Δημιουργία Κλάσεων/Αντικειμένων/Συσχετίσεων/Συνδέσεων

Δημιουργία κλάσεων και αντικειμένων

Θεωρούμε ένα υποθετικό σύστημα που αναφέρεται σε Φοιτητές και Μαθήματα. Για τη μοντελοποίηση των Φοιτητών δημιουργούμε την κλάση Student. Η κλάση διαθέτει ως ιδιότητες το όνομα και τον αριθμό μητρώου του φοιτητή, ενώ από πλευράς μεθόδων πέραν των κατασκευαστών και των μεθόδων πρόσβασης διαθέτει και μια μέθοδο εκτύπωσης των τιμών των ιδιοτήτων.

```
public class Student {

	private String name;
	private String id;

	public Student(String text1, String text2) {
		name = text1;
		id = text2;
	}

	public Student(String text1) {
		name = text1;
		id = "9999";
	}

	public String getName() {
		return name;
	}

	public String getID() {
		return id;
	}

	public void printInfo() {
		System.out.println("Name: " + name);
		System.out.println("AM: " + id);
	}

}
```

Η κατασκευή αντικειμένων της κλάσης Student μπορεί να πραγματοποιηθεί είτε στη main είτε σε οποιαδήποτε μέθοδο κλάσης:

```
public class Main {

	public static void main(String[] args) {
		Student S1 = new Student("John", "1209");
		Student S2 = new Student("Nick");

		S1.printInfo();
```

```
                S2.printInfo();
        }
}
```

Το τμήμα της εντολής Student S1 = new Student("John", "1209"); μετά τον τελεστή "=", δηλαδή:

new Student("John", "1209");

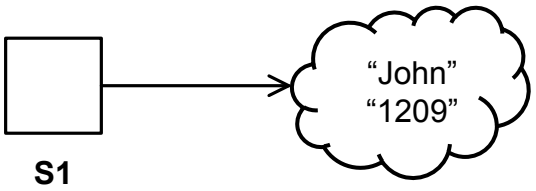
δημιουργεί ένα αντικείμενο της κλάσης Student καλώντας τον αντίστοιχο κατασκευαστή και μεταβιβάζοντας ως ορίσματα το όνομα και τον ΑΜ του φοιτητή.

Το τμήμα της εντολής πριν από τον τελεστή "=", δηλαδή

Student S1

δημιουργεί μια **αναφορά** τύπου Student, δηλαδή μια μεταβλητή της οποίας το περιεχόμενο μπορεί να λάβει ως τιμή τις διευθύνσεις αντικειμένων τύπου Student (ουσιαστικά πρόκειται για έναν "δείκτη").

Μετά την εκτέλεση της εντολής, ανατίθεται στην αναφορά S1 η διεύθυνση του αντικειμένου που δημιουργήθηκε και κατά συνέπεια η αναφορά "δείχνει" πλέον στο δημιουργηθέν αντικείμενο. Αυτό μπορεί να παρασταθεί διαγραμματικά ως εξής:



Επειδή όλες οι ενέργειες επί του αντικειμένου που δημιουργήθηκε πραγματοποιούνται μέσω της αναφοράς S1, καταχρηστικά αναφερόμαστα συχνά στο ίδιο το αντικείμενο ως 'S1'. Ωστόσο θα πρέπει να γίνει κατανοητή η διαφορά μεταξύ μιας αναφοράς ενός αντικειμένου (η οποία προτού αρχικοποιηθεί μπορεί να μην "δείχνει" πουθενά) και του ίδιου του αντικειμένου.

