Πανεπιστήμιο Δυτικής Αττικής
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Αντικειμενοστρεφής Προγραμματισμός

Εργαστήριο 5

Κληρονομικότητα

Η κληρονομικότητα αποτελεί ένα από τα κυριότερα χαρακτηριστικά μιας αντικειμενοστρεφούς γλώσσας προγραμματισμού. Στη C++ η κληρονομικότητα επιτυγχάνεται με το να επιτρέπεται σε μια κλάση να κληρονομεί και να διαχειρίζεται τα στοιχεία μιας άλλης κλάσης μέσα από τη διακήρυξη της. Η πρώτη κλάση ονομάζεται παραγόμενη (derived class) και η δεύτερη βασική (base class).

Η γενική μορφή της διακήρυξης μιας παραγόμενης κλάσης είναι:

class όνομα παραγόμενης κλάσης **: τύπος προσπέλασης** όνομα βασικής κλάσης{…}

Ο τύπος προσπέλασης δηλώνει τον τρόπο προσπέλασης των στοιχείων της βασικής κλάσης από την παραγόμενη και είναι ένας από τις λέξεις κλειδιά: public, protected ή private.

Στη C++ οι κλάσεις αναγνωρίζουν τρεις κατηγορίες στοιχείων : public, protected και private. Τα public στοιχεία μιας κλάσης είναι προσπελάσιμα από κάθε συνάρτηση του προγράμματος. Τα private στοιχεία είναι προσπελάσιμα μόνο από τις συναρτήσεις μέλη ή από τις φιλικές συναρτήσεις της κλάσης. Τα στοιχεία που διακηρύσσονται ως protected είναι προσπελάσιμα μόνο από τις συναρτήσεις μέλη και τις φιλικές συναρτήσεις της κλάσης καθώς και *από τις από τις αντίστοιχες συναρτήσεις των παραγόμενων από αυτή κλάσεων*.

Εάν κατά τη διακήρυξη μιας παραγόμενης κλάσης από μια βασική ο τύπος προσπέλασης είναι:

1. Public:
   Τότε όλα τα public και protected στοιχεία της βασικής κλάσης γίνονται αντίστοιχα public και protected στοιχεία της παραγόμενης κλάσης.
2. Protected:
   Τότε όλα τα public και protected στοιχεία της βασικής κλάσης γίνονται protected στοιχεία της παραγόμενης κλάσης.
3. Private:
   Τότε όλα τα public και protected στοιχεία της βασικής κλάσης γίνονται private στοιχεία της παραγόμενης κλάσης.

Σημειώνεται ότι δεν υπάρχει η δυνατότητα προσπέλασης στα private στοιχεία της βασικής κλάσης.

Ο παρακάτω πίνακας συνοψίζει τις 3x3 περιπτώσεις:

Access	Public	protected	private
Μέλη της ίδιας κλάσης	Ναι	Ναι	Ναι
Μέλη παραγόμενης κλάσης	Ναι	Ναι	Όχι
Μη μέλη	Ναι	Όχι	Όχι

Παραδείγματα:

Παράδειγμα 1

Σε αυτό το παράδειγμα, οι δύο μεταβλητές της Shape είναι public, συνεπώς είναι ορατές και από τα μέλη της Triangle και από τον «έξω κόσμο», δηλ. την main.

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

// A class for two-dimensional objects.
class Shape {
public:
  double width;
  double height;

  void showDim() {
    cout << "Width and height are " <<
              width << " and " << height << "\n";
  }
};

