**Tutorial - A 2D-Shape Calculator (1/2)**

**OOP Design and development**

V1.0

28/03/2022

In the tutorial we are going design and develop a calculator for 2D-shapes. We are going to use this program as an example to walk through the steps of design and developing software using the OOP programming pattern.

* Step 1. Identify requirements
* Step 2. Design classes
* Step 3. Develop classes
* Step 4. Develop the main program

Below are the topics of this tutorial.

**Topics**

[**1.** **Step 1 – Identify software requirements** 2](#_Toc99403751)

[**2.** **Step 2 – Design Classes** 2](#_Toc99403752)

[**What is class notation** 2](#_Toc99403753)

[**How to represent class relationship** 2](#_Toc99403754)

[**Design class relationships** 3](#_Toc99403755)

[**Design class Shape** 3](#_Toc99403756)

[**Design class for each Shape** 4](#_Toc99403757)

[**Student activity 1** 4](#_Toc99403758)

[**Step summary** 5](#_Toc99403759)

[**(Optional) Challenge 1** 5](#_Toc99403760)

[**(Optional) Challenge 2** 5](#_Toc99403761)

[**3.** **Step 3 – Implement classes in C#** 6](#_Toc99403762)

[**Implement class Shape** 6](#_Toc99403763)

[**Implement class Rectangle** 6](#_Toc99403764)

[**Implement class Circle** 7](#_Toc99403765)

[**Test the classes** 8](#_Toc99403766)

[**Student activity 2** 8](#_Toc99403767)

[**Step summary** 9](#_Toc99403768)

[**(Optional) Challenge 3** 9](#_Toc99403769)

[**(Optional) Challenge 4** 9](#_Toc99403770)

[**4.** **Step 4 – Implement the UI** 9](#_Toc99403771)

[**Student activity 3** 11](#_Toc99403772)

[**Student activity 4** 11](#_Toc99403773)

[**Step summary** 11](#_Toc99403774)

[**(Optional) Challenge 5** 11](#_Toc99403775)

[**(Optional) Challenge 6** 11](#_Toc99403776)

[**Summary** 12](#_Toc99403777)

1. **Step 1 – Identify software requirements**

Here are the requirements for the “A 2D-Shape Calculator” program

|  |  |
| --- | --- |
| 1 | User is able to use a command line-based console interface to operate the calculator. |
| 2 | User is able to choose the target shape from a predefined list, including square, rectangle, circle, trapezoid. |
| 3 | User is able to choose what to calculate for each selected geometry shape from a predefined list, including area and perimeter. |
| 4 | (Optional) User is able to stay in the program and use the functions as long as it needed; and user is able to choose the “exit” menu item to quit the program. |

1. **Step 2 – Design Classes**

**What is class notation**

Use the following notation for class design and an example has been given.

Class Name

Field 1

Field 2

……

Method 1

Method 2

……

Vehicle

Make

Model

Year

……

ShiftGear(TargetGear)

Maintain(Task)

……

**How to represent class relationship**

Use the following diagram to represent inheritance relationships between base classes and sub-classes, and an example has been given.

Base class

Sub class

Vehicle

Car

Truck

Bus

Hatchback

Sedan

SUV

**Design class relationships**

In OOP, “[inheritance](https://www.w3schools.com/cs/cs_inheritance.php)” defines the “is a” relationship, e.g. a **car** is a **vehicle** or a **SUV** is a **car**. For the calculator program, we could design a “shape” base class and classes for each individual shape. Here is the class diagram.

Shape

Trapezoid

Square

Rectangle

Circle

Then we need to define the members of each class.

**Design class Shape**

For the Shape class, we leverage the “[abstract](https://www.w3schools.com/cs/cs_abstract.php)” method coding pattern to define the structure of the GetArea and GetPerimeter methods. Note a “abstract” method defines input/output information of a method without providing the implementation. The input/output information of a method include: 1) method name; 2) argument structure; 3) return type.

And Shape class has no member field.

abstract Shape

(No member field)

abstract double GetArea()

abstract double GetPerimeter()

**Design class for each Shape**

Rectangle, Square, Circle, Trapezoid classes each derives from the Shape class and each has to override and implement the GetArea() and GetPerimeter() methods.

Here are the designs for class Rectangle and Circle.

Rectangle

double Length

double Height

override double GetArea()

override double GetPerimeter()

Circle

double Radius

override double GetArea()

override double GetPerimeter()

**Student activity 1**

Design class Square and Trapezoid, insert your design in the box below.

Note: a Trapezoid include 5 parameters

* Base “a”
* Base “b”
* Side “c”
* Side “d”
* Height “h”

Chart, shape, rectangle

Description automatically generated

|  |
| --- |
|  |

**Step summary**

Now we have the design of all the classes. Next step is to use a programming language to implement it.

**(Optional) Challenge 1**

Implement a “Type” field for all shapes and the “Type” field indicate if the shape is a rectangle or a square or anything else.