Η αποστολή ενός μηνύματος στο αντικείμενο ισοδυναμεί με την κλήση μιας μεθόδου του αντικειμένου. Για παράδειγμα, η αποστολή του μηνύματος printInfo() στο αντικείμενο προς το οποίο δείχνει η αναφορά S1 με σκοπό να εκτυπωθούν στην κονσόλα οι τιμές των ιδιοτήτων του πραγματοποιείται με την εντολή:

S1.printInfo();

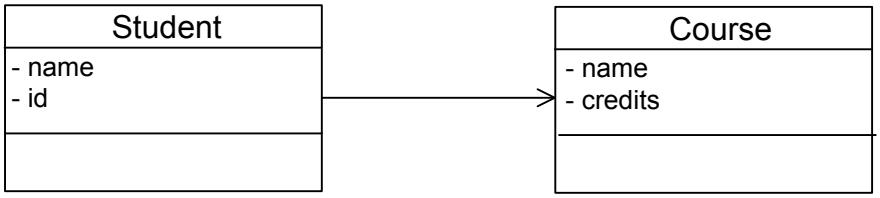
Δημιουργία συσχετίσεων μεταξύ κλάσεων

Για τη μοντελοποίηση ενός μαθήματος δημιουργούμε την κλάση Course:

```
public class Course {
```

```
        private String name;
        private int credits;

        public Course(String text, int number) {
                name = text;
                credits = number;
        }

        public String getName() {
                return name;
        }

        public int getCredits() {
                return credits;
        }

}
```

Για να μοντελοποιήσουμε το γεγονός ότι ένας φοιτητής παρακολουθεί ένα μάθημα (και άρα σχετίζεται με αυτό) δημιουργούμε μια **συσχέτιση** μεταξύ των κλάσεων Student και Course με κατεύθυνση από την πρώτη προς τη δεύτερη (μονόδρομη συσχέτιση). Μια τέτοια συσχέτιση σε ένα διάγραμμα κλάσεων απεικονίζεται ως εξής:



Η συσχέτιση υποδηλώνει ότι ένα αντικείμενο τύπου Student "γνωρίζει" το αντικείμενο Course με το οποίο σχετίζεται και επιτρέπει την αποστολή μηνυμάτων από το αντικείμενο Student προς το αντικείμενο Course (όχι όμως και το ανάποδο, δεδομένου ότι η συσχέτιση έχει μονόδρομη κατεύθυνση). Η συσχέτιση υλοποιείται στην κλάση Student ως μια ιδιότητα που είναι αναφορά τύπου Course:

private Course myCourse;

προφανώς, αν η αναφορά αυτή δεν λάβει συγκεκριμένη τιμή δεν "δείχνει" πουθενά. Μπορούμε να αναθέσουμε τιμή σε μια αναφορά όπως και σε οποιαδήποτε άλλη ιδιότητα. Αν υποθέσουμε ότι η αναφορά λαμβάνει τιμή μέσω κατάλληλης μεθόδου set, η αντίστοιχη μέθοδος θα ήταν ως εξής:

```
        public void setCourse(Course c) {
                myCourse = c;
        }
```

Για να συσχετίσουμε ένα συγκεκριμένο αντικείμενο Student με ένα αντικείμενο Course θα πρέπει, αφού δημιουργηθούν και τα 2 αντικείμενα, να καλέσουμε τη μέθοδο setCourse() επί του αντικειμένου Student μεταβιβάζοντας ως όρισμα το αντικείμενο Course. Για παράδειγμα στη main:

```
public class Main {

	public static void main(String[] args) {
		Student S1 = new Student("John", "1209");
		Student S2 = new Student("Nick");

		Course C1 = new Course("Java", 5);

		S1.setMyCourse(C1);	//δημιουργία σύνδεσης Student S1->Course C1
		S2.setMyCourse(C1);

		S2.printInfo();
		S2.printInfo();
	}
}
```

Μετά την εκτέλεση της εντολής S1.setMyCourse(C1); η αναφορά myCourse του αντικειμένου S1 "δείχνει" πλέον στο αντικείμενο C1. Από το σημείο αυτό και μετά υφίσταται σύνδεση (link) μεταξύ των δύο αντικειμένων.
Επισημαίνεται ότι οι συσχετίσεις υφίστανται στατικά μεταξύ κλάσεων και δημιουργούν την "υποδομή" για τη σύνδεση αντικειμένων. Απλώς και μόνο η ύπαρξη μια συσχέτισης δεν οδηγεί αυτομάτως και σε δημιουργία συνδέσεων μεταξύ των αντικειμένων.
Οι συνδέσεις δημιουργούνται δυναμικά κατά τη διάρκεια της εκτέλεσης και μπορούν να μεταβάλλονται ή να καταστρέφονται. (για παράδειγμα η αναφορά myCourse του αντικειμένου S1 μπορεί να τροποποιηθεί στη συνέχεια ώστε να "δείχνει" σε άλλο μάθημα).

