What is Polymorphism?

In this lesson, we will be learning about the basics of polymorphism with the implementation details.

WE'LL COVER THE FOLLOWING

- Definition
- Shape Class
  - Implementation
- Rectangle Class
  - Implementation
- Circle Class
  - Implementation
- Explanation of Code

> The word **Polymorphism** is a combination of two Greek words, **Poly** means *many* and **Morph** means *forms*.

Definition

When we say *polymorphism* in programming that means something which exhibits many forms or behaviors. So far, we have learned that we can add new data and functions to a class through inheritance. But what about if we want our derived class to inherit a method from the base class and have a different implementation for it? That is when polymorphism comes in, a fundamental concept in OOP paradigm.

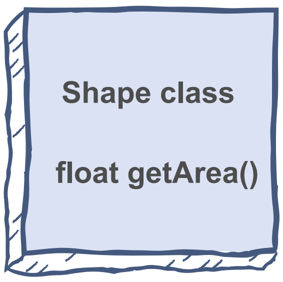
Shape Class

We are considering here the example of **Shape** class, which is base class for many shapes like *Rectangle and Circle*. This class contains a function

getArea() which calculates the area for the *derived* classes.

Implementation

Let's look at the implementation of **Shape** class:

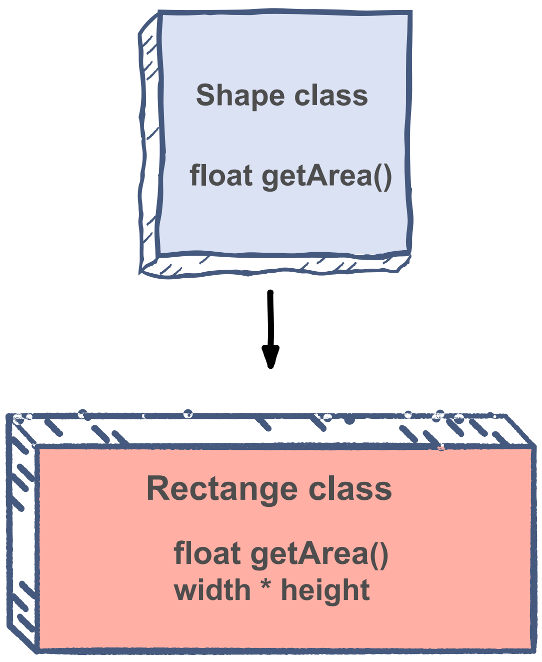


Shape class

```
// A simple Shape interface which provides a method to get the Shape's area
class Shape {
  public:
  float getArea(){}
};
```

Rectangle Class

Consider the **Rectangle** class which is derived from *Shape* class. It has two data members, i.e., *width* and *height* and it returns the *Area* of the rectangle by using **getArea()** function.



Rectangle class

Implementation

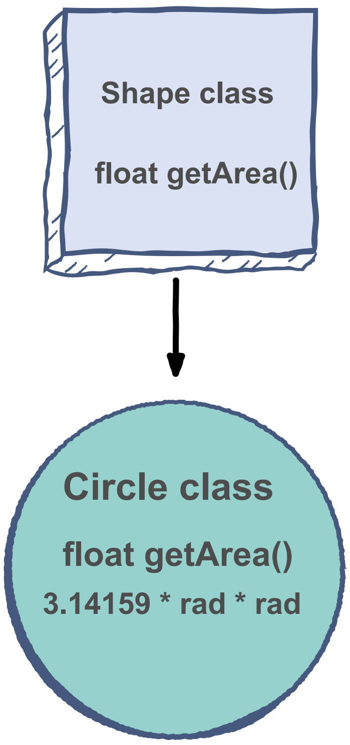
Let’s look at the implementation of the **Rectangle** class:

```
// A Rectangle is a Shape with a specific width and height
class Rectangle : public Shape {   // derived form Shape class
  private:
  float width;
  float height;

  public:
  Rectangle(float wid, float heigh) {
    width = wid;
    height = heigh;
  }
  float getArea(){
    return width * height;
  }
};
```

Circle Class

Consider the **Circle** class which is derived from *Shape* class. It has one data member, i.e., *radius* and it returns the *Area* of the circle by using **getArea()** function.



Circle class

Implementation

Let’s look at the implementation of the **Circle** class:

```
// A Circle is a Shape with a specific radius
class Circle : public Shape {
  private:
  float radius;

  public:
  Circle(float rad){
    radius = rad;
  }
  float getArea(){
    return 3.14159f * radius * radius;
  }
};
```

Now, if we merge all the classes then by calling the **getArea()** function, let’s see what happened:

```
#include <iostream>
using namespace std;

// A simple Shape interface which provides a method to get the Shape's area
```

```
class Shape {
  public:
  float getArea(){}

};

// A Rectangle is a Shape with a specific width and height
class Rectangle : public Shape {   // derived form Shape class
  private:
  float width;
  float height;

  public:
  Rectangle(float wid, float heigh) {
    width = wid;
    height = heigh;
  }
  float getArea(){
    return width * height;
  }
};

// A Circle is a Shape with a specific radius
class Circle : public Shape {
  private:
  float radius;

  public:
  Circle(float rad){
    radius = rad;
  }
  float getArea(){
    return 3.14159f * radius * radius;
  }
};

int main() {
  Rectangle r(2, 6);    // Creating Rectangle object

  Shape* shape = &r;   // Referencing Shape class to Rectangle object

  cout << "Calling Rectangle getArea function: " << r.getArea() << endl;      // Calls Rectar
  cout << "Calling Rectangle from shape pointer: " <<  shape->getArea() << endl <<endl; // Ca

  Circle c(5);    // Creating Circle object

  shape = &c;   // Referencing Shape class to Circle object

  cout << "Calling Circle getArea function: " << c.getArea() << endl;
  cout << "Calling Circle from shape pointer: " <<shape->getArea() << endl << endl;
}
```

Explanation of Code

Polymorphism only works with a pointer and reference types, so we have created a **Shape** pointer, and pointed to the *derived* class objects. But due to multiple existences of the same functions in classes, it will get confused

between which class **getArea()** function it's calling. The derived classes function has a different implementation but the same name and that's why we are not getting the expected output.

In the next lesson, we'll be learning about the fundamental concept of **overriding**.