Αντικειμενοστραφής Προγραμματισμός στη Μ2000

Γιώργος Καρράς
Δημιουργός της Μ2000

Πρόλογος

Ένα σύγχρονο προγραμματιστικό υπόδειγμα, ο αντικειμενοστραφής προγραμματισμός, έχει ενταχθεί στην ύλη του Λυκείου. Η ΓΛΩΣΣΑ, η ψευδογλώσσα που χρησιμοποιείται στο βιβλίο της ΑΕΠΠ δεν υποστηρίζει τα αντικείμενα, με συνέπεια να περιορίζεται η μάθηση σε μια αφηρημένη υπόσταση αντικειμένων, με διαγράμματα που συμβολίζουν την σύνδεση των αντικειμένων. Αυτό που λείπει είναι η εμπειρική μάθηση μέσω απτών παραδειγμάτων. Για τον δομημένο προγραμματισμό έχουν γραφτεί διερμηνευτές και μέσω αυτών μπορεί ο μαθητής να κατανοήσει το τρόπο προγραμματισμού σε αυτό το υπόδειγμα. Η Μ2000 προτείνεται για χρήση σε όλα το φάσμα της εκπαίδευσης βάσει δυο χαρακτηριστικών: Τη χρήση ελληνικών δεσμευμένων λέξεων, και τη χρήση πολλών προγραμματιστικών παραδειγμάτων συμπεριλαμβανόμενου του αντικειμενοστραφή προγραμματισμού. Η ΓΛΩΣΣΑ του σχολείου αποτυγχάνει στο δεύτερο, ενώ άλλες γλώσσας, όπως η Python, έχουν λεξιλόγιο στα αγγλικά. Το κείμενο αυτό έχει άδεια GNU FDL 1.3 (3 November 2008). Ο διερμηνευτής της Μ2000 και το περιβάλλον της έχει άδεια GNU GPL v.3 (29 June 2007), ανήκει στο ελεύθερο λογισμικό.

Εισαγωγή στο προγραμματισμό

Η γλώσσα Μ2000 είναι μια γλώσσα με επιρροές από την BASIC και την FORTH. Ως προς την BASIC περιέχει ένα μεγάλο σύνολο εντολών. Ως προς την FORTH, περιέχει δυο στοιχεία της, την δυνατότητα να αυξάνουμε το υπάρχον λεξιλόγιο με νέα τμήματα κώδικα, και την χρήση μιας ειδικής στοίβας, του Σωρού Τιμών, κατά την κλήση υποπρογραμμάτων και για γενική χρήση. Ένα από τα αποτελέσματα αυτών των επιρροών είναι η απουσία μιας εντολή RUN, διότι η εκτέλεση ενός τμήματος γίνεται με το όνομά του (σαν να πρόκειται για μια νέα εντολή της γλώσσας).

Για την εγκατάσταση του προγράμματος κάνουμε εκτέλεση του εκτελέσιμου αρχείου M2000Language.exe. Αν το έχουμε κατεβάσει από σύνδεσμο που προτείνεται από το τόπο της Μ2000, στο georgekarras.blogspot.gr τότε το αρχείο έχει υπογραφή (την βλέπουμε στις ιδιότητες). Μπορούμε να βγάλουμε τον αποκλεισμό που εισάγει αυτόματα το σύστημα για αρχεία exe, και να το εκτελέσουμε. Μπορούμε πριν την εκτέλεση να χειριστούμε το πιστοποιητικό (την υπογραφή) καταχωρώντας την μη αυτόματα στο κατάλογο πιστοποιητικών ρίζας του συστήματος, πάλι από τις ιδιότητες του αρχείου.

Στη πρώτη εκτέλεση γράφουμε την εντολή **Ρυθμίσεις** και επιλέγουμε από τη φόρμα διαλόγου Ρυθμίσεις, τον τρόπο αρχικής εμφάνισης της κονσόλας, χρώματα, γραμματοσειρά και διάστιχο, και τη γλώσσα των μηνυμάτων ως char type. Επιλέξτε Greek γιατί αρχικά η Μ2000 είναι ρυθμισμένη στα αγγλικά (char type Latin). Η εντολή Ρυθμίσεις εκτελείται και με **Ctrl+U** από την κονσόλα.

Για κάθε εντολή υπάρχει η Βοήθεια. Το Βοήθεια Όλα δείχνει όλα τα θέματα της βοήθειας στα ελληνικά, ενώ το Βοήθεια All δείχνει όλα τα θέματα της βοήθειας στα αγγλικά.

Το κενό πρόγραμμα

Ένα κενό πρόγραμμα περιλαμβάνει ένα τμήμα με ένα όνομα και καμία εντολή, οπότε η εκτέλεσή του δεν κάνει τίποτα! Έχει σημασία όμως η σύνταξή του. Παρατηρήστε ότι χρησιμοποιούνται αγκύλες για να δηλώσουν το μέρος του κώδικα που σχετίζεται με το τμήμα. Αντίθετα με τη Python, η M2000 δεν χρησιμοποιεί εσοχές για να ορίσει μπλοκ κώδικα (δομή ακολουθίας), αλλά αγκύλες και σε ορισμένες δομές εναλλακτικά με δήλωση τέλους ακολουθίας. Αντίθετα με τη Java η αλλαγή γραμμής δηλώνει αλλαγή εντολής, έτσι δεν βάζουμε το χαρακτήρα του ερωτηματικού ";" ως τέλος εντολής.

```
Τμήμα Αλφα {
}
Αλφα
```

***Πρόγραμμα 1**: Κενό Πρόγραμμα*

Στο παραπάνω πρόγραμμα ο διερμηνευτής πρώτα βρίσκει τον ορισμό της Αλφα και τον καταχωρεί σε μια λίστα ορισμών. Μετά βρίσκει την Αλφα ως δήλωση και επειδή υπάρχει στη λίστα ως τμήμα εκτελεί το τμήμα. Αυτό το πρόγραμμα μπορεί να βρίσκεται σε ένα αρχείο και να το φορτώσει και να το εκτελέσει ο διερμηνευτής. Εναλλακτικά από την κονσόλα της M2000, που ξεκινάει αν δεν έχουμε δώσει κάποιο πρόγραμμα για εκτέλεση, μπορούμε να εκτελέσουμε αυτά τα βήματα (σε παρένθεση φαίνεται η ενέργεια που γίνεται):

1. **Συγγραφή Αλφα** (πατάμε το Enter, ανοίγει ο διορθωτής προγράμματος)
2. (πατάμε ένα Enter, για να περιέχει κάτι το πρόγραμμα, και μετά το ESC για να κλείσει ο διορθωτής)
3. **Αλφα** (πατάμε Enter και τρέχει το πρόγραμμα χωρίς να κάνει τίποτα.

Στα βήματα 1 και 3 βλέπουμε το δρομέα στη κονσόλα (μια κάτω γραμμή _ να αναβοσβήνει) και εκεί γράφουμε στο βήμα 1 το Συγγραφή Αλφα (ή Σ Αλφα για συντόμευση) και στο βήμα 3 το Αλφα.

Το πρόγραμμα HelloWorld

Σε όλες τις γλώσσες δίνουν ένα πρόγραμμα που δείχνει το Hello World στην οθόνη του υπολογιστή. Σε σχέση με το κενό πρόγραμμα έχει μια μόνο δήλωση ή εντολή, με το αναγνωριστικό Τύπωσε και το αλφαριθμητικό "Hello World".

```
Τμήμα HelloWorld {
    Τύπωσε "Hello Wolrd"
}
HelloWorld
```

***Πρόγραμμα 2**: HelloWorld*

Και ένα παράδειγμα με χρήση γραφικής διεπαφής, όπου το Hello World γράφεται στο τίτλο της φόρμας (Παράθυρο):

```
Τμήμα Γραφική_Διεπαφή {
    Όρισε Φόρμα1 Φόρμα
    Με Φόρμα1 ,"Title", "Hello World"
    Μέθοδος Φόρμα1 ,"Show", 1
    Όρισε Φόρμα1 Τίποτα
}
Γραφική_Διεπαφή
```

***Πρόγραμμα 3**: HelloWorld με Γραφική Διεπαφή*

Χειρισμός Προγραμμάτων

Όπως είδαμε οι εντολές από την κονσόλα της M2000 εκτελούνται όταν πατάμε το πλήκτρο Enter.

- Για να σώσουμε το πρόγραμμα στο δίσκο τότε δίνουμε το **Σώσε Αλφα** ή όποιο άλλο όνομα θέλουμε. Αν θέλουμε να σώσουμε ξανά το πρόγραμμα μετά από αλλαγές πατάμε το συνδυασμό πλήκτρων **CTRL+A** το οποίο σώνει αφού ρωτήσει αν σίγουρα θέλουμε να γράψουμε πάνω στο παλιό στο δίσκο. Πάντα σώνεται και η προηγούμενη έκδοση με το ίδιο όνομα και κατάληξη bck, ενώ η κανονική κατάληξη είναι η gsb.
- Για να σβήσουμε τη λίστα τμημάτων δίνουμε την εντολή **Νέο**
- Για να φορτώσουμε το πρόγραμμα τότε δίνουμε την εντολή **Φόρτωσε Αλφα** ή όποιο άλλο όνομα είχαμε δώσει στην εντολή Σώσε. Η φόρτωσε εμφανίζεται και με τον συνδυασμό **CTRL+Φ**
- Για να δούμε τι έχουμε φορτωμένο με ονόματα τότε δίνουμε την εντολή **Τμήματα ?** (μαζί με το αγγλικό ερωτηματικό). Αν δεν έχουμε τίποτα φορτωμένο θα μας δείξει ό,τι υπάρχει στο δίσκο. Η εντολή αυτή εκτελείται με και με το **CTRL+N**.
- Για να δούμε τι έχουμε φορτωμένο και τι έχουμε στο δίσκο τότε δίνουμε την εντολή **Τμήματα**.

Με τις εντολές, που μέχρι τώρα γνωρίσαμε, μπορούμε να φτιάχνουμε τμήματα, να τα σώνουμε, να τα φορτώνουμε, να βλέπουμε τα φορτωμένα τμήματα και να βλέπουμε τι έχουμε στο δίσκο.

Διακριτά στοιχεία της M2000

Στη M2000 η σύνταξη της γλώσσας δηλώνει συνάμα και τα διακριτά στοιχεία της. Όλα τα διακριτά στοιχεία ανήκουν στις Εντολές της γλώσσας.

Σύνταξη Εντολών

Κάθε εντολή χωρίζεται σε αριστερή και δεξιά έκφραση. Ενδέχεται μια εντολή να μην έχει δεξιά έκφραση.

- Μια εντολή όπως η Νέο δεν έχει δεξιά έκφραση.
- Μια εντολή όπως η Τύπωσε 10, 20 ή ? 10, 20 (το σύμβολο "?" ως αριστερή έκφραση είναι η συντομογραφία της Τύπωσε), έχει αριστερή έκφραση το Τύπωσε ή το ? και δεξιά έκφραση

το 10, 20, δηλαδή μια λίστα δεξιών εκφράσεων με διαχωριστικό το κόμμα. Οι δεξιές εκφράσεις μπορούν να περιέχουν λίστα με δεξιές εκφράσεις.

- Μια εντολή όπως η A=10 έχει μια αριστερή έκφραση με έναν τελεστή, εδώ το "=" και μια δεξιά μια σταθερή τιμή το 10. Το 10 λέγεται σταθερή τιμή γιατί δεν αλλάζει κάθε φορά που εκτελείται αυτή η εντολή. Ο τελεστής στην αριστερή έκφραση μπορεί να είναι τελεστής εκχώρησης τιμής όπως στο παράδειγμα ή τελεστής διαμόρφωσης τιμής πχ το "++" που αυξάνει την τιμή του Α κατά ένα, χωρίς να δέχεται δεξιά έκφραση δηλαδή Α++ ως έχει.
- Μια εντολή μπορεί να έχει μια αριστερή έκφραση με δεξιά ενσωματωμένη όπως το Ρουτίνα1(10,20) η οποία στη δεξιά ενσωματωμένη έκφραση έχουμε μια λίστα δεξιών εκφράσεων τα 10 και 20.
- Μια εντολή μπορεί να έχει και δεξιά ενσωματωμένη και τελεστή και πιθανόν μια δεξιά έκφραση δεξιά από τον τελεστή. Το ΠίνακαςΑ(10)+=50 είναι μια εντολή με ενσωματωμένη δεξιά έκφραση το 10, το τελεστή += και την δεξιά έκφραση 50.

Όνομα Εντολής

Το όνομα μιας εντολής μπορεί να έχει χαρακτήρα στην αρχή και να ακολουθούν χαρακτήρες αριθμοί ή η κάτω παύλα _. Στους χαρακτήρες μπορούν να είναι τα σύμβολα "[" και "]". Το όνομα [] είναι δεσμευμένο. Στα γράμματα δεν είναι σημαντικά τα πεζά ή τα κεφαλαία και οι τόνοι στα ελληνικά. Ο τελεστής "=" χρησιμοποιείται και σε εντολή ως αριστερή έκφραση έτσι το =10 ως μια εντολή λέει "επιστροφή τιμής 10".

Αριθμοί

Οι σταθεροί αριθμοί δεν χρειάζονται το σημείο των δεκαδικών (στο πρόγραμμα είναι πάντα η τελεία), για να δηλωθούν ως πραγματικοί. Εξ ορισμού οι αριθμοί είναι πραγματικοί (διπλής ακρίβειας). Υπάρχουν τύποι αριθμητικοί που απαιτούν κάποια σύμβολα στην αρχή ή στο τέλος ή στην αρχή και το τέλος. Για παράδειγμα ο αριθμός 0x100& είναι ο αριθμός 256 σε δεκαεξαδική μορφή και το σύμβολο & στο τέλος σημαίνει ότι λογίζεται ως αριθμός με πρόσημο, δηλαδή το 0xFFFFFFFF& είναι το -1 σε ακέραιο 32bit, ενώ το 0xFFFFFFFF είναι αριθμός χωρίς πρόσημο και είναι ο αριθμός 4294967295. Ενώ το 0xFFFF% είναι το -1 σε ακέραιο 16bit.

Ο 10 είναι διπλός (πραγματικός), ο 10& είναι μακρύς (ακέραιος 32bit), ο 10% είναι ακέραιος (16bit), ο 10~ είναι απλός (πραγματικός), ο 10@ είναι αριθμός με 29 ψηφία (Decimal), ο 10# είναι ο λογιστικός (Currency). Συνήθως με χρήση πραγματικών καλύπτουμε πολλά παραδείγματα. Οι τελεστές ΔΙΑ και ΥΠΟΛ (ή ΥΠΟΛΟΙΠΟ) δουλεύουν σωστά με πραγματικούς, χωρίς να χρειάζεται να καταφύγουμε σε ακέραιους.

Αλφαριθμητικά

Τα σταθερά αλφαριθμητικά χρησιμοποιούν τα διπλά εισαγωγικά ως αρχή και τέλος, ή τις αγκύλες. Οι αγκύλες χρησιμοποιούνται και για τη δομή ακολουθίας, αλλά αναγνωρίζει ο διερμηνευτής όπως και ο χρωματιστής προγράμματος στο διορθωτή, πότε οι αγκύλες είναι μέρος δομής άρα είναι μέρος δομής ακολουθίας και πότε είναι αλφαριθμητικό (δηλαδή ανήκει σε δεξιά έκφραση). Οι αγκύλες στα αλφαριθμητικά μπορούν να περιέχουν αλλαγές γραμμών.

Άσπρο Διάστημα

Σε όλες τις γλώσσες ορίζεται τι είναι άσπρο διάστημα, δηλαδή τι λογαριάζεται σαν διάστημα ενώ δεν φαίνεται σαν διάστημα. Στη Μ2000 άσπρο διάστημα είναι ο χαρακτήρας TAB (κωδικός 9) και ο χαρακτήρας NBS ή non breaking space (κωδικός 160), πέρα από το διάστημα (κωδικός 32).

Διαχωριστικά Εντολών

Μεταξύ των εντολών σε μια γραμμή μπορεί να βρίσκεται η άνω και κάτω τελεία ως διαχωριστικό εντολής. Μια αλλαγή γραμμής είναι διαχωριστικό αλλαγής εντολής σε μια δομή ακολουθίας. Διαχωριστικά εντολών είναι και οι αγκύλες, οι οποίες δηλώνουν έναν μπλοκ εντολών. Έτσι αν σε μια εντολή ξεκινάμε με αγκύλη πρέπει κάπου να κλείσει η αγκύλη και αυτό δηλώνει ότι ξεκινάει μια ξεχωριστή δομή ακολουθίας. Στη δομή επιλογής θα δούμε ότι μια σειρά εντολών ως το τέλος μιας γραμμής μπορεί να θεωρηθεί ως μια διακλάδωση που μπορεί να εκτελεστεί ή όχι, δηλαδή ως ένα μπλοκ χωρίς να είναι μπλοκ εντολών.

Οι χαρακτήρες $ και % στα ονόματα

Όταν ένα όνομα έχει το $ στο τέλος του (ή ακριβώς πριν το άνοιγμα παρενθέσεων), τότε λέμε ότι είναι ένα όνομα για αλφαριθμητικά, που επιστρέφουν ή εκχωρούνται (ανάλογα το τι είναι) αλφαριθμητικές τιμές.

Όταν ένα όνομα έχει το % στο τέλος του (ή ακριβώς πριν το άνοιγμα παρενθέσεων) τότε λέμε ότι είναι μια αριθμητική τιμή που δεν κρατάει δεκαδικά. Η αποκοπή δεκαδικών σε αυτόν τον τύπο γίνεται στο μισό, όπως στο τρόπο που γίνεται στο σχολείο, έτσι το 0.5 γίνεται 1 ενώ το -0.5 γίνεται -1.

Τύποι Εκφράσεων

Στη Μ2000 υπάρχουν δυο βασικοί τύποι εκφράσεων, ο αριθμητικός και ο αλφαριθμητικός. Στους αλφαριθμητικούς τα ονόματα έχουν το $ στο τέλος. Στους αριθμητικούς τα ονόματα μπορεί να έχουν και το % στο τέλος. Στους αριθμητικούς τύπους συμπεριλαμβάνονται και οι λογικές εκφράσεις. Οι λογικές εκφράσεις επιστρέφουν τύπο Λογικό, ενώ κάθε μη μηδενική τιμή λογίζεται ως Αληθές ή Αληθής και η μηδενική τιμή ως Ψευδές ή Ψευδής. Οι σταθερές Αληθές ή Αληθής έχουν τιμή -1, και οι Ψευδές ή Ψευδής τιμή 0 (δεν είναι τύπου Λογικός, αλλά πραγματικός). Για τα παραδείγματα θα χρησιμοποιήσουμε τη εντολή Τύπωσε στην συντομογραφία της με το αγγλικό ερωτηματικό ?. Η Τύπωσε δέχεται μια λίστα εκφράσεων και κατά την εκτέλεση ελέγχει πριν εκτελέσει την κάθε έκφραση τι τύπου είναι για να την δώσει στον κατάλληλο επεξεργαστή εκφράσεων. Στα δεξιά από την εντολή είναι μια επεξήγηση που διαχωρίζεται από την εντολή με την απόστροφο '.

Παραδείγματα

```
? 12*234                    ' αριθμητική έκφραση.
? "α">"β"                   ' λογική έκφραση, ανήκει στις αριθμητικές εκφράσεις.
? Α>5 και Β<3               ' λογική έκφραση.
? "αλφα"                    ' Αλφαριθμητική έκφραση
```

? "α" + "β"	' Αλφαριθμητική έκφραση
? "α1" ~ "α[12]"	' λογική έκφραση
Τα επόμενα τα λογαριάζουμε χωρίς τα περιεχόμενα μέσα στις παρενθέσεις.	
? Δεξί$(A$, 3)	' Δεξί$(), έχει το $ άρα είναι αλφαριθμητική έκφραση.
? Δεξί$(A$, 3)<>"αβγ"	' Δεξί$()<>"αβγ" λογική έκφραση (τελεστής σε αλφαριθμητικά).
? (1,"αλφα",3,4)#Τιμή$(1)	' ()#Τιμή$(), έχει το $ άρα είναι αλφαριθμητική έκφραση.
? ("α", β$)	' () αριθμητική έκφραση.
? Μήκος(α$)	' Μήκος() αριθμητική έκφραση
? &A	' Αλφαριθμητική έκφραση (σχετίζεται με το όνομα Α).

Επιστροφή Αντικειμένων

Στις εκφράσεις ενδέχεται να έχουμε επιστροφή αντικειμένων, είτε αυτά λογίζονται ως τιμές είτε λογίζονται ως δείκτες σε τιμές. Όταν λογίζονται ως τιμές τότε αυτό που θα πάρουμε είναι ένα αντίγραφο τιμής, όταν λογίζονται ως δείκτες θα πάρουμε αντίγραφο του δείκτη. Ειδικότερα για αντικείμενα θα δούμε σε άλλο κεφάλαιο.

Παραδείγματα Εκφράσεων με Αντικείμενα

? (1,2,3,4,5)	' το (1,2,3,4,5) είναι αντικείμενο (αυτόματη εμφάνιση στοιχείων)
(A$, B)=("Γιώργος", 1000)	' το A$ θα πάρει το όνομα και το Β την τιμή από το πίνακα.
Μ=("Γιώργος", 1000)	' το Μ έχει το δείκτη στο πίνακα
? Μήκος(Μ)=2	' Η Μήκος() χειρίζεται τιμής και αντικείμενα.
Πίνακας Α(2)=2	' Ο πίνακας Α() έχει δυο στοιχεία Α(0) και Α(1) με τιμή 2 σε καθένα.
Μ=Α()	' το Μ δείχνει τώρα στον Α()
Πίνακας Β()	' Ο πίνακας Β() είναι χωρίς στοιχεία
Β()=Α()	' Ο πίνακας Β() έχει ένα αντίγραφο του πίνακα Α()
Λ=Β()	' το Λ δείχνει τον πίνακα Β()
? Μ Είναι Λ	' Μας επιστρέφει τιμή 0 ή Ψευδής, το Μ δεν είναι το Λ
Λ=Α()	' Τώρα το Λ δείχνει το Α()
? Μ Είναι Λ	' Μας επιστρέφει τιμή -1 ή Αληθής, το Μ είναι το Λ

Ο τελεστής Είναι χρησιμοποιείται στα αντικείμενα, και στους δείκτες σε αντικείμενα. Όταν λέμε ότι το Μ είναι το Λ εννοούμε ότι αυτό που δείχνει το Μ είναι αυτό που δείχνει το Λ. Έχουμε δυο διαφορετικά ονόματα με το καθένα να έχει δικό του χώρο καταχώρησης στη μνήμη. Αυτό που καταχωρούν εδώ λέγεται δείκτης, και στο Μ και στο Λ είναι το ίδιο. Ο τελεστής δεν αναφέρεται

στις μεταβλητές αλλά στα περιεχόμενά τους. Δεν κοιτάει για "ίσα" στοιχεία, αλλά για ίδιο αντικείμενο.

Η εντολή Β()=Α() κάνει αντιγραφή του Α() πίνακα στον Β(). Αυτό γίνεται γιατί όταν ένας πίνακας (όχι δείκτης σε πίνακα) είναι στην αριστερή έκφραση και η δεξιά έκφραση φέρνει έναν πίνακα (οτιδήποτε, είτε πίνακα είτε δείκτη σε πίνακα), τότε γίνεται αντιγραφή των στοιχείων του δεξιού πίνακα στον αριστερό. Δείτε ότι ο Β() δεν είχε οριστεί με διαστάσεις. Αυτό δεν ενοχλεί τον διερμηνευτή, θα αντιγράψει τον Α() ως έχει, με τις διαστάσεις του (εδώ έχει μία, με δύο στοιχεία).

Σε μια εκχώρηση τιμής, με μια εντολή Κ=Λ, δεν φαίνεται αν έχουμε εκχώρηση τιμής ή δείκτη σε αντικείμενο. Αυτό που μας ενδιαφέρει είναι: τι είναι το Κ (πχ μπορεί να είναι νέο όνομα) και τι το Λ (δεν μπορεί να είναι νέο όνομα). Αν το Λ είναι ο αριθμός 3 και το Κ ένα νέο όνομα τότε το Κ θα πάρει την ίδια τιμή με το Λ, έτσι τα Κ και Λ είναι τιμές, θα έχουμε εκχώρηση τιμής. Αν το Λ είναι δείκτης σε πίνακα, τότε το Κ θα γίνει δείκτης σε πίνακα, και θα πάρει την ίδια τιμή του Λ, επειδή έχουμε δείκτη. Λέμε ότι το Κ είναι δείκτης σε αντικείμενο που δείχνει το Λ, διότι αυτό είναι πιο σημαντικό από το να σκεφτόμαστε ότι στην ουσία εκχώρηση τιμής έγινε, και να λέγαμε ότι το Κ είναι αντίγραφο του Λ. Ο τύπος του δείκτη χαρακτηρίζει το όνομα ως αντικείμενο του ίδιου τύπου.

Δείτε στο παράδειγμα πιο πάνω, ότι το Μ έδειχνε σε ένα πίνακα που σχηματίσθηκε με παρενθέσεις και τιμές, και όχι με μια εντολή Πίνακας. Αυτό δεν αλλάζει το γεγονός ότι έχουμε πίνακες. Αυτό που τα διαφοροποιεί είναι ότι ο πίνακας με όνομα με παρενθέσεις μπορεί να πάρει νέα τιμή όπως το Β()=Α(), και αν πριν την εκχώρηση ένας δείκτης έδειχνε το Β() θα συνεχίσει να το δείχνει, γιατί η εκχώρηση τιμής (εδώ των τιμών του Α()) δεν αλλάζει το αντικείμενο φέρνοντας ένα νέο αντικείμενο αλλά το αλλάζει εσωτερικά, εδώ του βάζει διάσταση και στοιχεία.

Το παράδειγμα με τον πίνακα είναι μια περίπτωση, όπου δεν ορίζεται η αντιγραφή από τη δεξιά έκφραση, από αυτό που δίνουμε, αλλά από αυτό που παίρνει, από την αριστερή έκφραση. Το παράδειγμα με την αριθμητική τιμή, ορίζεται η αντιγραφή από την δεξιά έκφραση. Η δεξιά έκφραση επιστρέφει μια τιμή που εξάγει ο επεξεργαστής αριθμητικής έκφρασης. Ακόμα και μια τιμή να υπάρχει, όπως το 3 στο Χ=3 θα περάσει από τον επεξεργαστή αυτόν και θα γραφτεί (άρα θα αντιγραφεί) σε ένα αποτέλεσμα, και αυτό το αποτέλεσμα θα αντιγραφεί τελικά στο χώρο μνήμης του Χ.

Δομή Ακολουθίας

Η πιο γενική δομή στο προγραμματισμό είναι η δομή ακολουθίας. Όλες οι εντολές βρίσκονται σε μια νοητή σειρά εκτέλεσης, από τη πρώτη έως τη τελευταία. Μια δομή ακολουθίας μπορεί να περιέχει δομές επιλογής και δομές επανάληψης, αλλά λογαριάζουμε κάθε μια από αυτές τις δομές ως μια εντολή, στη σειρά με τις άλλες εντολές της ακολουθίας. Μια δομή ακολουθίας είναι σαν μια εντολή, σαν μια ολοκληρωμένη ενέργεια, που μπορούμε να αναφερθούμε περιφραστικά σε αυτήν με ακρίβεια, όπου κάθε φορά που θα περάσουμε σε αυτήν θα εφαρμοστεί η ενέργεια μέχρι το πέρας της, το τέλος της ακολουθίας.

Είναι σημαντικό να μπορούμε να περιγράφουμε τι κάνει μια ακολουθία, όχι ειδικά με λεπτομέρεια αλλά επί της ουσίας, πώς μπαίνουμε στην ακολουθία (με τι), τι γίνεται (πάνω σε αυτό που είχαμε κατά την είσοδο) και τι περιμένουμε να έχει γίνει (σε αυτό) στο τέλος.

Χειρισμός Δομής Ακολουθίας

Αντίθετα με το δομημένο προγραμματισμό στη Μ2000 μπορούμε να εκτελέσουμε έξοδο από τη δομή ακολουθίας ή να την χειριστούμε σαν δομή επανάληψης. Θα επισημανθεί όπου συμβαίνει παραβίαση του δομημένου προγραμματισμού. Η ακολουθία πρέπει να έχει μια είσοδο και μια έξοδο. Στη Μ2000 μπορούμε να το αλλάξουμε αυτό, αλλά δεν είναι αναγκαίο να το κάνουμε, γιατί μπορούμε να έχουμε ξεχωριστές δομές για ξεχωριστές ολοκληρωμένες ενέργειες. Έτσι αντί να εφαρμόσουμε αριθμούς γραμμών ή ετικέτες που δείχνουν σημεία στην ακολουθία και άλματα σε αυτά τα σημεία, κάτι που κάνει τον κώδικα να λέγεται σπαγγέτι προγραμματισμός, ομαδοποιούμε τη κάθε διαφορετική ακολουθία σε ξεχωριστές ρουτίνες, δηλαδή σε επώνυμες δομές ακολουθίας, και εκτελούμε την κατάλληλη κάθε φορά.

Θα δούμε αργότερα όλα τα είδη των επώνυμων δομών ακολουθίας, ενώ ήδη έχουμε δει ένα, το Τμήμα, χωρίς να δούμε επακριβώς τα χαρακτηριστικά του.

Σημειώσεις

Έχει αναφερθεί σε παραδείγματα η σημείωση με το χαρακτήρα ' (απόστροφος). Ο λόγος που βάζουμε σημειώσεις είναι για να θυμηθούμε κάτι ιδιαίτερο, ή για να μπορεί να διαβάσει άλλος το πρόγραμμα και να ενημερωθεί κατάλληλα. Η απόστροφος χρησιμοποιείται στο τέλος μιας γραμμής αλλά μπορεί να μπει και η πλάγια γραμμή \.

Αν θέλουμε να αφαιρέσουμε μια γραμμή με εντολές χωρίς να χαθεί ο χρωματιστός στο διορθωτή βάζουμε το ΣΗΜ στην αρχή της εντολή (ισχύει για όλη τη γραμμή).

Ορισμένες φορές θέλουμε να αφαιρέσουμε ένα κομμάτι κώδικα χωρίς να το σβήσουμε, γιατί θέλουμε αργότερα να το βάλουμε σε λειτουργία. Ο τρόπος να "σκιάσουμε" κώδικα, για να μην εκτελείται, είναι να τον κάνουμε μια σημείωση πολλαπλών γραμμών με την ΣΗΜ { } όπου το μπλοκ χρωματίζεται όπως τα αλφαριθμητικά.

Δομή Επιλογής

Μια δομή επιλογής επιλέγει μια διαφορετική ακολουθία υπό μια συνθήκη. Ουσιαστικά θέλουμε να αλλάξουμε μια λειτουργία του προγράμματος όταν συμβαίνει κάτι. Στον δομημένο προγραμματισμό οι δομές επιλογής περιέχουν τις ακολουθίες που ελέγχουν.

Εδώ αξίζει να αναφερθεί η έννοια της φωλιασμένης δομής. Μια δομή είναι αρκετή να σχηματίσει μια ακολουθία εντολών (και μια εντολή να έχουμε θεωρείται μια ακολουθία μιας εντολής). Έτσι μια ακολουθία επιλογής σε μια δομή επιλογής μπορεί να περιέχει ή να είναι απλά μια άλλη δομή επιλογής.

Θα μπορούσε κανείς να σκεφτεί ότι θα υπάρχει τρόπος να έχουμε μεταβλητή διάρθρωση φωλιασμένων επιλογών. Αυτό δεν γίνεται με κώδικα "στατικό". Στατικός είναι ο κώδικας που δεν αλλάζει δομή. Όταν αναφερόμαστε σε δομές προγράμματος, όπως αυτές για επιλογές, λέμε για στατικό κώδικα, έτσι ότι έχουμε φωλιασμένο είναι ξεκάθαρο, φανερό πριν την εκτέλεση. Αυτό που δεν είναι ξεκάθαρο είναι το αν θα εκτελεστεί κάποια επιλογή (κάποια ακολουθία εντολών). Στη Μ2000 με συναρτήσεις λάμδα μπορούμε να έχουμε δομές που δημιουργούνται δυναμικά κατά την εκτέλεση. Στο αρχείο info, που δίνεται με την εγκατάσταση του περιβάλλοντος της γλώσσας, υπάρχει το PermStep τμήμα όπου δημιουργούνται συνθέσεις συναρτήσεων στο βάθος που θέλουμε,

όπου τελικά παίρνουμε μια συνάρτηση και αυτή δίνει σε κάθε εκτέλεση τον επόμενο συνδυασμό μιας σειράς συμβόλων ή αριθμών, συν με αναφορά μια τιμή που λέει αν οι συνδυασμοί τέλειωσαν.

Στη Μ2000 υπάρχουν οι παρακάτω δομές και σχήματα:

- Επιλογή μιας ακολουθίας αν συμβαίνει μια συνθήκη (μια λογική έκφραση), δομή: **Αν Τότε**
- Επιλογή μιας από δυο ακολουθίες. Το αποτέλεσα μιας λογικής έκφραση επιτρέπει μια από τις δυο ακολουθίες να εκτελεστούν, δομή: **Αν Τότε Αλλιώς**
- Έλεγχος διαδοχικά πολλών λογικών εκφράσεων μέχρι να βρεθεί μια που να δώσει συγκεκριμένη ακολουθία για εκτέλεση, δομή: **Αν Τότε Αλλιώς.Αν Τότε Αλλιώς**
- Επιλογή βάσει μιας αριθμητικής ή αλφαριθμητικής έκφρασης και επιμέρους πολλών συνθηκών. Εδώ έχουμε μια πολλαπλή επιλογή, με πολλές ακολουθίες, με δυνατότητα να ενώσουμε ακολουθίες. Η δυνατότητα ένωσης δεν ανήκει στο δομημένο προγραμματισμό, δομή: **Επίλεξε Με ...Τέλος Επιλογής**. *Η Επίλεξε γράφεται και Επέλεξε*.
- Επιλογή βάσει αριθμού από λίστα ετικετών για άλμα. Δεν ανήκει στο δομημένο προγραμματισμό, δομή: **Από** Α **Προς** 100, 200, άλφα, βήτα
- Επιλογή βάσει αριθμού από λίστα ετικετών για κλήση απλής ρουτίνας. Επειδή μια ρουτίνα με κλήση ετικέτας ενδέχεται να έχει πολλές εισόδους (να έχει και άλλες ετικέτες), και αυτή δεν ανήκει στο δομημένο προγραμματισμό, δομή: **Από** Α **Διαμέσου** 100, 200, άλφα, βήτα
- Άλμα σε ετικέτα. Χωρίς συνθήκη. **Προς** 100 ή Προς Άλφα
- Κλήση ρουτίνας με ετικέτα. Χωρίς συνθήκη. **Διαμέσου** 100 ή Διαμέσου Άλφα

Συναρτήσεις Αν() και Αν$()

Εκτός από τις περιπτώσεις που θέλουμε να αλλάξουμε τη ροή εκτέλεσης σε μια άλλη ακολουθία, θα θέλαμε να επιλέξουμε βάσει συνθήκης μια έκφραση, ή αριθμητικά. Υπάρχει δηλαδή η επιλογή ως συνάρτηση, σε έκφραση: **Αν(** Α=Β -> Αληθές, Ψευδές**)**, **Αν$(**Α=Β -> "Ναι", "Όχι"**)**, και με πολλαπλή επιλογή έκφρασης: **Αν(**Μ -> 100, 200, 400, 550, 800**)**, **Αν$(**Μ -> "Κ", "Λ", "Ο","Α"**)**. Σε κάθε περίπτωση ο διερμηνευτής εκτελεί μόνο την έκφραση που επιλέγουμε.

Παραδείγματα Δομών Επιλογής

Στα παραδείγματα θα αναφερθούν οι τρόποι σύνταξης των δομών επιλογής. Στο παρακάτω παράδειγμα μετά το αναγνωριστικό **Τότε** ακολουθούν δυο εντολές, οι οποίες αποτελούν μια ακολουθία μέχρι το τέλος της γραμμής.

```
Αν Χ=100 Τότε Τύπωσε "Το Χ έχει τιμή 100" : Τύπωσε "Τέλος"
```

Μεταξύ του **Τότε** και του **Αλλιώς** έχουμε μια ακολουθία, και μετά το Αλλιώς μέχρι το τέλος γραμμής μια δεύτερη ακολουθία.

```
Αν Χ=100 Τότε Τύπωσε "Το Χ έχει τιμή 100" Αλλιώς Τύπωσε "Επαναφορά Τιμής" : Χ=100
```

***Πρόγραμμα 4**: Δομή Αν χωρίς αγκύλες σε μια γραμμή*

Παράδειγμα δομών επιλογής με χρήση μπλοκ κώδικα:

```
Αν Χ=100 Τότε {Τύπωσε "Το Χ έχει τιμή 100" : Τύπωσε "Τέλος" }
Αν Χ=100 Τότε {Τύπωσε "Το Χ έχει τιμή 100"} Αλλιώς {Τύπωσε "Επαναφορά Τιμής" : Χ=100}
```

***Πρόγραμμα 5**: Δομή Αν με αγκύλες σε μια γραμμή*

Τα μπλοκ κώδικα μπορούν να ανοίξουν με γραμμές, επίσης μπορούμε να βγάλουμε το διαχωριστικό εντολών (άνω και κάτω τελεία). Βάζουμε εσοχές με διαστήματα ή με Tab.

```
Αν Χ=100 Τότε {
    Τύπωσε "Το Χ έχει τιμή 100"
    Τύπωσε "Τέλος"
}
Αν Χ=100 Τότε {
    Τύπωσε "Το Χ έχει τιμή 100"
} Αλλιώς {
    Τύπωσε "Επαναφορά Τιμής"
    Χ=100
}
```

***Πρόγραμμα 6**: Δομή Αν με αγκύλες σε πολλές γραμμές*

Μπορούμε να επιλέξουμε στο διορθωτή κείμενο που θα βάλουμε ή θα βγάλουμε εσοχές (με TAB βάζουμε, με Shift TAB βγάζουμε. Μπορούμε με Ctrl+Enter να βάλουμε αγκύλες και αυτόματα γραμμή με εσοχή ανάμεσά τους.

Η Αλλιώς μπορεί να γραφεί στη θέση της Τότε σε μια Αν, αλλά τότε δεν θα γράψουμε άλλη Αλλιώς.

Στο παράδειγμα παρακάτω η Αλλιώς.Αν συντάσσεται με την Τότε ή την Αλλιώς. Αν συνταχθεί με την αλλιώς τότε είναι σαν Όχι (συνθήκη) Τότε. Με το τρόπο αυτό γλιτώνουμε ένα τελεστή. Μπορούμε να βάλουμε αγκύλες ή όχι. Αν δεν βάλουμε αγκύλες και ακολουθούμε τη σύνταξη όπου η δομή ξεχωρίζει από τις ακολουθίες, σε δικές της γραμμές τότε βάζουμε το Τέλος Αν στο τέλος.

```
Αν Χ=100 Τότε {
    Τύπωσε "Το Χ έχει τιμή 100"
} Αλλιώς.Αν Χ=10 Τότε {
    Τύπωσε "Το Χ έχει τιμή 10"
} Αλλιώς.Αν Χ=50 ή Χ<0 Αλλιώς {
    Τύπωσε "Το Χ δεν είναι 50 και δεν είναι μικρότερο του μηδέν"
} Αλλιώς.Αν Χ=-20 Τότε {
```

```
    Τύπωσε "Το Χ έχει τιμή -20"
} Αλλιώς {
    Τύπωσε "Επαναφορά Τιμής"
    Χ=100
}
```

***Πρόγραμμα 7**: Δομή Αν με Αλλιώς.Αν με αγκύλες σε πολλές γραμμές*

Με Τέλος Αν στο τέλος της δομής:

```
Αν Χ=100 Τότε
    Τύπωσε "Το Χ έχει τιμή 100"
Αλλιώς.Αν Χ=10 Τότε
    Τύπωσε "Το Χ έχει τιμή 10"
Αλλιώς.Αν Χ=50 ή Χ<0 Αλλιώς
    Τύπωσε "Το Χ δεν είναι 50 και δεν είναι μικρότερο του μηδέν"
Αλλιώς.Αν Χ=-20 Τότε
    Τύπωσε "Το Χ έχει τιμή -20"
Αλλιώς
    Τύπωσε "Επαναφορά Τιμής"
    Χ=100
Τέλος Αν
```

***Πρόγραμμα 8:** Δομή Αν με Αλλιώς με Τέλος Αν (χωρίς αγκύλες)*

Η επιλογή με αριθμητική έκφραση και πολλαπλές συνθήκες για την έκφραση αυτή. Σε αυτό το άρθρο δεν θα δούμε πώς ενώνονται δυο οι περισσότερες ακολουθίες.

```
Επίλεξε Με Χ
Με 100, 200
    Τύπωσε "Σε μια γραμμή1" : Τύπωσε "Σε μια γραμμή2"
Με 500, >1000
    Χ+=10
Με <50
    {     \\ χρήση μπλοκ για περισσότερες από μια γραμμές
        Χ++
    }
Με 70 έως 80
    Χ/=10
Αλλιώς Με
    Χ-=50
Τέλος Επιλογής
```

***Πρόγραμμα 9**: Δομή Επίλεξε Με Τέλος Επιλογής*

Η εντολή Επίλεξε γράφεται και ως Επέλεξε.

Η χρήση μπλοκ είναι χρήσιμη και όταν χρησιμοποιούμε τη Προς ή GOTO. Η εντολή Προς δουλεύει μέσα σε αγκύλες. Ο διερμηνευτής κοιτάει αν το άλμα είναι εντός του μπλοκ με αγκύλες,

και αν όχι πάει ένα μπλοκ πιο πάνω, με τελευταίο το μπλοκ του επώνυμου κώδικα. Άλματα εκτός επώνυμου κώδικα δεν μπορούν να γίνουν, και σε αυτήν την περίπτωση γίνεται τερματισμός του κώδικα. Η Διαμέσου μπορεί να εκτελεστεί από οπουδήποτε αρκεί ο κώδικας να υπάρχει στο χώρο του επώνυμου κώδικα, με συγκεκριμένη ετικέτα για είσοδο. Τα άλματα και οι κλήσεις ρουτινών με ετικέτες καταγράφονται την πρώτη φορά που ζητούνται για γρήγορη εκτέλεση σε νέα ζήτηση. Υπάρχει συνδυασμός της Προς με αριθμό και της δομής Αν, όπου το Προς μπορούμε να το παραλείψουμε. Αν όμως το άλμα είναι προς ετικέτα πρέπει να δοθεί η Προς με την ετικέτα.

```
Αν A=B Τότε 100 Αλλιώς 500
```

***Πρόγραμμα 10**: Δομή Αν με άλματα σε αριθμητικές ετικέτες*

Όταν χρησιμοποιούμε ετικέτες και αριθμούς τους κρατάμε στην αριστερή άκρη των γραμμών. Αυτή η ακολουθία ενώ έχει ανάμεσα αλλαγές ροής με άλματα, έχει μια αρχή και ένα τέλος. Δηλαδή εξωτερικά είναι μια τυπική ακολουθία.

```
        X=Τυχαίος(1, 4)
        Από X Προς 100, 200, άλφα, βήτα
άλφα:
100     Τύπωσε "Μονός Αριθμός"; X
        Προς 300
βήτα:   \\ μόνο σημειώσεις σε ετικέτες
200     Τύπωσε "Ζυγός Αριθμός"; X
300     Τύπωσε "Τέλος Σπαγγέτι"
```

***Πρόγραμμα 11**: Χρήση Εντολής Προς (Goto)*

Οι ετικέτες πρέπει να διατηρούν τα γράμματα (πεζά και κεφαλαία) και τους τόνους ως έχουν.

Δομή Επανάληψης

Υπάρχουν τέσσερις δομημένες δομές επανάληψης στη Μ2000, συν μια ειδική:

- Για καθορισμένο αριθμό επαναλήψεων, δομή: Για Επόμενο
- Για έλεγχο στην αρχή της ακολουθίας προς επανάληψη, δομή: Ενώ Τέλος Ενώ
- Για έλεγχο στο τέλος της ακολουθίας προς επανάληψη, δομή: Επανάλαβε Μέχρι
- Χωρίς έλεγχο επανάληψης, δομή: Επανάλαβε Πάντα (Επανέλαβε Πάντα)
- Χρήση Μπλοκ ως δομή επανάληψης

Η Επανάλαβε γράφεται και Επανέλαβε.

Παραδείγματα Δομών Επανάληψης

Η Για εκτελεί τουλάχιστον μια φορά την ενσωματωμένη ακολουθία. Αν η αρχική τιμή είναι μεγαλύτερη από την τελική τότε έχουμε μέτρηση προς τα κάτω. Εξ ορισμού το βήμα είναι ανά ένα, αλλά μπορούμε να το αλλάξουμε. Από το βήμα ο διερμηνευτής κρατάει μόνο την απόλυτη τιμή. Εξαίρεση αποτελεί η περίπτωση που η αρχική τιμή και η τελική είναι ίδια. Τότε αν δεν υπάρχει η Ανά μετά την ακολουθία δεν θα έχουμε αλλαγή τιμής στην μεταβλητή ελέγχου. Αν χρησιμοποιηθεί

η Ανά το πρόσημο καθορίζει αν θα είναι προς τα κάτω ή προς τα πάνω η τιμή της μεταβλητής ελέγχου. Μπορούμε να αλλάξουμε τη μεταβλητή ελέγχου στην επανάληψη, αλλά αυτή η τιμή δεν είναι η πραγματική τιμή ελέγχου αλλά αντίγραφό της, έτσι δεν εξαρτιέται από αυτήν την μεταβλητή η επανάληψη. Στην επόμενη επανάληψη της ακολουθίας η μεταβλητή ελέγχου θα πάρει νέα τιμή από τη πραγματική μεταβλητή ελέγχου (είναι κρυφή).

```
Για ι=1 έως 10 : Τύπωσε ι : Επόμενο ι

Για ι=1 έως 10 ανά 2
    Τύπωσε ι
Επόμενο ι


Για ι=1 έως 10 {Τύπωσε ι}


Για ι=1 έως 10 Ανά 2 {
    Τύπωσε ι
}
```

***Πρόγραμμα 12**: Δομή Επανάληψης Για σε διάφορες διαμορφώσεις*

Δομή επανάληψης Ενώ. Είτε χρησιμοποιήσουμε μπλοκ είτε όχι η δομή έχει μπλοκ.

```
X=10 : Ενώ X>0 : X-- : Τύπωσε X: Τέλος Ενώ

X=10
Ενώ X>0
    X--
    Τύπωσε X
Τέλος Ενώ


X=10
Ενώ X>0 { X-- : Τύπωσε X}


X=10
Ενώ X>0 {
    X--
    Τύπωσε X
}
```

***Πρόγραμμα 13**: Δομή Επανάληψης Ενώ σε διάφορες διαμορφώσεις*

Δομή επανάληψης Επανέλαβε. Είτε χρησιμοποιήσουμε μπλοκ είτε όχι η δομή έχει μπλοκ.

```
X=10 : Επανέλαβε : X--:Τύπωσε X: Μέχρι X=0

X=10
Επανέλαβε
```

```
    Χ--
    Τύπωσε Χ
Μέχρι Χ=0


Χ=10
Επανέλαβε { Χ-- : Τύπωσε Χ} Μέχρι Χ=0


Χ=10
Επανέλαβε {
    Χ--
    Τύπωσε Χ
} Μέχρι Χ=0
```

***Πρόγραμμα 14**: Δομή Επανάληψης Επανέλαβε Μέχρι σε διάφορες διαμορφώσεις*

Υπάρχει μια επανάληψη που δεν τερματίζει. Εκτός αν χρησιμοποιήσουμε δική μας έξοδο. Εδώ δεν έχουμε δομημένο προγραμματισμό.

```
Χ=10
Επανέλαβε
    Χ--
    Αν Χ<0 Τότε Έξοδος
    Τύπωσε Χ
Πάντα


Χ=10
Επανέλαβε {
    Χ--
    Αν Χ<0 Τότε Έξοδος
    Τύπωσε Χ
} Πάντα
```

***Πρόγραμμα 15**: Δομή Επανάληψης Επανέλαβε Πάντα σε διάφορες διαμορφώσεις*

Το μπλοκ ως ειδική δομή επανάληψης: Το μπλοκ κώδικα γίνεται δομή επανάληψης. Σε κάθε μπλοκ κώδικα, οποιασδήποτε δομής, ακόμα και απλό μπλοκ έχει μια κρυφή σημαία (flag) που μπορεί να έχει μια από δυο θέσεις, την άνω και την κάτω. Κάθε φορά που ξεκινάει ένα μπλοκ η σημαία είναι σε θέση κάτω και όταν ένα μπλοκ τερματίζει κανονικά τότε ο διερμηνευτής ελέγχει τη θέση της σημαίας, και αν την βρει στην άνω θέση τότε επαναλαμβάνει το μπλοκ. Η εντολή Κυκλικά (Loop) βάζει τη σημαία στην άνω θέση (ή την αφήνει στην άνω θέση αν ήδη είναι). Αυτή η εντολή μπορεί να μπει οπουδήποτε μέσα σε ένα μπλοκ, ακόμα και σε μια γραμμή με την Αν Τότε ή Αν Αλλιώς. Αν δεν δώσουμε την εντολή αυτή σε μια από τις επαναλήψεις του μπλοκ, τότε το μπλοκ θα τερματίσει, κανονικά στο τέλος του. Εδώ έχουμε δομημένο προγραμματισμό γιατί η αρχή και το τέλος της επανάληψης είναι πάντα προκαθορισμένα. Έχουμε την ευκολία να επιλέξουμε το χειρισμό της σημαίας, οποτεδήποτε.

```
\\ Όπως η Επανέλαβε Πάντα
X=10
{
      X--
      Αν X<=0 Αλλιώς Κυκλικά
      Τύπωσε X
}
\\ Όπως η Επανέλαβε Μέχρι
X=10
{
      X--
      Τύπωσε X
      Αν X>0 Τότε Κυκλικά
}
\\ Όπως η Ενώ
X=10
{
      Αν X>1 Τότε Κυκλικά
      X--
      Τύπωσε X
}
```

***Πρόγραμμα 16**: Διαμόρφωση μπλοκ κώδικα σε διάφορες δομές επανάληψης*

Παράδειγμα εξομοίωσης της For της BASIC

Στο παράδειγμα έχουν χρησιμοποιηθεί τα Αρχή, Τέλος και Βήμα τα οποία αποτελούν εντολές της γλώσσας, αλλά μπορούν να χρησιμοποιηθούν ως μεταβλητές, το παράδειγμα έχει όλο μπλε αναγνωριστικά γιατί ο χρωματιστής του διορθωτή της M2000 τα αναγνωρίζει ως γνωστά. Επίσης χρησιμοποιούμε τη Σύγκρινε() η οποία παίρνει δυο μεταβλητές, ή δυο στοιχεία από πίνακες, είτε για αριθμούς είτε για αλφαριθμητικά (δεν παίρνει αριθμητικές εκφράσεις). Αυτή η συνάρτηση δίνει -1, 0 ή 1. Υπάρχει ο τελεστής <=> που χρησιμοποιείται με αριθμητικές εκφράσεις ή αλφαριθμητικές εκφράσεις και δίνει το ίδιο αποτέλεσμα. Το 1<=>2 δίνει -1 δηλαδή το αριστερό είναι μικρότερο. Στο 2<=>1 δίνει 1, δηλαδή το δεξιό είναι μικρότερο. Η συνάρτηση Σημ() δίνει -1, 0 ή 1, όταν έχουμε αρνητικό, μηδέν, ή θετικό (είναι το SGN() της BASIC).

