Δείκτες

Συνέχεια

Η μεταβλητή this

- Κάθε αντικείμενο έχει μια μεταβλητή με όνομα this που είναι δείκτης προς το ίδιο το αντικείμενο.
- Η μεταβλητή this δεν δηλώνεται αλλά μπορεί να χρησιμοποιείται από οποιαδήποτε συνάρτηση-μέλος της κλάσης στην οποία ανήκει το αντικείμενο.
- Το this είναι πολύ χρήσιμο σε συγκεκριμένες περιπτώσεις
  - Στην υπερφόρτωση τελεστών (θα τη δούμε αργότερα)
  - Όταν μια συνάρτηση-μέλος καλεί μια συνάρτηση εκτός της κλάσης μπορεί να χρησιμοποιήσει το this για να της περάσει ένα δείκτη προς το συγκεκριμένο στιγμιότυπο από το οποίο την κάλεσε.

Παράδειγμα

Χωρίς χρήση του this

```
class Test {
   private:
      double x=0.0;
   public:
      double getX() {
         return x;
      }
      void setX(double x1) {
         x = x1;
      }
};
```

Η ίδια κλάση με χρήση του this

```
class Test {
   private:
      double x=0.0;
   public:
      double getX() {
         return this->x;
      }
      void setX(double x) {
         this->x = x;
      }
};
```

Στο συγκεκριμένο παράδειγμα το this δεν μας προσφέρει κάτι ουσιαστικό.
Υπάρχουν όμως περιπτώσεις που διευκολύνει πολύ τη σύνταξη του προγράμματος.

Δείκτες προς συναρτήσεις

- Είδαμε ότι ένας δείκτης δείχνει τη διεύθυνση κάποιας μεταβλητής.
- Μπορούμε να ορίζουμε και δείκτες προς συναρτήσεις.
- Στην περίπτωση αυτή ο δείκτης δείχνει τη διεύθυνση από την οποία ξεκινάει ο εκτελέσιμος κώδικας της συνάρτησης.
- Σύνταξη:
  - Δήλωση μεταβλητής

    τύπος αποτελέσματος συνάρτησης(* όνομα μεταβλητής)(τύποι ορισμάτων συνάρτησης)

    π.χ: float (*f)(float);

    Η μεταβλητή f είναι δείκτης προς μια συνάρτηση που δέχεται μια παράμετρο τύπου float και επιστρέφει αποτέλεσμα τύπου float.
  - Ανάθεση τιμής σε μεταβλητή

    f = sqrt;

    Η μεταβλητή f δείχνει στην αρχή της συνάρτησης sqrt
  - Κλήση συνάρτησης μέσω δείκτη

    y = (*f)(24.0);

    Καλείται η συνάρτηση που δείχνει η μεταβλητή f περνώντας ως παράμετρο την τιμή 24.0 και το αποτέλεσμα αποθηκεύεται στην μεταβλητή y.
- Μπορούμε να ορίζουμε πίνακες από δείκτες προς συναρτήσεις.
- Δεν μπορούμε να κάνουμε πράξεις με δείκτες προς συναρτήσεις.

Παράδειγμα:

Έστω ότι θέλουμε να υπολογίζουμε την τιμή της έκφρασης (f(x1)+f(x2))/2 για διάφορες μαθηματικές συναρτήσεις f

Για να λύσουμε το πρόβλημα θα φτιάξουμε μια συνάρτηση C++ με όνομα mid που θα δέχεται τρεις παραμέτρους εισόδου:
(α) Έναν δείκτη f προς μια συνάρτηση που δέχεται μια παράμετρο τύπου double και επιστρέφει αποτέλεσμα τύπου double.
(β) Δύο τιμές double με ονόματα x1 και x2.
Το αποτέλεσμα της συνάρτησης θα είναι τύπου double και θα ισούται με την τιμή (f(x1)+f(x2))/2 που θέλουμε να υπολογίσουμε.

Η συνάρτηση main θα καλεί την mid περνώντας της μια διαφορετική συνάρτηση κάθε φορά.

```
#include <iostream>
#include <cmath>

using namespace std;

double mid(double (*f)(double), double x1, double x2) {
    return ((*f)(x1)+(*f)(x2))/2;
}
int main()
{
    cout << "Displaying (f(1)+f(3))/2 where f is:\n";
    cout << "Square root: " << mid(sqrt,1.0,3.0) << endl;
    cout << "Logarithm: " << mid(log, 1.0, 3.0) << endl;
    cout << "Sine: " << mid(sin, 1.0, 3.0) << endl;
    return 0;
}
```

Δείκτες προς μέλη

- Μια ειδική κατηγορία δεικτών που δεν δείχνουν προς συγκεκριμένες μεταβλητές ή συναρτήσεις αλλά προς μεταβλητές-μέλη ή συναρτήσεις-μέλη κάποιας κλάσης.
- Μπορούν να χρησιμοποιηθούν για πρόσβαση στη μεταβλητή-μέλος ή τη συνάρτηση-μέλος οποιουδήποτε στιγμιοτύπου της κλάσης.
- Δηλώνονται όπως οι συνηθισμένοι δείκτες, μόνο που πριν από το * μπαίνει το όνομα της κλάσης ακολουθούμενο από τον τελεστή ::
  - Παράδειγμα:

    int c::*a;