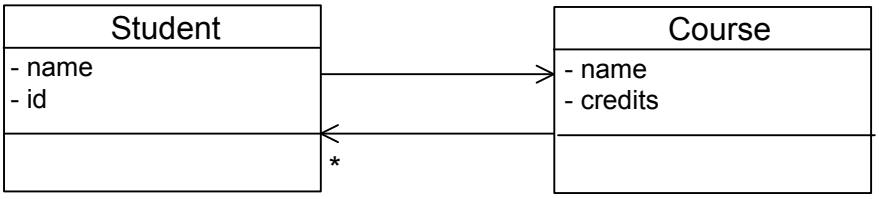
Ένα αντικείμενο τύπου Student μπορεί να αξιοποιήσει τη γνώση του συσχετισμένου μαθήματος ώστε να αποστείλει μηνύματα σε αυτό είτε για την εκτέλεση λειτουργιών είτε για την άντληση δεδομένων. Στον παρακάτω πλήρη κώδικα της κλάσης Student η μέθοδος printInfo έχει τροποποιηθεί ώστε μαζί με τα στοιχεία του φοιτητή να εκτυπώνονται και στα στοιχεία του συσχετιζόμενου μαθήματος.

```
public class Student {

	private String name;
	private String id;
	private Course myCourse;  //αναφορά προς αντικείμενο τύπου Course

	public Student(String text1, String text2) {
		name = text1;
		id = text2;
	}

	public Student(String text1) {
		name = text1;
		id = "9999";
```

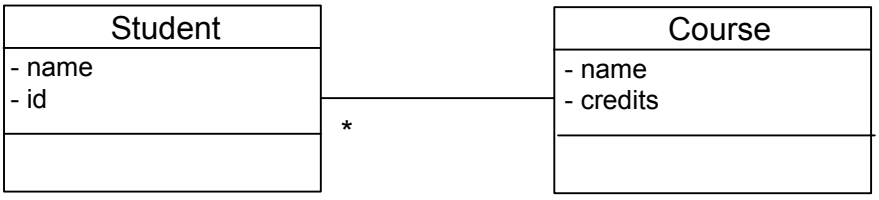
```
        }

        public void setMyCourse(Course c) {
                myCourse = c;
        }

        public String getName() {
                return name;
        }

        public String getID() {
                return id;
        }

        public void printInfo() {
                System.out.println("Name: " + name);
                System.out.println("AM: " + id);
                System.out.println("Course: " + myCourse.getName());
                System.out.println("Credits: " + myCourse.getCredits());
        }

}
```

Δημιουργία αμφίδρομης συσχέτισης με πολλαπλότητα

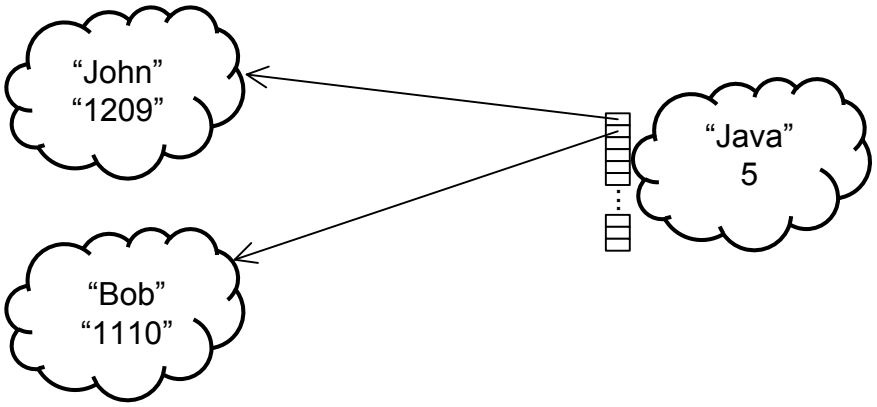
Στην περίπτωση που θέλουμε να μοντελοποιήσουμε και το ότι ένα μάθημα "γνωρίζει" τους φοιτητές που είναι εγγεγραμμένοι σε αυτό θα πρέπει να δημιουργήσουμε και μια συσχέτιση αντίθετης φοράς (από την κλάση Μάθημα προς την κλάση Φοιτητής). Επιπλέον, επειδή στη συνήθη περίπτωση πολλοί φοιτητές εγγράφονται σε ένα μάθημα, η αντίστοιχη συσχέτιση πρέπει να έχει πολλαπλότητα "πολλά", που υποδηλώνει ότι ένα αντικείμενο τύπου Μάθημα μπορεί να συσχετιστεί με πολλά αντικείμενα τύπου Φοιτητής. Σε επίπεδο διαγράμματος κλάσεων μια τέτοια αμφίδρομη συσχέτιση απεικονίζεται ως εξής (το σύμβολο '*' υποδηλώνει "πολλά":