Αν το βήμα είναι αρνητικό, Σημ(Βήμα)=-1 τότε θέλουμε το Τέλος να είναι ίσο ή μικρότερο με την Αρχή δηλαδή 0 ή 1, άρα δεν θέλουμε το -1

Αν το βήμα είναι θετικό, Σημ(Βήμα)=1 τότε θέλουμε η Αρχή να είναι ίση ή μικρότερη με το Τέλος δηλαδή 0 ή -1, άρα δεν θέλουμε το 1.

Μόλις ξεκινήσει το μπλοκ της Αν, αναλαμβάνουν οι δυο τελευταίες εντολές. Η μία προσθέτει το βήμα, είτε είναι θετικό είτε αρνητικό είτε μηδέν. Κάνουμε πάλι την ίδια σύγκριση και αν δεν είναι εντάξει, δηλαδή αν γυρίσει η κατάσταση της σύγκρισης, θα τερματίσει το μπλοκ. Αν όμως συνεχίζεται η αρχική κατάσταση, τότε κοιτάμε αν το βήμα είναι 0 και αν δεν είναι βάζουμε τη σημαία επανάληψης στην άνω θέση και έτσι το μπλοκ επαναλαμβάνει.

Με μηδέν βήμα έχουμε μια εκτέλεση του μπλοκ και έξοδο

```
Αρχή=1 : Τέλος=10 : Βήμα=1
Αν Σύγκρινε(Αρχή,Τέλος)<>Σημ(Βήμα) Τότε {
    Τύπωσε Αρχή
    Αρχή+=Βήμα
    Αν Σύγκρινε(Αρχή,Τέλος)<>Σημ(Βήμα) Τότε Αν Βήμα<>0 Τότε Κυκλικά
}
Αρχή=10 : Τέλος=1 : Βήμα=-1
Αν Σύγκρινε(Αρχή,Τέλος)<>Σημ(Βήμα) Τότε {
    Τύπωσε Αρχή
    Αρχή+=Βήμα
    Αν Σύγκρινε(Αρχή,Τέλος)<>Σημ(Βήμα) Τότε Αν Βήμα<>0 Τότε Κυκλικά
}
```

***Πρόγραμμα 17**: Εξομοίωση της For (Για) της BASIC*

Στοιχεία μη δομημένου προγραμματισμού για επαναλήψεις

Υπάρχουν τέσσερις εντολές, **Έξοδος**, **Διέκοψε**, **Συνέχισε**, **Ξεκίνα** για μη δομημένο προγραμματισμό που σχετίζονται με τις επαναλήψεις. Στην επανάληψη Για Επόμενο δεν υπάρχει μπλοκ εντολών όπως στην Για {}. Μπορούμε να κάνουμε έξοδο από αυτό το Για με την Έξοδος Για. Σε όλες τις άλλες επαναλήψεις αρκεί η **Έξοδος** (Exit). Αν θέλουμε να φύγουμε από όλα τα φωλιασμένα μπλοκ, τότε χρησιμοποιούμε τη **Διέκοψε** (Break)

Υπάρχει το μπλοκ Δες { } το οποίο τερματίζει κανονικά ή όταν συμβεί λάθος χωρίς να διακοπεί το πρόγραμμα έξω από την Δες {} ή Try {}. Αυτή η εντολή σταματάει και τη διέκοψε. Στο παράδειγμα, οι επαναλήψεις με τη Για με μπλοκ, σταματάνε με τη Διέκοψε. Αν δεν είχαμε βάλει την Διέκοψε, ή αν χρησιμοποιούσαμε την Δες Μεταβλητή { } μια πιο ελαφριά έκδοση της Δες τότε η διέκοψε θα διέκοπτε μέχρι και τον επώνυμο κώδικα που εκτελούμε αυτό το πρόγραμμα. Θα μπορούσαμε να διακόψουμε τις επαναλήψεις με ένα άλμα έξω από τα δυο μπλοκ, με την Προς.

```
Δες {
    για ι=1 έως 10 {
        για κ=1 έως 10 {
            Αν ι>2 και κ> 2 Τότε Διέκοψε
        }
    }
}
Τύπωσε ι, κ
```

***Πρόγραμμα 18**: Χρήση της Διέκοψε (Break) εντός μια Δες (Try)*

Μια άλλη εντολή που βολεύει είναι η Συνέχισε. Αυτή η εντολή κάνει μια δομή να τερματίζει χωρίς να κάνει έξοδο από το μπλοκ, αν το μπλοκ ανήκει σε μια από τις τέσσερις δομές επανάληψης. Στα απλά μπλοκ, αν η σημαία είναι σε άνω θέση τότε θα γίνει επανάληψη άμεση αλλιώς γίνεται έξοδος. Το παράδειγμα θα μας τυπώσει τα 1 2 8 9 10

```
για ι=1 έως 10
    αν ι>=3 και ι<=7 τότε συνέχισε
    \\ τυπώνουμε ανά στήλη, Δες το κόμμα μετά το ι
    τύπωσε ι,
επόμενο ι
τύπωσε
```

***Πρόγραμμα 19**: Χρήση της Συνέχισε (Continue)*

Τέλος η εντολή Ξεκίνα κάνει ένα μπλοκ να ξεκινάει άμεσα, χωρίς έλεγχο της σημαίας, και στην εκκίνηση η σημαία θα πάρει τη κάτω θέση αν δεν ήταν στη κάτω θέση. Το παρακάτω πρόγραμμα δίνει ότι και το προηγούμενο:

```
ι=0
{
    ι++
    αν ι>=3 και ι<=7 τότε ξεκίνα
    \\ τυπώνουμε ανά στήλη, Δες το κόμμα μετά το ι
    τύπωσε ι,
    αν ι<10 τότε κυκλικά
}
τύπωσε
```

***Πρόγραμμα 20**: Χρήση της Ξεκίνα (Restart) σε μπλοκ κώδικα*

Ειδικές Δομές Ακολουθίας

Η Μ2000 πέρα από τις βασικές δομές ακολουθίας, επιλογής και επανάληψης έχει και άλλες δομές που σχηματίζονται με λέξεις και αγκύλες. Όταν εκτελείται μια εντολή της Μ2000 έστω σε ένα σημείο στο κώδικα, υπάρχει ένα αντικείμενο, το λεγόμενο αντικείμενο εκτέλεσης το οποίο κρατάει κάποια βασικά στοιχεία. Ένα από αυτά είναι ο τρέχον σωρός τιμών. Ένα άλλο είναι το τρέχον αντικείμενο εισόδου/εξόδου. Η χρήση των αντικειμένων εκτέλεσης δίνει την δυνατότητα να τρέχουν νήματα. Κάθε νήμα έχει δικό του αντικείμενο εκτέλεσης, και μπορεί να ασχολείται με διαφορετικό αντικείμενο εισόδου/εξόδου (δηλαδή ένα διαφορετικό επίπεδο από αυτό της Οθόνης της κονσόλας της Μ2000, ή το επίπεδο μιας φόρμας - ενός παραθύρου του προγραμματιζόμενου γραφικού περιβάλλοντος). Εκτός όμως από αυτά τα δυο στοιχεία που αναφέρθηκαν, κρατάει αρκετά άλλα, σχετικά με την εκτέλεση, και για τον έλεγχο των λαθών.

Έλεγχος Λαθών

Είδαμε ήδη τη δομή Δες { }. Ο έλεγχος λαθών έχει δυο όψεις, η μια όψη είναι να αφαιρέσουμε λάθη, και ίσως να τα αντιμετωπίσουμε, και η άλλη να δώσουμε εσκεμμένα λάθη, γιατί ο κώδικας βρίσκει μια κατάσταση που προβλέπεται να γυρίζει λάθος. Ένα λάθος που δεν ελέγχεται οδηγεί στο τερματισμό εκτέλεσης. Σε κάποιες περιπτώσεις ο τερματισμός δεν γίνεται στο κεντρικό πρόγραμμα αλλά σε κάποιο επιμέρους όπως στην εξυπηρέτηση γεγονότος, και τότε δεν μεταφέρεται στο κύριο πρόγραμμα, και πιθανόν να εμφανιστεί στο τερματισμό ενός προγράμματος, επειδή υπάρχει μια εσωτερική λίστα που κρατούνται τα σφάλματα, μέχρι να αναγνωριστεί ένα λάθος οπότε και διαγράφονται από αυτήν. Το Δες {} κρατάει μια Διέκοψε όπως έχουμε δει, είναι η ισχυρότερη δομή

για σφάλματα, δεν εμφανίζει στη Λάθος τιμή, μόνο το μήνυμα στη Λάθος$, όπου μπορούμε να δούμε το λάθος, Η Λάθος$ αν διαβαστεί σβήνει το περιεχόμενό της, για το επόμενο λάθος όταν παρουσιαστεί.

```
Δες {
    Τύπωσε 10/0
}
Λ$=Λάθος$
\\ δίνει Πρόβλημα: διαίρεση με το μηδέν
Αν Λ$<>"" Τότε Τύπωσε "Πρόβλημα:";Λ$
```

***Πρόγραμμα 21**: Χρήση της δομής Δες (Try)*

Το πιο ελαφρύ Δες Μεταβλητή {} δίνει Αληθές στη μεταβλητή αν το πρόβλημα είναι μέσα στο μπλοκ και όχι σε κάτι που καλούμε μέσα από το μπλοκ. Στη Λάθος και την Λάθος$ θα μπουν τιμές.

```
Δες Οκ {
    Τύπωσε 10/0
}
\\ δίνει Πρόβλημα: διαίρεση με το μηδέν
Αν Όχι Οκ Τότε Τύπωσε "Πρόβλημα:";Λάθος$
```

***Πρόγραμμα 22**: Χρήση της παραλλαγής της δομής Δες με μεταβλητή*

Αν θέλουμε μπορούμε να προκαλέσουμε λάθος, με την Λάθος ως εντολή.

```
Δες Οκ {
    Λάθος " νέο λάθος"
}
\\ δίνει Πρόβλημα: νέο λάθος
Αν Λάθος ή Όχι Οκ Τότε Τύπωσε "Πρόβλημα:";Λάθος$

Δες Οκ {
    Λάθος " νέο λάθος - 5005"
}
\\ δίνει Πρόβλημα: νέο λάθος
Κ=Λάθος
Αν Κ ή Όχι Οκ Τότε Τύπωσε "Πρόβλημα:";Λάθος$, Κ=5005
```

***Πρόγραμμα 23**: Χρήση της εντολής Λάθος*

Γιατί να θέλουμε να προκαλέσουμε λάθος; Μπορεί σε μια δομή ακολουθίας να φτάσουμε σε ένα σημείο όπου δεν υπάρχει κάτι που μπορούμε να κάνουμε, είναι μια κατάσταση που δεν έχει λογική για να συνεχιστεί, οπότε διακόπτουμε με την Λάθος.

Υπάρχουν συναρτήσεις που ελέγχουν το λάθος άμεσα, μια από αυτές είναι και εκφράσεις, η Έγκυρο(). Στο παράδειγμα έχουμε διαίρεση με το μηδέν:

```
X=10 : Z=0 : Τύπωσε Έγκυρο(X/Z)=Ψευδές
```

***Πρόγραμμα 24**: Χρήση της συνάρτησης Έγκυρο()*

Άλλα Είδη Δομών

Υπάρχουν αρκετές άλλες δομές που χωρίζονται σε είδη που θα αναφερθούν στο άρθρο αυτό μόνο επιγραμματικά. Αυτές οι δομές είναι μόνο με μπλοκ με αγκύλες:

- Δομές σε σχέση με το χρόνο, δομή: Κάθε
- Δομές σε σχέση με νήματα, δομές: Κύριο.Έργο, Μετά και Δομή Νήμα
- Δομή συγχρονισμού, δομή: Μέρος
- Δομές σε σχέση με αντικείμενα, δομές: Σωρός και Για (με αντικείμενο)
- Δομές σε σχέση με γραφικά, δομή Πάχος, Δομή Χρώμα, Δομή Πένα, Δομή Επίπεδο, Δομή Περιθώριο. Αυτές οι δομές είτε επιλέγουν κάποιο επίπεδο για την εκτέλεση των εντολών προς επίπεδα, και των γραφικών, είτε καθορίζουν για όσο το μπλοκ είναι ενεργό ένα χαρακτηριστικό, όπως το χρώμα και τύπο φόντου ή το χρώμα πένας ή το πάχος και είδος της γραμμής.

Επώνυμος Κώδικας

Ο επώνυμος κώδικας έχει την ιδιότητα να καλείται με όνομα. Η κλήση μπορεί ή όχι να επιστρέφει τιμή, και κατά τη κλήση μπορούμε να δίνουμε ορίσματα, δηλαδή τιμές.

Εξ ορισμού οι κλήσεις με ορίσματα γίνονται με πέρασμα με τιμή, ενώ υπάρχει και το πέρασμα με αναφορά, όπου στέλνουμε μια ισχνή αναφορά και αυτό που καλούμε κάνει την ισχνή μια πραγματική αναφορά συνδέοντας σε ένα όνομα τον χώρο καταχώρησης του ονόματος που δώσαμε με αναφορά. Για παράδειγμα αν καλέσουμε ένα τμήμα Αλφα με ορίσματα 100, 200 δηλαδή Αλφα 100, 200 τότε οι δυο τιμές θα μπουν σε ένα σωρό τιμών και θα διαβαστούν με μια Διάβασε σε δυο μεταβλητές. Στη Μ2000 αν ζητάμε πέρασμα με αναφορά δεν γίνεται να μην δώσουμε μια ισχνή αναφορά. Αν στο τμήμα Βήτα δίνουμε το &Α, Β ως ορίσματα, τότε στο σωρό τιμών μπαίνει η ισχνή αναφορά της Α και η τιμή του Β. Στη πλευρά του Βήτα (στο κώδικά του) ακολουθεί η Διάβασε έστω Διάβασε &ΑΑ, ΒΒ όπου το ΑΑ θα γίνει αναφορά στο Α και το ΒΒ θα πάρει τη τιμή του Β. Αν αλλάξει τιμή το ΑΑ θα αλλάξει και το Α (αυτό σημαίνει με αναφορά), ενώ αν αλλάξει τιμή του ΒΒ δεν θα αλλάξει το Β.

Είδη Ορισμών

Υπάρχουν τα παρακάτω μπλοκ ορισμών:

- Μπλοκ Τμήματος - Είναι ένα επώνυμο μπλοκ ακολουθίας
- Μπλοκ Συνάρτησης - Είναι ένα επώνυμο μπλοκ ακολουθίας
- Μπλοκ Λάμδα Συνάρτησης - (σχεδόν ίδιο με το μπλοκ συνάρτησης)
- Μπλοκ Ομάδας (έχει δικό του διερμηνευτή)
- Μπλοκ Κλάσης (σχεδόν ίδιο με το μπλοκ ομάδας)

Όλα τα παραπάνω μπορεί να είναι Γενικά ή Τοπικά ή Μέλος Ομάδας. Η κλάση περιέχει ορισμό Ομάδας, και καλείται σαν γενική συνάρτηση, όταν δεν ανήκει σε ομάδα ή ως μέλος ομάδας αν ανήκει σε ομάδα. Μια ομάδα μπορεί να περιέχει οποιουσδήποτε ορισμούς με μπλοκ καθώς και δικά της ξεχωριστά μπλοκ, που θα δούμε στον αντικειμενοστραφή προγραμματισμό.

Ο επώνυμος κώδικας μπορεί να φορτωθεί κατά την εκτέλεση κώδικα. Τα τμήματα και οι συναρτήσεις με μπλοκ είναι σαν κλειστά κουτιά. Στη Μ2000 έχουν προτεραιότητα τα τοπικά ονόματα, και αν ένα όνομα υπάρχει ως γενικό και ως τοπικό, τότε το τοπικό σκιάζει το γενικό. Επειδή κάθε τμήμα και συνάρτηση μπορούν να έχουν δικά τους τμήματα και συναρτήσεις, το καθένα αποτελεί ένα πρόγραμμα από μόνο του. Η διαφορά με μια ομάδα που και αυτή μπορεί να έχει τμήματα και συναρτήσεις είναι ότι στην ομάδα διατηρούμε τιμές, δηλαδή λέμε ότι η ομάδα έχει μια κατάσταση. Αυτό σημαίνει ότι κάθε φορά που καλούμε μεθόδους της ομάδας, η συμπεριφορά της εξαρτιέται από τις τιμές που έχουν καταχωρηθεί.

Σε κάποιες γλώσσες όπως και στη Μ2000 υπάρχει η δυνατότητα στις συναρτήσεις και στα τμήματα (στις διαδικασίες σε άλλες γλώσσες) να ορίσουμε στατικές μεταβλητές. Με αυτή την έννοια θα έλεγε κανείς ότι μοιάζουν με τις ομάδες, αλλά οι ομάδες έχουν διάφορες μεθόδους, ενώ ένα τμήμα είναι μια μοναδική μέθοδος, αν σκεφτούμε ότι μπορεί να διατηρεί κατάσταση σε στατικές μεταβλητές. Δεν μπορούμε για την ίδια κατάσταση να καλέσουμε άλλη μέθοδο.

Υπάρχουν ορισμοί χωρίς μπλοκ (είναι πάντα τοπικοί):

- Στατική Ρουτίνα (Ρουτίνα Τέλος Ρουτίνας)
  Χρήση Έξοδος Ρουτίνας, κλήση είτε με τη Διαμέσου ή άμεσα με το όνομα.
  Το όνομα έχει πάντα παρενθέσεις ακόμα και αν δεν πάρει παραμέτρους.
- Στατική Συνάρτηση (Συνάρτηση Τέλος Συνάρτησης)
  Χρήση Έξοδος Συνάρτησης, κλήση μόνο σε έκφραση με @ πριν το όνομα.
  Συναρτήσεις και των δυο βασικών τύπων, αριθμητικού και αλφαριθμητικού

Ορισμούς χωρίς μπλοκ έχουν μόνο οι ορισμοί με μπλοκ: Τμήμα, Συνάρτηση, Λάμδα και έμμεσα η Ομάδα και η Κλάση σε μέλη Τμήματα, Συναρτήσεις, Λάμδα συναρτήσεις.

Θέαση & Ζωή Ορισμών Επώνυμου Κώδικα

Όλοι οι ορισμοί είναι δυναμικοί, δηλαδή ισχύουν μέχρι να διαγραφεί ο χώρος που τα έφτιαξε. Επίσης εκτός από τους στατικούς ορισμούς όλοι οι άλλοι μπορούν να αλλαχθούν ή να επιλεγούν πριν χρησιμοποιηθούν από εναλλακτικούς ορισμούς.

Οι στατικοί ορισμοί έχουν θέαση στο χώρο που υπάρχουν. Εδώ αυτό που λέμε κατάσταση στην Ομάδα βάσει μελών, και κατάσταση σε Τμήματα και Συναρτήσεις βάσει στατικών μεταβλητών, συμβαίνει να υπάρχει με τη χρήση των μεταβλητών του τμήματος ή της συνάρτησης που ανήκουν.

Στο παράδειγμα η ΤελικήΤιμή είναι θεατή στη ρουτίνα Παραγοντικό(), σε κάθε εκτέλεσή της (έχουμε κλήση του εαυτού της, αναδρομή). Το τμήμα είναι ένα **κλειστό κουτί**. Δέχεται μια παράμετρο και επιτελεί μια εργασία. Στο παράδειγμα καλούμε το τμήμα 28 φορές. Η ρουτίνα δεν είναι κλειστό κουτί, γιατί έχει θέαση στο τμήμα.

```
Τμήμα ΕμφάνισεΠαραγοντικό (Ν){
    Ν=Ακ(Ν)
    Αν Ν<0 τότε Λάθος "Όχι αρνητικός αριθμός για παραγοντικό"
    \\ ο Αριθμός είναι ο Decimal (27 ψηφία)
    Κάνε ΤελικήΤιμή ως Αριθμός=1
    Παραγοντικό(Ν)
    Τύπωσε Μορφή$("Παραγοντικό {0}!={1}", Ν, ΤελικήΤιμή)
    Τέλος
    Ρουτίνα Παραγοντικό(Χ)
        Αν Χ<1 Τότε Έξοδος Ρουτίνας
        ΤελικήΤιμή*=Χ
        Παραγοντικό(Χ-1)
    Τέλος Ρουτίνας
}
Για Ι=0 έως 27
    ΕμφάνισεΠαραγοντικό ι
Επόμενο
```

***Πρόγραμμα 25**: Παραγωγή Παραγοντικού με χρήση Ρουτίνας και Αναδρομή*

Θα γράψουμε το προηγούμενο παράδειγμα με μια κανονική συνάρτηση για παραγοντικό:

```
Τμήμα ΕμφάνισεΠαραγοντικό (Ν){
    \\ ο αριθμός 1@ είναι σταθερά τύπου Decimal (27 ψηφία)
    Συνάρτηση Παραγοντικό(Χ) {
        Αν Χ<1 Τότε =1@ : Έξοδος
        =Χ*Παραγοντικό(Χ-1)
    }
    Ν=Ακ(Ν)
    Αν Ν<0 τότε Λάθος "Όχι αρνητικός αριθμός για παραγοντικό"
    Τύπωσε Μορφή$("Παραγοντικό {0}!={1}", Ν, Παραγοντικό(Ν))
}
Για Ι=0 έως 27
    ΕμφάνισεΠαραγοντικό ι
Επόμενο
```

***Πρόγραμμα 26**: Παραγωγή Παραγοντικού με Συνάρτηση και Αναδρομή*

Οι ρουτίνες αν δεν έχουν μπλοκ κατά τη διαδοχή των κλήσεων έχουν μεγάλο αριθμό κλήσεων αναδρομής. Μπορούμε να ορίσουμε όριο με την Όριο.Αναδρομής ακόμα και εκατομμύρια κλήσεις. Οι συναρτήσεις έχουν ένα μέγιστο περίπου 5450 κλήσεων. Βεβαίως εδώ στο παραγοντικό δεν μπορούμε να ανέβουμε πολύ και να διατηρήσουμε την ακρίβεια μέχρι και τις μονάδες. Αν βάλουμε στις επαναλήψεις αριθμό μεγαλύτερο από 27 τότε ο αριθμός τύπου Decimal θα βγάλει λάθος υπερχείλισης.

Τα παραδείγματα δόθηκαν για να αντιληφθεί ο αναγνώστης τη θέαση ονομάτων, το τι λέμε "κλειστό κουτί".

Οι απλές συναρτήσεις δεν έχουν την δυνατότητα των ρουτινών για μεγάλη αναδρομή γιατί καλούνται μέσα σε εκτελεστή εκφράσεων. Το ίδιο πρόγραμμα με απλή (στατική) συνάρτηση. Για να κληθεί χρειάζεται το @ πριν από το όνομα (έτσι ξεχωρίζει ο διερμηνευτής τι συνάρτηση καλούμε).

```
Τμήμα ΕμφάνισεΠαραγοντικό (Ν){
    \\ ο αριθμός 1@ είναι σταθερά τύπου Decimal (27 ψηφία)
    Ν=Ακ(Ν)
    Αν Ν<0 τότε Λάθος "Όχι αρνητικός αριθμός για παραγοντικό"
    \\ οι στατικές απλές συναρτήσεις καλούνται με το @ στην αρχή του ονόματος
    Τύπωσε Μορφή$("Παραγοντικό {0}!={1}", Ν, @Παραγοντικό(Ν))
    Συνάρτηση Παραγοντικό(Χ)
        Αν Χ<1 Τότε =1@ : Έξοδος Συνάρτησης
        =Χ*@Παραγοντικό(Χ-1)
    Τέλος Συνάρτησης
}
Για Ι=0 έως 27
    ΕμφάνισεΠαραγοντικό ι
Επόμενο
```

***Πρόγραμμα 27**: Παραγωγή Παραγοντικού με απλή Συνάρτηση και αναδρομή*

Πολίτες Πρώτης Τάξης

Δυο από αυτούς τους ορισμούς, η λάμδα συνάρτηση και η Ομάδα λέγονται πολίτες πρώτης τάξης, διότι έχουν την δυνατότητα να μεταφέρονται, και με το τρόπο αυτό να μην διαγράφονται όταν διαγραφεί ο χώρος που τα δημιούργησε. Οι πολίτες πρώτης τάξης μπορούν να επιστραφούν από κώδικα, και να περαστούν ως αντίγραφα ή με αναφορά, σε κλήσεις κώδικα.

Παράδειγμα Επιστροφής Ομάδας

```
Συνάρτηση  Αντικείμενο_Αλφα (Κ){
    Ομάδα Άλφα {
        Χ=10*Κ, Υ=20*Κ, Ζ=30*Κ
    }
    =Άλφα
}
Α1=Αντικείμενο_Αλφα(2)
Α2=Αντικείμενο_Αλφα(5)
Τύπωσε Α1.Χ=20, Α2.Χ=50
```

***Πρόγραμμα 28**: Επιστροφή Αντικειμένου από Συνάρτηση*

Οι λάμδα συναρτήσεις είναι μεταβλητές και συναρτήσεις ταυτόχρονα. Έχουν δυνατότητα να κρατάνε τιμές στα λεγόμενα κλεισίματα, τα οποία ορίζουμε έξω από το μπλοκ κώδικα και ισχύουν για αυτόν τον κώδικα. Μπορούμε να εξομοιώσουμε την λάμδα συνάρτηση με μια ομάδα, σχεδόν σε όλες τις λειτουργίες.

Παράδειγμα Επιστροφής Λάμδα Συνάρτησης

Στο παράδειγμα η Κάπα είναι μια λάμδα που γυρνάει μια άλλη λάμδα. Όταν την καλούμε δίνουμε μια παράμετρο Χ. Αυτή γίνεται κλείσιμο στην επιστρεφόμενη λάμδα. Έτσι η Λάμδα στη Ζ3 έχει το 3**Υ ενώ η Ζ4 έχει το 5**Υ, όπου το ** (ή ^) είναι η ύψωση σε δύναμη. Αν δώσουμε το Ζ3=Ζ5 τότε η Ζ3 θα πάρει το αντίγραφο της Ζ5.

```
Κάπα=Λάμδα (Χ) ->{
    =Λάμδα Χ (Υ)-> {
        =Χ**Υ
    }
}
Ζ3=Κάπα(3)
Ζ5=Κάπα(5)
Τύπωσε Ζ3(3)=27, Ζ5(2)=25
```

***Πρόγραμμα 29**: Επιστροφή Λάμδα Συνάρτησης από Συνάρτηση*

Καταχώρηση Τιμών

Κάθε γλώσσα έχει τρόπο να καταχωρούμε πληροφορίες. Είδαμε ότι η Μ2000 έχει δυο βασικούς τύπους εκφράσεων, την αριθμητική και την αλφαριθμητική (γράμματα). Ομοίως έχουμε την καταγραφή τιμών από αυτές τις εκφράσεις σε ονόματα, ώστε αργότερα τις τιμές να τις χρησιμοποιήσουμε ξανά.

Οι καταχωρήσεις τιμών έχουν τα παρακάτω χαρακτηριστικά:

- Θέαση: Γενική, Τοπική, Μέλος (Αντικειμένου)
- Έχουν Όνομα ή έχουν Θέση: Τιμή σε θέση είναι η τιμή σε θέση πίνακα.
- Σταθερή ή Μεταβλητή: Οι σταθερές δέχονται μια φορά τιμή.
- Στατική ή Δυναμική καταχώρηση: Στη δυναμική καταχώρηση οι καταχωρήσεις διαγράφονται με τη διαγραφή του χώρου που ορίστηκε η καταχώρηση.

Παραδείγματα Τιμών Γενικών, Τοπικών, Μελών Ομάδας

Το επόμενο παράδειγμα δείχνει τη διαφορά Γενικής, Τοπικής και Μέλους Ομάδας. Δείχνει πότε το <= χρησιμοποιείται για εκχώρηση για αποφυγή του = το οποίο δημιουργεί τοπικές όταν δεν υπάρχουν τοπικές με το ίδιο όνομα, ανεξάρτητα αν υπάρχουν γενικές (θα τις σκιάσει όπως θα δούμε στο παράδειγμα)

```
Γενική Χ=22222
Τμήμα ΚλειστόΚουτί {
    Γενική Χ=10
    Χ<=40
    Τύπωσε Χ=40
    \\ Τώρα δημιουργούμε την τοπική Χ, και σκιάζουμε την γενική Χ
    Χ=30
```

```
    Τύπωσε Χ=30
    Ομάδα Άλφα {
        Χ=10
        Τμήμα Εμφάνισε {
            \\ το πρώτο Χ είναι η γενική
            \\ το δεύτερο είναι το Αυτό.Χ (προαιρετικά βάζουμε το Αυτό)
            Τύπωσε Χ=40, .Χ=10, Αυτό.Χ=10
            \\ αν δώσουμε το Χ=5000 θα φτιάξουμε τοπική Χ
            .Χ<=5000
        }
    }
    Τύπωσε Άλφα.Χ=10
    Άλφα.Εμφάνισε
    Τύπωσε Άλφα.Χ=5000
}
ΚλειστόΚουτί
\\ τώρα δεν υπάρχει καμία μεταβλητή, μόνο η Χ
Τύπωσε Χ=22222
```

***Πρόγραμμα 30**: Χρήση Μεταβλητών, Τοπικών, Γενικών και Μελών Ομάδων*

Παράδειγμα Τιμών Πολιτών Πρώτης Τάξης

Στο επόμενο παράδειγμα θα δούμε δυο πρώτης τάξης πολίτες με όνομα και σε θέση. Τα ονόματα δεν αλλάζουν τύπο όταν πάρουν την πρώτη τιμή, ή οριστούν απευθείας με τύπο. Οι θέσεις πίνακα όμως αλλάζουν τύπο. Ο νέος τύπος διώχνει τον παλιό τύπο. Εξαίρεση υπάρχει όταν στη θέση πίνακα υπάρχει αντικείμενο που ορίζει πώς θα αλλαχθεί η τιμή του (προχωρημένο θέμα).

Φτιάχνουμε μια ομάδα **Αλφα** με τρία μέλη (εδώ είναι δημόσια εξ ορισμού), το Χ, το Λ και τη συνάρτηση Κ(). Το μέλος Λ επειδή είναι λάμδα συνάρτηση έχει και το Λ() ως όνομα συνάρτησης.

Η ομάδα Άλφα έχει όνομα, δηλαδή δημιουργήθηκε σε ένα μέρος το οποίο με το πέρας εκτέλεσής του θα διαγραφεί. Δείχνουμε πρώτα ότι μπορούμε να δημιουργήσουμε μια Λ1 από την Αλφα.Λ, και αυτή είναι μια επώνυμη Λάμδα όπως και η Αλφα.Λ και θα διαγραφούν την ίδια στιγμή, στο πέρας εκτέλεσης του κώδικα. Μετά φτιάχνουμε ένα πίνακα και στη θέση 0 βάζουμε ένα αντίγραφο της Αλφα και στην θέση 1 ένα αντίγραφο της Αλφα.Λ και έτσι έχουμε πολίτες πρώτης τάξης σε θέσεις και όχι σε ονόματα. Αν ο κώδικας ανήκει σε μια συνάρτηση τότε μια εντολή =Α() θα δώσει ως επιστροφή τον πίνακα Α(), και με αυτόν τον τρόπο οι πολίτες πρώτης τάξης θα επιστραφούν από το χώρο που δημιουργήθηκαν. Δείτε ότι η συνάρτηση Κ() χρησιμοποιεί το Χ μέλος της Αλφα. Το Χ είναι η κατάσταση της Άλφα. Δείτε ότι στο Α(2) μπορούμε να καταχωρήσουμε ένα αντίγραφο με την τρέχουσα κατάσταση, και αργότερα να το αντιγράψουμε στο Α(0) επαναφέροντας την κατάσταση σε πρότερο βήμα. Δείτε επίσης ότι μπορούμε να χειριστούμε την ομάδα και την λάμδα στο πίνακα απευθείας δίνοντας την θέση που μας ενδιαφέρει.

```
    Ομάδα Αλφα {
        Χ=3
        Λ=Λάμδα (Χ)->Χ**2
```

```
    Συνάρτηση Κ (Ν) {
        =.Χ**Ν
        .Χ++
    }
}
Λ1=Αλφα.Λ
\\ σώνουμε την Άλφα.Λ λάμδα σε μια νέα Λ1
Τύπωσε Λ1(5)=25
\\ Δημιουργούμε ένα πίνακα μιας διάστασης με θέσεις από 0 έως 4
\\ με αρχική τιμή 100. Τα στοιχεία πίνακα μπορούν να αλλάξουν τύπο τιμής.
Πίνακας Α(5)=100
\\ στη θέση 0 βάζουμε αντίγραφο της Ομάδας Αλφα
Α(0)=Αλφα
\\ στη θέση Α(1) βάζουμε αντίγραφο της Λάμδα Λ
Α(1)=Αλφα.Λ
Τύπωσε Αλφα.Λ(3)=9, Αλφα.Χ=3
Τύπωσε Αλφα.Κ(3)=27, Αλφα.Χ=4
Τύπωσε Αλφα.Κ(3)=64, Αλφα.Χ=5
\\ Χρήση θέσεων 0 και 1
Τύπωσε Α(0).Χ=3, Α(1)(4)=16
Τύπωσε Α(0).Λ(3)=9, Α(0).Κ(3)=27, Α(0).Χ=4
\\ Σώνουμε το Α(0) στο Α(2)
Α(2)=Α(0)
Τύπωσε  Α(0).Κ(3)=64, Α(0).Χ=5
\\ Επιστρέφουμε τη τιμή του Α(2) στο Α(0)
Α(0)=Α(2)
Τύπωσε  Α(0).Κ(3)=64, Α(0).Χ=5
```

***Πρόγραμμα 31**: Χρήση Πολιτών Πρώτης Τάξης (Λάμδα Συνάρτησης & Ομάδας)*

Παράδειγμα Σταθερών-Απαριθμήσεων-Τελικών Μελών Ομάδων

Παρακάτω θα δούμε ένα άλλο παράδειγμα με σταθερές, δηλαδή ονόματα που παίρνουν μια φορά τιμή. Θα δούμε και μια Λ που ορίζεται ως σταθερή (μια Ομάδα δεν μπορεί να οριστεί ως σταθερή, αλλά μπορούν να οριστούν μέλη του ως Τελικά). Φαίνεται και η Απαρίθμηση, ένας τύπος που φτιάχνεται με σταθερές.

```
Σταθερή Εκατό=100
Απαρίθμηση Κατοικίδιο {Γάτα, Σκύλος, Χελώνα}
\\ η Μ πήρε τύπο Κατοικίδιο
Μ=Γάτα
Μ++
Τύπωσε Εκφρ$(Μ)="Σκύλος", Μ=2, Μ=Σκύλος
Τύπωσε Τύπος$(Μ)="Κατοικίδιο"
Σταθερή Λ=Λάμδα (Χ, Υ)->Χ^Υ
Τύπωσε Λ(2,3)=8
```

```
Δες {
    \\ δεν μπορεί να αλλάξει είναι σταθερή
    Λ=Λάμδα ->5000
}
Τύπωσε Λ(2,3)=8
\\ Στις ομάδες λέγεται Τελική και όχι Σταθερή
Ομάδα Άλφα {
    Τελική Μ=100
}
Τύπωσε Αλφα.Μ
Δες {
    \\ η σταθερή δεν έχει τελεστές ++ και --, καθώς και -=, +=, *=, /=
    Αλφα.Μ++
}
Τύπωσε Αλφα.Μ
```

***Πρόγραμμα 32**: Χρήση Σταθερών, Απαριθμήσεων και τελικών Μελών Ομάδων*

Παράδειγμα Καταχώρηση Στατικών Τιμών

Στο παρακάτω παράδειγμα θα χρησιμοποιήσουμε στατική μεταβλητή σε τμήμα. Στατικές και τοπικές δεν μπορούν να έχουν ίδιο όνομα. Αν το δει κάτι τέτοιο ο διερμηνευτής θα βγάλει λάθος.

```
Τμήμα Επόμενος_Αριθμός {
    Στατική Μ=1
    Τύπωσε Μ
    Μ++
}
Για κ=2 έως 5
    Επόμενος_Αριθμός
Επόμενο
```

***Πρόγραμμα 33**: Χρήση Στατικών Μεταβλητών*

Αν αυτό το πρόγραμμα το έχουμε σε ένα τμήμα έστω Α στη κονσόλα της Μ2000, τότε δείτε τι γίνεται. Την πρώτη φορά που θα το τρέξουμε από τη κονσόλα θα μας δείξει από το 1 έως το 4. Την επόμενη φορά που θα το τρέξουμε θα δείξει από το 5 έως το 8. Αν βάλουμε την εντολή Καθαρό στο τέλος του τμήματος θα καθαρίσουν οι μεταβλητές (θα γίνονταν έτσι και αλλιώς επειδή θα τελείωνε η εκτέλεση του τμήματος) και επίσης θα καθαρίσουν και οι στατικές (αυτό δεν γίνεται αυτόματα, γίνεται με την Καθαρό όταν το επιλέγουμε). Αν ξεκινήσουμε ξανά την Α θα δούμε τα 1 2 3 4.

Αν η Α είναι στη κονσόλα και καλείται με άμεση κλήση σημαίνει ότι είναι γενικό τμήμα. Αν φτιάξουμε ένα Β και σε αυτό καλέσουμε το Α τότε για την Β το Α θα κρατάει διαφορετικό σύνολο στατικών. Η σχέση Στατικών και τμήματος Α έχει να κάνει με το από που καλούνται. Αυτό συμβαίνει γιατί η καταχώρηση γίνεται σε αυτό που καλεί (και συνεχίζει προς την αρχή των κλήσεων). Οι στατικές χρησιμοποιούνται σε νήματα, μέρη τμημάτων που τρέχουν παράλληλα.

Μαζικοί Καταχωρητές Τιμών

Για την ευκολία στο προγραμματισμό υπάρχουν έτοιμα αντικείμενα για μαζική καταχώρηση τιμών.

- Πίνακες, τα στοιχεία γράφονται σε θέσεις.
  Οι πίνακες είναι δυναμικοί, μπορούμε να αλλάζουμε μέγεθος χωρίς να σβήνουμε στοιχεία (εκτός αν δεν γίνεται, πχ να μειώσουμε το μέγεθος).
  Δυο διεπαφές, με όνομα με παρενθέσεις (Α()) και με δείκτη πχ Α.
- Καταστάσεις, τα στοιχεία γράφονται ως κλειδί ή με ζεύγη κλειδί και τιμή.
  Εύρεση, Εγγραφή, Διαγραφή σε Ο(1). Χρήση πάντα με δείκτη.
- Σωροί τιμών. Τα στοιχεία έχουν σειρά. Μπορούν εύκολα να μετακινηθούν και να πολλαπλασιαστούν. Στους σωρούς βάζουμε ή βγάζουμε στοιχεία, ή τα διαβάζουμε, αλλά δεν αλλάζουμε τιμές ενώ είναι στο σωρό. Χρήση πάντα με δείκτη.

Σε όλους του καταχωρητές μπορούμε να βάλουμε δείκτη οποιουδήποτε άλλου, να βάλουμε απλές τιμές, να βάλουμε αντικείμενα τύπου Ομάδας, να βάλουμε Λάμδα συναρτήσεις.

Παράδειγμα Πίνακα

Ένας από τους μαζικούς καταχωρητές είναι ο πίνακας που έχουμε δει. Ο πίνακας έχει δυο διεπαφές. Τη διεπαφή με ονόματα με παρενθέσεις. Όταν ο πίνακας είναι σε αριστερή έκφραση τότε αντιγράφει τη δομή και τα στοιχεία του πίνακα, όπως στην εντολή Α()=Β():

```
Πίνακας Α(), Β(5)
Β(0):=1,2,3,4,5
Α()=Β()
Α(0)+=100
Α(4)+=100
Τύπωσε Α(0)=101, Β(0)=1
Τύπωσε Α(4)=105, Β(4)=5
```

***Πρόγραμμα 34**: Χρήση Πίνακα - Εκχώρηση (αντιγραφή) Πίνακα σε Πίνακα*

Παράδειγμα Αυτόματου Πίνακα (tuple)

Το tuple ή αυτόματος πίνακας, δίνει ένα δείκτη στο αντικείμενο. Το **(,)** είναι ο κενός πίνακας, το **(1,)** είναι πίνακας με ένα στοιχείο, το 1. Οι αυτόματοι πίνακες έχουν βάση το 0 (το πρώτο στοιχείο είναι στη θέση 0).

```
Α=(1,2,3,4,5)
Τύπωσε Μήκος(Α)=5, Α#Μεγ()=5, Α#Μικ()=1, Α#Τιμή(2)=3, Α#Αθρ()=15
```

***Πρόγραμμα 35**: Χρήση Αυτόματου Πίνακα (Tuple)*

Με χρήση λάμδα συναρτήσεων μπορούμε να εφαρμόσουμε φίλτρα και πακετάρισμα:

```
α=(1,2,3,4,5,6,7,8,9)
πακ1=λάμδα ->{
    βάλε αριθμός+αριθμός
}
```

```
ζυγός=λάμδα (χ)->χ υπόλοιπο 2=0
β=α#Φίλτρο(ζυγός, (,))
Τύπωσε β ' 2 4 6 8
Τύπωσε α#Φίλτρο(ζυγός)#Πακ(πακ1)=20
```

***Πρόγραμμα 36**: Αυτόματοι Πίνακες με Φίλτρα και Πακετάρισμα με Λάμδα Συναρτήσεις*

Η τελευταία γραμμή εφαρμόζει το φίλτρο ζυγός και στον πίνακα που δίνει εφαρμόζουμε το πακετάρισμα. Θα μπορούσε να γραφτεί με την #Αθρ() γιατί κάνει ακριβώς το ίδιο βγάζει το άθροισμα των στοιχείων:

```
Τύπωσε α#Φίλτρο(ζυγός)#Αθρ()=20
```

***Πρόγραμμα 37**: Χρήση της #Αθρ() για εξαγωγή αθροίσματος από Αυτόματο Πίνακα*

Παράδειγμα Κατάστασης Τιμών

Μια **κατάσταση** έχει ζεύγη κλειδιών και τιμών. Παρακάτω έχουμε ένα παράδειγμα που θα μπει σε ένα τμήμα. Το παράδειγμα διαμορφώνει την κονσόλα με 60 χαρακτήρες επί 32 γραμμές, δίνει χρώμα φόντου ματζέντα (από τα 0 έως 15 που είναι τα καθορισμένα χρώματα των Windows, μπορούμε να ορίσουμε το ματζέντα με το #FF00FF όπως στην Html), ορίζει το χρώμα πένας το κίτρινο (νούμερο 14). Μπορούμε να ορίσουμε πλάγια γράμματα, και να ορίσουμε προσωρινά την πένα με μια δομή πένας.

Οι ρουτίνες εδώ βλέπουν στοιχεία του τμήματος, όπως το Αρ_Αναφοράς, και την κατάσταση Αυτοκίνητο. Σε κάθε τιμή της κατάστασης βάζουμε κλειδί ένα αλφαριθμητικό και τιμή έναν αυτόματο πίνακα με δυο στοιχεία. Η συνάρτηση Ημέρα() παίρνει ημερομηνία ως αλφαριθμητικό και την μετατρέπει σε αριθμό. Η συνάρτηση Ημέρα$() κάνει το ανάποδο, παίρνει αριθμό και εμφανίζει την ημερομηνία που αντιστοιχεί στον αριθμό. Στην ρουτίνα ΕμφάνσεΌλα() φτιάχνουμε τοπικές, τη Σύνολο και τη Μ. Η Μ είναι ένα αντικείμενο Επαναλήπτης, ο οποίος κρατάει μια σύνδεση με το Αυτοκίνητο και διέρχεται από όλα τα στοιχεία του. Την αλλαγή τιμής του την κάνει η δομή Ενώ η οποία βλέπει ότι έχει αντικείμενο και ενεργεί διαφορετικά από την γνωστή συμπεριφορά της. Το Μ^ δίνει τον αριθμό στοιχείου (εδώ έχει βάση το 0, οπότε προσθέτουμε ένα για να φαίνεται ότι το πρώτο είναι το 1). Επίσης στην Εκφρ$(Μ!) η συνάρτηση Εκφρ$() βλέπει ότι έχει Κατάσταση (το Μ έχει το Αυτοκίνητο εντός του), και τραβάει το κλειδί (το δηλώνει το θαυμαστικό). Η ρουτίνα Εμφάνισε() δουλεύει για κάθε κλειδί, ακόμα και αυτά που δεν υπάρχουν. Έτσι θα κάνει έλεγχο αν το κλειδί υπάρχει. Για το πρόγραμμα αυτό ο έλεγχος δεν χρειάζεται αφού τα κλειδιά τα διαβάζουμε από την κατάσταση. Αν σε ένα κλειδί είχαμε αλφαριθμητικό θα το παίρναμε με το Αυτοκίνητο$(Κλειδι$), δηλαδή το όνομα της κατάστασης θα είχε το $ στο τέλος. Η κατάσταση φαίνεται σαν πίνακας, αλλά τα κλειδιά της είναι είτε αριθμοί είτε αλφαριθμητικά, και σε κάθε περίπτωση εσωτερικά είναι αλφαριθμητικά.

```
Φόρμα 60, 32
Οθόνη 5
Πένα 14
Πλάγια 1
```

```
Πένα 12 {Τύπωσε "Παράδειγμα με Κατάσταση  Τιμών και Πίνακες ως τιμές"}
Πλάγια 0
Αρ_Αναφοράς=1
Απαρίθμηση Είδος {Αξία=0, Άδεια}
Κατάσταση Αυτοκίνητο = "ZH12343":=(1500, Ημέρα("12-3-1998") ), "MA23033":=(2300,
Ημέρα("2-5-2002"))
Προσθήκη Αυτοκίνητο, "ZZH23451":=(5000, Ημέρα("26-8-2008"))
Ταξινόμηση Αυτοκίνητο
ΕμφάνισεΌλα()
Αφαίρεση Αυτοκίνητο, "ZZH23451"
Ταξινόμηση Αυτοκίνητο
ΕμφάνισεΌλα()
Τέλος
Ρουτίνα ΕμφάνισεΌλα()
    Τοπική Σύνολο=Μήκος(Αυτοκίνητο)
    Τοπική Μ=Κάθε(Αυτοκίνητο)
    Πένα 11 {Τύπωσε "Αναφορά:";Αρ_Αναφοράς}
    Αρ_Αναφοράς++
    Ενώ Μ
        Πένα 15 { Τύπωσε Μορφή$("Καταχώρηση: {0}/{1}",Μ^+1, Σύνολο) }
        Εμφάνισε(Εκφρ$(Μ!))
    Τέλος Ενώ
Τέλος Ρουτίνας
Ρουτίνα Εμφάνισε(Ποιο$)
    Αν Υπάρχει(Αυτοκίνητο, Ποιο$) Τοτε
        Τύπωσε "Στοιχεία: "; Ποιο$
        Τύπωσε "Τιμή Πώλησης: ";Αυτοκίνητο(Ποιο$)(Αξία)
        Τύπωσε "Ημ/νια πρώτης άδειας: ";Ημέρα$(Αυτοκίνητο(Ποιο$)(Άδεια))
    Αλλιώς
        Τύπωσε "Δεν υπάρχει το αυτοκίνητο με πινακίδα: ";Ποιο$
    Τέλος Αν
Τέλος Ρουτίνας
```

***Πρόγραμμα 38**: Καταχώρηση, Ταξινόμηση και Αναζήτηση Αυτοκινήτων σε Κατάσταση Ειδών*

Το πρόγραμμα αυτό θέλει μια επιπλέον ανάλυση. Στο κύριο μέρος του προγράμματος φτιάχνουμε αυτό που ορίζεται ως κατάσταση προγράμματος (state). Αυτό αποτελείται από στοιχεία του προγράμματος που κρατάνε δεδομένα. Πραγματικά τα δεδομένα εδώ είναι οι τρεις πίνακες και τα τρία κλειδιά, ένα για κάθε πίνακα. Όλα αυτά μαζί μπαίνουν σε ένα όνομα, μια κατάσταση τιμών. Η κατάσταση (state) του προγράμματος μεταβάφεται. Εδώ αφαιρούμε ένα στοιχείο, και καλούμε πάλι την ΕμφάνισεΌλα. Αυτό που δεν φαίνεται με μια πρώτη ματιά είναι ότι το πρόγραμμα έχει δυο επίπεδα πληροφορίας, όπου για το καθένα κρατάει διαφορετική κατάσταση προγράμματος. Το ένα επίπεδο είναι αυτό που δηλώνεται με τον καταχωρητή μαζικών τιμών, την Κατάσταση. Το άλλο επίπεδο είναι η αναφορά κατάστασης, η οποία διαχωρίζεται από όποια άλλη βάσει ενός αριθμού αναφοράς. Η μεταβλητή Αρ_Αναφοράς είναι στοιχείο του δεύτερου επίπεδου, και εμφανίζεται σαν

επικεφαλίδα στην αρχή της κάθε αναφοράς, καθώς επίσης αυξάνεται κατά ένα ώστε να δείχνει διαφορετικό αριθμό σε κάθε αναφορά.