    Η μεταβλητή a είναι δείκτης προς μια ακέραια μεταβλητή-μέλος της κλάσης c.
- Παίρνουν ως τιμή τη διεύθυνση μιας μεταβλητής ή συνάρτησης μέλους της κλάσης (όχι κάποιου στιγμιοτύπου)
  - Παράδειγμα:

    a = &c::x; //Ο δείκτης a δείχνει τη μεταβλητή-μέλος x της κλάσης c.
- Χρησιμοποιούνται με τον τελεστή .* σε οποιοδήποτε στιγμιότυπο της κλάσης.
  - Παράδειγμα:

    c c1, c2; //Δήλωση δύο στιγμιοτύπων της κλάσης c

    c1.*a = 3; //Η μεταβλητή-μέλος x του στιγμιοτύπου c1 παίρνει την τιμή 3.

    c2.*a = 5; //Η μεταβλητή-μέλος x του στιγμιοτύπου c2 παίρνει την τιμή 5.

Κληρονομικότητα

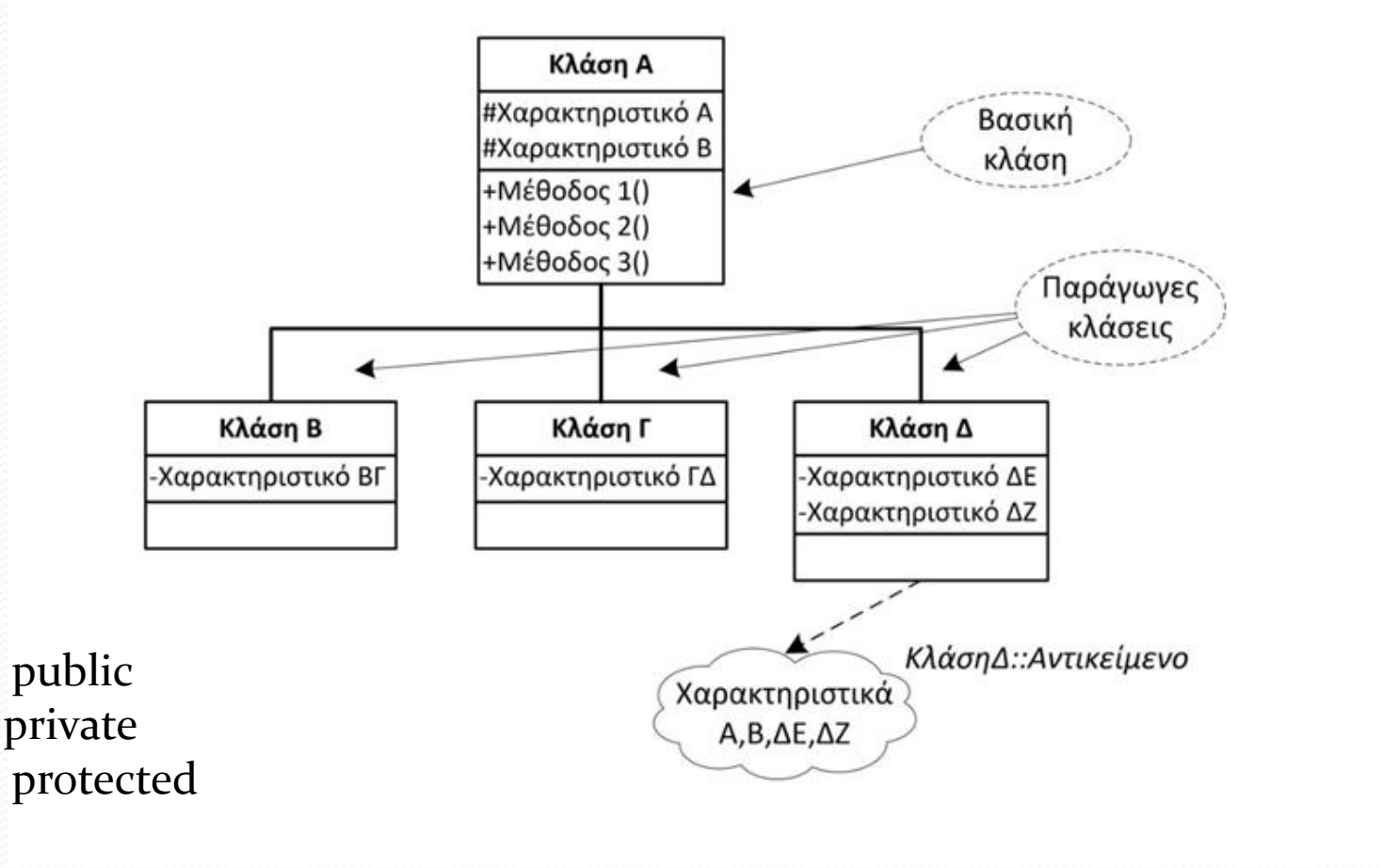
Τι είναι η κληρονομικότητα

- Ο μηχανισμός που επιτρέπει σε μια κλάση να κληρονομεί ιδιότητες και συμπεριφορές μιας άλλης κλάσης.
- Είναι ένα από τα πιο ισχυρά χαρακτηριστικά του αντικειμενοστραφούς προγραμματισμού.
- Η κλάση που κληρονομείται ονομάζεται βασική κλάση (base class) ή υπερκλάση (super class) ή κλάση γονέας (parent class).
- Η κλάση που κληρονομεί ονομάζεται παραγόμενη κλάση (derived class) ή υποκλάση (subclass) ή κλάση-παιδί (child class).

Πλεονεκτήματα της κληρονομικότητας

- Βασικό πλεονέκτημα της κληρονομικότητας είναι ότι επιτρέπει την επαναχρησιμοποίηση του κώδικα. Μετά την σύνταξη και την αποσφαλμάτωσή του, ο κώδικας της βασικής κλάσης μπορεί να χρησιμοποιηθεί από διάφορες παράγωγες κλάσεις, καθεμιά από τις οποίες είναι προσαρμοσμένη σε κάποιες ιδιαίτερες συνθήκες χρήσης του προγράμματος.
- Η επαναχρησιμοποίηση του κώδικα
  - Εξοικονομεί χρόνο και χρήμα
  - Κάνει το πρόγραμμα πιο αξιόπιστο και πιο ευανάγνωστο.

Σχηματική παράσταση κληρονομικότητας



+: public
-: private
#: protected

