+ GetPerimeter():float
+ GetArea(): float



class Exception (Built into C#)
+ Exception (message: string)

class IllegalShapeException
+ IllegalShapeException(message: string)

Project CS201

Requirements

Create a new Visual Studio project, download the two sample data files, and place the data files into the source code folder for the project.

Create class **IllegalShapeException** in its own file. It inherits from class **Exception** and contains:

- Your name properly documented as the author. A class description is not required.
- **public IllegalShapeException(string message) : base (…)** . A constructor that has a string parameter that represents a message. After the colon in the header, it calls the parent constructor (**base**(…)) with a string that consists of "Illegal Shape: " followed by the **message** parameter.
- The body will just consist of an empty pair of braces – there is no code in the body.

Create **class Shape** in its own file**.** It is an **abstract** class and contains

- Your name properly documented as the author. A class description is not required.
- one **public** auto-implemented property named **Color** that holds string.
- **public Shape()**
  - Assigns "White" to the property.
- **public Shape(String newColor)**
  - Assigns **newColor** to the property.
- **public override String ToString()**
  - (Overrides the Object class ToString method)
  - Returns a string holding the name of the class type (use **base.getType().Name**) followed by a comma and the shape's color.

    Example return value: "Circle,Red"
- **an abstract method named GetPerimeter() with a return type of double**
- **an abstract method named GetArea() with a return type of double**

Create **class Circle** in its own file. It **inherits** from class **Shape** and contains

- Your name properly documented as the author. A class description is not required.
- A **private** field (instance variable) named **radius** that will hold a double
- A public property name **Radius**
  - get: returns the value stored in the radius field
  - set: **If** the value is <= 0, the set will throw a new **IllegalShapeException** object with an argument of "Circle radius of value" where value is formatted to 1 digit of precision. **Else**, value will be assigned to the radius field.
- **public Circle() : base()**
  - Calls the base (parent) parameterless constructor (already done)
  - Assigns 1 to the Radius property
- **public Circle(string newColor, double newRadius) : base (newColor)**
  - Sends the color formal parameter to parent constructor (already done)
  - Assigns newRadius to the Radius property
- **public override String ToString()**
  - (Overrides the Shape class ToString method)
  - Creates and returns a string that is built from:
    - the returned result of the parent's ToString() method followed by a comma and the circle's Radius formatted to one digit of precision.

    Example return value: "Circle,Red,4.5"
- **public double GetPerimeter()**
  - Returns the circumference ("perimeter") of the circle. The computation can be performed in the return statement. Use **Math.PI** directly in the computation and utilize the Radius property.
- **public double GetArea()**

- Returns the area of the circle. The computation can be performed in the return statement. Use **Math.PI** directly in the computation and the Radius property. Remember that Math.Pow also exists.

Create **class Rectangle** in its own file. It **inherits** from class **Shape** and contains

- Your name properly documented as the author. A class description is not required.
- A **private** field (instance variable) named **length** that will hold a double
- A **private** field (instance variable) named **width** that will hold a double
- A public property name **Length**
  - get: returns the length
  - set: **If** the value is <= 0, the set will throw a new **IllegalShapeException** object with an argument of "Rectangle length of value" where value is formatted to 1 digit of precision. **Else**, value will be assigned to the length field.
- A public property name **Width**
  - get: returns the width
  - set: **If** the value is <= 0, the set will throw a new **IllegalShapeException** object with an argument of "Rectangle width of value" where value is formatted to 1 digit of precision. **Else**, value will be assigned to the width field.
- **public Rectangle()**
  - Call the parameterless constructor of the parent (See how base was called for the Circle)
  - Assigns 1 to the Length and Width properties
- **public Rectangle(string newColor, double newLength, double newWidth)**
  - Sends the color formal parameter to the parent constructor (See how base was called for the Circle)
  - Assigns newLength to the Length property
  - Assigns newWidth to the Width property
- **public override String ToString()**
  - (Overrides the Shape class ToString method)
  - Creates and returns a string that is built from:
    - the returned result of the parent's ToString() method followed by a comma and the rectangle's Length formatted to one digit of precision followed by a comma and the rectangle's Width formatted to one digit of precision.

