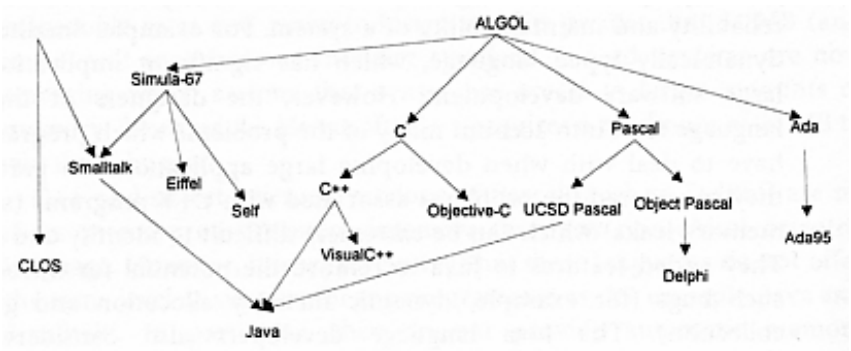
2 Αντικειμενοστρεφής προγραμματισμός

2.1 Τα δεδομένα στο επίκεντρο

- Στον δομημένο προγραμματισμό, η προσοχή μας εστιάζεται στην επεξεργασία (processing), δηλαδή στις ενέργειες που πρέπει να εκτελεστούν – τα δεδομένα έρχονται στο προσκήνιο αργότερα, όταν είναι προφανές τι λειτουργίες πρέπει να εκτελεστούν πάνω τους. Αυτό είναι καλό, γιατί μπορούμε να επιλέξουμε δομές δεδομένων που να αποδίδουν βέλτιστα για το σύνολο των απαιτούμενων λειτουργιών. Ωστόσο, εάν αργότερα χρειαστεί να τροποποιήσουμε το πρόγραμμα και να προσθέσουμε μια νέα λειτουργία στα δεδομένα, η δόμηση του προγράμματος χρειάζεται εκτεταμένες αλλαγές.
- Κατά την αντικειμενοστρεφή (object-oriented, O-O) προσέγγιση, το σύστημα αποτελείται από αλληλεπιδρώντα αντικείμενα. Κάθε αντικείμενο έχει ιδιότητες ή χαρακτηριστικά που οι τιμές τους αναπαριστάνουν την τρέχουσα κατάστασή του. Επίσης απαντά σε ένα σύνολο ενεργειών που ορίζουν τη συμπεριφορά του.
  - Πιο συγκεκριμένα, το αντικείμενο περιλαμβάνει: (α) δεδομένα που αναπαριστάνουν τις ιδιότητές του, και (β) λειτουργίες που υλοποιούν τις πράξεις που μπορούν να εφαρμοστούν πάνω του. Το πρόγραμμα δημιουργεί αντικείμενα και καλεί μεθόδους τους για να τα διαχειριστεί.
- Ας συγκρίνουμε τις δύο προσεγγίσεις με ένα παράδειγμα. Θέλουμε να φτιάξουμε ένα πρόγραμμα που λειτουργεί πάνω σε διάφορα γεωμετρικά σχήματα, πχ ευθύγραμμα τμήματα, κύκλους, παραλληλόγραμμα, κλπ. Θέλουμε τα προβάλουμε, να υπολογίζουμε το εμβαδόν τους, να τα μετακινούμε, να βρίσκουμε αν τέμνονται, κά.
  - Κατά το δομημένο προγραμματισμό, το πρόγραμμα οργανώνεται με βάση τις λειτουργίες. Για παράδειγμα, θα έχουμε μια μέθοδο που θα σχεδιάζει το σχήμα που της δίνεται ως παράμετρος:

```
public double epifaneia(Sxhma s) {
  switch (s.type) {
    case KYKLOS: return Math.PI * s.aktina * s.aktina;
    case PARALLHLOGRAMMO: /* αντίστοιχος κώδικας */
    ...
  } //end switch
} //end epifaneia
```

  - Κατά τον αντικειμενοστρεφή προγραμματισμό, το πρόγραμμα οργανώνεται με βάση τα σχήματα. Για κάθε σχήμα ορίζονται τα χαρακτηριστικά του και υλοποιούνται στην κλάση του οι λειτουργίες στις οποίες ανταποκρίνεται. Έτσι μπορούμε να προσθέσουμε σχήματα στη συλλογή μας χωρίς να αλλάζουμε τον υφιστάμενο κώδικα των σχημάτων που έχουμε ήδη φτιάξει.
- Κύριοι στόχοι του αντικειμενοστρεφούς προγραμματισμού είναι η **επαναχρησιμοποίηση κώδικα** (software reuse) και η **ακρίβεια** της **μοντελοποίησης** του συστήματος που προσομοιώνουμε με το πρόγραμμα.
- Χαρακτηριστικά αντικειμενοστρεφών συστημάτων ανάπτυξης λογισμικού:
  - Αρθρωσιμότητα (modularity): διαίρεση σε μικρότερα κομμάτια, βλέπε και δομημένος προγραμματισμός
  - Αφαίρεση (abstraction): επικεντρώνει την προσοχή στις κύριες ή σχετικές ιδιότητες του αντικειμένου και αγνοεί τις δευτερεύουσες ή επουσιώδεις.
  - **Ενθυλάκωση** (encapsulation): προσφέρει προστασία ιδιωτικών δεδομένων και δίνει τη δυνατότητα να αλλάξει η εσωτερική αναπαράσταση των αντικειμένων, χωρίς αυτό να γίνεται αντιληπτό στα σημεία κλήσης των μεθόδων.
  - Ιεραρχία και κληρονομικότητα (inheritance): εξειδίκευση κλάσεων μέσω επέκτασης της συμπεριφοράς άλλων κλάσεων
- Γλώσσες που υποστηρίζουν αντικειμενοστρεφή προγραμματισμό είναι οι Smalltalk, Java , C++.
  - Ειδικά οι δύο πρώτες είναι πλήρως αντικειμενοστρεφείς: όλα τα δεδομένα (μπορούν να) είναι αντικείμενα και τα προγράμματα είναι κλάσεις. Επίσης υποστηρίζουν τα παραπάνω τέσσερα χαρακτηριστικά.
  - Καταρχήν είναι εφικτό να αναπτύξει κάποιος ένα αντικειμενοστρεφές πρόγραμμα, χωρίς να χρησιμοποιήσει μια αμιγώς αντικειμενοστρεφή γλώσσα, όπως αυτές που αναφέρονται παραπάνω[1]. Άλλες γλώσσες έχουν επεκταθεί με αντικειμενοστρεφή χαρακτηριστικά, όπως η Visual Basic. Στη δεκαετία του '70 είχαμε ένα παρόμοιο φαινόμενο όταν οι πρώτες διαδικαστικές γλώσσες, όπως η Basic και η Fortran επεκτείνονταν με μηχανισμούς κλήσης υπορουτινών και περάσματος παραμέτρων για να υποστηρίξουν τις αρχές του δομημένου προγραμματισμού.



Εικόνα 25 Πρόγονοι της Java

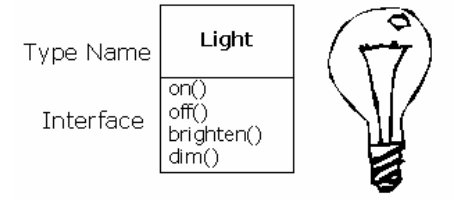
[1] Και το αντίστροφο ισχύει, και μάλιστα συμβαίνει συχνά: μπορεί κάποιος να χρησιμοποιεί μια αντικειμενοστρεφή γλώσσα προγραμματισμού κι όμως να αναπτύσσει μη αντικειμενοστρεφή προγράμματα.

- Ο Alan Kay συνοψίζει πέντε χαρακτηριστικά της Smalltalk, της πρώτης αντικειμενοστρεφούς γλώσσας προγραμματισμού που ισχύουν σε μεγάλο βαθμό και για τη Java:
  - Τα πάντα είναι αντικείμενα.[2]
  - Το πρόγραμμα είναι μια ομάδα αντικειμένων που λένε το ένα στο άλλο τι να κάνει στέλνοντας μηνύματα. Τα μηνύματα δηλαδή είναι το ανάλογο των υπορουτινών στις διαδικαστικές γλώσσες προγραμματισμού.
  - Κάθε αντικείμενο έχει τη μνήμη του και αποτελείται από άλλα αντικείμενα.
  - Κάθε αντικείμενο έχει έναν τύπο. Ακολουθώντας την ορολογία, κάθε αντικείμενο είναι στιγμιότυπο μιας κλάσης.
  - Όλα τα αντικείμενα του ίδιου τύπου μπορούν να λάβουν τα ίδια μηνύματα.

2.2 Η έννοια της αφαίρεσης

- Όλες οι γλώσσες προγραμματισμού είναι εργαλεία για τον ορισμό αφαιρέσεων[3] (abstractions). Αν και οι γλώσσες είναι υπολογιστικά ισοδύναμες, δηλαδή όλες μπορούν να λύσουν το ίδιο σύνολο προβλημάτων, η πολυπλοκότητα των προβλημάτων που μπορεί να λύσει ένας προγραμματιστής, εξαρτάται από το είδος και την ποιότητα της αφαίρεσης που υποστηρίζει η γλώσσα.
  - Όσο πιο υψηλού επιπέδου είναι μια γλώσσα, τόσο πιο μεγάλη αφαιρετική ικανότητα προσφέρει. Υπάρχουν γλώσσες που προσφέρουν συγκεκριμένες αφαιρέσεις και μπορούν να χρησιμοποιηθούν για να εκφράσουν λύσεις προβλημάτων που σχετίζονται με αυτές τις αφαιρέσεις. Για παράδειγμα, η Lisp είναι καλό εργαλείο για προβλήματα που αντιμετωπίζονται με λίστες και η Prolog για προβλήματα που λύνονται με εξαντλητική διερεύνηση του χώρου λύσεων μετά την εφαρμογή περιορισμών (constraints).
  - Αντίθετα, η assembly που είναι πολύ χαμηλού επιπέδου, προσφέρει στον προγραμματιστή μόνον τη δυνατότητα να χρησιμοποιεί μνημονικά ονόματα για τις εντολές του μικροεπεξεργαστή και τις θέσεις μνήμης του προγράμματος. Κατά τα άλλα ο προγραμματιστής εκφράζει τη λύση του προβλήματος χρησιμοποιώντας τις εντολές της γλώσσας μηχανής του υπολογιστή – στόχος.

- Με την αντικειμενοστρεφή προσέγγιση, τα προβλήματα επιλύονται δημιουργώντας αφαιρέσεις («μοντέλα») των αντικειμένων που εμπλέκονται στο πρόβλημα. Έτσι όταν κάποιος διαβάζει το πρόγραμμα, βλέπει αντικείμενα και λειτουργίες που έχουν ανάλογα στον χώρο του προβλήματος (problem space) και όχι στον χώρο της γλώσσας προγραμματισμού ή του υπολογιστή.
  - Παράδειγμα: Ας πούμε ότι σχεδιάζουμε ένα πρόγραμμα που ελέγχει το φωτισμό ενός καταστήματος. Ο φωτισμός γίνεται με λάμπες. Το αντικείμενο «λάμπα» (Light) έχει μια κατάσταση (state) και λειτουργίες για άναμμα, σβήσιμο. Επίσης, έχει δυνατότητα για μείωση / αύξηση φωτισμού.



Εικόνα 26 Το αντικείμενο λάμπα

  - Η εσωτερική αναπαράσταση της κατάστασης της λάμπας μπορεί να είναι ένας αριθμός που υποδηλώνει πόσο αναμμένη ή σβηστή είναι. Δείτε στην ενότητα 2.14 Υλοποίηση κλάσης λάμπα σε Java για μια υλοποίηση. Σημειώστε ότι η αναπαράσταση μπορεί να αλλάξει, πχ αντί για πραγματικός αριθμός που παίρνει τιμές μεταξύ 0 και 1, μπορούμε να χρησιμοποιήσουμε έναν ακέραιο με κάποιο εύρος $[0..n)$. Παρότι η υλοποίηση των υπορουτινών που υλοποιούν τη «συμπεριφορά» της λάμπας, δηλαδή οι εντολές στις οποίες ανταποκρίνεται θα αλλάξει, κάποιος που χρησιμοποιεί αντικείμενα της κλάσης, δεν θα αντιληφθεί την αλλαγή. Αυτό επιτυγχάνεται με την ενθυλάκωση.
  - Αφού φτιάξουμε την κλάση `Lampa` που αντιστοιχεί στο «πρότυπο» ή την προδιαγραφή, το επόμενο βήμα είναι να παράγουμε ένα ή περισσότερα «στιγμιότυπα» (instances) αντικειμένων, που αντιστοιχούν σε συγκεκριμένες λάμπες του καταστήματος. Πρέπει σε αυτό το σημείο να κατανοείτε τη διαφορά ανάμεσα σε μια κλάση (ορίζει τις ιδιότητες και τις λειτουργίες κάθε αντικειμένου της κλάσης) και στα αντικείμενα που έχουν κατασκευαστεί με βάση την κλάση.
  - Ας υποθέσουμε ότι μερικές λάμπες μπορούν μόνο να ανάβουν και να σβήνουν κι όχι να αυξομειώνουν τη ένταση του φωτισμού, ενώ σε κάποιες άλλες μπορούμε να αλλάξουμε το χρώμα. Τότε μπορούμε να ορίσουμε άλλες κατηγορίες ή κλάσεις που είναι κατά βάση λάμπες, αλλά τροποποιούν / επεκτείνουν τη συμπεριφορά της «βασικής» λάμπας.
  - Αν οι λάμπες έχουν διάφορα χρώματα, τότε πρέπει να συμπεριλάβουμε στις ιδιότητές τους το χρώμα. Στην περίπτωση που το χρώμα τους είναι σταθερό, θα επιτρέπουμε να ορίζεται μόνο κατά την κατασκευή της κάθε λάμπας και δεν θα υπάρχει δυνατότητα αλλαγής του αργότερα. Αντίθετα, αν η λάμπα μπορεί να αλλάζει χρωματισμούς, τότε θα πρέπει να παράσχουμε επιπρόσθετα μια λειτουργία που θα θέτει το τρέχον χρώμα της λάμπας στο αντίστοιχο αντικείμενο.

- Τα βήματα που ακολουθούμε για να σχεδιάσουμε τα αντικείμενα είναι ως εξής. (1) Ξεκινούμε επιλέγοντας τις οντότητες του προβλήματος που θα απεικονιστούν ως αντικείμενα. (2) Ορίζουμε μοντέλα αυτών των οντοτήτων όπου περιλαμβάνονται οι ιδιότητες και οι λειτουργίες τους. Προσπαθούμε να κάνουμε αυτά τα μοντέλα να προσομοιάζουν όσο το

---

[2] Βέβαια στη Java, *σχεδόν* όλα είναι αντικείμενα. Για λόγους επιδόσεων, οι βασικοί τύποι δεδομένων (αριθμοί, χαρακτήρες) *δεν* είναι αντικείμενα, εκτός αν ο προγραμματιστής χρησιμοποιήσει τις κλάσεις περιτυλίγματος (wrapper classes) `Integer`, `Long`, κτλ.

