ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
Κ10: ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ
Εξετάσεις 7 Φεβρουαρίου 2019

1. (α′) Δώστε δύο λόγους για τους οποίους η γλώσσα Java μπορεί να θεωρηθεί πιο "Αντικειμενοστραφής" γλώσσα από ότι η C++.

   (β′) Δίδεται το παρακάτω πρόγραμμα Java:

```
public class Main{
  public static void main(String[] args){
    System.out.println("Initial A count: ");
    A.print_count();
    System.out.println("Initial B count: " );
    B.print_count();

    A a1 = new A();
    a1.print(); a1.print_count(); a1.scope();
    B b1 = new B();
    b1.print(); b1.print_count(); b1.scope();
    a1 = b1;
    a1.print(); a1.print_count(); a1.scope(); } }

class A{
   private int data;
   private static int count;

   static {System.out.println("Initializing A ");}

   public A(){ this(10); }
   public A(int i) { data = i;
            System.out.println("An A just constructed with data: " + data); }
   public void print() {
                  System.out.println("Printing message for A data: " + data);
                  count++; }
   public static void print_count() { System.out.println(count); }
   public void scope() { System.out.println("Being in A's scope!"); } }

class B extends A{
   private int data;
   private static int count;

   static {System.out. println("Initializing B ");}

   public B() { this(100,200); }
   public B(int i, int j) { super(i);
            data = j;
            System.out.println("A B just constructed with data: " + data); }
   public void print() { super.print();
                  System.out.println("Printing message for B data: " + data);
                  count++; }
   public static void print_count() { System.out.println(count); }
   public void scope() { System.out.println("Being in B's scope!"); } }
```

Επίσης, δίδεται και το παρακάτω πρόγραμμα C++:

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

class A{
   int data;
   static int count;
  public:
   A(int i = 10): data(i) { cout << "An A just constructed with data: "
                                 << data << endl; }
   void print() { cout << "Printing message for A data: "
                       << data << endl;
                  count++; }
   static void print_count() { cout << count << endl; }
   void scope() { cout<< "Being in A's scope!" << endl; } };

class B : public A{
   int data;
   static int count;
  public:
   B(int i = 100, int j = 200 ): A(i), data(j)
         { cout << "A B just constructed with data: "
                << data << endl; }
   void print() {  A::print();
                   cout << "Printing message for B data: "
                        << data << endl;
                   count++; }
   static void print_count() { cout << count << endl; }
   void scope() { cout<< "Being in B's scope!" << endl; } };

int A::count=0;
int B::count=0;

int main(){
   cout << "Initial A count: " << endl;
   A::print_count();
   cout << "Initial B count: " << endl;
   B::print_count();

   A* a1 = new A();
   a1->print(); a1->print_count(); a1->scope();

   B* b1 = new B();
   b1->print(); b1->print_count(); b1->scope();

