ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ ΑΝΤΙΣΤΡΟΦΗΣ ΜΗΧΑΝΙΚΗΣ ΓΙΑ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗ ΣΥΣΤΗΜΑΤΑ ΛΟΓΙΣΜΙΚΟΥ

ΛΟΥΡΔΑΣ ΒΑΣΙΛΕΙΟΣ

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων Καθηγητής Χατζηγεωργίου Αλέξανδρος

Εξεταστές Ρεφανίδης Ιωάννης

Τμήμα Εφαρμοσμένης Πληροφορικής

Πανεπιστήμιο Μακεδονίας

Θεσσαλονίκη

Ιανουάριος 2007

ABSTRACT

The purpose of this project is the development of an application for the reverse engineering of an object-oriented software system. Any software of big scale, apart from its complexity and size, is characterized by its simultaneous development by a lot of programmers and the existence of multiple generations at the duration of the maintainance (multi-person, multi-version systems). As a result, the software design is being rendered "foggy" and obscure. In consequence, the existence of tools that export information from existing code regarding the structure of a system is judged particularly important.

Parsing of a file (or directory of files) and the imprinting of information in the form of UML class diagrams of object-oriented code will be implemented in the project context. At this stage, the existing bibliography as well as existing tools of reverse engineering will be examined in order to determine the functional requirements and investigate the problems of other approaches.

ΠΕΡΙΛΗΨΗ

Αντικείμενο της εργασίας είναι η ανάπτυξη εφαρμογής για την αντίστροφη μηχανική (reverse engineering) ενός αντικειμενοστραφούς συστήματος λογισμικού. Οποιοδήποτε έργο λογισμικού μεγάλης κλίμακας, εκτός από την πολυπλοκότητα και το μέγεθός του, χαρακτηρίζεται από την ταυτόχρονη ανάπτυξή του από πολλούς προγραμματιστές και την ύπαρξη πολλαπλών γενεών κατά τη διάρκεια της συντήρησης (multi-person, multi-version systems). Το γεγονός αυτό έχει ως αποτέλεσμα η σχεδίαση του λογισμικού να καθίσταται με την πάροδο του χρόνου «θολή» και δυσνόητη. Κατά συνέπεια, η ύπαρξη εργαλείων που εξάγουν από υπάρχοντα κώδικα πληροφορίες που αφορούν τη δομή του συστήματος κρίνεται ιδιαιτέρως σημαντική.

Στα πλαίσια της εργασίας θα υλοποιηθεί ανάγνωση (parsing) ενός αρχείου (ή ενός καταλόγου αρχείων) αντικειμενοστραφούς κώδικα και η αποτύπωση της πληροφορίας που περιλαμβάνει, υπό μορφή διαγραμμάτων κλάσεων της UML (Unified Modeling Language). Στο στάδιο αυτό θα εξεταστεί η υπάρχουσα βιβλιογραφία καθώς και υπάρχοντα εργαλεία αντίστροφης μηχανικής με σκοπό να καθοριστούν οι λειτουργικές απαιτήσεις και να διερευνηθούν τα προβλήματα άλλων προσεγγίσεων.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή

Το παρόν κείμενο εκπονήθηκε στα πλαίσια της διπλωματικής εργασίας για το Πρόγραμμα Μεταπτυχιακών Σπουδών του τμήματος Εφαρμοσμένης Πληροφορικής στην ειδίκευση «Συστήματα Υπολογιστών» του Πανεπιστημίου Μακεδονίας.

Ο στόχος της εργασίας είναι η ανάπτυξη εφαρμογής για την αντίστροφη μηχανική κώδικα bytecode της Java και η εξαγωγή του αντίστοιχου διαγράμματος κλάσεων της UML.

Αντίστροφη μηχανική (reverse engineering) χαρακτηρίζεται ως η διαδικασία ανακάλυψης των τεχνολογικών αρχών που διέπουν ένα σύστημα, διαδικασία που περιλαμβάνει ανάλυση της δομής και της λειτουργίας του. Συχνά η διαδικασία εμπλέκει το διαχωρισμό των τμημάτων του συστήματος και τη λεπτομερή μελέτη τους, για να κατασκευαστεί ένα παρόμοιο σύστημα που κάνει ακριβώς ό,τι και το αρχικό, χωρίς όμως να έχει μεσολαβήσει αντιγραφή των τμημάτων του πρώτου (Wikipedia 2006ς).

Στο έργο των μηχανικών λογισμικού έρχεται να βοηθήσει η δημιουργία και εξέλιξη εργαλείων μοντελοποίησης χρησιμοποιώντας κοινά πρότυπα, όπως είναι η Unified Modeling Language (UML). Υπάρχουν φορές που ο μηχανικός λογισμικού έρχεται αντιμέτωπος με συστήματα λογισμικού που έγραψε τρίτος και δεν υπάρχει (ή υπάρχει αλλά είναι ελλειπής) τεκμηρίωση γύρω από την αρχιτεκτονική αυτών των συστημάτων. Σε τέτοιες περιπτώσεις, η ύπαρξη εργαλείων που εφαρμόζουν αντίστροφη μηχανική και εξάγουν πληροφορίες και διαγράμματα που θα βοηθήσουν τον μηχανικό λογισμικού στην κατανόηση αυτών των συστημάτων, θεωρείται εξαιρετικά χρήσιμη και επιβεβλημένη.

Η εφαρμογή της παρούσας εργασίας έρχεται να καλύψει ένα από τα κενά αυτά. Από ένα σύστημα κλάσεων bytecode της Java, η εφαρμογή μπορεί να εξάγει το διάγραμμα κλάσεων της UML που αντιστοιχεί στο σύστημα αυτό, για την κατανόηση των σχέσεων μεταξύ των

κλάσεων, καθώς και της δομής τους.

Στο δεύτερο κεφάλαιο γίνεται μια ιστορική αναδρομή γύρω από τη γλώσσα μοντελοποίησης UML. Δίνεται ένας ορισμός για το είναι γλώσσα μοντελοποίησης και πως μπορεί η UML να χρησιμοποιηθεί. Στη συνέχεια, γίνεται μια σύντομη αναφορά στην αρχιτεκτονική της γλώσσας (γλωσσικές μονάδες και επίπεδα συμμόρφωσης) και στα είδη διαγραμμάτων της UML. Η κατηγοριοποίηση των διαγραμμάτων γίνεται σύμφωνα με το μοντέλο 4+1 που είχε προτείνει ο Kruchten (1995) και παράλληλα γίνεται μια σύντομη περιγραφή για κάθε διάγραμμα που υποστηρίζει η UML 2.0. Στο τέλος του κεφαλαίου γίνεται μια εκτενής περιγραφή των διαγραμμάτων κλάσεων.

Στο τρίτο κεφάλαιο περιγράφεται η εφαρμογή που αναπτύχθηκε στα πλαίσια της παρούσας εργασίας. Περιγράφονται οι βιβλιοθήκες BCEL και JGraph που χρησιμοποιήθηκαν κατά την ανάπτυξή της, με παραδείγματα κώδικα για το πως ο προγραμματιστής μπορεί να τις χρησιμοποιήσει. Στη συνέχεια, γίνεται περιγραφή των τεχνικών εύρεσης των σχέσεων μεταξύ των κλάσεων, πως δηλαδή η εφαρμογή ανακαλύπτει τα είδη των διαφόρων σχέσεων μεταξύ των κλάσεων ενός συστήματος. Ακολουθεί περιγραφή της αρχιτεκτονικής της εφαρμογής, όπου περιγράφονται συνοπτικά οι κλάσεις κάθε πακέτου της εφαρμογής και, στη συνέχεια, ένας οδηγός χρήσης της εφαρμογής με εικόνες. Επίσης, δίνεται ένα παράδειγμα διαγράμματος κλάσεων ενός συστήματος, όπως το δημιουργεί η εφαρμογή. Τέλος, περιγράφονται δυσκολίες και προβλήματα κατά την ανάπτυξη της εφαρμογής και γίνεται λόγος για πιθανές βελτιώσεις.

ΚΕΦΑΛΑΙΟ 2

Εισαγωγή στη Unified Modeling Language (UML)

2.1 Εισαγωγή

Στο κεφάλαιο αυτό γίνεται μια σύντομη εισαγωγή στη γλώσσα μοντελοποίησης αντικειμενοστραφών συστημάτων UML. Στόχος του κεφαλαίου είναι να μεταδώσει στον αναγνώστη τις βασικές έννοιες γύρω από τη UML και να εμβαθύνει στα *διαγράμματα κλάσεων (class diagrams)*.

Αρχικά, θα γίνει μια ιστορική αναδρομή γύρω από τη UML και τις συνθήκες που οδήγησαν στη δημιουργία της γλώσσας. Στη συνέχεια, θα δοθεί ο ορισμός της UML, θα γίνει μια σύντομη αναφορά στην αρχιτεκτονική της γλώσσας, καθώς και μια σύντομη περιγραφή στα είδη διαγραμμάτων που υποστηρίζει. Τέλος, βάρος θα δοθεί στα διαγράμματα κλάσεων.

2.2 Ιστορική αναδρομή

Στη δεκαετία του '80, υπήρξαν αλλαγές στο τοπίο των *αντικειμενοστραφών γλωσσών προγραμματισμού (object–oriented languages)*. Η Smalltalk είναι πλέον μια γλώσσα που μπορεί να χρησιμοποιήσει ο καθένας και η γέννηση της C++ σηματοδοτεί τη δεκαετία.

Ταυτόχρονα, γεννιέται η ανάγκη για την ύπαρξη τρόπων μοντελοποίησης των αντικειμένων με «γραφικό τρόπο». Η πρώτη βιβλιογραφία που πραγματεύεται αυτό το θέμα εμφανίστηκε κάπου μεταξύ 1988 και 1992. Στο διάστημα εκείνο υπήρξαν διάφορες προτάσεις σχετικά με τη μοντελοποίηση των αντικειμένων και το σχεδιασμό στο λογισμικό. Ο Grady Booch ανέπτυξε τη *μέθοδο Booch* (Wikipedia 2006α) που αποτελούνταν από μια αντικειμενοστραφή γλώσσα μοντελοποίησης και μια μέθοδο που χρησιμοποιούνταν στην αντικειμενοστραφή ανάλυση και σχεδιασμό. Ο Booch ανέπτυξε τη μεθοδολογία κατά τη διάρκεια της εργασίας του στην εταιρία

Rational Software (που υπάρχει και σήμερα και ανήκει στην IBM) (M. Fowler 2004α). Ο James Rumbaugh, σε συνεργασία με τους Blaha, Premerlani, Eddy και Lorensen, ανέπτυξε την *Τεχνική Μοντελοποίησης Αντικειμένων (Object Modeling Technique – OMT)* που ήταν επίσης μια μορφή αντικειμενοστραφούς γλώσσας μοντελοποίησης. Από τις δύο προαναφερόμενες προτάσεις, η OMT θεωρούνταν ότι ήταν καταλληλότερη για αντικειμενοστραφή ανάλυση (object–oriented analysis), ενώ η μέθοδος του Booch θεωρούνταν καταλληλότερη για αντικειμενοστραφή σχεδιασμό (object-oriented design). Ο Rumbaugh, και ενώ δούλευε στην εταιρία General Electric, το 1994 προσλήφθηκε από τη Rational Software. Εκεί δουλεύοντας με τον Booch, ξεκίνησαν έχοντας ως βάση τις μεθόδους τους για να δημιουργήσουν την *Ενοποιημένη Μέθοδο (Unified Method)*. Βοήθεια στην προσπάθειά τους αυτή ήρθε να δώσει ο Ivar Jacobson. Ο Ivar, εργαζόμενος στην εταιρία Objectory AB, ανέπτυξε το 1992 τη γλώσσα μοντελοποίησης αντικειμένων *Object-Oriented Software Engineering (OOSE)*. Εκτός από τη γλώσσα μοντελοποίησης, η OOSE όριζε και μεθοδολογία. Το 1995, η Objectory AB εξαγοράστηκε από τη Rational Software. Οι Booch, Rumbaugh και Jacobson ήταν γνωστοί με το παρατσούκλι *Three Amigos*, λόγω των συχνών διαφορών που είχαν σχετικά με τις προτιμήσεις μεθοδολογίας του καθένα (M. Fowler 2004β).

Οι παραπάνω είχαν κατά βάση κοινές ιδέες με μικρές διαφορές μεταξύ τους. Οι ίδιες έννοιες εμφανίζονταν με διαφορετική *σημειογραφία (notation)*, μπερδεύοντας έτσι τον πελάτη. Έτσι προκύπτει η ανάγκη για προτυποποίηση. Σε αυτό έπαιξε ρόλο το *Object Management Group (OMG)*, μια κοινοπραξία έντεκα εταιριών (μεταξύ των οποίων ήταν οι Hewlett Packard, IBM, Sun Microsystems, Apple Computer, American Airlines) που ιδρύθηκε το 1989 με αρχικό σκοπό τον ορισμό προτύπων για κατανεμημένα αντικειμενοστραφή συστήματα (Wikipedia 2006β). Τα τελευταία χρόνια, το OMG επικεντρώνεται στη μοντελοποίηση προγραμμάτων, συστημάτων και επιχειρησιακών διεργασιών και στον ορισμό προτύπων βασισμένα σε μοντέλα σε περίπου είκοσι κάθετες αγορές. Ένα από τα γνωστότερα πρότυπα του OMG υπήρξε το *CORBA (Common Object Request Broker Architecture)*.

Έτσι, μετέπειτα από πιέσεις των προμηθευτών εργαλείων CASE φοβούμενοι ότι η Rational Software θα δημιουργούσε ένα πρότυπο που θα έλεγχε η ίδια, το OMG εκδίδει τον Ιούνιο του 1996 ένα Request for Proposal για τη δημιουργία ενός προτύπου γλώσσας μοντελοποίησης, που θα επέτρεπε τους προμηθευτές εργαλείων CASE να δημιουργούν εργαλεία που να συνεργάζονται μεταξύ τους. Το ίδιο έτος, συστήνεται μια άλλη κοινοπραξία με όνομα *UML Partners*, κάτω από την ηγεσία των Three Amigos με σκοπό τη δημιουργία του προτύπου *Unified Model-*

ing Language ως απάντηση στην πρόταση του Request for Proposal του OMG. Η έκδοση 1.0 κατατέθηκε στο OMG τον Ιανουάριο του 1997. Παράλληλα, τα μέλη της κοινοπραξίας UML Partners, έκαναν ορισμένες τελικές αλλαγές στη *σημασιολογία (semantics)* της πρότασης και ενσωμάτωσαν προτάσεις από τρίτους με αποτέλεσμα την έκδοση της πρότασης 1.1 που κατατέθηκε στο OMG τον Αύγουστο του ίδιου έτους και αποτέλεσε την πρώτη επίσημη έκδοση του προτύπου της UML.

Η εξέλιξη όμως του προτύπου δεν σταμάτησε εκεί. Ακολούθησαν οι εκδόσεις 1.2, 1.3, 1.4 και 1.5. Η έκδοση 1.4.2 καθιερώθηκε ως το διεθνές πρότυπο *ISO/IEC 19501:2005 Information technology – Open Distributed Processing – Unified Modeling Language (UML) Version 1.4.2* του οργανισμού ISO. Αυτή τη στιγμή, η UML βρίσκεται στην έκδοση 2.0, με την έκδοση 2.1 να πρόκειται να ανακοινωθεί σύντομα (Wikipedia 2006δ).

2.3 Ορισμός

Η UML αποτελεί μια ανοικτή γλώσσα προδιαγραφών για τη μοντελοποίηση αντικειμένων. Είναι γλώσσα γενικού σκοπού και περιλαμβάνει μια προτυποποιημένη γραφική σημειογραφία για να απεικονίσει το αφηρημένο μοντέλο ενός συστήματος, που ονομάζεται *μοντέλο UML*. Βοηθά στην περιγραφή και σχεδιασμό συστημάτων λογισμικού (και όχι μόνο), ιδιαίτερα συστημάτων που είναι κτισμένα γύρω από την έννοια της αντικειμενοστρέφειας. Είναι επεκτάσιμη καθώς παρέχει μηχανισμό *προφίλ* για την προσαρμογή στις εκάστοτε ανάγκες. Αν μια έννοια δεν υπάρχει, μπορεί να δημιουργηθεί με τη χρήση *στερεοτύπων* (M. Fowler 2004ς).

Η UML έχει εφαρμοστεί και συνεχίζει να εφαρμόζεται σε πολυάριθμους τομείς της ζωής, όπως τον τραπεζικό και ασφαλιστικό τομέα, στον τομέα υγείας, στην άμυνα, στην κατανεμημένη υπολογιστική, στον τομέα της αγοράς και της προμήθειας, κτλ. Λόγω του γεγονότος ότι ξεκίνησε από τον χώρο του σχεδιασμού λογισμικού, δεν αποτελεί έκπληξη το γεγονός ότι εκεί γίνεται η μεγαλύτερη χρήση της. Όταν εφαρμόζεται στο λογισμικό, η UML προσπαθεί να γεφυρώσει το χάσμα μεταξύ της αρχικής πρωτότυπης ιδέας για ένα λογισμικό και της υλοποίησής της (D. Pilone and Pitman 2005).

Ο Martin Fowler (2004δ) περιγράφει τους τρεις τρόπους με τους οποίους μπορεί να χρησιμοποιηθεί η UML:

- Ως *σκίτσο*. Ο ενδιαφερόμενος θέλει να εξωτερικεύσει κύρια σημεία των σκέψεών του. Τα σκίτσα αυτά είναι πρόχειρα και θα μπορούσαν για παράδειγμα να σκιαγραφηθούν σε ένα

ασπροπίνακα.

- Ως *προσχέδιο*. Ο ενδιαφερόμενος παρέχει μια λεπτομερή προδιαγραφή ενός συστήματος με τα διαγράμματα που του παρέχει η UML. Για το σκοπό αυτό υπάρχουν εργαλεία που δημιουργούν τα διαγράμματα αυτά, ενώ παρέχουν τη δυνατότητα της εμπρός (forward) και αντίστροφης (reverse) μηχανικής (engineering) ώστε πηγαίος κώδικας και μοντέλο UML να είναι πλήρως συγχρονισμένα.

- Ως *γλώσσα προγραμματισμού*. Σε αυτή την περίπτωση, γίνεται η μετάβαση από το μοντέλο UML στον εκτελέσιμο κώδικα (και όχι απλά τμήματα του κώδικα όπως στην εμπρός μηχανική), που σημαίνει ότι μοντελοποιείται κάθε άποψη του συστήματος. Θεωρητικά, μπορεί κανείς να διατηρεί το μοντέλο επ΄ αόριστον και να δημιουργεί τον κώδικα για την *«ανάπτυξη» (deployment)* σε διαφορετικά συστήματα.

Από τα παραπάνω, γίνεται κατανοητό ότι η UML χρησιμοποιείται ακόμα κι όταν απαιτείται μεγάλος βαθμός λεπτομέρειας σε ένα περίπλοκο σύστημα. Ο βαθμός λεπτομέρειας της UML σε ένα σύστημα λογισμικού εξαρτάται ως ένα βαθμό από τη μέθοδο ανάπτυξης που ακολουθούν οι προγραμματιστές.

2.4 Αρχιτεκτονική γλώσσας

Το πρότυπο της UML ορίζεται σε δύο έγγραφα του Object Management Group, το Unified Modeling Language: Infrastructure version 2.0 (formal/05-07-05) (O.M.G. 2006) και Unified Modeling Language: Superstructure version 2.0 (formal/05-07-04) (O.M.G. 2005). Το δύο έγγραφα είναι συμπληρωματικά μεταξύ τους κα καθορίζουν το πλήρες πρότυπο της γλώσσας. Το πρώτο καθορίζει τις βασικές γλωσσικές δομές που απαιτούνται στη UML 2.0 και το δεύτερο τις δομές στο επίπεδο χρήστη.

Η UML είναι μια γλώσσα με ιδιαίτερα ευρύ φάσμα εφαρμογής σε πολλούς τομείς εφαρμογών. Κατά συνέπεια, δεν είναι κατ΄ ανάγκη απαραίτητες οι πλήρεις δυνατότητες μοντελοποίησης που παρέχει. Έτσι, κρίνεται απαραίτητη η δυνατότητα δόμησης της γλώσσας σε *αρθρώματα (modules)* προκειμένου να χρησιμοποιούνται κάθε φορά τα τμήματα εκείνα που κρίνονται ως απαραίτητα για τη μοντελοποίηση της εφαρμογής. Από την άλλη, μία τέτοια δόμηση έχει το μειονέκτημα ότι μπορεί να υπάρξουν προβλήματα κατά την ανταλλαγή μοντέλων μεταξύ των εργαλείων UML. Για το σκοπό αυτό κρίθηκε κατάλληλη για την επίλυση τέτοιων προβλημά-

των η προτυποποίηση *επιπέδων συμμόρφωσης (compliance levels)*, ώστε να διευκολύνεται η ανταλλαγή μοντέλων από διαφορετικά εργαλεία που συμμορφώνονται στο ίδιο επίπεδο.

2.4.1 Γλωσσικές μονάδες (Language units)

Οι έννοιες μοντελοποίησης της γλώσσας ομαδοποιούνται στις *γλωσσικές μονάδες (language units)*. Μια γλωσσική μονάδα αποτελείται από μια συλλογή από έννοιες μοντελοποίησης με στενή μεταξύ τους *σύζευξη (coupling)* που παρέχουν στο χρήστη τη δυνατότητα να απεικονίζει πτυχές του συστήματος σύμφωνα με συγκεκριμένο πρότυπο παράδειγμα. Αυτό σημαίνει ότι με αυτή τη «διαμέριση» της UML ο χρήστης μπορεί να ασχοληθεί με τα τμήματα εκείνα της γλώσσας που θεωρεί απαραίτητα για τα μοντέλα της εφαρμογής του και όταν στο μέλλον οι ανάγκες του αλλάξουν, λόγω του αρθρωτού χαρακτήρα της γλώσσας, μπορεί να προσθέσει ό,τι χρειάζεται. Από τη σκοπιά αυτή, ένας χρήστης δεν απαιτείται να γνωρίζει όλα τα σκέλη της γλώσσας για να τη χρησιμοποιήσει αποτελεσματικά. Για παράδειγμα, η γλωσσική μονάδα δραστηριοτήτων (activities language unit) παρέχει τα απαραίτητα για την μοντελοποίηση συμπεριφοράς που βασίζεται σε πρότυπο παράδειγμα τύπου διαγράμματος ροής.

2.4.2 Επίπεδα συμμόρφωσης (Compliance levels)

Η διαστρωμάτωση της γλώσσας χρησιμοποιείται ως η βάση για τον ορισμό των επιπέδων συμμόρφωσης της UML. Οι έννοιες της μοντελοποίησης της γλώσσας διαμερίζονται σε οριζόντια στρώματα αυξανομένων δυνατοτήτων που ονομάζονται *επίπεδα συμμόρφωσης*. Ορισμένα στοιχεία της γλώσσας υπάρχουν στα ανώτερα επίπεδα συμμόρφωσης. Το πρότυπο Infrastructure ορίζει δύο επίπεδα, ενώ το Superstructure ορίζει τέσσερα, τα οποία και περιγράφονται παρακάτω:

Επίπεδο 0 (Level 0 – L0) Το επίπεδο αυτό ορίζεται και στο Infrastructure και περιέχει μία μόνο γλωσσική μονάδα για τη μοντελοποίηση δομών βασισμένων σε κλάσεις που συναντώνται σε πολλές αντικειμενοστραφείς γλώσσες προγραμματισμού, παρέχοντας έτσι μία βασικού επιπέδου δυνατότητα μοντελοποίησης.

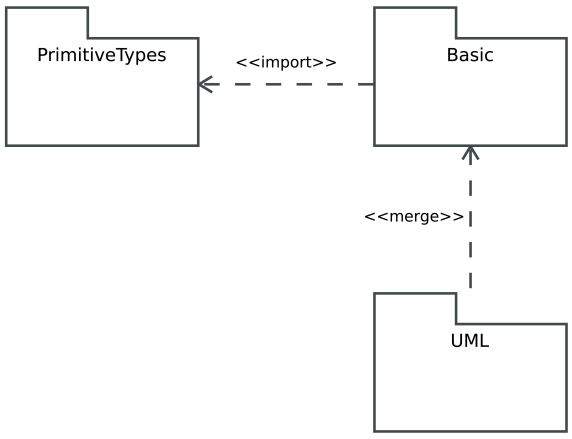
Επίπεδο 1 (Level 1 – L1) Το επίπεδο αυτό παρέχει νέες γλωσσικές μονάδες και επεκτείνει τις δυνατότητες του επιπέδου 0. Συγκεκριμένα, προσθέτει μονάδες για τις περιπτώσεις χρήσης, τις αλληλεπιδράσεις, τις δομές, τις ενέργειες και τις δραστηριότητες.

Επίπεδο 2 (Level 2 – L2) Επεκτείνει τις δυνατότητες του επιπέδου 1 και προσθέτει νέες γλωσσικές μονάδες για την ανάπτυξη (deployment), τις μηχανές κατάστασης και τα προφίλ.