Για λόγους απλούστευσης του διαγράμματος, όταν μια συσχέτιση είναι αμφίδρομη δεν απεικονίζεται ως δύο διαφορετικές ακμές αλλά ως μια γραμμή χωρίς βέλη (υπονοούνται βέλη και προς τις δύο κατευθύνσεις). Επίσης, για την απλοποίηση του παραδείγματος υποθέτουμε ότι σε ένα μάθημα μπορούν να εγγραφούν το πολύ 100 φοιτητές (από 0 έως 100) και κατά συνέπεια το σύμβολο της πολλαπλότητας τροποποιείται ανάλογα:



Πώς υλοποιείται όμως μια τέτοια συσχέτιση από την κλάση Course προς την κλάση Student; Ακριβώς όπως η συσχέτιση από την κλάση Student προς την κλάση Course υλοποιήθηκε ως μια αναφορά τύπου Course μέσα στην κλάση Student, κατά ανάλογο τρόπο απαιτείται η προσθήκη πολλών αναφορών τύπου Student μέσα στην κλάση Course. Προφανώς δεν έχει νόημα η δημιουργία 100 ιδιοτήτων τύπου Student, αλλά η δημιουργία ενός πίνακα 100 θέσεων όπου κάθε θέση είναι μια αναφορά που μπορεί εν δυνάμει να "δείξει" προς ένα αντικείμενο τύπου Student. Αν δηλαδή θεωρήσουμε ένα αντικείμενο Course μπορούμε να θεωρήσουμε ότι φιλοξενεί έναν πίνακα αναφορών προς Student, που επιτρέπει στο μάθημα να συσχετιστεί με έως 100 φοιτητές, όπως φαίνεται στο παρακάτω σχήμα:



Σε επίπεδο κώδικα απαιτείται η προσθήκη μιας ιδιότητας στην κλάση Course υπό μορφή πίνακα 100 θέσεων όπου κάθε θέση αποτελεί αναφορά προς αντικείμενα Student:

```
private Student[] enrolledStudents  = new Student[100];
```

Τονίζεται για ακόμα μια φορά η αναγκαιότητα χρήσης ονομάτων που εκφράζουν όσο το δυνατόν περισσότερο την σημασία κάθε μεταβλητής. Επίσης, αξίζει να σημειωθεί ότι ένας πίνακας αποτελεί αντικείμενο στην Java και κατά συνέπεια πριν από τη χρήση του πίνακα απαιτείται η κατασκευή του αντίστοιχου αντικειμένου με χρήση της δεσμευμένης λέξης new.

