# Ευρετικοί Κανόνες Αντικειμενοστρεφούς Σχεδίασης

***Object-Oriented Design Heuristics***

# Ευρετικοί Κανόνες

Πέρας της φάσης "Σχεδίαση Λογισμικού":

"Πρόκειται για καλή σχεδίαση ή όχι ?"

Συνήθως η απάντηση δίδεται από τους guru της ομάδας !

Ο Arthur J. Riel σε μία προσπάθεια να εξαλείψει την αναγκαιότητα τέτοιων εμπειρογνωμόνων, κατέγραψε 60 εμπειρικούς κανόνες.

• Οι κανόνες αυτοί δεν είναι αυστηρές προδιαγραφές που πρέπει να τηρούνται υποχρεωτικά.

• Πολλοί κανόνες έρχονται σε αντίθεση μεταξύ τους

• Όπως κάθε engineering activity υπόκειται σε συμβιβασμούς

• Πρόκειται για εμπειρικούς κανόνες -> Δυσκολία στην αυτοματοποίηση

# Ευρετικοί Κανόνες

Παραδείγματα τέτοιων κανόνων που έχουμε δει ήδη (ερμηνεύονται με βάση τις αρχές Σχεδίασης):

• *Όλα τα δεδομένα θα πρέπει να βρίσκονται κρυμμένα στην κλάση τους*

• *Οι χρήστες μιας κλάσης θα πρέπει να εξαρτώνται από τη δημόσια διασύνδεσή της αλλά μία κλάση δεν θα πρέπει να εξαρτάται από τους χρήστες της*

• *Ελαχιστοποιείστε τον αριθμό των μηνυμάτων που μπορεί να δεχθεί μία κλάση*

• *Υλοποιείστε μία ελάχιστη δημόσια διασύνδεση την οποία υλοποιούν όλες οι κλάσεις (π.χ. λειτουργίες copy, έλεγχο ισότητας, printing κλπ)*

• *Μην γεμίζετε τη δημόσια διασύνδεση μιας κλάσης με στοιχεία τα οποία οι χρήστες της κλάσης δεν μπορούν ή δεν ενδιαφέρονται να χρησιμοποιήσουν*

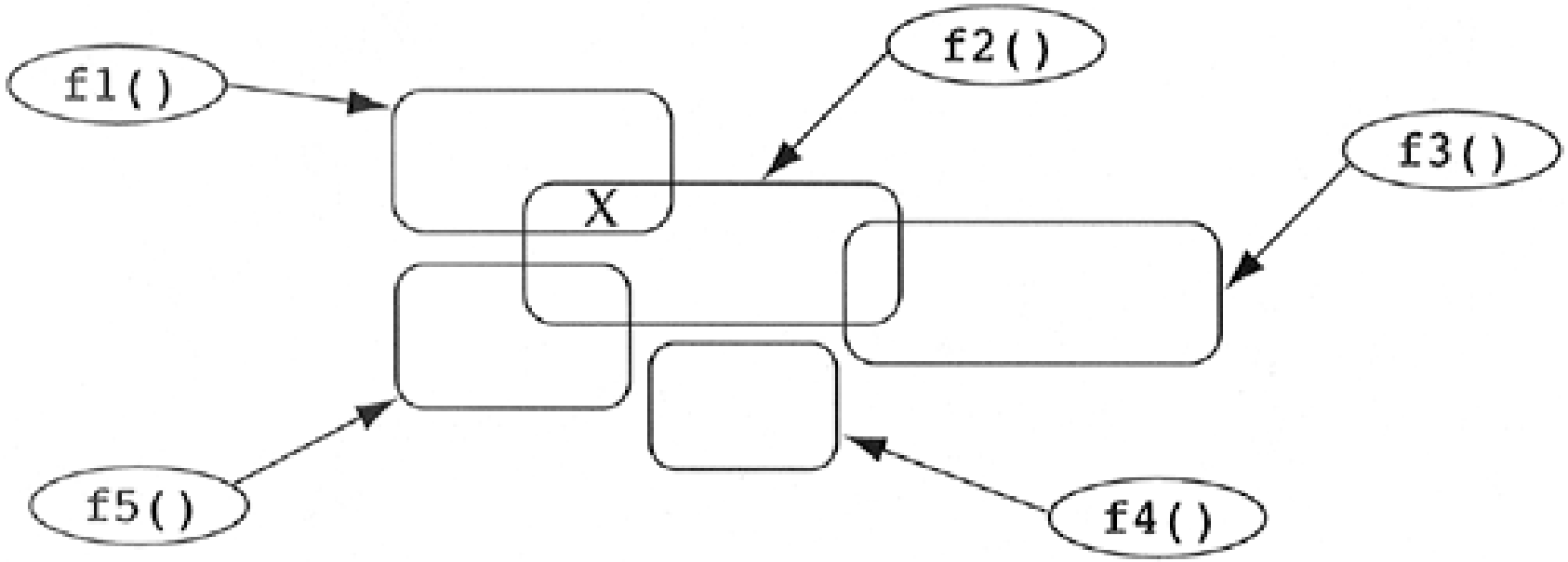
• *Κάθε κλάση θα πρέπει να ενσωματώνει μία και μόνο μία κύρια αφαίρεση*

• *Συσχετιζόμενα δεδομένα και συμπεριφορά θα πρέπει να διατηρούνται στο ίδιο σημείο*

# Ευρετικοί Κανόνες

**Σύγκριση δομών διαδικασιακών και αντικειμενοστρεφών συστημάτων**

Θεωρούμε μία εφαρμογή η οποία μπορεί να διασπασθεί σε πέντε συναρτήσεις.



# Ευρετικοί Κανόνες

Οι δομές δεδομένων δημιουργούνται ως δεύτερη σκέψη, κατά τη διάρκεια της υλοποίησης των συναρτήσεων.

Τα μέλη της ομάδας ανάπτυξης αντιλαμβάνονται κατά την υλοποίηση ότι ορισμένες συναρτήσεις μπορούν να μοιράζονται τμήματα των υποκείμενων δεδομένων.

Στον διαδικασιακό προγραμματισμό οι εξαρτήσεις των δεδομένων (**δηλαδή τα δεδομένα από τα οποία εξαρτάται ένα τμήμα κώδικα**) ανακαλύπτονται εξετάζοντας απλά την υλοποίηση των συναρτήσεων. (τυπικές παράμετροι, τοπικές μεταβλητές και προσπελαζόμενες καθολικές μεταβλητές).

Είναι όμως προβληματικό το αντίθετο: Ο εντοπισμός των διαδικασιακών εξαρτήσεων ενός δεδομένου (**δηλαδή οι συναρτήσεις οι οποίες εξαρτώνται από ένα δεδομένο**). Στο διαδικασιακό προγραμματισμό δεν υπάρχει σαφής συσχέτιση μεταξύ δεδομένων και λειτουργικότητας.

# Ευρετικοί Κανόνες

Έστω ότι τροποποιείται (κατόπιν αλλαγών στις απαιτήσεις) η δομή δεδομένων X

Με βάση το διάγραμμα: Αλλαγές μόνο στις `f1()` και `f2()`

Αν όμως (την προηγούμενη εβδομάδα), ένας προγραμματιστής δημιούργησε τη συνάρτηση `f6()` εν αγνοία των προγραμματιστών των `f1()` και `f2()` και η οποία εξαρτάται επίσης από τα δεδομένα που συμβολίζονται ως X:

Είναι πλέον πιθανό, να πραγματοποιηθούν όλες οι αλλαγές στα δεδομένα και στις συναρτήσεις `f1()` και `f2()` χωρίς όμως το πρόγραμμα τα λειτουργεί σωστά.

Το πρόβλημα έγκειται στη μη τεκμηριωμένη εξάρτηση δεδομένων/συμπεριφοράς λόγω της μονόδρομης συσχέτισης μεταξύ κώδικα και δεδομένων.

