



Polymorphism



Problem Scenario

Sometimes due to **different conditions**, we want our functions act differently.

This is called polymorphism.

Poly = many

Morphism = shapes (states)

Polymorphism: Types

We have discussed **two types** of Polymorphism.

1. Dynamic Polymorphism
2. Static Polymorphism

Dynamic Polymorphism

|| Problem Scenario: Extend the Code

We want to develop a system such it allows to create three types of shapes.

Rectangle: to represent a rectangle, width and height are required. Formula to find the area is $w * h$.

Square: to represent a square, a single side (s) is sufficient. Formula to find the area of square is $s * s = s^2$

Circle: to represent a radius (r) is enough. Formula to find the area of Circle is $2 * \pi * r^2$

Problem Scenario

1. System is required to save information of these three shapes.
2. In main method any type and any number of shape objects could be created.
3. All these objects should be created through the respective UI Class and should be added in to a **SINGLE COMMON LIST**. We do NOT want to create separate DLs for each type of shape.
4. When a program runs it shows all objects, object type (**type of shape**) and the **area of the shape**.

Problem Scenario: DriverProgram

```
class Program
{
    static void Main(string[] args)
    {
        Rectangle r = RectangleUI.createShape();
        ShapeDL.addToList(r);
        Circle c = CircleUI.createShape();
        ShapeDL.addToList(c);
        Square s = SquareUI.createShape();
        ShapeDL.addToList(s);
        Rectangle r1 = RectangleUI.createShape();
        ShapeDL.addToList(r1);
        Circle c1 = CircleUI.createShape();
        ShapeDL.addToList(c1);

        ShapeUI.showAreas(ShapeDL.getList());
        Console.ReadKey();
    }
}
```

```
Enter Width: 1
Enter Height: 1
Enter radius: 2
Enter Side: 4
Enter Width: 2
Enter Height: 2
Enter radius: 5
1.The shape is Rectangle and its area is 1
2.The shape is Circle and its area is 25.1327412287183
3.The shape is Square and its area is 16
4.The shape is Rectangle and its area is 4
5.The shape is Circle and its area is 157.07963267949
```

Problem Scenario: Hint 01

One thing is obvious, that we need three separate classes each representing the corresponding shape and all these classes will have following two methods.

- `getArea()`
- `getShapeType()`

Problem Scenario: Hint 01

Rectangle

- width:int
- height:int

- + getArea()
- + getShapeType()

Circle

- radius:int

- + getArea()
- + getShapeType()