Με λίγη προσοχή παραπάνω βλέπουμε ότι το πρόγραμμα φέρει στοιχεία, δεν τα παίρνει από κάπου αλλού, όπως ένα αρχείο. Το πρόγραμμα δηλαδή από μόνο του είναι σαν ένα αντικείμενο το οποίο έχει μια κατάσταση και καλεί μεθόδους που την αλλάζουν. Θα μπορούσαμε να βάζαμε και άλλες λειτουργίες, αλλά σε κάθε περίπτωση η αρχική κατάσταση είναι προδιαγεγραμμένη. Μια ιδέα θα ήταν να διατηρούσε κάποιος ένα αρχείο, όχι με απλές γραμμές, αλλά με κώδικα, δηλαδή να έμπαινε στο κώδικα προγράμματος και να αφαιρούσε κωδικούς ή να πρόσθετε κωδικούς και όταν έτρεχε το αρχείο να ετοιμαζόταν η κατάσταση τιμών, και να οδηγούσε σε μια εισαγωγή κλειδιού (την πινακίδα του αυτοκινήτου) για να δείξει τα στοιχεία αν υπάρχει το κλειδί.

Παράδειγμα με Σωρό Τιμών

Ένα απλό σχέδιο, χωρίς ταξινόμηση, αλλά με διατήρηση της σειράς των στοιχείων μπορεί να γίνει με χρήση σωρού τιμών. Για να το κάνουμε πιο ενδιαφέρον θα κάνουμε εκτύπωση σε εκτυπωτή (επιλέξτε εκτύπωση σε pdf).

```
\\ επιλογή εκτυπωτή
Εκτυπωτής ?
Τύπωσε "Επιλεγμένος Εκτυπωτής: "; Εκτυπωτής$
\\ επιλογή ιδιοτήτων εκτύπωσης
Εκτυπωτης !

Πάρκινγκ_Α=Σωρός
Σωρός Πάρκινγκ_Α {
    Σειρά "ZH12343", 1500, "12-3-1998"
    Σειρά "MA23033", 2300, "2-5-2002"
    Σειρά"ZZH23451", 5000, "26-8-2008"
}
Αντίγραφο_Α=Σωρός(Πάρκινγκ_Α)
Σωρός Αντίγραφο_Α {
    Εκτύπωση Ναι
    \\ στον εκτυπωτή γράφουμε με μαύρο χρώμα!
    Πένα 0
    Αναφορά 2, "Λίστα Αυτοκινήτων στο Πάρκινγκ Α"
    Τύπωσε '
    Ενώ όχι κενό {
        Διάβασε Πινακίδα$, Τιμή_Πώλησης, Ημ_Αδειας$
        Τύπωσε "Στοιχεία: "; Πινακίδα$
        Τύπωσε "Τιμή Πώλησης: ";Τιμή_Πώλησης
        Τύπωσε "Ημ/νια πρώτης άδειας: ";Ημ_Αδειας$
    }
    Εκτύπωση 'Οχι
}
```

```
Τύπωσε Μήκος(Πάρκινγκ_Α)=9
Τύπωσε Μήκος(Αντίγραφο_Α)=0
```

***Πρόγραμμα 39**: Καταχώρηση Αυτοκινήτων σε Σωρό και Εκτύπωση Σωρού σε Εκτυπωτή*

Οι σωροί γράφουν στοιχεία σε διάταξη, χωρίς κλειδιά. Μπορούμε να βάζουμε με τη Βάλε ή με τη Σειρά. Η Σειρά τα βάζει όπως τα βλέπουμε, στο τέλος του σωρού, έτσι το τελευταίο στοιχείο, το "26-8-2008" θα μπει τελευταίο. Εδώ η χρήση λέγεται FIFO, διότι το πρώτο που διαβάζουμε είναι και το πρώτο που μπήκε στο σωρό (First In First Out, ή το πρώτο που μπαίνει βγαίνει πρώτο). Από το σωρό σηκώνουμε στοιχεία με τη Διάβασε (το ίδιο κάνει κάθε συνάρτηση και τμήμα με χρήση του τρέχοντος σωρού τιμών). Με την δομή Σωρός Αντικείμενο_Σωρός {} αλλάζουμε τον τρέχον σωρό με το Αντικείμενο_Σωρός. Στο παράδειγμα χρησιμοποιήσαμε το Αντίγραφο_Α το οποίο δημιουργήσαμε με τη Σωρός(Πάρκινγκ_Α) για να δείξουμε στο τέλος ότι αυτό άδειασε από στοιχεία. Η Κενό δίνει αληθής αν ο σωρός δεν έχει στοιχεία, οπότε με την Ενώ όχι κενό εκτελούμε την δομή επανάληψης για όλα τα στοιχεία.

Η Τύπωσε εμφανίζει τα στοιχεία στο τρέχον επίπεδο εξόδου, αρχικά στη κονσόλα της Μ2000, αλλά εδώ με την Εκτυπωτής Ναι αλλάξαμε την έξοδο στο φύλλο του εκτυπωτή. Η εκτύπωση θα γίνει μόλις δώσουμε το Εκτυπωτής Όχι ή μια εντολή για αλλαγή σελίδας. Αν γράφουμε συνέχεια με την τύπωσε, θα γίνει αυτόματα αλλαγή σελίδας. Στο φύλλο του εκτυπωτή δεν έχουμε ολίσθηση γραμμών όπως στη κονσόλα αλλά αλλαγή σελίδας. Το φύλλο του εκτυπωτή το χρησιμοποιούμε όπως και τη κονσόλα αλλά στο μέρος της εξόδους (όχι για εισαγωγή). Έτσι μπορούμε να κάνουμε σχέδια και να εμφανίσουμε εικόνες.

Περισσότερα παραδείγματα με σωρούς και καταστάσεις δεν θα μπουν σε αυτό το άρθρο, υπάρχουν στο μικρό εγχειρίδιο της Μ2000, το οποίο περιέχεται στο αρχείο εγκατάστασης της γλώσσας και φαίνεται στο μενού Μ2000 στα προγράμματα στο μενού Έναρξη, των Windows. Το αρχείο είναι pdf και επειδή περιέχει το κανονικό αρχείο κειμένου μπορεί να ανοίξει στο LibreOffice έκδοση 6. Από εκεί είναι καλύτερο να αντιγράψει κανείς τα παραδείγματα παρά από το pdf από το πρόγραμμα ανάγνωσης του pdf.

Τι είναι ένα αντικείμενο;

Ένα αντικείμενο είναι μια προγραμματιστική οντότητα, το οποίο έχει μια καθορισμένη συμπεριφορά, που ορίζεται από τη διεπαφή του, ένα σύνολο δημόσιων ιδιοτήτων και μεθόδων, και σχετίζεται με την τρέχουσα κατάστασή του η οποία ορίζεται εσωτερικά και είναι ιδιωτική. Δυο ίδια αντικείμενα, σημαίνει ότι είναι δυο αντικείμενα με ίδια κατάσταση και ίδια συμπεριφορά. Έτσι σαν παρουσίες είναι δυο διαφορετικά στιγμιότυπα ενός αφηρημένου αντικειμένου, μιας ιδέας δηλαδή. Όταν έχουμε ίδια συμπεριφορά τότε λέμε ότι έχουμε ίδιο τύπο. Ο τύπος αντικειμένου είναι μια λεκτική ονομασία που προσδιορίζει το αφηρημένο αντικείμενο. Η αφαίρεση συνίσταται στην επιλεκτική χρήση μέρους των ιδιοτήτων και της συμπεριφοράς ενός φυσικού αντικειμένου.

Το αφηρημένο αντικείμενο

Κατά το σχεδιασμό, στο νου μας έχουμε το φυσικό αντικείμενο και το αφηρημένο αντικείμενο που στηρίζεται στο φυσικό και είναι ένας ορισμός που θέλουμε να δηλώνει το φυσικό αντικείμενο. Για

να δουλέψουν όμως τα αντικείμενα πρέπει να υλοποιηθούν σε προγραμματιστικά αντικείμενα. Η υλοποίηση χρειάζεται δυο πράγματα: Να έχουμε ένα ορισμό αντικειμένου και έναν μηχανισμό που θα μας δώσει σε ένα όνομα ένα προγραμματιστικό αντικείμενο.

Το προγραμματιστικό αντικείμενο είναι η παρουσία του αφηρημένου αντικειμένου και κατ΄ επέκταση δηλώνει το φυσικό φυσικό.

Ιδιότητες και Μέθοδοι Αντικειμένων

Τα αντικείμενα έχουν μέλη, ιδιότητες, και μεθόδους, όπου στις παρουσίες τους αντιστοιχούν σε τιμές και σε εκτελέσιμες ενέργειες. Θα αναφέρουμε τα αντικείμενα εδώ ως παρουσίες, τα προγραμματιστικά αντικείμενα.

Ένα αντικείμενο μπορεί να έχει άλλα αντικείμενα μέσα του, είτε είναι μέρος της παρουσίας του, είτε είναι δείκτης σε άλλο αντικείμενο. Για να γίνει πιο καθαρό ένα εσωτερικό αντικείμενο είτε είναι μέλος όπως όλα τα άλλα μέλη, οπότε δεν υπάρχει δείκτης σε αυτό, είτε είναι μέλος ένας δείκτης σε ένα αντικείμενο που σημαίνει ότι και άλλα αντικείμενα μπορούν να δείχνουν το ίδιο αντικείμενο ή άλλοι δείκτες σε άλλες θέσεις να δείχνουν το ίδιο αντικείμενο. Επίσης το είδος του αντικειμένου που δείχνει ο δείκτης μπορεί να διαφοροποιείται. Κατά μια έννοια όταν δεν έχουμε δείκτη στο εσωτερικό αντικείμενο είναι προκαθορισμένο, και αυτό που μπορεί να αλλάξει είναι η κατάστασή του, αλλά όχι ο τύπος του.

Είδη Αντικειμένων στην Μ2000

Στη Μ2000 τα αντικείμενα είναι τριών ειδών:

- Εσωτερικά αντικείμενα που ορίζονται με τιμές άμεσα.
  Πίνακας, Αυτόματος Πίνακας (Tuple), Σωρός, Κατάσταση (αυτοί λέγονται και μαζικοί καταχωρητές), Διάρθρωση Μνήμης (χειρισμός μνήμης), Γεγονός (με μεταβλητή λίστα συναρτήσεων), Έγγραφο, Λάμδα συναρτήσεις.

- Εσωτερικά και εξωτερικά αντικείμενα.
  Τα αντικείμενα αυτά ορίζονται με την Όρισε (Declare), και είναι τύπου COM (ένα μοντέλο προγραμματισμού αντικειμένων). Αυτά τα αντικείμενα είναι είτε βιβλιοθήκες μεθόδων είτε πραγματικά αντικείμενα όπως φόρμες χρήστη (GUI), ή εξωτερικό όπως το Word, ή το Sapi (για χειρισμό ομιλίας).

- Αντικείμενα Χρήστη, οι Ομάδες.

Αυτό που μας ενδιαφέρει σε αυτό το άρθρο είναι η χρήση της Ομάδας, γιατί σε αυτό έχουμε τη δυνατότητα να εφαρμόζουμε ορισμούς, να κατασκευάζουμε δικά μας αντικείμενα.

Παράδειγμα χρήσης εξωτερικού αντικειμένου

Στο παράδειγμα που ακολουθεί χρησιμοποιούμε το SAPI και ειδικότερα το SpVoice αντικείμενο, με το οποίο θα κάνουμε ένα κείμενο να ακουστεί ενώ ταυτόχρονα θα εμφανίζονται οι λέξεις καθώς θα τις ακούμε. Για να πετύχει αυτό πρέπει το αντικείμενο να το ανοίξουμε με χρήση γεγονότων. Εκτός από τον καθορισμό της χρήσης ΜεΓεγονότα στην όρισε το συγκεκριμένο αντικείμενο απαιτεί να πάρει μια τιμή η οποία ανά bit ορίζει ποια γεγονότα θα στέλνει. Ζητάμε τρία γεγονότα και για κάθε

ένα έχουμε τρεις συναρτήσεις εξυπηρέτησης γεγονότων. Αυτές οι συναρτήσεις καλούνται όπως οι ρουτίνες, με θέαση στο τμήμα. Για το λόγο αυτό όπου έχουμε νέες μεταβλητές βάζουμε το Τοπική και στην Διάβασε ακόμα βάζουμε το Νέο ώστε σίγουρα να διαβαστούν σε νέες μεταβλητές. Όσες φτιάχνουμε θα διαγραφούν μόλις εξυπηρετηθεί το γεγονός, στο πέρας εκτέλεσης της συνάρτησης.

Στα αντικείμενα αυτά καλούμε μεθόδους με την Μέθοδος. Αν επιστρέφουν τιμή τότε βάζουμε στο τέλος Ως Όνομα (αν επιστρέφουν αλφαριθμητικό βάζουμε Όνομα$). Αν θέλουμε να προσπελάσουμε ιδιότητες χρησιμοποιούμε την Με ΌνομαΑντικειμένου. Δείτε ότι υπάρχει και η Με στην Επίλεξε Με. Εκεί δεν μπερδεύεται ο Διερμηνευτής γιατί περιμένει την Με της Επίλεξε σε συγκεκριμένη θέση, επειδή μεταξύ δυο Με μπορεί να υπάρχει ή μια γραμμή εντολών ή ένα μπλοκ εντολών.

Στο πρόγραμμα εμφανίζουμε τις λέξεις κάνοντας και έναν έλεγχο αν χωράει στην γραμμή και αν όχι τότε αλλάζουμε γραμμή.

```
Τμήμα ΧρήσηΓεγονότων {
    Όρισε ΜεΓεγονότα sp "SAPI.SpVoice"
    Αυτό$={Rosetta Code is a programming chrestomathy site.
    The idea is to present solutions to the same task in as many different languages as
possible, to demonstrate how languages are similar and different, and to aid a person with a
grounding in one approach to a problem in learning another. Rosetta Code currently has 913
tasks, 214 draft tasks, and is aware of 707 languages, though we do not (and cannot) have
solutions to every task in every language.}
    ΤέλοςΡεύματος=False
    ΤελικήΘέση=-1
    ΠλάτοςΚειμένου=0
    Συνάρτηση sp_Word {
        Διάβασε Νέο ΑριθμόςΡεύματος, ΘέσηΡεύματος, ΘέσηΧαρακτήρα, ΜήκοςΛέξης
        Αν ΤελικήΘέση=ΘέσηΧαρακτήρα Τότε Έξοδος
        ΤελικήΘέση=ΘέσηΧαρακτήρα
        Τοπική φ$=" "
        Αν ΠλάτοςΚειμένου=ΘέσηΧαρακτήρα+ΜήκοςΛέξης Τότε φ$=". "
        Αν ΜήκοςΛέξης+θέση+2>Πλάτος Τότε Τύπωσε
        Τύπωσε Mid$(Αυτό$, ΘέσηΧαρακτήρα+1, ΜήκοςΛέξης);φ$;
        Αν φ$=". " Τότε Τύπωσε
        Ανανέωση
    }
    Συνάρτηση sp_EndStream {
        Ανανέωση
        ΤέλοςΡεύματος=Αληθής
    }
    Συνάρτηση sp_Sentence {
        Διάβασε Νέο ΑριθμόςΡεύματος, ΘέσηΡεύματος, ΘέσηΧαρακτήρα, ΜήκοςΛέξης
        Αν ΜήκοςΛέξης>0 και όχι ΘέσηΧαρακτήρα=0 Τότε Τύπωσε
        Τύπωσε "  ";
```

```
            ΠλάτοςΚειμένου=ΘέσηΧαρακτήρα+ΜήκοςΛέξης-1
        }
        Σταθερή SVEEndInputStream = 4
        Σταθερή SVEWordBoundary = 32
        Σταθερή SVESentenceBoundary = 128
        Σταθερή SVSFlagsAsync = 1&

        Με sp, "EventInterests", SVEWordBoundary + SVEEndInputStream +
SVESentenceBoundary
        Μέθοδος sp, "Speak", Αυτό$, SVSFlagsAsync
        Ενώ Όχι ΤέλοςΡεύματος {Αναμονή 10}
        Αναμονή 100
}
ΠαλιάΈνταση=Ένταση
Ένταση 100
ΧρήσηΓεγονότων
Ένταση ΠαλιάΈνταση
```

***Πρόγραμμα 40**: Χρήση Εξωτερικού Αντικειμένου με Γεγονότα*

Η Ομάδα (Αντικείμενο) στη M2000

Η M2000 δεν γράφτηκε στις πρώτες εκδόσεις με σκοπό να έχει αντικείμενα χρήστη. Μόνο τα αντικείμενα COM ήταν στο σχεδιασμό της, για να γίνεται ο λεγόμενος αυτοματισμός γραφείου, να επικοινωνεί ο διερμηνευτής με το Word ή το Excel και παρόμοια προγράμματα τα οποία είναι αντικείμενα του συστήματος όταν είναι εγκατεστημένα.

Σε αυτό το άρθρο θα δούμε πως διαμορφώθηκε η ιδέα της Ομάδας στη M2000 ως αντικείμενο, το λεγόμενο και ως αντικείμενο χρήστη, με την έννοια ότι σχεδιάζεται από τον προγραμματιστή. Η λέξη ομάδα (group στην αγγλική ορολογία της M2000), δηλώνει αυτό που περιέχει στοιχεία. Έχουμε δει ότι υπάρχουν έτοιμα αντικείμενα, όπως η κατάσταση που είναι σαν μια ομάδα στοιχείων από ζεύγη κλειδιών και τιμών. Η Ομάδα ως ιδέα μπήκε στη γλώσσα πριν που μπουν οι μαζικοί καταχωρητές. Θα δούμε παρακάτω τα βήματα της εξέλιξης της ομάδας. Η εξέλιξη βασίστηκε στο τι θέλουμε να πετύχουμε με τη Ομάδα, και για το λόγο αυτό θα αναφέρεται ο σκοπός, ή ιδέα, πίσω από το κάθε βήμα εξέλιξης.

Πέρασμα Ομάδας με τιμή και με αναφορά

Η πρώτη ιδέα για την Ομάδα ήταν να περιέχει μια ομάδα μεταβλητών με σκοπό να περνάει η ομάδα με τιμή ή με αναφορά σε μια κλήση, και έτσι να μεταφέρουμε ομάδα μεταβλητών. Στο παράδειγμα που ακολουθεί καλούμε το ΚάνεΚάτι με πέρασμα με τιμή, και καλούμε το ΚάνεΚάτι2 με πέρασμα με αναφορά, της ομάδας Αλφα. Όλα τα μέλη της Αλφα είναι εξ ορισμού δημόσια. Δεν έχουμε δώσει κάποιο τύπο στην Αλφα. Η Τύπος$(Αλφα) γυρίζει πάντα "Group" όταν έχουμε ομάδα. Θα δούμε αργότερα πως χρησιμοποιούνται οι τύποι. Εδώ η ομάδα Αλφα θα διαγραφεί στο τέλος εκτέλεσης του τμήματος που περιέχει το κώδικα.

```
Ομάδα Αλφα {
    Χ=10, Υ=30, α$="Όνομα"
    Πίνακας Α(1 έως 100)=1
}
Τμήμα ΚάνεΚάτι (Κ) {
    Τύπωσε Κ.α$
    Για ι=1 έως 100
        Κ.Α(ι)=ι
    Επόμενο
    Τύπωσε Κ.Α()#Αθρ()=5050
}
ΚάνεΚάτι Αλφα
Τύπωσε Αλφα.Α()#Αθρ()=100
Τμήμα ΚάνεΚάτι2 (&Κ) {
    Τύπωσε Κ.α$
    Για ι=1 έως 100
        Κ.Α(ι)=ι
    Επόμενο
    Τύπωσε Κ.Α()#Αθρ()=5050
}
ΚάνεΚάτι2 &Αλφα
Τύπωσε Αλφα.Α()#Αθρ()=5050
```

Χρήση μεθόδων - Πέρασμα με αναφορά μεθόδου ομάδας

Η επόμενη ιδέα ήταν να μπορεί η ομάδα να έχει μεθόδους, δηλαδή τμήματα και συναρτήσεις και να υπάρχει ένας τρόπος αυτά τα τμήματα να βλέπουν τα άλλα μέλη. Κανονικά όπως έχουμε ήδη δει, ένα τμήμα βλέπει ό,τι ορίζει ή κάτι γενικό. Εδώ έχουμε ένα βαθμό θέασης ακόμα τα μέλη του αντικειμένου που ανήκει ως μέλος το τμήμα.

Στο παράδειγμα έχουμε ένα αντικείμενο Αλφα με ιδιωτικό μέλος τη μεταβλητή μετρητής. Έχουμε δημόσια μέλη μια μέθοδο Εμφάνισε και μια ΧρήσηΜεθόδου. Η δεύτερη μέθοδος είναι τύπου συνάρτησης. Οι συναρτήσεις στη Μ2000 μπορούν να περαστούν με αναφορά.

Έχουμε ένα ξεχωριστό τμήμα Καταναλωτής ο οποίος ζητάει μια συνάρτηση. Εντός ο Καταναλωτής χρησιμοποιεί την συνάρτηση και αυτή η χρήση καταγράφεται στο αντικείμενο Αλφα, στο μετρητή του. Πριν την κλήση του καταναλωτή η Αλφα.Εμφάνισε δίνει 0 για τιμή μετρητή. Μετά τη κλήση δίνει τιμή μετρητή 10.

Πράγματι δέκα φορές χρησιμοποιείται η συνάρτηση. Η κλήση της συνάρτησης λέγεται κλήση προς τα πίσω (CallBack) διότι στην ουσία εκτελείται η συνάρτηση στην ομάδα άλφα έξω από το τμήμα Καταναλωτής. Ενώ σε ένα πέρασμα με αναφορά η επιστροφή τιμής γίνεται στο πέρας της κλήσης, εδώ η κλήση προς τα πίσω έγινε δέκα φορές, πριν τερματίσει η εκτέλεση του τμήματος Καταναλωτής.

```
Ομάδα Αλφα {
Ιδιωτικό:
```

```
    μετρητής=0
Δημόσιο:
    Τμήμα Εμφάνισε {
        Τύπωσε "Τιμή μετρητή:";.μετρητής
    }
    Συνάρτηση ΧρήσηΜεθόδου (Κ){
        =Κ**2
        .μετρητής++
    }
}
Τμήμα Καταναλωτής (&Μέθοδος()) {
    Για ι=1 έως 10
        Τύπωσε Μέθοδος(ι)
    Επόμενο
}
Αλφα.Εμφάνισε
Καταναλωτής &Αλφα.ΧρήσηΜεθόδου()
Αλφα.Εμφάνισε
```

***Πρόγραμμα 42**: Πέρασμα με αναφορά Μεθόδου Ομάδας*

Παρατηρήστε ότι ο διαχωρισμός ιδιωτικό και δημόσιο, δίνει στο αντικείμενο ένα μέρος που αποκρύπτεται, δηλαδή δεν υπάρχει το Αλφα.μετρητής (δεν είναι δημόσιο μέλος) και αυτό λέγεται **ενθυλάκωση**. Επίσης δίνει ένα μέρος δημόσιο το οποίο ορίζει αυτό που λέμε **διεπαφή**. Στο αντικείμενο Αλφα η διεπαφή έχει δυο μεθόδους.

Έχουμε ήδη δει ότι το Αλφα ως τιμή είναι ένας πολίτης πρώτης τάξης. Έχουμε δει παράδειγμα που η ομάδα μπαίνει σε θέση σε πίνακα. Εξ ορισμού αν Β είναι νέο όνομα και Αλφα είναι μια ομάδα τότε το Β=Αλφα κάνει το Β ένα αντίγραφο του Αλφα.

Η Ομάδα ως Πρότυπο

Είδαμε ήδη ότι μια ομάδα μπορεί να δημιουργήσει άλλη ομάδα, σε νέο όνομα ή ανώνυμη σε θέση σε πίνακα ή άλλο μαζικό καταχωρητή. Μια ιδέα εδώ είναι οι πίνακες ομάδων. Ενώ μπορούμε σε ένα πίνακα να βάζουμε οποιαδήποτε ομάδα σε οποιαδήποτε θέση, ο πίνακας ομάδας παίρνει εξ ορισμού μια ομάδα και αναπαράγει το ίδιο πρότυπο σε κάθε θέση. Μπορούμε να το σκεφτούμε αυτό ως κάθε θέση του πίνακα να έχει μια αρχική κατάσταση μιας ομάδας πρότυπο.

Στο παράδειγμα που ακολουθεί μια ομάδα έχει τρεις ιδιότητες, οι οποίες είναι ομάδες και αυτές. Μια ομάδα μπορεί να δέχεται τιμή και να δίνει τιμή, ή και τα δύο. Υπάρχει στον ορισμό μιας ομάδας η Ιδιότητα η οποία είναι μια μακροεντολή που αφήνει τον πραγματικό κώδικα (λέγεται syntactic sugar). Οι ιδιότητες ως ομάδες μπορούν να έχουν δικές τους ιδιότητες. Όλες οι ιδιότητες είναι δημόσιες. Εδώ έχουμε στην Ομάδα Τετράγωνο, το Μήκος_Πλευράς. Την ορίζουμε ως ομάδα, βάζουμε μια ιδιότητα Μέγεθος, και την ορίζουμε σαν Ιδιότητα μετά. Αυτό γίνεται γιατί η Ιδιότητα είναι και αυτή Ομάδα, και μπορούμε σε μια ομάδα να προσθέσουμε και άλλα στοιχεία ορισμού. Στην ιδιότητα εμβαδόν έχουμε μόνο την Αξία, δηλαδή είναι μόνο για ανάγνωση. Επειδή μια ομάδα

δεν μπορεί να δει κάτι έξω από τον δικό της ορισμό, ειδικά για τις ομάδες μέσα σε άλλες ομάδες, υπάρχει μια δυνατότητα να δει μέσω μιας αναφοράς κάτι από τη γονική ομάδα. Έτσι στο Εμβαδόν στο μέλος Αξία βλέπουμε το Μήκος_Πλευράς μέσω της αναφοράς του, το Μ.

Η Ομάδα Άλφα είναι έτοιμη για χειρισμό. Στο παράδειγμα φτιάχνουμε ένα πίνακα με τιμή τη τιμή της Άλφα, δηλαδή μια ανώνυμη ομάδα για κάθε θέση του πίνακα. Εδώ δεν έχουμε δείκτη σε ομάδα, δηλαδή κάθε θέση έχει ξεχωριστή ομάδα, παρόλο που όλες έχουν την ίδια αρχική κατάσταση. Στο παράδειγμα διαφοροποιούμε την ομάδα στο Α(0). Για να μην γράφουμε το κώδικα της εμφάνισης των στοιχείων, χρησιμοποιούμε μια απλή ρουτίνα, με ετικέτα, χρήση της Διαμέσου και της εντολής Επιστροφής. Οι απλές ρουτίνες είναι σαν να βρίσκεται ο κώδικας στο σημείο που τις καλούμε. Δεν έχουν χειρισμό τοπικών μεταβλητών όπως οι ρουτίνες με ονόματα με παρενθέσεις (Ρουτίνα/Τέλος Ρουτίνας)

```
Ομάδα Τετράγωνο {
    Ομάδα Μήκος_Πλευράς {
        Ιδιότητα Μέγεθος{
            Θέσε {
                Αν Αξία>4 τότε Αξία=4
                Αν Αξία<0.5 τότε Αξία=0.5
            }, Αξια
        }=1
    }
    Ιδιότητα Μήκος_Πλευράς {
        Θέσε {
            Αν Αξία<100 τότε Αξία=100
        },
        Αξία {
            Αξία=Αξία*.Μέγεθος
        }
    }=100
    Ιδιότητα Εμβαδόν {
        Αξία {
            Ένωσε γονικό Μήκος_Πλευράς στη Μ
            Αξία=Μ*Μ
        }
    }
    Ιδιότητα Συντεταγμένες=(0,0)
}
Πίνακας Τετράγωνο(10)=Τετράγωνο
Τετράγωνο(0).Συντεταγμένες=(1000, 2000)
Τετράγωνο(0).Μήκος_Πλευράς.Μέγεθος=2

Για Τετράγωνο(0) {
    Διαμέσου Απλή_Ρουτίνα
```

```
}
Για Τετράγωνο(1) {
    Διαμέσου Απλή_Ρουτίνα
}

Τέλος
Απλή_Ρουτίνα:
    Τύπωσε "Μήκος Πλευράς: ";.Μήκος_Πλευράς
    Τύπωσε "Εμβαδόν: ";.Εμβαδόν
    Τύπωσε "Συντεταγμένες: "; .Συντεταγμένες
Επιστροφή
```

***Πρόγραμμα 43**: Ομάδα Τετράγωνο - Χρήση Ιδιοτήτων*

Οι τέσσερις Ιδέες των Αντικειμένων στις Ομάδες

Στον αντικειμενοστραφή προγραμματισμό υπάρχουν οι παρακάτω τέσσερις κύριες ιδέες:

1. Κληρονομικότητα
2. Ενθυλάκωση
3. Αφαίρεση
4. Πολυμορφισμός

Αυτές οι ιδέες υλοποιούνται με διάφορους τρόπους σε διαφορετικές γλώσσες που συχνά οδηγούν σε παρερμηνείες.

Η **Κληρονομικότητα** σαν ιδέα μας λέει ότι ένα αντικείμενο Α μπορεί να κληρονομεί άλλο αντικείμενο Β, με συνέπεια να έχει τις ιδιότητες και μεθόδους του αντικειμένου Β που κληρονομεί.

Η **Ενθυλάκωση** σαν ιδέα μας λέει ότι μόνο ένα επιλεγμένο μέρος των πληροφοριών του αντικειμένου εκτίθεται έξω από το αντικείμενο. Αναφέρεται σε ιδιότητες όχι σε μεθόδους.

Η **Αφαίρεση** σαν ιδέα μας λέει ότι μόνο ένα επιλεγμένο σύνολο μεθόδων, τις λεγόμενες υψηλού επιπέδου μεθόδους, δίνονται για πρόσβαση στο αντικείμενο.

Ο **Πολυμορφισμός** σαν ιδέα μας λέει ότι μια μέθοδος είναι αυτή που φαίνεται ονομαστικά χωρίς να είναι απαραίτητο να υλοποιείται με τον ίδιο τρόπο σε άλλο αντικείμενο με το ίδιο όνομα μεθόδου.

Σε πολλά βιβλία υπάρχει μια σύγχυση και προς τις τέσσερις ιδέες! Οι συγχύσεις ξεκινούν από το πως υλοποιούνται οι παραπάνω ιδέες σε γλώσσες προγραμματισμού.

Σίγουρα μια παρεξήγηση υπάρχει μεταξύ Αφαίρεσης και Ενθυλάκωσης, διότι και οι δυο στηρίζονται στο τι είναι θεατό. Αν έχω μια ιδιωτική μέθοδο, τότε αυτό δεν ανήκει στην Ενθυλάκωση αλλά στην Αφαίρεση. Δηλαδή έχω μια μέθοδο που δεν προβάλλω έξω από το αντικείμενο, ή με άλλα λόγια έχω μια κρυμμένη λειτουργικότητα.

Επίσης μια άλλη παρεξήγηση έχει να κάνει με δυο δυνατότητες των μεθόδων, της Υπερφόρτωσης ορισμών και της Υποσκέλισης μεθόδου. Κάποια φορά τον Πολυμορφισμό τον αναφέρουν στην μία ή στην άλλη δυνατότητα, και καμιά φορά και στις δυο. Η ιδέα του πολυμορφισμού έχει να κάνει με

την Υποσκέλιση μεθόδου. Επειδή σε κάποιες γλώσσες ο μεταφραστής έδινε την δυνατότητα της Υπερφόρτωσης ορισμών σε συναρτήσεις, η δυνατότητα αυτή ονομάστηκε Πολυμορφισμός. Αυτό όμως δεν είναι ιδέα των αντικειμένων. Η ιδέα του Πολυμορφισμού στα αντικείμενα είναι ξεκάθαρη. Έχουμε δυο αντικείμενα Α1 και Α2, να κληρονομούν από ένα άλλο Β, και σε αυτό υπάρχει μια μέθοδος Μ. Το ένα αντικείμενο κρατάει τη μέθοδο Μ που κληρονόμησε, ενώ το άλλο την έχει αλλάξει με δική του. Όταν σε μια μέθοδο, οπουδήποτε είναι αυτή, ζητάμε το αντικείμενο Χ να είναι όπως το Β, και εκεί καλούμε τη μέθοδο Χ.Μ, τότε αν βάλουμε το Α1 ως Χ αφού είναι Β θα κληθεί η Μ όπως του Β αλλά αν βάλουμε το Α2 θα κληθεί η δική του Μ που δεν είναι ίδια με αυτή του Β παρά μόνο στο όνομα και στην υπογραφή (τον τύπο και αριθμό παραμέτρων). Αυτός είναι ο Πολυμορφισμός στα αντικείμενα. Επειδή όμως υπάρχει και δυνατότητα Πολυμορφισμού στις μεθόδους ενός αντικειμένου, ως Υπερφόρτωση, αυτό φαντάζει ότι είναι μορφή πολυμορφισμού των αντικειμένων. Δεν είναι έτσι, παραμένει ένας πολυμορφισμός μεθόδων ανεξάρτητα αν η μέθοδος είναι σε αντικείμενο, ή είναι μια συνάρτηση γενική ή στατική.

Αφού είδαμε για την Ενθυλάκωση, την Αφαίρεση και τον Πολυμορφισμό, το τι παρεξηγήσεις γίνονται ας πάμε στο μεγάλο θέμα του τι είναι κληρονομικότητα πραγματικά!

Κληρονομικότητα Αντικειμένων

Η Visual Basic 6, η γλώσσα που γράφτηκε ο διερμηνευτής και το περιβάλλον της Μ2000, έχει αντικείμενα με κλάσεις. Αν ψάξει κανείς να βρει τρόπο να φτιάξει μια κλάση κληρονομώντας από άλλη κλάση, μάλλον θα απογοητευτεί. Αυτό που μπορεί να κάνει κανείς είναι να ορίσει μια διεπαφή (interface) και να ορίσει ότι έστω δυο κλάσεις η Α και η Β υλοποιούν την συγκεκριμένη διεπαφή, εκτός από ότι άλλο δικό τους θέλουν. Η συμμορία των τεσσάρων (Gang of Four), που αναφέρονται στο κεφάλαιο με τα προγραμματιστικά παραδείγματα, έλεγε ότι πρέπει να σκεφτόμαστε τα αντικείμενα ως διεπαφές, όχι το πώς κάνουν κάτι, αλλά το πως τα χειριζόμαστε μέσω της διεπαφής τους (του δημόσιου προσώπου κατά μια έννοια). Η γλώσσα του 1997 ακολουθεί αυτό το χαρακτηριστικό που γράφτηκε το 1994, εκδόθηκε το 1995 από τους GoF, δηλαδή σχετικά πρόσφατα για μια τέτοια γλώσσα. Επίσης μας λένε ότι είναι προτιμότερο να χρησιμοποιούμε αντικείμενα μέσα σε αντικείμενα παρά τη κληρονομικότητα αντικειμένου. Εδώ είναι η διαφορά όταν ένα αντικείμενο Έχει Ένα, από το να Είναι Ένα. Επίσης η VB 6 μπορεί να έχει σε ένα αντικείμενο οποιαδήποτε άλλα, με δείκτες, δηλαδή και εδώ ακολουθεί την οδηγία των τεσσάρων.

Πολλές γλώσσες κατασκευάστηκαν με περιορισμούς ως προς το εύρος της κληρονομικότητας. Άλλες επιτρέπουν την πολλαπλή κληρονομικότητα και άλλες όχι. Δεν γίνεται αυτό από παραξενιά, αλλά από κατασκευαστικούς λόγους. Η πολλαπλή κληρονομικότητα ως έκφραση μπορεί να σημαίνει πολλά πράγματα. Αν έχουμε το αντικείμενο Α ως αντικείμενο Β συν κάποιες δικές του ιδιότητες και μεθόδους τότε λέμε ότι το Α κληρονομεί από το Β. Τώρα αν πάρουμε άλλο ένα Κ ως Α και βάλουμε και μερικές ιδιότητες και μεθόδους, τότε το Κ θα είναι και Α και Β. Θα μπορούσε να πει κανείς ότι το Κ έχει πολλαπλή κληρονομικότητα, επειδή είναι Α και Β ταυτόχρονα. Γενικά δεν θεωρείται αυτή η σχέση μεταξύ Κ, Α και Β ως πολλαπλή κληρονομικότητα αλλά ως μια ιεραρχία κληρονομικότητας, επειδή το Α είναι και Β. Αν το βλέπαμε σε ένα σχήμα το Β θα είχε ένα βελάκι στο Α και το Α στο Κ. Πολλαπλή κληρονομικότητα έχουμε όταν η ιεραρχία ξεκινάει από περισσότερες αφετηρίες και όχι από μια. Έτσι αν τα Α και Β είναι διαφορετικά αντικείμενα, δηλαδή δεν ισχύει το Α να είναι Β ή το Β να είναι Α, τότε το Κ αν φτιαχτεί από τα Α και Β τότε το Κ έχει πολλαπλή κληρονομικότητα, αφού τα Α και Β είναι δυο διαφορετικές αφετηρίες στην ιεραρχία.

Ο τρόπος κατασκευής των αντικειμένων προγραμματιστικά καθορίζει και τις δυνατότητές τους. Εδώ λοιπόν θα γίνει η αναφορά των δυο διαδεδομένων τρόπων κατασκευής: Την κατασκευή βάσει πρότυπου και την κατασκευή βάσει κλάσης. Η JavaScript (1995) χρησιμοποιεί το πρότυπο, η Python (1989) την κλάση, όμως και οι δυο γλώσσες μπορούν να κάνουν αλλαγές στην σύσταση των τελικών αντικειμένων. Η Java (1995) χρησιμοποιεί κλάσεις και δεν επιδέχεται αλλαγές σύστασης στα τελικά αντικείμενα, επιπλέον δεν υποστηρίζει πολλαπλή κληρονομικότητα.

Δημιουργία Αντικειμένων σε τέσσερις γλώσσες

Ακολουθούν ενδεικτικά προγράμματα για τέσσερις γλώσσες, JavaScript, Python και Java και M2000. Και οι τέσσερις γλώσσες είναι διερμηνευτές, με τη διαφορά ότι η Java κάνει μετάφραση σε ενδιάμεσο κώδικα και ο διερμηνευτής είναι η εικονική μηχανή που εκτελεί το κώδικα. Αυτό την κάνει πιο γρήγορη από τις άλλες γλώσσες. Εφάμιλλη της Java είναι η C# που επίσης τρέχει σε εικονική μηχανή, και όπως και η java δηλώνονται οι τύποι απαραίτητα. Οι JavaScript, Python δουλεύουν χωρίς ειδικές δηλώσεις στις τυπικές παραμέτρους. Η M2000 δουλεύει και με τύπους και χωρίς, επειδή είναι στραμμένη προς την εκπαίδευση, και στα αρχικά στάδια είναι πιο εύκολο όταν δεν έχουμε μεγάλο κώδικα, αλλά έναν λιτό και λειτουργικό, να αποτυπώσουμε τη σκέψη μας.

JavaScript

Δείτε ότι η name δεν υπάρχει στο αντικείμενο person. Η const κάνει τα ονόματα που ακολουθούν να μην μπορούν να αλλάξουν τιμή, όμως επειδή είναι δείκτες σε αντικείμενα, αυτό που έχουν, που δείχνει ο δείκτης, μπορεί να μεταβληθεί.

```
const person = {
  isHuman: false,
  printIntroduction: function() {
    console.log(`My name is ${this.name}. Am I human? ${this.isHuman}`);
  }
};
const me = Object.create(person);
me.name = 'Matthew'; // "name" είναι ιδιότητα του "me", αλλά όχι του "person"
me.isHuman = true; // οι κληρονομημένες ιδιότητες μπορούν να αλλαχθούν
me.printIntroduction();
// εξαγωγή: "My name is Matthew. Am I human? true"
```

Python

Πιο απλά στην Python, ορίζουμε ιδιότητες ακόμα και μέσα στον κατασκευαστή. Λειτουργεί με κλάσεις ενώ διατηρεί το πλεονέκτημα να αυξάνει/μειώνει τα μέλη του. Δείτε ότι για τα name και

age δεν δηλώθηκαν τύποι. Δείτε επίσης ότι η πρώτη παράμετρος της __init__() συνάρτησης είναι η self (θα μπορούσε να ήταν οποιοδήποτε όνομα), και αναφέρεται στο τρέχον αντικείμενο.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

Java

Μια γλώσσα που βασίζεται σε μια θαυμάσια εικονική μηχανή, που τρέχει παντού. Εδώ έχουμε κλάσεις, με την πιο καθαρή έννοια (ή πιο γνωστή για την ακρίβεια). Δείτε ότι σε κάθε όνομα μπαίνει ο τύπος. Παραδόξως στις συναρτήσεις δεν μπαίνει δήλωση ότι ακολουθεί συνάρτηση αλλά προσδιορίζεται από τις παρενθέσεις στο όνομα και το μπλοκ κώδικα που ακολουθεί. Δηλώνεται όμως αν γυρίζει κάποιο αποτέλεσμα ή όχι (void), και το επίπεδο θέασης (πχ public για δημόσιο).

```
public class Puppy {
  int puppyAge;
  public Puppy(String name) {
    // Ο κατασκευαστής έχει μια παράμετρο, name.
    System.out.println("Name chosen is :" + name );
  }
  public void setAge( int age ) {
    puppyAge = age;
  }
  public int getAge( ) {
    System.out.println("Puppy's age is :" + puppyAge );
    return puppyAge;
  }
  public static void main(String []args) {
    /* Δημιουργία Αντικειμένου */
```

```
    Puppy myPuppy = new Puppy( "tommy" );
    /* Κλήση μεθόδου για να θέσει την ηλικία age */
    myPuppy.setAge( 2 );
    /* Κλήση μεθόδου για να διαβάσει την ηλικία age */
    myPuppy.getAge( );
    /* Άμεση ανάγνωση ηλικίας επειδή δεν είναι ιδιωτική */
    System.out.println("Variable Value :" + myPuppy.puppyAge );
  }
}
```

M2000

Η M2000 χρησιμοποιεί το πρότυπο, αλλά και τις κλάσεις όπως θα δούμε παρακάτω:

```
\\ δημιουργία αντικειμένου με ορισμό (literal)
\\ δεν έχουμε δηλώσει τύπο
Ομάδα Άλφα {
    X=10
    Συνάρτηση X2 {
        =.X*2
    }
}
Τύπωσε  Άλφα.X, Άλφα.X2()
\\ δημιουργία με αντιγραφή (cloning)
Βήτα=Άλφα

Κλάση Δέλτα {
    X=10
    Συνάρτηση X2 {
        =.X*2
    }
Κλάση:
    Τμήμα Δέλτα (.X) {
    }
}
Κάπα=Δέλτα(100)
Τύπωσε Κάπα.X, Κάππα.X2()
Τύπωσε Κάπα είναι τύπος Δέλτα = Αληθές
```

```
\\ Δήλωση κλάσης για εσωτερική ομάδα σε ομάδα
\\ δήλωση τύπου σε ομάδα
Ομάδα Ζήτα {
      Τύπος: Ζήτα1
      Δέλτα Μ(300)
}
Τύπωσε Ζήτα.Μ.Χ, Ζήτα.Μ.Χ2()
Τύπωσε Ζήτα είναι τύπος Ζήτα1


\\ συγχώνευση αντικειμένων (Κληρονομικότητα βάσει αντικειμένων)
Μ=Ζήτα Με Δέλτα(500)
Τύπωσε Μ είναι τύπος Ζήτα1 = Αληθές
Τύπωσε Μ είναι τύπος Δέλτα = Αληθές
Τύπωσε Μ.Μ είναι τύπος Δέλτα = Αληθές
\\ Κληρονομικότητα Κλάσεων
\\ Φτιάχνεται μια κλάση με πρώτο ορισμό αυτό της Δέλτα
Κλάση Θήτα Ως Δέλτα {
      Συνάρτηση Χ2 {
            =.Χ**2
      }
Κλάση:
      Τμήμα Θήτα {
            .Δέλτα
      }
}
Λ=Θήτα(3)
Τύπωσε Λ.Χ2()=9
Τύπωσε Λ είναι τύπος Δέλτα = Αληθές
Τύπωσε Λ είναι τύπος Θήτα = Αληθές
```

***Πρόγραμμα 44**: Δημιουργία Αντικειμένου Ομάδα*

Αυτό που κάνει η Μ2000 είναι να χρησιμοποιεί ένα αντικείμενο ενός μοναδικού τύπου, του Group. Αν σε οποιαδήποτε ομάδα παραπάνω δώσουμε το Τύπος$() πχ το Τύπος$(Λ) θα μας δώσει το Group. Μπορούμε να ορίσουμε έναν ή περισσότερους τύπους, άμεσα όπως στην ομάδα Ζήτα του παραδείγματος. Οι ορισμοί τύπων χρησιμοποιούνται όταν θέλουμε να πάρουμε μια τυπική παράμετρο (σε ένα τμήμα ή μια συνάρτηση κατά τη κλήση) και θέλουμε να έχει έναν συγκεκριμένο τύπο, δηλαδή τον χρησιμοποιούμε για ασφάλεια. Μια κλάση δίνει τον τύπο της και όποιον άλλο κληρονομεί αυτόματα στο τελικό. Η συγχώνευση αντικειμένων κάνει συγχώνευση και στους τύπους (αν υπάρχουν).

Το αντικείμενο Group, μια κλάση γραμμένη στη VB6, έχει μια λίστα για δεδομένα και μια για μεθόδους. Όσο το αντικείμενο είναι επώνυμο, δεν κρατάει στοιχεία παρά μόνο αναφορές στο που είναι τα στοιχεία. Αν το αντικείμενο είναι ανώνυμο και δεν χρησιμοποιείται τότε τα στοιχεία είναι ενσωματωμένα σε ιδιωτικό χώρο στο αντικείμενο. Αν χρησιμοποιείται, τότε έχει γίνει επώνυμο με όνομα που του δίνει ο διερμηνευτής, άρα τα στοιχεία είναι εκεί που τα τοποθετεί ο διερμηνευτής και όχι στο αντικείμενο. Δηλαδή όταν ζητάμε το Ζήτα.Μ.Χ ο διερμηνευτής το βρίσκει άμεσα και όχι με εύρεση του Ζ, μετά του Μ στο Ζ και μετά του Χ στο Ζ. Για το λόγο αυτό μπορούμε να περάσουμε το Ζήτα.Μ.Χ ως αναφορά. Η μεταβλητή που θα φτιαχτεί ως αναφορά του Ζήτα.Μ.Χ είναι στον ίδιο χώρο με αυτό. Όταν περάσουμε με αναφορά μια ομάδα ο διερμηνευτής φτιάχνει μια νέα ομάδα και σε αυτήν οι αναφορές δείχνουν τις ίδιες αναφορές που φαίνονται στην αρχική. Έτσι αν αυξήσουμε τα στοιχεία της ομάδας που περάσαμε με αναφορά δεν θα πειραχθεί η αρχική, ενώ τα τυχόν νέα μέλη θα μπουν με αναφορά σε αυτά τα νέα που θα δημιουργήσει ο διερμηνευτής, και δεν θα έχουν σχέση με την αναφερόμενη αρχική ομάδα. Το αντικείμενο Ομάδα που περνάμε με αναφορά μπορεί να κληρονομήσει από άλλο ενώ διατηρεί την αναφορά στο αρχικό.

Έχει σημασία πως λειτουργεί κάθε γλώσσα, για να αντιληφθούμε το πώς ένα αντικείμενο Α, που κληρονομεί από το Β, αφομοιώνει το αντικείμενο Β. Στη Μ2000 γίνεται πιο απλά από άλλες γλώσσες επειδή χρησιμοποιεί συγχώνευση. Έτσι τα ιδιωτικά μέλη του Β γίνονται ιδιωτικά μέλη του Α, άρα ως προς την κληρονομικότητα δεν λειτουργεί η ενθυλάκωση. Εσωτερικά το Α δεν έχει καμία δυνατότητα να διαχωρίσει τα μέλη του Β. Αν θέλουμε το Α να έχει ένα Β, και να είναι ένα Β τότε μπορούμε να βάλουμε στην Α ένα μέλος που θα είναι Β, και ταυτόχρονα να κάνουμε το Α να είναι Β. Σαν το μοτίβο Adapter (που θα δούμε αργότερα), με τη διαφορά ότι η προσαρμογή αφορά το Β, μέλος του Α και κάνει το Α να είναι ένα Β.

Κατασκευή Ομάδας με Κλάση

Η αρχική ιδέα για τις κλάσεις ήταν να μπορούν να ετοιμάζουν αντικείμενα βάσει ενός κατασκευαστή. Δηλαδή να είναι συναρτήσεις που να επιστρέφουν ομάδες. Τότε ήρθε και η ιδέα να ξεκινάει μια ομάδα με κάποια μέλη που δεν θα συμπεριληφθούν στο τελικό. Ο κατασκευαστής για παράδειγμα δεν μας ενδιαφέρει να βρίσκεται στο τελικό αντικείμενο. Έτσι ορίστηκε η τρίτη ετικέτα η **Κλάση:** (οι ετικέτες στη δομή της ομάδας μπορούν να έχουν ή όχι τόνους, δεν χρησιμοποιούνται για άλματα). Ενώ η κλάση είναι συνάρτηση, η εξυπηρέτηση γίνεται από ένα τμήμα με το ίδιο όνομα με αυτό της κλάσης. Σε αυτό το τμήμα δίνεται ο σωρός τιμών. Μπορούμε να κάνουμε έλεγχο και με ένα τμήμα να εξυπηρετούμε διαφορετικές εισαγωγές ορισμάτων. Με αυτό το τρόπο πετυχαίνουμε το **πολυμορφισμό** στη Μ2000, στον τρόπο που λέγεται Υπερφόρτωση (overloading). Σε άλλες γλώσσες γίνεται με σειρά μεθόδων με διαφορετικές υπογραφές (δηλαδή τύπους ορισμάτων) ενώ εδώ έχουμε μία μέθοδο και σε αυτήν κάνουμε έλεγχο υπογραφών και να επιλέξουμε τη σωστή ακολουθία (δομή επιλογής). Ο διερμηνευτής εξ ορισμού δεν ελέγχει τον αριθμό και τον τύπο ορισμάτων όταν εκτελεί μια κλήση στο μέρος της κλήσης. Ο έλεγχος γίνεται στο μέρος αυτού που καλούμε. Αυτό δίνει πολλές δυνατότητες στη Μ2000, και ειδικά στις συναρτήσεις δίνει τις λεγόμενες Variadic, τις συναρτήσεις που παίρνουν οποιοδήποτε αριθμό και τύπο ορισμάτων. Η χρήση σε άλλες γλώσσες ξεχωριστών μεθόδων με ίδιο όνομα και διαφορετικές υπογραφές (αριθμό ή και τύπο τυπικών παραμέτρων) βοηθάει στον διορθωτή να εμφανίζει επιλογές. Εδώ μας ενδιαφέρει η γνώση να προέρχεται από τεκμηρίωση, δηλαδή να υπάρχει μια περιγραφή του τι κάνει κάθε μέθοδος και πώς μπορεί να συνταχθεί.

