Από τη UML στον Κώδικα

Μέρος Β

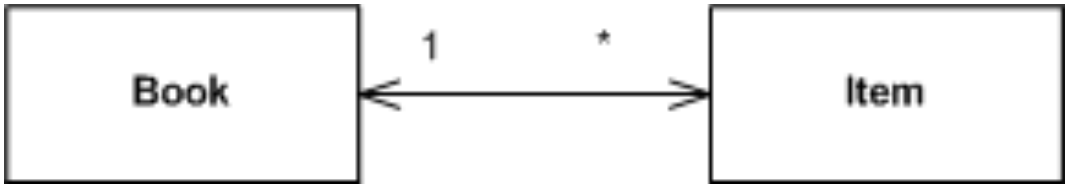
περιεχόμενα παρουσίασης

- Αμφίδρομες συσχετίσεις
- Συσσωμάτωση
- Σύνθεση
- Διαγράμματα ακολουθίας

αμφίδρομες συσχετίσεις

- Μία αμφίδρομη συσχέτιση υλοποιείται με δύο μονόδρομες. Υπάρχει όμως μία πολύ σημαντική διαφορά. Όλες οι αλλαγές στη μία μονόδρομη συσχέτιση θα πρέπει να απεικονίζονται στην άλλη.
- Μία αμφίδρομη συσχέτιση μεταξύ δύο κλάσεων σημαίνει αυτόματα ότι οι δύο κλάσεις είναι αμοιβαία εξαρτώμενες.
- Η σύζευξη των κλάσεων με αμφίδρομη συσχέτιση είναι υψηλότερη της σύζευξης με μονόδρομη. Οι αμφίδρομες συσχετίσεις δυσχεραίνουν και τη συντήρηση του λογισμικού.

αμφίδρομες συσχετίσεις




```
public class Item {
    private Book book;
    public Book getBook() { return book; }
}

public class Book {
    private Set<Item> items = new HashSet<Item>();
    public void addItem(Item item) {//…}
    public void removeItem(Item item) {//…}
}
```

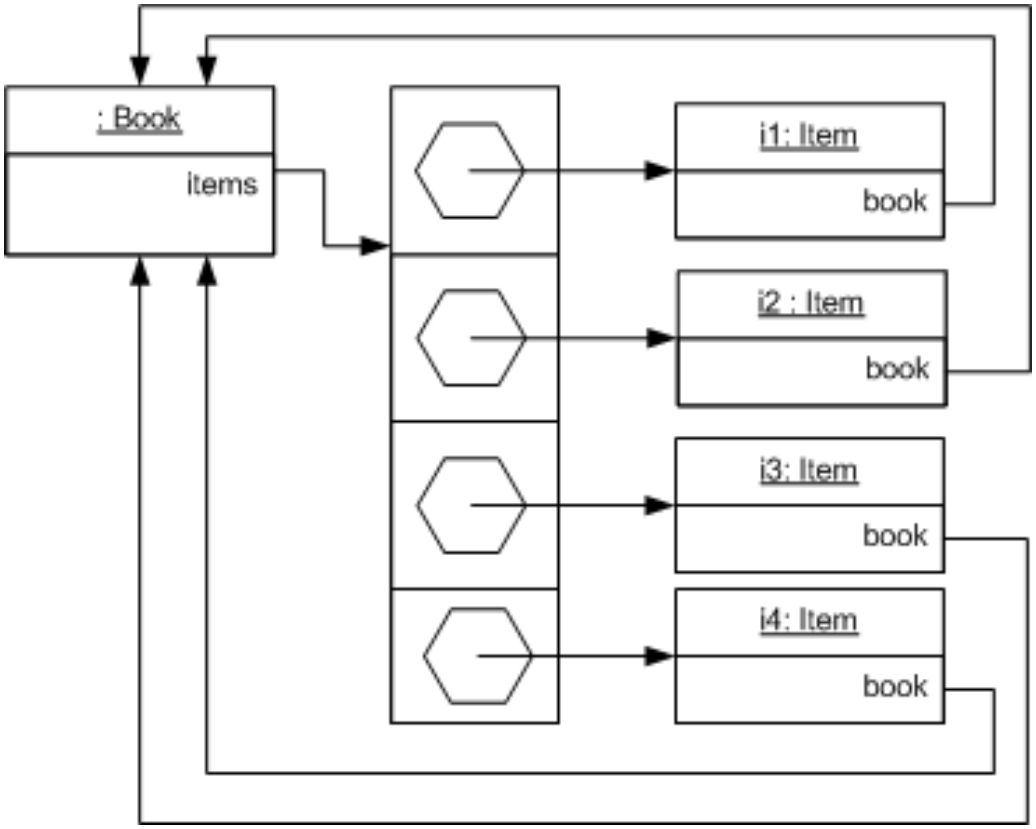
αμφίδρομες συσχετίσεις

```
public class Item {
    private Book book;

    public void setBook(Book book) {
        if (this.book != null) {
            this.book.friendItems().remove(this);
        }
        this.book = book;
        if (this.book != null) {
            this.book.friendItems().add(this);
        }
    }

    public Book getBook() {
        return book;
    }
}
```

αμφίδρομες συσχετίσεις

```
public class Book {
    private Set<Item> items = new HashSet<Item>();

    public Set<Item> getItems() {  return new HashSet<Item>(items);   }

    public void addItem(Item item) {
        if (item != null) {  item.setBook(this); }
    }

    public void removeItem(Item item) {
        if (item != null) {  item.setBook(null); }
    }

    Set<Item> friendItems() {
        return items;
    }
}
```

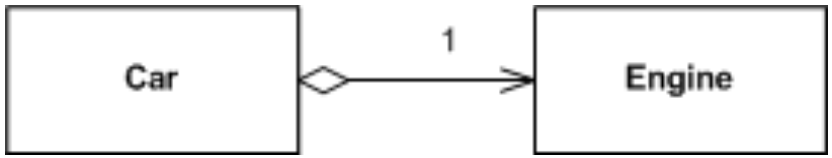
αμφίδρομες συσχετίσεις



συσσωμάτωση

- Η διαφορά μεταξύ της απλής συσχέτισης και της συσσωμάτωσης είναι περισσότερο εννοιολογική.
- Η Java δεν προσφέρει κάποια διάκριση μεταξύ της απλής συσχέτισης και της συσσωμάτωσης.
- Αν ο σχεδιαστής χρησιμοποιήσει τη συσσωμάτωση στη σχεδίαση, θα πρέπει να συμφωνήσει με τους προγραμματιστές για την ερμηνεία της συσσωμάτωσης κατά την υλοποίηση.
- Μία πιθανή εννοιολογική ερμηνεία της συσσωμάτωσης (χωρίς να είναι και η μόνη) είναι ότι το όλο δεν μπορεί να λειτουργήσει χωρίς το τμήμα του.

συσσωμάτωση



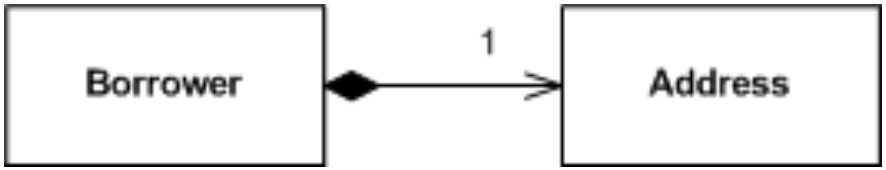

```
public class Car {
    Engine engine;

    public void drive() throws CarException {
        if (engine == null ) {
            throw new CarException();
        }
        // ο κώδικας της οδήγησης
    }
}
```

Δεν μπορούμε να οδηγήσουμε ένα αυτοκίνητο χωρίς τον κινητήρα του, οπότε η κλήση της μεθόδου drive, χωρίς να υπάρχει κινητήρας, δίδει εξαίρεση.

σύνθεση

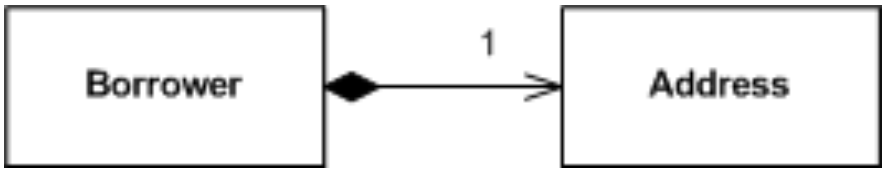
- Η σύνθεση είναι μία ιδιαίτερη περίπτωση της συσχέτισης όλου / τμήματος, μόνο που συνοδεύεται με κάποιους περιορισμούς.
- Οι περιορισμοί είναι ότι το αντικείμενο τμήμα ανήκει αποκλειστικά σε ένα αντικείμενο όλο και ότι το αντικείμενο όλο διαχειρίζεται πλήρως τον κύκλο ζωής του αντικειμένου τμήμα.

σύνθεση (μεταβίβαση)




```
public class Borrower {
  private Address address = new Address();

  public void setStreet(String street) { address.setStreet(street);  }

  public String getStreet() { return address.getStreet();  }

  public void setNumber(String number) {address.setNumber(number); }

  public String getNumber() { return address.getNumber();  }
//Όλες οι μέθοδοι πρόσβασης της Address
}
```

σύνθεση (αντίγραφα αντικειμένων)

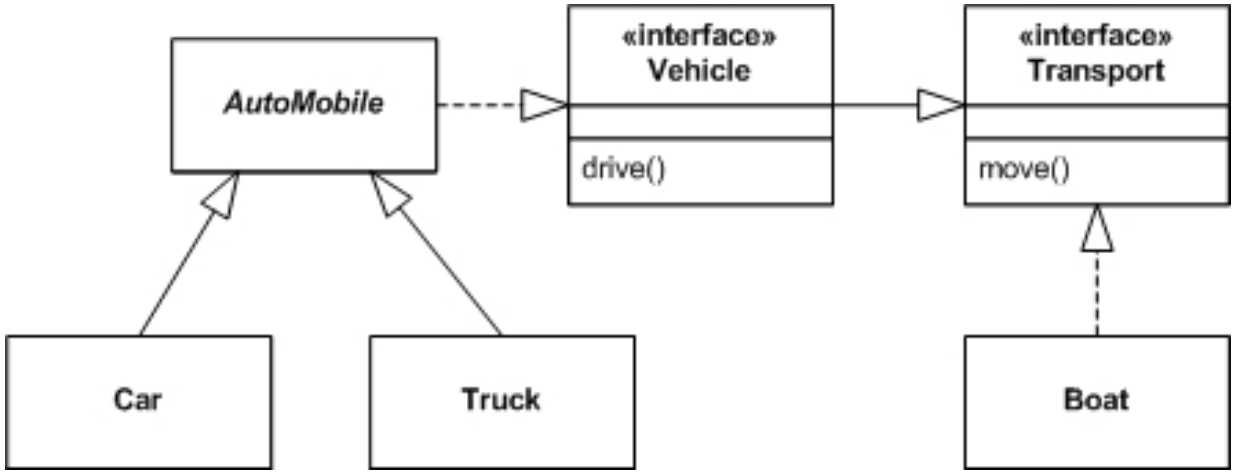



```
public class Borrower  {
    private Address address;

    public void setAddress(Address address) {
        this.address = address == null ? null : new Address(address);
    }

    public Address getAddress() {
        return address == null ? null : new Address(address);
    }
}
```

κληρονομικότητα και διεπαφές

- Η χρήση της κληρονομικότητας, της υλοποίησης και των διεπαφών παρέχει συγκεκριμένες οδηγίες στον προγραμματιστή για τη μετάβαση από το σχέδιο στον κώδικα.
- Η μεταφορά της κληρονομικότητας και της υλοποίησης των διεπαφών από τη UML στη Java είναι και πάλι σχεδόν αυτόματη. Η μεγαλύτερη ίσως ασυνέπεια μεταξύ της UML και της Java είναι ότι η Java δεν υποστηρίζει την πολλαπλή κληρονομικότητα.

κληρονομικότητα και διεπαφές



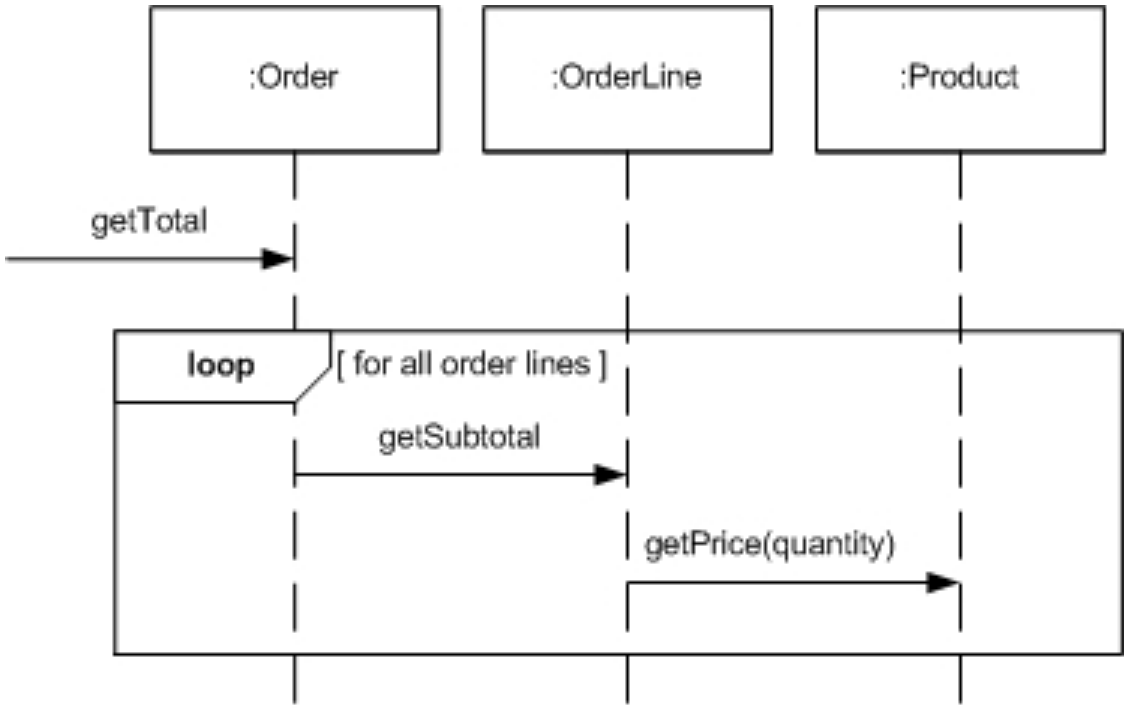
κληρονομικότητα και διεπαφές

```
public interface Transport {
    public void move();
}
public interface Vehicle extends Transport {
    public void drive();
}

public abstract class AutoMobile implements Vehicle {

    public void drive() { ....    }

    public void move() { .... }
}
```

κληρονομικότητα και διεπαφές

```
public class Car extends AutoMobile  {
    // Η κλάση Car κληρονομεί την υλοποίηση  της μεθόδου drive.
    // Η μέθοδος drive μπορεί να επαναοριστεί
}

public class Truck extends AutoMobile {
    // Η κλάση Truck κληρονομεί την υλοποίηση της μεθόδου drive.
    // Η μέθοδος drive μπορεί να επαναοριστεί
}

public class Boat implements Transport{
    public void move() {
    // Η κλάση Boat υλοποιεί τη μέθοδο move }
}
```

διαγράμματα ακολουθίας



διαγράμματα ακολουθίας

```
public class Order {
    private Set<OrderLine> orderLines = new HashSet<OrderLine>();

    public int getTotal() {
        int total = 0;
        for(OrderLine orderLine : orderLines)  {
            total += orderLine.getSubTotal();
        }
        return total;
    }
}
```

διαγράμματα ακολουθίας

```
public class OrderLine  {
     private int quantity;    private Product product;

     public int getSubTotal() {
       return product.getPrice(quantity);
     }
}

public class Product {
     private int price;

   public int getPrice(int quantity) {
        return price * quantity;
   }
}
```