3/8/2025

Muhammad Arham Usman

FAST NUCES

Object Oriented Programming

Lab 5

Task 1

# Source Code:

/\*

Task-01:

Create a class called Square with the following attributes:

• sideLength – float variable

• area – float variable

• allareas – static float variable

Create the following methods for your class:

• Constructors (default and parameterized (only takes sideLength as input))

• Function to calculate area, the calculated area should always be added to the allareas

variable each time an area is calculated. (assume it’s called only once for each object)

• Getters and Setters for all variables

In your main functions create 3 Squares with different sideLength. Call the area method for each of

those Square objects. After each time the area method is called, call that square’s getters for area and

allareas to display the updated values.

\*/

#include <iostream>

#include <random>

using namespace std;

class Square{

    float sideLength, area;

    static float allAreas;

    public:

    //constructor

    Square(float sideLength=0){

        this->sideLength=sideLength;

        area=0;

        if (sideLength!=0){

            calculateArea();

        }

    }

    //destructor

    ~Square(){

        allAreas-=area;

    }

    //Setter

    void setSideLength(float Length){

        sideLength=Length;

        if (area==0){

            calculateArea();

        }

        else{

            allAreas-=area;

            area=0;

            calculateArea();

        }

    }

    void setArea(){

        calculateArea();

    }

    //Getter

    float getSideLength(){

        return sideLength;

    }

    float getArea(){

        return area;

    }

    float getAllAreas(){

        return allAreas;

    }

    //Method

    void calculateArea(){

        if (area==0){ //check if area is being calculated for the first time

            area = sideLength\*sideLength;

            allAreas+=area;

        }

    }

};

float Square::allAreas=0;

int main(){

    Square squares[3];

    for (int i=0; i<3; i++){

        squares[i].setSideLength(rand()%10);

        squares[i].calculateArea();

        cout<<"Printing info for square"<<i+1<<endl;

        cout<<"Length: "<<squares[i].getSideLength()<<endl;

        cout<<"Area: "<<squares[i].getArea()<<endl;

        cout<<"Total Area: "<<squares[i].getAllAreas()<<endl;

        cout<<"----------\n";

    }

}

# Screen Shot:



Task 2

# Source Code:

/\*

Task-02:

Create a class called LoanHelper, which helps the user calculate their loan payments. The class

should have a variable that stores interest rate for the loan as a user defined constant value. Aside

from the that, it also stores the amount for the loan taken and amount of months that the user will

repay the loan in. The loan repayment should be calculated on a monthly basis, and the interest rate

should be applied over the monthly return amount. The output should be something like:

“You have to pay 999 every month for 12 monthsto repay your loan”

Note: first divide loan amount by number of months, then apply the interest rate on it. Interest rate

should be a value between 0-0.5%

\*/

#include <iostream>

using namespace std;

class LoanHelper{

    float interest\_rate;

    float amount\_of\_loan, amount\_of\_months;

    public:

    //Constructor

    LoanHelper(float InterestRate=0, float LoanAmount=0, float Months=0){

    }

    //Setter

    void setInterest(float InterestRate){

        if (InterestRate>=0&&InterestRate<=0.5){

            interest\_rate=InterestRate;

        }

        else{

            cout<<"ERROR! Interest Rate should be 0-0.5.\n";

        }

    }

    void setLoanAmount(float LoanAmount){

        if (LoanAmount<0){

            cout<<"ERROR! Loan Amount should be a positive integer.\n";

        }

        else{

            amount\_of\_loan=LoanAmount;

        }

    }

    void setMonths(float Months){

        if ((Months==0&&amount\_of\_loan==0)||Months>0){

            amount\_of\_months=Months;

        }

        else{

            cout<<"Error! Months should be greater than 0\n";

        }

    }

    //Methods

    void calculateLoan(){

        if (amount\_of\_loan==0){

            cout<<"You have repaid all your loan.\n";

        }

        else{

            if (amount\_of\_months==0){

                cout <<"MONTHS should be greater than 0.\n";

            }

            else {

                cout <<"You have to pay "<<(amount\_of\_loan\*(1+(interest\_rate/100)))/amount\_of\_months<<" every month for "<< amount\_of\_months<<" months to repay your loan.\n";

            }

        }

    }

};

int main(){

    LoanHelper l1;

    l1.setInterest(0.5);

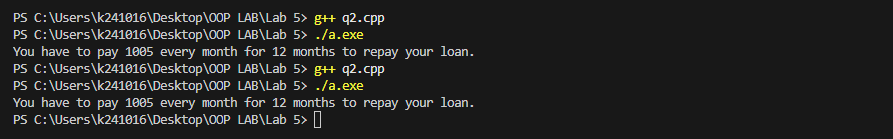
    l1.setLoanAmount(12000);

    l1.setMonths(12);

    l1.calculateLoan();

}

# Screen Shot:



Task 3

# Source Code:

/\*

Create a class called ValidateString. The purpose of this class will be check if the given characters in a

string are alphabet only. Numbers and symbols will mean that the string is invalid. By using a

parameterized constructor, create multiple objects for your class that accept different strings. Create a

constant function that checks whether the string variable is valid or not.

