Week # 9

Abstract Classes in C++

Objective:
Understand and implement the concept of Abstract Classes in object-oriented programming (OOCP).

Introduction:

Abstract classes in Object-Oriented Programming (OOP) are classes that cannot be instantiated on their own and are designed to be inherited by other classes. They serve as a blueprint for other classes, providing a template for them to follow. An abstract class typically includes one or more abstract methods, which are methods declared without any implementation. Derived classes that inherit from an abstract class must provide implementations for these abstract methods. The primary purpose of abstract classes is to define a common interface for all derived classes while allowing the implementation details to be handled by the subclasses. This promotes code reusability and ensures that certain methods are implemented in all derived classes.

Key Points:

Cannot Instantiate: You cannot create an object of an abstract class.

Abstract Methods: These are methods declared without implementation. Derived classes must implement these methods.

Concrete Methods: Abstract classes can also have concrete methods with implementation.

Purpose: Abstract classes are used to define a template for other classes, enforcing a contract for subclasses to follow.

Pure Virtual Functions:

A pure virtual function in C++ is a virtual function that has no implementation in the base class and must be overridden by derived classes. It is declared by assigning = 0 at the end of the function declaration. Classes containing one or more pure virtual functions are considered abstract classes and cannot be instantiated.

The primary purpose of pure virtual functions is to define an interface that must be implemented by derived classes, enforcing a contract that specific functions will be provided.

Key Points:

Declaration: A pure virtual function is declared with = 0 at the end of the function declaration.

Abstract Classes: A class containing at least one pure virtual function is abstract and cannot be instantiated.

Implementation in Derived Classes: Derived classes must provide implementations for all pure virtual functions from the base class.

Polymorphism: Pure virtual functions enable polymorphic behavior, allowing for runtime method binding.

```
#include <iostream>

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

    // Concrete method
    void info() const {
        std::cout << "This is a shape." << std::endl;
    }

    // Virtual destructor
    virtual ~Shape() {}
};

// Derived class Circle
class Circle : public Shape {
public:
    // Implementing the pure virtual function
    void draw() const override {
        std::cout << "Drawing a Circle." << std::endl;
    }
};

// Derived class Square
class Square : public Shape {
public:
    // Implementing the pure virtual function
    void draw() const override {
```

```
        std::cout << "Drawing a Square." << std::endl;
    }
};

int main() {
    // Shape shape; // This line would cause a compilation error because Shape is abstract

    // Create instances of derived classes
    Shape* circle = new Circle();
    Shape* square = new Square();

    // Call the draw method on each derived class object
    circle->draw(); // Outputs: Drawing a Circle.
    square->draw(); // Outputs: Drawing a Square.

    // Call the info method from the base class
    circle->info(); // Outputs: This is a shape.
    square->info(); // Outputs: This is a shape.

    // Clean up
    delete circle;
    delete square;

    return 0;
}
```

Explanation:

Abstract Base Class Shape:
Contains a pure virtual function draw(), making it an abstract class. This function must be implemented by all derived classes.
Includes a concrete method info() with a defined implementation that can be used by all derived classes.
Provides a virtual destructor to ensure proper cleanup of derived class objects.

Derived Class Circle:
Inherits from Shape.
Implements the pure virtual function draw() with the override keyword. This provides the specific behavior for drawing a circle.

Derived Class Square:
Inherits from Shape.
Implements the pure virtual function draw() with the override keyword. This provides the specific behavior for drawing a square.

Main Function:

Demonstrates that attempting to instantiate a Shape object directly would result in a compilation error because Shape is abstract.
Creates objects of Circle and Square, assigns them to variables of type Shape, and calls their draw() methods, demonstrating polymorphism.
Calls the info() method, which is a concrete method from the base class, on both Circle and Square objects, demonstrating that derived classes can use concrete methods from the base class.

Summary:

A pure virtual function in C++ is a way to enforce that derived classes provide specific functionality. By defining an interface with pure virtual functions in an abstract base class, you ensure that any class derived from it implements these functions, thereby enabling polymorphic behavior.

Exercise:

Question 1: Shape Area Calculation

Problem Statement:
Create an abstract class Shape with a pure virtual function area(). Derive two classes Circle and Rectangle from Shape. Implement the area() function in both derived classes. Write a program to calculate the area of a circle and a rectangle.

Question 2: Employee Salary Calculation

Problem Statement:
Create an abstract class Employee with a pure virtual function calculateSalary(). Derive two classes PermanentEmployee and ContractEmployee from Employee. Implement the calculateSalary() function in both derived classes. Write a program to calculate the salary of a permanent employee and a contract employee.

Question 3: Vehicle Fuel Efficiency

Problem Statement:
Create an abstract class Vehicle with a pure virtual function fuelEfficiency(). Derive two classes Car and Truck from Vehicle. Implement the fuelEfficiency() function in both derived classes. Write a program to calculate the fuel efficiency of a car and a truck.

Question 4: Payment Processing System

Problem Statement:
Create an abstract class Payment with a pure virtual function processPayment(). Derive two classes CreditCardPayment and PaypalPayment from Payment. Implement the processPayment() function in both derived classes. Write a program to process a payment through credit card and PayPal.

Question 5: Animal Sound Simulation

Problem Statement:
Create an abstract class Animal with a pure virtual function makeSound(). Derive two classes Dog and Cat from Animal. Implement the makeSound() function in both derived classes. Write a program to simulate the sounds of a dog and a cat.