"Ss. Cyril and Methodius" University in Skopje
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Object oriented programming

Exercises 10

Version 1.0, 4 May, 2017

Table of Contents

1. Static members and exceptions

1.1. Problem

Each card payment has its advantages. The state, motivated to increase usage of cards, offers better payment conditions. Model a class `Card` and classes `Master` and `Maestro` that derive from it. A card is described with its ID and the balance.

On payment with Maestro card, on each payment there is a discount of 5% for ALL users of the card. This percentage is fixed and can not be changed!

On paying with Master card, if the limit of the card is over 6000 MKD then the discount is 10%, instead of the standard 3$ for cards with limit less then 6000 MKD. The discount of 10% is same for ALL users, but it can be changed from the National bank.

Solution `oop_av101_en.cpp`

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

class Card {
private:
    char id[16];
    double balance;

public:
    Card(char* id = "", double  balance = 0) {
        this->balance = balance;
        strcpy(this->id, id);
    }
    void showBalance() {
        cout << id << ": " << balance << endl;
    }

    void deposit(double  amount) {
        this->balance += amount;
    }

    virtual double withdraw(double amount) {
        if (this->balance > amount) {
            this->balance -= amount;
            return amount;
        }
        else return 0;
    }

protected:
    virtual double withdraw(double amount, double limit) {
        if (this->balance + limit > amount) {
            this->balance -= amount;
            return amount;
        }
        else return 0;
    }
};
```

```
class Maestro : public Card {
private:
    char password[10];
    const static double discount; //static member of the class

public:
    Maestro(char* password = "", char* id = "", double  balance = 0) : Card(id, balance) {
        strcpy(this->password, password);
    }

    static double  getPopust() { //static function accessing a static member
        return discount;
    }

    double withdraw(double price) {
        double  amount = price * (1 - discount); // non-static functions can use static
members
        // static data members
        return Card::withdraw(amount);
    }
};
const double Maestro::discount = 0.05; // initialization of static member

class Master : public Card {
private:
    double  limit;
    const static double discount1;  // fixed discount
    static double discount2;       // discount that can be changed

public:
    Master(double  limit = 0, char* id = "", double  balance = 0) : Card(id, balance) {
        this->limit = limit;
    }

    static double getDiscount1() {
        return discount1;
    }

    static double getDiscount2() {
        return discount2;
    }

    static void setDiscount2(double  discount2) {
        Master::discount2 = discount2;
    }