Όπως ακριβώς η κλάση Student είχε εφοδιαστεί με μέθοδο (setMyCourse) που επέτρεπε στα αντικείμενα άλλων κλάσεων να θέτουν τιμή στην αναφορά myCourse ενός φοιτητή (και ουσιαστικά να συνδέουν έναν φοιτητή με ένα μάθημα), και η κλάση Course θα πρέπει να εφοδιαστεί με μέθοδο που να επιτρέπει την καταχώρηση τιμών στον πίνακα enrolledStudents (ουσιαστικά να καταχωρούν φοιτητές σε ένα μάθημα). Λόγω της χρήσης του πίνακα απαιτείται η ύπαρξη ενός μετρητή εγγεγραμμένων φοιτητών, έτσι ώστε κάθε νέος φοιτητής να εγγράφεται στην επόμενη ελεύθερη θέση του πίνακα enrolledStudents. Για το λόγο αυτό προστίθεται μια ιδιότητα counter στην κλάση Μάθημα που αντιπροσωπεύει τον αριθμό των ήδη εγγεγραμμένων φοιτητών:

```
private int counter = 0;          //δηλώνουμε ρητά ότι ο μετρητής μηδενίζεται κατά
                                  //την κατασκευή ενός μαθήματος (αν ο μηδενισμός μιας
                                  //int μεταβλητής είναι εγγυημένος στην Java)
```

Η μέθοδος καταχώρησης ενός φοιτητή σε μάθημα, σε απλή μορφή (χωρίς τον έλεγχο της υπέρβασης των διαθέσιμων θέσεων του πίνακα) θα μπορούσε να γραφεί ως εξής:

```
public void addStudent(Student s) {
```

```
        enrolledStudents[counter] = s;          //καταχώρηση της τιμής της αναφοράς s
                                                //στην επόμενη ελεύθερη θέση του πίνακα
        counter++;                              //αύξηση του μετρητή ώστε να δείχνει στην
                                                //επόμενη ελεύθερη θέση

}
```

Έχοντας στη διάθεσή μας μια συσχέτιση μεταξύ Μαθήματος και Φοιτητών, μπορούμε στη μέθοδο main (ή στον κώδικα οποιασδήποτε άλλης κλάσης) να δημιουργήσουμε αντικείμενα τύπου φοιτητής, να δημιουργήσουμε αντικείμενα τύπου Μάθημα και να εγγράψουμε φοιτητές σε κάποιο μάθημα, αποστέλλοντας το μήνυμα addStudent() στο αντίστοιχο αντικείμενο Μάθημα, μεταβιβάζοντας ως όρισμα το αντικείμενο Φοιτητή (την αναφορά προς αυτό):

```
public class Main {

        public static void main(String[] args) {
                Student S1 = new Student("John", "1209");
                Student S2 = new Student("Nick", "1709");

                Course C1 = new Course("Math101");

                S1.setMyCourse(C1);
                S2.setMyCourse(C2);

                C1.addStudent(S1);
                C1.addStudent(S2);
        }
}
```

από τις εντολές

```
                C1.addStudent(S1);
                C1.addStudent(S2);
```

και μετά, το αντικείμενο C1 (μάθημα Math101) είναι συνδεδεμένο με τους φοιτητές S1 και S2, και κατά συνέπεια μπορεί να αποστείλει μηνύματα σε αυτά.

Για παράδειγμα, μπορούμε να εφοδιάσουμε την κλάση Μάθημα με μια μέθοδο εκτύπωσης των στοιχείων των φοιτητών που είναι εγγεγραμμένοι σε ένα μάθημα. Για την υλοποίηση της λειτουργίας, απαιτείται η αλληλεπίδραση του μαθήματος με τους εγγεγραμμένους φοιτητές (αποστέλλοντας μηνύματα) ώστε να προκληθεί η εκτύπωση του ονόματος και του αριθμού μητρώου κάθε φοιτητή. Δεδομένου ότι οι φοιτητές μπορεί να είναι πολλοί, απαιτείται σάρωση της δομής δεδομένων που περιλαμβάνει τις αναφορές προς τους εγγεγραμμένους φοιτητές και την αποστολή κατάλληλου μηνύματος εκτύπωσης των στοιχείων. Η αντίστοιχη μέθοδος στην κλάση Course είναι:

```
        public void printStudentsInfo() {
                for(int i=0; i<counter; i++) {
                        enrolledStudents[i].printInfo();
                }
        }
```

Δημιουργία αμφίδρομης συσχέτισης με πολλαπλότητα - χρήση δομής ArrayList

Οι πίνακες ως δομές δεδομένων έχουν δύο μειονεκτήματα: είναι στατικοί ως προς το μέγεθος (δηλαδή το μέγεθός τους δεν μπορεί να μεταβληθεί κατά τη διάρκεια της εκτέλεσης του προγράμματος) και είναι στατικοί ως προς τον τύπο δεδομένων (δηλαδή αν δηλώσουμε ότι ένας πίνακας περιλαμβάνει στοιχεία του τύπου Α, δεν μπορούμε στην πορεία να εισάγουμε στοιχεία άλλου τύπου).

