Μεθοδολογία Προγραμματισμού

Καλή και κακή σχεδίαση

Νικόλαος Πεταλίδης

Τμήμα Μηχανικών Πληροφορικής και Επικοινωνιών
Διεθνές Πανεπιστήμιο της Ελλάδος

Εισαγωγή
Εαρινό Εξάμηνο

Ανεξαρτησία συστατικών

- Είπαμε πολλές φορές ότι στη σχεδίαση μιας κλάσης πρέπει να προσέξουμε να είναι όσο το δυνατόν περισσότερο ανεξάρτητη.
- Τι σημαίνει όμως αυτό;
- Πότε μπορούμε να πούμε ότι έχουμε τελικά μια «καλή» κλάση;

Σύζευξη και συνεκτικότητα

Δίνουν έμφαση στη δομή του συστήματος ανεξάρτητα από το πως μεταβάλλεται στο χρόνο

- Στη σχεδίαση τα συστατικά θα πρέπει να είναι όσο το δυνατόν ανεξάρτητα το ένα από το άλλο
- Για να αναγνωρίσουμε και να μετρήσουμε το βαθμό ανεξαρτησίας μεταξύ των συστατικών χρησιμοποιούμε δύο έννοιες:
  - Σύζευξη
  - Συνεκτικότητα

Είδη σύζευξης

- Σύζευξη περιεχομένου (κακή)
- Σύζευξη κοινών δεδομένων
- Σύζευξη ελέγχου
- Σύζευξη αντιγράφου
- Σύζευξη δεδομένων
- Μη Σύζευξη (καλή)

Σύζευξη περιεχομένου

- Όταν το ένα συστατικό α αλλάζει το περιεχόμενο ενός άλλου συστατικού β

  Παράδειγμα Μια συνάρτηση Α αλλάζει μια εντολή που βρίσκεται σε μια συνάρτηση Β.

  Παράδειγμα Μια συνάρτηση Α περιέχει μια εντολή goto σε μια συνάρτηση Β

Προβλήματα στη σύζευξη περιεχομένου

- Σχεδόν οποιαδήποτε αλλαγή στο B, ακόμα και αλλαγή του compiler που χρησιμοποιείται προϋποθέτει αλλαγή και στο A.
- Αδύνατον να χρησιμοποιηθεί το A χωρίς να χρησιμοποιηθεί και το B

Σύζευξη κοινών δεδομένων

- Όταν δύο συστατικά μοιράζονται κοινά δεδομένα

Παράδειγμα Η συνάρτηση `functionA()` και η συνάρτηση `functionB()` μπορούν ΚΑΙ να διαβάσουν ΚΑΙ να αλλάξουν μια καθολική μεταβλητή (global variable)

Παράδειγμα Δύο διαφορετικές συναρτήσεις έχουν πρόσβαση στην ίδια βάση δεδομένων και μπορούν να αλλάξουν ΤΗΝ ΙΔΙΑ ΕΓΓΡΑΦΗ

Παράδειγμα Δύο διαφορετικές κλάσεις έχουν πρόσβαση (ανάγνωση/εγγραφή) στο ίδιο public attribute μιας κλάσης

Προβλήματα στη σύζευξη κοινών δεδομένων

- Είναι αντίθετη στις αρχές του δομημένου προγραμματισμού
- Ο κώδικας που παράγεται δε διαβάζεται
- Πότε σταματά ο βρόγχος;

```
while (global == 0) {
  if (x > 25) {
    functionA();
  } else {
    functionB();
  }
```

Προβλήματα στη σύζευξη κοινών δεδομένων (συνχ.)

- Τα συστατικά μπορούν να έχουν "παρενέργειες". Αυτό τα κάνει δυσανάγνωστα αφού πρέπει κανείς να διαβάσει ολόκληρο το συστατικό με προσοχή να δει αν αλλάζει κάποια καθολική μεταβλητή.
- Αν αλλάξει η καθολική μεταβλητή πρέπει να αλλάξει και κάθε συστατικό που τη χρησιμοποιεί
- Η επαναχρησιμοποίηση είναι δύσκολη. Πρέπει να υπάρχει το ίδιο σύνολο καθολικών μεταβλητών όποτε επαναχρησιμοποιείται το συστατικό
- Κάθε συστατικό έχει πρόσβαση σε περισσότερα δεδομένα από όσα χρειάζεται. Αυτό κάνει ευκολότερη τη δημιουργία λαθών από απροσεξία

Πλεονεκτήματα στη σύζευξη κοινών δεδομένων

- Όταν ένα συστατικό έχει ένα τεράστιο αριθμό μεταβλητών και πρέπει να αρχικοποιηθούν, αντί να περαστούν σαν παράμετροι όποτε χρησιμοποιείται το συστατικό, ορίζονται σαν καθολικές μεταβλητές και αρχικοποιούνται έτσι πιο γρήγορα

Σύζευξη ελέγχου

- Όταν το ένα συστατικό περνά παραμέτρους στο άλλο που ελέγχουν τη συμπεριφορά του του ή μεταξύ ενός συστατικού και των εσωτερικών τμημάτων του
- Συνήθως εμφανίζεται σε συστατικά που παρουσιάζουν λογική συνεκτικότητα (δείτε τις παρακάτω διαφάνειες)

Παράδειγμα σύζευξης ελέγχου

- Η συνάρτηση `progressmeter(int n)` στο `netbsdsrc/usr.bin/ftp/fetch.c` παίρνει τρία ορίσματα
  - -1 αρχικοποιεί το μετρητή προόδου και τον ρυθμίζει για μια ενημέρωση/δευτερόλεπτο
  - 0 ενημερώνει την προβολή του μετρητή προόδου
  - 1 σταματά την ενημέρωση του μετρητή.
- Όποιος καλεί αυτή τη συνάρτηση "ελέγχει" και αποφασίζει ποια λειτουργία θα εκτελέσει

Προβλήματα στη σύζευξη ελέγχου

- Τα συστατικά δεν είναι ανεξάρτητα.
- Το συστατικό που καλείται πρέπει να ξέρει την εσωτερική δομή του συστατικού που το καλεί
- Επηρεάζει την επαναχρησιμοποίηση

Σύζευξη αντιγράφου

- Πολλές γλώσσες επιτρέπουν μόνο απλές μεταβλητές σαν παραμέτρους σε συναρτήσεις ή μεθόδους
  - Το όνομα ενός υπαλλήλου
  - Ο κωδικός ενός προϊόντος
- Αλλά πολλές γλώσσες επιτρέπουν ολόκληρες δομές
  - Το struct υπάλληλος
  - Το struct υπάλληλος
- Σύζευξη αντιγράφου έχουμε όταν ένα συστατικό Α καλεί ένα συστατικό Β και περνά μια ολόκληρη δομή σε αυτό σαν παράμετρο και επιπλέον το Β χρειάζεται να ξέρει μόνο συγκεκριμένα πεδία από αυτή τη δομή

Παράδειγμα σύζευξης αντιγράφου

- Σε μια συνάρτηση όπως αυτή; `function calculateWithHolding(employeeRecord R);` είναι δύσκολο να πείτε χωρίς να δείτε τον κώδικα ποια πεδία του R αλλάζουν ή χρησιμοποιούνται
- Θα έπρεπε να περνάν σαν παράμετροι *μόνο* συγκεκριμένα πεδία

Προβλήματα στη σύζευξη αντιγράφου

- Δεν είναι εύκολο να ξεκαθαρίσεις τι κάνει το κάθε συστατικό
- Η επαναχρησιμοποίηση δεν είναι εύκολη
- Τα συστατικά αποκτούν πρόσβαση σε δεδομένα που δε χρειάζονται

Προσοχή στη σύζευξη αντιγράφου

- Δεν είναι κακό να περνάτε ολόκληρες δομές αν αυτό είναι απαραίτητο:

```
M
invertatrix (Moriginalatrix, Minvertedatrix);
printRecord(warehouseRecord);
```

Σύζευξη δεδομένων

- Όταν το ένα συστατικό περνά δεδομένα στο άλλο
- Είναι η καλύτερη σύζευξη που μπορείτε να πετύχετε

```
displayTimeOfArrival (flightNumber);
computeProduct (firstNumber, secondNumber);
getJobWithHighestPriority (jobQueue);
```

Αντικειμενοστραφής προγραμματισμός και σύζευξη

- Ένα από τα πλεονεκτήματα των αντικειμενοστραφών συστημάτων είναι ότι παρουσιάζουν μικρή σύζευξη καθώς κάθε τάξη μπορεί και περιορίζει τις πράξεις που γίνονται στα δεδομένα της αλλά και περιέχει τους ορισμούς των πράξεων αυτών

Συνεκτικότητα

- Αναφέρεται στην εσωτερική συνοχή ενός συστατικού
- Όσο πιο μεγάλη συνοχή έχει ένα συστατικό τόσο περισσότερο σχετίζονται μεταξύ τους τα εσωτερικά του μέρη και εξυπηρετούν καλύτερα το συνολικό σκοπό του
- Ένα συστατικό με μεγάλη συνοχή δεν έχει παραπανίσια κομμάτια!

Είδη συνεκτικότητας

- Λειτουργική Καλή (υψηλή)
- Σειριακή
- Επικοινωνιακή
- Διαδικασιακή
- Χρονική
- Λογική
- Συμπτωματική Κακή (χαμηλή)

Συμπτωματική συνεκτικότητα

- Όταν τα μέρη δε σχετίζονται μεταξύ τους
- Παράδειγμα:

```
function doItAll(char *par1, int par2)
printNextLine();
ReverseString(par1)
par2 += 7;
return par2
```

- Συνήθως προκύπτει από εντολές του τύπου
  Γράψτε μια συνάρτηση που αποτελείται από 10-20 εντολές

Προβλήματα συμπτωματικής συνεκτικότητας

- Το συστατικό δε συντηρείται εύκολα γιατί είναι δυσνόητο
- Δε μπορεί να επαναχρησιμοποιηθεί
- Διορθώνεται όμως εύκολα: Φτιάξτε τόσα χωριστά συστατικά όσες και οι ενέργειες που έχετε

Λογική συνεκτικότητα

- Όταν διάφορα λογικά σχετιζόμενες συναρτήσεις ή δεδομένα τοποθετούνται στο ίδιο συστατικό
- Για παράδειγμα έχετε μια συνάρτηση η οποία γράφει δεδομένα σε συσκευές. Επιλέγετε τη συσκευή στην οποία θα γράψει περνώντας μια παράμετρο. Π.χ. Αν περαστεί ο κωδικός 1 γράφει σε δίσκο, Αν περαστεί ο κωδικός 2 γράφει στη μνήμη
- Στις περισσότερες από αυτές τις περιπτώσεις η συνάρτηση αυτή θα παίρνει και άλλες παραμέτρους που θα τους χρησιμοποιεί ανάλογα με τον κωδικό που περάσατε

Παράδειγμα λογικής συνεκτικότητας

```
function code = 7;
^^IwriteData(op code,
             dummy 1, // not used if code=7
             dummy 2,// not used if code=7
             dummy 3,// not used if code=7);
```

Παράδειγμα λογικής συνεκτικότητας

Το συστατικό στα δεξιά περιέχει κώδικα που διαχειρίζεται όλες τις εγγραφές/αναγνώσεις. Βάζετε μια καινούργια συσκευή. (πχ. Ένα νέο δίσκο). Πρέπει να αλλάξετε τα κομμάτια 1, 2, 3, 4, 7, 8. Τι θα γίνει όμως με τις συσκευές που ήδη χρησιμοποιούσαν αυτόν τον κώδικα;

1. Code for all input and output
2. Code for input only
3. Code for output only
4. Code for disk and tape I/O
5. Code for disk I/O
6. Code for tape I/O
7. Code for disk input
8. Code for disk output
9. Code for tape input
10. Code for tape output
⋮ ⋮ ⋮
37. Code for keyboard input

Προβλήματα λογικής συνεκτικότητας

- Οι διεπαφές δεν είναι ξεκάθαρες
- Δεν είναι ξεκάθαρο που βρίσκεται και τι κάνει κάθε κομμάτι
- Η επαναχρησιμοποίηση είναι δύσκολη

Χρονική συνεκτικότητα

- Όταν τα μέρη σχετίζονται με βάση το χρονισμό τους
- Ένα συστατικό που κάνει αρχικοποίηση

```
class initialize {
//Αρχικοποίησε μεταβλητέςπρογράμματος
//Άδειασε τονπίνακα Sales
//Διάβασε τηνπρώτηεγγραφήαπότονπίνακα Transactions
//Εμφάνισε μήνυμαστηνοθόνη
```

Παράδειγμα Χρονικής συνεκτικότητας

- Η κλάση Compiler της μηχανής Jasper

```
public class Compiler {
   public boolean compile() {
      if (javac == null) return true;
   }
    public void setJavaCompiler(JavaCompiler javac)
      this.javac = javac;
}
```

Επεξήγηση παραδείγματος

- Η μέθοδος `setJavaCompiler()` έχει ενταχθεί στην κλάση γιατί πρέπει να την καλέσουμε χρονικά πριν την μέθοδο `compile()`
- Τι θα γίνει αν ο χρήστης απλά ξεχάσει να την καλέσει; Υπάρχει κάτι που υποχρεώνει το χρήστη να το κάνει;

Προβλήματα χρονικής συνεκτικότητας

- Οι ενέργειες που εκτελούνται μέσα στο συστατικό έχουν μόνο μικρή σχέση μεταξύ τους αλλά συνήθως μεγάλη σχέση με ενέργειες που γίνονται σε άλλα συστατικά.
  - Αρχικοποιούμε τον πίνακα Sales αλλά άλλες ενέργειες όπως «ενημέρωσε τον Sales» «τύπωσε τον Sales» βρίσκονται σε άλλα συστατικά
  - Αν θέλουμε να αλλάξουμε τη δομή του πίνακα σε ποια συστατικά πρέπει να αλλάξουμε τον κώδικα;
- Δεν επιτρέπει την επαναχρησιμοποίηση

Διαδικασιακή συνεκτικότητα

- Διαφορετικές λειτουργίες ομαδοποιούνται σε ένα συστατικό επειδή πρέπει να εκτελεστούν με συγκεκριμένη σειρά
- Συνήθως οι λειτουργίες δεν έχουν κάποιο κοινό δεδομένο, αλλά απλά όταν ολοκληρώνεται η μία περνάει τον έλεγχο στην άλλη

```
function makeRepairs() {
read part number
update repair record on master file
}
```

Προβλήματα διαδικασιακής συνεκτικότητας

- Οι ενέργειες μέσα στο συστατικό είναι μόνο χαλαρά συνδεδεμένες μεταξύ τους.
- Δεν επιτρέπει την επαναχρησιμοποίηση

Πότε είναι απαραίτητη

- Όταν φτιάχνεται ένα συστατικό που ελέγχει (βρίσκεται πάνω από) όλα τα άλλα

Επικοινωνιακή συνεκτικότητα

- Ομαδοποιούνται συναρτήσεις που δρουν στο ίδιο σύνολο δεδομένων. Μοιάζει με τη χρονική αλλά εδώ κάθε λειτουργία δρα στα ίδια δεδομένα και η σειρά με την οποία γίνονται οι λειτουργίες δεν είναι τόσο σημαντική
- Παράδειγμα
  - Ενημέρωσε μια εγγραφή στη βάση ΚΑΙ γράψε τα στοιχεία της εγγραφής σε ένα αρχείο
  - Υπολόγισε νέες συντεταγμένες ΚΑΙ τύπωσέ τες στην οθόνη

Προβλήματα επικοινωνιακής συνεκτικότητας

- Οι ενέργειες είναι πιο στενά συνδεδεμένες αλλά όχι με το βέλτιστο τρόπο
- Δεν είναι εύκολη η επαναχρησιμοποίηση γιατί κάθε συστατικό τελικά εκτελεί πάνω από μια ενέργειες
- Καλό θα ήταν να χωριστεί σε περισσότερα συστατικά, ένα για κάθε ενέργεια

Ακολουθιακή συνεκτικότητα

- Όταν ένα συστατικό εκτελεί μια σειρά από ενέργειες, αλλά κάθε μία είναι ορισμένη ανεξάρτητα από την άλλη και όλες ενεργούν πάνω στο ίδιο σύνολο δεδομένων
- Παράδειγμα

In salesRegion.c

```
struct SalesRegion {}; //definition
initSalesRegion() //function
updateSalesRegion() //function
printSalesRegion() //function
```

Λειτουργική συνεκτικότητα

- Κάθε τμήμα επεξεργασίας είναι απαραίτητο για την απόδοση μιας μόνο λειτουργίας
- Παράδειγμα
  - Υπολόγισε τη θερμοκρασία του δωματίου
  - Υπολόγισε το εμβαδόν ενός τετραγώνου

Γιατί είναι καλή

- Γιατί η επαναχρησιμοποίηση είναι εύκολη
- Γιατί η εύρεση και διόρθωση λαθών είναι εύκολη
- Η λειτουργική συνεκτικότητα είναι ιδεατή και όχι πάντα δυνατή

Ανάλυση περιπτώσεων χρήσης

- Πολλές φορές είναι χρήσιμο προκειμένου να καταλήξουμε σε μια καλή σχεδίαση να ξεκινήσουμε αναλύοντας μια περίπτωση χρήσης με βάση γενικές «κλάσεις ανάλυσης»

Κλάση ανάλυσης

- Αποτελεί μια γενικευμένη κλάση η οποία κατά τη φάση της σχεδίασης θα αντιστοιχηθεί σε περισσότερες από μια τάξεις
- Επικεντρώνεται στις *λειτουργικές απαιτήσεις* και αναβάλει το χειρισμό των *μη λειτουργικών απαιτήσεων* στη φάση σχεδιασμού και υλοποίησης
- Δεν περιέχει μεθόδους. Το πως συμπεριφέρεται ορίζεται ουσιαστικά από τις υποχρεώσεις της
- Έχει χαρακτηριστικά (τα οποία αργότερα μπορεί να γίνουν κλάσεις)
- Σχετίζεται με άλλες κλάσεις

Κατηγορίες κλάσεων ανάλυσης

Φανταστείτε ότι έχετε μόνο τρία είδη κλάσεων στη διάθεσή σας

κλάσεις ορίου Κλάσεις που είναι υπεύθυνες για να στέλνουν δεδομένα σε χαρακτήρες ή να παίρνουν δεδομένα από αυτούς.

κλάσεις οντότητας Κλάσεις που είναι υπεύθυνες για να αποθηκεύουν δεδομένα (γνωστές και ως *κλάσεις δεδομένων*)

κλάσεις ελέγχου κλάσεις που κάνουν ελέγχους, ορίζουν με ποια σειρά πρέπει να γίνουν διάφορες ενέργεια και «συνδέουν» τις ενέργειες των κλάσεων ορίου με τις κλάσεις οντότητας.

Κλάσεις ορίου (Boundary classes)

- Χρησιμοποιούνται για να περιγράψουν τη διασύνδεση μεταξύ του συστήματος και των χαρακτήρων
- Αυτή η διασύνδεση περιλαμβάνει συνήθως λήψη και παρουσίαση πληροφορίας και αιτήσεις από και προς το σύστημα
- Είναι συνήθως γενικεύσεις τερματικών, εκτυπωτών, παραθύρων κτλ
- Οι τάξεις αυτές περιγράφουν το ΤΙ επιτυγχάνεται από τη διασύνδεση και όχι το ΠΩΣ
- Κάθε τέτοια τάξη συνδέεται με τουλάχιστον ένα χαρακτήρα

Κλάσεις ορίου (Boundary classes

- Χρησιμοποιούνται για να περιγράψουν τη διασύνδεση μεταξύ του συστήματος και των χαρακτήρων
- Αυτή η διασύνδεση περιλαμβάνει συνήθως λήψη και παρουσίαση πληροφορίας και αιτήσεις από και προς το σύστημα
- Είναι συνήθως γενικεύσεις τερματικών, εκτυπωτών, παραθύρων κτλ
- Οι τάξεις αυτές περιγράφουν το ΤΙ επιτυγχάνεται από τη διασύνδεση και όχι το ΠΩΣ
- Κάθε τέτοια τάξη συνδέεται με τουλάχιστον ένα χαρακτήρα

Χαρακτηριστικά (attributes) κλάσεων ορίου

- Οι τάξεις ορίου που επικοινωνούν με ανθρώπους έχουν συνήθως χαρακτηριστικά που αφορούν πληροφορίες (πχ text fields, labels etc)
- Οι τάξεις ορίου που επικοινωνούν με άλλα συστήματα έχουν χαρακτηριστικά κάποιου πρωτοκόλλου επικοινωνίας

Κλάσεις οντότητας (Entity Class)

- Αναπαριστούν πληροφορία η οποία διατηρείται για κάποιο χρονικό διάστημα
- Στις περισσότερες περιπτώσεις προέρχονται κατευθείαν από το πεδίο εφαρμογής (domain model)

Χαρακτηριστικά κλάσεων οντοτήτων

- Τα χαρακτηριστικά μιας κλάσης οντοτήτων είναι συνήθως αρκετά φανερά
- Συνήθως μπορείτε να τα βρείτε από τα χαρακτηριστικά των εννοιών που έχετε αναπαραστήσει στο domain model

Ανακαλύπτοντας κλάσεις οντοτήτων

- Αρχικά μπορούν να σκιαγραφηθούν οι βασικές τάξεις οντοτήτων από το domain model.
- Οι σχέσεις που υπάρχουν στο domain model μπορούν να χρησιμοποιηθούν για να αποκαλύπτουν και τις σχέσεις ανάμεσα στις τάξεις οντοτήτων

Κλάσεις ελέγχου

- Αναπαριστούν συνεργασίες, μεταφορές, έλεγχο άλλων αντικειμένων. Αναπαριστούν υπολογισμούς
- Σε γενικές γραμμές η «δυναμική» του συστήματος αναπαριστάται από τις τάξεις ελέγχου

Χαρακτηριστικά κλάσεων ελέγχου

- Συνήθως οι κλάσεις ελέγχου δεν έχουν κάποια χαρακτηριστικά γιατί συνήθως έχουν και σύντομη διάρκεια ζωής.
- Πολλές φορές όμως μπορεί να έχουν χαρακτηριστικά που συσσωρεύονται, πχ μετρητές, που χρησιμοποιούνται σε μια περίπτωση χρήσης

Κάρτες CRC

- Χρησιμοποιούνται πολλές φορές να περιγράψουν με μεγαλύτερη λεπτομέρεια τη λειτουργία μιας κλάσης ανάλυσης
- Τα αρχικά τους προέρχονται από τη φράση Class Responsibility Collaboration

Order	
Check items are in stock	Order Line
Determine the price	Order Line
Check for valid payment	Customer
Dispatch to delivery address	

Κλάσεις ανάλυσης και MVC

- Η διαίρεση ενός συστήματος χρησιμοποιώντας τις κλάσεις ανάλυσης σε 3 βασικά επίπεδα βρίσκει εφαρμογή ως αρχιτεκτονική λύση σε διάφορα συστήματα και πολλές φορές συναντάται με τα αρχικά Model-View-Controller.
- Στην περιγραφή που κάναμε προηγουμένως
  - οι κλάσεις δεδομένων ανήκουν στο model
  - οι κλάσεις ορίου στο view
  - οι κλάσεις ελέγχου στο controller

Το αρχιτεκτονικό μοτίβο MVC

- Πρωτοπαρουσιάστηκε από τον Trygve Reenskaug, έναν προγραμματιστη Smalltalk στο Xerox Palo Alto Research Center το 1979
- βοηθά στη μείωση της σύζευξης μεταξύ των δεδομένων και της επιχειρηματικής λογικής από τον τρόπο που αυτά εμφανίζονται στο χρήστη

Βασικά συστατικά του MVC

Model Αναπαριστά δεδομένα και κανόνες που αφορούν στην πρόσβαση και στον έλεγχο των δεδομένων.

View Εμφανίζει τα δεδομένα ενός μοντέλου. Αν αλλάξει το μοντέλο το view πρέπει να ενημερώσει την εμφάνιση όπως απαιτείται.

- **Push model** Το view ενημερώνεται από το μοντέλο μέσω events
- **Pull model** Το view ρωτά το μοντέλο για τα ποιο πρόσφατα δεδομένα

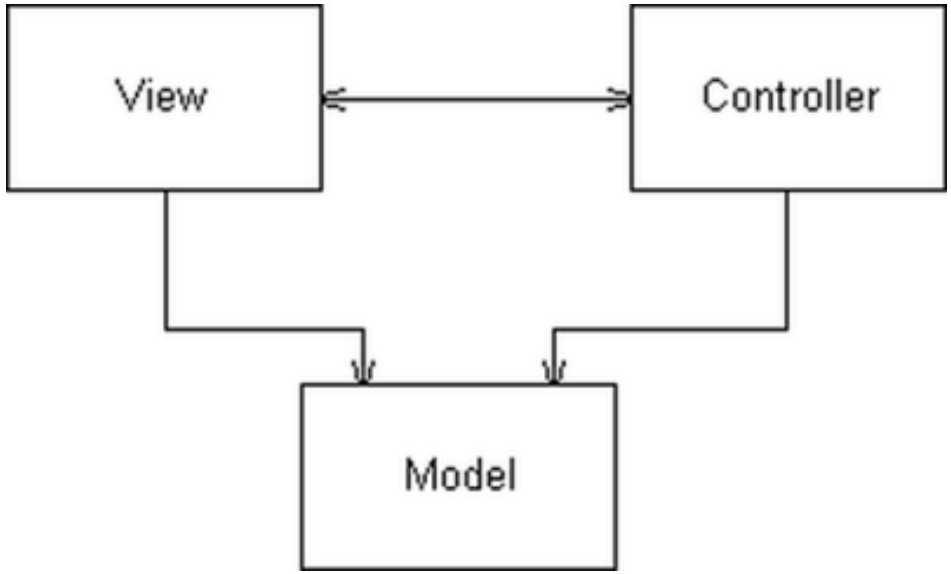
Controller Μεταφράζει τις ενέργειες του χρήση στο view σε ενέργειες στο μοντέλο

Δύο διαφορετικές υλοποιήσεις του MVC

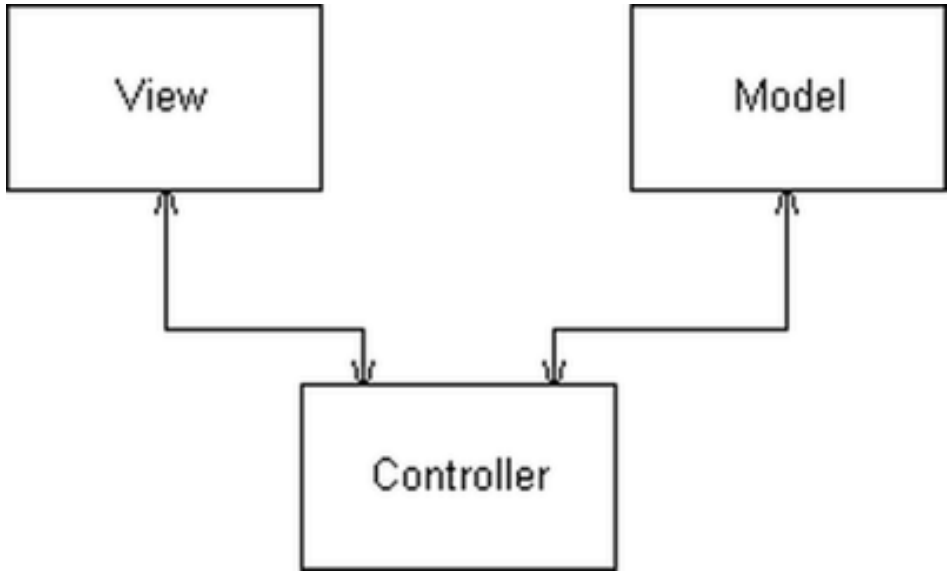
Το μοτίβο του MVC εμφανίζεται με δύο διαφορετικές τρόπους

- Μπορεί το view να ενημερώνεται από το μοντέλο απευθείας
- Μπορεί το view να ενημερώνεται μέσω του controller

Πρώτη προσέγγιση



Δεύτερη προσέγγιση



Πλεονεκτήματα της δεύτερης προσέγγισης

- Μεγαλύτερη ανεξαρτησία του μοντέλου από το view

Το MVC στη Java

- Στη Java χρησιμοποιείται μια τροποποιημένη έκδοση του MVC
- Το View και ο Controller είναι μία κλάση (Component)
- Κάποιες από τις ευθύνες ενός component ανατίθενται σε άλλες κλάσεις που είναι υπεύθυνες για τη δημιουργία του Pluggable Look & Feel

Παράδειγμα αντιστοίχησης στη Java

Component	**Model Interface**	**Είδος**
JButton	ButtonModel	GUI
JComboBox	ComboBoxModel	data
JTable	TableModel	data
JTable	TableColumnModel	GUI

Είδη μοντέλων στο Swing

GUI Interfaces που ορίζουν οπτικά θέματα όπως αν ένα κουμπί πατήθηκε η όχι

data Αναπαριστούν κάποια δεδομένα που έχουν νόημα για την εφαρμογή (για παράδειγμα οι πελάτες που εμφανίζονται σε ένα Table

Βιβλιογραφία

Για το σημερινό μάθημα θεωρείστε ως κομμάτι της ύλης και τα ακόλουθα

- Amy Fowler, A Swing Architecture Overview, The Inside Story on JFC Component Design,`http://www.eecs.yorku.ca/course_archive/2004-05/W/3461/FowlerArticle.pdf`
- Robert Exckstein, JAVA SE Application Design With MVC, 2007 (Διαθέσιμο στο elearning)

Τα παραδείγματα πραγματικού κώδικα στην ενότητα σύζευξης και συνοχής προέρχονται από το βιβλίο Diomidis Spinellis. Code Quality: The Open Source Perspective. Addison Wesley, 2006