Универзитет „Св. Кирил и Методиј“ - Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Објектно ориентирано програмирање

Аудиториски вежби 9

Содржина

1. Повеќекратно наследување

1.1. Задача

Да се состави класа за автомобил со млазен погон кој наследува својства од две класи, автомобил и млазен авион (дијамант проблем).

Решение `oop_av91.cpp`

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

class Vehicle {
public:
    Vehicle() {
        cout << "Vehicle Constructor" << endl;
    }
    virtual ~Vehicle() {
        cout << "Vehicle Destructor" << endl;
    }
    virtual void accelerate() const {
        cout << "Vehicle Accelerating" << endl;
    }
    void setAcceleration(double a) {
        acceleration = a;
    }
    double getAcceleration() const {
        return acceleration;
    }
protected:
    double acceleration;
};

class Car : public Vehicle {
public:
    Car(){
        cout << "Car Constructor" << endl;
    }
    virtual void accelerate() const {
        cout << "Car Accelerating" << endl;
    }
    virtual void drive() const {
        cout << "Car Driving" << endl;
    }
    virtual ~Car() {
        cout << "Car Destructor" << endl;
    }
};

class Jet : public Vehicle {
public:
    Jet() {
        cout << "Jet Constructor" << endl;
    }
    virtual ~Jet() {
        cout << "Jet Destructor" << endl;
    }
    virtual void fly() const {
        cout << "Jet flying" << endl;
    }
};
class JetCar : public Car, public Jet {
public:
    JetCar(){
        cout << "JetCar Constructor" << endl;
    }
    virtual ~JetCar() {
```

```
        cout << "JetCar Destructor" << endl;
    }
    virtual void drive() const {
        cout << "JetCar driving" << endl;
    }
    virtual void fly() const {
        cout << "JetCar flying" << endl;
    }
};

void analyzeCarPerformance(Car *testVehicle) {
    testVehicle->drive();
    //функцијата drive() може да се повика и со покажувач кон основната и
    //кон изведената класа. Оваа функција е дефинирана и во двете класи
}
void analyzeJetPerformance(Jet *testVehicle) {
    testVehicle->fly();
    //fly() е дефинирана и во основната и во изведената класа (Jet и JetCar)
}
int main() {
    Car myCar;
    Jet myJet;
    JetCar myJetCar;
    cout << endl << endl;
    cout << "Car testing in progress" << endl;
    analyzeCarPerformance(&myCar);
    analyzeCarPerformance(&myJetCar);
    cout << "Jet testing in progress" << endl;
    analyzeJetPerformance(&myJet);
    analyzeJetPerformance(&myJetCar);
    cout << endl << endl;
    return 0;
}
```

Излез од програмата е:

```
Vehicle Constructor
Car Constructor
Vehicle Constructor
Jet Constructor
Vehicle Constructor
Car Constructor
Vehicle Constructor
Jet Constructor
JetCar Constructor

Car testing in progress
Car Driving
JetCar driving
Jet testing in progress
Jet flying
JetCar flying

JetCar Destructor
Jet Destructor
Vehicle Destructor
Car Destructor
Vehicle Destructor
Jet Destructor
Vehicle Destructor
Car Destructor
Vehicle Destructor
```

Како што може да се забележи кога се креира објект од класата `JetCar`, конструкторот на класата `Vehicle` се повикува два пати.

```
Vehicle Constructor
Car Constructor
Vehicle Constructor
Jet Constructor
JetCar Constructor
```

Исто така и деструкторот на класата `Vehicle` се повикува два пати при повик на деструктор на објект од класата `JetCar`.

```
JetCar Destructor
Jet Destructor
Vehicle Destructor
Car Destructor
Vehicle Destructor
```

За да се избегне оваа појава при повеќекратното наследување се воведува виртуелно наследување за класите `Car` и `Jet` од класата `Vehicle`.

