Task 5: Abstract Class and Pure Virtual Functions (C++)

Objective:
The objective of this task is to understand and implement abstract classes and pure virtual functions in C++. This task demonstrates how abstraction enforces derived classes to provide their own implementation of essential behaviors.

Problem Statement

Enhance the Shape class to demonstrate abstraction by performing the following:
- Modify the draw() function to be a pure virtual function
- Ensure that all derived classes implement the draw() function

Concepts Used

- Abstract Class
- Pure Virtual Function
- Runtime Polymorphism
- Inheritance
- Function Overriding

Source Code (C++)

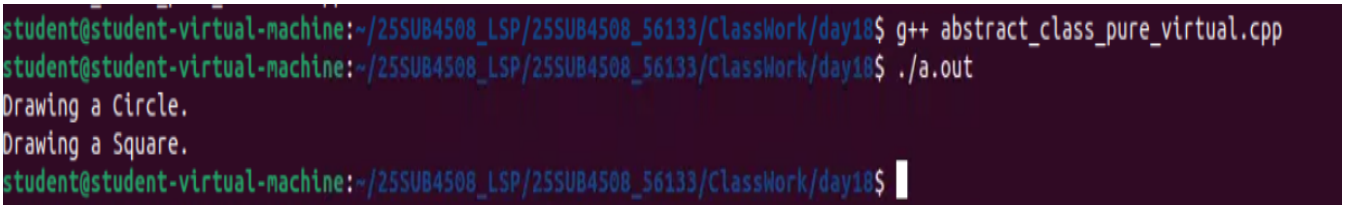
```
#include <iostream>
using namespace std;

// Abstract base class
class Shape {
public:
        // Pure virtual function
        virtual void draw() = 0;
};

// Derived class Circle
class Circle : public Shape {
public:
```

```
        void draw() override {
        cout << "Drawing a Circle." << endl;
        }
};

// Derived class Square
class Square : public Shape {
public:
        void draw() override {
        cout << "Drawing a Square." << endl;
        }
};

int main() {
        Shape* shape1;
        Shape* shape2;

        Circle c;
        Square s;

        shape1 = &c;
        shape2 = &s;

        shape1->draw();
        shape2->draw();

        return 0;
}
```

Explanation of the Code

1. The Shape class is declared as an abstract class by defining a pure virtual function draw().
2. The pure virtual function is declared using '= 0', which means it has no implementation in Shape.
3. Because of the pure virtual function, objects of Shape cannot be created.
4. Circle and Square classes inherit from Shape.
5. Both derived classes provide their own implementation of the draw() function.
6. Base class pointers are used to achieve runtime polymorphism.
7. The correct draw() method is invoked based on the actual object type.

Program Output



Conclusion

This task demonstrates abstraction using abstract classes and pure virtual functions in C++. By enforcing derived classes to implement essential functions, abstraction ensures a consistent interface while allowing flexible and specific behavior in child classes.