ΤΕΙ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΑΝΟΙΚΤΑ ΑΚΑΔΗΜΑΙΚΑ ΜΑΘΗΜΑΤΑ

ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ(Θ)

Ενότητα 5: ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

ΔΙΔΑΣΚΩΝ: ΠΑΡΙΣ ΜΑΣΤΟΡΟΚΩΣΤΑΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΤΕ

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.

- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο ΤΕΙ Κεντρικής Μακεδονίας» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ενότητα 5

ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

ΔΙΔΑΣΚΩΝ: ΠΑΡΙΣ ΜΑΣΤΟΡΟΚΩΣΤΑΣ

Περιεχόμενα ενότητας

1. Κληρονομικότητα
2. Ιεραρχία κληρονομικότητας
3. Σχέσεις «Is - A» και «Has - A»
4. Βασική κλάση βάσης και παραγόμενες
5. Παραδείγματα κληρονομικότητας
6. Φίλιες συναρτήσεις
7. Φίλιες κλάσεις
8. Βασικές και παραγόμενες κλάσεις
9. Ιδιότητες προστατευόμενων μεταβλητών _ μελών
10. Αλλαγή προσδιορισμού πρόσβασης
11. Επίπεδα κληρονομικότητας
12. Μελέτη περίπτωσης: Ιεραρχία κληρονομικότητας τριών επιπέδων
13. Πολλαπλή κληρονομικότητα
14. Περιεκτικότητα

Σκοποί ενότητας

Κληρονομικότητα

Η κληρονομικότητα είναι ένα από τα πιο ισχυρά χαρακτηριστικά του αντικειμενοστραφούς προγραμματισμού.

Είναι ο μηχανισμός που επιτρέπει σε μία κλάση να κληρονομεί όλη τη συμπεριφορά και τις ιδιότητες μίας άλλης κλάσης.

Η κλάση που κληρονομεί ονομάζεται παράγωγη ή απορρέουσα κλάση (*derived class*), ενώ η κλάση που παρέχει την κληρονομικότητα ονομάζεται βασική κλάση (*base class*).

Ένα από τα πλεονεκτήματα της κληρονομικότητας είναι ότι επιτρέπει την επαναχρησιμοποίηση του κώδικα. Αφού γραφεί μία βασική κλάση και γίνει εκσφαλμάτωση, δε χρειάζεται να την ξαναπειράξουμε. Μπορούμε να την προσαρμόσουμε να λειτουργεί σε διάφορες συνθήκες.

Η επαναχρησιμοποίηση υπάρχοντος κώδικα εξοικονομεί χρόνο, χρήμα και αυξάνει την αξιοπιστία του προγράμματος.

Κληρονομικότητα

Υποστηρίζονται 3 τύποι κληρονομικότητας:

(1) public

Κάθε αντικείμενο μίας παραγόμενης κλάσης είναι αντικείμενο και της βασικής κλάσης.

Αντικείμενα μίας βασικής κλάσης δεν είναι αντικείμενα της παραγόμενης κλάσης.

Παράδειγμα: Όλα τα αυτοκίνητα είναι οχήματα, αλλά δεν ισχύει το αντίστροφο.

Επιτρέπεται η προσπέλαση των μη ιδιωτικών μελών της βασικής κλάσης.

Η παραγόμενη κλάση μπορεί να επιφέρει αλλαγές στα ιδιωτικά μέλη της βασικής κλάσης, μέσω κληρονομούμενων μη ιδιωτικών μεθόδων.

(2) private

Αντίστοιχη με τη σχέση σύνθεσης *(θα μελετηθεί αργότερα)*.

(3) protected

Χρησιμοποιείται σπάνια.

Κληρονομικότητα

Ιεραρχία κλάσεων

-**Άμεση βασική κλάση**: Κληρονομείται απ' ευθείας (ιεραρχία ενός επιπέδου).

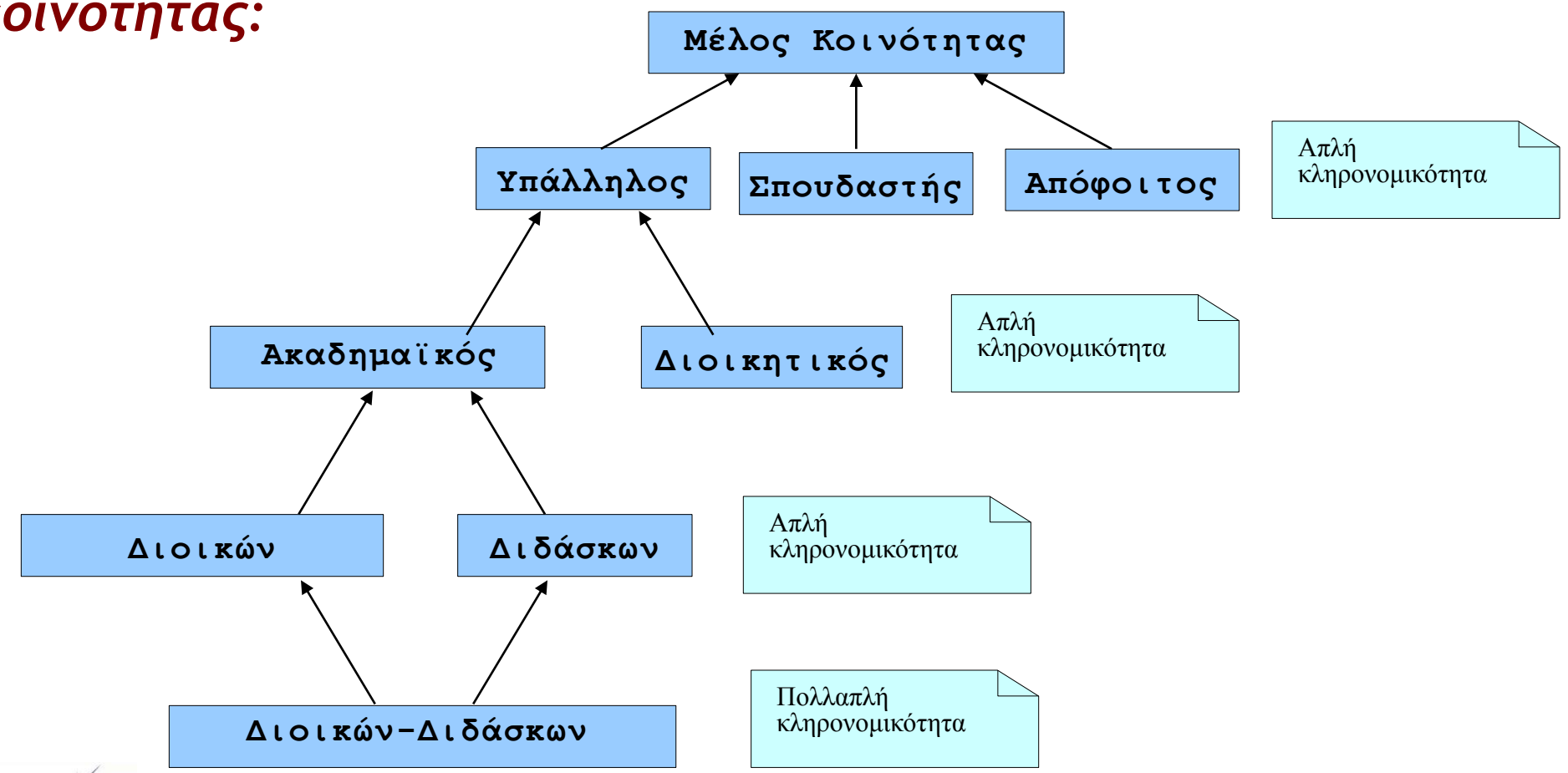
-**Έμμεση βασική κλάση**: Κληρονομείται σε ιεραρχία δύο ή περισσότερων επιπέδων.

-**Απλή κληρονομικότητα**: Μία παραγόμενη κλάση συνδέεται με μία μόνο βασική κλάση.

