# ΣΧΕΔΙΑΣΗ ΑΝΤΙΚΕΙΜΕΝΩΝ ΜΕ ΑΡΜΟΔΙΟΤΗΤΕΣ

Ορισμός σχεδιαστικών προτύπων
Εφαρμογή των 9 GRASP προτύπων

# Γενικά…

- Εξαιρετικά σημαντικά:
  - η απόφαση για το που ανήκουν οι μέθοδοι, και
  - πως αλληλεπιδρούν τα αντικείμενα
- Κρίσιμο βήμα – στην καρδιά της ανάπτυξης ΑΣ συστήματος
- Τα πρότυπα GRASP είναι:
  - ένα βοήθημα εκμάθησης και κατανόησης της ουσιαστικής σχεδίασης αντικειμένων, καθώς η αιτιολόγηση της προσέγγισής της γίνεται με ένα μεθοδικό, λογικό, και επεξηγηματικό τρόπο.
- Η προσέγγιση της κατανόησης και της χρησιμοποίησης των σχεδιαστικών αρχών βασίζεται στα *πρότυπα και στην ανάθεση αρμοδιοτήτων.*

# Αρμοδιότητες και Μέθοδοι

- Η UML ορίζει μια αρμοδιότητα ως:
  - «ένα συμβόλαιο ή υποχρέωση μιας οντότητας»
  - Υλοποίηση με μεθόδους (δεν είναι μέθοδος)

  Οι υποχρεώσεις είναι δύο τύπων:
  - Πράξη
    - Κάτι που κάνει μόνο του (πχ. δημιουργία αντικειμένου, ενός υπολογισμού)
    - Έναρξη λειτουργίας σε άλλη κλάση
    - Έλεγχος και συντονισμός ενεργειών σε άλλες κλάσεις
  - Γνώση
    - σχετικά με ιδιωτικά δεδομένα
    - σχετικά σε σχετιζόμενα αντικείμενα
    - σχετικά με πράγματα που μπορεί να παράγει ή να υπολογίσει

# Σχεδιαστικά Πρότυπα (1/2)

- Αποτελούν ονοματισμένα, κωδικοποιημένα ζεύγη «προβλημάτων / λύσεων», τα οποία έχουν προταθεί από έμπειρους δημιουργούς με βάση συγκεκριμένες σχεδιαστικές αρχές.

*Παράδειγμα Παρουσίασης Προτύπου:*

- *Όνομα προτύπου:* Information Expert
- *Λύση*: Ανάθεση αρμοδιότητας σε κλάση η οποία έχει την πληροφορία για να την εκπληρώσει
- *Πρόβλημα που επιλύει*: ποια είναι η βασική αρχή ανάθεσης αρμοδιότητας σε αντικείμενα;

# Σχεδιαστικά Πρότυπα (2/2)

- Η ιδέα ενός προτύπου αναφέρεται σε επαναληπτικές εφαρμογές.
- Δεν εκφράζουν νέες ιδέες σχεδίασης, αλλά κωδικοποιούν υπάρχουσες εμπειρίες σχεδιαστικών αρχών.
- Όσο συχνότερα και ευρύτερα χρησιμοποιούνται τόσο καλύτερα.
- Τα πρότυπα έχουν δηλωτικά ονόματα, για να:
  - κατανοούμε και να απομνημονεύουμε μια έννοια,
  - βοηθούν στην επικοινωνία μεταξύ δημιουργών
- Εφαρμόζονται κατά την δημιουργία Διαγραμμάτων Αλληλεπίδρασης και Κωδικοποίησης.

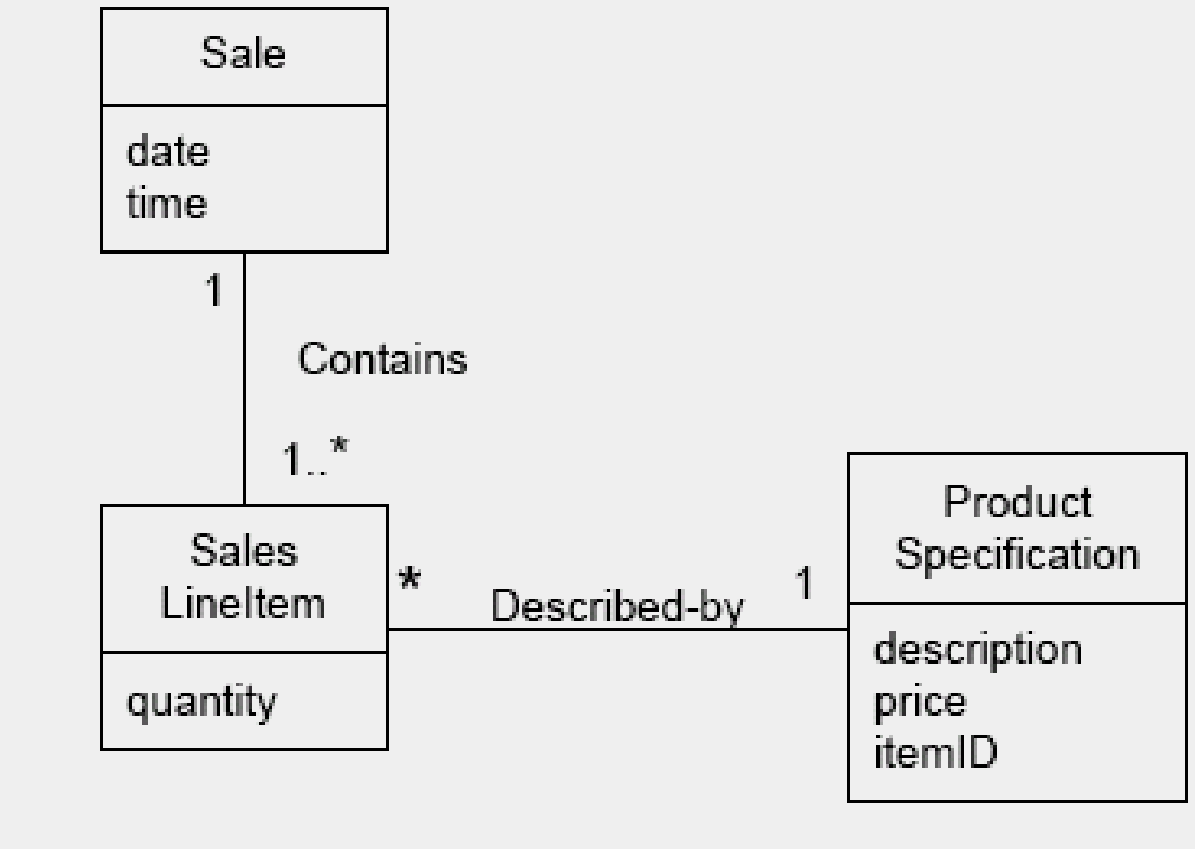
# GRASP Πρότυπα Σχεδίασης

1. Φορέας πληροφορίας (Information Expert)
2. Δημιουργός (Creator)
3. Υψηλή Συνοχή (High Cohesion)
4. Μειωμένη Σύζευση (Low Coupling)
5. Ελεγκτής (Controller)
6. Πολυμορφισμού
7. Indirection
8. Pure Fabrication
9. Protected Variations

