

# Προγραμματισμός Ηλεκτρονικών Υπολογιστών

## Φιλοσοφίες γλωσσών προγραμματισμού

Διδάσκων: Στάθης Ζάχος

Επιμέλεια Διαφανειών: Μάκης Αρσένης  
CoReLab

ΣΗΜΜΥ - Ε.Μ.Π.

Ιανουάριος 2017

- 1 Φιλοσοφία Γλωσσών Προγραμματισμού
- 2 Συναρτησιακός Προγραμματισμός (Functional Programming)
- 3 Λογικός Προγραμματισμός (Prolog)
- 4 Object-Oriented Programming

# Προγραμματιστικά Μοντέλα

Προγραμματιστικό Μοντέλο	Γλώσσες
Προστακτικός Προγραμματισμός (Imperative Programming)	FORTRAN, Algol, COBOL Pascal, C, Ada, Pascal
Συναρτησιακός Προγραμματισμός (Functional Programming)	LISP, ML, Scheme, Haskell
Λογικός Προγραμματισμός (Logic Programming)	Prolog
Αντικειμενοστρεφής Προγραμματισμός (object-oriented programming)	Simula, Smalltalk, C++ Java, C#

- 1 Φιλοσοφία Γλωσσών Προγραμματισμού
- 2 Συναρτησιακός Προγραμματισμός (Functional Programming)
- 3 Λογικός Προγραμματισμός (Prolog)
- 4 Object-Oriented Programming

- Σε μία γλώσσα συναρτησιακού προγραμματισμού, η αποτίμηση μιας συνάρτησης δίνει πάντα το ίδιο αποτέλεσμα για τις ίδιες τιμές των παραμέτρων της.
- Η σημαντική αυτή ιδιότητα δεν ισχύει κατ' ανάγκη στις γλώσσες προστακτικού προγραμματισμού
- Στις προστακτικές γλώσσες αυτό συμβαίνει λόγω:
  - Μεταβλητών που ορίζονται και αλλάζουν τιμές εκτός του σώματος της συνάρτησης (global variables)
  - Εξάρτησης από την κατάσταση (state) του υπολογισμού
  - Άλλων παρενεργειών (side-effects) που μπορεί να υπάρχουν στο πρόγραμμα

**Υπενθύμιση:** Στα μαθηματικά οι συναρτήσεις εξαρτώνται μόνο από τα ορίσματά τους

- Σε μία γλώσσα συναρτησιακού προγραμματισμού, η αποτίμηση μιας συνάρτησης δίνει πάντα το ίδιο αποτέλεσμα για τις ίδιες τιμές των παραμέτρων της.
- Η σημαντική αυτή ιδιότητα δεν ισχύει κατ' ανάγκη στις γλώσσες προστακτικού προγραμματισμού
- Στις προστακτικές γλώσσες αυτό συμβαίνει λόγω:
  - Μεταβλητών που ορίζονται και αλλάζουν τιμές εκτός του σώματος της συνάρτησης (global variables)
  - Εξάρτησης από την κατάσταση (state) του υπολογισμού
  - Άλλων παρενεργειών (side-effects) που μπορεί να υπάρχουν στο πρόγραμμα

Υπενθύμιση: Στα μαθηματικά οι συναρτήσεις εξαρτώνται μόνο από τα ορίσματά τους

- Σε μία γλώσσα συναρτησιακού προγραμματισμού, η αποτίμηση μιας συνάρτησης δίνει πάντα το ίδιο αποτέλεσμα για τις ίδιες τιμές των παραμέτρων της.
- Η σημαντική αυτή ιδιότητα δεν ισχύει κατ' ανάγκη στις γλώσσες προστακτικού προγραμματισμού
- Στις προστακτικές γλώσσες αυτό συμβαίνει λόγω:
  - Μεταβλητών που ορίζονται και αλλάζουν τιμές εκτός του σώματος της συνάρτησης (global variables)
  - Εξάρτησης από την κατάσταση (state) του υπολογισμού
  - Άλλων παρενεργειών (side-effects) που μπορεί να υπάρχουν στο πρόγραμμα

Υπενθύμιση: Στα μαθηματικά οι συναρτήσεις εξαρτώνται μόνο από τα ορίσματά τους

- Σε μία γλώσσα συναρτησιακού προγραμματισμού, η αποτίμηση μιας συνάρτησης δίνει πάντα το ίδιο αποτέλεσμα για τις ίδιες τιμές των παραμέτρων της.
- Η σημαντική αυτή ιδιότητα δεν ισχύει κατ' ανάγκη στις γλώσσες προστακτικού προγραμματισμού
- Στις προστακτικές γλώσσες αυτό συμβαίνει λόγω:
  - Μεταβλητών που ορίζονται και αλλάζουν τιμές εκτός του σώματος της συνάρτησης (global variables)
  - Εξάρτησης από την κατάσταση (state) του υπολογισμού
  - Άλλων παρενεργειών (side-effects) που μπορεί να υπάρχουν στο πρόγραμμα