-**Πολλαπλή κληρονομικότητα**: Μία παραγόμενη κλάση συνδέεται με πολλές βασικές κλάσεις. Θα πρέπει να χρησιμοποιείται με προσοχή.

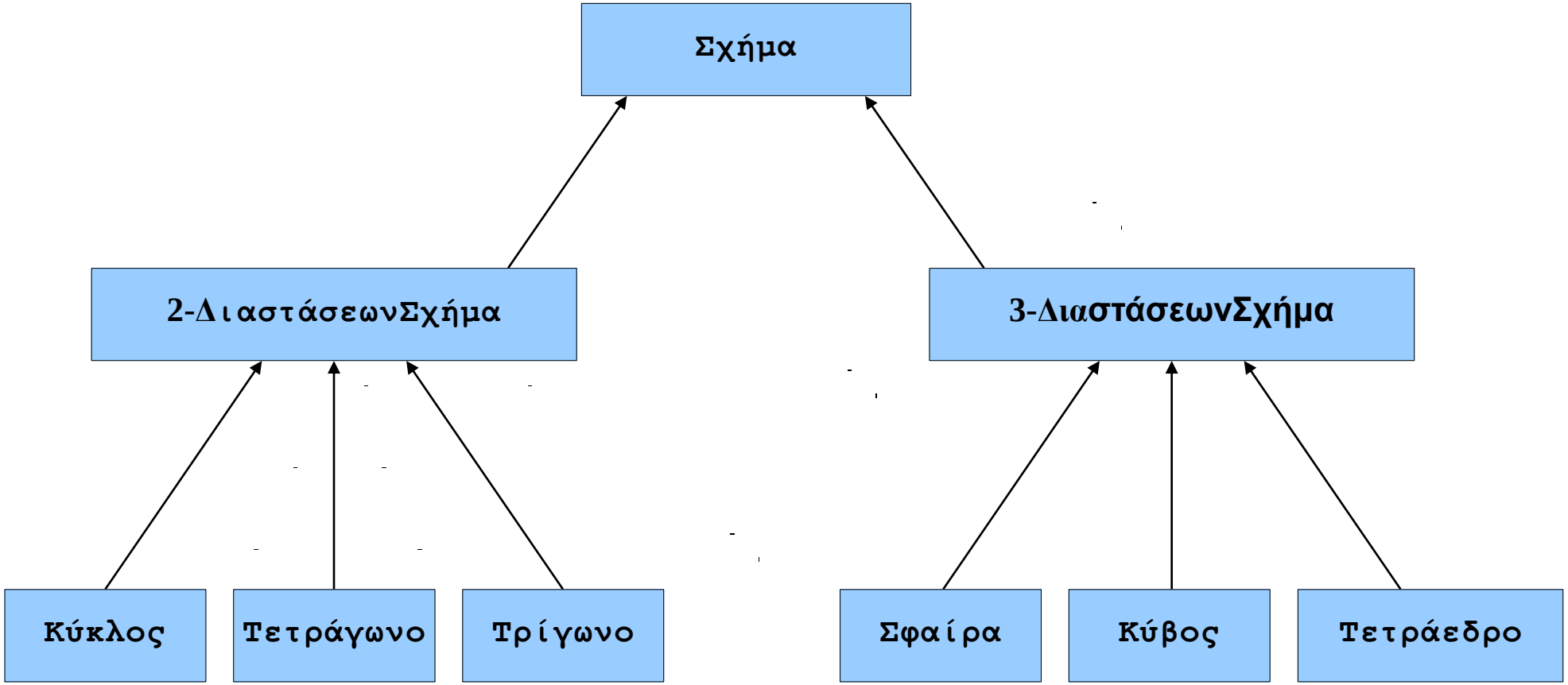
Ιεραρχία κληρονομικότητας

Ιεραρχία κληρονομικότητας για μέλη ακαδημαϊκής κοινότητας:



Ιεραρχία κληρονομικότητας

Ιεραρχία κληρονομικότητας για σχήματα:



Σχέσεις «Is - A» και «Has - A»

- **Σχέση «Is – A»: υποδηλώνει σχέση κληρονομικότητας**
  - **Ένα αντικείμενο μίας παραγόμενης κλάσης μπορούμε να το χειριστούμε και ως αντικείμενο της βασικής κλάσης.**
  - **Παράδειγμα: *Αυτοκίνητο*, *Φορτηγό*, *Μοτοσικλέτα* *Is a* *Όχημα***
    - **Οι ιδιότητες/συμπεριφορά της κλάσης Όχημα ισχύουν και για τις κλάσεις Αυτοκίνητο, Φορτηγό, Μοτοσικλέτα.**

- **Σχέση «Has - A»: υποδηλώνει σχέση σύνθεσης**
  - **Ένα αντικείμενο περιέχει ένα ή περισσότερα αντικείμενα άλλων κλάσεων ως μέλη.**
  - **Παράδειγμα: *Αυτοκίνητο* *has* 1 *Μηχανή*, 1 *Τιμόνι*, 4 *Πόρτες*κ.τ.λ.**

Βασική κλάση βάσης και παραγόμενες

- Ένα αντικείμενο μίας παραγόμενης κλάσης είναι ένα (is a) αντικείμενο και της βασικής κλάσης.
  - Παράδειγμα: ένα Τρίγωνο είναι ένα Γεωμετρικό Σχήμα.
    - Η κλάση Τρίγωνο κληρονομεί την κλάση Γεωμετρικό Σχήμα.
    - Γεωμετρικό Σχήμα: βασική κλάση (base class)
    - Τρίγωνο: παραγόμενη κλάση (derived class)
- Η βασική κλάση αντιπροσωπεύει μία γενικότερη έννοια απ' ότι η παραγόμενη κλάση (σχέση γενίκευσης/ειδίκευσης).
  - Παράδειγμα:
    - Base class: Όχημα
      - Αυτοκίνητο, Φορτηγό, Βάρκα, Ποδηλάτο, …
    - Derived class: Αυτοκίνητο
      - Το αυτοκίνητο είναι μια ειδική κατηγορία οχημάτων

Παραδείγματα κληρονομικότητας

Base class	Derived classes
Σπουδαστής	Μεταπτυχιακός Προπτυχιακός
Σχήμα	Κύκλος Τρίγωνο Ορθογώνιο
Δάνειο	Φοιτητικό Καταναλωτικό Στεγαστικό
Υπάλληλος	Πλήρους Απασχόλησης Μερικής Απασχόλησης
Λογαριασμός	Όψεως Ταμιευτηρίου

Βασικές και παραγόμενες κλάσεις

Τύπος κληρονομικότητας public

Ορίζεται με την εντολή:

```
class TwoDimensionalShape : public Shape
```

Η προσπέλαση των ιδιωτικών μελών της βασικής κλάσης δεν μπορεί να γίνει απ' ευθείας, παρόλα αυτά τα ιδιωτικά μέλη κληρονομούνται και μπορούμε να τα χειριστούμε μέσω των μη ιδιωτικών κληρονομούμενων μεθόδων.

Τα δημόσια και προστατευμένα μέλη της βασικής κλάσης κληρονομούνται και είναι δυνατή η απ' ευθείας προσπέλασή τους (με χρήση του ονόματος του μέλους). Οι φίλιες συναρτήσεις δεν κληρονομούνται.

Η μεταβλητή-μέλος αυτής της κατηγορίας μπορεί να προσπελασθεί από συναρτήσεις-μέλη μέσα στην δική της κλάση ή σε οποιαδήποτε κλάση που κληρονομεί τη δική της κλάση. Δεν μπορεί να προσπελασθεί από συναρτήσεις έξω απ' αυτές τις κλάσεις.

Συναρτήσεις που δηλώνονται στο public τμήμα μίας κλάσης με το πρόθεμα friend, αλλά ορίζονται όπως οι κανονικές συναρτήσεις εκτός της κλάσης, χωρίς το class_name ::, και προσπελαύνουν τα ιδιωτικά μέλη των αντικειμένων της κλάσης. Κατά συνέπεια δε αποτελούν συναρτήσεις-μέλη.