Επίπεδο 3 (Level 3 – L3) Επεκτείνει τις δυνατότητες του επιπέδου 2 και προσθέτει νέες γλωσσικές μονάδες για τη μοντελοποίηση της ροής της πληροφορίας, για τα πρότυπα και τα πακέτα. Το μοντέλο αντιπροσωπεύει το σύνολο της γλώσσας της UML.

Στο έγγραφο Infrastructure μετά το επίπεδο συμμόρφωσης 0 υπάρχει το επίπεδο *Δομών Μεταμοντέλου (Metamodel Constructs)* που προσθέτει μία επιπλέον γλωσσική μονάδα για υψηλού επιπέδου κλάσεις και δομές που χρησιμοποιούνται για την κατασκευή μετα-μοντέλων όπως η ίδια η UML.

Η διαστρωμάτωση με τα επίπεδα συμμόρφωσης επιτυγχάνεται με ένα μηχανισμό που ονομάζεται *ενσωμάτωση πακέτου (package merging)*. Με αυτό τον τρόπο είναι δυνατή η επέκταση των εννοιών μοντελοποίησης με νέα χαρακτηριστικά. Έτσι, όλα τα επίπεδα συμμόρφωσης ορίζονται ως επέκταση στο μοναδικό πακέτο `UML` που υπάρχει σε όλα τα επίπεδα και καθένα «χτίζει» πάνω του. Στο σχήμα **2.4.1** φαίνεται το διάγραμμα πακέτων του επιπέδου 0. Στο διάγραμμα απεικονίζεται το άδειο πακέτο `UML` που ενσωματώνει τα περιεχόμενα του πακέτου `Basic` από το UML Infrastructure το οποίο με τη σειρά του χρησιμοποιεί (η δήλωση `<<import>>`) το πακέτο `PrimitiveTypes`. Το πακέτο `Basic` περιέχει τις βασικές έννοιες όπως κλάση, πακέτο, τύπος δεδομένων, κτλ. που ορίζονται στο UML Infrastructure.



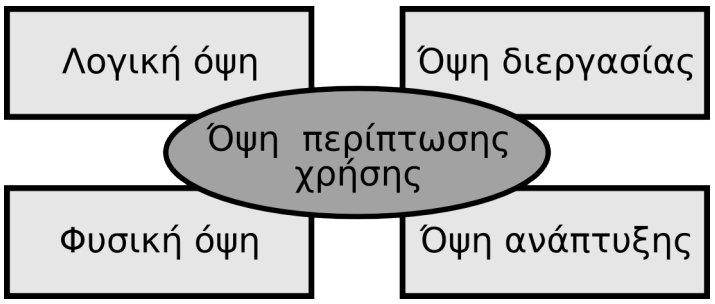
Σχήμα **2.4.1** Το διάγραμμα πακέτων του επιπέδου συμμόρφωσης 0 της UML

Τα διαγράμματα των υπόλοιπων επιπέδων δεν αναφέρονται για λόγους συντομίας. Ο αναγνώστης μπορεί να ανατρέξει στο έγγραφο του UML Infrastructure για περισσότερες λεπτο-

μέρειες.

2.5 Είδη διαγραμμάτων της UML

Ο Kruchten (1995), σε άρθρο του, είχε προτείνει το *μοντέλο 4+1* για να περιγράψει την αρχιτεκτονική των συστημάτων λογισμικού. Το μοντέλο 4+1 βασίζεται σε πολλαπλές, ταυτόχρονες όψεις που καθεμία αιχμαλωτίζει μια συγκεκριμένη πλευρά ενός συστήματος. Βάσει του μοντέλου αυτού, τα διαγράμματα της UML μπορούν να ταξινομηθούν στις όψεις του μοντέλου 4+1 του Kruchten. Η υποδιαίρεση φαίνεται στο σχήμα **2.5.2**.



Σχήμα **2.5.2** Μοντέλο όψεων 4+1 κατά Kruchten

Κάθε όψη στο μοντέλο 4+1 ερμηνεύεται ως εξής (K. Hamilton and Miles 2006):

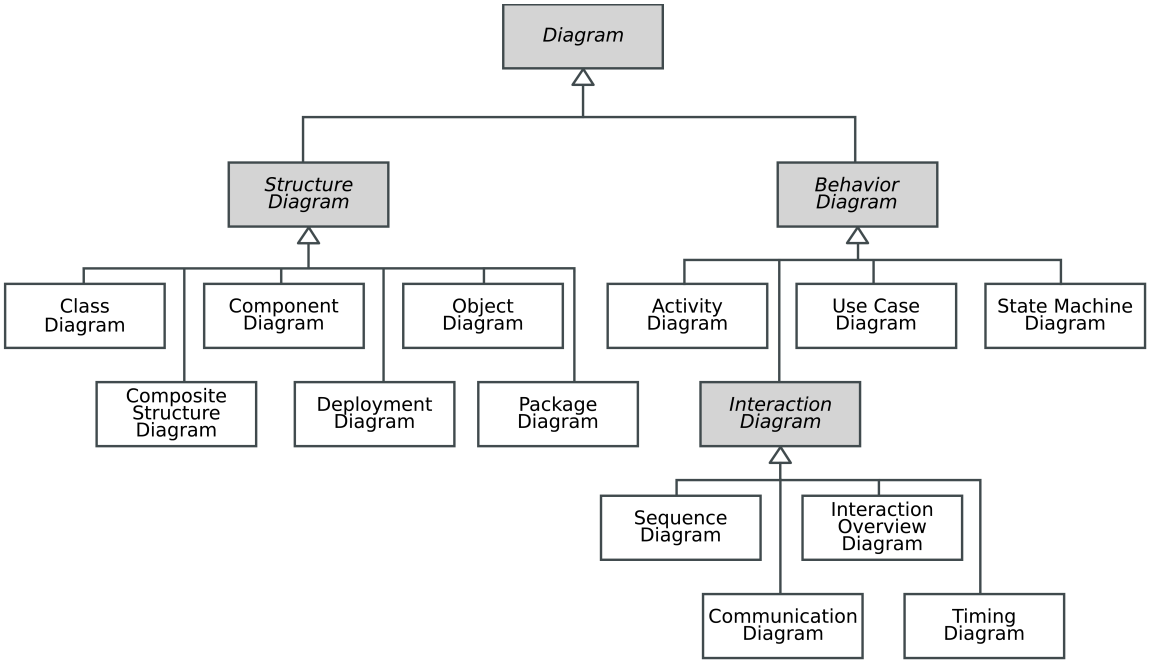
Λογική όψη Περιγράφει τις αφηρημένες έννοιες των τμημάτων ενός συστήματος. Χρησιμοποιείται για να μοντελοποιήσει τα μέρη από τα οποία απαρτίζεται ένα σύστημα και πως αυτά αλληλεπιδρούν μεταξύ τους. Τα είδη των διαγραμμάτων που ανήκουν στη λογική όψη είναι τα διαγράμματα *κλάσεων*, *αντικειμένων*, *κατάστασης μηχανής*, *αλληλεπίδρασης* και *σύνθετων δομών*.

Όψη διεργασίας Περιγράφει τις διαδικασίες μέσα στο σύστημα. Ιδιαίτερα χρήσιμη όταν πρέπει να παρουσιάσει κανείς με οπτικό τρόπο το τι συμβαίνει στο σύστημα. Σε αυτή την όψη ανήκουν τα διαγράμματα *δραστηριοτήτων*.

Όψη ανάπτυξης Περιγράφει πως τα τμήματα ενός συστήματος οργανώνονται σε αρθρώματα και *συστατικά (components)*. Χρήσιμο όταν θέλει κανείς να διαχειρίζεται τα επίπεδα της αρχιτεκτονικής ενός συστήματος. Στην όψη αυτή ανήκουν τα διαγράμματα *πακέτων* και *συστατικών*.

Φυσική όψη Περιγράφει πως η σχεδίαση του συστήματος, όπως αυτή περιγράφεται στις τρεις προηγούμενες όψεις, δημιουργείται ως ένα σύνολο οντοτήτων του πραγματικού κόσμου. Τα διαγράμματα *ανάπτυξης* που ανήκουν στην όψη αυτή δείχνουν πώς τα αφηρημένα τμήματα θα υπάρξουν σε ένα «ανεπτυγμένο» σύστημα.

Όψη περίπτωσης χρήσης Περιγράφει τη λειτουργικότητα του συστήματος όπως αυτή φαίνεται από έξω. Η όψη αυτή είναι απαραίτητη προκειμένου να δείξει στον ενδιαφερόμενο τι υποτίθεται ότι πρέπει να κάνει το σύστημα. Από την όψη αυτή εξαρτώνται όλες οι υπόλοιπες και αυτός είναι ο λόγος που το μοντέλο ονομάζεται 4+1. Στην όψη ανήκουν τα διαγράμματα *περιπτώσεων χρήσης*, *περιγραφών* και *επισκόπησης*.



Σχήμα **2.5.3** Η ιεραρχία των διαγραμμάτων της UML

Το σχήμα **2.5.3** (O.M.G. 2005, σελ. 660) δείχνει την ιεραρχία των διαγραμμάτων της UML. Τα διαγράμματα που είναι με γκρι υπόβαθρο και έχουν το όνομά τους με πλάγια γράμματα δεν είναι πραγματικά, αλλά πρόκειται για κατηγορίες διαγραμμάτων.

2.5.1 Λογική όψη

Στο υποκεφάλαιο αυτό περιγράφονται τα διαγράμματα που ανήκουν στη λογική όψη σύμφωνα με το μοντέλο 4+1 όπως περιγράφεται στο §2.5. Μεταξύ αυτών είναι και τα διαγράμματα κλάσεων,

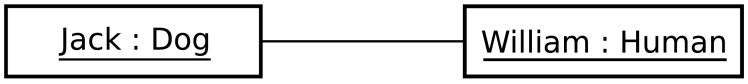
τα οποία όμως αναλύονται εκτενέστερα στο §2.6.

Διαγράμματα αντικειμένων (Object diagrams)

Τα *αντικείμενα (objects)* αποτελούν την καρδιά ενός αντικειμενοστραφούς συστήματος. Σε ένα σύστημα που σχεδιάστηκε και στη συνέχεια χρησιμοποιείται από τους τελικούς χρήστες, τα αντικείμενα είναι τα στιγμιότυπα των κλάσεων που σχεδιάστηκαν προσεκτικά και διαμορφώνουν τα μέρη του συστήματος. Λόγω του γεγονότος ότι τα διαγράμματα αντικειμένων δείχνουν *στιγμιότυπα (instances)*, πολλές φορές ονομάζονται διαγράμματα στιγμιοτύπων.

Στα διαγράμματα τα ονόματα των στοιχείων που υπάρχουν είναι υπογραμμισμένα και κάθε όνομα έχει τη μορφή 'όνομα στιγμιοτύπου : όνομα κλάσης', για παράδειγμα 'Jack : Dog' (βλ. σχήμα **2.5.4**). Όταν παραλείπεται το όνομα του στιγμιοτύπου, για παράδειγμα : Dog, τότε πρόκειται για περίπτωση ανώνυμου αντικειμένου. Αυτό δεν πρέπει να φαίνεται περίεργο, εφόσον υπάρχουν περιπτώσεις που δημιουργούνται ανώνυμα αντικείμενα, όπως συμβαίνει για παράδειγμα στη Java κατά τη δημιουργία ενός ανώνυμου αντικειμένου του τύπου `ActionListener` που αποτελεί σύνηθες φαινόμενο.

Τα διαγράμματα αντικειμένων είναι χρήσιμα στο να δείχνουν πως τα αντικείμενα συνδέονται μεταξύ τους. Τα διαγράμματα κλάσεων ορίζουν με ακρίβεια μια δομή, αλλά σε περιπτώσεις που είναι δύσκολο η δομή αυτή να γίνει κατανοητή, τότε τα διαγράμματα αντικειμένων μπορούν να ξεκαθαρίσουν την κατάσταση.



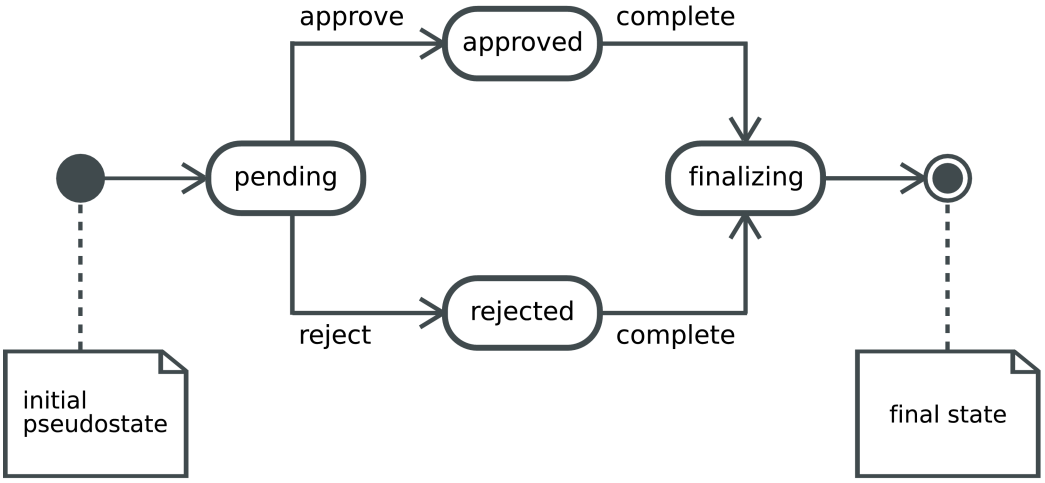
Σχήμα **2.5.4** Παράδειγμα διαγράμματος αντικειμένων

Διαγράμματα κατάστασης μηχανής (State machines diagrams)

Τα *διαγράμματα κατάστασης μηχανής (state machines diagrams)* χρησιμεύουν για να δείχνουν πως ένα αντικείμενο συμπεριφέρεται ανάλογα με την κατάσταση που βρίσκεται. Υπάρχουν περιπτώσεις, που αλλαγές στην κατάσταση των αντικειμένων προκαλούν αλλαγή στην συμπεριφορά τους. Ένα αντικείμενο μπορεί να βρίσκεται σε διάφορες *καταστάσεις (states)*. Οι καταστάσεις συμβολίζονται με ορθογώνια με στρογγυλοποιημένες άκρες. Ένα αντικείμενο μπορεί να μεταβαίνει από μια κατάσταση σε άλλη μέσω της *μετάβασης (transition)*. Η τελευταία συμβολίζεται με ένα βέλος. Μια κατάσταση θεωρείται *ενεργή (active)* όταν γίνεται εισαγωγή σε αυτή μέσω

μιας μετάβασης και *ανενεργή (inactive)* όταν γίνεται εξαγωγή από αυτή μέσω μιας μετάβασης. Το γεγονός ή *σκανδάλη (trigger)* που προκαλεί την αλλαγή της κατάστασης αναγράφεται κατά μήκος του βέλους μετάβασης. Στα διαγράμματα αυτού του είδους, συνήθως υπάρχει αυτό που ονομάζεται *αρχική ψευδοκατάσταση (initial pseudostate)* και *τελική κατάσταση (final state)* που συμβολίζουν αντίστοιχα την αρχή και το τέλος της κατάστασης μηχανής.

Στο σχήμα **2.5.5** φαίνεται το διάγραμμα κατάστασης μηχανής για ένα λογαριασμό που αρχικά είναι στην κατάσταση αναμονής (pending) και ανάλογα με την ενέργεια, μεταβαίνει είτε στην κατάσταση έγκρισης (approved), είτε στην κατάσταση απόρριψης (rejected) για να καταλήξει στο τέλος στην κατάσταση τερματισμού (finalizing).



Σχήμα **2.5.5** Παράδειγμα διαγράμματος κατάστασης μηχανής

Διαγράμματα αλληλεπίδρασης (interaction diagrams)

Τα διαγράμματα αλληλεπίδρασης περιγράφουν πως ομάδες αντικειμένων συνεργάζονται σε κάποια συμπεριφορά. Στην μεγάλη αυτή κατηγορία ανήκουν τα *διαγράμματα ακολουθίας*, *επικοινωνίας*, *χρονισμού* και *επισκόπησης*.

Διαγράμματα ακολουθίας (Sequence diagrams) Από τις διάφορες μορφές διαγραμμάτων αλληλεπίδρασης που ορίζει η UML οι πιο συχνά χρησιμοποιούμενες είναι τα *διαγράμματα ακολουθίας*. Τυπικά, τα διαγράμματα ακολουθίας αιχμαλωτίζουν τη συμπεριφορά ενός σεναρίου και δείχνουν πως τα διάφορα τμήματα του συστήματος αλληλεπιδρούν μεταξύ τους.

Στα διαγράμματα ακολουθίας λαμβάνουν μέρος τα διάφορα τμήματα του συστήματος που ονομάζονται *συμμετέχοντες (participants)*. Οι συμμετέχοντες στο διάγραμμα ακολουθίας το-

ποθετούνται στην ίδια ευθεία στον οριζόντιο άξονα, χωρίς δύο ή περισσότεροι να αλληλοεπικαλύπτονται, ενώ η σειρά τοποθέτησής τους παίζει το δικό της ρόλο. Τα ονόματα των συμμετεχόντων είναι της μορφής

`όνομα´ `[επιλογέας]´ : `όνομα κλάσης´ [ref `αποσύνθεση´]

όπου όνομα είναι το όνομα του στιγμιοτύπου που λαμβάνει μέρος, επιλογέας είναι ένα προαιρετικό τμήμα του ονόματος που δείχνει για ποιο συγκεκριμένο στιγμιότυπο πρόκειται αν ένα στοιχείο έχει πολλές τιμές (για παράδειγμα ένας πίνακας), όνομα κλάσης είναι το όνομα του τύπου και ref αποσύνθεση είναι ένα προαιρετικό τμήμα που δείχνει σε ένα άλλο διάγραμμα ακολουθίας, το οποίο δείχνει λεπτομέρειες για το πως ο συμμετέχων επεξεργάζεται τα μηνύματα.

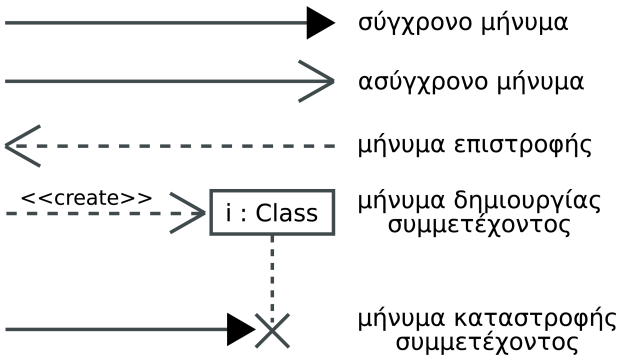
Ο χρόνος σε ένα διάγραμμα ακολουθίας ξεκινά από την κορυφή και αυξάνει καθώς μετακινείται το μάτι προς τα κάτω. Πρέπει να σημειωθεί ότι τα διαγράμματα αυτά έχουν σχέση με την ακολουθία των ενεργειών και όχι με τη διάρκεια.

Τα *γεγονότα (events)* είναι το μικρότερο τμήμα μιας αλληλεπίδρασης και είναι ένα σημείο όπου κάτι συμβαίνει. Σε ένα διάγραμμα ακολουθίας, συμβαίνει αλληλεπίδραση όταν ένας συμμετέχων στέλνει ένα *μήνυμα (message)* σε έναν άλλο. Ένα μήνυμα συνοδεύεται από την περιγραφή ή την υπογραφή του. Η μορφή της υπογραφής είναι

[`ιδιότητα´ = ]`όνομα μηνύματος´ (`παράμετροι´) : `τύπος επιστρεφόμενης τιμής´

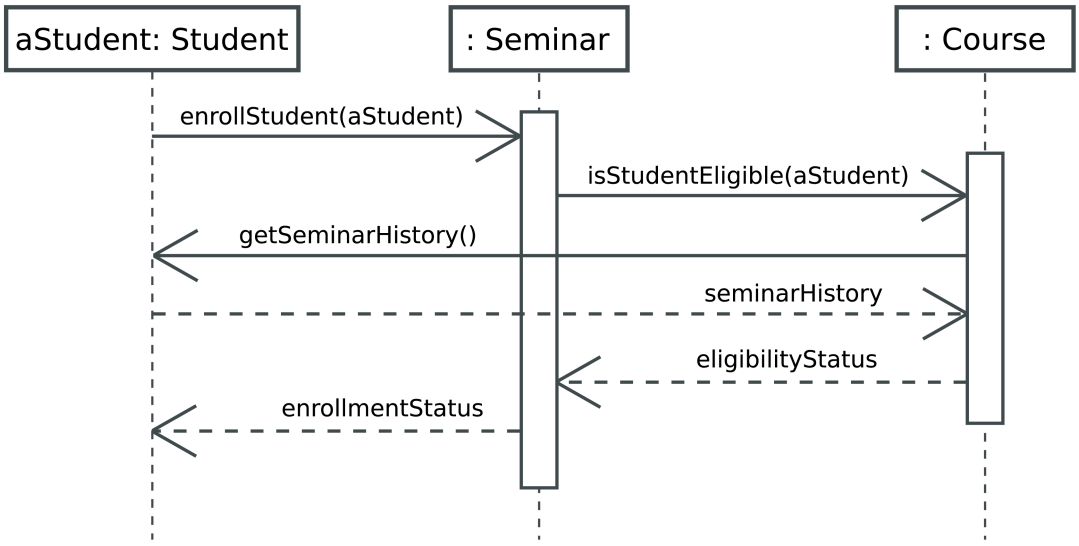
όπου ιδιότητα είναι ένα προαιρετικό όνομα μιας μεταβλητής που θα πάρει την τιμή που επιστρέφει το μήνυμα και αφορά τον καλών, όνομα μηνύματος είναι το όνομα της λειτουργίας που καλείται, παράμετροι είναι οι παράμετροι της λειτουργίας στη μορφή όνομα : τύπος και χωρίζονται με το κόμμα στην περίπτωση πολλών παραμέτρων και τύπος επιστρεφόμενης τιμής είναι ο τύπος της τιμής επιστροφής της λειτουργίας. Όταν ένα μήνυμα περνάει από έναν συμμετέχων σε έναν άλλον, τότε ο τελευταίος ενεργοποιείται προκειμένου να εκτελέσει μια λειτουργία. Στην περίπτωση αυτή, ο συμμετέχων λέγεται ότι είναι *ενεργός (active)* και στο διάγραμμα αυτό φαίνεται με μια κάθετη μπάρα κάτω από το στιγμιότυπό του.

Τα είδη των μηνυμάτων φαίνονται στο σχήμα **2.5.6**. Σύγχρονα ονομάζονται τα μηνύματα για τα οποία ο καλών πρέπει να περιμένει το αποτέλεσμα προτού συνεχίσει. Το αντίθετο είναι τα ασύγχρονα μηνύματα, τα οποία καλεί ο συμμετέχων και συνεχίζει χωρίς να περιμένει. Ένας συμμετέχων σε ένα διάγραμμα ακολουθίας δεν είναι απαραίτητα στη ζωή καθόλη τη διάρκεια, αλλά μπορεί να δημιουργείται και να καταστρέφεται μετά από την κλήση μηνυμάτων από άλλους συμμετέχοντες. Για το σκοπό αυτό υπάρχουν τα μηνύματα δημιουργίας και καταστροφής των συμμετεχόντων.



Σχήμα **2.5.6** Οι τύποι μηνυμάτων που στέλνονται μεταξύ των συμμετεχόντων

Ένα πιο ολοκληρωμένο διάγραμμα ακολουθίας απεικονίζεται στο σχήμα **2.5.7**, όπου εμφανίζεται η διαδικασία εγγραφής ενός μαθητή σε ένα σεμινάριο. Το αντικείμενο μαθητής (student) περνάει μηνύματα στο σεμινάριο (seminar) το οποίο με τη σειρά του περνάει μηνύματα στο μάθημα (course) και κατά τη διαδικασία αυτή επιστρέφονται αντικείμενα (τα βέλη με τις διακεκομμένες γραμμές) στους καλώντες συμμετέχοντες.



Σχήμα **2.5.7**: Ένα απλό διάγραμμα ακολουθίας που περιγράφει τη διαδικασία εγγραφής ενός μαθητή σε ένα σεμινάριο

Διαγράμματα επικοινωνίας (communication diagrams) Στην κατηγορία των διαγραμμάτων αλληλεπίδρασης ανήκουν και τα *διαγράμματα επικοινωνίας (communication diagrams)*, τα οποία στη UML 1.x ήταν γνωστά ως *διαγράμματα συνεργασίας (collaboration dia-*

grams). Αυτά δε διαφέρουν πολύ από τα διαγράμματα ακολουθίας. Αντί σε κάθε συμμετέχων να ζωγραφίζεται μια γραμμή ζωής, οι συμμετέχοντες μπορούν να τοποθετούνται χωρίς περιορισμό στο διάγραμμα ενώ μεταξύ τους υπάρχουν σύνδεσμοι με αριθμούς που δείχνουν τη σειρά των μηνυμάτων που ανταλλάσονται μεταξύ τους.