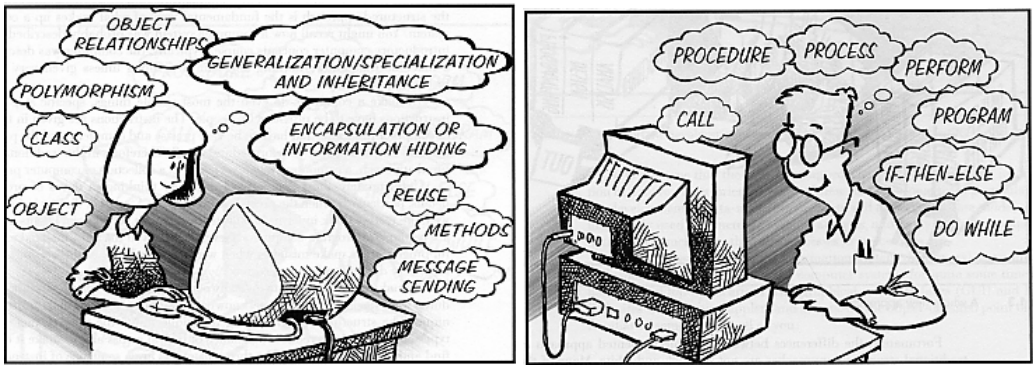
[3] Μια οπτική γωνία που μπορεί κάποιος να δει τα προγράμματα είναι ως κατασκευές που εκτελούνται πάνω σε γενικής χρήσης υπολογιστικές μηχανές και διαμορφώνουν (εξειδικεύουν) τη συμπεριφορά τους, ώστε να λύνουν κάποιο πρόβλημα. Το πρόγραμμα «σκεπάζει» τις λεπτομέρειες της εσωτερικής του λειτουργίας, καθώς και της συσκευής πάνω στην οποία εκτελείται και την «μεταμφιέζει» σε μια συσκευή που λύνει το πρόβλημα, δηλαδή αποδέχεται εισόδους κατά τις προδιαγραφές και παράγει τις νόμιμες εξόδους.

δυνατόν πιστότερα τις οντότητες του προβλήματος. (3) Κατασκευάζουμε στιγμιότυπα των μοντέλων και εφαρμόζουμε λειτουργίες επί αυτών.

- Ακόμη κι αν κάποιος προγραμματίζει ακολουθώντας πιστά το μοντέλο του αντικειμενοστρεφούς προγραμματισμού, όταν υλοποιεί την εσωτερική οργάνωση των αντικειμένων θα χρησιμοποιήσει τεχνικές δομημένου προγραμματισμού, πχ την οργάνωση του κώδικα σε μπλοκ εντολών, την κλήση σε υπορουτίνες, κλπ. Άρα τελικά οι δύο τεχνικές δεν είναι ξένες ή εχθρικές – αντίθετα, συνδυάζονται αρμονικά όποτε χρειάζεται.

2.3 Μειονεκτήματα του Ο-Ο προγραμματισμού

- Ασαφείς έννοιες και δυσνόητοι όροι. Επίσης, χρησιμοποιούνται διαφορετικοί όροι από τους συνηθισμένους των διαδικαστικών γλωσσών προγραμματισμού.
- Διαφορετική θεώρηση. Αρκετοί έχουν συνηθίσει την προσέγγιση του δομημένου προγραμματισμού και βρίσκουν πολύ διαφορετική την αντικειμενοστρεφή προσέγγιση.
- Έλλειψη επιδόσεων. Εκτός από τη διερμηνεία στην ιδεατή μηχανή, υπάρχουν εγγενείς αιτίες που προκαλούν αργά προγράμματα: η δυναμική σύνδεση των μηνυμάτων με τις μεθόδους, η δυναμική εκχώρηση μνήμης, κά.



Εικόνα 27 Έννοιες αντικειμενοστρεφούς και δομημένου προγραμματισμού

2.4 Αντικείμενο = δεδομένα + συμπεριφορά σε ενθυλάκωση

- Η **κλάση** (class) είναι ένα περίγραμμα ή ένα προσχέδιο για το πώς θα λειτουργεί και τι πληροφορίες ή ιδιότητες θα περιλαμβάνει ένα αντικείμενο. Αυτό το «καλούπι» χρησιμοποιείται για να παράγει **αντικείμενα** (objects). Το κάθε αντικείμενο έχει μια μοναδική ταυτότητα για να ξεχωρίζει από τα «αδέλφια» του.
  - Στην απλούστερη περίπτωση, μια κλάση μπορεί να ορίζει μόνον ιδιότητες και καθόλου συμπεριφορά για τα αντικείμενα που παράγει, δηλαδή να έχει μόνο μεταβλητές στιγμιότυπου. Τότε η κλάση λειτουργεί όπως οι τύποι δεδομένων οριζόμενοι από το χρήστη (user-defined data types) στις κλασικές γλώσσες προγραμματισμού.
  - Το σύνολο των ιδιοτήτων ενός αντικειμένου, που είναι δηλωμένα στην ενότητα «**μεταβλητές στιγμιότυπου**» στην Εικόνα 28, αποτελούν την κατάστασή του. Ένα κλάσμα έχει ιδιότητες τον αριθμητή και τον παρονομαστή του – δείτε την κλάση `Klasma` στην ενότητα 2.15 Κλασματικοί αριθμοί για μια υλοποίηση..
  - Παρατηρήστε ότι οι μεταβλητές στιγμιότυπου δηλώνονται σε επίπεδο κλάσης κι όχι μέσα σε κάποια μέθοδο. Αυτό το συναντούμε για πρώτη φορά – μέχρι τώρα όλες οι μεταβλητές δηλώνονταν στο εσωτερικό μιας μεθόδου, πχ της `main`.
  - Παρότι οι μεταβλητές στιγμιότυπου αρχικοποιούνται κατά τη δήλωσή τους σε προκαθορισμένη τιμή (μηδέν οι αριθμητικές, ψευδές οι λογικές), είναι ορθή προγραμματιστική τεχνική να τις αρχικοποιεί ο προγραμματιστής πριν διαβάσει την τιμή τους. Κατ' αντιδιαστολή οι τοπικές μεταβλητές *απαιτούν* αρχικοποίηση από τον προγραμματιστή πριν να διαβαστεί η τιμή τους – δεν αρχικοποιούνται αυτόματα από το σύστημα.

```
public class SomeClass { // το όνομα της κλάσης

// μεταβλητές στιγμιότυπου
/* εδώ δηλώνονται οι μεταβλητές που αντιστοιχούν
στις ιδιότητες κάθε αντικειμένου */

// κατασκευαστές
/* εδώ ορίζονται μία ή περισσότερες δημόσιες συναρ-
τήσεις με τις οποίες κατασκευάζονται νέα αντικείμε-
να – κυρίως αρχικοποιούν τις ιδιότητες */

// μέθοδοι ανάγνωσης get και εγγραφής set
/* εδώ ορίζονται δημόσιες συναρτήσεις getSomePrope-
rty και setSomeProperty που επιτρέπουν την πρόσβα-
ση και την αλλαγή στις τιμές ιδιοτήτων του αντικει-
μένου */

/* εδώ ορίζονται άλλες δημόσιες μέθοδοι που ορίζουν
τη συμπεριφορά των αντικειμένων της κλάσης */

/* εδώ ορίζονται βοηθητικές μέθοδοι ιδιωτικής εμβέ-
λειας που χρησιμοποιούνται από τις δημόσιες μεθό-
δους */

} //end class
```

Εικόνα 28 Ένας σκελετός για τον ορισμό μιας κλάσης Java

- **Απόκρυψη πληροφορίας** (information hiding) επιτυγχάνουμε περιορίζοντας την εμβέλεια των ιδιοτήτων στο εσωτερικό της κλάσης. Η δημόσια διεπαφή (public interface) της κλάσης αποτελείται από ενέργειες στις οποίες μπορεί να ανταποκριθεί το αντικείμενο.
  - Παράδειγμα: στο ραδιοκασετόφωνο αυτοκινήτου, η διεπαφή του αποτελείται από χειριστήρια και υποδοχές για κε-

ραία, ηχεία, κτλ που γνωρίζεις τι κάνουν ή πώς συνδέονται, χωρίς να ξέρεις πώς λειτουργούν.

- Μπορούμε να δηλώσουμε κάποιες ιδιότητες ως ιδιωτικές και κάποιες άλλες ως δημόσιες και το ίδιο ισχύει και για τις μεθόδους.
- Μια αρχή που ακολουθούν οι στρατιωτικοί είναι ότι η διάχυση της πληροφορίας πρέπει να γίνεται on-a-need-to-know-basis – αυτό σημαίνει ότι καθένας πρέπει να γνωρίζει μόνον ότι είναι απαραίτητο για να εκτελέσει σωστά την αποστολή του. Η αρχή αυτή εφαρμόζεται στον προγραμματισμό με τον περιορισμό της εμβέλειας όσο το δυνατόν περισσότερων ονομάτων μέσα στην κλάση. Όταν ένα όνομα (μεταβλητής ή μεθόδου) δηλώνεται **`private`**, η εμβέλειά του περιορίζεται στο εσωτερικό της κλάσης. Αντίθετα όταν δηλώνεται **`public`**, είναι ορατό και προσπελάσιμο από παντού. Όταν εισάγουμε την κληρονομικότητα, θα δούμε ότι η Java υποστηρίζει ακόμα μια δήλωση εμβέλειας, την **`protected`**.
- Αν περιορίσουμε την εμβέλεια των μεταβλητών στιγμιότυπου δηλώνοντάς τις ως **`private`**, είναι απαραίτητο να χρησιμοποιήσουμε ειδικές μεθόδους ανάγνωσης και εγγραφής μέσω των οποίων θα προσπελαύνονται.

2.5 Κατασκευαστές και καταστροφείς αντικειμένων

- Είναι οι ειδικές εκείνες μέθοδοι που αναλαμβάνουν τη δημιουργία νέων αντικειμένων και την καταστροφή των υπαρχόντων.

- Οι **κατασκευαστές** (constructors) στην Java πρέπει να έχουν ίδιο όνομα με την κλάση της οποίας αντικείμενα κατασκευάζουν και δεν επιστρέφουν τιμή. Ακόμη κι αν δεν γράψουμε κώδικα για κανένα κατασκευαστή, η Java από μόνη της φτιάχνει έναν «αόρατο» κατασκευαστή δίχως παραμέτρους που αρχικοποιεί τις μεταβλητές στιγμιότυπου στις εξ ορισμού τιμές τους.
  - Πριν να κατασκευάσουμε ένα αντικείμενο πρέπει να δηλώσουμε μια μεταβλητή που θα «δείχνει» αυτό το αντικείμενο και μέσω της οποίας θα προσπελαύνονται οι ιδιότητες και οι μέθοδοί του. Για παράδειγμα, η εντολή `Light lampaSaloniou = new Light(0);` δηλώνει τη μεταβλητή `lampaSaloniou` να μπορεί να δεχθεί ως τιμή αντικείμενα τύπου `Light` και έπειτα κατασκευάζει ένα τέτοιο αντικείμενο με τη δεσμευμένη λέξη **`new`** και κλήση του κατασκευαστή `Light(0)`. Σημειώνεται ότι το πρώτο `Light` είναι δήλωση της κλάσης, ενώ το `Light(0)` είναι κλήση μιας μεθόδου – του κατασκευαστή.
  - Τιμή της μεταβλητής `lampaSaloniou` είναι ένα προσδιοριστικό του συγκεκριμένου αντικειμένου. Τυπώνοντάς την βγαίνει κάτι σαν `Light@2a9835`. Βέβαια δεν έχει νόημα να τυπώσουμε την τιμή μιας μεταβλητής αντικειμένου - απλώς μέσω αυτής καλούνται οι μέθοδοι του αντικειμένου στο οποίο «στοχεύει» και ανακαλούνται οι δημόσιες ιδιότητές του.
  - Για να γίνει πλήρως κατανοητή η διάκριση ανάμεσα σε ένα αντικείμενο και σε μια μεταβλητή που δείχνει προς αυτό, δείτε το διπλανό κώδικα. Οι δύο μεταβλητές `lamp` και `sameLamp` αναφέρονται στο ίδιο αντικείμενο γι' αυτό λέγονται **ψευδώνυμα** (aliases).

```
Light lamp, sameLamp; //δηλώνει δύο μεταβλητές
lamp = new Light(0); //φτιάχνει ένα αντικείμενο
  //και θέτει τη μεταβλητή lamp να δείχνει σε αυτό
sameLamp = lamp; //δεν δημιουργείται άλλο αντικείμενο
```

Εικόνα 29 Ψευδώνυμα

- Οι **καταστροφείς** (destructors) αντικειμένων, εάν υπάρχουν, πρέπει να έχουν το όνομα μεθόδου `finalize`. Η βασικότερη λειτουργία τους είναι η απόδοση πίσω στο σύστημα των πόρων που απελευθερώνονται από την διαγραφή του αντικειμένου και η συντήρηση διαφόρων δομών δεδομένων ή μετρητών στιγμιότυπων, πχ μπορεί ο προγραμματιστής να διατηρεί το πλήθος των αντικειμένων μιας κλάσης. Δείτε το παράδειγμα 2.15 Κλασματικοί αριθμοί.

- **Διαχείριση μνήμης**
  - Στο διπλανό παράδειγμα, το αντικείμενο `Light` που δημιουργείται στη γραμμή Γ και προσπελαύνεται μέσω της μεταβλητής `h`, προκαλεί αδυναμία προσπέλασης του πρώτου αντικειμένου που είχε κατασκευαστεί στη γραμμή Α. Έτσι το αντικείμενο αυτό καταλαμβάνει πόρους (μνήμη), αλλά είναι αδύνατο να χρησιμοποιηθεί πλέον στο πρόγραμμα, αφού δεν υπάρχει κάποια μεταβλητή που να οδηγεί σε αυτό.
  - **Συλλογή απορριμμάτων** (garbage collection) είναι ο μηχανισμός αυτόματης αποδέσμευσης της μνήμης, όταν πια δεν χρησιμοποιείται το αντικείμενο που την καταλαμβάνει. Υλοποιείται από μια νηματική διεργασία (thread process) που τρέχει στο παρασκήνιο (background) με χαμηλή προτεραιότητα. Εάν θελήσουμε να επιβάλλουμε εκτέλεση της συλλογής απορριμμάτων σε συγκεκριμένη στιγμή, η εντολή είναι `System.gc()`.

```
Light h; Light k;
h = new Light(0); //A
k = new Light(0); //B
/* ... */
h = new Light(1); //Γ
k = null;
```

Εικόνα 30 Μη προσπελάσιμο αντικείμενο

  - Συλλογή απορριμμάτων έχει υλοποιηθεί από παλιά (πχ στις γλώσσες Lisp, Smalltalk), αλλά σε σύγχρονες γλώσσες τη διαχείριση μνήμης την αναλαμβάνει ο ίδιος ο προγραμματιστής, γιατί καταναλώνει πολύ χρόνο όταν γίνεται αυτόματα. Η Java υποστηρίζει συλλογή απορριμμάτων, αλλά αρκετοί μεταγλωττιστές έχουν προβλήματα στην υλοποίηση.
  - **Διαρροή μνήμης** (memory leak) έχουμε όταν δεν απελευθερώνονται τμήματα μνήμης που δεν είναι πλέον προσπελάσιμα από κανέναν. Μπορεί να οδηγήσει σε μείωση της διαθέσιμης μνήμης ή και εξάντλησή της.