Square

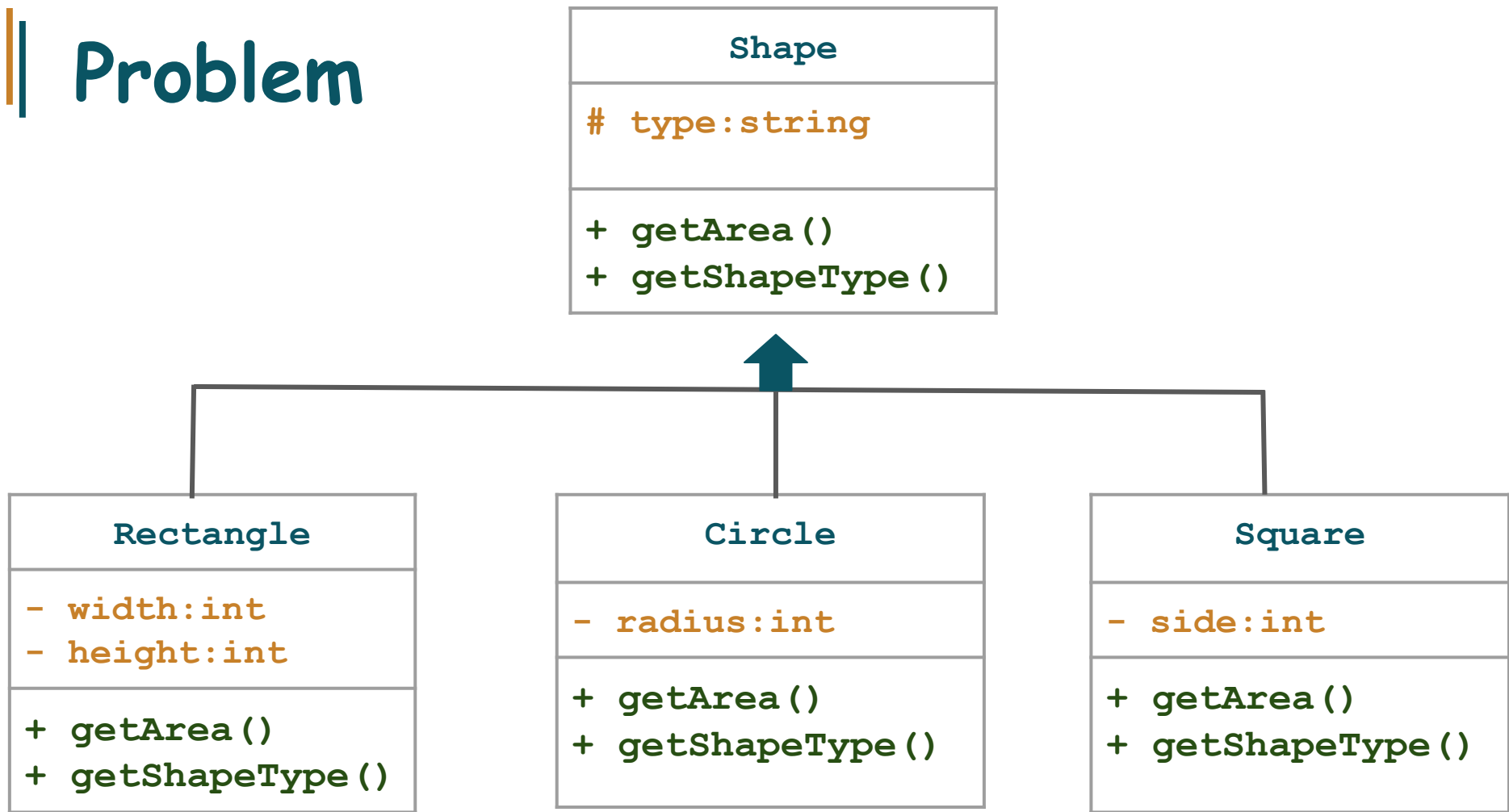
- side:int

- + getArea()
- + getShapeType()

Function Overriding

But for **single list** that contains different type of shapes we need to create a **general** (common) class and all classes should extend the parent class

Problem



Problem Scenario: Solution

We can create a parent class **Shape** with two methods.

```
class Shape
{
    public double getArea()
    {
        return 0;
    }

    public string getShapeType()
    {
        return "undefined.";
    }
}
```

Problem Scenario: CircleBL

```
class Circle:Shape
{
    private int radius;
    public Circle (int radius)
    {
        this.radius=radius;
    }
    public new double getArea()
    {
        double area;
        area=2*Math.Pow(radius,2)*Math.PI;
        return area ;
    }
    public new string getShapeType()
    {
        return "Circle";
    }
}
```

Problem Scenario: RectangleBL & SquareBL

```
class Rectangle:Shape
{
    private int width;
    private int height;
    public Rectangle(int width,int height)
    {
        this.width=width;
        this.height=height;
    }
    public new double getArea()
    {
        return width * height;
    }
    public new string getShapeType()
    {
        return "Rectangle";
    }
}
```

```
class Square:Shape
{
    private int side;
    public Square (int side)
    {
        this.side=side;
    }
    public new double getArea()
    {
        double area;
        area=Math.Pow(side,2);
        return area ;
    }
    public new string getShapeType()
    {
        return "Square";
    }
}
```

List of Shape Object

Now, we can easily create a **single list of shape** class that can contain all type of shapes but there is one problem with it.

Can you **highlight** the **problem** ?

List of Shape Object

Currently, only the functions of parent class are called.

```
class Program
{
    static void Main(string[] args)
    {
        Rectangle r = RectangleUI.createShape();
        ShapeDL.addIntoList(r);
        Circle c = CircleUI.createShape();
        ShapeDL.addIntoList(c);
        Square s = SquareUI.createShape();
        ShapeDL.addIntoList(s);
        Rectangle r1 = RectangleUI.createShape();
        ShapeDL.addIntoList(r1);
        Circle c1 = CircleUI.createShape();
        ShapeDL.addIntoList(c1);

        ShapeUI.showAreas(ShapeDL.getList());
        Console.ReadKey();
    }
}
```

```
Enter Width: 1
Enter Height: 1
Enter radius: 2
Enter Side: 4
Enter Width: 2
Enter Height: 2
Enter radius: 5
```

```
1.The shape is undefined. and its area is 0
2.The shape is undefined. and its area is 0
3.The shape is undefined. and its area is 0
4.The shape is undefined. and its area is 0
5.The shape is undefined. and its area is 0
```


What we want to Accomplish?

If we call `getArea()` and `getShapeType()` it should be called according to the respective child shape not the functions of the parent class.

Dynamic Behaviour

Can Parent Class could have **dynamic behaviour** at run time ?

Dynamic Behaviour

Can Parent Class could have **dynamic behaviour** at run time ?