What happens if you do not make your function constant? Add a comment right above your function

explaining why we make a function constant.

\*/

#include <iostream>

#include <string>

using namespace std;

class ValidateString{

    string sentence;

    public:

    //Constructor

    ValidateString(string sentence=""){

        this->sentence=sentence;

    }

    //Setter

    void setString(string String){

        sentence=String;

    }

    //Getter

    string getString(){

        return sentence;

    }

    //const make sure that the function don't change the original string

    const bool validate\_string(){

        if (sentence==""){

            cout << "Set a string first"; return false;

        }

        for (int i=0; i<sentence.length(); i++){

            if ((sentence.at(i)>='a'&&sentence.at(i)<='z')||(sentence.at(i)>='A'&&sentence.at(i)<='Z')||(sentence.at(i)==' ')){}

            else{

                return false;

            }

        }

        return true;

    }

};

int main(){

    ValidateString \*Strings=new ValidateString[3];

    Strings[0].setString("Hello World");

    Strings[1].setString("Hello 123");

    Strings[2].setString("Hello @!@");

    for (int i=0; i<3; i++){

        cout << "String "<< i+1<<": "<<Strings[i].getString()<<endl;

        if (Strings[i].validate\_string()){

            cout<<"String is Valid.\n\n";

        }

        else{

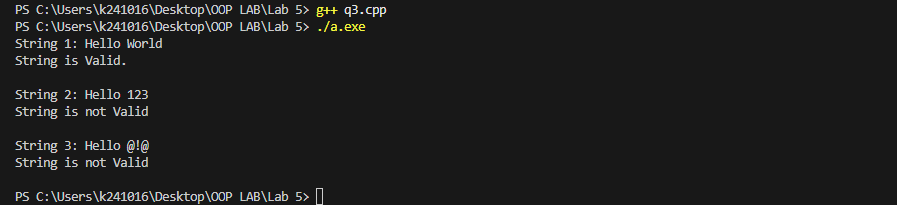
            cout << "String is not Valid\n\n";