- Γνωρίζουμε ότι τα μέλη μιας κλάσης (μεταβλητές και συναρτήσεις) μπορούν να είναι δημόσια (public), ιδιωτικά (private) ή προστατευμένα (protected).
- Στη C++ οι χαρακτηρισμοί αυτοί χρησιμοποιούνται και για τον τύπο της κληρονομικότητας (τον τρόπο με τον οποίο παράγεται μια παράγωγη κλάση από μια βασική).
- Η δήλωση μιας παράγωγης κλάσης στον κώδικα γίνεται ως εξής:

class <παράγωγη κλάση> : <τύπος> <βασική κλάση>

- Ο τύπος της κληρονομικότητας προσδιορίζει τον τρόπο με τον οποίο κληρονομούνται τα χαρακτηριστικά της βασικής κλάσης από την παράγωγη.
  - Μπορεί να είναι public, private ή protected.

- Όπως γνωρίζουμε, γενικά ισχύουν τα εξής

Όταν τα δεδομένα δηλωθούν	**Έχει πρόσβαση**		
	Η κλάση στην οποία ανήκουν	Οι παραγόμενες κλάσεις	Κάθε μέθοδος έξω από την κλάση στην οποία ανήκουν
private	✓	-----	-----
protected	✓	✓	-----
public	✓	✓	✓

- Η γενικές αυτές αρχές εξειδικεύονται ανάλογα με τον τρόπο που κληρονομούνται τα χαρακτηριστικά.

Κληρονομικότητα δημόσιου τύπου (public)

- Τα δημόσια μέλη της βασικής κλάσης γίνονται δημόσια μέλη της παραγόμενης.
- Τα προστατευμένα μέλη της βασικής κλάσης γίνονται προστατευμένα μέλη της παραγόμενης.
- Τα ιδιωτικά μέλη της βασικής κλάσης δεν είναι προσβάσιμα από τον κώδικα της παραγόμενης.
  - Υπάρχουν όμως και στα στιγμιότυπα της παραγόμενης κλάσης και μπορεί να γίνει πρόσβαση σε αυτά μέσω μη ιδιωτικών μεθόδων της βασικής.
- Ο δημόσιος τρόπος χρησιμοποιείται συνήθως όταν η κληρονομικότητα εκφράζει μια σχέση “is-a”.
  - Ένα αντικείμενο της παράγωγης κλάσης είναι μια ειδική περίπτωση αντικειμένου της βασικής κλάσης
    - Για παράδειγμα, ένα στιγμιότυπο της κλάσης «τετράγωνο» είναι μια ειδική περίπτωση της κλάσης «επίπεδο σχήμα».

