Advanced Scripting   
Inheritance, Enumerations

Last Updated: 3/9/2024 5:31 PM Version 1  
Document Prepared for: CYBER360 Student

# Name Click here to enter name ID Click here to enter id

# Instructions

Save a copy of this document. Answer all questions directly in this document. You will save and upload this completed document as your homework submission.

# Overview

Now let’s extend the class we already coded in the previous exercise. We’ll make a new class for three-dimensional cylindrical columns, based on the Circle class, with a new property and new methods. We will also create an *enumerated* data type to specify the Color property in both classes.

# Requirements

* PowerShell
* VS Code
* Your **Methods.ps1** script file from the previous exercise.

# Setup

Create a new script file named **Oo.ps1**. Copy all of the contents from **Methods.ps1** into this new script.

In your **Oo.ps1** editor pane, hover your mouse pointer to the left of the first line, **Class Circle {** , and tap the downward pointing ▼ mark that appears there. This will collapse (hide) all of the rest of the Circle class code. The ▼ mark changes to rightward pointing ► when the code is collapsed. You should also notice the line numbers of the collapsed code are also hidden.

# Task 1—Create a new class to extend the capabilities of Circle class

Powershell uses the colon character **:** to denote *inheritance* from another class.

## Steps

1. Add the following code to your new script file:   
   A screen shot of a computer

   Description automatically generated   
   The colon **:** tells PowerShell to let a Column *inherit* all the members of a Circle.
2. Run your script.
3. Create a new instance of Column, and look at the results.   
   $c=[Column]::new()  
   $c
4. It should look exactly like your circle objects from the previous lab. Look at the instance members:   
   $c|get-member
5. Look at the static members:   
   $c|get-member -static

# Task 2—Add a property

## Steps

1. Add a new height property to your column class:   
   A screenshot of a computer

   Description automatically generated
2. Create an instance. Notice the new property.   
   [Column]::new()

# Task 3—Add a Method

Now let’s provide support for a **Volume** method. We’ll create a static method and an instance method.

## Steps

1. The volume of a column is the area of its circular cross-section multiplied by its height. So all we need to do is call the base class method to calculate the **Area**, then multiply it by **$Height**. For our static method we will need to pass two arguments, Diameter and Height. Add the following code.   
   A screen shot of a computer

   Description automatically generated
2. Use the new class to calculate the volume of a column:   
   **[Column]::Volume(2,2)**
   1. What is the Volume? Click or tap here to enter text.
3. Now add an instance method to calculate the Volume. Again, it will require no arguments because it will use the object’s Height and Diameter properties.
   1. Test it ‘till it works!

# Task 4—Add a constructor

Adding constructors to an inheriting class happens pretty much the same as with the base class. However, you may wish to call the base class’ constructor from your inherited class. PowerShell uses the colon character **:** and the keyword **Base** to call a base class constructor.

## Steps

1. Add a constructor that accepts both a Diameter and Height as parameters. Make it call the base constructor and pass the Diameter argument to it:   
    A screen shot of a computer program

   Description automatically generated
2. Create an instance of the class, passing in values for Diameter and Height:   
   [column]::new(4,10)
   1. Notice both the Diameter and Height have the respective specified values.

# Task 5—Create an Enumeration

Powershell uses the keyword **enum** to define an enumeration. *Enumerations* are simple data types that let scripters use a descriptive word to represent a fixed numeric value. Computers must represent *everything* using numbers, but humans work better with descriptive words; enumerations efficiently facilitate both points of view. Let’s create an enumeration for colors, then use it in our Circle class.

## Steps

1. Create the following enumeration at the top of your file, above the Circle class:   
   A computer screen shot of a program code

   Description automatically generated   
   (*Note*: its good practice to leave a space before and after each **=** in an **enum**’s definition.)
2. Run your code, then look at our new **colors** enum. There’s a static method **GetNames** for that:   
   [enum]::GetNames([colors])
3. Create a variable of type **colors** and set it to red:   
   [colors]$v='red'
4. Interrogate it:   
   $v
   1. What is the value of **$v**? Click or tap here to enter text.
5. Change $v to green:   
   $v='green'
6. See if you can convert the color to its corresponding integer value:   
   [int]$v
   1. Did it convert? Click or tap here to enter text. If yes, what is the value? If not, what happened instead? Click or tap here to enter text.
7. Set the value to yellow:   
   $v='yellow'
   1. Was it successful? Click or tap here to enter text. If yes, what is the value? If not, what happened instead? Click or tap here to enter text.
8. Set the value to 1:   
   $v=1   
   $v
   1. Was it successful? Click or tap here to enter text.
   2. What was returned? Click or tap here to enter text.
9. Experiment: are the enum’s words case sensitive? Click or tap here to enter text.
10. Constrain the datatype of the Color property in the Circle class to **[colors]**.   
    A screen shot of a computer

    Description automatically generated
11. Create a new instance of Column.
12. What is the color of your new Column object? Click or tap here to enter text.
13. Explain: how did it get that color value? Click or tap here to enter text.

# Wrap-up

Copy all of the contents of your **Oo.ps1** script here:

Click or tap here to enter text.

# Deliverable

Upload this document with completed answers to I-Learn Canvas.