2.6 Μέθοδοι ανάγνωσης και εγγραφής

- Οι **μέθοδοι ανάγνωσης** (reader ή getter ή accessor) είναι δημόσιες μέθοδοι που συνήθως το όνομά τους αρχίζει από `get`, δεν παίρνουν κάποια παράμετρο ως είσοδο και μόνον επιστρέφουν μια τιμή. Καθιστούν δυνατή την ανάγνωση της τιμής των μεταβλητών στιγμιότυπου (instance variables) που για λόγους απόκρυψης πληροφορίας (information hiding) είναι συνήθως ορισμένες ως ιδιωτικές μέσα στην κλάση.

- Οι **μέθοδοι εγγραφής** (writer ή setter ή mutator) είναι δημόσιες μέθοδοι που συνήθως το όνομά τους αρχίζει από `set`, παίρνουν παραμέτρους τις τιμές που επιθυμούμε να θέσουμε σε μεταβλητές στιγμιότυπου και δεν επιστρέφουν τίποτε[4]. Χρησιμοποιούνται για να αλλάξουν την τιμή των ιδιωτικών μεταβλητών μιας κλάσης, καθώς η απευθείας ανάθεση σε αυτές μιας τιμής δεν είναι δυνατή, όταν γίνεται έξω από την κλάση.
  - Στις μεθόδους εγγραφής συχνά συμπεριλαμβάνεται κώδικας που ελέγχει ότι οι τιμές που δίνονται στις ιδιότητες του αντικειμένου αντιστοιχούν σε μια «νόμιμη» ή συνεπή κατάσταση για το αντικείμενο. Για παράδειγμα στην Εικόνα 31, το κλάσμα αποκλείεται να αποκτήσει μηδενικό παρονομαστή. Απαγορεύοντας απευθείας προσπέλαση στις μεταβλητές στιγμιότυπου με τη δήλωσή τους ως **`private`**, επιβάλλουμε ότι ο μόνος τρόπος ορισμού / αλλαγής τους είναι μέσω των κατασκευαστών και των μεθόδων εγγραφής στις οποίες υπάρχουν έλεγχοι ότι τα στιγμιότυπα της κλάσης μας δεν τίθενται σε «παράνομη» κατάσταση. Αυτό μας γλιτώνει από πρόσθετους ελέγχους αργότερα στη ζωή του στιγμιότυπου, πχ δεν χρειάζεται να ελέγξουμε για διαίρεση δια 0, όταν υπολογίζουμε την τιμή του κλάσματος

```
public float computeRatio() { return (float) arithmitis / paronomastis; }
```

```
//μεταβλητές στιγμιοτύπου
private int arithmitis;     //αριθμητής
private int paronomastis;   //παρονομαστής

//τυπική μέθοδος ανάγνωσης
public int getArithmitis () {
  return this.arithmitis;
}

//τυπική μέθοδος εγγραφής
public void setFraction(int arithmitis,
                        int paronomastis) {
  this.arithmitis = arithmitis;
  this.paronomastis =
       ((paronomastis==0)?1: paronomastis);
}

//τυπική μέθοδος εκτύπωσης
public String toString() {
  return arithmitis + "/" + paronomastis;
}
```

Εικόνα 31 Δείγματα μεθόδων get , set, toString

- Τυπικά, σε μια κλάση που ορίζει έναν τύπο αντικειμένων, θα συναντήσουμε (α) μεταβλητές στιγμιότυπου που αναπαριστούν τις ιδιότητες, (β) κατασκευαστές, (γ) μεθόδους ανάγνωσης / εγγραφής, (δ) μεθόδους που υλοποιούν τη συμπεριφορά του αντικειμένου.
  - Επίσης, συνηθίζεται να υπάρχει μια μέθοδος `toString` (δείτε Εικόνα 31) που επιστρέφει κάποιας μορφής μετατροπή του αντικειμένου σε συμβολοσειρά, ώστε να μπορεί να εκτυπωθεί. Για παράδειγμα,

```
Klasma x = new Klasma(3, 2);
System.out.println(x.toString()); //τυπώνει 3.0/2.0, όπως αναμενόταν
System.out.println(x); //πάλι τυπώνει 3.0/2.0 αφού καλεί εμμέσως την toString()
```

- Παράδειγμα: ζητείται η κατασκευή ενός ηλεκτρονικού καταστήματος μέσω του οποίου οι πελάτες επιλέγουν προϊόντα από έναν κατάλογο, υποβάλουν τις παραγγελίες τους και τις εξοφλούν με πιστωτική κάρτα. Κατασκευάζουμε κλάσεις για τα προϊόντα, τους πελάτες και τις παραγγελίες. Η κλάση για το προϊόν μπορεί να ενσωματώνει χαρακτηριστικά όπως κωδικός, περιγραφή, τιμή, διαθέσιμη ποσότητα και μεθόδους δημιουργίας, ενημέρωσης, κλπ.

2.7 Ροή εκτέλεσης

- Κατά το αντικειμενοστρεφές μοντέλο, αντί κλήσεων υπορουτινών, η εκτέλεση του προγράμματος εκτυλίσσεται με ανταλλαγή μηνυμάτων (messages) μεταξύ αντικειμένων.
  - Η σύνταξη των κλήσεων είναι ως εξής: *παραλήπτης.μέθοδος()* ή *αντικείμενο.ρήμα()*, πχ `παράθυρο.κλείσε()` ή `λάμπαΣαλονιού.άναψε()`. Οι παρενθέσεις χρειάζονται για να διαφοροποιείται η ανταλλαγή μηνυμάτων από την αναφορά σε ιδιότητες αντικειμένων και περικλείουν τη λίστα των πραγματικών παραμέτρων (actual parameters).
  - Τα μηνύματα τα συλλαμβάνει το αντικείμενο - παραλήπτης και εκτελεί την ομώνυμη μέθοδο. Ο παραλήπτης του μηνύματος δεν καθορίζεται κατά τη μεταγλώττιση, γιατί κατά τη μεταγλώττιση ο παραλήπτης δεν υφίσταται ως αντικείμενο και επειδή η μέθοδος μπορεί να είναι υπερφορτωμένη, άρα ο μεταγλωττιστής δεν γνωρίζει ποια από τις εκδοχές της να αντιστοιχίσει στην κλήση. Μόνο κατά την εκτέλεση (runtime) συσχετίζεται η αποστολή του μηνύματος με τον παραλήπτη. Αυτό λέγεται **δυναμική** ή **καθυστερημένη σύνδεση** (dynamic / late binding).
  - Μπορούμε να αναφερόμαστε στις ιδιότητες του αντικειμένου με τη σύνταξη *object.property* πχ `παράθυροΠρόσοψης.ύψος`, ή `λάμπαΣαλονιού.χρώμα` υπό την προϋπόθεση ότι έχουν δημόσια εμβέλεια.

- Ο κύκλος λειτουργίας ενός αντικειμενοστρεφούς συστήματος περιλαμβάνει τα βήματα: (α) εκκίνηση εφαρμογής, (β) δημιουργία αντικειμένων, (γ) παραμονή σε αναμονή γεγονότος, (δ) αναγνώριση γεγονότος, (ε) αναζήτηση αντικειμένου παραλήπτη, (στ) εκτέλεση χειριστή γεγονότος, (ζ) περαιτέρω γεγονότα ή αναμονή.

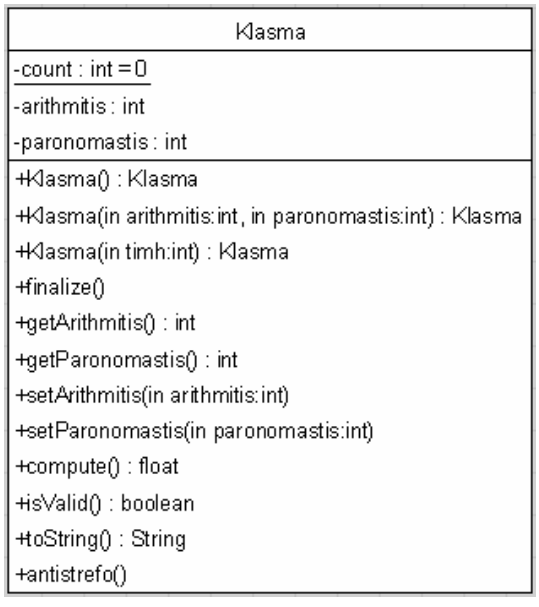
[4] Μερικές φορές είναι χρήσιμο να επιστρέφουν μια τιμή που υποδηλώνει ότι η εγγραφή της τιμής έγινε κανονικά στην ιδιότητα του αντικειμένου.

2.8 Υπερφόρτωση / Πολυμορφισμός

- Στην ίδια κλάση μπορούμε να ορίσουμε μεθόδους με το ίδιο όνομα, αρκεί να διαφέρουν στις παραμέτρους τους κατά το πλήθος, τον τύπο ή απλώς τη σειρά. Τότε το όνομα της μεθόδου είναι υπερφορτωμένο (overloaded method). Κατά την κλήση του ονόματος, το περιβάλλον εκτέλεσης θα εκτελέσει την μέθοδο της οποίας η λίστα των τυπικών παραμέτρων ταιριάζει με τις πραγματικές παραμέτρους. Συνήθως υπερφορτωμένες μέθοδοι είναι οι κατασκευαστές, δηλαδή οι μέθοδοι με τις οποίες αρχικοποιείται η κατάσταση των νεοδημιουργηθέντων αντικειμένων. Οι υπερφορτωμένες μέθοδοι συνήθως εκτελούν παρόμοιες λειτουργίες.
  - Προκύπτει ότι το όνομα μιας μεθόδου *δεν* είναι αρκετό για τον πλήρη προσδιορισμό της – χρειάζεται και η λίστα των παραμέτρων της με τους τύπους τους. Όλα αυτά ονομάζονται **υπογραφή** (signature) της υπερφορτωμένης μεθόδου.
  - Το σώμα της μεθόδου που έχει κληθεί καθορίζεται από το αντικείμενο – παραλήπτη κατά τη διάρκεια της εκτέλεσης κι όχι κατά τη μετάφραση (**καθυστερημένη σύνδεση**, late binding). Το χαρακτηριστικό αυτό βελτιώνει την προσαρμοστικότητα και αυξάνει την επαναχρησιμοποίηση, αλλά είναι πηγή σφαλμάτων και καθυστέρησης στην εκτέλεση.
- Η λειτουργία που επιτελεί ένας πολυμορφικός τελεστής εξαρτάται από το αντικείμενο στο οποίο εφαρμόζεται.
  - Σε αντίθεση με τη C++, η Java δεν επιτρέπει στον προγραμματιστή να υπερφορτώσει τελεστές, παρότι ορισμένοι τελεστές της είναι υπερφορτωμένοι από τη γλώσσα, όπως το `+`.

2.9 Κλήση μεθόδων

- Για την κλήση μιας μεθόδου, η συνήθης σύνταξη είναι
  αντικείμενο`.`*μέθοδος*`(` *λίσταΠραγματικώνΠαραμέτρων* `)`
  που εφαρμόζει τη συγκεκριμένη μέθοδο πάνω σε ένα αντικείμενο, πχ `textArea.setText("abc")`. Η σύνταξη αυτή ισχύει για τις **μεθόδους στιγμιότυπων**.
- Ωστόσο, σε μερικές περιπτώσεις συναντούμε τη σύνταξη
  Κλάση`.`*μέθοδος*`(` *λίσταΠραγματικώνΠαραμέτρων* `)`
  που εφαρμόζει τη μέθοδο στην κλάση, πχ `Integer.parseInt(sBuffer)`, ή `Math.sqrt(16)`. Σε αυτή την περίπτωση, κατά τον ορισμό της στην κλάση, η μέθοδος έχει οριστεί ως **`static`**. **Στατικές μέθοδοι** μπορούν να κληθούν ακόμη κι αν δεν έχουν δημιουργηθεί στιγμιότυπα της κλάσης.
  - Για να χρησιμοποιήσουμε μια μέθοδο πχ από μια βιβλιοθήκη, δεν αρκεί να γνωρίζουμε το όνομά της - πρέπει να γνωρίζουμε την «υπογραφή» της, δηλαδή την επικεφαλίδα του ορισμού της: αν είναι στατική ή όχι, τι τύπο δεδομένων επιστρέφει, το πλήθος και τον τύπο των παραμέτρων της. Πχ η μέθοδος στιγμιότυπου που μετατρέπει μια συμβολοσειρά σε κεφαλαία συντάσσεται `sName.toUpperCase()` και όχι `String.toUpperCase(sName)`.
  - Τα διαγράμματα κλάσεων (class diagrams) είναι μια εποπτική απεικόνιση των ιδιοτήτων και των μεθόδων μιας κλάσης. Για κάθε κλάση αναγράφονται διαδοχικά: (α) όνομα, (β) ιδιότητες και (γ) μέθοδοι. Η εμβέλεια του κάθε ονόματος υποδηλώνεται με + όταν είναι δημόσια (**`public`**) και με – όταν είναι ιδιωτική (**`private`**). Στην Εικόνα 32 φαίνεται το διάγραμμα της κλάσης `Klasma` - ο κώδικάς της βρίσκεται στην ενότητα 2.15.