Η Java διαθέτει μια πλούσια συλλογή ισχυρών, καλά σχεδιασμένων και αποδοτικών δομών δεδομένων (Collection API) που ενδείκνυνται για όλα σχεδόν τα προβλήματα που μπορεί να αντιμετωπίσει ένας προγραμματιστής. Η πλέον συνηθισμένη και εύκολη δομή δεδομένων είναι η ArrayList που συμπεριφέρεται ως δυναμικός πίνακας (άρα είναι εύκολη στη χρήση) και επιπλέον δεν ισχύουν οι 2 περιορισμοί των πινάκων. Το μέγεθος της δομής μεταβάλλεται αυτόματα (μεγαλώνει) αν παραστεί ανάγκη κατά την εκτέλεση του προγράμματος.

Για να χρησιμοποιήσουμε ένα ArrayList πρέπει να δημιουργήσουμε ένα στιγμιότυπο (αντικείμενο) της αντίστοιχης κλάσης:

```
ArrayList enrolledStudents = new ArrayList();
```

Σε μια δομή τύπου ArrayList μπορούν να καταχωρηθούν αντικείμενα οποιουδήποτε τύπου (δεδομένου ότι οποιαδήποτε κλάση στην Java είναι υποκλάση της Object). Το πρόβλημα με αυτή την "ελευθερία" είναι ότι σε περίπτωση που "εξαχθεί" ένα αντικείμενο από μια θέση της δομής απαιτείται ρητή μετατροπή τύπου ώστε να μπορέσουμε να "χρησιμοποιήσουμε" το αντικείμενο. Π.χ. αν θέλουμε να εξάγουμε το 6ο στοιχείο μιας δομής στην οποία έχουν καταχωρηθεί αντικείμενα τύπου Student θα πρέπει να γίνει μετατροπή τύπου ως εξής:

```
Student sixthStudent = (Stundet) enrolledStudents.get(5);
```

Κάτι τέτοιο δεν είναι πάντοτε βολικό, ειδικά για τις περιπτώσεις όπου γνωρίζουμε ότι σε μια δομή ArrayList θα καταχωρηθούν αντικείμενα ενός μόνο τύπου. Για το λόγο αυτό μπορούμε να αξιοποιήσουμε τη δυνατότητα που παρέχεται από την έκδοση της Java 1.5 και μετά των generics (γενικευμένων τύπων) που επιτρέπουν να δηλώσουμε μια δομή δεδομένων υποδεικνύοντας τον τύπο δεδομένων των στοιχείων που θα περιλαμβάνουν. Στην περίπτωση αυτή δεν απαιτείται μετατροπή τύπου κατά την εξαγωγή στοιχείων και επιπλέον δεν επιτρέπεται η εισαγωγή στοιχείων άλλου τύπου (type safety).
Η δήλωση είναι ως εξής:

```
ArrayList<Student> enrolledStudents = new ArrayList<Student>();
```

Για την καταχώρηση ενός στοιχείου σε μια δομή ArrayList χρησιμοποιείται η μέθοδος add() και για την εξαγωγή ενός στοιχείου που βρίσκεται στη θέση με αριθμοδείκτη i η μέθοδος get(i). Για την εύρεση του πλήθους στοιχείων μιας δομής ArrayList διατίθεται η μέθοδος size().
Επομένως, ο πλήρης κώδικας της κλάσης Course με τη χρήση ArrayList είναι:

```
import java.util.*;

public class Course {

	private String name;
	private ArrayList<Student> enrolledStudents;
	private int counter;
```

```
	public Course(String text) {
		name = text;
		enrolledStudents = new ArrayList<Student>();
		counter = 0;
	}

	public void addStudent(Student s) {
		enrolledStudents.add(s);
	}

	public void printStudentsInfo() {
		System.out.println("Course: " + name);
		System.out.println("--------");
		for(int i=0; i<enrolledStudents.size(); i++) {
			Student s = enrolledStudents.get(i);
			s.printInfo();
		}
	}

	public String getName() {
		return name;
	}

}
```