Φίλιες συναρτήσεις

```
#include <iostream>
using namespace std;
class Rectangle
{
private:
        int width, height;
public:
        Rectangle()
        {}
        Rectangle (int x, int y) : width(x), height(y)
        {}
        int area()
        {
                return width * height;
        }
```

Φίλιες συναρτήσεις

```
        friend Rectangle duplicate (const Rectangle &param);
}; // τέλος της κλάσης Rectangle
Rectangle duplicate(const Rectangle &param)
{
        Rectangle res;
        res.width = param.width*2;
        res.height = param.height*2;
        return res;
}  // ορισμός της φίλιας συνάρτησης χωρίς σύνδεση με την κλάση
main () {
        Rectangle foo;
        Rectangle bar (2,3);
        foo = duplicate (bar);
        cout << foo.area() << endl;
}
```

Τυπικές περιπτώσεις χρήσης των φίλιων συναρτήσεων είναι λειτουργίες που διεξάγονται ανάμεσα σε δύο διαφορετικές κλάσεις, προσπελαύνοντας ιδιωτικά ή προστατευμένα μέλη.

Φίλιες κλάσεις

Παρόμοια με τις φίλιες συναρτήσεις, μία φίλια κλάση έχει μέλη με πρόσβαση σε ιδιωτικά ή προστατευμένα μέλη έτερης κλάσης:

```
#include <iostream>
using namespace std;
class Square;
class Rectangle
{
private:
        int width, height;
public:
        int area ()
        {
                return (width * height);
        }
```

Φίλιες κλάσεις

```
        void convert (Square a);
};  // τέλος της κλάσης Rectangle
class Square
{
        friend class Rectangle;
private:
        int side;
        public:
        Square (int a) : side(a)
        {}
};  // τέλος της κλάσης Square
void Rectangle::convert (Square a)
{
        width = a.side; height = a.side;
}
```

Φίλιες κλάσεις

```
main () {
        Rectangle rect;
        Square sqr (4);
        rect.convert(sqr);
        cout << rect.area();
}
```

Σε αυτό το παράδειγμα η κλάση **Rectangle** είναι φίλια κλάση της κλάσης **Square**, επιτρέποντας στις συναρτήσεις μέλη της **Rectangle** να έχουν πρόσβαση στα ιδιωτικά και προστατευμένα μέλη της **Square**. Η **Rectangle** έχει πρόσβαση στη μεταβλητή-μέλος **Square::side**, η οποία περιγράφει την πλευρά του τετραγώνου.

Αξίζει να σημειωθεί ότι στην αρχή του προγράμματος υπάρχει κενή δήλωση της κλάσης **Square**. Η δήλωση είναι απαραίτητη διότι η κλάση **Rectangle** χρησιμοποιεί τη **Square** (ως παράμετρο στο μέλος **convert**) και η **Square** χρησιμοποιεί τη **Rectangle** (δηλώνοντάς την ως φίλια).

Φίλιες κλάσεις

Δεν υπάρχει αμοιβαιότητα στην έννοια της φίλιας κλάσης, καθώς το ποιος είναι φίλιος προς ποιον καθορίζεται ρητά και δεν υποννοείται: Στο παράδειγμα η κλάση **Rectangle** θεωρείται φίλια της κλάσης **Square**, αλλά η **Square** δε θεωρείται φίλια από τη **Rectangle**. Κατά συνέπεια, οι συναρτήσεις-μέλη της κλάσης **Rectangle** μπορούν να έχουν πρόσβαση στα ιδιωτικά και προστατευμένα μέλη της **Square** αλλά όχι τούμπαλιν. Εξυπακούεται ότι σε περίπτωση που η **Square** δηλωθεί φίλια κλάση της **Rectangle**, αποκτά τη σχετική πρόσβαση.

Η ιδιότητα των φίλιων συναρτήσεων και κλάσεων δεν είναι μεταβιβάσιμη: Η φίλια κλάση μίας κλάσης φίλιας προς τρίτη δε θεωρείται φίλια προς την τρίτη, εκτός κι αν ορισθεί ρητά.

Βασικές και παραγόμενες κλάσεις

Προσπέλαση προστατευόμενων μελών μελών: Ενδιάμεσο επίπεδο προστασίας δεδομένων μεταξύ public και private.

- **Η προσπέλαση των προστατευόμενων μελών είναι εφικτή σε:**
  - **Μέλη της βασικής κλάσης**
  - **Φίλιες συναρτήσεις της βασικής κλάσης**
  - **Μέλη της παραγόμενης κλάσης**
  - **Φίλιες συναρτήσεις της παραγόμενης κλάσης**

Ιδιότητες προστατευόμενων μεταβλητών – μελών

Πλεονεκτήματα:

- **Οι παραγόμενες κλάσεις μπορούν να αλλάξουν τις τιμές των πεδίων απ' ευθείας.**
- **Υπάρχει μικρή βελτίωση της ταχύτητας καθώς αποφεύγεται η κλήση των μεθόδων set/get.**

Μειονεκτήματα:

- **Δεν προσφέρονται για έλεγχο εγκυρότητας τιμών, καθώς η παραγόμενη κλάση μπορεί να δώσει μη-επιτρεπτή τιμή.**
- **Δημιουργία σχέσεων εξάρτησης:**
  - **Οι μέθοδοι της παραγόμενης κλάσης είναι πιο πιθανόν τώρα να εξαρτώνται από την υλοποίηση της βασικής κλάσης.**
  - **Εάν αλλάξει η υλοποίηση της βασικής κλάσης μπορεί να χρειαστεί να τροποποιήσουμε και την παραγόμενη κλάση.**

Κληρονομικότητα

```
class Account
{
protected:
    float  balance;
public:
    Account()
    {
        balance = 0;
    }
    Account(float  balance1)
    {
        balance = balance1;
    }
```

Κληρονομικότητα

```
    void  withdraw(float  money)
    {
          if (money <= balance)          balance = balance - money;
          else
              cout << "Το ποσό ανάληψης υπερβαίνει το τρέχον!" << endl;
    }
    void deposit(float  money)
    {
          balance += money;
    }
    float getBalance()
    {
          return  balance;
    }
};  // τέλος της κλάσης account
```

Κληρονομικότητα

```
class AccInter : public Account
{
public:
    void  interest()
    {
          balance += balance*0.1;
    }
};
main()
{
   AccInter  a1;
   cout << “Τρέχον ποσό λογαριασμού a1 = “ << a1.getBalance() << endl;
   a1.deposit(100);
   cout << “Τρέχον ποσό λογαριασμού a1 = “ << a1.getBalance() << endl;
   a1.interest();
   cout << “Τρέχον ποσό λογαριασμού a1 = “ << a1.getBalance() << endl;
}
```

Κληρονομικότητα

*Στο προηγούμενο παράδειγμα ορίζονται δύο κλάσεις, η **Account** και η **AccInter**.*

Η πρόταση

```
class AccInter : public Account
```

*καθορίζει τη σχέση της κληρονομικότητας. Η **AccInter** είναι η παράγωγη κλάση και κληρονομεί όλες τις δυνατότητες της βασικής κλάσης **Account**.*

*Στη **main()** δημιουργούμε ένα αντικείμενο της κλάσης **AccInter**:*

```
AccInter  a1;
```

*Στο αντικείμενο δίνεται αρχική τιμή 0, αν και δεν υπάρχει συνάρτηση εγκατάστασης στην κλάση **AccInter**. Όταν δεν υπάρχει συνάρτηση εγκατάστασης στην παράγωγη κλάση, χρησιμοποιείται η συνάρτηση εγκατάστασης από τη βασική κλάση.*

Κληρονομικότητα

- Το αντικείμενο **a1** της κλάσης **AccInter** χρησιμοποιεί τις συναρτήσεις **deposit()** και **getBalance()** της κλάσης **Account**. Κατά τον ίδιο τρόπο, όταν ο μεταγλωττιστής δε βρίσκει κάποια συνάρτηση στην παράγωγη κλάση - δηλαδή την κλάση στην οποία το **a1** είναι μέλος - αναζητά τη συνάρτηση αυτή στη βασική κλάση.