Υπενθύμιση: Στα μαθηματικά οι συναρτήσεις εξαρτώνται μόνο από τα ορίσματά τους



- Σε μία γλώσσα συναρτησιακού προγραμματισμού, η αποτίμηση μιας συνάρτησης δίνει πάντα το ίδιο αποτέλεσμα για τις ίδιες τιμές των παραμέτρων της.
- Η σημαντική αυτή ιδιότητα δεν ισχύει κατ' ανάγκη στις γλώσσες προστακτικού προγραμματισμού
- Στις προστακτικές γλώσσες αυτό συμβαίνει λόγω:
  - Μεταβλητών που ορίζονται και αλλάζουν τιμές εκτός του σώματος της συνάρτησης (global variables)
  - Εξάρτησης από την κατάσταση (state) του υπολογισμού
  - Άλλων παρενεργειών (side-effects) που μπορεί να υπάρχουν στο πρόγραμμα

Υπενθύμιση: Στα μαθηματικά οι συναρτήσεις εξαρτώνται μόνο από τα ορίσματά τους

- Σε μία γλώσσα συναρτησιακού προγραμματισμού, η αποτίμηση μιας συνάρτησης δίνει πάντα το ίδιο αποτέλεσμα για τις ίδιες τιμές των παραμέτρων της.
- Η σημαντική αυτή ιδιότητα δεν ισχύει κατ' ανάγκη στις γλώσσες προστακτικού προγραμματισμού
- Στις προστακτικές γλώσσες αυτό συμβαίνει λόγω:
  - Μεταβλητών που ορίζονται και αλλάζουν τιμές εκτός του σώματος της συνάρτησης (global variables)
  - Εξάρτησης από την κατάσταση (state) του υπολογισμού
  - Άλλων παρενεργειών (side-effects) που μπορεί να υπάρχουν στο πρόγραμμα

**Υπενθύμιση:** Στα μαθηματικά οι συναρτήσεις εξαρτώνται μόνο από τα ορίσματά τους

- **Δήλωση Συναρτήσεων:**

```
inc(n) = n + 1
```

```
f(t) = t * inc(t)
```

- **Δήλωση Τιμών:**

```
x = f(6)
```

```
y = f(f(2))
```

- **Τύποι:**

```
inc, f :: Int -> Int
```

```
x, y :: Int
```

- **Δήλωση Συναρτήσεων:**

```
inc(n) = n + 1
```

```
f(t) = t * inc(t)
```

- **Δήλωση Τιμών:**

```
x = f(6)
```

```
y = f(f(2))
```

- **Τύποι:**

```
inc, f :: Int -> Int
```

```
x, y :: Int
```

- **Δήλωση Συναρτήσεων:**

`inc(n) = n + 1`

`f(t) = t * inc(t)`

- **Δήλωση Τιμών:**

`x = f(6)`

`y = f(f(2))`

- **Τύποι:**

`inc, f :: Int -> Int`

`x, y :: Int`

# Αναγωγές

$inc(n) = n+1$	ή διαφορετικά για οικονομία παρενθέσεων:	$inc\ n = n+1$
$f(t) = t * inc(t)$		$f\ t = t * inc\ t$
$y = f(f(2))$		$y = f\ (f\ 2)$

Παράδειγμα εσωτερικότερων αναγωγών:

$$\begin{aligned}y &\rightarrow f((f(2))) \rightarrow f(2 * inc(2)) \rightarrow f(2*(2+1)) \rightarrow f(2*3) \\ &\rightarrow f(6) \rightarrow 6 * inc(6) \rightarrow 6*(6+1) \rightarrow 6*7 \rightarrow 42\end{aligned}$$

Παράδειγμα εξωτερικότερων αναγωγών/οκνηρή αποτίμηση (lazy evaluation):

$$\begin{aligned}y &\rightarrow f(f(2)) \rightarrow f(2) * inc(f(2)) \\ &\rightarrow (2 * inc(2)) * (f(2) + 1) \\ &\rightarrow (2*(2+1)) * (2*inc(2)+1) \\ &\rightarrow 6 * (2*(2+1) + 1) \rightarrow 42\end{aligned}$$

# Αναγωγές

$$\begin{array}{l|l} \text{inc}(n) = n+1 & \text{ή διαφορετικά} \\ f(t) = t * \text{inc}(t) & \text{για οικονομία} \\ y = f(f(2)) & \text{παρενθέσεων:} \end{array} \quad \begin{array}{l} \text{inc } n = n+1 \\ f \ t = t * \text{inc } t \\ y = f (f \ 2) \end{array}$$

Παράδειγμα εσωτερικότερων αναγωγών:

$$\begin{aligned} y &\rightarrow f((f(2))) \rightarrow f(2 * \text{inc}(2)) \rightarrow f(2*(2+1)) \rightarrow f(2*3) \\ &\rightarrow f(6) \rightarrow 6 * \text{inc}(6) \rightarrow 6*(6+1) \rightarrow 6*7 \rightarrow 42 \end{aligned}$$

Παράδειγμα εξωτερικότερων αναγωγών/οκνηρή αποτίμηση (lazy evaluation):

$$\begin{aligned} y &\rightarrow f(f(2)) \rightarrow f(2) * \text{inc}(f(2)) \\ &\rightarrow (2 * \text{inc}(2)) * (f(2) + 1) \\ &\rightarrow (2*(2+1)) * (2*\text{inc}(2)+1) \\ &\rightarrow 6 * (2*(2+1) + 1) \rightarrow 42 \end{aligned}$$

# Αναγωγές

$inc(n) = n+1$	ή διαφορετικά για οικονομία παρενθέσεων:	$inc\ n = n+1$
$f(t) = t * inc(t)$		$f\ t = t * inc\ t$
$y = f(f(2))$		$y = f\ (f\ 2)$

Παράδειγμα εσωτερικότερων αναγωγών:

$$\begin{aligned}y &\rightarrow f((f(2))) \rightarrow f(2 * inc(2)) \rightarrow f(2*(2+1)) \rightarrow f(2*3) \\ &\rightarrow f(6) \rightarrow 6 * inc(6) \rightarrow 6*(6+1) \rightarrow 6*7 \rightarrow 42\end{aligned}$$

Παράδειγμα εξωτερικότερων αναγωγών/οκνηρή αποτίμηση (lazy evaluation):

$$\begin{aligned}y &\rightarrow f(f(2)) \rightarrow f(2) * inc(f(2)) \\ &\rightarrow (2 * inc(2)) * (f(2) + 1) \\ &\rightarrow (2*(2+1)) * (2*inc(2)+1) \\ &\rightarrow 6 * (2*(2+1) + 1) \rightarrow 42\end{aligned}$$



```
f(x, y) = if (x == 0) then 1 else f(x-1, f(x, y))
```

Η οκνηρή αποτίμηση δίνει π.χ.:

$$f(2, 0) \rightarrow f(1, f(2, 0)) \rightarrow f(0, f(1, f(2, 0))) \rightarrow 1.$$

ενώ με τις εσωτ. αναγωγές έχουμε:

$$f(2, 0) \rightarrow f(1, f(2, 0)) \rightarrow f(1, f(1, f(2, 0))) \rightarrow \dots$$

`f(x, y) = if (x == 0) then 1 else f(x-1, f(x, y))`

Η οκνηρή αποτίμηση δίνει π.χ.:

$$f(2, 0) \rightarrow f(1, f(2, 0)) \rightarrow f(0, f(1, f(2, 0))) \rightarrow 1.$$

ενώ με τις εσωτ. αναγωγές έχουμε:

$$f(2, 0) \rightarrow f(1, f(2, 0)) \rightarrow f(1, f(1, f(2, 0))) \rightarrow \dots$$

$f(x, y) = \text{if } (x == 0) \text{ then } 1 \text{ else } f(x-1, f(x, y))$

Η οκνηρή αποτίμηση δίνει π.χ.:

$$f(2, 0) \rightarrow f(1, f(2, 0)) \rightarrow f(0, f(1, f(2, 0))) \rightarrow 1.$$

ενώ με τις εσωτ. αναγωγές έχουμε:

$$f(2, 0) \rightarrow f(1, f(2, 0)) \rightarrow f(1, f(1, f(2, 0))) \rightarrow \dots$$

```
twice :: (Int -> Int, Int) -> Int  
twice(f, x) = f(f(x))
```

```
inc(n) = n + 1  
plus2(x) = twice(inc, x)
```

```
plusN :: Int -> (Int -> Int)  
plusN(n) = f  
  where f(x) = x + n
```

# Currying

```
add :: (Int, Int) -> Int
add(x, y) = x + y
```

Μπορεί να γραφεί διαφορετικά με την τεχνική Currying ως εξής:

```
add' :: Int -> (Int -> Int)
add'(x) = f
  where f(y) = x + y
```

Δηλαδή η `add'` δέχεται το πρώτο όρισμα `x` και επιστρέφει μια συνάρτηση η οποία δέχεται το δεύτερο όρισμα `y` και επιστρέφει το άθροισμα `x + y`. Ισχύει λοιπόν η παρακάτω ισοδυναμία:

$$\text{add}(x, y) == (\text{add}'(x))(y)$$

```
factorial n =  
  if n <= 1 then 1  
    else n * factorial (n-1)
```

```
gcd (n, 0) = n  
gcd (n, m) = if (n < m) then gcd (m, n)  
              else gcd (m, n 'mod' m)
```

# Συμπεράσματα

Η απόδειξη της ορθότητας ενός προγράμματος μπορεί να γίνει εύκολα ελέγχοντας τις αντικαταστάσεις (αναγωγές). Η απόδειξη ορθότητας αναδρομικών συναρτήσεων γίνεται με χρήση μαθηματικής επαγωγής.

Πλεονεκτήματα συναρτησιακού προγραμματισμού:

- Συντομία (2-10 φορές μικρότερος κώδικας)
- Ευκολία στην κατανόηση
- Λιγότερα σφάλματα εκτέλεσης
- Επαναχρησιμοποίηση, αφαίρεση, δόμηση
- Αυτόματη διαχείριση μνήμης

Μειονεκτήματα

- Μειωμένη απόδοση στην εκτέλεση
- Μεγαλύτερες απαιτήσεις μνήμης

# Συμπεράσματα

Η απόδειξη της ορθότητας ενός προγράμματος μπορεί να γίνει εύκολα ελέγχοντας τις αντικαταστάσεις (αναγωγές). Η απόδειξη ορθότητας αναδρομικών συναρτήσεων γίνεται με χρήση μαθηματικής επαγωγής.

Πλεονεκτήματα συναρτησιακού προγραμματισμού:

- Συντομία (2-10 φορές μικρότερος κώδικας)
- Ευκολία στην κατανόηση
- Λιγότερα σφάλματα εκτέλεσης
- Επαναχρησιμοποίηση, αφαίρεση, δόμηση
- Αυτόματη διαχείριση μνήμης

Μειονεκτήματα

- Μειωμένη απόδοση στην εκτέλεση
- Μεγαλύτερες απαιτήσεις μνήμης



# Συμπεράσματα

Η απόδειξη της ορθότητας ενός προγράμματος μπορεί να γίνει εύκολα ελέγχοντας τις αντικαταστάσεις (αναγωγές). Η απόδειξη ορθότητας αναδρομικών συναρτήσεων γίνεται με χρήση μαθηματικής επαγωγής.

Πλεονεκτήματα συναρτησιακού προγραμματισμού:

- Συντομία (2-10 φορές μικρότερος κώδικας)
- Ευκολία στην κατανόηση
- Λιγότερα σφάλματα εκτέλεσης
- Επαναχρησιμοποίηση, αφαίρεση, δόμηση
- Αυτόματη διαχείριση μνήμης

Μειονεκτήματα

- Μειωμένη απόδοση στην εκτέλεση
- Μεγαλύτερες απαιτήσεις μνήμης

# Συμπεράσματα

Η απόδειξη της ορθότητας ενός προγράμματος μπορεί να γίνει εύκολα ελέγχοντας τις αντικαταστάσεις (αναγωγές). Η απόδειξη ορθότητας αναδρομικών συναρτήσεων γίνεται με χρήση μαθηματικής επαγωγής.

Πλεονεκτήματα συναρτησιακού προγραμματισμού:

- Συντομία (2-10 φορές μικρότερος κώδικας)
- Ευκολία στην κατανόηση
- Λιγότερα σφάλματα εκτέλεσης
- Επαναχρησιμοποίηση, αφαίρεση, δόμηση
- Αυτόματη διαχείριση μνήμης

Μειονεκτήματα

- Μειωμένη απόδοση στην εκτέλεση
- Μεγαλύτερες απαιτήσεις μνήμης

# Συμπεράσματα

Η απόδειξη της ορθότητας ενός προγράμματος μπορεί να γίνει εύκολα ελέγχοντας τις αντικαταστάσεις (αναγωγές). Η απόδειξη ορθότητας αναδρομικών συναρτήσεων γίνεται με χρήση μαθηματικής επαγωγής.

Πλεονεκτήματα συναρτησιακού προγραμματισμού:

- Συντομία (2-10 φορές μικρότερος κώδικας)
- Ευκολία στην κατανόηση
- Λιγότερα σφάλματα εκτέλεσης
- Επαναχρησιμοποίηση, αφαίρεση, δόμηση
- Αυτόματη διαχείριση μνήμης

Μειονεκτήματα

- Μειωμένη απόδοση στην εκτέλεση
- Μεγαλύτερες απαιτήσεις μνήμης

# Συμπεράσματα

Η απόδειξη της ορθότητας ενός προγράμματος μπορεί να γίνει εύκολα ελέγχοντας τις αντικαταστάσεις (αναγωγές). Η απόδειξη ορθότητας αναδρομικών συναρτήσεων γίνεται με χρήση μαθηματικής επαγωγής.

Πλεονεκτήματα συναρτησιακού προγραμματισμού:

- Συντομία (2-10 φορές μικρότερος κώδικας)
- Ευκολία στην κατανόηση
- Λιγότερα σφάλματα εκτέλεσης
- Επαναχρησιμοποίηση, αφαίρεση, δόμηση
- Αυτόματη διαχείριση μνήμης

Μειονεκτήματα

- Μειωμένη απόδοση στην εκτέλεση
- Μεγαλύτερες απαιτήσεις μνήμης

# Συμπεράσματα

Η απόδειξη της ορθότητας ενός προγράμματος μπορεί να γίνει εύκολα ελέγχοντας τις αντικαταστάσεις (αναγωγές). Η απόδειξη ορθότητας αναδρομικών συναρτήσεων γίνεται με χρήση μαθηματικής επαγωγής.

Πλεονεκτήματα συναρτησιακού προγραμματισμού:

- Συντομία (2-10 φορές μικρότερος κώδικας)
- Ευκολία στην κατανόηση
- Λιγότερα σφάλματα εκτέλεσης
- Επαναχρησιμοποίηση, αφαίρεση, δόμηση
- Αυτόματη διαχείριση μνήμης

Μειονεκτήματα

- Μειωμένη απόδοση στην εκτέλεση
- Μεγαλύτερες απαιτήσεις μνήμης

# Συμπεράσματα

Η απόδειξη της ορθότητας ενός προγράμματος μπορεί να γίνει εύκολα ελέγχοντας τις αντικαταστάσεις (αναγωγές). Η απόδειξη ορθότητας αναδρομικών συναρτήσεων γίνεται με χρήση μαθηματικής επαγωγής.

Πλεονεκτήματα συναρτησιακού προγραμματισμού:

- Συντομία (2-10 φορές μικρότερος κώδικας)
- Ευκολία στην κατανόηση
- Λιγότερα σφάλματα εκτέλεσης
- Επαναχρησιμοποίηση, αφαίρεση, δόμηση
- Αυτόματη διαχείριση μνήμης

Μειονεκτήματα

- Μειωμένη απόδοση στην εκτέλεση
- Μεγαλύτερες απαιτήσεις μνήμης

- 1 Φιλοσοφία Γλωσσών Προγραμματισμού
- 2 Συναρτησιακός Προγραμματισμός (Functional Programming)
- 3 Λογικός Προγραμματισμός (Prolog)
- 4 Object-Oriented Programming

- Το πρόγραμμα είναι εκφρασμένο σε μια μορφή συμβολικής λογικής με δήλωση των σχέσεων μεταξύ των δεδομένων που διαχειρίζεται.
- Η εκτέλεση του προγράμματος ισοδυναμεί με τη διεξαγωγή συλλογισμών σε αυτή τη λογική.
- Βιβλιογραφία: W.F. Clocksin and C.S. Mellish, Programming in Prolog, Springer



- Το πρόγραμμα είναι εκφρασμένο σε μια μορφή συμβολικής λογικής με δήλωση των σχέσεων μεταξύ των δεδομένων που διαχειρίζεται.
- Η εκτέλεση του προγράμματος ισοδυναμεί με τη διεξαγωγή συλλογισμών σε αυτή τη λογική.
- Βιβλιογραφία: W.F. Clocksin and C.S. Mellish, Programming in Prolog, Springer

- Το πρόγραμμα είναι εκφρασμένο σε μια μορφή συμβολικής λογικής με δήλωση των σχέσεων μεταξύ των δεδομένων που διαχειρίζεται.
- Η εκτέλεση του προγράμματος ισοδυναμεί με τη διεξαγωγή συλλογισμών σε αυτή τη λογική.
- Βιβλιογραφία: W.F. Clocksin and C.S. Mellish, Programming in Prolog, Springer

## Γεγονότα:

```
male(john).
male(george).
female(mary).
female(jenny).
parent(john,george).
parent(mary,george).
parent(john,jenny).
parent(mary,jenny).
```

## Κανόνες:

```
father(X,Y) :- parent(X,Y), male(X).
mother(X,Y) :- parent(X,Y), female(X).
human(X) :- male(X).
human(X) :- female(X).
brother(X,Y) :- male(X),
                parent(Z,X),
                parent(Z,Y).
sister(X,Y) :- female(X),
               parent(Z,X),
               parent(Z,Y).
```

?- male(john).

yes

?- male(mary).

no

?- male(peter).

no

?- male(X).

X = john ;

X = george ;

no

?- brother(george,jenny).

yes

?- human(X).

X = john;

X = george;

X = mary;

X = jenny;

no

?- mother(X,george).

X = mary;

no

?- sister(X,Y).

X = jenny, Y = george;

X = jenny, Y = jenny;

X = jenny, Y = george;

X = jenny, Y = jenny;

no

# Παραδείγματα I

```
engineer(X) :- graduate(X), experience(X).  
rational(X) :- X = A/B, integer(A), integer(B), not(B = 0).
```

Μέγιστος Κοινός Διαιρέτης:

```
gcd(X, 0, X).  
gcd(0, X, X).  
gcd(X, Y, D) :- X < Y,  
                Y1 is Y mod X,  
                gcd(X, Y1, D).  
gcd(X, Y, D) :- Y < X,  
                gcd(Y, X, D).
```

```
engineer(X) :- graduate(X), experience(X).  
rational(X) :- X = A/B, integer(A), integer(B), not(B = 0).
```

## Μέγιστος Κοινός Διαίρετης:

```
gcd(X, 0, X).  
gcd(0, X, X).  
gcd(X, Y, D) :- X < Y,  
                Y1 is Y mod X,  
                gcd(X, Y1, D).  
gcd(X, Y, D) :- Y < X,  
                gcd(Y, X, D).
```

## Παραγοντικό:

```
factorial(0, 1).  
factorial(N, Result) :- N > 0,  
                        N1 is N-1,  
                        factorial(N1, Facmin),  
                        Result is Facmin*N.
```

- 1 Φιλοσοφία Γλωσσών Προγραμματισμού
- 2 Συναρτησιακός Προγραμματισμός (Functional Programming)
- 3 Λογικός Προγραμματισμός (Prolog)
- 4 Object-Oriented Programming



# Object-Oriented Programming

## Γλώσσες

Simula (Norway), Smalltalk (Xerox, California), C++ (Denmark), Eiffel (France), Java (Sun Microsystems), C# (Microsoft)

Χαρακτηριστικά:

- messages αντί για procedures (functions)
- objects αντί για data
- classes (families of objects) αντί για types
- Information Hiding
- Data Abstraction
- Dynamic Binding
- Inheritance

## Γλώσσες

Simula (Norway), Smalltalk (Xerox, California), C++ (Denmark), Eiffel (France), Java (Sun Microsystems), C# (Microsoft)

### Χαρακτηριστικά:

- messages αντί για procedures (functions)
- objects αντί για data
- classes (families of objects) αντί για types
- Information Hiding
- Data Abstraction
- Dynamic Binding
- Inheritance

## Γλώσσες

Simula (Norway), Smalltalk (Xerox, California), C++ (Denmark), Eiffel (France), Java (Sun Microsystems), C# (Microsoft)

### Χαρακτηριστικά:

- messages αντί για procedures (functions)
- objects αντί για data
- classes (families of objects) αντί για types
- Information Hiding
- Data Abstraction
- Dynamic Binding
- Inheritance

## Γλώσσες

Simula (Norway), Smalltalk (Xerox, California), C++ (Denmark), Eiffel (France), Java (Sun Microsystems), C# (Microsoft)

### Χαρακτηριστικά:

- messages αντί για procedures (functions)
- objects αντί για data
- classes (families of objects) αντί για types
- Information Hiding
- Data Abstraction
- Dynamic Binding
- Inheritance

## Γλώσσες

Simula (Norway), Smalltalk (Xerox, California), C++ (Denmark), Eiffel (France), Java (Sun Microsystems), C# (Microsoft)

### Χαρακτηριστικά:

- messages αντί για procedures (functions)
- objects αντί για data
- classes (families of objects) αντί για types
- Information Hiding
- Data Abstraction
- Dynamic Binding
- Inheritance

## Γλώσσες

Simula (Norway), Smalltalk (Xerox, California), C++ (Denmark), Eiffel (France), Java (Sun Microsystems), C# (Microsoft)

### Χαρακτηριστικά:

- messages αντί για procedures (functions)
- objects αντί για data
- classes (families of objects) αντί για types
- Information Hiding
- Data Abstraction
- Dynamic Binding
- Inheritance

## Γλώσσες

Simula (Norway), Smalltalk (Xerox, California), C++ (Denmark), Eiffel (France), Java (Sun Microsystems), C# (Microsoft)

### Χαρακτηριστικά:

- messages αντί για procedures (functions)
- objects αντί για data
- classes (families of objects) αντί για types
- Information Hiding
- Data Abstraction
- Dynamic Binding
- Inheritance

## Γλώσσες

Simula (Norway), Smalltalk (Xerox, California), C++ (Denmark), Eiffel (France), Java (Sun Microsystems), C# (Microsoft)

### Χαρακτηριστικά:

- messages αντί για procedures (functions)
- objects αντί για data
- classes (families of objects) αντί για types
- Information Hiding
- Data Abstraction
- Dynamic Binding
- Inheritance



# Object-Oriented Programming

- Η επίλυση προβλημάτων γίνεται μέσω αντικειμένων (objects) που αλληλεπιδρούν: δεδομένα που ξέρουν να εφαρμόζουν μεθόδους στον εαυτό τους.
- Οι μέθοδοι μιας κλάσης (class methods) ορίζουν λειτουργίες που η κλάση ξέρει πώς να κάνει — δεν ορίζουν αντικείμενα της κλάσης. Οι κλήσεις μεθόδων είναι κάτι σαν τις συνηθείς κλήσεις συναρτήσεων στις μη αντικειμενοστρεφείς γλώσσες.
- Το πρόγραμμα είναι οργανωμένο ως ένα σύνολο από αλληλεπιδρώντα αντικείμενα. Κάθε αντικείμενο περιέχει:
  - δεδομένα (data), που χαρακτηρίζουν την κατάστασή του
  - μεθόδους (methods), δηλαδή κώδικα που υλοποιεί τη συμπεριφορά του

- Η επίλυση προβλημάτων γίνεται μέσω αντικειμένων (objects) που αλληλεπιδρούν: δεδομένα που ξέρουν να εφαρμόζουν μεθόδους στον εαυτό τους.
- Οι μέθοδοι μιας κλάσης (class methods) ορίζουν λειτουργίες που η κλάση ξέρει πώς να κάνει — δεν ορίζουν αντικείμενα της κλάσης. Οι κλήσεις μεθόδων είναι κάτι σαν τις συνηθεις κλήσεις συναρτήσεων στις μη αντικειμενοστρεφείς γλώσσες.
- Το πρόγραμμα είναι οργανωμένο ως ένα σύνολο από αλληλεπιδρώντα αντικείμενα. Κάθε αντικείμενο περιέχει:
  - δεδομένα (data), που χαρακτηρίζουν την κατάστασή του
  - μεθόδους (methods), δηλαδή κώδικα που υλοποιεί τη συμπεριφορά του

- Η επίλυση προβλημάτων γίνεται μέσω αντικειμένων (objects) που αλληλεπιδρούν: δεδομένα που ξέρουν να εφαρμόζουν μεθόδους στον εαυτό τους.
- Οι μέθοδοι μιας κλάσης (class methods) ορίζουν λειτουργίες που η κλάση ξέρει πώς να κάνει — δεν ορίζουν αντικείμενα της κλάσης. Οι κλήσεις μεθόδων είναι κάτι σαν τις συνηθείς κλήσεις συναρτήσεων στις μη αντικειμενοστρεφείς γλώσσες.
- Το πρόγραμμα είναι οργανωμένο ως ένα σύνολο από αλληλεπιδρώντα αντικείμενα. Κάθε αντικείμενο περιέχει:
  - δεδομένα (data), που χαρακτηρίζουν την κατάσταση του
  - μεθόδους (methods), δηλαδή κώδικα που υλοποιεί τη συμπεριφορά του

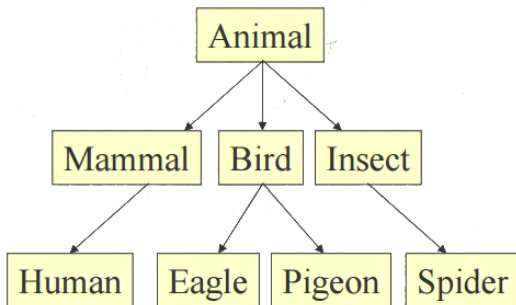
- Η επίλυση προβλημάτων γίνεται μέσω αντικειμένων (objects) που αλληλεπιδρούν: δεδομένα που ξέρουν να εφαρμόζουν μεθόδους στον εαυτό τους.
- Οι μέθοδοι μιας κλάσης (class methods) ορίζουν λειτουργίες που η κλάση ξέρει πώς να κάνει — δεν ορίζουν αντικείμενα της κλάσης. Οι κλήσεις μεθόδων είναι κάτι σαν τις συνηθεις κλήσεις συναρτήσεων στις μη αντικειμενοστρεφείς γλώσσες.
- Το πρόγραμμα είναι οργανωμένο ως ένα σύνολο από αλληλεπιδρώντα αντικείμενα. Κάθε αντικείμενο περιέχει:
  - δεδομένα (data), που χαρακτηρίζουν την κατάσταση του
  - μεθόδους (methods), δηλαδή κώδικα που υλοποιεί τη συμπεριφορά του

- Η επίλυση προβλημάτων γίνεται μέσω αντικειμένων (objects) που αλληλεπιδρούν: δεδομένα που ξέρουν να εφαρμόζουν μεθόδους στον εαυτό τους.
- Οι μέθοδοι μιας κλάσης (class methods) ορίζουν λειτουργίες που η κλάση ξέρει πώς να κάνει — δεν ορίζουν αντικείμενα της κλάσης. Οι κλήσεις μεθόδων είναι κάτι σαν τις συνηθείς κλήσεις συναρτήσεων στις μη αντικειμενοστρεφείς γλώσσες.
- Το πρόγραμμα είναι οργανωμένο ως ένα σύνολο από αλληλεπιδρώντα αντικείμενα. Κάθε αντικείμενο περιέχει:
  - δεδομένα (data), που χαρακτηρίζουν την κατάσταση του
  - μεθόδους (methods), δηλαδή κώδικα που υλοποιεί τη συμπεριφορά του

```
class Circle {  
    real x, y, r;  
    Circle (real radius) {  
        x = 0; y = 0; r = radius;  
    }  
    real circumference() { return 2*3.14*r; }  
    real area() { return 3.14*r*r; }  
}
```

# Ιεραρχίες κλάσεων Κληρονομικότητα (inheritance)

Εξειδίκευση των αντικειμένων μιας κλάσης, υποστηρίζοντας πρόσθετη ή διαφοροποιημένη συμπεριφορά.



- Αφαίρεση, δόμηση, επαναχρησιμοποίηση
- Κατά την ανάλυση και τη σχεδίαση: objects ανάλογα με interface και όχι με implementation.
- Απόκρυψη όσο το δυνατόν περισσότερο της υλοποίησης.
- Επαναχρησιμοποίηση των αντικείμενων.
- Ελαχιστοποίηση διαδράσεων (interactions) μεταξύ αντικειμένων.



- Αφαίρεση, δόμηση, επαναχρησιμοποίηση
- Κατά την ανάλυση και τη σχεδίαση: objects ανάλογα με interface και όχι με implementation.
- Απόκρυψη όσο το δυνατόν περισσότερο της υλοποίησης.
- Επαναχρησιμοποίηση των αντικείμενων.
- Ελαχιστοποίηση διαδράσεων (interactions) μεταξύ αντικειμένων.

- Αφαίρεση, δόμηση, επαναχρησιμοποίηση
- Κατά την ανάλυση και τη σχεδίαση: objects ανάλογα με interface και όχι με implementation.
- Απόκρυψη όσο το δυνατόν περισσότερο της υλοποίησης.
- Επαναχρησιμοποίηση των αντικείμενων.
- Ελαχιστοποίηση διαδράσεων (interactions) μεταξύ αντικειμένων.

- Αφαίρεση, δόμηση, επαναχρησιμοποίηση
- Κατά την ανάλυση και τη σχεδίαση: objects ανάλογα με interface και όχι με implementation.
- Απόκρυψη όσο το δυνατόν περισσότερο της υλοποίησης.
- Επαναχρησιμοποίηση των αντικείμενων.
- Ελαχιστοποίηση διαδράσεων (interactions) μεταξύ αντικειμένων.

- Αφαίρεση, δόμηση, επαναχρησιμοποίηση
- Κατά την ανάλυση και τη σχεδίαση: objects ανάλογα με interface και όχι με implementation.
- Απόκρυψη όσο το δυνατόν περισσότερο της υλοποίησης.
- Επαναχρησιμοποίηση των αντικείμενων.
- Ελαχιστοποίηση διαδράσεων (interactions) μεταξύ αντικειμένων.

Συνεπώς:

- Εύκολη και γρήγορη ανάπτυξη πρωτότυπων συστημάτων λογισμικού.
- Εύκολη τροποποίηση για επαναχρησιμοποίηση και εύκολος εντοπισμός λαθών.

Όμως:

- απώλεια αποδοτικότητας, κόστος μεταγλώττισης, ...

Συνεπώς:

- Εύκολη και γρήγορη ανάπτυξη πρωτότυπων συστημάτων λογισμικού.
- Εύκολη τροποποίηση για επαναχρησιμοποίηση και εύκολος εντοπισμός λαθών.

Όμως:

- απώλεια αποδοτικότητας, κόστος μεταγλώττισης, ...

Συνεπώς:

- Εύκολη και γρήγορη ανάπτυξη πρωτότυπων συστημάτων λογισμικού.
- Εύκολη τροποποίηση για επαναχρησιμοποίηση και εύκολος εντοπισμός λαθών.

Όμως:

- απώλεια αποδοτικότητας, κόστος μεταγλώττισης, ...