Από τη UML στον Κώδικα

Μέρος Α

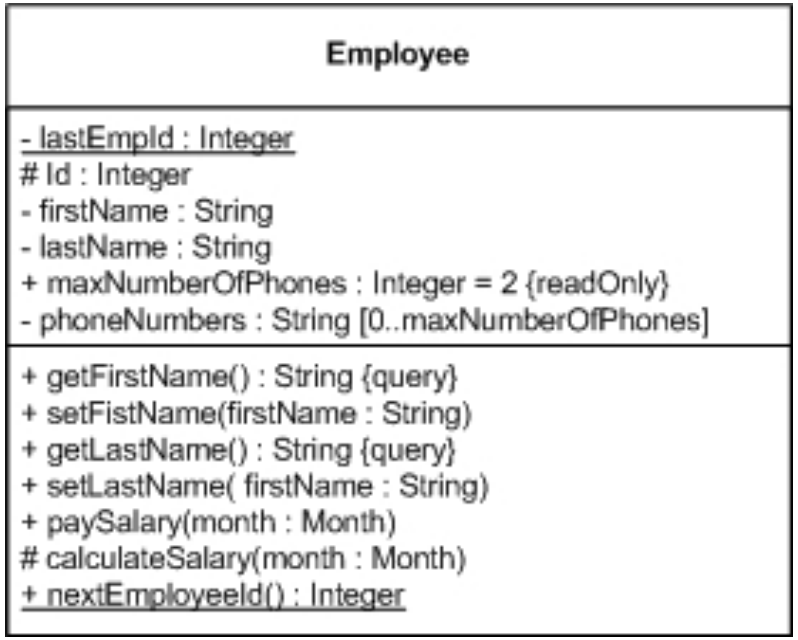
περιεχόμενα παρουσίασης

- Κλάσεις
- Ισότητα αντικειμένων
- Μονόδρομες συσχετίσεις με πολλαπλότητα «ένα»
- Μονόδρομες συσχετίσεις με πολλαπλότητα «πολλά»
- Συλλογές

από το σχέδιο στον κώδικα

- Όταν ο σχεδιαστής του λογισμικού προσεγγίζει τη σχεδίαση των βασικών δομικών μονάδων του λογισμικού, επικοινωνεί τις σχεδιαστικές του επιλογές στον προγραμματιστή.
- Επειδή ακριβώς η σχεδίαση με τα διαγράμματα κλάσεων της UML είναι πολύ κοντά και στον προγραμματισμό, ο σχεδιαστής του λογισμικού θα πρέπει να γνωρίζει πώς μεταφέρεται το σχέδιο στον κώδικα.
- Επίσης, θα πρέπει σε περιπτώσεις όπου η UML αφήνει περιθώρια ερμηνειών των στοιχείων μοντελοποίησης, να υπάρξει μία εκ των προτέρων συμφωνία μεταξύ σχεδιαστή και του προγραμματιστή για την ερμηνεία τους.