Διαγράμματα χρονισμού (timing diagrams) Τα *διαγράμματα χρονισμού (timing diagrams)*, όπως φανερώνει και το όνομά τους, έχουν σχέση με το χρόνο. Ενώ τα διαγράμματα ακολουθίας δίνουν βάση στη σειρά των μηνυμάτων μεταξύ των συμμετεχόντων και τα διαγράμματα επικοινωνίας δείχνουν τους συνδέσμους μεταξύ τους, το κενό όσον αφορά τον παράγοντα χρόνο κατά τις διαδικασίες αυτές συμπληρώνουν τα διαγράμματα χρονισμού. Τα διαγράμματα αυτά χρησιμοποιούνται όταν ο παράγοντας χρόνος παίζει σημαντικό ρόλο στη σχεδίαση του συστήματος και τέτοιου είδους διαγράμματα συναντιώνται σε συστήματα *πραγματικού χρόνου (real time)* ή *ενσωματωμένα (embedded).* Στα διαγράμματα χρονισμού, κάθε γεγονός συσχετίζεται με πληροφορίες όπως το πότε συμβαίνει, πόσο χρόνο παίρνει για ένα συμμετέχοντα να πάρει το μήνυμα που σχετίζεται με το γεγονός αυτό και για πόσο χρόνο ο λαμβάνων συμμετέχων θα είναι στη συγκεκριμένη κατάσταση.

Διαγράμματα επισκόπησης αλληλεπίδρασης (interaction overview diagrams) Τα *διαγράμματα επισκόπησης αλληλεπίδρασης (interaction overview diagrams)* είναι ένας συνδυασμός των διαγραμμάτων ακολουθίας και δραστηριοτήτων. Τα διαγράμματα αυτά παρέχουν μια υψηλότερου επιπέδου όψη του πως τα τμήματα του συστήματος έρχονται σε αλληλεπίδραση μεταξύ τους σε σχέση με τα διαγράμματα ακολουθίας, επικοινωνίας και χρονισμού. Κάθε τμήμα του διαγράμματος μπορεί να είναι από μόνο του μια πλήρης αλληλεπίδραση, που ανάλογα με τις εκάστοτε ανάγκες μπορεί να απεικονιστεί ως διάγραμμα ακολουθίας ή χρονισμού. Παρόμοια με ένα διάγραμμα δραστηριοτήτων η αλληλεπίδραση ξεκινά με τον αρχικό κόμβο και τελειώνει με τον τελικό, καθώς επίσης μπορεί να έχει αποφάσεις, παράλληλες ενέργειες ή και *βρόχους (loops).*

Με τα διαγράμματα επισκόπησης αλληλεπίδρασης, κλείνει το κεφάλαιο των διαγραμμάτων αλληλεπίδρασης στη UML 2.0.

Σύνθετες δομές (composite structures)

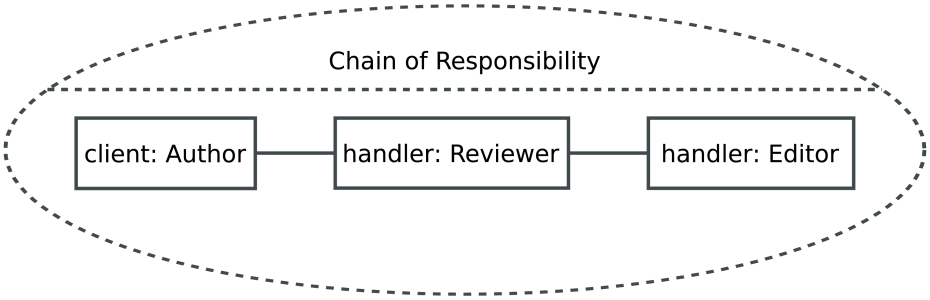
Τα διαγράμματα *σύνθετων δομών (composite structures)* είναι νέο χαρακτηριστικό στην έκδοση 2.0 της UML. Τα διαγράμματα αυτά επιτρέπουν τη διάσπαση ενός σύνθετου αντικειμένου σε τμήματα για την καλύτερη κατανόησή του. Χρησιμοποιούνται εκεί που τα διαγράμματα κλάσεων δεν μπορούν να αποτυπώσουν ορισμένες λεπτομέρειες του συστήματος, σε περιπτώσεις όπως οι εσωτερικές δομές μιας κλάσης και ο τρόπος συνεργασίας μεταξύ τους ή ο τρόπος χρησιμοποίησης μιας κλάσης από το σύστημα και η απεικόνιση *προτύπων σχεδίασης (design patterns)*, δηλαδή αντικειμένων που συνεργάζονται για την επίτευξη ενός στόχου. Τα διαγράμματα αυτά είναι παρόμοια με τα διαγράμματα κλάσεων, με τη διαφορά ότι μοντελοποιούν μια συγκεκριμένη χρήση μιας δομής. Ενώ τα διαγράμματα κλάσεων μοντελοποιούν τη στατική όψη των δομών των κλάσεων, τα διαγράμματα σύνθετων δομών χρησιμοποιούνται για να δείξουν δομές κατά την εκτέλεση (run-time), τα χρησιμοποιούμενα πρότυπα και τις σχέσεις μεταξύ στοιχείων που μπορεί να μη δείχνουν τα στατικά διαγράμματα.

Στα διαγράμματα σύνθετων δομών υπάρχει η έννοια της *θύρας (port)* που είναι το σημείο αλληλεπίδρασης της κλάσης με τον εξωτερικό κόσμο και αντιπροσωπεύει μοναδικούς τρόπους χρησιμοποίησης της κλάσης, που συνήθως είναι από διαφορετικούς πελάτες. Παρόμοια με τα διαγράμματα συστατικών (βλ. §2.5.3 στη σελίδα 27), οι κλάσεις έχουν τις *απαραίτητες (required)* και *παρεχόμενες (provided)* διασυνδέσεις. Οι *συνεργασίες (collaborations)* είναι ένας τρόπος για να δείξει κανείς την ύπαρξη ενός προτύπου σχεδίασης ή γενικότερα τη συνεργασία μεταξύ των κλάσεων προκειμένου να επιτύχουν ένα σκοπό. Στο σχήμα **2.5.8** απεικονίζεται η συνεργασία μεταξύ των τριών κλάσεων που υλοποιούν το πρότυπο σχεδίασης Chain of Responsibility. Στο σχήμα, μεταξύ των κλάσεων υπάρχουν γραμμές που συνδέουν τις κλάσεις μεταξύ τους απεικονίζοντας έτσι τη συσχέτιση μεταξύ τους. Οι γραμμές αυτές ονομάζονται *σύνδεσμοι (connectors)*.

Στο σχήμα **2.5.9** εμφανίζεται ένα στιγμιότυπο κλάσεων με εσωτερική δομή. Οι ιδιότητες και τα μέρη του, τα οποία εμφανίζονται και εκείνα ως ιδιότητες, έχουν την σημειογραφία

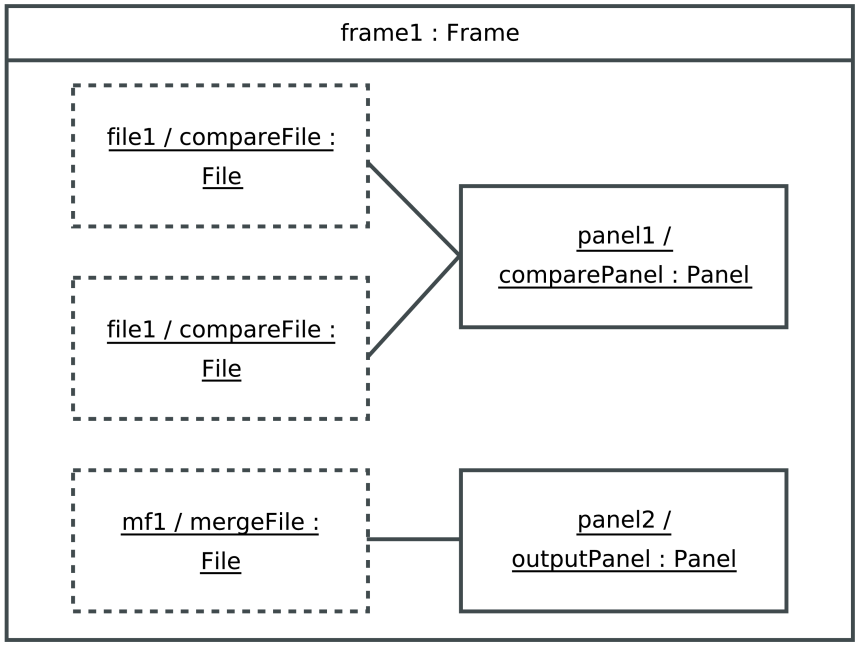
`{όνομα} / ρόλος : τύπος`

όπου `όνομα` το όνομα του στιγμιοτύπου, `ρόλος` το όνομα του ρόλου που χρησιμοποιείται μέσα στο στιγμιότυπο της κλάσης και `τύπος` το όνομα του τύπου του στιγμιοτύπου. Τα ονόματα των τμημάτων υπογραμμίζονται για να δείξουν ότι είναι στιγμιότυπα. Στο ίδιο σχήμα απεικονίζονται οι έννοιες της *ιδιότητας (property)* και του *τμήματος (part)*. Τα πρώτα εμφανίζονται



Σχήμα **2.5.8**: Παράδειγμα διαγράμματος σύνθετης δομής συνεργασίας (collaboration) που απεικονίζει την ύπαρξη του προτύπου σχεδίασης Chain of Responsibility κατά τη διάρκεια έγκρισης του περιεχομένου σε ένα ιστοτόπο

με διακεκομμένες γραμμές και τα δεύτερα με συνεχείς. Οι ιδιότητες είναι αντικείμενα που έχουν *συσχέτιση (association)* με την κλάση που τις περιέχει και μπορεί να μοιράζονται άλλα στιγμιότυπα κλάσεων.

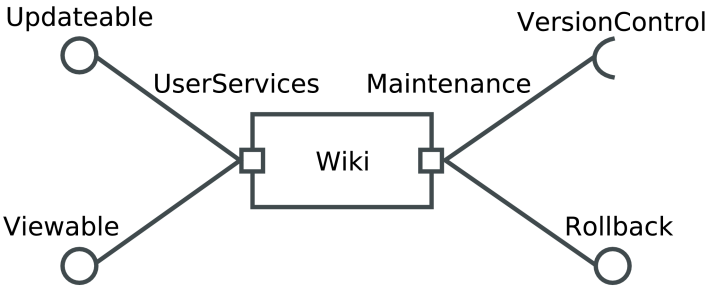


Σχήμα **2.5.9**: Παράδειγμα διαγράμματος σύνθετης δομής που απεικονίζει στιγμιότυπο κλάσης με τα μέρη και τμήματα που περιέχει

Στο σχήμα **2.5.10** απεικονίζεται μία κλάση με όνομα `Wiki` που απεικονίζει ένα τμήμα της λειτουργικότητας ενός wiki[1]. Η κλάση υλοποιεί τις παρεχόμενες διασυνδέσεις `Updateable`

[1]Τα wiki είναι ένας τύπος ιστοχώρου που επιτρέπει στους επισκέπτες να προσθέτουν, διαγράφουν ή

και `Viewable` που συσχετίζονται με τη θύρα `UserServices` επιτρέποντας σε άλλες κλάσεις να ενημερώνουν και να βλέπουν το wiki μέσω αυτών των διασυνδέσεων. Επίσης, στη θύρα `Maintenance` υπάρχει η παρεχόμενη διασύνδεση `Rollback` για να επιτρέπει στους διαχειριστές του wiki να *απορρίπτουν (roll back)* τις αλλαγές σε αυτό, ενώ για την υλοποίηση του χαρακτηριστικού αυτού, χρειάζεται η απαιτούμενη διασύνδεση `VersionControl`. Οι θύρες απεικονίζονται με τα μικρά τετράγωνα στις κάθετες ακμές του ορθογωνίου της κλάσης.



Σχήμα **2.5.10**: Παράδειγμα διαγράμματος σύνθετης δομής που απεικονίζει τμήμα της λειτουργικότητας ενός wiki

2.5.2 Όψη διεργασίας

Στο υποκεφάλαιο αυτό περιγράφονται τα διαγράμματα δραστηριοτήτων που ανήκουν στην όψη διεργασίας.

Διαγράμματα δραστηριοτήτων (Activity diagrams)

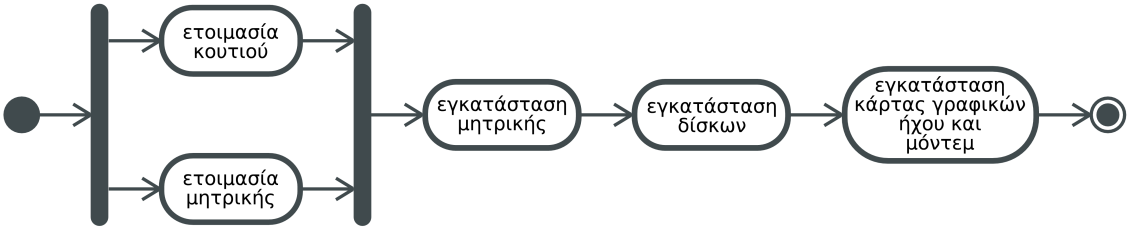
Ενώ τα διαγράμματα περιπτώσεων χρήσης (βλ. §2.5.5) δείχνουν το *τι* θα κάνει το σύστημα, τα *διαγράμματα δραστηριοτήτων (activity diagrams)* δείχνουν το *πώς* το σύστημα θα επιτύχει τους στόχους του. Τα διαγράμματα αυτά δείχνουν υψηλού επιπέδου λειτουργίες που δένουν μεταξύ τους για να αναπαραστήσουν μια διεργασία του συστήματος.

Τα διαγράμματα δραστηριοτήτων μοντελοποιούν ιδιαίτερα εύστοχα *επιχειρησιακές διεργασίες (business processes)*. Η επιχειρησιακή διεργασία είναι ένα σύνολο από συνεργαζόμενες εργασίες για την επίτευξη ενός επιχειρησιακού στόχου, όπως για παράδειγμα η αποστολή μιας παραγγελίας στον πελάτη. Στα είδη αυτών των διαγραμμάτων υπήρξαν σημαντικές αλλαγές από την έκδοση 1.0 της UML στην έκδοση 2.0. Στην πρώτη τα διαγράμματα δραστηριοτήτων

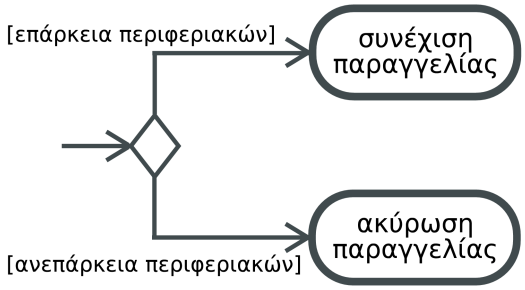
τροποποιούν κάποιο από το περιεχόμενό του, χωρίς να χρειάζεται απαραίτητα είσοδος σε αυτό με όνομα χρήστη και συνθηματικό.

υπήρχαν ως ειδική περίπτωση των διαγραμμάτων κατάστασης, ενώ στη δεύτερη αυτό το «δέσιμο» εξαφανίστηκε, με αποτέλεσμα να χρησιμοποιούνται αρκετά λόγω και του γεγονότος ότι έχουν ομοιότητες με τα *διαγράμματα ροής (flowchart diagrams)*.

Στο σχήμα **2.5.11** απεικονίζεται ένα απλό διάγραμμα δραστηριοτήτων που δείχνει τη διαδικασία συναρμολόγησης ενός προσωπικού υπολογιστή. Στο σχήμα φαίνονται τα στάδια έναρξης και τερματισμού (παρόμοια με τα διαγράμματα κατάστασης μηχανής), οι *ενέργειες (actions)*, που συμβολίζονται με τα στρογγυλεμένα ορθογώνια, ενώ με τις κάθετες μπάρες συμβολίζεται η *διχάλωση (fork)* και η *συνένωση (join)*, διαδικασίες που συμβολίζουν αντίστοιχα την έναρξη παράλληλων ενεργειών και την ένωσή τους και πάλι σε μία ενέργεια.



Σχήμα **2.5.11**: Παράδειγμα διαγράμματος δραστηριοτήτων με διχάλωση (fork) και συνένωση (join)



Σχήμα **2.5.12** Παράδειγμα διαγράμματος δραστηριοτήτων με την έννοια της απόφασης

Στο σχήμα **2.5.12** απεικονίζεται ένα απόσπασμα διαγράμματος δραστηριοτήτων στο οποίο υπάρχει η έννοια της *απόφασης (decision)*, που απεικονίζεται με το ρόμβο. Η απόφαση βασίζεται στις *συνθήκες (conditions)* που αναγράφονται μέσα σε [ ] σε κάθε ακμή.

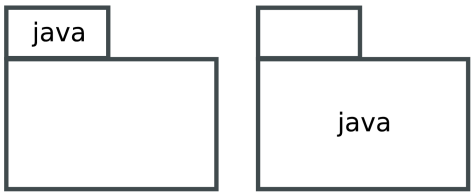
2.5.3 Όψη ανάπτυξης

Στο υποκεφάλαιο αυτό περιγράφονται τα διαγράμματα πακέτων και συστατικών που ανήκουν στην όψη ανάπτυξης.

Διαγράμματα πακέτων (Package diagrams)

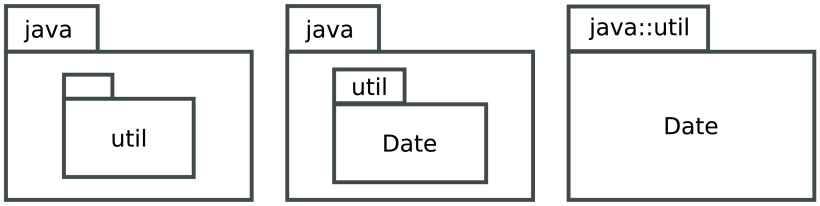
Κατά τη διάρκεια ανάπτυξης μιας μεγάλης εφαρμογής με πολλές κλάσεις, πρέπει να υπάρχει ένας τρόπος οργάνωσης των κλάσεων προκειμένου ο προγραμματιστής να μπορεί να βγάζει νόημα από τον κυκεώνα των κλάσεων. Ένας τρόπος οργάνωσης είναι να δημιουργηθούν λογικές ομάδες κλάσεων. Για παράδειγμα κλάσεις που συσχετίζονται με τη γραφική διασύνδεση της εφαρμογής θα μπορούσαν να συγκεντρωθούν σε μια ομάδα.

Για το σκοπό αυτό, στη UML υπάρχουν τα *διαγράμματα πακέτων (package diagrams)*. Το *πακέτο* είναι μια δομή που ομαδοποιεί μεμονωμένα στοιχεία σε ομάδες υψηλότερου επιπέδου. Για παράδειγμα, στη γλώσσα προγραμματισμού Java υπάρχει η έννοια του πακέτου, στη C# και C++ η έννοια του *ονόματος χώρου (namespace)*, παρόλο που αυτά δεν είναι 100% το ίδιο, εξυπηρετούν παρόμοιους σκοπούς. Τα διαγράμματα πακέτων χρησιμοποιούνται για να δει κανείς τις εξαρτήσεις μεταξύ των πακέτων. Ένα πακέτο μπορεί να περιέχει μέσα του είτε άλλα πακέτα, είτε κλάσεις. Ο συνδυασμός του ονόματος πακέτου και του ονόματος της κλάσης είναι μοναδικός σε μια εφαρμογή. Το πλήρες όνομα μιας κλάσης είναι ο συνδυασμός του ονόματος του πακέτου και του όνοματός της, για παράδειγμα στη Java υπάρχει η κλάση `Date` που υπάρχει σε δύο πακέτα, στα `java.util` και `java.sql`. Έτσι, το πλήρες όνομα της κλάσης (στη UML) είναι `java::util::Date` ή `java::sql::Date` αντίστοιχα. Στο σχήμα **2.5.13** απεικονίζεται το πακέτο με όνομα `java` με δύο διαφορετικούς τρόπους απεικόνισης. Στο σχήμα **2.5.14** εμφανίζονται τρεις διαφορετικοί τρόποι απεικόνισης φωλιασμένων πακέτων, ενώ στο δεύτερο και τρίτο πακέτο εμφανίζεται και η κλάση `Date`.



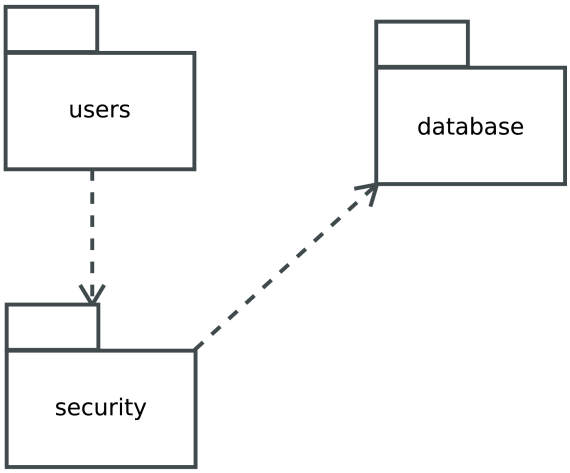
Σχήμα **2.5.13** Ένα πακέτο με δύο διαφορετικούς τρόπους απεικόνισης

Στο σχήμα **2.5.15** απεικονίζονται οι εξαρτήσεις μεταξύ των πακέτων `users`, `security` και `database`. Το πακέτο `users` εξαρτάται από το `security` το οποίο με τη σειρά του εξαρτάται από το `database`. Παράλληλα, υπάρχει η έννοια της *εισαγωγής (import)* με την οποία ένα πακέτο μπορεί να εισάγει ένα άλλο πακέτο, ώστε να μπορεί να αναφέρεται στα στοιχεία του δεύτερου χωρίς να χρειάζεται να χρησιμοποιεί το πλήρες όνομά του. Ένα τέτοιο παράδειγμα απεικονίζει

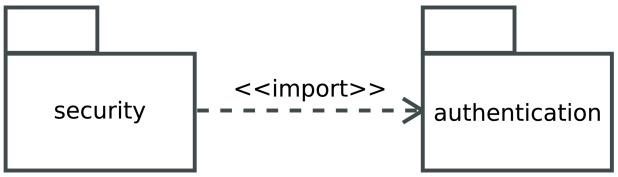


Σχήμα **2.5.14** Φωλιασμένα πακέτα και συμπερίληψη κλάσης σε αυτά

το σχήμα **2.5.16**, στο οποίο το πακέτο `security` εισάγει το πακέτο `authentication`.



Σχήμα **2.5.15** Εξαρτήσεις μεταξύ πακέτων



Σχήμα **2.5.16** Εισαγωγή του πακέτου `authentication` από το `security`

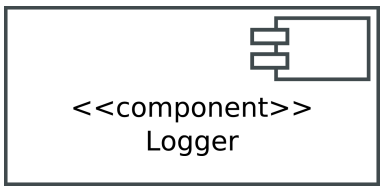
Διαγράμματα συστατικών (Component diagrams)

Το *συστατικό (component)* αποτελεί ένα ενθυλακωμένο, επαναχρησιμοποιήσιμο τμήμα λογισμικού που μπορεί εύκολα να αντικατασταθεί. Μπορεί να σκεφτεί κανείς τα συστατικά ως «τούβλα» που συνδυαζόμενα δίνουν μορφή στο λογισμικό. Τα συστατικά εκτελούν συγκεκριμένη λειτουργικότητα, για παράδειγμα, ένας *γραμματικός αναλυτής (parser)* για XML ή ένας *καταγραφέας συμβάντων ημερολογίου (logger)* αποτελούν περιπτώσεις συστατικών. Για το λό-

γο αυτό, τα συστατικά (όπως και γενικότερα οι κλάσεις σε ένα αντικειμενοστραφές λογισμικό) πρέπει να έχουν χαλαρή *σύζευξη (coupling)* ώστε αλλαγές σε αυτά να μην επηρεάζουν άλλες κλάσεις. Έτσι, επιβάλλεται τα συστατικά να χρησιμοποιούνται μέσω *διασυνδέσεων (interfaces)* για να παρέχουν την κατάλληλη *αφαίρεση (abstraction)*.

Στη UML ένα συστατικό μπορεί να εφαρμόζει ό,τι εφαρμόζουν και οι κλάσεις. Η κύρια διαφορά μεταξύ ενός συστατικού και μιας κλάσης είναι ότι το πρώτο γενικά έχει μεγαλύτερες ευθύνες από τη δεύτερη.

Στο σχήμα **2.5.17** φαίνεται η σημειογραφία ενός συστατικού στη UML 2.0. Στην έκδοση 1.x η παράσταση είναι λίγο διαφορετική.



Σχήμα **2.5.17** Σημειογραφία ενός συστατικού στη UML

Οι *παρεχόμενες διασυνδέσεις (provided interfaces)* είναι οι διασυνδέσεις που υλοποιεί ένα συστατικό και περιγράφουν τις υπηρεσίες που αυτό παρέχει. Οι *απαραίτητες διασυνδέσεις (required interfaces)* είναι οι διασυνδέσεις που χρειάζεται το συστατικό για να λειτουργήσει και περιγράφουν τις υπηρεσίες που αυτό χρειάζεται για τη λειτουργία του.