    Example return value: "Rectangle,Red,4.5,3.0"
- **public double GetPerimeter()**
  - Returns the perimeter of the rectangle by using the Length and Width properties.
- **public double GetArea()**
  - Returns the area of the rectangle by using the Length and Width properties.

class Program in its own file. It contains

- any needed using statements (For example, you will need System.Collections.Generic and System.IO)
- Your name properly documented as the author. A class description is not required.
- (There will be no instance variables or properties)
- Use the following code in method **main**. You will need to create the list where the comment indicates: This is a local variable in main (not an instance variable or property).

```
static void Main(string[] args)
{
        // Create an empty list named shapes that will hold Shape values.


        //GetShapes(shapes, "shapes.txt");
        GetShapes(shapes, "shapesWithBadData.txt");
        Console.WriteLine("Valid Shapes:");
        ShowShapes(shapes);
}
```

When wanting to also test invalid shapes, comment out the first call to getShapes and uncomment the second.

- Create a **static void** method named **GetShapes** that takes a list of Shape objects (i.e. List<Shape> shapes) as a parameter as well as a string respresenting a filename.
  - Declare a **StreamReader** variable named **inFile** and assign null.
  - Insert a try-catch that tries to open the filename parameter from the source code folder for this project. **A relative path name must be used (not an absolute path)**. Catch the **Exception** object, display only the message stored in the exception, and exit with a value of 1.
  - **while** there are more records in the file (this while loop goes after/below the catch)
    - **try** the following:
      - split the record just read into a data array.
      - **If** the first element in the data array is a "Rectangle" (to make case-insensitive, use a method such as String.Equals(str1, str2, StringComparison.OrdinalIgnoreCase))
        - Use the **Add** method for the shapes list to add a **new** Retangle with the color and the length and width. Remember to convert the length and width to doubles.
      - **else if** the first element in the data array is a "Circle" (to make case-insensitive, use a method such as String.Equals(str1, str2, StringComparison.OrdinalIgnoreCase))
        - Use the **Add** method for the shapes list to add a **new** Circle with the color and the radius

      FYI: There will be no else – Any other shape is just ignored and not added to the list
    - **catch** an **IllegalShapeException** object:
      - Display only the message stored in the object. (Note: The program does not exit here)
  - close the file
- Create a **static void** method named **ShowShapes** that takes a list of Shape objects as a parameter (i.e. List<Shape> shapes). There are no other parameters.
  - Use a **for** or **foreach** loop to loop through each shape in the shape objects list sent as the parameter. See the sample output after "Valid shapes: " in the sample runs. The body of this loop will
      - display the returned result of the shape's ToString() method
      - display the returned result from GetPerimeter() and GetArea(). The returned results from GetPerimeter() and GetArea() are to be displayed with 1 digit of precision and will have labels of units or sq. units.
      - There is a blank line also displayed.
  - Remember to match the exact output shown.

Other Notes

- Follow the details of the requirements and code according to those requirements.
- All coding guidelines must be followed; however, except for inserting your name as the author where needed, you are not required to insert additional documentation. **Remember to realign code as needed and split lines longer than 80 chars onto two or more lines.**

Sample Runs

Run 1 (shapes.txt file not found. Your path will differ)

```
Could not find file 'C:\Users\vango\source\repos\ShapeProject\shapes.txt'.
```

Run 2 (Using shapes.txt)

```
Valid Shapes:
Circle,Yellow,3.0
Perimeter: 18.8 units
Area: 28.3 sq. units

Circle,Red,4.5
Perimeter: 28.3 units
```

```
Area: 63.6 sq. units

Rectangle,Blue,3.0,4.0
Perimeter: 14.0 units
Area: 12.0 sq. units

Rectangle,Red,3.5,5.1
Perimeter: 17.2 units
Area: 17.9 sq. units

Circle,Light Green,2.5
Perimeter: 15.7 units
Area: 19.6 sq. units

Rectangle,Orange,4.0,4.0
Perimeter: 16.0 units
Area: 16.0 sq. units
```

Run 3 (Using shapesWithBadData.txt)

```
Illegal Shape: Circle radius of -3.0
Illegal Shape: Rectangle length of -3.5
Illegal Shape: Rectangle width of -4.0
Illegal Shape: Rectangle length of -2.0
Valid Shapes:
Circle,Red,2.5
Perimeter: 15.7 units
Area: 19.6 sq. units

Rectangle,Blue,3.5,4.5
Perimeter: 16.0 units
Area: 15.8 sq. units

Circle,Gray,2.1
Perimeter: 13.2 units
Area: 13.9 sq. units

Rectangle,Dark Purple,5.0,4.0
Perimeter: 18.0 units
Area: 20.0 sq. units
```

Submission

- Before due date/time: Upload all 5 source files that you created.