Εικόνα 32 Διάγραμμα κλάσης

- Οι **παράμετροι** πάντοτε εγκλείονται σε παρενθέσεις και διαχωρίζονται με κόμμα.
  - Οι παράμετροι στην κλήση μιας μεθόδου ονομάζονται πραγματικές παράμετροι (actual parameters).
  - Οι παράμετροι στην επικεφαλίδα του ορισμού της μεθόδου ονομάζονται τυπικές παράμετροι (formal parameters).
  - Κατά την κλήση μιας μεθόδου αντιστοιχίζονται οι πραγματικές με τις τυπικές παραμέτρους, άρα το πλήθος τους πρέπει να είναι ίδιο – η Java δεν υποστηρίζει άμεσα προαιρετικές (optional) παραμέτρους, όπως πχ η Visual Basic. Μπορούμε ωστόσο να επιτύχουμε το ίδιο αποτέλεσμα υπερφορτώνοντας τη μέθοδο, δηλ. ορίζοντάς την πολλές φορές με το ίδιο όνομα με διαφορετικό αριθμό παραμέτρων.
- Τα ονόματα που δηλώνονται σε επίπεδο κλάσης χρησιμοποιώντας τη δεσμευμένη λέξη **`static`** δεν αναπαράγονται κάθε φορά που δημιουργείται ένα αντικείμενο της κλάσης και προσπελαύνονται με αναφορά στο όνομα της κλάσης – όχι το όνομα ενός συγκεκριμένου στιγμιότυπου της κλάσης. Συνήθως δηλώνονται **σταθερές κλάσης**, πχ

```
public class Euro {
   public static final CONVERSION_RATE = 340.75; …
```

  στην οποία αναφερόμαστε με `Euro.CONVERSION_RATE`. Τα υπάρχοντα πακέτα της Java ορίζουν αρκετές στατικές σταθερές, όπως `Math.PI`.
- Σε μερικές περιπτώσεις χρειάζεται να δημιουργήσουμε ένα και μοναδικό στιγμιότυπο μιας κλάσης. Μπορεί να αναρω-

τηθείτε αν αξίζει να κατασκευάσουμε ένα αντικείμενο και μεθόδους στιγμιότυπου γι αυτό, ή είναι καλύτερο να χρησιμοποιήσουμε στατικές μεταβλητές και μεθόδους. Η επικρατέστερη άποψη είναι να αποφεύγονται οι στατικές μέθοδοι, διότι:

- Παραβιάζουν το αντικειμενοστρεφές μοντέλο σχεδίασης.
- Στο μέλλον μπορεί να απαιτηθεί η δημιουργία και άλλων στιγμιότυπων.

- Το όνομα μιας μεταβλητής βασικού τύπου (πχ **int**, **double**) λειτουργεί σαν μια ετικέτα για μια περιοχή μνήμης όπου αποθηκεύεται η τιμή της μεταβλητής. Αντίθετα, το όνομα μιας μεταβλητής αντικειμένου (πχ `String`) είναι μια ετικέτα για μια περιοχή μνήμης που βρίσκεται η διεύθυνση του στιγμιότυπου.

2.10 Πέρασμα παραμέτρων

- Στη Java το πέρασμα παραμέτρων (parameter passing) γίνεται με τιμή (call by value). Ωστόσο, ο μηχανισμός λειτουργεί διαφορετικά, ανάλογα με το αν η παράμετρος είναι μια απλή μεταβλητή κάποιου από τους βασικούς τύπους δεδομένων, ή αν είναι αντικείμενο.
  - Για τους βασικούς τύπους δεδομένων (**int**, **float**, **double**, κλπ), οι πραγματικές και οι τυπικές παράμετροι είναι διαφορετικές μεταβλητές και δεν μοιράζονται τον ίδιο χώρο μνήμης. Κατά την κλήση της μεθόδου, εκχωρείται μνήμη για τις τυπικές παραμέτρους, η τιμή των πραγματικών ανατίθεται στην αντίστοιχη τυπική παράμετρο. Κατά συνέπεια, ό,τι αλλαγές κι αν γίνουν στις τιμές των τυπικών παραμέτρων στο εσωτερικό της μεθόδου δεν αντανακλώνται στις πραγματικές παραμέτρους. Κατά την επιστροφή από την μέθοδο, ο χώρος των τυπικών παραμέτρων επιστρέφεται στον διαθέσιμο χώρο του συστήματος.

```
public class ParameterPassing {
  public static void main(String args[]){
    double a = 9.5; //βασικός τύπος δεδομένων
    int b[] = {4, 2, 7}; //πίνακας
    System.out.println(a + " " + b[0]); //9.5 4
    //πραγματικές παράμετροι είναι τα a, b
    pass2(a, b);
    System.out.println(a + " " + b[0]); //9.5 -4
  }

  // τυπικές παράμετροι είναι τα x ,y
  private static void pass2(double x, int[] y) {
    x = -x;
    y[0] = -y[0];
    System.out.println(x + " " + y[0]); //-9.5 -4
  }
}
```

Εικόνα 33 Πέρασμα απλής μεταβλητής και αντικειμένου

  - Εξηγήστε γιατί η μέθοδος `swap` στην Εικόνα 34 αποτυγχάνει να αντιμεταθέσει τις τιμές των πραγματικών παραμέτρων.
  - Όταν η παράμετρος είναι αντικείμενο (πχ πίνακας, συμβολοσειρά, ή άλλο αντικείμενο κλάσης που έχει ορίσει ο χρήστης) μεταβιβάζεται η αναφορά του. Πάλι δημιουργείται αντίγραφο της αναφοράς, αλλά όχι αντίγραφο του αντικειμένου προς το οποίο δείχνει η αναφορά. Έτσι εξοικονομείται μνήμη και χρόνος, καθώς δεν γίνεται εκχώρηση / αποδέσμευση μνήμης, ούτε και αντιγραφή τιμών για ολόκληρο το αντικείμενο.

```
  ...
  int a = 1, c = 3;
  swap (a, c); //κλήση
  ... // παραμένουν a = 1 και c = 3

public static void swap(int x, int y) {
  int temp = x;
  x = y;
  y = temp;
}
```

Εικόνα 34 Πέρασμα με τιμή

- Για λόγους απλότητας και περιορισμού των παρενεργειών, η Java υποστηρίζει αποκλειστικά **πέρασμα παραμέτρων με τιμή**[5]. Άρα ο καλών μπορεί να τροφοδοτεί την καλούμενη μέθοδο με τιμές που η μέθοδος θα χρησιμοποιεί στο εσωτερικό της για να διαμορφώνει τη συμπεριφορά της. Πώς όμως επιστρέφει μια μέθοδος τα αποτελέσματα που υπολόγισε πίσω σε αυτόν που την κάλεσε, ειδικά όταν αυτά τα αποτελέσματα είναι βασικού τύπου δεδομένων;
  - Αν απαιτείται να επιστραφεί ένα μόνο αποτέλεσμα, τα πράγματα είναι εύκολα: δηλώνουμε στην επικεφαλίδα τον τύπο του αποτελέσματος που θέλουμε να επιστραφεί και επιστρέφουμε την τιμή του με μια εντολή `return`.
  - Αν χρειάζεται να επιστρέψουμε πολλά αποτελέσματα, και με δεδομένο ότι δεν υποστηρίζεται επιστροφή αποτελεσμάτων μέσω των παραμέτρων, το μόνο που μπορούμε να κάνουμε είναι να δηλώσουμε δημόσιες μεταβλητές, οπότε επιτρέπεται η πρόσβαση και η αλλαγή τους από την μέθοδο (και από οπουδήποτε αλλού). Αυτό είναι μια αναγκαία, αλλά άσχημη και επικίνδυνη πρακτική διότι δεν είναι εύκολα ορατές οι παρενέργειες που προκαλεί η μέθοδος στο περιβάλλον της. Μια εναλλακτική λύση είναι να «πακετάρουμε» τις τιμές που θέλουμε να επιστρέψουμε σε κάποιο αντικείμενο.

2.11 Σύνθεση

- Σύνθεση (composition) είναι ένας μηχανισμός μοντελοποίησης όπου ορίζουμε μια κλάση με συστατικά που είναι ορισμένα από άλλες κλάσεις. Για παράδειγμα, ένας Φοιτητής ορίζεται να έχει όνομα που είναι ένα αντικείμενο `String` και ημερομηνίαΕγγραφής που είναι μια `Date`, δηλαδή αντικείμενο μιας άλλης κλάσης (από τη βιβλιοθήκη java.util.Date).

[5] Σε άλλες γλώσσες προγραμματισμού, ο προγραμματιστής δηλώνει τον τρόπο περάσματος παραμέτρων στη λίστα των παραμέτρων που βρίσκεται στην επικεφαλίδα της υπορουτίνας, πχ στη Visual Basic δηλώνει `ByVal` ή `ByRef`.

- Μέχρι τώρα έχουμε δει αναδρομικές μεθόδους, δηλαδή μεθόδους που καλούν τον εαυτό τους υπό προϋποθέσεις (αν δεν υπάρχει κάποια προϋπόθεση, τότε η αναδρομικές κλήσεις θα συνεχιζόταν μέχρι να εξαντληθούν οι διαθέσιμη μνήμη του υπολογιστή). Ωστόσο, μπορούμε να ορίσουμε και αναδρομικές κλάσεις. Δείτε ένα παράδειγμα στην Εικόνα 35, όπου για κάθε άτομο ορίζουμε διάφορες ιδιότητες, μεταξύ των οποίων και η μητέρα και ο πατέρας του, που είναι αντικείμενα άτομο.

```
class Person {
  String name;
  Person mother;
  Person father;
  …
}
```

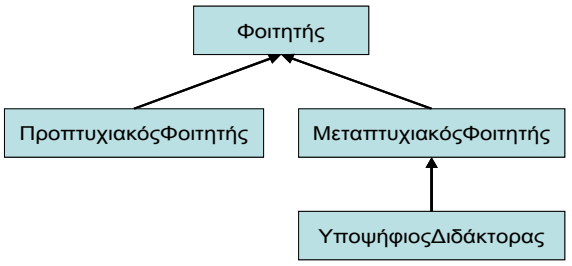
Εικόνα 35 Αναδρομικές κλάσεις

  - Η δυνατότητα αναδρομικής σύνθεσης στις κλάσεις είναι πολύ χρήσιμη για να ορίζουμε «αλυσίδες» κόμβων που περιέχουν πληροφορία. Στην απλούστερη περίπτωση, κάθε κόμβος της αλυσίδας περιέχει κάποια πληροφορία και ένα σύνδεσμο προς τον επόμενο κόμβο στην αλυσίδα. Περισσότερα γι αυτή την τεχνική στο μάθημα επιλογής «Αλγόριθμοι & Δομές Δεδομένων».

2.12 Κληρονομικότητα

- Κληρονομικότητα (inheritance) είναι ένας μηχανισμός επαναχρησιμοποίησης που επιτρέπει σε νεότερες κλάσεις να επεκτείνουν υπάρχουσες κλάσεις, όσον αφορά τα δεδομένα τους, όσο και τη συμπεριφορά τους. Η κληρονομικότητα μπορεί να συνεχίζεται σε αρκετές γενιές. Οι κλάσεις – απόγονοι (υποκλάσεις ή παράγωγες κλάσεις, subclasses ή derived classes) επιτρέπεται όχι μόνο να επεκτείνουν, αλλά και να τροποποιούν την υπάρχουσα συμπεριφορά των προγόνων τους (υπέρ-κλάσεις, super classes).
  - Παράδειγμα: Τόσο οι προπτυχιακοί όσο και οι μεταπτυχιακοί φοιτητές μοιράζονται ορισμένες κοινές ιδιότητες, πχ ημερομηνία εγγραφής και λειτουργίες, πχ δηλώνουν μαθήματα που πρέπει να περάσουν. Οι μεταπτυχιακοί ωστόσο έχουν την επιπρόσθετη βαθμό πτυχίου. Ορίζουμε μια κλάση «Φοιτητές» που έχει τις κοινές ιδιότητες, όπως ονοματεπώνυμο, ημερομηνία εγγραφής, και μεθόδους δήλωσηΜαθήματος, κλπ. Ορίζουμε τις κλάσεις «ΜεταπτυχιακόςΦοιτητής» και «ΠροπτυχιακόςΦοιτητής» να κληρονομούν την κλάση «Φοιτητής» και να προσθέτουν τις ιδιότητες και τις μεθόδους που τους προσιδιάζουν και δεν είναι κοινές. Επίσης, μπορούν να επαναορίζουν την μέθοδο δήλωσηΜαθήματος, υπερκαλύπτοντας τον ορισμό που έχει στην υπέρ-κλάση, καθώς οι μεταπτυχιακοί έχουν διαφορετική διαδικασία επιλογής μαθημάτων από τους προπτυχιακούς.

- **Απλή κληρονομικότητα** έχουμε όταν η υποκλάση βασίζεται μόνον σε μια άλλη υπέρ-κλάση. Τέτοιου τύπου κληρονομικότητα υποστηρίζει η Java μέσω της δεσμευμένης λέξης **extends** που συντάσσεται στην επικεφαλίδα της υποκλάσης και δηλώνει το όνομα της προγονικής κλάσης. Η απλή κληρονομικότητα απεικονίζεται γραφικά ως ένα ανεστραμμένο δέντρο που στο πάνω μέρος του βρίσκεται η υπέρ-κλάση η οποία συνδέεται με γραμμές με τις υποκλάσεις της (δείτε Εικόνα 36).



Εικόνα 36 Απεικόνιση κληρονομικότητας

  - Μια υποκλάση μπορεί να (επανα)ορίζει με διαφορετικό τύπο μια μεταβλητή στιγμιότυπου που είναι ήδη ορισμένη στην υπέρ-κλάση της. Επίσης μια υποκλάση μπορεί να (επανα)ορίζει μια μέθοδο που είναι ήδη ορισμένη με την ίδια ακριβώς υπογραφή στην υπέρ-κλάση της.

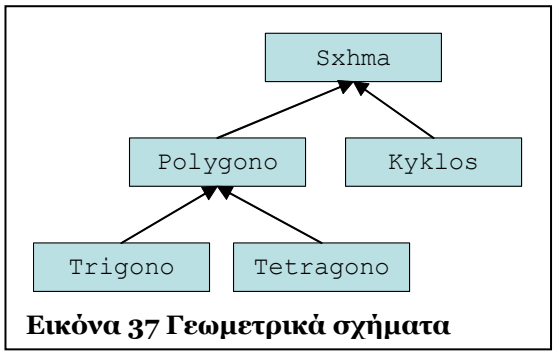
- Υποστήριξη κληρονομικότητας από τη Java
  - Για να αναφερθούμε στην προγονική κλάση όταν βρισκόμαστε στο σώμα της υποκλάσης, χρησιμοποιούμε τη λέξη `super`[6].
  - Έχουμε δει τα προσδιοριστικά εμβέλειας **public** και **private** που επιτρέπουν / απαγορεύουν πρόσβαση έξω από την κλάση αντίστοιχα. Ενδιάμεσα, ένα όνομα (μεταβλητή ή μέθοδος) δηλωμένο ως **protected**, είναι ορατό στις υποκλάσεις που κληρονομούν την τρέχουσα κλάση και στις κλάσεις του ίδιου πακέτου στο οποίο είναι ορισμένο.
  - Οι κατασκευαστές της υπέρ-κλάσης δεν κληρονομούνται από τις υποκλάσεις της.
  - Στη Java όλες οι κλάσεις κληρονομούν άμεσα ή έμμεσα από την υπέρ-κλάση `Object`. Έτσι όταν θέλουμε να αναφερθούμε σε αντικείμενο του οποίου την κλάση δεν γνωρίζουμε δηλώνουμε την αναφορά του ως `Object`. Δείτε την ενότητα 2.16 Πολύφωτο για ένα παράδειγμα. Για να προσδιορίσουμε την κλάση ενός αντικείμενου που έχει χαρακτηριστεί ως `Object`, χρησιμοποιούμε τον τελεστή **instanceof**.
  - Με το προσδιοριστικό **final** στην επικεφαλίδα μιας κλάσης απαγορεύουμε τη δημιουργία υποκλάσεων που να κληρονομούν από αυτή την κλάση. Έτσι δεν υπάρχει περίπτωση κάποιος να ορίσει κλάση που να κληρονομεί την **final** και να αλλάζει την υλοποίηση κάποιων μεθόδων της ή να αποκτήσει προσπέλαση στα **protected** δεδομένα / μεθόδους της (που μπορεί να τα έχει κληρονομήσει από κάποια υπέρ-κλάση της). Ως **final** μπορεί να προσδιοριστεί και μια μέθοδος οπότε δεν μπορεί να οριστεί σε κάποια υποκλάση.
  - Αν μια κλάση δηλωθεί ως **public**, τότε μπορεί να χρησιμοποιηθεί από οποιαδήποτε άλλη κλάση ακόμη και αν δεν ανήκουν στο ίδιο πακέτο (`package`). Υπενθυμίζεται ο κανόνας ότι μια **public** κλάση πρέπει να αποθηκεύεται σε ομώνυμο αρχείο με την κατάληξη `.java`[7]. Αν μια κλάση δεν είναι **public**, ονομάζεται «φιλική» (friendly) και τότε μπορεί να χρησιμοποιηθεί μόνον από κάποια κλάση του ίδιου πακέτου.

