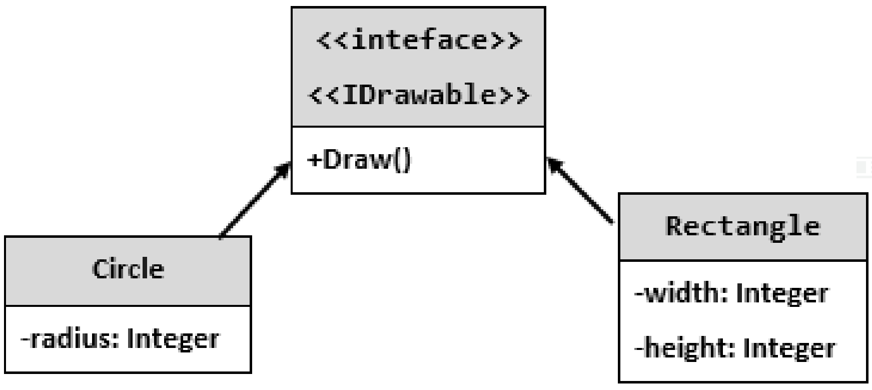
Lab: Basic Interfaces and Encapsulation

Shapes

NOTE: You need a public **`StartUp`** class with the namespace **`Shapes`**.

Build **hierarchy** of **interfaces** and **classes**:



You should be able to use the class like this:

StartUp.cs

```
var radius = int.Parse(Console.ReadLine());
IDrawable circle = new Circle(radius);

var width = int.Parse(Console.ReadLine());
var height = int.Parse(Console.ReadLine());
IDrawable rect = new Rectangle(width, height);

circle.Draw();
rect.Draw();
```

Examples

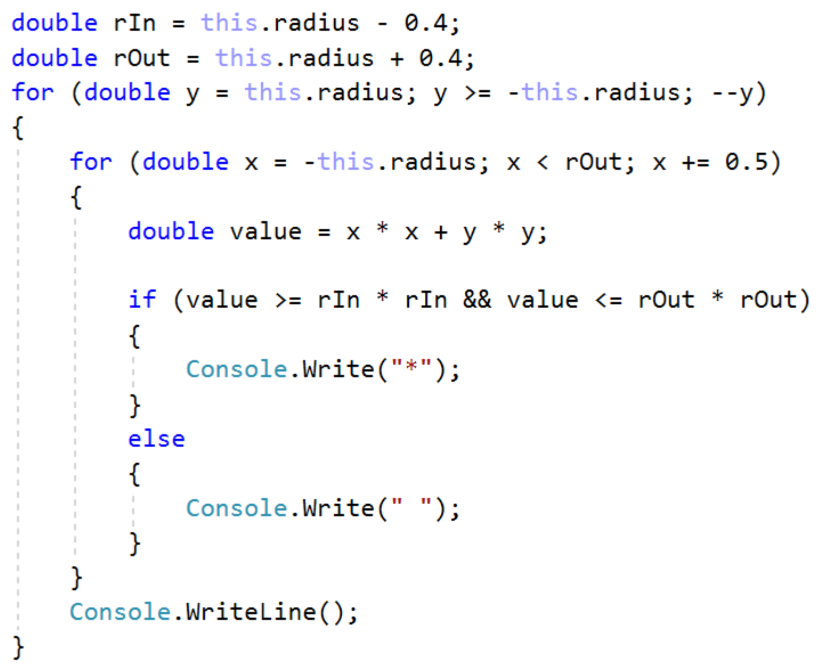
Input	Output
3 4 5	see below

```
Input
3
4
5
```

```
Output
   *******
 **       **
**         **
*           *
**         **
 **       **
   *******
****
*  *
*  *
*  *
****
```

Solution

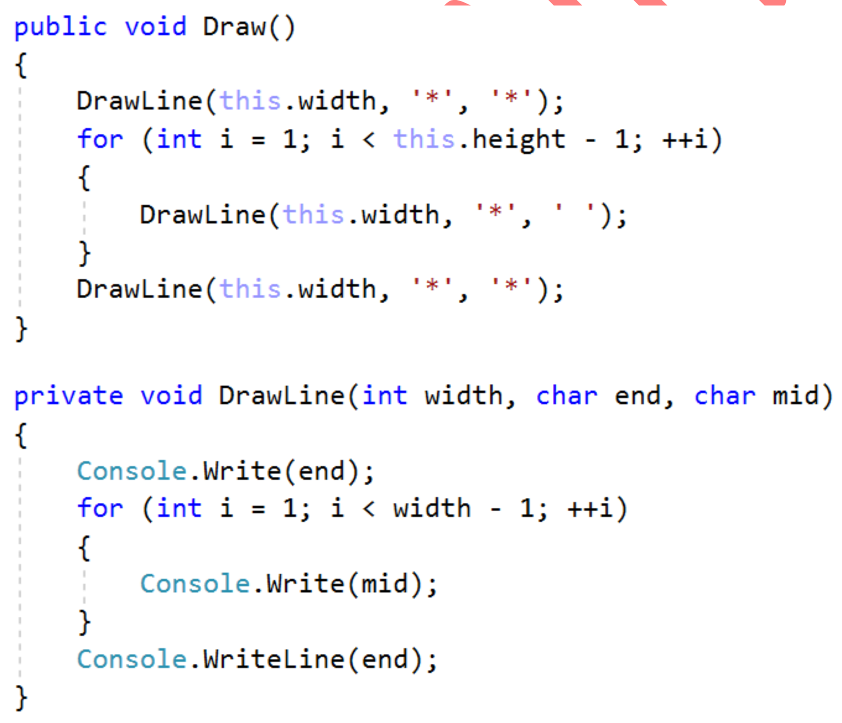
The algorithm for drawing a circle is:



```
double rIn = this.radius - 0.4;
double rOut = this.radius + 0.4;
for (double y = this.radius; y >= -this.radius; --y)
{
    for (double x = -this.radius; x < rOut; x += 0.5)
    {
        double value = x * x + y * y;

        if (value >= rIn * rIn && value <= rOut * rOut)
        {
            Console.Write("*");
        }
        else
        {
            Console.Write(" ");
        }
    }
    Console.WriteLine();
}
```