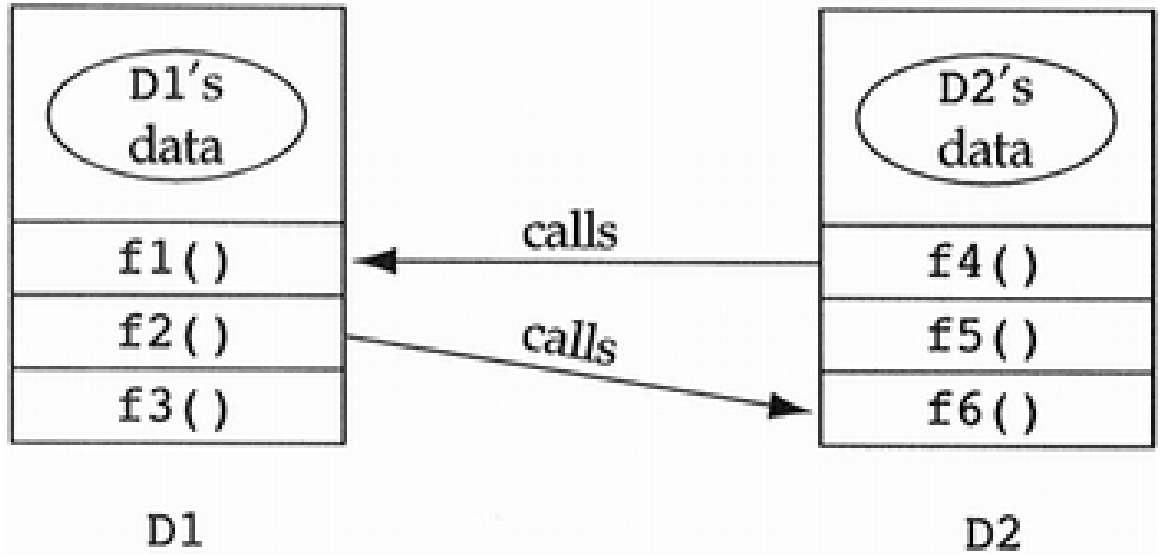
Σχεδόν όλα τα μέρη συστήματος που βασίζονται σε διαδικασιακό τρόπο ανάπτυξης έχουν μία αλυσιδωτή δομή εξαρτήσεων (spaghetti-like code) όπου κάθε τμήμα κώδικα αποτελεί μέρος αυτής.

# Ευρετικοί Κανόνες

Στον αντικειμενοστρεφή τρόπο σχεδίασης η συμπεριφορά του συστήματος οδηγεί τη διαδικασία αποσύνθεσης και των δεδομένων.

Το αποτέλεσμα είναι η σχεδίαση του συστήματος ως μία συλλογή αποκεντρωμένων τμημάτων δεδομένων με σαφώς καθορισμένες δημόσιες διασυνδέσεις.

Οι μόνες εξαρτήσεις της λειτουργικότητας από τα δεδομένα είναι οι λειτουργίες της δημόσιας διασύνδεσης που εξαρτώνται από τα συσχετιζόμενα δεδομένα.



# Παράδειγμα

Εφαρμογή καταχώρησης αριθμών φορολογικού μητρώου

# Παράδειγμα

Διαδικασιακός προγραμματισμός

```
void IssueNew(int A[], int *size);
void checkValidity(int A[], int size);
void countByLastDigit(int A[], int size, int B[]);
void printNumbers(int A[]);
main()
{
int numbers[];
int currentsize;
int count[9];
issueNew(numbers, &currentsize);
checkValidity(numbers, currentsize);
countBylastdigit(numbers, currentsize, count);
Printnumbers(count);
}
```

# Παράδειγμα

**Τι θα συμβεί αν αυξήσω το μέγεθος των αριθμών φορολογικού μητρώου σε δεκαψήφιους?**

# Ευρετικοί Κανόνες

Ωστόσο:

η εικόνα σε ένα διαδικασιακό σύστημα δεν είναι πάντοτε τόσο κακή

ούτε βέβαια η εικόνα ενός αντικειμενοστρεφούς συστήματος τόσο ρόδινη

# Ευρετικοί Κανόνες

**Καλή εξέλιξη σε ένα διαδικασιακό σύστημα**

Στην πράξη κάθε δομή δεδομένων τοποθετείται σε ξεχωριστό αρχείο όπου επίσης περιλαμβάνονται οι συναρτήσεις που εξαρτώνται από τη συγκεκριμένη δομή.

Κατά συνέπεια το σύστημα είναι ευκολότερα συντηρήσιμο.

Στην ουσία χρησιμοποιείται μία σύμβαση, σύμφωνα με την οποία τα δεδομένα και οι συναρτήσεις που λειτουργούν επί αυτών ομαδοποιούνται σε μία λογική οντότητα, ότι ακριβώς κάνει και μία κλάση στον αντικειμενοστρεφή προγραμματισμό. Εδώ δεν υπάρχουν κλάσεις, και για αυτό χρησιμοποιείται ένα μηχανισμός χαμηλοτέρου επιπέδου.

Ωστόσο, ένας προγραμματιστής θα πρέπει:

- να κατανοήσει την αναγκαιότητα αυτής της σύμβασης
- να την τηρεί πιστά πάντοτε. Το τελευταίο είναι και το δυσκολότερο.

# Ευρετικοί Κανόνες

**Κακή εξέλιξη σε ένα αντικειμενοστρεφές σύστημα**

Τι μπορεί να "στραβώσει" σε ένα αντικειμενοστρεφές σχέδιο ώστε αυτό να μην εξελιχθεί καλά;

Υπάρχουν δύο κύρια είδη τέτοιων προβλημάτων:

• Πρόβλημα "**θεϊκής κλάσης**" (God class) (μη ομοιόμορφη κατανομή της λειτουργικότητας του συστήματος)

• Πρόβλημα "**πολλαπλασιασμού των κλάσεων**" (proliferation of classes) ( δημιουργία υπερβολικού αριθμού κλάσεων)

Το πρόβλημα των θεϊκών κλάσεων εμφανίζεται σε δύο μορφές, τη μορφή συμπεριφοράς και τη μορφή δεδομένων.

# Ευρετικοί Κανόνες

**Το πρόβλημα της θεϊκής κλάσης - Μορφή Συμπεριφοράς**

Συνηθισμένο σφάλμα προγραμματιστών διαδικασιακού τρόπου ανάπτυξης κατά τη μετάβαση στο αντικειμενοστρεφές μοντέλο:

Οι προγραμματιστές αυτοί προσπαθούν να δημιουργήσουν τον κεντρικό μηχανισμό ελέγχου που υπάρχει σε ένα διαδικασιακό σύστημα.

Το αποτέλεσμα είναι μία υπερβολικά ανεπτυγμένη κλάση που αναλαμβάνει τις περισσότερες αρμοδιότητες, αφήνοντας μικρό μέρος λεπτομερειακών εργασιών στις υπόλοιπες στοιχειώδεις κλάσεις.

