BITS Pilani Hyderabad Campus

2nd Semester 2024-25 (CS F211: DSA) Lab-sheet 2

In the previous lab sheet, we saw structure of C++ programs including, classes, objects, constructors, member functions, class inheritance (derived class), and access modifiers (public, private, and protected). This lab will continue with additional features of C++.

Q.1 Run-time Polymorphism in C++. In this problem your task is to create a vehicle rental service to calculate rental charges for different types of vehicles. You need to implement the following steps, whose part code is given below.

Step 1: Create a base class named "Vehicle" with attributes, vehicleNumber (string), rentPerDay (float). The base class should have a method named as calculateRent (int days) as a Pure virtual function.

Step 2: Create derived class named as "Car" with additional attribute, driverFee(int float). Override calculateRent() to include driver fee: rent = (rentPerDay × days) + driverFee. Create another derived calss named as "Bike". Override calculateRent() to compute rent: rent = rentPerDay × days.

Step 3: Write a main() function to create an array of vehicles to store car and bike objects. Use polymorphism to compute and display rent for all vehicles.

Your output should look something similar to:

Vehicle: Car (Number: TS-08-5795)
Rental Duration: 3 days
Total Rent: 180
Driver Fee: 30

Vehicle: Bike (Number: TS-08-8543)
Rental Duration: 2 days
Total Rent: 40

C++ code:

```
#include <iostream>
#include <string>
using namespace std;
class Vehicle {
protected:
  string vehicleNumber;
  float rentPerDay;

public:
    Vehicle(string number,float rent):vehicleNumber(number), rentPerDay(rent)
    {}
    // Pure virtual function for calculating rent
    virtual float calculateRent(int days) = 0;
    virtual void displayDetails(int days) {
       cout << "Vehicle Number: " << vehicleNumber << endl;
       cout << "Rent for " << days << " day(s):" << calculateRent(days)<< endl;
    }

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


// Derived class: Car. Find out the missing code at ?
class Car: public Vehicle {
private:
    ? ;

public:
    Car(string number, float rent, float fee) : Vehicle(number, rent),
    driverFee(fee) {}

    // Override calculateRent
    float calculateRent(int days) override {
        return (rentPerDay * days) + driverFee;
    }
```

```
    // Override displayDetails
    void displayDetails(int days) override {
        cout << "Vehicle Type: Car" << endl;
        Vehicle::displayDetails(days);
        cout << "Driver Fee:" << driverFee << endl;
    }
};

// Derived class: Bike. Find out missing code marked as?
class Bike : public Vehicle {
public:
    // Constructor
    Bike(string number, float rent) : Vehicle(number, rent) {}

    // Override calculateRent
    ?
    }

    // Override displayDetails
    ?
    }
};

int main() {
    // Array of Vehicle pointers
    Vehicle* vehicles[2];

    // Create Car and Bike objects
    vehicles[0]=new Car("TS-08-5795", 50.0, 30.0);//Rent:50/day,Driver Fee: 30
    vehicles[1]=new Bike("TS-08-8543", 20.0);    // Rent: 20/day

    // Rental duration for each vehicle
    int rentalDays[] = {3, 2};

    // Display details and calculate rent for each vehicle
    for (int i = 0; i < 2; i++) {
        vehicles[i]->displayDetails(rentalDays[i]);
    }

    // Clean up memory
    for (int i = 0; i < 2; i++) {
        delete vehicles[i];
    }

    return 0;
}
```

Task: Extend the above code by adding a new derived class "Truck" with attributes like, loadCapacity (float, in tons) and extraChargePerTon (float). Override the calculateRent() method to calculate the rent based on:

`Rent = (rentPerDay * days) + (loadCapacity * extraChargePerTon)`

Also, add a feature to calculate and apply discounts on rental charges based on the rental duration. Add a method named `applyDiscount()` in the base class `Vehicle` to handle discounts with the following rule:

- If a customer rents a vehicle for more than 5 days, he or she is eligible for a 10% discount on the total rent.

Ensure this method is virtual, so that the derived classes (like Car, Bike, or Truck) can override it, if necessary.

You need to integrate it with Rent Calculation also. Use the applyDiscount() method inside the calculateRent() method to adjust the final rent amount. Display both the rent before discount and the final rent after applying the discount.

Q.2 Friend class and friend function: Friend class allows all member functions of one class to access private/protected members of another class. A Friend function is useful for non-member functions that need access to private data.
Below program designs two classes, ClassA and ClassB, where ClassB is a friend class of ClassA, and a friend function showValues accesses private members of both classes. Implements a method in ClassB to modify the private member of ClassA. Creates objects, displays initial values using showValues, modifies ClassA's value via ClassB, and displays updated values.

