Object-Oriented Programming

Student's Name

Department, Institutional Affiliation

Course Name and Number

Professor's Name

Due Date

Object-Oriented Programming

Object-Oriented Programming (OOP) is a way of structuring and designing code that emphasizes creating reusable and modular components called “objects”. An object contains both data, which represents its attributes, and operations, which are the methods that manipulate that data.

Core Principles of OOP

Encapsulation

Encapsulation is a key concept in Object-Oriented Programming (OOP) that plays a fundamental role in organizing code. It revolves around grouping data and its associated methods within a class. One of the benefits of encapsulation is that it enables data hiding, which means that the internal state of an object is protected from direct external access.

Inheritance

Inheritance is a powerful concept in programming that enables the creation of new classes by building upon existing ones. It works by establishing a relationship between a base class and a derived class. The derived class inherits the properties and behaviors defined in the base class, which allows for code reuse and specialization.

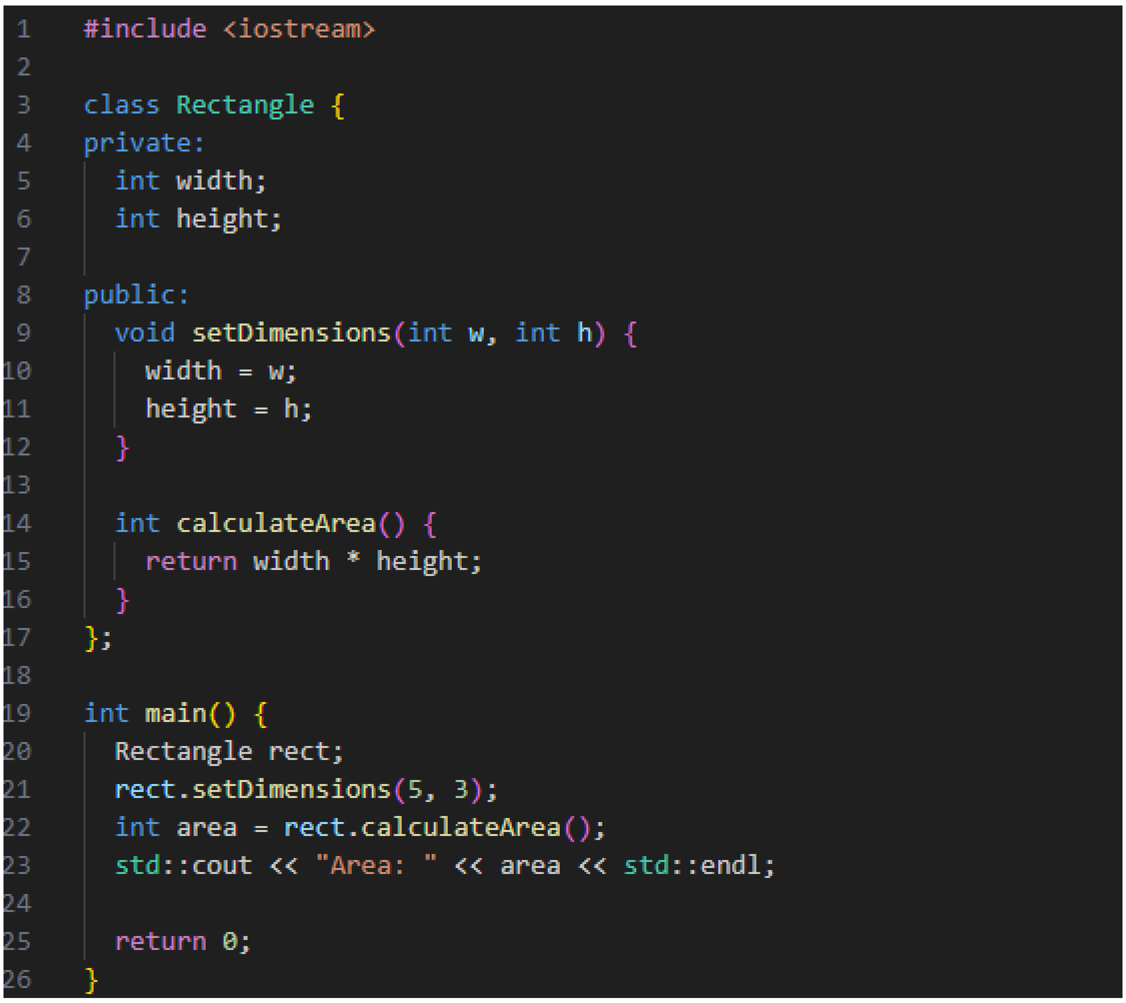
Polymorphism

Polymorphism is a concept in programming that allows objects to exhibit different behaviors or forms depending on the situation they are in. On the other hand, function overriding involves redefining a method in a derived class that already exists in its base class. This enables the derived class to provide its own implementation of the method, allowing for customized behavior while maintaining a consistent interface.