Ένας αριθμός ευρετικών κανόνων μπορούν να βοηθήσουν για την αποφυγή δημιουργίας τέτοιων κλάσεων:

# Ευρετικοί Κανόνες

**Ευρετικός Κανόνας 1**

*Η λειτουργικότητα του συστήματος θα πρέπει να κατανέμεται οριζοντίως όσο το δυνατόν πιο ομοιόμορφα, δηλαδή, οι κλάσεις υψηλού επιπέδου σε ένα σχέδιο θα πρέπει να μοιράζονται ομοιόμορφα την εργασία που πρέπει να πραγματοποιηθεί.*

**Ευρετικός Κανόνας 2**

*Μή δημιουργείτε θεϊκές κλάσεις/αντικείμενα στο σύστημά σας. Να είστε ιδιαίτερα καχύποπτοι για κλάσεις των οποίων το όνομα περιλαμβάνει τους όρους* `Driver`, `Manager, System, Subsystem` *ή παρόμοιες έννοιες.*

**Ευρετικός Κανόνας 3**

*Να είστε ιδιαίτερα προσεκτικοί με τις κλάσεις οι οποίες έχουν πολλές μεθόδους πρόσβασης στη δημόσια διασύνδεσή τους. Η ύπαρξη πολλών υποδηλώνει ότι συσχετιζόμενα δεδομένα και συμπεριφορά δεν διατηρούνται μαζί σε ένα σημείο.*

**Ευρετικός Κανόνας 4**

*Να είστε ιδιαίτερα προσεκτικοί με τις κλάσεις που εμφανίζουν πολλή συμπεριφορά η οποία δεν σχετίζεται με επικοινωνία με άλλα αντικείμενα (noncommunicating behavior) δηλαδή μεθόδους οι οποίες δρουν σε ένα συγκεκριμένο υποσύνολο των μελών δεδομένων της κλάσης. Οι θεϊκές κλάσεις παρουσιάζουν συχνά πολλή συμπεριφορά τέτοιου τύπου.*

# Ευρετικοί Κανόνες

Η ύπαρξη θεϊκών κλάσεων σε ένα σύστημα, συνεπάγεται την εμφάνιση πολλών μεθόδων `get()` και `set()` στις υπόλοιπες κλάσεις.

Η αναγκαιότητα τέτοιων μεθόδων οφείλεται σε δύο λόγους:

- Είτε η κλάση που χρησιμοποιεί τις `gets()` και `sets()` υλοποιεί μία στρατηγική (policy) μεταξύ δύο ή περισσοτέρων κλάσεων
- είτε βρίσκεται στα όρια μεταξύ ενός αντικειμενοστρεφούς μοντέλου και της γραφικής διασύνδεσης χρήστη

# Ευρετικοί Κανόνες

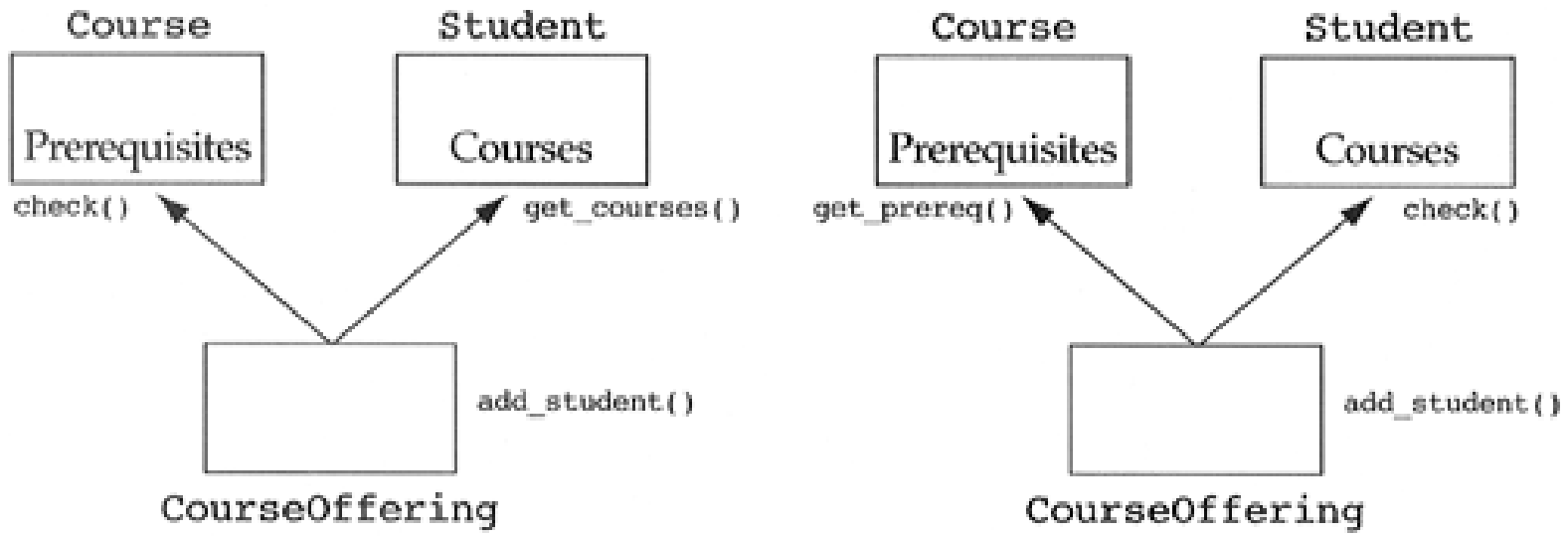
Παράδειγμα: μοντέλο ενός ακαδημαϊκού συστήματος όπου υπάρχουν μαθήματα (courses), προσφερόμενες τάξεις (course offerings) και φοιτητές (student).

Τα μαθήματα περιλαμβάνουν στατική πληροφορία (περιγραφή, αριθμός διδακτικών μονάδων, προαπαιτούμενα),

οι προσφερόμενες τάξεις διατηρούν στατική και δυναμική πληροφορία (το προσφερόμενο μάθημα, αίθουσα, ώρες διδασκαλίας, λίστα ατόμων που παρακολουθούν),

ενώ η κλάση φοιτητής περιέχει πληροφορία που αφορά τους φοιτητές (ονοματεπώνυμο, αριθμός μητρώου, λίστα μαθημάτων που έχει περάσει επιτυχώς κλπ).

# Ευρετικοί Κανόνες




Μία λειτουργία της "Προσφερόμενης Τάξης" αφορά την προσθήκη ενός φοιτητή.

Πριν από την προσθήκη, η κλάση θα πρέπει να ελέγξει ότι ο συγκεκριμένος φοιτητής που της δίνεται έχει περάσει τα προαπαιτούμενα. Πώς γίνεται αυτός ο έλεγχος;

Από τη μία ο φοιτητής γνωρίζει ποιά μαθήματα έχει περάσει αλλά από την άλλη το κάθε μάθημα γνωρίζει τα προαπαιτούμενά του.

Κατ' ελάχιστον, απαιτείται η λήψη της πληροφορίας από τη μία κλάση και η μεταβίβασή της στην άλλη. Οι δύο εναλλακτικές φαίνονται στο σχήμα. Σε κάθε σχεδίαση απαιτείται η ύπαρξη μιας μεθόδου πρόσβασης `get()`.

# Ευρετικοί Κανόνες

Jacobson's Objectory method: και στις δύο περιπτώσεις παραβιάζεται ένας κανόνας "η στρατηγική δεν θα πρέπει να τοποθετείται μέσα σε κλάσεις του προβλήματος οι οποίες εμπλέκονται στη λήψη της απόφασης".