Yes, Object Oriented Programming allows **Parent Class** to use the functionality of its **Child Class**.
This functionality will be assigned at **runtime**.

Problem Scenario: Extend the Code

To implement it in the code, we can use our previous knowledge of inheritance and Function Overriding.

Problem Scenario: Extend the Code

We can create following functions in parent class with **virtual** keyword and **override** these in corresponding child classes

1. `getArea()` and
2. `getShapeType()`.

Problem Scenario: Solution

We can create a parent class **Shape** with two methods.

```
class Shape
{
    public virtual double getArea()
    {
        return 0;
    }

    public virtual string getShapeType()
    {
        return "undefined.";
    }
}
```

Problem Scenario: CircleBL

```
class Circle:Shape
{
    private int radius;
    public Circle (int radius)
    {
        this.radius=radius;
    }
    public override double getArea()
    {
        double area;
        area=2*Math.Pow(radius,2)*Math.PI;
        return area ;
    }
    public override string getShapeType()
    {
        return "Circle";
    }
}
```

Problem Scenario: RectangleBL & SquareBL

```
class Rectangle:Shape
{
    private int width;
    private int height;
    public Rectangle(int width,int height)
    {
        this.width=width;
        this.height=height;
    }
    public override double  getArea()
    {
        return width * height;
    }
    public override string getShapeType()
    {
        return "Rectangle";
    }
}
```

```
class Square:Shape
{
    private int side;
    public Square (int side)
    {
        this.side=side;
    }
    public override double getArea()
    {
        double area;
        area=Math.Pow(side,2);
        return area ;
    }
    public override string getShapeType()
    {
        return "Square";
    }
}
```


Problem Scenario: ShapeDL

```
class ShapeDL
{
    private static List<Shape> shapesList = new List<Shape>();

    public static void addIntoList(Shape r)
    {
        shapesList.Add(r);
    }

    public static List<Shape> getList()
    {
        return shapesList;
    }
}
```

Problem Scenario: UI


```
class CircleUI
{
    public static Circle createShape()
    {
        Circle c;
        Console.Write("Enter radius: ");
        int radius = int.Parse(Console.ReadLine());
        c = new Circle(radius);
        return c;
    }
}
```

```
class SquareUI
{
    public static Square createShape()
    {
        Square s;
        Console.Write("Enter Side: ");
        int side = int.Parse(Console.ReadLine());
        s = new Square(side);
        return s;
    }
}
```

```
class RectangleUI
{
    public static Rectangle createShape()
    {
        Rectangle r;
        Console.Write("Enter Width: ");
        int width = int.Parse(Console.ReadLine());
        Console.Write("Enter Height: ");
        int height = int.Parse(Console.ReadLine());
        r = new Rectangle(width, height);
        return r;
    }
}
```

Problem Scenario: ShapeUI

```
class ShapeUI
{
    public static void showAreas(List<Shape> shapesList)
    {
        string message;
        for (int idx = 0; idx < shapesList.Count; idx++)
        {
            message = "The shape is {0} and its area is {1}";
            message = (idx + 1) + "." + message;
            Shape s = shapesList[idx];
            Console.WriteLine(message, s.getShapeType(), s.getArea());
        }
    }
}
```



Problem Scenario: ShapeUI

```
class ShapeUI
{
    public static void showAreas(List<Shape> shapesList)
    {
        string message;
        for (int idx = 0; idx < shapesList.Count; idx++)
        {
            message = "The shape is {0} and its area is {1}";
            message = (idx + 1) + "." + message;
            Shape s = shapesList[idx];
            Console.WriteLine(message, s.getShapeType(), s.getArea());
        }
    }
}
```

Problem Scenario: DriverProgram

```
class Program
{
    static void Main(string[] args)
    {
        Rectangle r = RectangleUI.createShape();
        ShapeDL.addIntoList(r);
        Circle c = CircleUI.createShape();
        ShapeDL.addIntoList(c);
        Square s = SquareUI.createShape();
        ShapeDL.addIntoList(s);
        Rectangle r1 = RectangleUI.createShape();
        ShapeDL.addIntoList(r1);
        Circle c1 = CircleUI.createShape();
        ShapeDL.addIntoList(c1);

        ShapeUI.showAreas(ShapeDL.getList());
        Console.ReadKey();
    }
}
```

Problem Scenario: DriverProgram

```
class Program
{
    static void Main(string[] args)
    {
        Rectangle r = RectangleUI.createShape();
        ShapeDL.addIntoList(r);
        Circle c = CircleUI.createShape();
        ShapeDL.addIntoList(c);
        Square s = SquareUI.createShape();
        ShapeDL.addIntoList(s);
        Rectangle r1 = RectangleUI.createShape();
        ShapeDL.addIntoList(r1);
        Circle c1 = CircleUI.createShape();
        ShapeDL.addIntoList(c1);

        ShapeUI.showAreas(ShapeDL.getList());
        Console.ReadKey();
    }
}
```

Problem Scenario: Output

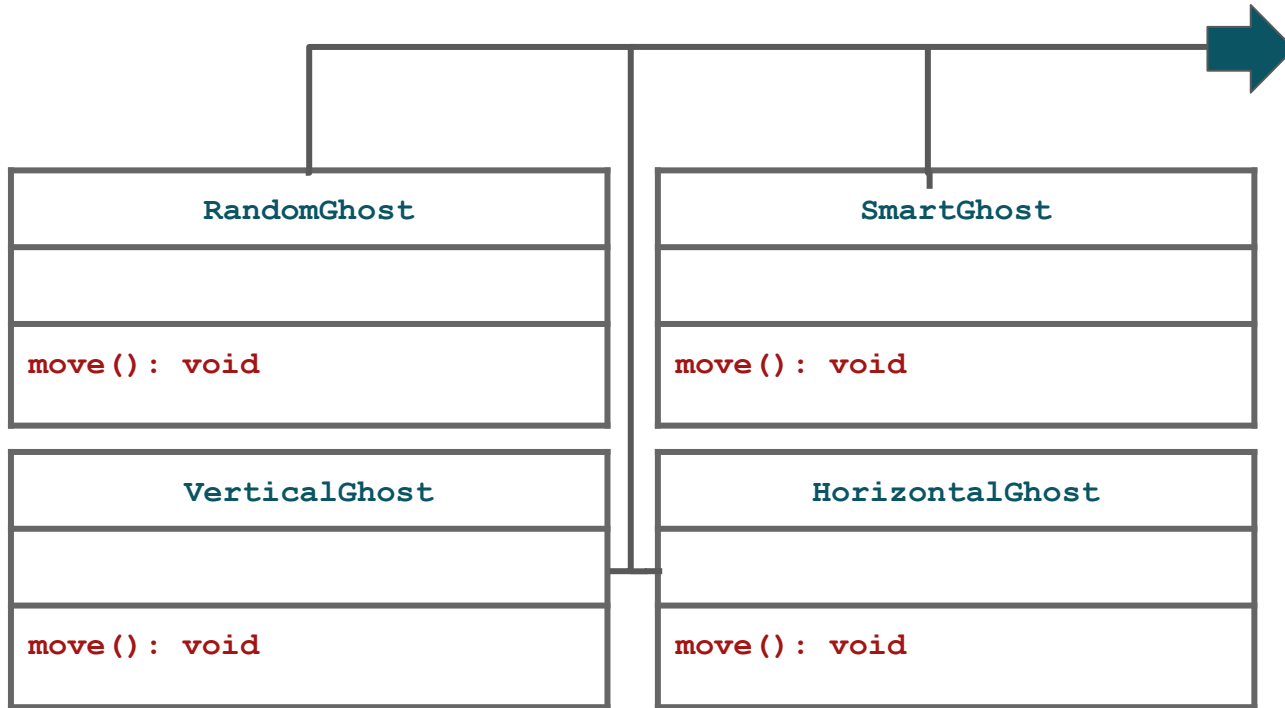
```
Enter Width: 1
Enter Height: 1
Enter radius: 2
Enter Side: 4
Enter Width: 2
Enter Height: 2
Enter radius: 5
1.The shape is Rectangle and its area is 1
2.The shape is Circle and its area is 25.1327412287183
3.The shape is Square and its area is 16
4.The shape is Rectangle and its area is 4
5.The shape is Circle and its area is 157.07963267949
```

Challenge

- List down the Scenario that can be improved your game code using dynamic polymorphism.
- Implementation Detail.



Example: Pacman



Ghost
<pre>X: int Y: int ghostDirection: string ghostCharacter: char previousItem: char</pre>
<pre>setDirection(ghostDirection: String): void getDirection(): string remove(): void draw(): void getCharacter(): char move(): void</pre>

Conclusion

- Object Oriented Programming offers us a way to extend the functionality of the parent class through function **overriding**.
- Function overriding is called **Dynamic Polymorphism**, because it decides at **run time** which function will be called.
- In function overriding, the **name and parameter list** of the function **should be same**.
- When we assign child class object to parent object and we want the functionality of parent object function is replaced with the child functionality, we add **virtual** keyword in the parent class and **override** keyword in child class.



Learning Objective

Implement **Dynamic Polymorphism** through Parent Child Relation and Overriding.

