CS224 Object Oriented Programming and Design Methodologies

Lab 09 - Abstract Classes

Department of Computer Science
Dhanani School of Science and Engineering
Habib University

Fall 2025

Contents

1 Introduction

1.1 Guidelines

1. Use of AI for code generation or debugging is strictly prohibited. Violations will result in a grade of 0 and potential disciplinary action.
2. Attendance in lab is mandatory. Absence results in automatic 0.
3. Submit your lab by the deadline on Canvas. Late submissions are not accepted.

1.2 Objectives

- Understand abstract classes and pure virtual functions in C++.
- Practice implementing polymorphism through abstract base classes.
- Learn to design a class hierarchy that uses dynamic binding.

1.3 Directory Structure

Each lab follows the same structure:

```
lab/
  manual/
     lab.pdf
  src/
     *.hpp
     *.cpp
  tests/ (optional)
     *.cpp
  makefile (optional)
```

2 Abstract Classes

An abstract class in C++ serves as a blueprint for derived classes. It cannot be instantiated directly and contains at least one pure virtual function. This concept supports polymorphism by ensuring all derived classes implement specific functionality in their own way.

3 Code Example

```cpp
#include <iostream>
#include <string>
#include <vector>
using namespace std;

// Abstract base class
class TradingCard {
```

```
protected:
    string name;
    int hp; // Hit Points

public:
    TradingCard(string n, int h) : name(n), hp(h) {}

    // Pure virtual function
    virtual void attack() = 0;

    // Common function for all cards
    void showInfo() {
        cout << "Card: " << name << " | HP: " << hp << endl;
    }

    // Virtual destructor (important for polymorphism)
    virtual ~TradingCard() = default;
};

// Derived class 1
class PokemonCard : public TradingCard {
public:
    PokemonCard(string n, int h) : TradingCard(n, h) {}

    void attack() override {
        cout << name << " used Thunderbolt!" << endl;
    }
};

// Derived class 2
class YugiohCard : public TradingCard {
public:
    YugiohCard(string n, int h) : TradingCard(n, h) {}

    void attack() override {
        cout << name << " attacks with Dark Magic!" << endl;
    }
};

// Derived class 3
class CustomCard : public TradingCard {
public:
    CustomCard(string n, int h) : TradingCard(n, h) {}

    void attack() override {
        cout << name << " performs Code Slash!" << endl;
    }
};

int main() {
    // Create a vector of pointers to TradingCard (The Base Class)
    vector<TradingCard*> deck;

    // Dynamically add cards to the vector
    deck.push_back(new PokemonCard("Pikachu", 90));
    deck.push_back(new YugiohCard("Dark Magician", 2500));
    deck.push_back(new CustomCard("Code Knight", 1800));

```

```
    // Iterate through the vector and use polymorphism
    for (auto* card : deck) {
        card->showInfo();
        card->attack();
        cout << endl;
    }

    // Clean up dynamically allocated memory
    for (auto* card : deck) {
        delete card;
    }

    return 0;
}
```

Listing 1: Trading Card Polymorphism Example

Expected Output

```
Card: Pikachu | HP: 90
Pikachu used Thunderbolt!

Card: Dark Magician | HP: 2500
Dark Magician attacks with Dark Magic!

Card: Code Knight | HP: 1800
Code Knight performs Code Slash!
```

2 Lab Exercises

1. Numbers (abstract class)

In this lab you will practice abstract classes and const correctness.

Define a class NumInterface whose only role is to provide a common interface (member function prototypes) for the derived classes. It should provide the functions: display(), increment(). You'll guess the full prototype of each function by what you think would be their functionality in the derived classes. In this abstract class the function bodies would be empty.

There is no point to store any data since we do not know what kind of number we are dealing with. You should make it an abstract class, i.e., it should be impossible to create its object.

Define your class such that the first line in main(), on compilation, produces the error as described in the comments. Once you have successfully produced the compile error, comment that line to pass the test case.

2. Numbers (const correctness)

From the base class NumInterface that you wrote in Q1, derive two more classes with following interfaces:

1. NumWhole: // this should store the values of whole numbers, choose an appropriate data type as private member!

   - NumWhole(val) // constructor
   - getval() // getter
   - setval() // setter
   - display() // should display the value and do nothing else.
   - increment() // should increment the stored value and do nothing else
   - operator+ // should be able to add two whole numbers using the "+" operator

2. NumComplex: // this should store the values of signed complex numbers, choose appro- priate data types to store the real and imaginary parts as private members!

   - NumComplex(real, imag) // constructor
   - getreal() // getter
   - setreal() // setter
   - getimag() // getter
   - setimag() // setter
   - display() // should display the value and do nothing else.
   - increment() // should increment the stored value (real++, imag++) and do nothing else
   - operator+ // should be able to add two complex numbers using the "+" operator

The interfaces of the functions (number of args as well as their types + return type) are delib- erately missed for you to guess.

The main function given should produce the output as provided in the comments.

3. Hotel Rooms

In a Hotel there are two types of rooms Simple Hotel room and Apartment room. Apartment are expensive as compared to Simple rooms.

Define a class HotelRoom which has two int members for storing the number of bathrooms and bedrooms. Define a constructor which initializes these members. also define a function named get price , which calculates the prices using this formulas: 50× ()Number of bedrooms)+100× (number of bathrooms)

Define another class called Apartment which inherits from HotelRoom. Define a relevant con- structor for it. This class also has a get price function which calculate the price using the above formula plus 100.

Write a main() which takes int n as input and creates a vector of type HotelRoom* of size n. Then takes n lines of input. each line contains the type of room, number of bed rooms and number of bathrooms. You have to create an object based on the type of room and add it into the array. After taking the input, loop through all the rooms and calculate the total price(profit) of all the rooms and print it.

Sample Case 0
Sample input format: 2 standard 3 1 apartment 1 1
Sample Output 500
Explanation In the sample we have one standard room with 3 bedrooms and 1 bathroom, and one apartment with one bedrooms and 1 bathroom. The price for the room is $3 \times 50 + 100 = 250$. The price for the apartment is $50 + 100 + 100 = 250$. Thus the hotel profit is $250 + 250 = 500$ as the sum of prices of both rooms.