Στο σχήμα **2.5.18** απεικονίζεται ένα απλό διάγραμμα συστατικών. Το συστατικό `Order` χρησιμοποιεί τη διασύνδεση `Cart` του συστατικού `Shopping Cart`, το οποίο με τη σειρά του χρησιμοποιεί τη διασύνδεση `DataSource` του συστατικού `MySQLDB`.

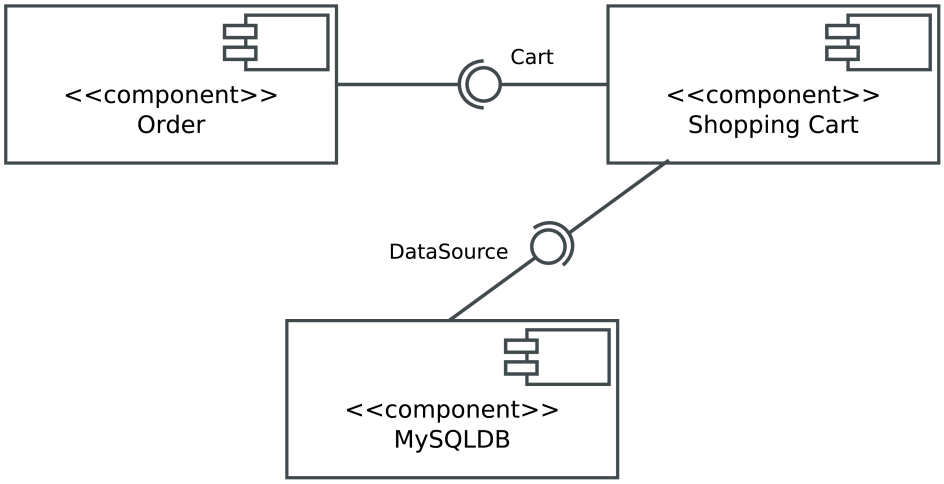
2.5.4 Φυσική όψη

Στο υποκεφάλαιο αυτό περιγράφονται τα διαγράμματα ανάπτυξης που ανήκουν στη φυσική όψη.

Διαγράμματα ανάπτυξης (Deployment diagrams)

Όπως αναφέρθηκε στο §2.5 στη σελίδα 15, τα *διαγράμματα ανάπτυξης (deployment diagrams)* δείχνουν τη φυσική όψη του συστήματος και δίνουν μορφή στο λογισμικό δείχνοντας πως τα διάφορα τμήματα επικοινωνούν μεταξύ τους και πως αυτά ανατίθενται στο *υλικό (hardware)*.

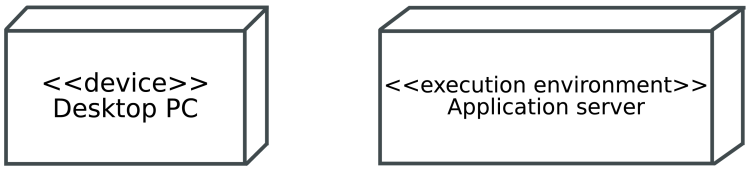
Ο *κόμβος (node)* είναι κάτι και μπορεί να εκτελέσει ένα κομμάτι λογισμικού που μπορεί να υπάρχει με τη μορφή υλικού ή ενός *περιβάλλοντος εκτέλεσης (execution environment)*. Το



Σχήμα **2.5.18** Ένα απλό διάγραμμα συστατικών

περιβάλλον εκτέλεσης είναι ειδικό λογισμικό το οποίο μπορεί να φιλοξενεί και να εκτελεί άλλο λογισμικό. Παραδείγματα κόμβων υλικού είναι ένας διακομιστής, ένα σταθμός εργασίας ή ένας οδηγός δίσκου. Παραδείγματα κόμβων περιβάλλοντος εκτέλεσης είναι ένα λειτουργικό σύστημα, ένα Java EE container, ένας διακομιστής ιστοσελίδων ή εφαρμογών.

Η σημειογραφία ενός κόμβου υλικού και περιβάλλοντος εκτέλεσης εμφανίζεται στο σχήμα **2.5.19**. Το σχήμα απεικονίζει ένα τυπικό ηλεκτρονικό υπολογιστή, καθώς και ένα διακομιστή εφαρμογών.

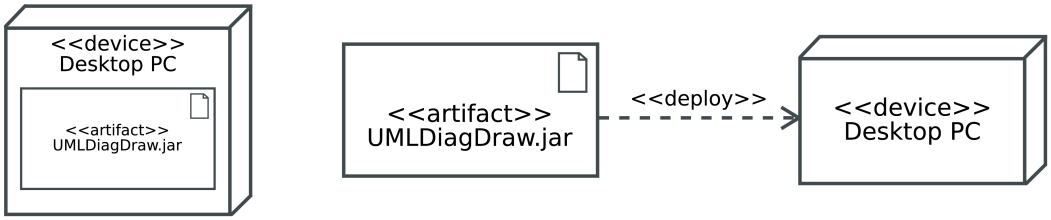


Σχήμα **2.5.19**: Ο κόμβος ενός τυπικού ηλεκτρονικού υπολογιστή και ενός διακομιστή εφαρμογών σε ένα διάγραμμα ανάπτυξης

Η σημειογραφία του λογισμικού απεικονίζεται στο σχήμα **2.5.20**. Το λογισμικό συμβολίζεται με το στερεότυπο `<<artifact>>`.

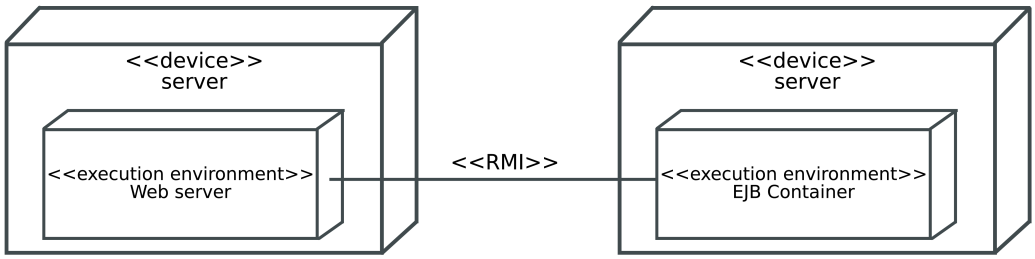
Σχήμα **2.5.20** Ο κόμβος ενός λογισμικού σε ένα διάγραμμα ανάπτυξης

Η ανάπτυξη του λογισμικού στον υπολογιστή είναι το πιο απλό διάγραμμα ανάπτυξης και απεικονίζεται στο σχήμα **2.5.21**.



Σχήμα **2.5.21** Το απλούστερο διάγραμμα ανάπτυξης με δύο διαφορετικές σημειογραφίες

Οι κόμβοι επικοινωνούν μεταξύ τους για να φέρουν σε πέρας την εργασία. Για την επικοινωνία αυτή χρησιμοποιούνται τα *μονοπάτια επικοινωνίας (communication paths)* που εμφανίζονται με συνεχείς γραμμές μεταξύ των κόμβων. Το σχήμα **2.5.22** απεικονίζει την επικοινωνία μεταξύ δύο διαφορετικών περιβαλλόντων εκτέλεσης, ενός διακομιστή ιστοσελίδων και ενός EJB container. Το μονοπάτι επικοινωνίας έχει το στερεότυπο `<<RMI>>` που δείχνει ότι οι κόμβοι επικοινωνούν μέσω του πρωτοκόλλου Remote Method Invocation (RMI).



Σχήμα **2.5.22** Δύο κόμβοι επικοινωνούν μεταξύ τους μέσω του μονοπατιού επικοινωνίας

2.5.5 Όψη περίπτωσης χρήσης

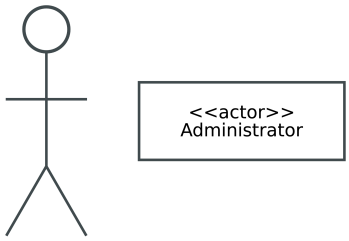
Στο υποκεφάλαιο αυτό περιγράφονται τα διαγράμματα περιπτώσεων χρήσης, περιγραφών και επισκόπησης που ανήκουν στην όψη περίπτωσης χρήσης.

Διαγράμματα περιπτώσεων χρήσης (Use case diagrams)

Η *περίπτωση χρήσης (use case)* είναι μια περίπτωση όπου το σύστημα εκπληρώνει μία ή περισσότερες απαιτήσεις του χρήστη. Η περίπτωση χρήσης αποτυπώνει ένα τμήμα της λειτουργικότητας που παρέχει το σύστημα. Οι περιπτώσεις χρήσης είναι η καρδιά του συστήματος, γιατί επηρεάζουν και καθοδηγούν όλα τα υπόλοιπα στοιχεία του συστήματος, όπως φαίνεται και

στο σχήμα **2.5.2** του μοντέλου του Kruchten. Οι περιπτώσεις χρήσης αποτελούν ένα άριστο σημείο εκκίνησης σχεδίασης, ανάπτυξης, δοκιμής και τεκμηρίωσης για κάθε πλευρά ενός αντικειμενοστραφούς συστήματος. Ένας άλλος ορισμός που δίνει ο Fowler (M. Fowler 2004ε) για την περίπτωση χρήσης είναι ότι αποτελεί ένα σύνολο *σεναρίων (scenarios)* που δένονται μαζί γύρω από ένα κοινό σκοπό του χρήστη. Το σενάριο είναι μια σειρά από βήματα που ακολουθεί ο χρήστης κατά τη διάδρασή του με το σύστημα.

Σε ένα διάγραμμα περίπτωσης χρήσης, κάτι που έρχεται σε διάδραση με το σύστημα ονομάζεται *actor*. Η σημειογραφία του actor απεικονίζεται στο σχήμα **2.5.23**. Η σημειογραφία είναι δύο μορφών, είτε με το «ανθρωπάκι», είτε με το ορθογώνιο και το στερεότυπο `<<actor>>`. Ο actor δεν είναι υποχρεωτικό να είναι μόνο άνθρωπος, αλλά μπορεί για παράδειγμα να είναι το σύστημα ενός τρίτου σε μια εφαρμογή επιχείρησης-σε-επιχείρηση (business-to-business).

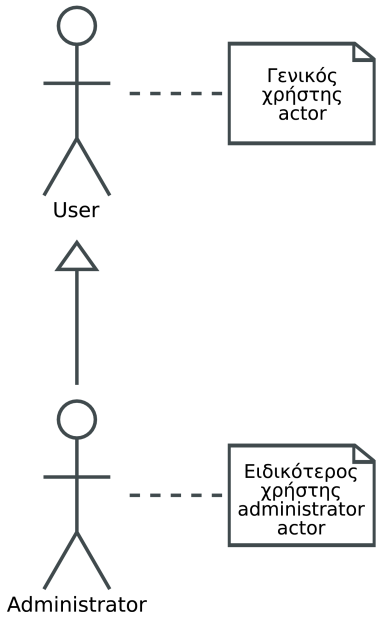


Σχήμα **2.5.23**: Ο actor στο διάγραμμα περίπτωσης χρήσης με δύο διαφορετικές σημειογραφίες

Για τους actor υπάρχει κληρονομικότητα όπως και στις κλάσεις. Στο σχήμα **2.5.24** απεικονίζεται ο actor `Administrator` που είναι μια ειδική περίπτωση του actor `User`.

Η περίπτωση χρήσης στο αντίστοιχο διάγραμμα συμβολίζεται με ένα οβάλ σχήμα μέσα στο οποίο υπάρχει η περιγραφή της διάδρασης που αντιπροσωπεύει, όπως για παράδειγμα φαίνεται στο σχήμα **2.5.25**.

Οι δύο αυτές έννοιες (actor και περίπτωση χρήσης) συνδυάζονται μεταξύ τους μέσω των *γραμμών επικοινωνίας (communication lines)*. Μια γραμμή επικοινωνίας θα συνδέσει τον actor `Administrator` με την περίπτωση χρήσης για να δείξει ότι το πρώτο συμμετέχει στο δεύτερο. Το σχήμα **2.5.26** δείχνει το συνδυασμό αυτό, ενώ στο σχήμα **2.5.27** απεικονίζεται ένα λίγο πιο σύνθετο διάγραμμα με τρεις actor που λαμβάνουν μέρος στη διαδικασία της εισόδου (login) σε ένα σύστημα.



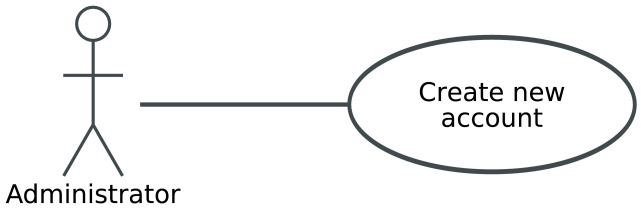
Σχήμα **2.5.24**: Ο actor στο διάγραμμα περίπτωσης χρήσης με δύο διαφορετικές σημειογραφίες

Σχήμα **2.5.25** Η περίπτωση χρήσης συμβολίζεται με ένα οβάλ σχήμα

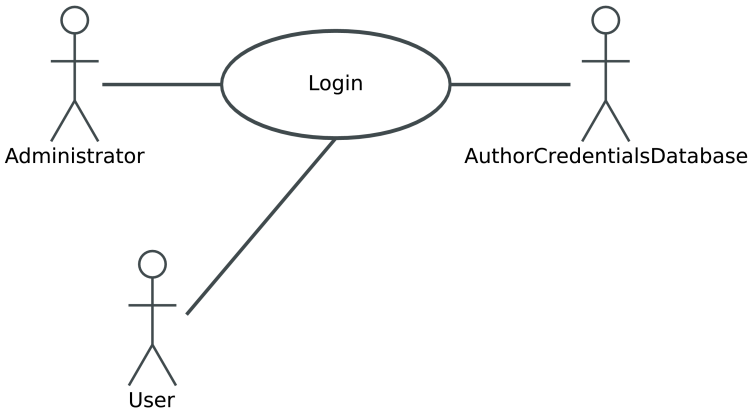
2.6 Διαγράμματα κλάσεων (class diagrams)

Τα διαγράμματα κλάσεων είναι από τα διαγράμματα της UML που χρησιμοποιούνται ευρέως. Στο §2.5.1 στη σελίδα 17 περιγράφηκαν τα διαγράμματα αντικειμένων. Ενώ τα τελευταία περιγράφουν στιγμές του συστήματος μέσω των στιγμιοτύπων, τα διαγράμματα κλάσεων περιγράφουν τη δομή του συστήματος και τους διαφορετικούς τύπους αντικειμένων που υπάρχουν. Παράλληλα, απεικονίζουν τις στατικές συσχετίσεις μεταξύ των τύπων των αντικειμένων. Η UML χρησιμοποιεί το γενικό όρο *χαρακτηριστικό (feature)* για να περιγράψει τις ιδιότητες και λειτουργίες μια κλάσης.

Σε ένα διάγραμμα κλάσης μια κλάση συμβολίζεται με ένα ορθογώνιο, το οποίο χωρίζεται σε τρεις περιοχές μέσω οριζόντιων γραμμών. Στην πρώτη περιοχή (πάνω), αναγράφεται το όνομα της κλάσης, στη δεύτερη περιοχή (μέση) αναγράφονται οι *ιδιότητες (attributes)* και στην τρίτη περιοχή (κάτω) αναγράφονται οι *λειτουργίες (operations)*. Από τις τρεις αυτές περιοχές, μόνο η πρώτη είναι υποχρεωτική. Μια κλάση μπορεί να αναγράφει το όνομα και τις ιδιότητές

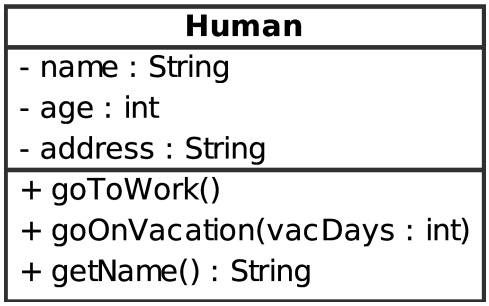


Σχήμα **2.5.26** Το πιο απλό διάγραμμα περίπτωσης χρήσης



Σχήμα **2.5.27**: Ένα πιο σύνθετο διάγραμμα περίπτωσης χρήσης με τρεις actor και μια περίπτωση χρήσης

της, το όνομα και τις λειτουργίες της ή και τα τρία χαρακτηριστικά μαζί. Ένα παράδειγμα μιας κλάσης Human φαίνεται στο σχήμα **2.6.28**. Πολλές φορές για λόγους συντομίας παραλείπεται η αναγραφή των ιδιοτήτων και λειτουργιών μιας κλάσης.



Σχήμα **2.6.28** Η σημειογραφία της κλάσης στο διάγραμμα κλάσεων

2.6.1 Αφηρημένες κλάσεις (Abstract classes)

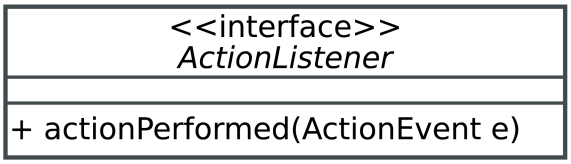
Μια *αφηρημένη (abstract)* κλάση είναι μια κλάση που παρέχει την *υπογραφή (signature)* μιας ή περισσότερων λειτουργιών, αλλά η κλάση δεν παρέχει την υλοποίηση αυτών των λειτουργιών.

Ουσιαστικά, σε μια αφηρημένη κλάση μία ή περισσότερες λειτουργίες είναι αφηρημένες. Οι αφηρημένες κλάσεις χρησιμοποιούνται για να αναγνωρίζουν κοινή λειτουργικότητα μεταξύ κλάσεων. Όταν μια κλάση περιέχει τουλάχιστον μία αφηρημένη λειτουργία, τότε η κλάση θεωρείται αφηρημένη ανεξάρτητα αν περιέχει λειτουργίες τις οποίες υλοποιεί. Στην περίπτωση αυτή, στο διάγραμμα κλάσεων το όνομα της κλάσης εμφανίζεται με πλάγια γράμματα.

Αντικείμενο αφηρημένης κλάσης δε μπορεί να δημιουργηθεί. Αντίθετα, μπορεί να δημιουργηθεί ένα αντικείμενο μιας υποκλάσης της αφηρημένης κλάσης που υλοποιεί τις αφηρημένες λειτουργίες της τελευταίας.

2.6.2 Διασυνδέσεις (Interfaces)

Οι *διασυνδέσεις (interfaces)* είναι μια ειδική περίπτωση τύπων. Θα μπορούσαν να παρομοιαστούν με αφηρημένες κλάσεις των οποίων **όλες** οι λειτουργίες είναι αφηρημένες. Μια διασύνδεση μπορεί να περιέχει ιδιότητες και λειτουργίες όπως μια κλάση. Μια διασύνδεση στο διάγραμμα κλάσεων συμβολίζεται με το στερεότυπο `<<interface>>` πάνω από το όνομά της, ενώ οι λειτουργίες της συμβολίζονται με ίσια γράμματα, γιατί σε μια διασύνδεση οι λειτουργίες είναι εξ' ορισμού αφηρημένες. Στο σχήμα **2.6.29** φαίνεται η διασύνδεση *ActionListener* της γλώσσας προγραμματισμού Java.



Σχήμα **2.6.29** Παράδειγμα διασύνδεσης

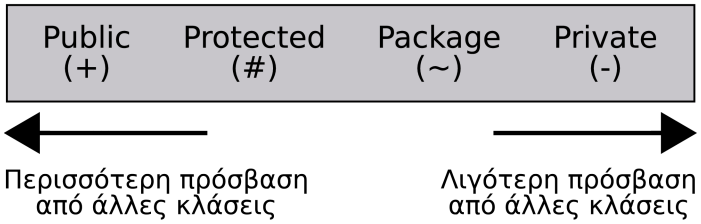
Αντικείμενα μιας διασύνδεσης μπορούν να δηλωθούν, αλλά θα δημιουργηθούν μέσω κλάσεων που *υλοποιούν (implement)* τη διασύνδεση. Η υλοποίηση γίνεται μέσω μιας συσχέτισης που θα συζητηθεί παρακάτω (βλ. §2.6.6 στη σελίδα 40).

Μια κλάση πολλές φορές απαιτεί ορισμένες διασυνδέσεις για να δουλεύει. Στην περίπτωση αυτή η κλάση *απαιτεί (requires)* τις διασυνδέσεις αυτές. Στην περίπτωση που μια κλάση μπορεί να χρησιμοποιηθεί στη θέση μιας ή περισσοτέρων διασυνδέσεων, τότε η κλάση *παρέχει (provides)* τις διασυνδέσεις αυτές.

2.6.3 Ορατότητα (Visibility)

Η *ορατότητα (visibility)* ορίζει το πως μια κλάση επιτρέπει την πρόσβαση στις ιδιότητες και μεθόδους της προς τον έξω κόσμο. Με τη φράση «έξω κόσμο» εννοούνται οι άλλες κλάσεις που ανήκουν είτε στο ίδιο πακέτο είτε σε διαφορετικό με την εν λόγω κλάση.

Υπάρχουν τέσσερα είδη ορατότητας στη UML. Στο σχήμα **2.6.30** φαίνονται ποια είναι αυτά.



Σχήμα **2.6.30** Τα τέσσερα είδη ορατότητας στη UML

Όπως φαίνεται και στο σχήμα, από αριστερά προς τα δεξιά ο βαθμός ορατότητας μικραίνει.

Public (δημόσια) Όταν μια ιδιότητα ή λειτουργία έχει τη δημόσια ορατότητα, τότε είναι ορατή από οποιονδήποτε.

Protected (προστατευμένη) Όταν μια ιδιότητα ή λειτουργία μιας κλάσης έχει την προστατευμένη ορατότητα, τότε είναι ορατή μόνο από τις υποκλάσεις της αρχικής.

Package (πακέτου) Όταν μια ιδιότητα ή λειτουργία μιας κλάσης έχει την ορατότητα πακέτου, τότε είναι ορατή μόνο από τις κλάσεις που ανήκουν στο ίδιο πακέτο με αυτό της αρχικής.

Private (ιδιωτική) Όταν μια ιδιότητα ή λειτουργία μιας κλάσης έχει την ιδιωτική ορατότητα, τότε δεν είναι ορατή από οποιαδήποτε άλλη κλάση.

Σε κάθε περίπτωση οι ιδιότητες και λειτουργίες μιας κλάσης εννοείται ότι είναι ορατές από την ίδια την κλάση, γι' αυτό και δεν αναφέρεται παραπάνω.

2.6.4 Ιδιότητες (Attributes)

Οι ιδιότητες μιας κλάσης είναι τα κομμάτια πληροφορίας που απαρτίζουν την κατάσταση ενός αντικειμένου. Σε μια κλάση, οι ιδιότητες αναγράφονται ως εξής:

`ορατότητα´ / `όνομα´ : `τύπος´ `πολλαπλότητα´ = `εξ´ ορισμού τιμή´ {`ειδικές ιδιότητες/περιορισμοί´}

Το *όνομα* είναι το όνομα της ιδιότητας. Το / μπροστά από το όνομα υποδηλώνει ότι η συγκεκριμένη ιδιότητα κληρονομείται από τη γονική κλάση.

Ο *τύπος* είναι ο τύπος της ιδιότητας, που μπορεί να είναι πρωτογενής (πχ. `int`, `boolean`, `String`, κτλ.) ή κλάση.

Η *πολλαπλότητα (multiplicity)* είναι ένδειξη του αριθμού των αντικειμένων μιας ιδιότητας. Οι συνήθεις ενδείξεις πολλαπλότητας είναι:

1 Υπάρχει ακριβώς ένα στιγμιότυπο της ιδιότητας.

0..1 Υπάρχει το πολύ ένα στιγμιότυπο της ιδιότητας.

***** Υπάρχει απροσδιόριστος αριθμός στιγμιοτύπων της ιδιότητας. Το `0..*` γράφεται απλά `*`.

x..y Υπάρχουν από x έως και y στιγμιότυπα της ιδιότητας. Όταν τα x και y είναι ο ίδιος αριθμός, τότε μπορεί να αναγράφεται μόνο ο ένας, για παράδειγμα το `2..2` μπορεί να γραφεί ως 2.

Η *εξ´ ορισμού τιμή* είναι αυτό που φανερώνει η περιγραφή της, η αρχική τιμή που δίνεται στην ιδιότητα κατά τη δημιουργία στιγμιοτύπου της κλάσης.

Οι *ειδικές ιδιότητες/περιορισμοί* είναι μια λίστα από ιδιότητες/περιορισμούς, όπως για παράδειγμα `{readonly}` που δείχνει ότι η τιμή της συγκεκριμένης ιδιότητας δεν αλλάζει, `{ordered}` που δείχνει ότι η σειρά των αντικειμένων σε μια δομή παίζει ρόλο (αυτό φυσικά έχει νόημα για δομές που αποθηκεύουν πολλά αντικείμενα) ή `{unique}` που υποδεικνύει ότι σε μια δομή δεν επιτρέπεται η ύπαρξη των ίδιων αντικειμένων παραπάνω από μία φορά.