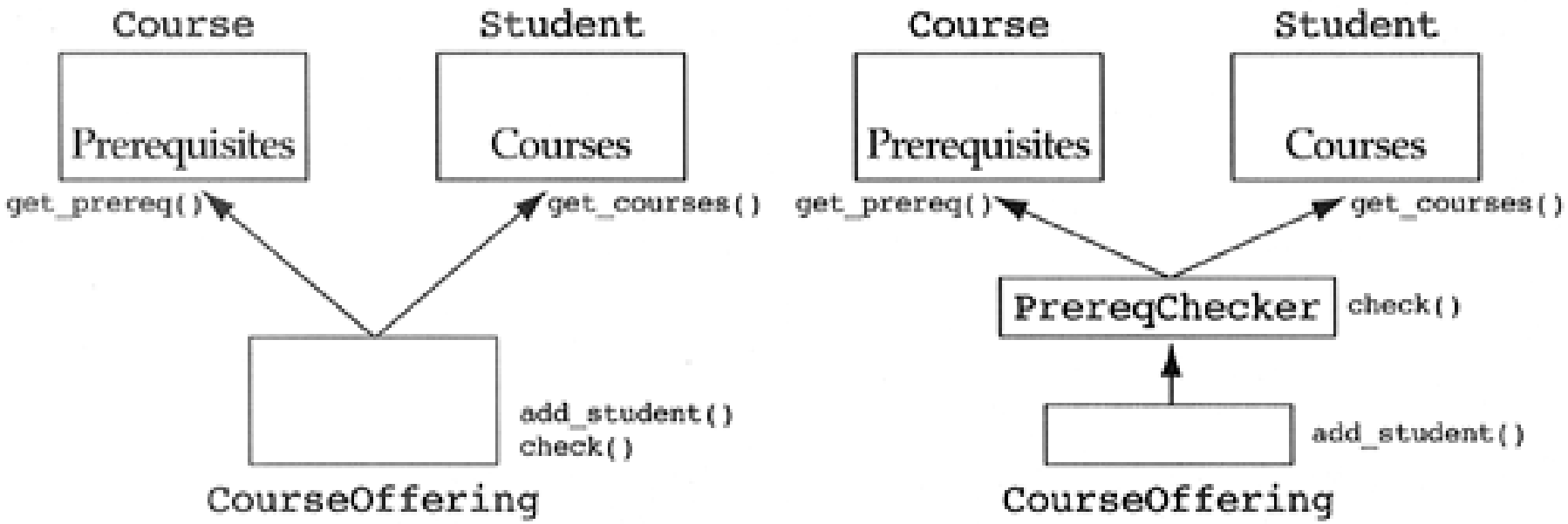
Υποστηρίζεται, ότι οι κλάσεις του πεδίου αχρηστεύονται γιατί συνδέονται με το πεδίο του προβλήματος που θέτει τη στρατηγική.

Σύμφωνα με τη μεθοδολογία, η λύση συνίσταται:

• είτε στο να λάβει η κλάση CourseOffering και τη λίστα των προαπαιτούμενων και τη λίστα των μαθημάτων που έχουν εξεταστεί επιτυχώς και να αποφασίσει η ίδια,

• είτε να δημιουργηθεί μία ξεχωριστή κλάση `ΈλεγκτηςΠροαπαιτουμένων` για την εκτέλεση αυτής της λειτουργίας

# Ευρετικοί Κανόνες




Η τελευταία κλάση είναι μία ειδική περίπτωση κλάσης Ελεγκτή (Controller class) που περιέχει μόνο συμπεριφορά.

Παρόλο που μία κλάση ελεγκτής καθιστά τις υπόλοιπες κλάσεις περισσότερο επαναχρησιμοποιήσιμες, μία τέτοια κλάση δεν έχει λόγο ύπαρξης.

Τι θα έκαναν σε μία τέτοια περίπτωση οι κλάσεις Φοιτητής και Μάθημα; Σε μία τέτοια σχεδίαση, η κλάση ελεγκτής πραγματοποιεί τα πάντα ενώ οι υπόλοιπες περιλαμβάνουν μόνο συναρτήσεις get() και set().

Στην ουσία καταργείται το αντικειμενοστρεφές μοντέλο διαχωρίζοντας δεδομένα και συμπεριφορά.

# Ευρετικοί Κανόνες

Υπάρχει μία περίπτωση όπου είναι αναγκαία η ύπαρξη μεθόδων πρόσβασης και αναφέρεται σε αρχιτεκτονικές όπου ένα αντικειμενοστρεφές μοντέλο αλληλεπιδρά με τη γραφική διασύνδεση χρήστη.
Ο ευρετικός κανόνας εδώ είναι ότι το μοντέλο θα πρέπει να είναι ανεξάρτητο από τη γραφική διασύνδεση, παρόλο που η τελευταία θα πρέπει να επιτρέπει την παρακολούθηση και τροποποίηση ορισμένων από τα δεδομένα του μοντέλου.
Σε μία τέτοια αρχιτεκτονική, η γραφική διασύνδεση χρειάζεται την ύπαρξη μεθόδων get() και set() στις κλάσεις του μοντέλου.
Σε τέτοια συστήματα οι κλάσεις του μοντέλου σπανίως έχουν ενδιαφέρουσα συμπεριφορά. Σε συστήματα διαχείρισης δεδομένων για παράδειγμα, οι κλάσεις επικοινωνούν με ένα DBMS, ανακτούν εγγραφές, τις προωθούν, τις τροποποιούν και επανεγγράφουν στη βάση.

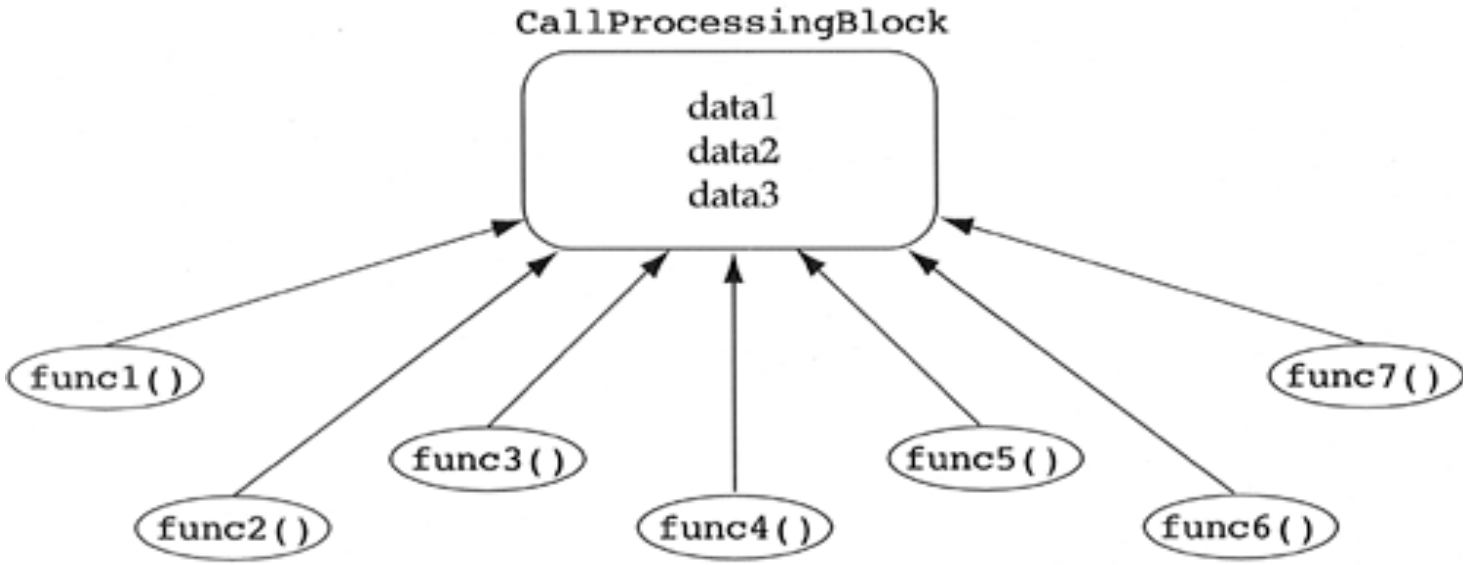
**Ευρετικός Κανόνας 5**

*Σε εφαρμογές που αποτελούνται από ένα αντικειμενοστρεφές μοντέλο που αλληλεπιδρά με διασύνδεση χρήστη, το μοντέλο θα πρέπει να είναι ανεξάρτητο από τη διασύνδεση. Η διασύνδεση θα πρέπει να εξαρτάται από το μοντέλο.*

# Ευρετικοί Κανόνες

**Πρόβλημα της θεϊκής κλάσης - Μορφή Δεδομένων**

Το πρόβλημα της περίπτωσης αυτής εμφανίζεται συχνά κατά τη μετατροπή ενός υπάρχοντος συστήματος λογισμικού (legacy system) σε μία νέα αντικειμενοστρεφή σχεδίαση.