    double withdraw(double price) {
        if (this->limit < 6000) {
            double  amount = price * (1 - discount1);
            return Card::withdraw(amount, limit);
        }
        else {
            double  amount = price * (1 - discount2);
            return Card::withdraw(amount, limit);
        }
    }
};
const double Master::discount1 = 0.03; // on initialization you do not write the keyword
static
double Master::discount2 = 0.1;
```

1.2. Problem

Create a class `Register` that should represent the register of a store where customers can pay in cash or with card. Each register has two daily amounts, one for the cash payments and other for the card payments. Also, each object from the class is created on different day, so for each object we store the day, month and year when the

register is opened.

In the class `Register` there is a function `pay()` that enables accepting payments. For cash payments there is function with definition `pay(double)` with the amount that should be paid.

For payments with cards create a function with definition `pay(double, Card&)` that as argument also accepts and a reference to the card for the payment. Calling this function should update the balance of the register and the card.

In the class define a function `show()` that will show the information of the register.

Solution `oop_av102_en.cpp`

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

class Card {
private:
    char id[16];
    double balance;

public:
    Card(char* id = "", double  balance = 0) {
        this->balance = balance;
        strcpy(this->id, id);
    }
    void showBalance() {
        cout << id << ": " << balance << endl;
    }

    void deposit(double  amount) {
        this->balance += amount;
    }

    virtual double withdraw(double amount) {
        if (this->balance > amount) {
            this->balance -= amount;
            return amount;
        }
        else return 0;
    }

protected:
    virtual double withdraw(double amount, double limit) {
        if (this->balance + limit > amount) {
            this->balance -= amount;
            return amount;
        }
        else return 0;
    }
};

class Maestro : public Card {
private:
    char password[10];
    const static double discount; //static member of the class

public:
    Maestro(char* password = "", char* id = "", double  balance = 0) : Card(id, balance) {
        strcpy(this->password, password);
    }

    static double  getPopust() { //static function accessing a static member
```

```
        return discount;
    }

    double withdraw(double price) {
        double  amount = price * (1 - discount); // non-static functions can use static
members
        // static data members
        return Card::withdraw(amount);
    }
};
const double Maestro::discount = 0.05; // initialization of static member

class Master : public Card {
private:
    double  limit;
    const static double discount1;  // fixed discount
    static double discount2;       // discount that can be changed

public:
    Master(double  limit = 0, char* id = "", double  balance = 0) : Card(id, balance) {
        this->limit = limit;
    }

    static double getDiscount1() {
        return discount1;
    }

    static double getDiscount2() {
        return discount2;
    }

    static void setDiscount2(double  discount2) {
        Master::discount2 = discount2;
    }

    double withdraw(double price) {
        if (this->limit < 6000) {
            double  amount = price * (1 - discount1);
            return Card::withdraw(amount, limit);
        }
        else {
            double  amount = price * (1 - discount2);
            return Card::withdraw(amount, limit);
        }
    }
};
const double Master::discount1 = 0.03; // on initialization you do not write the keyword
static
double Master::discount2 = 0.1;

class Register {
private:
    double  cashBalance;
    double  cardBalance;
    int day, month, year;

public:

    Register(double  cashBalance, int day, int month, int year) {
        this->cashBalance = cashBalance;
        this->cardBalance = 0;
        this->day = day;
        this->month = month;
        this->year = year;
    }

    void pay(double amount) {
        this->cashBalance += amount;
    }

    void pay(double amount, Card &c) {
        this->cardBalance += c.pay(amount);
    }

    void show() {
        cout << "Day: \t" << day << endl;
```

```
        cout << "Month: \t" << month << endl;
        cout << "Year: " << year << endl;
        cout << "Balance: " << this->balance() << endl;
        cout << endl;
    }

    double  balance() {
        return this->cardBalance + this->cashBalance;
    }
};

int main() {
    Register daily(10000, 22, 4, 2014);
    Card *k;
    daily.show();

    cout << "Paying in cash!" << endl;
    daily.pay(5000);
    daily.show();

    k = new Master(10000.00, "1234567890123456", 54000.00);
    cout << "Paying with card!" << endl;
    daily.pay(10000.00, *k);
    daily.show();

    k = new Maestro("password", "1234567890123456", 54000.00);
    cout << "Paying with card!" << endl;
    daily.pay(10000, *k);
    daily.show();

    Master::setPopust2(0.07);
    k = new Master(10000, "4567891234567890", 3000);
    cout << "Paying with card!" << endl;
    daily.pay(10000, *k);
    daily.show();
    return 0;
}
```

1.3. Problem

Part of the products in one store after the new policy must have some amount of discount. For this purpose the system of the store should be extended with an abstract class for `Discount`. This class should keep the course ratios of euros and dollars in MKD and methods that each class that derives it should implement.

- `float discount_price();`
- `float price();`
- `void print_rule();`

For each product `Product` keep the name and the price and implement constructor and needed methods.

Products are divided in several types: `FoodProduct`, `Drinks` and `Cosmetics`.

Following the new policy hte food is not on discount. The drinks, with alcohol and with price larger than 20 euros have discount of 5%, and alcohol free from the brand

Coca-Cola have discount of 10%. All cosmetic products with price larger than 5 euros have discount of 12%, and those with price larger than 20 dollars have discount of 14%.

Compute the total price of all products with the discount.

