



Abstraction



Problem Scenario

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 $w \times h$.

Triangle: to represent a Triangle, base (b) and height (h) are required. Formula to find the area of triangle is $(h \times b) / 2$

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

Problem Scenario

One way to implement this is through **Dynamic Polymorphism** that we have already discussed.

Problem Scenario

Every shape has its own attributes according to its definition.

Like **Circle** needs to save radius and **Triangle** needs to save base and height.

Therefore, a **separate class** for each type of shape will be created.

Problem Scenario

In addition, Every shape must have **two** functions:

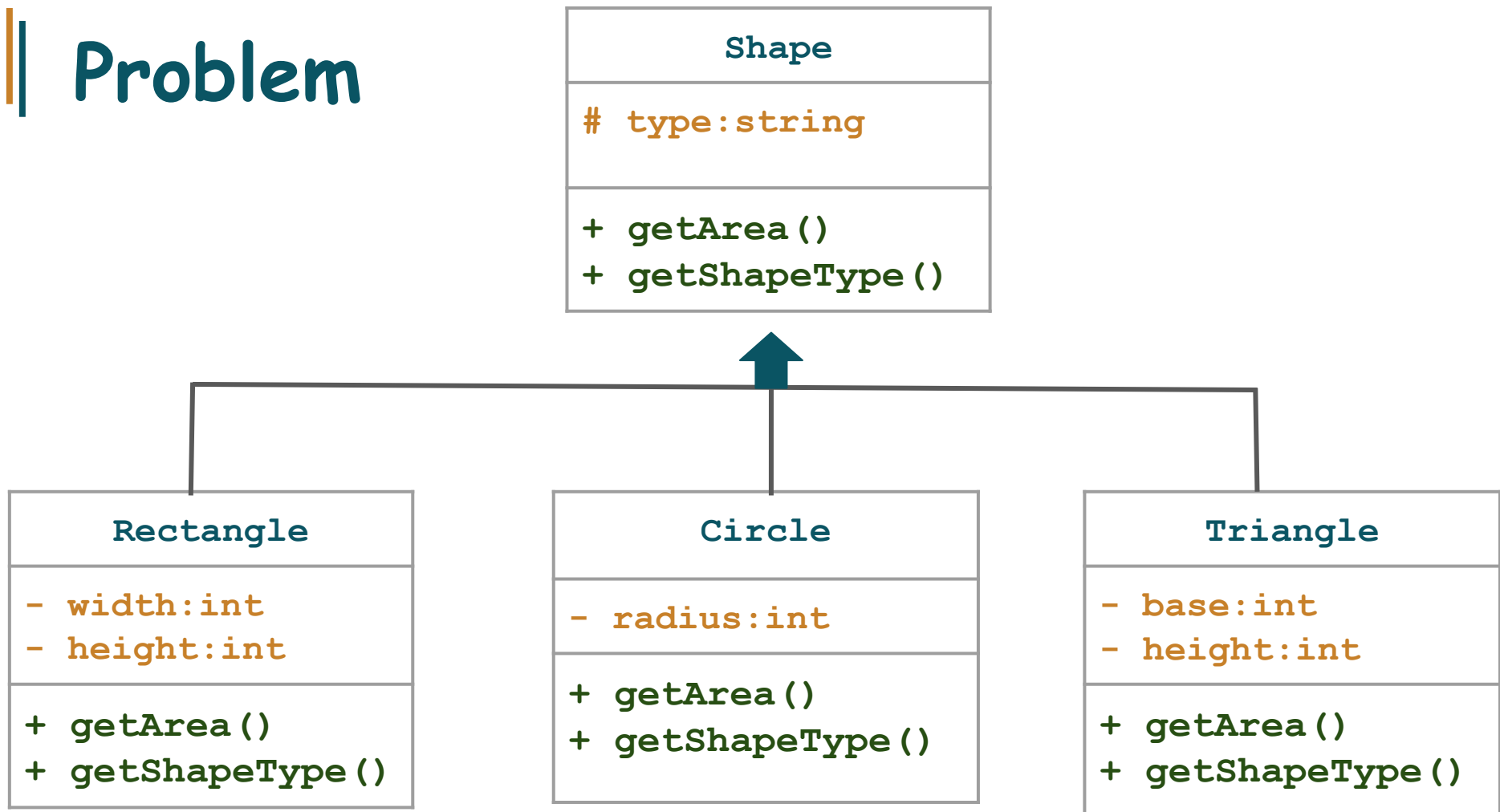
- `getArea()`
- `getShapeType()`.

Therefore, in order to maintain a single list and to obtain the **dynamic behaviour**, we can create a **parent class** with those two methods and all other classes **inherit** this class.

Problem Scenario

As child classes, have different definition for `getArea()` and `getShapeType()`, therefore all these children overrides the parent methods.

Problem



Problem

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

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

```
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";
    }
}
```


Problem with the Shape Class

Do you see any Issue with the Shape class ?

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

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

Problem with the Shape Class

Here we know `getArea()` and `getShapeType()` methods should be in Shape class but we can not give their definition because we do not know how they work.

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

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

Problem with the Shape Class

But to add the functions within the Shape class we have added the functions with some invalid implementation.

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

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

Problem with the Shape Class

We encounter such scenarios many times during the modeling of the real world problems.

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

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

Problem with the Shape Class

Sometimes we know there should be some function but we do know what it will do. Its implementation depends on the class that extends it.

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

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

Problem with the Child Class

Similarly, at the child end, how can we guarantee child implement the virtual function ?

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

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

```
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";
    }
}
```

| What we need.

We need some mechanism where we can define the functions that has only **header** (what shall be the input and output of function) but do **not have any body** and also it should be essential for children to provide definition of this method.

Abstraction

We need some mechanism where we can define the functions that has only **header** (what shall be the input and output of function) but **no body** and also it should be essential for children to provide definition of this method.

Such Mechanism in OOP is called **Abstraction**.

Abstraction

Abstraction means when something is an **idea** and does **not exist** in **concrete form** (an event) in current context.

In Code, We can declare a method **abstract** and leave its body for the child class to implement.

Abstraction

Abstraction means when something is an **idea** and does **not exist** in **concrete form** (an event) in current context.

In Code, We can declare a method **abstract** and leave its body for the child class to implement.

Here We have

```
public abstract double getArea();
```

Abstraction

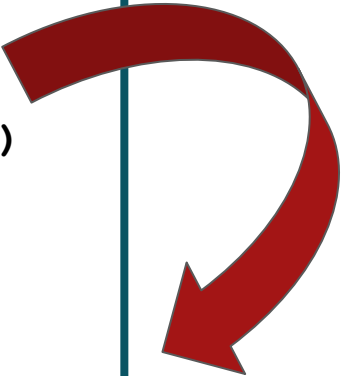
However, when declare an **abstract** method in a **class** we get an error message.

```
class Shape {  
    public abstract double getArea();  
    public virtual string getShapeType()  
    {  
        return "undefined.";  
    }  
}
```

Abstraction

However, when declare an **abstract** method in a **class** we get an error message.

```
class Shape {  
    public abstract double getArea();  
    public virtual string getShapeType()  
    {  
        return "undefined.";  
    }  
}
```



✓ C# Program.cs

2

✗ 'Shape.getArea()' is abstract but it is contained in non-abstract type '...' (CS0513) [8, 32]

✗ 'Shape.getArea()' is abstract but it is contained in non-abstract... csharp(CS0513) [8, 32]

Abstraction

If a class contains an **abstract** method, **class** should be **declared as abstract** too.

```
abstract class Shape {  
    public abstract double getArea();  
    public virtual string getShapeType()  
    {  
        return "undefined.";  
    }  
}
```

Abstraction

If a class contains an **abstract** method, **class** should be **declared as abstract** too.



```
abstract class Shape {  
    public abstract double getArea();  
    public virtual string getShapeType()  
    {  
        return "undefined.";  
    }  
}
```

Abstraction

Objects of abstract class can not be created because its definition is not complete.



```
abstract class Shape {  
    public abstract double getArea();  
    public virtual string getShapeType()  
    {  
        return "undefined.";  
    }  
}
```

Abstraction

Abstract class instance can be created through it child class.



```
abstract class Shape {  
    public abstract double getArea();  
    public virtual string getShapeType()  
    {  
        return "undefined.";  
    }  
}
```


Abstraction

It is must for child class that **inherits abstract class** to provide definition of all abstract methods of the parent class.



```
abstract class Shape {  
    public abstract double getArea();  
    public virtual string getShapeType()  
    {  
        return "undefined.";  
    }  
}
```

Abstraction

It is must for child class that **inherits abstract class** to provide definition of all abstract methods of the parent class.

```
abstract class Shape {  
    public abstract double getArea();  
    public virtual string getShapeType()  
    {  
        return "undefined.";  
    }  
}
```

```
class Rectangle: Shape  
{  
    private int width;  
    private int height;  
    public override double getArea()  
    {  
        return this.width * height;  
    }  
    //Other part of code  
}
```

| Abstraction

If the child class that **inherits** the abstract class, **does not provide** the definition of all abstract methods of the Parent Class, then the child class will also be an **Abstract Class**

Abstraction: Advantage?

Then what is the advantage of declaring the **abstract methods**, when we were already doing same work with **virtual methods** ?

```
abstract class Shape {  
    public abstract double getArea();  
    public virtual string getShapeType()  
    {  
        return "undefined.";  
    }  
}
```

```
class Rectangle: Shape  
{  
    private int width;  
    private int height;  
    public override double getArea()  
    {  
        return this.width * height;  
    }  
    //Other part of code  
}
```

Abstraction: Advantage

Abstraction applies more strict conditions on the child classes that they must give the definition of the abstract method or otherwise they become abstract as well.

Abstraction: Advantage

It is philosophical more sound that if class can not provide some of its implementation then it should be abstract and also its object can not be created.

Conclusion

- An **abstract class** is a class that is declared abstract.
- **Abstraction** means when something is an idea and not exist in concrete form (an event) in current context.
- Abstract classes **cannot be instantiated**, they can only be inherited.
- When an abstract class is inherited, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the **subclass must also be declared abstract**.



Learning Objective

Implement **Abstraction** in **Parent Class** when it is known **WHAT** should be a functionality but do not know **HOW** it will be achieved.



Self Assessment: Output?

```
abstract class Bank
{
    public virtual void issueCard()
    {
        Console.WriteLine("Card is issued");
    }
    public abstract int getInterestRate();
}
```

```
class HBL : Bank
{
    public override int getInterestRate()
    { return 7; }
}
```

```
public static void main(string[] args)
{
    Bank b;
    b = new HBL();
    Console.WriteLine("HBL Rate of Interest is: " + b.getInterestRate() + "%");
    b.issueCard();
}
```

Self Assessment: Output?

```
abstract class Bank
{
    public virtual void issueCard()
    {
        Console.WriteLine("Card is issued");
    }
    public abstract int getInterestRate();
}
```

```
class MCB : Bank
{
    public override void issueCard()
    {
        Console.WriteLine("MCB Card is issued");
    }
}
```

```
public static void main(string[] args)
{
    Bank b;
    b = new MCB();
    Console.WriteLine("MCB Rate of Interest is: " + b.getInterestRate() + "%");
    b.issueCard();
}
```