Υπάρχον σύστημα επεξεργασίας κλήσεων

# Ευρετικοί Κανόνες

Κατά τη μετατροπή του συστήματος η ομάδα ανάπτυξης αποφασίζει να ενσωματώσει τη δομή δεδομένων σε μία κλάση, προσθέτοντας σε αυτή μεθόδους `get()` και `set()` για όλα τα μέλη δεδομένων.

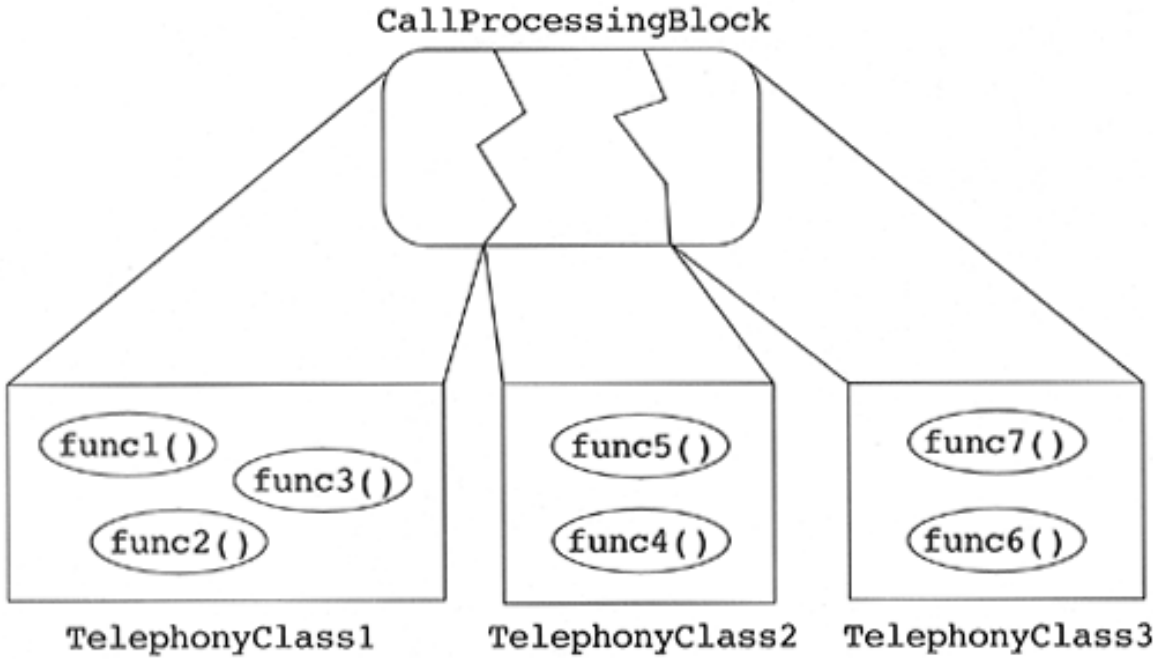
Οι συναρτήσεις επεξεργασίας μετατρέπονται σε κλάσεις - ελεγκτές οι οποίες χρησιμοποιούν τις μεθόδους πρόσβασης και τροποποίησης της κλάσης των δεδομένων.

Το νέο σύστημα υπερτερεί από το αρχικό. Ωστόσο, η σχεδίασή του δεν είναι καλής ποιότητας.

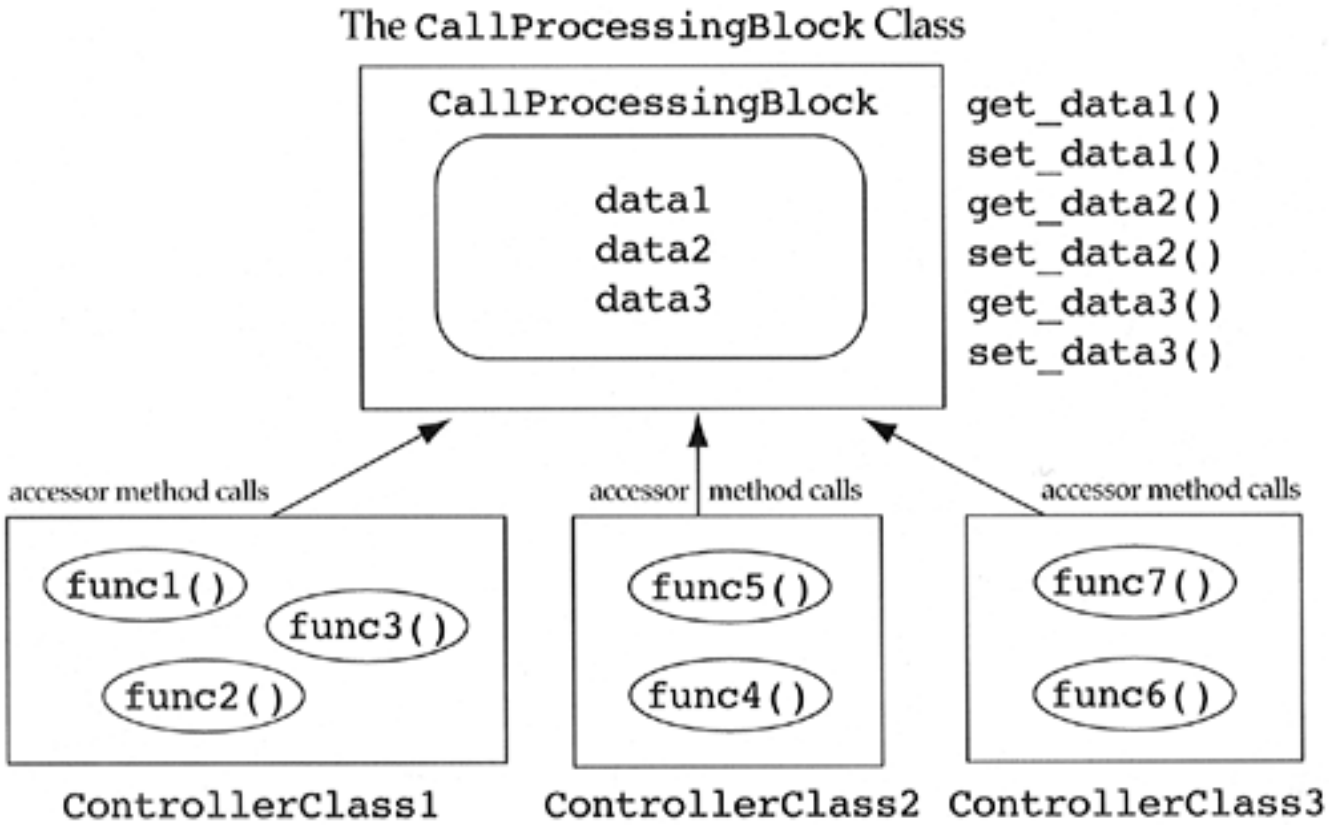
Η κλάση `CallProcessingBlock` είναι μία θεϊκή κλάση που διατηρεί όλα τα δεδομένα, ενώ οι κλάσεις-ελεγκτές παραβιάζουν την αρχή της ενσωμάτωσης (συγκέντρωσης δεδομένων και μεθόδων στο ίδιο σημείο).

# Ευρετικοί Κανόνες

Η προηγούμενη αντιμετώπιση είναι η ευκολότερη λύση.
Ορθή σχεδίαση: διαχωρισμός της κλάσης που διατηρεί τα δεδομένα σε τμήματα, βάσει των συναρτήσεων επεξεργασίας.



# Ευρετικοί Κανόνες




Αντικειμενοστρεφές σύστημα χαμηλής ποιότητας

# Ευρετικοί Κανόνες

Πρόβλημα Πολλαπλασιασμού των Κλάσεων (Proliferation of Classes)

3 ευρετικοί κανόνες των οποίων η παραβίαση οδηγεί στο ανωτέρω πρόβλημα

***Διαγράψτε τις άσχετες κλάσεις από τη σχεδίαση***

Μία άσχετη κλάση είναι αυτή που η συμπεριφορά της δεν έχει κανένα νόημα για το πεδίο του προβλήματος. Συνήθως πρόκειται για κλάσεις που δεν έχουν άλλες λειτουργίες εκτός από set, get και print.