κλάσεις

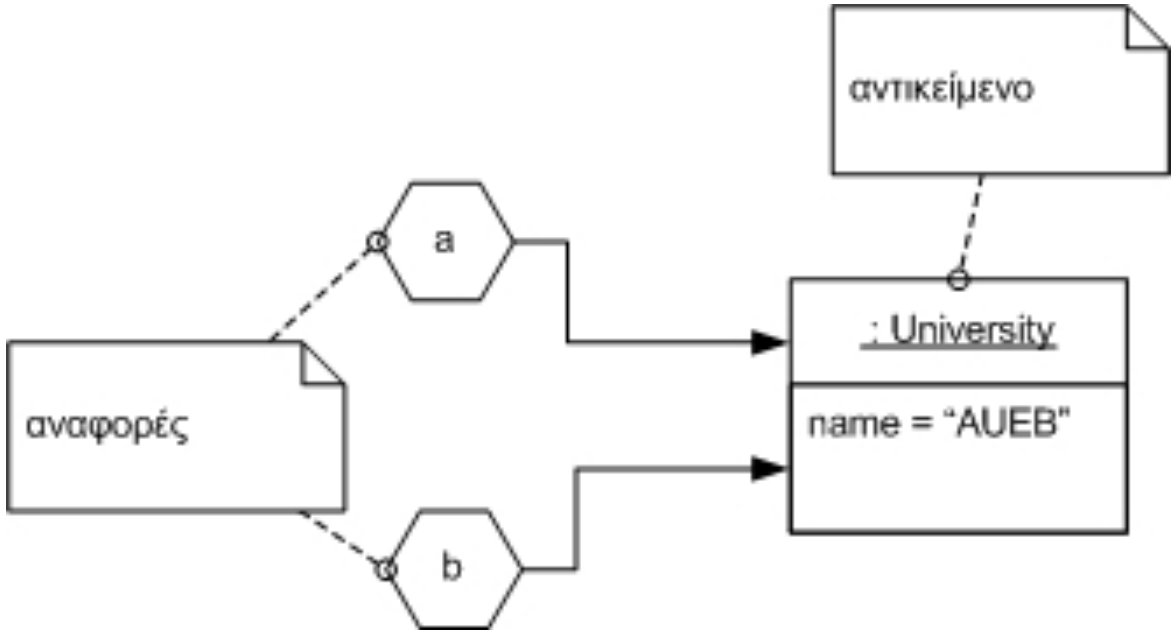



```
public class Employee {
      private static int lastEmpId ;
      protected int Id;
      private String firstName;
      private String lastName;


      public final int maxNumberOfPhones = 2;
      private String[] phoneNumbers = new
String [maxNumberOfPhones];


     public String getFirstName() {}
      public void setFistName(String firstName) {}
      public String getLastName() {}
      public void setLastName(String lastName){}
      public void paySalary(Month month){}
      protected void calculateSalary(Month
      month) {}
      public static int nextEmployeeId(){}
}
```

κλάσεις

- Η υλοποίηση των κλάσεων, των ιδιοτήτων και των λειτουργιών της UML είναι σχεδόν αυτόματη διαδικασία. Οι κλάσεις της UML μετατρέπονται σε κλάσεις της Java, οι ιδιότητες γίνονται πεδία και οι λειτουργίες μέθοδοι.
- Ένα σημείο που χρήζει κάποιας προσοχής είναι οι ορατότητες της UML. Οι ορατότητες, όπως ορίζονται στη UML, έχουν όμοια αντιστοιχία με τις ορατότητες της Java με κάποιες λεπτές διαφορές. Για παράδειγμα η προστατευμένη (protected) ορατότητα της Java είναι ταυτόχρονα και ορατότητα πακέτου (package).

ισότητα αντικειμένων

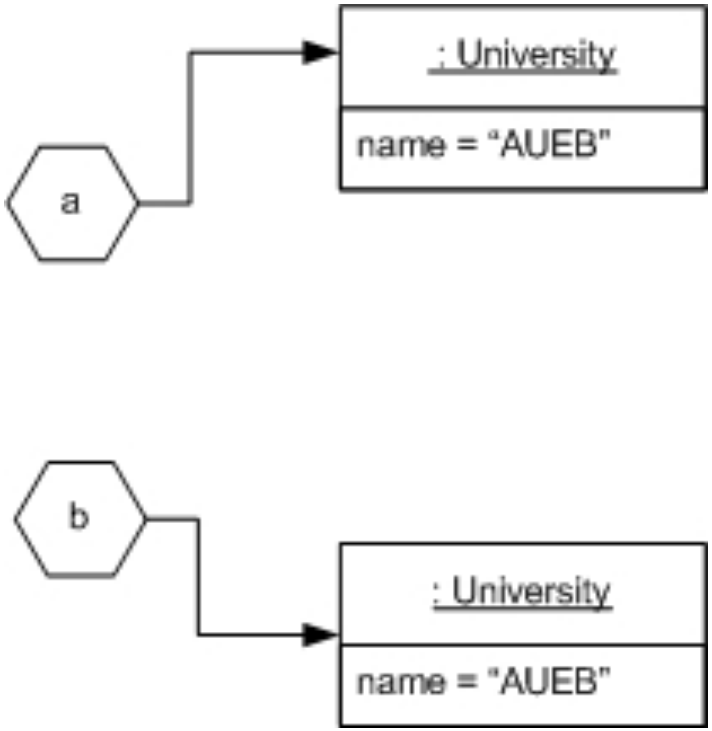


University a = new University();

a.setName("AUEB");