**(Optional) Challenge 2**

A square is a special rectangle. Implement the following design. Note, you may need to use the concept of “virtual” method. Refer to the last example in [this tutorial](https://www.w3schools.com/cs/cs_polymorphism.php).

Shape

Trapezoid

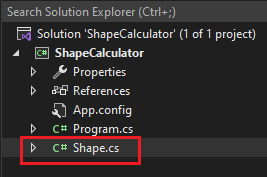
Square

Rectangle

Circle

1. **Step 3 – Implement classes in C#**

Based on the design, let’s do the coding. In Visual Studio 🡪 create a console application 🡪 create a class file “Shape.cs”



And then implement the classes according to the design

**Implement class Shape**

Text

Description automatically generated

**Implement class Rectangle**

Text

Description automatically generated

**Implement class Circle**

Text

Description automatically generated

**Test the classes**

Text

Description automatically generated

Check the result with any online calculator tools.

**Student activity 2**

Implement the class Square and Trapezoid, insert your code in the box below.

|  |
| --- |
|  |

**Step summary**

Now we have implemented the classes. Next step is build the console-based UI for the program.

**(Optional) Challenge 3**

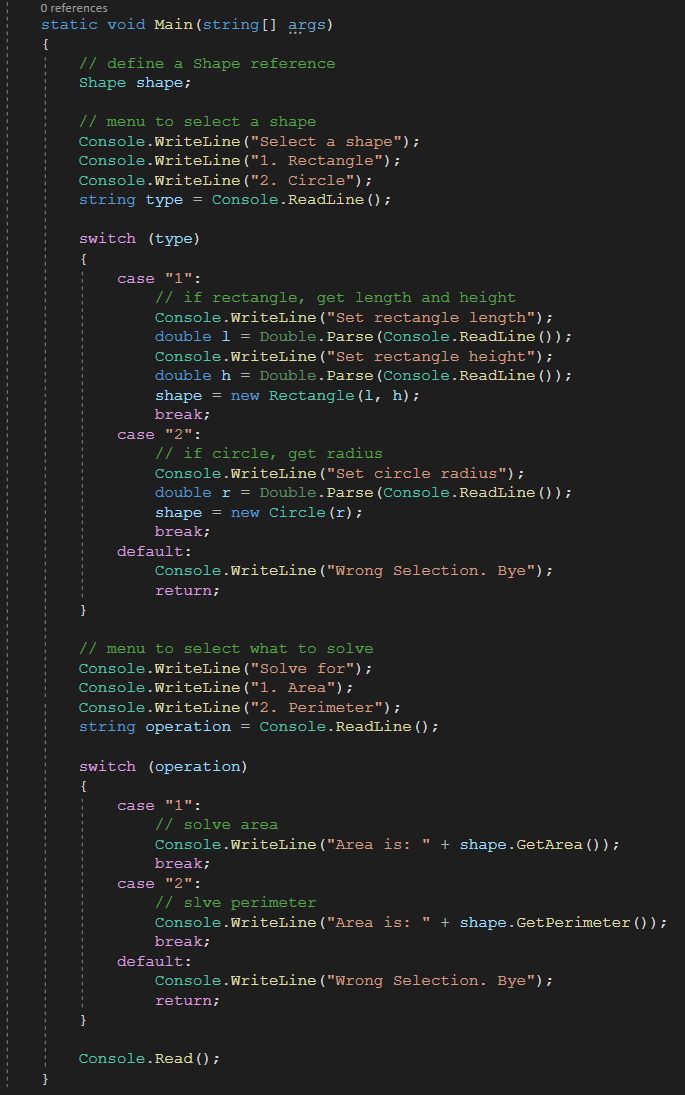
The design of Shape class above is has no fields and only one abstract method, which make the class “completely abstract”. Use “[Interface](https://www.w3schools.com/cs/cs_interface.php)”, instead of class, to define Shape

**(Optional) Challenge 4**

Use [C# Enums](https://www.w3schools.com/cs/cs_enums.php) to implement the “Type” field in Challenge 1 above.

1. **Step 4 – Implement the UI**

Implement a menu driven UI in the main method.



**Student activity 3**

Implement the UI for Square and Trapezoid, insert your code in the box below.

|  |
| --- |
|  |

**Student activity 4**

Fully test the calculator and record the results in the box blow.

|  |
| --- |
|  |

**Step summary**

Now we have competed the calculator.

**(Optional) Challenge 5**

Implement this requirement.

|  |  |
| --- | --- |
| 4 | (Optional) User is able to stay in the program and use the functions as long as it needed; and user is able to choose the “exit” menu item to quit the program. |

**(Optional) Challenge 6**

Detect illegal user inputs and handle the errors to improve the robustness of the program.

**Summary**

In this tutorial, we used the 2D shape calculator program as an example and walked through the general development steps. In the next tutorial - A 2D-Shape Calculator (2/2) – we will use this program to talk about software test.

End of this tutorial