Στο σχήμα **2.6.28** φαίνονται τρεις ιδιότητες με τα ονόματα και τους τύπους τους.

Παράλληλα μια ιδιότητα μπορεί να είναι στατική, δηλαδή όλες οι κλάσεις να μοιράζονται το ίδιο στιγμιότυπο της ιδιότητας (ονομάζεται και στιγμιότυπο κλάσης). Σε αυτή την περίπτωση, η περιγραφή της ιδιότητας είναι υπογραμμισμένη, όπως για παράδειγμα
`- count : int`

2.6.5 Λειτουργίες (Operations)

Η λειτουργία μιας κλάσης υποδηλώνει το *τι* μπορεί να κάνει μια κλάση, αλλά όχι απαραίτητα το *πως* πρέπει να το κάνει. Μια λειτουργία είναι περισσότερο μια υπόσχεση ή ένα μικρό συμβόλαιο

που δηλώνει ότι μια κλάση θα έχει κάποια συμπεριφορά ανάλογη με αυτό που φανερώνει το όνομά της. Το σύνολο των λειτουργιών μιας κλάσης περιλαμβάνει το σύνολο της συμπεριφοράς μιας κλάσης.

Η πλήρης σύνταξη UML για μια λειτουργία είναι:

`ορατότητα´ `όνομα´ (`παράμετροι´) : `τύπος επιστροφής´ {`ειδικές ιδιότητες´}

Για την ορατότητα ισχύει ό,τι και για τις ιδιότητες.

Το όνομα είναι το όνομα της λειτουργίας.

Οι παράμετροι είναι η λίστα παραμέτρων που δέχεται η λειτουργία. Κάθε παράμετρος είναι της μορφής `κατεύθυνση´ `όνομα´ : `τύπος´ = `εξ´ ορισμού τιμή´ με διαχωριστή το , (κόμμα). Η κατεύθυνση ορίζει αν η παράμετρος θα περάσει τιμή στη λειτουργία (**in**), θα πάρει τιμή μέσω της λειτουργίας και την επιστρέψει στον καλών (**out**) ή και τα δύο (**inout**). Η κατεύθυνση μπορεί να παραληφθεί. Σε αυτή την περίπτωση αυτή θεωρείται ότι πρόκειται για in παράμετρο.

Ο τύπος επιστροφής είναι ο τύπος που επιστρέφει η λειτουργία. Μπορεί να είναι το τίποτα (void) ή κάποιος από τους τύπους που μπορεί να είναι μια ιδιότητα.

Όπως και στις ιδιότητες, μια λειτουργία μπορεί να είναι στατική. Στην περίπτωση αυτή αναγράφεται με υπογραμμισμένα γράμματα.

Αφηρημένες λειτουργίες (Abstract operations)

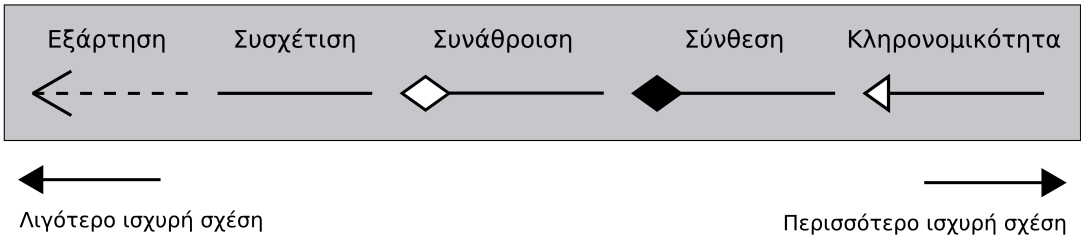
Στο §2.6.1 γίνεται αναφορά στις αφηρημένες κλάσεις. Η ύπαρξη μιας λειτουργίας ως αφηρημένη έχει ως αποτέλεσμα να θεωρείται αφηρημένη και η κλάση στην οποία ανήκει. Όπως το όνομα μιας αφηρημένης κλάσης αναγράφεται με πλάγια γράμματα, έτσι και τα ονόματα των αφηρημένων μεθόδων αναγράφονται επίσης με πλάγια γράμματα.

2.6.6 Σχέσεις (Relationships)

Σε ένα διάγραμμα κλάσεων της UML οι κλάσεις από μόνες τους δεν λένε πολλά για το σύστημα. Αυτό που θα δώσει ζωή στο διάγραμμα κλάσεων είναι οι *σχέσεις (relationships)* μεταξύ των κλάσεων, από τις οποίες ο ενδιαφερόμενος μπορεί να κατανοήσει πως οι κλάσεις συσχετίζονται μεταξύ τους.

Υπάρχουν πέντε είδη σχέσεων, τα οποία απεικονίζονται στο σχήμα **2.6.31**. Η ισχύς μιας σχέσης μεταξύ δύο κλάσεων ορίζεται ανάλογα με το πως αυτές σχετίζονται μεταξύ τους. Σε κάποια είδη σχέσεων, μπορεί να υπάρξει κατεύθυνση (η φορά του βέλους). Για παράδειγμα,

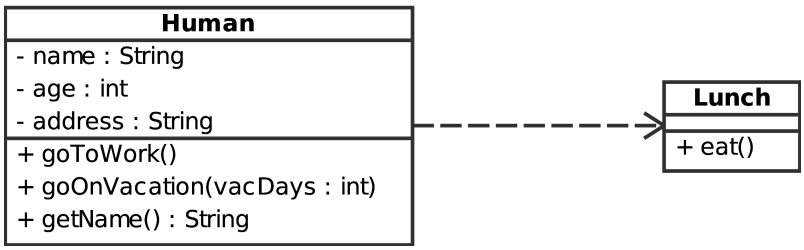
στο σχήμα **2.6.32** φαίνεται ότι ένα αντικείμενο `Human` εξαρτάται από ένα αντικείμενο `Lunch` και όχι το αντίστροφο (από ένα στιγμιότυπο δηλαδή του τύπου `Human` μπορεί να πάρει κανείς την αναφορά προς το στιγμιότυπο του τύπου `Lunch`).



Σχήμα **2.6.31** Τα είδη σχέσεων μεταξύ κλάσεων

Εξάρτηση (Dependence)

Η *εξάρτηση* μεταξύ δύο κλάσεων υποδηλώνει ότι μια κλάση χρειάζεται να ξέρει πως πρέπει να δουλέψει με αντικείμενα μιας άλλης κλάσης. Αυτό υποννοεί ότι αλλαγές της μιας κλάσης, επηρεάζουν την άλλη. Στο σχήμα **2.6.32** η κλάση `Human` εξαρτάται από την κλάση `Lunch`. Για παράδειγμα, μέσα στη λειτουργία `goToWork()` μπορεί να δημιουργείται ένα προσωρινό αντικείμενο του τύπου `Lunch` και να καλείται η λειτουργία `eat()` της τελευταίας.



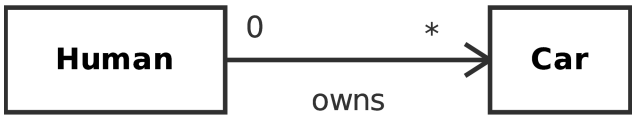
Σχήμα **2.6.32** Εξάρτηση μεταξύ δύο κλάσεων

Η σχέση εξάρτησης υποδηλώνει ότι δύο κλάσεις *μπορούν* να δουλέψουν μαζί και θεωρείται ως η ασθενέστερη σχέση μεταξύ τους.

Συσχέτιση (Association)

Η *συσχέτιση* μεταξύ δύο κλάσεων υποδηλώνει ότι η μία θα περιέχει αναφορά προς αντικείμενο της άλλης με τη μορφή μέλους. Το μέλος μπορεί να είναι είτε μια ιδιότητα της κλάσης, είτε μια παράμετρος μιας λειτουργίας. Αυτό σημαίνει ότι μια ιδιότητα κλάσης που είναι τύπος άλλης

κλάσης μπορεί να παρασταθεί και ως συσχέτιση μεταξύ της πρώτης και της δεύτερης. Στο σχήμα **2.6.33** η κλάση `Human` μπορεί να περιέχει οποιονδήποτε αριθμό από αντικείμενα τύπου `Car`. Στο σχήμα αναγράφεται η πολλαπλότητα μεταξύ των δύο (`0..*`), το όνομα της σχέσης `owns` και η κατευθυντικότητα (από τον άνθρωπο, μπορεί να βρει κανείς ποια αυτοκίνητα έχει, ενώ το αντίστροφο δεν ισχύει – χάριν παραδείγματος, το αυτοκίνητο δεν έχει μυαλό για να γνωρίζει σε ποιον ανήκει).



Σχήμα **2.6.33** Συσχέτιση μεταξύ δύο κλάσεων

Συνάθροιση (Aggregation)

Η *συνάθροιση* είναι μια μορφή ισχυρότερης συσχέτισης μεταξύ δύο κλάσεων. Σημαίνει ότι μία κλάση *περιέχει* ένα αντικείμενο μιας άλλης κλάσης και πιθανόν να μοιράζεται το αντικείμενο.

Αρκετοί συγχέουν τη συνάθροιση με τη συσχέτιση. Η διαφορά των δύο είναι απλή και έγκειται στο ότι στη συνάθροιση, μια κλάση που *περιέχεται* σε μια άλλη, είναι *τμήμα* της. Για παράδειγμα, στο σχήμα **2.6.34** παρατηρεί κανείς ότι η κλάση `Airplane` περιέχεται στην κλάση `Airport`.

Σχήμα **2.6.34** Συνάθροιση κλάσεων

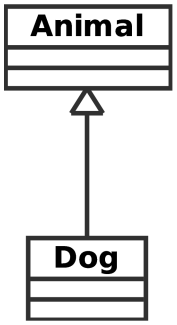
Σύνθεση (Composition)

Η *σύνθεση* δε διαφέρει πολύ από τη συνάθροιση. Η διαφορά τους έγκειται στο ότι στη σύνθεση, τα αντικείμενα της περιεχόμενης κλάσης *δεν* μοιράζονται από άλλα αντικείμενα. Η διαφορά στη σημειογραφία σε σχέση με τη συνάθροιση είναι ότι το διαμάντι είναι «γεμάτο». Για παράδειγμα, στο σχήμα **2.6.35** παρατηρεί κανείς ότι η κλάση `Engine` περιέχεται στην κλάση `Car` με τη διαφορά ότι η μηχανή μπορεί να υπάρχει σε ένα αυτοκίνητο μόνο.

Σχήμα **2.6.35** Σύνθεση κλάσεων

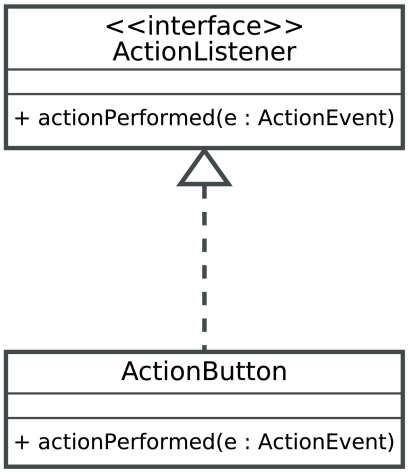
Κληρονομικότητα (Inheritance)

Η *κληρονομικότητα* είναι σχέση που υποδηλώνει εξειδίκευση κλάσεων. Δηλαδή μια κλάση που κληρονομεί μια άλλη υποδηλώνει ότι είναι του τύπου της πρώτης αλλά ταυτόχρονα πιο εξειδικευμένη. Στο σχήμα **2.6.36** η κλάση `Dog` κληρονομεί την `Animal`.



Σχήμα **2.6.36** Κληρονομικότητα κλάσεων

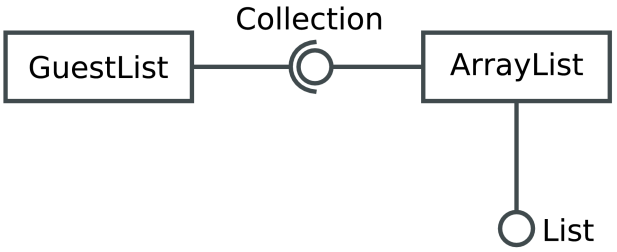
Μια ειδική περίπτωση της σχέσης αυτής είναι η *υλοποίηση (implementation)* ή *πραγματοποίηση (realization)* που αφορά κλάσεις και διασυνδέσεις. Όπως αναφέρθηκε προηγουμένως, οι διασυνδέσεις είναι αφηρημένοι τύποι που περιέχουν αφηρημένες λειτουργίες τις οποίες πρέπει μια κλάση να υλοποιήσει με κώδικα προκειμένου η τελευταία να χρησιμοποιηθεί ως αντικείμενο του τύπου της διασύνδεσης. Αυτό ισχύει για όλες τις λειτουργίες της διασύνδεσης. Όταν μια κλάση υλοποιεί ή πραγματοποιεί μια διασύνδεση, τότε πρέπει εκτός από άλλες (πιθανές) δικές της λειτουργίες να παρέχει την υλοποίηση των λειτουργιών της διασύνδεσης. Αυτού του είδους η σχέση έχει παρόμοια σημειογραφία με αυτήν της κληρονομικότητας, με τη διαφορά ότι η γραμμή μεταξύ κλάσης και διασύνδεσης είναι διακεκομμένη. Στο σχήμα **2.6.37**) απεικονίζεται η κλάση `ActionButton` που υλοποιεί τη διασύνδεση `ActionListener` που περιέχει τη μοναδική αφηρημένη λειτουργία `actionPerformed()`.



Σχήμα **2.6.37** Υλοποίηση διασύνδεσης

2.6.7 Εξάρτηση κλάσης από διασύνδεση

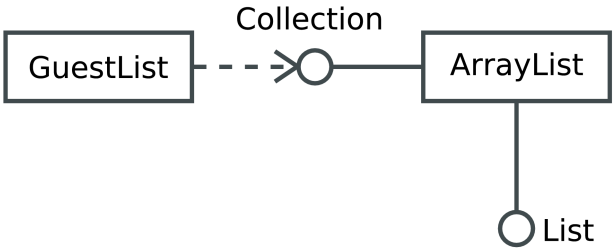
Σε ένα καλά σχεδιασμένο αντικειμενοστραφές σύστημα λογισμικού, η σχεδίαση γίνεται πάντα σε διασυνδέσεις, ώστε αλλαγές στις υλοποιήσεις να μην επηρεάζουν τους πελάτες-κλάσεις των διασυνδέσεων αυτών. Για παράδειγμα, έστω η κλάση `GuestList` στη Java η οποία εξαρτάται από τη διασύνδεση `Collection` προκειμένου να κρατάει σε μια συλλογή μια λίστα ονομάτων (`String`). Όμως ακριβώς επειδή η `Collection` είναι διασύνδεση, ουσιαστικά η `GuestList` εξαρτάται από μια υλοποίηση της πρώτης, για παράδειγμα την κλάση `ArrayList`. Άρα σε επίπεδο υλοποίησης η `GuestList` εξαρτάται από την `ArrayList`. Αυτό φαίνεται με τη *σημειογραφία της μπάλας (ball notation)* στο σχήμα **2.6.38**. Η σημειογραφία του σχήματος **2.6.38** είναι καινούρια και ανήκει στην έκδοση 2.0 της UML. Στο σχήμα **2.6.39** φαίνεται η αντίστοιχη σημειογραφία για την έκδοση UML 1.x.



Σχήμα **2.6.38** Εξάρτηση κλάσης από υλοποίηση διασύνδεσης (έκδοση UML 2.0)

Δηλαδή σε επίπεδο κώδικα, το παραπάνω θα μπορούσε να είναι ως εξής:

```
Collection guests = new ArrayList();
```



Σχήμα **2.6.39** Εξάρτηση κλάσης από υλοποίηση διασύνδεσης (έκδοση UML 1.x)

όπου `guests` είναι μια ιδιότητα της κλάσης `GuestList`.

Παράλληλα, στα σχήματα **2.6.38** και **2.6.39** απεικονίζεται η σημειογραφία της περίπτωσης διασύνδεσης που παρέχει μια διασύνδεση (κύκλος ενωμένος μέσω γραμμής με την κλάση). Η κλάση `ArrayList`, εκτός της διασύνδεσης `Collection`, παρέχει και τη διασύνδεση `List`.

2.6.8 Qualified associations

Σε επίπεδο υλοποίησης, υπάρχουν δομές που επιτρέπουν την αποθήκευση αντικειμένων χρησιμοποιώντας ως δείκτη άλλα αντικείμενα. Παραδείγματα τέτοιων δομών είναι οι συσχετιστικοί πίνακες, τα λεξικά, τα map και hash. Στη Java για παράδειγμα, υπάρχει η κλάση `Map` που επιτρέπει την εισαγωγή αντικειμένων χρησιμοποιώντας για την προσπέλασή τους άλλο αντικείμενο. Έστω ο κώδικας σε Java:

```
String name;
Person person;
  .
  .
Map names = new HashMap();
names.put(name, person);
```

Με αυτό τον κώδικα ορίζεται ένα αντικείμενο τύπου `Map` στο οποίο αποθηκεύονται αντικείμενα τύπου `Object`[2]. Στη συνέχεια, εισάγεται στη δομή αυτή ένα αντικείμενο τύπου `Person` με όνομα `person` και δείκτη ένα αντικείμενο `String` με όνομα `name`. Για να πάρει κανείς το αντικείμενο τύπου `Person` δεδομένου ενός ονόματος, πρέπει να δώσει για παράδειγμα

```
Person p = (Person)names.get("Lourdas");
```

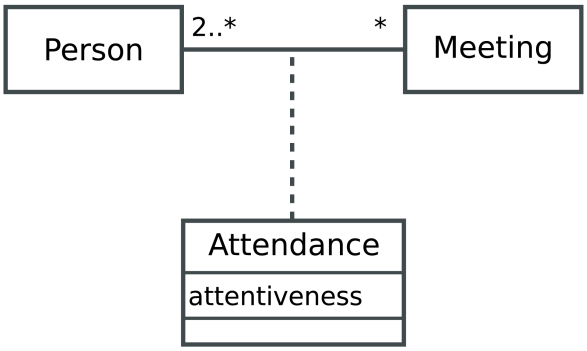
[2]Η κλάση `Object` στη Java είναι η κλάση γονέας όλων των υπολοίπων της γλώσσας.

και το p θα δείχνει σε ένα αντικείμενο τύπου Person. Η σημειογραφία για τέτοιες δομές απεικονίζεται στο σχήμα **2.6.40**.

Σχήμα **2.6.40**: Class qualifier (αντιστοίχιση αντικειμένου τύπου String σε αντικείμενα τύπου Person)

2.6.9 Κλάση συσχέτισης (Association class)

Μια *κλάση συσχέτισης (association class)* δεν είναι κάτι το ιδιαίτερο. Είναι μια κανονική κλάση και χρησιμοποιείται σε περιπτώσεις που η συσχέτιση μεταξύ δύο κλάσεων δεν είναι μια απλή ένωση. Για παράδειγμα, έστω οι κλάσεις Person, Meeting και Attendance. Το σχήμα **2.6.41** παρουσιάζει το διάγραμμα με τις τρεις αυτές κλάσεις. Η κλάση Attendance είναι η κλάση συσχέτισης στην περίπτωση αυτή.

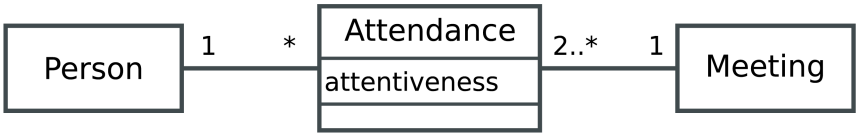


Σχήμα **2.6.41** Η κλάση Attendance είναι η κλάση συσχέτισης

Το ισοδύναμο του διαγράμματος στο σχήμα **2.6.41** χωρίς την κλάση συσχέτισης φαίνεται στο σχήμα **2.6.42**. Υπάρχει ωστόσο μια μικρή διαφορά που εισάγει η έννοια της κλάσης συσχέτισης. Η διαφορά είναι ότι μεταξύ δύο αντικειμένων των τύπων Person και Meeting υπάρχει **ένα** μόνο στιγμιότυπο της κλάσης Attendance.

2.6.10 Κλάσεις πρότυπα (Template classes)

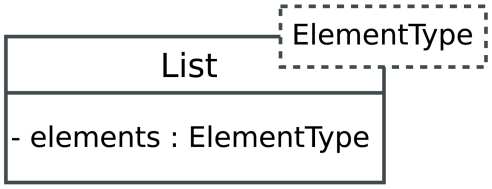
Ορισμένες γλώσσες προγραμματισμού σήμερα (όπως η C++, C#, Java, κά.) παρέχουν τη δυνατότητα στον προγραμματιστή να γράφει «γενικό» κώδικα. Αυτό σημαίνει ότι μπορεί ο



Σχήμα **2.6.42**: Το ισοδύναμο του διαγράμματος του σχήματος **2.6.41** χωρίς την κλάση συσχέτισης

προγραμματιστής να γράφει κώδικα που χειρίζεται αντικείμενα των οποίων ο τύπος δεν έχει αποφασιστεί στη φάση της μεταγλώττισης. Για παράδειγμα, μπορεί να υπάρχει μια κλάση τύπου λίστας που να διαχειρίζεται αντικείμενα οποιουδήποτε τύπου. Στην περίπτωση αυτή, πρέπει όλα τα αντικείμενα που θα εισάγονται στη λίστα να είναι του ίδιου τύπου, ανεξάρτητα από το ποιος είναι αυτός. Μπορεί κανείς να σκεφτεί τη συγγραφή «γενικού» κώδικα ως τη χρησιμοποίηση αφαιρέσεων (abstractions) σε επίπεδο τύπων, ωστόσο δεν πρέπει να συγχέεται με την έννοια της διασύνδεσης.

Μία τέτοια περίπτωση κλάσεως ονομάζεται ***κλάση πρότυπο*** *(template class)*[3]. Στο σχήμα **2.6.43** απεικονίζεται η σημειογραφία της UML για τις κλάσεις πρότυπα. Υπάρχει μια κλάση `List` που διαχειρίζεται αντικείμενα οποιουδήποτε τύπου `ElementType`[4] σε μια δομή λίστας.



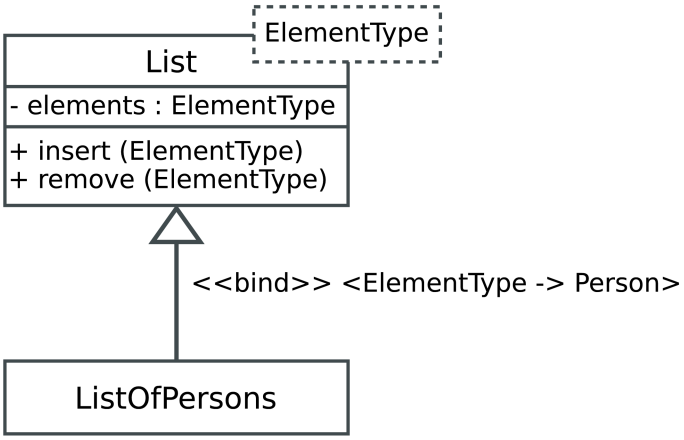
Σχήμα **2.6.43** Η σημειογραφία της κλάσης πρότυπο στη UML

Για να χρησιμοποιηθεί μια κλάση πρότυπο, πρέπει να γίνει μια διαδικασία γνωστή ως *«δέσιμο» παραμέτρου (parameter bind)*. Η σημειογραφία της διαδικασίας αυτής απεικονίζεται στο σχήμα **2.6.44**. Με το στερεότυπο `«bind»` και το `<ElementType -> Person>` η κλάση `ListOfPersons` «κληρονομεί» την `List` ορίζοντας ταυτόχρονα ως κλάση λειτουργίας της την `Person`. Ο τρόπος αυτός της σημειογραφίας ονομάζεται *derivation*, αλλά ο τύπος `ListOfPersons` δεν κληρονομεί τον τύπο `List`.

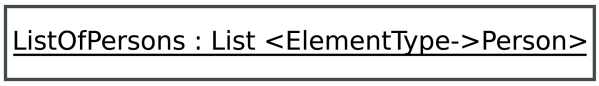
Στο σχήμα **2.6.45** απεικονίζεται ένας εναλλακτικός τρόπος για το «δέσιμο» παραμέτρου.

[3]Ονομάζεται και *παραμετροποιήσιμη (parameterized)* κλάση.

[4]Το `ElementType` είναι ένα ενδεικτικό όνομα του γενικού τύπου, συνήθως το όνομα αυτό είναι `T`.



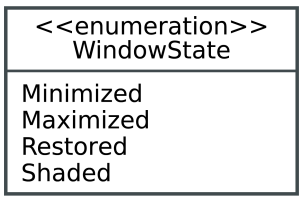
Σχήμα **2.6.44**: Ένας τρόπος σημειογραφίας για το «δέσιμο» παραμέτρου σε μια κλάση πρότυπο



Σχήμα **2.6.45** Εναλλακτική σημειογραφία για το «δέσιμο» παραμέτρου σε μια κλάση πρότυπο

2.6.11 Απαριθμήσεις (Enumerations)

Οι *απαριθμήσεις (enumerations)* είναι ειδικοί τύποι που περιέχουν ένα προκαθορισμένο σύνολο μόνο ιδιοτήτων που δεν έχουν τιμή τίποτα άλλο εκτός από το όνομά τους. Οι τύποι αυτοί συμβολίζονται με το στερεότυπο `<<enumeration>>`. Η σημειογραφία απεικονίζεται στο σχήμα **2.6.46** και δείχνει τις απαριθμήσεις για την κατάσταση ενός παραθύρου σε ένα γραφικό περιβάλλον.