- Το δεδομένο **balance** στην κλάση **Account** έχει δηλωθεί ως **protected**. Αυτό σημαίνει ότι το δεδομένο μπορεί να προσπελασθεί από συναρτήσεις-μέλη μέσα στην δική του κλάση ή σε οποιαδήποτε κλάση που κληρονομεί τη δική του κλάση. Δεν μπορεί να προσπελασθεί από συναρτήσεις έξω απ' αυτές τις κλάσεις, όπως π.χ. η **main()**.

Κληρονομικότητα

Πίνακας προσπελασιμότητας
(η παράγωγη κλάση παράγεται με δημόσια πρόσβαση)

Μέλη βασικής κλάσης	Προσπελάσιμα από τη δική της κλάση	Προσπελάσιμα από την παράγωγη κλάση	Προσπελάσιμα από αντικείμενα έξω από την κλάση
public	ναι	ναι	ναι
protected	ναι	ναι	όχι
private	ναι	όχι	όχι

Κληρονομικότητα

Θα πρέπει να τονισθεί ότι η κληρονομικότητα δεν λειτουργεί αντίστροφα. Η βασική κλάση δεν κληρονομεί τις δυνατότητες της παράγωγης κλάσης.

Για παράδειγμα, εάν δηλώσουμε στη **main()**

```
Account a2;
```

το αντικείμενο αυτό μπορεί να χρησιμοποιήσει τις συναρτήσεις-μέλη της δικής του κλάσης, αλλά δεν μπορεί, για παράδειγμα, να χρησιμοποιήσει τη συνάρτηση **interest()** της κλάσης **AccInter**.

Κληρονομικότητα

Παράδειγμα: (1) Υλοποίηση χωρίς χρήση protected μεταβλητών-μελών

```
class rectangle
{
private:
        float side_a, side_b;
public:
        συναρτήσεις δόμησης / αποδόμησης    ...
        float area()                    {
                return side_a * side_b;
        }
        void show()             {
                cout << side_a << "x" << side_b << endl;
        }
        void set_sides(float a, float b)            {
                side_a = a;             side_b = b;
        }
};      // τέλος της βασικής κλάσης rectangle
```

theory_5_inheritence_1.cpp

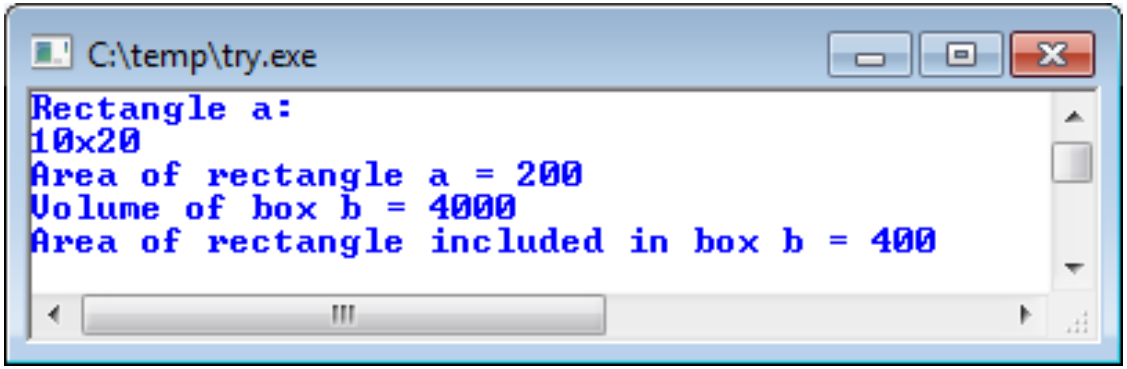
Κληρονομικότητα

```
class box : public rectangle
{
private:
    float side_c;
public:
    συναρτήσεις δόμησης / αποδόμησης      ...
    float volume()
    {
        return side_c * area();
    }
    void set_side_c(float c)
    {
        side_c = c;
    }
};  // τέλος της παράγωγης κλάσης box
```

Επειδή οι μεταβλητές – μέλη είναι ιδιωτικές, η απορρέουσα κλάση δεν έχει πρόσβαση σε αυτές, οπότε καλείται η συνάρτηση area για να συνεισφέρει στον υπολογισμό του όγκου.

Κληρονομικότητα

```
main()              {
    rectangle a;
    box b;
    a.set_sides(10,20);
    cout << “Rectangle a: ” << endl;
    a.show;
    cout << “Area of rectangle a = ” << a.area() << endl;
    b.set_sides(20,20);
    b.set_side_c(10);
    cout << “Volume of box b = ” << b.volume() << endl;
    cout << “Area of rectangle included in box b = ” << b.area();
}
```



Κληρονομικότητα

(2) Υλοποίηση με χρήση protected μεταβλητών-μελών

```
class rectangle   {
protected:
        float side_a, side_b;
public:
        ...
};          // τέλος της κλάσης rectangle
class box : public rectangle
{
private:
    float side_c;
public:
    float volume()          {
        return side_a * side_b * side_c;
    }
    ...
};  // τέλος της παράγωγης κλάσης box
```

Επειδή οι μεταβλητές – μέλη της βασικής κλάσης protected, η απορρέουσα κλάση έχει πρόσαβση σε αυτές, οπότε δεν απαιτείται κλήση της συνάρτησης area για να υπολογισθεί ο όγκος.

Κληρονομικότητα

Συναρτήσεις δόμησης της παράγωγης κλάσης

Στο προηγούμενο πρόγραμμα, εάν θελήσουμε να αποδώσουμε αρχική τιμή στο αντικείμενο **a1** της παράγωγης κλάσης αυτό δε θα είναι εφικτό, γιατί ενώ ο μεταγλωττιστής χρησιμοποιεί μία συνάρτηση δόμησης από τη βασική κλάση όταν είναι χωρίς ορίσματα, δεν μπορεί να το κάνει για συναρτήσεις με ορίσματα. Για να το πετύχουμε αυτό, **πρέπει να γράψουμε συναρτήσεις δόμησης για την παράγωγη κλάση.**

```
class AccInter : public Account
{
public:
        AccInter() : Account()
        {}
        AccInter(float  bal) : Account(bal)
        {}
        void  interest()
        {
                balance += balance*0.1;
        }
};
```

Η τιμή για την bal της AccInter θα αναζητηθεί στην bal της Account.

Κληρονομικότητα

Οπότε τώρα μπορούμε να δημιουργήσουμε αντικείμενα της παράγωγης κλάσης και να τα αποδώσουμε αρχικές τιμές:

```
main()
{
   AccInter  a1, a2(100);
}
```

Όταν δημιουργούμε αντικείμενα στη **main()**, τότε καλούνται οι ανάλογες συναρτήσεις δόμησης για να αποδώσουν αρχικές τιμές. Με τη δήλωση

```
AccInter  a1;
```

εκτελείται η πρόταση

```
AccInter() : Account()
```

όπου η συνάρτηση δόμησης **AccInter()** καλεί την αντίστοιχη συνάρτηση δόμησης **Account()** για την αρχικοποίηση του αντικειμένου.