University b = a;

ισότητα αντικειμένων



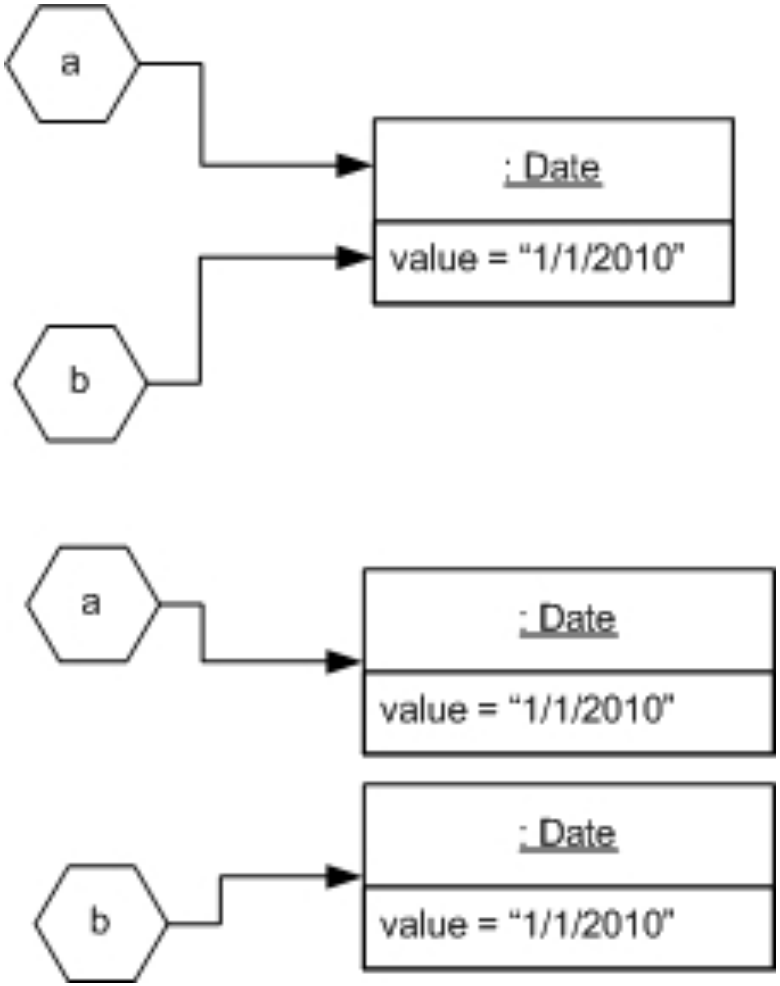
University a = new University();

a.setName("AUEB");

University b = new University();

b.setName("AUEB”);

ισότητα αντικειμένων



Όταν συγκρίνουμε δύο αναφορές σε αντικείμενα απλών τύπων, πχ ημερομηνιών, μας ενδιαφέρει εάν οι δύο αναφορές αναφέρονται στις ίδιες ημερομηνίες. Θέλουμε επομένως σύγκριση βάσει των τιμών και όχι βάσει των ταυτοτήτων των αντικειμένων.