Κληρονομικότητα Κλάσεων

```
Κλάση Εμφανιστής {
    Τμήμα Εμφάνισε {
        Λάθος "Αφηρημένη"
    }
}
Τμήμα Καταναλωτής (Μ ως Εμφανιστής) {
    Μ.Εμφάνισε
}

Κλάση Πρώτη ως Εμφανιστής {
Ιδιωτικό:
    Χ=10, Υ=20
Δημόσιο:
    Τμήμα Εμφάνισε {
        Τύπωσε .Χ, .Υ
    }
Κλάση:
    Τμήμα Πρώτη {
        Αν Ταύτιση("ΑΑ") Τότε
            Διάβασε .Χ, .Υ
        Τέλος Αν
    }
}
Μ=Πρώτη()
Καταναλωτής Μ      ' 10 20
Ζ=Πρώτη(100,200)
Καταναλωτής Ζ      ' 100 200

Κλάση Δεύτερη ως Εμφανιστής {
Ιδιωτικό:
    Πρώτη Μ1(50, 80)
    Πρώτη Μ2
    Πρώτη Μ3(1, 2)
Δημόσιο:
    Τμήμα Εμφανισε {
        .Μ1.Εμφάνισε
        .Μ2.Εμφάνισε
        .Μ3.Εμφάνισε
    }
}
Ζ1=Δεύτερη()
Καταναλωτής Ζ1
```

***Πρόγραμμα 45**: Κληρονομικότητα με Κλάσεις*

Η ιδέα της κληρονομικότητας είναι να έχουμε μια βάση, εδώ την κλάση Εμφανιστής και ένα τμήμα που να ζητάει μια ομάδα που να είναι Εμφανιστής. Η κλάση λέει ότι πρέπει να υπάρχει μια σωστή μέθοδος Εμφάνισε. Η κλάση Πρώτη λέμε να είναι και κλάση Εμφανιστής. Αν δεν βάλουμε την μέθοδο Εμφάνισε τότε θα κληθεί εκείνη με το λάθος! Η κλάση Πρώτη έχει τρία μέρη. Το ιδιωτικό με τη "κατάσταση" (state) σε δυο μεταβλητές Χ και Υ. Το δημόσιο που είναι ίδιο με αυτό της κλάσης Εμφανιστής και ένα που δηλώνει ότι τα παρακάτω δεν θα περιέχονται, η ετικετα Κλάση (έχει μια άνω και κάτω τελεία αμέσως μετά το όνομα). Εκεί έχουμε το κατασκευαστή που κοιτάει αν υπάρχουν δυο αριθμοί (αυτό λέει το "ΑΑ" αν θέλαμε γράμματα, δηλαδή αλφαριθμητικά θα βάζαμε το "ΓΓ", δείτε το Βοήθεια Ταύτιση() από τον διερμηνευτή για να δείτε τι άλλο υπάρχει).

Στο τμήμα Καταναλωτής βάζουμε μια αντιγραφή ομάδας και εκεί ελέγχεται ο τύπος. Οι κλάσεις δίνουν τύπους αυτόματα στις ομάδες. Εδώ δόθηκε ο τύπος Εμφανιστής και ο τύπος Πρώτη. Μπορεί μια κλάση να κληρονομεί από περισσότερες από μια κλάσεις. Πρώτα γράφεται ο κώδικας της πλέον δεξιάς και στο τέλος γράφεται ο κώδικας που βλέπουμε στη κλάση. Η συνάρτηση Εμφάνισε έχει γραφτεί δυο φορές, και η δεύτερη την άλλαξε. Αν είχαμε βάλει Συνάρτηση Τελική Όνομα.. τότε όποια άλλη προσπάθεια για αλλαγή κώδικα θα αστοχούσε σιωπηλά (χωρίς να φέρει λάθος, γιατί είναι κανονική λειτουργία).

Στο δεύτερο μέρος φτιάχνουμε μια άλλη κλάση που κληρονομεί από τη κλάση Εμφανιστής. Σε αυτή την κλάση έχουμε ιδιωτικά μέλη τρεις ομάδες, τις οποίες κατασκευάζουμε στη κλήση της κλάσης Δεύτερη ως Δεύτερη(). Η πρώτη() είναι γενική και για το λόγο αυτό φαίνεται στη Δεύτερη(). Και η Δεύτερη() είναι γενική. Όλες οι κλάσεις που ορίζονται σε τμήματα και συναρτήσεις είναι γενικές. Μπορούμε να έχουμε ορισμούς κλάσεων μέσα σε ομάδες και κλάσεις. Αυτοί οι ορισμοί δεν είναι γενικοί είναι μέλη της ομάδας και μπορεί να είναι και ιδιωτικά (και αυτά μπορούν να έχουν κληρονομικότητα, από κλάσεις μέσα ή έξω από τη ομάδα).

Η δεύτερη φάση θέλει να δείξει το Έχει Ένα (ενώ η κληρονομικότητα ορίζει το Είναι Ένα), Έτσι η ομάδα Ζ1 **είναι ένα** αντικείμενο Εμφανιστής και **έχει τρία** αντικείμενα της κλάσης Πρώτη. Το τμήμα Καταναλωτής, δεν ενδιαφέρεται το τι έχει το Ζ1, μόνο το τι είναι το Ζ1.

Στον αντικειμενοστραφή προγραμματισμό πρέπει να σκεφτόμαστε με τη διεπαφή. Όπως φαίνεται στο πρόγραμμα του παραδείγματος αμέσως μετά την διεπαφή στη κλάση Εμφανιστής μπήκε ο Καταναλωτής των αντικειμένων και εκεί μόνο η Εμφάνισε χρησιμοποιείται. Το τι έχει εν τέλει ένα αντικείμενο δεν μας ενδιαφέρει!

Ακολουθεί ένα ακόμα παράδειγμα, θα το βρείτε στο αρχείο info με το όνομα Κλάση1. Το αρχείο info δίνεται στην εγκατάσταση, και στο Readme.txt έχει οδηγίες γι'αυτό.

Το παράδειγμα βγάζει την παρακάτω αναφορά από μια ρουτίνα που "πιστεύει" ότι έχει πάντα το ίδιο αντικείμενο, με τύπο Κατοικίδιο. Όμως κάθε κλάση που κληρονομεί από το Κατοικίδιο κάνει κάποιες αλλαγές και αυτές φαίνονται στην εξαγωγή του αποτελέσματος:

```
Τελική Αναφορά
Με φωνάζουν Αζόρ και είμαι σκύλος και είμαι κατοικίδιο με 4 πόδια και όλη μέρα γαβγίζω
Με φωνάζουν Ψιψίνα και είμαι γάτα και είμαι κατοικίδιο με 4 πόδια και όλη μέρα νιαουρίζω
Με φωνάζουν Φλοξ και είμαι καναρίνι με 2 πόδια και φτερά και όλη μέρα κελαηδάω
Με φωνάζουν Ντορή και είμαι κατοικίδιο με 4 πόδια και όλη μέρα χλιμιντρίζω
```

```
Κλάση Ζώο {
Ιδιωτικό:
   πόδια=0
   ήχος$="τίποτα"
Δημόσιο:
   Συνάρτηση Τελική ΠόσαΠόδια() {
      =.πόδια
   }
   Συνάρτηση ΚάνωΉχο$() {
      =.ήχος$
   }
}
Κλάση Κατοικίδιο ως Ζώο {
Ιδιωτικό:
   όνομα$
   Μήνυμα$="Με φωνάζουν {0} και είμαι κατοικίδιο με {1} πόδια και όλη μέρα {2}"
Δημόσιο:
   Συνάρτηση Μήνυμα$() {
      =.Μήνυμα$
   }
   Συνάρτηση ΜεΦωνάζουν$() {
      =.όνομα$
   }
Κλάση:
   Τμήμα Κατοικίδιο {
      Διάβασε .όνομα$, .ήχος$, .πόδια
   }
}
Κλάση Σκύλος ως Κατοικίδιο {
   Συνάρτηση ΜεΦωνάζουν$() {
      =.όνομα$+" και είμαι σκύλος"
   }
Κλάση:
   Τμήμα Σκύλος(.Όνομα$)    {
      .πόδια<=4
      .ήχος$<="γαβγίζω"
   }
}
Κλάση Γάτα ως Κατοικίδιο {
   Συνάρτηση ΜεΦωνάζουν$() {
      =.όνομα$+" και είμαι γάτα"
   }
```

```
Κλάση:
   Τμήμα Γάτα(.Όνομα$)    {
      .πόδια<=4
      .ήχος$<="νιαουρίζω"
   }
}
Κλάση Πουλί ως Κατοικίδιο {
Ιδιωτικό:
   τύπος$="κάτι"
   Μήνυμα$="Με φωνάζουν {0} με {1} πόδια και φτερά και όλη μέρα {2}"
Δημόσιο:
   Συνάρτηση ΜεΦωνάζουν$() {
      =.όνομα$+" και είμαι "+.τύπος$
   }
Κλάση:
   Τμήμα Πουλί(.Όνομα$, .τύπος$)    {
      .πόδια<=2
      .ήχος$<="κελαηδάω"
   }
}
Εγγραφο ΌλαΜαζι$={Τελική Αναφορά
}
Αζορ=Σκύλος("Αζόρ")
Τι(Αζορ)
Ψιψίνα=Γάτα("Ψιψίνα")
Τι(Ψιψίνα)
Φλοξ=Πουλί("Φλοξ", "καναρίνι")
Τι(Φλοξ)
ΆλλοΚατοικίδιο=Κατοικίδιο("Ντορή", "χλιμιντρίζω", 4)
Τι(ΆλλοΚατοικίδιο)
\\ στέλνουμε στο πρόχειρο την αναφορά
Πρόχειρο ΌλαΜαζι$


Τέλος
\\ Η ρουτίνα δέχεται οτιδήποτε είναι κατοικίδιο
\\ έτσι δέχεται Κατοικίδιο, Γάτα, Σκύλος, Πουλί
Ρουτίνα Τι(Α ως Κατοικίδιο)
   Τοπική Εξ$=Μορφή$(Α.Μήνυμα$(),Α.ΜεΦωνάζουν$(), Α.ΠόσαΠόδια(), Α.ΚάνωΉχο$())
   ΌλαΜαζι$=Εξ$+{
   }
   Αναφορά Εξ$
Τέλος Ρουτίνας
```

***Πρόγραμμα 46**: Παράδειγμα με κλάση Ζώο*

Ο κώδικας του παραπάνω παραδείγματος είναι κατανοητός. Η εντολή Αναφορά εμφανίζει κείμενο με αναλογική γραφή και αυτόματη αναδίπλωση λέξεων, εφαρμόζοντας διάφορα στυλ φορμαρίσματος. Το βασικό εδώ στυλ είναι η στοίχιση αριστερά.

Στη ρουτίνα Τι() ορίζουμε τοπικές με στόχο να σκιάσουμε άλλες με ίδιο όνομα. Αυτή είναι μια καλή πρακτική για τις ρουτίνες αφού βλέπουν ότι έχει το τμήμα. Η ΌλαΜαζί$ φαίνεται σαν αλφαριθμητικό αλλά είναι έγγραφο. Ένα έγγραφο έχει κάθε παράγραφο σε ξεχωριστό αλφαριθμητικό. Αυτό βελτιώνει πολλά με κύριο να προσθέτει γρήγορα στο τέλος κομμάτια παραγράφων. Η εκχώρηση στο έγγραφο κάνει εγγραφή στο τέλος, δηλαδή προσθέτει (λέγεται append στα αγγλικά). Εδω μας ενδιαφέρει να δώσουμε το αλφαριθμητικό (εξάγουμε ένα ενιαίο αλφαριθμητικό), στο πρόχειρο των Windows. Η ρουτίνα μας βολεύει γιατί έχει πρόσβαση στην ΟλαΜαζί$ δεν χρειάζεται να την περάσουμε σαν όρισμα.

Δείκτες σε Ομάδες

Σε αυτό το άρθρο δεν θα αναφερθούν ιδιαίτερες δυνατότητες των αντικειμένων, πέραν αυτών που χρειάζονται για να ολοκληρωθούν τα προγραμματιστικά μοτίβα. Εδώ λοιπόν θα εξηγηθεί η χρήση των δεικτών ομάδων.

Ο δείκτης σε ομάδα στη Μ2000 έχει τη δυνατότητα να δείχνει οποιαδήποτε ομάδα, και να αλλάζει να δείχνει άλλη ομάδα ή την μηδενική ομάδα (Null). Η μηδενική ομάδα είναι τύπος μηδενικός και null (μπορούμε να χρησιμοποιήσουμε όποιο θέλουμε για τον έλεγχο τύπου). Αυτό που λέμε δείκτη στην ομάδα, άλλες γλώσσες το έχουν ως τον μοναδικό τρόπο να αναφέρονται σε αντικείμενα. Αυτό συμβαίνει στην Python, όπου τα πάντα είναι δείκτες σε κάτι. Το μειονέκτημα σε αυτή την αντιμετώπιση είναι ότι δεν περνάει σε κλήση η τιμή αλλά ο δείκτης ως αντίγραφο, ως τιμή ενός άλλου δείκτη. Συνέπεια αυτού είναι ότι το αντικείμενο θα περάσει και ό,τι αλλαγές γίνουν θα επιστρέψουν, με την διαφορά από το πέρασμα με αναφορά ότι αν αλλαχθεί ο δείκτης, αυτός δεν θα γυρίσει, άρα την στιγμή της αλλαγής θα απομονωθεί το αντικείμενο που αρχικά περάσαμε. Αυτή τη λειτουργικότητα την έχουμε και με τους δείκτες σε ομάδα στη Μ2000. Υπάρχει όμως και το πέρασμα του δείκτη με αναφορά. Αυτό σημαίνει ότι αν αλλαχθεί ο δείκτης, θα φανεί και στο μέρος της κλήσης (αυτό δεν μπορεί να γίνει στη Python, δεν έχει πέρασμα με αναφορά, πρέπει να γίνει επιστροφή τιμής ο νέος δείκτης, ως επιστροφή μεθόδου)

Επειδή υπάρχουν δυο ειδών αντικείμενα τύπου ομάδα, βάσει του τρόπου χειρισμού από το διερμηνευτή, όπως έχει αναφερθεί, τα επώνυμα και αυτά που είναι σε θέση (ανώνυμα), ο δείκτης εσωτερικά προσαρμόζεται να δείχνει το επώνυμο ή το ανώνυμο αντικείμενο. Όταν δείχνει το επώνυμο αντικείμενο γίνεται έλεγχος αν το επώνυμο υπάρχει, πριν την χρήση.

- Το επώνυμο αντικείμενο διαγράφεται όταν ο δημιουργός του τερματίσει την εκτέλεση.
- Το ανώνυμο επειδή το κρατάει δείκτης διαγράφεται όταν κανένας δείκτης δεν το δείχνει. Έτσι αν έχουμε ένα ανώνυμο αντικείμενο σε ένα δείκτη, είμαστε σίγουροι ότι το κρατάμε στη "ζωή". Υπάρχουν ανώνυμες ομάδες που έχουν μοναδικό δείκτη, και ανώνυμες με δυο ή περισσότερους δείκτες. Αυτή η διαφορά δεν φαίνεται στο κώδικα.

Εδώ ας μείνει στο νου ότι η ζωή του επώνυμου με δείκτη σχετίζεται με το αν εκτελείται ο χώρος που φτιάχτηκε, ενώ του ανώνυμου με δείκτη δεν εξαρτιέται από τίποτα άλλο παρά από τους δείκτες σε αυτό.

```
Κλάση Κάτι {
    Χ=10
    Τμήμα Εμφάνισε {
        Τύπωσε .Χ
    }
    Διαγραφή {
        Τύπωσε "Διαγράφτηκα"
    }
}
\\ επώνυμο αντικείμενο Μ
Μ=Κάτι()
\\ Δείκτες σε επώνυμο - δεν καλείται η διαγραφή.
Δ1->Μ
Δ11=Δείκτης(Μ)
\\ Δείκτες σε ανώνυμο
Δ2->(Μ)
Δ22=Δείκτης((Μ))
Δ3->Κάτι()
Δ33=Δείκτης(Κάτι())
Δ4=Δ3
\\ σύνολο πραγματικών δεικτών 4, θα δούμε τέσσερα Διαγράφτηκα
\\ συν ένα για το επώνυμο Μ, που δώσαμε με εντολή Καθαρό
Καθαρό Μ
```

***Πρόγραμμα 47**: Χρήση Δεικτών και του μέλους Διαγραφή*

Στο παράδειγμα φαίνεται και η λειτουργία Διαγραφή που καλείται σε διαγραφή ομάδας, αυτόματα σε δείκτες σε ανώνυμη ομάδα, ή με εντολή δική μας με τη Καθαρό. Όπου μπήκαν παρενθέσεις στο Μ στη κλήση της Δείκτης() είτε στη μορφή με συνάρτηση είτε στη μορφή με τελεστή (είναι το ίδιο), έκαναν την εσωτερική συνάρτηση να εκτελέσει έκφραση και έτσι να πάρει αντίγραφο αντικειμένου.

Στο παράδειγμα παραπάνω δεν φάνηκε πως χρησιμοποιούμε τις ιδιότητες και τις μεθόδους. Ας δούμε ένα μικρό παράδειγμα:

```
Κλάση Κάτι {
    Χ=10
    Τμήμα Εμφάνισε {
        Τύπωσε .Χ
    }
}
Μ->Κάτι()
```

```
M=>X+=500
M=>Εμφάνισε

Για M {
    .X+=5000
    M=>X+=1000
    .Εμφάνισε
}
```

***Πρόγραμμα 48**: Χρήση Μεθόδων και Ιδιοτήτων σε Ομάδα με Δείκτη*

Δείτε το παχύ βέλος =>, αντί να βάλουμε την τελεία, ο δείκτης έχει αυτή την παραξενιά, θέλει το βέλος. Υπάρχουν κάποιες εντολές που δεν δέχονται το M=>ΚάτιΤις γιατί αυτό έχει μια ιδιαίτερη λειτουργία. Στη περίπτωση αυτή χρησιμοποιούμε την Για αντικείμενο { }. Αυτή η εντολή μπορεί να πάρει μια σειρά αντικείμενα (μέχρι δέκα στο σύνολο των φωλιασμένων Για αντικείμενο σε ένα τμήμα ή συνάρτηση). Το πρώτο αντικείμενο χρησιμοποιεί μια τελεία, το δεύτερο δύο κ.ο.κ. Εσωτερικά ο διερμηνευτής έχει ανοίξει το αντικείμενο, το έχει κάνει επώνυμο. Αυτό σημαίνει ότι αν θέλουμε μπορούμε να περάσουμε με αναφορά ένα μέλος ή και το ίδιο με το Αυτό και το ..Αυτό για το δεύτερο κ.ο.κ. Δηλαδή χωρίς να γνωρίζουμε το όνομα που αποδίδει ο διερμηνευτής χειριζόμαστε τα αντικείμενα πίσω από τους δείκτες. Η Για αντικείμενο ανοίγει και άλλα αντικείμενα κλειστά που δεν έχουν δείκτες, όπως σε πίνακα ή σε κατάσταση. Να γιατί στο παράδειγμα μέσα στην Για M { } χρησιμοποιούμε τις τελείες. Η εντολή M=>X+=1000 χρησιμοποιεί το M μέσα στη Για. Τη δεδομένη στιγμή γνωρίζει ο διερμηνευτής ότι είναι ανοικτό το αντικείμενο και χρησιμοποιεί το δείκτη ως δείκτη σε επώνυμη ομάδα.

Έχουμε δει σε προηγούμενο παράδειγμα όπου βάζουμε τρία αντικείμενα συγκεκριμένου τύπου σε ένα αντικείμενο. Αυτά τα αντικείμενα ήταν ανοικτά αντικείμενα, χωρίς δείκτη. Σε μια υποθετική άλλη περίπτωση με τρία εσωτερικά αντικείμενα με δείκτη, τα αντικείμενα θα είναι συνδεδεμένα με το αντικείμενο και ο τύπος τους μπορεί να διαφοροποιείται. Αυτή είναι μια δυνατότητα που έχει πλεονεκτήματα όταν θέλουμε να έχουν όλα έναν βασικό τύπο, αλλά να διαφοροποιούνται. Θα δούμε τέτοια αντικείμενα στα προγραμματιστικά μοτίβα.

Ένας άλλος λόγος που χρησιμοποιούμε δείκτες, είναι για να φτιάξουμε δομές δεδομένων, τα λεγόμενα δένδρα ή σε πιο γενική μορφή τους γράφους.

Προγραμματιστικά Μοτίβα Αντικειμένων

Τα σχεδιαστικά μοτίβα δημοσιεύτηκαν από τη λεγόμενη συμμορία των τεσσάρων (Gang of Four), στο έργο "Design Patterns: Elements of Reusable Object-Oriented Software" (1994), των Erich Gamma, Richard Helm, Ralph Johnson και John Vlissides (Ελληνοαμερικάνος, πέθανε νέος 44 ετών από καρκίνο στον εγκέφαλο, δέκα χρόνια μετά τη συγγραφή του έργου). Το έργο αυτό έφερε τον αντικειμενοστραφή προγραμματισμό σε άλλο επίπεδο. Αυτό το βιβλίο έδωσε δυο μεγάλης αξίας προτάσεις:

- Προγραμμάτισε με τη Διεπαφή όχι με την Υλοποίηση
- Η Σύνθεση προέχει της Κληρονομικότητας

Το πρώτο μας λέει να σκεφτόμαστε το πώς φαίνεται στο χρήστη ένα αντικείμενο, όχι στο πώς έχει γραφτεί. Αυτό που φαίνεται λέγεται διεπαφή και περιέχει τα δημόσια μέλη του, τις μεθόδους και τις ιδιότητες που μπορούμε να χειριστούμε.

Το δεύτερο μας λέει ότι καλύτερα ένα αντικείμενο να έχει άλλα αντικείμενα αντί να κληρονομεί από άλλα αν είναι δυνατόν. Με αυτό το τρόπο τα εσωτερικά αντικείμενα γίνονται ιδιωτική υπόθεση του αντικειμένου που τα περιέχει, και η διεπαφή αυτού καθορίζεται μόνο από αυτό.

Ακολουθεί μια λίστα με τα μοτίβα με τις αγγλικές ονομασίες. Κάθε ονομασία έχει ένα σύνδεσμο σε μια σελίδα στο georgekarras.blogspot.gr τον επίσημο τόπο της M2000, όπου ο κώδικας παράδειγα είναι με τις αγγλικές εντολές της γλώσσας. Σε αυτό το άρθρο θα δούμε τα ίδια προγράμματα με ελληνικές εντολές και με επεξηγήσεις.

Διαβάζοντας τα προγραμματιστικά μοτίβα έχουμε μια ιδέα για το τι μπορεί να γίνει με τον αντικειμενοστραφή προγραμματισμό.

Υπάρχουν τρεις ενότητες μοτίβων, όπως ακολουθούν. Σε κάθε ενότητα το αγγλικό όνομα περιέχει σύνδεσμο που οδηγεί στο τόπο της M2000 και ειδικότερα σε αναρτήσεις με το αντίστοιχο πρόγραμμα επίδειξης του κάθε μοτίβου.

Μοτίβα Δημιουργίας

Singleton Pattern, Factory Method Pattern, Factory Pattern, Abstract Factory Pattern, Builder Pattern, Prototype Pattern, Pool Pattern

Μοτίβα Συμπεριφοράς

Memento Pattern, Mediator Pattern, Observer Pattern, Null Object Pattern, Visitor Pattern, Interpreter Pattern, Iterator Pattern, Strategy Pattern, Command Pattern, State Pattern Template Method Pattern, Chain of Responsibility Pattern

Μοτίβα Δομικά

Adapter Pattern, Bridge Pattern, Composite Pattern, Decorator Pattern, Flyweight Pattern, Facade Pattern, Proxy Pattern

Μοτίβο Singleton

Σκοπός αυτού του μοτίβου είναι να δείξει πώς μπορούμε ένα αντικείμενο να είναι μοναδικό παρόλο που το αντιγράφουμε! Το παράδειγμα έχει δυο μέρη. Το πρώτο μέρος έχει το τμήμα CheckSingleton. Σε αυτό το τμήμα φτιάχνουμε μια γενική λάμδα συνάρτηση Ένα. Θέλουμε αυτή η συνάρτηση να είναι ο κατασκευαστής του μοναδικού αντικειμένου.

Το αντικείμενό μας παίρνει μια τιμή και την καταχωρεί σε μια ιδιωτική μεταβλητή την μνήμη. Η μεταβλητή αυτή δέχεται έναν πίνακα ενός στοιχείου, κρατώντας ένα δείκτη στο πίνακα. Οι δείκτες σε πίνακα έχουν τελεστές που εφαρμόζονται σε όλα τα στοιχεία. Έτσι έχουμε μια δημόσια μέθοδο, την Αύξησε που εκτελεί τον τελεστή += στο δείκτη Αυτό.μνήμη. Μέσα στη λάμδα Ένα δημιουργούμε ένα επώνυμο αντικείμενο, το Signleton. Δίνουμε ως τύπο το Ένα. Δίνουμε μια ειδική μέθοδο τη Θέσε που χρησιμοποιείται στην εκχώρηση τιμής. Επειδή δεν θέλουμε να γίνεται

εκχώρηση πετάμε το αντικείμενο που υποτίθεται εκχωρούν στο τρέχον. Το αντικείμενο γυρνάει τιμή τη πρώτη τιμή του πίνακα στο δείκτη Αυτό.μνήμη. Η λάμδα Ένα επιστρέφει δείκτη σε αντικείμενο ομάδα. Ακολουθούν σημειώσεις στο πρόγραμμα.

Η συνάρτηση Ομάδα() γυρνάει μια ομάδα, και εδώ επειδή η ομάδα Signleton επιστρέφει τιμή, που δεν είναι ομάδα, με τη χρήση αυτής της συνάρτησης θα πάρουμε ένα αντίγραφο, από το οποίο ζητάμε ένα δείκτη και τον γράφουμε στη μ που είναι ένα κλείσιμο στη λάμδα. Σκοπός μας είναι αυτό να γίνει τη πρώτη φορά που ο μ είναι Null (είναι επίσης Signleton, μια ομάδα που γνωρίζει ο διερμηνευτής, και κάθε φορά η Δείκτης() χωρίς όρισμα ή το ->0& επιστρέφει δείκτη στο Null, με τύπο Null και Μηδενικός (η ελληνική ονομασία). Και τα δύο ονόματα ισχύουν για το μηδενικό αντικείμενο. Η M2000 αντί να έχει δείκτη με τιμή μηδέν, έχει δείκτη σε ένα μοναδικό αντικείμενο, και έτσι ελέγχουμε με το τελεστή Είναι Τύπος όποτε χρειαστεί. Πράγματι στη Ένα, έχουμε μια δομή επιλογής με τη συνθήκη **μ είναι τύπος μηδενικός**. Όταν δημιουργείται η Ένα τότε παίρνει σε μια λίστα όλα τα κλεισίματα, και εδώ έχει ένα μόνο το μ.

Ο λόγος που τη **μνήμη** (μεταβλητή) την έχουμε με δείκτη, είναι απλός. Μπορούμε να έχουμε πραγματικά διαφορετικά αντικείμενα, που να δείχνουν με το ίδιο δείκτη στο ίδιο αντικείμενο, εδώ τον αυτόματο πίνακα (tuple). Η αντιγραφή ομάδας γίνεται ως ρηχή αντιγραφή σε πίνακες που ορίζονται με την Πίνακας, ενώ στους δείκτες σε πίνακα έχουμε αντιγραφή δείκτη. Όσα και να φτιάξουμε αντικείμενα, με την Ένα ως συνάρτηση ή με αντιγραφή από άλλο επώνυμο που έχει πάρει την SIngleton ομάδα, θα έχει τον ίδιο δείκτη για το tuple. Έτσι φαινομενικά θα φαίνεται ότι έχουμε το ίδιο αντικείμενο. Σε κάποια άλλη διαμόρφωση η μνήμη θα μπορούσε να έδειχνε με δείκτη ένα άλλο αντικείμενο. Και εδώ το μοτίβο Singleton θα δούλευε σωστά, αφού πάντα ο δείκτης θα αντιγράφεται!

```
Τμήμα CheckSingleton {
    Γενική Ένα=Λάμδα μ=Δείκτης() (μιαΤιμή=0)-> {
        Αν μ είναι τύπος μηδενικός τότε
            Ομάδα Singleton {
                Τύπος:Ένα
            Ιδιωτικό:
                μνήμη=(μιαΤιμή,)
            Δημόσιο:
                Τμήμα Αύξησε (χ) {
                    .μνήμη+=χ
                }
                Θέσε {Πέτα}
                Αξία {
                    =.μνήμη#Τιμή(0)
                }
            }
            \\ επειδή η ομάδα γυρνάει αξία, τραβάμε αντίγραφο με την Ομάδα()
            μ->Ομάδα(Singleton)
        Τέλος Αν
        =μ
```

```
    }
    \\ η Ενα() γυρίζει δείκτη σε ομάδα
    κ=Ένα(100)
    \\ ο δείκτης για να εκτελέσει τη μέθοδο Αξία χρειάζεται το Εκφρ() (έκφραση)
    Τύπωσε Έκφρ(κ)=100
    μ=Ένα()
    Τύπωσε Έκφρ(μ)=100
    Τύπωσε κ είναι μ = Αληθής
    Τύπωσε κ είναι τύπος Ένα = Αληθής
    \\ όταν έχουμε δείκτη βάζουμε => εκεί που θα βάζαμε μια τελεία
    κ=>Αύξησε 500
    Τύπωσε Έκφρ(κ)=600
    \\ αντίγραφο από δείκτη σε ομάδα σε νέα επώνυμη ομάδα ζ
    ζ=Ομάδα(κ)
    \\ η ζ είναι επώνυμη ομάδα δίνει άμεσα την Αξία, αλλά παίζει και με το Εκφρ(ζ)
    Τύπωσε Έκφρ(ζ)=600, ζ=600
    ζ.Αύξησε 1000
    Τύπωσε ζ=1600, Έκφρ(μ)=1600, Έκφρ(κ)=1600
    \\ για να μπει η ομάδα και όχι η τιμή της βάζουμε το Ομάδα(ζ), αντίγραφο της ζ
    Βάλε Ομάδα(ζ)
    \\ όπωε μπήκε στο σωρό τιμών βγαίνει στη νέα βήτα
    Διάβασε βήτα
    Βήτα.Αύξησε 1000
    Τύπωσε ζ=2600, Έκφρ(μ)=2600, Έκφρ(κ)=2600
    \\ ενώ το Ένα() γυρίζει δείκτη το Ομάδα δέλτα=Ένα() φτιάχνει επώνυμη ομάδα
    Ομάδα δέλτα=Ένα()
    \\ επειδή το δέλτα είναι επώνυμη χρησιμοποιούμε τη τελεία μεταξύ ονόματος και
μεθόδου
    δέλτα.Αύξησε 1000
    Τύπωσε ζ=3600, βήτα=3600, δέλτα=3600, Έκφρ(μ)=3600, Έκφρ(κ)=3600
    μ=>Αύξησε 400
    Τύπωσε ζ=4000, βήτα=4000, δέλτα=4000, Έκφρ(μ)=4000, Έκφρ(κ)=4000
    Κλάση άλφα {
        χ=100
    }
    \\ εδώ δίνουμε μια άλφα() για συγχώνευση
    \\ αλλά δεν γίνεται γιατί υπάρχει η μέθοδος Θέσε...
    \\ ...που πετάει την τιμή (το αντικείμενο)...
    \\ ...και δεν το αφήνει να συγχωνευθεί
    ζ=άλφα()
    Τύπωσε Έγκυρο(ζ.χ)=Ψευδής
    \\ ένας δείκτης όμως αλλάζει αντικείμενο με άλλο δείκτη
    κ->άλφα()
    Τύπωσε κ είναι τύπος άλφα
```

```
    Τύπωσε κ=>χ=100
    \\ βάζουμε τη λάμδα στο σωρό τιμών, ως αποτέλεσμα του τμήματος
    Βάλε Ένα
}
CheckSingleton
\\ εδώ διαβάζουμε τη τιμή του σωρού χωρίς να την πετάξουμε
Γενική Κάτι=ΤιμήΣωρού()
\\ τώρα την πετάμε
Πέτα
\\ το Κάτι είναι λάμδα συνάρτηση έτσι έχει και το Κάτι ως μεταβλητή
\\ και το Κάτι() ως συνάρτηση
\\ κάνουμε το μ ομάδα και όχι δείκτη σε ομάδα
Ομάδα μ=Κάτι()
Τύπωσε μ=4000
μ.Αύξησε 1000
Τύπωσε μ=5000
ζ=Κάτι()
ζ=>Αύξησε 1000
Τύπωσε Έκφρ(ζ)=6000, μ=6000
```

***Πρόγραμμα 49**: Μοτίβο Singleton*

Μοτίβο Factory Pattern

Αυτό είναι κάπως μεγάλο, και καλό είναι να το διαβάσει κανείς προς το τέλος. Μπήκε εδώ για να ακολουθηθεί η σειρά όπως εμφανίζονται τα αγγλικά μοτίβα.

Στο Factory Method σκοπός μας είναι να συνδέσουμε μια **Factory** συνάρτηση (ή και περισσότερες) που θα φτιάχνει αντικείμενα βάσει ενός τύπου αρχείου, για τα οποία η **Εφαρμογή**, μια άλλη κλάση δεν έχει γνώση περί τύπων εκτός του βασικού τύπου **Έγγραφο**. Αυτό σημαίνει ότι στην Εφαρμογή θα χρησιμοποιηθούν μόνο αυτά που έχει το Έγγραφο, οι μέθοδοι: **άνοιξε**, **σώσε** και **κλείσε**.

Στο παράδειγμα η κλάση Έγγραφο έχει τρεις μεθόδους που δίνουν λάθος αν χρησιμοποιηθούν. Με το τρόπο αυτό μιμούμαστε την αφηρημένη κλάση. Από την κλάση **Έγγραφο** και την κλάση **ΑπλόΚείμενο** θα φτιάξουμε τρεις κλάσεις, την **έγγραφοΙστού**, την **ΤοΈγγραφο** και την **έγγραφοPdf**. Οι τρεις κλάσεις θα δημιουργούν ένα αντικείμενο με τρεις τύπους, οι δυο εκ των οποίων θα είναι το Έγγραφο και το ΑπλόΚείμενο.

Η κλάση Εφαρμογή έχει μια δημόσια συνάρτηση που δίνει λάθος αν χρησιμοποιηθεί (είναι αφηρημένη), και μια ιδιωτική επίσης "αφηρημένη". Έτσι όταν φτιάχνουμε το αντικείμενο ηΕφαρμογή, δίνουμε δυο αντικείμενα, το ένα από την κλάση Εφαρμογή και το άλλο είναι ο ΣτιβαρόςΔημιουργός. Ο ΣτιβαρόςΔημιουργός είναι αυτός που αλλάζει τις αφηρημένες συναρτήσεις για να δουλέψει η Εφαρμογή. Η Εφαρμογή κρατάει μια λίστα εγγράφων, από οποιοδήποτε έγγραφο παράγει, το κομμάτι που έδωσε ο ΣτιβαρόςΔημιουργός. Δείτε ότι αυτή η ομάδα είναι φτιαγμένη με ορισμό Ομάδα. Δεν έχει τύπο, και δεν προσθέτει τύπο στο τελικό αντικείμενο. Η αλλαγή πετυχαίνεται στα έτοιμα αντικείμενα, όχι στις κλάσεις. Εδώ έχουμε κληρονομικότητα βάσει αντικειμένων.

Δείτε ότι η Εφαρμογή καταχωρεί τα έγγραφα με ένα νούμερο. Όταν θέλουμε να δουλέψουμε με ένα έγγραφο τότε ζητάμε το δείκτη του εγγράφου βάσει του αριθμού του. Μπορούμε να κρατάμε τον αριθμό του εγγράφου καθώς αυτό καταχωρείται στη λίστα του αντικειμένου, ή να ζητάμε βάσει του γνωστού αριθμού (κάνει έλεγχο αν υπάρχει το έγγραφο με τον αριθμό που ζητάμε).

Η κλάση Εφαρμογή δημιουργεί ένα αντικείμενο το οποίο γυρίζει τιμή. Αυτό γίνεται στο μέλος Αξία. Αυτή λειτουργεί σαν συνάρτηση, και αν πάρει τιμή>0 τότε κάνει όπως περιγράφτηκε παραπάνω έλεγχο και δίνει τιμή ή το μηδέν. Οι μεταβλητές Κείμενο_1 και Κείμενο_2 κρατάνε τον αριθμό του εγγράφου (1 και 2), και με τους αριθμούς αυτούς το Έγγραφο μας δίνει το αντικείμενο για να το χρησιμοποιήσουμε έξω από αυτό. Παίρνουμε το ΤοΚείμενο1 και το ΤοΚείμενο2 ως δείκτες. Αν δεν υπάρχουν τα αντικείμενα (πχ τα έχουμε κλείσει) τότε θα πάρουμε δείκτη στο ΑπλόΚείμενο. Κατόπιν θα δώσουμε ένα κείμενο με το **ΤοΚείμενο1=>βάλε_κείμενο "html text"**, όπου το παχύ βέλος => χρησιμοποιείται ως τελεία, επειδή έχουμε δείκτη, και καλούμε την βάλε_κείμενο με το κείμενο που θέλουμε. Κατόπιν θα καλέσουμε μια ρουτίνα η οποία ελέγχει αν το αντικείμενο είναι δείκτης στο ΑπλόΚείμενο. Όντως είναι και αυτό το κάνουμε εκεί γιατί θα χρησιμοποιήσουμε την διεπαφή της κλάσης ΑπλόΚείμενο,

Το αντικείμενο ηΕφαρμογή δεν γνωρίζει ότι στη λίστα του έχει αντικείμενα του τύπου ΑπλόΚείμενο. Το μόνο που γνωρίζει είναι να χρησιμοποιεί την διεπαφή Έγγραφο. Παρόλο που τα έγγραφα δεν έχουν κώδικα στις υλοποιήσεις του Έγγραφου, το πρόγραμμα λειτουργεί, και μπορούν αργότερα να προστεθούν στοιχεία.

Το πλεονέκτημα της Factory Method είναι ότι η κλάση Εφαρμογή μπορεί να δουλέψει για οποιοδήποτε άλλο πρόγραμμα στο οποίο το ΑπλόΚείμενο, και οι στιβαρές κλάσεις, όπως αυτές που θα κληρονομήσουν το Έγγραφο και το ΑπλόΚείμενο καθώς και ο ΣτιβαρόςΔημιουργός μπορούν να αλλάξουν για άλλους τύπους αρχείων, ακόμα και να αλλάξουν ονόματα.

Τα σταθερά στοιχεία του μοτίβου είναι η κλάση **Έγγραφο**, η κλάση **Εφαρμογή** και η απαίτηση για αλλαγή της *Factory Method* **ΠάρεΤύπο$()** και **ΦτιάξεΈγγραφο()**, Η συμπεριφορά του αντικειμένου που δίνει η κλάση Εφαρμογή δεν αλλάζει. Δείτε ότι η ΠάρεΤύπο$() είναι ιδιωτική.

```
Κλάση Έγγραφο {
   Τμήμα άνοιξε {
      Λάθος "Δεν έχει υλοποιηθεί ακόμα "+όνομα.τμήματος$
   }
   Τμήμα σώσε (όνομαΑρχείου$) {
      Λάθος "Δεν έχει υλοποιηθεί ακόμα "+όνομα.τμήματος$
   }
   Τμήμα κλείσε {
      Λάθος "Δεν έχει υλοποιηθεί ακόμα "+όνομα.τμήματος$
   }
}
Κλάση ΑπλόΚείμενο {
   ιδιότητα βρώμικο {Αξία}=Ψευδής
   ιδιότητα κείμενο$ {Αξία}=""
```

```
    Τμήμα βάλε_κείμενο (ένα_κείμενο$) {
        .[βρώμικο] <= .[κείμενο]$<>ένα_κείμενο$ ή .[βρώμικο]
        Άλλαξε .[κείμενο]$, ένα_κείμενο$
    }
}
Κλάση έγγραφοΙστού ως ΑπλόΚείμενο ως Έγγραφο{
    Τμήμα άνοιξε {
    }
    Τμήμα σώσε (όνομαΑρχείου$) {
    }
    Τμήμα κλείσε {
    }
}
Κλάση ΤοΈγγραφο ως ΑπλόΚείμενο ως Έγγραφο {
    Τμήμα άνοιξε {
    }
    Τμήμα σώσε (όνομαΑρχείου$) {
    }
    Τμήμα κλείσε {
    }
}
Κλάση έγγραφοPdf ως ΑπλόΚείμενο ως Έγγραφο {
    Τμήμα άνοιξε {
    }
    Τμήμα σώσε (όνομαΑρχείου$) {
    }
    Τμήμα κλείσε {
    }
}


Κλάση Εφαρμογή {
ιδιωτικό:
    αρ_εγγράφου=0, έγγραφα=Λίστα
    Συνάρτηση ΠάρεΤύπο$(όνομαΑρχείου$) {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα "+όνομα.τμήματος$
    }
Δημόσιο:
    Αξία (νούμερο=0){
        Αν νούμερο=0 Τότε
            =.αρ_εγγράφου
        Αλλιώς.Αν Υπάρχει(.έγγραφα, νούμερο) Τότε
            =νούμερο
        Τέλος Αν
```

```
    }
    Συνάρτηση εξαγωγήΔείκτη(νούμερο) {
        Αν Υπάρχει(.έγγραφα, νούμερο) Τότε
            =Εκφρ(.έγγραφα)
        Αλλιώς
            =Δείκτης() ' Μηδενική Ομάδα
        Τέλος Αν
    }
    Τμήμα ΠέταΈγγραφο (νούμερο) {
        Αν Υπάρχει(.έγγραφα, νούμερο) Τότε delete .έγγραφα, νούμερο
    }
    Συνάρτηση ΦτιάξεΈγγραφο(τύπος$){
        Λάθος "Δεν έχει υλοποιηθεί ακόμα "+όνομα.τμήματος$
    }
    Τμήμα ΝέοΈγγραφο(τύπος$){
        Έγγραφο->(.ΦτιάξεΈγγραφο(τύπος$))
            Αν Έγγραφο Είναι Τύπος Έγγραφο Τότε
            .αρ_εγγράφου++
            Προσθήκη .έγγραφα, .αρ_εγγράφου:=Έγγραφο
            Έγγραφο=>άνοιξε
        Αλλιώς
            Λάθος "Λάθος τύπος αρχείου"
        Τέλος Αν
    }
    Τμήμα ΆνοιξεΈγγραφο(όνομαΑρχείου$) {
        Έγγραφο=Δείκτης(.ΦτιάξεΈγγραφο(.ΠάρεΤύπο$(όνομαΑρχείου$)))
        Αν Έγγραφο Είναι Τύπος Έγγραφο Τότε
            .αρ_εγγράφου++
            Προσθήκη .έγγραφα, .αρ_εγγράφου:=Έγγραφο
            Έγγραφο=>άνοιξε
        Αλλιώς
            Λάθος "Λάθος τύπος αρχείου"
        Τέλος Αν
    }
}
Ομάδα ΣτιβαρόςΔημιουργός {
ιδιωτικό:
    Συνάρτηση Τελική ΠάρεΤύπο$(όνομαΑρχείου$) {
        όνομαΑρχείου$=πεζ$(όνομαΑρχείου$)
        Αν όνομαΑρχείου$ ~ "*.html" Τότε
            ="html"
        Αλλιώς.Αν όνομαΑρχείου$ ~ "*.odt" Τότε
            ="ιδιωτικός"
        Αλλιώς.Αν όνομαΑρχείου$ ~ "*.pdt" Τότε
```

```
            ="pdf"
        Τέλος Αν
    }
Δημόσιο:
    Συνάρτηση Τελική ΦτιάξεΈγγραφο(τύπος$){
        Αν τύπος$="html" Τότε
            =έγγραφοΙστού()
        Αλλιώς.Αν τύπος$="ιδιωτικός" Τότε
            =ΤοΈγγραφο()
        Αλλιώς.Αν τύπος$="pdf" Τότε
            =έγγραφοPdf()
        Αλλιώς
            Λάθος "Δεν υπάρχει τέτοιος "+τύπος$
        Τέλος Αν
    }
}
ηΕφαρμογή = Εφαρμογή() με ΣτιβαρόςΔημιουργός
ηΕφαρμογή.ΝέοΈγγραφο "html"
Κείμενο_1=ηΕφαρμογή()
ηΕφαρμογή.ΝέοΈγγραφο "pdf"
Κείμενο_2=ηΕφαρμογή()
ΤοΚείμενο1=Αν( Κείμενο_1<>0-> ηΕφαρμογή.εξαγωγήΔείκτη(Κείμενο_1),
ΑπλόΚείμενο())
ΤοΚείμενο1=>βάλε_κείμενο "html text"
Εμφάνισε(ΤοΚείμενο1)
ΤοΚείμενο2=Αν( Κείμενο_2<>0-> ηΕφαρμογή.εξαγωγήΔείκτη(Κείμενο_2),
ΑπλόΚείμενο())
ΤοΚείμενο2=>βάλε_κείμενο "pdf text"
Εμφάνισε(ΤοΚείμενο2)


Τέλος
Ρουτίνα Εμφάνισε(Έγγραφο ως *ΑπλόΚείμενο)
    Τύπωσε "βρώμικο: ";Έγγραφο=>βρώμικο=Αληθής
    Τύπωσε "Κείμενο : ";Έγγραφο=>κείμενο$
Τέλος Ρουτίνας
```

***Πρόγραμμα 50**: Μοτίβο Factory Pattern*

Μοτίβο Factory

Είδαμε το Factory Method, και τώρα θα δούμε το πιο απλό Factory. Ένα αντικείμενο Παραγωγός παράγει βάσει κωδικών άλλα αντικείμενα.

Μοτίβο Factory με καταχωρημένες κλάσεις

Η ΠαραγωγήΠροϊόντος είναι ο Παραγωγός μας ή Factory. Σε αυτόν υπάρχει η γνώση των προϊόντων που θα δημιουργήσει, βάσει μιας σειράς κωδικών, 101 και 102. Μπορούμε να δώσουμε τον αριθμό ή το όνομα που αντιστοιχεί στον αριθμό (είναι μια σταθερά). Η συνάρτηση δημιούργησεΠροϊόν δέχεται έναν αριθμό που πρέπει να περιέχεται στη λίστα αριθμών της απαρίθμησης ΓνωστάΠροϊόντα. Αν ο κωδικός δεν υπάρχει θα πάρουμε λάθος εδώ, το οποίο το ελέγχουμε με την Δες ΟΚ {} και δίνουμε τον Μηδενικό τύπο.

```
Κλάση Προϊόν {

}
Κλάση ΠροϊόνΈνα ως Προϊόν {

}
Κλάση ΠροϊόνΔύο ως Προϊόν {

}
Ομάδα ΠαραγωγήΠροϊόντος {
    Απαρίθμηση ΓνωστάΠροϊόντα {Κωδ1=101, Κωδ2=102}
    Συνάρτηση δημιούργησεΠροϊόν {
       Δες οκ {
           Διάβασε κωδικός ως .ΓνωστάΠροϊόντα
           Αν κωδικός=Κωδ1 Τότε
              =ΠροϊόνΈνα()
           Αλλιώς.Αν κωδικός=Κωδ2 Τότε
              =ΠροϊόνΔύο()
           Τέλος Αν
       }
       Αν όχι Οκ τότε =Δείκτης() ' Μηδενικός τύπος Ομάδας
    }
}
Π1=ΠαραγωγήΠροϊόντος.δημιούργησεΠροϊόν(101)
Τύπωσε Π1 είναι τύπος Προϊόν
Τύπωσε Π1 είναι τύπος ΠροϊόνΈνα
Π2=ΠαραγωγήΠροϊόντος.δημιούργησεΠροϊόν(ΠαραγωγήΠροϊόντος.Κωδ2)
Τύπωσε Π2 είναι τύπος Προϊόν
Τύπωσε Π2 είναι τύπος ΠροϊόνΔύο
Π3=ΠαραγωγήΠροϊόντος.δημιούργησεΠροϊόν(0)
Τύπωσε Π3 είναι τύπος Μηδενικός
```

***Πρόγραμμα 51**: Μοτίβο Factory - Παράδειγμα 1*

Μοτίβο Factory με δυναμική καταχώρηση αντικειμένων

Εδώ έχουμε μια παραλλαγή, όπου ο Παραγωγός έχει τρόπο να καταχωρεί αντικείμενα με μια μέθοδο Καταχώρησε. Ότι κάναμε στο προηγούμενο με τη διαφορά ότι δίνουμε τις καταχωρήσεις με ορίσματα τον κωδικό, ένα αλφαριθμητικό πχ "101" και "102". Κατά την ετοιμασία των αντικειμένων καταγράφεται και ο αριθμός σειράς.

Το ενδιαφέρον εδώ είναι ότι καταχωρούμε αντικείμενα που το καθένα φτιάχνει ένα άλλο αντικείμενο. Δείτε την Ομάδα ΠροϊόνΈνα. Αυτή η ομάδα έχει μέλος μια κλάση, το ΠροϊόνΈνα το οποίο κληρονομεί από τη κλάση Προϊόν. Αυτή η ομάδα επιστρέφει τιμή (δέχεται και ένα όρισμα, το νούμερο για τον αριθμό σειράς), Οι κλάσεις στις ομάδες δεν είναι γενικές. Αυτό δεν τις χαλάει να φτιάχνονται κληρονομώντας είτε από άλλη κλάση εντός της ομάδας είτε από μια γενική. Η Κλάση Προϊόν είναι γενική (δεν είναι μέλος Ομάδας).

Οι Ομάδες σε αντίθεση με τις Κλάσεις δημιουργούν άμεσα αντικείμενο. Εδώ δεν έχουμε δώσει τύπο. Οι κλάσεις δημιουργούν αυτόματα τύπο, και αν κληρονομούν από άλλες βάζουν και τους τύπους των άλλων κλάσεων.