Κατά τη διαγραφή τέτοιων κλάσεων δεν απομακρύνεται και η πληροφορία (μέλη δεδομένων) που διατηρούν, αλλά ενσωματώνεται ως ιδιότητα σε άλλες κλάσεις.

# Ευρετικοί Κανόνες

***Διαγράψτε κλάσεις που βρίσκονται εκτός συστήματος***

Κλασσικό παράδειγμα μία αφαίρεση η οποία αποστέλλει μηνύματα προς το σύστημα αλλά δεν λαμβάνει κανένα μήνυμα από οποιαδήποτε κλάση.

Παράδειγμα αισθητήρων: φυσικές διατάξεις ή συσκευές που παρέχουν πληροφορία χρήσιμη για το σύστημα.

Ωστόσο δεν ενδιαφέρουν συνήθως οι μέθοδοι συλλογής, επεξεργασίας και καταγραφής των δεδομένων (πρόκειται για λειτουργίες στο επίπεδο του hardware) και κατά συνέπεια δεν υπάρχει λόγος μοντελοποίησής τους ως μέθοδοι κλάσεων.

Παράδειγμα ATM. Το ερώτημα που τίθεται είναι συνήθως "Ο πελάτης του συστήματος είναι κλάση αφού στέλνει μηνύματα προς το ATM;". Ωστόσο μόνο η αποστολή μηνυμάτων δεν συνιστά συμπεριφορά στο πεδίο του προβλήματος. Ο πελάτης δεν λαμβάνει κανένα μήνυμα το οποίο να ενεργοποιεί λειτουργίες.

# Ευρετικοί Κανόνες

*Μην μετατρέπετε λειτουργίες σε κλάσεις. Να είστε καχύποπτοι για κλάσεις των οποίων το όνομα είναι ένα ρήμα, ή παράγωγο ρήματος, ειδικά αυτές που παρουσιάζουν μόνο μία συμπεριφορά με νόημα (εξαιρούνται μέθοδοι set, get και print).*

Κλάσεις που εμφανίζονται σε μία σχεδίαση να έχουν μόνο μία ουσιαστική λειτουργία θα πρέπει να εξεταστούν με προσοχή. Οι λειτουργίες αυτές (μαζί με τυχόν δεδομένα που χρησιμοποιούν) ενδεχομένως να μπορούν να ενσωματωθούν σε κάποια κλάση που απομένει να εντοπιστεί.

# Ευρετικοί Κανόνες

*Κλάσεις - Διαμεσολαβητές*

(Meiler Page-Jones - OOPSLA'87):

"Σε μία αντικειμενοστρεφή φάρμα υπάρχει μία αντικειμενοστρεφής αγελάδα με αντικειμενοστρεφές γάλα. Θα πρέπει η αντικειμενοστρεφής αγελάδα να στείλει στο αντικειμενοστρεφές γάλα το μήνυμα "παρήγαγε τον εαυτό σου" ή θα πρέπει το αντικειμενοστρεφές γάλα να στείλει το μήνυμα στην αντικειμενοστρεφή αγελάδα "άρμεξε τον εαυτό σου";

Παρόλο που το παράδειγμα μπορεί να φαίνεται αστείο, εμφανίζεται διαρκώς, όπως για παράδειγμα στο μοντέλο μιας βιβλιοθήκης: Δεν είναι εύκολο να αποφασίσει κανείς αν ένα βιβλίο αποστέλλει μήνυμα "αρχειοθέτησης του" στη βιβλιοθήκη, ή εάν το μήνυμα αποστέλλεται από τη βιβλιοθήκη στο βιβλίο.

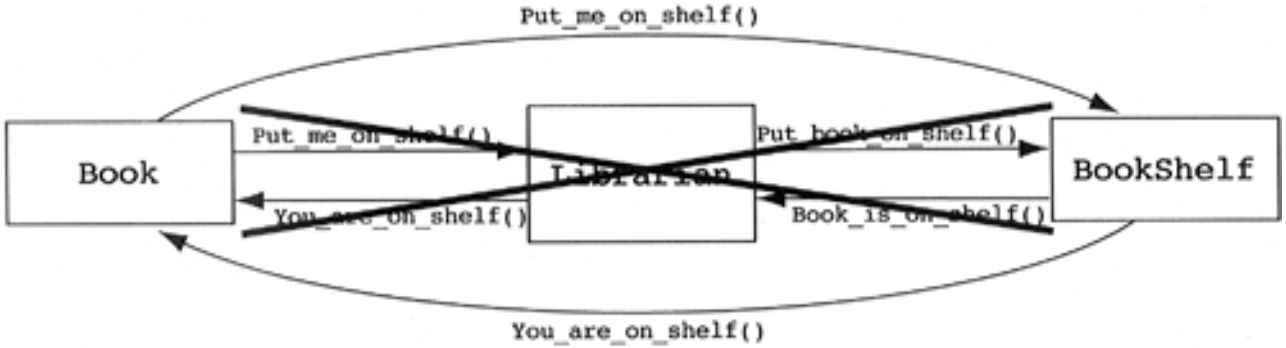
Το πρόβλημα και στις δύο περιπτώσεις είναι κοινό. Υπάρχει ένα κοινό στοιχείο - κλειδί το οποίο λείπει. Στο πρώτο παράδειγμα είναι ένας αγρότης και στο δεύτερο ένας βιβλιοθηκονόμος. Είναι οι αφαιρέσεις αυτές κλάσεις;

# Ευρετικοί Κανόνες

Κατά τη διάρκεια της ανάλυσης, τέτοιες **κλάσεις-διαμεσολαβητές** μοντελοποιούνται καθώς αποτελούν μέρος του πραγματικού κόσμου.


Ωστόσο, κατά τη διάρκεια της σχεδίασης, λαμβάνοντας υπόψη και τους διάφορους ευρετικούς κανόνες, το μοντέλο του πραγματικού κόσμου τροποποιείται.

Τίθεται για παράδειγμα το ερώτημα "ποιά η χρήση ενός βιβλιοθηκονόμου στο σύστημα"; Αν πρόκειται μόνο για μία κλάση η οποία λαμβάνει μηνύματα και τα ξαναστέλνει, τότε γιατί να μην διαγραφεί από τη σχεδίαση μειώνοντας τον αριθμό των κλάσεων και συσχετίσεων;



# Ευρετικοί Κανόνες

*Κλάσεις - διαμεσολαβητές εντοπίζονται συχνά κατά τη διάρκεια της ανάλυσης μιας εφαρμογής. Κατά τη σχεδίαση, πολλοί διαμεσολαβητές αποδεικνύονται άσχετοι και θα πρέπει να διαγράφονται από το μοντέλο.*

Αναφορικά με το ποιά κλάση πρέπει να αποστέλλει το μήνυμα, συνήθως σε λογικό επίπεδο δεν υπάρχει απάντηση, καθώς καμία λύση δεν μειονεκτεί έναντι της άλλης. Ορισμένες φορές σε επίπεδο φυσικής σχεδίασης υπάρχουν μικρά πλεονεκτήματα.

# Ευρετικοί Κανόνες

## ΚΛΗΡΟΝΟΜΙΚΟΤΗΤΑ – Γενίκευση – Εξειδίκευση

- Θεμελιώδης μηχανισμός στην ανάλυση και σχεδίαση συστημάτων
- χρησιμοποιείται:

  α) για να εκφραστούν τα κοινά στοιχεία (συμπεριφορά και ιδιότητες) μεταξύ δύο ή περισσοτέρων κλάσεων, περίπτωση στην οποία αναφερόμαστε και ως **γενίκευση**,

  β) για να προσδιοριστεί ότι μία κλάση αποτελεί ειδικότερη κατηγορία μιας άλλης κλάσης, περίπτωση στην οποία αναφερόμαστε και ως **εξειδίκευση**.

Δεν είναι το ίδιο ?

# Ευρετικοί Κανόνες

Γενικεύσεις εντοπίζονται συνήθως στην πρώτη έκδοση ενός συστήματος:

Τα μέλη της ομάδας ανάπτυξης ανακαλύπτουν κατά τη σχεδίαση ότι ορισμένες κλάσεις έχουν κοινή διασύνδεση ή κοινά δεδομένα. Αυτή η κοινή πληροφορία ενσωματώνεται σε μία γενικότερη υπερκλάση.

Σχέσεις εξειδίκευσης εντοπίζονται κατά την ανάπτυξη μεταγενέστερων εκδόσεων

Καθώς το λογισμικό εξελίσσεται για να συμπεριλάβει επιπρόσθετη λειτουργικότητα, ορισμένες λειτουργίες/δεδομένα απαιτούν ειδικό χειρισμό που δεν μπορεί να ενταχθεί στις υπάρχουσες κλάσεις.

# Ευρετικοί Κανόνες

Σύνηθες πρόβλημα κατανόησης: σύγχυση μεταξύ κληρονομικότητας και περιεκτικότητας.

Αν μία κλάση Β (ορολογία: **παράγωγη κλάση**, **υποκλάση**, **απόγονος** ή **κλάση παιδί**) κληρονομεί μία κλάση Α (ορολογία: **βασική κλάση**, **υπερκλάση**, **πρόγονος** ή **γονική κλάση**), αποκτά ένα αντίγραφο των δεδομένων της Α. Ωστόσο, αν η κληρονομικότητα σήμαινε μόνο την αντιγραφή των δεδομένων της βασικής κλάσης στην παράγωγη κλάση, αυτή η σχέση θα ήταν άχρηστη. Η σχέση της περιεκτικότητας έχει ακριβώς την ίδια σημασιολογία.

Ωστόσο, με τη σχέση της περιεκτικότητας η περιέχουσα κλάση δεν αποκτά πρόσβαση στις λειτουργίες της περιεχόμενης κλάσης.

Ειδοποιός διαφορά στην κληρονομικότητα: Η παράγωγη κλάση διατηρεί τη λειτουργικότητα της βασικής κλάσης

# Ευρετικοί Κανόνες

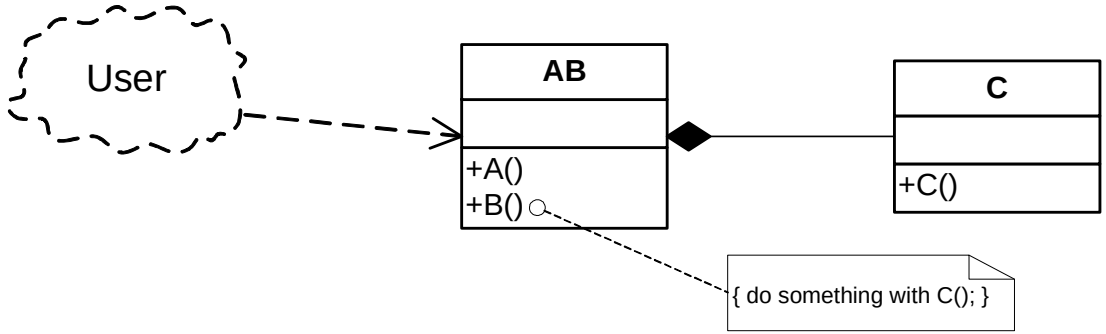
*Η κληρονομικότητα θα πρέπει να χρησιμοποιείται μόνο για τη μοντελοποίηση μιας ιεραρχίας εξειδικεύσεων*

Περιεκτικότητα: σχεδίαση black-box

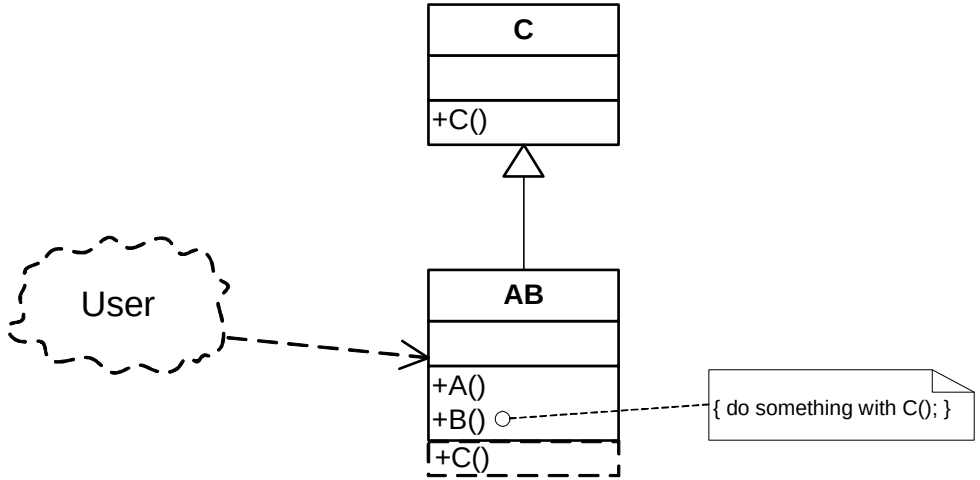
Κληρονομικότητα: σχεδίαση white-box καθώς ο χρήστης της παράγωγης κλάσης, για να την χρησιμοποιήσει, πρέπει να γνωρίζει τα δεδομένα και τις λειτουργίες της βασικής κλάσης.

Αν χρησιμοποιούμε κληρονομικότητα εκεί όπου θα μπορούσαμε να χρησιμοποιήσουμε περιεκτικότητα, αποκαλύπτουμε χωρίς λόγο λεπτομέρειες υλοποίησης στους χρήστες μιας κλάσης.

# Ευρετικοί Κανόνες




Στη διασύνδεση της κλάσης AB η μέθοδος C δεν είναι "ορατή" και κατά συνέπεια ο χρήστης δεν μπορεί να στείλει μήνυμα C στην AB, ούτε φυσικά να παρακάμψει τη λειτουργία B για την επίτευξη του ίδιου σκοπού.




Στη διασύνδεση της κλάσης AB η μέθοδος C είναι "ορατή" ("Άνοιγμα της σχεδίασης")

# Ευρετικοί Κανόνες

*Οι παράγωγες κλάσεις πρέπει να έχουν γνώση των βασικών τους κλάσεων εξ'ορισμού, αλλά οι βασικές κλάσεις δεν θα πρέπει να γνωρίζουν οτιδήποτε για τις παράγωγες κλάσεις.*

Αν οι βασικές κλάσεις γνωρίζουν τις παράγωγές τους κλάσεις, τότε υπονοείται ότι κατά την προσθήκη μιας νέας παράγωγης κλάσης, ο κώδικας της βασικής κλάσης θα πρέπει να τροποποιηθεί. Αυτό αποτελεί παραβίαση της αρχής της αντιστροφής των εξαρτήσεων καθώς η αφαίρεση (βασική κλάση) θα εξαρτάται από την υλοποίηση (παράγωγες κλάσεις).

Βάθος ιεραρχιών κληρονομικότητας

*Θεωρητικά, οι ιεραρχίες κληρονομικότητας πρέπει να έχουν μεγάλο βάθος - όσο μεγαλύτερο βάθος τόσο καλύτερα.*

Θεωρητικό κίνητρο: με την ύπαρξη μιας μεγάλης ταξινομίας αφαιρέσεων, μία νέα παράγωγη κλάση μπορεί να διασχίσει την ιεραρχία κληρονομώντας την πιο εκλεπτυσμένη αφαίρεση (Πετρελαιοκίνητο_Αυτοκίνητο_4x4)

# Ευρετικοί Κανόνες

*Στην πράξη, οι ιεραρχίες κληρονομικότητας δεν θα πρέπει να έχουν μεγαλύτερο βάθος από ότι ένας μέσος άνθρωπος μπορεί να κρατήσει στη μνήμη βραχείας διάρκειας. Μία συνηθισμένη τιμή για αυτό το βάθος είναι έξι.*

Κακή εμπειρία από έργα λογισμικού με υπερβολικά μεγάλες τιμές (12 έως 17 επίπεδα).