Κληρονομικότητα ιδιωτικού τύπου (private)

- Τα δημόσια και τα προστατευμένα μέλη της βασικής κλάσης γίνονται ιδιωτικά μέλη της παραγόμενης.
  - Είναι προσβάσιμα από τον κώδικα της παραγόμενης κλάσης αλλά δεν είναι προσβάσιμα από κλάσεις που παράγονται από την παραγόμενη ή από κώδικα που βρίσκεται εκτός της παραγόμενης κλάσης.
- Τα ιδιωτικά μέλη της βασικής κλάσης δεν είναι προσβάσιμα από τον κώδικα της παραγόμενης.
  - Υπάρχουν όμως και στα στιγμιότυπα της παραγόμενης κλάσης και μπορεί να γίνει πρόσβαση σε αυτά με κατάλληλη χρήση μη ιδιωτικών μεθόδων.
- Εάν δεν δηλώσουμε τον τύπο της κληρονομικότητας υπονοείται ο τύπος private.
  - π.χ. η δήλωση class A:B {…} ισοδυναμεί με class A:private B {…}

Κληρονομικότητα προστατευμένου τύπου (protected)

- Τα δημόσια και τα προστατευμένα μέλη της βασικής κλάσης γίνονται προστατευμένα μέλη της παραγόμενης.
  - Είναι προσβάσιμα από τον κώδικα της παραγόμενης κλάσης και από κλάσεις που παράγονται από αυτήν.
  - Δεν είναι προσβάσιμα από κώδικα που βρίσκεται εκτός της παραγόμενης κλάσης.
- Τα ιδιωτικά μέλη της βασικής κλάσης δεν είναι προσβάσιμα από τον κώδικα της παραγόμενης.
  - Υπάρχουν όμως και στα στιγμιότυπα της παραγόμενης κλάσης και μπορεί να γίνει πρόσβαση σε αυτά με κατάλληλη χρήση μη ιδιωτικών μεθόδων.

```
class Base {
  public:
    int x;
  protected:
    int y;
  private:
    int z;
};

class PublicDerived: public Base {
  // x is public
  // y is protected
  // z is not accessible from PublicDerived
};

class ProtectedDerived: protected Base {
  // x is protected
  // y is protected
  // z is not accessible from ProtectedDerived
};

class PrivateDerived: private Base {
  // x is private
  // y is private
  // z is not accessible from PrivateDerived
};
```

Αποκατάσταση είδους πρόσβασης

- Αναφέραμε ότι εάν μια κλάση-παιδί δηλωθεί με τύπο κληρονομικότητας “private”, τα μη ιδιωτικά μέλη της κλάσης-γονέα γίνονται ιδιωτικά μέλη της κλάσης-παιδί.
- Εάν θέλουμε, μπορούμε να επαναφέρουμε συγκεκριμένα μέλη στο είδος πρόσβασης που είχαν αρχικά στην βασική κλάση (public ή protected).
- Για να το κάνουμε αυτό, προσθέτουμε στην αντίστοιχη περιοχή δηλώσεων της παράγωγης κλάσης μια γραμμή της μορφής

  βασική κλάση::μέλος;
- Μετά τον τελεστή :: γράφουμε απλώς το όνομα του μέλους, χωρίς τύπο δεδομένων και χωρίς λίστα ορισμάτων ή παρενθέσεις (αν πρόκειται για συνάρτηση-μέλος).
- Η τεχνική αυτή δεν μπορεί να αλλάξει το είδος πρόσβασης με το οποίο είχε δηλωθεί το μέλος στην βασική κλάση. Απλώς το επαναφέρει όπως είχε δηλωθεί.

Φιλικές συναρτήσεις και κλάσεις

- Κώδικας εκτός της κλάσης είναι δυνατόν να αποκτήσει πρόσβαση σε προστατευμένα ή ιδιωτικά μέλη της κλάσης εάν ανήκει σε κλάσεις ή συναρτήσεις που έχουν δηλωθεί μέσα στην κλάση ως φιλικές.
- Οι φιλικές συναρτήσεις και κλάσεις δηλώνονται μέσα στην κλάση χρησιμοποιώντας το πρόθεμα friend.
  - Ο ορισμός τους γίνεται έξω από την κλάση.
    - Δεν αποτελούν μέλη της κλάσης.
    - Δεν μπορούν να χρησιμοποιήσουν τον δείκτη this.

```
#include <iostream>

using namespace std;

class Square;
class Rectangle {
  private:
     float width, height;
  public:
     float area() {
        return width*height;
     }
     void convert(Square a);
};  //Τέλος της κλάσης Rectangle

class Square {
   friend class Rectangle;
   private:
      float side;
   public:
      Square(float a):side(a)
      {}
} ; //Τέλος της κλάσης Square
```

Η κλάση Square δηλώνεται εδώ χωρίς να ορίζεται για να μπορεί να αναφερθεί μέσα στην κλάση Rectangle .

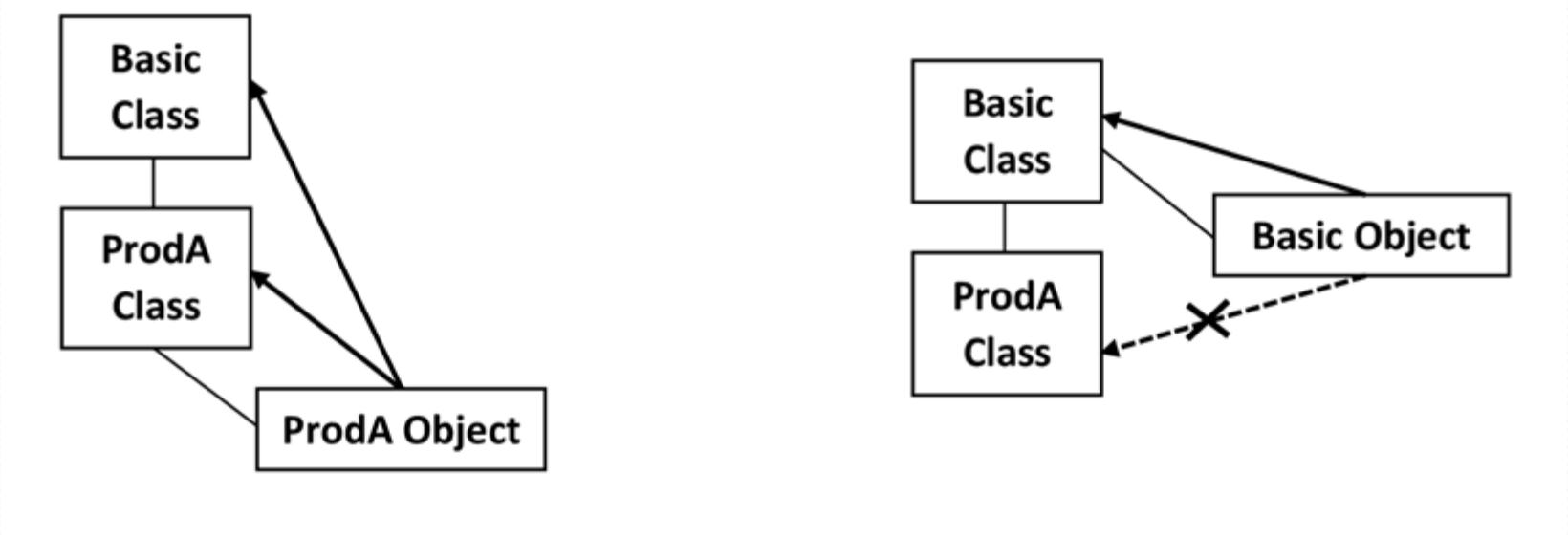
Η κλάση Rectangle δηλώνεται φιλική προς την κλάση Square και επομένως ο κώδικας της Rectangle έχει πρόσβαση σε ιδιωτικά μέλη της Square

```
void Rectangle::convert(Square a) {
    width = a.side;
    height = a.side;
}

int main()
{
    Rectangle r;
    Square s(4);
    r.convert(s);
    cout << r.area();
    return 0;
}
```

Κώδικας της Rectangle που χρησιμοποιεί ιδιωτικά μέλη της Square.

- Στον κώδικά μας πρέπει να δηλώνουμε σαφώς ποια κλάση είναι φιλική προς ποια.
  - Δεν υπάρχουν υπονοούμενες σχέσεις φιλίας.
  - Εάν η κλάση Α έχει δηλωθεί ως friend μέσα στην κλάση Β δεν σημαίνει ότι ο κώδικας της Β μπορεί να βλέπει ιδιωτικά μέλη της Α.
  - Εάν η κλάση Α είναι friend της Β και η Β είναι friend της Γ δεν συνεπάγεται ότι η Α είναι friend της Γ.
- Συναρτήσεις και κλάσεις που είναι φιλικές προς μια παράγωγη κλάση έχουν την ίδια πρόσβαση στα μέλη της βασικής κλάσης που έχει και ο κώδικας της παράγωγης κλάσης.

- Η κληρονομικότητα δεν λειτουργεί αντίστροφα.
  - Ένα στιγμιότυπο της παράγωγης κλάσης έχει τα μέλη της βασικής.
  - Ένα στιγμιότυπο της βασικής κλάσης δεν έχει τα μέλη της παράγωγης.



Παράγωγες κλάσεις με πολλές βασικές

- Μια παράγωγη κλάση μπορεί να έχει περισσότερες από μια βασικές κλάσεις.
- Ο τύπος της κληρονομικότητας δηλώνεται για κάθε βασική κλάση ξεχωριστά.

```
#include <iostream>

using namespace std;

class base1 {
protected:
   int x;
public:
   void showx() {cout << x << endl;}
};

class base2 {
protected:
   int y;
public:
   void showy() {cout << y << endl;}
};

class derived: public base1, public base2 {
public:
   void set(int i, int j) {x=i; y=j;}
};
```

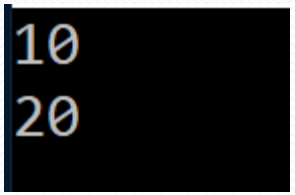
Η μεταβλητή x κληρονομείται από την κλάση base1 και η μεταβλητή y από την κλάση base2.

```
int main()
{
   derived ob;

   ob.set(10,20);
   ob.showx();
   ob.showy();
   return 0;
}
```

Κληρονομήθηκε από την κλάση base1

Κληρονομήθηκε από την κλάση base2

Μετά την εκτέλεση του προγράμματος βλέπουμε στην οθόνη τα εξής:



Συναρτήσεις κατασκευής – συναρτήσεις καταστροφής

- Τόσο η βασική κλάση όσο και η παράγωγη μπορεί να περιέχουν συναρτήσεις κατασκευής ή/και συναρτήσεις καταστροφής (constructors-destructors).
- Κατά τη δημιουργία ενός στιγμιοτύπου της παράγωγης κλάσης καλείται πρώτα η συνάρτηση κατασκευής της βασικής κλάσης (αν υπάρχει) και μετά της παράγωγης.
  - Το μέρος του αντικειμένου που περιλαμβάνει τα βασικά χαρακτηριστικά του (βασική κλάση) δημιουργείται πριν από το μέρος που περιλαμβάνει τα επιπλέον χαρακτηριστικά που προσθέτει η παράγωγη κλάση.
- Αντίθετα, κατά την καταστροφή του στιγμιοτύπου καλείται πρώτα η συνάρτηση καταστροφής της παράγωγης κλάσης και μετά της βασικής.
  - Όταν καταστρέφεται το μέρος που περιλαμβάνει τα βασικά χαρακτηριστικά, το αντικείμενο παύει να υπάρχει. Επομένως τα επιπλέον χαρακτηριστικά πρέπει ήδη να έχουν καταστραφεί.

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

class base {
public:
   base() {cout << "Constructing base" << endl;}
   ~base() {cout << "Destructing base" << endl;}
};

class derived: public base {
public:
   derived() {cout << "Constructing derived" << endl;}
   ~derived() {cout << "Destructing derived" << endl;}
};

int main()
{
   derived ob;

   return 0;
}
```

Το πρόγραμμα δεν κάνει τίποτα. Απλώς δημιουργεί ένα αντικείμενο και αμέσως το καταστρέφει.

Αποτελέσματα στην οθόνη

```
Constructing base
Constructing derived
Destructing derived
Destructing base
```

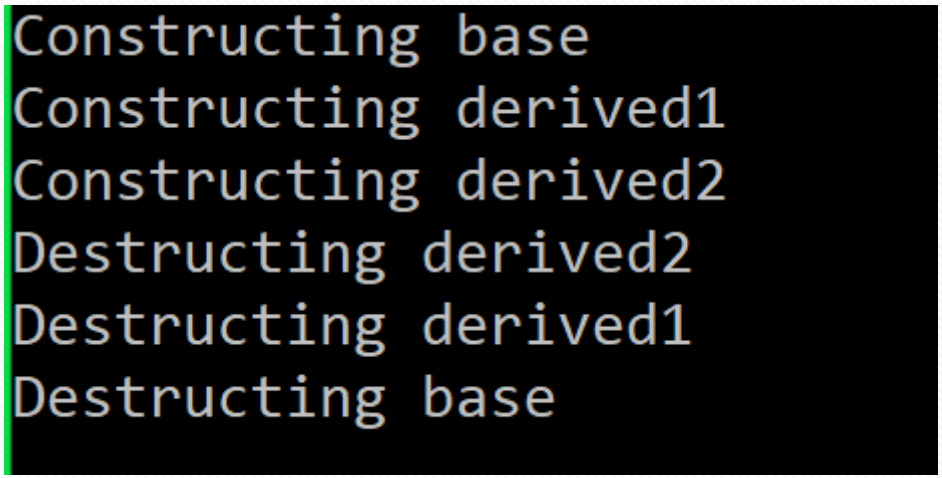
Πολλαπλά επίπεδα κληρονομικότητας

- Οι συναρτήσεις κατασκευής (εφόσον υπάρχουν) καλούνται με τη σειρά παραγωγής των κλάσεων.
  - Η συνάρτηση κατασκευής κάθε κλάσης-παιδιού καλείται μετά τη συνάρτηση κατασκευής της κλάσης-γονέα.
- Οι συναρτήσεις καταστροφής (εφόσον υπάρχουν) καλούνται με σειρά αντίστροφη από τη σειρά παραγωγής των κλάσεων.
  - Η συνάρτηση καταστροφής κάθε κλάσης-παιδιού καλείται μετά τη συνάρτηση καταστροφής της κλάσης-γονέα.

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

class base {
public:
   base() {cout << "Constructing base" << endl;}
   ~base() {cout << "Destructing base" << endl;}
};

class derived1: public base {
public:
   derived1() {cout << "Constructing derived1" << endl;}
   ~derived1() {cout << "Destructing derived1" << endl;}
};

class derived2: public derived1 {
public:
   derived2() {cout << "Constructing derived2" << endl;}
   ~derived2() {cout << "Destructing derived2" << endl;}
};
```

```
int main()
{
    derived2 ob;

    return 0;
}
```