Ο παραγωγός μας έχει ενσωματώσει και τους ορισμούς των κλάσεων, όχι απλά αντικείμενα όπως το προηγούμενο παράδειγμα!

```
Κλάση Προϊόν {
Ιδιωτικό:
    αριθμόςΣειράς
}
Ομάδα ΠροϊόνΈνα {
    Κλάση ΠροϊόνΈνα ως Προϊόν {
    Κλάση:
            Τμήμα ΠροϊόνΈνα (.αριθμόςΣειράς) {
                Τύπωσε "Προϊόν Ένα,"+Γραφή$(.αριθμόςΣειράς)
            }
    }
    Αξία (νούμερο) {
        =.ΠροϊόνΈνα(νούμερο)
    }
}
Ομάδα ΠροϊόνΔύο {
    Κλάση ΠροϊόνΔύο ως Προϊόν {
    Κλάση:
            Τμήμα ΠροϊόνΔύο (.αριθμόςΣειράς) {
                Τύπωσε "Προϊόν Δύο,"+Γραφή$(.αριθμόςΣειράς)
            }
    }
    Αξία (νούμερο) {
        =.ΠροϊόνΔύο(νούμερο)
```

```
    }
}
Ομάδα ΠροϊόνΠαραγωγής {
ιδιωτικό:
     ΚαταχώρησηΠροιόντων=Λίστα
     αριθμόςΣειράς=10001
Δημόσιο:
    Τμήμα Καταχώρησε (κωδ$, κάτι ως Ομάδα) {
        κωδ$=πεζ$(κωδ$)
        Αν Υπάρχει(.ΚαταχώρησηΠροιόντων, κωδ$) Τότε λάθος "Χρησιμοποιείται ο
κωδικός"
        Προσθήκη .ΚαταχώρησηΠροιόντων, κωδ$:=Ομάδα(κάτι)
    }
    Function ΦτιάξεΠροϊόν(κωδ$) {
        κωδ$=πεζ$(κωδ$)
        Αν Υπάρχει(.ΚαταχώρησηΠροιόντων, κωδ$) Τότε
            Κλάση=εκφρ(.ΚαταχώρησηΠροιόντων)
            =Κλάση(.αριθμόςΣειράς)
            .αριθμόςΣειράς++
        Αλλιώς
            =Δείκτης() ' Μηδενικός Τύπος
        Τέλος Αν
    }
}
ΠροϊόνΠαραγωγής.Καταχώρησε "101", Ομάδα(ΠροϊόνΈνα)
ΠροϊόνΠαραγωγής.Καταχώρησε "102", Ομάδα(ΠροϊόνΔύο)
Π1=ΠροϊόνΠαραγωγής.ΦτιάξεΠροϊόν("101")
Τύπωσε Π1 είναι τύπος Προϊόν
Τύπωσε Π1 είναι τύπος ΠροϊόνΈνα
Π2=ΠροϊόνΠαραγωγής.ΦτιάξεΠροϊόν("102")
Τύπωσε Π2 είναι τύπος Προϊόν
Τύπωσε Π2 είναι τύπος ΠροϊόνΔύο
```

***Πρόγραμμα 52**: Μοτίβο Factory - Παράδειγμα 2*

Μοτίβο Abstract Factory

Ο αφηρημένος Παραγωγός είναι ένα μοτιβος που σε μια μέθοδο μιαΛειτουργίεια ενός αντικειμένου τύπου Δημιουργός, φτιάχνουμε ένα αντικείμενο τύπου Προϊόν. Μπορούμε να φτιάξουμε με βάση τον Δημιουργό άλλες κλάσεις, και με βάση το Προϊόν άλλα προϊόντα που θα αντικαθιστούν το Προϊόν στην μιαΛειτουργία.

Ο Δημιουργός έχει μια λειτουργία (μια μέθοδο **μιαΛειτουργία**) που καλεί μια συνάρτηση **ΜέθοδοςΠαραγωγός()**, η οποία δεν έχει υλοποιηθεί. Η υλοποίηση θα γίνει από τον εκάστοτε Στιβαρό Δημιουργό. Παράλληλα έχουμε τη κλάση Προϊόν, από την οποία κληρονομούν οι κλάσεις ΣτιβαρόΠροϊόν και ΣτιβαρόΠροϊόν2. Για αυτά τα δυο προϊόντα φτιάχνουμε δυο δημιουργούς που

κληρονομούν από τη κλάση Δημιουργός, τα ΣτιβαρόςΔημιουργός και ΣτιβαρόςΔημιουργός2. Όλες οι κλάσεις είναι γενικές οπότε είναι φανερές στο τμήμα Πελάτης. Στο συγκεκριμένο τμήμα φτιάχνουμε το αντικείμενο Δημιουργός από την κλάση ΣτιβαρόςΔΗμιουργός και τη Δημιουργός2 από τη ΣτιβαρόςΔημιουργός2. Σε κάθε περίπτωση καλούμε τη μιαΛειτουργία, η οποία καλεί τον ξεχωριστή ΜέθοδοςΠαραγωγός() και δημιουργεί μέσα στη μίαΛειτουργία το προϊόν. Η χρήση των προϊόντων γίνεται μέσα στην μιαΛειτουργία του δημιουργού, όπου το προϊόν είναι Προϊόν εκτός από κάποιο από τα δύο στιβαρά προϊόντα.

```
Κλάση Προϊόν {
Κλάση:
    Τμήμα Προϊόν {
        Αναφορά "Ένα νέο Προϊόν"
    }
}
Κλάση ΣτιβαρόΠροϊόν ως Προϊόν {
Κλάση:
    Τμήμα ΣτιβαρόΠροϊόν {
        .Προϊόν
        Αναφορά "Ένα νέο ΣτιβαρόΠροϊόν είναι ένα Προϊόν"
    }
}
Κλάση ΣτιβαρόΠροϊόν2 ως Προϊόν {
Κλάση:
    Τμήμα ΣτιβαρόΠροϊόν2 {
        .Προϊόν
        Αναφορά "Ένα νέο ΣτιβαρόΠροϊόν2 είναι ένα Προϊόν"
    }
}
Κλάση Δημιουργός {
    Τμήμα μιαΛειτουργία {
        Αναφορά "Μια λειτουργία του Δημιουργού - χρησιμοποιεί τη ΜέθοδοςΠαραγωγός"
        Προϊόν = .ΜέθοδοςΠαραγωγός()
    }
    Συνάρτηση ΜέθοδοςΠαραγωγός {
        Λάθος "Δεν έχει ακόμα υλοποιηθεί "+όνομα.τμήματος$
    }
Κλάση:
    Τμήμα Δημιουργός {
        Αναφορά "Ένας νέος Δημιουργός"
    }
}
Κλάση ΣτιβαρόςΔημιουργός ως Δημιουργός {
    Συνάρτηση Τελική ΜέθοδοςΠαραγωγός {
        Τύπωσε "ΜέθοδοςΠαραγωγός 001"
        =ΣτιβαρόΠροϊόν()
    }
Κλάση:
    Τμήμα ΣτιβαρόςΔημιουργός {
        .Δημιουργός
        Αναφορά "Ο νέος ΣτιβαρόςΔημιουργός είναι ένας Δημιουργός"
```

```
    }
}
Κλάση ΣτιβαρόςΔημιουργός2 ως Δημιουργός {
    Συνάρτηση Τελική ΜέθοδοςΠαραγωγός {
        Τύπωσε "ΜέθοδοςΠαραγωγός 002"
        =ΣτιβαρόΠροϊόν2()
    }
Κλάση:
    Τμήμα ΣτιβαρόςΔημιουργός2 {
        .Δημιουργός
        Αναφορά "Ο νέος ΣτιβαρόςΔημιουργός2 είναι ένας Δημιουργός"
    }
}
Τμήμα Πελάτης {
    Αναφορά 2, "Ο ΣτιβαρόςΔημιουργός βάζει μια νέα ΜέθοδοςΠαραγωγός στη Κλάση
Δημιουργός"
    Δημιουργός = ΣτιβαρόςΔημιουργός()
    Δημιουργός.μιαΛειτουργία
    Δημιουργός2= ΣτιβαρόςΔημιουργός2()
    Δημιουργός2.μιαΛειτουργία
}
Πελάτης
```

***Πρόγραμμα 53**: Μοτίβο Abstract Factory*

Μοτίβο Builder

Το μοτίβο Builder ή Κτίστης, χρησιμοποιείται όταν θέλουμε για ένα τύπο αντικειμένου εισαγωγής X τύπου να εφαρμόσουμε ένα αλγόριθμο Y που θα δώσει ένα συγκεκριμένο τύπο Z. Ο τύπος Z είναι ο τύπος του Κτίστη. Μπορούμε να έχουμε διαφορετικούς τύπους εισαγωγής αλλά σε κάθε περίπτωση θέλουμε το τελικό αντικείμενο να είναι ο τύπος του Κτίστη ή άλλο αντικείμενο που να είναι τύπος Κτίστη (αλλά να κάνει άλλη τελική μετατροπή). Το στοιχείο κλειδί εδώ είναι ότι ο αλγόριθμος Y είναι ένα αντικείμενο που κρατάει μια αναφορά στο Z (τον Κτίστη). Όταν η λειτουργία του Y τελειώσει τότε το αποτέλεσμα θα έχει κτιστεί στο Z και από εκεί θα πάρουμε το αποτέλεσμα. Σχηματικά $X \rightarrow Y(\&Z) \rightarrow Z$ όπου το X πάει στο Y που έχει την αναφορά στο Z και μας αφήνει το Z με το αποτέλεσμα (δεν το επιστρέφει, αλλά το αφήνει με αλλαγή κατάστασης).

Το παράδειγμα κάνει μετατροπή εγγράφου RTF σε κείμενο ASCII. Υποθέτουμε ότι το RTF κείμενο έχει στοιχεία το **c** που είναι ο χαρακτήρας, το **p** που είναι η αλλαγή παραγράφου, και επιστρέφει και ένα ακόμα στοιχείο, το τερματισμό του. Αυτή είναι η απλή κωδικοποίηση του RTF που εδώ θα μετατρέψουμε σε ASCII. Κάθε στοιχείο λέγεται token, και ένας μεταφραστής στοιχείων λέγεται parser. Αυτό το μοτίβο χρησιμοποιείται για μεταφραστές, parsers.

Στο παράδειγμα υπάρχει μια κλάση **Πελάτης** με μια μέθοδο **δημιουργία_ASCII_Κείμενο** που δέχεται ένα όρισμα του τύπου **Έγγραφο.** Ο τύπος Έγγραφο αναφέρεται σε μια κλάση Έγγραφο που για το παράδειγμα έχει ένα δοκιμαστικό κείμενο, και μια μέθοδο **πάρεΕπόμενοΣτοιχείο()**. Ο Πελάτης φτιάχνει το αντικείμενο **ascii_Κτίστης** από την κλάση **ASCII_Μετατροπέας**. Αυτή η κλάση είναι η υλοποίηση της κλάσης **Μετατροπέας_Κειμένου**. Φτιάχνουμε ένα ακόμα αντικείμενο το **RTF_Αναγνώστης** από την κλάση RTF_Αναγνώστης με όρισμα μια αναφορά στο

ascii_Κτίστης. Αν το αντικείμενο που θα δώσουμε δεν είναι του τύπου Μετατροπέας_Κειμένου θα πάρουμε λάθος. Μπορούμε να δώσουμε διαφορετικό μετατροπέα κειμένου αρκεί να είναι τύπου Μετατροπέας_Κειμένου. Ο RTF_Αναγνώστης έχει μια μέθοδο μετάφρασεRTF που θα κάνει την μετάφραση του εγγράφου από στοιχεία ενός RTF εγγράφου σε Ascii, επειδή του δώσαμε το αντικείμενο ascii_Κτίστης. Το αποτέλεσμα της μετάφρασης θα το πάρουμε από το ascii_Κτίστης με τη μέθοδο **ΠάρεΑποτέλεσμα$()** σε μια μεταβλητή ASCII_Κείμενο$, και αμέσως μετά θα το δείξουμε με την Αναφορά, η οποία εμφανίζει κείμενο με παραγράφους στη κονσόλα με δυνατότητα να σταματάει στα ¾ των γραμμών της κονσόλας και να περιμένει να πατήσουμε διάστημα ή πλήκτρο του ποντικιού για να συνεχίσει.

Το παράξενο της υλοποίησης εδώ είναι ότι περάσαμε μια αναφορά σε μια κλήση και αυτή έμεινε μετά την κλήση! Στην ουσία κρατήσαμε την ισχνή αναφορά για χρήση αργότερα. Η ισχνή αναφορά είναι ένα αλφαριθμητικό. Η Για αντικείμενο {} έχει μια παραλλαγή όπου αντί για αντικείμενο δίνουμε αλφαριθμητικό και αυτό το δέχεται ως ισχνή αναφορά σε αντικείμενο. Θα μπορούσε να υλοποιηθεί το πρόγραμμα με δείκτη στον Κτίστη. Το μπλοκ της Για {} χρησιμοποιείται για επανάληψη με την εντολή Κυκλικά όσο δεν έρχεται το EOF (αυτό το όνομα είναι συντομογραφία του End of File).

```
Κλάση Μετατροπέας_Κειμένου {
    Τμήμα μετατροπήΧαρακτήρα (χαρακτήρας) {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα "+όνομα.τμήματος$
    }
    Τμήμα μετατροπήΠαραγράφου {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα "+όνομα.τμήματος$
    }
}
Κλάση ASCII_Κείμενο$ {
Ιδιωτικό:
    τοΚείμενο$=""
Δημόσιο:
    Τμήμα πρόσθεσεΧαρακτήρα (χαρακτήρας$) {
        .τοΚείμενο$+=χαρακτήρας$
    }
    Αξία {
        =.τοΚείμενο$
    }
}
Κλάση ASCII_Μετατροπέας ως Μετατροπέας_Κειμένου {
Ιδιωτικό:
    ASCII_Κείμενο$ ASCII_ΚείμενοΟμάδα
Δημόσιο:
    Τμήμα Τελικό μετατροπήΧαρακτήρα(χαρακτήρας) {
        .ASCII_ΚείμενοΟμάδα.πρόσθεσεΧαρακτήρα Χαρ$(χαρακτήρας)
    }
```

```
    Τμήμα Τελικό μετατροπήΠαραγράφου {
        .ASCII_ΚείμενοΟμάδα.πρόσθεσεΧαρακτήρα {
        }
    }
    Συνάρτηση ΠάρεΑποτέλεσμα$ {
        =.ASCII_ΚείμενοΟμάδα$
    }
}
Κλάση Έγγραφο {
Ιδιωτικό:
    ΣτοιχείοΤέλους=0
    δοκιμαστικό$="ccpcc", θέση_χαρακτήρα=1
Δημόσιο:
    Συνάρτηση πάρεΕπόμενοΣτοιχείο() {
        Αν .θέση_χαρακτήρα>len(.δοκιμαστικό$) Τότε
            =.ΣτοιχείοΤέλους
        Αλλιώς
            =Κωδ(Μεσ$(.δοκιμαστικό$, .θέση_χαρακτήρα, 1))
            .θέση_χαρακτήρα++
        Τέλος Αν
    }
}
Κλάση RTF_Αναγνώστης {
Ιδιωτικό:
    κτίστης$
Δημόσιο:
    Τμήμα μετάφρασεRTF (Κείμενο_Εισαγωγής ως Έγγραφο) {
        Απαρίθμηση σημάδια {
            EOF=0
            CHAR=Κωδ("c")
            PARA=Κωδ("p")
        }
        Κάνε χαρακτήρας ως μακρύς
        For .κτίστης$ {
            χαρακτήρας=Κείμενο_Εισαγωγής.πάρεΕπόμενοΣτοιχείο()
            Αν χαρακτήρας=EOF Τότε Έξοδος Αλλιώς Κυκλικά
            Επίλεξε Με χαρακτήρας
            Με CHAR
                .μετατροπήΧαρακτήρα χαρακτήρας
            Με PARA
                .μετατροπήΠαραγράφου
            Τέλος Επιλογής
        }
        Τύπωσε "Τέλος Μετάφρασης"
```

```
    }
Κλάση:
    Τμήμα RTF_Αναγνώστης(.κτίστης$) {
        Ένωσε Ισχνή .κτίστης$ στη Κτίστης
        Αν Όχι Κτίστης είναι τύπος Μετατροπέας_Κειμένου Τότε Λάθος "Χρειάζομαι έναν
αντικείμενο τύπου Μετατροπέας_Κειμένου"
    }
}
Κλάση Πελάτης {
    Τμήμα δημιουργία_ASCII_Κείμενο(Κείμενο_Εισαγωγής ως Έγγραφο){
        ascii_Κτίστης=ASCII_Μετατροπέας()
        RTF_Αναγνώστης = RTF_Αναγνώστης(&ascii_Κτίστης)
        RTF_Αναγνώστης.μετάφρασεRTF Κείμενο_Εισαγωγής
        ASCII_Κείμενο$ = ascii_Κτίστης.ΠάρεΑποτέλεσμα$()
        Αναφορά ASCII_Κείμενο$
    }
}
Πελάτης=Πελάτης()
Κείμενο_Εισαγωγής=Έγγραφο()
Πελάτης.δημιουργία_ASCII_Κείμενο Κείμενο_Εισαγωγής
Τύπωσε "Αυτό είναι ένα παράδειγμα του μοτίβου Κτίστης (Κτίστης)"
```

***Πρόγραμμα 54**: Μοτίβο Builder*

Μοτίβο Prototype

Κάποιες φορές η δημιουργία ενός αντικειμένου μέσω μιας κλάσης, και του κατασκευαστή της, είναι χρονοβόρα σε σχέση με την αντιγραφή ενός αντικειμένου που ήδη έχει κατασκευαστή σε ένα άλλο αντικείμενο.

Το παράδειγμα δείχνει επακριβώς τη χρήση του μοτίβου: Υπάρχει η κλάση Ανάλυση_Στοιχείων. Όταν φτιάχνουμε ένα Σ1_Σύνολο με μια σειρά στοιχεία, υποθέτουμε ότι έχουμε τη χρονοβόρα διαδικασία (πχ τα στοιχεία ήρθαν από αρχείο, ή από το διαδίκτυο). Θέλουμε να εφαρμόσουμε διάφορους τρόπους ανάλυσης πάνω σε αυτά τα στοιχεία. Τα στοιχεία έχουν γραφεί σε ένα αντικείμενο σωρός, ακριβώς το ίδιο που χρησιμοποιείται στον τρέχον σωρό, και ειδικότερα έχουμε πάρει άμεσα το σωρό τιμών της κλάσης κατά τη κλήση. Στη Σ1_Σύνολο η ιδιωτική μεταβλητή στοιχεία κρατάει έναν δείκτη στο σωρό τιμών. Αν αντιγράψουμε την Σ1_Σύνολο θα αντιγράψουμε τον δείκτη, οπότε έχουμε μια μέθοδο την Αντίγραφο() που κάνει αντιγραφή του σωρού σε νέο σωρό. Όταν εκτελούμε μια ανάλυση ο σωρός πετάει τα στοιχεία του. Αν λοιπόν δίναμε ένα απλό αντίγραφο θα είχαμε πρόβλημα, γιατί δεν θα υπήρχαν στοιχεία! Όταν εκτελούμε μια ανάλυση δίνουμε ένα αντίγραφο, και αυτό το καταναλώνει η Ανάλυση. Δείτε όμως ότι χρησιμοποιούμε δυο κλάσεις την ΣτιβαρήΑνάλυση1 και την ΣτιβαρήΑνάλυση2 που κληρονομούν από την Ανάλυση. Η απλή Ανάλυση απλά μας δείχνει τα στοιχεία! Η ΣτιβαρήΑνάλυση1 βγάζει τη σούμα ενώ η ΣτιβαρήΑνάλυση1 βγάζει το μέγιστο. Δείτε ότι αρχική τιμή μέγιστου δώσαμε το -Άπειρο. Επίσης δείτε ότι και οι δυο στιβαρές αναλύσεις καλούν στον κατασκευαστή τους τον κατασκευαστή της κλάσης Ανάλυση. Επειδή οι κατασκευαστές είναι τμήματα, με την κλήση του επόμενου τμήματος

(εδώ του κατασκευαστή της κλάσης Ανάλυση) περνάμε τον τρέχον σωρό στο επόμενο. Οι κατασκευαστές δεν θα υπάρχουν στο τελικό αντικείμενο επειδή είναι μετά την ετικέτα Κλάση:

Η μέθοδος **ΠάρεΕπόμενο(&στοιχείο)** παίρνει μια μεταβλητή με αναφορά και διαβάζει από το σωρό στοιχεία το στοιχείο τραβώντας την τιμή στην κορυφή του σωρού. Η μέθοδος επιστρέφει τιμή Αληθής αν πράγματι η στοιχείο πήρε νέα τιμή.

Η μέθοδος **Προσθήκη_Στοιχείων** παίρνει το αντικείμενο Οποιοσδήποτε_Αναλυτής και επειδή είναι ίδιου τύπου (Ανάλυση_Στοιχείων) έχει πρόσβαση στην ιδιωτική μεταβλητή στοιχεία. Οπότε ανοίγει το δικό του σωρό .στοιχεία και βάζει τα στοιχεία από το Οποιοσδήποτε_Αναλυτή, και αυτή η λειτουργία αδειάζει τον σωρό στοιχείων του Οποιοσδήποτε_Αναλυτής. Επειδή ο Οποιοσδήποτε_Αναλυτής πέρασε με τιμή, θα διαγραφεί στην επιστροφή από το Προσθήκη_Στοιχείων. Αυτό που πετυχαίνουμε στο Προσθήκη_Στοιχείων είναι να γίνει μεταφορά στοιχείων που είναι ιδιωτικά, και μόνο αντικείμενα του τύπου Ανάλυση_Στοιχείων μπορούν να το κάνουν αυτό.

Στο παράδειγμα στο τελική φάση χρησιμοποιούμε δυο φορές την Προσθήκη_Στοιχείων για να βάλουμε δυο φορές τα στοιχεία! Τη δεύτερη φορά δεν δίνουμε αντίγραφο και βλέπουμε ότι πράγματι διαγράφτηκαν τα στοιχεία από το Σ1_Σύνολο.

```
Κλάση Ανάλυση_Στοιχείων {
ιδιωτικό:
    στοιχεία=Σωρός
Δημόσιο:
    Ομάδα Μέγεθος_Στοιχείων {
         Αξία {
              Ένωσε γονικό στοιχεία στο Α
              =Μήκος(Α)
         }
    }
    Τμήμα Προσθήκη_Στοιχείων (Οποιοσδήποτε_Αναλυτής ως Ανάλυση_Στοιχείων){
        Σωρός .στοιχεία {
            Σειρά ! Οποιοσδήποτε_Αναλυτής.στοιχεία
        }
    }
    Συνάρτηση ΠάρεΕπόμενο(&στοιχείο) {
        Σωρός .στοιχεία {
            Αν Κενό Τότε Έξοδος
            Διάβασε στοιχείο
            =Αληθής
        }
    }
    Συνάρτηση Αντίγραφο() {
        Αντίγραφο=Αυτό
        Αντίγραφο.στοιχεία<=Σωρός(.στοιχεία)
        =Αντίγραφο
    }
Κλάση:
```

```
    Τμήμα Ανάλυση_Στοιχείων {
        \\ το [] αφαιρεί τον τρέχον σωρό και δίνει το δείκτη σε αυτό που αφαίρεσε
        \\ στον τρέχον σωρό βάζει νέο σωρό
        .στοιχεία<=[]
    }
}

Κλάση Ανάλυση {
ιδιωτικό:
Ανάλυση_Στοιχείων Αναλυτής_Στοιχείων
Δημόσιο:
    Τμήμα Προσθήκη_Στοιχείων {
        .Αναλυτής_Στοιχείων.Προσθήκη_Στοιχείων
    }
    Τμήμα Κάνε_Ανάλυση {
        ι=0
        Ενώ .Αναλυτής_Στοιχείων.ΠάρεΕπόμενο(&ι)
            Τύπωσε ι,
        Τέλος Ενώ
        Τύπωσε
    }
Κλάση:
    Τμήμα Ανάλυση (Οποιοσδήποτε_Αναλυτής ως Ανάλυση_Στοιχείων) {
        .Προσθήκη_Στοιχείων Οποιοσδήποτε_Αναλυτής
    }
}


Κλάση ΣτιβαρήΑνάλυση1 ως Ανάλυση {
    Τμήμα Τελικό Κάνε_Ανάλυση {
        ι=0
        σούμα=0
        Ενώ .Αναλυτής_Στοιχείων.ΠάρεΕπόμενο(&ι)
            σούμα+= ι
        Τέλος Ενώ
        Τύπωσε "Σούμα = "; σούμα
    }
Κλάση:
    Τμήμα ΣτιβαρήΑνάλυση1 {
        .Ανάλυση
    }
}
Κλάση ΣτιβαρήΑνάλυση2 ως Ανάλυση {
    Τμήμα Τελικό Κάνε_Ανάλυση {
        ι=0
```

```
            Μέγιστο=-Άπειρο
            Ενώ .Αναλυτής_Στοιχείων.ΠάρεΕπόμενο(&ι)
                Αν ι>Μέγιστο Τότε Μέγιστο=ι
            Τέλος Ενώ
            Τύπωσε "Μέγιστο = "; Μέγιστο
        }
    Κλάση:
        Τμήμα ΣτιβαρήΑνάλυση2 {
            .Ανάλυση
        }
    }
    \\ Υποτίθεται ότι η δημιουργία του Σ1_Σύνολο είναι χρονοβόρα
    \\ ενώ το Σ1_Σύνολο.Αντίγραφο() είναι μια γρήγορη αντιγραφή
    Σ1_Σύνολο=Ανάλυση_Στοιχείων(100,200,150,100,400,200,100)
    Τύπωσε Σ1_Σύνολο.Μέγεθος_Στοιχείων=7
    Α1=ΣτιβαρήΑνάλυση1(Σ1_Σύνολο.Αντίγραφο())
    Α1.Κάνε_Ανάλυση  ' δείχνει 1250
    Α2=ΣτιβαρήΑνάλυση2(Σ1_Σύνολο.Αντίγραφο())
    Α2.Κάνε_Ανάλυση  ' δείχνει 400
    Α1.Προσθήκη_Στοιχείων Σ1_Σύνολο.Αντίγραφο()
    Α1.Προσθήκη_Στοιχείων Σ1_Σύνολο
    Τύπωσε Σ1_Σύνολο.Μέγεθος_Στοιχείων=0
    Α1.Κάνε_Ανάλυση  ' δείχνει 2500
```

***Πρόγραμμα 55**: Μοτίβο Prototype*

Μοτίβο Pool

Το μοτίβο Pool ή Μάζωμα είναι ένα αμφιλεγόμενη χρηστικότητας μοτίβο (δεν δίνεται από τη συμμορία των τεσσάρων). Στη βάση του κρατάει μια δεξαμενή (μάζωμα θα το ονομάζουμε εδώ) αντικειμένων. Ο αντίλογος για τέτοια χρήση δηλώνεται για γλώσσες που χρησιμοποιούν αυτόματο συλλέκτη ανενεργών αντικειμένων. Η Μ2000 δεν χρησιμοποιεί αυτόματο συλλέκτη, επειδή τα δικά της αντικείμενα διαγράφονται με χρήση μετρητή δεικτών. Έτσι αν ο δείκτης μηδενιστεί τότε το αντικείμενο διαγράφεται (είναι το κλασικό σύστημα στα COM αντικείμενα, σε αυτά στηρίζεται η Μ2000 εσωτερικά). Οι επώνυμες ομάδες ξέρουμε ότι διαγράφονται στο πέρας του μπλοκ που τα δημιούργησε (είτε είναι το τμήμα ή συνάρτηση είτε το Για αντικείμενο {}). Οι ανώνυμες χωρίς δείκτη θα διαγραφούν όταν αυτό που τις κρατάει διαγραφεί (όλες οι ομάδες έχουν δείκτη εσωτερικά αλλά η έκφραση δείκτης σε ομάδα είναι για το αν ο δείκτης είναι χρηστικός από τον προγραμματιστή, αν έχει εκδοθεί κατά μια έννοια).

Η χρήση του Μαζώματος είναι να συνδυάσει αντικείμενα με ένα περιορισμένο αριθμό πηγών. Όταν το αντικείμενο χρησιμοποιηθεί τότε η πηγή δίνεται πίσω στο μάζωμα. Η πηγή είναι και αυτό ένα αντικείμενο.

Το παράδειγμα είναι μεγάλο, αλλά δείχνει όλα τα στοιχεία του μοτίβου. Για το σκοπό του παραδείγματος έχουμε δυο κλάσεις στοιχείων ΤοΣτοιχείο και το ΤοΣτοιχείο1 καθώς και το Τίποτα.

Όλη η λογική είναι στην κλάση Τροφοδοσία, η οποία μας φτιάχνει το αντικείμενο Μάζωμα. Αυτό το αντικείμενο έχει τα παρακάτω:

Ιδιότητα: ΤελευταίοΛάθος είναι μια ομάδα που γυρίζει τιμή.

Μέθοδος: ΠάρεΈνα() είναι μια λάμδα συνάρτηση με κλείσιμο το πρώτο όνομα πηγής "a", η οποία μπορεί να δώσει ένα περιορισμένο αριθμό (6) και κάθε πηγή θα έχει το όνομα από το a, b… και επειδή δέχεται ένα όρισμα, μια Άλλη ομάδα, επιστρέφει μια συγχώνευση ομάδας (κληρονομικότητα βάσει αντικειμένων) των Άλλη και Πηγή.

Μέθοδος: ΒάλεΈνα είναι μια μέθοδος που παίρνει ένα αντικείμενο που πρέπει να είναι τουλάχιστον τύπου Πηγή, και κάνει το εξής: βρίσκει το όνομα πηγής πχ το "c" από μια λίστα και αφαιρεί την κρατημένη πηγή και την βάζει πάλι στο μάζωμα (είναι σωρός τιμών, εδώ χρησιμοποιείται ως FIFO).

Για το παράδειγμα υπάρχουν και δυο ακόμα μέθοδοι, για να βλέπουμε τι έχει η λίστα (άλλη ονομασία για την Κατάσταση Ειδών) και σωρός. Στο σωρό βλέπουμε με μια Peek τύπου συνάρτηση που διαβάζει όχι μόνο από την κορυφή αλλά από όπου θέλουμε, και εδώ γίνεται με έναν Επαναλήπτη (οι μαζικοί καταχωρητές όπως Κατάσταση, Πίνακας και Σωρός έχουν επαναλήπτες, αντικείμενα που διατρέχουν τα στοιχεία τους με χρήση της Ενώ Τέλος Ενώ, ή Ενώ {})

Ας δούμε τι κάνει το παράδειγμα: Πρώτα φτιάχνει ένα πίνακα δέκα στοιχείων. Αμέσως μετά προσπαθεί να τραβήξει δέκα πηγές. Δεν δίνουμε την Άλλη, αλλά δεν υπάρχει πρόβλημα γιατί στη ΠάρεΈνα έχουμε δημιουργήσει μια άδεια Άλλη και στην Διάβασε θα πάρει αυτήν. Αν δώσουμε μια Άλλη στο σωρό τιμών τότε η Διάβασε Άλλη θα κάνει συγχώνευση η οποία θα συγχωνεύσει την άδεια Άλλη με αυτήν που δίνουμε. Γενικά η Διάβασε βγάζει λάθος όταν στο όνομα που διαβάζει δεν υπάρχει τιμή και δεν βρίσκει τιμή στο σωρό τιμών. Εδώ έχουμε ήδη τιμή στην Άλλη (ας είναι και η κενή ομάδα). Επίσης με το Ομάδα Άλλη {} δεν δώσαμε τύπο, οπότε δεν αλλάζει κάτι στην συγχώνευση κατά το διάβασμα. Κατά την φάση της ακολουθίας επανάληψης για να πάρουμε τις πηγές θα βγει λάθος και το αναγνωρίζουμε και εξερχόμαστε από την επανάληψη.

Στην επόμενη φάση βάζουμε πίσω όλες τις πηγές και καθαρίζουμε τον πίνακα.
Τώρα ξεκινάει το Δοκιμαστικό() η ρουτίνα που θα καλέσουμε δυο φορές.
Στη ρουτίνα Δοκιμαστικό() βάζουμε τέσσερα αντικείμενα στην ΠάρεΈνα() και παίρνουμε τέσσερις πηγές. Η ΠάρεΈνα() δεν ελέγχει τι αντικείμενα δίνουμε. Παρακολουθούμε τι πήγε στην Λίστα, με την χρήστη της μεθόδου Λίστα_στο_ΜάζωμαΠίσω.
Μετά ακολουθεί μια ακολουθία επανάληψης για τα τέσσερα στοιχεία, με έλεγχο τύπου κάθε στοιχείο και αναφορές από το καθένα.
Τέλος ακολουθεί μια ακόμα επανάληψη για να βάλουμε πίσω τα στοιχεία με την ΒάλεΈνα, διαγράφουμε τον πίνακα και ελέγχουμε τι έχουμε στο σωρό, με την μέθοδο Λίστα_στο_Μάζωμα.
Για να παρακολουθήσουμε πως χρησιμοποιούμε τις πηγές, έχουμε έναν αθροιστή, μια μνήμη, για κάθε πηγή η οποία αυξάνεται όταν μια πηγή δίνει αντίγραφό της για χρήση, και η εμφάνιση των στοιχείων του σωρού και της λίστας (ιδιωτικά στη κλάση Τροφοδοσία), γίνεται με ζεύγη, γράμμα πηγής και αριθμός χρήσης της.

```
Κλάση ΤοΣτοιχείο {
Ιδιωτικό:
```

```
    X=10, Y=40, Z=500
Δημόσιο:
    Τμήμα ΔείξεΣτοιχεία {
        Τύπωσε Μορφή$("X={0}, Y={1}, Z={2}", .X, .Y, .Z)
    }
Κλάση:
    Τμήμα ΤοΣτοιχείο (.X, .Y, .Z) {
    }
}
Κλάση ΤοΣτοιχείο1 {
Ιδιωτικό:
    X=10, Y=40
Δημόσιο:
    Τμήμα ΔείξεΣτοιχεία {
        Τύπωσε Μορφή$("X={0}, Y={1}", .X, .Y)
    }
Κλάση:
    Τμήμα ΤοΣτοιχείο1 (.X, .Y) {
    }
}
Κλάση Τροφοδοσία {
Ιδιωτικό:
    Μάζωμα=Σωρός
    ΜάζωμαΠίσω=Λίστα
    ΛάθοςΜου=Ψευδής
Δημόσιο:
    Ομάδα ΤελευταίοΛάθος {
        Αξία {
            Ένωσε γονικό ΛάθοςΜου στη μ
            =μ
        }
    }
    ΠάρεΈνα=Λάμδα α$="a"  ->{
        .ΛάθοςΜου<=Ψευδής
        Ομάδα Άλλη {
        }
        Διάβασε Άλλη ως Ομάδα
        Αν Μήκος(.Μάζωμα)=0 Τότε
            Αν Μήκος(.ΜάζωμαΠίσω)<6 Τότε
                Ομάδα Πηγή {
                Τύπος: Πηγή
                Ιδιωτικό
                    μνήμη=0
                Δημόσιο:
```

```
                Τελική όνομα$=α$
                Τμήμα Τελικό Αντιγραφή_μνήμης (Π ως Πηγή) {
                    .μνήμη<=Π.μνήμη+1
                }
                Τμήμα Τελικό Δείξε_Στοιχεία_Μνήμης {
                    Τύπωσε Μορφή$("({0},{1})",.όνομα$, .μνήμη),
                }
            }
            α$=Χαρ$(Κωδ(α$)+1)
        Αλλιώς
            .ΛάθοςΜου<=Αληθής
            Διέκοψε
        Τέλος Αν
    Αλλιώς
        Σωρός .Μάζωμα {Διάβασε Πηγή}
    Τέλος Αν
    Προσθήκη .ΜάζωμαΠίσω, Πηγή.όνομα$:=Πηγή
    =Πηγή με Άλλη
}
Τμήμα ΒάλεΈνα (p ως Πηγή){
    .ΛάθοςΜου<=Ψευδής
    Αν Υπάρχει(.ΜάζωμαΠίσω, p.όνομα$) Τότε
        p1=Έκφρ(.ΜάζωμαΠίσω)
        p1.Αντιγραφή_μνήμης p
        Σωρός .Μάζωμα {
            Σειρά p1
        }
        Αφαίρεση .ΜάζωμαΠίσω, p1.όνομα$
    Αλλιώς
        .ΛάθοςΜου<=Αληθής
    Τέλος Αν
}
Τμήμα Λίστα_στο_Μάζωμα {
    μ=Κάθε(.Μάζωμα)
    Ενώ μ
        Για Αυτό {
            k=ΤιμήΣωρού(μ)
            k.Δείξε_Στοιχεία_Μνήμης
        }
    Τέλος Ενώ
    Τύπωσε
}
Τμήμα Λίστα_στο_ΜάζωμαΠίσω {
    μ=Κάθε(.ΜάζωμαΠίσω)
```

```
        Ενώ μ
            Για Αυτό {
                k=Έκφρ(μ)
                k.Δείξε_Στοιχεία_Μνήμης
            }
        Τέλος Ενώ
        Τύπωσε
    }
}
Μάζωμα=Τροφοδοσία()
Ομάδα Τίποτα {
    Τύπος: Τίποτα
}
Πίνακας Καταχωρητής(10)
Για i=0 έως 9
    Καταχωρητής(i)=Μάζωμα.ΠάρεΈνα()
    Αν Μάζωμα.ΤελευταίοΛάθος Τότε Τύπωσε "Δεν υπάρχει Πηγή για το ";i : Έξοδος Για
Επόμενο i
Για i=5 έως 0
    Μάζωμα.ΒάλεΈνα Καταχωρητής(i)
Επόμενο I
Πίνακας Καταχωρητής(10)=0
Δοκιμαστικό()
Δοκιμαστικό()
Τέλος
Ρουτίνα Δοκιμαστικό()
    Καταχωρητής(0)=Μάζωμα.ΠάρεΈνα(ΤοΣτοιχείο(100,300,500))
    Καταχωρητής(1)=Μάζωμα.ΠάρεΈνα(Τίποτα)
    Καταχωρητής(2)=Μάζωμα.ΠάρεΈνα(ΤοΣτοιχείο())
    Καταχωρητής(3)=Μάζωμα.ΠάρεΈνα(ΤοΣτοιχείο1(20,20))
    Μάζωμα.Λίστα_στο_ΜάζωμαΠίσω
    Για i=0 έως 3 {
        Για Καταχωρητής(i) {
            Τύπωσε .όνομα$
            Αν Αυτό είναι τύπος ΤοΣτοιχείο ή Αυτό είναι τύπος ΤοΣτοιχείο1 Τότε
                .ΔείξεΣτοιχεία
             Αλλιώς.Αν Αυτό είναι τύπος Τίποτα Τότε
                Τύπωσε "Τίποτα στην Καταχωρητής(";i;")"
             Τέλος Αν
        }
    }
    Για i=0 έως 3
        Μάζωμα.ΒάλεΈνα Καταχωρητής(i)
    Επόμενο I
```

```
    Πίνακας Καταχωρητής(10)=0
    Μάζωμα.Λίστα_στο_Μάζωμα
Τέλος Ρουτίνας
```

***Πρόγραμμα 56**: Μοτίβο Pool*

Μοτίβο Memento

Το μοτίβο Memento, έχει στόχο να δίνει σε ένα αντικείμενο δυνατότητα επαναφοράς σε πρότερη κατάσταση.

Στο παράδειγμα έχουμε μια κλάση **Καταγραφικό** με μια μέθοδο **Καταχώρηση** η οποία δεν αλλάζει, είναι τελική και μια μέθοδο **Επαναφορά** η οποία δεν έχει υλοποιηθεί. Αυτή τη κλάση την κληρονομεί η κλάση **Δημιουργός**, η οποία ορίζει την Επαναφορά. Ως έχει το πρόγραμμα θεωρούμε ότι η αντιγραφή του Αυτό στην Καταχώρηση επαρκεί για να κρατήσει την κατάσταση διαφορετικά θα πρέπει να έχουμε μια κλήση σε μια εξειδικευμένη μέθοδο Αντιγραφή(). Η μέθοδος Καταχώρηση επιστρέφει δείκτη σε ένα αντικείμενο τύπου **Καταγραφή_Επόπτη**.

Η κλάση Δημιουργός δεν γνωρίζει που σώνεται ο δείκτης του αντιγράφου της. Η μέθοδος επαναφορά δέχεται ένα δείκτη του τύπου Δημιουργός, εξάγει την ομάδα, και την συγχωνεύει με το Αυτό, έτσι ανακτά πάλι την πρότερη κατάσταση.

Για το παράδειγμα έχουμε μια κλάση Επόπτης η οποία κρατάει ένα δείκτη που θα βάζει αντικείμενο του τύπου Καταγραφή_Επόπτη. Αυτόν τον δείκτη τον χρησιμοποιεί σε μια μέθοδο ΚάνεΚάτι όπου δέχεται με αναφορά ένα αντικείμενο Δημιουργός, εκτελεί μια μέθοδο του δημιουργού την Αλλαγή_Μνήμης (άρα αλλάζει την κατάστασή του), μας δείχνει την αλλαγή με την μέθοδο του δημιουργού την Εμφάνισε_Μνήμη, και τότε κρατάει μια καταχώρηση από το μέρος του δημιουργού που είναι το Καταγραφικό. Αμέσως μετά αλλάζουμε πάλι την μνήμη του δημηιουργού και δείχνουμε την αλλαγή.

Ο επόπτης έχει ακόμα μια μέθοδο την ΕπαναφοράΚατάστασης η οποία ζητάει με αναφορά ένα αντικείμενο δημιουργός και σε αυτό και με χρήση του δείκτη που είχε φυλάξει, καλεί την επαναφορά του δημιουργού δίνοντας ως όρισμα την επιστροφή από το την συνάρτηση Επαναφορά() του αντικειμένου που δείχνει ο δείκτης του επόπτη Δείκτης_σε_ομάδα.

Θα μπορούσε κανείς να βάλει στον επόπτη ένα σωρό και σε αυτόν να βάζει πολλά αντικείμενα τύπου **Καταγραφή_Επόπτη** ή να έχει δυο σωρούς ώστε να γίνεται το UNDO και το REDO. Θα μπορούσε να σκεφτεί κανείς ότι δεν χρειάζεται το αντικείμενο που μεταφέρει τον δημιουργό. Επειδή ο δημιουργός δίνει την δυνατότητα αντιγραφής μπορούμε απευθείας να δώσουμε στο δείκτη στον επόπτη τον δείκτη ενός αντίγραφου του δημιουργού. Η διαφορά με το Memento είναι ότι βάζει την Καταχώρηση μέσα στο Δημιουργό, έτσι ώστε ο Δημιουργός να έχει τον έλεγχο της δημιουργία του αντικειμένου Καταγραφή_Επόπτη. Για παράδειγμα θα μπορούσε να καταχωρήσει ένα αντικείμενο που συμπεριλαμβάνεται στο Δημιουργό, και είναι ιδιωτικό. Η επαναφορά πάλι γίνεται μέσα στο Δημιουργό άρα θα μπορεί να αλλάξει το ιδιωτικό αντικείμενο με την πρότερη κατάσταση ενώ απ΄ έξω δεν γίνεται!

```
Κλάση Καταγραφικό {
Δημόσιο:
    Συνάρτηση Τελική Καταχώρηση {
        Δείκτης_σε_ομάδα->(Αυτό)
        Ομάδα Εσωτερική {
            Τύπος: Καταγραφή_Επόπτη
        Ιδιωτικό:
            Καταχώρηση_Δείκτη=Δείκτης_σε_ομάδα
        Δημόσιο:
            Συνάρτηση Επαναφορά() {
                =.Καταχώρηση_Δείκτη
            }
        }
        ->(Εσωτερική)
    }
    Τμήμα Επαναφορά {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα "+όνομα.τμήματος$
    }
}
Κλάση Δημιουργός ως Καταγραφικό {
Ιδιωτικό:
    μνήμη=1
Δημόσιο:
    Τμήμα Αλλαγή_Μνήμης {
        .μνήμη+=100
    }
    Τμήμα Εμφάνισε_Μνήμη {
        Τύπωσε "μνήμη=";.μνήμη
    }
    Τμήμα Επαναφορά (Δείκτης_σε_Δημιουργό ως *Δημιουργός) {
        Αυτό=Ομάδα(Δείκτης_σε_Δημιουργό)
    }
}
Κλάση Επόπτης {
Ιδιωτικό:
    Δείκτης_σε_ομάδα=Δείκτης()
Δημόσιο:
    Τμήμα ΚάνεΚάτι (&Δημιουργός ως Δημιουργός){
        Δημιουργός.Αλλαγή_Μνήμης
        Δημιουργός.Εμφάνισε_Μνήμη
        .Δείκτης_σε_ομάδα<=Δημιουργός.Καταχώρηση()
        Δημιουργός.Αλλαγή_Μνήμης
        Δημιουργός.Εμφάνισε_Μνήμη
    }
```

```
    Τμήμα ΕπαναφοράΚατάστασης (&Δημιουργός ως Δημιουργός){
        Αν .Δείκτης_σε_ομάδα Είναι Τύπος Καταγραφή_Επόπτη Τότε
            Δημιουργός.Επαναφορά .Δείκτης_σε_ομάδα=>Επαναφορά()
            Δημιουργός.Εμφάνισε_Μνήμη
        Τέλος Αν
    }
}
Επόπτης=Επόπτης()
Δημιουργός=Δημιουργός()
Επόπτης.ΚάνεΚάτι &Δημιουργός
Επόπτης.ΕπαναφοράΚατάστασης  &Δημιουργός
```

***Πρόγραμμα 57**: Μοτίβο Memento*

Μοτίβο Mediator

Το μοτίβο Mediator είναι ο μεσολαβητής μεταξύ αντικειμένων τα οποία επικοινωνούν με γεγονότα. Δεν έχουμε δει σε αυτό το άρθρο την χρήση γεγονότων στις ομάδες. Εδώ γίνεται χρήση δυο διαφορετικών τύπων γεγονότων. Ο ένας τύπος παρέχεται από τις ομάδες και ανήκει στα ελαφριά γεγονότα. Ελαφρύ γεγονός είναι αυτό που συνδέει άμεσα ένα αντικείμενο ομάδα με συναρτήσεις εξυπηρέτησης γεγονότων. Σε αυτά τα γεγονότα δεν μπορούν να προστεθούν συναρτήσεις, και μπορούν να δοθούν οποιοσδήποτε αριθμός ορισμάτων. Ο άλλος τύπος είναι το αντικείμενο Γεγονός. Ένα αντικείμενο γεγονός μπορεί να περιέχεται σε μια Ομάδα. Αν περιέχεται σε ομάδα τότε είναι με δείκτη, που σημαίνει ότι αν αντιγράφουμε Ομάδες με αντικείμενα Γεγονότα τότε αντιγράφεται ο δείκτης τους. Στη χρήση τους μέσα σε μεθόδους και έξω από αντικείμενα είναι τιμές, δηλαδή μπορούν να αλλάξουν τιμή. Ένα αντικείμενο γεγονός κρατάει μια λίστα συναρτήσεων, και έχει δηλωθεί με μια υπογραφή, δηλαδή με ένα ορισμένο τρόπο που θα διαβάζει τον σωρό τιμών. Όταν καλούμε ένα γεγονός (αντικείμενο) τότε όλες οι συναρτήσεις διαδοχικά θα πάρουν τα ίδια ορίσματα. Βεβαίως μπορούμε να έχουμε ορίσματα με αναφορά που σημαίνει ότι μπορούμε να περιπλέξουμε τα πράγματα!

Στο παράδειγμα έχουμε δυο κλάσεις την **Συνάδελφος** και την **Μεσολαβητής** οι οποίες πρέπει να υλοποιηθούν με άλλες, και έτσι γίνεται με την **ΣτιβαρόςΣυνάδελφος** και **ΣτιβαρόςΜεσολαβητής**. Φτιάχνουμε πρώτα ένα αντικείμενο Μεσολαβητής, στο οποίο δίνουμε ένα μέρος του τμήματος για να μπορεί να το καλεί πίσω (callback). Χρησιμοποιούμε μια συνάρτηση Οκν$() η οποία παίρνει μια έκφραση και παράγει μια συνάρτηση για οκνηρή αποτίμηση, ή παίρνει μια αναφορά συνάρτησης, παίρνει τον κώδικά της και φτιάχνει μια άλλη συνάρτηση που τρέχει σαν να είναι το τμήμα που εκτελέστηκε η Οκν$(). Με αυτό το τρόπο βάζουμε τη συνάρτηση ΟποιοδήποτεΌνομα (το όνομά της δεν έχει σημασία), η οποία θα κληθεί ως τμήμα και θα έχει θέαση σε ότι έχει το τμήμα, θα κληθεί σαν ρουτίνα. Παράλληλα ορίσαμε τον Μεσολαβητή ΜεΓεγονότα, δηλαδή θα χρησιμοποιηθεί ένα ελαφρύ γεγονός, το "τελείωσε" το οποίο στη κλήση από τον Μεσολαβητή περιμένει να βρει μια συνάρτηση Μεσολαβητής_τελείωσε. Και αυτή η συνάρτηση θα κληθεί σαν να είναι ρουτίνα του τμήματος.

Ο Μεσολαβητής έχει μια μέθοδο **μεσολάβηση** την οποία πρέπει να υλοποιεί ο ΣτιβαρόΜεσολαβητής (η κλάση από την οποία τον δημιουργήσαμε), αφού κληρονομεί από την κλάση Μεσολαβητής. Σε αυτήν την μέθοδο δίνουμε με αναφορά τον Συνάδελφο ο οποίος έχει φτιαχτεί από την κλάση ΣτιβαρόςΣυνάδελφος και η οποία κλάση κληρονομεί από τη κλάση Συνάδελφος, μια μέθοδο **πάρε_αυτά**. Αυτή η μέθοδος χρειάζεται δυο γεγονότα (αντικείμενα) ως ορίσματα.

Ο Μεσολαβητής μόλις πάρει τον Συνάδελφο αρχίζει να τον περνάει σε μια ακολουθία επανάληψης όπου καλεί μια δική του μέθοδο Έργο1 και σε αυτήν στέλνει την αναφορά του Συνάδελφου. Η επανάληψη τελειώνει όταν ένα αθροιστής γίνει μεγαλύτερος από το 24000. Προς το παρόν δεν φαίνεται που ο αθροιστής αλλάζει! Στη μέθοδο Έργο1 γίνεται κλήση της μεθόδου πάρε_αυτά του Συναδέλφου με ορίσματα τα Αυτό.ένα και Αυτό.δύο δηλαδή δυο γεγονότα ιδιωτικά στο Μεσολαβητή. Αν δούμε τον κατασκευαστή του Μεσολαβητή στην κλάση ΣτιβαρόςΜεσολαβητής και τη μέθοδο ΣτιβαρόςΜεσολαβητής θα δούμε ότι βάζουμε στο γεγονός Αυτό.ένα δυο συναρτήσεις την ιδιωτική ΚάνεΚάτι1() και την εξωτερική (αυτή που δώσαμε ως callback), ενώ στο γεγονός Αυτό.δύο μια συνάρτηση, την ιδιωτική ΚάνεΚάτι2().

Στην Κλάση ΣτιβαρόςΣυνάδελφος έχουμε υλοποιήσει τη μέθοδο πάρε_αυτά και εκεί καλούμε τα γεγονότα βάζοντας την τιμή 100+.Χ και στο δεύτερο δίνουμε με αναφορά την .Χ

Η εξωτερική ΟποιαδήποτεΌνομα παίρνει την τιμή 100+.Χ και την προσθέτει σε μια μεταβλητή άθροισμα που ήδη υπάρχει στο τμήμα. Η εσωτερική ΚάνεΚάτε1 αυξάνει το άθροισμα. Η εσωτερική ΚάνεΚάτι2 παίρνει με αναφορά το .Χ και αυξάνει την τιμή κατά 20000.

Με απλά λόγια ο Συνάδελφος κατά τη μεσολάβηση επηρεάζει την δική του κατάσταση, τη κατάσταση του Μεσολαβητή και την κατάσταση του τμήματος που είναι ο μεσολαβητής, χωρίς να έχει γνώση του τι κάνει! Όταν τελειώσει τη μεσολάβηση ο Μεσολαβητής καλεί το γεγονός "τελείωσε". Το μοτίβο σχετίζεται με την μεσολάβηση δυο ή περισσότερων συναδέλφων, που κληρονομούν από τη κλάση Συνάδελφος. Στο παράδειγμα έγινε μια αλλαγή για να φανεί πώς από τον Μεσολαβητή μπορούν να γίνουν κλήσεις πριν τερματίσει η κλήση στον πρώτο συνάδελφο. Μπορούμε στη μεσολάβηση να ορίζουμε ότι το έργο1 θα γίνεται σε δυο ή τρεις συναδέλφους. Ένα γεγονός μπορεί να μην εξυπηρετηθεί και αυτό δεν εμφανίζεται ως λάθος. Η μέθοδος του Συναδέλφου μπορεί να έχει ό,τι επίδραση θέλουμε, αφού στην πράξη δεν γνωρίζει ο Συνάδελφος τι καλεί πραγματικά. Επίσης και ο Μεσολαβητής με δικά του γεγονότα μπορεί και αυτός να καλεί και αν πάρει απόκριση, έστω καλεί με τιμή με αναφορά, δηλαδή έχει αλλαγή στην τιμή, να αλλάξει συμπεριφορά βάσει της τιμής!

```
Κλάση Συνάδελφος {
    Τμήμα πάρε_αυτά (γεγονός1, γεγονός2) {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα "+ονομα.τμήματος$
    }
}
Κλάση ΣτιβαρόςΣυνάδελφος ως Συνάδελφος {
    Χ=2000
    Τμήμα πάρε_αυτά (γεγονός1, γεγονός2){
        Κάλεσε Γεγονός γεγονός1, 100+.Χ
        Κάλεσε Γεγονός γεγονός2, &.Χ
```

```
    }
}
Κλάση Μεσολαβητής {
    Τμήμα Μεσολάβηση {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα "+ονομα.τμήματος$
    }
}
Κλάση ΣτιβαρόςΜεσολαβητής ως Μεσολαβητής {
    Γεγονός "τέλειωσε"
Ιδιωτικό:
    αθροιστής=0
    Τμήμα Έργο1 (&εκείνο ως Συνάδελφος) {
        εκείνο.πάρε_αυτά .ένα, .δύο
    }
    Γεγονός ένα {
        Διάβασε x
    }
    Γεγονός δύο {
        Διάβασε &x
    }
    Συνάρτηση ΚάνεΚάτι1(Νέο μια_τιμή) {
        Τύπωσε "μια_τιμή=";μια_τιμή
        .αθροιστής+=μια_τιμή
    }
    Συνάρτηση ΚάνεΚάτι2(Νέο &μια_τιμή) {
        μια_τιμή+=20000
    }
Δημόσιο:
   Συνάρτηση Αθροιστής() {
        =.αθροιστής
   }
    Τμήμα Μεσολάβηση (&ΆλλοςΣυνάδελφος ως Συνάδελφος) {
        Επανέλαβε
            .Έργο1 &ΆλλοςΣυνάδελφος
        Μέχρι .αθροιστής>=2400
        Κάλεσε Γεγονός "τέλειωσε"
    }
Κλάση:
    Τμήμα ΣτιβαρόςΜεσολαβητής (&m()) {
        Γεγονός .ένα Νέο &.ΚάνεΚάτι1(), m()
        Γεγονός .δύο Νέο &.ΚάνεΚάτι2()
    }
}
άθροισμα=0
```

```
Συνάρτηση ΟποιοδήποτεΌνομα {
    Διάβασε Νέο μιά_τιμή
    Τύπωσε "Από συνάρτηση ΟποιοδήποτεΌνομα:", μιά_τιμή
    άθροισμα+=μιά_τιμή
}
ΣτιβαρόςΣυνάδελφος=ΣτιβαρόςΣυνάδελφος()
Ομάδα ΜεΓεγονότα Μεσολαβητής=ΣτιβαρόςΜεσολαβητής(Οκν$(&ΟποιοδήποτεΌνομα()))
Συνάρτηση Μεσολαβητής_τέλειωσε {
    Τύπωσε "Τέλειωσε ", άθροισμα
}
Μεσολαβητής.Μεσολάβηση &ΣτιβαρόςΣυνάδελφος
Τύπωσε Μεσολαβητής.Αθροιστής()=24200
Τύπωσε άθροισμα=24200
```

***Πρόγραμμα 58**: Μοτίβο Mediator*

Μοτίβο Observer

Το μοτίβο Observer δείχνει προγραμματιστικά πως χρησιμοποιούμε γεγονότα, χωρίς τη χρήση των δυνατοτήτων των ομάδων για γεγονότα και χωρίς τη χρήση των αντικειμένων γεγονότων που περιγράφτηκε στο παράδειγμα του μοτίβου Mediator.

Το μοτίβο χρησιμοποιείται για αντικείμενα που δημοσιεύουν τις αλλαγές τους σε ένα υποκείμενο που κρατάει ευρετήριο αντικειμένων τύπου Παρατηρητή. Ένα αντικείμενο Παρατηρητής, έχει ένα όνομα, δυο συναρτήσεις εξυπηρέτησης έξω από το υποκείμενο, και μια σειρά αντικειμένων που δύναται να αλλάξει η κατάστασή τους. Το υποκείμενο περιοδικά ελέγχει τους παρατηρητές και κάθε παρατηρητής ελέγχει τα δικά του αντικείμενα. Όταν κάποιο αντικείμενο αλλάξει κατάσταση ενημερώνει το υποκείμενο και εκτελεί την συνάρτηση εξυπηρέτησης.

Το παράδειγμα διαφέρει από το αντίστοιχο στα αγγλικά ως προς το αντικείμενο που δημιουργείται από τη κλάση Υποκείμενο. Εδώ το αντικείμενο που παίρνουμε το κρατάμε σε δείκτη ενώ στο αντίστοιχο αγγλικό το κρατάμε σε επώνυμη ομάδα. Έχουν γίνει αλλαγές για να έχουμε χαλαρή σύνδεση του Παρατηρητή με το Υποκείμενο (θα εξηγηθεί παρακάτω)

Το παράδειγμα έχει μια σειρά κλάσεων, ακολουθεί μια λίστα κλάσεων και διεπαφής (μέθοδοι και ιδιότητες που μπορούν να χρησιμοποιηθούν έξω από τα αντικείμενα):

Κλάση **Υποκείμενο**:
    ΒάλεΠαρατηρητή, ΒγάλεΠαρατηρητή, Ενημέρωσε, Παρατήρησε, Διαγραφή
Κλάση **ΚάθεΚλικ**:
    Αφηρημένη ορίζει μια μέθοδο: διάβασέ_το
Κλάση **ΈναΚλικ** ως ΚάθεΚλικ:
    τελική μέθοδος διάβασέ_το, Διαγραφή.
Κλάση **ΚλικΠοντικιού** ως ΚάθεΚλικ:
    τελική μέθοδος διάβασέ_το, Διαγραφή.
Κλάση **Παρατηρητής**:
    όνομα$ (μόνο ανάγνωση), Κάνε_Κάτι, Διαγραφή.

Η μέθοδος διαγραφή καλείται όταν κανένας δείκτης δεν δείχνει το αντικείμενο, κατά τη φάση διαγραφής. Εδώ θέλουμε να κληθεί για όλα τα αντικείμενα που φτιάχνουμε. Επίσης μπήκε ένα κείμενο στην αρχή και χωρισμός οθόνης από την πέμπτη γραμμή, επειδή η μέθοδος Παρατήρησε εμφανίζει περιοδικά την φράση Παρατηρώ και έτσι έχουμε ολίσθηση στο κάτω μέρος (από την πέμπτη γραμμή) της οθόνης, ενώ το πάνω μέρος δείχνει τι μπορούμε να κάνουμε για να αλλάξουμε τη κατάσταση των αντικειμένων. Επίσης μπήκε φραγή στο πλήκτρο Esc, ώστε να μην διακοπεί το πρόγραμμα, αν πατηθεί κατά λάθος το Esc.

Με τη κλάση ΈναΚλικ ορίζουμε ένα αντικείμενο που ελέγχει ένα πλήκτρο, εδώ το "Α". Με την κλάση ΚλικΠοντικιού ορίζουμε ένα αντικείμενο που ελέγχει ένα από τα πλήκτρα του ποντικιού.

Η κλάση Παρατηρητής δέχεται ένα όνομα, δυο αναφορές συναρτήσεων, και ακολουθούν ορίσματα του τύπου ΚάθεΚλικ, δηλαδή ΈναΚλικ ή ΚλικΠοντικιού.

Το πρόγραμμα εκτελεί την μέθοδο **Υποκείμενο=>Παρατήρησε**. Αυτή η μέθοδος λειτουργεί μέχρι να πατήσουμε το διάστημα και μπορούμε να την κάνουμε να είναι αργή ή γρήγορη στην απόκριση αλλάζοντας τον αριθμό χιλιοστών δευτερολέπτου στην εντολή **Αναμονή**. Στο διάστημα αυτό αν πατήσουμε κάποιο πλήκτρο του ποντικιού ή το Α θα πάρουμε απάντηση από το σύστημα. Ειδικά για το μεσαίο πλήκτρο του ποντικιού έχουμε έναν μετρητή (έχει φτιαχτεί στο τμήμα που τρέχει το πρόγραμμα) και στο οποίο έχουν θέαση όλες οι συναρτήσεις που φτιάχτηκαν με την Οκν$() και νοούνται ως κλήση προς τα πίσω. Έτσι όταν ο μετρητής περάσει το 3 τότε βγάζει τον Παρατηρητή του από το Υποκείμενο. Για να βγει χρειάζεται μόνο το όνομα του Παρατηρητή (που είναι το κλειδί στη λίστα που έχει καταχωρηθεί).

Στην μέθοδο κάνε_κάτι του Παρατηρητή, με το πρώτο αντικείμενο που θα δείξει αλλαγή (ότι έχει πατημένο πχ το Α ή κάποιο πλήκτρο του ποντικιού) τότε κάνει ότι έχει να κάνει και με μια Διέκοψε τερματίζει την μέθοδο. Με αυτόν τον τρόπο δεν θα πάρουμε δυο φορές το σήμα "αλλαγής" του Παρατηρητή. Αν θέλουμε μπορούμε σε δυο ή περισσότερους Παρατηρητές να βάλουμε το ίδιο αντικείμενο, έστω για το "Α" έτσι ώστε αν πατήσουμε το Α να παραχθούν γεγονότα από τον κάθε Παρατηρητή που το περιέχει.

Επειδή δεν θέλουμε ο Παρατηρητής να βλέπει άμεσα το Υποκείμενο, δίνουμε μια συνάρτηση ως κλήση προς τα πίσω και μέσω αυτής δίνουμε το όνομα του Παρατηρητή στην μέθοδο Ενημέρωση του Υποκείμενου. Την συνάρτηση την κρατάμε σε αλφαριθμητικό και επειδή θέλουμε να περάσουμε τιμή την συνδέουμε με ένα όνομα συνάρτησης και καλούμε αυτό το όνομα με την παράμετρο που θέλουμε.

Έχει χρησιμοποιηθεί μια Για Αυτό { } που ανοίγει ένα μπλοκ προσωρινών ορισμών, και έτσι οι μεταβλητές που δημιουργούμε (για να φαίνεται πιο κατανοητός ο κώδικας) διαγράφονται στο τέλος του.

Χαλαρή & Σφιχτή Σύνδεση (Loose and Tight Coupling)

Στο παράδειγμα έχουμε χαλαρή σύνδεση (loose coupling) για τον Παρατηρητή προς το Υποκείμενο, ενώ το Υποκείμενο έχει σφιχτή σύνδεση (tight coupling) με τον Παρατηρητή, αφού εκτός από τον δείκτη που κρατάει προς αυτόν, στη ΒάλεΠαρατηρητή και στην Παρατήρησε χρησιμοποιεί τη διεπαφή του. Το Υποκείμενο εξαρτιέται από τον Παρατηρητή, έτσι αν αλλάξουμε όνομα στη Κάνε_Κάτι ή στο όνομα$ τότε θα πρέπει να κάνουμε αλλαγές στο Υποκείμενο. Σε

αντίθεση αν κάνουμε αλλαγές στην διεπαφή του Υποκείμενου, ο Παρατηρητής δεν επηρεάζεται, αφού δεν χρησιμοποιεί την διεπαφή του Υποκείμενου μέσα στο κώδικά του.

```
Διαφυγή Όχι ' βγάζουμε εκτός το Esc πλήκτρο
Οθόνη ,0
Αναφορά {Μοτίβο Observer
    Πάτα τα πλήκτρα του ποντικιού, και το μεσαίο
    Πάτα το πλήκτρο Α (ανεξάρτητα γλώσσας και πεζών/κεφαλαίων)
    Πάτα το διάστημα για έξοδο
}
Οθόνη ,4
Κλάση Υποκείμενο {
Ιδιωτικό:
    ευρετήριο=Λίστα
Δημόσιο:
    Τμήμα ΒάλεΠαρατηρητή (Παρατηρητής ως *Παρατηρητής) {
        Προσθήκη .ευρετήριο, Παρατηρητής=>όνομα$:=Παρατηρητής
    }
    Τμήμα ΒγάλεΠαρατηρητή (κλειδί$) {
        Αν Δεν Υπάρχει(.ευρετήριο, κλειδί$) Τότε Έξοδος
        Αφαίρεση .ευρετήριο, κλειδί$
    }
    Τμήμα Ενημέρωσε(όνομα$){
         Τύπωσε "Εισερχόμενο γεγονός ";όνομα$
    }
    Τμήμα Παρατήρησε {
        κάτι=Κάθε(.ευρετήριο)
        Ενώ κάτι {
            Παρατηρητής=Έκφρ(κάτι)
            Παρατηρητής=>κάνε_κάτι
        }
        Τύπωσε "Παρατηρώ"
        Αναμονή 100
        Αν ΕνΚομ$<>" " Τότε Κυκλικά
    }
    Διαγραφή {
         Τύπωσε "Διαγράφω το Υποκείμενο"
    }
}
Κλάση ΚάθεΚλικ {
    Συνάρτηση διάβασέ_το {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
```

```
}
Κλάση ΈναΚλικ ως ΚάθεΚλικ {
    νούμερο=0
    Συνάρτηση Τελική διάβασέ_το {
        Αν Πατημένο(.νούμερο) Τότε
            =Αληθής
        Τέλος Αν
    }
    Διαγραφή {
        Τύπωσε "Διαγράφω το ΈναΚλικ ";.νούμερο
    }
Κλάση:
    Τμήμα ΈναΚλικ (.νούμερο) {
    }
}
Κλάση ΚλικΠοντικιού ως ΚάθεΚλικ {
    νούμερο=0
    Συνάρτηση Τελική διάβασέ_το {
        Αν Δυαδικό.Και(Δείκτης, .νούμερο)=.νούμερο Τότε
            =Αληθής
        Τέλος Αν
    }
    Διαγραφή {
        Τύπωσε "Διαγράφω το ΚλικΠοντικιού ";.νούμερο
    }
Κλάση:
    Τμήμα ΚλικΠοντικιού (.νούμερο) {
    }
}
Κλάση Παρατηρητής {
Ιδιωτικό:
    ενημέρωση_υποκειμένου$
    κλικ=Λίστα, αναφορά_συνάρτησης$
Δημόσιο:
    Ιδιότητα όνομα$ {Αξία} ' δεν μπορεί να αλλάξει απ' έξω
    Διαγραφή {
        .κλικ<=Λίστα
        Τύπωσε "Διαγράφω το ";.όνομα$
    }
    Τμήμα κάνε_κάτι {
        κάτι=Κάθε(.κλικ)
        Ενώ κάτι
            κάποιο_κλικ=Έκφρ(κάτι)
            Αν κάποιο_κλικ=>Διάβασέ_το() Τότε
```

```
                Τύπωσε "Παρατηρητής.κάνε_κάτι: Κλήθηκε από ";.όνομα$
                Ένωσε Ισχνή .ενημέρωση_υποκειμένου$ στη ενημέρωση()
                Κάλεσε ενημέρωση(.όνομα$)
                Κάλεσε .αναφορά_συνάρτησης$
                Διέκοψε
            Τέλος Αν
        Τέλος Ενώ
    }
Κλάση:
    Τμήμα Παρατηρητής(.[όνομα]$, .αναφορά_συνάρτησης$, .ενημέρωση_υποκειμένου$,
Κάποιο_κλικ ως *ΚάθεΚλικ) {
        Προσθήκη .κλικ, Μήκος(.κλικ):=Κάποιο_κλικ
        Ενώ όχι Κενό
            Διάβασε Κάποιο_κλικ
            Αν όχι Κάποιο_κλικ είναι τύπος ΚάθεΚλικ Τότε Λάθος "δεν είναι σωστός τύπος"
            Προσθήκη .κλικ, Μήκος(.κλικ):=Κάποιο_κλικ
        Τέλος Ενώ
    }
}
Συνάρτηση Ενημέρωσε (Νέο όνομα$){
    \\ δεν θέλουμε ο Παρατηρητής να συνδέεται άμεσα με το Υποκείμενο.
    Υποκείμενο=>Ενημέρωσε όνομα$
}
Συνάρτηση Πλήκτρο1 {
    Τύπωσε "Εξυπηρέτηση για το γεγονός: Πλήκτρο 1"
}
Συνάρτηση Πλήκτρο2 {
    Τύπωσε "Εξυπηρέτηση για το γεγονός: Πλήκτρο 2"
}
Μέτρητής1=0
Συνάρτηση Μεσαίο {
    Τύπωσε "Εξυπηρέτηση για το γεγονός: Μεσαίο Πλήκτρο"
    Μέτρητής1++
    Αν Μετρητής1>3 τότε Υποκείμενο=>ΒγάλεΠαρατηρητή "Μεσαίο"
}
Κάλεσε_Ενημέρωση$=Οκν$(&Ενημέρωσε())
Κάλεσε_Πίσω1$=Οκν$(&Πλήκτρο1())
Κάλεσε_Πίσω2$=Οκν$(&Πλήκτρο2())
Κάλεσε_Πίσω3$=Οκν$(&Μεσαίο())
Υποκείμενο->Υποκείμενο()
Για Αυτό {
    \\ μπλοκ για προσωρινούς ορισμούς
    ΚλικΑ=Δείκτης(ΈναΚλικ(Asc("Α")))
    ΚλικΠοντικοου1=Δείκτης(ΚλικΠοντικιού(1))
```

```
    ΚλικΠοντικιου2=Δείκτης(ΚλικΠοντικιού(2))
    ΚλικΠοντικιου4=Δείκτης(ΚλικΠοντικιού(4))
    Παρατηρητής->Παρατηρητής("Πλήκτρο1", Κάλεσε_Πίσω1$, Κάλεσε_Ενημέρωση$,
ΚλικΑ, ΚλικΠοντικοου1)
    Υποκείμενο=>ΒάλεΠαρατηρητή Παρατηρητής
    Παρατηρητής=Δείκτης(Παρατηρητής("Πλήκτρο2", Κάλεσε_Πίσω2$,
Κάλεσε_Ενημέρωση$, ΚλικΠοντικιου2))
    Υποκείμενο=>ΒάλεΠαρατηρητή Παρατηρητής
    Παρατηρητής=Δείκτης(Παρατηρητής("Μεσαίο", Κάλεσε_Πίσω3$,
Κάλεσε_Ενημέρωση$, ΚλικΠοντικιου4))
    Υποκείμενο=>ΒάλεΠαρατηρητή Παρατηρητής
}
Υποκείμενο=>Παρατήρησε
Τύπωσε "Τέλος"
Διαφυγή Ναι ' ενεργοποιούμε ξανά το πλήκτρο Esc
```

***Πρόγραμμα 59**: Μοτίβο Observer*

Μοτίβο Null Object

Το μοτίβο Null Object, ή Μηδενικό Αντικείμενο, δημιουργεί ένα αντικείμενο όπως δηλώνει μια αφηρημένη κλάση, χωρίς όμως να κάνει κάτι, απλά διατηρεί την διεπαφή. Με αυτόν τον τρόπο δεν βάζουμε το πραγματικό Null αντικείμενο. Στη Μ2000 ο Μηδενικός τύπος είναι ο τύπος του αντικειμένου που γυρίζει η συνάρτηση Δείκτης() και το αντίστοιχο ->0&. Το πραγματικό μηδενικό αντικείμενο έχει την ιδιότητα στη συγχώνευση με μια άλλη ομάδα να εξαφανίζεται. Έτσι αν Α->0& και Β=Κάτι() τότε το Γ->(Α με Β) θα δώσει στο Γ ένα μόνο τύπο το Κάτι. Το ίδιο θα γίνει και ανάποδα Γ->(Β με Α). Αυτές τις ιδιότητες δεν τις έχει η ΜηδενικήΟμάδα του παραδείγματος. Έτσι την χρησιμοποιούμε μόνο όπου δεν θέλουμε να χειριστούμε την ομάδα τύπου Μηδενικός (Null), αλλά μια ομάδα με διεπαφή μεν αλλά χωρίς να κάνει κάτι.

Στο πρόγραμμα ο Πελάτης λαμβάνει τρία αντικείμενα, και τα καταχωρεί σε έναν πίνακα (στην ουσία μπαίνουν αντίγραφα στο πίνακα, γιατί οι ομάδες πρώτη, δεύτερη και τρίτη είναι επώνυμες, δεν είναι δείκτες). Εδώ έχουμε τις δυο Για, την δομή επανάληψης και την δομή για τα αντικείμενα. Φαίνεται η διαφορά τους, η δομή επανάληψης έχει αμέσως μετά το όνομα μεταβλητής τον τελεστή εκχώρησης.

```
Κλάση Αφηρημένη {
    Τμήμα Κάνε_Κάτι {
        Λάθος "Αφηρημένη"
    }
}
Κλάση ΜηδενικήΟμάδα ως Αφηρημένη {
    Τμήμα Τελικό Κάνε_Κάτι {
        \\ Κανε κάτι
    }
}
Κλάση Καταγραφέας {
Ιδιωτικό:
```

```
    ο_κωδικός_μου=0
Κλάση:
    Τμήμα Καταγραφέας (.ο_κωδικός_μου) {
    }
}
Κλάση Στιβαρή ως Καταγραφέας ως Αφηρημένη {
    Τμήμα Τελικό Κάνε_Κάτι {
        Print "Καταγραφή σε Αρχείο από ";.ο_κωδικός_μου
    }
Κλάση:
    Τμήμα Στιβαρή {
        .Καταγραφέας
    }
}
Κλάση ΣτιβαρήΆλλη ως Καταγραφέας ως Αφηρημένη {
    Τμήμα Τελικό Κάνε_Κάτι {
        Print "Καταγραφή στην Κονσόλα από ";.ο_κωδικός_μου
    }
Κλάση:
    Τμήμα ΣτιβαρήΆλλη {
        .Καταγραφέας
    }
}
Τμήμα Πελάτης(πρώτη ως Αφηρημένη, δεύτερη ως Αφηρημένη, τρίτη ως Αφηρημένη) {
    Πίνακας πίνακας_για_ομάδες(3)
    πίνακας_για_ομάδες(0)=πρώτη
    πίνακας_για_ομάδες(1):=δεύτερη, τρίτη
    Για ι=0 έως 2
        Για πίνακας_για_ομάδες(ι) {
            .Κάνε_Κάτι
        }
    Επόμενο ι
}
Πελάτης ΜηδενικήΟμάδα(), Στιβαρή(1001), ΣτιβαρήΆλλη(2002)
```

***Πρόγραμμα 60**: Μοτίβο Null Object*

Μοτίβο Visitor

Ένα από τα πιο ωραία μοτίβα είναι το μοτίβο Visitor, ή Επισκέπτης. Αν έχουμε μια δομή δένδρου φτιαγμένη με αντικείμενα, τότε μπορούμε να χρησιμοποιήσουμε το μοτίβο Επισκέπτης για να περάσει από όλα τα στοιχεία του και να συλλέξει πληροφορίες.

Στο παράδειγμα χρησιμοποιούμε δείκτες σε ομάδες. Υπάρχουν διαφορές ως προς την αγγλική έκδοση, η Ομάδα Πελατών είναι δείκτης (στην αγγλική είναι επώνυμη ομάδα),και επιπλέον έχουν μπει σε όλα τα αντικείμενα η Διαγραφή που μας ανακοινώνει ποιο αντικείμενο διαγράφεται. Έχουν κρατηθεί τα ίδια στοιχεία που δίνουμε στην αγγλική έκδοση.

Το παράδειγμα έχει μια σειρά κλάσεων, ακολουθεί μια λίστα κλάσεων και διεπαφής (μέθοδοι και ιδιότητες που μπορούν να χρησιμοποιηθούν έξω από τα αντικείμενα):

Κλάση **Επισκέπτης**:
Αφηρημένη ορίζει μία δημόσια μέθοδο: Επίσκεψη (είναι τελική, δεν αλλάζει)
Κλάση **Επισκέψιμη**:
Αφηρημένη ορίζει μία δημόσια μέθοδο: Αποδοχή
Κλάση **Στοιχείο** ως Επισκέψιμη:
τελική μέθοδος Αποδοχή, Διαγραφή και Ιδιότητα Όνομα$
Κλάση **Παραγγελία** ως Επισκέψιμη:
τελική μέθοδος Αποδοχή, Βάλε_Στοιχείο, Διαγραφή και Ιδιότητα Όνομα$
Κλάση **Πελάτης** ως Επισκέψιμη:
τελική μέθοδος Αποδοχή, Βάλε_Παραγγελία, Διαγραφή και Ιδιότητα Όνομα$
Κλάση **Ομάδα_Πελατών** ως Επισκέψιμη:
τελική μέθοδος Αποδοχή, Βάλε_Πελάτη, Διαγραφή
Κλάση **Γενική_Αναφορά** ως Επισκέπτης:
Επίσκεψη, Εμφάνισε_Αποτελέσματα, Διαγραφή.

Όλες οι ιδιότητες Όνομα$ είναι μόνο για ανάγνωση

Για να φτιάξουμε το δένδρο εργαζόμαστε σαν να πλέκουμε με βελονάκι. Η θηλιά στο πλεκτό κρατάει την κλωστή δηλαδή το αντικείμενο. Η θηλιά είναι φτιαγμένη από κλωστή, δηλαδή είναι αντικείμενο. Η κλωστή είναι επιλογή μας, ως προς τα χαρακτηριστικά της, δηλαδή είναι τα δεδομένα που δίνουμε για να φτιαχτεί ένα αντικείμενο. Έτσι η επιλογή κλωστής είτε είναι επιλογή έτοιμου αντικειμένου είτε είναι ετοιμασία αντικειμένου με επιλογές στο κατασκευαστή του. Η σειρά που θα φτιάξουμε τις κλωστές, τα αντικείμενα, δεν παίζει ρόλο όσο ο κατασκευαστής τους δεν χρειάζεται κάποιο αντικείμενο ως επιλογή μας (ως όρισμα).

1. Στο πρόγραμμα ξεκινάμε από τη Ομάδα_Πελατών, την δημιουργούμε και βάζουμε ένα δείκτη σε αυτήν από τη κλάση Ομάδα_Πελατών
2. Μετά ασχολούμαστε με τη Παραγγελία, στην οποία ο κατασκευαστής της δέχεται ως δεύτερο όρισμα ένα στοιχείο, έτσι με μια κατασκευή Παραγγελίας έχουμε ένα αντικείμενο Παραγγελία το οποίο διαθέτει ένα αντικείμενο Στοιχείο. Επειδή θέλουμε να βάλουμε και άλλο στοιχείο καλούμε την μέθοδο Βάλε_Στοιχείο του αντικειμένου Παραγγελία και δίνουμε ως όρισμα ένα δείκτη από ένα νέο Στοιχείο (με τις επιλογές του στοιχείου, το όνομά του εδώ).
3. Φτιάχνουμε το Μ1 από τη κλάση Πελάτης και βάζουμε την Παραγγελία με τη μέθοδο Βάλε_Παραγγελία.
4. Φτιάχνουμε μια Παραγγελία1 ως αντικείμενο από τη κλάση Παραγγελία και επίσης την βάζουμε στο Πελάτη Μ1.
5. Βάζουμε το Πελάτη Μ1 στην Ομάδα_Πελατών. Εδώ τελειώσαμε με τις πρώτες θηλιές
6. Φτιάχνουμε μια νέα παραγγελία και δίνουμε το δείκτη της στο Παραγγελία1, με ένα στοιχείο. Προσθέτουμε δυο ακόμα στοιχεία.

7. Φτιάχνουμε έναν νέο Πελάτη και βάζουμε το δείκτη του στο Μ1. Βάζουμε στο Μ1 την Παραγγελία1
8. Βάζουμε το Πελάτη Μ1 στην Ομάδα_Πελατών. (η διαφορά από το βήμα 5 είναι ότι το Μ1 στο βήμα 8 δείχνει σε άλλο αντικείμενο σε σχέση με το αντικείμενο στο βήμα 5, επειδή στο 7 αλλάξαμε το αντικείμενο που έδειχνε το Μ1 στο νέο αντικείμενο που του δώσαμε με δείκτη).

Επειδή έχουμε χρησιμοποιήσει το Για Αυτό { } το οποίο στο πέρας εκτέλεσης καθαρίζει όλα τα ονόματα που δημιουργήθηκαν, τα Παραγγελία, Παραγγελία1 και Μ1 θα διαγραφούν. Επειδή όμως οι δείκτες που κράταγαν μέχρι τέλους είναι ήδη κρατημένοι (εκεί που τους έχουμε βάλει), η διαγραφή δεν θα επιφέρει κάτι στα αντικείμενα. Μας μένει μόνο η **Ομάδα_Πελατών**. Αυτός ο δείκτης κρατάει όλο το δένδρο (ή πλεκτό). Όπως στο πλεκτό αν τραβήξουμε τη κλωστή θα χαλάσει, έτσι και εδώ αν καταργηθεί ο δείκτης, πχ γράφοντας στη μεταβλητή Ομάδα_Πελατών έναν άλλο δείκτη ή τον Μηδενικό δείκτη, ή αν διαγραφεί λόγο τερματισμού του τρέχοντος τμήματος (κάπου τρέχει όλο το πρόγραμμα του παραδείγματος, και αυτό είναι ένα τμήμα), τότε θα διαγραφούν όλα τα αντικείμενα που βρίσκονται σε αυτό. Αυτό θα το δούμε στη κονσόλα της Μ2000 επειδή έχουμε βάλει σε όλα τα αντικείμενα (κλωστές) να μας δηλώνουν πότε διαγράφονται, με την μέθοδο Διαγραφή.

Τώρα έχουμε όλα τα στοιχεία που θέλουμε να μας τα επισκεφθεί ο επισκέπτης, ένα αντικείμενο της κλάσης Γενική_Αναφορά που κληρονομεί από τη κλάση Επισκέπτης. Για να ξεκινήσει η επίσκεψη αρκεί στο αντικείμενο Ομάδα_Πελατών να γίνει αποδοχή, δηλαδή να εκτελεστεί η μέθοδος αποδοχή η οποία παίρνει τον δείκτη ενός Επισκέπτη (όχι ειδικά του Γενική_Αναφορά). Έτσι μπορούμε να έχουμε πολλούς διαφορετικούς επισκέπτες, και καθένας να δημιουργεί την δική του αναφορά αποτελέσματος.

Στο παράδειγμα θα πάρουμε την παρακάτω αναφορά:

```
Πελάτης: Bob
Παραγγελία : 1001
    1. AZX100
    2. ZZ12-23
Παραγγελία : 1002
    1. AZX101
Πελάτης: John
Παραγγελία : 1003
    1. KKX112
    2. BZ212-6
    3. BZ212-7
```

Η Γενική_Αναφορά φτιάχνει αυτό το κείμενο και η μέθοδος Εμφάνισε_Αποτελέσματα στέλνει την αναφορά στο πρόχειρο (και έτσι αντιγράφηκε στο άρθρο), και παράλληλα το εμφανίζει και στην κονσόλα.

```
Κλάση Επισκέπτης {
Ιδιωτικό:
    Τμήμα Επισκέψιμο_Στοιχείο (κάτι ως Στοιχείο) {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
    Τμήμα Επισκέψιμη_Παραγγελία (κάτι ως Παραγγελία) {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
    Τμήμα Επισκέψιμος_Πελάτης (κάτι ως Πελάτης) {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
Δημόσιο:
    Τμήμα Τελικό Επίσκεψη (κάτι ως *Επισκέψιμη){
        Αν κάτι είναι τύπος Στοιχείο Τότε
            .Επισκέψιμο_Στοιχείο κάτι
        Αλλιώς.Αν κάτι είναι τύπος Παραγγελία Τότε
            .Επισκέψιμη_Παραγγελία κάτι
        Αλλιώς.Αν κάτι είναι τύπος Πελάτης Τότε
            .Επισκέψιμος_Πελάτης κάτι
        Τέλος Αν
    }
}
Κλάση Επισκέψιμη {
    Τμήμα Αποδοχή (Επισκέπτης ως *Επισκέπτης){
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
}
Κλάση Στοιχείο ως Επισκέψιμη {
    Ιδιότητα Όνομα$ {Αξία}
    Τμήμα Τελικό Αποδοχή (Επισκέπτης ως *Επισκέπτης){
    \\ δεν κάνει τίποτα
    }
    Διαγραφή {
        Τύπωσε "Το στοιχείο ";.Όνομα$;" Διαγράφτηκε"
    }
Κλάση:
    Τμήμα Στοιχείο (.[Όνομα]$) {
    }
}
Κλάση Παραγγελία ως Επισκέψιμη {
Ιδιωτικό:
    Στοιχεία=Ουρά
Δημόσιο:
    Ιδιότητα Όνομα$ {Αξία}
    Τμήμα Τελικό Αποδοχή (Επισκέπτης ως *Επισκέπτης){
        στοιχείο=Κάθε(.Στοιχεία)
        Ενώ στοιχείο
            Επισκέπτης=>Επίσκεψη Εκφρ(στοιχείο)
        Τέλος Ενώ
    }
```

```
    Διαγραφή {
        Τύπωσε "Η Παραγγελία ";.Όνομα$;" Διαγράφτηκε"
    }
    Τμήμα Βάλε_Στοιχείο (ένα_Όνομα$){
        Προσθήκη .Στοιχεία, ένα_Όνομα$:=Δείκτης(Στοιχείο(ένα_Όνομα$))
    }
Κλάση:
    Τμήμα Παραγγελία (.[Όνομα]$, Όνομα_Στοιχείου$="") {
        Αν Όνομα_Στοιχείου$<>"" Τότε .Βάλε_Στοιχείο Όνομα_Στοιχείου$
    }
}
Κλάση Πελάτης ως Επισκέψιμη {
Ιδιωτικό:
    Παραγγελίες=Λίστα
Δημόσιο:
    Ιδιότητα Όνομα$ {Αξία}
    Τμήμα Τελικό Αποδοχή (Επισκέπτης ως *Επισκέπτης){
        στοιχείο=Κάθε(.Παραγγελίες)
        Ενώ στοιχείο
            Επισκέπτης=>Επίσκεψη Εκφρ(στοιχείο)
        Τέλος Ενώ
    }
    Τμήμα Βάλε_Παραγγελία (Παραγγελία ως *Παραγγελία) {
        Προσθήκη .Παραγγελίες, Παραγγελία=>Όνομα$:=Παραγγελία
    }
    Διαγραφή {
        Τύπωσε "Ο Πελάτης ";.Όνομα$;" Διαγράφτηκε"
    }
Κλάση:
    Τμήμα Πελάτης (.[Όνομα]$) {
    }
}
Κλάση Ομάδα_Πελατών ως Επισκέψιμη {
Ιδιωτικό:
    Πελάτες=Λίστα
Δημόσιο:
    Τμήμα Τελικό Αποδοχή (Επισκέπτης ως *Επισκέπτης){
        στοιχείο=Κάθε(.Πελάτες)
        Ενώ στοιχείο
            Επισκέπτης=>Επίσκεψη Εκφρ(στοιχείο)
        Τέλος Ενώ
    }
    Τμήμα Βάλε_Πελάτη (Πελάτης ως *Πελάτης) {
        Προσθήκη .Πελάτες, Πελάτης=>Όνομα$:=Πελάτης
    }
    Διαγραφή {
        Τύπωσε "Η Ομάδα Πελατών Διαγράφτηκε"
    }
}
Κλάση Γενική_Αναφορά ως Επισκέπτης {
```

```
Ιδιωτικό:
    Στοιχείου_Νούμερο, Παραγγελίας_Νούμερο, Πελάτη_Νούμερο
    Έγγραφο Αποτελέσματα$
    Τμήμα Τελικό Επισκέψιμο_Στοιχείο (κάτι ως *Στοιχείο) {
        .Στοιχείου_Νούμερο++
        .Αποτελέσματα$<=Μορφή$("{0::-6}. {1}",.Στοιχείου_Νούμερο, κάτι=>Όνομα$)+{
        }
    }
    Τμήμα Τελικό Επισκέψιμη_Παραγγελία (κάτι ως *Παραγγελία) {
        .Παραγγελίας_Νούμερο++
        .Στοιχείου_Νούμερο<=0
        .Αποτελέσματα$<=Μορφή$("Παραγγελία : {0}",κάτι=>Όνομα$)+{
        }
        κάτι=>Αποδοχή Δείκτης(Αυτό)
    }
    Τμήμα Τελικό Επισκέψιμος_Πελάτης (κάτι ως *Πελάτης) {
        .Πελάτη_Νούμερο++
        .Παραγγελίας_Νούμερο<=0
        .Αποτελέσματα$<=Μορφή$("Πελάτης: {0}", κάτι=>Όνομα$)+{
        }
        κάτι=>Αποδοχή Δείκτης(Αυτό)
    }
Δημόσιο:
    Τμήμα Εμφάνισε_Αποτελέσματα {
        Αναφορά .Αποτελέσματα$
        Πρόχειρο .Αποτελέσματα$
        Καθαρό .Αποτελέσματα$
        .Πελάτη_Νούμερο<=0
    }
}
Ομάδα_Πελατών->Ομάδα_Πελατών()

Για Αυτό {
    Παραγγελία1=Παραγγελία("1001", "AZX100")
    Παραγγελία1.Βάλε_Στοιχείο "ZZ12-23"
    Μ1=Πελάτης("Bob")
    Μ1.Βάλε_Παραγγελία Δείκτης((Παραγγελία1))
    Παραγγελία1=Παραγγελία("1002", "AZX101")
    Μ1.Βάλε_Παραγγελία Δείκτης((Παραγγελία1))
    Ομάδα_Πελατών=>Βάλε_Πελάτη Δείκτης((Μ1))
    Παραγγελία1=Παραγγελία("1003", "KKX112")
    Παραγγελία1.Βάλε_Στοιχείο "BZ212-6"
    Παραγγελία1.Βάλε_Στοιχείο "BZ212-7"
    Μ1=Πελάτης("John")
    Μ1.Βάλε_Παραγγελία Δείκτης((Παραγγελία1))
    Ομάδα_Πελατών=>Βάλε_Πελάτη Δείκτης((Μ1))
}
Γενική_Αναφορά->Γενική_Αναφορά()
```

```
Ομάδα_Πελατών=>Αποδοχή Γενική_Αναφορά
Γενική_Αναφορά=>Εμφάνισε_Αποτελέσματα
```

***Πρόγραμμα 61**: Μοτίβο Visitor*

Μέθοδος Διαγραφή

Αμέσως μετά την εκτέλεση του προγράμματος (είναι σε ένα τμήμα έστω Α) βλέπουμε όλες τις διαγραφές των αντικειμένων, γιατί η Ομάδα_Πελατών διαγράφεται ως μεταβλητή, και κατά τη διαγραφή ενημερώνει το αντικείμενο να κατεβάσει κατά ένα έναν εσωτερικό απαριθμητή συνδέσεων, και επειδή δεν υπάρχει άλλη σύνδεση, θα μηδενιστεί άρα θα ξεκινήσει η διαγραφή και θα κληθεί το μέλος Διαγραφή. Αν είχαμε επώνυμη ομάδα δεν θα γίνονταν αυτό, επειδή στις επώνυμες εξ ορισμού έχει βγει ο αυτοματισμός της κλήσης της Διαγραφής. Όμως επειδή οι επώνυμες ομάδες έχουν πάντα μοναδικό δείκτη, η διαγραφή του ονόματος προκαλεί άμεσα τη διαγραφή του αντικειμένου.

Μπορούμε να εκκινήσουμε την μέθοδο Διαγραφή με την Καθαρό Όνομα_Ομάδας.

Ο λόγος που δεν γίνεται αυτόματη διαγραφή του επώνυμου είναι αυτός: Σε ένα αντικείμενο κλειστό όταν καλείται αυτόματα η Διαγραφή το αντικείμενο ανοίγει σε ένα επώνυμο με άλλο δείκτη αλλά με ό,τι έχει το προς διαγραφή αντικείμενο, δηλαδή στην ουσία το επώνυμο είναι εκ των πραγμάτων το κλειστό αντικείμενο παρόλο που έχει άλλο πραγματικό δείκτη (που σημαίνει ότι είναι δυο διαφορετικά αντικείμενα). Αυτό το επώνυμο αντικείμενο έχει όνομα που του δίνει ο διερμηνευτής. Έτσι στο προς διαγραφή αντικείμενο δεν δίνουμε δείκτη και δεν πρέπει γιατί το σύστημα θα διαγράψει τη μνήμη του μετά το τέλος της μεθόδου Διαγραφή. Χρησιμοποιούμε το επώνυμο για να κάνουμε ό,τι χρειάζεται να γίνει στην Διαγραφή.

Μοτίβο Interpreter

Το μοτίβο Interpreter, ή Διερμηνευτής, χρησιμοποιείται για να μπορεί να διαχειρίζεται μια διερμηνεία από ένα δένδρο κόμβων. Η διερμηνεία δίνει ένα δένδρο το οποίο μπορεί να εκτιμηθεί, δηλαδή να βγει ένα αποτέλεσμα.

Θα δούμε δυο παραδείγματα για το μοτίβο Διερμηνευτή. Και στα δυο παραδείγματα θέλουμε το αποτέλεσμα του 3 3 4 + 8 + + (**ανάποδη πολωνική γραφή** για μαθηματικές εκφράσεις, όπου δεν χρησιμοποιούνται παρενθέσεις). Η συγκεκριμένη έκφραση εκτελείται με επόμενο αποτέλεσμα 3 7 8 + + μετά 3 15 + και μετά 18 και αυτό είναι το αποτέλεσμα (δεν υπάρχει άλλος τελεστής +). Οι αριθμοί λέγονται τερματικά ή φύλλα, και ο τελεστής λέγεται μη τερματικό ή κλαδί. Στο ελληνικό παράδειγμα εδώ θα χρησιμοποιήσουμε τα ονόματα Φύλλο και Κλαδί. Όταν υπάρχει κλαδί πρέπει να υπάρχουν δυο φύλλα, ή δυο κλαδιά ή ένα κλαδί και ένα φύλλο. Η πρόοδος της διερμηνείας τελειώνει επιτυχώς όταν δεν υπάρχει κλαδί, και το τελικό στοιχείο είναι φύλλο. Η τιμή του είναι η τιμή επιστροφής

Στο πρώτο παράδειγμα έχουμε δυο κλάσεις. Τη κλάση Περιεχόμενο και τη κλάση Έκφραση. Από την πρώτη κλάση φτιάχνουμε ένα δείκτη το Περιεχόμενο. Η κλάση έχει τις μεθόδους ΠρόσθεσεΦύλλο και ΠρόσθεσεΚλαδί. Για κάθε εισαγωγή βάζουμε είδος προσθήκης με έναν αριθμό και το σύμβολο$ που δίνουμε ως όρισμα. Φτιάχνουμε ένα δεύτερο αντικείμενο από τη

κλάση Έκφραση ως δείκτης Έκφραση. Καλούμε την μέθοδο Διερμήνευσε με όρισμα το δείκτη σε Περιεχόμενο, που έχουμε ως Περιεχόμενο.

Αξίζει να περιγραφεί η εσωτερική λειτουργία της Διερμήνευσε. Πρώτα αδειάζει ο τρέχον σωρός τιμών. Οι συναρτήσεις έχουν δικό τους σωρό τιμών, και αν είχαμε βάλει δυο ή περισσότερα ορίσματα θα είχαν μπει στο σωρό τιμών χωρίς να είναι λάθος. Έτσι αδειάζοντας το σωρός ξέρουμε ότι ξεκινάμε με κενό σωρό. Μετά παίρνουμε στην μεταβλητή στοιχεία (τοπική) την τιμή Περιεχόμενο=>στοιχεία, το οποίο είναι μια ιδιότητα μόνο για ανάγνωση, και μας δίνει αντίγραφο του σωρού τιμών που πραγματικά κρατάει. Ακολουθεί μια ακολουθία επανάληψης όσο το μήκος του σωρού στοιχεία δεν είναι μηδέν, δηλαδή όσο υπάρχουν στοιχεία. Στην ακολουθία το πρώτο πράγμα που κάνουμε είναι να ανοίξουμε το σωρό στοιχεία και να διαβάσουμε τα δυο πρώτα μέρη του πρώτου στοιχείου, το είδος και το σύμβολο$. Μόλις βγούμε από τη δομή Σωρός {} επανέρχεται ο τρέχον σωρός. Ανάλογα τι μας λέει το είδος κάνουμε ένα από τα παρακάτω: Αν το είδος είναι 1 βάζουμε την αριθμητική τιμή που λέει το σύμβολο στο τρέχον σωρό, στη κορυφή του. Αν το είδος είναι 2 τότε τραβάμε τους δυο πρώτους αριθμούς από την κορυφή και βάζουμε το αποτέλεσμα στη κορυφή. Στο τέλος της ακολουθίας η τιμή που μένει στο σωρό τιμών είναι το αποτέλεσμα, και την διαβάζουμε με την Αριθμός (Η αριθμός τραβάει την τιμή από το σωρό, όπως μια Διάβασε Α). Στον σωρό στοιχεία έχουμε ζεύγη είδος και σύμβολο$, και έχουν μπει με την Σειρά, δηλαδή ως FIFO. Στον τρέχον σωρό της μεθόδου Διερμήνευση κάνουμε χρήση του σωρού τιμών ως LIFO, ασχολούμαστε με τα τελευταία που μπαίνουν.

Εμφάνιση Σωρού Τιμών

Για να φανεί πώς γίνεται η διαδικασία, στην εντολή Σημ Σωρός πάμε το δρομέα μετά το Σημ και πατάμε enter, έτσι η Σωρός πάει σε νέα γραμμή και θα εκτελεστεί. Η εντολή Σωρός χωρίς ορίσματα δείχνει το τρέχον σωρό τιμών στη κονσόλα. Έτσι βλέπουμε πως διαμορφώνεται σε κάθε επανάληψη.

```
3
3       3
4       3       3
7       3
8       7       3
15      3
18
```

```
Κλάση Περιεχόμενο {
    \\ η ιδιότητα είναι μακροεντολή,
    \\ η οποία φτιάχνει ό,τι χρειάζεται με εντολές της Μ2000
    \\ και δημιουργεί την [στοιχεία] ως ιδιωτική στην κλάση.
    Ιδιότητα στοιχεία {
        Αξία {
            Αξία=Σωρός(Αξία) ' Αντίγραφο το σωρού
        }
```

```
    }=Σωρός
    Τμήμα ΠρόσθεσεΦύλλο (σύμβολο$) {
        \\ αν είχαμε το .στοιχεία τότε η προσθήκη...
        \\ θα πήγαινε σε ένα σωρό που θα χάνονταν!
        \\ έτσι διαβάζουμε την ιδιωτική μεταβλητή
        \\ που αντιστοιχεί στην ιδιότητα.
        Σωρός .[στοιχεία] {Σειρά 1, σύμβολο$}
    }
    Τμήμα ΠρόσθεσεΚλαδί (σύμβολο$) {
        Σωρός .[στοιχεία] {Σειρά 2, σύμβολο$}
    }
}
Κλάση Εκφραση {
    Συνάρτηση Τελική Διερμήνευσε(Περιεχόμενο as *Περιεχόμενο) {
        Άδειασε ' καθαρίζει τον τρέχον σωρό
        στοιχεία=Περιεχόμενο=>στοιχεία
        Ενώ Μήκος(στοιχεία)>0
            Σωρός στοιχεία {
                Διάβασε είδος, σύμβολο$
            }
            Αν είδος=1 Τότε
                Βάλε Τιμή(σύμβολο$)
            Αλλιώς.Αν σύμβολο$="+" Τότε
                Βάλε Αριθμός+Αριθμός
            Τέλος Αν
            Σημ Σωρός
        Τέλος Ενώ
        =Αριθμός
    }
}
Περιεχόμενο->Περιεχόμενο()
Για Περιεχόμενο {
    .ΠρόσθεσεΦύλλο "3"
    .ΠρόσθεσεΦύλλο "3"
    .ΠρόσθεσεΦύλλο "4"
    .ΠρόσθεσεΚλαδί "+"
    .ΠρόσθεσεΦύλλο "8"
    .ΠρόσθεσεΚλαδί "+"
    .ΠρόσθεσεΚλαδί "+"
}
Εκφραση->Εκφραση()
Τύπωσε Εκφραση=>Διερμήνευσε(Περιεχόμενο)=18
```

***Πρόγραμμα 62**: Μοτίβο Interpreter*

Επαναχρησιμοποίηση Κλάσεων

Το δεύτερο πρόγραμμα φτιάχνει το λεγόμενο AST, ή Abstract Syntax Tree, ένα δένδρο σύνταξης προς εκτέλεση. Χρησιμοποιούμε την ίδια κλάση Περιεχόμενο. Η **επαναχρησιμοποίηση** κλάσεων είναι μια καλή πρακτική. Η κλάση Έκφραση έχει μεν μια δημόσια μέθοδο Διερμήνευσε αλλά έχει διαφορετική επιστροφή. Μας επιστρέφει ένα δένδρο AST φτιαγμένο με tuple, αυτόματο πίνακα. Μπορούμε σε ένα πίνακα να βάζουμε άλλους πίνακες. Με αυτόν τον τρόπο ο πίνακας που επιστρέφουμε έχει μια τιμή το είδος και ένα ή δυο στοιχεία ακόμα ανάλογα το είδος. Το κάθε στοιχείο μπορεί να είναι είδος 1 οπότε έχει ένα αριθμό (από το σύμβολο αριθμό που δίνουμε) ή να είναι είδος 2 οπότε έχει δυο άλλα φύλλα, δηλαδή δυο άλλους πίνακες που μπορεί να είναι οποιοδήποτε είδος.

Ο τρόπος να παράγουμε το AST απαιτεί κάποιες συναρτήσεις και μια αναφορά στο σωρό με τα σύμβολα να είναι όλα ιδιωτικά. Αφού πάρουμε το αντίγραφο του σωρού από το Περιεχόμενο το αντιστρέφουμε, για να ξεκινήσουμε από τα σύμβολα προς τους αριθμούς (σαν να έχουμε απλή πολωνική γραφή). Επιστρέφουμε το αποτέλεσμα της ιδιωτικής εφαρμογή(). Σε αυτήν μπορούμε εύκολα να προσθέσουμε πράξεις. Όταν καλούμε την Πράξη_Πρόσθεσης δεν κάνουμε πρόσθεση αλλά φτιάχνουμε το κόμβο που μπορεί να έχει κόμβους ή φύλλα ή και κόμβο και φύλλο, καλώντας για κάθε θέση την εφαρμογή() ξανά! Έχουμε δηλαδή αναδρομή. Θα μπορούσαμε να βάλουμε απ' ευθείας την επιστροφή της Πράξη_Πρόσθεσης στη κλήση της αλλά εδώ την έχουμε χωριστά για να διαβάζουμε καλύτερα το πρόγραμμα. Δείτε επίσης ότι στην εφαρμογή γίνεται χρήση της ιδιωτικής στη κλάση σεΧρήση που είναι ο δείκτης στο σωρό τιμών. Αυτή η ιδιωτική είναι συνέχεια θεατή σε κάθε εκτέλεση της εφαρμογής().

Αφού πάρουμε το AST, μπορούμε να το εκτελέσουμε όσες φορές θέλουμε, με τη συνάρτηση Εκτέλεση_AST() η οποία διατρέχει τον πίνακα και όταν βρίσκει το είδος 1 γυρίζει την τιμή, αλλιώς αν βρει το είδος 2 τότε για τα στοιχεία 1 και 2 (δεύτερο και τρίτο) εκτελεί χωριστά τον εαυτό του (αναδρομική κλήση) προσθέτοντας τα δυο αποτελέσματα.

Παρατηρούμε ότι τόσο για τη δημιουργία του AST όσο και για την εκτέλεση του AST κάνουμε χρήση της αναδρομής. Στο πρώτο απλό πρόγραμμα, αποφεύγουμε την αναδρομή με χρήση του τρέχοντος σωρού.

```
Κλάση Περιεχόμενο {
    Ιδιότητα στοιχεία {
        Αξία {
            Αξία=Σωρός(Αξία)
        }
    }=Σωρός
    Τμήμα ΠρόσθεσεΦύλλο (σύμβολο$) {
        Σωρός .[στοιχεία] {Σειρά 1, σύμβολο$}
    }
    Τμήμα ΠρόσθεσεΚλαδί (σύμβολο$) {
        Σωρός .[στοιχεία] {Σειρά 2, σύμβολο$}
    }
}
```

```
Κλάση Έκφραση {
Ιδιωτικό:
    σεΧρήση=Σωρός
    Συνάρτηση Πράξη_Πρόσθεσης() {
        =(2, .εφαρμογή(), .εφαρμογή())
    }
    Συνάρτηση εφαρμογή() {
        Σωρός .σεΧρήση {
            Διάβασε σύμβολο$, είδος
        }
        Αν είδος=1 Τότε
            =(1, Τιμή(σύμβολο$))
        Αλλιώς.Αν σύμβολο$="+" Τότε
            =.Πράξη_Πρόσθεσης()
        Τέλος Αν
    }
Δημόσιο:
    Συνάρτηση Τελική Διερμήνευσε(Περιεχόμενο ως *Περιεχόμενο) {
        .σεΧρήση<=Περιεχόμενο=>στοιχεία
        Σωρός .σεΧρήση {
            Φέρε 1, -Μέγεθος.Σωρού ' αντιστρέφει τα στοιχεία
        }
        =.εφαρμογή()
    }
}
Συνάρτηση Εκτέλεση_Ast(δένδρο_AST ως πίνακας) {
    Αν δένδρο_AST#Τιμή(0)=1 Τότε
        =δένδρο_AST#Τιμή(1)
    Αλλιώς.Αν δένδρο_AST#Τιμή(0)=2 Τότε
        =Εκτέλεση_Ast(δένδρο_AST#Τιμή(1))+Εκτέλεση_Ast(δένδρο_AST#Τιμή(2))
    Τέλος Αν
}
Περιεχόμενο->Περιεχόμενο()
for Περιεχόμενο {
    .ΠρόσθεσεΦύλλο "3"
    .ΠρόσθεσεΦύλλο "3"
    .ΠρόσθεσεΦύλλο "4"
    .ΠρόσθεσεΚλαδί "+"
    .ΠρόσθεσεΦύλλο "8"
    .ΠρόσθεσεΚλαδί "+"
    .ΠρόσθεσεΚλαδί "+"
}
Έκφραση->Έκφραση()
```

```
AST=Έκφραση=>Διερμήνευσε(Περιεχόμενο)
Τύπωσε Εκτέλεση_Ast(AST)=18
```

***Πρόγραμμα 63**: Μοτίβο Interpreter (AST)*

Μοτίβο Iterator

Το μοτίβο Iterator ή Επαναλήπτης, εφαρμόζεται σε αντικείμενα που έχουν μια λίστα αντικειμένων και θέλουμε να παίρνουμε ένα προς ένα τα αντικείμενα από αυτήν. Το πως η λίστα υλοποιείται δεν είναι γνωστό, αλλά ο χειρισμός είναι ο ίδιος για κάθε υλοποίηση.

Μια κλάση ΣτιβαρόςΕπαναλήπτης κληρονομεί από την κλάση Επαναλήπτης, και υλοποιεί τις αφηρημένες μεθόδους της. Ειδικά για την M2000 λόγω των δυο κύριων τύπων τον αριθμητικό και τον αλφαριθμητικό η μέθοδος Τρέχον_Στοιχείο υπάρχει και για αλφαριθμητικά ως Τρέχον_Στοιχείο$. Σε άλλες γλώσσες που δεν ξεχωρίζουν τα ονόματα αν γυρίζουν αλφαριθμητικά, έχουμε μια για κάθε επιστροφή.

Στο παράδειγμα έχουμε δυο κλάσεις να κληρονομούν από την κλάση Επαναλήπτης. Η μία μας δείχνει ένα προς ένα τους χαρακτήρες ενός αλφαριθμητικού που δίνουμε στον κατασκευαστή της ενώ η άλλη μας δείχνει ένα προς ένα τα ορίσματα ως αριθμοί που δίνουμε στον κατασκευαστή της. Και στις δυο περιπτώσεις έχουμε το ίδιο: Φτιάχνουμε ένα αντικείμενο. Καλούμε την μέθοδο Πρώτο_Στοιχείο και μετά σε μια Ενώ Τέλος Ενώ ελέγχουμε την μέθοδο Επόμενο_Στοιχείο και μέσα στο σώμα της ακολουθίας επανάληψης χρησιμοποιούμε την μέθοδο Τρέχον_Στοιχείο για να πάρουμε αριθμό ή την Τρέχον_Στοιχείο$ για να πάρουμε αλφαριθμητικό. Αν κάνουμε λάθος στην τύπο της κλήσης του τρέχοντος στοιχείου θα ενεργοποιηθεί το Λάθος στην αντίστοιχη μέθοδο που θα λέει ότι δεν έχει υλοποιηθεί!

```
Κλάση Επαναλήπτης {
    Τμήμα Πρώτο_Στοιχείο() {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
    Συνάρτηση Επόμενο_Στοιχείο() {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
    Συνάρτηση Τρέχον_Στοιχείο() {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
    Συνάρτηση Τρέχον_Στοιχείο$() {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
}
Κλάση ΣτιβαρόςΕπαναλήπτης ως Επαναλήπτης {
Ιδιωτικό:
    τιμή_αλφαριθμητική$, μετρητής=0
Δημόσιο:
    Τμήμα Πρώτο_Στοιχείο() {
```

```
        .μετρητής<=0
    }
    Συνάρτηση Επόμενο_Στοιχείο() {
        Αν .μετρητής<Μήκος(.τιμή_αλφαριθμητική$) Τότε .μετρητής++ : =Αληθής
    }
    Συνάρτηση Τρέχον_Στοιχείο$() {
        =Μεσ$(.τιμή_αλφαριθμητική$, .μετρητής, 1)
    }
Κλάση:
    Τμήμα ΣτιβαρόςΕπαναλήπτης (.τιμή_αλφαριθμητική$) {
    }
}
Κλάση ΣτιβαρόςΕπαναλήπτης1 ως Επαναλήπτης {
Ιδιωτικό:
    τιμές=(,), μετρητής=0, μέγιστη_τιμή
Δημόσιο:
    Τμήμα Πρώτο_Στοιχείο() {
        .μετρητής<=-1
    }
    Συνάρτηση Επόμενο_Στοιχείο() {
        Αν .μετρητής<.μέγιστη_τιμή Τότε .μετρητής++ : =Αληθής
    }
    Συνάρτηση Τρέχον_Στοιχείο() {
        =.τιμές#Τιμή(.μετρητής)
    }
Κλάση:
    Τμήμα ΣτιβαρόςΕπαναλήπτης1 {
        .τιμές<=Πίνακας([])
        .μέγιστη_τιμή<=Μήκος(.τιμές)-1
    }
}
Ε1=ΣτιβαρόςΕπαναλήπτης("ABCDEF")
Ε1.Πρώτο_Στοιχείο
Ενώ Ε1.Επόμενο_Στοιχείο()
    Τύπωσε Ε1.Τρέχον_Στοιχείο$()
Τέλος Ενώ
Ε2=ΣτιβαρόςΕπαναλήπτης1(10,3,12,5,2,1,300,-5,3)
Ε2.Πρώτο_Στοιχείο
Ενώ Ε2.Επόμενο_Στοιχείο()
    Τύπωσε Ε2.Τρέχον_Στοιχείο()
Τέλος Ενώ
```

***Πρόγραμμα 63**: Μοτίβο Iterator*

Μοτίβο Strategy

Το μοτίβο Strategy αλλάζει συμπεριφορά σε ένα αντικείμενο αλλάζοντας ένα εσωτερικό αντικείμενο. Βασίζεται σε μια αφηρημένη κλάση (ή διεπαφή σε άλλες γλώσσες), και μπαίνει σε μια κλάση όπως στο παράδειγμα στη κλάση Ρομπότ όπου μπορεί να αλλαχθεί. Στο παράδειγμα ορίζουμε τρεις συμπεριφορές που κληρονομούν από την κλάση Είδος_Συμπεριφοράς. Και οι τρεις έχουν επακριβώς ότι δηλώνεται στο Είδος_Συμπεριφοράς. Αν είχαμε διαφοροποιήσεις τότε θα έπρεπε να χρησιμοποιήσουμε στο Ρομπότ δείκτη σε ομάδα, επειδή μια ομάδα που μπαίνει επώνυμα σε άλλη διατηρεί τα μέλη της ως αυτά που δηλώθηκαν, και μόνο κατά την κατασκευή της κλάσης που την περιέχει μπορεί να προσθέσει στοιχεία ή να αλλάξει τιμές στα στοιχεία της. Η διαφοροποίηση περιέχεται αμέσως μετά το παράδειγμα.

Η έξοδος του προγράμματος δίνει τα παρακάτω:

```
Κανονική Συμπεριφορά: Αγνόησε τα άλλα Ρομπότ
Επιλογή: 0 για Ρομπότ R2
Επιθετική Συμπεριφορά: Επίθεση σε άλλα Ρομπότ
Επιλογή: 1 για Ρομπότ R2
Αμυντική Συμπεριφορά: Αν δεις άλλα Ρομπότ φεύγα
Επιλογή:-1 για Ρομπότ R2
```

```
Κλάση Είδος_Συμπεριφοράς {
    Συνάρτηση εντολήΜετακίνησης() {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
}

Κλάση Επιθετική_Συμπεριφορά ως Είδος_Συμπεριφοράς {
    Συνάρτηση εντολήΜετακίνησης() {
        Τύπωσε "Επιθετική Συμπεριφορά: Επίθεση σε άλλα Ρομπότ"
        =1
    }
}


Κλάση Αμυντική_Συμπεριφορά ως Είδος_Συμπεριφοράς {
    Συνάρτηση εντολήΜετακίνησης() {
        Τύπωσε "Αμυντική Συμπεριφορά: Αν δεις άλλα Ρομπότ φεύγα"
        =-1
    }
}


Κλάση Κανονική_Συμπεριφορά ως Είδος_Συμπεριφοράς {
    Συνάρτηση εντολήΜετακίνησης() {
```

```
            Τύπωσε "Κανονική Συμπεριφορά: Αγνόησε τα άλλα Ρομπότ"
            =0
        }
    }

    Κλάση Ρομπότ {
    Ιδιωτικό:
        Είδος_Συμπεριφοράς Συμπεριφορά
    Δημόσιο:
        Ιδιότητα Όνομα$ {Αξία}
        Τμήμα Βάλε_Συμπεριφορά (κάτι ως Είδος_Συμπεριφοράς) {
            .Συμπεριφορά<=κάτι
        }
        Συνάρτηση Πάρε_Συμπεριφορά() {
            =.Συμπεριφορά
        }
        Τμήμα Κίνηση {
            μήνυμα$=Γραφή$(.Συμπεριφορά.εντολήΜετακίνησης())
            Τύπωσε "Επιλογή:"+μήνυμα$+" για Ρομπότ "+.[Όνομα]$
        }
    Κλάση:
        Τμήμα Ρομπότ (.[Όνομα]$) {
            .Συμπεριφορά<=Κανονική_Συμπεριφορά()
        }
    }
    P1=Ρομπότ("R2")
    P1.Κίνηση
    P1.Βάλε_Συμπεριφορά Επιθετική_Συμπεριφορά()
    P1.Κίνηση
    P1.Βάλε_Συμπεριφορά Αμυντική_Συμπεριφορά()
    P1.Κίνηση
```

***Πρόγραμμα 64**: Μοτίβο Strategy*

Η αλλαγή στη Ρομπότ για να μετατρέπει τις συμπεριφορές που δίνουμε σε δείκτες. Σε περίπτωση αντιγραφής του ρομπότ, θα έχουμε αντιγραφή δείκτη. Οι εσωτερικές ομάδες σε δείκτες δεν μπορούν να δουν σε καμία περίπτωση τα στοιχεία της γονικής ομάδας.

```
Κλάση Ρομπότ {
Ιδιωτικό:
    Συμπεριφορά=Δείκτης()
Δημόσιο:
    Ιδιότητα Όνομα$ {Αξία}
    Τμήμα Βάλε_Συμπεριφορά (κάτι ως Είδος_Συμπεριφοράς) {
```

```
        .Συμπεριφορά<=Δείκτης((κάτι)) 'μετατροπή σε δείκτη
    }
    Συνάρτηση Πάρε_Συμπεριφορά() {
        =Ομάδα(.Συμπεριφορά) ' μετατροπή από δείκτη
    }
    Τμήμα Κίνηση {
        μήνυμα$=Γραφή$(.Συμπεριφορά=>εντολήΜετακίνησης())
        Τύπωσε "Επιλογή:"+μήνυμα$+" για Ρομπότ "+.[Όνομα]$
    }
Κλάση:
    Τμήμα Ρομπότ (.[Όνομα]$) {
        .Συμπεριφορά<=Δείκτης(Κανονική_Συμπεριφορά())
    }
}
```

***Πρόγραμμα 65**: Μοτίβο Strategy - Κλάση Ρομπότ με δείκτες*

Μοτίβο Command

Στο μοτίβο Command μας ενδιαφέρει να σώσουμε μηνύματα τα οποία θα εκτελεστούν αργότερα αν δεν μπορούν να εκτελεστούν άμεσα. Τα μηνύματα τα λέμε εδώ Εντολές.

Στο πρόγραμμα υπάρχει μια κλάση Εντολή, από την οποία θα φτιάξουμε στιβαρές κλάσεις, κάθε στιβαρή κλάση διαχειρίζεται αντικείμενα ενός υποδοχέα, τέλος υπάρχει μια κλάση σε ρόλο εκτελεστή, εδώ η κλάση Πράκτορας, αυτή κρατάει μια ουρά εντολών. Κάθε εντολή περιέχει μια κλήση σε μια μέθοδο του υποδοχέα. Αν μια εντολή δεν μπορεί να εκτελεστεί τότε περιμένει να εκτελεστεί. Κατά την εκτέλεση ο Πράκτορας καλεί την μέθοδο Βάλε_Εντολή_Από_Ουρά μέχρι να γυρίσει ψευδές, δηλαδή να έχει αδειάσει η ουρά. Επειδή την ώρα που βάζουμε εντολές στον Πράκτορα, δοκιμάζει αν τρέχουν αμέσως ενδέχεται η ουρά να είναι άδεια όταν πάμε στη φάση εκτέλεσης από την ουρά.

```
    Κλάση Εντολή {
        Συνάρτηση Εκτέλεσε {
            Λάθος "Δεν έχει υλοποιηθεί ακόμα"
        }
    }
    \\ Κλάση υποδοχέας
    Κλάση Χρηματιστήριο {
        Τμήμα Αγοράζω {
            Τύπωσε "Θες να αγοράσεις μετοχές"
        }
        Τμήμα Πουλάω {
            Τύπωσε "θες να πουλήσεις μετοχές"
        }
    }
```

```
\\ Κλάση εκτελεστής
Κλάση Πράκτορας {
Ιδιωτικό:
    Ουρά_Εντολών=Σωρός
Δημόσιο:
    Τμήμα Βάλε_Εντολή (Εντολή ως *Εντολή) {
        Σωρός .Ουρά_Εντολών {
            Σειρά Εντολή
            Διάβασε Εντολή1
            Αν Όχι Εντολή1=>Εκτέλεσε() Τότε Βάλε Εντολή1
        }
    }
        Συνάρτηση Βάλε_Εντολή_Από_Ουρά {
        Σωρός .Ουρά_Εντολών {
            Αν Κενό Τότε Εξοδος
            =Αληθές
            Διάβασε Εντολή1
            Αν Όχι Εντολή1=>Εκτέλεσε() Τότε Βάλε Εντολή1
            Αν Κενό Τότε =Ψευδής
        }
    }
}
\\ ΣτιβαρήΕντολή
Κλάση Εντολή_Αγοράζω_Μετοχές ως Εντολή {
Ιδιωτικό:
    Χρηματιστήριο Μετοχές
Δημόσιο:
    Συνάρτηση Εκτέλεσε() {
        .Μετοχές.Αγοράζω
        =Αληθές
    }
Κλάση:
    Τμήμα Εντολή_Αγοράζω_Μετοχές (οιΜετοχές ως Χρηματιστήριο) {
        .Μετοχές<=οιΜετοχές
    }
}
\\ ΣτιβαρήΕντολή
Κλάση Εντολή_Πουλάω_Μετοχές ως Εντολή {
Ιδιωτικό:
    Χρηματιστήριο Μετοχές
Δημόσιο:
    Συνάρτηση Εκτέλεσε {
        Παράγοντας=Τυχαίος(1,10)<3
        Αν Παράγοντας Τότε
```

```
            .Μετοχές.Πουλάω : = Αληθές
        Αλλιώς
            Τύπωσε "Δεν πουλάω τώρα, περιμένω!"
        Τέλος Αν
    }
Κλάση:
    Τμήμα Εντολή_Πουλάω_Μετοχές (οιΜετοχές ως Χρηματιστήριο) {
        .Μετοχές<=οιΜετοχές
    }
}
Πράκτορας=Πράκτορας()
Μετοχές=Χρηματιστήριο()
Εντολή_Αγοράς->Εντολή_Αγοράζω_Μετοχές(Μετοχές)
Εντολή_Πώλησης-> Εντολή_Πουλάω_Μετοχές(Μετοχές)
Πράκτορας.Βάλε_Εντολή Εντολή_Αγοράς
Πράκτορας.Βάλε_Εντολή Εντολή_Πώλησης
Ενώ Πράκτορας.Βάλε_Εντολή_Από_Ουρά()
    Αναμονή 100
Τέλος Ενώ
```

***Πρόγραμμα 66**: Μοτίβο Command*

Μοτίβο State

Το μοτίβο State (κατάσταση) φαίνεται ίδιο με το μοτίβο Strategy, αλλά διαφέρει ως προς την αλλαγή συμπεριφοράς. Ενώ στο Strategy βάζαμε τη συμπεριφορά που θέλαμε εδώ η συμπεριφορά αλλάζει από την τρέχουσα συμπεριφορά. Μπορεί η αλλαγή να συμβεί μετά από μερικές φορές που θα χρησιμοποιηθεί μια συμπεριφορά.

Στο παράδειγμα έχουμε την αφηρημένη Τύπος_Κατάστασης με μια μέθοδο για υλοποίηση την Γράψε_Όνομα. Φτιάχνουμε δυο κλάσεις ως στιβαρές κλάσεις που κληρονομούν από τη Τύπος_Κατάστασης. Φτιάχνουμε την Κατάσταση_Πεζά και την Κατάσταση_Πολλαπλά_Κεφαλαία. Η τελευταία εκτελείται δυο φορές πριν δώσει νέα κατάσταση.

Το πρόγραμμα δίνει αυτήν την έξοδο:

```
monday
TUESDAY
WEDNESDAY
thursday
FRIDAY
SATURDAY
sunday
```

```
Κλάση Τύπος_Κατάστασης {
    Συνάρτηση Γράψε_Όνομα (όνομα$) {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
}

Κλάση Κατάσταση_Πεζά ως Τύπος_Κατάστασης {
    Συνάρτηση Γράψε_Όνομα (όνομα$) {
        Τύπωσε Πεζ$(όνομα$)
        =Δείκτης(Κατάσταση_Πολλαπλά_Κεφαλαία())
    }
}


Κλάση Κατάσταση_Πολλαπλά_Κεφαλαία ως Τύπος_Κατάστασης {
Ιδιωτικό:
    Μετρητής = 0
Δημόσιο:
    Συνάρτηση Γράψε_Όνομα (όνομα$) {
        Τύπωσε Κεφ$(όνομα$)
        .Μετρητής++
        Αν .Μετρητής=2 Τότε
            =Δείκτης(Κατάσταση_Πεζά())
            .Μετρητής<=0
        Αλλιώς
            =Δείκτης()
        Τέλος Αν
    }
}


Κλάση ΠεριεχόμενοΚατάστασης {
Ιδιωτικό:
    Τύπος_Κατάστασης=Δείκτης()
Δημόσιο:
    Τμήμα Γράψε_Όνομα(όνομα$) {
        τι=.Τύπος_Κατάστασης=>Γράψε_Όνομα(όνομα$)
        Αν τι είναι τύπος Τύπος_Κατάστασης Τότε .Τύπος_Κατάστασης<=τι
         }
Κλάση:
    Τμήμα ΠεριεχόμενοΚατάστασης {
        .Τύπος_Κατάστασης->Κατάσταση_Πεζά()
    }
}
Τμήμα ΔείξεΚατάσταση {
    Περιεχόμενο=ΠεριεχόμενοΚατάστασης()
```

```
    Περιεχόμενο.Γράψε_Όνομα "Monday"
    Περιεχόμενο.Γράψε_Όνομα "Tuesday"
    Περιεχόμενο.Γράψε_Όνομα "Wednesday"
    Περιεχόμενο.Γράψε_Όνομα "Thursday"
    Περιεχόμενο.Γράψε_Όνομα "Friday"
    Περιεχόμενο.Γράψε_Όνομα "Saturday"
    Περιεχόμενο.Γράψε_Όνομα "Sunday"
}
ΔείξεΚατάσταση
```

***Πρόγραμμα 67**: Μοτίβο State*

Μοτίβο Template Method

Στο μοτίβο Template Method, η σχέδιο μεθόδων, έχουμε μια μέθοδο που εκτελεί μια σειρά μεθόδων, οι οποίες μπορούν να διαφοροποιηθούν. Η μέθοδος Template δεν διαφοροποιείται, και ανήκει σε μια κλάση που κληρονομείται από στιβαρές κλάσεις, όπου οι εσωτερικές μέθοδοι, που δεν έχουν υλοποιηθεί, υλοποιούνται.

Στο παράδειγμα η κλάση Ταξίδι υλοποιεί την κλάση Πρόγραμμα_Ταξιδιού, με μια σειρά ιδιωτικών μεθόδων, μέσα σε μια τελική μέθοδο Πρόγραμμα_Ταξιδιού. Οι ιδιωτικές μέθοδοι δεν έχουν υλοποιηθεί. Φτιάχνουμε τις κλάσεις ΠακέτοΑ και ΠακέτοΒ, που κληρονομούν από το Ταξίδι, όπου σε αυτές υλοποιούνται οι μέθοδοι όπως τις χρειάζεται η μέθοδος Πρόγραμμα_Ταξιδιού. Φτιάχνουμε το αντικείμενο Ταξίδι από το ΠακέτοΑ και εκτελούμε το Πρόγραμμα_Ταξιδιού, και αμέσως μετά αλλάζουμε το Ταξίδι δίνοντας το ΠακέτοΒ και εκτελούμε πάλι το Πρόγραμμα_Ταξιδιού.

```
Κλάση Ταξίδι {
Δημόσιο:
    Τμήμα Τελικό Πρόγραμμα_Ταξιδιού {
        .ΤαξίδιΠροςΠροορισμό
        .Μέρα_Πρώτη
        .Μέρα_Δεύτερη
        .Μέρα_Τρίτη
        .ΤαξίδιΕπιστροφής
    }
Ιδιωτικό:
    Τμήμα ΤαξίδιΠροςΠροορισμό {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
    Τμήμα Μέρα_Πρώτη {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
    Τμήμα Μέρα_Δεύτερη() {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
    Τμήμα Μέρα_Τρίτη() {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
```

```
    }
    Τμήμα ΤαξίδιΕπιστροφής {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
}
Κλάση ΠακέτοΑ ως Ταξίδι {
Ιδιωτικό:
    Τμήμα ΤαξίδιΠροςΠροορισμό {
        Τύπωσε "Οι τουρίστες έρχονται αεροπορικώς ..."
    }
    Τμήμα Μέρα_Πρώτη {
        Τύπωσε "Οι τουρίστες πάνε στο ενυδρείο ..."
    }
    Τμήμα Μέρα_Δεύτερη {
        Τύπωσε "Οι τουρίστες πάνε στη παραλία ..."
    }
    Τμήμα Μέρα_Τρίτη {
        Τύπωσε "Οι τουρίστες πάνε στα βουνά ..."
    }
    Τμήμα ΤαξίδιΕπιστροφής {
        Τύπωσε "Οι τουρίστες επιστρέφουν αεροπορικώς ..."
    }
}
Κλάση ΠακέτοΒ ως Ταξίδι {
Ιδιωτικό:
    Τμήμα ΤαξίδιΠροςΠροορισμό {
        Τύπωσε "Οι τουρίστες έρχονται με τρένο ..."
    }
    Τμήμα Μέρα_Πρώτη {
        Τύπωσε "Οι τουρίστες επισκέπτονται το βουνό ..."
    }
    Τμήμα Μέρα_Δεύτερη {
        Τύπωσε "Οι τουρίστες πάνε στην παραλία ..."
    }
    Τμήμα Μέρα_Τρίτη {
        Τύπωσε "Οι τουρίστες πάνε στο ζωολογικό κήπο ..."
    }
    Τμήμα ΤαξίδιΕπιστροφής {
        Τύπωσε "Οι τουρίστες επιστρέφουν με τρένο ..."
    }
}
Τύπωσε "Πρώτο Ταξίδι"
Ταξίδι=ΠακέτοΒ()
Ταξίδι.Πρόγραμμα_Ταξιδιού
Τύπωσε "Επόμενο Ταξίδι"
Ταξίδι=ΠακέτοΑ()
Ταξίδι.Πρόγραμμα_Ταξιδιού
```

***Πρόγραμμα 68**: Μοτίβο Template Method*

Μοτίβο Chain of Responsibility

Στο μοτίβο Chain of Responsibility, δημιουργούμε για μια κλάση Χειριστής (handler), στιβαρές κλάσεις, όπου ορίζουμε στον καθένα την μη υλοποιημένη μέθοδο του Χειριστή: Χειρισμός_Αιτήματος. Το Αίτημα είναι ένα αντικείμενο που δημιουργούμε με μια κλάση Αίτημα, η οποία έχει δυο μεθόδους, την πάρε_Περιγραφή$ και την πάρε_Τιμή. Οι στιβαροί χειριστές είναι σφιχτά συνδεδεμένοι (tight coupling) με το Αίτημα. Η κλάση χειριστής δεν έχει ιδέα περί του αντικειμένου αιτήματος. Έτσι μπορούμε να έχουμε για διαφορετικά αιτήματα (ως κλάσεις) τους ανάλογους στιβαρούς χειριστές.

Η αλυσίδα της υπευθυνότητας παίρνει ένα αίτημα στο πρώτο χειριστή, τη κεφαλή της αλυσίδας και αν δεν ικανοποιηθεί σε αυτόν τότε προωθείται στον επόμενο χειριστή, κ.ο.κ.

Στην ελληνική έκδοση του προγράμματος οι Χειρ1, Χειρ2, Χειρ3 είναι δείκτες σε αντικείμενα. Έχουμε δώσει στη Κλάση Χειριστής το μέλος Διαγραφή για να το πάρουν όλοι οι στιβαροί χειριστές. Επίσης καθαρίζουμε τη μνήμη του τρέχοντος τμήματος με την Καθαρό και τυπώνουμε ένα μήνυμα μετά. Στην εκτέλεση βλέπουμε τρία μηνύματα Διαγράφτηκα! Πριν το τελευταίο μήνυμα.

Τα αιτήματα δεν τα περνάμε με αναφορά, είναι σαν τιμές, περνάνε με αντιγραφή.

```
Κλάση Αίτημα {
Ιδιωτικό:
    μια_τιμή, μια_περιγραφή$
Δημόσιο:
    Συνάρτηση πάρε_Περιγραφή$ {
        =.μια_περιγραφή$
    }
    Συνάρτηση πάρε_Τιμή {
        =.μια_τιμή
    }
Κλάση:
    Τμήμα Αίτημα (.μια_περιγραφή$, .μια_τιμή) {
    }
}
Κλάση Χειριστής {
Ιδιωτικό:
    διάδοχος_χειριστής=Δείκτης()
Δημόσιο:
    Τμήμα θέσεΔιάδοχοΧειριστή (Χειριστής ως *Χειριστής) {
        .διάδοχος_χειριστής<=Χειριστής
    }
    Τμήμα Χειρισμός_Αιτήματος {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
    Διαγραφή {
         Τύπωσε "Διαγράφτηκα!"
    }
}
```

```
Κλάση  Στιβαρός_Χειριστής_Ένα ως Χειριστής {
    Τμήμα Χειρισμός_Αιτήματος (Αίτημα ως Αίτημα) {
        Αν Αίτημα.πάρε_Τιμή() < 0 Τότε
            Αναφορά "Τις αρνητικές τιμές χειρίζεται ο Στιβαρός_Χειριστής_Ένα: " +
Αίτημα.πάρε_Περιγραφή$()
            Αναφορά Μορφή$("Τιμή: {0}", Αίτημα.πάρε_Τιμή())
        Αλλιώς
            .διάδοχος_χειριστής=>Χειρισμός_Αιτήματος Αίτημα
        Τέλος Αν
    }
}
Κλάση Στιβαρός_Χειριστής_Δύο ως Χειριστής {
    Τμήμα Χειρισμός_Αιτήματος (Αίτημα ως Αίτημα) {
        Αν Αίτημα.πάρε_Τιμή() = 0 Τότε
            Αναφορά "Τις μηδενικές τιμές χειρίζεται ο Στιβαρός_Χειριστής_Δύο: " +
Αίτημα.πάρε_Περιγραφή$()
            Αναφορά Μορφή$("Τιμή: {0}", Αίτημα.πάρε_Τιμή())
        Αλλιώς
            .διάδοχος_χειριστής=>Χειρισμός_Αιτήματος Αίτημα
        Τέλος Αν
    }
}
Κλάση Στιβαρός_Χειριστής_Τρία ως Χειριστής {
    Τμήμα Χειρισμός_Αιτήματος (Αίτημα ως Αίτημα) {
        Αν Αίτημα.πάρε_Τιμή() > 0 Τότε
            Αναφορά "Τις θετικές τιμές χειρίζεται ο Στιβαρός_Χειριστής_Τρία: " +
Αίτημα.πάρε_Περιγραφή$()
            Αναφορά Μορφή$("Τιμή: {0}", Αίτημα.πάρε_Τιμή())
        Τέλος Αν
    }
}
\\ Δημιουργούμε τους χειριστές
Χειρ1->Στιβαρός_Χειριστής_Ένα()
Χειρ2->Στιβαρός_Χειριστής_Δύο()
Χειρ3->Στιβαρός_Χειριστής_Τρία()


\\ Δημιουργούμε την αλυσίδα
\\ οι δείκτες είναι απλές αναφορές σε επώνυμες ομάδες.
Χειρ1=>θέσεΔιάδοχοΧειριστή Χειρ2
Χειρ2=>θέσεΔιάδοχοΧειριστή Χειρ3


\\ Στέλνουμε αιτήματα στην αλυσίδα (από τον Χειρ1)
```

```
Χειρ1=>Χειρισμός_Αιτήματος Αίτημα("Αρνητική Τιμή ", -1)
Χειρ1=>Χειρισμός_Αιτήματος Αίτημα("Μηδενική Τιμή ", 0)
Χειρ1=>Χειρισμός_Αιτήματος Αίτημα("Θετική Τιμή ", 1)
Χειρ1=>Χειρισμός_Αιτήματος Αίτημα("Θετική Τιμή ", 2)
Χειρ1=>Χειρισμός_Αιτήματος Αίτημα("Αρνητική Τιμή ", -5)


\\ σβήνουμε όλες τις μεταβλητές του τρέχοντος τμήματος
Καθαρό
Τύπωσε "όλα τα αντικείμενα διαγράφτηκαν"
```

***Πρόγραμμα 69**: Μοτίβο Chain of Responsibility*

Μοτίβο Adapter

Το μοτίβο Adapter, ή Προσαρμογέας, χρησιμοποιείται όταν έχουμε μια κλάση, Προσαρμόσιμη, που θα χρησιμοποιηθεί με διαφορετική διεπαφή. Στο παράδειγμα η Προσαρμόσιμη έχει διεπαφή τις μεθόδους μέθοδοςΒ και συνΒ. Θέλουμε να χρησιμοποιήσουμε την Διεπαφή_Κάτι, η οποία ορίζει τα μέθοδοςΑ και συνΑ. Φτιάχνουμε ένα προσαρμογέα (adapter), ο οποίος έχει έναν εσωτερικό δείκτη για αντικείμενο τύπου Προσαρμόσιμη, επειδή έτσι ορίζουμε στον κατασκευαστή της να είναι το όρισμα Προσαρμόσιμη. Ο προσαρμογέας υλοποιεί την Διεπαφή_Κάτι, και έχει με σφιχτή σύνδεση (tight coupling) την γνώση για την Προσαρμόσιμη.

Στο παράδειγμα σε έναν Πελάτη, ένα αντικείμενο κλάσης Πελάτη, περιμένει ένα αντικείμενο τύπου Διεπαφή_Κάτι. Αντί να δώσουμε ένα τέτοιο αντικείμενο, δίνουμε έναν προσαρμογέα, που είναι τύπου Διεπαφή_Κάτι αλλά έχει δείκτη ιδιωτικό προς μια Προσαρμόσιμη και υλοποιεί τις μεθόδους με κλήσεις προς τις μεθόδους της Προσαρμόσιμης.

Τεχνικά στη Μ2000 η κλήση τμήματος από τμήμα μεταφέρει τον τρέχον σωρό τιμών, οπότε δεν χρειάζεται να πάρουμε τα ορίσματα σε τυπικές παραμέτρους και να τα δώσουμε στην νέα κλήση. Η κλήση συνάρτησης σε έκφραση (όχι ως τμήμα με την Κάλεσε) δημιουργεί νέο σωρό τιμών. Και εδώ δεν θα φτιάξουμε τυπικές παραμέτρους για να τις δώσουμε ως ορίσματα στην κλήση της συνΒ στην Προσαρμόσιμη. Το [] είναι αναγνωριστικό μόνο για ανάγνωση και παίρνει τον τρέχοντα σωρό τιμών ως δείκτη και αφήνει για τρέχοντα ένα νέο κενό σωρό τιμών. Το ![] βάζει τα στοιχεία του σωρού από το δείκτη που μας έδωσε το [] στον σωρό της κλήσης της ΣυνΒ, έτσι μεταφέρουμε το σωρό ως έχει.

```
Κλάση Προσαρμόσιμη {
    Τμήμα μέθοδοςΒ (Χ) {
        Τύπωσε "Χ=";Χ
    }
    Συνάρτηση συνΒ (Χ, Υ) {
        =Χ**Υ
    }
}
Κλάση Διεπαφή_Κάτι {
    Τμήμα μέθοδοςΑ {
```

```
            Λάθος "Δεν έχει υλοποιηθεί ακόμα"
        }
        Συνάρτηση συνΑ {
            Λάθος "Δεν έχει υλοποιηθεί ακόμα"
        }
}


Κλάση Προσαρμογέας ως Διεπαφή_Κάτι {
Ιδιωτικό:
    εσωτερική=Δείκτης()
Δημόσιο:
    Τμήμα μέθοδοςΑ {
        \\ στις κλήσεις τμημάτων ο σωρός περνάει ως έχει
        .εσωτερική=>μέθοδοςΒ
    }
    Συνάρτηση συνΑ {
        \\ Περνάμε τα ορίσματα της συνΑ στην συνΒ με το ![]
        \\ γιατί οι συναρτήσεις ξεκινούν με δικό τους σωρό
        =.εσωτερική=>συνΒ(![])
    }
Κλάση:
    Τμήμα Προσαρμογέας (Προσαρμόσιμη ως *Προσαρμόσιμη){
        .εσωτερική<=Προσαρμόσιμη
    }
}
Κλάση Πελάτης {
    Τμήμα Κάνε_Κάτι (Προσαρμογέας ως Διεπαφή_Κάτι) {
        Προσαρμογέας.μέθοδοςΑ 100
        Τύπωσε Προσαρμογέας.συνΑ(2, 3)=8
    }
}
Πελάτης=Πελάτης()
Προσαρμογέας1=Προσαρμογέας(Δείκτης(Προσαρμόσιμη()))
Πελάτης.Κάνε_Κάτι Προσαρμογέας1
```

***Πρόγραμμα 70**: Μοτίβο Adapter*

Μοτίβο Bridge

Το μοτίβο Bridge, ή Γέφυρα, κάνει κάτι εξαιρετικό, διαχωρίζει την αφηρημένη περιγραφή από την υλοποίηση, με τρόπο ώστε η υλοποίηση και η αφηρημένη κλάση να μπορούν να επεκταθούν και μαζί και χώρια! Πραγματικά αν και το παράδειγμα που ακολουθεί είναι μικρό, είναι δύσκολο να κατανοηθεί, έχει οκτώ κλάσεις, αν δεν πάρουμε τα πράγματα με μια σειρά, η καλύτερα να ξεκινήσουμε με το τι ορίζουμε στο πρόγραμμα:

Κλάση **Διεπαφή_Αφηρημένη**:
Αφηρημένη ορίζει μία δημόσια μέθοδο: Λειτουργία
Κλάση **Αφαίρεση1** ως Διεπαφή_Αφηρημένη:
μέθοδος Λειτουργία, ο Κατασκευαστής παίρνει δείκτη τύπου Διεπαφή_κάθε_υλοποιητή
Κλάση **Αφαίρεση2** ως Διεπαφή_Αφηρημένη:
μέθοδος Λειτουργία, μέθοδος Λειτουργία1 (δείχνει ότι η αφαίρεση2 μπορεί να επεκταθεί)
ο Κατασκευαστής παίρνει δείκτη τύπου Διεπαφή_κάθε_υλοποιητή
Κλάση **Αφαίρεση3** ως Αφαίρεση2:
μέθοδος Λειτουργία, μέθοδος Λειτουργία1 (αλλαγή της μεθόδου από αυτήν στην Αφαίρεση2)
ο Κατασκευαστής παίρνει δείκτη τύπου Διεπαφή_κάθε_υλοποιητή
Κλάση **Διεπαφή_κάθε_υλοποιητή**:
Αφηρημένη ορίζει μία δημόσια μέθοδο: λειτουργίαΥλοποίησης
Κλάση **Διεπαφή_υλοποιητή1** ως Διεπαφή_κάθε_υλοποιητή:
μέθοδος λειτουργίαΥλοποίησης
Κλάση **Διεπαφή_υλοποιητή2** ως Διεπαφή_κάθε_υλοποιητή:
μέθοδος λειτουργίαΥλοποίησης
Κλάση **Διεπαφή_υλοποιητή3** ως Διεπαφή_υλοποιητή2:
μέθοδος λειτουργίαΥλοποίησης, λειτουργίαΥλοποίησης2 (εδώ κάνουμε επέκταση της κλάσης Διεπαφή_υλοποιητή2

Σε κάθε περίπτωση θέλουμε να καλούμε τη μέθοδο **Λειτουργία** από ένα αντικείμενο του τύπου **Διεπαφή_Αφηρημένη**, και κάποια στιγμή θέλουμε να επεκτείνουμε την διεπαφή με ένα απόγονο από τη κλάση Διεπαφή_Αφηρημένη, εδώ στο πρόγραμμα την Αφαίρεση3.

Η Διεπαφή_Αφηρημένη, απλά λέει ότι θα πρέπει να υπάρχει μια μέθοδος Λειτουργία. Ας δούμε τι γίνεται με μια στιβαρή κλάση, την Αφηρημένη1 που κληρονομεί από την Διεπαφή_Αφηρημένη: Αυτή η κλάση δημιουργεί ένα αντικείμενο με ένα όρισμα του τύπου Διεπαφή_κάθε_υλοποιητή το οποίο είναι δείκτης και τον καταχωρεί στην ιδιωτική: **υλοποίηση**. Υλοποιεί την Λειτουργία ως όφειλε με την κλήση της λειτουργίαςΥλοποίησης στο δείκτη υλοποίηση. Καλούμε δηλαδή τη Λειτουργία και το αντικείμενο καλεί τη λειτουργίαΥλοποίησης. Γεφυρώνουμε ένα αντικείμενο τύπου Διεπαφή_Αφηρημένη με ένα αντικείμενο τύπου Διεπαφή_Υλοποιητή. Αυτό το έχουμε και σε άλλα μοτίβα, όπως αυτό του Adapter. Ποια είναι λοιπόν η διαφορά;

Συνεχίζουμε και φτιάχνουμε την **Αφαίρεση2** με μια πρόσθετη Λειτουργία1. Αυτή είναι μια επέκταση. **Αυτό σημαίνει ξεχωριστή επέκταση Αφαίρεσης**

Μπορούμε παράλληλα να κάνουμε επέκταση και στη Διεπαφή_Υλοποιητή, εκεί που είχαμε μόνο τη **Διεπαφή_υλοποιητή1** τώρα θα έχουμε την **Διεπαφή_υλοποιητή2** η οποία απλά αλλάζει την λειτουργίαΥλοποίησης. **Αυτό σημαίνει ότι έχουμε ξεχωριστή επέκταση Υλοποίησης**

Δείχνουμε ότι η Αφαίρεση1 λειτουργεί με οποιαδήποτε Διεπαφή_υλοποιητή. Επίσης δείχνουμε ότιο στην Αφαίρεση2 η Λειτουργία1 δεν έχει σύνδεση με το αντικείμενο που δείχνει ο δείκτης υλοποίηση. Αν καλέσουμε την Λειτουργία1 θα μας εμφανίσει το μήνυμα ***Επέκταση Αφαίρεσης***.

Στο τρίτο μέρος κάνουμε κάνουμε επέκταση και της Αφηρημένης2 στην Αφηρημένη3 όπου η Λειτουρία1 καλεί την λειτουργίαΥλοποίησης2. Το αντικείμενο που δέχεται ο κατασκευαστής της

Αφηρημένης3 είναι τύπου **Διεπαφή_υλοποιητή3** ο οποίος προκύπτει από τη Διεπαφή_υλοποιητή2 που προκύπτει από την Διεπαφή_κάθε_υλοποιητή. Έτσι η Αφηρημένη3 δεν μπορεί να δουλέψει με κάτι που δεν είναι τύπος Διεπαφή_υλοποιητή3. **Εδώ έχουμε επέκταση και της Αφαίρεσης και της Υλοποίησης**, με σφιχτή σύνδεση (tight coupling), αφού η Αφαίρεση2 εξαρτιέται από την Διεπαφή_υλοποιητή3.

```
Κλάση Διεπαφή_Αφηρημένη {
    Τμήμα Λειτουργία {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
}
Κλάση Αφαίρεση1 ως Διεπαφή_Αφηρημένη {
ιδιωτικό:
    υλοποίηση=Δείκτης()
Δημόσιο:
    Τμήμα Λειτουργία {
        .υλοποίηση=>λειτουργίαΥλοποίησης
    }
Κλάση:
    Τμήμα Αφαίρεση1 (μια_υλοποίηση ως *Διεπαφή_κάθε_υλοποιητή) {
        .υλοποίηση<=μια_υλοποίηση
    }
}
Κλάση Αφαίρεση2 ως Διεπαφή_Αφηρημένη {
ιδιωτικό:
    υλοποίηση=Δείκτης()
Δημόσιο:
    Τμήμα Λειτουργία {
        .υλοποίηση=>λειτουργίαΥλοποίησης
    }
    Τμήμα Λειτουργία1 {
        Τύπωσε "Επέκταση Αφαίρεσης"
    }
Κλάση:
    Τμήμα Αφαίρεση2 (μια_υλοποίηση ως *Διεπαφή_κάθε_υλοποιητή) {
        .υλοποίηση<=μια_υλοποίηση
    }
}

Κλάση Διεπαφή_κάθε_υλοποιητή {
    Τμήμα λειτουργίαΥλοποίησης {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
}
Κλάση Διεπαφή_υλοποιητή1 ως Διεπαφή_κάθε_υλοποιητή {
    Τμήμα λειτουργίαΥλοποίησης {
        Τύπωσε "Αποτέλεσμα από Διεπαφή_υλοποιητή1"
```

```
    }
}
Κλάση Διεπαφή_υλοποιητή2 ως Διεπαφή_κάθε_υλοποιητή {
    Τμήμα λειτουργίαΥλοποίησης {
        Τύπωσε "Αποτέλεσμα από Διεπαφή_υλοποιητή2"
    }
}
Αντικείμενο=Αφαίρεση1(Δείκτης(Διεπαφή_υλοποιητή1()))
Αντικείμενο.Λειτουργία
Αντικείμενο=Αφαίρεση1(Δείκτης(Διεπαφή_υλοποιητή2()))
Αντικείμενο.Λειτουργία
Αντικείμενο=Αφαίρεση2(Δείκτης(Διεπαφή_υλοποιητή1()))
Αντικείμενο.Λειτουργία
Αντικείμενο.Λειτουργία1
\\ επέκταση και στην Αφαίρεση και στην Διεπαφή_υλοποιητή
Κλάση Αφαίρεση3 ως Αφαίρεση2 {
ιδιωτικό:
    υλοποίηση=Δείκτης()
Δημόσιο:
    Τμήμα Λειτουργία {
        .υλοποίηση=>λειτουργίαΥλοποίησης
    }
    Τμήμα λειτουργία1 {
        .υλοποίηση=>λειτουργίαΥλοποίησης2
    }
Κλάση:
    Τμήμα Αφαίρεση3 (μια_υλοποίηση ως *Διεπαφή_υλοποιητή3) {
        .υλοποίηση<=μια_υλοποίηση
    }
}
Κλάση Διεπαφή_υλοποιητή3 ως Διεπαφή_υλοποιητή2 {
    Τμήμα λειτουργίαΥλοποίησης2 {
        Τύπωσε "Αποτέλεσμα 2 από Διεπαφή_υλοποιητή3"
    }
}
Αντικείμενο=Αφαίρεση3(Δείκτης(Διεπαφή_υλοποιητή3()))
Αντικείμενο.Λειτουργία
Αντικείμενο.Λειτουργία1
Τύπωσε Αντικείμενο είναι τύπος Διεπαφή_Αφηρημένη
Τύπωσε Αντικείμενο είναι τύπος Αφαίρεση3
```

***Πρόγραμμα 71**: Μοτίβο Bridge*

Μοτίβο Composite

Το μοτίβο Composite, ή Σύνθεση, δουλεύει για δομές δένδρων. Έχουμε δει δομή δένδρου στο μοτίβο Observer. Εδώ έχουμε κάτι πιο απλό, όπως περιγράφονται οι κλάσεις στο πίνακα:

Κλάση **Διεπαφή_Εξαρτήματος**:

Αφηρημένη ορίζει μία δημόσια μέθοδο: Λειτουργία

Κλάση **Εξάρτημα** ως Διεπαφή_Εξαρτήματος:

μέθοδος Λειτουργία

Κλάση **Φύλλο** ως Εξάρτημα:

νέα μέθοδος Λειτουργία

Κλάση **Σύνθετο** ως Εξάρτημα:

νέα μέθοδος Λειτουργία, μέθοδος βάλε_παιδιά

Η δημιουργία δένδρου γίνεται με παρόμοιο τρόπο με το δένδρο στο μοτίβο Observer. Δημιουργούμε ένα σύνθετο, βάζουμε παιδιά με δείκτες από Εξάρτημα. Επειδή το σύνθετο είναι εξάρτημα μπορούμε να φτιάξουμε άλλο σύνθετο και να βάλουμε το προηγούμενο στο νέο.