Επιπρόσθετα, ο σχεδιαστής του συστήματος θα πρέπει να είναι σε θέση να γνωρίζει για μία δεδομένη κλάση, το σύνολο των μεθόδων και ιδιοτήτων που κληρονομεί για να μπορεί να τη χρησιμοποιήσει κατάλληλα.

*Όλες οι αφηρημένες κλάσεις πρέπει να είναι βασικές κλάσεις*

Αν από μία κλάση δεν μπορούν να κατασκευαστούν στιγμιότυπα, τότε η κλάση αυτή θα πρέπει να κληρονομείται από κλάσεις που θα δημιουργούν αντικείμενα. Αν δεν ισχύει αυτό, τότε η λειτουργικότητα της αφηρημένης κλάσης δεν μπορεί να προσπελαστεί από πουθενά.

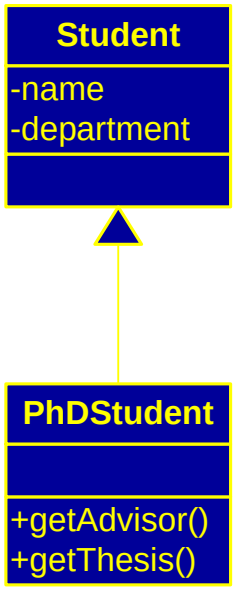
# Ευρετικοί Κανόνες

*Όλες οι βασικές κλάσεις πρέπει να είναι αφηρημένες*

Άλλη διατύπωση: "Καμία κλάση δεν πρέπει να κληρονομεί από κάποια συγκεκριμένη κλάση"

Παράδειγμα: Θεωρούμε ένα σύστημα ενός πανεπιστημίου όπου μοντελοποιούμε τις έννοιες ενός φοιτητή (Student) και ενός διδακτορικού φοιτητή (PhDStudent)

Απολύτως φυσιολογικά, ένας διδακτορικός φοιτητής αποτελεί ειδική περίπτωση ενός φοιτητή και αποφασίζεται η μοντελοποίηση των δύο εννοιών με κληρονομικότητα



# Ευρετικοί Κανόνες

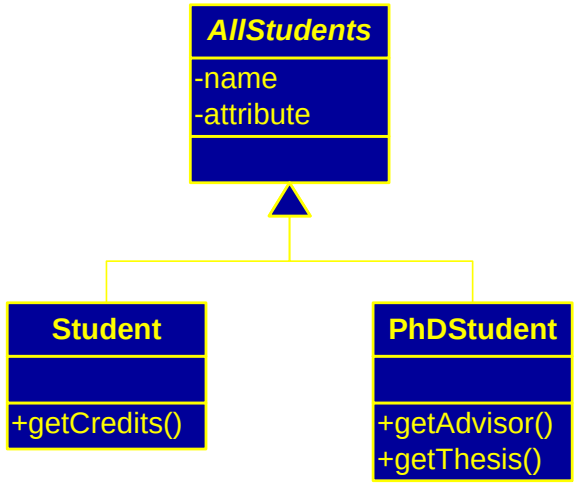
Νέα απαίτηση: ζητείται η προσθήκη σε κάθε φοιτητή μιας μεθόδου `getCredits()`.

Μπορεί να προστεθεί η συγκεκριμένη λειτουργικότητα στην Student χωρίς να προστεθεί και στην PhDStudent, την οποία όμως δεν αφορά;

Αυτό ακριβώς είναι το πρόβλημα συντηρησιμότητας που δημιουργείται όταν μία συγκεκριμένη κλάση κληρονομεί μία άλλη συγκεκριμένη κλάση.

Το ανωτέρω (σημαντικό σε πολλές περιπτώσεις πρόβλημα) μπορεί να αποφευχθεί αν θεωρήσουμε ότι και οι δύο κλάσεις έχουν κάτι κοινό. Η κοινή πληροφορία ενσωματώνεται σε μία αφηρημένη βασική κλάση (AllStudents)

# Ευρετικοί Κανόνες




Επίλυση προβλήματος ιδιαίτερα επίπονη.

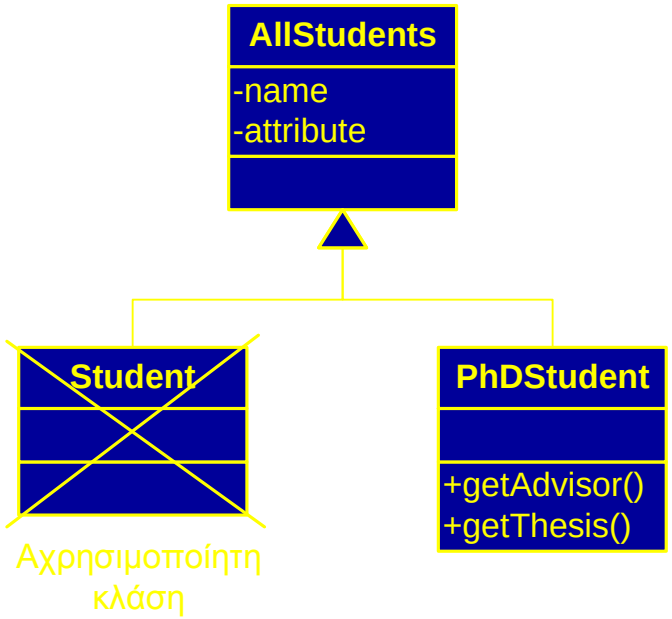
Ενώ ο κώδικας της κλάσης Student θα έπρεπε ούτως ή άλλως να τροποποιηθεί (για την προσθήκη της νέας μεθόδου), πλέον θα πρέπει να τροποποιηθεί και ο κώδικας της κλάσης PhDStudent, να επαναμεταγλωττιστεί και να μεταγλωττιστούν και ελεγχθούν εκ νέου όλες οι μονάδες λογισμικού που χρησιμοποιούν την PhDStudent.

Ακόμα και αν κάποιος αποφάσισε να μην αλλάξει τη σχεδίαση, εκφυλίζοντας απλά την υλοποίηση της μεθόδου getCredits() στην κλάση PhDStudent, θα έπρεπε να αλλάξει τον κώδικα της, χωρίς να ελπίζει ότι τέτοια προβλήματα δεν θα δημιουργηθούν και στο μέλλον.

# Ευρετικοί Κανόνες

Οι ευρετικοί κανόνες δεν ισχύουν απόλυτα

Έστω ότι δεν υπήρχαν τέτοιες αλλαγές στις απαιτήσεις. Τότε η "βελτιωμένη σχεδίαση" θα οδηγούσε σε μία παράγωγη κλάση η οποία δεν θα είχε καμία διαφορά από τη βασική της κλάση - κατά συνέπεια θα ήταν κενή



# Ευρετικοί Κανόνες

Κλάσεις υποκλάσεις και ρόλοι

• *Να είστε σίγουροι ότι οι αφαιρέσεις που μοντελοποιείτε είναι κλάσεις και όχι ρόλοι που έχουν κάποια αντικείμενα*

# Ευρετικοί Κανόνες

•Ανάλυση περιπτώσεων βάσει της τιμής μιας ιδιότητας πρέπει συνύθως να αποφεύγεται. Η κλάση που περιλαμβάνεται στους ελέγχους θα πρέπει να αποσυντεθεί σε μια ιεραρχία κλάσεων όπου κάθε τιμή της ιδιότητας αντιστοιχίζεται σε μια παράγωγη κλάση.

## Ευρετικοί Κανόνες

- Δεν θα πρέπει να μοντελοποιείται η δυναμική σημασιολογία μιας κλάσης μέσω κληρονομικότητας.