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

// Forward declaration of class ClassB
class ClassB;

class ClassA {
private:
    int valueA;

public:
    ClassA(int val) : valueA(val) {}

    // Declare a friend function
    friend void showValues(const ClassA& a, const ClassB& b);

    // Declare a friend class
    friend class ClassB;
};

class ClassB {
private:
    int valueB;

public:
    ClassB(int val) : valueB(val) {}

    // Friend function has access to private members of both classes
    friend void showValues(const ClassA& a, const ClassB& b);

    // A member function of ClassB accessing private member of ClassA
    void modifyValueOfA(ClassA& a, int newVal) {
        a.valueA = newVal; // Direct access to ClassA's private member
    }
};

// Friend function definition
void showValues(const ClassA& a, const ClassB& b) {
    cout << "Value in ClassA: " << a.valueA << endl;
    cout << "Value in ClassB: " << b.valueB << endl;
}

int main() {
    ClassA objA(10); // Create an object of ClassA
    ClassB objB(20); // Create an object of ClassB

    cout << "Initial values:" << endl;
    showValues(objA, objB);

    // Use ClassB's member function to modify ClassA's private data
    objB.modifyValueOfA(objA, 50);

    cout << "\nAfter modifying ClassA's value using ClassB:" << endl;
    showValues(objA, objB);

    return 0;
}
```

Task: Enhance the above code by adding a third class, ClassC. Make ClassC a friend class of both ClassA and ClassB. Add a method in ClassC called swapValues() that swaps the private members of ClassA and ClassB. Use the existing friend function to display the values before and after the swap.

Q.3 Templates in C++ are a powerful feature that allows you to write generic code that works seamlessly with different data types. They provide a way to create functions and classes that can be used with various data types without the need to duplicate code. This not only promotes code reusability but also enhances the efficiency of your programs by eliminating redundant implementations. Templates enable us to define functions or classes with placeholders for data types. When we use a template function or class, we provide the specific data type as an argument, and the compiler generates the appropriate code for that data type. For example, if we need to write functions for tasks like sorting and searching that handles data of different datatypes, we can write a single template function and pass the datatype as an argument.

```
1. template <typename T>
2. T add(T a, T b)
3. {
4. T result = a + b;
5. return result;
6. }
7. int main()
8. {
9. cout << add<int>(1,2)<<endl; // Call add for int
10. cout <<add<double>(1.2,2.3)<<endl; // Call add for double
11. }
```

Here, the template keyword signifies that the following function or class will be a template. The typename keyword denotes that the T specifier that follows it is a data type parameter. The word typename can also be replaced with the word Class like so:

```
template < class T>
```

In the above example, the compiler creates two overloaded functions, for int and double because those are the data types which are being used in the function calls.

Task: Write a C++ program to demonstrate the use of function templates and class templates to perform operations on different data types. Create a function template to find maximum and minimum of two values. Test it with different data types (int, double, char). Create a class template named Calculator that can add, multiple, and subtract numbers. Use the class template with at least two different data types, int and double.

Q.4 In this program, implement multiple inheritance with virtual inheritance. Virtual inheritance ensures that a base class (Engine) is shared among all derived classes (Car, Truck) and is initialized only once in the final derived class (Vehicle). Without virtual inheritance, there would be ambiguity about which Engine constructor to call if multiple paths to Engine exist in the inheritance hierarchy.

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

// Base class Engine
class Engine {
protected:
    string engineType;
    int engineCapacity; // in liters

public:
    Engine(string type, int capacity)
        : engineType(type), engineCapacity(capacity) {}

    void displayEngineDetails() {
        cout << "Engine Type: " << engineType << endl;
        cout << "Engine Capacity: " << engineCapacity << " liters" << endl;
    }
};
// Derived class Car, inherits Engine
class Car : virtual public Engine {
protected:
    int numOfDoors;

public:
    Car(string type, int capacity, int doors)
        : Engine(type, capacity), numOfDoors(doors) {}
```

```
    void displayCarDetails() {
        cout << "Car Details:" << endl;
        displayEngineDetails();
        cout << "Number of Doors: " << numOfDoors << endl;
    }
};

// Derived class Truck, inherits Engine
class Truck : virtual public Engine {
protected:
    int loadCapacity; // in tons

public:
    Truck(string type, int capacity, int load)
        : Engine(type, capacity), loadCapacity(load) {}

    void displayTruckDetails() {
        cout << "Truck Details:" << endl;
        displayEngineDetails();
        cout << "Load Capacity: " << loadCapacity << " tons" << endl;
    }
};

// Derived class that inherits both Car and Truck. Ex. of multiple inheritance
class Vehicle : public Car, public Truck {
public:
    Vehicle(string type, int engineCapacity, int doors, int load)
        : Engine(type, engineCapacity), Car(type, engineCapacity, doors),
                                        Truck(type, engineCapacity, load) {}

    void displayVehicleDetails() {
        cout << "Vehicle Details:" << endl;
        displayCarDetails();
        displayTruckDetails();
    }
};

int main() {
    // Create a Vehicle object that inherits from both Car and Truck
    Vehicle myVehicle("V8", 6, 4, 10); // Engine Type: V8, Engine Capacity: 6L,
                                                    4 Doors, Load Capacity: 10 tons

    // Display vehicle details
    myVehicle.displayVehicleDetails();

    return 0;
}
```

Task: Extend the above code to add exception handling (by adding below class) to check for invalid engine capacities, negative doors, or load capacities.

```
// Exception class for invalid input
class InvalidInputException : public exception {
   public: const char* what() const noexcept override {
        return "Invalid input: Value cannot be negative!";
   }
};
```

In the constructor class of engine, add if (capacity < 0) { throw InvalidInputException();}, and similarly for car (if doors < 0) and for truck (if load < 0).

Refs: cplusplus.com/doc/tutorial/