Implementation in C++

Figure 1

Below is an example of a class definition for a "Rectangle" object:

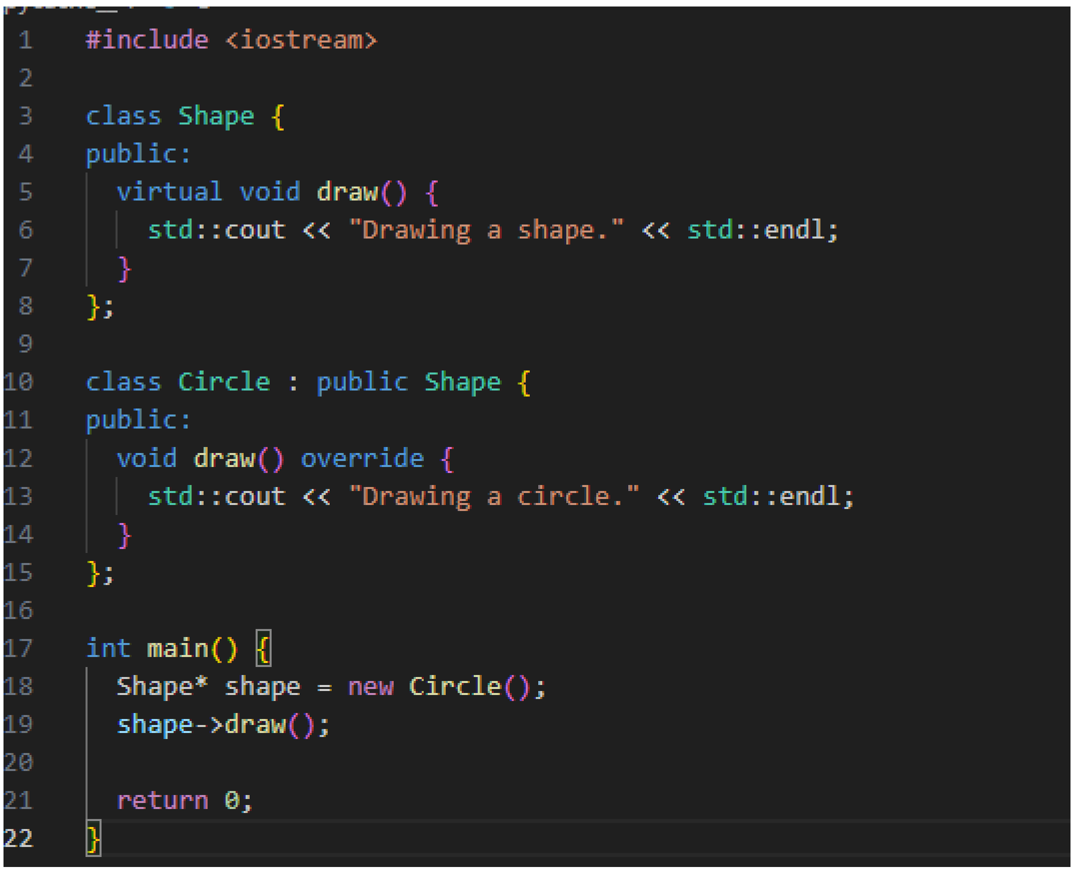


```cpp
#include <iostream>

class Rectangle {
private:
    int width;
    int height;

public:
    void setDimensions(int w, int h) {
        width = w;
        height = h;
    }

    int calculateArea() {
        return width * height;
    }
};

int main() {
    Rectangle rect;
    rect.setDimensions(5, 3);
    int area = rect.calculateArea();
    std::cout << "Area: " << area << std::endl;

    return 0;
}
```

In the example above, the "width" and "height" attributes of the Rectangle class are encapsulated and made private. Access to these attributes is provided through public methods, such as "setDimensions()" and "calculateArea()".

Figure 2

In the figure below, the "Shape" class is the base class, and the "Circle" class is derived from it.



```cpp
#include <iostream>

class Shape {
public:
    virtual void draw() {
        std::cout << "Drawing a shape." << std::endl;
    }
};

class Circle : public Shape {
public:
    void draw() override {
        std::cout << "Drawing a circle." << std::endl;
    }
};

int main() {
    Shape* shape = new Circle();
    shape->draw();

    return 0;
}
```

The "draw()" method is overridden in the derived class, allowing for polymorphic behavior.