Σχήμα **2.6.46** Οι απαριθμήσεις στη UML

2.6.12 Ενεργές κλάσεις (Active classes)

Μια *ενεργός κλάση (active class)* έχει στιγμιότυπα, καθένα από τα οποία εκτελεί και ελέγχει τα δικά του *νήματα (threads)*[5] ελέγχου. Οι κλήσεις των μεθόδων γίνονται σε ένα νήμα πελάτη ή στο νήμα της ενεργούς κλάσης. Στο σχήμα **2.6.47** απεικονίζεται η ενεργός κλάση `MainApp`.

Σχήμα **2.6.47** Μία ενεργός κλάση

[5]Τα νήματα μπορούν να θεωρηθούν ως υπο-διεργασίες μιας διεργασίας ενός λειτουργικού συστήματος, αλλά διαφέρουν στον τρόπο με τον οποίο μοιράζονται τους πόρους σε σχέση με τις διεργασίες.

ΚΕΦΑΛΑΙΟ 3

Εφαρμογή UMLDiagDraw

3.1 Εισαγωγή

Στο κεφάλαιο αυτό περιγράφεται η εφαρμογή *αντίστροφης μηχανικής (reverse engineering)* UMLDiagDraw που αναπτύχθηκε στα πλαίσια της εργασίας. Σκοπός της εφαρμογής είναι η ανάγνωση ενός καταλόγου αρχείων bytecode της Java και η εξαγωγή των συσχετίσεων μεταξύ των κλάσεων αυτών σε μορφή διαγράμματος κλάσεων της UML.

Αρχικά, θα γίνει μια περιγραφή των βιβλιοθηκών BCEL και JGraph που χρησιμοποιήθηκαν κατά την ανάπτυξη της εφαρμογής. Στη συνέχεια, θα γίνει μια περιγραφή του τρόπου εύρεσης των συσχετίσεων μεταξύ των κλάσεων και μια συνοπτική περιγραφή για την αρχιτεκτονική της εφαρμογής. Θα ακολουθήσει ένας σύντομος οδηγός χρήσης της εφαρμογής με εικόνες και τέλος θα αναφερθούν δυσκολίες και προβλήματα κατά την πορεία ανάπτυξής της.

3.2 Στόχος της εφαρμογής UMLDiagDraw

Ο σκοπός της εφαρμογής UMLDiagDraw είναι η αντίστροφη μηχανική κώδικα bytecode της Java και η εξαγωγή του διαγράμματος κλάσεων της UML για τις κλάσεις που περιλαμβάνονται στον κώδικα bytecode. Η εφαρμογή αναπτύχθηκε σε Java και περιλαμβάνει γραφική διεπαφή για την αλληλεπίδραση με το χρήστη (επιλογή καταλόγου με τα αρχεία του bytecode και εμφάνιση του διαγράμματος των κλάσεων).

Στο §3.6 στη σελίδα 63 υπάρχει ένας οδηγός χρήσης της εφαρμογής, όπου ο αναγνώστης μπορεί να διαβάσει για τις δυνατότητες της εφαρμογής.

3.3 Βιβλιοθήκες που χρησιμοποιήθηκαν

Η εφαρμογή βασίστηκε κυρίως σε δύο βιβλιοθήκες της Java. Η μία είναι το *Byte Code Engineering Library (BCEL)* και η άλλη το *JGraph*. Η πρώτη χρησιμοποιείται για την ανάγνωση του bytecode και την εξαγωγή πληροφοριών για τις κλάσεις. Η δεύτερη χρησιμοποιείται για το σχηματισμό των γραφικών και συγκεκριμένα των ορθογωνίων που παριστάνουν τις κλάσεις και των γραμμών που παριστάνουν τις συσχετίσεις μεταξύ των κλάσεων.

3.3.1 Βιβλιοθήκη Byte Code Engineering Library

Η βιβλιοθήκη Byte Code Engineering Library (BCEL) είναι μια βιβλιοθήκη που παρέχει στον προγραμματιστή τη δυνατότητα να αναλύει, να δημιουργεί και να χειρίζεται το bytecode της Java. Ως γνωστόν, ως bytecode εννοούνται τα μεταγλωττισμένα αρχεία της Java των οποίων τα ονόματα έχουν κατάληξη `.class`. Η βιβλιοθήκη παρέχει τη δυνατότητα δημιουργίας από την αρχή ενός `.class` αρχείου ή αλλαγής ενός υπάρχοντος, **χωρίς** να μεσολαβεί η διαδικασία της μεταγλώττισης (BCEL 2006).

Η βιβλιοθήκη είναι γραμμένη σε Java και είναι ανοικτό λογισμικό (διανέμεται με την άδεια Apache License 2.0). Αποτελεί ένα υποέργο του Apache Software Foundation και η ηλεκτρονική του διεύθυνση είναι http://jakarta.apache.org/bcel/index.html.

Η βιβλιοθήκη παρέχει αρκετές λειτουργίες στον προγραμματιστή. Από αυτές, στα πλαίσια της εργασίας, το ενδιαφέρον θα επικεντρωθεί στις δυνατότητες της βιβλιοθήκης για την εξαγωγή των πληροφοριών μιας κλάσης.

Εύρεση των πληροφοριών μιας κλάσης

Η κλάση `SyntheticRepository` του BCEL ορίζει μια *αποθήκη (repository)* στην οποία «φορτώνονται» κλάσεις από το σύστημα αρχείων. Η κλάση, κατά τη δημιουργία της, δέχεται την πλήρη διαδρομή ενός καταλόγου που περιέχει αρχεία `.class`. Στον παρακάτω κώδικα φαίνεται η δημιουργία του αντικειμένου `SyntheticRepository`. Η μέθοδος `getInstance()` λειτουργεί ως μέθοδος «εργοστάσιο» που δημιουργεί αντικείμενα τύπου `SyntheticRepository`.

```
SyntheticRepository repository = SyntheticRepository.getInstance(
                                    new ClassPath("/classfiles"));
```

Η παράμετρος που δέχεται η `getInstance()` είναι ένα αντικείμενο του τύπου `ClassPath`. Το αντικείμενο αυτό δέχεται ως παράμετρο στον δομητή την πλήρη διαδρομή ενός καταλόγου

που περιέχει αρχεία .class (στο παραπάνω παράδειγμα είναι ο κατάλογος "/classfiles"). Το αντικείμενο τύπου ClassPath χρησιμοποιείται για τη φόρτωση των αρχείων .class που βρίσκονται σε ένα δεδομένο κατάλογο αρχείων στο σύστημα.

Στη συνέχεια και μέσω του αντικειμένου του τύπου SyntheticRepository μπορεί ο προγραμματιστής να δημιουργήσει ένα αντικείμενο του τύπου JavaClass. Το αντικείμενο αυτό δημιουργείται καλώντας τη μέθοδο loadClass() της κλάσης SyntheticRepository με παράμετρο το όνομα της κλάσης από τον κατάλογο που δόθηκε αρχικά κατά τη δημιουργία του SyntheticRepository. Αυτό φαίνεται στον παρακάτω κώδικα.

```
JavaClass repClass = repository.loadClass("TypeInfo");
```

Στη συνέχεια, με το αντικείμενο του τύπου JavaClass ο προγραμματιστής μπορεί να λάβει όλες τις πληροφορίες για την κλάση. Για το σκοπό αυτό η κλάση JavaClass παρέχει μεθόδους για κάθε ιδιότητα της κλάσης. Ως ιδιότητα, εννοείται κάθε μορφής χαρακτηριστικό που έχει μια κλάση, όπως οι μεταβλητές που υπάρχουν δηλωμένες, οι μέθοδοι, η ορατότητα της κλάσης, κτλ. Ακολουθεί ενδεικτικός κώδικας.

```
// the package name of the class
String packageName = repClass.getPackageName();
// the name of the super class
String superClassName = repClass.getSuperclassName();
// true if the class is an interface
boolean isInterface = repClass.isInterface();
// true if the class is abstract
boolean isAbstract = repClass.isAbstract();
// the implemented interfaces of the class (names)
String[] implInterfaces = repClass.getInterfaceNames();
// the fields (attributes) of the class
Field[] fields = repClass.getFields();
// the methods of the class
Method[] methods = repClass.getMethods();
```

Το παραπάνω παράδειγμα κώδικα δείχνει τον τρόπο με τον οποίο παίρνει κάποιος τις πληροφορίες για την κλάση. Οι τύποι Field και Method είναι ειδικοί τύποι του BCEL που χρησιμοποιούνται από τον προγραμματιστή για τη λήψη πληροφοριών για τις ιδιότητες και μεθόδους της

κλάσης. Οι τύποι αυτοί παρέχουν μεθόδους στον προγραμματιστή για να βρίσκει για παράδειγμα το όνομα της ιδιότητας ή μεθόδου, την ορατότητα, κτλ. Ακολουθεί παράδειγμα κώδικα για τις ιδιότητες.

```
// for each field
for(Field _f : fields)
{
  // the name of the field
  String fieldName = _f.getName();
  // visibility
  if (_f.isPrivate())
    ...
  else if (_f.isPublic())
    ...
}
```

Ακολουθεί παράδειγμα για τις μεθόδους.

```
// for each method
for(Method _m : methods)
{
  // the name of the field
  String methodName = _m.getName();
  // visibility
  if (_m.isPrivate())
    ...
  else if (_m.isPublic())
    ...
  // the method's opcodes
  Code _code = _m.getCode();
}
```

Εκτός των υπολοίπων, η κλάση `Method` παρέχει τη μέθοδο `getCode()` (όπως φαίνεται στο παραπάνω παράδειγμα) που επιστρέφει τον κώδικα της υπό εξέτασης μεθόδου σε opcodes[1] για τη Java virtual machine.

3.3.2 Βιβλιοθήκη JGraph

Η βιβλιοθήκη JGraph είναι μια βιβλιοθήκη που παρέχει τη λειτουργικότητα για το ζωγράφισμα και την ανάλυση γράφων. Παρέχει ένα απλό application programming interface που επιτρέπει την οπτικοποίηση, αυτόματη διάταξη και ανάλυση των γράφων. Παραδείγματα εφαρμογών είναι διαγράμματα διεργασιών, ροής, δικτύου και βάσεων δεδομένων, κλάσεων UML, κά. (JGraph Ltd. 2006)

Ένας γράφος αποτελείται από τις *κορυφές (vertices)* ή κόμβους και τις *ακμές (edges)* που συνδέουν τους κόμβους μεταξύ τους. Στην ορολογία του JGraph και οι δύο αυτοί όροι περιγράφονται με τον ίδιο όρο *κελί (cell)*.

Η βιβλιοθήκη διανέμεται σε δύο εκδόσεις. Η μία είναι ανοικτού κώδικα και διανέμεται με την άδεια GNU Lesser General Public License v.2.1. Η δεύτερη ονομάζεται JGraphLayout, πωλείται για εμπορική χρήση της βιβλιοθήκης και παρέχει δυνατότητες αυτόματης διάταξης των κελιών στο γράφο και ανάλυσης των γράφων, δυνατότητες που δεν παρέχει η δωρεάν έκδοση. Η συγκεκριμένη έκδοση διανέμεται με άδεια που επιτρέπει την ακαδημαϊκή χρήση της βιβλιοθήκης. Η JGraphLayout «χτίζει» πάνω στην ανοικτού κώδικα JGraph και παρέχει κλάσεις με επιπλέον λειτουργικότητα από αυτήν της JGraph. Μετά από επικοινωνία του γράφοντος με την εταιρία JGraph Ltd., που είναι ο δημιουργός του JGraph, δόθηκε η άδεια για τη χρήση της βιβλιοθήκης στα πλαίσια της παρούσας εργασίας. Η ηλεκτρονική διεύθυνση για τη βιβλιοθήκη είναι http://www.jgraph.com.

Η βιβλιοθήκη ακολουθεί το πρότυπο σχεδίασης Model-View-Controller. Υπάρχουν δηλαδή ξεχωριστά αντικείμενα, με το καθένα να επιτελεί το δικό του σκοπό (όψη, μοντέλο και ελεγκτής)[2].

Στον πυρήνα της βιβλιοθήκης του JGraph υπάρχει η κλάση `org.jgraph.JGraph` η οποία

[1] Το *opcode* είναι συντομογραφία του Operation Code που είναι οι οδηγίες σε γλώσσα μηχανής ενός προγράμματος σε μια δεδομένη αρχιτεκτονική (στην περίπτωση αυτή για την εικονική μηχανή της Java).

[2] Για περισσότερες πληροφορίες σχετικά με το πρότυπο σχεδίασης Model-View-Controller ο αναγνώστης μπορεί να ανατρέξει στην ηλεκτρονική διεύθυνση http://en.wikipedia.org/wiki/Model-view-controller.

κληρονομεί την JComponent του πακέτου javax.swing της Java. Η εφαρμογή που αναπτύχθηκε στα πλαίσια της εργασίας παρέχει γραφική διεπαφή. Έτσι και η κλάση JGraph «δένει» με ένα αντικείμενο μιας υποκλάσης του JComponent, όπως είναι η JFrame.

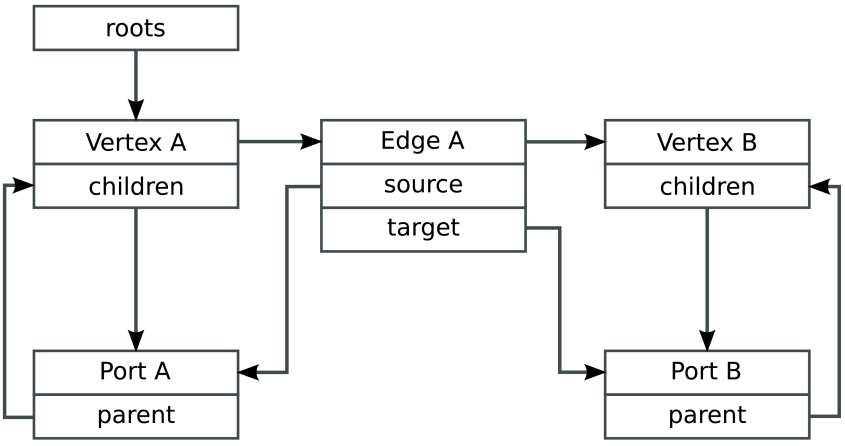
Για κάθε κορυφή αντιστοιχούν τρεις κλάσεις (λόγω του προτύπου σχεδίασης Model-View-Controller): η GraphCell, η CellView και η CellViewRenderer. Για την ακρίβεια, τα παραπάνω δεν είναι κλάσεις, αλλά διασυνδέσεις που πρέπει να υλοποιήσει ο προγραμματιστής για να δουλέψει με το JGraph. Οι εξ' ορισμού υλοποιήσεις των διασυνδέσεων αυτών στη βιβλιοθήκη του JGraph είναι η κλάση DefaultGraphCell, VertexView και VertexRenderer αντίστοιχα. Η VertexRenderer, που είναι η υπεύθυνη για το σχεδιασμό των κελιών του γράφου, κληρονομεί την κλάση javax.swing.JLabel της βιβλιοθήκης Swing της Java.

Για τις ακμές ισχύει αντίστοιχα ό,τι και για τις κορυφές. Παρέχονται οι κλάσεις DefaultEdge, EdgeView και EdgeRenderer.

Εκτός από τις κορυφές και τις ακμές, υπάρχει η έννοια της *θύρας (port)*. Οι θύρες είναι αντικείμενα που συσχετίζονται με τις κορυφές μέσω μιας συσχέτισης αντίστοιχα παιδιού – γονέα. Οι θύρες χρησιμοποιούνται από τις ακμές για να συνδέσουν δύο κορυφές μέσω της συσχέτισης πηγή – στόχος. Οι εξ' ορισμού κλάσεις στο JGraph που υλοποιούν τη λειτουργικότητα των θυρών είναι οι DefaultPort, PortView και PortRenderer. Στο σχήμα **3.3.1** φαίνεται πως τα αντικείμενα κορυφή, ακμή και θύρα «δένουν» μεταξύ τους. Οι κορυφές και ακμές βρίσκονται στο ίδιο επίπεδο, ενώ οι θύρες είναι παιδιά των κορυφών. Οι ακμές συνδέουν δύο κορυφές μέσω των ιδιοτήτων πηγή (source) και στόχος (target). Πάντα υπάρχει μια κορυφή που δείχνει στην πρώτη κορυφή, που δείχνει στην επόμενη ακμή ή κορυφή. Οι κορυφές και ακμές που βρίσκονται στο πρώτο επίπεδο ονομάζονται και *ρίζες (roots)*. Αυτό που υλοποιείται στην περίπτωση του σχήματος είναι ένας γράφος που δείχνει από το Vertex A στο Vertex B μέσω της ακμής Edge A.

Δημιουργία γράφου

Για τη δημιουργία νέου γράφου, χρειάζονται καταρχήν τα αντικείμενα τύπου GraphModel, GraphLayoutCache και JGraph. Ο παρακάτω κώδικας δείχνει τη δημιουργία των αντικειμένων. Οι δομητές των αντικειμένων τύπου GraphModel και JGraph χρειάζονται επιπλέον παραμέτρους. Συγκεκριμένα, για το GraphLayoutCache είναι απαραίτητο το αντικείμενο τύπου GraphModel και ένα επιπλέον αντικείμενο που λειτουργεί ως «εργοστάσιο» δημιουργίας αντικειμένων τύπου CellView.



Σχήμα **3.3.1** Η συσχέτιση μεταξύ κορυφών, ακμών και θυρών των αντικειμένων του JGraph

```
GraphModel model = new DefaultGraphModel();
GraphLayoutCache view = new GraphLayoutCache(model,
                                new DefaultCellViewFactory());
JGraph graph = new JGraph(model, view);
```

Η δημιουργία των κελιών γίνεται δημιουργώντας αντικείμενα τύπου `GraphCell`, ενώ για τις ακμές αντικείμενα τύπου `Edge`. Προκειμένου να μπορεί ένα κελί (ή κορυφή) να συνδέεται με κάποιο άλλο μέσω της ακμής (edge), θα πρέπει σε κάθε κελί να προστεθεί ένα αντικείμενο τύπου `Port`. Η διαδικασία αυτή φαίνεται στον παρακάτω κώδικα.

```
// create the first cell
GraphCell cell1 = new DefaultGraphCell();
Port port1 = new DefaultPort();
cell1.add(port1);

// create the second cell
GraphCell cell2 = new DefaultGraphCell();
Port port2 = new DefaultPort();
cell2.add(port2);

// create the edge that will connect the cells
Edge edge = new DefaultEdge();
edge.setSource(port1);  // set the source to the first cell
```

```
edge.setSource(port2);  // set the source to the second cell
```

Πρέπει να σημειωθεί ότι ο δομητής της κλάσης `DefaultGraphCell` μπορεί να δεχθεί ως παράμετρο ένα αντικείμενο τύπου `Object`. Στην περίπτωση αυτή, το αντικείμενο αυτό «δένεται» με το συγκεκριμένο κελί και μπορεί να είναι οτιδήποτε επιθυμεί ο προγραμματιστής. Μάλιστα, στην ορολογία του JGraph, το αντικείμενο αυτό ονομάζεται *αντικείμενο χρήστη (user object)*.

Αυτό που μένει τώρα είναι τα αντικείμενα αυτά να εισαχθούν στο γράφημα. Πριν γίνει αυτό όμως, μπορεί ο προγραμματιστής να θέσει σε αυτό το σημείο κάποιες παραμέτρους που αφορούν σε διάφορες ρυθμίσεις των κελιών, όπως το χρώμα, το μέγεθος του κελιού, τις διαστάσεις του, κά. Αυτό μπορεί να γίνει μέσω της κλάσης `GraphConstants`, που διαθέτει ένα πλήθος στατικών μεθόδων για τον ορισμό τέτοιων παραμέτρων. Οι ιδιότητες των αντικειμένων του JGraph αποθηκεύονται σε δομές τύπου map. Για παράδειγμα, για να αλλάξει ο προγραμματιστής την ιδιότητα του `AUTOSIZE` του κελιού σε `true`, πρέπει να δώσει

```
// set the AUTOSIZE property of a cell to true
GraphConstants.setAutoSize(cell.getAttributes(), true);
```

Με τον τρόπο αυτό (χρησιμοποιώντας την κλάση `GraphConstants`) ορίζονται οι ιδιότητες των αντικειμένων του JGraph.

Η εισαγωγή των κελιών στο γράφημα γίνεται με κλήση της μεθόδου `insert()` του αντικειμένου της όψης του JGraph, στην περίπτωση του παραδείγματος, του `view`. Η μέθοδος καλείται με παραμέτρους αντικείμενα τύπου `Object` που είναι είτε τα κελιά, είτε οι ακμές. Οι θύρες δεν χρειάζεται να εισαχθούν ρητά, γιατί ως παιδιά των κελιών, εισάγονται αυτόματα. Έτσι, ο κώδικας για την εισαγωγή των κελιών και της ακμής του παραδείγματος είναι

```
// insert the JGraph objects to the view object
view.insert(new Object[cell1, cell2]);
view.insert(edge);
```

Όπως αναφέρθηκε παραπάνω, η κλάση `JGraph` κληρονομεί την κλάση `JComponent` του Swing API της Java. Συνήθως στις εφαρμογές που χρησιμοποιούν τη βιβλιοθήκη JGraph, χρησιμοποιείται ένα αντικείμενο τύπου `JScrollPane` με παράμετρο το αντικείμενο του `JGraph`. Αυτό γίνεται ώστε σε περίπτωση που τα κελιά με τις ακμές δεν χωράνε στο παράθυρο της εφαρμογής, ο χρήστης να μπορεί να μετακινηθεί μέσω των μπαρών κύλισης. Τέλος, το αντικείμενο τύπου `JComponent` προστίθεται στο κεντρικό *πλαίσιο (frame)* του παραθύρου της εφαρμογής.

```
// attach the JGraph object to a JScrollPane object
JScrollPane scrollPane = new JScrollPane(graph);
  .
  .
  .
// add to the main application frame
frame.getContentPane().add(scrollPane);
```

Σε γενικές γραμμές, αυτά είναι τα (ελάχιστα) βήματα που πρέπει να ακολουθήσει ο προγραμματιστής για να προσθέσει ένα απλό γράφο στην εφαρμογή του.

Δημιουργία εξειδικευμένου γράφου

Υπάρχουν περιπτώσεις που είναι επιθυμητή η δημιουργία εξειδικευμένων κελιών ή ακμών σε ένα γράφο. Μια τέτοια περίπτωση είναι η εφαρμογή της παρούσας εργασίας. Στην εφαρμογή χρειάστηκε η δημιουργία κελιών που τα ορθογώνιά τους έπρεπε να περιέχουν δύο επιπλέον οριζόντιες γραμμές για το διαχωρισμό των ιδιοτήτων και λειτουργιών.

Για τη δημιουργία εξειδικευμένων κελιών απαιτείται η δημιουργία νέας κλάσης που κληρονομεί την `DefaultGraphCell`. Για την εφαρμογή, δημιουργήθηκε η κλάση `ClassGraphCell` και αποτελεί το αντικείμενο χρήστη. Στη συνέχεια, δημιουργήθηκε μια νέα υποκλάση της `VertexView` με όνομα `ClassView`. Η κλάση `ClassView` είναι η κλάση της όψης, άρα και η υπεύθυνη για τη δημιουργία των γραφικών αντικειμένων του γράφου. Παράλληλα, η ίδια κλάση περιέχει μια *εσωτερική (inner)* κλάση με όνομα `ClassRenderer`. Η κλάση αυτή είναι που κάνει το εξειδικευμένο ζωγράφισμα με *υπέρβαση (override)* της μεθόδου `paint(Graphics g)`. Τέλος, δημιουργήθηκε μια κλάση με όνομα `ClassViewFactory`, υποκλάση της `DefaultCellViewFactory` που επιστρέφει στιγμιότυπα της εξειδικευμένης κλάσης `ClassView`.

Λόγω του ότι δημιουργείται μια νέα κλάση ως «εργοστάσιο», θα πρέπει να δηλωθεί με κάποιο τρόπο η κλάση αυτή, ώστε να επιστρέφει τα σωστά αντικείμενα του τύπου `ClassGraphCell`. Αυτό γίνεται κατά τη δημιουργία του αντικειμένου της όψης του τύπου `GraphLayoutCache`, όπου περνιέται ως δεύτερη παράμετρος η νέα κλάση για τη δημιουργία των αντικειμένων των κελιών.

```
// register the new cell factory with the view object
GraphLayoutCache view = new GraphLayoutCache(model, new ClassViewFactory());
```

Μηχανισμοί αυτόματης διάταξης του JGraphLayout

Όπως αναφέρθηκε προηγουμένως (σελίδα 51), για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκε η βιβλιοθήκη JGraphLayout, η οποία διανέμεται κάτω από εμπορική άδεια, αλλά μπορεί να χρησιμοποιηθεί και για ακαδημαϊκούς σκοπούς. Η επιπλέον δυνατότητα, που χρησιμοποιήθηκε στην εφαρμογή, σε σχέση με το δωρεάν JGraph είναι η αυτόματη διάταξη των κόμβων στο γράφο.