```
class Car: virtual public Vehicle
...
class Jet: virtual publuc Vehicle
...
```

Точно решение `oop_av911.cpp`

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

class Vehicle {
public:
    Vehicle() {
        cout << "Vehicle Constructor" << endl;
    }
    virtual ~Vehicle() {
        cout << "Vehicle Destructor" << endl;
    }
    virtual void accelerate() const {
        cout << "Vehicle Accelerating" << endl;
    }
    void setAcceleration(double a) {
        acceleration = a;
    }
    double getAcceleration() const {
        return acceleration;
    }
protected:
    double acceleration;
};

class Car : virtual public Vehicle {
public:
    Car(){
        cout << "Car Constructor" << endl;
    }
    virtual void accelerate() const {
        cout << "Car Accelerating" << endl;
    }
    virtual void drive() const {
        cout << "Car Driving" << endl;
    }
    virtual ~Car() {
        cout << "Car Destructor" << endl;
    }
```

```
};

class Jet : virtual public Vehicle {
public:
    Jet() {
        cout << "Jet Constructor" << endl;
    }
    virtual ~Jet() {
        cout << "Jet Destructor" << endl;
    }
    virtual void fly() const {
        cout << "Jet flying" << endl;
    }
};
class JetCar : public Car, public Jet {
public:
    JetCar(){
        cout << "JetCar Constructor" << endl;
    }
    virtual ~JetCar() {
        cout << "JetCar Destructor" << endl;
    }
    virtual void drive() const {
        cout << "JetCar driving" << endl;
    }
    virtual void fly() const {
        cout << "JetCar flying" << endl;
    }
};

void analyzeCarPerformance(Car *testVehicle) {
    testVehicle->drive();
    //функцијата drive() може да се повика и со покажувач кон основната и
    //кон изведената класа. Оваа функција е дефинирана и во двете класи
}
void analyzeJetPerformance(Jet *testVehicle) {
    testVehicle->fly();
    //fly() е дефинирана и во основната и во изведената класа (Jet и JetCar)
}
int main() {
    Car myCar;
    Jet myJet;
    JetCar myJetCar;
    cout << endl << endl;
    cout << "Car testing in progress" << endl;
    analyzeCarPerformance(&myCar);
    analyzeCarPerformance(&myJetCar);
    cout << "Jet testing in progress" << endl;
    analyzeJetPerformance(&myJet);
    analyzeJetPerformance(&myJetCar);
    cout << endl << endl;
    return 0;
}
```

1.2. Задача

Да се имплементира класа `Product` за која се чуваат името и цената. Да се имплементира апстрактна класа `Discount` во која има два чисто виртуелни методи за цена и за цена со попуст. Од овие класи да се изведат класите:

- `FoodProduct` за која дополнително се чува бројот на калории;
- `DigitalProduct` за која дополнително се чува големината (во MB)

Да се имплементира надворешна функција `total_discount` која ќе пресметува вкупен попуст на неколку продукти на попуст кои ги прима како аргумент.

Решение `oop_av92.cpp`

```
#include <iostream>
#include <cstring>
using namespace std;
class Discount {
public:
    virtual float getDiscount_price() = 0;
    virtual float getPrice() = 0;
};
class Product {
protected:
    char name[100];
    float price;
public:
    Product(const char *name = "", const float price = 0) {
        strcpy(this->name, name);
        this->price = price;
    }
    float getPrice() {
        return price;
    }
};

class DigitalProduct : public Product, public Discount {
private:
    float size;
public:
    DigitalProduct(const char *name = "", const float price = 0, const float size = 0){
        strcpy(this->name, name);
        this->price = price;
        this->size = size;
    }
    // се препокриваат функциите од апстрактната класа
    float getDiscount_price() {
        // попустот е 10%
        return 0.9 * getPrice();
    }
    // и двете класи Product и Discount имаат функција getPrice, која се
    // препокрива во изведената. Оваа ја користи getPrice од Product
    float getPrice() {
        return Product::getPrice();
    }

};

class FoodProduct : public Product, public Discount {
private:
    float callories;
public:
    FoodProduct(const char *name = "", const float price = 0, const float callories = 0) :
Product(name, price) {
        this->callories = callories;
    }
    float getDiscount_price() {
        // попустот е 20%
        return .8 * getPrice();
    }
    // и двете класи Product и Discount имаат функција getPrice, која се
    // препокрива во изведената. Оваа ја користи getPrice од Product
    float getPrice() {
        return Product::getPrice();
    }
};

float total_discount(Discount **d, int n) {
    float price = 0;
```

```
    for (int i = 0; i < n; ++i) {
        // повик на функцијата getPrice од класата FoodProduct или
        // DigitalProduct соодветно, затоа што во Discount функцијата е виртуелна
        price += d[i]->getPrice();
    }
    float discount = 0;
    for (int i = 0; i < n; ++i) {
        discount += d[i]->getDiscount_price();
    }
    return price - discount;
}
int main() {
    Discount **d = new Discount*[3];
    d[0] = new FoodProduct("Cheese", 450, 1200);
    d[1] = new FoodProduct("Wine", 780, 250);
    d[2] = new DigitalProduct("WOW", 380, 400);
    cout << "Difference: " << total_discount(d, 3) << endl;
    for (int i = 0; i < 3; ++i) {
        delete d[i];
    }
    delete[] d;
    return 0;
}
```

2. Изворен код од примери и задачи

https://github.com/finki-mk/OOP/

Source code ZIP