HY352 : ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ,
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ,
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΔΑΣΚΩΝ
Αντώνιος Σαββίδης

CSD

ΕΝΟΤΗΤΑ 4

ΣΤΟΙΧΕΙΑ ΟΝΤΟΚΕΝΤΡΙΚΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Αριθμός διαλέξεων 7, Διάλεξη 1η

CSD

Περιεχόμενα

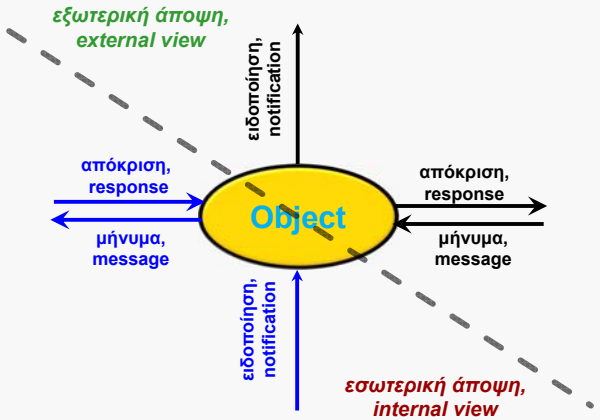
- *Ή έννοια του αντικειμένου (θεωρία)*
- Η γένεση των μεθόδων (πράξη)

CSD

Η έννοια του αντικειμένου (1/5)

- Τα αντικείμενα είναι λειτουργικές οντότητες οι οποίες δέχονται έναν πεπερασμένο αριθμό καλά ορισμένων μηνυμάτων, τα οποία περιέχουν συγκεκριμένες παραμέτρους / δεδομένα, και επιστρέφουν κατάλληλες αποκρίσεις.
  - Οι αποκρίσεις αυτές μπορεί να είναι και ασύγχρονες, δηλ. να αποστέλλονται αφού η αποστολή των μηνυμάτων, ως συνάρτηση έχει «επιστρέψει».
  - Τόσο οι αποστολείς των μηνυμάτων όσο και οι παραλήπτες είναι επίσης αντικείμενα
- Ένα σύστημα είναι ένα σύμπλεγμα από αλληλεπιδρώντα αντικείμενα
- Εάν θυμηθούμε το component-based architecture θα δούμε ότι μπορούμε αμέσως να θεωρήσουμε πως component = object
- ***Άρα με μία οντοκεντρική προσέγγιση κάθε αρχιτεκτονικό τμήμα είναι και ένα αντικείμενο***

Η έννοια του αντικειμένου (2/5)



Η έννοια του αντικειμένου (3/5)

- Η έννοια αποστολής μηνύματος → αντικείμενο
  - Τα μηνύματα αντιστοιχούν σε λειτουργίες (συναρτήσεις) τις οποίες μπορεί να επιτελέσει ένα αντικείμενο.
    - Ένα μήνυμα ενσωματώνει πληροφορία σχετικά με την ταυτότητα της συνάρτησης, και τις παραμέτρους κλήσεις,
    - …ενώ η απόκριση ενσωματώνει τα επιστρεφόμενα από την κλήση αποτελέσματα.

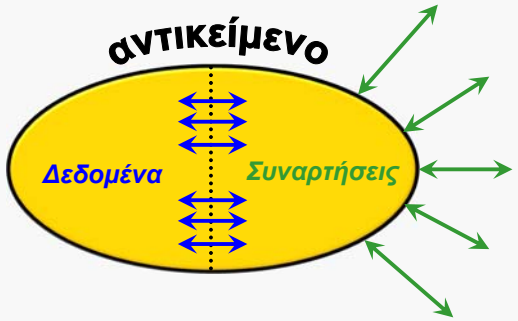
```
Object Array myArray;
GetMaximum(1...10)  →  myArray  →  MaxElement;
   μήνυμα            αντικείμενο     σύγχρονη απόκριση
```

```
Array myArray;
Element maxElement = myArray.GetMaximum(1, 10);
   σύγχρονη απόκριση   αντικείμενο      μήνυμα
```


Η έννοια του αντικειμένου (4/5)

- Τα αντικείμενα ενσωματώνουν δεδομένα (δηλ. τοπικές μεταβλητές) και συναρτήσεις.
- Τα δεδομένα σπανίως είναι διαθέσιμα σε άλλα αντικείμενα
  - *ιδιωτική χρήση – private use*
- Από τις συναρτήσεις τις οποίες περιέχει ένα αντικείμενο, μόνο ένα υποσύνολο γίνεται διαθέσιμο σε άλλα αντικείμενα
  - *δημόσια χρήση – public use*

Η έννοια του αντικειμένου (5/5)



Περιεχόμενα

- Ή έννοια του αντικειμένου (θεωρία)
- *Η γένεση των μεθόδων (πράξη)*
  - *Μία απλουστευμένη επιχείρηση εξομοίωσης των θεμελιωδών μεθόδων οντοκεντρικού προγραμματισμού στη γλώσσα C*
    - *...kids don't do this at home*

Η γένεση των μεθόδων (1/20)

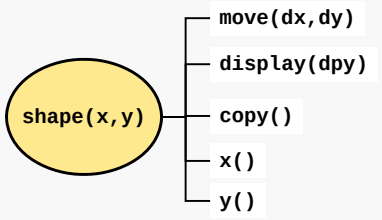
- Ο οντοκεντρικός προγραμματισμός γεννήθηκε στα όρια εφαρμογής του δομημένου προγραμματισμού και της προσπάθειας επίτευξης βέλτιστης κατάτμησης

→ Θα ερευνήσουμε τον τρόπο με τον οποίο αυτή η νέα στρατηγική οδήγησε στον εντοπισμό χαρακτηριστικών ανεπαρκειών των διαδικαστικών γλωσσών (όπως η C), κυρίως λόγω περιορισμών σχετικά με:

- → την δυνατότητα να προγραμματίσουμε το σχεδιαστικό μοντέλο απ' ευθείας στον κώδικα
- → την υποστήριξη υλοποίησης της αντιστοίχησης του μοντέλου σε κώδικα με ένα καθαρό, και εύκολο στη διαχείριση προγραμματιστικό τρόπο.

Η γένεση των μεθόδων (2/20)

- *Ενθυλάκωση και απόκρυψη πληροφορίας (1/3)*
  - Encapsulation and information hiding



- Τα αντικείμενα εμπεριέχουν ταυτόχρονα δεδομένα και συναρτήσεις με τρόπους που δεν είναι ορατοί στο εξώτερο προγραμματιστικό περιβάλλον.
- Οι εσωτερικές δομές δεδομένων καθώς και η υλοποίηση των συναρτήσεων είναι πλήρως αδιαφανείς.

→ Αυτές οι ιδιότητες ορίζουν την έννοια *κάψουλας κώδικα*

- *code capsule*

Η γένεση των μεθόδων (3/20)

- *Ενθυλάκωση και απόκρυψη πληροφορίας (2/3)*

...τεχνική υλοποίησης στη C

```
// shape.h, δημόσια διαθέσιμοι τύποι και συναρτήσεις.

typedef void* shape_t;  // Αδιαφανής τύπος αντικειμένου shape.

extern shape_t  shape_create (float x, float y);
extern void     shape_move (shape_t shape, float dx, float dy);
extern shape_t  shape_copy (shape_t shape);
extern void     shape_display (shape_t shape, display_t display);
extern float    shape_x (shape_t shape);
extern float    shape_y (shape_t shape);
extern void     shape_destroy (shape_t shape);
```

Best practices: (i) all functions are prefixed with the class name; (ii) all functions accept the object as the first argument; and (iii) constructors / destructors are clearly supported.

Η γένεση των μεθόδων (3/20)

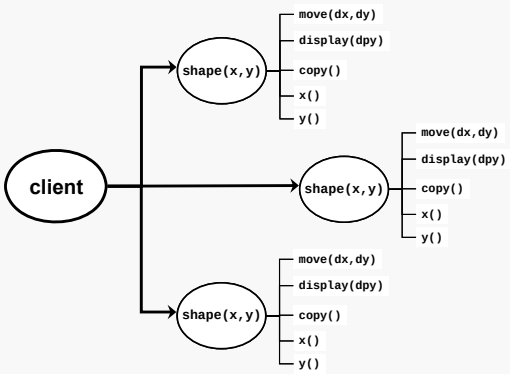
- *Ενθυλάκωση και απόκρυψη πληροφορίας (3/3)*

Αντίστοιχες περιπτώσεις

```
Σχετικά καλής ποιότητας API
FILE*     fopen (const char* filename, const char* mode );
int       fclose (FILE* stream);
int       fflush (FILE* stream );
size_t    fwrite (const void* buffer, size_t size, size_t count, FILE* stream );
int       fprintf (FILE* stream, const char *format, ... );

Κακής ποιότητας API
int       socket(int domain, int type, int protocol);
int       close(int f);
int       connect (int sockfd, const sockaddr* serv_addr, socklen_t addrlen);
int       bind (int sockfd, const sockaddr* addr, socklen_t addrlen);
int       accept (int sockfd, sockaddr* addr, socklen_t* addrlen);
```


Η γένεση των μεθόδων (4/20)

- *Δημιουργία πολλαπλών ανεξάρτητων στιγμιότυπων (1/2)*
  - Instance creation



- Πρέπει να υποστηρίζεται η δημιουργία πολλαπλών στιγμιότυπων για τον τύπου του αντικειμένου, ως ανεξάρτητων λειτουργικών οντοτήτων.
- Κάθε στιγμιότυπο διατηρεί τη δική του εσωτερική κατάσταση, χωρίς να επηρεάζεται από τα υπόλοιπα.

Η γένεση των μεθόδων (5/20)

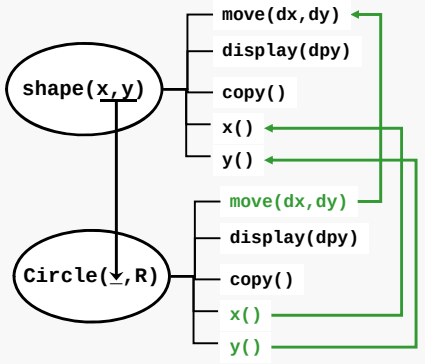
- *Δημιουργία πολλαπλών ανεξάρτητων στιγμιότυπων (2/2)*

...τεχνική υλοποίησης στη C

```
// shape_private.h, ο εσωτερικός κρυφός τύπος.
struct shape_private_t { float x, y; };

// shape.c, η εσωτερική κρυφή υλοποίηση.
shape_t shape_create (float x, float y) {
   shape_private_t* p;
   p = (shape_private_t)* malloc(sizeof(shape_private_t));
   p->x  = x;
   p->y  = y;
   return (shape_t) p;
}

void shape_move (shape_t shape, float dx, float dy) {
   shape_private_t* p = (shape_private_t*) shape;
   p->x += dx; p->y += dy;
}
```


Η γένεση των μεθόδων (6/20)

- *Κληρονομικότητα και επαναχρησιμοποίηση (1/3)*
  - Inheritance and re-use



- Η υλοποίηση συγγενών τύπων αντικειμένων, τα οποία συνιστούν εννοιολογικές εξειδικεύσεις του αυθεντικού τύπου, θα πρέπει να επιτρέπεται με τη μέγιστη δυνατή επαναχρησιμοποίηση κώδικα.
- Εισάγεται η έννοια της κληρονομικότητας,δηλ. σχέσεων isa, τόσο για τις συναρτήσεις όσο και για τα δεδομένα.

CSD

Η γένεση των μεθόδων (7/20)

- *Κληρονομικότητα και επαναχρησιμοποίηση (2/3)*

...τεχνική υλοποίησης στη C

```
// circle.h, δημόσια διαθέσιμοι τύποι και συναρτήσεις.
typedef void* circle_t; // Αδιαφανής τύπος αντικειμένου circle.
extern circle_t circle_create (float cx, float cy, float radius);
extern void     circle_display (circle_t circle, display_t dpy);
extern circle_t circle_copy (circle_t circle);
extern float    circle_radius (circle_t circle);
extern void     circle_destroy (circle_t circle);
extern shape_t  circle_to_shape (circle_t);

// Επαναχρησιμοποίηση συναρτήσεων του shape τύπου αντικειμένου.
#define circle_move(circle,dx,dy) \
       shape_move(circle_to_shape(circle), dx, dy);
#define circle_x(circle) \
       shape_x(circle_to_shape(circle))
#define circle_y(circle) \
       shape_y(circle_to_shape(circle))
```


CSD

Η γένεση των μεθόδων (8/20)

- *Κληρονομικότητα και επαναχρησιμοποίηση (3/3)*

...τεχνική υλοποίησης στη C

```
// circle_private.h, ο εσωτερικός κρυφός τύπος.
struct circle_private_t { shape_t shape; float radius; };

// circle.c, η εσωτερική κρυφή υλοποίηση.
circle_t circle_create (float x, float y, float radius) {
   circle_private_t* p;
   p = (circle_private_t)* malloc(sizeof(circle_private_t));
   p->shape        = shape_create(x,y);
   p->radius       = radius;
   return (circle_t) p;
}
// Μετατροπή σε στιγμιότυπο του κληροδότη τύπου (super-type).
shape_t circle_to_shape (circle_t circle) {
   return ((circle_private_t*) circle)->shape;
}
```


CSD

Η γένεση των μεθόδων (9/20)

- *Υπερφόρτωση συναρτήσεων (1/3)*
  - *Function overloading*
  - Τα όνομα μίας συνάρτησης να εμφανίζεται με εναλλακτικούς ορισμούς (definitions = υλοποιήσεις), για διαφορετικές αντίστοιχες δηλώσεις (declarations, ή αλλιώς υπογραφές – signatures).
    - F (X x, Y y, Z z)
    - F(X x, W w)
    - F(A a, Bb)
  - Ο προγραμματιστής μπορεί και καλεί μία μόνο συνάρτηση με διαφορετικούς τύπους πραγματικών ορισμάτων (actual arguments).

CSD

Η γένεση των μεθόδων (10/20)

- *Υπερφόρτωση συναρτήσεων (2/3)*

...τεχνική υλοποίησης στη C

int	add(int, int)	int	add(float, float)
float	add(int, int)	float	add(float, float)
int	add(float, int)	int	add(int, float)
float	add(float, int)	float	add(int, float)

```
// add.h, δήλωση «υπερφορτωμένης συνάρτησης add.
extern void add (char* format,...);

// Παράδειγμα χρήσης.
int i = 20, j = 30;
float x = 3.14;
int result; // Λαμβάνει το αποτέλεσμα.
add("ifii", i, x, j, (void*) &result);
```

Δίνεται ο τύπος του κάθε ορίσματος με τους χαρακτήρες **i f u d**, ακολουθούν τα αντίστοιχα ορίσματα, με τελευταίο το αποτέλεσμα ως τύπο pointer

Η γένεση των μεθόδων (11/20)

- *Υπερφόρτωση συναρτήσεων (3/3)*

...τεχνική υλοποίησης στη C

```
// add.c
void add (char* format, ...) {

   va_list args;
   va_start(args, format);
   double sum = 0;

   // Αθροίζουμε τα ορίσματα.
   //
   while (*format && format[1]) {
      if (*format=='i')
         sum += va_arg(args, int);
      else
      if (format=='f')
          sum += va_arg(args, float);
      ++format;
   }
```

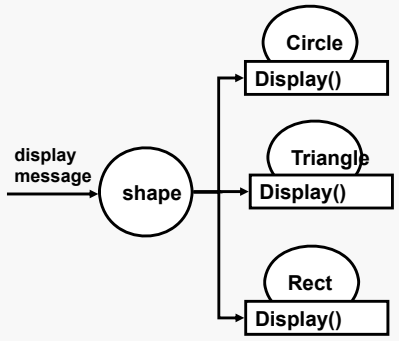
```
   // Εκχωρούμε το άθροισμα στη
   // μεταβλητή του αποτελέσματος.
   //
   ++format;
   void* result = va_arg(args, void*);
   if (*format=='i')
      *((int*) result) = (int) sum;
   else
   if (*format==f')
      *((float*) result) = (float) sum;

   va_end(args);
}

*** Έχουν αφαιρεθεί, για λόγους
ευκολίας παρουσίασης, όλοι οι έλεγχοι
λαθών.
```


Η γένεση των μεθόδων (12/20)

- *Υπερφόρτωση τελεστών*
  - Operator overloading
    - Π.χ., A + B, X * Y, με A, B, X, Y να είναι τύποι αντικειμένων διαφορετικοί από τους εγγενείς τύπους της γλώσσας προγραμματισμού (για τους οποίους η σημασιολογία των τελεστών + και * είναι αμετάβλητη).
  - Δεν υποστηρίζεται στη C.
    - Η σημασιολογία των τελεστών στη C δεν είναι επεκτάσιμη σε διαφορετικούς από τους εγγενείς τύπους.
    - Μόνο εξομοίωση με συναρτήσεις μπορεί να εφαρμοστεί.

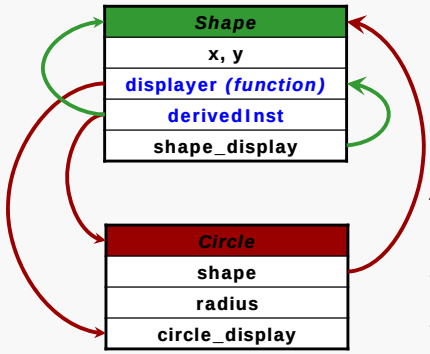
Η γένεση των μεθόδων (13/20)

- *Πολυμορφισμός και δυναμική αντιστοίχιση (1/5)*
  - Polymorphism and late binding



- Πολυμορφισμός υποστηρίζεται όταν μία συνάρτηση επιτελεί την ίδια λειτουργία σε διαφορετικούς τύπους αντικειμένων μέσω ενός κοινού API το οποίο μοιράζονται οι διαφορετικοί τύποι.
- Μόνο κατά την εκτέλεση, ανάλογα με το συγκεκριμένο τύπο κάθε στιγμιότυπου, η κατάλληλη συνάρτηση επιλέγεται και καλείται. Αυτό ονομάζεται δυναμική αντιστοίχιση συνάρτησης.

Η γένεση των μεθόδων (14/20)



Όταν δημιουργείται ένα shape instance, τίθεται η default display συνάρτηση, ενώ ως δείκτη στο στιγμιότυπο κληρονόμου κλάσης θέτουμε το ίδιο το shape instance

Όταν δημιουργείται ένα circle instance, φτιάχνει ένα shape instance, θέτει ως display συνάρτηση την δική του, και ως στιγμιότυπο κληρονόμου κλάσης θέτει τον εαυτό του.

Η γένεση των μεθόδων (15/20)

- *Πολυμορφισμός και δυναμική αντιστοίχιση (2/5)*

...τεχνική υλοποίησης στη C

```
// object.h, για όλους τους τύπους αντικειμένων.
typedef void* object_t;

// shape.h, επεκτάσεις
#include "object.h"
extern object_t shape_derivedinstance (shape_t shape);

// shape_private.h, τροποποιήσεις.
typedef void (*displayfunc_t)(shape_t shape, display_t display);
struct shape_private_t {
    float          x, y;
    displayfunc_t  displayer;   // Display συνάρτηση κληρονόμου τύπου.
    object_t       derivedInst; // Στιγμιότυπο κληρονόμου τύπου.
};
```


Η γένεση των μεθόδων (16/20)

- *Πολυμορφισμός και δυναμική αντιστοίχιση (3/5)*

...τεχνική υλοποίησης στη C

```
// shape.c, τροποποιήσεις.
static void shape_display_private (shape_t shape, display_t display) {
   // Default υλοποίηση display συνάρτησης για τον βασικό τύπο shape.
}
shape_t shape_create (float x, float y) {
   shape_private_t* p;
   p = (shape_private_t)* malloc(sizeof(shape_private_t));
   p->x            = x;
   p->y            = y;
   p->displayer    = shape_display_private;
   p->derivedInst  = p;
   return (shape_t) p;
}
object_t shape_derivedinstance (shape_t shape) {
   return ((shape_private_t*) shape)->derivedInst;
}
```


Η γένεση των μεθόδων (17/20)

- *Πολυμορφισμός και δυναμική αντιστοίχιση (4/5)*

...τεχνική υλοποίησης στη C

```
// shape.c, τροποποιήσεις (συνέχεια).
void shape_display (shape_t shape, display_t display) {
   (*((shape_private_t*) shape)->displayer)(shape, display);
}
// circle.c, τροποποιήσεις - #include shape_private.h is required!
static void circle_display_private (shape_t shape, display_t display) {
 // Υλοποίηση display συνάρτησης για τον τύπο circle.
}
circle_t circle_create (float x, float y, float radius) {
   circle_private_t* p;
   p = (circle_private_t*) malloc(sizeof(circle_private_t));
   p->shape                = shape_create(x,y);
   p->radius               = radius;
   p->shape->derivedInst   = (object_t) p;
   p->shape->displayer     = circle_display_private;
   return (circle_t) p;
}
```


Η γένεση των μεθόδων (18/20)

- *Πολυμορφισμός και δυναμική αντιστοίχιση (5/5)*

```
shape_t shapes[N]; unsigned total = 0;
void displayer (display_t d) {
    unsigned i;
    for (i=0; i<total; ++i) shape_display(shapes[i], d); // Πολυμορφισμός
}
void add (shape sh) { assert(total < N); shapes[total++] = sh; }
…
circle_t    circle   = circle_create(10,10,5);
triangle_t  triangle = triangle_create(0,0,10,10,20,20);
add(triangle_to_shape(triangle)); up-casting
add(circle_to_shape(circle));

1. shape_display (shapes[i], display):
(*((shape_private_t*) shape)->displayer)(shape, display);
2. circle_display_private(shape, display):
Εσωτερικά έχουμε (circle_t) shape_derivedinstance(shape) down-casting
```

Η γένεση των μεθόδων (19/20)

- **Περίληψη**
  - *Ενθυλάκωση και απόκρυψη πληροφορίας*
    - Encapsulation and information hiding
  - *Δημιουργία πολλαπλών ανεξάρτητων στιγμιότυπων*
    - Instance creation
  - *Κληρονομικότητα και επαναχρησιμοποίηση*
    - Inheritance and re-use
  - *Υπερφόρτωση συναρτήσεων και τελεστών*
    - Function and operator overloading
  - *Πολυμορφισμός και δυναμική αντιστοίχιση*
    - Polymorphism and late binding

Η γένεση των μεθόδων (20/20)

- *Είναι εφικτή η διαδικαστική υλοποίηση όλων των στοιχείων OOP*;
  - Χωρίς εγγενή υποστήριξη από τη γλώσσα απαιτούνται προηγμένες τεχνικές, υιοθετώντας σταδιακά όλο και πιο πολύπλοκα σχεδιαστικά «καλούπια» κώδικα
  - Τοποθετείται ψηλά ο πήχης για τους προγραμματιστές, αφού θα πρέπει να μάθουν να διαχειρίζονται τέτοιου είδους πρότυπα υλοποίησης ώστε να υποστηριχθούν οι δομές OOP
  - → Σε αυτό το σημείο υπεισέρχονται οι OOP γλώσσες, παρέχοντας όλο τον απαραίτητο εγγενή εξοπλισμό,
    - →και ταυτόχρονα μεταφέροντας την ευθύνη διαχείρισης των πολύπλοκων σχημάτων κώδικα στους κατασκευαστές των compilers