        }

    }

}

# Screen Shot:



Task 4

# Source Code:

**/\***

**Task-04:**

**Create a BankAccount class. Which contains following details and functionalties:**

**Attributes: The BankAccount class has three private member variables: accountNumber,**

**accountHolderName, and balance.**

**Create Constructor: The class has a constructor that takes parameters to initialize the account**

**details (accountNumber, accountHolderName, and balance).**

**Create following Member Functions:**

**• deposit(double amount): Adds the specified amount to the account balance.**

**• withdraw(double amount): Subtracts the specified amount from the account balance, if**

**sufficient funds are available.**

**• display(): Displays the account details including the account number, account holder name,**

**and balance.**

**In the main() function, create an array accounts of BankAccount objects. The array contains three**

**elements, each representing a different bank account.**

**Initialize Each BankAccount object with specific account details such as account number, holder**

**name, and initial balance.**

**Perform following operations:**

**• Iterate through each account in the accounts array.**

**• For each account, display the account details using the display() function.**

**• Perform Two Transactions:**

**• Deposits 500.0 rupees into the account.**

**• Withdraws 200.0 rupees from the account.**

**• After each transaction, display the updated account details, including the new balance.**

**\*/**

**#include <iostream>**

**using namespace std;**

**class BankAccount{**

**string accountNumber, accountHolderName;**

**double balance;**

**public:**

**BankAccount(string AccountNumber="", string AccountHolderName="", double Balance){**

**accountHolderName=AccountHolderName;**

**accountNumber=AccountNumber;**

**balance=Balance;**

**}**

**void deposit(const double amount){**

**balance+=amount;**

**cout<<amount<<" is deposited successfully.\n";**

**}**

**bool withdraw(const double amount){**

**if (amount>balance){**

**cout<<"Transaction failed! Sufficient balance not available.\n";**

**return false;**

**}**

**balance-=amount;**

**cout<<amount<<"is withdrawed successfully.\n";**

**return true;**

**}**

**void display() const{**

**cout<<"-----\n";**

**cout<<"Account Number: "<<accountNumber<<endl;**

**cout<<"Account Holder Name: "<<accountHolderName<<endl;AC**

**cout<<"Balance: "<<balance<<endl;**

**cout<<"-----\n";**

**}**

**};**

**int main() {**

**BankAccount bop[]={BankAccount("0105 0203 04", "Ali", 100),**

**BankAccount("0100 0200 07", "Bilal", 150),**

**BankAccount("0109 0205 09", "Charlie", 200)**

**};**

**for (int i=0; i<3; i++){**

**bop[i].display();**

**bop[i].deposit(500.0);**

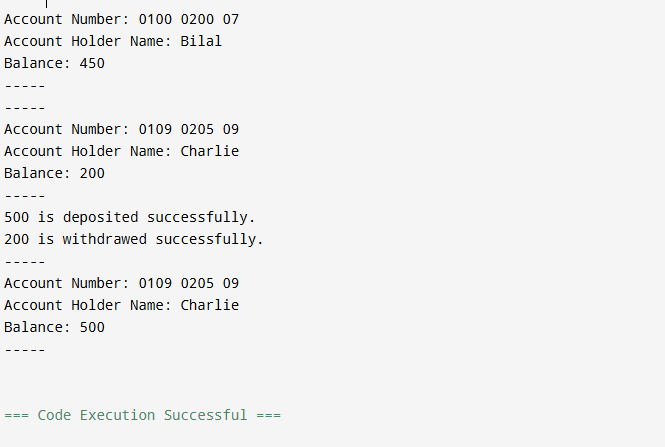
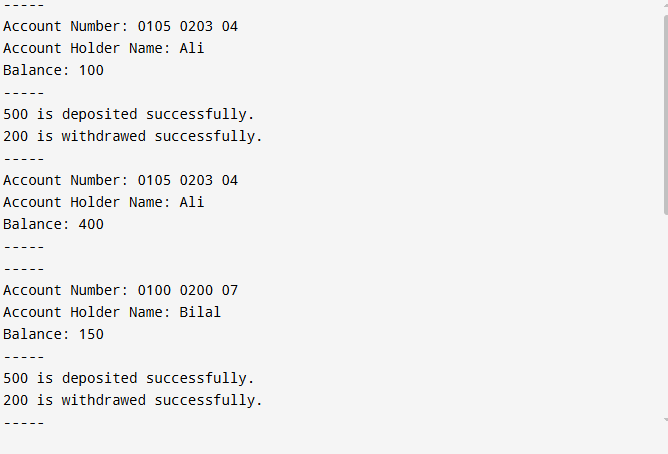
**bop[i].withdraw(200.0);**

**bop[i].display();**

**}**

**}**

# Screen Shot:



Task 5

# Source Code:

/\*

Keeping in mind our previous car example, write a class that represents a car, and useaggregation and

composition to combine different components like engine, wheels, headlights and steering to create the

car object.

Hint: create the individual classes firstbeforeperforming the composition. Remember

thatfor aggregation, you will need pointers! You�ll be needing constructors and setters to

set these values in case of aggregation. Same hint applies to other questions.

\*/

#include <iostream>

using namespace std;

class Engine

{

int size;

int horsepower;

public:

Engine(int Size = 0, int HP = 0)

{

size = Size;

horsepower = HP;

}

void displayEngineInfo()

{

cout << "Engine Size: " << size << "L, Horsepower: " << horsepower << " HP" << endl;

}

};

class Wheel

{

string type;

public:

Wheel(string Type = "Alloy")

{

type = Type;

}

void displayWheelInfo()

{

cout << "Wheel Type: " << type << endl;

}

};

class Headlight

{

string type;

public:

Headlight(string Type = "LED")

{

type = Type;

}

void displayHeadlightInfo()

{

cout << "Headlight Type: " << type << endl;

}

};

class Steering

{

string type;

public:

Steering(string Type = "Power")

{

type = Type;

}

void displaySteeringInfo()

{

cout << "Steering Type: " << type << endl;

}

};

class Car

{

Engine engine; // Composition

Headlight headlight; // Composition

Steering steering; // Composition

Wheel \*wheels[4]; // Aggregation

string name;

public:

Car()

{

engine = Engine();

headlight = Headlight();

steering = Steering();

name = "Car";

for (int i = 0; i < 4; i++)

{

wheels[i] = nullptr;

}

}

Car(string Name, Wheel \*Wheels[4], Engine engine, Headlight headlight, Steering steering) : name(Name), engine(engine), headlight(headlight), steering(steering)