Για το μηχανισμό της αυτόματης διάταξης σε ένα γράφο, υπάρχει η διασύνδεση `JGraphLayout` και η κλάση `JGraphFacade`. Οι κλάσεις που υλοποιούν τη διασύνδεση εκτελούν τις μαθηματικές λειτουργίες για τη δημιουργία της διάταξης, ενώ η κλάση ενεργεί πάνω στο γράφο και παρέχει διάφορες μεθόδους για την εξαγωγή πληροφοριών σχετικά με αυτόν. Με τον τρόπο αυτό, διαχωρίζεται η λειτουργικότητα της εμφάνισης από τον ίδιο τον αλγόριθμο της διάταξης.

Για να χρησιμοποιηθεί ένας αλγόριθμος αυτόματης διάταξης σε ένα γράφο, πρέπει να δημιουργηθεί ένα αντικείμενο του τύπου `JGraphFacade` που κρατάει τις πληροφορίες και τις ρυθμίσεις του γράφου. Ο δομητής για το αντικείμενο αυτό, χρειάζεται ένα αντικείμενο του τύπου `JGraph` ώστε να γνωρίζει σε ποιο γράφο αναφέρεται. Στη συνέχεια, απαιτείται η δημιουργία ενός αντικειμένου του τύπου `JGraphLayout` καλώντας έναν από τους δομητές των κλάσεων που υλοποιούν αυτή τη διασύνδεση. Τέτοιες κλάσεις είναι ουσιαστικά κλάσεις που υλοποιούν διαφορετικούς αλγόριθμους διάταξης στο γράφο. Έτσι, υπάρχουν κλάσεις για τη διάταξη σε σχήμα δέντρου (tree layout) διαφόρων μορφών, κλάσεις για τη διάταξη σε σχήμα που θυμίζει οργανική δομή (organic layout), ενώ η εφαρμογή χρησιμοποιεί μια διάταξη που ονομάζεται ιεραρχική διάταξη (hierarchical layout). Αν ο προγραμματιστής επιθυμεί μια διάταξη σε σχήμα δέντρου για παράδειγμα, πρέπει να καλέσει το δομητή μιας από τις κλάσεις `JGraphCompactTreeLayout` ή `JGraphTreeLayout`. Στη συνέχεια, καλείται η μέθοδος `run()` της `JGraphLayout` με παράμετρο το αντικείμενο του τύπου `JGraphFacade` που δημιουργήθηκε προηγουμένως. Η μέθοδος εκτελεί τον αλγόριθμο της διάταξης, αλλά για να εφαρμοστεί και στο γράφημα, απαιτούνται δύο επιπλέον ενέργειες. Όλα τα παραπάνω, συνοψίζονται στο παρακάτω παράδειγμα κώδικα.

```
// obtain the root user objects
Object[] roots = getRootObjects();

// create the JGraphFacade and JGraphLayout objects
JGraphFacade facade = new JGraphFacade(graph, roots);
```

```
JGraphLayout layout = new JGraphTreeLayout();  // use a tree layout

// run the layout algorithm
layout.run(facade);

// obtain a map of the resulting attribute changes from the facade
Map nested = facade.createNestedMap(true, true);

// apply the results to the actual graph
graph.getGraphLayoutCache().edit(nested);
```

Στο παράδειγμα αυτό, ο αναγνώστης θα παρατηρήσει ότι υπάρχει ένας πίνακας από αντικείμενα `Object` που είναι οι ρίζες από τις οποίες ξεκινάει το δέντρο. Τα αντικείμενα αυτά είναι απαραίτητα όταν χρησιμοποιείται ο αλγόριθμος της διάταξης δέντρων και σε αυτά αποθηκεύονται τα αντικείμενα χρήστη (user objects) και όχι αντικείμενα των κελιών (του τύπου `GraphCell`).

Με τα παραπάνω βήματα, γίνεται η αυτόματη διάταξη των κελιών του γράφου, χωρίς κόπο από την πλευρά του προγραμματιστή.

3.4 Τεχνικές εύρεσης των σχέσεων μεταξύ των κλάσεων

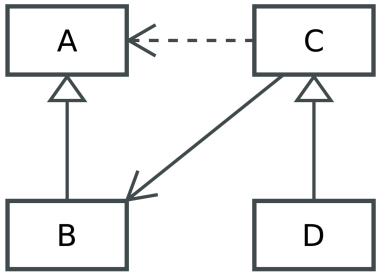
Έστω n ο αριθμός των αρχείων `.class` για τα οποία πρέπει να δημιουργηθεί το διάγραμμα κλάσεων. Η εφαρμογή χρησιμοποιεί ένα διδιάστατο πίνακα μήκους $n * n$ για να απεικονίσει τις σχέσεις μεταξύ των κλάσεων. Σε κάθε κελί τοποθετείται ένας χαρακτήρας που υποδηλώνει τη σχέση της κλάσης από την αντίστοιχη γραμμή προς την κλάση στην αντίστοιχη στήλη του κελιού. Για παράδειγμα, έστω οι κλάσεις A, B, C και D. Ένας υποθετικός πίνακας σχέσεων είναι ο **3.1**. Το διάγραμμα κλάσεων που αντιστοιχεί στον πίνακα απεικονίζεται στο σχήμα **3.4.2**.

Οι χαρακτήρες που χρησιμοποιούνται στην απεικόνιση των συσχετίσεων μεταξύ των κλάσεων είναι:

- **S** Για την κληρονομικότητα.
- **I** Για την υλοποίηση/πραγματοποίηση διασύνδεσης
- **D** Για την εξάρτηση

Πίνακας **3.1** Πίνακας σχέσεων μεταξύ των κλάσεων A, B, C και D

	A	B	C	D
A				
B	S			
C	D	U		
D			S	



Σχήμα **3.4.2** Το διάγραμμα κλάσεων που αντιστοιχεί στον πίνακα συσχετίσεων **3.1**

- **U** Για την συσχέτιση
- **H** Για τη συνάθροιση ή σύνθεση

Πρέπει να σημειωθεί ότι στη Java οι έννοιες της συνάθροισης ή σύνθεσης δεν υφίστανται, λόγω της αρχιτεκτονικής της γλώσσας.

Ο παραπάνω πίνακας δημιουργείται κατά την εύρεση των συσχετίσεων μεταξύ των κλάσεων που υπάρχουν στον κατάλογο που επέλεξε ο χρήστης. Για την ακρίβεια, στα κελιά του πίνακα εισάγονται συμβολοσειρές του τύπου

<συσχέτιση>:<πολλαπλότητα>

όπου συσχέτιση είναι ένας από τους χαρακτήρες που περιγράφονται παραπάνω και πολλαπλότητα είναι η πολλαπλότητα για την αντίστοιχη σχέση, στη μορφή $x:y$, όπου τα x και y μπορούν να είναι ένα από τα 0, 1 ή $*$. Για παράδειγμα, αν μια μέθοδος μιας κλάσης A χρησιμοποιούσε ένα πίνακα από μια κλάση B σε μια παράμετρό της, τότε η συσχέτιση στον πίνακα θα ήταν `D:1:*`.

Από τα παραπάνω, γίνεται αντιληπτό ότι έχει σημασία ο τρόπος με τον οποίο διαβάζεται ο πίνακας. Δηλαδή, για τον πίνακα **3.1** ισχύει:

`B` → `A : S` (Η B κληρονομεί την A)

`C` → `A : D` (Η C εξαρτάται από την A)

`C` → `B` : `U` (Η C συσχετίζεται με την B)

`D` → `C` : `S` (Η D κληρονομεί την C)

Μια κλάση που έχει ως ιδιότητα – μέλος ή μια από τις μεθόδους της έχει ως παράμετρο ένα στιγμιότυπο μιας άλλης κλάσης θεωρείται ότι συσχετίζεται (**U**) με τη δεύτερη. Αν στο σώμα μιας μεθόδου της (όχι στις παραμέτρους) δημιουργεί και χρησιμοποιεί αντικείμενο μιας άλλης κλάσης, τότε (η πρώτη κλάση) θεωρείται ότι εξαρτάται (**D**) από τη δεύτερη.

3.5 Αρχιτεκτονική εφαρμογής

Ο πηγαίος κώδικας της εφαρμογής οργανώνεται σε πακέτα. Ακολουθεί η συνοπτική περιγραφή των πακέτων:

`gr.uom.umldiagdraw` Το πρώτο πακέτο περιέχει την κλάση εκκίνησης `MainApp` της εφαρμογής.

`gr.uom.umldiagdraw.exceptions` Περιέχει τις ορισμένες από τον προγραμματιστή *εξαιρέσεις (exceptions)* της εφαρμογής. Το πακέτο περιέχει τις κλάσεις `AbstractException`, `ClassFilesNotFound` και `PathNotFoundException`, από τις οποίες οι δύο τελευταίες κληρονομούν την πρώτη.

`gr.uom.umldiagdraw.factory` Περιέχει την κλάση `Factory` που παρέχει στατικές μεθόδους για την επιστροφή στιγμιοτύπων αντικειμένων κλάσεων που χρησιμοποιεί η εφαρμογή. Λειτουργεί ως κλάση «εργοστάσιο».

`gr.uom.umldiagdraw.graph` Περιέχει τις κλάσεις `ClassGraphCell`, `ClassViewFactory` και `ClassView`. Είναι οι εξειδικευμένες κλάσεις του JGraph που χρησιμοποιούνται για τη σχεδίαση των γραφικών του γράφου που απεικονίζουν τις κλάσεις του διαγράμματος κλάσεων.

`gr.uom.umldiagdraw.graphics` Περιέχει τις κλάσεις `MainFrame` και `AboutFrame`. Από αυτές, η πρώτη αποτελεί την κλάση που υλοποιεί την γραφική διεπαφή της εφαρμογής και αποτελεί την κλάση ελέγχου των υπολοίπων, με αποτέλεσμα να είναι η μεγαλύτερη (σε γραμμές κώδικα) κλάση όλης της εφαρμογής. Η δεύτερη υλοποιεί ένα απλό παράθυρο διαλόγου που εμφανίζει σύντομες πληροφορίες για την εφαρμογή.

`gr.uom.umldiagdraw.logging` Περιέχει την κλάση `MyLogger` που λειτουργεί ως κλάση «εργοστάσιο» για την επιστροφή αντικειμένου του τύπου `org.apache.log4j.Logger`. Το αντικείμενο αυτό χρησιμοποιείται για την εμφάνιση χρήσιμων μηνυμάτων ενημερωτικού τύπου ή μηνυμάτων που χρησιμεύουν κατά την *αποσφαλμάτωση (debugging)* της εφαρμογής. Η κλάση χρησιμοποιεί τη βιβλιοθήκη **Log4j**, ένα από τα υποέργα του Apache Software Foundation[3].

`gr.uom.umldiagdraw.readers` Περιέχει τις κλάσεις `TypeInfoReader` και `TypeInfoReaderBCEL`. Η πρώτη είναι αφηρημένη κλάση που κληρονομεί η δεύτερη και υλοποιεί τις μεθόδους για την ανάγνωση των πληροφοριών των κλάσεων χρησιμοποιώντας τη βιβλιοθήκη BCEL.

`gr.uom.umldiagdraw.readers.utils` Περιέχει την κλάση `DirectoryFinder` που παρέχει μεθόδους για την εύρεση των υποκαταλόγων ενός καταλόγου και χρησιμοποιείται ως «βοηθητική» κλάση από την `TypeInfoReaderBCEL`.

`gr.uom.umldiagdraw.relations` Περιέχει τις κλάσεις `RelationMatch` και `RelationMatchForJava`. Η πρώτη είναι αφηρημένη κλάση που κληρονομεί η δεύτερη και υλοποιεί τις μεθόδους για την εύρεση των σχέσεων των κλάσεων για τη Java. Η κλάση αυτή συμπληρώνει το διδιάστατο πίνακα, όπως περιγράφεται στο §3.4 στη σελίδα 57.

`gr.uom.umldiagdraw.types` Περιέχει τους ορισμένους από τον προγραμματιστή τύπους δεδομένων που χρησιμοποιεί η εφαρμογή. Το πακέτο αυτό παρέχει τις παρακάτω κλάσεις:

`AccessEnum` Είναι μια απαρίθμηση για τα είδη ορατότητας (ιδιωτική, προστατευμένη, πακέτου και δημόσια).

`Attribute` Είναι μια κλάση που περιέχει τις μετα-πληροφορίες για μια ιδιότητα κλάσης.

`MethodType` Είναι μια κλάση που περιέχει τις μετα-πληροφορίες για μια λειτουργία κλάσης.

`ModifierEnum` Είναι μια απαρίθμηση για τον τρόπο αποθήκευσης ενός τύπου [τελικός (final), στατικός (static) ή αφηρημένος (abstract)].

`ParamType` Είναι μια κλάση που περιέχει τις μετα-πληροφορίες για μια παράμετρο λειτουργίας μιας κλάσης.

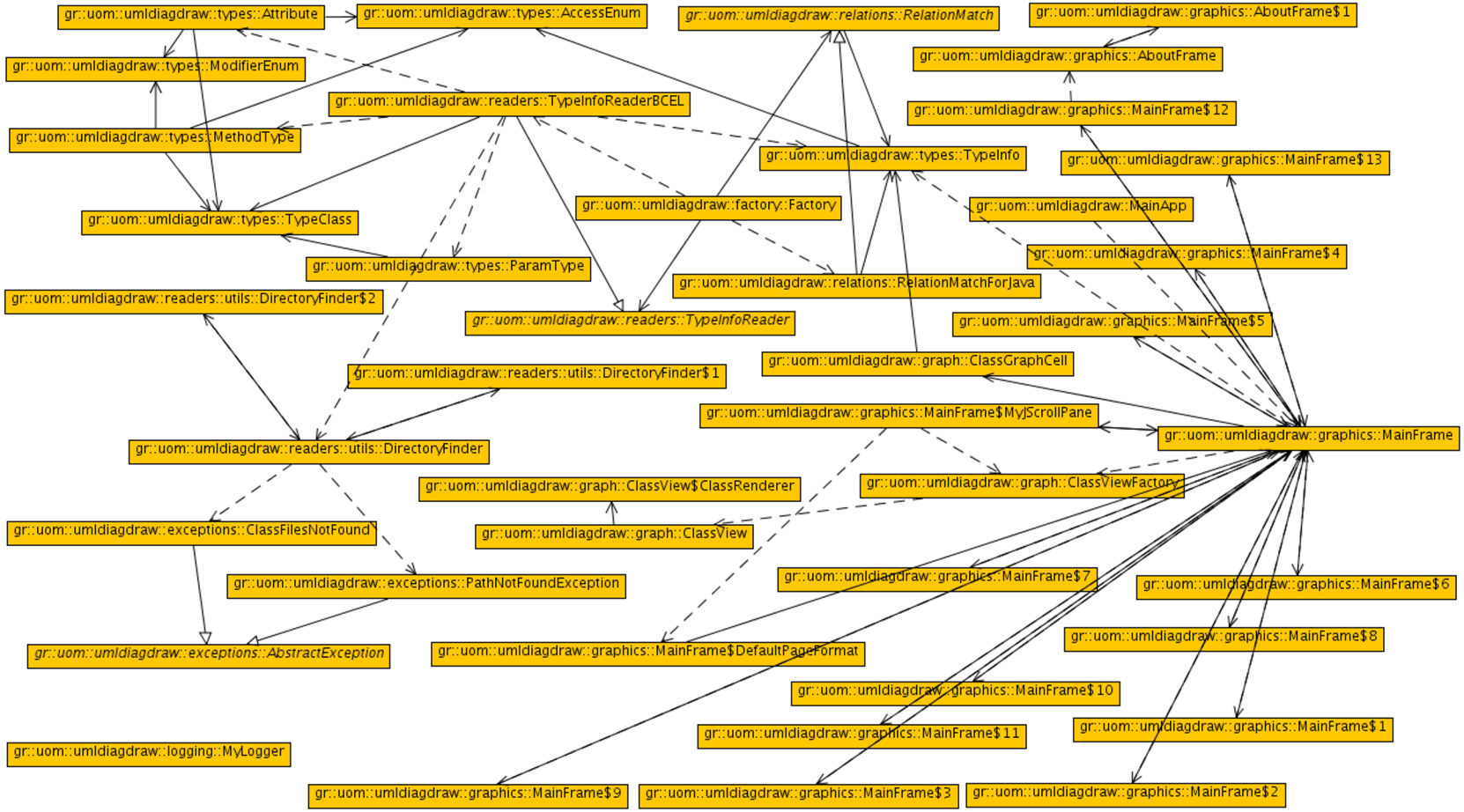
[3]Για περισσότερες πληροφορίες σχετικά με τη βιβλιοθήκη Log4j ο αναγνώστης μπορεί να ανατρέξει στην ηλεκτρονική διεύθυνση http://logging.apache.org/

`TypeClass` Είναι μια κλάση που περιγράφει τον τύπο μιας ιδιότητας, παραμέτρου μεθόδου ή τον τύπο επιστροφής έτσι όπως χρησιμοποιείται εσωτερικά από τις υπόλοιπες κλάσεις του πακέτου. Για παράδειγμα, ένας τύπος ακεραίου συμβολίζεται ως `int`, ενώ ένα αντικείμενο μιας κλάσης `java.util.Map` ως '`class:java.util.Map`'.

`TypeInfo` Είναι μια κλάση που περιέχει τις μετα-πληροφορίες για μια κλάση. Είναι η κύρια κλάση που χρησιμοποιούν τα υπόλοιπα πακέτα για την αποθήκευση των πληροφοριών για ένα τύπο. Η κλάση `TypeInfo` συσχετίζεται με τις υπόλοιπες κλάσεις του πακέτου αυτού με συσχετίσεις ένα προς πολλά.

Η εφαρμογή απαιτεί τουλάχιστον Java 5 για την εκτέλεσή της, διότι χρησιμοποιεί αρκετά από τα νέα χαρακτηριστικά της έκδοσης 5 της γλώσσας, όπως generics, απαριθμήσεις, νέοι βρόχοι `for`, κτλ.

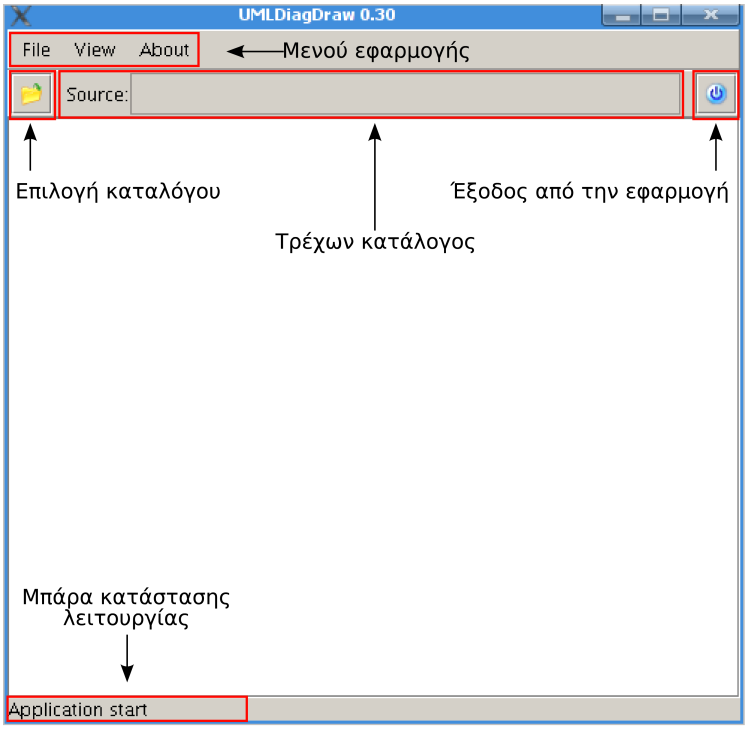
Στο σχήμα **3.5.3** απεικονίζεται το διάγραμμα κλάσεων της εφαρμογής, όπως αυτό δημιουργείται από την ίδια την εφαρμογή. Όπως γίνεται αντιληπτό, η κλάση `MainFrame` είναι αυτή που έχει τις περισσότερες συσχετίσεις με τις υπόλοιπες κλάσεις, λόγω του γεγονότος ότι είναι η κλάση που διαχειρίζεται τη γραφική διεπαφή της εφαρμογής.



Σχήμα **3.5.3** Το διάγραμμα κλάσεων της εφαρμογής UMLDiagDraw όπως παράγεται από την ίδια

3.6 Σύντομος οδηγός χρήσης

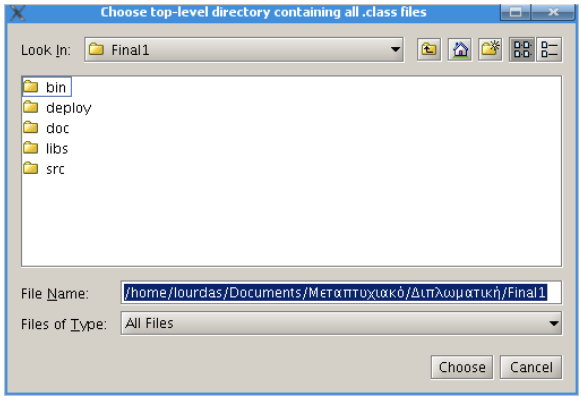
Η κεντρική οθόνη της εφαρμογής απεικονίζεται στο σχήμα **3.6.4**. Υπάρχει η μπάρα μενού με τρία μενού, η μπάλα εργαλείων με δύο κουμπιά, το ένα για την επιλογή καταλόγου προέλευσης και το άλλο για την έξοδο από την εφαρμογή, και παράλληλα υπάρχει και μια ετικέτα που εμφανίζει το πλήρες όνομα του καταλόγου που επέλεξε ο χρήστης.



Σχήμα **3.6.4** Η κεντρική οθόνη της εφαρμογής UMLDiagDraw

Πατώντας στο μενού File → Browse directory... ή στο πρώτο εικονίδιο με το άνοιγμα φακέλου στην μπάρα εργαλείων εμφανίζεται το παράθυρο επιλογής καταλόγου (σχήμα **3.6.5**). Αφού ο χρήστης επιλέξει ένα κατάλογο που περιέχει το bytecode των κλάσεων της εφαρμογής του, η εφαρμογή διαβάζει τα αρχεία, εξάγει τις πληροφορίες που χρειάζεται μέσω του BCEL και τέλος δημιουργεί το γράφο, εμφανίζοντας με τον τρόπο αυτό το διάγραμμα κλάσεων για τις συγκεκριμένες κλάσεις. Το σχήμα **3.6.6** εμφανίζει το διάγραμμα κλάσεων για ένα υποθετικό σύστημα.

Σε μεγάλα διαγράμματα, όπου οι κλάσεις είναι αρκετές και είναι δύσκολο ο χρήστης να έχει την «εικόνα» του συστήματος, η εφαρμογή δίνει τη δυνατότητα της σμίκρυνσης (zoom out) του γράφου σε τέσσερα επίπεδα (25%, 50%, 75% και 100%) του αρχικού μεγέθους (το σχήμα **3.6.7**

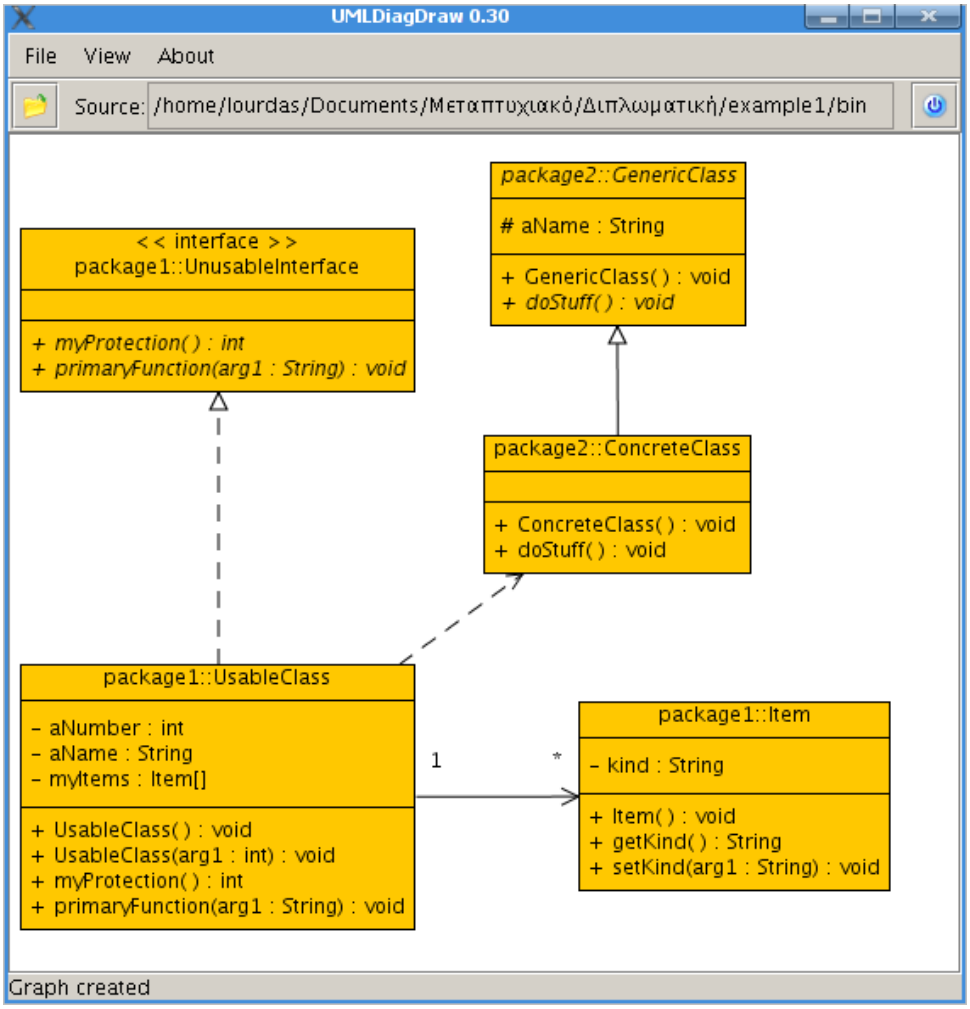


Σχήμα **3.6.5** Το παράθυρο επιλογής καταλόγου

δείχνει τον γράφο του σχήματος σε 50% σμίκρυνση). Η επιλογή του επιπέδου της σμίκρυνσης γίνεται από το μενού View με επιλογή των αντίστοιχων επιλογών (25% zoom ... 100% zoom).

