

**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
Κ10:ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ**

Εξετάσεις 15 Σεπτεμβρίου 2014

1. (α') Η αφαίρεση στα δεδομένα ή η έννοια της κληρονομικότητας είναι πιο χαρακτηριστική για τον αντικειμενοστραφή προγραμματισμό; Δώστε ένα παράδειγμα τύπου όπου η αφαίρεση στα δεδομένα είναι χρήσιμη στην υλοποίησή του αλλά η κληρονομικότητα δεν είναι απαραίτητη.

- (β') Δίδεται το παρακάτω πρόγραμμα C++:

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

class A{ int data;
public:
    A(int i=100) : data(i)
        { cout << "I just constructed an A with: " << data << endl;}
    A(const A& a) : data(a.data)
        { cout << "I just constructed an A by copying with data "
            << data << endl;}
    ~A() { cout << "Destructing an A with data " << data << endl;}
    int get_data() { return data; }
    void change() { data += 100; } };

class FatA{
    A& data1; A& data2;
public:
    FatA(A& a) : data1(a),data2(a)
        { cout << "I just constructed a FatA with: "
            << data1.get_data() << " " << data2.get_data() << endl;}
    ~FatA() { cout << "Destructing a FatA " << endl;}
    void change() { data1.change(); data2.change(); } };

class BigFatA{
    FatA& data1; FatA& data2;
public:
    BigFatA(FatA& fatA) : data1(fatA),data2(fatA)
        { cout << "I just constructed a BigFatA " << endl; }
    ~BigFatA() { cout << "Destructing a BigFatA " << endl;}
    void change() { FatA cdata1 = data1; cdata1.change();
                    FatA cdata2 = data2; cdata2.change(); }
};

int main(){
    A a;
    FatA fatA(a); fatA.change();

    BigFatA bigFatA(fatA);
    bigFatA.change();

    return 0; }
```

Δώστε το αποτέλεσμα της εκτέλεσής του, αιτιολογώντας την απάντησή σας.

2. Δίδεται το παρακάτω πρόγραμμα C++

```

#include <iostream>
using namespace std;

class Dish{
protected:
    int cost; int cal;
public:
    Dish(int cst=0, int cl=150) : cost(cst), cal(cl)
        { cout << "A Dish was created " << endl; }
    ~Dish() { cout << "A Dish will be destroyed" << endl; }
    void eat() { cout << "Eating a Dish with cal " << cal
        << " and cost " << cost << endl; }
    int get_cost() { return cost; }
    int get_cal() { return cal; } };

class FirstCourse : public Dish{
public:
    FirstCourse(int cst, int cl) : Dish(cst,cl)
        {cout << "A FirstCourse was created with " <<
            cost << " and " << cal << endl; }
    ~FirstCourse() { cout << "A FirstCourse will be destroyed" << endl; } };

class Starter : public FirstCourse{
public :
    Starter(int cst, int cl) : FirstCourse(cst, cl)
        { cout << "A Starter was created " << endl ;}
    ~Starter() { cout << "A Starter will be destroyed" << endl; }
    void eat() { FirstCourse::eat();
        cout << "Eating Starter " << endl; } };

class Salad : public FirstCourse{
public :
    Salad(int cst, int cl) : FirstCourse(cst, cl)
        { cout << "A Salad was created " << endl ;}
    ~Salad() { cout << "A Salad will be destroyed" << endl; }
    void eat() { FirstCourse::eat();
        cout << "Eating Salad " << endl; } };

class MainCourse : public Dish{
public:
    MainCourse(int cst, int cl) : Dish(cst,cl)
        { cout << "A MainCourse was created with " <<
            cost << " and " << cal << endl; }
    ~MainCourse() { cout << "A MainCourse will be destroyed" << endl; } };

class Dessert : public Dish{
public:
    Dessert(int cst, int cl) : Dish(cst,cl)
        { cout << "A Dessert was created " << endl; }
    ~Dessert() { cout << "A Dessert will be destroyed" << endl; }
    void eat() { Dish::eat();
        cout << "Eating Dessert " << endl; } };

```