{

for (int i = 0; i < 4; i++)

{

this->wheels[i] = Wheels[i];

}

}

void displayCarInfo()

{

cout << "Car Name: " << name << endl;

engine.displayEngineInfo();

headlight.displayHeadlightInfo();

steering.displaySteeringInfo();

for (int i = 0; i < 4; i++)

{

if (wheels[i] != nullptr)

{

wheels[i]->displayWheelInfo();

}

else

{

cout << "Wheels not set!" << endl;

}

}

}

};

int main()

{

Wheel bbs\_1("Alloy");

Wheel bbs\_2("Alloy");

Wheel bbs\_3("Alloy");

Wheel bbs\_4("Alloy");

Wheel \*carWheels[] = {&bbs\_1, &bbs\_2, &bbs\_3, &bbs\_4};

string carName = "Toyota";

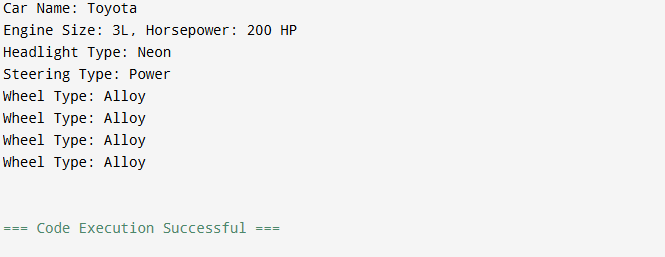
Car car1(carName, carWheels, Engine(3, 200), Headlight("Neon"), Steering("Power"));

car1.displayCarInfo();

return 0;

}

# Screenshot:



Task 6

# Source Code:

/\*

Keeping in mind our previous car example, write a class that represents a car, and useaggregation and

composition to combine different components like engine, wheels, headlights and steering to create the

car object.

Hint: create the individual classes firstbeforeperforming the composition. Remember

thatfor aggregation, you will need pointers! You’ll be needing constructors and setters to

set these values in case of aggregation. Same hint applies to other questions.

\*/

#include <iostream>

using namespace std;

class GraphicsRenderingEngine{

string resolution;

public:

GraphicsRenderingEngine(string resolution=""){

this->resolution=resolution;

cout<<"Graphics Rendering Engine created\n";

}

~GraphicsRenderingEngine(){

cout<<"Graphics Rendering Engine destroyed\n";

}

};

class InputHandler{

string eventManager;

public:

InputHandler(string EventManager=""){

eventManager=EventManager;

cout<<"Input Handler created\n";

}

~InputHandler(){

cout<<"InputHandler destroyed\n";

}

};

class PhysicsEngine{

float gravity;

public:

PhysicsEngine(float Gravity=0){

gravity=Gravity;

cout<<"Physics Engine created\n";

}

~PhysicsEngine(){

cout<<"Physics Engine destroyed\n";

}

};

class GameEngine{

GraphicsRenderingEngine gre;

PhysicsEngine pe;

InputHandler ih;

public:

GameEngine(string resolution="", string eventManager="", float gravity=0){

gre=GraphicsRenderingEngine(resolution);

pe=PhysicsEngine(gravity);

ih=InputHandler(eventManager);

cout<<"Game Engine created\n";

}

~GameEngine(){

cout<<"Game Engine destroyed\n";

}

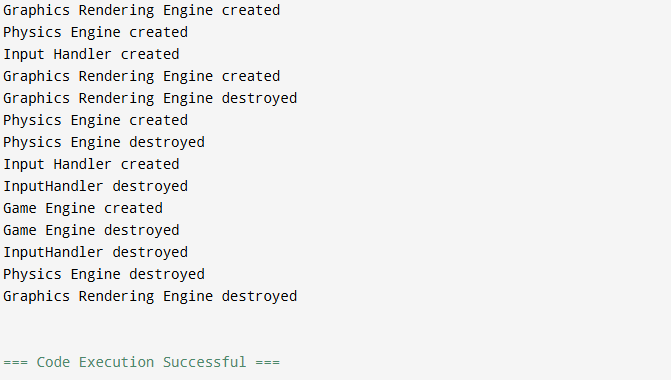
};

int main(){

GameEngine ge("1024x2048", "startGame", 9.812);

}

# Screen Shot:



Task 7

# Source Code:

/\*

Implement a restaurant ordering system that holds information about the restaurant’s menus, menu

items, orders, and payments. Identify the relationship between the five entities mentioned. Keep in

mind the following information as well:

a) Menu Items hold two things: food name and food price.

b) Menu is a class that holds an array of menu items. You can have different functions to add and

remove items, or display the entire menu.

c) Restaurant ordering system has one menu.

d) Any staff member can place an order to the system on behalf of a customer. Theorder class

consists of one or more menu items and a payment.

e) Payment is a class that holds the amount of money that a customer needs to pay.This is

generated when the order is placed.