Αποτέλεσμα στην οθόνη



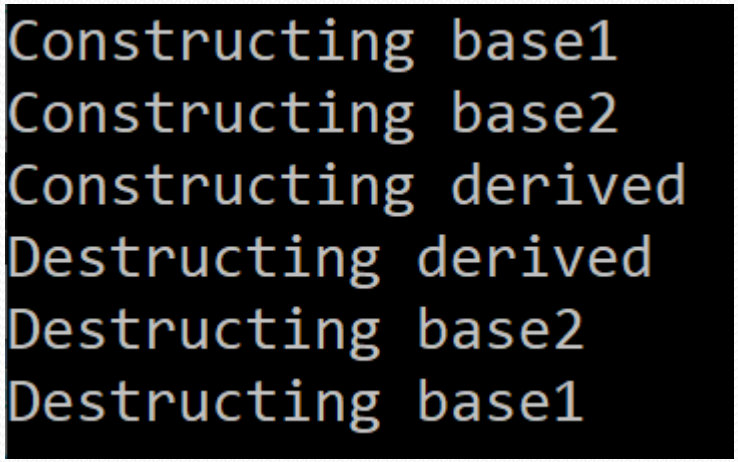
Πολλές βασικές κλάσεις

- Όταν η παράγωγη κλάση κληρονομεί περισσότερες από μια βασικές κλάσεις οι συναρτήσεις κατασκευής τους καλούνται με τη σειρά που δηλώνονται, από τα αριστερά προς τα δεξιά.
- Οι συναρτήσεις καταστροφής καλούνται με την αντίστροφη σειρά.

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

class base1 {
public:
   base1() {cout << "Constructing base1" << endl;}
   ~base1() {cout << "Destructing base1" << endl;}
};

class base2 {
public:
   base2() {cout << "Constructing base2" << endl;}
   ~base2() {cout << "Destructing base2" << endl;}
};

class derived: public base1, public base2 {
public:
   derived() {cout << "Constructing derived" << endl;}
   ~derived() {cout << "Destructing derived" << endl;}
};
```

```
int main()
{
    derived ob;

    return 0;
}
```

Αποτέλεσμα στην οθόνη



Κλήση συνάρτησης κατασκευής βασικής κλάσης με παραμέτρους

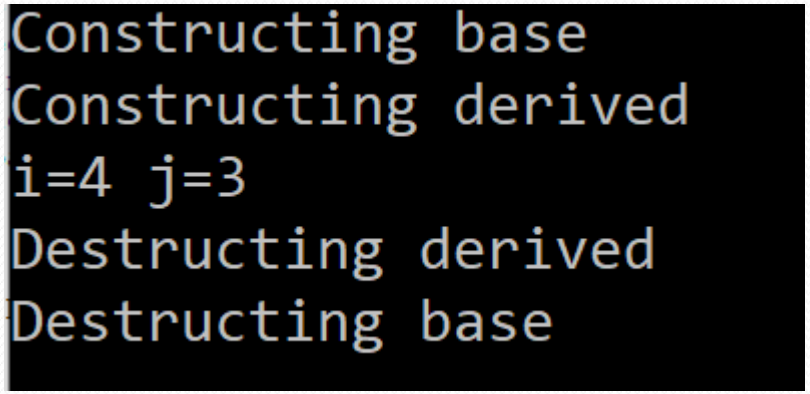
- Είδαμε ότι κατά τη δημιουργία ενός στιγμιοτύπου μιας παράγωγης κλάσης καλούνται αυτόματα και οι συναρτήσεις κατασκευής της βασικής κλάσης (ή των βασικών κλάσεων), εφόσον φυσικά υπάρχουν.
- Τι γίνεται όμως αν σε κάποιες από τις συναρτήσεις αυτές θέλουμε να περάσουμε και παραμέτρους;
- Στην περίπτωση αυτή φτιάχνουμε μια συνάρτηση κατασκευής της παράγωγης κλάσης με όλες τις παραμέτρους που θέλουμε να περάσουμε σε κλάσεις-γονείς και δηλώνουμε ρητά ποιες παραμέτρους θέλουμε να περάσουμε σε ποιες κλάσεις.

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

class base {
protected:
   int i;
public:
   base(int x) {
      i = x;
      cout << "Constructing base" << endl;
   }
   ~base() {cout << "Destructing base" << endl;}
};

class derived: public base {
protected:
   int j;
public:
   derived(int x,int y):base(y) {
      j = x;
      cout << "Constructing derived" << endl;
   }
   ~derived() {cout << "Destructing derived" << endl;}
   void show() {cout << "i=" << i << " j=" << j << endl;}
};
```

Η παράμετρος x χρησιμοποιείται από τη συνάρτηση κατασκευής της derived ενώ η παράμετρος y διαβιβάζεται στη συνάρτηση κατασκευής της βασικής κλάσης.

```
int main()
{
   derived ob(3,4);
   ob.show();
   return 0;
}
```