// Triangle is derived from Shape.
class Triangle : public Shape {
public:
  string style;

  double area() {
    return width * height / 2;
  }

  void showStyle() {
    cout << "Triangle is " << style << "\n";
  }
};

int main() {
  Triangle t1;

  t1.width = 4.0;
  t1.height = 4.0;
  t1.style = "isosceles";

  cout << "Info for t1:\n";
  t1.showStyle();
  t1.showDim();
  cout << "Area is " << t1.area() << "\n";

  return 0;
}
```

Άσκηση 1

Να δημιουργήσετε μια κλάση Insurance η οποία θα αναπαριστά ένα ασφαλιστικό συμβόλαιο. Η κλάση περιγράφεται από τις ακόλουθες μεταβλητές:

- τα στοιχεία του πελάτη (όνομα, επίθετο, ΑΦΜ),
- την ηλικία του πελάτη
- το μέγιστο ποσό κάλυψης (π.χ. 500.000 ευρώ)

Επίσης θα πρέπει να ορίσετε και μια συνάρτηση display( ) για την εκτύπωση των στοιχείων του συμβολαίου στην οθόνη.

Όλες οι μεταβλητές της κλάσης να είναι public.

Άσκηση 2

Σε συνέχεια της 1ης άσκησης, να υλοποιήσετε δύο κλάσεις που επεκτείνουν την λειτουργικότητα της InsurancePolicy:

- LifeInsurance
- AutoInsurance

Η κλάση AutoInsurance προσθέτει και άλλη μια παράμετρο, την **ηλικία του αυτοκινήτου**.

Και οι δύο κλάσεις υλοποιούν μια συνάρτηση (**calculateCost**) για τον υπολογισμό του ετήσιου κόστους της ασφάλισης.

Στην κλάση LifeInsurance, το κόστος υπολογίζεται σύμφωνα με την παρακάτω σχέση:
Δόση = ηλικία + 2*κάλυψη

Στην κλάση AutoInsurance, ο τύπος γίνεται :
Δόση = -1*ηλικία + 10*κάλυψη+ 10* ηλικία αυτοκινήτου

Παρατηρήστε ότι, στην περίπτωση της Ασφάλειας αυτοκινήτου, ο συντελεστής της ηλικίας είναι αρνητικός, καθώς η εταιρία θεωρεί ότι όσο μεγαλώνει ο οδηγός τόσο πιο ασφαλής γίνεται η οδήγησή του, με αποτέλεσμα την μείωση του κόστους ασφάλισης.

Και εδώ, όλες οι μεταβλητές μπορούν να είναι public.

Θα πρέπει λοιπόν:

- Να υλοποιήσετε τις δύο νέες κλάσεις
- Να υλοποιήσετε την συνάρτηση calculateCost και στις 2 κλάσεις
- Να υπερφορτώσετε την συνάρτηση display( ) και στις 2 κλάσεις ώστε να εμφανίζει και το κόστος
- Στην main, να δημιουργήσετε αντικείμενα και από τις δύο κλάσεις και να εκτυπώσετε τις πληροφορίες τους στην οθόνη.

Σημείωση: Η ασφαλιστική κάλυψη μετριέται σε χιλιάδες ευρώ

Παράδειγμα 2

Σε αυτό το παράδειγμα, οι δύο μεταβλητές της Shape είναι private (by default), συνεπώς δεν είναι ορατές ούτε από τα μέλη της Triangle ούτε βεβαίως από τον «έξω κόσμο».

```
// Access to private members is not granted to derived classes.
class Shape {
  // these are now private
  double width;
  double height;
public:
  void showDim() {
    cout << "Width and height are " <<
              width << " and " << height << "\n";
  }
};

// Triangle is derived from Shape.
class Triangle : public Shape {
public:
  string style;

  double area() {
```

```
    return width * height / 2;
  }

  void showStyle() {
    cout << "Triangle is " << style << "\n";
  }
};
```

Παράδειγμα 3

Μια λύση για να αποκτήσουμε πρόσβαση στις δύο μεταβλητές της Shape είναι να χρησιμοποιήσουμε accessor functions (που θα είναι βεβαίως public).

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

// A class for two-dimensional objects.
class Shape {
  // these are private
  double width;
  double height;
public:

  void showDim() {
    cout << "Width and height are " <<
              width << " and " << height << "\n";
  }

  // accessor functions
  double getWidth() { return width; }
  double getHeight() { return height; }
  void setWidth(double w) { width = w; }
  void setHeight(double h) { height = h; }
};

// Triangle is derived from Shape.
class Triangle : public Shape {
public:
  string style;

  double area() {
    return getWidth() * getHeight() / 2;
  }

  void showStyle() {
    cout << "Triangle is " << style << "\n";
  }
};

int main() {
  Triangle t1;

  t1.setWidth(4.0);
  t1.setHeight(4.0);
  strcpy(t1.style, "isosceles");

  cout << "Info for t1:\n";
  t1.showStyle();
  t1.showDim();
  cout << "Area is " << t1.area() << "\n";

  return 0;
```

```
}
```