```

class Meal{
    static const int service = 1;
    Dish* starter;
    Dish* salad;
    Dish* maincourse;
    Dish* dessert;

public:
    Meal() { cout << "A Meal has been created " << endl; }
    ~Meal() { cout << "A Meal will be destroyed" << endl; }
    void order() { starter = new Starter(4, 100);
                  salad = new Salad(6, 50);
                  maincourse = new MainCourse(10, 500);
                  dessert = new Dessert(4, 450); }
    void eat() { starter->eat(); salad->eat();
                maincourse->eat(); dessert->eat();}
    int pay_bill() { return starter->get_cost() + salad->get_cost() +
                    maincourse->get_cost() + dessert->get_cost() + service;}
    int evaluate_cals() { return starter->get_cals() + salad->get_cals() +
                        maincourse->get_cals() + dessert->get_cals(); } };

int main(){
    Meal meal;
    meal.order();
    meal.eat();
    cout << "You are going to pay " << meal.pay_bill() << " euros " << endl;
    cout << "You are going to get " << meal.evaluate_cals()
         << " calories " << endl ;
    return 0; }

```

Δώστε το αποτέλεσμα της εκτέλεσής του, αιτιολογώντας την απάντησή σας.

3. Έστω ότι έχουμε να υλοποιήσουμε σε C++ μία προσομοίωση ενός θερινού σινεμά. Έστω ότι η προσομοίωση αυτή θεωρεί τις εξής κλάσεις:

- κλάση: “Ταμείο” που προσομοιώνει την εξυπηρέτηση στο ταμείο του θερινού
- κλάση: “Bar” που προσομοιώνει την εξυπηρέτηση στο μπαρ του θερινού
- κλάση: “Cinema” που προσομοιώνει την εξυπηρέτηση στο θερινό σινεμά.

Η κλάση “Ταμείο”:

- έχει μία ουρά θεατών (**customers**)

Η κλάση “Ταμείο” χαρακτηρίζεται από την εξής συμπεριφορά:

- Αρχικά το ταμείο είναι άδειο.
- Στο ταμείο, ένας θεατής προστίθεται (**add**) στο τέλος της ουράς του.
- Στο ταμείο εξυπηρετείται (**serve**) ο πρώτος θεατής κάθε φορά, διαγράφοντάς τον από την ουρά.

Η κλάση “Bar”:

- έχει μία ουρά θεατών (**customers**)
- αποθηκεύει το τζίρο των πωλήσεων (**M**)

Η κλάση “Bar” χαρακτηρίζεται από την εξής συμπεριφορά:

- Αρχικά το μπαρ είναι άδειο.
- Στο μπαρ, ένας θεατής προστίθεται (**add**) στο τέλος της ουράς του.
- Στο μπαρ εξυπηρετείται (**serve**) ο πρώτος θεατής κάθε φορά, συμβάλλοντας στο τζίρο με το ποσό αγοράς του, και διαγράφεται από την ουρά.

Η κλάση “Cinema”:

- έχει ένα ταμείο (**cashier**)
- έχει ένα μπαρ (**bar**)
- έχει μια σταθερά που αντιστοιχεί στο αντίτιμο του εισιτηρίου (**ticket_price**)
- αποθηκεύει τον τρέχοντα συνολικό τζίρο του (**M**)

Η κλάση “Cinema” χαρακτηρίζεται από την εξής συμπεριφορά:

- Αρχικά, ανατίθεται το αντίτιμο του εισιτηρίου του και ο συνολικός τζίρος είναι μηδέν.
- Κάνει εξυπηρέτηση, δημιουργώντας θεατή και προσθέτοντάς τον στο ταμείο. Κατόπιν, εξυπηρετεί το ταμείο, αυξάνοντας το συνολικό τζίρο κατά το αντίτιμο του εισιτηρίου. Αν ο θεατής που εξυπηρετήθηκε θέλει να αγοράσει κάτι από το μπαρ, τον προσθέτει στο μπαρ και τυχαία επιλέγει να εξυπηρετήσει το μπαρ ή όχι (**serve**)

Ο θεατής χαρακτηρίζεται από το ποσό αγοράς του από το μπαρ. Το ποσό αυτό δημιουργείται τυχαία κατά τη δημιουργία του θεατή. Αν δεν θέλει να αγοράσει κάτι από το μπαρ, το ποσό αυτό είναι μηδέν.

Να υλοποιήσετε σε C++ τις παραπάνω κλάσεις, κάνοντας παραδοχές στις προδιαγραφές, όπου (και αν) το θεωρείτε απαραίτητο, τεκμηριώνοντάς τις.

Σημείωση: Στα θέματα στα οποία σας ζητείται να βρείτε αποτελέσματα, όποτε αυτά επαναλαμβάνονται ή η αιτιολόγηση είναι ίδια με κάτι που ήδη έχει αιτιολογηθεί, μην επαναλαμβάνετε. Απλά κάντε τη σχετική αναφορά ομοιότητας.