αντικείμενα τιμές

- Η ισότητα καθορίζεται βάσει της κατάστασης (τιμές των πεδίων) και όχι βάσει ταυτότητας.
- Είναι κατά κανόνα αμετάβλητες κλάσεις. Μετά τη δημιουργία των αντικειμένων δεν αλλάζει η κατάστασή τους.
- Οι πράξεις επί των αντικειμένων δεν αλλάζει την κατάστασή τους αλλά επιστρέψουν νέα αντικείμενα.
- Για το ζήτημα της ισότητας πρέπει να επαναορίσουμε τις μεθόδους equals και hashCode.

παράδειγμα: κλάση ZipCode

```
public class ZipCode {
    private String value;

    public ZipCode(String zipcode) {
        value = zipcode;
    }

    public String getValue() {
        return value;
    }
}
```

η μέθοδος eqauls της κλάσης ZipCode

```
public boolean equals(Object other) {
    if (other == null) {
        return false;
    }

    if (this == other) {
        return true;
    }

    if (! (other instanceof ZipCode)) {
        return false;
    }

    ZipCode theZipCode = (ZipCode) other;
    return value == null ? theZipCode.value == null : value.equals(theZipCode.value);
}
```

Η μέθοδος hashCode της κλάσης ZipCode

```
public int hashCode() {
    return value == null ? 0 : value.hashCode();
}
```

συσχετίσεις

Τα δύο σημαντικότερα στοιχεία που θα πρέπει να έχει υπόψη του ο σχεδιαστής για μία συσχέτιση είναι

- η πολλαπλότητα και
- η πλοηγησιμότητα

στα άκρα των συσχετίσεων.

μονόδρομες συσχετίσεις πολλαπλότητας «ένα»


```
public class Book {
private Publisher publisher;

    public void setPublisher(Publisher publisher)  {
            this.publisher = publisher;
    }

    public Publisher getPublisher()  {
            return publisher;
    }
}
```

μονόδρομες συσχετίσεις πολλαπλότητας «ένα»

- Για να υλοποιήσουμε την απλή συσχέτιση όπου κάθε αντικείμενο της κλάσης Book γνωρίζει ένα αντικείμενο της κλάσης Publisher, δημιουργούμε ένα πεδίο publisher με τύπο Publisher εντός της κλάσης Book .
- Την ονομασία του πεδίου την πήραμε από το όνομα του άκρου της συσχέτισης του σχήματος.
- Η μέθοδος setPublisher ενημερώνει το πεδίο και η μέθοδος getPublisher το επιστρέφει

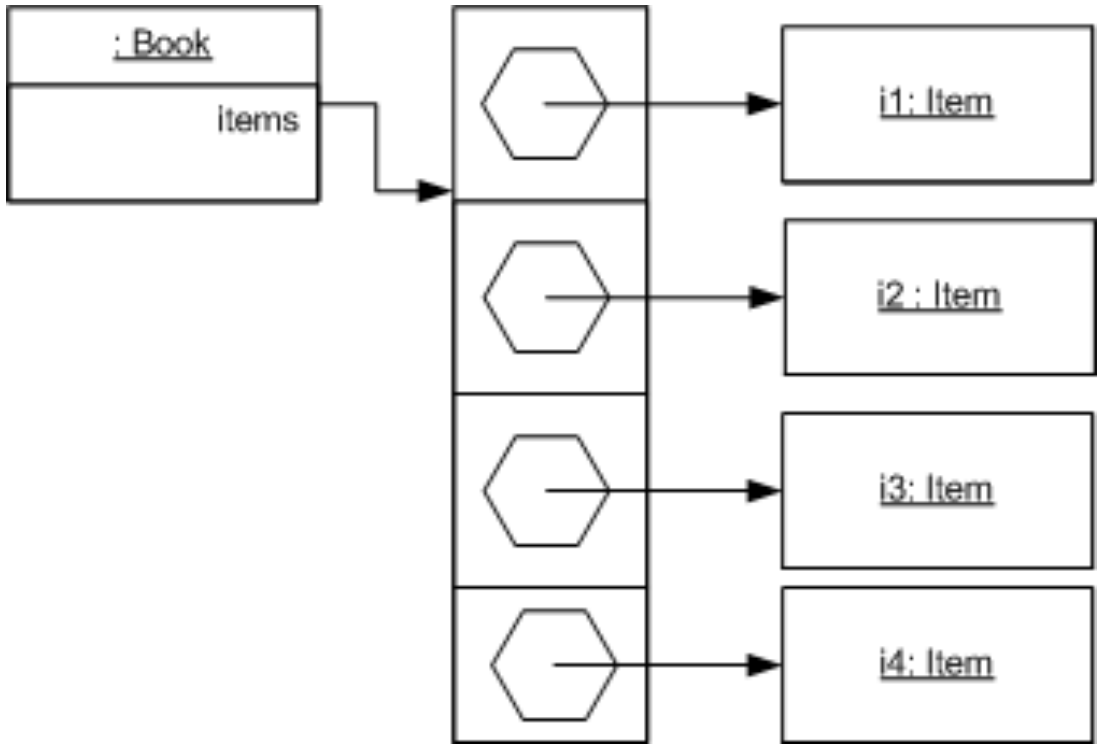
μονόδρομες συσχετίσεις πολλαπλότητας «πολλά»


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

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

μονόδρομες συσχετίσεις πολλαπλότητας «πολλά»

- Θα πρέπει να χρησιμοποιήσουμε μία συλλογή στην κλάση Book που θα περιέχει αναφορές προς τα συσχετιζόμενα αντικείμενα της κλάσης Item.
- Μία συλλογή που μπορεί να χρησιμοποιηθεί είναι τα σύνολα της Java.
- Για τα σύνολα η Java παρέχει τη διεπαφή Set και ορισμένες κλάσεις που την υλοποιούν, όπως οι κλάσεις HashSet και TreeSet.

συλλογές



Ένα αντικείμενο της κλάσης Book έχει την αναφορά items προς μία συλλογή. Αν και πολλές φορές λέμε ότι μία συλλογή περιέχει αντικείμενα, στην πραγματικότητα περιέχει αναφορές προς αντικείμενα.

συλλογές

- Εφόσον η συλλογή items αρχικοποιείται σε συλλογή HashSet, θα ήταν λογικό να επιστρέφουμε HashSet και από την getItems(). Προτιμούμε όμως να επιστρέφουμε τη συλλογή ως τη διεπαφή Set και όχι την υλοποίηση που είναι η κλάση HashSet.
- Οι πελάτες της κλάσης Book δε γνωρίζουν την υλοποίηση της συλλογής των αντικειμένων, γνωρίζουν μόνο ότι η συλλογή είναι σύνολο.
- Η καλή πρακτική για το χειρισμό των συλλογών είναι να δημοσιεύονται στους πελάτες οι διεπαφές (π.χ. Set) και όχι οι κλάσεις που τις υλοποιούν (π.χ. HashSet). Αυτό γιατί οι υλοποιήσεις των συλλογών μπορεί να αλλάζουν, αλλά δε θέλουμε οι αλλαγές αυτές να επηρεάζουν τους πελάτες.

συλλογές

- Η υλοποίηση αυτή διευκολύνει ακόμα περισσότερο την αλλαγή της υλοποίησης της συλλογής items. Η αλλαγή στην αρχικοποίηση της συλλογής σε items = new TreeSet() δεν επηρεάζει τους πελάτες της κλάσης Book
- Έτσι, όταν θέλουμε να χρησιμοποιήσουμε τη συλλογή items, θα έχουμε κάποιο κώδικα όπως:
  Book oneBook = new Book();
  Item oneItem = new Item();
  oneBook.getItems().add(oneItem);
  oneBook.getItems().remove(oneItem);
- Με το τμήμα του κώδικα παραπάνω, δημιουργούνται δύο αντικείμενα oneBook και oneItem. Εισάγουμε το αντικείμενο oneItem στη συλλογή items και αμέσως μετά το απομακρύνουμε από τη συλλογή.

συλλογές

- Δεν υπάρχει κάποια μέθοδος setItems στην κλάση Book, γιατί δε θέλουμε οι πελάτες να τροποποιούν απευθείας τη συλλογή, αλλά μόνο μέσω των μεθόδων εισαγωγής και διαγραφής αντικειμένων. Εάν υπήρχε, θα μπορούσε να υπάρξει ο κώδικας:

  oneBook.setItems(null);

- κώδικας που διαγράφει όλη τη συλλογή, κάτι που προφανώς δεν είναι επιθυμητό.

συλλογές

Βοηθητικές μέθοδοι για την πρόσθεση και απομάκρυνση :

```
 public class Book {
//…
        public void addItem(Item item) {
                if (item != null )  {  items.add(item);  }
         }

        public void removeItem(Item item) {
                if (item != null) {             items.remove(item);     }
        }
}
```

συλλογές

- Οι βοηθητικές μέθοδοι μας βοηθούν, αλλά ο πελάτης εξακολουθεί να έχει πρόσβαση στη συλλογή μέσω του παρακάτω κώδικα:

  Set<Item> myItems = oneBook.getItems()
  myItems.add(aSecondItem)

- Μπορεί επομένως ο πελάτης να προσθέσει απευθείας αντικείμενα στη συλλογή, χωρίς να χρησιμοποιήσει τις βοηθητικές μεθόδους. Θα επιθυμούσαμε επομένως να ενθυλακώσουμε πλήρως τη συλλογή και να εξαναγκάσουμε τους πελάτες της κλάσης Book να χρησιμοποιήσουν τις βοηθητικές μεθόδους addItem και removeItem.

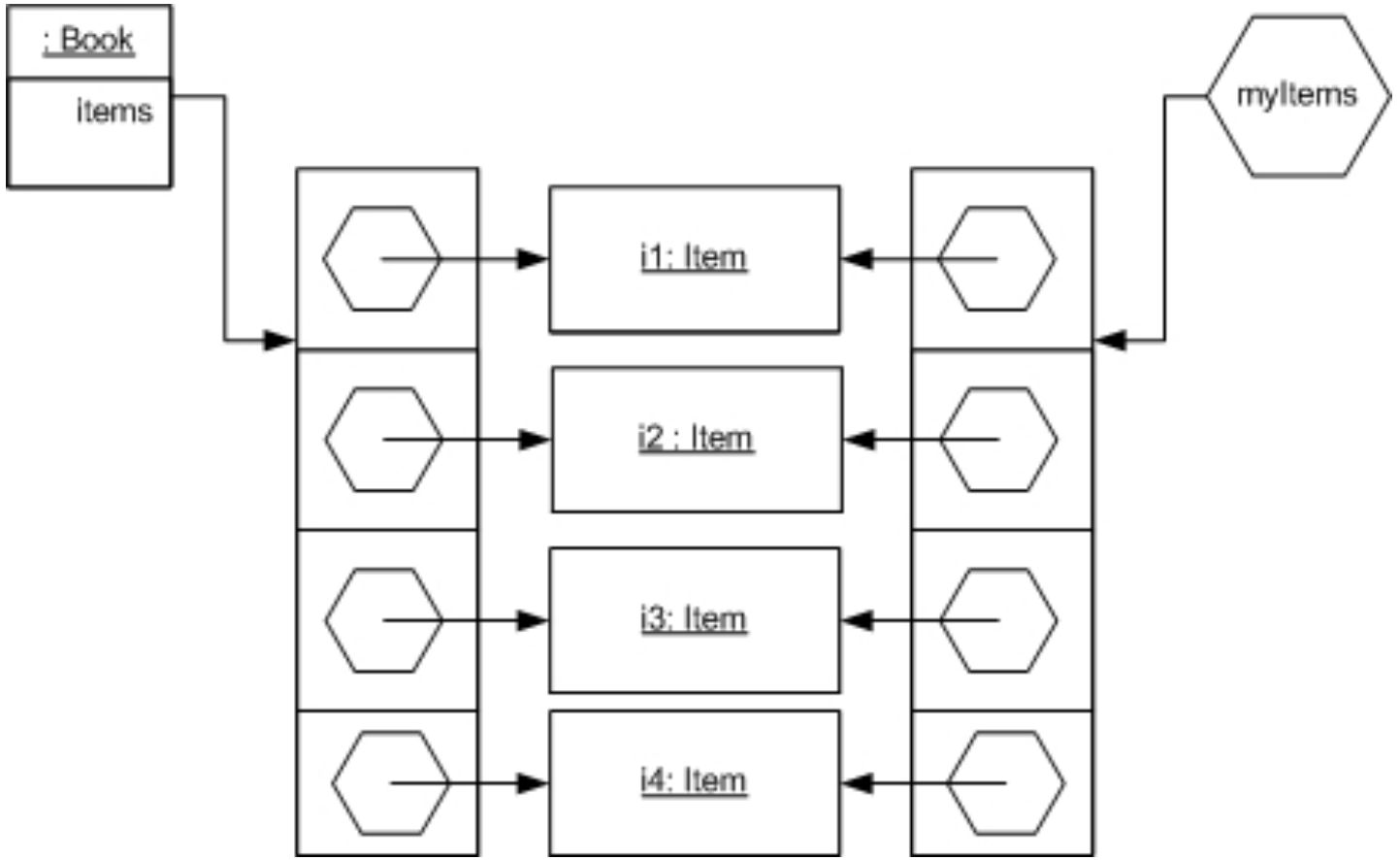
συλλογές

- Η ενθυλάκωση αυτή γίνεται με την επιστροφή αντιγράφου της συλλογής και όχι την ίδια τη συλλογή. Αυτό επιτυγχάνεται με την τροποποίηση της μεθόδου getItems παρακάτω:

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

- Η getItems επιστρέφει πλέον αντίγραφα της συλλογής items και όχι την ίδια τη συλλογή. Τα αντίγραφα των συλλογών δίνουν πρόσβαση στα αντικείμενα που περιέχουν και αφήνουν φυσικά την πλήρη χρήση τους, όπως για παράδειγμα την τροποποίηση των πεδίων τους. Αυτό που απαγορεύουν είναι η απευθείας πρόσβαση στη συλλογή.

αντίγραφα συλλογών



αντίγραφα συλλογών

- Το σχήμα παρουσιάζει την εικόνα ενός αντιγράφου της συλλογής. Εάν υποθέσουμε ότι έχουμε τον κώδικα:

  Set<Item> myItems = oneBook.getItems();

  τότε η συλλογή myItems είναι το αντίγραφο της συλλογής του αντικειμένου oneBook.
- Μέσω της συλλογής myItems έχουμε πρόσβαση σε όλα τα αντικείμενα και μπορούμε να αλλάξουμε την κατάστασή τους. Η πρόσθεση και η απομάκρυνση αντικειμένων γίνεται πλέον στο αντίγραφο της συλλογής και όχι στην αρχική. Για να γίνει πρόσθεση ή απομάκρυνση αντικειμένων στην αρχική συλλογή θα πρέπει να χρησιμοποιηθούν οι βοηθητικές μέθοδοι addItem και removeItem.

αντίγραφα συλλογών

- Έστω για παράδειγμα ο κώδικας:

  ```
  oneBook.addItem(oneItem);
  oneBook.getItems().add(secondItem);
  ```

- Η πρώτη γραμμή του κώδικα θα εισαγάγει το αντικείμενο oneItem στην (ενθυλακωμένη) συλλογή items. Η δεύτερη γραμμή όμως θα εισαγάγει το αντικείμενο secondItem σε αντίγραφο της συλλογής items. Η συλλογή items δε θα αλλάξει.

συλλογές για πολλαπλότητα πολλά

Η τελική εικόνα της υλοποίησης γίνεται:

```
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 ) {  items.add(item);   }
}

public void removeItem(Item item) {
        if (item != null) {  items.remove(item);  }
}
}
```