Παράλληλα, υπάρχει η δυνατότητα απόκρυψης των ιδιοτήτων και λειτουργιών των κλάσεων, λειτουργικότητα που υπάρχει σε αρκετές παρόμοιες εφαρμογές αντίστροφης μηχανικής. Οι λειτουργίες αυτές υπάρχουν στο μενού View ως επιλογές που είναι είτε ενεργοποιημένες, είτε όχι. Τα ονόματά τους είναι αντίστοιχα "Show class attributes" και "Show class operations". Εξ' ορισμού, οι επιλογές αυτές είναι ενεργοποιημένες. Αν ο χρήστης έχει ήδη επιλέξει ένα κατάλογο αρχείων και έχει ήδη σχηματιστεί το διάγραμμα κλάσεων της εφαρμογής, τότε υπάρχει η δυνατότητα απενεργοποίησης των επιλογών αυτών εκ των υστέρων. Απενεργοποιώντας λοιπόν τη μία (ή και τις δύο) επιλογές αυτές, ο γράφος ξανασχηματίζεται χωρίς να δείχνει τις ιδιότητες (ή/και λειτουργίες) των κλάσεων. Το σχήμα **3.6.8** δείχνει το διάγραμμα του υποθετικού συστήματος χωρίς τις ιδιότητες και λειτουργίες των κλάσεων.

Κάποιες επιπλέον δυνατότητες της εφαρμογής είναι η εκτύπωση του γραφήματος μέσω της αντίστοιχης επιλογής τους μενού (File → Print...) και της αποθήκευσής του σε αρχείο εικόνας τύπου PNG (Portable Network Graphics) (μέσω της επιλογής File → Save as image...). Για τη λειτουργία αυτή, η εφαρμογή ζητά από το χρήστη μέσω του κατάλληλου παραθύρου διαλόγου το όνομα του αρχείου. Αν η αποθήκευση του αρχείου `.png` γίνει χωρίς πρόβλημα, ο χρήστης ενημερώνεται με το κατάλληλο μήνυμα (σχήμα **3.6.9**). Τέλος, μια πρόσθετη δυνατότητα της εφαρμογής είναι η αλλαγή του χρώματος των κλάσεων στο διάγραμμα. Αυτό γίνεται από το μενού View → Choose class color... που εμφανίζει ένα παράθυρο επιλογής της χρωματικής απόχρωσης.



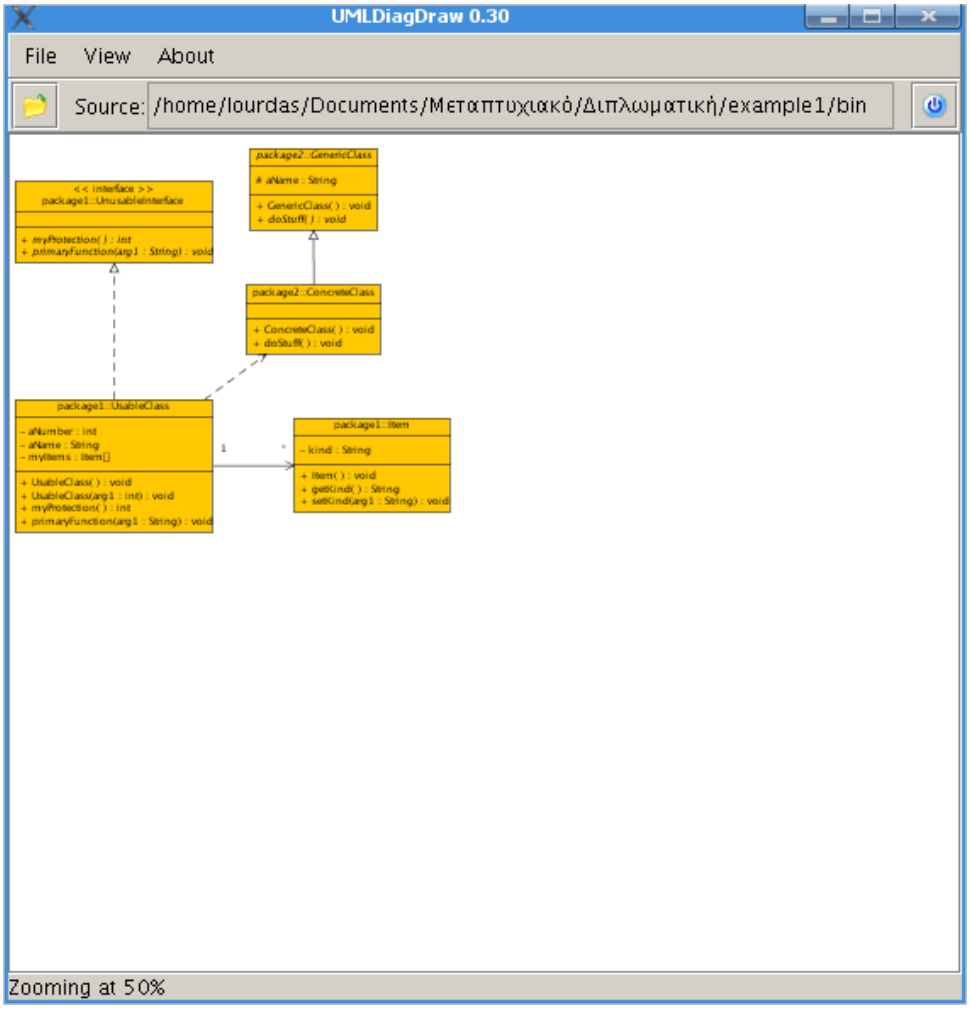
Σχήμα **3.6.6** Το διάγραμμα κλάσεων για ένα υποθετικό σύστημα

3.7 Προβλήματα – προτάσεις για βελτίωση – συμπεράσματα

3.7.1 Προβλήματα

Στη Java 5 μια σημαντική προσθήκη αφορά στον χειρισμό των τύπων συλλογών και λιστών. Συγκεκριμένα, μέσω των *generics* είναι δυνατόν να ξέρει ο μεταγλωττιστής κατά τη φάση της μεταγλώττισης τι είδους αντικείμενα θα εισαχθούν σε μια συλλογή ή λίστα. Τα generics έχουν παρόμοια χρήση με τα *πρότυπα (templates)* της C++. Για παράδειγμα, αντί του κλασικού

```
Collection myCollection = new ArrayList();
myCollection.add("computer");
```

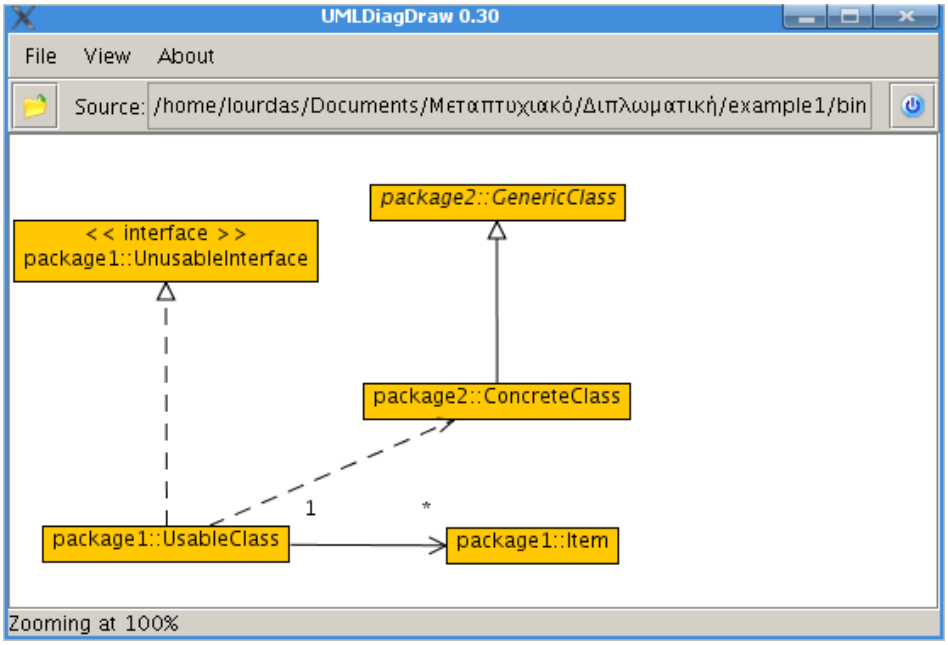


Σχήμα **3.6.7** Το διάγραμμα κλάσεων του σχήματος **3.6.6** σε σμίκρυνση 50%

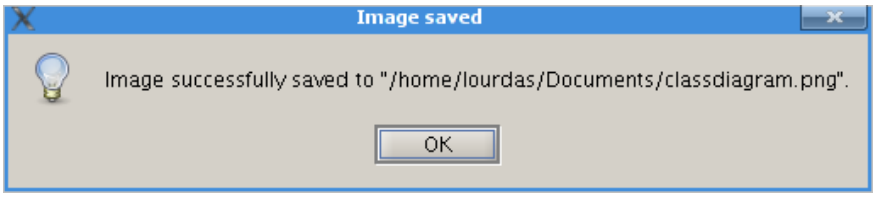
```
myCollection.add("keyboard");
   .
   .
   .
Iterator i = myCollection.iterator();
String word = (String)i.next();
```

με τα generics η λύση είναι αρκετά κομψότερη και ασφαλέστερη, εφόσον δεν γίνονται περιττές (και κάποιες φορές επικίνδυνες) *μετατροπές τύπων (casts)*.

```
Collection<String> myCollection = new ArrayList()<String>;
myCollection.add("computer");
```



Σχήμα **3.6.8**: Το διάγραμμα κλάσεων του σχήματος **3.6.6** χωρίς την απεικόνιση των ιδιοτήτων και λειτουργιών των κλάσεων



Σχήμα **3.6.9**: Η εφαρμογή ενημερώνει το χρήστη για την επιτυχή αποθήκευση του διαγράμματος σε εικόνα τύπου PNG

```
myCollection.add("keyboard");
  .
  .
  .
Iterator i = myCollection.iterator();
String word = i.next();
```

Ωστόσο, αυτό είναι χαρακτηριστικό του μεταγλωττιστή και δεν υπάρχει διαφορά στο παραγόμενο bytecode. Έτσι, ενώ σε αντικείμενα τύπου συλλογών, λιστών, map, κλπ. θα μπορούσε η εφαρμογή να ανακαλύπτει το είδος των σχέσεων τέτοιων αντικειμένων με τον περιεχόμενο τύπο, αυτό τελικά δεν επιτεύχθηκε. Σχέσεις δηλαδή μεταξύ κλάσεων μέσω αντικειμένων τύπου

Collection, Map, κτλ. με τις κλάσεις των αντικειμένων που αποθηκεύουν δεν κατέστη δυνατό να βρεθούν. Στον κώδικα της εφαρμογής, σε πολλά σημεία χρησιμοποιείται μια δομή τύπου Map<String, TypeInfo> που είναι αντιστοίχιση ονόματος τύπου με τον τύπο που κρατάει τις μετα-πληροφορίες για τον τύπο αυτό. Για παράδειγμα, η κλάση MainFrame έχει ιδιότητα του τύπου αυτού, ωστόσο η εφαρμογή αδυνατεί να βρει τη συσχέτιση μεταξύ των δύο, ακριβώς για το λόγο ότι δεν μπορεί να ανακαλύψει το είδος των αντικειμένων που αποθηκεύει η δομή Map.

Ένα πρόβλημα που προέκυψε κατά την ανάπτυξη ήταν το πως θα εξαχθούν πληροφορίες για το σώμα μιας μεθόδου. Το BCEL παρέχει τη μέθοδο getCode() της κλάσης Method που επιστρέφει τα opcodes του σώματος μιας μεθόδου. Ωστόσο, έπρεπε να βρεθούν τα συγκεκριμένα opcodes που χρησιμοποιούνται όταν δημιουργούνται νέα αντικείμενα με τον τελεστή new. Αυτό ισχύει είτε για απλά αντικείμενα, είτε για πολυδιάστατους πίνακες. Για παράδειγμα, έστω ο κώδικας

```
int[] numbers;
String[] strings;
String[][] multistrings;
String string;

numbers = new int[10];
strings = new String[5];
string = new String("test");
multistrings = new String[10][];
```

Ο κώδικας αυτός μεταφράζεται σε opcodes ως εξής:

```
0:    bipush    10
2:    newarray    <int>
4:    pop
5:    iconst_5
6:    anewarray   <java.lang.String> (15)
9:    pop
10:   new   <java.lang.String> (15)
13:   dup
14:   ldc   "test" (17)
```

```
16:   invokespecial java.lang.String.<init> (Ljava/lang/String;)V (19)
19:   pop
20:   bipush    10
22:   multianewarray  <[[Ljava.lang.String;>  1 (22)
26:   pop
27:   return
```

Δηλαδή, το `numbers = new int[10];` μεταφράζεται[4] ως

```
bipush 10
newarray <int>
```

Το `strings = new String[5];` ως

```
iconst_5 anewarray <java.lang.String>
```

Το `string = new String("test");` μεταφράζεται ως

```
pop
new <java.lang.String>
dup
ldc "test"
invokespecial java.lang.String.<init> (Ljava/lang/String;)V
```

Και το `multistrings = new String[10][];` ως

```
bipush 10
multianewarray <[[Ljava.lang.String;>
```

Από τα παραπάνω, γίνεται αντιληπτό ότι έπρεπε να βρεθούν οι γραμμές που περιείχαν τα opcodes: `new`, `newarray`, `anewarray` και `multianewarray`. Επίσης το πρόβλημα υπήρξε και στην ανάλυση του τύπου δεδομένων κατά τη διαδικασία δημιουργίας του νέου αντικειμένου. Σε απλούς τύπους και μονοδιάστατους πίνακες δηλώνεται μέσα σε <> (για κλάσεις δηλώνεται το πλήρες όνομα της κλάσης όπως `java.lang.String` στο παράδειγμα παραπάνω), ενώ σε πολυδιάστατους πίνακες δηλώνεται και πάλι μέσα σε <>, αλλά μετά το πρώτο < υπάρχουν τόσες αγκύλες '[' όσες οι διαστάσεις του πίνακα και ακολουθεί το όνομα του τύπου[5].

[4] Η αντιστοιχία των εντολών σε Java στα αντίστοιχα opcodes στο παράδειγμα αυτό ενδέχεται να μην είναι 100% ακριβής.

[5] Για περισσότερες πληροφορίες ο αναγνώστης μπορεί να απευθυνθεί στο Java Virtual Machine Specification.

Μια παρατήρηση είναι ότι στο bytecode δεν κρατώνται τα ονόματα των μεταβλητών των παραμέτρων μιας μεθόδου. Για παράδειγμα, για μια μέθοδο με την υπογραφή `void add(String string1, String string2);`, το BCEL θα επιστρέψει ως ονόματα των παραμέτρων τα `arg1` και `arg2`. Το φαινόμενο αυτό δεν συμβαίνει όταν ο κώδικας έχει μεταγλωττιστεί με την παράμετρο `-g` στον μεταγλωττιστή (`javac`), διαδικασία ωστόσο που δεν είναι συνηθισμένη.

Η τεκμηρίωση που παρέχουν οι βιβλιοθήκες BCEL και JGraph, κρίνεται για την πρώτη από μέτρια ως ικανοποιητική και για τη δεύτερη ως αρκετά καλή. Και τα δύο παρέχουν το Javadoc για τα API τους, αλλά σε κάποια σημεία αυτό είναι είτε ελλειπές είτε ανύπαρκτο για ορισμένες ιδιότητες κλάσεων ή μεθόδους. Επίσης τα παραδείγματα που περιέχουν τα Javadoc είναι ελάχιστα. Ωστόσο, για το JGraph υπάρχει λεπτομερές εγχειρίδιο που παρέχεται δωρεάν από την ιστοσελίδα του[6] και παράλληλα παρέχεται υποστήριξη μέσω φόρουμ συζητήσεων[7], η οποία κυμαίνεται σε αρκετά καλά επίπεδα. Εκεί μπορούν να βρεθούν αρκετά παραδείγματα κώδικα ή λύσεις σε προβλήματα. Το BCEL περιορίζεται μόνο σε μια ιστοσελίδα τύπου σύντομου εγχειριδίου[8], ωστόσο, κάποια παραδείγματα κώδικα μπορούν να βρεθούν και σε άλλες ιστοσελίδες. Γενικά, θα μπορούσε να υπάρξει καλύτερη τεκμηρίωση και στις δύο βιβλιοθήκες όσον αφορά στο Javadoc και ιδιαίτερα στο BCEL, γιατί όσον αφορά στο JGraph το εγχειρίδιο και το φόρουμ υποστήριξής του καλύπτει τα κενά του Javadoc του API του.

3.7.2 Προτάσεις για βελτίωση

Η εφαρμογή θα μπορούσε να επεκταθεί περαιτέρω προς την αναζήτηση για τύπους συλλογών, λιστών, map, κτλ. προκειμένου να βρεθούν επιπλέον συσχετίσεις μεταξύ των κλάσεων ενός συστήματος. Μάλιστα, αυτό δε θα μπορούσε να γίνει μόνο προς την κατεύθυνση για αναζήτηση του τελεστή `new`, αλλά και προς την κατεύθυνση εύρεσης των κλήσεων των μεθόδων τύπου `add` (και παρόμοιων) με τα οποία προσθέτει ο προγραμματιστής αντικείμενα στους ειδικούς αυτούς τύπους.

Μία επιπλέον δυνατότητα που θα μπορούσε να υπάρξει στην εφαρμογή θα ήταν η απευθείας επέμβαση στον πηγαίο κώδικα μιας κλάσης και η ταυτόχρονη απεικόνιση των αλλαγών που επιφέρουν αυτές στο διάγραμμα των κλάσεων. Υπάρχουν εργαλεία που παρέχουν τέτοια δυνατότητα, όπως το Borland Together, δίνοντας έτσι τη δυνατότητα στο χρήστη να βλέπει

[6] http://www.jgraph.com/pub/jgraphmanual.pdf
[7] http://www.jgraph.com/forum
[8] http://jakarta.apache.org/bcel/manual.html

το πως μια αλλαγή στον κώδικα, αλλάζει τις συσχετίσεις των κλάσεων στο διάγραμμα. Μια τέτοιου είδους δυνατότητα ωστόσο απαιτεί αρκετή προσπάθεια και δεν ήταν στα πλαίσια της παρούσας εργασίας.

3.7.3 Συμπεράσματα

Τις τελευταίες δεκαετίες συμβαίνουν άλματα στη μηχανική λογισμικού, τόσο όσον αφορά στις τεχνολογίες που υπάρχουν, όσο και στα διαθέσιμα εργαλεία λογισμικού. Αρκεί να δει κανείς τι εργαλεία για το σχεδιασμό και την ανάπτυξη του λογισμικού υπάρχουν τώρα και τι υπήρχαν πριν από δέκα και παραπάνω χρόνια. Όσο οι ανάγκες και η πολυπλοκότητα των συστημάτων αυξάνει, τόσο θα βελτιώνονται τα εργαλεία. Παράλληλα, σε όλες αυτές τις διαδικασίες ρόλο παίζει ο παράγοντας *προτυποποίηση*. Η ύπαρξη κοινών (και ανοικτών) προτύπων βοηθά στη συνεργασία μεταξύ σχεδιαστών – αναλυτών και προγραμματιστών που ανήκουν σε διαφορετικές ομάδες, ενώ η δημιουργία τέτοιων προτύπων γίνεται μέσα από διαδικασίες που λαμβάνουν μέρος όλοι.

Στόχος της εργασίας ήταν η ανάπτυξη ενός εργαλείου για την κατανόηση ενός συστήματος λογισμικού, μέσω μιας γλώσσας μοντελοποίησης, που στη συγκεκριμένη περίπτωση είναι η UML. Όντας απλό στη χρήση του, ο γράφων ελπίζει το εργαλείο αυτό να βοηθήσει στην κατανόηση της δομής ενός συστήματος λογισμικού που βρίσκεται με τη μορφή αρχείων στον υπολογιστή.

ΒΙΒΛΙΟΓΡΑΦΙΑ

BCEL (2006). *BCEL - Byte Code Engineering Library (BCEL)*. Ανάκτηση 1/11/2006 από World Wide Web: http://jakarta.apache.org/bcel/manual.html.

D. Pilone and Pitman, N. (2005). *Learning UML 2.0*, O'Reilly.

JGraph Ltd. (2006). *JGraph and JGraph Layout Pro User Manual.*

K. Hamilton and Miles, P. (2006). *Learning UML 2.0*, O'Reilly.

M. Fowler (2004α). *UML Distilled Third Edition*, Boston, Pearson Education, Inc., σ. 21–23.

M. Fowler (2004β). *UML Distilled Third Edition*, Boston, Pearson Education, Inc., σ. 7–9.

M. Fowler (2004ς). *UML Distilled Third Edition*, Boston, Pearson Education, Inc., σ. 1–1.

M. Fowler (2004δ). *UML Distilled Third Edition*, Boston, Pearson Education, Inc., σ. 2–3.

M. Fowler (2004ε). *UML Distilled Third Edition*, Boston, Pearson Education, Inc., σ. 99–100.

O.M.G. (2005). *Unified Modeling Language: Superstructure, version 2.0 formal/05-07-04.* Object Management Group.

O.M.G. (2006). *Unified Modeling Language: Infrastructure, version 2.0 formal/05-07-05.* Object Management Group.

Philippe Kruchten (1995). "architectural blueprints – the "4+1" view model of software architecture", *IEEE Software*, 12.

Wikipedia (2006α). *Booch method – Wikipedia, the free encyclopedia.* Ανάκτηση 3/11/2006 από World Wide Web: http://en.wikipedia.org/wiki/Booch_method.

Wikipedia (2006β). *Object Management Group – Wikipedia, the free encyclopedia.* Ανάκτηση 3/11/2006 από World Wide Web: http://en.wikipedia.org/wiki/Object_Management_Group.

Wikipedia (2006ς). *Reverse engineering – Wikipedia, the free encyclopedia.* Ανάκτηση 20/11/2006 από World Wide Web: http://en.wikipedia.org/wiki/Reverse_engineering.

Wikipedia (2006δ). *Unified Modeling Language – Wikipedia, the free encyclopedia.* Ανάκτηση 3/11/2006 από World Wide Web: http://en.wikipedia.org/wiki/Unified_Modeling_Language.

Γλωσσάρι

abstraction	αφαίρεση, 27, 43
abstract	αφηρημένος, 33, 60
active	ενεργός, 19
activities language unit	γλωσσική μονάδα δραστηριοτήτων, 13
activity diagram	διάγραμμα δραστηριοτήτων, 24
aggregation	συνάθροιση, 39
association	συσχέτιση, 22, 38
attribute	ιδιότητα, 32, 35
business process	επιχειρησιακή διεργασία, 24
business-to-business	επιχείρηση-σε-επιχείρηση, 31
cast	μετατροπή τύπου, 66
cell	κελί, 51
class diagram	διάγραμμα κλάσεων, 9, 31
collaboration diagram	διάγραμμα συνεργασίας, 21
collaboration	συνεργασία, 22
communication diagram	διάγραμμα επικοινωνίας, 21
communication line	γραμμή επικοινωνίας, 31
communication path	μονοπάτι επικοινωνίας, 30
compliance level	επίπεδο συμμόρφωσης, 12, 13
component diagram	διάγραμμα συστατικών, 27
component	συστατικό, 27
composite structure	σύνθετη δομή, 21