HY352 : ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ,
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ,
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΔΑΣΚΩΝ
Αντώνιος Σαββίδης

CSD

ΕΝΟΤΗΤΑ 4

ΣΤΟΙΧΕΙΑ ΟΝΤΟΚΕΝΤΡΙΚΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Αριθμός διαλέξεων 7, Διάλεξη 3η

CSD

Περιεχόμενα

- *Συναρτήσεις – μέλη, μυστικά*
- Κληρονομικότητα – κεφάλαιο Ι
  - Προγραμματιστικά πλεονεκτήματα
  - Το μοντέλο και τα βασικά στοιχεία
  - Κληρονομιά τύπων / δεδομένων
  - Κληρονομικότητα και κανόνες πρόσβασης
  - Κληρονομιά συναρτήσεων
  - Σειρά κλήσης constructor / destructor

CSD

Συναρτήσεις – μέλη, μυστικά (1/4)

- Θα αναλύσουμε τον τρόπο αντιμετώπισης των *συναρτήσεων – μέλη* από τους C++ compilers
- Έτσι θα γίνει καλύτερα κατανοητό τι συνεπάγεται η κλήση μίας τέτοιας συνάρτησης
  - και τι σημαίνει για την παραγωγή κώδικα πρακτικά η έννοια *μέλος – συνάρτηση*
  - η ανάλυση θα γίνει με την παρουσίαση του ισοδύναμου κώδικα σε C για τον ορισμό μίας κλάσης
    - έτσι λειτουργούσαν οι αρχικοί compiler για τη C++, οι οποίοι παρήγαγαν τελικό κώδικα σε C – θα δούμε μία απλοποιημένη μορφή
    - κάτι αντίστοιχο γίνεται και στην παραγωγή κώδικα μηχανής
  - Μπορείτε να εντοπίσετε αρκετές *σχεδιαστικές* ομοιότητες με τον κώδικα σε C που είδαμε

Συναρτήσεις – μέλη, μυστικά (2/4)

Καμία virtual συνάρτηση

C++ δηλώσεις και ορισμοί

```
class Shape {
  private:
    float x, y;
  public:
    void   Display (void);
    void   Move (const Point& atPoint)
               { x = atPoint.GetX(); y = atPoint.GetY(); Display(); }
    void   Move (float dx, float dy)
               { x += dx; y+= dy; Display(); }
    float  GetX (void) const { return x; }
    float  GetY (void) const { return y; }
    Shape (void) : x(0), y(0) {}
    Shape (float _x, float _y) : x(_x), y(_y) {}
   ~Shape(){}
};
```

Χρήση

```
{ Shape sh1; sh1.Move(-3, -5);
  Shape sh2(10, 20); float x = sh1.GetX();
  Point pt (40, 50); sh1.Move(pt); }
```


Συναρτήσεις – μέλη, μυστικά (3/4)

Παραγωγή κώδικα σε C για την κλάση

```
struct Shape { float x, y; };    // Μόνο δεδομένα περιέχονται !

void  Shape_Display_void (Shape* this);

void  Shape_Move_Point (Shape* this, Point* atPoint) {
  this->x = Point_GetX_void(atPoint);
  this->y = Point_GetY_void(atPoint);
  Shape_Display_void(this);
}
void Shape_Move_float_float (Shape* this, float dx, float dy) {
  this->x+= dx; this->y+= dy; Shape_Display_void(this);
}

float Shape_GetX_void (Shape* this) { return this->x; }
float Shape_GetY_void (Shape* this) { return this->y; }

Shape_Shape_void (Shape* this) { this->x =0, this->y = 0; }
Shape_Shape_float_float (Shape* this, float _x, float _y) {
  this->x = _x, this->y = _y;
}
Shape_Destructor(Shape* this){}
```

Η δημιουργία συναρτήσεων με μοναδικά ονόματα στην μετατροπή όλων των κλάσεων σε λίστα από συναρτήσεις βασίζεται σε έναν αλγόριθμο και η τεχνική ονομάζεται name decoration ή συνήθως *name mangling*

Συναρτήσεις – μέλη, μυστικά (4/4)

Χρήση στιγμιότυπων C++

```
{
1:  Shape sh1;
2:  sh1.Move(-3, -5);
3:  Shape sh2(10, 20);
4:  float x = sh1.GetX();
5:  Point pt(40, 50);
6:  sh1.Move(pt);
}
```

Χρήση στιγμιότυπων στη C

```
{
1: Shape sh1;
3: Shape sh2;
4: float x;
5: Point pt;

1: Shape_Shape_void(&sh1);
2: Shape_Move_float_float(&sh1,-3,-5);
3: Shape_Shape_float_float(&sh2, 10, 20);
4: x = Shape_GetX_void(&sh1);
5: Point_Point_float_float(&pt, 40, 50);
6: Shape_Move_Point(&sh1, &pt);

5: Point_Destructor(&pt);
3: Shape_Destructor(&sh2);
1: Shape_Destructor(&sh1);
}
```

δηλώσεις

κλήση constructors και members με τη σειρά

κλήση destructors

Περιεχόμενα

- Συναρτήσεις – μέλη, μυστικά
- Κληρονομικότητα – κεφάλαιο I
  - *Προγραμματιστικά πλεονεκτήματα*
  - Το μοντέλο και τα βασικά στοιχεία
  - Κληρονομιά τύπων / δεδομένων
  - Κληρονομικότητα και κανόνες πρόσβασης
  - Κληρονομιά συναρτήσεων
  - Σειρά κλήσης constructor / destructor

CSD

Προγραμματιστικά πλεονεκτήματα (1/2)

- Χρήση και εκμετάλλευση κώδικα και δεδομένων που έχουν ήδη οριστεί και αξιολογηθεί, χωρίς παραβίαση των κανόνων πρόσβασης
  - *Επαναχρησιμοποίηση - reusability*
- Υποστήριξη εφαρμογής παρόμοιων επεξεργασιών, με παρόμοιο, πάνω σε παρόμοια αντικείμενα (π.χ. display / move / copy, circle / triangle / rectangle).
  - *Πολυμορφισμός - polymorphism*

CSD

Προγραμματιστικά πλεονεκτήματα (2/2)

- Η κληρονομικότητα ως σχέση κληρονόμου – κληροδότη, δεν γεννάει και εξαρτήσεις μεταξύ αυτών των δύο;
  - Είναι αυτό το είδος της επαναχρησιμοποίησης ισχυρή εξάρτηση ή μία απλή αντικαταστάσιμη εξάρτηση;
  - Θα δούμε δύο τρόπους κληρονομικότητας και θα μελετήσουμε μειονεκτήματα και πλεονεκτήματα:
    - την απλή κληρονομικότητα όπου ο κληροδότης είναι συγκεκριμένος
    - την λεγόμενη mixin inheritance ή αλλιώς genericity όπου ο κληροδότης είναι παραμετροποιημένος
- Ο πολυμορφισμός βασίζεται στη δημιουργία αλγορίθμων πάνω σε super-classes γεγονός που μειώνει τις εξαρτήσεις
  - Αυτό δε δημιουργεί μία εγγενή εξάρτηση μεταξύ των πολυμορφικών αλγορίθμων και των super-classes; ποιος από τους δύο έρχεται πρώτος («η κότα έκανε το αυγό ή το αυγό την κότα;»)

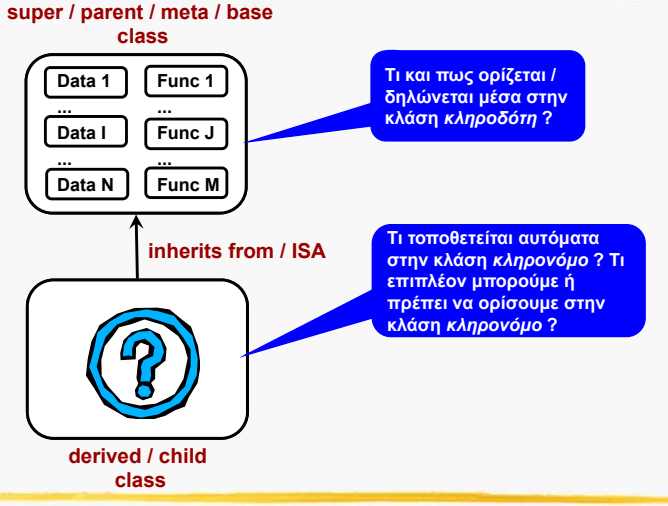
CSD

Περιεχόμενα

- Συναρτήσεις – μέλη, μυστικά
- Κληρονομικότητα – κεφάλαιο Ι
  - Προγραμματιστικά πλεονεκτήματα
  - *Το μοντέλο και τα βασικά στοιχεία*
  - Κληρονομιά τύπων / δεδομένων
  - Κληρονομιά συναρτήσεων
  - Σειρά κλήσης constructor / destructor

CSD

Το μοντέλο και τα βασικά στοιχεία (1/2)



CSD

Το μοντέλο και τα βασικά στοιχεία (2/2)

- Θα παρουσιάσουμε τους κανόνες που διέπουν την κληρονομικότητα, τόσο για δεδομένα όσο και για συναρτήσεις, δίνοντας απαντήσεις στις ακόλουθες ερωτήσεις:
  - τι δύναται να δηλωθεί ή οριστεί μέσα σε μία κληροδότη κλάση, και πως μπορεί ή επιβάλλεται να γίνει αυτό ?
  - τι κληρονομείται αυτόματα από τις κληρονόμους κλάσεις, και πως μπορούν τα όποια κληρονομημένα να χρησιμοποιούνται μέσω των αντίστοιχων κληρονόμων στιγμιότυπων ?
  - πως είναι δυνατό να επαναπροσδιορίσουμε μέσα σε μία κληρονόμο κλάση τον τρόπο χρήσης των κληρονομημένων ?

CSD

Περιεχόμενα

- Συναρτήσεις – μέλη, μυστικά
- Κληρονομικότητα – κεφάλαιο I
  - Προγραμματιστικά πλεονεκτήματα
  - Το μοντέλο και τα βασικά στοιχεία
  - *Κληρονομιά τύπων / δεδομένων*
  - Κληρονομικότητα και κανόνες πρόσβασης
  - Κληρονομιά συναρτήσεων
  - Σειρά κλήσης constructor / destructor

CSD

Κληρονομιά τύπων / δεδομένων (1/5)

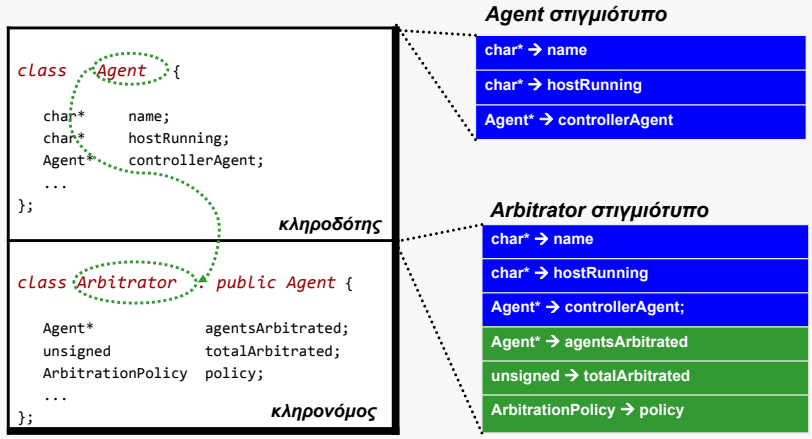
- Η κληρονόμος κλάση περιέχει τύπους και δεδομένα τα οποία ορίζονται ως:
  - ***τύποι και δεδομένα(κληρονόμου) = τύποι και δεδομένα(κληροδότη) ∪ τύποι και δεδομένα (σώμα κληρονόμου κλάσης)***
- Είναι απολύτως νόμιμος ο ορισμός μεταβλητών και τύπων στην κληρονόμο κλάση με αναγνωριστικά ονόματα τα οποία είναι ίδια με μεταβλητές και τύπους της κληροδότη κλάσης
- Στο χώρο ορισμού της κληροδότη κλάσης (δηλ. σώμα ορισμού και υλοποίηση συναρτήσεων - μελών), κληρονομημένες μεταβλητές και τύποι μπορούν να χρησιμοποιηθούν εάν και μόνο εάν είχαν οριστεί με τους χαρακτηρισμούς πρόσβασης:
  - *public ή protected*

CSD

Κληρονομιά τύπων / δεδομένων (2/5)

- *Σημείωση*
  - Υπάρχει σαφής διαχωρισμός μεταξύ:
    - του σώματος ορισμού μίας κλάσης, δηλ. το τι μπορεί να δηλωθεί ή οριστεί μέσα στο χώρο **class {...},**
      - *class definition body*
    - ...και της εικόνας της κλάσης κατά την εκτέλεση του προγράμματος, δηλ. τη δομή μνήμης στιγμιότυπου και την οργάνωση των περιεχομένων του
      - *class run-time memory model*

Κληρονομιά τύπων / δεδομένων (3/5)



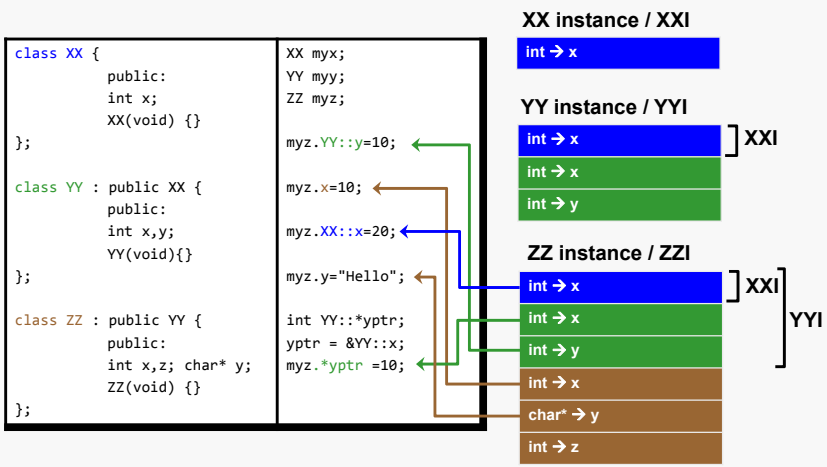
Κληρονομιά τύπων / δεδομένων (4/5)

```
class Circle : public Shape {
  private:
    float radius;
  public:
    Circle (float _x, float _y, float _radius) :
      Shape(_x, _y),      // Μεταβίβαση παραμέτρων στον κληροδότη constructor
      radius(_radius) {}
};

// Μεγάλη ευελιξία στον ορισμό τύπων

class A            { public: typedef bool Flag; };
class B : public A { public: typedef unsigned int Flag; };
class C : public A { public: struct  Flag { bool val; }; };
class D : public B { public: typedef B::Flag Flag; };
class E : public D { public: class   Flag { public: D::Flag flag; }; };
```


Κληρονομιά τύπων / δεδομένων (5/5)



Περιεχόμενα

- Συναρτήσεις – μέλη, μυστικά
- Κληρονομικότητα – κεφάλαιο I
  - Προγραμματιστικά πλεονεκτήματα
  - Το μοντέλο και τα βασικά στοιχεία
  - Κληρονομιά τύπων / δεδομένων
  - *Κληρονομικότητα και κανόνες πρόσβασης*
  - Κληρονομιά συναρτήσεων
  - Σειρά κλήσης constructor / destructor

CSD

Κληρονομικότητα και κανόνες πρόσβασης (1/2)

- Τα μέλη μίας κλάσης μπορούν να έχουν ένα από τους παρακάτω χαρακτηρισμούς πρόσβασης:
  - **private** ή **public**
  - ή **protected,** ο οποίος χρησιμοποιείται αποκλειστικά στην κληρονομικότητα.
    - Ένα protected μέλος είναι ουσιαστικά private εκτός της κλάσης ***X*** στην οποία ορίζεται, αλλά μπορεί να χρησιμοποιηθεί σε μία κλάση που είναι άμεσος κληρονόμος της ***X***.
- Η κληρονομικότητα πρέπει να χαρακτηρίζεται με έναν από τους τρεις διαφορετικούς χαρακτηρισμούς (δηλ. public, private, protected).
  - Αυτού του είδους ο χαρακτηρισμός ορίζει *τον τρόπο με τον οποίο οι αυθεντικοί χαρακτηρισμοί πρόσβασης των κληρονομημένων μελών τροποποιούνται μέσα στο χώρο της κληρονόμου κλάσης*

CSD

Κληρονομικότητα και κανόνες πρόσβασης (2/2)

χαρακτηρισμός κληρονομικότητας	χαρακτηρισμός πρόσβασης μέλους κληροδότη
	δυνατότητα χρήσης στην κληρονόμο κλάση
	πρόσβαση κληρονομημένου μέλους

	public	private	protected
public	ΝΑΙ	ΟΧΙ	ΝΑΙ
	public	private	protected
private	ΝΑΙ	ΟΧΙ	ΝΑΙ
	private	private	private
protected	ΝΑΙ	ΟΧΙ	ΝΑΙ
	protected	private	protected

Το αποτέλεσμα της private κληρονομικότητας είναι ότι όλα τα κληρονομημένα μέλη γίνονται private

Τα private μέλη του κληροδότη είναι απρόσιτα στην κληρονόμο

Τα protected μέλη του κληροδότη είναι πάντα προσιτά στην κληρονόμο

CSD

Περιεχόμενα

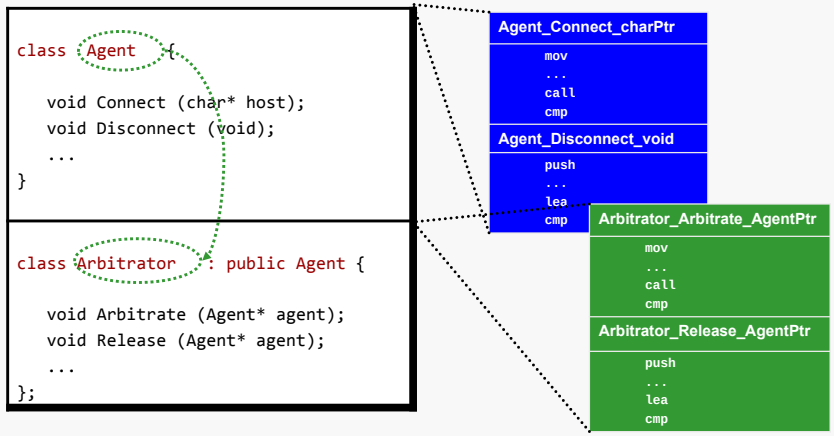
- Συναρτήσεις – μέλη, μυστικά
- Κληρονομικότητα – κεφάλαιο Ι
  - Προγραμματιστικά πλεονεκτήματα
  - Το μοντέλο και τα βασικά στοιχεία
  - Κληρονομιά τύπων / δεδομένων
  - Κληρονομικότητα και κανόνες πρόσβασης
  - *Κληρονομιά συναρτήσεων*
  - Σειρά κλήσης constructor / destructor

CSD

Κληρονομιά συναρτήσεων (1/11)

- Παρόμοιοι κανόνες με την κληρονομιά δεδομένων και τύπων ισχύουν.
- Η κληρονόμος κλάση περιέχει συναρτήσεις οι οποίες ορίζονται ως:
  - *συναρτήσεις(κληρονόμου) = συναρτήσεις(κληροδότη) ∪ συναρτήσεις(χώρος κληρονόμου κλάσης)*
  - Οι συναρτήσεις της κληρονόμου κλάσης ανήκουν σε διαφορετική εμβέλεια από αυτές της κληροδότη
    - Μπορούμε να έχουμε συναρτήσεις που να έχουν το ίδιο όνομα, και διαφορετικές / ίδιες υπογραφές (signatures)
  - Μέσα στην κληρονόμο κλάση, συναρτήσεις της κληροδότη μπορούν να χρησιμοποιηθούν μόνο μόνο εάν είχαν οριστεί με τους χαρακτηρισμούς πρόσβασης:
    - ***public ή protected***

Κληρονομιά συναρτήσεων (2/11)



...οι συναρτήσεις δεν αντιγράφονται, επαναχρησιμοποιούνται !

Κληρονομιά συναρτήσεων (3/11)

- Η κληρονομιά των συναρτήσεων, πέρα από την απλή επαναχρησιμοποίηση, αποκτάει μεγαλύτερη αξία με την δυνατότητα επαναπροσδιορισμού από την κληρονόμο κλάση των:
  - Χαρακτηριστικών πρόσβασης
  - Υπογραφής
  - Σημασιολογίας – λειτουργίας (υλοποίησης)
- *Προσοχή, δεν αναφερόμαστε σε πολυμορφικές συναρτήσεις με δυναμική αντιστοίχιση ακόμη αλλά εν γένει σε οποιουδήποτε είδους συνάρτηση*

Κληρονομιά συναρτήσεων (4/11)

αλλαγή χαρακτηρισμών πρόσβασης

```
class X {
    public: void F(void){}
    X(void){}
};
class Y : public X {
    private: void F(void){}
    public: Y(void){}
};
class Z : protected X {
     private: void F(void){}
     public: Z(void){}
};

Y y;
y.F();           // Error, το Y::F είναι private
((X*) &y)->F();  // Ok, αφού το X::F είναι public
Z z;             // Να το δοκιμάσουμε με το Z ?
((X*) &z)→F();  // Error, η protected κληρονομικότητα εμποδίζει το casting
void* p = &z;    // Εδώ όμως δεν μας εμποδίζει κανείς
((X*)p)->F();    // ...και ιδού παρακάμψαμε τον κανόνα !
```


Κληρονομιά συναρτήσεων (5/11)

- Η αλλαγή *υπογραφής μίας συνάρτησης* μπορεί να συνίσταται σε:
  - Αλλαγή τύπου επιστρεφόμενου αποτελέσματος
  - Αλλαγή τυπικών ορισμάτων - παραμέτρων
  - Αλλαγή τρόπου σύνδεσης (linkage)

Κληρονομιά συναρτήσεων (6/11)

Αλλαγή υπογραφής (1/2)

```
class X {
    public: void F(void){};
    X(void){}
};
class Y : public X {
    public:
    int   F(void){...}   // Αλλαγή τύπου αποτελέσματος
    char* F(void){...}   // Error, παράνομη υπερφόρτωση της int F(void)
    bool  F(int,int);    // Ok, αλλαγή τύπου αποτελέσματος και παραμέτρων
    int   F(int a=100);  // Ok, όμοια αλλαγή με προηγούμενη
    Y(void);
};

Y y;
y.F();           // Error, ποια F ? int F(default 100) ή int F(void) ?
y.X::F();        // Ok, χρήση απευθείας της X::F
```


Κληρονομιά συναρτήσεων (7/11)

Αλλαγή υπογραφής (2/2)

```
class X {
    public:
    static void F(X*);
    void G(void);
}

class Y : public X {
    public:
    void F(void) { X::F((X*) this); }
    static void G(Y* y) { ((X*)y)->G(); }
};

class Z : public Y {
    public:
    static void F(Z* z) { ((Y*)z)->F(); }
    void G(void) { Y::G((Y*) this); }
};
```

static → auto

auto → static

static ←auto

auto ← static

Κληρονομιά συναρτήσεων (8/11)

- *Αλλαγή σημασιολογίας – υλοποίησης (ενδεικτική ταξινόμηση)*
  - *Διεύρυνση ή επαύξηση* - augmentation
    - Επιπλέον λειτουργικότητα προστίθεται «πάνω» στην αυθεντική κληρονομούμενη συνάρτηση
  - *Απόρριψη ή αντικατάσταση* - overriding
    - Η κληρονομούμενη λειτουργικότητα της συνάρτησης απορρίπτεται, ενώ παρέχεται από την κληρονόμο κλάση νέα εξ αρχής υλοποίηση
  - *Περιορισμός* - restriction
    - Το πεδίο ορισμού της συνάρτησης, δηλ. το σύνολο των αποδεκτών πραγματικών παραμέτρων – actual arguments – κατά την κλήση της συνάρτησης, περιορίζεται, γεγονός που καθιστά τη συνάρτηση εφαρμόσιμη σε ένα μικρότερο πεδίο τιμών

Κληρονομιά συναρτήσεων (9/11)

παράδειγμα διεύρυνσης

```
class Window {
   public:
   virtual void OnMouseEnter (int x, int y) { SetCursor(FOCUS_CURSOR); }
};
class Button : public Window {
   public:
   void DisplayFocusBorder (void);
   virtual void OnMouseEnter (int x, int y) {
        DisplayFocusBorder();        // Επαύξηση λειτουργικότητας
        Window::OnMouseEnter(x,y);   // Κλήση κληρονομούμενης συνάρτησης
   }
};
class AuditoryButton : public Button {
   public:
   void PlayAudioOnFocus (void);
   virtual void OnMouseEnter (int x, int y) {
        PlayAudioOnFocus();          // Επαύξηση λειτουργικότητας
        Button::OnMouseEnter(x,y);   // Κλήση κληρονομούμενης συνάρτησης
   }
};
```

Κληρονομιά συναρτήσεων (10/11)

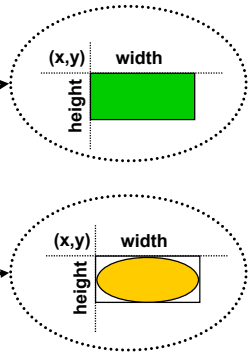
παράδειγμα απόρριψης

```
class Button {
   unsigned int width, height;
   public:
   virtual void Display(...);
};

class OvalButton : public Button {
   public:
   virtual void Display(...);
}

void Button::Display (...) {
   Κώδικας για τη ζωγραφική ορθογώνιου
      αντικειμένου Button
}

void OvalButton::Display (...) {
   Αντίστοιχος κώδικας για ζωγραφική ενός  οβάλ Button,
      στην οποία η Button::Display() δεν χρησιμοποιείται
}
```



Κληρονομιά συναρτήσεων (11/11)

παράδειγμα περιορισμού

```
class VideoConference {
   public:
   virtual bool Connect (char* host, char* userId, unsigned timeOut);
};

class GroupVideoConference : public VideoConference {
   private:
   char*  groupId;
   static bool InSameGroup (char* groupId, char* host, char* userId);

   public:
   bool Connect (char* host, char* userId, unsigned timeOut);
   GroupVideoConference (char* groupId);
};

bool GroupVideoConference::Connect (...) {
     if (InSameGroup(groupId, host, userId))
         return VideoConference::Connect(host, userId, timeOut);
     return false;
}
```

αρχική συνάρτηση κληροδότου κλάσης

περιορισμός αρχικής συνάρτησης

λογική περιορισμού πεδίου τιμών

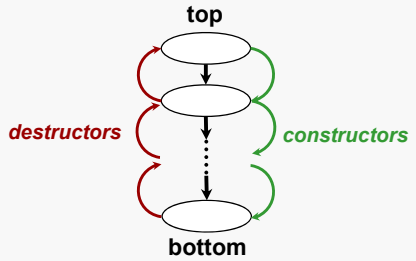
κλήσης αυθεντικής συνάρτησης

Περιεχόμενα

- Συναρτήσεις – μέλη, μυστικά
- Κληρονομικότητα – κεφάλαιο Ι
  - Προγραμματιστικά πλεονεκτήματα
  - Το μοντέλο και τα βασικά στοιχεία
  - Κληρονομιά δεδομένων
  - Κληρονομικότητα και κανόνες πρόσβασης
  - Κληρονομιά συναρτήσεων
  - *Σειρά κλήσης constructor / destructor*

Σειρά κλήσης constructor / destructor (1/2)

- Οι constructors στην ιεραρχία κληρονομικότητας καλούνται από πάνω προς τα κάτω - *downwards*
- Οι destructors στην ιεραρχία κληρονομικότητας καλούνται από κάτω προς τα πάνω - *upwards*



CSD

Σειρά κλήσης constructor / destructor (2/2)

```
class X {
public: X(void) { printf("X()\n"); }
       virtual ~X() { printf("~X()\n"); }
};

class Y : public X {
public: Y(void) { printf("Y()\n"); }
       ~Y() { printf("~Y()\n"); }
};

class Z : public Y {
public: Z(void) { printf("Z()\n"); }
       ~Z() { printf("~Z()\n"); }
};

int main (int, char**) { Z z; return 0; }
```

Εκτυπώνεται κατά την εκτέλεση:

```
X();
Y();
Z();
~Z();
~Y();
~X();
```

παράδειγμα