Also, create a function `changePrice(float)` in the class `Product` that enables changing the price of the product. If there is an attempt to set a negative price for the product, throw an exception (object from class `NegativeValueException`). Handle with the exception in the main function, where you list all the products of type `Cosmetics` and change their price.

Solution `oop_av103_en.cpp`

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

// exception class
class NegativeValueException {
private:
    char text[50];
public:
    NegativeValueException(char *text)
    {
        strcpy(this->text, text);
    }
    void print() { cout << text; }
};

class Discount {
public:
    static float euro;
    static float dollar;
    virtual float discount_price() = 0;
    virtual float price() = 0;
    virtual void print_rule() = 0;
};

float Discount::euro = 61.7;
float Discount::dollar = 44.5;

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;
    }
    void print() {
        cout << "Product{ name=" << name << ", price=" << price << "}" << endl;
    }
    void changePrice(float price) {
        if (price < 0) throw NegativeValueException("You entered a negative price for the
exception!\n");
        this->price = price;
    }
};
```

```
class Cosmetics : public Product, public Discount {
private:
    int weight;
public:
    Cosmetics(const char *name = "", const float price = 0,
              const int weight = 0) : Product(name, price) {
        this->weight = weight;
    }
    float discount_price() {
        if (getPrice() / Discount::dollar > 20)
            return 0.86 * getPrice();
        if (getPrice() / Discount::euro > 5)
            return 0.88 * getPrice();
        return getPrice();
    }
    float price() {
        return getPrice();
    }
    void print_rule() {
        cout << "All cosmetic products with price > 5 euro have discount of 12%, while
those with price > 20 dollars have discount 14%" << endl;
    }
};

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 discount_price() {
        return getPrice();
    }

    float price() {
        return getPrice();
    }

    void print_rule() {
        cout << "No discount on food" << endl;
    }
};

class Drinks : public Product, public Discount {
private:
    char brand[100];
    bool alcoholic;
public:
    Drinks(const char *name = "", const float price = 0,
           const char *brand = "", const bool alcoholic = false) : Product(name, price) {
        strcpy(this->brand, brand);
        this->alcoholic = alcoholic;
    }
    float discount_price() {
        if (this->alcoholic && (getPrice() / Discount::euro > 20))
            return 0.95 * getPrice();
        if (!this->alcoholic && (strcmp(this->brand, "Coca-Cola") == 0))
            return 0.90 * getPrice();
        return getPrice();
    }
    float price() { return getPrice(); }
    void print_rule() {
        cout << "All alcohol drinks with price > 20 euros have discount of 5%, while
alcohol free Coca-Cola have discount of 10% " << endl;
    }
};

float total_discount(Discount **d, int n) {
    float discount = 0;
    for (int i = 0; i < n; ++i) {
        discount += d[i]->discount_price();
        cout << "Original price: " << d[i]->price() << endl;
        cout << "Discounted: " << d[i]->discount_price() << endl;
```

```
        d[i]->print_rule();
    }
    return discount;
}

int main() {
    int n = 0;
    float newPrice;
    Discount **d = new Discount*[10];
    d[n++] = new FoodProduct("leb", 30);
    d[n++] = new Drinks("viski", 1350, "Jack Daniel's", true);
    d[n++] = new FoodProduct("sirenje", 390, 105);
    d[n++] = new Drinks("votka", 850, "Finlandia", true);
    d[n++] = new Cosmetics("krema", 720, 100);
    d[n++] = new Drinks("sok", 50, "Coca-Cola", false);
    d[n++] = new Cosmetics("parfem", 3500, 50);

    cout << "Total price of all products is: " << total_discount(d, n) << endl;

    cout << "Changing the price of the cosmetic products" << endl;
    for (int i = 0; i < n; ++i) {
        Cosmetics* c = dynamic_cast<Cosmetics *>(d[i]);
        if (c != 0) {
            c->print();
            cin >> newPrice;
            try {
                c->changePrice(newPrice);
            }
            catch (NegativeValueException i) {
                i.print();
            }
        }
    }

    for (int i = 0; i < n; ++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