Παράδειγμα 4

Η άλλη λύση για να αποκτήσουμε πρόσβαση στις δύο μεταβλητές της Shape είναι να τις δηλώσουμε ως protected που είναι μια ενδιάμεση κατάσταση μεταξύ public και private. Πιο συγκεκριμένα, μια τέτοια μεταβλητή είναι με ορατή από την θυγατρική κλάση αλλά όχι από την main.

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

// A class for two-dimensional objects.
class Shape {
protected:
  double width;
  double height;

public:
  // accessor functions
  double getWidth() { return width; }
  double getHeight() { return height; }
  void setWidth(double w) { width = w; }
  void setHeight(double h) { height = h; }

  void showDim() {
    cout << "Width and height are " <<
             width << " and " << height << "\n";
  }
};

// Triangle is derived from Shape.
class Triangle : public Shape {
public:
  char style[20];

  double area() {
    return width * height / 2;
  }

  void showStyle() {
    cout << "Triangle is " << style << "\n";
  }
};

int main() {
  Triangle t1;

  t1.setWidth(4.0);
  t1.setHeight(4.0);
  strcpy(t1.style, "isosceles");

  cout << "Info for t1:\n";
  t1.showStyle();
  t1.showDim();
  cout << "Area is " << t1.area() << "\n";

  return 0;
}
```

Άσκηση 3
Συνεχίζοντας τις ασκήσεις 1 και 2 , να «κρύψετε» τις μεταβλητές όλων των κλάσεων ώστε να μην είναι προσβάσιμες άμεσα από τον «έξω κόσμο» και να υλοποιήσετε accessor functions όπου είναι απαραίτητο.

Παράδειγμα 5
Η αρχικοποίηση των μεταβλητών της Shape και της Triangle μπορεί (και πρέπει) να γίνει με την χρήση κατασκευαστών. Θεωρητικά, ο κατασκευαστής της Triangle θα μπορούσε να αναλάβει να αρχικοποιήσει όλες τις μεταβλητές (αφού έχει πρόσβαση σε όλες), αλλά το σωστό είναι κάθε κλάση να αρχικοποιεί τις δικές της μεταβλητές με τον δικό της constructor. Αυτό που γίνεται στο τέλος, είναι ότι ο κατασκευαστής του Triangle πρέπει να καλέσει το αντίστοιχο της Shape.
Κάτι που θέλει προσοχή εδώ είναι το ότι ο default constructor της Triangle καλεί αυτόματα τον αντίστοιχο της Shape.

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

class Shape {
  double width;
  double height;
public:

  // Default constructor.
  Shape() {
    width = height = 0.0;
  }

  // Constructor for Shape.
  Shape(double w, double h) {
    width = w;
    height = h;
  }

  // Construct object with equal width and height.
  Shape(double x) {
    width = height = x;
  }

  void showDim() {
    cout << "Width and height are " <<
              width << " and " << height << "\n";
  }

  // accessor functions
  double getWidth() { return width; }
  double getHeight() { return height; }
  void setWidth(double w) { width = w; }
  void setHeight(double h) { height = h; }
};


class Triangle : public Shape {
  char style[20]; // now private

public:

  /* A default constructor. This automatically invokes
```

```
    the default constructor of Shape. */
  Triangle() {
    strcpy(style, "unknown");
  }

  // Constructor with three parameters.
  Triangle(char *str, double w, double h) : Shape(w, h) {
    strcpy(style, str);
  }

  // Construct an isosceles triangle.
  Triangle(double x) : Shape(x) {
    strcpy(style, "isosceles");
  }

  double area() {
    return getWidth() * getHeight() / 2;
  }

  void showStyle() {
    cout << "Triangle is " << style << "\n";
  }
};

int main() {
  Triangle t1;
  Triangle t2("right", 8.0, 12.0);
  Triangle t3(4.0);

  t1 = t2;

  cout << "Info for t1: \n";
  t1.showStyle();
  t1.showDim();
  cout << "Area is " << t1.area() << "\n";

  cout << "\n";

  cout << "Info for t2: \n";
  t2.showStyle();
  t2.showDim();
  cout << "Area is " << t2.area() << "\n";

  cout << "\n";

  cout << "Info for t3: \n";
  t3.showStyle();
  t3.showDim();
  cout << "Area is " << t3.area() << "\n";
  cout << "\n";
   return 0;
}
```