---

[6] Παράβαλε με τη δεσμευμένη λέξη **this** με την οποία αναφερόμαστε στο τρέχον αντικείμενο από το εσωτερικό της κλάσης του.

[7] Στο ίδιο αρχείο μπορούν να συνυπάρχουν και άλλες κλάσεις αλλά αυτές δεν μπορούν να είναι **public**.

- Με το προσδιοριστικό **abstract** στην επικεφαλίδα μιας κλάσης απαγορεύουμε τη δημιουργία στιγμιότυπων της κλάσης. Και τότε τι την θέλουμε την κλάση, αφού δεν μπορούμε να κατασκευάσουμε αντικείμενά της; Οι **abstract κλάσεις** περιέχουν κάποιες κανονικές μεθόδους που περιλαμβάνουν κώδικα υλοποίησης και κάποιες **abstract** μεθόδους, που έχουν επικεφαλίδα, αλλά δεν περιέχουν κώδικα υλοποίησης. Υποκλάσεις που επεκτείνουν την **abstract** κλάση μπορούν να συμπληρώσουν την υλοποίηση των **abstract** μεθόδων της και βέβαια είναι δυνατόν να κατασκευαστούν στιγμιότυπα αυτών των υποκλάσεων. Γενικά χρησιμοποιούμε μια abstract κλάση για να ομαδοποιήσουμε κάποιες κοινές μεθόδους των υποκλάσεών της, οπότε αποφεύγουμε να επαναλαμβάνουμε τον κώδικα των κοινών μεθόδων σε καθεμιά υποκλάση.

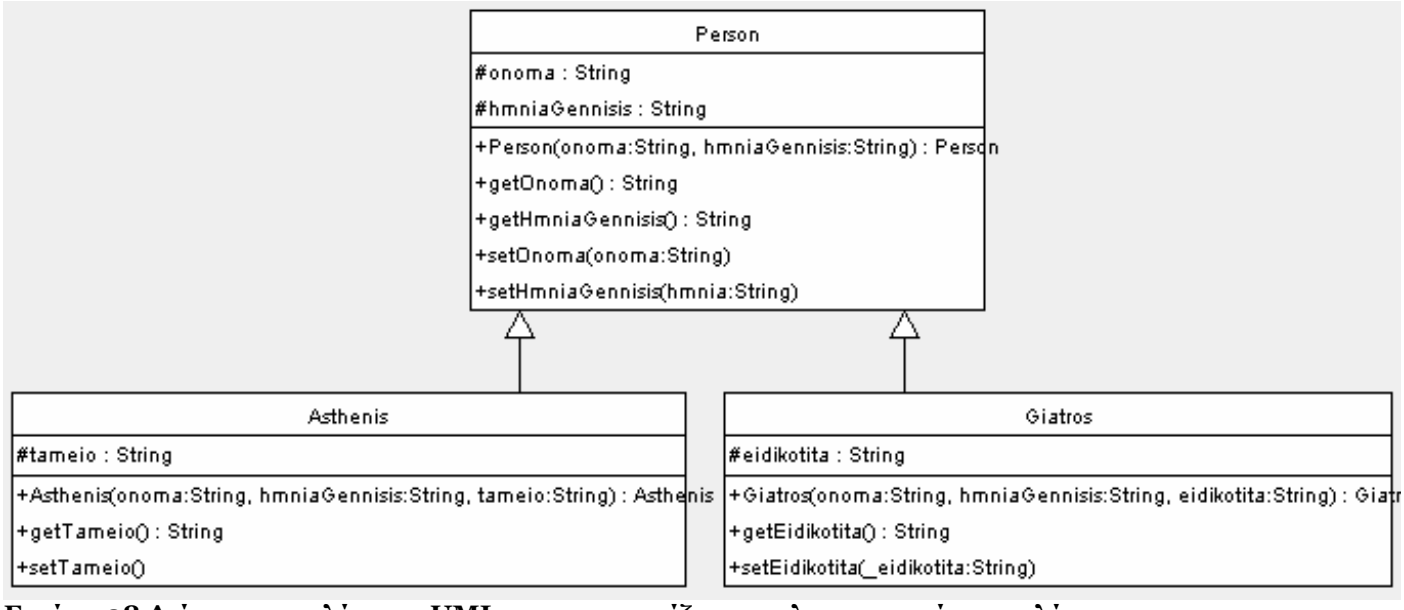


Εικόνα 37 Γεωμετρικά σχήματα

  - Κουίζ: γιατί είναι αντίφαση μια **abstract** μέθοδος να είναι και **final**; Γιατί μια κλάση δεν μπορεί να είναι **abstract** και **static**;
  - Παράδειγμα: Ορίζουμε μια **abstract** υπέρ-κλάση `Sxhma`, αφού δεν έχει νόημα να κατασκευάσουμε στιγμιότυπά της, επειδή είναι πολύ γενική και αφηρημένη. Η `Sxhma` έχει την ιδιότητα `Xroma`, άρα οποιαδήποτε υποκλάση της, πχ `Trigono`, `Kyklos`, θα την κληρονομεί και έτσι γλιτώνουμε από το να την ορίζουμε και να την ελέγχουμε σε κάθε υποκλάση. Στη `Sxhma` ορίζουμε και την **abstract** μέθοδο `ypologiseEmvadon()`, που δεν έχει σώμα – μόνον επικεφαλίδα. Αυτό μας βολεύει γιατί δεν υπάρχει τρόπος (μαθηματικός τύπος) που να υπολογίζει το εμβαδόν *κάθε* σχήματος, όμως έχουμε καταφέρει να δεσμεύσουμε όλες τις υποκλάσεις να την υλοποιήσουν με ακριβώς αυτό το όνομα και υπογραφή (εννοείται ότι το σώμα θα διαφέρει για κάθε υποκλάση).

- Η κληρονομικότητα είναι ένας **μηχανισμός μοντελοποίησης**, δηλαδή επιτρέπει να κατασκευάσουμε προγράμματα που θα απεικονίζουν κατά το δυνατόν πιστά τις οντότητες του πραγματικού κόσμου των οποίων τη λειτουργία προσομοιώνουν. Όταν προγραμματίζουμε με αντικειμενοστρεφή τρόπο και θέλουμε να επεκτείνουμε τις ιδιότητες ενός αντικειμένου μπορούμε είτε να προσθέσουμε νέες μεταβλητές στιγμιότυπου, ή να επεκτείνουμε την κλάση του αντικειμένου ορίζοντας μια υποκλάση της με τις πρόσθετες ιδιότητες. Για να επιλέξουμε μπορούμε να σκεφτούμε ως εξής:
  - Όταν η σχέση είναι κλάσης – υποκλάσης, τη διατυπώνεις με τη φράση " ... *είναι-ένα* ... " (**is-a** relation). Για παράδειγμα, ο άνθρωπος είναι ένα θηλαστικό, ή το αυτοκίνητο είναι ένα τετράτροχο που με τη σειρά του είναι ένα όχημα.
  - Διαφορετικά, η φράση " ... *έχει-ένα* ... " (**has-a** ή part-whole relation) υποδηλώνει ότι θα πρέπει να προσθέσουμε μεταβλητές στιγμιότυπου (δες ενότητα 2.11). Για παράδειγμα, ο άνθρωπος έχει χέρια και πόδια, ή το αυτοκίνητο έχει κυβισμό, χρώμα, κλπ.

- Ας δούμε ένα απλό παράδειγμα. Θέλουμε να διαχειριστούμε διάφορες λειτουργίες ενός νοσοκομείου και αρκετές από αυτές αφορούν γιατρούς και ασθενείς. Θα μπορούσαμε να υλοποιήσουμε δύο ανεξάρτητες κλάσεις με τις απαιτούμενες ιδιότητες και μεθόδους, ωστόσο παρατηρούμε ότι γιατροί και ασθενείς μοιράζονται κοινά χαρακτηριστικά που τα υλοποιούμε δύο φορές. Για να το αποφύγουμε αυτό, φτιάχνουμε μια κλάση που υλοποιεί ό,τι αφορά ένα άτομο, είτε είναι γιατρός, είτε ασθενής. Έπειτα φτιάχνουμε ξεχωριστές υποκλάσεις για γιατρούς και ασθενείς που κληρονομούν τα κοινά χαρακτηριστικά των ατόμων και υλοποιούν τα ιδιαίτερα χαρακτηριστικά που τους αφορούν. Στην ενότητα 2.16



Εικόνα 38 Διάγραμμα κλάσεων UML που απεικονίζει την κληρονομικότητα κλάσεων

Γιατροί και ασθενείς (κληρονομικότητα) υπάρχει μια υλοποίηση του παραδείγματος.

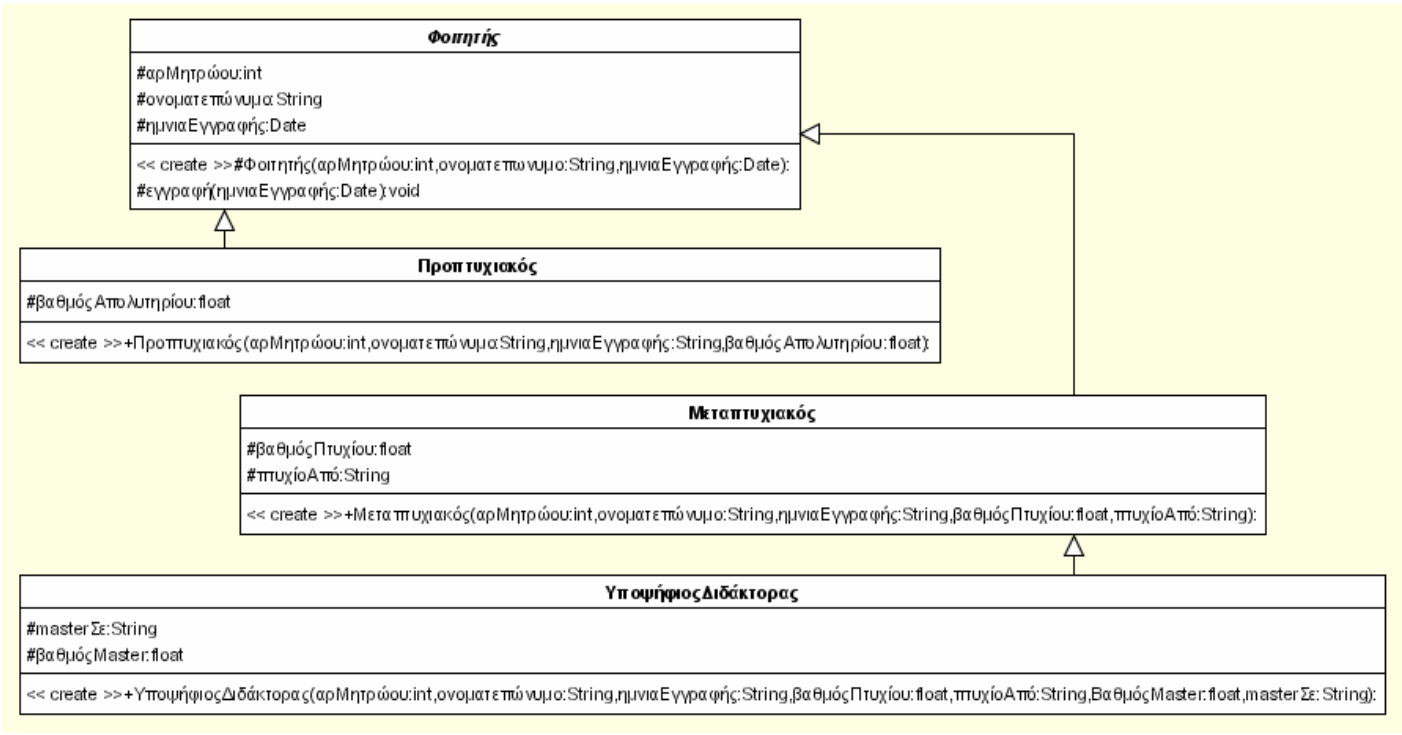
  - Στην Εικόνα 38 φαίνονται οι τρεις κλάσεις με τις ιδιότητες και τις μεθόδους τους και η σχέση κληρονομικότητας απεικονίζεται με τα βέλη. Το διάγραμμα απεικονίζει τις κλάσεις ως κουτιά τα οποία συνδέονται με βέλη που δείχνουν τη σχέση κληρονομικότητας. Η εμβέλεια υποδηλώνεται με # όταν είναι **protected**, με + όταν είναι **public** και με

– όταν είναι **`private`**.

- Όταν ένα πρόγραμμα αποτελείται από πολλές κλάσεις που θέλουμε να τις ομαδοποιήσουμε για να μπορούν να ανταλλάσσουν μηνύματα, τις εντάσσουμε στο ίδιο **πακέτο** με την εντολή `package όνομαΠακέτου`. Τα μεταγλωττισμένα αρχεία `.class` τοποθετούνται σε ένα φάκελο στο σημείο που δείχνει η μεταβλητή συστήματος `CLASSPATH`.
  - Για να χρησιμοποιήσουμε κώδικα του πακέτου Χ από κάποια κλάση άλλου πακέτου Υ, δηλώνουμε στην αρχή της κλάσης **`import`** `X.ΌνομαΚλάσης` ή γενικά **`import`** `X.*;`

- Όταν κατασκευάζουμε ένα μεγάλο πρόγραμμα, είναι πιθανόν να πρέπει να μοντελοποιήσουμε αρκετές οντότητες που θα συνδέονται μεταξύ τους με σχέσεις κληρονομικότητας. Γι αυτή την περίπτωση υπάρχουν εργαλεία που επιτρέπουν στον προγραμματιστή να σχεδιάζει τις κλάσεις και τη σχέση τους σε μορφή διαγράμματος κλάσης **UML** (Universal Modeling Language), όπως φαίνεται και στο σχήμα και έπειτα παράγουν αυτόματα τον σκελετό των κλάσεων σε κάποια γλώσσα προγραμματισμού. Ένα τέτοιο εργαλείο που παράγει κώδικα Java είναι το Poseidon for UML – μια ελεύθερη έκδοσή του μπορείτε να βρείτε στο http://www.gentleware.com (community edition for Windows, έκδοση 3.0.1, ~24Mb). Ο προγραμματιστής έπειτα αρκεί να «γεμίσει» τα σώματα των μεθόδων με κώδικα. Τόσο το σχήμα στην Εικόνα 38, όσο και ο σκελετός του προγράμματος της ενότητας 2.16 Γιατροί και ασθενείς (κληρονομικότητα) προήλθαν από το εργαλείο Poseidon. Ένα αντίστοιχο εργαλείο είναι και το Rational Rose της IBM.

- Η κληρονομικότητα πέρα από την αναλογία της με αντικείμενα και καταστάσεις του πραγματικού κόσμου, μας χρησιμεύει για:
  - να αποφεύγουμε τη συγγραφή του ίδιου κώδικα, αφού ο κοινό κώδικας συγκεντρώνεται στην υπέρ-κλάση και οι υποκλάσεις τον κληρονομούν από εκείνη, και
  - να ορίζουμε ένα κοινό «πρωτόκολλο» σε ένα σύνολο κλάσεων, όταν θέλουμε όλες οι υποκλάσεις να «απαντούν» με ένα κοινό τρόπο για (κάποιες από) τις λειτουργίες τους.



Εικόνα 39 Διάγραμμα κλάσεων UML για φοιτητές

  - Στην Εικόνα 39 βλέπουμε άλλο ένα παράδειγμα κληρονομικότητας. Η κλάση `Φοιτητής` είναι δηλωμένη **`abstract`**, άρα δεν επιτρέπεται να κατασκευαστούν στιγμιότυπά της. Η χρησιμότητά της είναι ότι κληρονομείται από τις παράγωγες κλάσεις `Προπτυχιακός` και `Μεταπτυχιακός`. Επιπλέον, ένας `ΥποψήφιοςΔιδάκτορας` είναι-ένας `Μεταπτυχιακός`. Στο διάγραμμα πρόκειται να προστεθούν όλες οι μέθοδοι στις οποίες «απαντούν» οι κλάσεις.

- **Πολλαπλή κληρονομικότητα** έχουμε όταν μια κλάση κληρονομεί ή επεκτείνει πολλές. Η πολλαπλή κληρονομικότητα παρέχει μεγαλύτερες δυνατότητες μοντελοποίησης, αλλά είναι πιο δύσχρηστη και «επικίνδυνη». Η γλώσσα C++ υποστηρίζει άμεσα πολλαπλή κληρονομικότητα. Στη Java η πολλαπλή κληρονομικότητα υλοποιείται έμμεσα ορίζοντας ότι μια κλάση υλοποιεί μία ή περισσότερες διεπαφές (**interface**[8]).

[8] Δεν έχει καμία σχέση με τη γραφική διεπαφή χρήστη (graphical user interface).

- Μια διεπαφή στη Java είναι ο σκελετός μιας κλάσης που εισάγεται με τη δεσμευμένη λέξη **interface**, αντί της **class**, και περιλαμβάνει δηλώσεις σταθερών ή επικεφαλίδες μεθόδων χωρίς το σώμα τους, δηλαδή την υλοποίησή τους. Μια κλάση που υλοποιεί τη διεπαφή έχει στην επικεφαλίδα της τη δεσμευμένη λέξη **implements** και περιλαμβάνει τις επικεφαλίδες των μεθόδων μαζί με τον κώδικα που υλοποιεί τη λειτουργικότητά τους. Ουσιαστικά μια διεπαφή «δεσμεύει» ονόματα για τις μεθόδους και τους τύπους των παραμέτρων τους, έτσι ώστε κάθε κλάση που επεκτείνει τη διεπαφή να υποχρεώνεται να ακολουθήσει τις προδιαγραφές αυτές.

```
[προσδιοριστικάΚλάσης] class όνομαΚλάσης
                       [extends όνομαΥπερΚλάσης]
                       [implements όνομαΔιεπαφής+ ] {
                           σώμαΚλάσης
                       }
```

Εικόνα 40 Η σύνταξη μιας επικεφαλίδας κλάσης

- Ενώ μια κλάση μπορεί να επεκτείνει (**extends**) μόνον μία άλλη υπέρ-κλάση (απλή κληρονομικότητα), είναι δυνατόν να υλοποιεί (**implements**) περισσότερες από μία διεπαφές (πολλαπλή κληρονομικότητα). Το συντακτικό για τον ορισμό μιας κλάσης της Java φαίνεται στην Εικόνα 40. Τα προσδιοριστικά καθορίζουν αν η κλάση είναι **public**, **final** ή **abstract**. Προσέξτε το σταυρό στην ενότητα **implements** που δηλώνει ένα ή περισσότερα ονόματα διεπαφής.
- Μια συνηθισμένη χρήση των διεπαφών είναι να ορίζουν σταθερές, όπως φαίνεται στην Εικόνα 41. Μια άλλη κλάση για να χρησιμοποιήσει τις σταθερές απλώς χρειάζεται να δηλώσει στην επικεφαλίδα ότι υλοποιεί (**implements**) τις διεπαφές.

```
public interface MyConstants {
        // Μεγέθη για πουκάμισα
  byte SIZE_SMALL = 0;    // όλες οι δηλώσεις σε
  byte SIZE_MEDIUM = 1;   // interface είναι εξ
  byte SIZE_LARGE = 2;    // ορισμού
  byte SIZE_X_LARGE = 3; // public και final
  byte SIZE_XX_LARGE = 4;
        // Τύποι πουκάμισων
  byte MONOXROMO = 0; // σταθερά
  byte EMPRIME = 1;
  byte RIGE = 2;
}
```

```
public class Poykamisa implements MyConstants {
  byte megethos = SIZE_MEDIUM; // ιδιότητες
  byte typos = MONOXROMO;

  public Poykamisa(byte megethos, byte typos) {
    if ((megethos >= SIZE_SMALL) &&
        (megethos <= SIZE_XX_LARGE))
      this.megethos = megethos;
  }
}
```

Εικόνα 41 Ορισμός και χρήση διεπαφής

2.13 Ασκήσεις

Α25. Φτιάξτε κλάση που διαχειρίζεται σημεία στο επίπεδο , δηλαδή σημεία που έχουν μια τετμημένη `x` και μια τεταγμένη `y`. Φτιάξτε μέθοδο `distance` που να υπολογίζει την απόσταση δύο σημείων. Φτιάξτε άλλη μέθοδο `a.isFurther(b)` που βρίσκει αν το `a` απέχει από την αρχή των αξόνων περισσότερο από το `b` (Υπόδειξη: μπορεί να χρησιμοποιεί την προηγούμενη). Τέλος, μια μέθοδο που θα ελέγχει αν τρία σημεία είναι στην ίδια ευθεία.

- Φτιάξτε μια άλλη κλάση που ορίζει διανύσματα ως διατεταγμένα ζεύγη σημείων. Μια μέθοδος των διανυσμάτων να υπολογίζει το μήκος τους. Κατασκευάστε μερικά διανύσματα και βρείτε το πιο μεγάλο.
- Επεκτείνετε την κλάση, έτσι ώστε τα σημεία να είναι στον τρισδιάστατο χώρο.

Α26. Φτιάξτε κλάση που περιέχει το ονοματεπώνυμο ενός φοιτητή, το εξάμηνο στο οποίο βρίσκεται και τους βαθμούς πέντε μαθημάτων του. Όλες οι ιδιότητες της κλάσης θα είναι ιδιωτικές. Φτιάξτε μεθόδους ανάγνωσης και εγγραφής για όλες τις ιδιότητες. Ειδικά για τους βαθμούς, θα υλοποιήσετε μέθοδο **float** `getBathmos(`**int** `arMath)` και **void** `setBathmos(`**int** `arMath, `**float** `fBathmos)`. Τέλος, κατασκευάστε μερικούς φοιτητές και χρησιμοποιήστε όλες τις μεθόδους τουλάχιστον μία φορά.

- Με βάση τους φοιτητές που έχετε κατασκευάσει υπολογίστε το μέσο όρο τους σε όλα τα μαθήματα στα οποία έχουν πάρει βαθμό. Υπόδειξη: εάν ένας μαθητής δεν έχει πάρει βαθμό σε κάποιο από τα πέντε μαθήματα, βάλτε του μια «αδύνατη» τιμή, πχ (-1).
- Υλοποιήστε κλάση για τους μεταπτυχιακούς φοιτητές που κληρονομεί από την αρχική κλάση.

Α27. Φτιάξτε κλάσεις για δισδιάστατα γεωμετρικά αντικείμενα με μεθόδους που υπολογίζουν την περίμετρο και το εμβαδόν τους. Για παράδειγμα, (α) τρίγωνο με πλευρές α, β, γ, (β) κύκλο ακτίνας ρ, (γ) παραλληλόγραμμο πλευρών α, β, (δ) τραπέζιο με παράλληλες πλευρές α, β και ύψος υ. Έπειτα φτιάξτε πρόγραμμα που θα δημιουργεί αντικείμενα συγκεκριμένου τύπου και ιδιοτήτων και θα τυπώνει πληροφορίες γι αυτά.

- Επεκτείνετε το πρόγραμμα προσθέτοντας κι άλλες κλάσεις για τα γεωμετρικά αντικείμενα (ε) έλλειψη και (στ) τομέα κύκλου.
- Τροποποιήστε τον κατασκευαστή του τριγώνου έτσι ώστε να ελέγχει ότι κάθε πλευρά του είναι μικρότερη από το άθροισμα των άλλων δύο και μεγαλύτερη από τη διαφορά τους (αλλιώς τι σόι τρίγωνο είναι αυτό). Επεκτείνετε την κλάση του τριγώνου με δυαδικές μεθόδους `einaiOrthoGonio`, `einaiSkalino`, `einaiIsoskeles`, `einaiIsoplevro`.

Α28. Φτιάξτε κλάσεις για στερεά αντικείμενα (α) παραλληλεπίπεδο (με πρόσθετο κατασκευαστή για κύβο), (β) σφαίρα, (γ) κώνος, (δ) κύλινδρος. Για κάθε στερεό θα συμπεριλάβετε στην κλάση του μεθόδους που υπολογίζουν τον όγκο και την επιφάνειά του. Μια άλλη κλάση θα δίνει επιλογή στο χρήστη για το αντικείμενο που θέλει να κατασκευάσει και στο τέλος

θα τυπώνει το συνολικό όγκο των αντικειμένων που κατασκευάστηκαν.

- Επεκτείνετε το πρόγραμμα προσθέτοντας κλάσεις για το στερεό αντικείμενο (ε) πυραμίδα. Επίσης, το πρόγραμμα θα πρέπει να σταματάει όταν η συνολική επιφάνεια των στερεών που κατασκευάστηκαν υπερβεί τις 10 τετραγωνικές μονάδες.

Α29. Φτιάξτε κλάση για το αντικείμενο ώρα χρησιμοποιώντας ιδιωτικά δεδομένα. Υλοποιήστε τις συνηθισμένες μεθόδους (κατασκευαστές, getter, setter). Επίσης, φτιάξτε μέθοδο `toString` που θα επιστρέφει την ώρα ως 23:26:43 ή 11:26:43μμ ανάλογα με διακόπτη (παράμετρο). Επίσης, υλοποιήστε μέθοδο που βρίσκει τη διαφορά ανάμεσα σε δύο ώρες.

- Επεκτείνετε το πρόγραμμα έτσι ώστε να διαβάζει τις ώρες προσέλευσης και αποχώρησης ενός εργαζομένου και όταν τερματίζει να τυπώνει τη συνολική διάρκεια που εργάστηκε, και πόσες φορές προσήλθε στην εργασία του. [Υπάρχει πιθανότητα ο εργαζόμενος να ξεκινήσει να δουλεύει στις 23:00:00 και να βγει στις 07:00:00.]

Α30. Φτιάξτε κλάση για ημερομηνίες χρησιμοποιώντας ιδιωτικά δεδομένα. Οι ιδιότητες θα πρέπει να ελέγχονται για εγκυρότητα. Υλοποιήστε μεθόδους για εκτύπωση και μέθοδο που επιστρέφει τη διαφορά σε ημέρες ανάμεσα σε δύο ημερομηνίες (θα πρέπει, βέβαια να λαβαίνετε υπόψη πόσες ημέρες έχει ο κάθε μήνας. Με δεδομένο ότι πχ η 3η Απριλίου 2003 είναι Πέμπτη, θα φτιάξετε μέθοδο που θα υπολογίζει την ημέρα της εβδομάδας για την ημερομηνία. Θα χρειαστεί να φτιάξετε ιδιωτικές μεθόδους που θα εντοπίζουν δίσεκτα έτη και πόσες ημέρες μεσολαβούν από την αρχή του έτους ως μια ημερομηνία και από μια ημερομηνία ως το τέλος του έτους.

- Μετρήστε πώς κατανέμονται οι Πρωτοχρονιές τον εικοστό πρώτο αιώνα (πόσες είναι Δευτέρες, πόσες Τρίτες, κοκ). Αν κάποιος γεννήθηκε την 21η Οκτωβρίου 1999, ποια ημερομηνία θα έχει ζήσει 10.000 ημέρες;

Α31. Ένα κατάστημα εμπορεύεται πουκάμισα διαφόρων τύπων, συγκεκριμένων χρωμάτων, μεγεθών και τιμών αγοράς και πώλησης. Φτιάξτε κλάση που να αποθηκεύει τα χαρακτηριστικά και να υλοποιεί τις βασικές λειτουργίες πάνω στα πουκάμισα. Έπειτα φτιάξτε πρόγραμμα που θα παραλαμβάνει πουκάμισα στο κατάστημα (ισοδυναμεί με δημιουργία αντικειμένου) και θα πουλάει μερικά από αυτά (ισοδυναμεί με καταστροφή αντικειμένου). Με την παραλαβή θα αυξάνεται το χρέος προς τον προμηθευτή και με την πώληση θα αυξάνεται ο τζίρος.

- Τροποποιήστε το παραπάνω πρόγραμμα με δεδομένο ότι το κατάστημα πουλάει τα πουκάμισα με ποσοστό κέρδους 17%. Επίσης, υποθέστε ότι ο επιχειρηματίας έχει €1.000 διαθέσιμο κεφάλαιο για να αγοράζει στοκ. Κάντε το πρόγραμμα να τερματίζει όταν το κατάστημα πουλήσει τόσα πουκάμισα ώστε το κέρδος να υπερβεί τα €200.

Α32. Όπως θα διαπιστώσατε στην άσκηση Α17, αν προσπαθήσετε να υπολογίσετε το ν παραγοντικό χρησιμοποιώντας ακεραίους **`int`**, θα φτάσετε ως ν=16. Με μεγάλους ακεραίους **`long`**, ως ν=39. Για μεγαλύτερα ν, χρησιμοποιήστε την κλάση περιτυλίγματος `BigInteger`. Τέτοιοι ακέραιοι είναι αντικείμενα κι όχι απλές μεταβλητές. Φτιάξτε πρόγραμμα που να υπολογίζει το ν παραγοντικό όλων των αριθμών μέχρι το 1000.

```
public static BigInteger factorial(long n) {
  BigInteger product = new BigInteger("1");
  BigInteger increment = new BigInteger("1");
  for (long i = 1L; i <= n; i++) {
    product = product.multiply(increment);
    increment = increment.add(BigInteger.ONE);
  }
  return product;
}
public static void main(String[] args) {
  BigInteger t = new BigInteger("0");
  for(long i = 1L; i <= 1000L; i++) {
    t = factorial(i);
    System.out.println("fac(" + i + ")=" + t.toString());
  }
}
```

Εικόνα 42 Χρήση κλάσης περιτυλίγματος

Α33. Φτιάξτε πρόγραμμα που να περιλαμβάνει μεθόδους που συγκρίνουν αντικείμενα της κλάσης Κλάσμα. Υπενθυμίζεται ότι το κλάσμα έχει δύο ιδιότητες και ότι ο παρονομαστής του αποκλείεται να γίνει 0. Εκτός από τους βασικούς τελεστές +, -, *, /, θα υλοποιήσετε τους σχεσιακούς τελεστές με τις μεθόδους `isEqual`, `isGreater` και μετά θα χρησιμοποιήσετε αυτές τις δύο για να υλοποιήσετε τις `isDifferent`, `isGreaterThanOrEqual`, `isLessThanOrEqual`, `isLessThan`. Στη `main` θα πρέπει να καλέσετε τουλάχιστον μία φορά όλες τις μεθόδους που φτιάξατε. Αριθμητική κλασματικών αριθμών. υλοποιήστε απλοποίηση, σχεσιακούς τελεστές σύγκρισης. Για μια (μερική) υλοποίηση δείτε την ενότητα 2.15 Κλασματικοί αριθμοί.

- Φτιάξτε αριθμομηχανή που λειτουργεί επί κλασμάτων. Το πρόγραμμα θα διαχειρίζεται έναν αριθμό πχ 5 κλασμάτων. Κατά τη βούληση του χρήστη θα μπορεί να θέτει (αλλάζει) τους όρους τους, να εκτελεί τις βασικές αριθμητικές και λογικές πράξεις, να τα τυπώνει και ό,τι άλλο εσείς κρίνετε χρήσιμο για μια αριθμομηχανή.

Α34. Φτιάξτε κλάση ΤραπεζικόςΛογαριασμός και υλοποιήστε τις απαραίτητες μεθόδους που θα τον πιστώνουν και θα τον χρεώνουν παίρνοντας όλες τις απαραίτητες προφυλάξεις. Φτιάξτε τρεις λογαριασμούς και έναν απλό διερμηνευτή που θα δέχεται εντολές για ανάληψη / κατάθεση σε αυτούς.

Α35. Φτιάξτε κλάση ATM (Αυτόματη Ταμειολογιστική Μηχανή) με την οποία κάνουν τραπεζικές κινήσεις οι συναλλασσόμενοι. Για την εκτέλεση των κινήσεων χρησιμοποιήστε τις μεθόδους της κλάσης ΤραπεζικόςΛογαριασμός. Δημιουργήστε δύο ATM.

Α36. Άλγεβρα συνόλων. Υλοποιήστε κατασκευαστές συνόλων και τις βασικές λειτουργίες: ανήκει, υποσύνολο, τομή, ένω-

ση, διαφορά.

- Αριθμομηχανή συνόλων. Φτιάξτε πρόγραμμα που θα διαχειρίζεται σύνολα: θα επιτρέπει την προσθήκη / διαγραφή στοιχείων, την εκτέλεση πράξεων επί αυτών και την εκτύπωση αποτελεσμάτων.

Α37. Φτιάξτε κλάσεις που να αντικατοπτρίζουν το παρακάτω σενάριο: Ένα όχημα έχει έναν κινητήρα και μπορεί να είναι αυτοκίνητο ή μοτοσικλέτα. Ο κινητήρας καίει κανονική ή αμόλυβδη βενζίνη και συγκεκριμένο κυβισμό και ιπποδύναμη. Τα αυτοκίνητα έχουν τέσσερις τροχούς, ενώ οι μοτοσικλέτες δύο. Για κάθε τροχό γνωρίζουμε τη διάμετρο και την πίεση στην οποία είναι φουσκωμένος. Οι λειτουργίες που μπορεί να εκτελέσουμε στο όχημα είναι: άναμμα / σβήσιμο της μηχανής, αλλαγή τροχού, έλεγχος και φούσκωμα τροχού.

Α38. Ένας κερματοδέκτης δέχεται κέρματα 1 και 2 ευρώ, καθώς επίσης και 50, 20, 10, 5, 2, 1 λεπτών. Φτιάξτε μεθόδους που εισάγουν κέρματα στον κερματοδέκτη, τυπώνουν το ποσό που έχει μέσα. Επίσης, φτιάξτε μέθοδο που επιστρέφει συγκεκριμένο ποσόν με τον ελάχιστο αριθμό κερμάτων. Εάν αυτό δεν γίνεται (εξαιτίας έλλειψης συγκεκριμένων κερμάτων), προσπαθεί να επιστρέψει το ποσόν με τα κέρματα που είναι διαθέσιμα στις θήκες του, αλλά πάλι φροντίζει να δώσει ελάχιστα κέρματα και ζητάει συγγνώμη που έδωσε πολλά «ψιλά». Αν δεν μπορεί να επιστρέψει το συγκεκριμένο ποσό, ζητάει συγγνώμη και δηλώνει αδυναμία.

Α39. Σχεδιάστε κλάση «Εργαζόμενος» που περιέχει ατομικά στοιχεία και δύο υποκλάσεις (derived) «Ωρομίσθιος» και «Μισθωτός» που προσθέτουν τα μισθολογικά στοιχεία του εργαζομένου.

Α40. Αριθμητική μιγαδικών αριθμών. Εκτός από τις τέσσερις βασικές αλγεβρικές πράξεις, υλοποιήστε μέτρο $\sqrt{x^2+y^2}$. Για μια υλοποίηση που περιλαμβάνει και διεπαφή ρυθμού χαρακτήρων, δείτε την ενότητα 2.17 Μιγαδικοί και διερμηνευτής.

Α41. Η κλάση `Nomisma` αναπαριστά τιμές νομισμάτων, όπως το ευρώ και το δολάριο. Για κάθε νόμισμα θα υπάρχει (α) το διεθνές τριψήφιο σύμβολό του, πχ EUR ή USD, (β) το πρόσημό του, + ή -, (γ) η ακέραια τιμή του, πχ 3 ή 142, και (δ) τα εκατοστά του[9], πχ 33 ή 99. Φτιάξτε κατασκευαστή `Nomisma("EUR", '+', 3, 33)` ή `Nomisma("USD", '-', 142, 99)` που να δημιουργεί «νόμιμα» νομίσματα, εκτυπωτικό `toString` που τυπώνει `EUR+3.33` ή `USD-142.99` και δύο μεθόδους που να προσθέτουν και να αφαιρούν νομίσματα. Επίσης φτιάξτε ένα κατασκευαστή ακόμη που θα "αναγνωρίζει" τιμές νομισμάτων όπως στα παραπάνω παραδείγματα.

- Φτιάξτε πρόγραμμα που θα υπολογίζει την συνολική αξία σε ευρώ των νομισματικών τιμών που έχουν δημιουργηθεί. Επιτρέπουμε νομίσματα σε EUR, USD, GBP. Για τα δύο τελευταία τηρούμε την ισοτιμία τους με το ευρώ.
- Μετά από πολύμηνη χρήση, διαπιστώνετε ότι στα αντικείμενα της κλάσης `Nomisma` εκτελούνται πολύ συχνά οι μέθοδοι της πρόσθεσης και της αφαίρεσης, οπότε για λόγους επιδόσεων σκεφτόσαστε να αλλάξετε την αναπαράσταση του αριθμητικού μέρους και του πρόσημου. Αντί για ένα χαρακτήρα, έναν ακέραιο και ένα μικρό ακέραιο (`byte`) θα χρησιμοποιήσετε έναν μεγάλο (προσημασμένο) ακέραιο `long`. Έτσι ο 1.33 αποθηκεύεται ως 133 και ο -143.99 ως -14399. Αλλάξτε την υλοποίηση της κλάσης, χωρίς ωστόσο να πειράξετε στο παραμικρό τις δημόσιες μεθόδους, έτσι ώστε να υπάρχει συμβατότητα με τα προγράμματα που ήδη χρησιμοποιούν την κλάση `Nomisma`.

2.14 Υλοποίηση κλάσης λάμπα σε Java

```
package lampa;
public class Light {

  // instance variable
  /* value ranges from 0 ... 1.
   * 0 means the light is off, 1 means the light is fully on
   * some value in between means light operates in reduced capacity
   */
  private float lightStatus;

  // constructors
  public Light() {
    lightStatus = (float) 0;
    System.out.println("A new light was created turned off.");
  }
  public Light(float initialStatus) {
    lightStatus = minimum(maximum(initialStatus, (float) 0), (float) 1);
    System.out.println("A new light was created with initial status " + lightStatus + ".");
  }

  // public methods
  public void on() {
  // turns the light on to full brightness
    lightStatus = (float) 1;
    System.out.println("The light was turned on.");
  }
```

[9] Διερευνήστε γιατί δεν είναι καλή ιδέα να αναπαραστήσουμε την τιμή του νομίσματος, πχ 3.33 με έναν float ή ακόμα και με double πραγματικό αριθμό; Σε μερικές γλώσσες, όπως στην «αρχαία» γλώσσα προγραμματισμού Cobol υπήρχε η αναπαράσταση αριθμών BCD (Binary coded Decimal) – βρείτε πώς λειτουργεί και γιατί χρησιμοποιείται παρόλο που είναι «σπάταλη».

```
  public void off() {
  // turns the light off to complete darkness
    lightStatus = (float) 0;
    System.out.println("The light was turned off.");
  }

  public void brighten() {
  // turns the light a little brighter
    lightStatus = minimum(1, (lightStatus + (float) 0.1));
    System.out.println("The light became brighter.");
  }

  public void dim() {
  // turns the light a little dimmer
    lightStatus = maximum(0, (lightStatus - (float) 0.1));
    System.out.println("The light became dimmer.");
  }

  // private methods
  private float minimum(float x, float y) {
  //returns the lesser of two floating numbers
    return (x < y)?x:y;
  }

  private float maximum(float x, float y) {
  //returns the bigger of two floating numbers
    return (x > y)?x:y;
  }

} //end class
```

2.15 Κλασματικοί αριθμοί και συλλογή απορριμάτων

```
package klasma;
public class Klasma {

  //μεταβλητή κλάσης
  private static int count = 0; //μετράει πόσα κλάσματα υπάρχουν

  //μεταβλητές στιγμιοτύπων
  private int arithmitis;   // ο αριθμητής του κλάσματος
  private int paronomastis; // ο παρονομαστής του κλάσματος

  //κατασκευαστές & καταστροφέας
  public Klasma() { // ο πιο απλός κατασκευαστής
    arithmitis = 0;
    paronomastis = 1;
    count ++;
    System.out.println("Πλήθος " + count);
  }

  public Klasma(int arithmitis, int paronomastis) {
    this.arithmitis = arithmitis;  // το this χρειαζεται για να ξερει σε οτι ...
    this.paronomastis = ((paronomastis==0)?1:paronomastis);  // αναφερομαστε ...
    count ++;                              // στην ιδιότητα και όχι στην παράμετρο
    System.out.println("Πλήθος " + count);
  }

  public Klasma(int timh) {
      this(timh, 1); //χρησιμοποιεί τον προηγούμενο κατασκευαστή
    }

  public void finalize() { // να και ο καταστροφέας
    count --;
    System.out.println("Πλήθος " + count);
  }

  //μέθοδοι ανάκλησης (get) και τοποθέτησης (set)
  public int getArithmitis() {
    return(arithmitis);
  }
  public int getParonomastis() {
    return(paronomastis);
  }
```

```
  public void setArithmitis(int arithmitis) {
    this.arithmitis=arithmitis;
  }
  public void setParonomastis(int paronomastis) {
    if (paronomastis != 0)
      this.paronomastis=paronomastis;
    // αν πας να βάλεις παρονομαστή το 0, αφήνει τον παλιό παρονομαστή
  }
  //άλλες μέθοδοι
  public float compute() {    // το cast με το float είναι απαραίτητο, γιατί ...
    return ((float) arithmitis) / paronomastis; // αλλιώς κάνει ακέραια διαίρεση
  }

  public String toString() {
    return arithmitis + "/" + paronomastis;
  }

  public void antistrefo() { // αντιστρέφει τυς όρους του κλάσματος
    if (arithmitis != 0) {
      int temp = paronomastis; paronomastis = arithmitis; arithmitis = temp;
    } // αν ο αριθμητής είναι 0, το κλάσμα παραμένει ως έχει
  }

  public void aplopoio() { // απλοποιεί το κλάσμα
    int mkdKlasmatos = mkd(arithmitis, paronomastis);
    arithmitis = arithmitis / mkdKlasmatos;
    paronomastis = paronomastis / mkdKlasmatos;
  }

  //ιδιωτικές μέθοδοι
  private static int mkd (int m, int n) {
    while (n!=0 && m!=0)
      if (m > n) m -=n; else n -=m;
    return Math.max(m, n);
  }
} //end class
```

```
package klasma;
public class KlasmaAlgebra {
  //όλες οι μέθοδοι επιστρέφουν νέο κλάσμα που έχει το αποτέλεσμα

  public static Klasma antistrefo(Klasma k ) {
    // δεν πειράζει το αρχικό κλάσμα - δημιουργεί άλλο που ειναι αντεστραμμένο
    return new Klasma(k.getParonomastis(), k.getArithmitis());
  }

  public static Klasma prostheto(Klasma k1, Klasma k2) {
    Klasma apot = new Klasma(); //προσωρινή μεταβλητή αντικειμένου
    apot.setParonomastis(k1.getParonomastis() * k2.getParonomastis());
    apot.setArithmitis(k1.getArithmitis() * k2.getParonomastis() +
                       k2.getArithmitis() * k1.getParonomastis()   );
    return apot;
  }

  public static Klasma afairo(Klasma k1, Klasma k2) {
    Klasma apot = new Klasma(k1.getArithmitis()   * k2.getParonomastis() -
                             k2.getArithmitis()   * k1.getParonomastis() ,
                             k1.getParonomastis() * k2.getParonomastis() );
    return apot;
  }

  public static Klasma pollaplasiazo(Klasma k1, Klasma k2) {
    return new Klasma(k1.getArithmitis()   * k2.getArithmitis()  ,
                      k1.getParonomastis() * k2.getParonomastis() );
  }

  public static Klasma diairo(Klasma k1, Klasma k2) {
    if (k2.getArithmitis() == 0)
      return new Klasma();
    else
      return new Klasma(k1.getArithmitis()   * k2.getParonomastis() ,
                        k1.getParonomastis() * k2.getArithmitis()    );
  }
} //end class
```

```
package klasma;
public class KlasmaDemo {

  public static void main(String[] args) {
    Klasma r = new Klasma(5, 7);
    System.out.println("Arithmitis=" + r.getArithmitis()+
                       " Paronomastis=" + r.getParonomastis() +
                       " Timh=" + r.compute() );
    r.setArithmitis(3); r.setParonomastis(4);
    System.out.println("To idio klasma meta apo allagh synteleston " + r.toString());

    printLines(2);
    Klasma s = new Klasma(1, 2);
    System.out.println("Ena allo klasma " + s); // δεν χρειάζεται να βάλω .toString()
    System.out.println("To antistrofo toy einai " + KlasmaAlgebra.antistrefo(s));

    // η παραπάνω εντολή έφτιαξε ένα (ανώνυμο) κλάσμα που δεν είναι προσπελάσιμο
    System.gc(); // τώρα καταστρέφεται αυτό το κλάσμα

    printLines(2);
    Klasma t = KlasmaAlgebra.prostheto(r, s);
    System.out.println("Prosthesi " + r + " syn " + s + " kanei " + t);
    t.aplopoio();
    System.out.println(" me aplopoiisi " + t);

    printLines(2);
    System.out.println("Afairesi " + r + " meion " + s + " kanei " + KlasmaAlgebra.afairo(r, s));
    System.exit(0);
  } //end main

  private static void printLines(int n){
    for (int i=1; i<=n; i++)
      System.out.println();
  }
} //end class
```

2.16 Πολύφωτο

Παραλλαγή της λάμπας όπως έχει υλοποιηθεί στην ενότητα 2.14. Η δεύτερη κλάση κατασκευάζει και διαχειρίζεται πίνακα από αντικείμενα μέσω ενός απλού διερμηνευτή διαταγών.

```
package polifoto;
public class Lampa {

  private final boolean ANAMMENH = true; //σταθερες της κλασης
  private final boolean SBHSMENH = false;

  private boolean katastasi; //ιδιοτητα των αντικειμενων Λαμπα

  public Lampa() {   //κατασκευαστης: φτιαχνει μια νεα σβηστη λαμπα
    katastasi = SBHSMENH;
    System.out.println("Κατασκευάστηκε μια νέα λάμπα");
  }

  public boolean getKatastasi() {   //επιστρεφει καταστση λαμπας γιατι ειναι ιδιωτικη
    return katastasi;
  }

  public void anapse() {   //αναβει τη λαμπα
    katastasi = ANAMMENH;
    System.out.println("Αναψε μια λάμπα");
  }

  public void sbhse() {   //σβηνει τη λαμπα
    katastasi = SBHSMENH;
    System.out.println("Εσβησε μια λάμπα");
  }
}
```

```
package polifoto;
import javax.swing.*;
```

```
public class Polifoto {
  public static void main(String[] args) {

    Object[] fotistiko; //το φωτιστικο ειναι ενας πινακας με στοιχεια αντικειμενα

    String sLampes = JOptionPane.showInputDialog("Πόσες λάμπες έχει το φωτιστικό;");
    int iLampes = Integer.parseInt(sLampes);
    fotistiko = new Object[iLampes]; //κατασκευαζεται πινακας με μεγεθος που του δωσαμε

    for(int i = 0; i < iLampes; i++) //εδω γεμιζει ο πινακας με στοιχεια
      fotistiko[i] = new Lampa(); //καθε στοιχειο ειναι μια νεα λαμπα

    String sEpiLampa; int iEpiLampa; //επιλογη λαμπας
    String sEpiLeit; char cEpiLeit; //επιλογη λειτουργιας
    Lampa epiLampa; //επιλεγμενη λαμπα πανω στην οποια θα γινει η λειτουργια
    while (true) {
      sEpiLampa = JOptionPane.showInputDialog("Επιλογή λάμπας 0 έως " + (iLampes - 1) +
                                              ". Αλλος αριθμός τερματίζει το πρόγραμμα.");
      iEpiLampa = Integer.parseInt(sEpiLampa);
      if ((iEpiLampa < 0) || (iEpiLampa>=iLampes)) break;
      epiLampa = (Lampa) fotistiko[iEpiLampa];

      sEpiLeit = JOptionPane.showInputDialog(
                 "Επιλογή λειτουργίας α=αναψε, σ=σβησε, κ=κατασταση");
      cEpiLeit = sEpiLeit.charAt(0);

      switch (cEpiLeit) {
        case 'α':
          epiLampa.anapse();
          break;
        case 'σ':
          epiLampa.sbhse();
          break;
        case 'κ':
          JOptionPane.showMessageDialog(null,"Η λάμπα " + iEpiLampa + " είναι " +
                                        (epiLampa.getKatastasi()?"αναμμένη":"σβηστή"));
          break;
      } //end switch

    } //end while

    for (int i = 0; i < fotistiko.length; i++) //τυπωνει την κατασταση ολων των λαμπων
      System.out.println("Η λαμπα " + i + " είναι " +
                         (((Lampa) fotistiko[i]).getKatastasi()?"αναμμένη":"σβηστή"));
    System.exit(0);
  } //end main
 } //end class
```

2.17 Μιγαδικοί και διερμηνευτής

```
package migadikoi;

public class Migadikos {

  //ιδιότητες μιγαδικού
  public double pragmat; //πραγματικό μέρος
  public double fantast; //φανταστικό μέρος

  //κατασκευαστές
  public Migadikos (){
    //κατασκευάζει 0+0i
  }

  public Migadikos (double pra, double fan){
     pragmat = pra; fantast = fan;
  }

  public void setMigadikos(double pra, double fan) {
     pragmat = pra; fantast = fan;
  }

  public String toString() {
    return pragmat + (fantast >= 0 ? "+" : "") + fantast + "i";
  }
```

```
  public double metro() {
    return Math.sqrt(pragmat * pragmat + fantast * fantast);
  }

  public Migadikos syzyghs() {
    return new Migadikos(pragmat, - fantast);
  }

  public void add(Migadikos x, Migadikos y) {
    pragmat = x.pragmat + y.pragmat;
    fantast = x.fantast + y.fantast;
  }
  public void subtract(Migadikos x, Migadikos y) {
    pragmat = x.pragmat - y.pragmat;
    fantast = x.fantast - y.fantast;
  }

  public void multiply(Migadikos x, Migadikos y) {
    pragmat = x.pragmat * y.pragmat - x.fantast * y.fantast;
    fantast = x.pragmat * y.fantast + x.fantast * y.pragmat;
  }

  public void divide(Migadikos x, Migadikos y) {
    double temp = y.metro(); temp = temp * temp;
    Migadikos w = new Migadikos();
    w.multiply(x, y.syzyghs());
    pragmat = w.pragmat / temp;
    fantast = w.fantast / temp;
  }
} //end class
```

```
package migadikoi;
import java.io.*;
public class MigadikoiDemo {

  public static void main(String[] args) throws IOException {
  /*Διερμηνευτής διαταγών που διαβάζονται από την είσοδο και έχουν τη μορφή χαρακτήρων.*/

    Migadikos x = new Migadikos();
    Migadikos y = new Migadikos();
    Migadikos z = new Migadikos();

    BufferedReader stdIn = new BufferedReader (new InputStreamReader(System.in));
    char cIn; //χαρακτήρας διαταγής ενέργειας
    double pra; double fan;

    do{ //επαναλάμβανε
      cIn = readChar(stdIn);
      switch (cIn){
        case 'X': pra = readNumber("Πραγματικό μέρος ", stdIn);
                  fan = readNumber("Φανταστικό μέρος ", stdIn);
                  x.setMigadikos(pra, fan);
                  break;
        case 'Y': pra = readNumber("Πραγματικό μέρος ", stdIn);
                  fan = readNumber("Φανταστικό μέρος ", stdIn);
                  y.setMigadikos(pra, fan);
                  break;
        case 'Z': System.out.println("X = " + x.toString() + "\n" +
                                     "Y = " + y.toString() + "\n" +
                                     "Z = " + z.toString());
                  break;
        case '+': z.add(x,y);
                  z.toString();
                  break;
        case '-': z.subtract(x, y);
                  z.toString();
                  break;
        case '*': z.multiply(x, y);
                  z.toString();
                  break;
        case '/': z.divide(x,y);
                  z.toString();
                  break;
        case 'T': break; //Πιάσε το για να μην το θεωρήσει άγνωστη πράξη
```

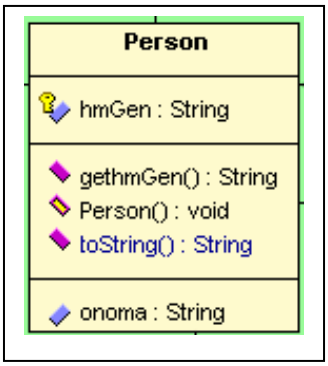
```
        default : System.out.println(printOdhgies());
      } //end switch
    } while (cIn !='T'); //end do - μέχρι
    System.out.println("Το πρόγραμμα τερματίστηκε.");
    System.exit(0);
  } //end main

  private static char readChar (BufferedReader stdIn) throws IOException {
  /*Διαβάζει μια συμβολοσειρά από την είσοδο και επιστρέφει τον πρώτο της χαρακτήρα
    στα κεφαλαία. Αν η συμβολοσειρά είναι κενή, επιστρέφει #. */
    System.out.print("Ενέργεια >");
    System.out.flush();
    String strIn = stdIn.readLine().toUpperCase();
    if (strIn.equals(""))
      return '#';
    else
      return Character.toUpperCase(strIn.charAt(0));
  } //end readChar

  private static double readNumber(String meros, BufferedReader stdIn) throws IOException {
    System.out.print("Αριθμός (" + meros + ") >");
    System.out.flush();
    return Double.parseDouble(stdIn.readLine());
  }

  private static String printOdhgies() {
    return "Πρέπει να εισάγεις ένα από τα:\n" +
           "  X       δίνει τιμή στον πρώτο μιγαδικό\n" +
           "  Y       δίνει τιμή στο δεύτερο μιγαδικό\n" +
           "  Z       τυπώνει τιμές των X, Y και του αποτελέσματος της πράξης που προηγήθηκε\n" +
           "  +,-,*,/ εκτελεί την αντίστοιχη πράξη\n" +
           "  T       τερματίζει το πρόγραμμα.";
  } //end printOdhgies
} //end class MigadikoiDemo
```

2.18 Γιατροί και ασθενείς (κληρονομικότητα)



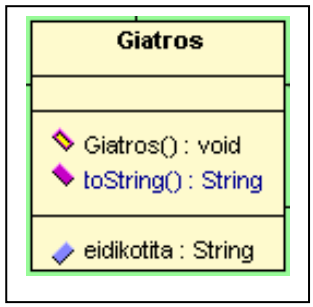

```
public class Person {

  protected String onoma; //ορατά εδώ και στις υποκλάσεις της Person
  protected String hmGen; // ημερομηνια γέννησης

  public Person(String onoma, String hmGen) { //κατασκευαστής
    this.onoma = onoma;
    this.hmGen = hmGen;
  }

  public String getOnoma() { return this.onoma; } //μόνον για ανάγνωση
  public String gethmGen() { return this.hmGen; }
  // δεν υπάρχουν set - οι ιδιότητες του ασθενούς απαγορεύεται να αλλάξουν

  public String toString() {
    return "Ον/νυμο: " + this.onoma + "\n" + // το this δεν είναι απαραίτητο
           "Ημ.γέν: " + this.hmGen;          // απλώς τονίζει ότι πρόκειται
  }                                          // για μεταβλητές στιγμιότυπου
}
```



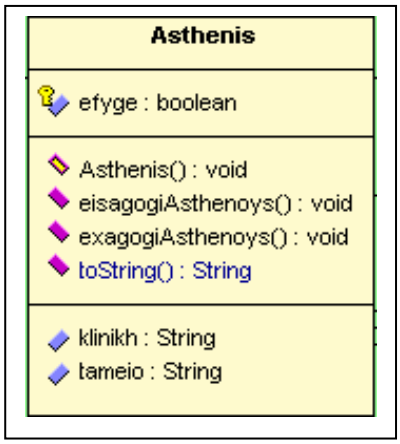

```
public class Giatros extends Person {

  protected String eidikotita;

  public Giatros(String onoma, String hmGen, String eidikotita) {
    super(onoma, hmGen); // καλεί τον κατασκευαστή της υπερκλάσης
    this.eidikotita = eidikotita;
  }

  public String getEidikotita() { return eidikotita; }

  public String toString() {
    return "ΓΙΑΤΡΟΣ" + "\n" + super.toString() + "\n" + "Ειδικότ: " + this.eidikotita;
  }
}
```




```
public class Asthenis extends Person {

  protected String tameio;
  protected String klinikh = "";    // συμπληρώνεται κατά την εισαγωγή
  protected boolean efyge = false; // γίνεται true όταν γίνει εξαγωγή

  public Asthenis(String onoma, String hmGen, String tameio) {
    super(onoma, hmGen);
    this.tameio = tameio;
  }

  public String getTameio() { return this.tameio; }
  public void setTameio() { this.tameio = tameio; }
  public String getKlinikh() { return this.klinikh; }

  public void eisagogiAsthenoys(String klinikh) {
    if (this.efyge) System.out.println("Δεν μπορει να γίνει εισαγωγή αφού ο ασθενής έχει φύγει");
    else if (!this.klinikh.equals("")) System.out.println("Εχει εισαχθεί στην " + this.klinikh);
    else this.klinikh = klinikh;
  }

  public void exagogiAsthenoys() {
    if (this.klinikh.equals("")) System.out.println("Δεν έχει εισαχθεί ακόμη");
    else                         this.efyge = true;
  }

  public String toString() {
    return "ΑΣΘΕΝΗΣ" + "\n" + super.toString() + "\n" + "Ταμείο: " + this.tameio + "\n" +
           "Κλινική: " + this.klinikh + "\n" + (this.efyge ? "Έφυγε" : "Νοσηλεύεται");
  }
}
```

```
public class AtomoDemo {
  public static void main(String[] args) {
    Person pers = new Person("Γιάννης Ιωάννου", "12/03/1970");
    System.out.println(pers.toString());

    Giatros giat = new Giatros("Πέτρος Βασιλείου", "18/02/1960", "Παιδίατρος");
    System.out.println(giat.toString());

    Asthenis ast = new Asthenis("Νικος Νικολάου", "03/06/1980", "ΙΚΑ");
    ast.eisagogiAsthenoys("Ορθοπεδική");    System.out.println(ast);
    ast.exagogiAsthenoys();                 System.out.println(ast);
  }
}
```