   a1 = b1;
   a1->print(); a1->print_count(); a1->scope(); }
```

Δώστε τα αποτελέσματα των δύο προγραμμάτων, αιτιολογώντας την απάντησή σας. Τα αποτελέσματα των δύο προγραμμάτων διαφέρουν. Αιτιολογήστε γιατί.

2. Αιτιολογώντας την απάντησή σας, δώστε το αποτέλεσμα της εκτέλεσης του παρακάτω προγράμματος C++. Στη συνέχεια, αποσχολιάστε το σχολιασμένο κομμάτι κώδικα. Δώστε πάλι το αποτέλεσμα της εκτέλεσης, αιτιολογώντας την απάντησή σας. Κοινές εκτυπώσεις ή/και αιτιολογήσεις δεν χρειάζεται να επαναληφθούν.

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

class A{
   int data1;
   int& data2;
  public:
   A(int d1, int& d2):data1(d1), data2(d2)
       {cout << "An A just constructed" << endl;}
   ~A(){cout << "An A to be destructed with: "
             << data1 << " and " << data2 << endl;}
   A& operator=(const A& a)
               { cout<< "I am performing an A assignment" << endl;
                 data1 = a.data1; data2 = a.data2;
                 return *this;}
   void inc() { data1++; data2++; }
   void print() {cout << data1 << "    " << data2 << endl;} };

class B{
   int data1;
   int* data2;
   A& data3;
  public:
   B(int d1, int* pd2, A& a):data1(d1), data2(pd2), data3(a)
       {cout << "A B just constructed" << endl;}
   ~B(){cout << "A B to be destructed with: "
             << data1 << " and " << *data2 << " and " << endl;
             data3.print(); }
   B& operator=(const B& b)
               { cout<< "I am performing a B assignment" << endl;
                 data1 = b.data1; data2 = b.data2; data3 = b.data3;
                 return *this;}
   void inc() { data1++; ++(*data2); data3.inc(); } };

int main(){
   int i1 = 10; int i2 = 20;
   A a1(i1, i1); A a2(i2, i2);
   B b1(i1, &i2, a1); B b2(i2, &i1, a2); B b3(b1);

   a1.inc(); a2.inc();
   b1.inc(); b2.inc(); b3.inc();
cout<< "i1=" << i1 << "    " << "i2=" << i2 << endl;
/*
   b2 = b3; b3.inc();
cout<< "i1=" << i1 << "    " << "i2=" << i2 << endl;
*/
}
```

3. Υλοποιήστε σε C++ την πληροφορία ενός ταξιδιωτικού πρακτορείου (TravelAgency). Το ταξιδιωτικό πρακτορείο έχει πελάτες (Customer) και προσφέρει τουριστικά πακέτα (Package_tour). Το τουριστικό πακέτο χαρακτηρίζεται από μια σειρά πόλεων (City), συνοδευόμενες από τον αριθμό των διανυκτερεύσεων σε κάθε μία από αυτές, και τη σειρά από τις συνδέσεις μετακίνησης μεταξύ πόλεων (Travel). Το πρακτορείο δίδει τιμές στα πακέτα. Η τιμή ενός πακέτου προκύπτει από το κόστος του πακέτου επαυξημένη με ένα ποσοστό που αναλογεί στο κέρδος του πρακτορείου. Το κόστος του πακέτου προκύπτει από το κόστος των συνδέσεων μετακίνησης από πόλη σε πόλη επαυξημένο με το συνολικό κόστος της διαμονής στις πόλεις. Ανά πάσα στιγμή, το πρακτορείο μπορεί να υπολογίσει το συνολικό κέρδος του από τα πακέτα που έχει πουλήσει.

   Ένας *πελάτης* έχει μια ταυτότητα και χαρακτηρίζεται από το σύνολο των πακέτων που έχει αγοράσει. Αρχικά έχει μόνο ταυτότητα και δεν έχει αγοράσει κανένα πακέτο. Αγοράζει ένα πακέτο (`buy_package`), προσθέτοντας το πακέτο στα πακέτα που έχει αγοράσει. Δίνεται η δυνατότητα να εκτυπωθούν τα πακέτα που έχει αγοράσει (`print`).

   Μια *πόλη* έχει ένα όνομα κι ένα κόστος διανυκτέρευσης και αρχικοποιείται με όλη την πληροφορία αυτή. Μπορούμε να ανακτήσουμε το όνομά της (`get_name`). Επίσης, μπορούμε να ανακτήσουμε το κόστος διανυκτέρευσης (`get_cost`).