Άσκηση 4
Να υλοποιήσετε τους απαραίτητους κατασκευαστές και των 3 κλάσεων.

Πολλαπλή Κληρονομικότητα

Είναι δυνατόν μια κλάση να παράγεται από περισσότερες από μια βασικές κλάσεις. Στην περίπτωση αυτή η νέα κλάση έχει προσπέλαση στα στοιχεία όλων των βασικών κλάσεων σύμφωνα με τους κανόνες της κληρονομικότητας.

Η γενική μορφή της διακήρυξης μιας παραγόμενης κλάσης, με τον ίδιο τύπο προσπέλασης, είναι:

> class όνομα παραγόμενης κλάσης : τύπος προσπέλασης όνομα βασικής κλάσης1,…, όνομα βασικής κλάσηςn{…}

Εάν οι βασικές κλάσεις περιέχουν κατασκευαστές που δεν δέχονται παραμέτρους, τότε η παραγόμενη κλάση δεν είναι απαραίτητο να έχει κατασκευαστή. Οι κατασκευαστές των βασικών κλάσεων θα εκτελεστούν αυτόματα με τη δημιουργία του αντικειμένου της παραγόμενης κλάσης. Η προτεραιότητα εκτέλεσης είναι από αριστερά προς τα δεξιά με τη σειρά που εμφανίζονται οι βασικές κλάσεις στη διακήρυξη της παραγόμενης.

Στην περίπτωση που οι κατασκευαστές των βασικών κλάσεων δέχονται παραμέτρους, πρέπει οπωσδήποτε η παραγόμενη κλάση να περιέχει κατασκευαστή.

Παράδειγμα

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

class A{
      protected:
                int a;
      public:
                A(int m);
};

class B{
        protected:
                  int b;
        public:
                  B(int m);
};

class C:public A, public B{
      private:
              int k;
      public:
              C(int i, int j);
              int set_k(void);
};

A::A(int m){
         a=m;
         cout<<"a="<<a<<"\n";
}

B::B(int m){
         b=m;
         cout<<"b="<<b<<"\n";
}

C::C(int i,int j):A(i),B(j){
         cout<<"To a kai to b phran times mesa apo ti C \n";
}

int C::set_k(void){
              k=a+b;
              return k;
}
```

```
int main(){
    int k,i;
    k=10;
    i=20;
    C c(k,i);
    cout<<"k="<<c.set_k()<<endl;

    system("PAUSE");
    return 1;
}
```

Υπάρχει επίσης η περίπτωση οι δύο μητρικές κλάσεις A και B να έχουν ως μέλος την ίδια παράμετρο (π.χ. `int a`) και αυτή να χρησιμοποιείται από αντικείμενα του τύπου C. Σε αυτή την περίπτωση, μιλάμε για δύο διαφορετικές μεταβλητές που έχουν όμως το ίδιο όνομα. Για να διακρίνουμε μεταξύ τους χρησιμοποιούμε τον scope operator ::

Δηλαδή: `A::i` και `B::i`