The algorithm for drawing a rectangle is:



```
public void Draw()
{
    DrawLine(this.width, '*', '*');
    for (int i = 1; i < this.height - 1; ++i)
    {
        DrawLine(this.width, '*', ' ');
    }
    DrawLine(this.width, '*', '*');
}

private void DrawLine(int width, char end, char mid)
{
    Console.Write(end);
    for (int i = 1; i < width - 1; ++i)
    {
        Console.Write(mid);
    }
    Console.WriteLine(end);
}
```

MathOperation

NOTE: You need a public **`StartUp`** class with the namespace **`Operations`**.

Create a class **MathOperations**, which should have 3 times method **`Add().`** Method **`Add()`** has to be invoked with:

- **`Add(int, int): int`**
- **`Add(double, double, double): double`**
- **`Add(decimal, decimal, decimal): decimal`**

You should be able to use the class like this:

StartUp.cs

`public static void Main()`
`{`
`    MathOperations mo = new MathOperations();`
`    Console.WriteLine(mo.Add(2, 3));`
`    Console.WriteLine(mo.Add(2.2, 3.3, 5.5));`
`    Console.WriteLine(mo.Add(2.2m, 3.3m, 4.4m));`
`}`

Examples

Output
5 11 9.9

Solution

Created MathOperation class should look like this:

```
public int Add(int a, int b)
{
    return a + b;
}

public double Add(double a, double b, double c)
{
    return a + b + c;
}

public decimal Add(decimal a, decimal b, decimal c)
{
    return a + b + c;
}
```

Animals

NOTE: You need a public **`StartUp`** class with the namespace **`Animals`**.

Create a class Animal, which holds two fields:

- name: string
- favouriteFood: string

Animal has one virtual method **`ExplainSelf(): string`.**
You should add two new classes - **Cat** and **Dog. Override** the **`ExplainSelf()`** method by adding concrete animal sound on a new line. (Look at examples below)

You should be able to use the class like this:

StartUp.cs

`Animal cat = new Cat("Pesho", "Whiskas");`
`Animal dog = new Dog("Gosho", "Meat");`
 	
`Console.WriteLine(cat.ExplainSelf());`
`Console.WriteLine(dog.ExplainSelf());`

Examples

Output
I am Pesho and my fovourite food is Whiskas MEEOW I am Gosho and my fovourite food is Meat DJAAF

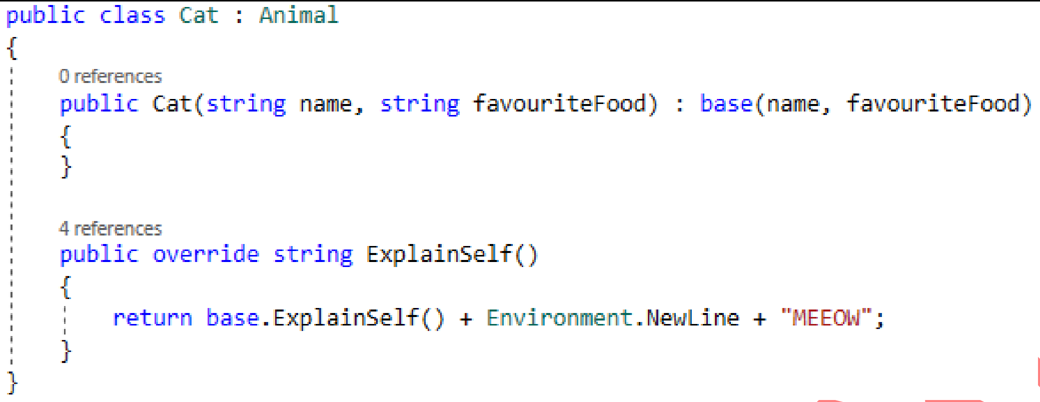
Solution

```
public class Animal
{
    2 references
    public string Name { get; protected set; }

    2 references
    public string FavouriteFood { get; protected set; }

    2 references
    protected Animal(string name, string favouriteFood)
    {
        this.Name = name;
        this.FavouriteFood = favouriteFood;
    }

    4 references
    public virtual string ExplainSelf()
    {
        return $"I am {this.Name} and my favourite food is {this.FavouriteFood}";
    }
}
```



```
public class Cat : Animal
{
    0 references
    public Cat(string name, string favouriteFood) : base(name, favouriteFood)
    {
    }

    4 references
    public override string ExplainSelf()
    {
        return base.ExplainSelf() + Environment.NewLine + "MEEOW";
    }
}
```

Shapes

NOTE: You need a public **`StartUp`** class with the namespace **`Shapes`**.

Create a class hierarchy, starting with **`abstract`** class **`Shape`**:

- **`Abstract` methods:**
  - **`CalculatePerimeter(): doulbe`**
  - **`CalculateArea(): double`**
- **`Virtual` methods:**
  - **`Draw(): string`**

Extend the **`Shape`** class with two children:

- **`Rectangle`**
- **`Circle`**

Each of them need to have:

- **Fields:**
  - **height and width for `Rectangle`**
  - **radius for `Circle`**
- **Encapsulation for these fields**
- **A public constructor**
- **Concrete methods for calculations (perimeter and area)**
- **Override methods for drawing**