Μεθοδολογία Προγραμματισμού

Abstract Κλάσεις και Interfaces

Νικόλαος Πεταλίδης

Τμήμα Μηχανικών Πληροφορικής και Επικοινωνιών
Διεθνές Πανεπιστήμιο της Ελλάδος

Εισαγωγή
Εαρινό Εξάμηνο

Generics

- Γράψτε ένα πρόγραμμα σε Java που υλοποιεί μια στοίβα από ακεραίους
- Γράψτε ένα πρόγραμμα σε Java που υλοποιεί μια στοίβα από String
- Σε τι διαφέρουν;

Generics

- Υπάρχουν στην Java από την έκδοση 1.5.
- Επιτρέπουν τον ορισμό κλάσεων χωρίς να αναφερόμαστε σε συγκεκριμένους τύπους
- Κάνουν τον κώδικα πιο καθαρό και πιο ασφαλή

Παραδείγματα

- Χωρίς generics

```
List myIntList = new LinkedList();
myIntList.add(new Integer(0));
Integer x = (Integer) myIntList.iterator().next();
```

- Με Generics

```
List<Integer> myIntList = new LinkedList<Integer>(
myIntList.add(new Integer(0));
Integer x = myIntList.iterator().next();
```

Παράδειγμα ορισμού Generics

```
public interface List<E> { void add(E x);
Iterator<E> iterator();
}

public interface Iterator<E> { E next();
boolean hasNext();
}
```

Επεξήγηση

- Το <E> είναι η παράμετρος
- Αντικαθίσταται από μια συγκεκριμένη κλάση όταν δημιουργούμε ένα αντικείμενο
- Μια κλάση που έχει generics μεταγλωτίζεται μόνο μια φορά
- Κατά σύμβαση τα ονόματα των παραμέτρων είναι μονά γράμματα και κεφαλαία

Περισσότερα παραδείγματα

```
class Pair <T> {
   public T first;
   public T second;
   public Pair (T f, T s) {
      first = f;
      second = s;
   }
   public Pair () {
     first = null;
     second = null;
  }
}
}
```

Πολλαπλές παράμετροι

```
class Pair2 <T,U> {
   public T first;
   public U second;
   public Pair2 (T f, U s) {
      first = f;
      second = s;
   }
   public Pair2 () {
    first = null;
    second = null;
  }
}
}
```

Δημιουργία Pair

```
Pair <String> pair =
  new Pair <String> ("1","2");
Pair2 <String,Integer> pair2 =
  new Pair2 <String,Integer> ("1",2);
```

Generic static algorithms

- Μπορεί κάποιος να ορίσει και generic μεθόδους

```
class Algorithms {
   public static <T> T getMiddle (T [ ] a) {
   return a [ a.length / 2 ];
  }
}
```