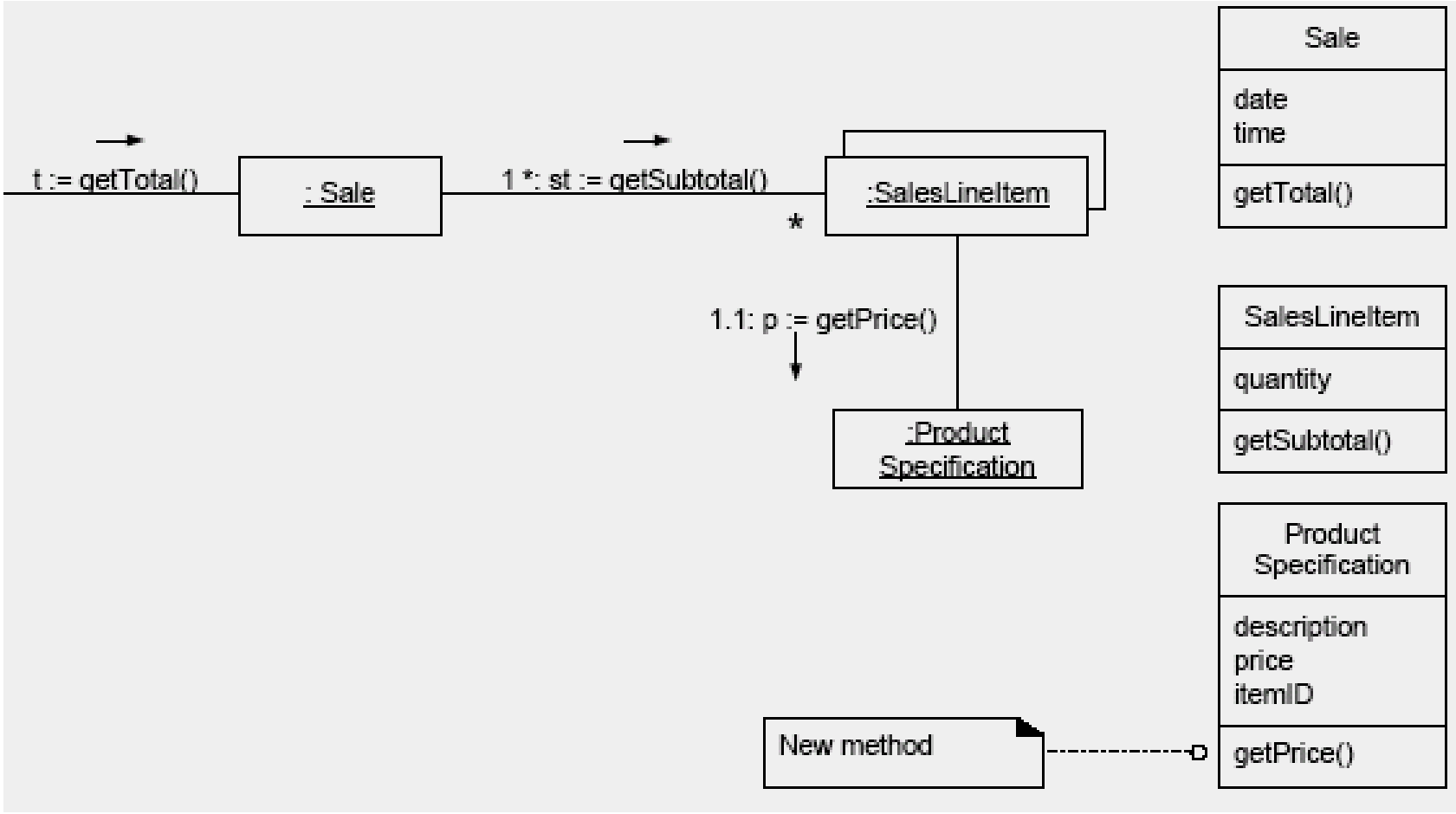
# «Φορέας Πληροφορίας» (1/4)

- **Πρόβλημα**: Ποια είναι η βασική αρχή ανάθεσης αρμοδιότητας σε αντικείμενα;
- **Λύση**: Ανάθεση αρμοδιότητας στον φορέα της πληροφορίας – μια κλάση η οποία έχει την πληροφορία για να εκπληρώσει την αρμοδιότητα
- Συμβάλει στην ευκολία κατανόησης, συντήρησης, επέκτασης, επαναχρησιμοποίησης
- *Παράδειγμα*:
  - *Που θα δημιουργηθεί το 'Γενικό σύνολο;'*

# «Φορέας Πληροφορίας» (2/4)



# «Φορέας Πληροφορίας» (3/4)



# «Φορέας Πληροφορίας» (4/4)

- Οι αρμοδιότητες αποφασίζονται στα Διαγράμματα Αλληλεπίδρασης
- Το πρότυπο εκφράζει το αυτονόητο, ότι τα αντικείμενα λειτουργούν σύμφωνα με τις πληροφορίες που φέρουν ή κατέχουν
- Πλεονεκτήματα:
  - Τηρείται η αρχή της ενθυλάκωσης πληροφοριών
  - Η συμπεριφορά κατανέμεται σε διάφορες κλάσεις υποστηρίζοντας τη συνοχή και σύζευξη

# «Δημιουργός» (Creator) (1/3)

- Λύση: Ανάθεση στην κλάση Β την αρμοδιότητα να δημιουργήσει ένα στιγμιότυπο της κλάσης Α, εάν συντρέχει ένας από τους κάτωθι λόγους:
  1. η Β *συνθέτει* αντικείμενα της Α
  2. η Β *περιλαμβάνει (συσσωματώνει)* αντικείμενα της Α
  3. η Β *καταγράφει* στιγμιότυπα αντικειμένων της Α
  4. η Β *χρησιμοποιεί* αντικείμενα της Α
  5. η Β *έχει τα δεδομένα αρχικοποίησης* αντικειμένων της Α

  Για περισσότερο από ένα λόγους να χρησιμοποιηθεί το 1, και 2
- Πρόβλημα: ποιος είναι αρμόδιος για την δημιουργία νέου στιγμιότυπου (αντικειμένου) κλάσης;
- *Παράδειγμα*:
  - *Ποιος ... για τη δημιουργία της SalesLineItem;*
- Πλεονέκτημα: Ελάχιστη δυνατή Σύνδεση

# «Δημιουργός» (Creator) (2/3)

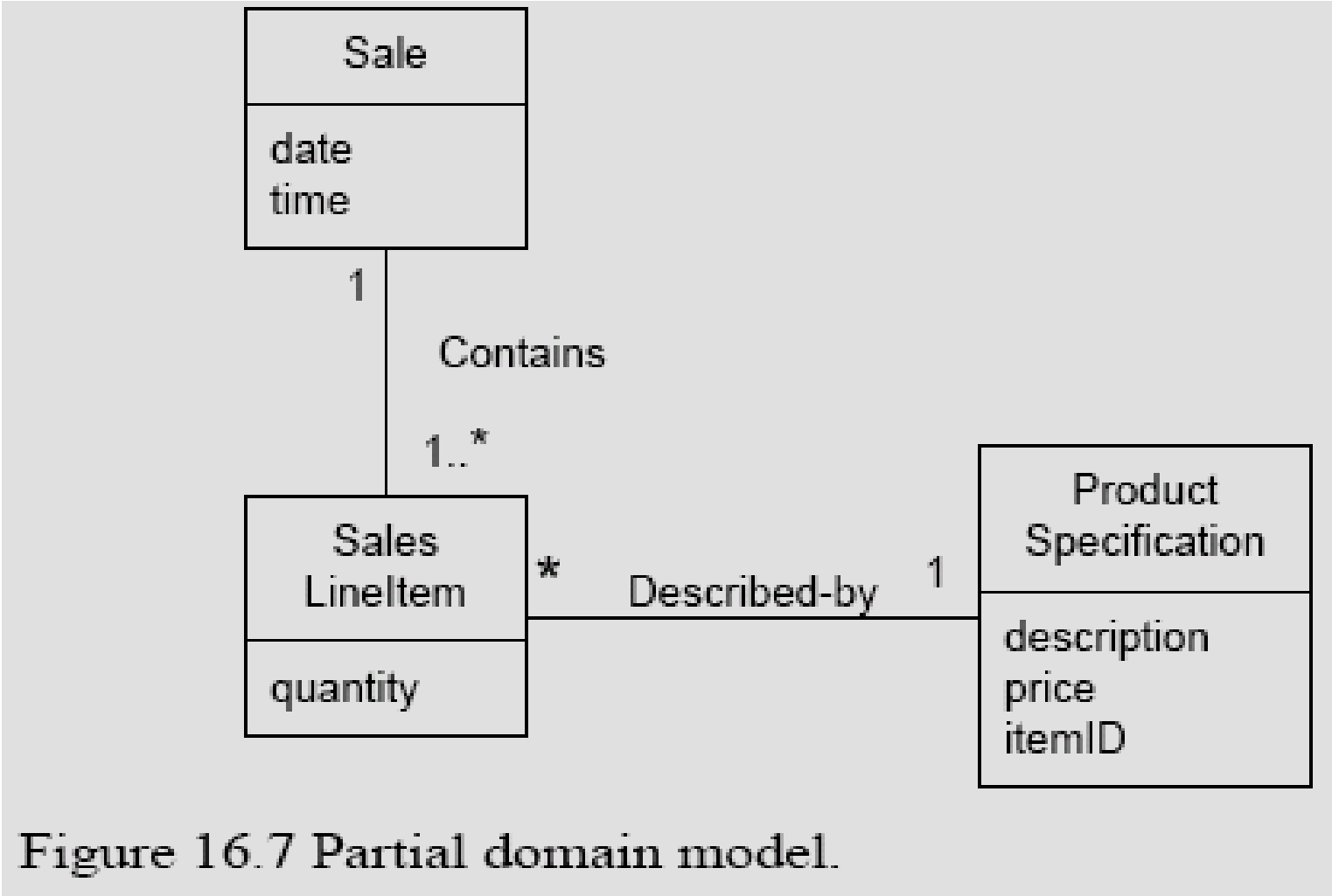



Figure 16.7 Partial domain model.

# «Δημιουργός» (Creator) (3/3)

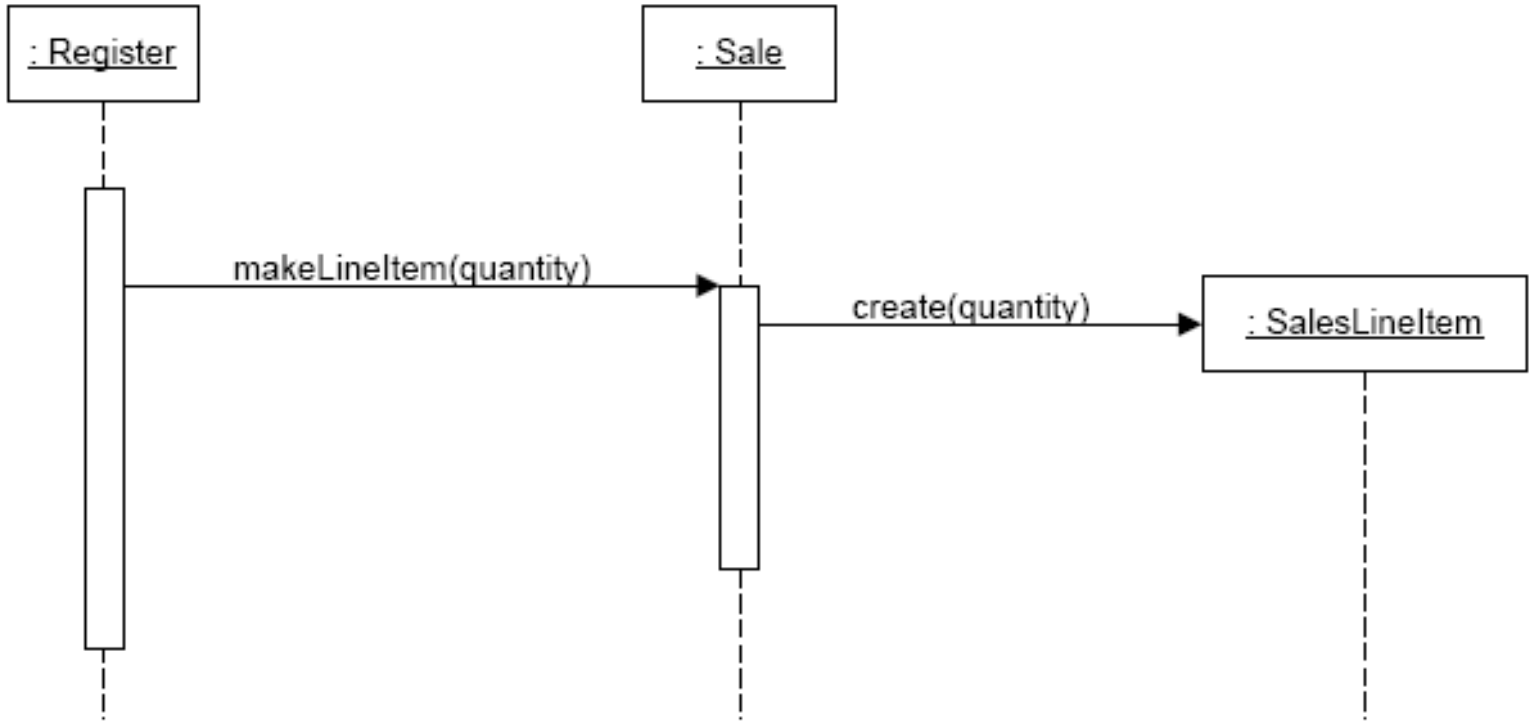



Figure 16.8 Creating a SalesLineItem.

# «Μείωση Σύζευξης» (Low Coupling) (1/5)

- Σύζευξη: μέτρο του βαθμού διασύνδεσης, γνώσης ή εξάρτησης με άλλα αντικείμενα
- Λύση: Ανάθεση μιας αρμοδιότητας ούτως ώστε η σύζευξη να παραμένει η ελάχιστη δυνατή
- Πρόβλημα: πως να διασφαλιστεί περιορισμένη εξάρτηση, περιορισμένες επιπτώσεις από αλλαγές, και αυξημένη επαναχρησιμοποίηση;
- Προβλήματα αυξημένου βαθμού σύζευξης:
  - Οι τοπικές αλλαγές επιφέρουν επιπτώσεις (αλλαγές) σε σχετιζόμενες κλάσεις
  - Δυσκολία κατανόησης μεμονωμένα
  - Δυσκολότερη επαναχρησιμοποίηση

# «Μείωση Σύζευξης» (Low Coupling) (3/5)

- *Παράδειγμα*:
  - *Μεταξύ 3 εννοιολογικών κλάσεων (Payment, Sale, Register), ποια κλάση θα* διασύνδεαι *το στιγμιότυπο της Payment με τη Sale;*

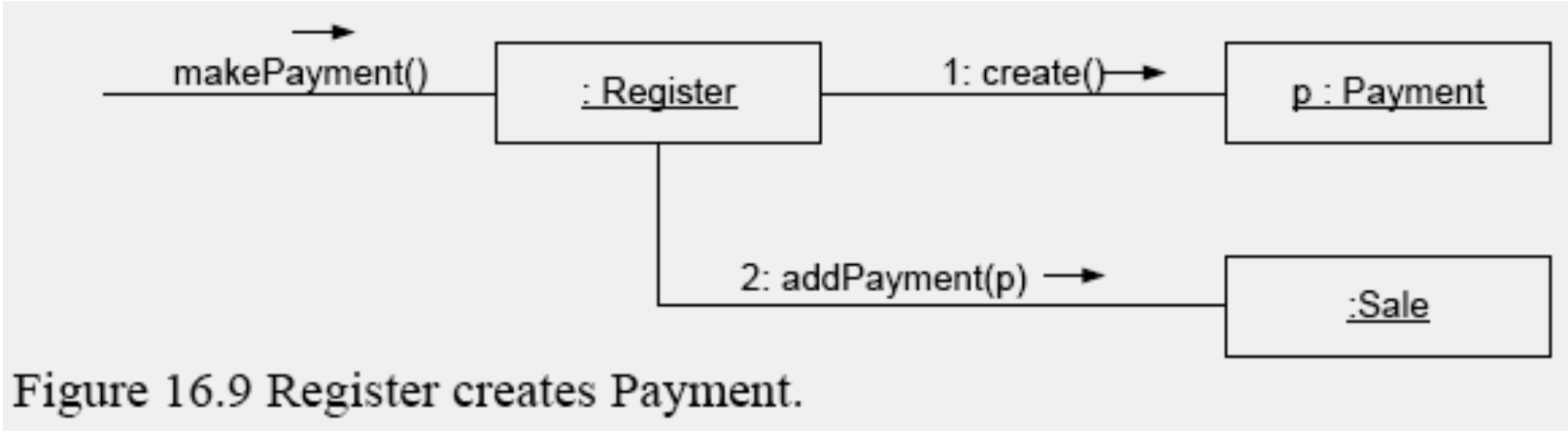



Figure 16.9 Register creates Payment.

# «Μείωση Σύζευξης» (Low Coupling) (4/5)

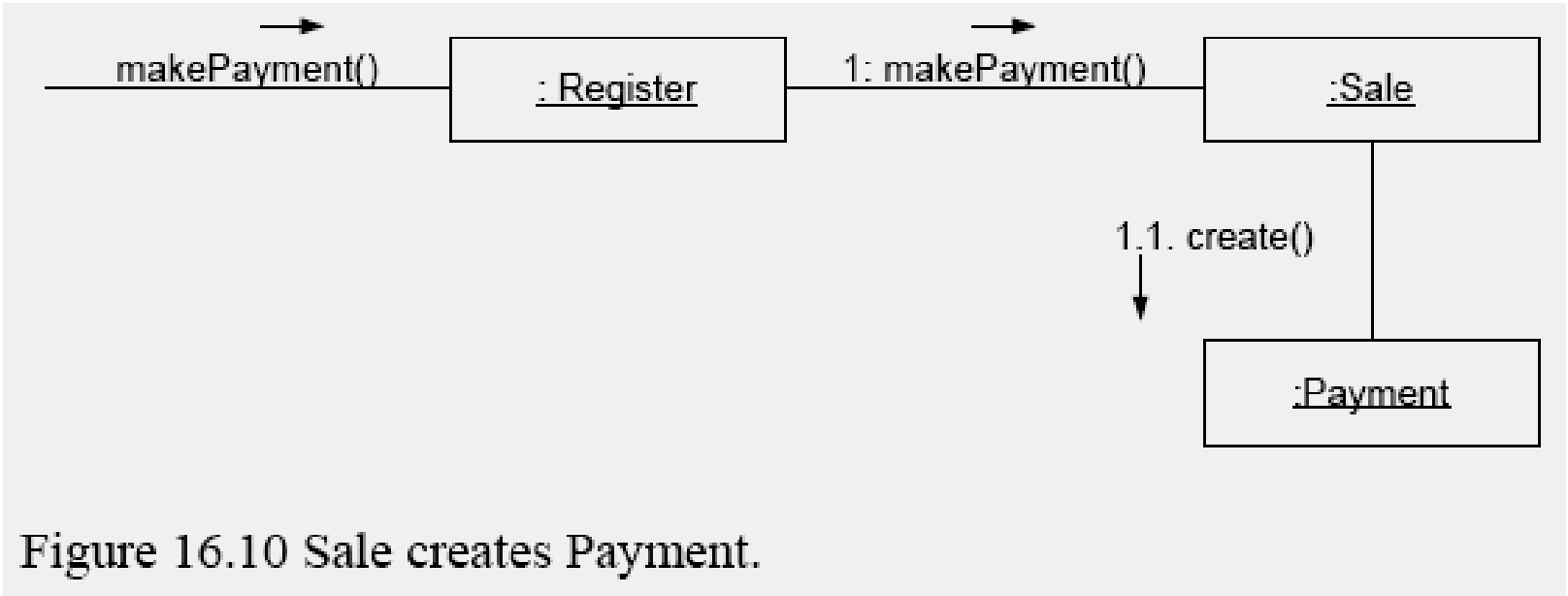



Figure 16.10 Sale creates Payment.

# «Μείωση Σύζευξης» (Low Coupling) (5/5)

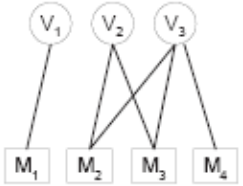
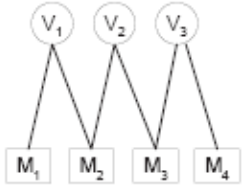
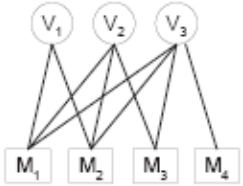
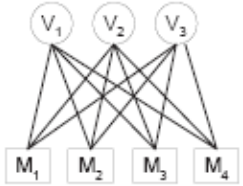
- Η Ελάχιστη Σύζευση είναι αξιολογητική αρχή
- Συνήθης μορφές Σύζευσης σε C++, Java, C#
  1. *TypeX* έχει χαρακτηριστικό-αναφορά του *TypeY*
  2. Αντικ. *TypeX* καλεί υπηρεσίες του αντικ. *TypeY*
  3. *TypeX* έχει μέθοδο που αναφέρεται στο αντικ. *TypeY*
  4. *TypeX* είναι άμεση ή έμμεση υποκλάση του *TypeY*
  5. *TypeX* είναι μια διασύνδεση, και η *TypeY* την υλοποιεί
- Πλεονεκτήματα:
  - Δεν επηρεάζεται από αλλαγές σε άλλα μέλη
  - Εύκολη η κατανόηση μεμονωμένα
  - Ευκολία επαναχρησιμοποίησης

# Υψηλή Συνοχή (High Cohesion) (1/4)

- Συνοχή: μέτρο του βαθμού σχέσης και «συγγένειας», των αρμοδιοτήτων σε μια κλάση
- Λύση: Ανάθεση μιας αρμοδιότητας ούτως ώστε η συνοχή να παραμένει μεγάλη
- Πρόβλημα: πως να διατηρηθεί η πολυπλοκότητα υπό έλεγχο;
- Προβλήματα μειωμένου βαθμού συνοχής:
  - Δυσκολία κατανόησης, επαναχρησιμοποίησης, συντήρησης
  - «συστέγαση» πολλών άσχετων μεταξύ τους αρμοδιοτήτων

# Υψηλή Συνοχή (High Cohesion) (2/4)

Table II. Examples for cohesion measures.

| Measure |  (a) class A |  (b) class B |  (c) class C |  (d) class D |
|---|---|---|---|---|
| LCOM1 | 3 | 3 | 0 | 0 |
| LCOM2 | 0 | 0 | 0 | 0 |
| LCOM3 | 2 | 1 | 1 | 1 |
| LCOM4 | 2 | 1 | 1 | 1 |
| Co | N/A[a] | 0 | 1 | 1 |
| LCOM5 | $\frac{2}{3}$ | $\frac{2}{3}$ | $\frac{1}{3}$ | 0 |
| Coh | $\frac{1}{2}$ | $\frac{1}{2}$ | $\frac{1}{3}$ | 1 |
| LCC | $\frac{1}{2}$ | 1 | 1 | 1 |
| TCC | $\frac{1}{2}$ | $\frac{1}{2}$ | 1 | 1 |

[a]Co is not defined for a class whose LCOM4 is not one.

# Υψηλή Συνοχή (High Cohesion) (3/4)

- *Παράδειγμα:*
  - *Το πρόβλημα της Σύζευσης (προηγ.)*

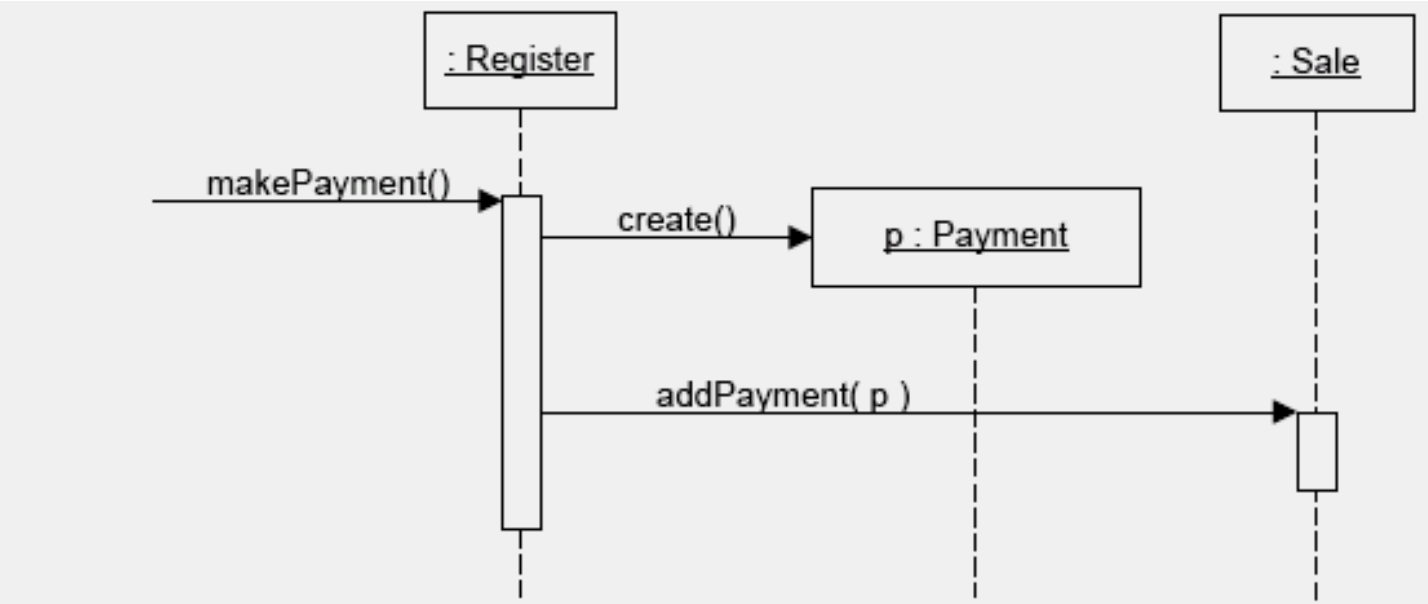



Figure 16.11 Register creates Payment.

# Υψηλή Συνοχή (High Cohesion) (4/4)

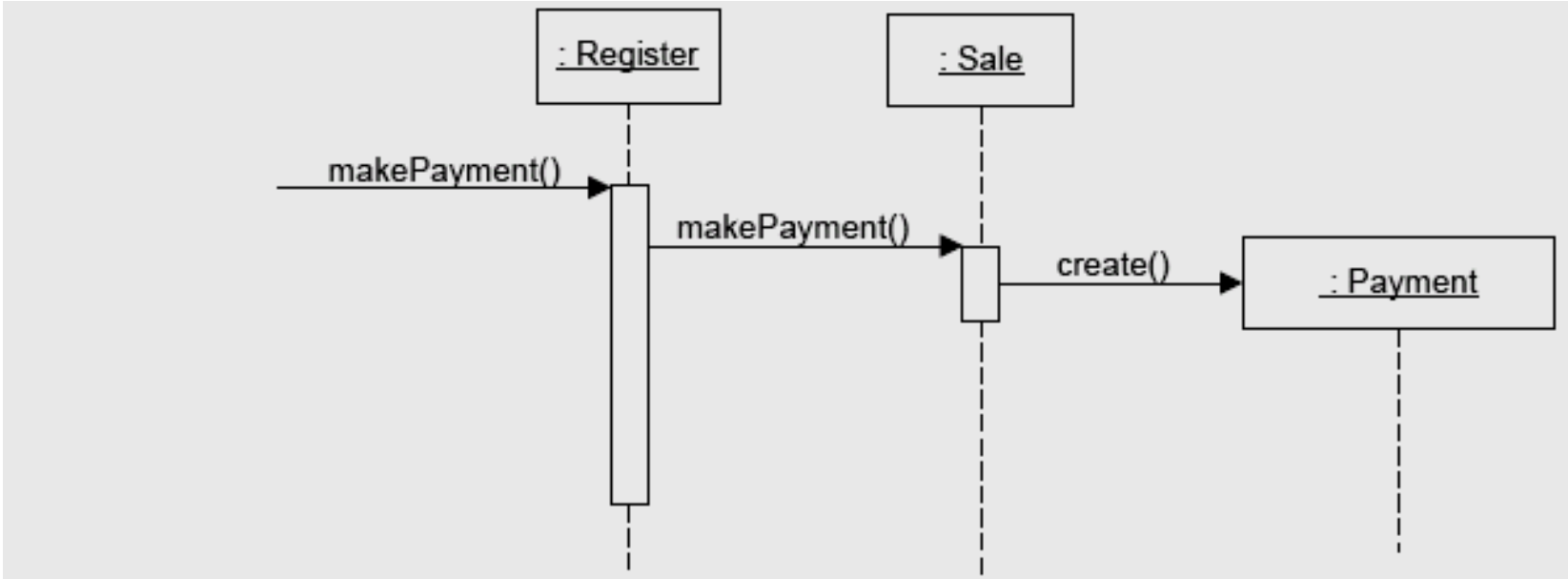



Figure 16.12 Sale creates Payment

# «Ελεγκτής» (Controller) (1/6)

- Ελεγκτής: ένα αντικείμενο, υπεύθυνο για να δεχτεί ή να διαχειριστεί ένα γεγονότος συστήματος
- Λύση: Ανάθεση της αρμοδιότητας αποδοχής ή διαχείρισης, μηνύματος γεγονότος συστήματος, σε κλάση που αντιπροσωπεύει μια από τις εξής επιλογές (πρόσοψη):
  - Ολόκληρο σύστημα, συσκευή, υποσύστημα
  - Ένα σενάριο ΠΧ, μέσα στο οποίο συμβαίνει το γεγονός συστήματος
- Πρόβλημα: ποιος είναι αρμόδιος για την διαχείριση ενός γεγονότος εισαγωγής στο σύστημα (από εξωτ. χρήστη);
- *Παράδειγμα*: εικ. 16.13, 16.14

# «Ελεγκτής» (Controller) (2/6)

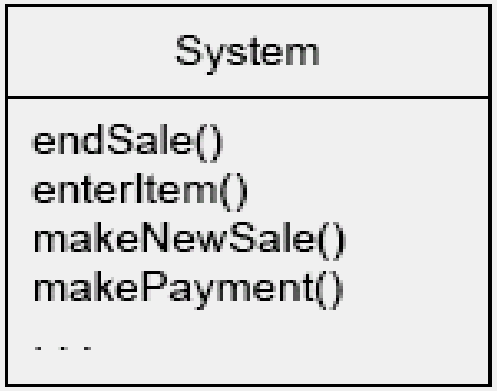



Figure 16.13 System operations associated with the system events.

- Συνιστάται να χρησιμοποιείται ένας **Ελεγκτής** για κάθε Περίπτωση Χρήσης.
- Ουσιαστικά ο Ελεγκτής δέχεται την αίτηση από το επίπεδο UI, συντονίζει και ελέγχει την δραστηριότητα, *παραπέμποντας* τη υλοποίηση σε άλλα αντικείμενα.
- Είναι από την πλευρά του *client.*

# «Ελεγκτής» (Controller) (3/6)

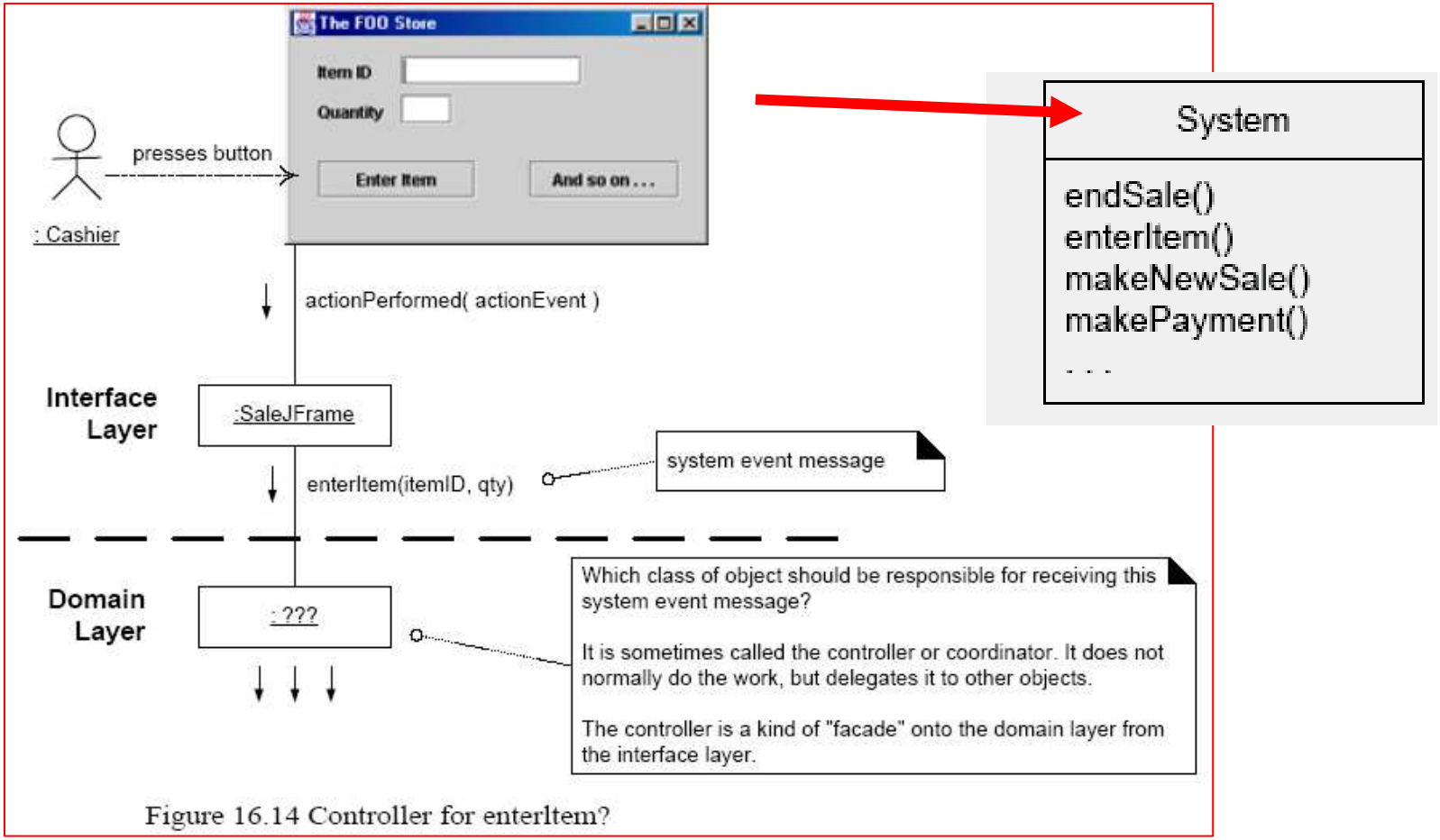



Figure 16.14 Controller for enterItem?

# «Ελεγκτής» (Controller) (4/6)

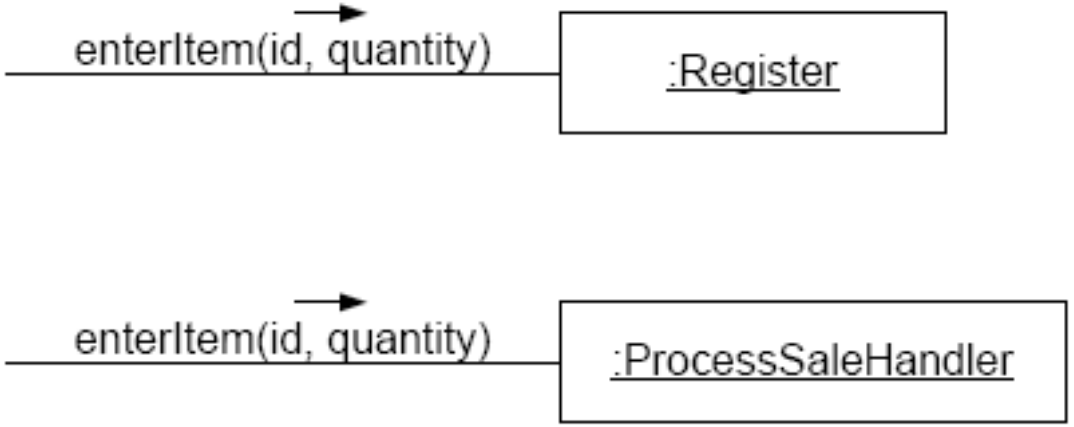



Figure 16.15 Controller choices.

# «Ελεγκτής» (Controller) (5/6)

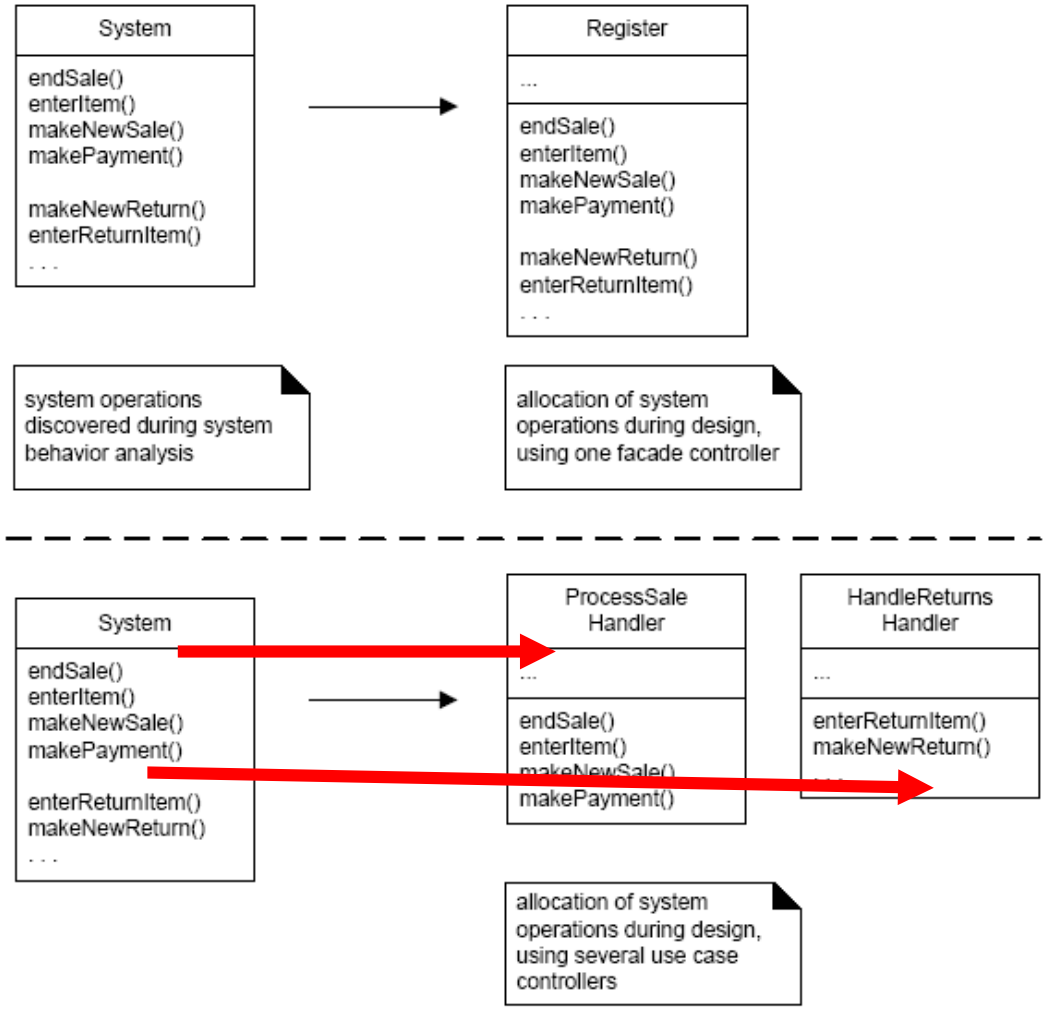



Figure 16.16 Allocation of system operations.

# «Ελεγκτής» (Controller) (6/6)

- Πλεονεκτήματα:
  - Αυξάνει την δυνατότητα επαναχρησιμοποίησης
  - Προσαρμόσιμη διεπιφάνεια
  - Ελέγχει τη σειρά εκτέλεσης λειτουργιών μιας Περίπτωσης Χρήσης
    - endSale() εκτελείται μετά την Payment()

# Πρότυπο Πολυμορφισμού (1/4)

Λύση όταν σχετικές εναλλακτικές λύσεις ή συμπεριφορές ποικίλουν σε τύπο (κλάση)

Πρόβλημα Πως να χειριστείς εναλλακτικές λύσεις βάσει *τύπου*; Πως να δημιουργήσεις προσαρμόσιμες μονάδες λογισμικού;

Παράδειγμα Στο NextGen POS διαφορετικοί εξωτ. υπολογιστές φόρου, με διαφορετική διασύνδεση και πρωτόκολλο (TCP socket, SOAP, Java RMI)

Οι «προσαρμοστές» είναι *εσωτ. αντικείμενα* που αντιπροσωπεύουν εξωτ. υπολογιστές φόρου