   Μια *σύνδεση μετακίνησης* μεταξύ πόλεων χαρακτηρίζεται από τις δύο πόλεις που συνδέει και το κόστος της. Η σύνδεση αυτή μπορεί να είναι είτε *αεροπορική πτήση* είτε *ακτοπλοϊκή σύνδεση* είτε *δρομολόγιο τρένου* είτε *δρομολόγιο λεωφορείου*. Στην περίπτωση της αεροπορικής πτήσης συμπεριλαμβάνεται κι η αεροπορική εταιρεία. Στην περίπτωση της ακτοπλοϊκής σύνδεσης, συμπεριλαμβάνεται και το όνομα του πλοίου. Όλες οι περιπτώσεις σύνδεσης μετακίνησης αρχικοποιούνται με την πληροφορία που τις χαρακτηρίζει. Πρέπει να παρέχεται η δυνατότητα να ανακτήσουμε το κόστος της (`get_cost`). Επίσης, πρέπει να δίνεται η δυνατότητα εκτύπωσης όλων των στοιχείων της (`print`). Κατά την εκτύπωση, εκτυπώνεται και το μέσο μετακίνησης.

   Ένα *πακέτο* χαρακτηρίζεται από τη σειρά των πόλεων που το απαρτίζουν όπου η κάθε πόλη συνοδεύεται από τον αριθμό των διανυκτερεύσεων σε αυτές —η πρώτη πόλη και η τελευταία είναι η πόλη αναχώρησης και η πόλη άφιξης, αντίστοιχα, οπότε δεν υπάρχει διανυκτέρευση σε αυτές—. Το πακέτο χαρακτηρίζεται επίσης, κι από τη σειρά των συνδέσεων μετακίνησης ανάμεσα στις πόλεις του. Ένα πακέτο αρχικοποιείται με όλη την παραπάνω πληροφορία. Για ένα πακέτο, μπορούμε να υπολογίσουμε το κόστος του (`get_cost`) αθροίζοντας τα κόστη διανυκτερεύσεων στις πόλεις που το αποτελούν καθώς και τα κόστη των μετακινήσεων μεταξύ των πόλεών του. Επίσης, μπορούμε να το εκτυπώσουμε (`print`), εκτυπώνοντας τα ονόματα των πόλεων που το απαρτίζουν, τον αριθμό διανυκτερεύσεων σε κάθε μία από αυτές και κάνοντας εκτύπωση και των συνδέσεων μετακίνησής του.

   Το *τουριστικό πρακτορείο*, αρχικά, δεν έχει κανένα πελάτη και κανένα τουριστικό πακέτο. Μπορεί να προστεθεί πελάτης (`add_customer`), προσθέτοντας τον στους ήδη υπάρχοντες. Μπορεί να προστεθεί τουριστικό πακέτο (`add_package_tour`), δημιουργώντας το και προσθέτοντάς το στα ήδη υπάρχοντα. Ένας ήδη υπάρχων πελάτης μπορεί να αγοράσει ένα πακέτο (`buy_package`). Αν δεν είναι στους ήδη υπάρχοντες, πρέπει και να προστεθεί σε αυτούς. Για ένα πακέτο, καθορίζουμε την τιμή του (`define_price`) αυξάνοντας το κόστος του κατά ένα ποσοστό κέρδους το οποίο αποτελεί σταθερά του πρακτορείου. Επίσης, μπορεί να υπολογιστεί το κέρδος που έχει το πρακτορείο με βάση τα πακέτα που έχουν αγοράσει οι πελάτες του (`evaluate_profit`). Τέλος, για έναν πελάτη, μπορεί να εκτυπώσει τα πακέτα που έχει αγοράσει (`print`).

   Υλοποιήστε την παραπάνω πληροφορία σε C++, υλοποιώντας κατάλληλες κλάσεις, ορίζοντας τα μέλη-δεδομένα που χρειάζονται και συναρτήσεις-μέλη που υλοποιούν την παραπάνω συμπεριφορά. Θεωρήστε δεδομένους τους τύπους δεδομένων που θα χρησιμοποιήσετε, δίδοντας, όμως, τις προδιαγραφές τους. (*Σημείωση:* Δεν χρειάζεται υλοποίηση συνάρτησης main).