Κληρονομικότητα

Παρόμοια, με τη δήλωση

```
AccInter  a2(100);
```

εκτελείται η πρόταση

```
AccInter(float  bal) : Account(bal)
```

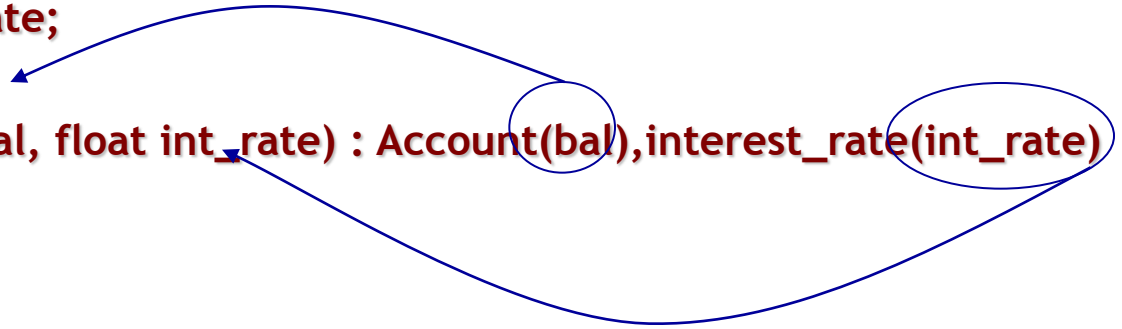
όπου καλείται η συνάρτηση δόμησης **AccInter()** με ένα όρισμα. Αυτή με τη σειρά της καλεί την αντίστοιχη συνάρτηση **Account()** με ένα όρισμα και της μεταβιβάζει το όρισμα, για να αποδοθεί ως αρχική τιμή στο αντικείμενο.

Κληρονομικότητα

Συναρτήσεις δόμησης της παράγωγης κλάσης
όταν η τελευταία έχει δικές της μεταβλητές-μέλη

Επεκτείνοντας το προηγούμενο πρόγραμμα, θεωρούμε ότι η παράγωγη κλάση έχει τη μορφή:

```
class AccInter : public Account
{
private:
        float interest_rate;
public:
        AccInter(float bal, float int_rate) : Account(bal),interest_rate(int_rate)
        {}
        void  interest()
        {
                balance += balance * interest_rate;
        }
};
```

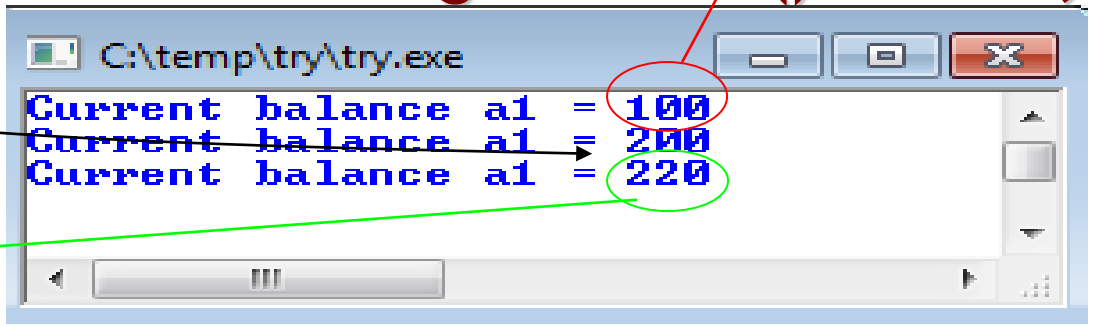


Κληρονομικότητα

```
main()
{
        AccInter  a1(100,0.1);
        cout << "Current balance a1 = " << a1.getBalance() << endl;
        a1.deposit(100);
        cout << "Current balance a1 = " << a1.getBalance() << endl;
        a1.interest();
        cout << "Current balance a1 = " << a1.getBalance() << endl;
}
```

100 + 100

*200 + (200 * 0.1)*



Κληρονομικότητα

Υπερφόρτωση συναρτήσεων-μελών

Σε μία παράγωγη κλάση μπορούμε να γράψουμε συναρτήσεις-μέλη που έχουν το ίδιο όνομα με κάποιες συναρτήσεις της βασικής κλάσης.

Στο παράδειγμα που εξετάζουμε, οι συναρτήσεις **deposit()** και **withdraw()** δέχονται ως όρισμα ένα χρηματικό ποσό και το προσθέτουν ή το αφαιρούν αντίστοιχα από το τρέχον ποσό του λογαριασμού. Δεν εξετάζεται όμως η περίπτωση κατά την οποία το ποσό που περνά ως όρισμα είναι αρνητικό. Θα μπορούσε βέβαια ο έλεγχος αυτός να προστεθεί κατευθείαν στις συναρτήσεις στη βασική κλάση. Για να μην «αλλοιώσουμε» όμως τον κώδικα της βασικής κλάσης, θα τοποθετήσουμε τον έλεγχο αυτό στην παράγωγη κλάση:

Κληρονομικότητα

```
class AccInter : public Account
{
public:
    AccInter() : Account()
    {}
    AccInter(float  bal) : Account(bal)
    {}
    void  interest() {
          balance += balance*0.1;
    }
    void  deposit(float  money)          {
          if (money>0)        Account::deposit(money);
          else                cout << “Το ποσό δεν είναι έγκυρο.”;
    }
     void  withdraw(float  money)        {
          if (money>0)        Account::withdraw(money);
          else                cout << “Το ποσό δεν είναι έγκυρο.”;
    }
};
```

Κληρονομικότητα

```
main()
{
   AccInter  a1;

   a1.deposit(100);
   cout << “Τρέχον ποσό λογαριασμού a1 = “ << a1.getBalance();
   a1.deposit(-10);
   cout << “Τρέχον ποσό λογαριασμού a1 = “ << a1.getBalance();
   a1.withdraw(50);
   cout << “Τρέχον ποσό λογαριασμού a1 = “ << a1.getBalance();
   a1.withdraw(-5);
   cout << “Τρέχον ποσό λογαριασμού a1 = “ << a1.getBalance();
}
```

Κληρονομικότητα

Όταν εκτελείται η πρόταση

```
a1.deposit(100);
```

καλείται η συνάρτηση **deposit()**, την οποία ο μεταγλωττιστής αναζητά στην παράγωγη κλάση, τη βρίσκει και την εκτελεί. Εκεί γίνεται ο έλεγχος εάν το όρισμα είναι θετικό ποσό και εφόσον είναι εκτελείται η πρόταση

```
Account::deposit(money);
```

όπου καλείται η συνάρτηση **deposit()** της βασικής κλάσης και της μεταβιβάζεται το ποσό ως όρισμα.

Κάτι ανάλογο ισχύει και για τη συνάρτηση **withdraw()**. Γενικά μπορούμε να πούμε ότι όταν υπάρχει η ίδια συνάρτηση στη βασική και στην παράγωγη κλάση, τότε εκτελείται η συνάρτηση στην παράγωγη κλάση (για αντικείμενα της παράγωγης κλάσης). Επίσης, να τονίσουμε ότι για να γίνει η κλήση των συναρτήσεων **deposit()** και **withdraw()** της βασικής κλάσης μέσα από τις αντίστοιχες συναρτήσεις της παράγωγης κλάσης, χρησιμοποιείται ο τελεστής διάκρισης εμβέλειας **::**, αλλιώς οι συναρτήσεις θα καλούσαν τον εαυτό τους και αυτό θα οδηγούσε το πρόγραμμα σε αποτυχία.

Κληρονομικότητα

Συνδυασμοί προσπέλασης

Υπάρχουν πολλές δυνατότητες προσπέλασης. Στο πρόγραμμα που ακολουθεί μπορούμε να δούμε διάφορους συνδυασμούς προσπέλασης, για να κατανοήσουμε ποιοι συνδυασμοί είναι έγκυροι και θα λειτουργήσουν:

```
#include <iostream.h>
class A                    // βασική κλάση
{
private:
        int privdataA;
protected:
        int protdataA;
public:
        int pubdataA;
};
```

Κληρονομικότητα

Συνδυασμοί προσπέλασης

```
class B : public A      // κλάση που παράγεται δημόσια
{
public:
    void funct( )
    {
    int a;
        a = privdataA;    // λάθος: μη προσπελάσιμο
        a = protdataA;    // σωστό
        a = pubdataA;     // σωστό
    }
};
```

Κληρονομικότητα

Συνδυασμοί προσπέλασης

```
class C : private A                    // κλάση που παράγεται ιδιωτικά
{
public:
    void funct( )
    {
          int a;
    a = privdataA;    // λάθος: μη προσπελάσιμο
        a = protdataA;    // σωστό
        a = pubdataA;     // σωστό
    }
};
```

Όλα τα δημόσια και προστατευμένα μέλη της βασικής κλάσης γίνονται ιδιωτικά μέλη της παράγωγης κλάσης: ενώ είναι προσβάσιμα από τα υπόλοιπα μέλη της παράγωγης κλάσης, δεν είναι διαθέσιμα σε κώδικα εκτός της κλάσης.

Κληρονομικότητα

Συνδυασμοί προσπέλασης

```
class D : protected A     // κλάση που παράγεται προστατευμένα
{
public:
    void funct( )
    {
          int a;
    a = privdataA;    // λάθος: μη προσπελάσιμο
        a = protdataA;    // σωστό
        a = pubdataA;     // σωστό
    }
};
```

Όλα τα δημόσια και προστατευμένα μέλη της βασικής κλάσης γίνονται προστατευμένα μέλη της παράγωγης κλάσης. Η παράγωγη κλάση έχει πρόσβαση τόσο στα δημόσια όσο και στα προστατευμένα μέλη που κληρονόμησε.

Κληρονομικότητα

Προσδιοριστικό πρόσβασης κληρονομικότητας	Μέλη βασικής κλάσης	Κληρονομούνται από την παράγωγη κλάση ως ...
public	public	δημόσια
public	private	ιδιωτικά χωρίς πρόσβαση
public	protected	προστατευμένα
private	public	ιδιωτικά
private	private	ιδιωτικά χωρίς πρόσβαση
private	protected	ιδιωτικά
protected	public	προστατευμένα
protected	private	ιδιωτικά χωρίς πρόσβαση
protected	protected	προστατευμένα

Κληρονομικότητα

Συνδυασμοί προσπέλασης

Στο πρόγραμμα ορίζεται μία βασική κλάση **A** όπου δηλώνονται ιδιωτικά, προστατευμένα και δημόσια δεδομένα. Επίσης με βάση την κληρονομικότητα, ορίζονται τρεις παράγωγες κλάσεις: Η **B** που παράγεται δημόσια, η **C** που παράγεται ιδιωτικά και η **D** που παράγεται προστατευμένα.

Είδαμε πριν ότι οι συναρτήσεις-μέλη των παράγωγων κλάσεων μπορούν να προσπελάσουν προστατευμένα και δημόσια δεδομένα της βασικής κλάσης. Επίσης, αντικείμενα της παράγωγης κλάσης **δεν μπορούν** να προσπελάσουν προστατευμένα και ιδιωτικά δεδομένα της βασικής κλάσης.

Αντικείμενα της παράγωγης κλάσης μπορούν να προσπελάσουν δημόσια δεδομένα της βασικής κλάσης, **μόνον εφόσον** η κλάση παράγεται δημόσια (π.χ. η **B**).

Αντικείμενα παράγωγης κλάσης που παράγεται ιδιωτικά (π.χ. η **C**) **δεν μπορούν** να προσπελάσουν ούτε δημόσια δεδομένα της βασικής κλάσης.

Αν δε δώσουμε κάποιον καθοριστή προσπέλασης όταν δημιουργούμε μία κλάση, υποτίθεται ότι είναι ιδιωτικός (*private*).

Αλλαγή προσδιορισμού πρόσβασης

Για να αλλαχθεί ο προσδιορισμός πρόσβασης για κάποιο μέλος της βασικής κλάσης (π.χ. μία συνάρτηση), θα πρέπει να γίνει η ακόλουθη προσθήκη στην παράγωγη κλάση:

```
class rectangle   {
private:
        float side_a, side_b;
public:
        float area()        {
                return side_a * side_b;
        }
        void show()         {
                cout << side_a << side_b << endl;
        }
        void set_sides(float a, float b)      {
                side_a = a;                   side_b = b;
        }
};
```

Έστω ότι επιδιώκεται η αλλαγή στον προσδιορισμό πρόσβασης της συνάρτησης **area**, ώστε να καταστεί ιδιωτική.

Αλλαγή προσδιορισμού πρόσβασης

Η αλλαγή προσδιορισμού γίνεται μέσα από την παρ

```
class box : public rectangle
{
private:
        float side_c;
        using rectangle :: area;
public:
        float volume()
        {
                return area() side_c;
        }
        void set_sides(float c)
        {
                side_c = c;
        }
};
```

Το προσδιοριστικό **using** χρησιμοποιείται μέσα στον χώρο των ιδιωτικών δηλώσεων, για να δηλώσει ότι η συνάρτηση **area()** της κλάσης **rectangle** θα κληρονομηθεί ως ιδιωτική κι όχι ως δημόσια, όπως καθορίζει το προσδιοριστικό κληρονομικότητας της κλάσης **box**.

Του προσδιοριστικού **using** της συνάρτησης-μέλος πρέπει να έπεται το όνομα της βασικής κλάσης, ακολουθούμενο από τον τελεστή **::**.

Δεν ορίζεται η λίστα παραμέτρων της συνάρτησης-μέλος, παρά μόνο το όνομά της χωρίς τις παρενθέσεις.

Επίπεδα κληρονομικότητας

Μία κλάση μπορεί να παραχθεί από μία άλλη κλάση, η οποία και η ίδια είναι παράγωγη.

```
class A
{};
class B : public A
{};
class C : public B
{};
```

Μελέτη περίπτωσης: Ιεραρχία κληρονομικότητας τριών επιπέδων

Σημείο -> Κύκλος -> Κύλινδρος

Σημείο

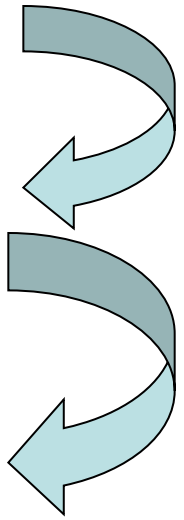
- Ζεύγος συντεταγμένων x-y

Κύκλος

- Ζεύγος συντεταγμένων x-y
- Ακτίνα

Κύλινδρος

- Ζεύγος συντεταγμένων x-y
- Ακτίνα
- Ύψος



Μελέτη περίπτωσης: Ιεραρχία κληρονομικότητας τριών επιπέδων

```
#include <cstdlib>
#include <iostream>
#include <iomanip>

using namespace std;

const double my_pi=3.14159;

class Point
{
private:
        int x;  // Τετμημένη
        int y;  // Τεταγμένη
public:
        Point(int x,int y);
        void setX(int x);
        int getX();
        void setY(int y);
        int getY();
        void print(); // εκτύπωση
                      // σημείου
}; // Τέλος της κλάσης Point
```

theory_5_hier_inh_1.cpp

Μελέτη περίπτωσης: Ιεραρχία κληρονομικότητας τριών επιπέδων

```
class Circle : public Point
{
private:
        double radius;  // Ακτίνα του κύκλου
public:
        Circle(int = 0,int = 0, double radius = 0.0);
        void setRadius(double radius);
        double getRadius();
        double getDiameter(); // Υπολογίζει τη διάμετρο
        double getCircumference();   // Υπολογίζει την περιφέρεια
        double getArea();            // Υπολογίζει το εμβαδόν
        void print();          // Εκτύπωση αντικειμένου κλάσης Circle
}; // Τέλος της κλάσης Circle
```

Κληρονομεί την Point.

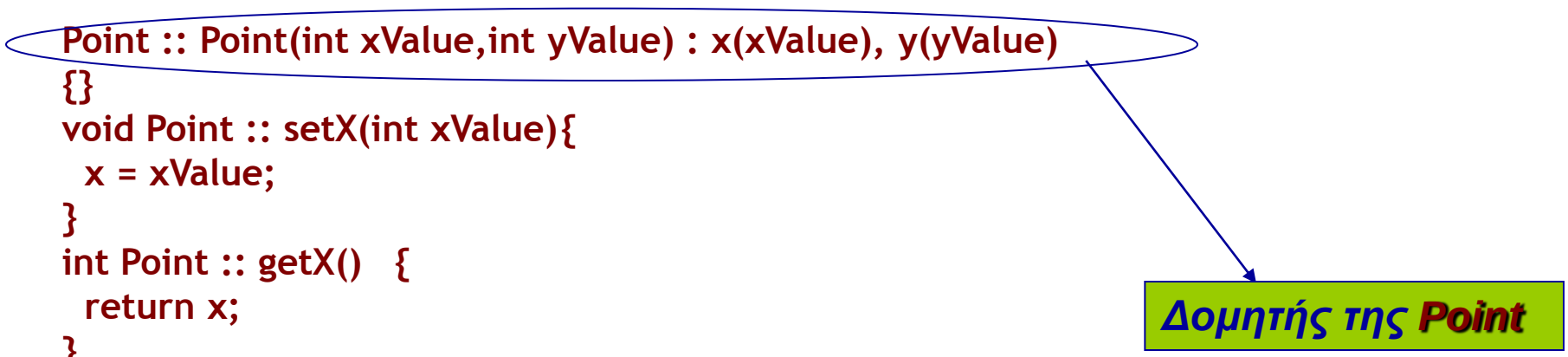
Μελέτη περίπτωσης: Ιεραρχία κληρονομικότητας τριών επιπέδων

```
class Cylinder : public Circle
{
private:
        double height;  // Ύψος του κυλίνδρου
public:
        Cylinder(int = 0,int = 0,double = 0.0, double = 0.0);
        void setHeight(double height);
        double getHeight();
        double getArea(); // Υπολογίζει το εμβαδόν
        double getVolume(); // Υπολογίζει τον όγκο
        void print();
}; // Τέλος της κλάσης Cylinder
```

Κληρονομεί την Circle, η οποία έχει κληρονομήσει την Point.

Μελέτη περίπτωσης: Ιεραρχία κληρονομικότητας τριών επιπέδων

```
Point :: Point(int xValue,int yValue) : x(xValue), y(yValue)
{}
void Point :: setX(int xValue){
  x = xValue;
}
int Point :: getX()   {
  return x;
}
void Point :: setY(int yValue){
  y = yValue;
}
int Point :: getY()   {
  return y;
}
void Point :: print()              {
  cout << '[' << x << ", " << y << ']';
}
```



Μελέτη περίπτωσης: Ιεραρχία κληρονομικότητας τριών επιπέδων

```
Circle::Circle(int xValue,int yValue,double radiusValue) :
                                        Point(xValue,yValue)
{
  setRadius(radiusValue); // ή αλλιώς ο κώδικας της setRadius
}
void Circle::setRadius(double radiusValue)
{
  if (radiusValue < 0.0) radius = 0.0;
  else radius = radiusValue;
}
double Circle::getRadius()
{
  return radius;
}
```

Δομητής της Circle, όπου αρχικοποιούνται και οι κληρονομούμενες μεταβλητές από την Point.

Μελέτη περίπτωσης: Ιεραρχία κληρονομικότητας τριών επιπέδων

```
double Circle::getDiameter()   {
        return 2 * getRadius();
}
double Circle::getCircumference()     {
        return my_pi * getDiameter();
}
double Circle::getArea()        {
        return my_pi * getRadius() * getRadius();
}
void Circle::print()      {
        cout << "Center = ";
        Point::print();
        cout << "; Radius = " << getRadius();
}
```

Εκ νέου ορισμός της συνάρτησης **print**, *ώστε να εκτυπώνεται* ***και*** *η ακτίνα του κύκλου.*

Καλείται η συνάρτηση **print** *της βασικής ως προς τη* **Circle** *κλάσης* **Point**, *με χρήση του τελεστή* **::**.

Μελέτη περίπτωσης: Ιεραρχία κληρονομικότητας τριών επιπέδων

```
Cylinder::Cylinder(int xValue, int yValue, double radiusValue, double
heightValue ) : Circle( xValue, yValue, radiusValue )
{
  setHeight( heightValue );
}
void Cylinder::setHeight(double heightValue)
{
  if (heightValue < 0.0) height = 0.0;
  else height = heightValue;
}
double Cylinder::getHeight()
{
  return height;
}
```

*Δομητής της **Cylinder**, όπου αρχικοποιούνται και οι κληρονομούμενες μεταβλητές από την **Circle** και, έμμεσα, αυτές της **Point**.*

Μελέτη περίπτωσης: Ιεραρχία κληρονομικότητας τριών επιπέδων

```
double Cylinder::getArea()
{
  return 2 * Circle::getArea() + getCircumference() * getHeight();
}

double Cylinder::getVolume()
{
  return Circle::getArea() * getHeight();
}
void Cylinder::print()
{
  Circle::print();
  cout << "; Height = " << getHeight();
}
```

Καλείται η συνάρτηση **getArea** *της βασικής ως προς τη* **Cylinder** *κλάσης* **Circle**, *με χρήση του τελεστή* **: :**.

Εκ νέου ορισμός της συνάρτησης **getArea**, *ώστε να υπολογίζεται πλέον το εμβαδόν του κυλίνδρου.*

Καλείται η συνάρτηση **print** *της βασικής ως προς τη* **Cylinder** *κλάσης* **Circle**, *με χρήση του τελεστή* **: :**.

Εκ νέου ορισμός της συνάρτησης **print**, *ώστε να εκτυπώνεται* **και** *το ύψος του κυλίνδρου.*

Μελέτη περίπτωσης: Ιεραρχία κληρονομικότητας τριών επιπέδων

```
main()
{
        Cylinder cylinder( 12, 23, 2.5, 5.7 );
        cout << "Initial cylinder's parameters\n";
        cout << "----------------------------\n";
        cout << "\tX-coordinate is " << cylinder.getX()
             << "\n\tY-coordinate is " << cylinder.getY()
             << "\n\tRadius is " << cylinder.getRadius()
             << "\n\tHeight is " << cylinder.getHeight();
        cylinder.setX(4);       // Νέα τετμημένη
        cylinder.setY(8);       // Νέα τεταγμένη
        cylinder.setRadius(5.5);  // Νέα ακτίνα
        cylinder.setHeight(10.0);   // Νέο ύψος
```

*Καλούνται οι έμμεσα κληρονο-μούμενες συναρτήσεις-μέλη της **Point**.*

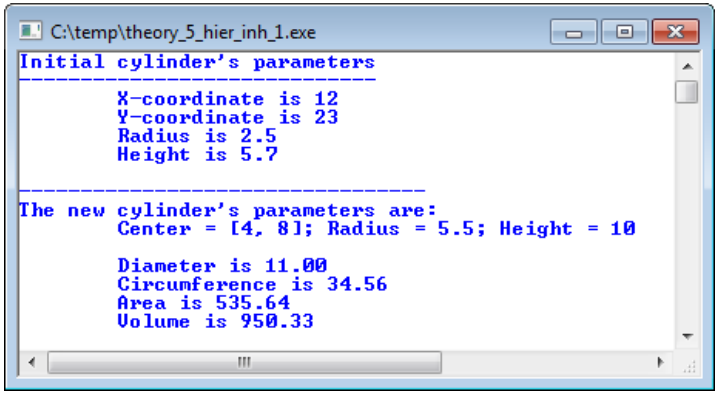
*Καλείται η άμεσα κληρονο-μούμενη συνάρτηση-μέλος της **Circle**.*

Μελέτη περίπτωσης: Ιεραρχία κληρονομικότητας τριών επιπέδων

```
    cout << "\n\n--------------------------------\n";
    cout << "The new cylinder's parameters are:\n\t";
    cylinder.print();
    cout << fixed << setprecision(2);
    cout << "\n\n\tDiameter is " << cylinder.getDiameter();
    cout << "\n\tCircumference is " << cylinder.getCircumference();
    cout << "\n\tArea is " << cylinder.getArea();
    cout << "\n\tVolume is " << cylinder.getVolume();
}
```

*Καλείται η επανακαθορισμένη συνάρτηση **print**.*

*Καλείται η επανακαθορισμένη συνάρτηση **getArea**.*



Πολλαπλή κληρονομικότητα

Μία κλάση μπορεί να παραχθεί από περισσότερες από μία κλάσεις.

```
class A
{};
class B
{};
class C : public A, public B
{};
```

Ανωτέρω ορίζονται δύο βασικές κλάσεις, οι **A** και **B**, και μία τρίτη κλάση η **C**, η οποία παράγεται και από την **A** και από την **B**.

Πολλαπλή κληρονομικότητα

Μία κλάση μπορεί να δημιουγηθεί μέσω κληρονομικότητας από περισσότερες της μίας βασικές κλάσεις, διαχωριζόμενες με κόμμα (,) στον κατάλογο των βασικών κλάσεων. Για παράδειγμα, εάν ένα πρόγραμμα διέθετε μία κλάση Output για εκτύπωση στην οθόνη και επιθυμούσαμε δύο κλάσεις Rectangle και Triangle να κληρονομούν τα μέλη της μαζί με εκείνα μίας τάξης Polygon, τότε θα γράφαμε τον ακόλουθο κώδικα:

```
#include <iostream>
using namespace std;
class Polygon
{
protected:
        int width, height;
```

Πολλαπλή κληρονομικότητα

```
public:
        Polygon (int a, int b) : width(a), height(b)
        {}
};      // τέλος της κλάσης Polygon
class Output
{
public:
        static void print (int i);
};      // τέλος της κλάσης Output
void Output :: print (int i)
{
        cout << i << endl;
}
```

Πολλαπλή κληρονομικότητα

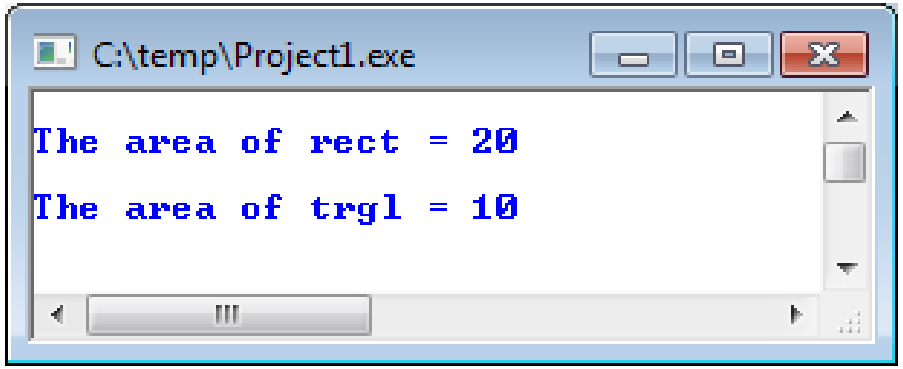
```
class Rectangle : public Polygon, public Output
{
public:
        Rectangle (int a, int b) : Polygon(a,b)
        {}
        int area ()
        {
                return width*height;
        }
};      // τέλος της κλάσης Rectangle
```

Πολλαπλή κληρονομικότητα

```
class Triangle : public Polygon, public Output
{
public:
        Triangle (int a, int b) : Polygon(a,b)
        {}
        int area ()
        {
                return width*height/2;
        }
};      // τέλος της κλάσης Triangle
```

Πολλαπλή κληρονομικότητα

```
main()
{
        Rectangle rect (4,5);
        Triangle trgl (4,5);
        cout << endl << “The area of rect = ”;
        rect.print(rect.area());
        cout << endl << “The area of trgl = ”;
        Triangle :: print(trgl.area());
}
```



theory_5_mul_inh.cpp

Περιεκτικότητα

Η κληρονομικότητα μας δίνει τη δυνατότητα να ορίσουμε μία σχέση ανάμεσα σε δύο κλάσεις **A** και **B**. Μία άλλη σχέση που μπορεί να ορισθεί καλείται **περιεκτικότητα**. Ακολούθως έχουμε την περίπτωση όπου ένα αντικείμενο της κλάσης **B** περιέχεται μέσα στην κλάση **A**.

```
class B
{
   ..................
};
class A
{
   ..................
   B  b;
    ..................
};
```

Περιεκτικότητα

```
#include <cstdlib>
#include <iostream>

using namespace std;

class Borrow
{
private:
        char studname[20];
        char bdate[10];
public:
        Borrow()
        {
                strcpy(studname, " ");
                strcpy(bdate, " ");
        }
```

theory_5_aggreg_1.cpp

Περιεκτικότητα

```
        Borrow(char studname1[], char bdate1[]) {
                strcpy(studname, studname1);
                strcpy(bdate, bdate1);
        }
        void readData()  {
                cout << "Give student name:";
                cin >> studname;
                cout << "Give date:";
                cin >> bdate;
        }
        void printData()  {
                cout << "Student name: " << studname << endl;
                cout << "Date borrowed: " << bdate << endl;
        }
};
```

Περιεκτικότητα

```
class Book
{
private:
        int number;         char title[30];
        Borrow b;
public:
        Book()    {
                number = 0;
                strcpy(title," ");
        }
        Book(int number0, char title0[], Borrow b0)
        {
                number = number0;
                strcpy(title, title0);
                b = b0;
        }
```

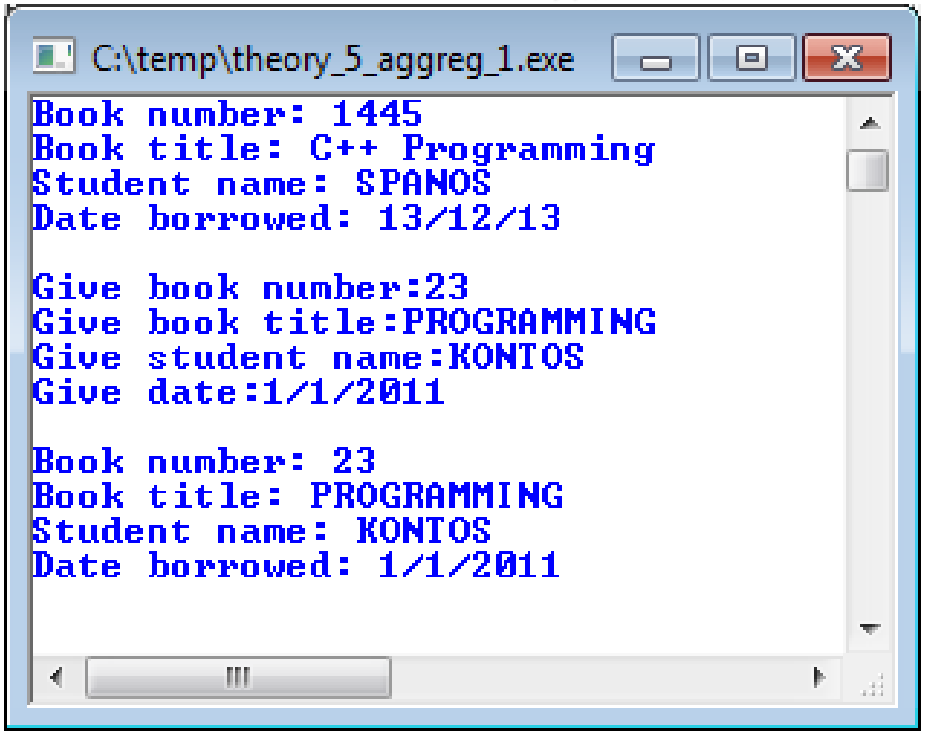
Περιεκτικότητα

```
        void readData()
        {
                cout << "Give book number:";
                cin >> number;
                cout << "Give book title:";
                cin >> title;
                b.readData();
        }
        void printData()
        {
                cout << "Book number: " << number << endl;
                cout << "Book title: " << title << endl;
                b.printData();
        }
};
```

Περιεκτικότητα

```
main()
{
        Book bk1(1445, "C++ Programming", Borrow("SPANOS",
                                                 "13/12/13")), bk2;

        bk1.printData();
        cout << endl;
        bk2.readData();
        cout << endl;
        bk2.printData();
}
```



Τέλος Ενότητας