Καλώντας generic μεθόδους

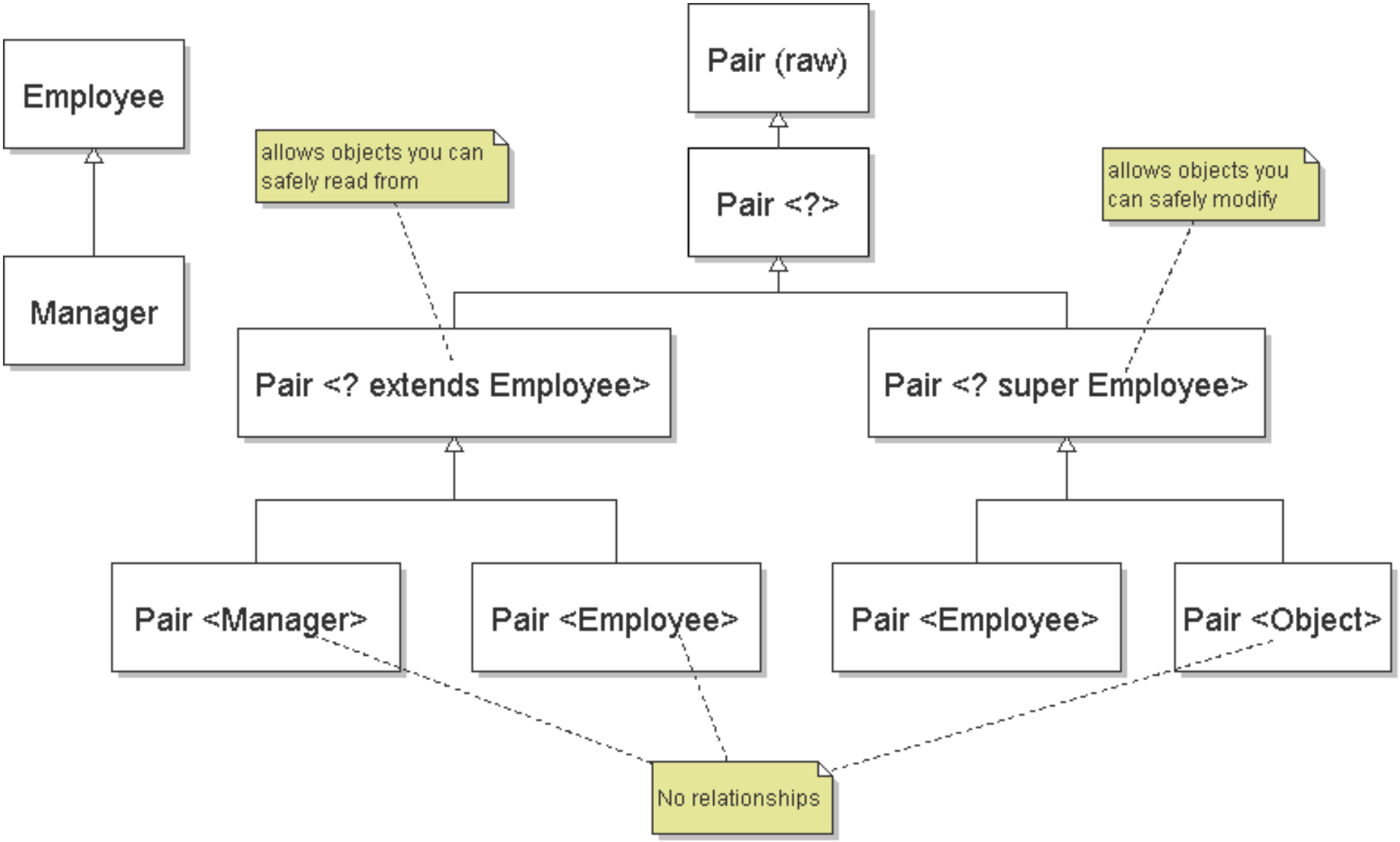
- Ορίζοντας τον τύπο

```
String s = Algorithms.<String>getMiddle(names);
```

- ή αφήνοντας τον compiler να τον καταλάβει

```
String s = Algorithms.getMiddle (names);
```

Κανόνες κληρονομικότητας



Επεξήγηση κανόνων

- Pair <Manager> ταιριάζει Pair <? **extends** Employee>
- Pair < Object > ταιριάζει Pair <? **super** Employee>

Unbounded wildcards

- Μερικές φορές θέλουμε να γράψουμε μεθόδους που δε χρησιμοποιούν ιδιότητες κάποιες συγκεκριμένης κλάσης. Για παράδειγμα θέλουμε να τυπώσουμε τα στοιχεία μιας λίστας

```
public void printStuff(Iterable<?> stuff) {
  for (Object item : stuff) {
    System.out.println(item);
  }
}
```

- Δείτε τη διαφορά

```
Collection<?> c = new ArrayList<String>();
c.add("foo"); //fails
Collection c2 = new ArrayList<String>();
c2.add("foo"); //ok
c2.add(new Integer(300)); //ok
c2.add(new Object()); //ok
```

Type erasure

- Το JVM δε γνωρίζει για generic types
- Ένας generic ορισμός τύπου μεταγλωττίζεται μία φορά και ένας απλός τύπος παράγεται
- Τι ξέρει το JVM για το Pair <T>

```
class Pair {
    public Object first;
    public Object second;
    public Pair(Object f, Object s) {...}
}
```

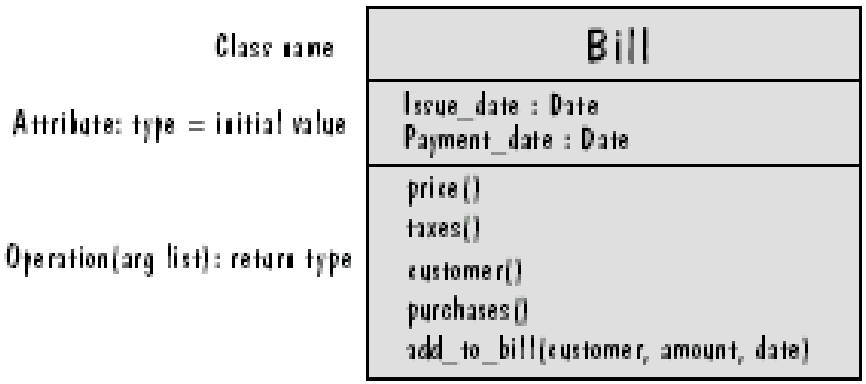
Unified Modeling Language

- Απόγονος των αντικειμενοστραφών μεθόδων ανάλυσης και σχεδιασμού που εμφανίστηκαν στα τέλη της δεκαετίας του 80
- Ενοποιεί τις μεθόδους των Booch, Rambaugh και Jacobson

Τι μας επιτρέπει να κάνουμε

- Να παράγουμε μοντέλα αντικειμενοστραφών προγραμμάτων με εύκολο τρόπο χωρίς να μπλεκόμαστε στις λεπτομέρειες γλωσσών προγραμματισμού όπως η C++ και η Java
- Μπορεί εύκολα ένα μοντέλο σε UML να μεταφραστεί σε μια αντικειμενοστραφή γλώσσα προγραμματισμού

Αναπαράσταση κλάσης στη UML

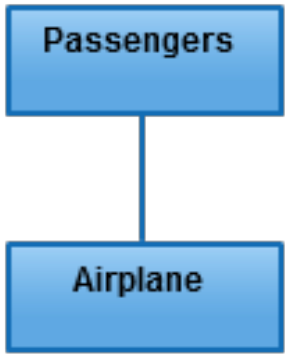


- Δεδομένα (Κατάσταση, χαρακτηριστικά)
- Συμπεριφορά (Ενέργειες, λειτουργίες, μετασχηματισμοί

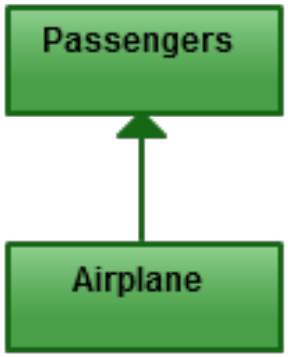
Συσχετίσεις

- Οι κλάσεις μπορεί να συσχετίζονται μεταξύ τους
- Η UML αναπαριστά τη συσχέτιση μεταξύ δύο κλάσεων πολύ απλά με μια γραμμή

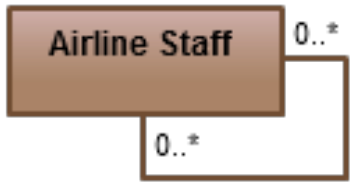
Συσχετίσεις - Παράδειγμα



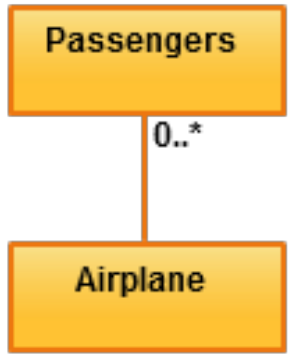
Association



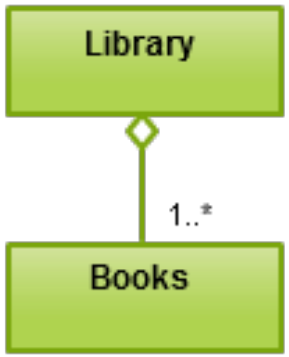
Directed Asscoation



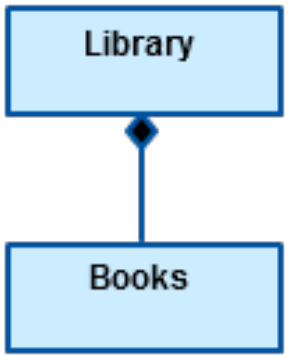
Reflexive Assciation



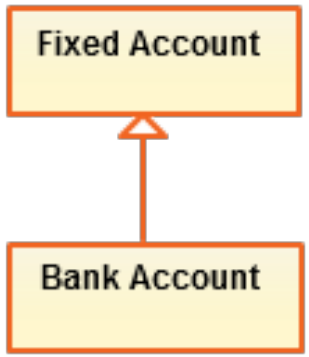
Multiplicity



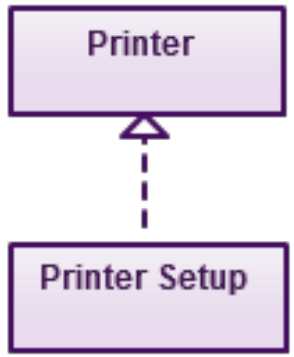
Aggregation



Composition



Inheritance



Realization

Απλή συσχέτιση - Association

```
class Passengers {
   Airplane airplane;
}

class Airplane {
    Passengers passengers;
}
```

Κατευθυνόμενη συσχέτιση - Directed Association

```
class Passengers {
}

class Airplane {
    Passengers passengers;
}
```

Αυτό-συσχέτιση Reflexive Association

```
class AirlineStaff {
    List<AirlineStaff> staff;
}
```

Συσχέτιση με πολλαπλότητα - Multiplicity

```
class Airplane {
    List<Passengers> passengers;
}
```

Συσσωμάτωση (Aggregation)

- Όταν μια κλάση είναι τμήμα μιας άλλης κλάσης

```
class Library {
    List<Book> books;
     //Hey! Same as Association
}
```

Σύνθεση (Composition)

- Όταν μια κλάση είναι ΠΑΝΤΑ αναπόσπαστο κομμάτι μιας άλλης κλάσης

```
class Library {
    List<Book> books;
     //Hey! Same as Association
}
```

Συσσωμάτωση - Σύνθεση: Που χρειάζονται

- ... πουθενά (αν απλώς θέλετε κάτι στα γρήγορα)!
- Είναι χρήσιμοι σε περιπτώσεις όπου θέλουμε να δείξουμε λεπτομέρειες αλλά πάντα μπορούν να αντικατασταθούν από μια συσχέτιση
- Η σύνθεση χρησιμοποιείται για να τονίσουμε ότι η διαγραφή μιας κλάσης συνεπάγεται και τη διαγραφή της άλλης

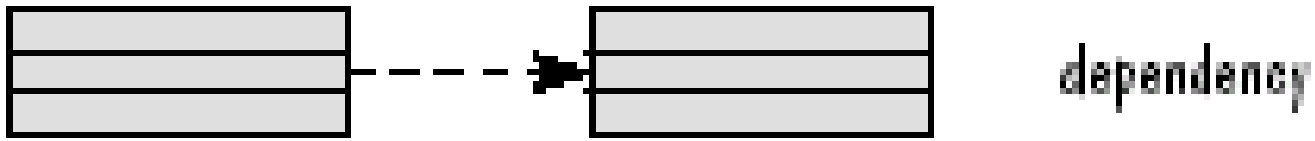
Κληρονομικότητα (Inheritance)

```
class BankAccount
    extends FixedAccount {
}
```

Υλοποίηση (Realization)

```
class PrinterSetup implements
    Printer {
}
```

Εξαρτήσεις



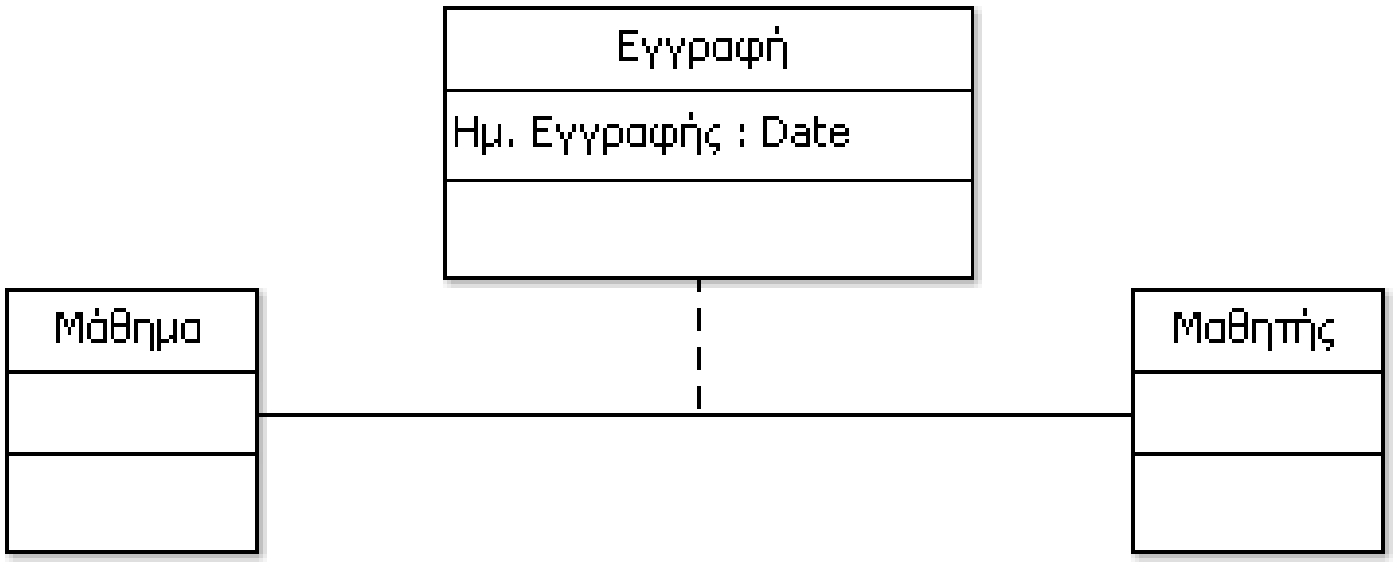
- Εξάρτηση υπάρχει αν μια κλάση χρησιμοποιεί μια άλλη κλάση κατά κάποιο τρόπο

```
class MyFile {
   public: MyFile(string filename);
   private: FILE *fp;
}
```

Κλάσεις συσχέτισης

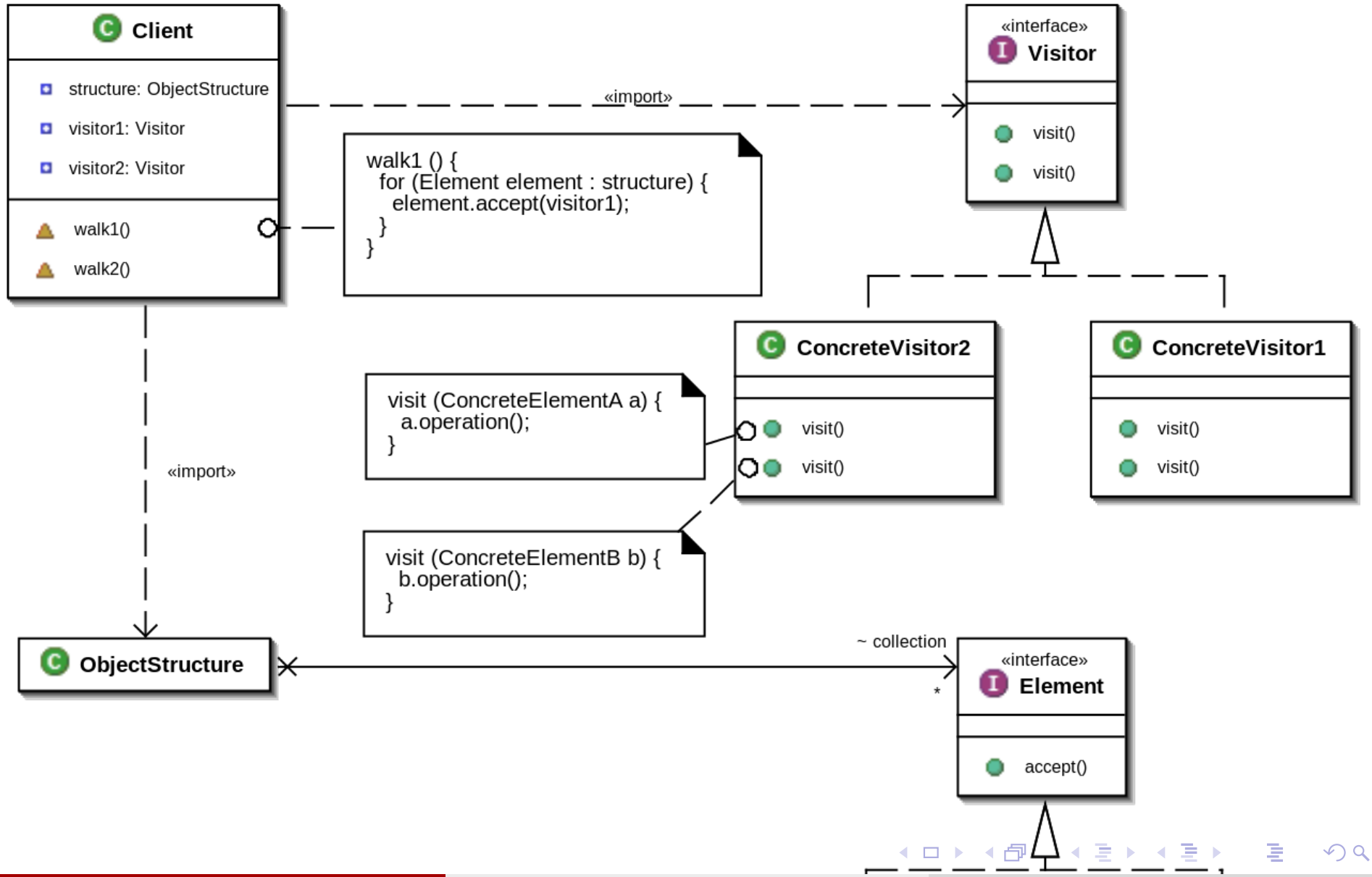
- Πολλές φορές θέλουμε να κρατήσουμε το "ιστορικό" της συσχέτισης μιας κλάσης με μια άλλη.
- Για παράδειγμα ποιες ημερομηνίας εξετάστηκε ένας φοιτητής σε ένα μάθημα. Αυτή η πληροφορία δεν αφορά ούτε το μάθημα ούτε το φοιτητή αλλά τη σχέση που έχουν αυτοί μεταξύ τους.
- Τέτοιες πληροφορίες μπορεί να αποθηκευτούν σε μια κλάση συσχέτισης

Κλάσεις συσχέτισης




```
class Registration {
    Lesson lesson;
    Student student;
}
```

Παράδειγμα


Client
structure: ObjectStructure
visitor1: Visitor
visitor2: Visitor
walk1()
walk2()
«import»
«interface»
Visitor
visit()
visit()
walk1 () {
for (Element element : structure) {
element.accept(visitor1);
}
}
ConcreteVisitor2
visit()
visit()
ConcreteVisitor1
visit()
visit()
visit (ConcreteElementA a) {
a.operation();
}
visit (ConcreteElementB b) {
b.operation();
}
«import»
ObjectStructure
~ collection
*
«interface»
Element
accept()