Αποτέλεσμα στην οθόνη



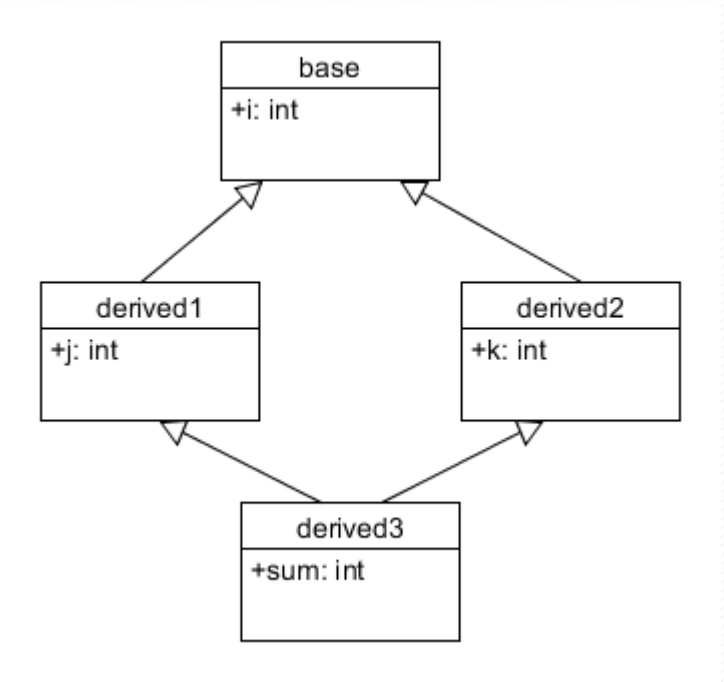
Γενική μορφή, με περισσότερες βασικές κλάσεις

```
derived-constructor(parameter list):base1(parameter list),
                                    base2(parameter list),
                                     …
                                     baseN(parameter list) {
 … κώδικας …
}
```

Ανάλογα με την εφαρμογή, η συνάρτηση κατασκευής μπορεί είτε να χρησιμοποιεί τις παραμέτρους που δέχεται είτε να τις διαβιβάζει σε βασικές κλάσεις (ή και τα δύο).

Εικονικές βασικές κλάσεις

- Είναι δυνατόν να υπάρξουν καταστάσεις όπου τα χαρακτηριστικά μιας βασικής κλάσης κληρονομούνται περισσότερες από μια φορές από μια παράγωγη κλάση.



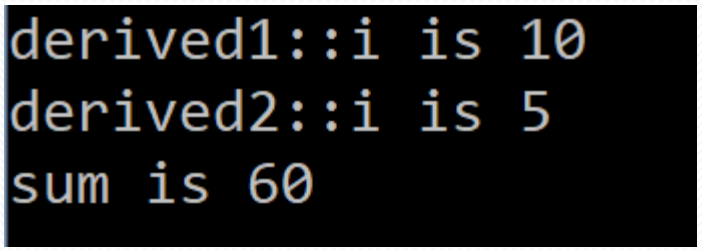
```
class base {
public:
   int i;
};

class derived1: public base {
public:
   int j;
};

class derived2: public base {
public:
   int k;
};

class derived3: public derived1, public derived2 {
public:
   int sum;
};
```

- Ένα στιγμιότυπο της κλάσης derived3 πρέπει να έχει τα χαρακτηριστικά και των δύο βασικών κλάσεων derived1 και derived2.
- Καθεμιά από αυτές έχει τα χαρακτηριστικά της βασικής κλάσης base, δηλαδή τη μεταβλητή i.
- Επομένως κάθε στιγμιότυπο της derived3 πρέπει να έχει δύο μεταβλητές i.
  - Πρόβλημα: Πώς τις ξεχωρίζουμε;
- Υπάρχουν δύο δυνατότητες
  - Χρήση του τελεστή :: για να δηλώνουμε ρητά ποια από τις μεταβλητές i εννοούμε κάθε φορά.
  - Δήλωση της κλάσης base ως εικονικής βασικής κλάσης (virtual base class).
    - Στην περίπτωση αυτή τα χαρακτηριστικά της δεν κληρονομούνται στις παράγωγες κλάσεις πάνω από μια φορά.

Χρήση του τελεστή ::

```
int main()
{
    derived3 obj;
    obj.derived1::i = 10;
    obj.derived2::i = 5;
    obj.j=20;
    obj.k=30;
    obj.sum = obj.derived1::i + obj.j +obj.k;
    cout << "derived1::i is " << obj.derived1::i << endl;
    cout << "derived2::i is " << obj.derived2::i << endl;
    cout << "sum is " << obj.sum << endl;
    return 0;
}
```

Αποτέλεσμα στην οθόνη



- Για να δηλώσουμε την κλάση base ως εικονική βασική κλάση προσθέτουμε τη λέξη virtual πριν από τον τύπο της κληρονομικότητας (public) στις δηλώσεις των παράγωγων κλάσεων derived1 και derived2.

```
class derived1: virtual public base {
public:
   int j;
};

class derived2: virtual public base {
public:
   int k;
};
```

- Κλάσεις που παράγονται από τις derived1 και derived2 κληρονομούν τα χαρακτηριστικά της base μόνο μια φορά.
  - Στιγμιότυπα των κλάσεων αυτών θα περιλαμβάνουν μόνο μια μεταβλητή i.