\*/

#include <iostream>

#include <string>

using namespace std;

class MenuItem {

string name;

int price;

public:

MenuItem(string name = "", int price = 0) {

this->name = name;

this->price = price;

}

string get\_name() const {

return name;

}

int get\_price() const {

return price;

}

void update\_price(int price) {

this->price = price;

}

};

class Menu {

MenuItem\* items;

int noOfitems;

int maxitems;

public:

Menu() {

items = new MenuItem[50];

maxitems = 50;

noOfitems = 0;

}

~Menu() {

delete[] items;

}

MenuItem get\_item(int index) const {

if (index < 1 || index > noOfitems) {

return MenuItem();

}

return items[index - 1];

}

void addItem(const MenuItem& item) {

if (noOfitems == maxitems) {

maxitems \*= 2;

MenuItem\* temp = new MenuItem[maxitems];

for (int i = 0; i < noOfitems; i++) {

temp[i] = items[i];

}

delete[] items;

items = temp;

}

items[noOfitems] = item;

noOfitems++;

}

void removeItem(const string& name) {

bool found = false;

for (int i = 0; i < noOfitems; i++) {

if (items[i].get\_name() == name) {

found = true;

}

if (found && i < noOfitems - 1) {

items[i] = items[i + 1];

}

}

if (found) {

noOfitems--;

cout << "Item removed successfully\n";

} else {

cout << "Error! Item not found\n";

}

}

void displayMenu() const {

cout << "\nDisplaying Menu:\n";

for (int i = 0; i < noOfitems; i++) {

cout << "Item " << i + 1 << ": Name: " << items[i].get\_name() << " || Price: " << items[i].get\_price() << endl;

}

cout << "----------\n";

}

int get\_noOfitems() const {

return noOfitems;

}

};

class Payment {

mutable int bill; // Declare bill as mutable so it can be modified in const functions

public:

Payment() : bill(0) {}

int generate\_bill(MenuItem\* items, int noOfitems) const {

bill = 0; // Now this is allowed because bill is mutable

cout << "\n\n=====Bill=====\n";

for (int i = 0; i < noOfitems; i++) {

cout << "Item " << i + 1 << ": Name: " << items[i].get\_name() << " || Price: " << items[i].get\_price() << endl;

bill += items[i].get\_price();

}

cout << "\nTotal Amount Payable: " << bill << endl;

cout << "==========\n";

return bill;

}

};

class Order {

Menu\* menu;

MenuItem\* items;

int noOfitems;

Payment p;

void addItem(const MenuItem& item) {

MenuItem\* temp = new MenuItem[noOfitems + 1];

for (int i = 0; i < noOfitems; i++) {

temp[i] = items[i];

}

temp[noOfitems++] = item;

delete[] items;

items = temp;

}

void placeOrder() const {

p.generate\_bill(items, noOfitems);

}

public:

Order() {

items = nullptr;

noOfitems = 0;

}

~Order() {

delete[] items;

}

void OrderFood() {

int itemIndex;

menu->displayMenu();

cout << "Enter the item number to order (or 0 to finish): ";

while (true) {

cin >> itemIndex;

if (itemIndex == 0) break;

if (itemIndex > 0 && itemIndex <= menu->get\_noOfitems()) {

addItem(menu->get\_item(itemIndex));

cout << "Item added to order.\n";

}

else {

cout << "Invalid item number.\n";

}

cout << "Enter another item number or 0 to finish: ";

}

placeOrder();

}

void setMenu(Menu\* menu) {

this->menu = menu;

}

};

class Restaurant {

Menu menu;

Order order;

public:

void addMenuItem(const MenuItem& item) {

menu.addItem(item);

}

void removeMenuItem(const string& name) {

menu.removeItem(name);

}

void displayMenu() {

menu.displayMenu();

}

void OrderFood() {

order.setMenu(&menu);

order.OrderFood();

}

};

int main() {

Restaurant restaurant;

restaurant.addMenuItem(MenuItem("Pizza", 2500));

restaurant.addMenuItem(MenuItem("Burger", 1500));

restaurant.addMenuItem(MenuItem("Pasta", 800));

restaurant.OrderFood();

}

# Screen Shot:

