Μεθοδολογία Προγραμματισμού

Μοτίβα σχεδίασης (Design Patterns)

Νικόλαος Πεταλίδης

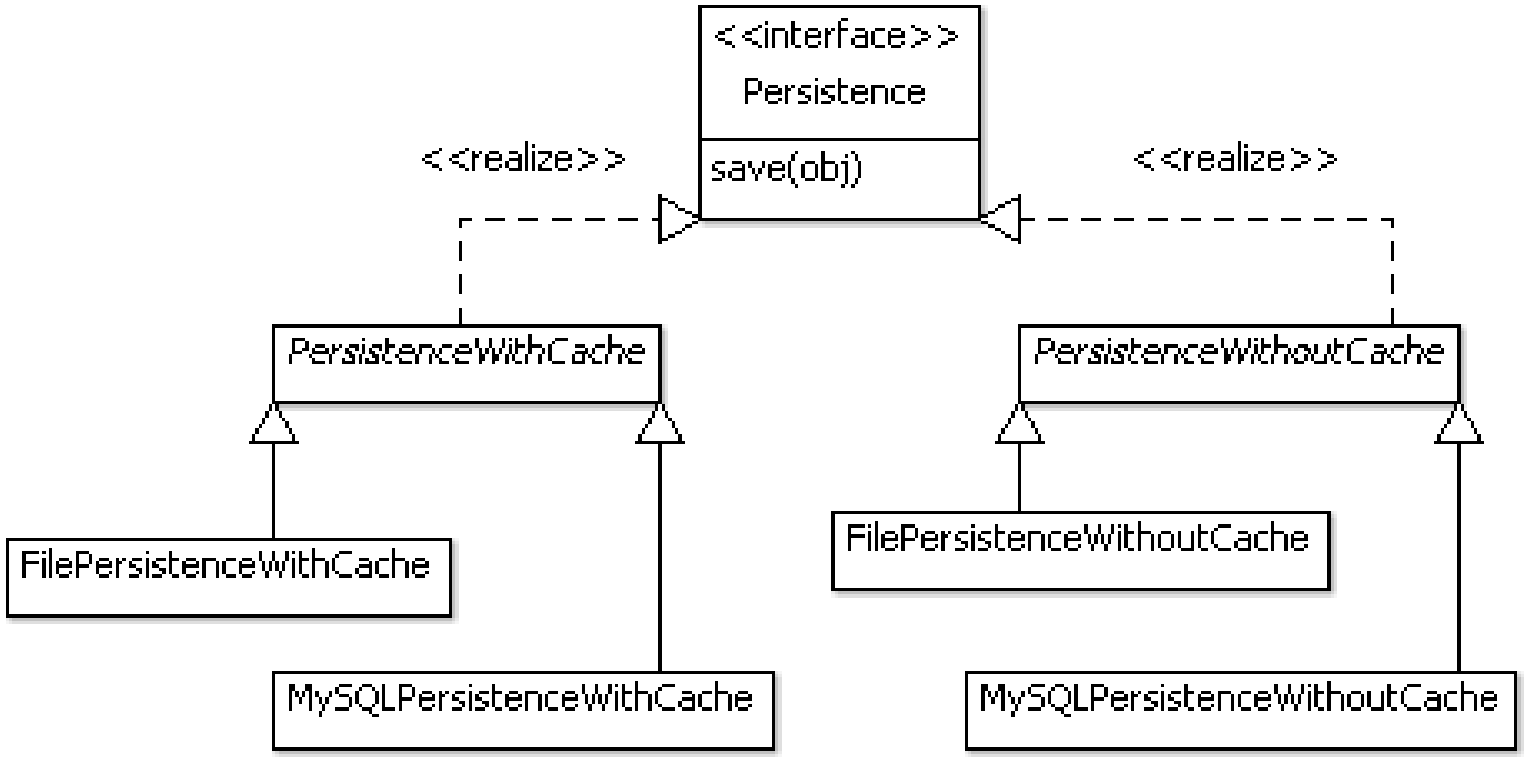
Τμήμα Μηχανικών Πληροφορικής και Επικοινωνιών
Διεθνές Πανεπιστήμιο της Ελλάδος

Εισαγωγή
Εαρινό Εξάμηνο

Bridge Pattern

- Θέλετε να σχεδιάσετε μια βιβλιοθήκη μόνιμης αποθήκευσης είτε σε αρχείο είτε σε βάση
- Η βιβλιοθήκη σας μπορεί να έχει διαφορετικούς τρόπους να αποθηκεύει (πχ χρησιμοποιώντας ή όχι cache) ΚΑΙ μπορεί να αποθηκεύει με διαφορετικές τεχνολογίες

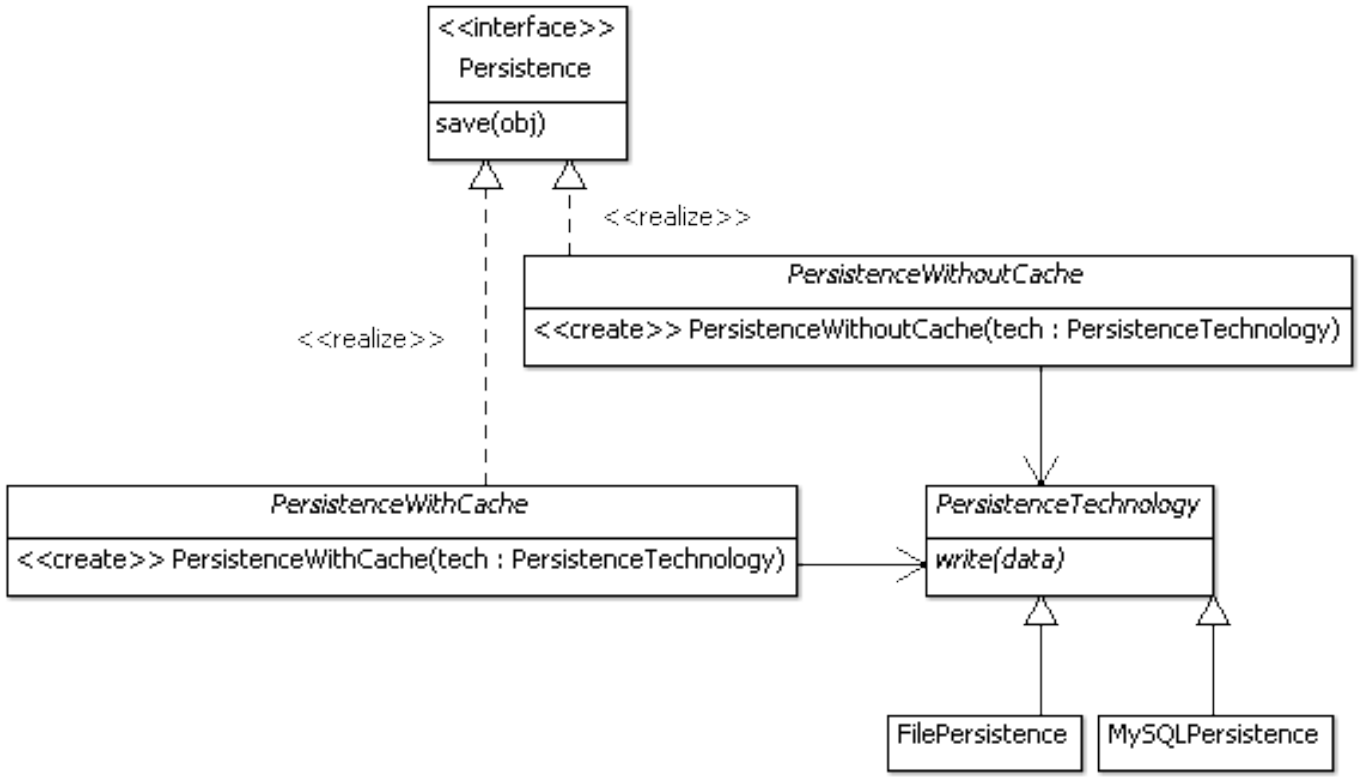
Μία (άσχημη) λύση



Γιατί είναι άσχημη;

- Έχετε δύο διαφορετικές ιεραρχίες
  - μία που αφορά τον τρόπο αποθήκευσης
  - μία που αφορά την τεχνολογία
- έχουν ενσωματωθεί σε μία

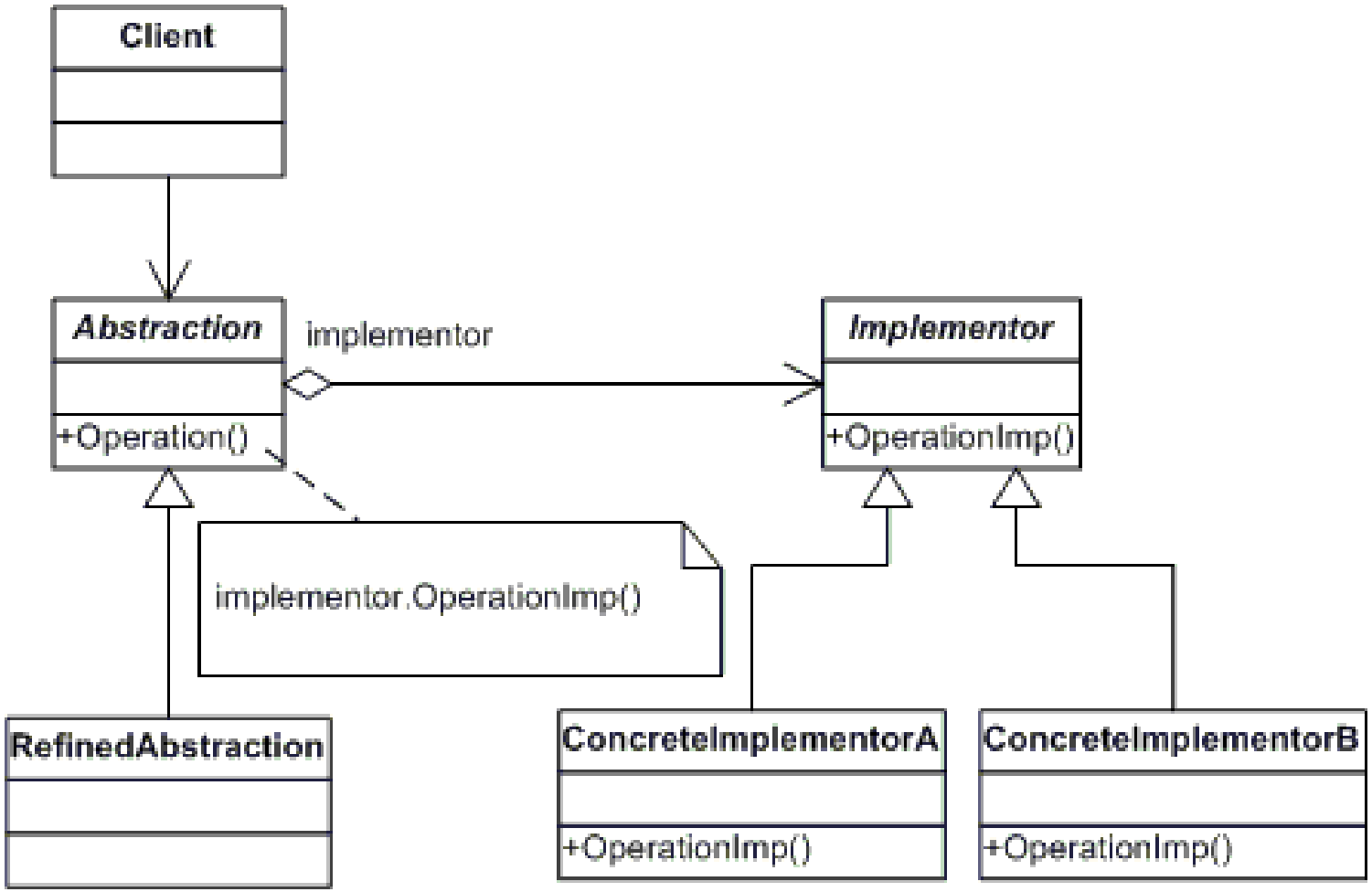
Μία καλύτερη λύση



Ο κώδικας;

```
class PersistenceWithCache {
 public void save()
{
   //do stuff related to chacing
   /decide that it is time to save
   tech.write(data)
}
```

Τυπικά το Bridge pattern



Bridge Pattern

- Χρησιμοποιείται για να αποσυνδέσει ένα interface από μία συγκεκριμένη υλοποίηση έτσι ώστε να μπορεί να εξειδικευθεί περισσότερο και το ένα και το άλλο
- Στο παράδειγμα:

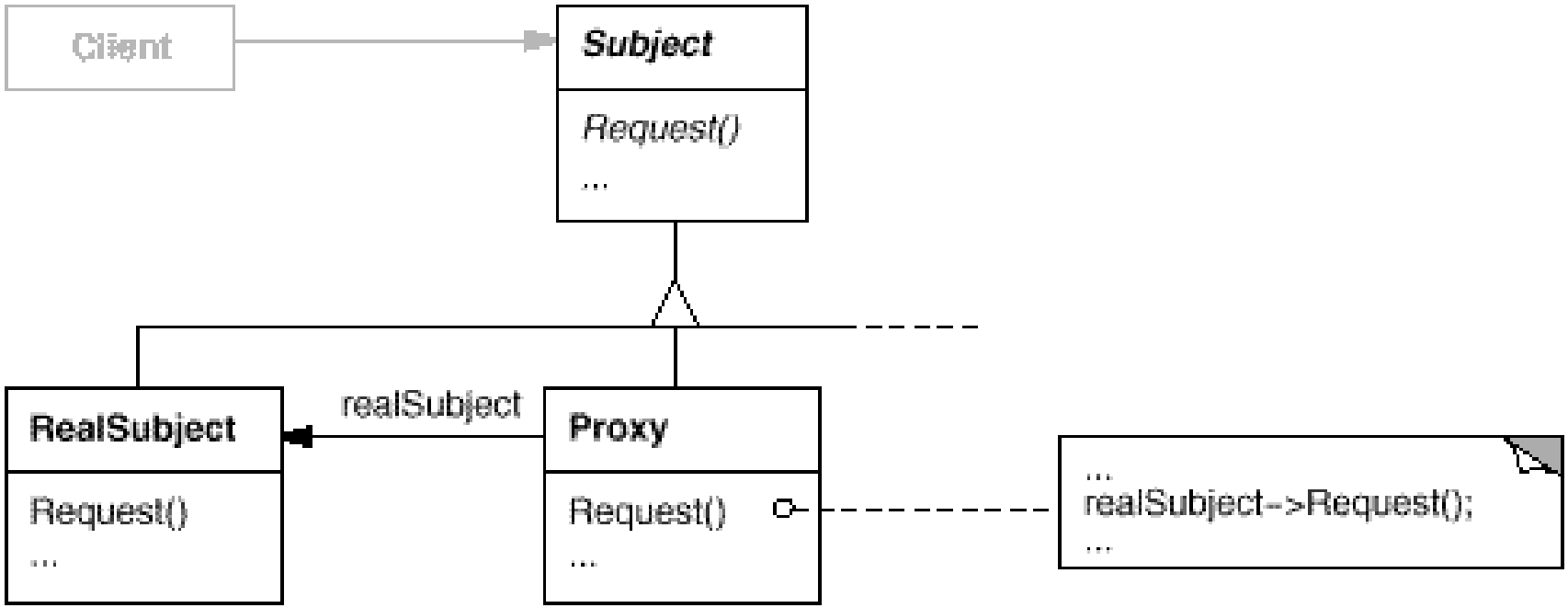
  interface Persistence

  υλοποίηση PersistenceTechnology

Proxy Pattern

- Πολλές φορές δεν είμαστε σίγουροι αν χρειαζόμαστε όλες τις πληροφορίες που περιέχονται σε ένα αντικείμενο μιας κλάσης
- Επιπλέον μπορεί να είναι χρονοβόρο η υπολογιστικά περίπλοκο να βρεθούν αυτές οι πληροφορίες
- Σε αυτήν την περίπτωση μπορούμε να χρησιμοποιήσουμε το μοτίβο Proxy

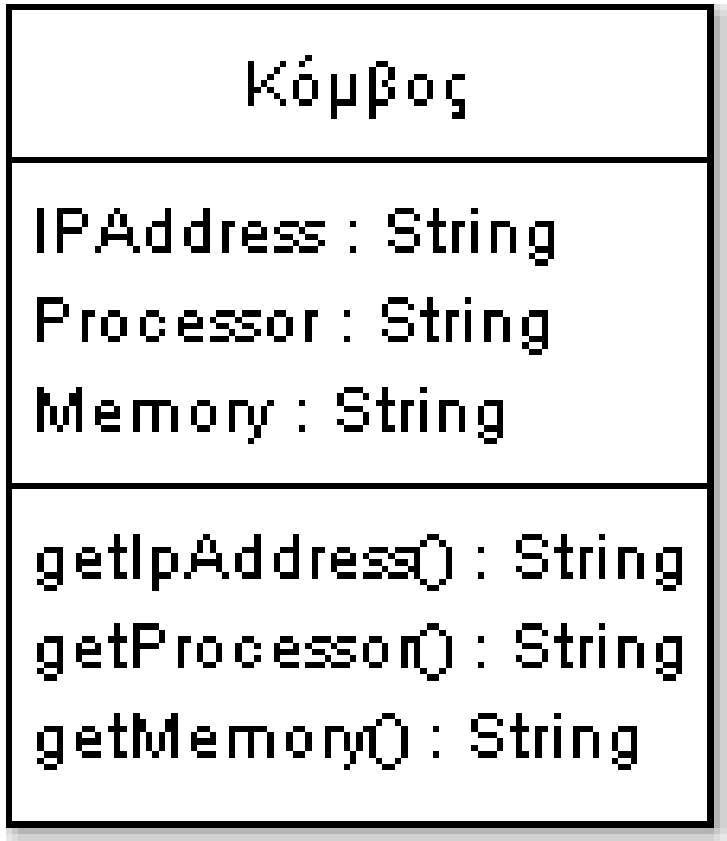
Σχέδιο Proxy



Περιγραφή

- Στο μοτίβο αυτό έστω ότι έχετε την κλάση «Κόμβος» ο οποίος αναπαριστά ένα δικτυακό κόμβο.
- Για να συμπληρωθεί η πληροφορία για ένα αντικείμενο αυτής της κλάσης απαιτείται χρονοβόρος σύνδεση με το δίκτυο

Παράδειγμα


Κόμβος
IPAddress : String
Processor : String
Memory : String
getIpAddress() : String
getProcessor() : String
getMemory() : String

Παράδειγμα

- θέλετε να δείξετε μια λίστα με όλους τους κόμβους του δικτύου, εμφανίζοντας μόνο το IP τους
- Ο χρήστης μπορεί να επιλέξει έναν από όλους τους κόμβους για να δει λεπτομέρειες, όπως επεξεργαστή και μνήμη
- πως θα μπορούσατε να σχεδιάσετε μια λύση όπου δεν καθυστερείτε να δείξετε τους κόμβους μαζεύοντας πληροφορίες που μπορεί ο χρήστης να μη ζητήσει ποτέ;

Σχεδίαση με Proxy Pattern

- Ορίστε ένα interface Node
- Ορίστε μια κλάση ProxyNode που υλοποιεί το Node
- Ορίστε μια κλάση RealNode που υλοποιεί το Node

Υλοποίηση με Proxy Pattern

```
class ProxyNode implements Node {
   private RealNode realNode;
   ...
   public String getMemory() {
      if (realNode==null) {
         realNode = new RealNode();
      }
      return realNode.getMemory();
   }
}
```

Mediator Pattern

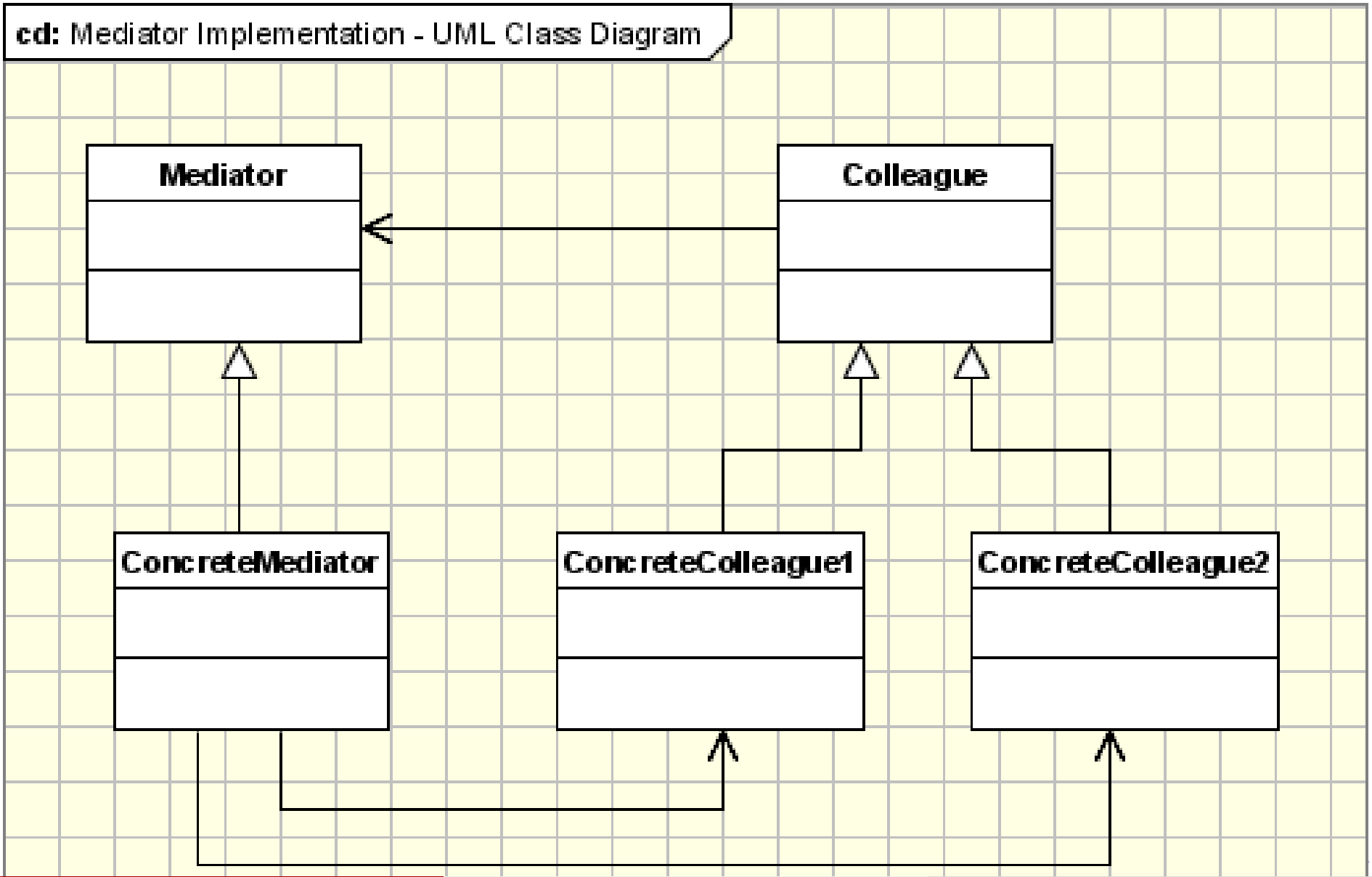
Σκεφθείτε όμως το ακόλουθο παράδειγμα:

- Σε μια φόρμα έχετε ένα TextBox και ένα κουμπί το οποίο πρέπει να γίνει "Enabled" όταν ο χρήστης συμπληρώσει κάτι στο TextBox.
- Η σύζευξη ανάμεσα στις δύο κλάσεις είναι υψηλή αν η TextBox πρέπει να γνωρίζει για το κουμπί που υπάρχει στη φόρμα. Πώς θα προχωρούσατε σε μια σχεδίαση που να μειώνει τη σύζευξη;

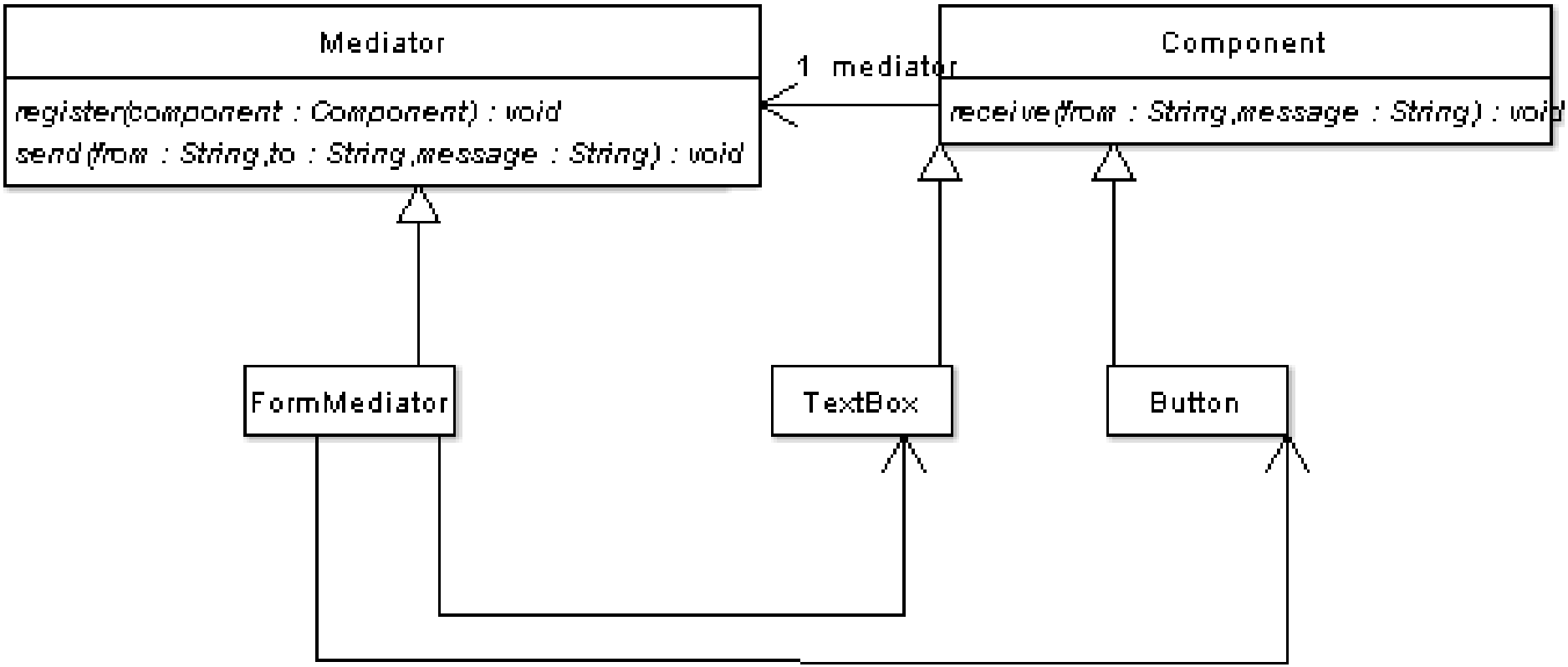
Mediator Pattern

- Μια λύση είναι να ορίσετε μια καινούργια κλάση
- Αυτή είναι υπεύθυνη για να μεταφέρει τα μηνύματα από ένα αντικείμενο μιας κλάσης σε ένα αντικείμενο μιας άλλης.
- Αυτή είναι και η δουλειά του Mediator Pattern

Γενική σχεδίαση mediator σε UML



Συγκεκριμένη σχεδίαση Mediator σε UML



Περιγραφή

- Κάθε κλάση η οποία πρέπει να επικοινωνήσει με άλλες (για παράδειγμα το TextBox και το Button) γνωρίζει το αντικείμενο που θα παίξει το ρόλο του Mediator.
- Κάθε αντικείμενο TextBox ή Button όταν ξεκινά «καταγράφει» (registers) τον εαυτό του στο αντικείμενο της Mediator (FormMediator)
- Στη συνέχεια όποτε θέλει να ενημερώσει άλλα αντικείμενα ενημερώνει πρώτα το FormMediator και αυτός με τη σειρά του τα αντικείμενα που ενδιαφέρονται για το μήνυμα αυτό

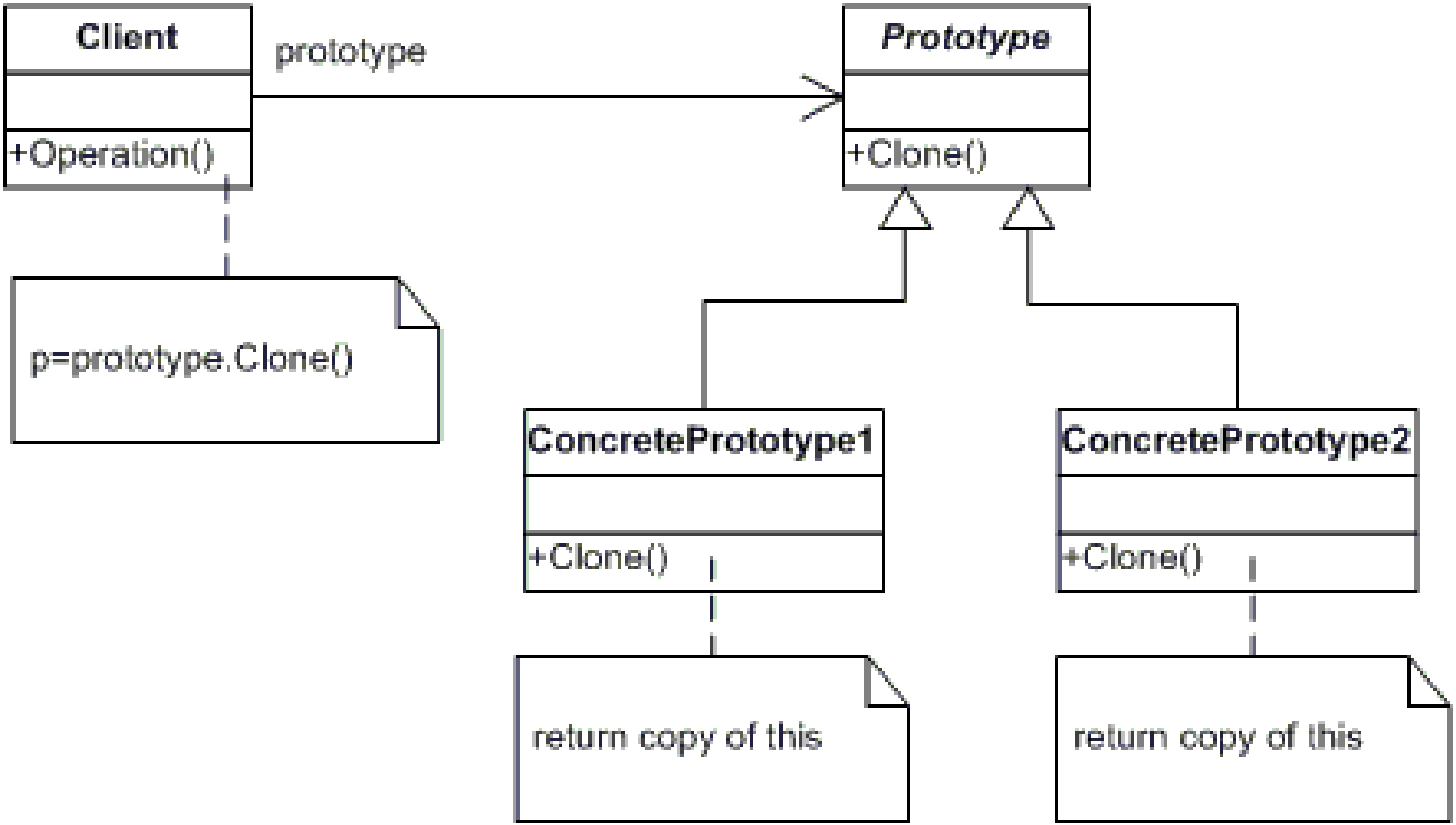
Που χρησιμοποιείται

- Όταν ένα σύνολο από αντικείμενα επικοινωνούν μεταξύ τους, και το αποτέλεσμα είναι πολλές και πολύπλοκες αλληλοεξαρτήσεις που είναι δύσκολο να κατανοηθούν
- Όταν η επαναχρησιμοποίηση ενός αντικειμένου είναι δύσκολη γιατί επικοινωνεί με πολλά αντικείμενα
- Όταν η συμπεριφορά μιας κλάσης είναι διαμοιρασμένη σε πολλές και είναι δύσκολο να επαναχρησιμοποιηθεί

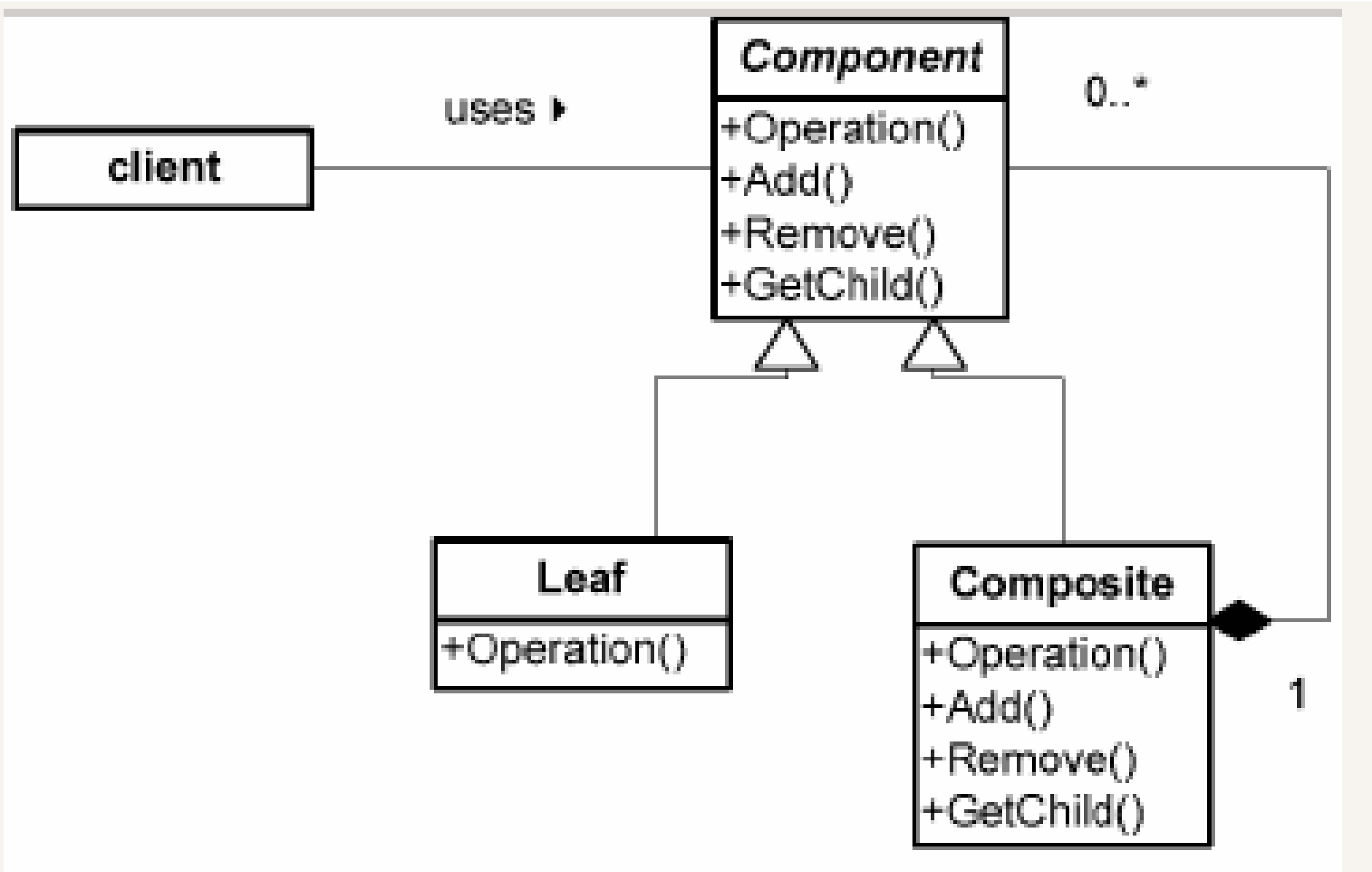
Composite Pattern

- Πολλές φορές οι προγραμματιστές δημιουργούν αντικείμενα που μπορεί να είναι μια συλλογή από άλλα αντικείμενα ή ένα αντικείμενο
- Για παράδειγμα σε ένα ένα δέντρο, ένας κόμβος ή αποτελείται από άλλους κόμβους ή είναι φύλλο
- Θα ήταν ωραίο να έχουμε πρόσβαση σε ένα αντικείμενο είτε είναι κόμβος είτε φύλλο με τον ίδιο τρόπο, δηλ. τις ίδιες μεθόδους.

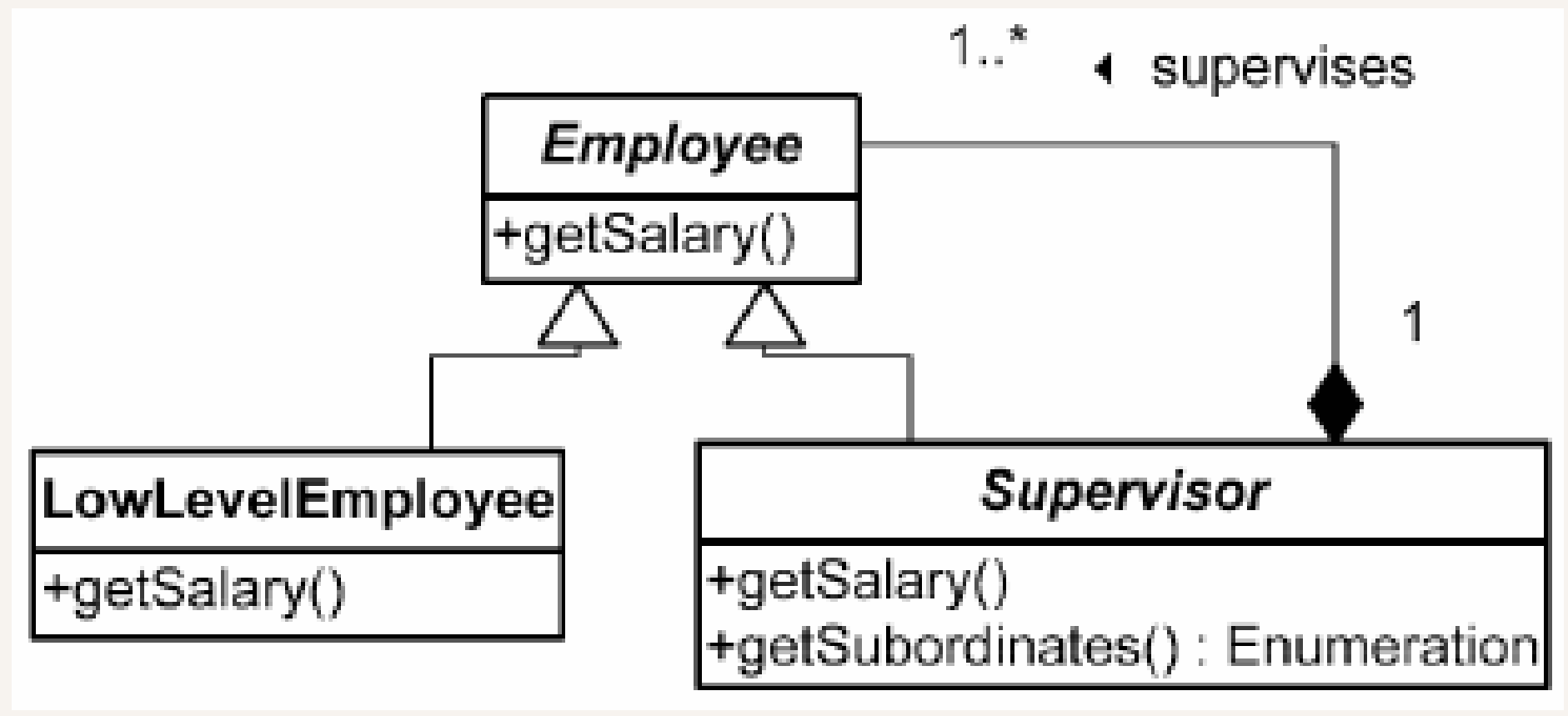
Composite Pattern σε UML



Παράδειγμα Composite Pattern σε UML



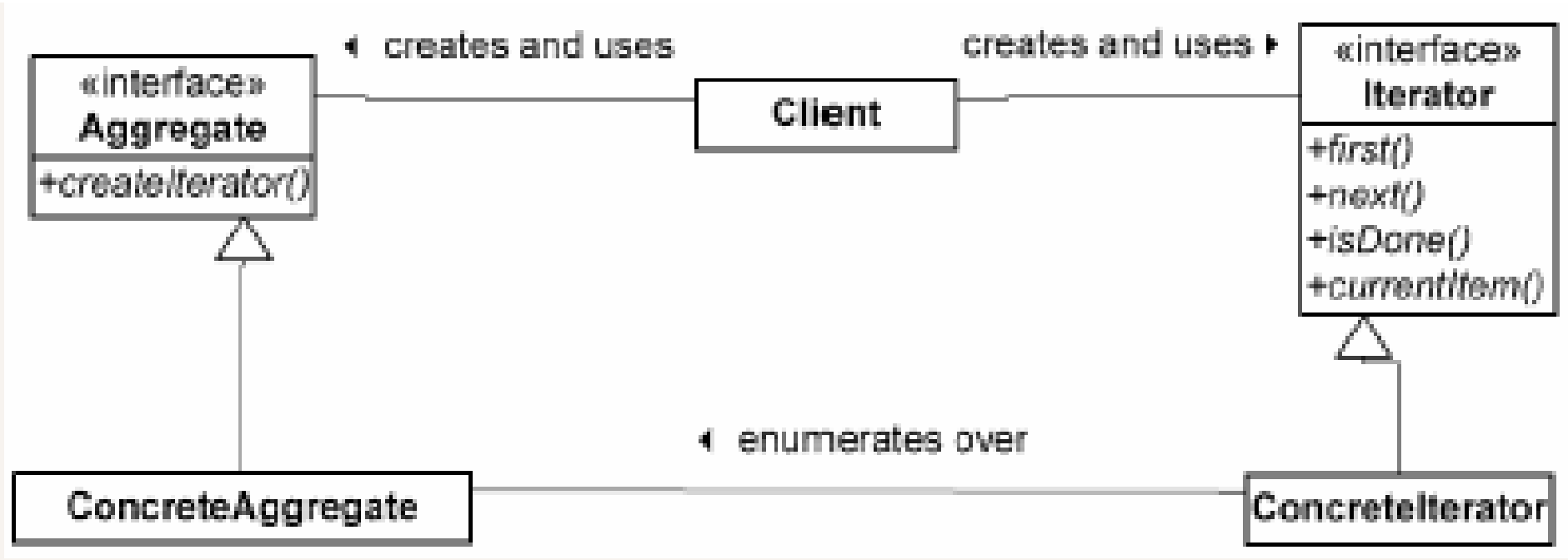
Άλλο Παράδειγμα Composite Pattern σε UML



Iterator Pattern

- Πολλές φορές θέλουμε να διατρέξουμε τα στοιχεία ενός Container είτε είναι λίστα, είτε πίνακας κτλ
- Θα θέλαμε να το κάνουμε αυτό χωρίς να γνωρίζουμε τη δομή του Container και χωρίς να χρειάζεται να προσθέτουμε στο interface του Container θέματα που αφορούν τον τρόπο με τον οποίο το διατρέχουμε

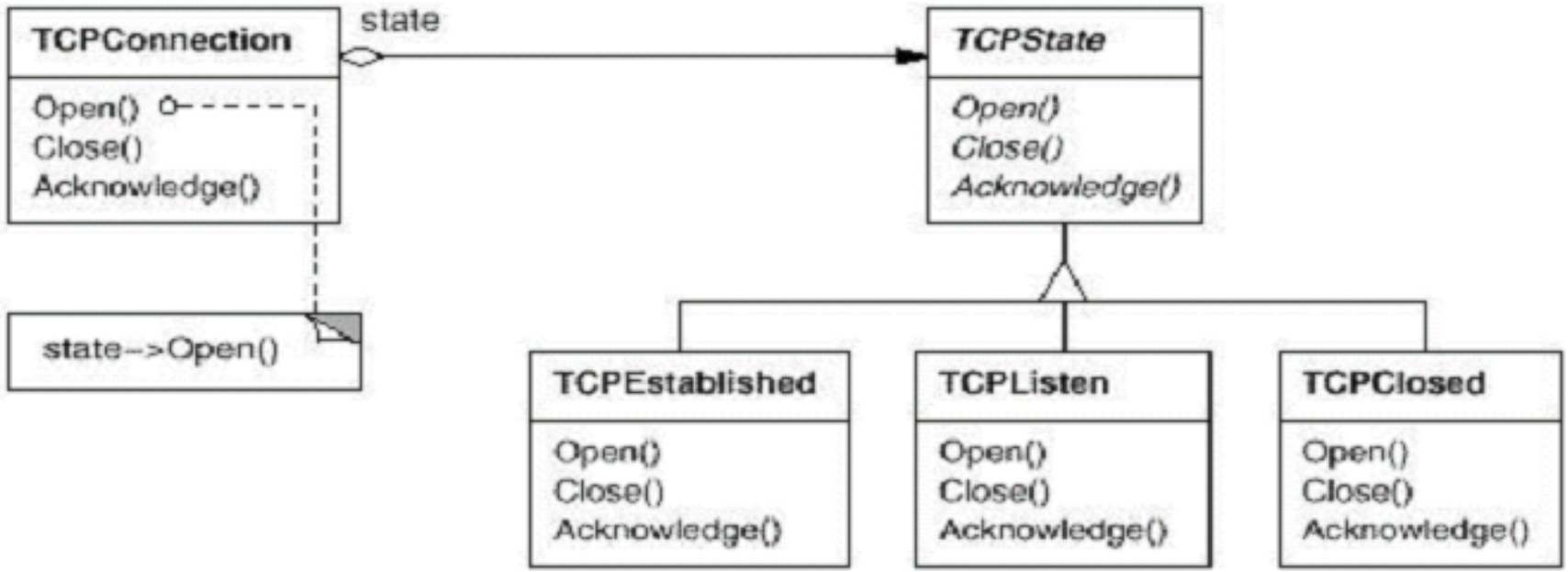
Άλλο Παράδειγμα Iterator Pattern σε UML



State Pattern

- Πολλές φορές έχουμε αντικείμενα τα οποία στη διάρκεια της ζωής τους περνούν από διάφορες καταστάσεις
- Θα θέλαμε να μπορούμε να αλλάξουμε τη συμπεριφορά του στις καταστάσεις από τις οποίες περνά ένα αντικείμενο χωρίς όμως να αλλάξουμε τον κώδικά του.
- Μπορούμε να το πετύχουμε αυτό με το State Pattern

Παράδειγμα State Pattern σε UML



Που χρησιμοποιείται

- Όταν η συμπεριφορά ενός αντικειμένου εξαρτάται από την κατάσταση στην οποία βρίσκεται
- Όταν υπάρχουν πολλά και δύσκολα ifs μέσα στον κώδικα για κάθε μία κατάσταση

Κώδικας χωρίς το state pattern

```
class TCPConnection {
 public void open()
{
 //code for all connections
   if (state == ConnectionEstablshed) {
   .....
   } else if (state == Listen) {
   ....
```

Κώδικας με το state pattern

```
class TCPConnection
   public void changeState(TCPState newState) {
      state = newState;
}
public void open() {
   state.open();
}
class TCPEstablished {
   void open(TCPConnection connection) {
   // do stuff
   //change state if necessary
   connection.changeState(new TCPClosed());
}
```