# Πρότυπο Πολυμορφισμού (2/4)

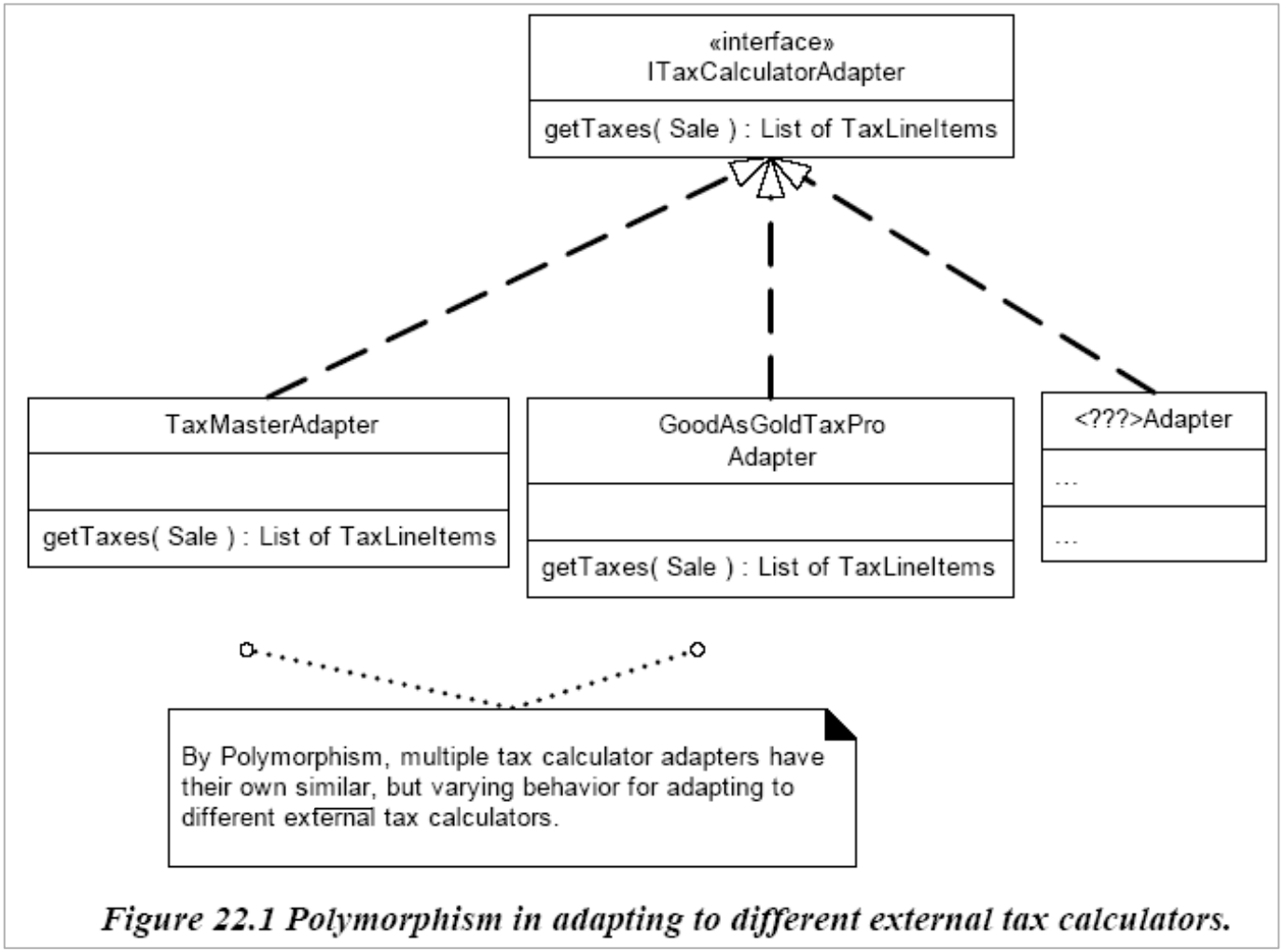



*Figure 22.1 Polymorphism in adapting to different external tax calculators.*

# Πρότυπο Πολυμορφισμού (3/4)

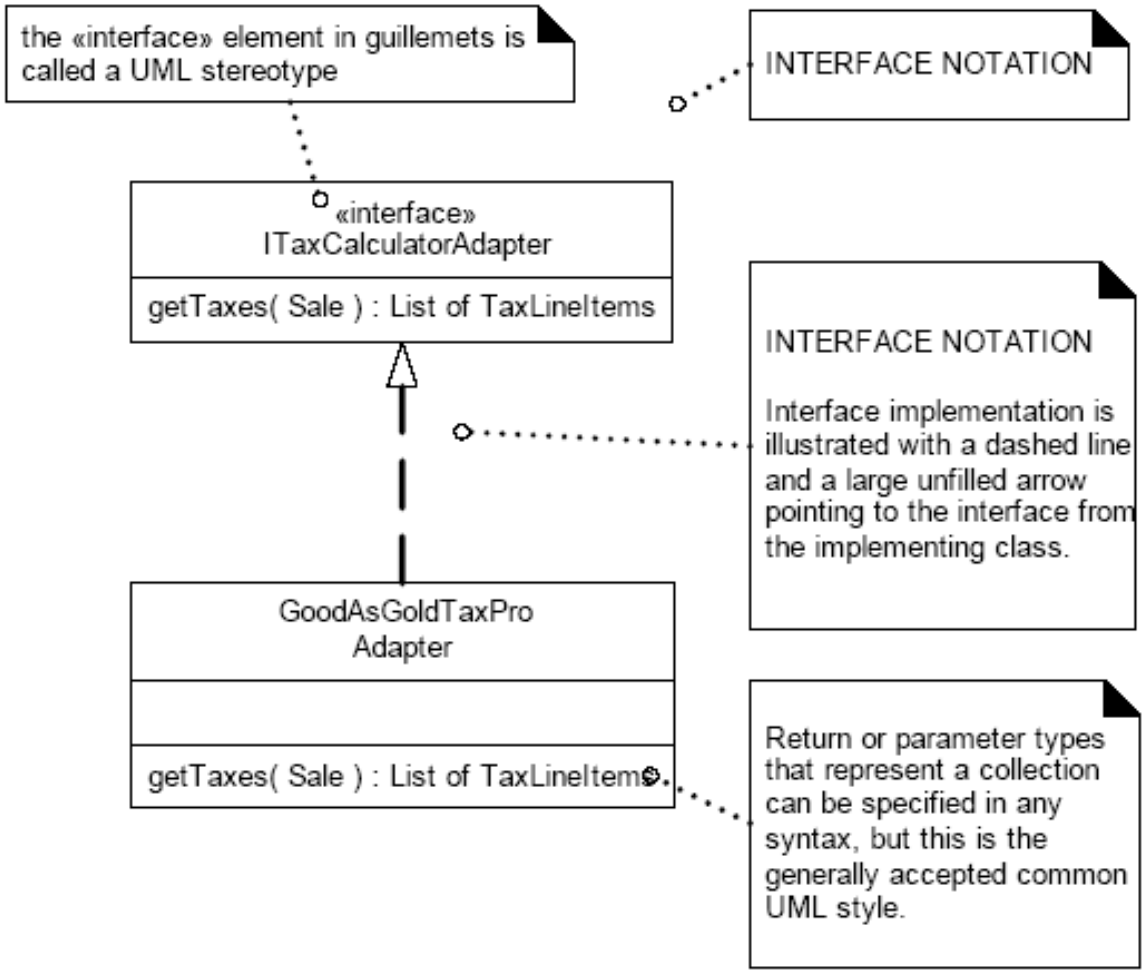



*Figure 22.2 UML notation for interfaces and return types.*

# Πρότυπο Πολυμορφισμού (4/4)

Πολυμορφισμός είναι μια θεμελιώδης αρχή σχεδίασης και οργάνωσης ενός συστήματος για τον χειρισμό παρόμοιων (συναφών) παραλλαγών

Οφέλη

- Εύκολη προσθήκη νέων επεκτάσεων για νέες παραλλαγές
- Εισαγωγή νέων υλοποιήσεων δίχως να επηρεάζονται οι εξυπηρετητές

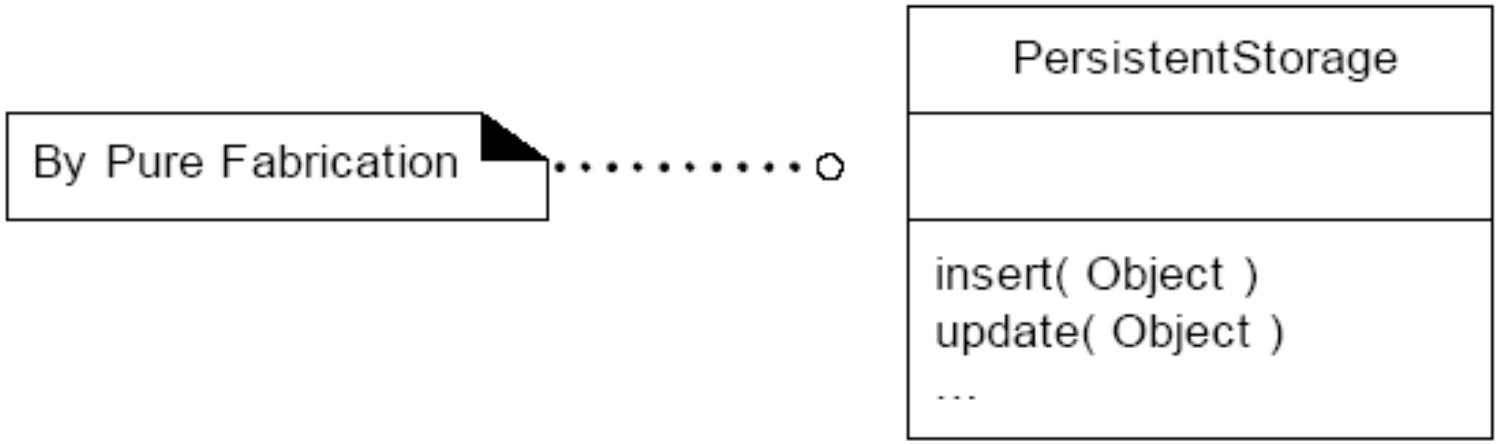
# Τεχνητή Επινόηση (Pure Fabrication) (1/3)

Λύση: Όρισε ένα σύνολο αρμοδιοτήτων υψηλής συνοχής σε μια τεχνητή κλάση η οποία δεν αντιπροσωπεύει κάποια έννοια της ΠΠ

Πρόβλημα: Σε ποιο αντικείμενο αναθέτουμε μία αρμοδιότητα χωρίς να παραβιαστούν οι κανόνες Συνοχής–Σύζευξης, αλλά η λύση του Φορέα Πληροφορίας δεν είναι η καταλληλότερη;

Παράδειγμα καταχώρηση στιγμιότυπων *Sale* σε ΒΔ.

# Τεχνητή Επινόηση (Pure Fabrication) (2/3)




Προβλήματα που επιλύει:

- Η *Sale* παραμένει καλοσχεδιασμένη
- Η *PersistentStorage* κλάση είναι συνεκτική
- Η *PersistentStorage* κλάση είναι πολύ γενική και επαχρησιμοποιήσιμη

# Τεχνητή Επινόηση (Pure Fabrication) (3/3)

Η σχεδίαση αντικειμένων περιλαμβάνει 2 ομάδες:

- Αντιπροσωπευτική ταξινόμηση (representational decomposition – από την ΠΠ, πχ. *'Sale'*)
  - Μειώνει το αντιπροσωπευτικό χάσμα
- Ταξινόμηση συμπεριφοράς (behavioral decomposition, πχ. *'TableOfContentsGenerator'*)
  - Ομαδοποίηση γενικών αρμοδιοτήτων
- Οφέλη
  - Υψηλή συνοχή, λόγω ομαδοποίησης σχετικών μεταξύ τους γενικών αρμοδιοτήτων
  - Δυνατότητα αύξησης επαναχρησιμοποίησης

# Indirection (1/3)

Λύση: Ανάθεσε την αρμοδιότητα σε ενδιάμεσο αντικείμενο το οποίο μεσολαβεί μεταξύ άλλων μονάδων ή υπηρεσιών, ώστε αυτά να μην συνδέονται άμεσα.

Πρόβλημα: Αποσύνδεση μεταξύ αντικειμένων με στόχο την μειωμένη σύζευξη και αύξηση της δυνατότητας επαναχρησιμοποίησης.

Παραδείγματα *TaxCalculatorAdapter*

Indirection + Πολυμορφισμός => προστασία της εσωτ. Σχεδίασης από διαφορετικές εξωτ. διασυνδέσεις (σχήμα 22.3)

# Indirection (2/3)

*PersistentStorage*

υποστηρίζει την indirection ανάμεσα στην *Sale* και την ΒΔ

Οφέλη

- Χαμηλή Σύζευξη (σύνδεση) μεταξύ αντικειμένων

# Indirection (3/3)

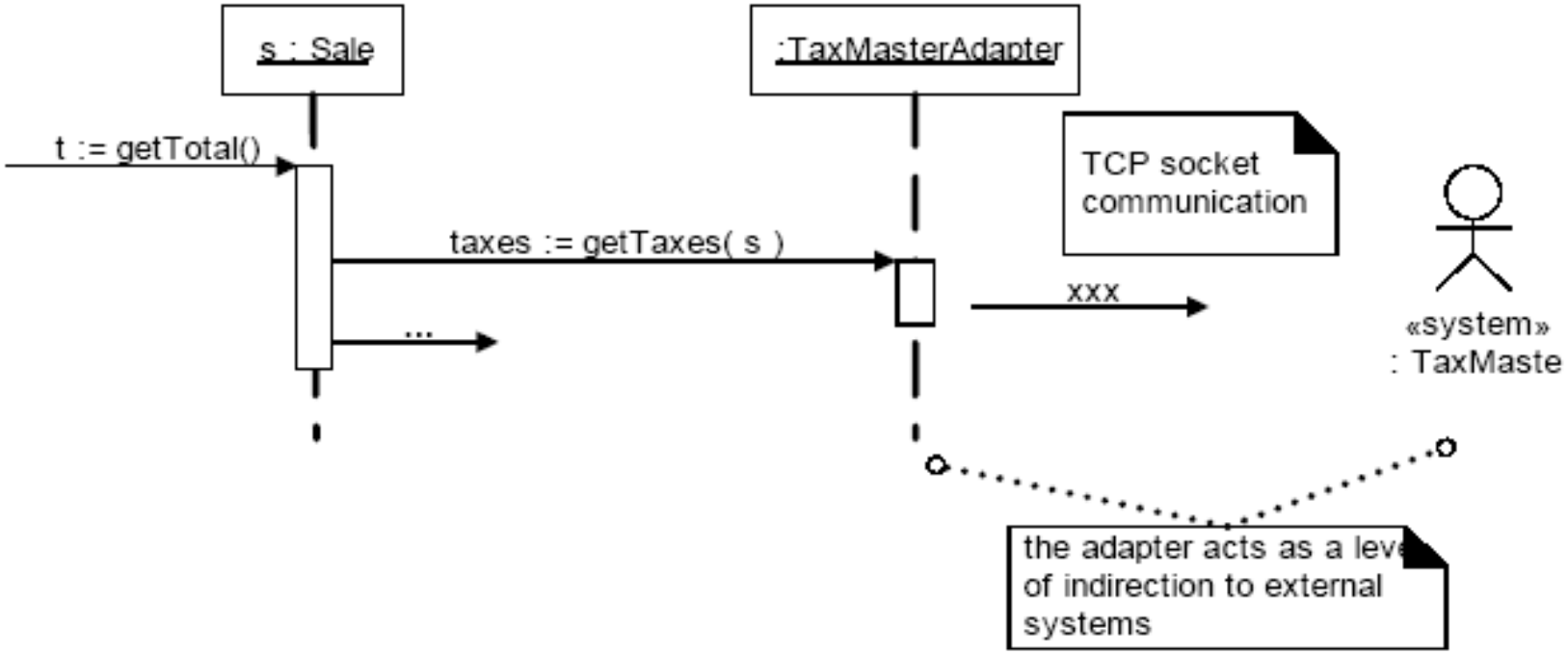



***Figure 22.3 Indirection via the adapter.***

# Protected Variations (1/4)

Λύση εντοπισμός προβλεπόμενων ασταθών σημείων. Δημιουργία σταθερής διασύνδεσης γύρω από αυτά.

Πρόβλημα Σχεδίαση αντικειμένων, υποσυστημάτων ή συστημάτων, ώστε οι αλλαγές σ' αυτά να μην επιφέρουν επιπτώσεις σε άλλα μέρη.

Παράδειγμα Τα σημεία αστάθειας και αλλαγών στις διασυνδέσεις των υπολογ. Φόρων (σχ. 22.1). Τα εσωτ. αντικείμενα συνεργάζονται πλέον με μια σταθερή διασύνδεση.

# Protected Variations (2/4)

*Μηχανισμοί υποστήριξης PVs*

*Core Protected Variations Mechanisms*

Η ενθυλάκωση (encapsulation), οι διασυνδέσεις, ο πολυμορφισμός και η Indirection.

*The Liskov Substitution Principle*

Αν για κάθε αντικείμενο *a1* του τύπου *S* υπάρχει αντικείμενο *a2* του τύπου *T* ώστε όλα τα προγράμματα *P* που αναφέρονται στο *T,* η συμπεριφορά του *P* δεν αλλάζει όταν το *a1* αντικαθίσταται από το *a2*, τότε το *S* είναι υποτύπος του *T.*

```
public void addTaxes( ITaxCalculatorAdapter calculator. Sale sale ){
    List taxLineItems = calculator.getTaxes( sale );
    // ... }
```

# Protected Variations (3/4)

*Σχεδίαση Απόκρυψης Δομής* (Law of Demeter)

Θεωρεί ότι μέσα σε μια μέθοδο τα μηνύματα θα αποστέλλονται μόνο στα ακόλουθα αντικείμενα:

1. Στο *this*
2. Σε παράμετρο της μεθόδου
3. Σε χαρακτηριστικό του *this*
4. Σε μέλος συλλογής (πχ. πίνακα) ο οποίος είναι χαρακτηριστικό του *this*
5. Σε αντικείμενο που δημιουργείται στην μέθοδο

# Protected Variations (4/4)

```
class Register
{
private Sale sale;

public void slightlyFragileMethod() {
    // sale.getPayment() sends a message to a "familiar" (passes #3)

    // but in sale.getPayment().getTenderedAmount()
    // the getTenderedAmount() message is to a "stranger" Payment

    Money amount = sale.getPayment().getTenderedAmount();

    // ...
  }
    // ...
  }
```