Χρήση του δείκτη this

Στο ανωτέρω πρόγραμμα κάθε φοιτητής διατηρεί μια αναφορά προς το μάθημα που παρακολουθεί και κάθε Μάθημα προς τους φοιτητές που είναι εγγεγραμμένοι σε αυτό. Κατά συνέπεια απαιτείται η καταχώρηση της αναφοράς ενός μαθήματος σε έναν φοιτητή και επίσης η καταχώρηση των φοιτητών που παρακολουθούν ένα Μάθημα.
Προφανώς κάτι τέτοιο θα ήταν άνευ νοήματος σε ένα πραγματικό σύστημα και το ζητούμενο θα ήταν για παράδειγμα, κατά την εγγραφή ενός φοιτητή σε ένα μάθημα, αυτομάτως να ενημερώνεται και ο φοιτητής για το μάθημα που παρακολουθεί. Αυτό επιτυγχάνεται σε αντικειμενοστρεφείς γλώσσες όπως η Java μόνο μέσω της αναφοράς **this**.

Κάθε αντικείμενο διατηρεί μια αναφορά που "δείχνει" προς το ίδιο το αντικείμενο. Η αναφορά αυτή λαμβάνεται από τη δεσμευμένη λέξη this. Με άλλα λόγια, όταν εκτελείται κώδικας επί ενός αντικειμένου (μέσα σε κάποια μέθοδο) η αναφορά this δείχνει πάντοτε προς το ίδιο το αντικείμενο, δηλαδή το αντικείμενο στο οποίο εκτελείται ο κώδικας. Επομένως, κάθε αντικείμενο έχει γνώση της αναφοράς του "εαυτού του" την οποία μπορεί να μεταβιβάσει σε άλλα αντικείμενα, αν θέλει να τους γνωστοποιήσει τη διεύθυνσή του.

Στα πλαίσια του ανωτέρου παραδείγματος, μέσα στον κώδικα της μεθόδου addStudent(Student s) στην κλάση Course, μετά την εγγραφή του φοιτητή στο μάθημα:

```
enrolledStudents.add(s);
```

υπάρχει η δυνατότητα αποστολής της αναφοράς this (που σύμφωνα με όσα αναφέρθηκαν θα "δείχνει" στο αντικείμενο Μάθημα όπου εκείνη τη στιγμή εκτελείται η εγγραφή φοιτητών) στον αντίστοιχο φοιτητή (που είναι γνωστός μέσω της αναφοράς s που λαμβάνεται ως παράμετρος). Κατά συνέπεια αποστέλλοντας το μήνυμα setMyCourse() προς τον φοιτητή s, με παράμετρο την αυτοαναφορά this, γνωστοποιούμε στον φοιτητή ποιο είναι το μάθημα και ο φοιτητή αυτομάτως καταχωρεί το μάθημα στην αναφορά myCourse που διαθέτει.

Με αυτόν τον τρόπο οποτεδήποτε καλείται η μέθοδος addStudent() επί ενός Μαθήματος, επιτυγχάνονται και οι 2 στόχοι, δηλαδή αφενός η εγγραφή του φοιτητής στο Μάθημα και αφετέρου η σύνδεση του Φοιτητή με το αντίστοιχο μάθημα. Ο κώδικας της μεθόδου addStudent() θα είναι ως εξής:

```
public void addStudent(Student s) {
	enrolledStudents.add(s);
	s.setMyCourse(this);
}
```

Οι σχετικές ενέργειες θα μπορούσαν να αναπαρασταθούν γραφικά ως εξής: