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

Object oriented programming

Exercises 9

Version 1.0, 18 April, 2017

Table of Contents

1. Multiple inheritance 1
    1.1. CarJet (The Diamond Problem) 1
    1.2. Product 4
2. Source code of the examples and problems 7

1. Multiple inheritance

1.1. CarJet (The Diamond Problem)

Compose a class for car with jet engine that derives from two classes, car and jet.

Soltuion `oop_av91_en.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;
}
```

Output of the program:

```
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
```

As we can notice when an object of class `JetCar` is created, the constructor of the class `Vehicle` is called twice.

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

Also the destructor of the class `Vehicle` is called twice when a `JetCar` object is destructing.

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

To avoid this problem with multiple inheritance a new type of `virtual` inheritance is introduced when classes `Car` and `Jet` derive from `Vehicle`.

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

Correct solution `oop_av911_en.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 (Јет 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. Product

Implement class `Product` that has a name and price. Implement abstract class `Discount` that has two pure virtual methods for price and discounted price. From these classes derive the following classes:

- `FoodProduct` that has additional info for the number of calories;
- `DigitalProduct` that has additional info for size (in MB)

Implement external function `total_discount` that will compute the total discount of few products that are on discount.

Solution `oop_av92_en.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;
    }
    // overriding functions from the base class
    float getDiscount_price() {
        // the discount is 10%
        return 0.9 * getPrice();
    }

    float getPrice() {
        return Product::getPrice();
    }

};

class FoodProduct : public Product, public Discount {
private:
    float calories;
public:
    FoodProduct(const char *name = "", const float price = 0, const float calories = 0) :
Product(name, price) {
        this->calories = calories;
    }
    float getDiscount_price() {
        // discount is 20%
        return .8 * getPrice();
    }

    float getPrice() {
        return Product::getPrice();
    }
};

float total_discount(Discount **d, int n) {
    float price = 0;
    for (int i = 0; i < n; ++i) {
        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. Source code of the examples and problems

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

Source code ZIP