Αφού τελειώσει η δημιουργία του δένδρου, κρατάμε μόνο ένα δείκτη, αυτό της ρίζας του δένδρου, και καλούμε τη Λειτουργία η οποία θα καλέσει όλα τα φύλλα διατρέχοντας όλα τα κλαδιά. Εδώ δεν έχουμε έναν Επισκέπτη να μπει σε όλα τα φύλλα, αλλά το ίδιο το σύνθετο καλεί ότι έχει!

Δεν έχει υλοποιηθεί η διαγραφή παιδιού βάσει ονόματος. Το πώς θα υλοποιείται το δένδρο είναι θέμα εσωτερικό της κλάσης Σύνθετο.

```
Κλάση Διεπαφή_Εξαρτήματος {
    Τμήμα λειτουργία {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
}
Κλάση Εξάρτημα ως Διεπαφή_Εξαρτήματος {
ιδιωτικό:
    όνομα$
Δημόσιο:
    Τμήμα λειτουργία {
        Τύπωσε "κάνε κάτι"
    }
    Διαγραφή {
        Τύπωσε "Διαγραφή ";.όνομα$
    }
}
Κλάση Φύλλο ως Εξάρτημα {
    Τμήμα λειτουργία {
        Τύπωσε "Κάνε μια λειτουργία για το Φύλλο:";.όνομα$
```

```
    }
Κλάση:
    Τμήμα Φύλλο (.όνομα$) {
    }
}
Κλάση Σύνθετο ως Εξάρτημα {
Ιδιωτικό:
    εξαρτήματα=Σωρός
Δημόσιο:
    Τμήμα λειτουργία {
     ένα_εξάρτημα=Κάθε(.εξαρτήματα)
     Τύπωσε "["+.όνομα$+"]"
     Ενώ ένα_εξάρτημα
          εξάρτημα=ΤιμήΣωρού(ένα_εξάρτημα)
          εξάρτημα=>λειτουργία
     Τέλος Ενώ
    }
    Τμήμα βάλε_παιδιά (Εξάρτημα ως *Εξάρτημα) {
        Σωρός .εξαρτήματα {Σειρά Εξάρτημα}
    }
Κλάση:
    Τμήμα Σύνθετο (.όνομα$) {
    }
}
εξαρτήματα->Σύνθετο("GroupA")
εξαρτήματα=>βάλε_παιδιά Δείκτης(Φύλλο("Shape1"))
εξαρτήματα=>βάλε_παιδιά Δείκτης(Φύλλο("Shape2"))
εξαρτήματα=>βάλε_παιδιά Δείκτης(Φύλλο("Shape3"))
εξαρτήματα1->Σύνθετο("Graphic")
εξαρτήματα1=>βάλε_παιδιά εξαρτήματα
εξαρτήματα->Σύνθετο("GroupB")
εξαρτήματα=>βάλε_παιδιά Δείκτης(Φύλλο("Shape4"))
εξαρτήματα1=>βάλε_παιδιά εξαρτήματα
εξαρτήματα->0& ' βάζουμε δείκτη με τύπο Μηδενικός (Null)
\\ εκτελούμε την λειτουργία για όλα τα φύλλα του δένδρου
εξαρτήματα1=>λειτουργία
```

***Πρόγραμμα 72**: Μοτίβο Composite*

Μοτίβο Decorator

Το μοτίβο Decorator, ή Διακοσμητής, στόχο έχει κατά τη φάση εκτέλεσης να μπορεί να πάρει ένα αντικείμενο μιας Οτιδήποτε κλάσης και να το επεκτείνει, να του δώσει νέο τύπο ή νέα λειτουργικότητα.

Ακολουθούν δυο παραδείγματα. Το πρώτο παράδειγμα κάνει χρήση της δυνατότητας της Μ2000 να συνδέει αντικείμενα σε ένα άλλο, και μάλιστα να φτιάχνει αντικείμενα που δεν έχουν τύπο, ώστε το τελικό αντικείμενο να μην αλλάζει τύπο ενώ έχει πάρει νέα μέλη. Κάνουμε έλεγχο ότι όντως δουλεύει η δυνατότητα μια μεθόδου ενός τύπου Οτιδήποτε να βλέπει ιδιωτικές μεταβλητές αντικειμένου που του έχει δοθεί ως τυπική παράμετρος του ίδιου τύπου. Εδώ έχουμε την εντολή

Ένα=Οτιδήποτε() με μια_Διακόσμηση η οποία συνδέει δυο αντικείμενα και δίνει το Ένα. Χρησιμοποιούμε την συνάρτηση Τμήμα() που επιστρέφει αληθής αν το τμήμα που γράφουμε σαν όρισμα υπάρχει (μπορούμε να δώσουμε και συνάρτηση). Η συνάρτηση αυτή δεν εκτελεί κάτι απλά ελέγχει αν το όνομα είναι μια μέθοδος.

```
Κλάση Οτιδήποτε {
Ιδιωτικό:
    η_τιμή_μου=100
Δημόσιο:
    Τμήμα Κάνε_Κάτι {
        Τύπωσε "κάτι"
    }
}
\\ ορισμός αντικειμένου, άμεσα, και χωρίς τύπο.
Ομάδα μια_Διακόσμηση {
    Τμήμα Κάνε_Κάτι {
        Τύπωσε "κάτι άλλαξε"
    }
    Τμήμα Κάνε_Κάτι_Αλλιώς (εδώ ως Οτιδήποτε){
        \\ η εδώ.η_τιμή_μου γίνεται να διαβαστεί ενώ είναι ιδιωτική
        \\ αν το τμήμα υπάρχει σε αντικείμενο με ίδιο τύπο: Οτιδήποτε
        Τύπωσε "Κάνε κάτι αλλιώς: ", .η_τιμή_μου+εδώ.η_τιμή_μου
    }
}
Ένα=Οτιδήποτε() με μια_Διακόσμηση
Ένα.Κάνε_Κάτι_Αλλιώς Οτιδήποτε()
Δύο=Οτιδήποτε()
Δύο.Κάνε_Κάτι
Τύπωσε Τμήμα(Δύο.Κάνε_Κάτι_Αλλιώς)=Ψευδής
\\ Προσθέτουμε αργότερα
Δύο=μια_Διακόσμηση
Δύο.Κάνε_Κάτι
Τύπωσε Τμήμα(Δύο.Κάνε_Κάτι_Αλλιώς)=Αληθής
```

***Πρόγραμμα 73**: Μοτίβο Decorator*

Στο δεύτερο παράδειγμα έχουμε μια κλάση Σχήμα_Κύκλου που φτιάχνει αντικείμενα τα οποία φτιάχνονται με ένα όρισμα ακτίνα, και έχουν μια μέθοδο Σχεδίασε που λαμβάνει το σημείο που θα τοποθετηθεί το σχήμα (το κέντρο του κύκλου). Η λειτουργία γίνεται με βασικό χρώμα (δεν το καθορίζουμε εδώ, αλλά υποθέτουμε ότι υπάρχει ένα βασικό χρώμα).

Έχουμε μια δεύτερη κλάση την Σχήμα_Κύκλου_με_Χρώμα, η οποία δέχεται όρισμα ένα χρώμα περιμέτρου. Παράλληλα έχει μια Σχεδίαση με αλλαγή για να χρησιμοποιεί το χρώμα περιμέτρου.

Αντί να έχουμε μια κλάση που να κληρονομεί από μια άλλη, τώρα συνδέουμε δυο αντικείμενα που ήδη έχουν κατασκευαστεί σε ένα αντικείμενο όπως φαίνεται στην εντολή παρακάτω, όπου το Κύκλος1 έχει ήδη κατασκευαστεί, **και ένα αντίγραφό του** συνδέεται με το αντικείμενο που δίνει η κλάση Σχήμα_Κύκλου_με_Χρώμα() και έτσι προκύπτει το Κύκλος2.

Κύκλος2=Κύκλος1 με Σχήμα_Κύκλου_με_Χρώμα(15)

Στην επόμενη εντολή έχουμε σύνδεση δυο αντικειμένων όπως μας τα δίνουν δυο κλάσεις. Σε κάθε μια εκτελούμε τον δικό της κατασκευαστή.

Κύκλος3=Σχήμα_Κύκλου(100) με Σχήμα_Κύκλου_με_Χρώμα(10)

```
Κλάση Σχήμα_Κύκλου {
Ιδιωτικό:
    μηνυμα$={Σχεδιάζω Κύκλο,
        με κέντρο στο σημείο {0},{1} με ακτίνα {2} με βασικό χρώμα
    }
    ακτίνα
Δημόσιο:
    Τμήμα Σχεδίασε (Χ, Υ) {
        Αναφορά Μορφή$(.μηνυμα$, Χ, Υ, .ακτίνα)
    }
Κλάση:
    Τμήμα Σχήμα_Κύκλου (.ακτίνα) {
    }
}
Κλάση Σχήμα_Κύκλου_με_Χρώμα {
Ιδιωτικό:
    Τελική μηνυμα$={Σχεδιάζω Κύκλο,
         με κέντρο στο σημείο {0},{1} με ακτίνα {2} με επιλογή χρώματος {3}
     }
     Χρώμα_Περιμέτρου=0
Δημόσιο:
    Τμήμα Σχεδίασε (Χ, Υ) {
        Αναφορά Μορφή$(.μηνυμα$, Χ, Υ, .ακτίνα,.Χρώμα_Περιμέτρου)
    }
Κλάση:
    Τμήμα Σχήμα_Κύκλου_με_Χρώμα (.Χρώμα_Περιμέτρου) {
    }
}
Κύκλος1=Σχήμα_Κύκλου(500)
Κύκλος1.Σχεδίασε 500, 400
\\ Κληρονομικότητα σε επίπεδο αντικειμένων.
Κύκλος2=Κύκλος1 με Σχήμα_Κύκλου_με_Χρώμα(15)
Κύκλος2.Σχεδίασε 120, 300
```

```
\\ Κληρονομικότητα σε επίπεδο αντικειμένων
\\ Εδώ κάνουμε συγχώνευση στο υπάρχον Κύκλος1
Κύκλος1=Σχήμα_Κύκλου_με_Χρώμα(13)
Κύκλος1.Σχεδίασε 500, 400
Κύκλος3=Σχήμα_Κύκλου(100) με Σχήμα_Κύκλου_με_Χρώμα(10)
Κύκλος3.Σχεδίασε 200, 200
```

Πρόγραμμα 74: *Μοτίβο Decorator - Σχήμα Κύκλος*

Μοτίβο Flyweight

Το μοτίβο Flyweight, ή Ελάφρυνση, δημιουργεί ελαφρότερα αντικείμενα όταν θέλουμε πολλά αντικείμενα.

Στο παράδειγμα που ακολουθεί, μας ενδιαφέρει να κινούμε στρατιώτες, οι οποίοι σχεδιάζονται με μεθόδους από ένα αντικείμενο ΓραφικήΑναπαράστασηΣτρατιώτη. Δεν χρειάζεται κάθε στρατιώτης να έχει ένα δικό του αντικείμενο γραφικής αναπαράστασης, αλλά μπορεί να κρατάει ένα δείκτη αναπαράσταση στο αντικείμενο ΓραφικήΑναπαράστασηΣτρατιώτη. Ας δούμε ποιες κλάσεις έχει το παράδειγμα:

Κλάση **Στρατιώτης**:
Αφηρημένη ορίζει μία δημόσια μέθοδο: κίνησηΣτρατιώτη
Κλάση **ΥλοποίησηΣτρατιώτη** ως Στρατιώτης:
μέθοδος κίνησηΣτρατιώτη
Κλάση **Νέος_Στρατιώτης** ως Στρατιώτης :
μέθοδος κίνησηΣτρατιώτη

Υπάρχουν δυο αντικείμενα, γενικά, δηλαδή μπορούν να κληθούν από παντού, όσο το τρέχον τμήμα εκτελείται. Το ένα είναι η **ΓραφικήΑναπαράστασηΣτρατιώτη** από την οποία θέλουμε κάθε στρατιώτης να έχει τον δείκτη του. Χωρίς το δείκτη δεν θα μπορούμε να καλούμε τις μεθόδους του. Το άλλο είναι ο **Παραγωγός_Στρατιώτη**. Αυτό είναι μια μέθοδος Factory, η οποία όμως παράγει έναν στρατιώτη και κάθε φορά που ζητάμε έναν μας δίνει τον ίδιο δείκτη. Στην ουσία μας δίνει ότι χρειαζόμαστε για να σχεδιάσουμε τον στρατιώτη μαζί με πρόσθετη λογική που μας δίνει η ΥλοποίησηΣτρατιώτη για την κίνησηΣτρατιώτη.

Η κλάση Νέος_Στρατιώτης, είναι ένας στρατιώτης και βλέπει το αντικείμενο Παραγωγός_Στρατιώτη, αφού είναι γενικό, και κρατάει το δείκτη του στρατιώτη που του δίνει το αντικείμενο. Παράλληλα έχει περισσότερη λογική για να κρατάει τη θέση του στρατιώτη. Έτσι η ΚίνησηΣτρατιώτη στο Νέος_Στρατιώτης καλεί την ΚίνησηΣτρατιώτη στον δείκτη στο Στρατιώτη ενώ παράλληλα κρατάει τις τελευταίες συντεταγμένες.

Γεμίζουμε ένα πίνακα ΣτρατιώτηςΠεδίου() με ΝέοΣτρατιώτη. Μετά καλούμε για κάθε στρατιώτη του πίνακα να μετακινηθεί σε ένα σημείο, που δίνουμε από το σωρό τιμών. Αρχικά η σχεδίαση θα γίνει μόνο για σχεδίαση και όχι για διαγραφή. Σε δεύτερη φάση δίνουμε άλλα στοιχεία στο σωρό τιμών και με μια ακολουθία επανάληψης πάλι εκτελούμε την κίνησηΣτρατιώτη, και εδώ έχουμε

πρώτα διαγραφή από το τελευταίο σημείο σχεδίασης και σχεδιασμός στο νέο ή αν το Υ < -1000 τότε δεν θα σχεδιαστεί ξανά.

```
Κλάση Στρατιώτης {
    Τμήμα κίνησηΣτρατιώτη {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
}
Γενική Ομάδα ΓραφικήΑναπαράστασηΣτρατιώτη {
    Τμήμα Σχεδίασε_Στρατιώτη (Χ, Υ){
        Τύπωσε "Στρατιώτης στο σημείο ", Χ, Υ
    }
    Τμήμα Διαγραφή_Στρατιώτη (Χ, Υ) {
        Τύπωσε "Διαγραφή Στρατιώτη από σημείο", Χ, Υ
    }
}
Κλάση ΥλοποίησηΣτρατιώτη ως Στρατιώτης {
Ιδιωτικό:
    Αναπαράσταση=Δείκτης(ΓραφικήΑναπαράστασηΣτρατιώτη)
Δημόσιο:
    Τμήμα κίνησηΣτρατιώτη(απόΧ, απόΥ, στοΧ, στοΥ) {
        Αν απόΥ>-1000 Τότε .Αναπαράσταση=>Διαγραφή_Στρατιώτη απόΧ, απόΥ
        Αν στοΥ>-1000 Τότε .Αναπαράσταση=>Σχεδίασε_Στρατιώτη στοΧ, στοΥ
    }
}
Γενική Ομάδα Παραγωγός_Στρατιώτη {
Ιδιωτικό:
    Στρατιώτης=Δείκτης()
Δημόσιο:
    Συνάρτηση Πάρε_Στρατιώτη {
        Αν .Στρατιώτης Είναι Τύπος Μηδενικός Τότε
            .Στρατιώτης<=Δείκτης(ΥλοποίησηΣτρατιώτη())
        Τέλος Αν
        =.Στρατιώτης
    }
}
Κλάση Νέος_Στρατιώτης ως Στρατιώτης {
Ιδιωτικό:
    Στρατιώτης=Παραγωγός_Στρατιώτη.Πάρε_Στρατιώτη()
    τωρινόΧ=-1000, τωρινόΥ=-1000
Δημόσιο:
    Τμήμα κίνησηΣτρατιώτη(στοΧ, στοΥ) {
        .Στρατιώτης=>κίνησηΣτρατιώτη .τωρινόΧ, .τωρινόΥ, στοΧ, στοΥ
        (.τωρινόΧ, .τωρινόΥ)=(στοΧ, στοΥ)
```

```
    }
}
Αδειασε ' αδειάζουμε τον τρέχον σωρό
\\ ο τελεστής << κάνει την Νέος_Στρατιώτης να εκτελείται για κάθε στοιχείο του πίνακα
Πίνακας ΣτρατιώτηςΠεδίου(1 έως 5)<<Νέος_Στρατιώτης()
Σειρά 100,200, 400,500,700,300,150,200, 30,160
Για ι=1 έως 5
    Για ΣτρατιώτηςΠεδίου(ι) {
        Διάβασε Θέση_Χ, Θέση_Υ
        .κίνησηΣτρατιώτη Θέση_Χ, Θέση_Υ
    }
Επόμενο
\\ νέες θέσεις
Σειρά 110,200, 440,500,700,350,150,-1000, 30,-1000
\\ τα στοιχεία θα μπουν από τον τρέχον σωρό
Για ι=1 έως 5
    Για ΣτρατιώτηςΠεδίου(ι) {
        .κίνησηΣτρατιώτη
    }
Επόμενο
Τύπωσε "Τέλος"
```

***Πρόγραμμα 75**: Μοτίβο Flyweight*

Μοτίβο Facade

Το μοτίβο Facade, ή Μετωπίδα, χρησιμοποιείται για να ελαφραίνουμε μια διεπαφή με μια πιο μικρή διεπαφή, που μπαίνει ως μέτωπο μπροστά από τη μεγάλη διεπαφή.

Συνήθως αυτό το μοτίβο δουλεύει με μια βιβλιοθήκη. Στο παράδειγμα μια συνάρτηση παίζει το ρόλο του πακέτου που περιέχει μερικές κλάσεις. Η επιστροφή της δίνει το αντικείμενο Μετωπίδα το οποίο έχει ιδιωτικά μέλη τρία αντικείμενα από τις κλάσεις που έχει το Πακέτο1. Αντί να χρησιμοποιούμε όλη τη διεπαφή των αντικειμένων αυτών, χρησιμοποιούμε τη διεπαφή της Μετωπίδας. Η διαφορά με το Adapter είναι ότι εδώ έχουμε μια λειτουργία που επιτελεί λειτουργίες με τα αντικείμενα που είναι κρυμμένα στη Μετωπίδα.

```
Συνάρτηση Πακέτο1 {
    Κλάση Κλάση1 {
        Τμήμα λειτουργίαΑ {
            Τύπωσε "Κλάση1 λειτουργίαΑ"
        }
    }
    Κλάση Κλάση2 {
    Ιδιωτικό:
        χ=10
```

```
    Δημόσιο:
        Τμήμα ΛειτουργίαΒ {
            Τύπωσε "Κλάση2 ΛειτουργίαΒ", .χ
        }
    Κλάση:
        Τμήμα Κλάση2(.χ) {
        }
    }
    Κλάση Κλάση3 {
        Ομάδα Εσωτερική1 {
        Ιδιωτικό:
            ζ=2000
        Δημόσιο:
            χ=100
            Τμήμα ΛειτουργίαΔ {
                Τύπωσε "Κλάση3 Εσωτερική1 ΛειτουργίαΔ", .ζ, .χ
            }
        }
        Τμήμα ΛειτουργίαΓ {
            Τύπωσε "Κλάση3 ΛειτουργίαΓ", .Εσωτερική1.χ,
Έγκυρο(.Εσωτερική1.ζ)=Ψευδής
        }
    }
    Κλάση Μετωπίδα {
    Ιδιωτικό:
        Κλάση1 Α
        Κλάση2 Β(300)
        Κλάση3 Γ
    Δημόσιο:
        Τμήμα ΚάνεΚάτι {
            .Α.λειτουργίαΑ
            .Β.ΛειτουργίαΒ
            .Γ.ΛειτουργίαΓ
            .Γ.Εσωτερική1.ΛειτουργίαΔ
        }
    }
    =Μετωπίδα()
}
Μέτωπο=Πακέτο1()
Τύπωσε Μέτωπο είναι τύπος Μετωπίδα
Μέτωπο.ΚάνεΚάτι
```

***Πρόγραμμα 76**: Μοτίβο Facade*

Μοτίβο Proxy

Το μοτίβο Proxy ή Πληρεξούσιος έχει στόχο να δίνει ένα αντικείμενο που να λειτουργεί όπως ένα άλλο αντικείμενο το (και τα δύο έχουν κοινή βάση, κληρονομούν από την ίδια κλάση), αλλά με αυξημένες δυνατότητες.

Η κοινή κλάση είναι το Υποκείμενο. Ο Πληρεξούσιος είναι ένα Υποκείμενο, αλλά έχει και ένα δείκτη για να κρατήσει ένα άλλο Υποκείμενο. Το ΠραγματικόΥποκείμενο είναι και αυτό ένα Υποκείμενο.

Φτιάχνουμε ένα Μ ως ΠραγματικόΥποκείμενο με όνομα αρχείου "alfa.bmp". Η μέθοδος Μέθοδος_Α μας δείχνει το όνομα του αρχείου. Η μέθοδος Μέθοδος_Β() επιστρέφει αληθές αν έχει φορτωθεί το αρχείο. Η μέθοδος Μέθοδος_C σχεδιάζει την εικόνα.

Με χρήση του Πληρεξούσιου φτιάχνουμε το Κ που τώρα είναι Πληρεξούσιος με δείκτη στο ΠραγματικόΥποκείμενο με όνομα αρχείου "beta.bmp". Η μέθοδος του Κ μέθοδος_Α λειτουργεί το ίδιο, με αυτή του Μ, και μας δείχνει το όνομα του αρχείου. Ζητάμε απευθείας να σχεδιάσουμε την εικόνα με την μέθοδο_C τώρα ο Πληρεξούσιος βλέπει ότι δεν έχει φορτώσει την εικόνα με την μέθοδο μέθοδο_Β() την οποία δεν καλέσαμε αλλά το έκανε ο Πληρεξούσιος. Αυτή είναι η αλλαγή στη λογική που έφερε ο Πληρεξούσιος.

```
Κλάση Υποκείμενο {
Ιδιωτικό:
    όνομα_αρχείου$, φορτωμένη_εικόνα
Δημόσιο:
    Τμήμα Μέθοδος_Α {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
    Συνάρτηση Μέθοδος_Β {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
    Τμήμα Μέθοδος_C {
        Λάθος "Δεν έχει υλοποιηθεί ακόμα"
    }
}
Κλάση Πληρεξούσιος ως Υποκείμενο {
    ΠραγματικόΥποκείμενο=Δείκτης()
    Τμήμα Μέθοδος_Α {
        .ΠραγματικόΥποκείμενο=>Μέθοδος_Α
    }
    Συνάρτηση Μέθοδος_Β {
        Αν Όχι .φορτωμένη_εικόνα Τότε
            .φορτωμένη_εικόνα<=.ΠραγματικόΥποκείμενο=>Μέθοδος_Β()
        Τέλος Αν
        =.φορτωμένη_εικόνα
```

```
    }
    Τμήμα Μέθοδος_C {
        Αν Όχι .φορτωμένη_εικόνα Τότε
            .φορτωμένη_εικόνα<=.ΠραγματικόΥποκείμενο=>Μέθοδος_B()
        Τέλος Αν
        .ΠραγματικόΥποκείμενο=>Μέθοδος_C
    }
Κλάση:
    Τμήμα Πληρεξούσιος (Υποκείμενο ως *Υποκείμενο) {
        .ΠραγματικόΥποκείμενο<=Υποκείμενο
    }
}
Κλάση ΠραγματικόΥποκείμενο ως Υποκείμενο {
    Τμήμα Μέθοδος_A {
        Τύπωσε "Το όνομα αρχείου εικόνας:"; .όνομα_αρχείου$
    }
    Συνάρτηση Μέθοδος_B {
        .φορτωμένη_εικόνα<=Αληθές
        Τύπωσε "Φορτώνω την εικόνα"
        =.φορτωμένη_εικόνα
    }
    Τμήμα Μέθοδος_C {
        Αν .φορτωμένη_εικόνα Τότε
            Τύπωσε "Σχεδιάζω την εικόνα"
        Τέλος Αν
    }
Κλάση:
    Τμήμα ΠραγματικόΥποκείμενο (.όνομα_αρχείου$) {
    }
}
M=ΠραγματικόΥποκείμενο("alfa.bmp")
M.Μέθοδος_A
Αν M.Μέθοδος_B() Τότε M.Μέθοδος_C


K=Πληρεξούσιος(Δείκτης(ΠραγματικόΥποκείμενο("beta.bmp")))
K.Μέθοδος_A
\\ Ο Πληρεξούσιος μπορεί να καλέσει τη  Μέθοδος_B πριν καλέσει την Μέθοδος_C
K.Μέθοδος_C
K.Μέθοδος_C
```

***Πρόγραμμα 77**: Μοτίβο Proxy*

Επίλογος

Σε αυτό το άρθρο είδαμε πώς η γλώσσα Μ2000 μπορεί να υποστηρίξει τον αντικειμενοστραφή προγραμματισμό, μέσα από τα παραδείγματα για τα γνωστά μοτίβα σχεδιασμού. Η γλώσσα μπορεί να συνδυάζει πολλά προγραμματιστικά παραδείγματα. Στο παράρτημα δίνονται μερικά παραδείγματα σε θέματα που δεν καλύφθηκαν από το κύριο άρθρο αλλά κρίθηκαν σκόπιμο να παραταθούν για μια ανάγνωση-ενημέρωση.

Αναφορές

Καρράς Γ. Επίσημος Τόπος της Μ2000: Πληροφορική Προγραμματισμός, Τελευταία πρόσβαση 22 Αυγ 2020, Ιστοχώρος: https://georgekarras.blogspot.com

GitHub (2015), Κώδικας της Μ2000, Τελευταία πρόσβαση 22 Αυγ 2020, Ιστοχώρος: https://github.com/M2000Interpreter/Version9

Παράρτημα

Συναρτησιακός Προγραμματισμός

Στο συναρτησιακό προγραμματισμό μας ενδιαφέρει η δυνατότητα να φτιάχνουμε συναρτήσεις από συναρτήσεις (σύνθεση συναρτήσεων). Αυτό μπορεί να γίνει με τις λάμδα συναρτήσεις. Εδώ όμως θα δείξουμε πως δομές επανάληψης και επιλογής μέσα σε λάμδα συναρτήσεις συνθέτουν μια δυναμική δομή, κατά την εκτέλεση. Όπως εξηγήθηκε στο άρθρο, οι δομές επιλογής που εξετάσαμε ήταν και είναι στατικές. Δηλαδή οι δομές είναι όπως τις βλέπουμε στο κώδικα με τη διαφορά ότι δεν ξέρουμε αν δεν τρέξουμε το πρόγραμμα ποιες επιλογές θα γίνουν, άρα ποιος τελικά κώδικας θα εκτελεστεί. Σε δυναμικές δομές, δεν ξέρουμε τον τελικό κώδικα, γιατί εξαρτιέται από τα ορίσματα. Στο παράδειγμα η συνάρτηση ΒήμαΜετάθεσης() παίρνει ένα πίνακα (ένα δείκτη σε πίνακα, για το λόγο αυτό δεν βάζουμε το **α()** αλλά το **α** ως δείκτη), και δίνει μια λάμδα συνάρτηση με ένα βάθος συναρτήσεων (όπου κάθε συνάρτηση περιέχει ως κλείσιμο (closure) μια άλλη συνάρτηση), ανάλογο του αριθμού των στοιχείων στο πίνακα.

Στο πρόγραμμα φαίνονται έτοιμες συναρτήσεις της Μ2000, για να χειριζόμαστε αυτόματους πίνακες (ή tuple), όπως: Μήκος() που δίνει τον αριθμό στοιχείων, το Πρώτο() που δίνει ένα πίνακα με ένα στοιχείο το πρώτο (ή μηδενικό πίνακα), το Επόμενα() το οποίο δίνει ένα πίνακα από άλλο πίνακα με όλα τα στοιχεία από το δεύτερο και μετά, το Ένωση() όπου δημιουργεί ένα νέο πίνακα από την ένωση δυο ή περισσοτέρων πινάκων.

Δείτε επίσης ότι όπου θέλουμε πέρασμα με αναφορά χρησιμοποιούμε το &. Πρέπει να χρησιμοποιηθεί και στη κλήση (για να μπει η ισχνή αναφορά) και στη Διάβασε για γίνει επίλυση ισχνής σε πραγματική αναφορά. Πχ στο γ1= Λάμδα (&φ, α) ->{ κώδικας } ο διερμηνευτής βάζει στο κώδικα το { Διάβασε &φ, α : κώδικας }. Πάντα υπάρχει η Διάβασε για να διαβάσει από το σωρό τιμών, τον οποίο χρησιμοποιεί ο διερμηνευτής κατά την κλήση τμημάτων και συναρτήσεων.

Προσέξτε επίσης ότι η επιστροφή τιμής σε μια συνάρτηση γίνεται με το = σαν εντολή και όταν εκτελείται δεν γίνεται τερματισμός της συνάρτησης, αλλά ο τερματισμός γίνεται στο τέλος του μπλοκ. Ειδικά για τις λάμδα συναρτήσεις αυτό το χαρακτηριστικό διευκολύνει για να

προετοιμάσουμε την επόμενη εκτέλεση της συνάρτησης (εδώ στο παράδειγμα αποφασίζουμε αν η περιστροφή των στοιχείων ενός εσωτερικού πίνακα ολοκληρώθηκε ή όχι).

```
Τμήμα ΒήμαΜεΒήμα {
   Συνάρτηση ΒήμαΜετάθεσης(α ως πίνακας) {
      γ1=Λάμδα (&φ, α) ->{
         =α
         φ=Αληθής
      }
      μ=Μήκος(α)
      Αν μ=0 Τότε Λάθος "Δεν υπάρχει κάτι στο πίνακα"
      γ=γ1
      Ενώ μ>1 {
         γ1=Λάμδα γ2=γ,π, μ=(,) (&φ, α) ->{
            Αν Μήκος(μ)=0 Τότε μ=α
            =Ένωση(Πρώτο(μ),γ2(&φ, Επόμενα(μ)))
            Αν φ Τότε
               φ=Ψευδής:π++: μ=Ένωση(Επόμενα(μ), Πρώτο(μ))
               Αν π=Μήκος(μ) Τότε π=0 : μ=(,): φ=Αληθής
            Τέλος Αν
         }
         γ=γ1
         μ--
      }
      =Λάμδα γ, α (&φ) -> {
            =γ(&φ, α)
      }
   }
   Αδειασε
   Σειρά (1,2,3,4), (100,200,300), ("A", "B", "C", "D")
   Σειρά ("DOG", "CAT", "BAT")
   Ενώ όχι κενό
      Διάβασε πιν1
      κ=Ψευδής
      ΒήμαΑ=ΒήμαΜετάθεσης(πιν1)
      Ενώ Όχι κ {Τύπωσε ΒήμαΑ(&κ)}
   Τέλος Ενώ
}
ΒήμαΜεΒήμα
```

***Πρόγραμμα 78**: Παράδειγμα Συναρτησιακού Προγραμματισμού*

Υπερκλάση στη Μ2000

Μπορούμε για κάθε αντικείμενο ομάδα να ορίσουμε μια Υπερκλάση. Η υπερκλάση αυτή διαφέρει από τη γενική ιδέα των κλάσεων που δημιουργούν ιεραρχία κλάσεων, όπου η βάση ή υπερκλάση δίνει μια υποκλάση.

Μια υπερκλάση στη Μ2000 είναι ένα αντικείμενο ομάδα που δεν δίνει διεπαφή άμεσα, παρά μόνο όταν ζητηθεί μέσα σε ένα αντικείμενο που συνδέεται με αυτήν. Ο λόγος ύπαρξης αυτού του τύπου υπερκλάσης είναι για να έχουμε διάφορα αντικείμενα που να συνδέονται με μια κοινή υπερκλάση, ένα κοινό αντικείμενο, που κρατάει κοινή κατάσταση (state), για όλα τα αντικείμενα που συνδέονται με αυτή.

Οι ιδιότητες της Υπερκλάσης στη Μ2000 είναι:

- Δημιουργούν αντικείμενα με σύνδεση στην υπερκλάση
- Έχουν μοναδικά μέλη που δεν κληρονομούνται
- Τα αντικείμενα διαβάζουν ιδιότητες ή να εκτελούν μεθόδους της Υπερκλάσης.
- Τα δημιουργημένα αντικείμενα μπορούν να επεκταθούν χωρίς να χάσουν την υπερκλάση.
- Κάθε ομάδα σε ομάδα μπορεί να έχει δική της υπερκλάση.
- Κατά τη συγχώνευση ομάδας αν η ομάδα που θα συγχωνευθεί σε άλλη έχει άλλη υπερκλάση τότε θα πάρει την νέα υπερκλάση (το δείκτη της). Αυτό θα δημιουργήσει πρόβλημα αν δεν έχουμε σωστό σχεδιασμό (αν δεν έχουν και οι δυο υπερκλάσεις την ίδια διεπαφή).

Σκοπός της υπερκλάση στη Μ2000 είναι να κρατάει στοιχεία μοναδικά, στα οποία θα έχουν πρόσβαση όλα τα αντικείμενα που δημιουργούνται από αυτήν. Με την εισαγωγή στο διερμηνευτή των δεικτών σε ομάδες μπορούμε να έχουμε πολλές ομάδες που να περιέχουν ένα δείκτη προς μια άλλη ομάδα και αυτή να κρατάει μια κοινή κατάσταση (κοινά στοιχεία). Παρόλα αυτά η υπερκλάση προστατεύει τη δική της κατάσταση, καλύτερα από μια άλλη ομάδα, αφού μόνο όσες ομάδες έχουν την ίδια υπερκλάση έχουν πρόσβαση σε αυτήν, ενώ η ίδια υπερκλάση ως ομάδα δεν βγάζει διεπαφή.

Στο πρόγραμμα που ακολουθεί βλέπουμε μια ετικέτα **Μοναδικό** η οποία ορίζει ιδιότητες και μεθόδους οι οποίες θα είναι μοναδικές για την Υπερκλάση. Επίσης ορίζουμε και άλλες ιδιότητες και μεθόδους. Η Αλφα ως μεταβλητή είναι μια ομάδα που δεν έχει τίποτα παρά μόνο ένα δείκτη στην Υπερκλάση. Με τη Αλφα μπορούμε να δημιουργούμε αντικείμενα. Τα αντικείμενα θα έχουν ότι έχει η Αλφα και δεν είναι μοναδικό. Εννοείται ότι και με όποια άλλη ομάδα που φτιάξαμε από την Άλφα μπορούμε να δημιουργούμε νέες ομάδες, με την ίδια σύνδεση στην Άλφα. Επίσης στο παράδειγμα γίνεται χρήση τελεστή ++ τον οποίο προγραμματίζουμε για την ομάδα που παράγει το Άλφα. Σημειώστε ότι τα Άλφα.Σύνολο και Άλφα++ δεν υπάρχουν γιατί η Αλφα δεν δείχνει να έχει τίποτα!

```
Υπερκλάση Άλφα {
Μοναδικό:
    Άθροισμα=10
```

```
    Τμήμα Διπλάσιο {
        .Άθροισμα*=2
    }
Ιδιωτικό:
    Συνάρτηση ΔεςΆθροισμα {
        Για Υπερκλάση {
            =.Άθροισμα
            .Διπλάσιο
        }
    }
Δημόσιο:
    Τελεστής "++" {
        Για Υπερκλάση {
            .Άθροισμα++
        }
    }
    Ιδιότητα Σύνολο {
        Αξία {
            Ένωσε γονικό ΔεςΆθροισμα() στη Δ()
            Αξία=Δ()
        }
    }
}
Α=Άλφα
Β=Άλφα
Τύπωσε Α.Σύνολο=10, Β.Σύνολο=20, Α.Σύνολο=40
Α++
Β++
Τύπωσε Β.Σύνολο=82, Α.Σύνολο=164
```

***Πρόγραμμα 79**: Χρήση Αντικειμένου Υπερκλάση*

Το ίδιο γίνεται, σχεδόν, με το παρακάτω πρόγραμμα με χρήση δείκτη σε ομάδα, με τη διαφορά ότι μπορούμε να έχουμε τη Διαγραφή που θα κληθεί όταν κανένας δείκτης δεν θα δείχνει το μοναδικό αντικείμενο που τα Α και Β δείχνουν μέσω της ιδιωτικής Υπ. Εδώ φτιάχνουμε την Άλφα η οποία όμως δεν είναι σαν την Υπερκλάση στο προηγούμενο παράδειγμα αλλά σαν τις Α και Β, δηλαδή υπάρχει η Άλφα.Σύνολο και η Αλφα++

```
Κλάση Αλφα {
Ιδιωτικό:
    Υπ=Δείκτης()
    Συνάρτηση ΔεςΆθροισμα {
        Για .Υπ {
            =.Άθροισμα
            .Διπλάσιο
```

```
        }
    }
Δημόσιο:
    Τελεστής "++" {
        Για .Υπ {
            .Άθροισμα++
        }
    }
    Ιδιότητα Σύνολο {
        Αξία {
            Ένωσε γονικό ΔεςΆθροισμα() στη Δ()
            Αξία=Δ()
        }
    }
Κλάση:
    Τμήμα Αλφα {
        Κλάση Υπερ {
            Άθροισμα=10
            Τμήμα Διπλάσιο {
                .Άθροισμα*=2
            }
            Διαγραφή {
                Τύπωσε "Διαγράφτηκα"
            }
        }
        .Υπ<=Δείκτης(Υπερ())
    }
}
Αλφα=Αλφα()
Α=Αλφα
Β=Αλφα
Τύπωσε Α.Σύνολο=10, Β.Σύνολο=20, Α.Σύνολο=40
Α++
Β++
Τύπωσε Β.Σύνολο=82, Α.Σύνολο=164
```

***Πρόγραμμα 80**: Χρήση Δείκτης σε Ομάδα ως Υπερκλάση*

Προγραμματισμός Τελεστών σε Ομάδες

Μπορούμε να ορίσουμε τελεστές για τις ομάδες. Εδώ δεν θα γίνει εκτεταμένη αναφορά στο θέμα των τελεστών. Το παράδειγμα που ακολουθεί περιέχεται, με επεξηγήσεις, και στο ελληνικό εγχειρίδιο της γλώσσας που συμπεριλαμβάνεται στο αρχείο εγκατάστασης (το βρίσκουμε στο

μενού Έναρξη, Προγράμματα, M2000, GreekManual). Το αρχείο pdf, όπως και το παρόν αρχείο, ανοίγει ως αρχείο κειμένου σε LibreOffice (και από εκεί είναι καλύτερα να γίνει αντιγραφή κώδικα).

```
Τμήμα Κλάσματα1 {
    Κλάση Κλάσμα {
        αριθμητής ως αριθμός, παρονομαστής ως αριθμός
        μκδ=λάμδα->0
        εκπ=λάμδα->0
        Τελεστής "+" {
            Διάβασε Τ
            παράγοντας=.εκπ(Τ.παρονομαστής, .παρονομαστής)
            .αριθμητής<= παράγοντας/Τ.παρονομαστής * Τ.αριθμητής +
παράγοντας/.παρονομαστής * .αριθμητής
            Αν .αριθμητής==0 Τότε παράγοντας=1
            .παρονομαστής<=παράγοντας
        }
        Τελεστής Μοναδιαίος {
            .αριθμητής-!
        }
        Τελεστής "-" {
            Διάβασε Τ
            Κάλεσε Τελεστή "+", -Τ
        }
        Τελεστής Υψηλός "*" {
            Διάβασε Τ
            γ1=.μκδ(Τ.αριθμητής,.παρονομαστής)
            γ2=.μκδ(.αριθμητής, Τ.παρονομαστής)
            Βάλε Τ.αριθμητής/γ1*.αριθμητής/γ2
            Βάλε Τ.παρονομαστής/γ2*.παρονομαστής/γ1
            Διάβασε .παρονομαστής, .αριθμητής

        }
        Συνάρτηση Αντίστροφο {
            Αν .αριθμητής==0 Τότε Λάθος "Διαίρεση με το μηδέν"
            επ=Αυτό
            πρόσημο=Σημ(επ.αριθμητής) : Αν πρόσημο<0 Τότε επ.αριθμητής-!
            Άλλαξε επ.αριθμητής, επ.παρονομαστής
            Αν πρόσημο<0 Τότε επ.αριθμητής-!
            =επ
        }
        Τελεστής Υψηλός "/" {
            Διάβασε Τ
            κάνε λογικό Πρόβλημα=Αληθές
            Δες {
                Κάλεσε Τελεστή "*", Τ.Αντίστροφο()
```

```
        Πρόβλημα~
    }
    Αν Πρόβλημα Τότε Λάθος "Διαίρεση με το μηδέν"
}
Συνάρτηση Ύψωση {
    Διάβασε δύναμη ως μακρύς
    επ=Αυτό
    κάνε λογικό Πρόβλημα=Αληθές
    Δες {
        επ.αριθμητής<=.αριθμητής^δύναμη
        επ.παρονομαστής<=.παρονομαστής^δύναμη
        Πρόβλημα~
    }
    Αν Πρόβλημα Τότε Λάθος "Ύψωση δύναμης εκτός περιοχής"
    =επ
}
Τελεστής "=" {
    Διάβασε Τ
    Κάνε Λογικό Τ=Αληθής, Κ=Ψευδής
    Αν Απόλ(Σημ(Τ.αριθμητής))+Απόλ(Σημ(.αριθμητής))=0 Τότε Βάλε Τ: Έξοδος
    Αν Σημ(Τ.αριθμητής) <>Σημ(.αριθμητής) Τότε Βάλε Κ : Έξοδος
    πΥπολ=Τ/Αυτό
    Βάλε πΥπολ.αριθμητής=1 και πΥπολ.παρονομαστής=1
}
Τελεστής ">" {
    Διάβασε Τ
    Κάνε Λογικό Κ
    Αν Απόλ(Σημ(Τ.αριθμητής))+Απόλ(Σημ(.αριθμητής))=0 Τότε Βάλε Κ: Έξοδος
    Αν Σημ(Τ.αριθμητής)=0 Τότε {
        Βάλε .αριθμητής>0
    } Αλλιώς {
        πΥπολ=Αυτό/Τ
        Βάλε πΥπολ.πραγματικός>1
    }
}
Τελεστής ">=" {
    Διάβασε Τ
    Αν Σημ(Τ.αριθμητής)=0 Τότε {
        Βάλε .αριθμητής>=0
    } Αλλιώς {
        πΥπολ=Αυτό/Τ
        Βάλε πΥπολ.πραγματικός>=1
    }
}
```

```
Τελεστής "<" {
    Διάβασε Τ
    Κάνε Λογικό Κ
    Αν Απόλ(Σημ(Τ.αριθμητής))+Απόλ(Σημ(.αριθμητής))=0 Τότε Βάλε Κ: Έξοδος
    Αν Σημ(Τ.αριθμητής)=0 Τότε {
        Βάλε .αριθμητής<0
    } Αλλιώς {
        πΥπολ=Αυτό/Τ
        Βάλε πΥπολ.πραγματικός<1
    }
}
Τελεστής "<=" {
    Διάβασε Τ
    Αν Σημ(Τ.αριθμητής)=0 Τότε {
        Βάλε .αριθμητής<=0
    } Αλλιώς {
        πΥπολ=Αυτό/Τ
        Βάλε πΥπολ.πραγματικός<=1
    }
}
Τελεστής "<>" {
    Διάβασε Τ
    Αν Σημ(Τ.αριθμητής)=0 Τότε {
        Βάλε .αριθμητής<>0
    } Αλλιώς {
        πΥπολ=Αυτό/Τ
        Βάλε πΥπολ.πραγματικός<>1
    }
}
Ομάδα πραγματικός {
    Αξία {
        Ένωσε γονικό αριθμητής, παρονομαστής στο ν, δ
        =ν/δ
    }
}
Ομάδα ΣεΛέξη$ {
   Αξία {
        Ένωσε γονικό αριθμητής, παρονομαστής στο ν, δ
        =Γραφή$(ν)+"/"+Γραφή$(δ,"")
    }
}
Κλάση:
Τμήμα Κλάσμα (.αριθμητής, .παρονομαστής) {
    Αν .παρονομαστής=0 Τότε Λάθος "Μηδενικός παρονομαστής"
```

```
            πρόσημο=Σημ(.αριθμητής)*Σημ(.παρονομαστής)
            .παρονομαστής<=Απόλ(.παρονομαστής)
            .αριθμητής<=Απόλ(.αριθμητής)*πρόσημο
            Αν .αριθμητής=0 τότε έξοδος
            μκδ1=λάμδα (α ως αριθμός, β ως αριθμός) -> {
                Αν α<β Τότε Άλλαξε α,β
                μ=α υπολ β
                Ενώ μ {
                    α=β:β=μ: μ=α υπολ β
                }
                =Απόλ(β)
            }
            μκδΤιμή=μκδ1(Απόλ(.αριθμητής), .παρονομαστής)
            Αν μκδΤιμή<.παρονομαστής και μκδΤιμή<>0 Τότε
                .παρονομαστής/=μκδΤιμή
                .αριθμητής/=μκδΤιμή
            Τέλος Αν
            .μκδ<=μκδ1
            .εκπ<=λάμδα μκδ=μκδ1 (α ως αριθμός, β ως αριθμός) -> {
                =α/μκδ(α,β)*β
            }
        }
    }
    Τύπωσε Κλάσμα(-3,3)<>Κλάσμα(-3,3) ' Ψευδής
    M=Κλάσμα(10, 150)
    N=Κλάσμα(2, 4)
    Τύπωσε "M.πραγματικός+N.πραγματικός=";M.πραγματικός+N.πραγματικός
    Τύπωσε "Z=M+N"
    Z=M+N
    Τύπωσε 10/150@+2/4@
    Τύπωσε "Z.πραγματικός="; Z.πραγματικός
    Τύπωσε "(";M.αριθμητής;"/"; M.παρονομαστής;") + (";N.αριθμητής;"/";
N.παρονομαστής;") = (";Z.αριθμητής;"/";Z.παρονομαστής;")"
    Τύπωσε M.ΣεΛέξη$+ " +"+N.ΣεΛέξη$+" ="+Z.ΣεΛέξη$
    Τύπωσε -10/150@+2/4@
    Z=-M+N
    Τύπωσε "-"+M.ΣεΛέξη$+ " +"+N.ΣεΛέξη$+" ="+Z.ΣεΛέξη$
    Τύπωσε Z.αριθμητής, Z.παρονομαστής, Z.αριθμητής/Z.παρονομαστής
    Τύπωσε -10/150@+2/4@
    Τύπωσε Z.πραγματικός
    Z=M-N
    Τύπωσε Z.αριθμητής, Z.παρονομαστής
    Τύπωσε 10/150@-2/4@
    Τύπωσε Z.πραγματικός
```

```
Z=M*N
Τύπωσε Ζ.αριθμητής, Ζ.παρονομαστής
Τύπωσε (10/150@)*(2/4@)
Τύπωσε Ζ.πραγματικός
Z=M/N
Τύπωσε Ζ.αριθμητής, Ζ.παρονομαστής
Τύπωσε (10/150@)/(2/4@)
Τύπωσε Ζ.ΣεΛέξη$
Τύπωσε Ζ.πραγματικός
Τύπωσε "Ζ Ύψωση 2 = ";
Z=Ζ.Ύψωση(2)
Τύπωσε Ζ.πραγματικός
Τύπωσε Ζ.ΣεΛέξη$;" = ";Έκφρ(Ζ.ΣεΛέξη$)
Τύπωσε Z=Z
Τύπωσε Z=N ' Ψευδής
Τύπωσε Z=-Z ' Ψευδής
ZZ=-Z
Τύπωσε ZZ=ZZ
Τύπωσε -Z=-Z
Τύπωσε Ζ.αριθμητής, Ζ.παρονομαστής
Τύπωσε Ζ.πραγματικός, Ζ.ΣεΛέξη$
\\ Πίνακας με κλάσματα
Πίνακας Κ(100)=Κλάσμα(1,1)
M=K(4)+K(3)
Τύπωσε Μ.πραγματικός
Τύπωσε Κ(4).ΣεΛέξη$
\\ πκ δείκτης σε νέο αντίγραφο του Ζ
πκ->(Ζ)
Τύπωσε πκ=>ΣεΛέξη$
Τύπωσε πκ=>ΣεΛέξη$+" +"
ZZZ=πκ+πκ
Τύπωσε ZZZ.ΣεΛέξη$+ " = ";
Τύπωσε πκ=>πραγματικός**2
Τύπωσε (1/ZZZ.πραγματικός)**2000=Άπειρο
Δες Οκ {
    KKK=ZZZ.Ύψωση(2000)
}
Αν όχι Οκ Τότε
    Αναφορά "Λάθος"+Λάθος$
    KKK=ZZZ.Ύψωση(2)
Τέλος Αν
Τύπωσε KKK.Πραγματικός
πκ->Ζ
Τύπωσε πκ=>πραγματικός
```

```
    Για πκ {
        Αυτό=Κλάσμα(3,4)
        Τύπωσε .πραγματικός
    }
    Τύπωσε Ζ.ΣεΛέξη$, Ζ.πραγματικός=0.75
    \\ αν αφαιρέσουμε το Υψηλό στο Τελεστή "*"
    \\ θα γίνουν "επίπεδες" οι πράξεις
    ΖΖΖ=Κλάσμα(10,1)-Κλάσμα(3,1)*Κλάσμα(2,1)
    Τύπωσε ΖΖΖ.ΣεΛέξη$, ΖΖΖ.πραγματικός=4
    ΖΖΖ=Κλάσμα(10,1)*Κλάσμα(3,1)-Κλάσμα(2,1)
    Τύπωσε ΖΖΖ.ΣεΛέξη$, ΖΖΖ.πραγματικός=28
    Τύπωσε Κλάσμα(2,3)>Κλάσμα(1,3) ' Αληθής
    Τύπωσε Κλάσμα(2,3)*Κλάσμα(1,2)>Κλάσμα(1,3) ' ψευδής
    Τύπωσε Κλάσμα(1,2)>Κλάσμα(1,3) ' αληθής
    Τύπωσε Κλάσμα(-1,2)<Κλάσμα(0,3) ' αληθής
}
Φόρμα 80, 50
Κλάσματα1
```

***Πρόγραμμα 81**: Κλάση Κλάσμα - Χρήση Προγραμματιζόμενων Τελεστών*

Πίνακας περιεχομένων

Πίνακας Προγραμμάτων