ΕΛΛΗΝΙΚΟ ΑΝΟΙΚΤΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

Ανάπτυξη συστήματος λογισμικού με UML και Java

Μισθοδοσία υπαλλήλων

Πάνος Φιτσιλής

2007

ΘΕΜΑΤΙΚΗ ΕΝΟΤΗΤΑ ΠΛΗ24

Για παρατηρήσεις και σχόλια στείλτε e-mail στο fitsilis@teilar.gr

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

[9]

1. Σκοπός της μελέτης περίπτωσης

Η παρούσα μελέτη περίπτωσης έχει τριπλό σκοπό:

1. Να παρουσιάσει τη χρήση αντικειμενοστρεφών μεθοδολογιών ανάπτυξης στη δημιουργία λογισμικού
2. Να παρουσιάσει την κατασκευή ενός συστήματος μισθοδοσίας υπαλλήλων με τη χρήση της γλώσσας UML.
3. Να παρουσιάσει το μετασχηματισμό του μοντέλου UML σε κώδικα Java. Η ανάπτυξη του κώδικα θα παρουσιασθεί βήμα-βήμα ενώ ταυτόχρονα γίνεται αναλυτική επεξήγηση του κώδικα.

Μετά τη μελέτη του παραδείγματος ο αναγνώστης θα είναι σε θέση :

1. Να χρησιμοποιήσει τη μεθοδολογία ICONIX για την ανάπτυξη ενός συστήματος λογισμικού με αντικειμενοστρεφή τρόπο.
2. Να αναπτύξει τα απαραίτητα μοντέλα που απαιτούνται στην κάθε φάση ανάπτυξης λογισμικού συστήματος με τη χρήση της γλώσσας UML.
3. Να κατασκευάσει τον κώδικα Java με βάση τα μοντέλα UML που αναπτύχθηκαν.
4. Να αποθηκεύει δεδομένα στη βάση δεδομένων.

2. Παρουσίαση της μελέτης περίπτωσης

Το σύστημα που καλούμαστε να μοντελοποιήσουμε είναι ένα σύστημα μισθοδοσίας που έχει αναπτυχθεί για τις ανάγκες μιας επιχείρησης. Στη γενική περίπτωση, η εφαρμογή μισθοδοσίας αποτελεί μια σύνθετη επιχειρηματική εφαρμογή αφού οφείλει να λάβει υπόψη μεγάλο αριθμό επιχειρηματικών κανόνων (business rules), που έχουν σχέση με το είδος της ασφάλειας του εργαζομένου, την προϋπηρεσία του εργαζομένου, την οικογενειακή κατάσταση κ.λπ.

Στο παράδειγμά μας παρουσιάζουμε μια απλοποιημένη μορφή μισθοδοσίας με περιορισμένο αριθμό απαιτήσεων. Στο παράδειγμα αυτό, η επιχείρηση έχει εργαζομένους διαφορετικών κατηγοριών, όπου κάθε κατηγορία εργαζομένων πληρώνεται με διαφορετικό τρόπο. Έτσι, η συγκεκριμένη εταιρεία έχει εργαζομένους:

- Πλήρους απασχόλησης
- Μερικής απασχόλησης

Μια ειδική κατηγορία εργαζομένων πλήρους απασχόλησης είναι οι *πωλητές,* οι οποίοι εκτός από το μηνιαίο μισθό τους λαμβάνουν πρόσθετη αμοιβή (bonus) η οποία εξαρτάται από το συνολικό ποσό των παραγγελιών που έλαβαν για τη δεδομένη χρονική περίοδο. Έτσι ένας πωλητής λαμβάνει:

- επιπλέον 5% του μισθού του, σε εβδομαδιαία βάση, εάν οι παραγγελίες που έλαβε είναι μικρότερες των 10.000 €.
- επιπλέον 7% του μισθού του, σε εβδομαδιαία βάση, εάν οι παραγγελίες που έλαβε είναι περισσότερες ή ίσες των 10.000 € και λιγότερες των 20.000 €.
- Για παραγγελίες μεγαλύτερες ή ίσες των 20.000 € λαμβάνουν επιπλέον 10% του μισθού τους σε εβδομαδιαία βάση.

Επιπλέον, το σύστημα μισθοδοσίας θα πρέπει να υπολογίζει το ποσό που αντιστοιχεί στις υπερωρίες των εργαζομένων. Ως υπερωρία θεωρείται η εργασία μετά τις 8 ώρες ημερησίως, η οποία αμείβεται με το ωρομίσθιο του κάθε εργαζομένου προσαυξημένο κατά 50%.

Για την ακριβή καταγραφή των ωρών εργασίας των εργαζομένων, η εταιρεία έχει

εγκαταστήσει σύστημα «Ελέγχου Προσέλευσης/Αποχώρησης Εργαζομένων», το οποίο καταγράφει την ώρα προσέλευσης και αποχώρησης του κάθε εργαζομένου. Για το λόγο αυτό, ο κάθε εργαζόμενος είναι εφοδιασμένος με προσωπική κάρτα εισόδου, ενώ στην είσοδο των εγκαταστάσεων της εταιρείας έχει εγκατασταθεί αναγνώστης καρτών.

Οι πωλητές εκτός από τη χρήση της κάρτας τους, όπως και κάθε εργαζόμενος, μπορούν να συνδέονται στο σύστημα και να καταχωρούν τις παραγγελίες που έχουν λάβει.

Το υπό ανάπτυξη σύστημα διαχειρίζεται ο υπεύθυνος μισθοδοσίας, ο οποίος έχει την αρμοδιότητα εισαγωγής, διαγραφής, τροποποίησης στοιχείων της μισθοδοσίας, καθώς επίσης και την παραγωγή της μισθοδοσίας.

Τέλος, το σύστημα θα πρέπει να διαθέτει δυνατότητα παραγωγής εκτυπώσεων:

- Φύλλο μισθοδοσίας ανά εργαζόμενο
- Συγκεντρωτική μισθοδοσία για όλη την εταιρεία
- Παραγγελίες ανά εργαζόμενο

3. Η μεθοδολογία ICONIX

Στο κεφάλαιο αυτό θα παρουσιάσουμε τη μεθοδολογία αντικειμενοστρεφούς ανάπτυξης ICONIX[1], η οποία με την αμεσότητα και την πληρότητά της μπορεί να ελαττώσει τον ανταγωνισμό ανάμεσα στις ομάδες ανάπτυξης και διαχείρισης ενός έργου λογισμικού, αυξάνοντας ταυτόχρονα την ποιότητα του παραγόμενου λογισμικού. Σύμφωνα με το συγγραφέα της ICONIX, Doug Rosenberg, η μεθοδολογία ICONIX είναι παλαιότερη της UML και της ενοποιημένης προσέγγισης (unified process) και δημιουργήθηκε από τη σύνθεση της μεθοδολογίας OMT του Jim Rumbaugh's, της μεθοδολογίας Objectory του Ivar Jacobson's, και της μεθοδολογίας του Grady Booch's. Πάντα σύμφωνα με τον Doug Rosenberg χρησιμοποιήθηκαν στοιχεία της OMT για την εκφραστικότητά τους στη μοντελοποίηση του πεδίου προβλήματος (problem domain), στοιχεία της μεθοδολογίας του Booch για τις δυνατότητές της στη αναλυτική σχεδίαση και στοιχεία της μεθοδολογίας Objectory για τη μοντελοποίηση των δυναμικών χαρακτηριστικών των συστημάτων.

Η ICONIX είναι απλούστερη και συντομότερη από την πολύ εκτενή μεθοδολογία Rational Unified Process (RUP). Όπως και η RUP, η ICONIX υιοθετεί την UML ως γλώσσα έκφρασης των απαιτήσεων και των προδιαγραφών του υπό σχεδίαση λογισμικού και είναι «καθοδηγούμενη από τις περιπτώσεις χρήσης». Ταυτόχρονα, αποφεύγει την πληθώρα μοντέλων χωρίς να παραλείπει τις διαδικασίες ανάλυσης και σχεδιασμού.

Επιπλέον, η μεθοδολογία ICONIX είναι ευέλικτη μέθοδος ανάπτυξης λογισμικού. Η ευελιξία (agility) στην ανάπτυξη συστημάτων λογισμικού είναι η ικανότητα της προσαρμογής και του επαναπροσδιορισμού ενός αναπτυσσόμενου και συνεχώς εξελισσόμενου συστήματος στην περίπτωση που εμφανίζονται αλλαγές στις αρχικές απαιτήσεις και παραδοχές. Οι ευέλικτες μέθοδοι αποτελούν την πλέον σύγχρονη

[1] Το παρόν κεφάλαιο κάνει μόνο μια σύντομη επισκόπηση της ICONIX. Θα βρείτε την πλήρη ανάπτυξη της μεθοδολογίας με αναλυτικά παραδείγματα στα βιβλία «Applying Use Case Driven Object Modeling with UML», και «Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example» των Doug Rosenberg και Kendall Scott (εκδ. Addison Wesley) καθώς και στην πιο πρόσφατη έκδοση των Doug Rosenberg, Matt Stephens και Mark Collins-Cope με τίτλο Agile Development with ICONIX Process—People, Process, and Pragmatism καθώς και στο βιβλίο με τίτλο «Αντικειμενοστρεφής Ανάπτυξη Λογισμικού με τη UML» των Γερογιάννη, Β., Κακαρόντζα, Γ., Καμέα, Α., Σταμέλου, Γ., και Φιτσιλή, Π. (εκδ. Κλειδάριθμος).

προσέγγιση και τάση στην ανάπτυξη λογισμικού. Το Agile Software Development Manifesto εκδόθηκε από μία ομάδα προγραμματιστών και συμβούλων επιχειρήσεων το 2001 (http://agilemanifesto.org/) (Cockburn, 2006; Anderson, 2003; Highsmith, 2002) και εστιάζεται στην αναγνώριση του ανθρώπινου παράγοντα ως τον πρωταρχικό παράγοντα επιτυχίας ενός έργου λογισμικού, τη συνεχή έμφαση στην αποτελεσματικότητα, την προσαρμοστικότητα και τη διαχείριση της αλλαγής (http://c2.com/cgi/wiki?ExtremeProgramming)[2].

Συνοψίζοντας, τα πλεονεκτήματα της ICONIX είναι:

- Χρησιμοποιεί UML αλλά περιέχει μόνο τον ελάχιστο αριθμό απαραίτητων βημάτων, αποφεύγοντας την «παράλυση» της ομάδας έργου λόγω υπερβολικής ανάλυσης.
- Παρέχει σε κάθε βήμα τη δυνατότητα ανίχνευσης του βαθμού υλοποίησης των απαιτήσεων – δεν επιτρέπει σε κανένα σημείο την απομάκρυνση από τις ανάγκες του χρήστη.
- Είναι επαναληπτική και αυξητική – το στατικό μοντέλο εκλεπτύνεται καθώς αναλύεται το δυναμικό μοντέλο – χωρίς να επιφέρει μεγάλη διαχειριστική επιβάρυνση.
- Είναι ευέλικτη.

Για να εφαρμόσουμε την ICONIX θα χρησιμοποιήσουμε μόνο τέσσερα UML διαγράμματα:

1. *Διάγραμμα περιπτώσεων χρήσης* για να αναπαραστήσουμε τα σενάρια χρήσης και τους χειριστές του συστήματος.
2. *Διάγραμμα κλάσεων* για να αναπαραστήσουμε το πεδίο προβλήματος του συστήματος αλλά και τη λεπτομερή στατική δομή του συστήματος.

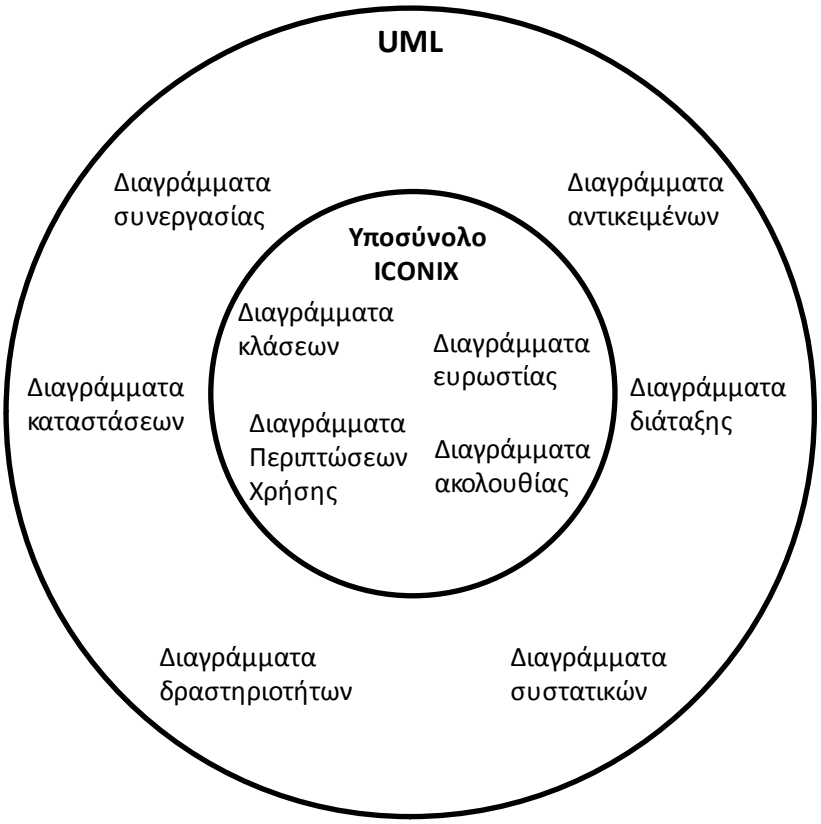
[2] Μια σύντομη εισαγωγή στις ευέλικτες μεθόδους παρουσιάζεται στο Παράρτημα Α.

[14]

3. *Διάγραμμα ευρωστίας* για να αναπαραστήσουμε τον τρόπο με τον οποίο η στατική δομή υλοποιεί τα σενάρια χρήσης του συστήματος.

4. *Διάγραμμα ακολουθίας* για να συσχετίσουμε λεπτομερώς τη δυναμική συμπεριφορά με τη στατική δομή του συστήματος.

Η σχέση του υποσυνόλου διαγραμμάτων της ICONIX με το σύνολο των διαθέσιμων διαγραμμάτων UML παρουσιάζεται στην Εικόνα 1.



Εικόνα 1: Το υποσύνολο ICONIX

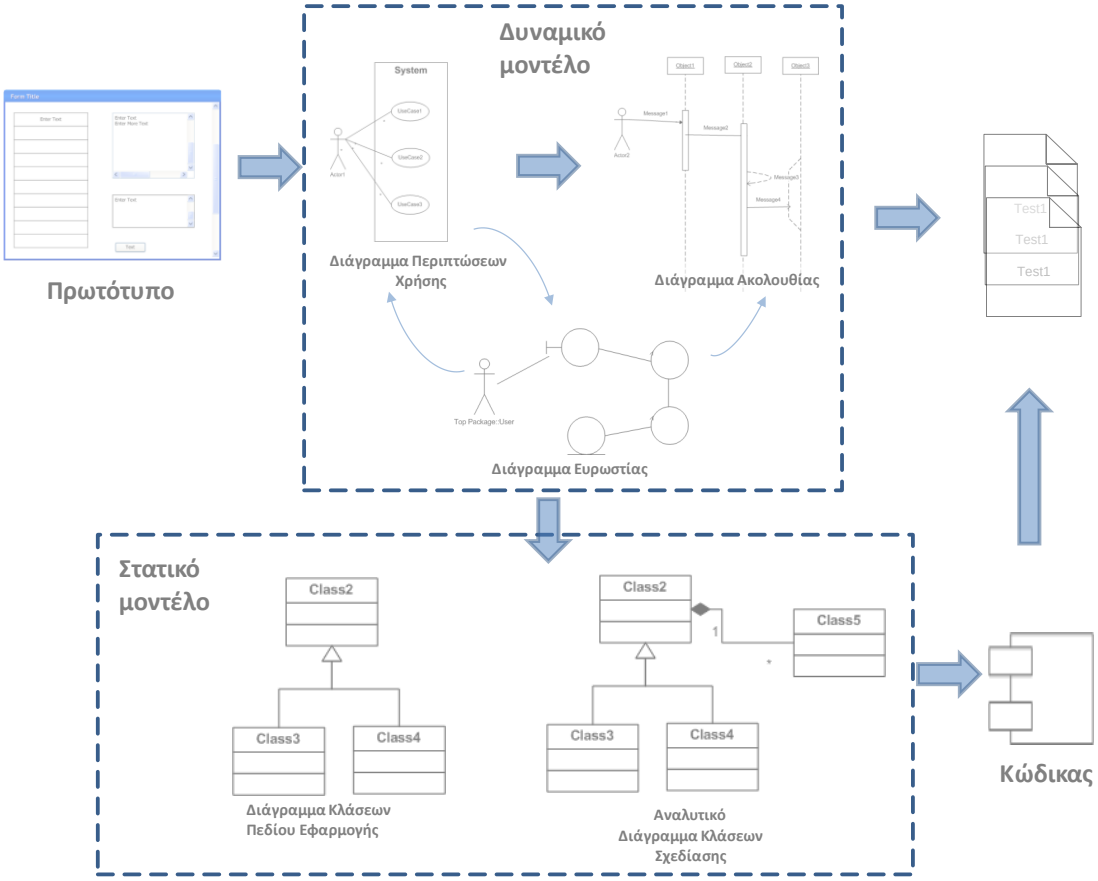
3.1 Δομή της μεθοδολογίας

Η μεθοδολογία ICONIX περιλαμβάνει τρεις φάσεις:

1. Ανάλυση απαιτήσεων
2. Αρχικός σχεδιασμός
3. Σχεδιασμός

Στα πλαίσια κάθε φάσης, αναπτύσσουμε ένα σύνολο μοντέλων, τα οποία συνολικά μας επιτρέπουν να φτάσουμε στην παραγωγή κώδικα ξεκινώντας από τις περιπτώσεις χρήσης. Η διαδικασία περιλαμβάνει ένα σύνολο ορόσημων (milestones) στα οποία έχουμε την ευκαιρία να ελέγξουμε την ποιότητα της δουλειάς.

Μια οπτικοποιημένη περιγραφή της μεθοδολογίας ICONIX φαίνεται στην Εικόνα 2. Συνοψίζοντας, ξεκινάμε με μια γρήγορη πρώτη εκτίμηση των αντικειμένων που βασίζεται εντελώς στην περιγραφή του προβλήματος, την οποία καταγράφουμε σε ένα συνοπτικό διάγραμμα κλάσεων. Στη συνέχεια, εφαρμόζουμε μια μακρά διαδικασία βασισμένη στην ανάλυση της δυναμικής συμπεριφοράς του συστήματος, κατά την οποία αναλύουμε λεπτομερώς την αναμενόμενη λειτουργία κάθε σεναρίου χρησιμοποιώντας διαγράμματα ευρωστίας (ένα είδος διαγράμματος συνεργασίας) και ακολουθίας, και ενημερώνοντας ταυτόχρονα το διάγραμμα κλάσεων του συστήματος. Έπειτα επιστρέφουμε στο σενάριο και αναλύουμε εκτενέστερα τη συμπεριφορά του συστήματος, επαναλαμβάνοντας τη διαδικασία. Με τον τρόπο αυτό αρχικά αποδομούμε το σύστημα με βάση τα όρια των περιπτώσεων χρήσης και στη συνέχεια, ανάλογα με τα αποτελέσματα της ανάλυσης των σεναρίων χρήσης, δημιουργούμε μοντέλα της δομής του συστήματος που είναι αρκετά λεπτομερή για να βασίσουμε σε αυτά την παραγωγή του κώδικα.



Εικόνα 2: Επισκόπηση της μεθοδολογίας ICONIX

Ακολουθώντας τη μεθοδολογία ICONIX, μας δίνεται η δυνατότητα να ελέγξουμε τα παραγόμενα μοντέλα τρεις φορές. Στο τέλος της ανάλυσης απαιτήσεων ελέγχουμε κυρίως τα διαγράμματα περιπτώσεων χρήσης σε συνδυασμό με το μοντέλο πεδίου προβλήματος. Στο τέλος του αρχικού και του τελικού σχεδιασμού ελέγχουμε τα διαγράμματα κλάσεων σε συνδυασμό με τα διαγράμματα ευρωστίας και τα διαγράμματα ακολουθίας, αντίστοιχα.

Επιγραμματικά τα βήματα της μεθοδολογίας ICONIX είναι τα παρακάτω:

- Βήμα 1: Προσδιορισμός των αντικειμένων του πεδίου προβλήματος.
- Βήμα 2: Προσδιορισμός των απαιτήσεων του συστήματος με χρήση περιπτώσεων χρήσης.
- Βήμα 3: Ανάλυση ευρωστίας με σκοπό τον εντοπισμό κενών στο μοντέλο του πεδίου προβλήματος (domain model), καθώς και σε μια πρώτη προσέγγιση της συμπεριφοράς του συστήματος, στα πλαίσια μιας περίπτωσης χρήσης.

[17]

- Βήμα 4: Λεπτομερής σχεδιασμός, προσδιορισμός της συμπεριφοράς των αντικειμένων με την ανάπτυξη διαγραμμάτων ακολουθίας.
- Βήμα 5: Ολοκλήρωση του στατικού μοντέλου, δηλαδή του διαγράμματος κλάσεων.
- Βήμα 6: Συγγραφή του κώδικα.
- Βήμα 7: Έλεγχος (testing) του συστήματος καθώς και έλεγχος αποδοχής (acceptance testing).

Επιπλέον, επιγραμματικά τα σημεία ελέγχου-ορόσημα που ορίζονται στη μεθοδολογία ICONIX είναι τα ακόλουθα:

Ορόσημο 1: Επισκόπηση απαιτήσεων

Ορόσημο 2: Προκαταρτική επισκόπηση σχεδιασμού

Ορόσημο 3: Κρίσιμη επισκόπηση σχεδιασμού

Ορόσημο 4: Παράδοση του συστήματος

3.2 Φάση 1. Ανάλυση Απαιτήσεων

Βήμα 1. Αναπαράσταση πεδίου προβλήματος

Περιλαμβάνει τον εντοπισμό εννοιολογικών οντοτήτων (αφαιρέσεων) – που προέρχονται κυρίως από τον πραγματικό κόσμο του πεδίου προβλήματος – που θα συγκροτήσουν τελικά το υπό ανάπτυξη σύστημα και την περιγραφή των σχέσεων ανάμεσα σε αυτές.

Ο στόχοι του πρώτου βήματος είναι δύο:

- να παραχθεί το αρχικό διάγραμμα κλάσεων του συστήματος από το μοντέλο του πεδίου προβλήματος.
- να δημιουργήσουμε πρωτότυπα του συστήματος με τεχνικές ταχείας πρωτοτυποποίησης

[18]

Βήμα 2. Σχεδίαση περιπτώσεων χρήσης

Βασικός στόχος του βήματος είναι να αναπτύξουμε το μοντέλο περιπτώσεων χρήσης. Το μοντέλο περιπτώσεων χρήσης καθορίζει τη δυναμική συμπεριφορά του συστήματος.

3.3 Σημείο ελέγχου 1. Επισκόπηση απαιτήσεων

Ο στόχος είναι να επιτευχθεί συμφωνία ανάμεσα σε όλες τις πλευρές (εμπλεκόμενους) ότι οι περιπτώσεις χρήσης που έχουν σχεδιαστεί, όταν εφαρμόζονται στα πλαίσια του μοντέλου του πεδίου προβλήματος, αντικατοπτρίζουν ακριβώς και πλήρως τις λειτουργικές απαιτήσεις του συστήματος, όπως αυτές περιγράφονται στα σενάρια λειτουργίας. Ταυτόχρονα, η ομάδα ανάπτυξης πρέπει να βεβαιωθεί ότι οι πελάτες γνωρίζουν τι θέλουν να κάνει το σύστημα – αν χρειαστεί, η ομάδα ανάπτυξης θα πρέπει να βοηθήσει τους πελάτες να συνειδητοποιήσουν και να εκφράσουν τι θέλουν να κάνει το σύστημα.

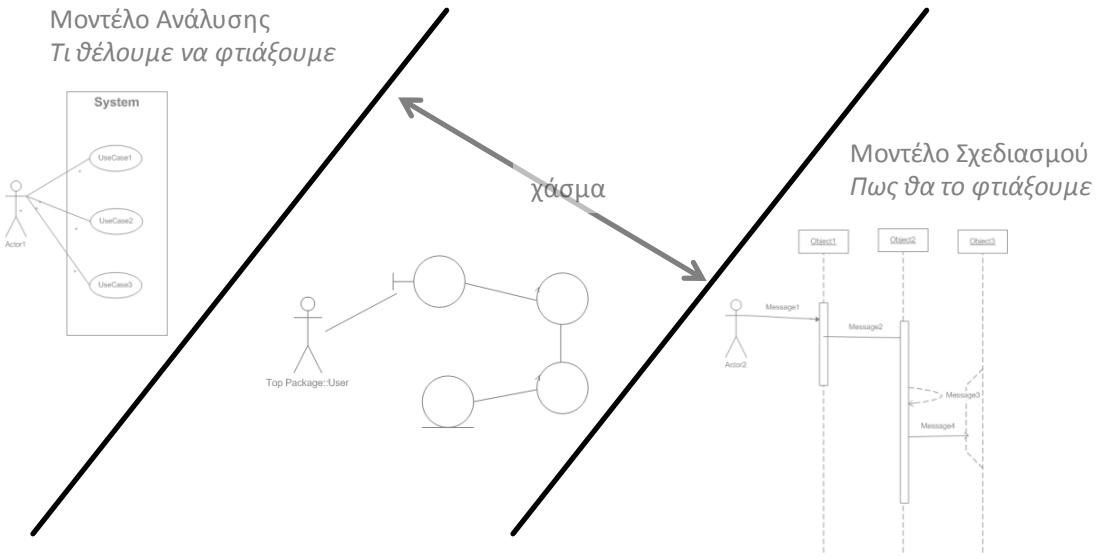
- Φροντίσουμε ώστε οι περιγραφές των περιπτώσεων χρήσης να είναι ακριβείς, συνοπτικές και – το κυριότερο – συγκεκριμένες, ώστε να είναι κατανοητές και από μη τεχνικό προσωπικό (π.χ. χρήστες, διαχειριστές).

- Βεβαιωθούμε ότι το μοντέλο του πεδίου προβλήματος αναπαριστά επακριβώς τα σχετικά με αυτό αντικείμενα του πραγματικού κόσμου και ότι οι περιγραφές των περιπτώσεων χρήσης αναφέρονται στα αντικείμενα αυτά.

3.4 Φάση 2. Αρχικός Σχεδιασμός

Βήμα 3. Ανάλυση ευρωστίας

Η ανάλυση ευρωστίας μας βοηθά να μεταβούμε από την ανάλυση του τι κάνει το σύστημα, στο σχεδιασμό, του πώς θα το κάνει. Πρόκειται για ένα καθοριστικό βήμα κάθε μεθοδολογίας ανάπτυξης λογισμικού. Στην ICONIX, θα πρέπει στο βήμα αυτό να απαντήσουμε στην ερώτηση «ποια αντικείμενα χρειαζόμαστε για να υλοποιήσουμε κάθε περίπτωση χρήσης;».



Εικόνα 3: Ο σκοπός του διαγράμματος ευρωστίας

Στόχος της ανάλυσης ευρωστίας είναι να παραχθούν διαγράμματα ευρωστίας – ένα για κάθε περίπτωση χρήσης (βασικό και εναλλακτικά σενάρια). Κάθε διάγραμμα θα αναπαριστά τα αντικείμενα του συστήματος που συμμετέχουν στα σενάρια και τον τρόπο με τον οποίο αλληλεπιδρούν. Το διάγραμμα συνεργασίας αναπαριστά το μοντέλο ευρωστίας – στην πραγματικότητα είναι ένα διάγραμμα κλάσεων, όπου όμως στη θέση κάθε κλάσης βρίσκεται ένα εννοιολογικό αντικείμενο, (τα αντικείμενα αυτά είναι λιγότερο αφαιρετικά και βρίσκονται πιο κοντά στο πραγματικό σύστημα). Ταυτόχρονα, θα εμπλουτίσουμε τις περιγραφές των περιπτώσεων χρήσης αλλά και το στατικό μοντέλο του συστήματος.

- Εμπλουτίζουμε το κείμενο των περιπτώσεων χρήσης, μειώνοντας την ασάφεια των περιγραφών, αλλά και το μοντέλο του πεδίου προβλήματος προσθέτοντας άμεσα όποια νέα αντικείμενα εντοπίστηκαν κατά τον αρχικό σχεδιασμό και καταγράφοντας βασικά κατηγορήματα στις πιο σημαντικές κλάσεις.

3.5 Σημείο ελέγχου 2. Προκαταρκτική επισκόπηση σχεδιασμού

Ο στόχος *της* προκαταρκτικής επισκόπησης σχεδιασμού είναι να βεβαιωθούμε ότι έχουμε εντοπίσει τα σημαντικά αντικείμενα του πεδίου προβλήματος και ότι αυτά είναι

αρκετά για να υλοποιήσουν την επιθυμητή συμπεριφορά του συστήματος. Για το σκοπό αυτό, οι (τώρα πια τελικές) περιγραφές των περιπτώσεων χρήσης και τα διαγράμματα συνεργασίας θα πρέπει να αποτελέσουν το συμβόλαιο ανάμεσα στην ομάδα ανάπτυξης και τους χρήστες.

3.6 Φάση 3. Σχεδιασμός

Βήμα 4. Λεπτομερής σχεδιασμός

Περιλαμβάνει τον σχεδιασμό της συμπεριφοράς των αντικειμένων που έχουμε εντοπίσει, μέσα από την αντιστοίχιση μεθόδων στις κλάσεις του συστήματος.

Ακολουθούμε μια επαναληπτική διαδικασία, επανεξετάζοντας λεπτομερώς τον αρχικό σχεδιασμό μέσα από τα σενάρια χρήσης και προσπαθώντας τώρα να περιγράψουμε επακριβώς πώς συνεργάζονται τα αντικείμενα που εντοπίσαμε. Πριν ξεκινήσουμε τη διαδικασία πρέπει να βεβαιωθούμε ότι έχουμε λεπτομερείς, ακριβείς και ορθές περιγραφές των περιπτώσεων χρήσης και ότι έχουμε «ανακαλύψει» όλα σχεδόν τα αντικείμενα που συνιστούν το σύστημα. Εάν για κάποιο από αυτά δεν είμαστε βέβαιοι ότι έχουμε πετύχει το καλύτερο δυνατό αποτέλεσμα, τότε δεν πρέπει να προχωρήσουμε στο βήμα 4, αλλά να επαναλάβουμε τα αντίστοιχα προηγούμενα βήματα.

Ο στόχος είναι να παραχθούν τα πλήρη μοντέλα της στατικής δομής και της δυναμικής συμπεριφοράς του συστήματος. Οι επιμέρους στόχοι της λεπτομερούς σχεδίασης περιλαμβάνουν:

- Την αντιστοίχιση συμπεριφοράς στα συνοριακά αντικείμενα, τα αντικείμενα οντοτήτων και τους ελεγκτές.
- Τη λεπτομερή σχεδίαση των αλληλεπιδράσεων κατά τη διάρκεια του χρόνου ανάμεσα στα αντικείμενα που εμπλέκονται στην υλοποίηση κάθε περίπτωσης χρήσης. Κατά τη διάρκεια της εκτέλεσης του λογισμικού, τα αντικείμενα κάθε κλάσης ανταλλάσσουν μηνύματα, προκαλώντας την εκτέλεση μεθόδων κάθε κλάσης.
- Την οριστική και πλήρη κατανομή πεδίων και μεθόδων στις κλάσεις.

3.7 Σημείο ελέγχου 3. Κρίσιμη επισκόπηση σχεδιασμού

Ο στόχος είναι να βεβαιωθούμε ότι ο τρόπος λειτουργίας του συστήματος (δηλαδή, η περιγραφή του «πώς επιτυγχάνεται η επιθυμητή συμπεριφορά»), όπως αντανακλάται στα διαγράμματα ακολουθίας και τα διαγράμματα κλάσεων, ταιριάζει ακριβώς με τις απαιτήσεις χρήσης του συστήματος (δηλαδή, την περιγραφή του «ποια είναι η επιθυμητή συμπεριφορά»), όπως περιγράφονται στις περιπτώσεις χρήσης. Η ανάλυση ευρωστίας έχει ήδη καταδείξει ότι η σχεδίαση μέσα στα πλαίσια του μοντέλου πεδίου προβλήματος είναι εφικτή. Τώρα πρέπει να βεβαιωθούμε ότι γνωρίζουμε τον τρόπο να υλοποιήσουμε τα συγκεκριμένα σχέδια. Είναι η «τελευταία στάση» πριν τον προγραμματισμό, γι'αυτό και προσπαθούμε να λύσουμε *όλα* τα εκκρεμή θέματα σχεδιασμού.

4 Το μοντέλο UML

Στη μελέτη θα ακολουθήσουμε τα βήματα της μεθοδολογίας ICONIX δίνοντας συμπληρωμικά στοιχεία σχετικά με τα διαγράμματα ή καλές πρακτικές όπου θεωρούμε σκόπιμο.

Ως ένα αρχικό βήμα πριν την επεξεργασία του προβλήματος θεωρούμε την απαρίθμηση των απαιτήσεων[3](requirements) όπως αυτές προκύπτουν από τη γενική περιγραφή της μελέτης περίπτωσης.

Η απαρίθμηση των απαιτήσεων θα μας βοηθήσει στην ιχνηλάτηση(tracing) αυτών κατά τη διάρκεια της ανάπτυξης του συστήματος. Γενικότερα, η διαχείριση απαιτήσεων (requirements management)[4] και το μοντέλο που προκύπτει αποτελεί τη βάση για τη σχεδίαση, τον έλεγχο και την τεκμηρίωση του συστήματος.

Η ιχνηλάτιση των απαιτήσεων επιτρέπει:

- Την καλύτερη αξιολόγηση των αλλαγών στις απαιτήσεις, η οποία θα ήταν απαραίτητη στην ανάπτυξη ενός πραγματικού συστήματος.
- Την αξιολόγηση των αποτελεσμάτων του ελέγχου (testing).
- Την επαλήθευση της ικανοποίησης των απαιτήσεων.
- Τη διαχείριση των αλλαγών.

Επιπλέον, εκτός από τις λειτουργικές απαιτήσεις θα πρέπει να λάβουμε υπόψη μας τις μη λειτουργικές απαιτήσεις (non-functional requirements)[5] του συστήματος.

---

[3] Απαίτηση είναι μια ικανότητα που θα πρέπει να διαθέτει το λογισμικό με σκοπό ο χρήστης να λύνει ένα πρόβλημα ή να πετυχαίνει ένα στόχο του ή να ικανοποιείται ένα πρότυπο ή ένας όρος του συμβολαίου κ.λπ.

[4] Διαχείριση απαιτήσεων είναι ο συστηματικός τρόπος εύρεσης, οργάνωσης και τεκμηρίωσης των απαιτήσεων του συστήματος καθώς και ο έλεγχος των αλλαγών που συμβαίνουν σε όλη τη διάρκεια του έργου

[5] Οι μη λειτουργικές απαιτήσεις προσδιορίζουν ιδιότητες του συστήματος και περιορισμούς στους οποίους υπόκειται το σύστημα. Στις πιο πολλές περιπτώσεις οι μη λειτουργικές απαιτήσεις είναι βασικές αφού η μη ικανοποίηση αυτών αχρηστεύει το σύστημα. Ως παραδείγματα μη λειτουργικών απαιτήσεων μπορούμε να αναφέρουμε την απόδοση (performance), τη διαθεσιμότητα (availability), την επεκτασιμότητα (scalability), την ευκολία χρήσης (usability), το είδος της γραφικής διεπαφής με

[23]

4.1 Απαρίθμηση των απαιτήσεων

No	Περιγραφή απαίτησης
1	Οι εργαζόμενοι της εταιρείας είναι δυο κατηγοριών: • Πλήρους απασχόλησης και • Μερικής απασχόλησης Μια ειδική κατηγορία εργαζομένων πλήρους απασχόλησης είναι οι *πωλητές.*
2	Η πληρωμή των εργαζομένων πλήρους απασχόλησης γίνεται με βάση το μηνιαίο μισθό λαμβάνοντας υπόψη τις ημέρες απουσίας.
3	Η πληρωμή των εργαζομένων μερικής απασχόλησης γίνεται με βάση τις πραγματικές ώρες εργασίας και το ωρομίσθιο του κάθε εργαζομένου.
4	Οι *πωλητές,* λαμβάνουν εκτός από το μηνιαίο μισθό πρόσθετη αμοιβή (bonus) η οποία εξαρτάται από το συνολικό ποσό των παραγγελιών που έλαβαν για τη δεδομένη χρονική περίοδο. Έτσι ένας πωλητής λαμβάνει • επιπλέον 5% του μισθού του, σε εβδομαδιαία βάση, εάν οι παραγγελίες που έλαβε είναι μικρότερες των 10.000 €. • επιπλέον 7% του μισθού του, σε εβδομαδιαία βάση, εάν οι παραγγελίες που έλαβε είναι περισσότερες των 10.000 € και λιγότερες των 20.000 €. • Για παραγγελίες μεγαλύτερες των 20.000 € λαμβάνουν επιπλέον 10% του μισθού τους σε εβδομαδιαία βάση.
5	Οι εργαζόμενοι πλήρους απασχόλησης λαμβάνουν υπερωρίες. Ως υπερωρία θεωρείται κάθε ώρα εργασίας μετά τις 8 ώρες ημερησίως, η οποία αμείβεται με το ωρομίσθιο του κάθε εργαζομένου προσαυξημένο κατά 50%. Οι υπερωρίες υπολογίζονται σε ημερήσια βάση.

το χρήστη (user interface) κ.λπ.

6	Για την ακριβή καταγραφή των ωρών εργασίας των εργαζομένων, η εταιρεία έχει εγκαταστήσει σύστημα «Ελέγχου Προσέλευσης/Αποχώρησης Εργαζομένων», το οποίο καταγράφει την ώρα προσέλευσης και αποχώρησης του κάθε εργαζομένου.
7	Κάθε εργαζόμενος έχει μια κάρτα εισόδου η οποία χρησιμοποιείται για την καταγραφή της ώρας προσέλευσης και αποχώρησης.
8	Ο υπεύθυνος μισθοδοσίας έχει την αρμοδιότητα εισαγωγής, διαγραφής και τροποποίησης των στοιχείων των εργαζομένων.
9	Ο υπεύθυνος μισθοδοσίας έχει την αρμοδιότητα παραγωγής μισθοδοσίας.
10	Το σύστημα θα πρέπει να διαθέτει δυνατότητα παραγωγής εκτυπώσεων τόσο ανά εργαζόμενο όσο και συγκεντρωτικές. Πιο συγκεκριμένα θα πρέπει να παράγει • Φύλλο μισθοδοσίας ανά εργαζόμενο • Συγκεντρωτική μισθοδοσία για όλη την εταιρεία • Παραγγελίες ανά εργαζόμενο
11	Τόσο, τα δεδομένα της μισθοδοσίας όσο και τα στοιχεία των υπαλλήλων θα αποθηκεύονται σε ΒΔ. Όταν ξεκινά η εφαρμογή θα πρέπει να διαβάζεται αυτόματα η πληροφορία των υπαλλήλων από τη ΒΔ και να δημιουργούνται τα κατάλληλα αντικείμενα (υπαλλήλων, κάρτας εισόδου, κ.ο.κ).[6]

4.2 Παραδοχές της μελέτης περίπτωσης

Οι παρακάτω παραδοχές έγιναν με σκοπό να παρουσιασθεί μια απλοποιημένη έκδοση του συστήματος που να ικανοποιεί με καλύτερο τρόπο τις εκπαιδευτικές ανάγκες που έθεσε η παρούσα μελέτη περίπτωσης.

[6] Αν και η απαίτηση αυτή δεν προσδιορίζεται σαφώς στη μελέτη περίπτωσης, θεωρούμε ότι υπονοείται καθώς η αποθήκευση των δεδομένων μιας επιχειρηματικής εφαρμογής σε μια βάση δεδομένων είναι προφανής και αναγκαία.

Οι παραδοχές αυτές είναι:

1. Κάθε εργαζόμενος έχει μια κάρτα εισόδου η οποία χρησιμοποιείται για την καταγραφή της ώρας προσέλευσης και αποχώρησης. Για την καταγραφή θα πρέπει να εξομοιωθεί η λειτουργία ενός αναγνώστη κάρτας καθώς και η προσέλευση/αναχώρηση των υπαλλήλων. Ο αναγνώστης κάρτας κάθε ημέρα και για κάθε υπάλληλο δέχεται δύο μηνύματα: Προσέλευση και Αποχώρηση. Κατά την προσέλευση η χρονική στιγμή εισαγωγής της κάρτας θα ορίζεται με τυχαίο τρόπο για το χρονικό διάστημα από 8:00 έως 8:15. Κατά την αναχώρηση η χρονική στιγμή εισαγωγής της κάρτας θα ορίζεται με τυχαίο τρόπο για το χρονικό διάστημα από 16:00 έως 16:30.
2. Το σύστημα «Ελέγχου Προσέλευσης/Αποχώρησης Εργαζομένων» αν και συνεργάζεται με το σύστημα μισθοδοσίας δεν αποτελεί τμήμα του.
3. Η εφαρμογή που παρουσιάζεται καταγράφει τους χρόνους εργασίας όλων των υπαλλήλων για τις εργάσιμες μέρες ενός μήνα. Για την απλοποίηση της λογικής υποθέτουμε πως δεν υπάρχουν απουσίες υπαλλήλων στη διάρκεια του μήνα και πως οι ωρομίσθιοι πληρώνονται στο τέλος κάθε μήνα.
4. Στην παρούσα μελέτη περίπτωσης δε θα ασχοληθούμε με τη διεπαφή χρήστη του συστήματος (graphical user interface) στο επίπεδο του κώδικα. Αν και το παράδειγμα, περιλαμβάνει γραφικές οθόνες (διαχείριση παραγγελιών) σκοπός είναι να παρουσιασθεί ο τρόπος με τον οποίο κάνουμε την πρωτοτυποποίηση του συστήματος και αναπτύσσουμε τη διεπαφή με το χρήστη. Για το λόγο αυτό δε δίνεται ιδιαίτερη έμφαση στην επεξήγηση των τεχνικών χαρακτηριστικών που σχετίζονται με την ανάπτυξη του GUI (Graphical User Interface).
5. Η παρούσα μελέτη περίπτωσης δεν έχει περιορισμούς εισόδου (login, password). Συνεπώς η λειτουργικότητα που παρουσιάζεται είναι κοινή για όλους τους χρήστες.

4.3 Φάση 1- Βήμα 1. Αναπαράσταση πεδίου προβλήματος

4.3.1 Εύρεση κλάσεων πεδίου προβλήματος

Στόχος του πρώτου βήματος της μεθοδολογίας ICONIX είναι η ανάπτυξη του αρχικού διαγράμματος κλάσεων του συστήματος από το μοντέλο του πεδίου προβλήματος.

Ένας από τους πιο απλούς τρόπους για την εύρεση των κλάσεων του πεδίου προβλήματος είναι η μελέτη των απαιτήσεων. Μερικοί απλοί κανόνες που βοηθούν είναι:

1) Τα ουσιαστικά αντιστοιχούν σε κλάσεις ή πεδία κλάσεων.

2) Τα ρήματα ή οι φράσεις που δίνουν ενέργεια γίνονται μέθοδοι ή/και συσχετίσεις (associations).

3) Οι κτητικές φράσεις συνήθως περιγράφουν ιδιότητες/πεδία των κλάσεων και συσχετίσεις.

Αν εφαρμόσουμε τους κανόνες χρησιμοποιώντας τη γενική περιγραφή του προβλήματος τότε καταλήγουμε σε ένα αριθμό υποψηφίων κλάσεων. Οι κλάσεις που μας ενδιαφέρουν και που βρήκαμε με τον τρόπο αυτό παρουσιάζονται υπογραμμισμένες, ενώ με τα λοξά γράμματα (italics) είναι οι μέθοδοι που έχουν εντοπισθεί.

No	**Περιγραφή απαίτησης**
1	Οι εργαζόμενοι της εταιρείας είναι δυο κατηγοριών: • Πλήρους απασχόλησης και • Μερικής απασχόλησης Μια ειδική κατηγορία εργαζομένων πλήρους απασχόλησης είναι οι πωλητές.
2	*Η πληρωμή των εργαζομένων πλήρους απασχόλησης* γίνεται με βάση το μηνιαίο μισθό λαμβάνοντας υπόψη τις ημέρες απουσίας.
3	*Η πληρωμή των εργαζομένων μερικής απασχόλησης* γίνεται με βάση τις πραγματικές ώρες εργασίας και το ωρομίσθιο του κάθε εργαζομένου.
4	*Οι πωλητές, λαμβάνουν εκτός από το μηνιαίο μισθό πρόσθετη αμοιβή (bonus)* η οποία εξαρτάται από το συνολικό ποσό των παραγγελιών που έλαβαν για τη δεδομένη χρονική περίοδο. Έτσι ένας πωλητής *λαμβάνει:* • επιπλέον 5% του μισθού του, σε εβδομαδιαία βάση, εάν οι παραγγελίες που έλαβε είναι μικρότερες των 10.000 €. • επιπλέον 7% του μισθού του, σε εβδομαδιαία βάση, εάν οι

No	Περιγραφή απαίτησης
	παραγγελίες που έλαβε είναι περισσότερες των 10.000 € και λιγότερες των 20.000 €. • Για παραγγελίες μεγαλύτερες των 20.000 € λαμβάνουν επιπλέον 10% του μισθού τους σε εβδομαδιαία βάση.
5	Οι εργαζόμενοι πλήρους απασχόλησης *λαμβάνουν υπερωρίες*. Ως υπερωρία θεωρείται η εργασία μετά τις 8 ώρες ημερησίως, η οποία αμείβεται με το ωρομίσθιο του κάθε εργαζομένου προσαυξημένο κατά 50%. *Οι υπερωρίες υπολογίζονται σε ημερήσια βάση*.
6	Για την ακριβή *καταγραφή των ωρών εργασίας* των εργαζομένων, η εταιρεία έχει εγκαταστήσει σύστημα «Ελέγχου Προσέλευσης/Αποχώρησης Εργαζομένων», το οποίο καταγράφει την ώρα προσέλευσης και αποχώρησης του κάθε εργαζομένου.
7	Κάθε εργαζόμενος έχει μια κάρτα εισόδου η οποία χρησιμοποιείται για την *καταγραφή της ώρας προσέλευσης και αποχώρησης*.
8	Ο υπεύθυνος μισθοδοσίας έχει την αρμοδιότητα *εισαγωγής, διαγραφής και τροποποίησης* των στοιχείων των εργαζομένων.
9	Ο υπεύθυνος μισθοδοσίας έχει την αρμοδιότητα *παραγωγής μισθοδοσίας*.
10	Το σύστημα θα πρέπει να διαθέτει δυνατότητα *παραγωγής εκτυπώσεων* τόσο ανά εργαζόμενο όσο και συγκεντρωτικές. Πιο συγκεκριμένα θα πρέπει να *παράγει:* • *Φύλλο μισθοδοσίας ανά εργαζόμενο* • *Συγκεντρωτική μισθοδοσία για όλη την εταιρεία* • *Παραγγελίες ανά εργαζόμενο.*
11	Τόσο τα δεδομένα της μισθοδοσίας όσο και τα στοιχεία των υπαλλήλων *θα αποθηκεύονται σε ΒΔ*. Όταν ξεκινά η εφαρμογή θα πρέπει να *διαβάζεται αυτόματα η πληροφορία των υπαλλήλων από τη ΒΔ* και να *δημιουργούνται τα*

No	Περιγραφή απαίτησης
	κατάλληλα αντικείμενα (υπαλλήλων, κάρτας εισόδου, κ.ο.κ).[7]

Συνοψίζοντας, για τις παραπάνω απαιτήσεις έχουμε:

Ουσιαστικό	**Σκεπτικό**
Εργαζόμενος	Είναι βασική οντότητα του συστήματος.
Εργαζόμενος πλήρους απασχόλησης	Είναι βασική οντότητα του συστήματος.
Εργαζόμενος μερικής απασχόλησης	Είναι βασική οντότητα του συστήματος.
Πωλητής	Είναι βασική οντότητα του συστήματος.
Μισθός	Είναι πεδίο της οντότητας του εργαζομένου. Λογικά μισθό θα πρέπει να έχουν οι πωλητές και οι πλήρους απασχόλησης εργαζόμενοι.
Ημέρα απουσίας	Είναι πεδίο της οντότητας του εργαζομένου.
Ώρα εργασίας	Είναι πεδίο. Θα πρέπει να σχετίζεται με τους εργαζόμενους.
Ωρομίσθιο	Είναι πεδίο του εργαζομένου μερικής απασχόλησης.
Υπερωρία	Είναι πεδίο του εργαζομένου πλήρους απασχόλησης.
Πρόσθετη αμοιβή	Είναι πεδίο του πωλητή.

[7] Αν και η απαίτηση αυτή δεν προσδιορίζεται σαφώς στη μελέτη περίπτωσης, θεωρούμε ότι υπονοείται καθώς η αποθήκευση των δεδομένων μιας επιχειρηματικής εφαρμογής σε μια βάση δεδομένων είναι προφανής και αναγκαία

Ουσιαστικό	**Σκεπτικό**
Παραγγελία	Είναι οντότητα του συστήματος. Αν και δεν αναφέρεται σαφώς κάθε παραγγελία σχετίζεται με προϊόντα.
Ποσό παραγγελιών	Είναι πεδίο της παραγγελίας.
Χρονική περίοδος	Αναφέρεται στην περίοδο υπολογισμού της μισθοδοσίας. Στη γενική περίπτωση θα μπορούσε να είναι ανεξάρτητη οντότητα.
Κάρτα εισόδου	Βασική οντότητα του συστήματος. Υπάρχει μια κάρτα εισόδου ανά εργαζόμενο. Εμμέσως πλην σαφώς η κάρτα εισόδου σχετίζεται άμεσα με τον αναγνώστη καρτών, συσκευή απαραίτητη για τη λειτουργία της.
Ώρα προσέλευσης	Είναι πεδίο που σχετίζεται με την είσοδο του εργαζομένου στην εταιρεία. Θα πρέπει να δημιουργούμε μια εγγραφή για κάθε είσοδο του εργαζομένου.
Ώρα αποχώρησης	Είναι πεδίο που σχετίζεται με την έξοδο του εργαζομένου στην εταιρεία. Θα πρέπει να δημιουργούμε μια εγγραφή για κάθε είσοδο του εργαζομένου.
Υπεύθυνος μισθοδοσίας	Είναι ρόλος του συστήματος. Δεδομένης της παραδοχής που έγινε στην παρούσα μελέτη περίπτωσης ότι «δεν υπάρχουν περιορισμοί εισόδου (login, password). Συνεπώς η λειτουργικότητα που παρουσιάζεται είναι κοινή για όλους τους χρήστες», η οντότητα αυτή δεν εμφανίζεται.
Μισθοδοσία	Βασική οντότητα του συστήματος.

Ουσιαστικό	**Σκεπτικό**
Στοιχεία εργαζομένου	Πεδία της κλάσεως εργαζόμενος.
Σύστημα	Το υπό ανάπτυξη σύστημα.
Δεδομένα μισθοδοσίας	Πεδία της οντότητα μισθοδοσίας.
Στοιχεία υπαλλήλων	Συνώνυμο του «στοιχεία εργαζομένου».

Ρήματα ή ρηματικές φράσεις

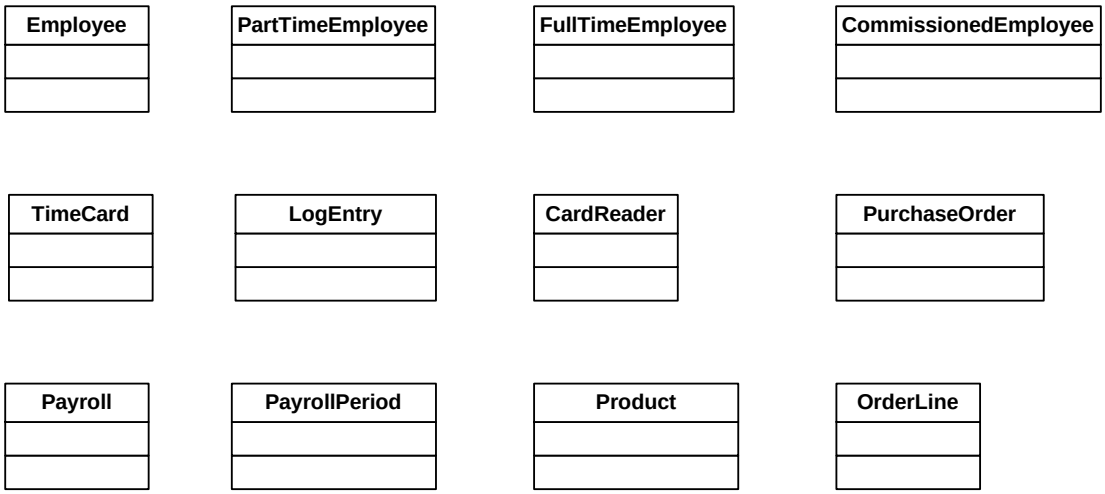
Αντίστοιχα, η ανάλυση για ρήματα ή φράσεις που υποδηλώνουν ενέργεια είναι οι παρακάτω:

- *Η πληρωμή των εργαζομένων πλήρους απασχόλησης*
- *Η πληρωμή των εργαζομένων μερικής απασχόλησης*
- *Λαμβάνουν πρόσθετη αμοιβή*
- *Λαμβάνουν υπερωρίες*
- *Παραγωγή εκτυπώσεων*
- *Οι υπερωρίες υπολογίζονται σε ημερήσια βάση*
- *Καταγραφή των ωρών εργασίας*
- *Ο υπεύθυνος μισθοδοσίας έχει την αρμοδιότητα εισαγωγής*
- *Ο υπεύθυνος μισθοδοσίας έχει την αρμοδιότητα διαγραφής*
- *Ο υπεύθυνος μισθοδοσίας έχει την αρμοδιότητα τροποποίησης*
- *Ο υπεύθυνος μισθοδοσίας έχει την αρμοδιότητα παραγωγής μισθοδοσίας*
- Να *διαβάζεται αυτόματα η πληροφορία των υπαλλήλων από τη ΒΔ* και να *δημιουργούνται τα κατάλληλα αντικείμενα*

Στην παρούσα φάση δεν ενδείκνυται η περαιτέρω ανάλυση αυτών των φράσεων αφού αρχική μας προτεραιότητα είναι να παρουσιάσουμε ένα σωστό και σταθερό μοντέλο κλάσεων του πεδίου προβλήματος και στη συνέχεια να αναθέσουμε αρμοδιότητες (responsibilities) στις κλάσεις αυτές.

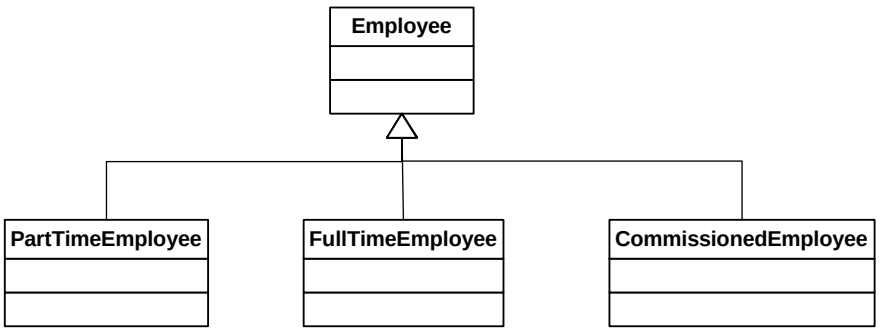
[31]

Στο σημείο αυτό είμαστε σε θέση να παράγουμε την πρώτη έκδοση του διαγράμματος κλάσεων όπως αυτό προκύπτει από το πεδίο προβλήματος. Στη συγκεκριμένη περίπτωση η επιλογή των κλάσεων που παρουσιάζονται βασίστηκε στην λεκτική ανάλυση των απαιτήσεων. Στην πραγματικότητα, η λεκτική ανάλυση δεν επαρκεί από μόνη της αφού η διαδικασία ανάπτυξης είναι πιο σύνθετη. Έτσι το αρχικό μοντέλο παράγεται μετά από σειρά συναντήσεων της ομάδας ανάπτυξης του λογισμικού, συναντήσεις με τον πελάτη, αντλώντας πληροφορία από αντίστοιχα μοντέλα άλλων παρόμοιων συστημάτων κ.λπ.

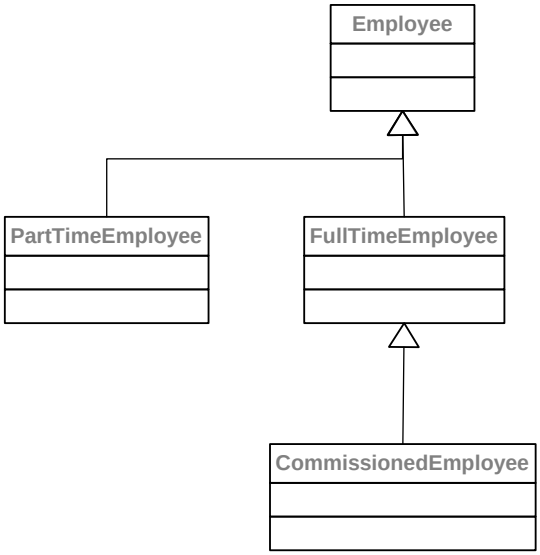


Εικόνα 4: Κλάσεις του πεδίου προβλήματος

Το επόμενο βήμα είναι να οργανώσουμε τις κλάσεις ώστε να εντοπίσουμε ιεραρχίες κληρονομικότητας καθώς και σχέσεις μεταξύ κλάσεων όπως αυτές προκύπτουν από το πεδίο προβλήματος.



(α)



(β)

Εικόνα 5: Σχέση κληρονομικότητας εργαζομένων

Στο αρχικό διάγραμμα κλάσεων υπάρχουν οι κλάσεις Employee (εργαζόμενος), PartTimeEmployee (εργαζόμενος μερικής απασχόλησης), FullTimeEmployee (εργαζόμενος πλήρους απασχόλησης) και CommissionedEmployee (πωλητής).

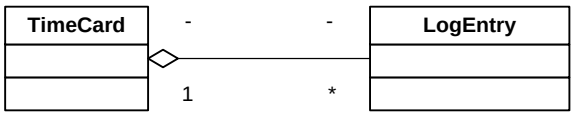
Σύμφωνα με την εκφώνηση του προβλήματος μπορούμε να δημιουργήσουμε μια ιεραρχία κληρονομικότητας όπου στην προσέγγιση (α) οι κλάσεις PartTimeEmployee, FullTimeEmployee και CommissionedEmployee κληρονομούν από την κλάση Employee. Στην προσέγγιση (β) η ιεραρχία κληρονομικότητας είναι τριών επιπέδων αφού θεωρούμε ότι οι πωλητές (CommissionedEmployee) είναι και αυτοί πλήρους απασχόλησης (FullTimeEmployee). Επιπλέον, η προσέγγιση (β) ικανοποιεί με καλύτερο τρόπο τις απαιτήσεις. Η τελική επιλογή για την ιεραρχία που θα επιλέξουμε θα γίνει σε επόμενη φάση αφού η επιλογή επηρεάζεται από τις μεθόδους και τα πεδία της που θα κατανείμουμε σε κάθε κλάση της ιεραρχίας.

Στη γενική περίπτωση, η κληρονομικότητα αυξάνει την επαναχρησιμοποίηση (reusability) του κώδικα, γεγονός που χαρακτηρίζεται θετικό και επιθυμητό. Ταυτόχρονα, όμως αυξάνει και τη σύζευξη μεταξύ των κλάσεων (coupling) μια και όταν χρησιμοποιούμε την κλάση παιδί έμμεσα χρησιμοποιούμε και τη μητρική του κλάση.

Επιπλέον, μεγάλες ιεραρχίες κλάσεων βέβαια, δημιουργούν ισχυρές εξαρτήσεις,

αφού η κλάση παιδί κληρονομεί από τη μητρική κλάση τόσο τα πεδία όσο και τις μεθόδους της. Αυτό δημιουργεί ισχυρό βαθμό συσχέτισης (cohesion) μεταξύ των κλάσεων που συμμετέχουν στην ιεραρχία. Το πρόβλημα αυτό έχει καταγραφεί στη βιβλιογραφία ως το πρόβλημα της εύθραυστης κλάσης βάσης (the fragile base class problem).

Προχωρώντας, στην εύρεση σχέσεων μεταξύ κλάσεων εξετάζουμε την επόμενη κλάση του πεδίου προβλήματος. Η κλάση TimeCard (κάρτα χρόνου) με την κλάση LogEntry (Εγγραφή χρόνου εργαζομένου) συνδέονται με σχέση συναρμολόγησης (aggregation). Η κλάση TimeCard προέκυψε απευθείας από την εκφώνηση του προβλήματος. Αντίθετα, η κλάση LogEntry προκύπτει από τη γενικότερη γνώση της ανάπτυξης συστημάτων, δηλαδή τη γνώση ότι κάθε φορά που ένας εργαζόμενος χρησιμοποιεί την κάρτα του, δημιουργεί μια εγγραφή. Έτσι, για κάθε εργαζόμενο έχουμε μία κάρτα εισόδου (Timecard) η οποία σχετίζεται με πολλές εγγραφές εισόδου (LogEntry) (μια για κάθε ημέρα). Στην Εικόνα 6 παρουσιάζεται η γραφική απεικόνιση αυτής της σχέσης συναρμολόγησης.



Εικόνα 6: Σχέση συναρμολόγησης μεταξύ κλάσεων TimeCard και LogEntry

Εναλλακτικά θα μπορούσαμε να ορίσουμε τη σχέση μεταξύ των κλάσεων TimeCard και LogEntry ως σχέση σύνθεσης (composition). Στην περίπτωση αυτή η σημασιολογία αλλάζει, αφού στη σχέση σύνθεσης τα αντικείμενα της κλάσης TimeCard και της κλάσης LogEntry έχουν την ίδια διάρκεια ζωής. Πιο συγκεκριμένα, στο παράδειγμά μας αυτό σημαίνει ότι αν ένας εργαζόμενος φύγει από την εταιρεία και διαγράψουμε την κάρτα του από το σύστημα, αυτόματα θα πρέπει να διαγραφούν και όλες οι εγγραφές εισόδου που σχετίζονται με αυτή την κάρτα. Αντίθετα, αν η σχέση ορισθεί ως συναρμολόγηση η διαγραφή μιας κάρτας δεν συνεπάγεται άμεση καταστροφή όλων των εγγραφών εισόδου. Μια επιπλέον παρατήρηση είναι ότι εάν επιλέξουμε να χρησιμοποιήσουμε σχέση σύνθεσης η πολλαπλότητα θα είναι πάντα 1, αφού μια LogEntry ανήκει πάντα σε μια TimeCard.

Στην Εικόνα 7 παρουσιάζεται η γραφική απεικόνιση αυτής της σχέσης σύνθεσης.

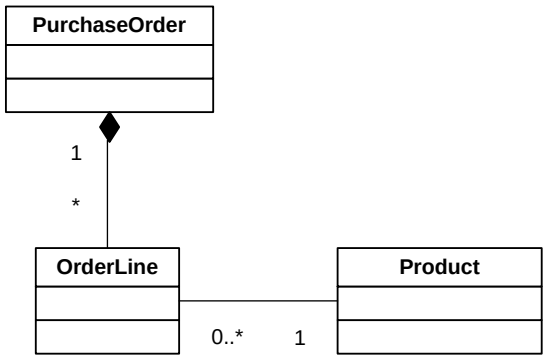
Εικόνα 7: Σχέση σύνθεσης μεταξύ κλάσεων TimeCard και LogEntry

Για το παράδειγμά μας θα προκρίνουμε τη σχέση σύνθεσης αφού τα αντικείμενα της κλάσης LogEntry υπάρχουν σε άμεση σύνδεση με τα αντικείμενα της κλάσης TimeCard.

Δραστηριότητα 1

Δώστε ένα παράδειγμα σχέσης σύνθεσης και ένα παράδειγμα σχέσης συναρμολόγησης. Εξηγείστε το σκεπτικό σας.

Αντίστοιχα, θα ορίσουμε σχέση σύνθεσης μεταξύ των κλάσεων PurchaseOrder (παραγγελία) και OrderLine (γραμμή παραγγελίας) (βλέπε Εικόνα 8).



Εικόνα 8: Σχέσεις μεταξύ κλάσεων PurchaseOrder, OrderLine και Product

Θεωρούμε, ότι η ανάλυση του πεδίου προβλήματος δεν επαρκεί στο σημείο που αφορά τη μοντελοποίηση των εννοιών της μισθοδοσίας με την κλάση Payroll και PayrollPeriod. Οι βασικοί λόγοι που μας οδηγούν σε αυτή την παρατήρηση είναι ότι:

- Θα πρέπει να τρέχουμε τη μισθοδοσία για κάθε περίοδο καθώς και
- Να αποθηκεύουμε τα αποτελέσματα της μισθοδοσίας ώστε να μπορέσουμε να τα επεξεργαστούμε για την παραγωγή των σχετικών αναφορών.

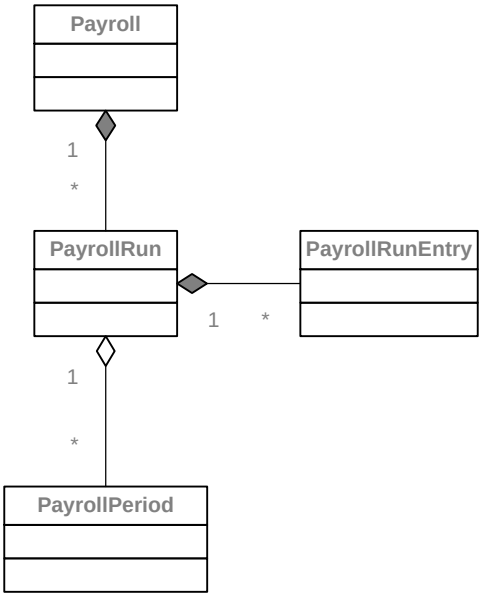
Συνεπώς, θα πρέπει να εισάγουμε δύο επιπλέον κλάσεις με σκοπό την ικανοποίηση των παραπάνω απαιτήσεων. Ως αποτέλεσμα είναι να έχουμε τέσσερις κλάσεις για τη μοντελοποίηση των απαιτήσεων παραγωγής μισθοδοσίας.

PayrollRun: Χρησιμοποιείται για το τρέξιμο της μισθοδοσίας μια χρονική περίοδο. Η κλάση PayrollRun συνδέεται με σχέση συναρμολόγησης με την κλάση PayrollRunEntry, αφού οι εγγραφές μισθοδοσίας δημιουργούνται αποκλειστικά από το τρέξιμο της μισθοδοσίας.

PayrollPeriod: Ορίζουμε την χρονική περίοδο για την οποία θα επεξεργαστούμε τα δεδομένα μισθοδοσίας. Λογικά τα αντικείμενα της PayrollRunPeriod προϋπάρχουν του τρεξίματος της μισθοδοσίας και αυτός είναι ο βασικός λόγος που συνδέουμε την κλάση PayrollRun με την κλάση PayrollRunPeriod με σχέση συναρμολόγησης και όχι σύνθεσης.

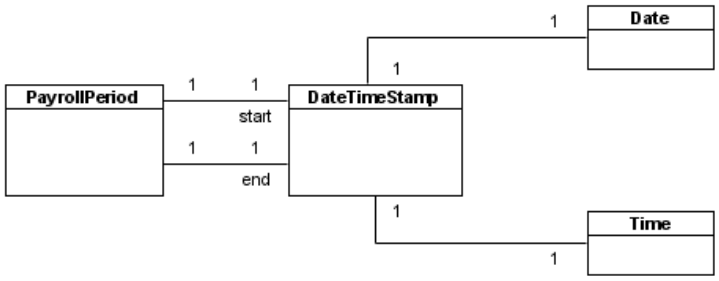
PayrollRunEntry: Η κλάση PayrollRunEntry είναι η κλάση που χρησιμοποιείται για την αποθήκευση των δεδομένων της μισθοδοσίας. Κάθε φορά που υπολογίζεται ο μισθός ενός εργαζομένου θα πρέπει να δημιουργηθεί ένα αντικείμενο της κλάσης PayrollRunEntry με σκοπό να αποθηκεύσουμε τα δεδομένα που παράγονται.

Payroll: Αποτελεί τη βασική κλάση του συστήματος καθώς και το σημείο εκκίνησης της εφαρμογής μισθοδοσίας. Η κλάση Payroll έχει ως ρόλο την αρχικοποιήσει το περιβάλλον της εφαρμογής μισθοδοσίας δημιουργώντας τα απαραίτητα αντικείμενα χρησιμοποιώντας τα δεδομένα που υπάρχουν στη βάση δεδομένων καθώς επίσης να δημιουργήσει το GUI της εφαρμογής ώστε να μπορέσει ο χρήστης να ξενικήσει την αλληλεπίδραση μαζί του.



Εικόνα 9: Οι κλάσεις του πακέτου Payroll

Στην Εικόνα 9 παρουσιάζεται η κλάση PayrollPeriod. Η έννοια της χρονικής περιόδου είναι μια συνηθισμένη έννοια η οποία υπάρχει σε αρκετές εφαρμογές. Ο M. Fowler πρότεινε για τη μοντελοποίηση της περιόδου τη δημιουργία μιας κλάσης TimePeriod, PayrollPeriod στην περίπτωσή μας όπως αυτή παρουσιάζεται στην Εικόνα 10. Οι λόγοι που μας οδηγούν στον ορισμό μιας ξεχωριστής κλάσης για τη μοντελοποίηση της χρονικής περιόδου είναι πολλοί. Για παράδειγμα, μια τέτοια κλάση θα πρέπει να ενσωματώνει επιχειρηματική λογική (business logic), δηλαδή ότι η ημερομηνία τέλους θα πρέπει να έπεται της ημερομηνίας αρχής, οι περίοδοι δεν θα πρέπει να επικαλύπτονται κ.λπ.

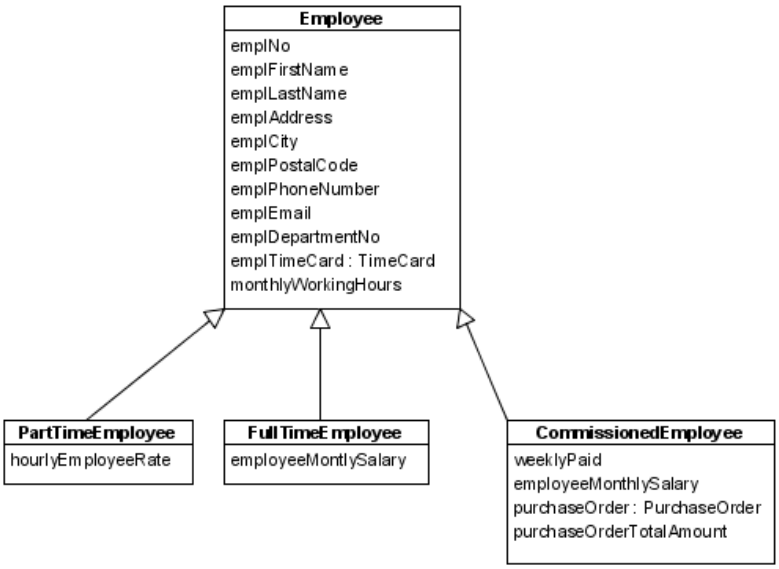


Εικόνα 10: Η μοντελοποίηση της κλάσης PayrollPeriod

Στο σημείο αυτό θα εστιάσουμε την προσοχή μας στον εντοπισμό των πεδίων των

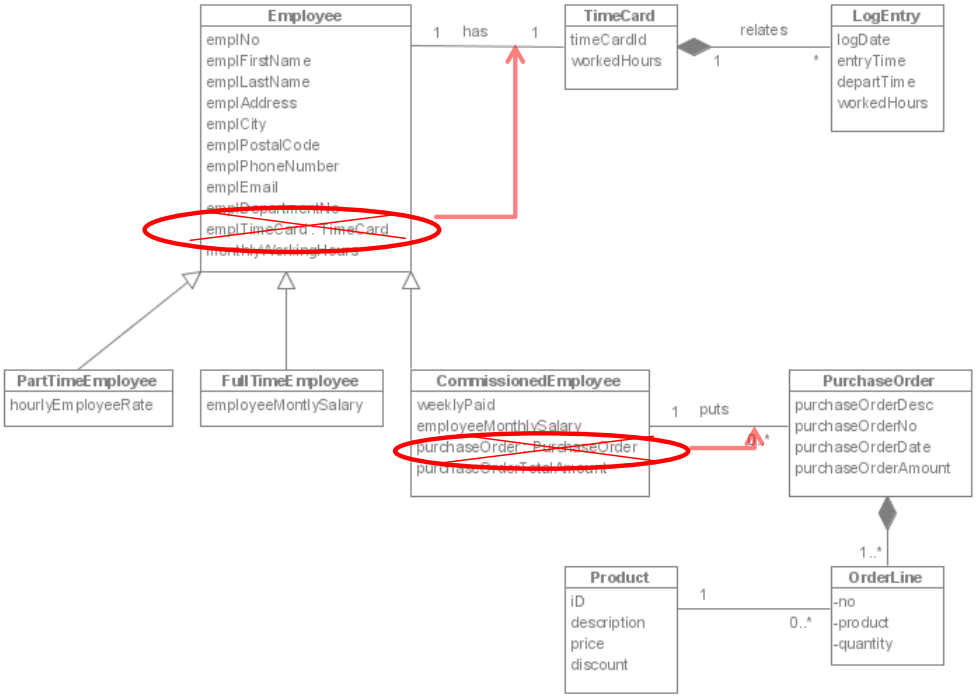
κλάσεων όπως αυτές προκύπτουν από το πεδίο προβλήματος.

Στην Εικόνα 11 παρουσιάζεται η κατανομή των πεδίων στην ιεραρχία κλάσεων Employee. Η κλάση Employee είναι η μητρική κλάση των κλάσεων PartTimeEmployee, FullTimeEmployee και CommissionedEmployee. Αυτό σημαίνει ότι όλα τα πεδία της κλάσης Employee κληρονομούνται από τις κλάσεις παιδιά. Έτσι για παράδειγμα, τα πεδία της κλάσης PartTimeEmployee είναι EmplNo, EmplFirstName, EmplLastName, EmplAddress, EmplCity, EmplPostalCode, EmplPhoneNumber, EmplEmail, EmplDepartmentNo, EmplTimeCard : TimeCard, MonthlyWorkingHours και το πεδίο HourlyEmployeeRate.



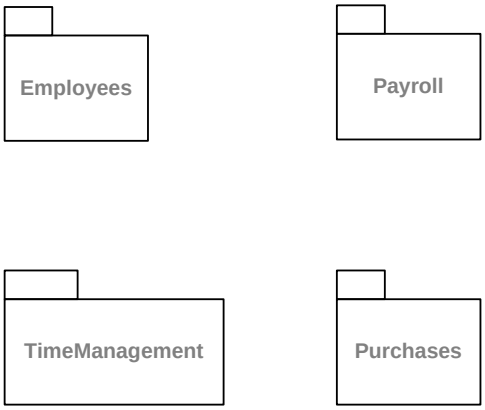
Εικόνα 11: Κατηγορήματα ιεραρχίας κλάσεων Employee

Συνεχίζοντας με την Εικόνα 11 παρατηρούμε ότι στην κλάση Employee υπάρχει ένα πεδίο με όνομα EmplTimeCard. Το πεδίο αυτό μοντελοποιεί την απαίτηση ότι κάθε Employee έχει μια κάρτα. Αντίστοιχα, το πεδίο PurchaseOrder μοντελοποιεί τη σχέση μεταξύ CommissionedEmployee και PurchaseOrder. Ο σωστός τρόπος για να παρουσιάσουμε αυτές τις σχέσεις στη φάση της ανάλυσης του πεδίου προβλήματος είναι να προσδιορίσουμε μεταξύ των κλάσεων σχέσεις συσχέτισης (association). Επομένως, η Εικόνα 11 θα πρέπει να τροποποιηθεί ώστε να παρουσιάζονται οι σχέσεις συσχέτισης.



Εικόνα 12: Απεικόνιση συσχετίσεων στο μοντέλο κλάσεων

Από την παραπάνω ανάλυση προκύπτουν με απόλυτο φυσικό τρόπο τα πακέτα κλάσεων του υπ'ανάπτυξη συστήματος. Οι κλάσεις ενός συστήματος είναι χρήσιμο να οργανώνονται σε πακέτα, τόσο για λόγους καλύτερου χειρισμού της πολυπλοκότητας όσο και για λόγους καλύτερης κατανόησης. Τα πακέτα προσφέρουν τη δυνατότητα ομαδοποίησης συναφών, υπό κάποια έννοια, κλάσεων μέσω της τοποθέτησης αυτών των κλάσεων στο ίδιο πακέτο. Επίσης, παρέχουν τη δυνατότητα ελέγχου των εξαρτήσεων αφού μπορούμε να περιορίσουμε την πρόσβαση στα πεδία του πακέτου εκθέτοντας μόνο ένα υποσύνολό τους. Για παράδειγμα, τα πεδία με πρόσβαση πακέτου (protected), δεν είναι ορατά έξω από το πακέτο τους. Τέλος, τα πακέτα παρέχουν ένα χώρο ονομάτων, αφού είναι δυνατόν να έχουμε σε διαφορετικά πακέτα στοιχεία με το ίδιο όνομα χωρίς να προκαλείται σύγχυση. Στην Εικόνα 13 παρουσιάζονται τα πακέτα του συστήματος μισθοδοσίας.

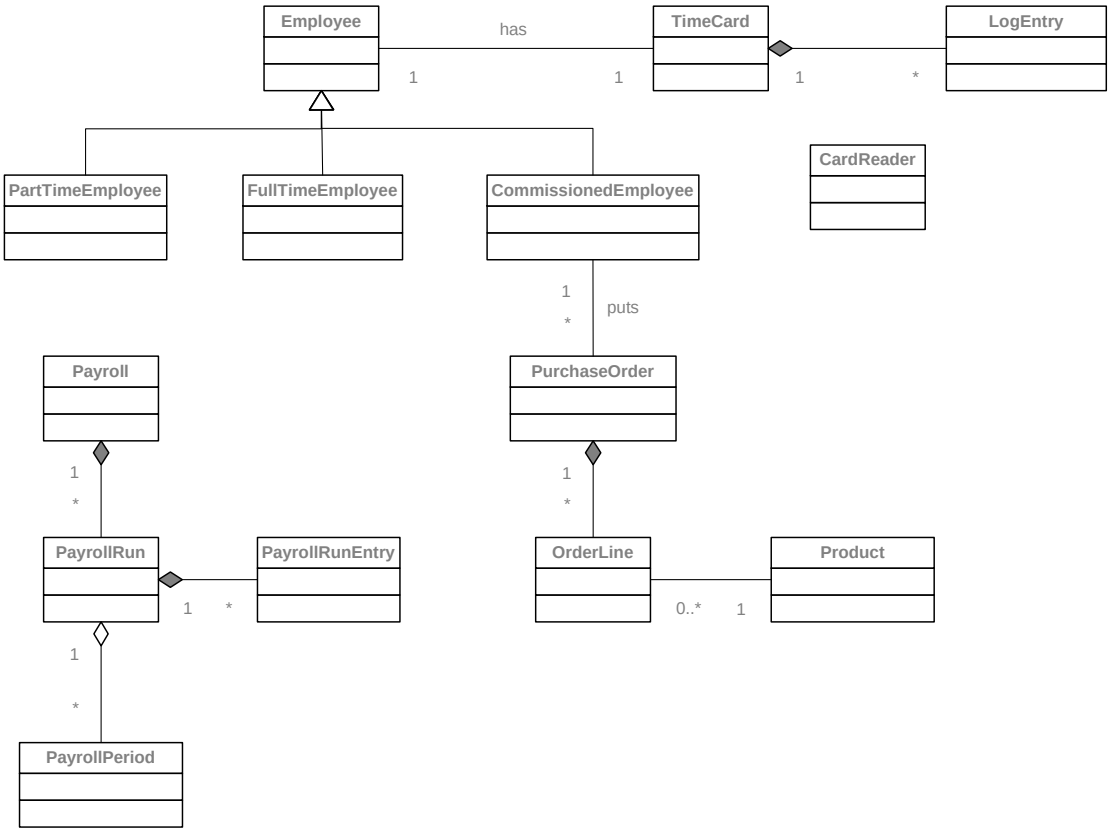


Εικόνα 13: Πακέτα συστήματος μισθοδοσίας.

Στο συγκεκριμένο παράδειγμα οι κλάσεις που ανήκουν στο κάθε πακέτο είναι:

- Πακέτο Employees: Employee, PartTimeEmployee, FullTimeEmployee, CommissionedEmployee.
- Πακέτο TimeManagement: TimeCard, CardReader, LogEntry.
- Πακέτο Payroll: Payroll, PayrollRun, PayrollPeriod, PayrollRunEntry.
- Πακέτο Purchases: PurchaseOrder, OrderLine, Product.

Στο σημείο αυτό μπορούμε να παραθέσουμε το συνολικό μοντέλο κλάσεων του συστήματος μισθοδοσίας όπως αυτό προκύπτει από την ανάλυση του πεδίου προβλήματος.



Εικόνα 14: Μοντέλο κλάσεων πεδίου προβλήματος

Το παραπάνω μοντέλο θα επεκταθεί στη συνέχεια με σχέσεις, πεδία, μεθόδους και νέες κλάσεις έτσι ώστε να μπορέσει να καλύψει πλήρως τις ανάγκες της παρούσας μελέτης περίπτωσης.

4.3.2 Σχεδιασμός πρωτότυπων του συστήματος

Ένας *δεύτερος βασικός στόχος του πρώτου βήματος* είναι η δημιουργία απλών πρωτοτύπων του συστήματος. Πολλές φορές κατά τη διάρκεια της ανάλυσης των απαιτήσεων, παρουσιάζονται αμφιβολίες σχετικά με τον τρόπο λειτουργίας του συστήματος, ή τις τεχνικές επιλογές. Για τη διερεύνηση λοιπόν των απαιτήσεων αλλά και για τον έλεγχο εναλλακτικών τεχνικών λύσεων χρησιμοποιείται η τεχνική της πρωτοτυποποίησης. Γενικότερα, η πρωτοτυποίηση μπορεί να είναι διερευνητική ή εξελικτική. Το διερευνητικό πρωτότυπο ή πρωτότυπο μιας χρήσης (throw-away prototype) είναι λογισμικό που αναπτύσσεται για τη διερεύνηση κάποιων από τα στοιχεία του συστήματος, της αποδοτικότητας μιας λύσης, την ελκυστικότητα μιας προσέγγισης κ.ο.κ. Επειδή ένα διερευνητικό πρωτότυπο επικεντρώνεται σε ένα συγκεκριμένο πρόβλημα ή

θέμα, με σκοπό του την εξαγωγή συμπερασμάτων, το ίδιο πρωτότυπο δεν μπορεί να χρησιμοποιηθεί ξανά σε επόμενη φάση της ανάπτυξης και είναι άχρηστο μετά το τέλος του πειράματος. Σε αντιδιαστολή, ένα εξελικτικό πρωτότυπο (evolutionary prototype) θα χρησιμοποιηθεί ξανά και σε επόμενη φάση της ανάπτυξης του συστήματος.

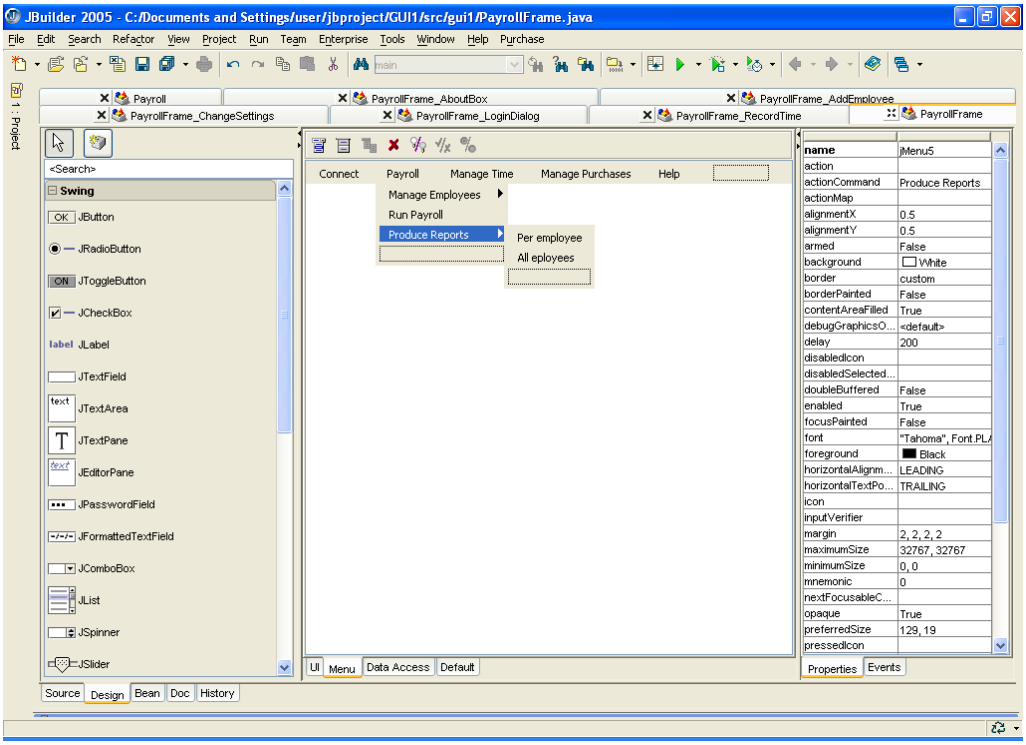
Αντίστοιχα, ο Constantine και Lockwood περιγράφουν τις βασικές αρχές πρωτοτυποποίησης της γραφικής διεπαφής του χρήστη. Μερικές από αυτές είναι οι παρακάτω:

- Η δομή της εφαρμογής θα πρέπει να είναι ξεκάθαρη (structure principle).
- Ο σχεδιασμός θα πρέπει να είναι απλός ώστε να απλοποιούνται οι εργασίες (simplicity principle).
- Θα πρέπει να είναι διαθέσιμη όλη η απαραίτητη πληροφορία και μόνο (visibility principle).
- Θα πρέπει να υπάρχει προσκόλληση στα πρότυπα.
- Θα πρέπει να υπάρχει συνέπεια σε όλο τον σχεδιασμό.
- Θα πρέπει να καλύπτονται όλες οι βασικές απαιτήσεις καθώς και τα δύσκολα σημεία.

Για την κατασκευή του πρωτότυπου θα πρέπει να χρησιμοποιήσουμε ένα γραφικό εργαλείο σχεδιασμού διεπαφής χρήστη. Τέτοια εργαλεία αποτελούν αναπόσπαστο τμήμα ολοκληρωμένων εργαλείων ανάπτυξης (IDE – Integrated Development Environment).

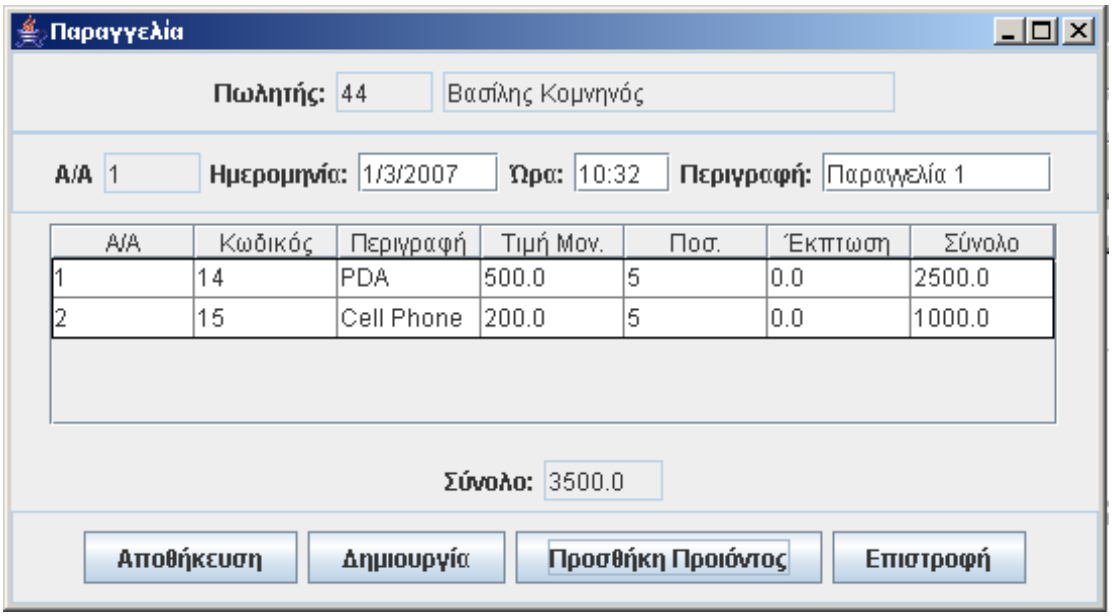
Τα IDE μας επιτρέπουν με γραφικό και εύκολο τρόπο να ορίσουμε χρησιμοποιώντας έτοιμα συστατικά (components) τη γραφική διεπαφή του συστήματος.

Για παράδειγμα, χρησιμοποιώντας το εργαλείο JBuilder μπορούμε να ορίσουμε τη δομή του μενού της εφαρμογής.



Εικόνα 15: Κατασκευή πρωτότυπου μενού εφαρμογής

Με αντίστοιχο τρόπο σχεδιάζουμε τις βασικές οθόνες του συστήματος έχοντας ως βασικό στόχο να δώσουμε στους χρήστες μια αρχική εικόνα του συστήματος, ενώ ταυτόχρονα επεξηγούμε τα δύσκολα σημεία.



Εικόνα 16: Διαχείριση παραγγελιών

4.4 Φάση 1- Βήμα 2. Σχεδίαση περιπτώσεων χρήσης

Το δεύτερο βήμα της πρώτης φάσης της μεθοδολογίας ICONIX *περιλαμβάνει* τον εντοπισμό και την καταγραφή, στη μεγαλύτερη δυνατή λεπτομέρεια, όλων των δυνατών ενεργειών των χρηστών και των αντίστοιχων αποκρίσεων του συστήματος.

4.4.1 Οι περιπτώσεις χρήσης

Το πρώτο βήμα για την ανάπτυξη του μοντέλου των Περιπτώσεων Χρήσης (ΠΧ) είναι η εύρεση των χειριστών (actors) του συστήματος. Ως χειριστή ορίζουμε ένα χρήστη του συστήματος με συγκεκριμένο ρόλο ή μια εξωτερική οντότητα του συστήματος η οποία αλληλεπιδρά με αυτό. Έτσι εκτός από χρήστες, οι χειριστές μπορεί να είναι άλλα συστήματα ή εξωτερικές συσκευές.

Για την εύρεση των χειριστών ανατρέχουμε στις απαιτήσεις όπως αυτές περιγράφονται στον ορισμό του προβλήματος. Μετά από μελέτη του ορισμού του προβλήματος εντοπίσαμε τους παρακάτω χειριστές:

- Employee
- FullTimeEmployee
- PartTimeEmployee
- CommissionedEmployee
- PayrollAdministrator

Όταν προσδιορίζουμε του χειριστές του συστήματος, υπάρχει πάντα ο προβληματισμός αν και κατά πόσο η λίστα είναι πλήρης, ή αν όλοι οι χειριστές που έχουν προσδιορισθεί είναι απαραίτητοι στη μοντελοποίηση του συστήματος κ.λπ. Για να απαντήσουμε σε αυτά τα ερωτήματα θα πρέπει να προσδιορίσουμε τις περιπτώσεις χρήσης στις οποίες συμμετέχει ο κάθε χειριστής και στη συνέχεια να ελέγξουμε κατά πόσο ικανοποιούνται όλες οι απαιτήσεις του συστήματος.

Προχωρούμε λοιπόν στο επόμενο βήμα το οποίο είναι η δημιουργία του διαγράμματος των περιπτώσεων χρήσης του συστήματος. Το διάγραμμα περιπτώσεων

χρήσης παρουσιάζει με γραφικό τρόπο, το πώς χρησιμοποιείται το σύστημα από τους χειριστές. Κάθε διακριτή χρήση του συστήματος καλείται περίπτωση χρήσης και ξεκινάει μετά από απαίτηση κάποιου χειριστή.

Για την εύρεση των περιπτώσεων χρήσης πρέπει να σκεφτούμε τους στόχους και τις απαιτήσεις του κάθε χρήστη από το σύστημα. Για κάθε έναν από τους διακριτούς στόχους θα πρέπει να καθορίσουμε πώς το σύστημα βοηθάει το χρήστη να τους εκπληρώσει και ποιες ενέργειες θα πρέπει να κάνει ο ίδιος.

Στο σημείο αυτό θα πρέπει να τονίσουμε ότι το μοντέλο των περιπτώσεων χρήσης αποτελείται:

Α) από το διάγραμμα περιπτώσων χρήσης το οποίο δίνει την εποπτική εικόνα του συστήματος καθώς και τις σχέσεις μεταξύ των περιπτώσεων χρήσης.

Β) από τη λεκτική περιγραφή των περιπτώσεων χρήσης, όπου περιγράφεται με λεπτομέρεια η λογική της κάθε περίπτωσης χρήσης.

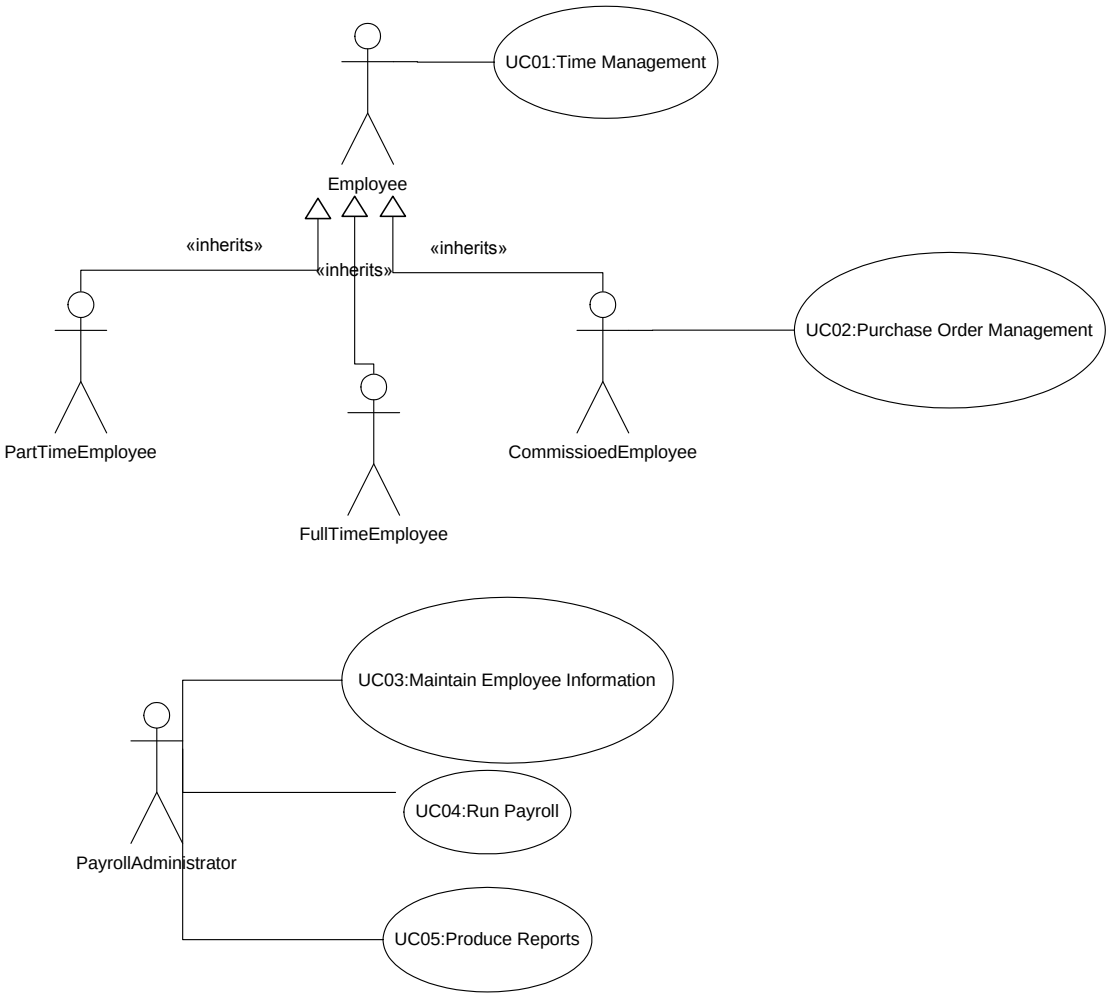
Στην Εικόνα 17 παρουσιάζεται το διάγραμμα περιπτώσεων χρήσης για το σύστημα μισθοδοσίας.

Η ΠΧ «Time Management» (Διαχείριση Χρόνου) σχετίζεται με όλους τους *Employees* (εργαζομένους). Όπως θα παρατηρήσατε ο χειριστής *Employee* συνδέεται με σχέση κληρονομικότητας ή σχέση γενίκευσης με τους χειριστές *PartTimeEmployee*, *FullTimeEmployee* και *CommissionedEmployee*. Η δημιουργία σχέσεων κληρονομικότητας μεταξύ των χειριστών απλοποιεί το μοντέλο των περιπτώσεων χρήσης αφού ομαδοποιεί την κοινή συμπεριφορά διαφορετικών ρόλων του συστήματος.

Η ΠΧ «Puchase Order Management» (Διαχείριση Εντολών Πωλήσεων) σχετίζεται αποκλειστικά με το χειριστή *CommissionedEmployee* (Πωλητής).

Τέλος, ο χειριστής *PayrollAdministrator* (Διαχειριστής Μισθοδοσίας) εκτελεί τις ΠΧ:

- Maintain Employee Information (Διαχείριση Εργαζομένων)
- Run Payroll (Εκτέλεση Μισθοδοσίας)
- Produce Reports (Παραγωγή αναφορών)



Εικόνα 17: Το διάγραμμα περιπτώσεων χρήσης

4.4.1.1 Προβληματισμοί σχετικοί με το μοντέλο περιπτώσεων χρήσης

1. *Οι περιπτώσεις χρήσης που παρουσιάζονται είναι υψηλού επιπέδου. Σε ποιο επίπεδο λεπτομέρειας θα πρέπει να εργαζόμαστε;*

   Ο αριθμός καθώς και η λεπτομέρεια στην οποία θα παρουσιάζονται οι περιπτώσεις χρήσης αποτελεί θέμα συζήτησης σε όλες τις ομάδες ανάπτυξης λογισμικού. Το βασικό ερώτημα είναι ποιο είναι το σωστό επίπεδο λεπτομέρειας που θα πρέπει να παρουσιάσω τις περιπτώσεις χρήσης;

   Θα πρέπει να έχω μια περίπτωση χρήσης υψηλού επίπεδου, όπως η ΠΧ «Maintain Employee Information», ή θα πρέπει να ορίσω ΠΧ με μεγαλύτερη λεπτομέρεια,

δηλαδή να αναλύσω την ΠΧ «Maintain Employee Information» σε υποπεριπτώσεις.

Η γενική απάντηση στην ερώτηση αυτή είναι ότι υπάρχουν διαφορετικά επίπεδα λεπτομέρειας, το καθένα εκ των οποίων ικανοποιεί διαφορετικούς στόχους.

Ο Cockburn στο βιβλίο του «Writing Effective Use Cases» αναφέρει ότι υπάρχουν τρία επίπεδα λεπτομέρειας:

1. Το συνοπτικό επίπεδο (summary level). Στο επίπεδο αυτό έχουμε πολύ μικρό αριθμό ΠΧ οι οποίες περιγράφουν τους βασικούς στόχους του συστήματος (system context).
2. Το επίπεδο των στόχων των χρηστών όπου περιγράφονται οι επιμέρους στόχοι των χρηστών (user goals).
3. Το επίπεδο διακριτών λειτουργιών (function level) , το οποίο μας επιτρέπει να περιγράψουμε συμπεριφορές πολύ χαμηλού επιπέδου σε επίπεδο λειτουργίας.

Ο βασικός μας στόχος θα πρέπει να είναι να περιγράψουμε αναλυτικά και με σαφήνεια τις ΠΧ που παρουσιάζουν τους στόχους των χρηστών.

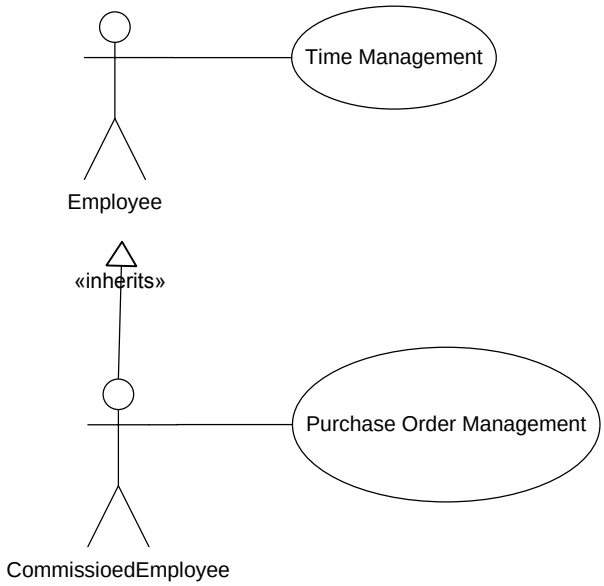
Επιπλέον, η διαδικασία του προσδιορισμού των ΠΧ περνάει από στάδια εκλέπτυνσης. Το μοντέλο των ΠΧ δημιουργείται βήμα-βήμα.

Το αρχικό βήμα είναι ο προσδιορισμός των χειριστών καθώς και των στόχων που έχει ο κάθε χειριστής. Στη συνέχεια προσδιορίζουμε το βασικό σενάριο για το οποίο δημιουργούμε την περίπτωση χρήσης. Ακολουθεί ο προσδιορισμός των συνθηκών αποτυχίας (failure conditions) της ΠΧ καθώς και ο τρόπος αντιμετώπισης των αποτυχιών (failure handling).

2. *Οι χειριστές PartTimeEmployee και FullTimeEmployee δεν εκτελούν καμία περίπτωση χρήσης. Θα πρέπει να εμφανίζονται στο μοντέλο;*

Οι χειριστές εκτός από το ερέθισμα (stimulus) που παράγουν ώστε να αρχίσει η εκτέλεση μιας ΠΧ, περιγράφουν τους ρόλους που υπάρχουν μέσα σε ένα σύστημα. Για το λόγο αυτό, αν και από άποψη λειτουργικότητας, το διάγραμμα της Εικόνα 18 είναι ισοδύναμο με αυτό της Εικόνα 17, η Εικόνα 17 περιέχει περισσότερη πληροφορία, αφού προσδιορίζει και τους ρόλους που αντιπροσωπεύουν οι

χειριστές *PartTimeEmployee* και *FullTimeEmployee.*



Εικόνα 18: Εναλλακτική λύση διαγράμματος περιπτώσεων χρήσης

3. *Θα πρέπει να εμφανίζεται ως χειριστής η συσκευή CardReader (αναγνώστης καρτών) αφού αλληλεπιδρά με το σύστημα;*

   Η συσκευή CardReader είναι μια απλή συσκευή εισόδου, μια απλή περιφερειακή συσκευή όπως το πληκτρολόγιο, το ποντίκι κ.α. και συνεπώς δεν είναι χειριστής.

   Εναλλακτικά, θα μπορούσαμε να πούμε ότι η συσκευή είναι δευτερεύων χειριστής στην ΠΧ TimeManagement για να καλύψουμε την περίπτωση που η συσκευή δεν είναι διαθέσιμη. Στην περίπτωση αυτή, θα πρέπει να υπάρχει μια ΠΧ επέκτασης CardReaderProblem στην ΠΧ TimeManagement.

4. *Θα πρέπει να εμφανίζεται ως χειριστής η βάση δεδομένων αφού είναι εξωτερικό σύστημα; Θα πρέπει να υπάρχει περίπτωση χρήσης «Αποθήκευση στη Βάση Δεδομένων»;*

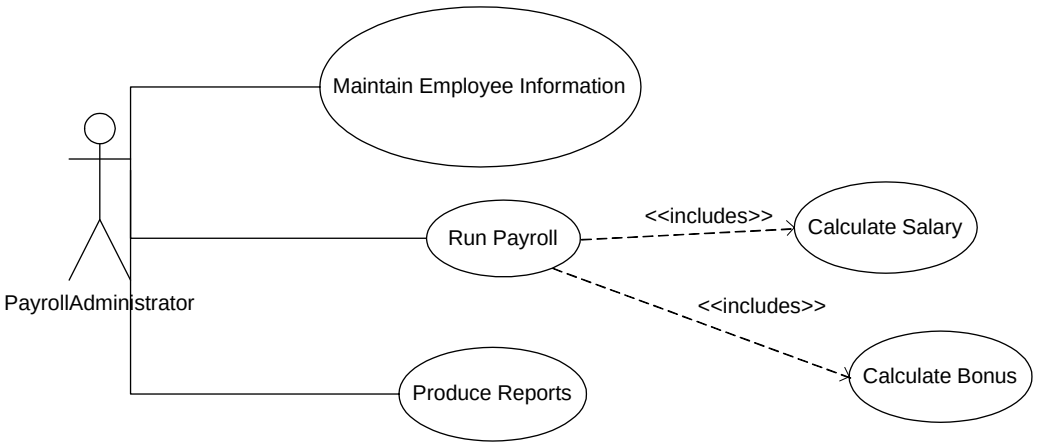
   Στο σημείο αυτό θα πρέπει να διευκρινίσουμε εάν η βάση δεδομένων αποτελεί μέρος του συστήματος ή όχι. Αν η βάση δεδομένων χρησιμοποιείται αποκλειστικά από το σύστημα μισθοδοσίας τότε μπορούμε να πούμε ότι αποτελεί μέρος του συστήματος, σε αντίθετη περίπτωση θα πρέπει να εμφανίζεται ως εξωτερικό

σύστημα δηλαδή να είναι χειριστής.

Σχετικά με το δεύτερο ερώτημα, εάν θα πρέπει να υπάρχει ΠΧ «Αποθήκευση στη Βάση Δεδομένων», η απάντηση είναι αρνητική. Αντί για αυτή την ΠΧ, θα μπορούσαμε να έχουμε μια ΠΧ με τίτλο «Αποθήκευση των στοιχείων εργαζομένου στη Βάση Δεδομένων», η οποία είναι συγκεκριμένη και ξεκινά από την απαίτηση του χειριστή να αποθηκεύσει δεδομένα. Ποτέ όμως, εκτός ίσως από την περίπτωση δημιουργία αντιγράφων ασφαλείας (backup), ο χειριστής δεν ζητά την καθολική αποθήκευση των δεδομένων όλου του συστήματος, και για το λόγο αυτό τέτοια ΠΧ δε θα πρέπει να υπάρχει.

5. *Η ΠΧ «Run Payroll» θα έπρεπε να αναλύεται περισσότερο αφού ο υπολογισμός της μισθοδοσίας είναι πολύπλοκος και εξαρτάται από την κατηγορία του κάθε εργαζομένου.*

Όντως, θα μπορούσαμε να αναλύσουμε περαιτέρω την ΠΧ «Run Payroll», σε επιμέρους ΠΧ που θα συνδέονταν με τη βασική ώστε για κάθε κατηγορία εργαζομένου να υπάρχει διαφορετική περίπτωση χρήσης. Η περίπτωση αυτή παρουσιάζεται στην Εικόνα 19, όπου η ΠΧ «Run Payroll» περιλαμβάνει (includes) τις ΠΧ «Calculate Salary» και «Calculate Bonus». Αυτό σημαίνει ότι για να ολοκληρωθεί η ΠΧ «Run Payroll» θα πρέπει να εκτελεσθεί η λογική, τόσο της ΠΧ «Calculate Salary», όσο και της ΠΧ «Calculate Bonus».



Εικόνα 19: Σχέση include μεταξύ περιπτώσεων χρήσης

Αποφύγαμε να ακολουθήσουμε αυτή την προσέγγιση γιατί:

- Ο απαιτούμενος υπολογισμός δεν είναι σύνθετος.
- Η ΠΧ αντιστοιχεί σε μια κλάση πράγμα που απλοποιεί το μοντέλο μας.

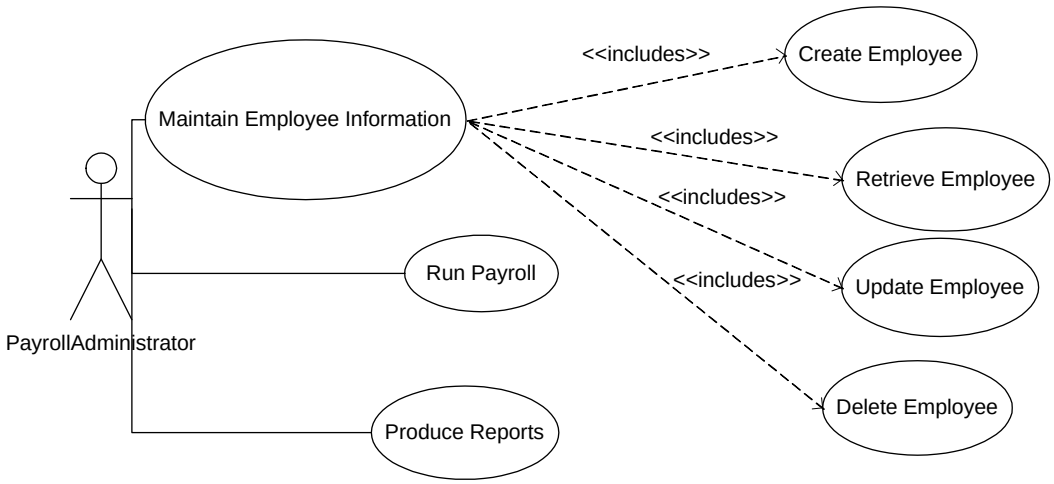
6. *Θα μπορούσαμε να αναλύσουμε παραπάνω την ΠΧ «Maintain Employee Information» σε εισαγωγή, τροποποίηση, διαγραφή εργαζομένου;*

   Η ΠΧ *«Maintain Employee Information»* θα μπορούσε να αναλυθεί με μεγαλύτερη λεπτομέρεια, εισάγοντας ΠΧ για κάθε μια από τις παραπάνω περιπτώσεις. Για τη δημιουργία αυτών των ΠΧ, υπάρχουν δύο εναλλακτικές περιπτώσεις.

   Α) Να αντικαθιστούσαμε την ΠΧ *«Maintain Employee Information»* με τις ΠΧ: «Create Employee», «Retrieve Employee», «Update Employee» και «Delete Employee».

   Β) Η ΠΧ *«Maintain Employee Information»* κάνει include τις περιπτώσεις χρήσης «Create Employee», «Retrieve Employee», «Update Employee» και «Delete Employee».

   Γενικά, ΠΧ όπως η *«Maintain Employee Information»* είναι πολύ συνηθισμένες, υπάρχουν σε όλα τα συστήματα και ονομάζονται ΠΧ CRUD (Create, Retrieve, Update, Delete).



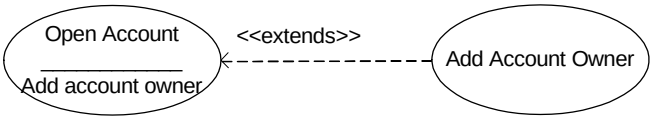
Εικόνα 20: Περίπτωση χρήσης CRUD

7. *Στην* Εικόνα 20 *χρησιμοποιείται μεταξύ των ΠΧ σχέση <<include>>. Θα μπορούσε να χρησιμοποιηθεί σχέση «extend»;*

   Η σχέση extend χρησιμοποιείται για να εξειδικεύσουμε μια βασική περίπτωση ΠΧ.

Το βασικό κριτήριο είναι ότι η βασική ΠΧ μπορεί να εκτελεσθεί ανεξάρτητα από την εξιδικεύουσα ΠΧ.

Για παράδειγμα, η ΠΧ «Open Account» είναι ανεξάρτητη από την ΠΧ «Add Account Owner». Όμως, όταν ο λογαριασμός ανοίγει από περισσότερους από ένα πελάτη, θα πρέπει να εκτελέσουμε επιπλέον ενέργειες-ελέγχους και συνεπώς χρειαζόμαστε την ΠΧ «Add Account Owner». Επίσης, θα μπορούσαμε να θεωρήσουμε ότι η εξειδικευμένη ΠΧ είναι μια εναλλακτική ροή της βασικής ροής της ΠΧ.



Εικόνα 21: Σχέση μεταξύ περιπτώσεων χρήσης τύπου extend

Αντίθετα, η σχέση include δεν αποτελεί μια ανεξάρτητη ΠΧ αλλά αποτελεί μέρος μιας μεγαλύτερης περίπτωσης χρήσης. Γενικά, η συμπερίληψη (include) είναι μία ειδική περίπτωση σχέσης, στην οποία σχετίζουμε δύο περιπτώσεις χρήσης, όπου η μία περίπτωση χρήσης συμπεριλαμβάνει/περιέχει την άλλη. Η έννοια της συμπερίληψης είναι υποχρεωτική, δηλαδή πάντα η μία περίπτωση χρήσης θα συμπεριλαμβάνει την άλλη. Χρησιμοποιούμε τη συμπερίληψη όταν η ίδια λειτουργικότητα περιλαμβάνεται σε περισσότερες από μία περιπτώσεις χρήσης.

Δραστηριότητα 2

Η Εικόνα 22 παρουσιάζει τις ΠΧ ενός συστήματος Διαχείρισης Έργων. Μελετήστε το διάγραμμα και απαντήστε στις ερωτήσεις:

1) Προσδιορίστε τους χειριστές και επεξηγήστε τις σχέσεις μεταξύ των ΠΧ
2) Ενημερώστε το διάγραμμα ΠΧ ώστε να συμπεριλάβει τις παρακάτω απαιτήσεις:
    a. Οι εργαζόμενοι (human resource) διαχειρίζονται το πλάνο επαγγελματικής ανάπτυξης (development plan). Η διαχείριση του πλάνου περιλαμβάνει τη διαχείριση του προφίλ του εργαζομένου, την πρόσβαση στο πρόγραμμα εκπαιδεύσεων εργαζομένων που είναι αποθηκευμένο σε βάση δεδομένων. Το πλάνο επαγγελματικής ανάπτυξης εργαζομένου θα πρέπει να αποθηκεύεται σε βάση

δεδομένων, η οποία δεν αποτελεί τμήμα του συστήματος διαχείρισης έργων.

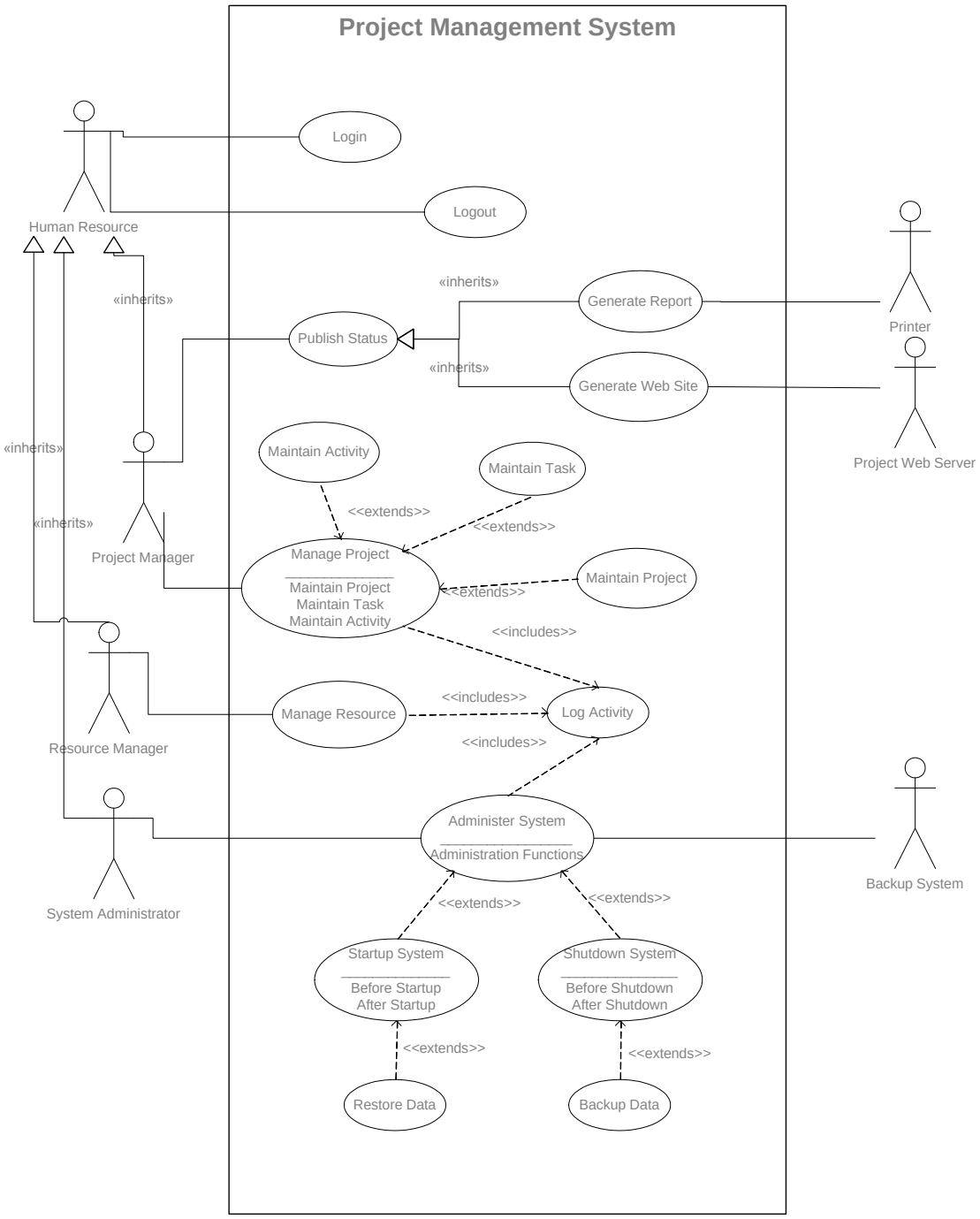
b. Ο διαχειριστής πόρων (resource manager) θα πρέπει να έχει πρόσβαση σε λειτουργικότητα που θα επιτρέπει τη διαχείριση πόρων. Μια επιλογή είναι η διαχείριση του προφίλ του πόρου. Η ίδια λειτουργικότητα θα πρέπει να είναι διαθέσιμη και στο διαχειριστή ανθρωπίνων πόρων.

c. Ο διαχειριστής έργου (project manager) ή ο διαχειριστής του συστήματος (administrator), θα πρέπει να είναι σε θέση να αποστέλλει e-mail. Το σύστημα e-mail δεν είναι μέρος του συστήματος διαχείρισης έργων. Όταν αποστέλλουμε e-mail θα πρέπει να έχουμε τη δυνατότητα της ασφαλούς αποστολής, με τη χρήση κρυπτογραφικού μηχανισμού, μηχανισμό τον οποίο θα προμηθευτούμε από εξειδικευμένη εταιρεία.

d. Το σύστημα e-mail όταν παραλαμβάνει e-mail ενημερώνει τους χρήστες. Ο χρήστης του συστήματος μπορεί να ελέγχει αν έχει παραληφθεί καινούργιο e-mail.

e. Όταν αποστέλλουμε ή παραλαμβάνουμε e-mail το σύστημα θα καταγράφει τη συναλλαγή.

3) Περιγράψτε τη σειρά – προτεραιότητα - με την οποία θα πρέπει να αναπτύξουμε τις ΠΧ ανεξάρτητα από τις απαιτήσεις των χρηστών.



Εικόνα 22: Παράδειγμα περιπτώσεων χρήσης «Σύστημα Διαχείρισης Έργων»

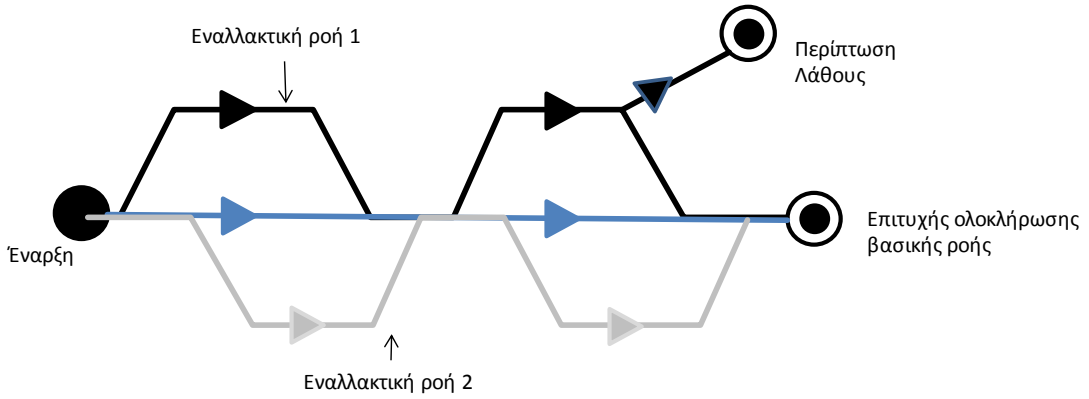
[53]

4.4.2 Η αναλυτική περιγραφή των περιπτώσεων χρήσης

Η λογική υλοποίηση των ΠΧ περιγράφεται χρησιμοποιώντας το πρότυπο της αναλυτικής περιγραφής των ΠΧ. Η λεκτική περιγραφή προσδιορίζει την αλληλεπίδραση των χειριστών με το σύστημα δίνοντας όλες τις εναλλακτικές περιπτώσεις – σενάρια που μπορούν να συμβούν κατά την εκτέλεση μιας ΠΧ.

Μια ΠΧ πρέπει να έχει μια συγκεκριμένη αρχή καθώς και ένα συγκεκριμένο τέλος. Από όλες τις εναλλακτικές ροές– σενάρια - μια χαρακτηρίζεται ως η βασική ροή ή το βασικό σενάριο.

Στη Εικόνα 23 παρουσιάζεται μια ΠΧ με μια βασική ροή, δύο εναλλακτικά σενάρια και μια περίπτωση λάθους.



Εικόνα 23: Βασική ροή και εναλλακτικές ροές μιας περίπτωσης χρήσης

Στη συνέχεια θα παρουσιάσουμε τις αναλυτικές περιγραφές δύο ΠΧ του παραδείγματός μας. Οι ΠΧ είναι:

- Εκτέλεση Μισθοδοσίας (Run Payroll) και
- Διαχείριση Παραγγελίας (Purchase Order management)

Τίτλος περίπτωσης χρήσης:
UC04:Εκτέλεση μισθοδοσίας (run payroll)
Σύντομη περιγραφή:
Ο διαχειριστής μισθοδοσίας τρέχει τη μισθοδοσία για μια περίοδο

Χειριστές:

Διαχειριστής μισθοδοσίας (payroll administrator)

Ροή γεγονότων

Βασική ροή

1. Ο διαχειριστής μισθοδοσίας επιλέγει «Εκτέλεση Μισθοδοσίας» από την κεντρική οθόνη του συστήματος
2. Επιλέγει την περίοδο για την οποία θα εκτελέσει τη μισθοδοσία
3. Το σύστημα υπολογίζει τη μισθοδοσία εργαζομένων
4. Το σύστημα υπολογίζει τη μισθοδοσία πωλητών
5. Το σύστημα υπολογίζει το bonus πωλητών
6. Το σύστημα υπολογίζει τη μισθοδοσία ωρομισθίων

Εναλλακτικές ροές

Εναλλακτική ροή 1

3α. Δεν υπάρχουν αποθηκευμένα στοιχεία χρόνου για τους εργαζόμενους

3β. Παρουσιάζεται ενημερωτικό μήνυμα

Εναλλακτική ροή 2

4α. Δεν υπάρχουν αποθηκευμένα στοιχεία χρόνου για τους πωλητές

4β. Παρουσιάζεται ενημερωτικό μήνυμα

Εναλλακτική ροή 3

5α. Δεν υπάρχουν αποθηκευμένα στοιχεία χρόνου για τους εργαζόμενους

5β. Παρουσιάζεται ενημερωτικό μήνυμα

Εναλλακτική ροή 4

6α. Δεν υπάρχουν αποθηκευμένες παραγγελίες για τους πωλητές

6β. Παρουσιάζεται ενημερωτικό μήνυμα

Ειδικές απαιτήσεις:

Δεν υπάρχουν

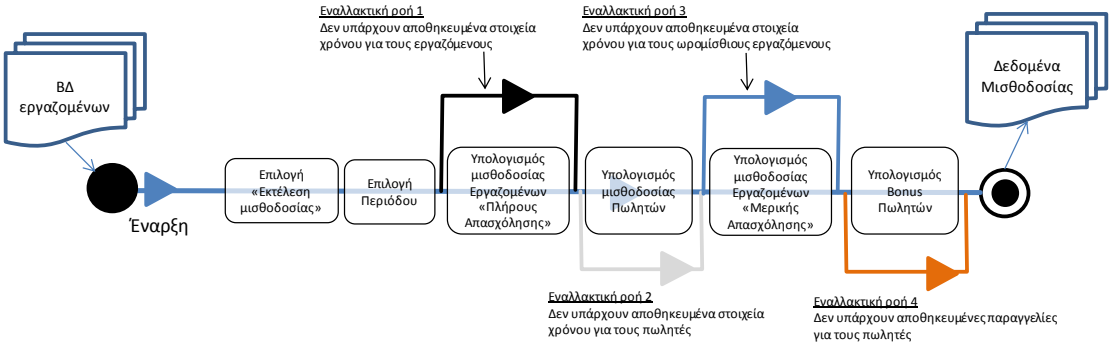
Κατάσταση εισόδου:

Να υπάρχει αρχείο/βάση δεδομένων εργαζομένων

Κατάσταση εξόδου:

Ενημέρωση εγγραφών μισθοδοσίας

Αν προσπαθούσαμε να παρουσιάσουμε με γραφικό τρόπο την ΠΧ θα είχαμε:



Εικόνα 24: Γραφική απεικόνιση περίπτωσης χρήσης υπολογισμού μισθοδοσίας

Συνεχίζουμε με την ΠΧ Διαχείριση Παραγγελίας. Θα πρέπει να μελετήσετε την αναλυτική περιγραφή της ΠΧ σε συνδυασμό με την Εικόνα 16 όπου παρουσιάζεται το πρωτότυπο της οθόνης του συστήματος.

Τίτλος περίπτωσης χρήσης:

UC02:Διαχείριση Παραγγελίας (Purchase Order Management)

Σύντομη περιγραφή:

Η ΠΧ αφορά την εισαγωγή μιας ή περισσότερων παραγγελιών για έναν Πωλητή.

Χειριστές: Πωλητής

Ροή γεγονότων

Βασική ροή

1. Η εφαρμογή εμφανίζει μια φόρμα για την εισαγωγή των στοιχείων της

παραγγελίας. Η φόρμα περιέχει λειτουργικότητα για: «Αποθήκευση» Παραγγελίας, «Δημιουργία» Νέας Παραγγελίας, «Προσθήκη Προϊόντος», «Επιστροφή». Η λειτουργία «Αποθήκευση Παραγγελίας» επιτρέπει την αποθηκεύσει της τρέχουσας παραγγελίας. Η λειτουργία «Δημιουργία Νέας Παραγγελίας» ξεκινά τη δημιουργία μιας νέας παραγγελίας για τον τρέχοντα Πωλητή. Η λειτουργία «Προσθήκη Προϊόντος» ανοίγει μια νέα φόρμα για την προσθήκη των προϊόντων της τρέχουσας παραγγελίας. Τέλος, η λειτουργία «Επιστροφή» μας δίνει τη δυνατότητα να κλείσουμε τη φόρμα εισαγωγής.

2. Η εφαρμογή εμφανίζει στη φόρμα αυτόματα τον κωδικό και το ονοματεπώνυμο του πωλητή.

3. Η εφαρμογή εμφανίζει τον αύξοντα αριθμό της παραγγελίας. Ο αύξων αριθμός της παραγγελίας πρέπει να είναι μοναδικός για κάθε παραγγελία και η τιμή του δίνεται αυτόματα από την εφαρμογή. Η εφαρμογή εμφανίζει επίσης την τρέχουσα αξία της παραγγελίας η οποία εξαρτάται από τα προϊόντα που έχουν προστεθεί στην παραγγελία.

4. Ο πωλητής εισάγει στα αντίστοιχα πλαίσια κειμένου της φόρμας την ημερομηνία και την ώρα, καθώς και μια περιγραφή της παραγγελίας.

5. Ο πωλητής επιλέγει «Προσθήκη Προϊόντος».

6. Η εφαρμογή εμφανίζει μια νέα φόρμα που επιτρέπει δύο λειτουργίες: «Προσθήκη» και «Επιστροφή». Η λειτουργία «Προσθήκη» προσθέτει στη λίστα των προϊόντων της τρέχουσας παραγγελίας το προϊόν που έχει επιλεγεί. Η λειτουργία «Επιστροφή» μας δίνει τη δυνατότητα να κλείσουμε τη φόρμα των προϊόντων, χωρίς να κάνουμε προσθήκη κάποιου προϊόντος.

7. Ο πωλητής εισάγει στα αντίστοιχα πλαίσια κειμένου τον κωδικό προϊόντος και την ποσότητα.

8. Ο πωλητής επιλέγει τη λειτουργία «Προσθήκη», ώστε το τρέχον προϊόν να προστεθεί στη λίστα προϊόντων της τρέχουσας παραγγελίας.

9. Προσθήκη του προϊόντος στη τρέχουσα παραγγελία, και επιστροφή στην αρχική

φόρμα.

10. Η εφαρμογή εμφανίζει στην αρχική φόρμα τη τρέχουσα αξία της παραγγελίας, συμπεριλαμβανομένων και των εκπτώσεων, και παρουσιάζει σε μια γραμμή, σε κατάλληλο στοιχείο ελέγχου της φόρμας, για το τελευταίο προϊόν που προστέθηκε:

    α) την περιγραφή του

    β) την τιμή του

    γ) τον συντελεστή έκπτωσης (0%, 5% ή 10%)

Επαναλαμβάνονται τα βήματα 5-10 μέχρι να συμπληρωθεί η λίστα των προϊόντων της τρέχουσας παραγγελίας.

11. Ο πωλητής επιλέγει τη λειτουργία «Αποθήκευση Παραγγελίας»

12. Ο πωλητής επιλέγει στη βασική φόρμα τη λειτουργία «Δημιουργία Νέας Παραγγελίας» για να εισάγει για λογαριασμό του μια νέα παραγγελία.

13. Ο έλεγχος μεταφέρεται στο βήμα 3 της βασικής ροής.

Εναλλακτικές Ροές:

Εναλλακτική ροή 1

4α. Ο πωλητής επιλέγει στη βασική φόρμα «Επιστροφή» για να ακυρώσει την εισαγωγή παραγγελίας.

4β. Η φόρμα εισαγωγής κλείνει και η περίπτωση χρήσης για τον τρέχοντα πωλητή τερματίζει.

Εναλλακτική ροή 2

7α. Ο Πωλητής επιλέγει στη φόρμα των προϊόντων τη λειτουργία «Επιστροφή» για να ακυρώσει την προσθήκη προϊόντος.

7β. Η φόρμα κλείνει και ο έλεγχος μεταφέρεται στο τέλος του βήματος 10 της βασικής ροής.

Εναλλακτική ροή 3

9α. Εσφαλμένος κωδικός προϊόντος

9β. Η εφαρμογή εμφανίζει μήνυμα σφάλματος και απορρίπτει την εισαγωγή.
9γ. Ο έλεγχος μεταφέρεται στο βήμα 7 της βασικής ροής.
Εναλλακτική ροή 4
12α. Ο Πωλητής επιλέγει τη λειτουργία «Επιστροφή».
12β. Η φόρμα εισαγωγής κλείνει και η περίπτωση χρήσης για τον τρέχοντα πωλητή τερματίζει.
Κατάσταση εισόδου:
Ο πωλητής έχει ταυτοποιηθεί. Ο πίνακας προϊόντων υπάρχει στη ΒΔ.
Κατάσταση εξόδου:
Μία ή περισσότερες παραγγελίες έχουν εισαχθεί. Η αποθήκευση γίνεται σε δομές της εφαρμογής (αντικείμενα), αλλά όχι στη ΒΔ.
Ειδικές απαιτήσεις:
Όχι

4.4.2.1 Προβληματισμοί σχετικοί με την αναλυτική περιγραφή των περιπτώσεων χρήσης.

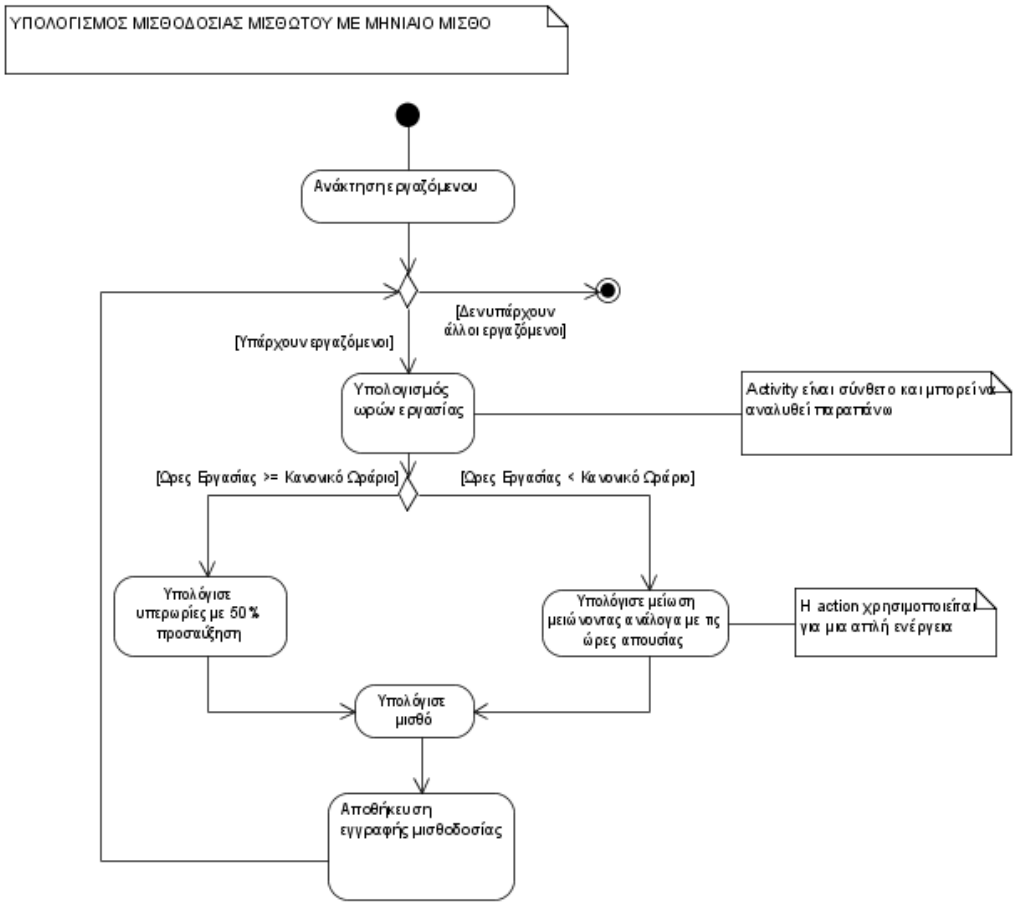
1. *Η ΠΧ «Υπολογισμός μισθοδοσίας» είναι πολύπλοκη. Πως θα μπορούσαμε να περιγράψουμε τη λογική του υπολογισμού της μισθοδοσίας;*

Η αναλυτική περιγραφή της λογικής μπορεί να γίνει με τη χρήση διαγραμμάτων δραστηριοτήτων (activity diagrams). Τα διαγράμματα δραστηριοτήτων περιγράφουν μια ακολουθία δραστηριοτήτων, μεταξύ των οποίων μπορούμε να έχουμε παράλληλες δραστηριότητες, συνθήκες κ.λπ.

Εκτός από την αναλυτική περιγραφή της λογικής των ΠΧ τα διαγράμματα δραστηριοτήτων συνήθως χρησιμοποιούνται και για την :

- Περιγραφή της ροής των εργασιών (workflow)
- Περιγραφή κάποιου πολύπλοκου αλγορίθμου
- Περιγραφή παράλληλων δραστηριοτήτων

[59]



Εικόνα 25: Διάγραμμα δραστηριότητας για τον υπολογισμό μισθοδοσίας μισθωτού

Δραστηριότητα 3

Φτιάξτε ένα διάγραμμα δραστηριοτήτων που περιγράφει ένα διαχειριστή έργου που τυπώνει μια αναφορά με τη χρήση ενός συστήματος διαχείρισης έργων. Προσδιορίστε τις δράσεις (actions) και τις μεταβάσεις ροής (flow transition).

---

2. *Πως εμφανίζεται η σχέση include μέσα στην αναλυτική περιγραφή των περιπτώσεων χρήσης;*

Εάν είχαμε επιλέξει να μοντελοποιήσουμε τις ΠΧ όπως παρουσιάζονται στην Εικόνα 19, τότε θα έπρεπε να τροποποιήσουμε την περίπτωση χρήσης «Εκτέλεση μισθοδοσίας»

(run payroll) ώστε στη βασική ροή να έχουμε:

Βασική ροή

1. Ο διαχειριστής μισθοδοσίας επιλέγει «Εκτέλεση Μισθοδοσίας» από την κεντρική οθόνη του συστήματος
2. Επιλέγει την περίοδο για την οποία θα εκτελέσει τη μισθοδοσία
3. Κάνουμε «Include» την ΠΧ «Calculate Salary» (Υπολογισμός Μισθού)
4. Κάνουμε «Include» την ΠΧ «Calculate Bonus» (Υπολογισμός Bonus)

3. *Πως θα περιγραφόταν η ΠΧ CRUD (βλέπε Εικόνα 20) στην αναλυτική περιγραφή των περιπτώσεων χρήσης;*

Στην Εικόνα 20, η ΠΧ «Maintain Employee Information» κάνει include τις ΠΧ «Create Employee», «Retrieve Employee», «Update Employee» και «Delete Employee». Στην περίπτωση αυτή η αναλυτική περιγραφή της ΠΧ «Maintain Employee Information» θα μπορούσε να έχει την παρακάτω δομή.

Βασική ροή

1. Κάνουμε «Include» την ΠΧ «Create Employee» (Δημιουργία Εργαζομένου)

Εναλλακτική ροή 1

1. Κάνουμε «Include» την ΠΧ «Retrieve Employee» (Ανάκτηση Στοιχείων Εργαζομένου)

Εναλλακτική ροή 2

1. Κάνουμε «Include» την ΠΧ «Update Employee» (Ενημέρωση Στοιχείων Εργαζομένου)

Εναλλακτική ροή 3

1. Κάνουμε «Include» την ΠΧ «Delete Employee» (Διαγραφή Εργαζομένου)

4.4.3 Χρήση περιπτώσεων χρήσης για υπολογισμό κόστους ανάπτυξης λογισμικού

Οι ΠΧ του συστήματος μπορούν να χρησιμοποιηθούν για τον υπολογισμό του

κόστους ανάπτυξης του συστήματος λογισμικού. Στη βιβλιογραφία έχουν προταθεί διάφορες εναλλακτικές μέθοδοι που χρησιμοποιούν ΠΧ για τον υπολογισμό του κόστους ανάπτυξης συστημάτων λογισμικού. Μεταξύ αυτών αξίζει να αναφέρουμε:

- συνδυασμός ΠΧ με σημεία λειτουργίας (function points) (Longstreet, 2005)
- συνδυασμός ΠΧ με γραμμές κώδικα LOC (Line of Code) (Smith, 1999)
- η μέθοδος των σημείων ΠΧ (use case points) (Karner, 1993).

Η μέθοδος υπολογισμού κόστους με τη χρήση σημείων περιπτώσεων χρήσης (Use Case Points - UCP) βασίζεται στη μέτρηση της πολυπλοκότητας των περιπτώσεων χρήσης και της αλληλεπίδρασης τους με τους χειριστές, με σκοπό να δώσει μια εκτίμηση σχετικά με το μέγεθος του υπ'ανάπτυξη συστήματος. Για τη μέτρηση των UCP ακολουθούμε τα παρακάτω βήματα:

1 Χαρακτηρίζουμε τον κάθε χειριστή ως απλό, μέτρια πολυπλοκότητας ή σύνθετο με σκοπό να υπολογίσουμε το αστάθμητο βάρος χειριστών (Unadjusted Actor Weight - UAW). Για τον υπολογισμό, μετρούμε τον αριθμό των απλών, μέτριας πολυπλοκότητας και σύνθετων χειριστών, πολλαπλασιάζουμε με το αντίστοιχο βάρος και αθροίζουμε τα γινόμενα.

Ταξινόμηση	**Παράδειγμα χειριστή**	**Βάρος**
Απλός	Ένα API καλά τεκμηριωμένο	1
Μέτριας πολυπλοκότητας	Ένα πρωτόκολλο όπως TCP/IP, HTTP, FTP ή ο χειριστής είναι βάση δεδομένων	2
Σύνθετος	Διεπαφή με άλλο σύστημα ή οποιαδήποτε διεπαφή με το χρήστη	3

2 Χαρακτηρίζουμε κάθε ΠΧ ως απλή, μέτριας πολυπλοκότητας ή σύνθετη με σκοπό να υπολογίσουμε το αστάθμητο βάρος ΠΧ (Unadjusted Use Case Weight - UUCW). Για τον

υπολογισμό, μετρούμε τον αριθμό των απλών, μέτριας πολυπλοκότητας και σύνθετων ΠΧ, πολλαπλασιάζουμε με το αντίστοιχο βάρος και αθροίζουμε τα γινόμενα.

Ταξινόμηση	**Παράδειγμα ΠΧ**	**Βάρος**
Απλή	<4 συναλλαγές	5
Μέτριας πολυπλοκότητας	≥ 4 και ≤ 7 συναλλαγών	10
Σύνθετη	≥ συναλλαγές	15

3 Υπολογίζουμε το συνολικό αριθμό αστάθμητων σημείων ΠΧ (Unadjusted Use Case Points – UUCP).

UUCP = UAW + UUCW

4 Υπολογίζουμε τους σταθμισμένους τεχνικούς παράγοντες και παράγοντες περιβάλλοντος. Κάθε παράγοντας παίρνει μια τιμή από 0 έως 5 ανάλογα με το βαθμό επίδρασης του παράγοντα στο έργο. Εάν ο παράγοντας βαθμολογηθεί με 0 σημαίνει ότι δεν επηρεάζει, ενώ τιμή 5 μεταφράζεται ότι ο παράγοντας είναι κρίσιμος για το έργο.

Παράγοντας	**Περιγραφή τεχνικών παραγόντων πολυπλοκότητας**	**Βάρος**
T1	Είναι κατανεμημένο σύστημα	2
T2	Υπάρχουν απαιτήσεις απόδοσης (performance) ή απόκρισης (response)	2
T3	Υπάρχουν απαιτήσεις για την αποδοτικότητα τελικών χρηστών	1
T4	Υπάρχει σύνθετη εσωτερική επεξεργασία	1

Παράγοντας	Περιγραφή τεχνικών παραγόντων πολυπλοκότητας	Βάρος
T5	Αναπτύσσεται επαναχρησιμοποιήσιμος κώδικας	1
T6	Ευκολία εγκατάστασης	0,5
T7	Ευχρηστία	0,5
T8	Φορητότητα (portability)	2
T9	Ευκολία αλλαγής	1
T10	Ταυτόχρονη λειτουργία	1
T11	Απαιτήσεις για ασφάλεια	1
T12	Παρέχει πρόσβαση σε τρίτους	1
T13	Ειδικές ανάγκες εκπαίδευσης χρηστών	1
Περιβαλλοντικοί παράγοντες		
F1	Οικειότητα με το πρόγραμμα	1,5
F2	Εμπειρία εφαρμογής	0,5
F3	Δοκιμάζει	1
F4	Ικανότητα του βασικού αναλυτή	0,5
F5	Ύπαρξη κινήτρων για την ομάδα έργου	1
F6	Σταθερές απαιτήσεις	2
F7	Μερικής απασχόλησης εργαζόμενοι	-1

Παράγοντας	Περιγραφή τεχνικών παραγόντων πολυπλοκότητας	Βάρος
F8	Δύσκολη γλώσσα προγραμματισμού	-1

5 Υπολογίζουμε το Tfactor από το άθροισμα όλων των τεχνικών παραγόντων πολυπλοκότητας.

6 Υπολογίζουμε τον παράγοντα τεχνικής πολυπλοκότητας (Technical Complexity Factor - TCF) με βάση τον τύπο

TCF = 0,6 + 0,01 * Tfactor

7 Υπολογίζουμε το Efactor από το άθροισμα όλων των παραγόντων περιβάλλοντος.

8 Υπολογίζουμε τον παράγοντα περιβάλλοντος (Environment Factor - EF) με βάση τον τύπο

EF = 1,4 –(0,03 * Efactor)

9 Υπολογίζουμε τα σταθμισμένα σημεία ΠΧ (Adjusted Use Case Points - AUCP) με βάση τον τύπο

AUCP = UUCP * TCF * EF

10 Η συνολική απαιτούμενη προσπάθεια είναι

Effort = AUCP * Παράγοντας Παραγωγικότητας

Σύμφωνα με τον Karner, ο παράγοντας παραγωγικότητας είναι 20, δηλαδή 20 ώρες εργασίας ανά AUCP.

Υπολογισμός προσπάθειας για σύστημα μισθοδοσίας

Για τον υπολογισμό της προσπάθειας που απαιτείται για την ανάπτυξη του συστήματος μισθοδοσίας ακολουθούμε βήμα – βήμα τον παραπάνω αλγόριθμο.

1 Υπολογισμός αστάθμητου βάρους χειριστών (Unadjusted Actor Weight - UAW)

Χειριστής	**Ταξινόμηση**	**Βάρος**
Employee	Σύνθετος	3
PartTimeEmployee	Σύνθετος	3
FullTimeEmployee	Σύνθετος	3
CommissionedEmployee	Σύνθετος	3
PayrollAdministrator	Σύνθετος	3
	Σύνολο UAW	**15**

2 Υπολογισμός αστάθμητου βάρους ΠΧ (Unadjusted Use Case Weight - UUCW)

Περίπτωση χρήσης	**Ταξινόμηση**	**Βάρος**
Time Management	Απλή	5
Purchase Order Management	Μέτρια (CRUD ΠΧ)	10
Maintain Employee Information	Μέτρια (CRUD ΠΧ)	10
Run Payroll	Μέτρια (CRUD ΠΧ)	10
Produce Reports	Υπάρχουν πολλές αναφορές	10
	Σύνολο	**45**

3 Υπολογισμός συνολικού αριθμού αστάθμητων σημείων ΠΧ (Unadjusted Use Case

Points – UUCP).

UUCP = UAW + UUCW = 15 + 45 = **60**

4 Υπολογισμός των σταθμισμένων τεχνικών παραγόντων και παραγόντων περιβάλλοντος. Κάθε παράγοντας παίρνει μια τιμή από 0 έως 5 ανάλογα με το βαθμό επίδρασης του παράγοντα στο έργο.

Παράγοντας	**Περιγραφή τεχνικών παραγόντων πολυπλοκότητας**	**Βάρος**	**Πολυπλοκότητα Έργου**	**Τιμή**
T1	Είναι κατανεμημένο σύστημα	2	0	0
T2	Υπάρχουν απαιτήσεις απόδοσης (performance) ή απόκρισης (response)	2	3	6
T3	Υπάρχουν απαιτήσεις για την αποδοτικότητα τελικών χρηστών	1	3	3
T4	Υπάρχει σύνθετη εσωτερική επεξεργασία	1	4	4
T5	Αναπτύσσεται επαναχρησιμοποιήσιμος κώδικας	1	0	0
T6	Ευκολία εγκατάστασης	0,5	0	0
T7	Ευχρηστία	0,5	3	1,5
T8	Φορητότητα (portability)	2	0	0

Παράγοντας	Περιγραφή τεχνικών παραγόντων πολυπλοκότητας	Βάρος	Πολυπλοκότητα Έργου	Τιμή
T9	Ευκολία αλλαγής	1	3	3
T10	Ταυτόχρονη λειτουργία	1	0	0
T11	Απαιτήσεις για ασφάλεια	1	0	0
T12	Παρέχει πρόσβαση σε τρίτους	1	0	0
T13	Ειδικές ανάγκες εκπαίδευσης χρηστών	1	0	0
			Tfactor	**17,5**
Περιβαλλοντικοί παράγοντες				
F1	Οικειότητα με το πρόγραμμα	1,5	3	4,5
F2	Εμπειρία πεδίου εφαρμογής	0,5	3	1,5
F3	Εμπειρία ανάπτυξης αντικειμενοστρεφών εφαρμογών	1	3	3
F4	Ικανότητα του βασικού αναλυτή	0,5	3	1,5
F5	Ύπαρξη κινήτρων για την ομάδα έργου	1	3	3
F6	Σταθερές απαιτήσεις	2	3	6

Παράγοντας	Περιγραφή τεχνικών παραγόντων πολυπλοκότητας	Βάρος	Πολυπλοκότητα Έργου	Τιμή
F7	Μερικής απασχόλησης εργαζόμενοι	-1	3	-3
F8	Δύσκολη γλώσσα προγραμματισμού	-1	3	-3
			Efactor	**13,5**

Στο παράδειγμά μας θεωρούμε ότι οι περιβαλλοντικοί παράγοντες επηρεάζουν μετρίως την ανάπτυξη του συστήματος.

5 Υπολογισμός Tfactor

Tfactor = 17,5

6 Υπολογισμός παράγοντα τεχνικής πολυπλοκότητας (Technical Complexity Factor - TCF) με βάση τον τύπο

TCF = 0,6 + 0,01 * Tfactor = 0,6 + 0,01 * 17,5 = 0,775

7 Υπολογισμός Efactor

Efactor = 13,5

8 Υπολογισμός του παράγοντα περιβάλλοντος (Environment Factor - EF)

EF = 1,4 –(0,03 * Efactor) = 1,4 –(0,03 * 13,5)=0,995

9 Υπολογισμός σταθμισμένων σημείων ΠΧ (Adjusted Use Case Points - AUCP)

AUCP = UUCP * TCF * EF = 60 * 0,775 *0,995 = 46,26

10 Υπολογισμός συνολικής απαιτούμενης προσπάθειας

Effort = AUCP * Παράγοντας Παραγωγικότητας = 46,26 * 20 = 925, 35

Σύμφωνα με τους παραπάνω υπολογισμούς χρειάζονται 925 ώρες για την ανάπτυξη του συστήματος. Θεωρώντας ότι κατά μέσο όρο ένας ανθρωπομήνας έχει 150 ώρες

εργασίας, χρειαζόμαστε 6,16 ανθρωπομήνες για να αναπτύξουμε το σύστημα.

4.5 Φάση 2 - Βήμα 3. Ανάλυση ευρωστίας

Όπως έχουμε αναφέρει, η ανάλυση ευρωστίας είναι μια απλή και χρήσιμη μέθοδος που γεφυρώνει το χάσμα μεταξύ της ανάλυσης του συστήματος και του σχεδιασμού. Πολύ συχνά υπάρχει προβληματισμός για

- το πώς θα μεταβούμε από το μοντέλο της ανάλυσης, το διάγραμμα κλάσεων πεδίου προβλήματος και το μοντέλο των ΠΧ, στο μοντέλο του σχεδιασμού,
- το αν το μοντέλο της ανάλυσης επαρκεί για την κατασκευή του συστήματος
- το πώς να εξασφαλίσουμε ότι έχουμε περιγράψει όλες τις εναλλακτικές ροές των ΠΧ ή ότι όλες οι απαιτούμενες λειτουργίες του συστήματος έχουν ανατεθεί σε κλάσεις.

Η ανάλυση ευρωστίας περιλαμβάνει τον εντοπισμό και την ταξινόμηση των αντικειμένων του συστήματος σε τρεις κατηγορίες:

- Αντικείμενα οντοτήτων (entity objects), τα οποία αναπαριστούν οντότητες του πεδίου εφαρμογής. Συνήθως αναπαριστούν παθητικά αντικείμενα τα οποία αντιστοιχούν σε πίνακες βάσεων δεδομένων ή σε συμβατικά αρχεία, που χρειάζεται να υπάρχουν και μετά το πέρας της εκτέλεσης της ΠΧ.

- Συνοριακά αντικείμενα (boundary objects), τα οποία υλοποιούν την επικοινωνία των χειριστών (ανθρώπων ή άλλων συστημάτων) με το σύστημα. Συνήθως περιλαμβάνουν αντικείμενα οθόνης (π.χ. παράθυρα, μενού κα). Για τον εντοπισμό τους θα βασιστούμε στο πρωτότυπο του συστήματος και στα αποτελέσματα του βήματος 2 (ιδιαίτερα στις περιγραφές των περιπτώσεων χρήσης)

- Αντικείμενα ελέγχου (ή ελεγκτές) (control objects), τα οποία υλοποιούν τις υπηρεσίες του συστήματος και συνδέουν τα συνοριακά αντικείμενα (δηλαδή τους χειριστές) με τα αντικείμενα οντοτήτων (δηλαδή τα αποθηκεύσιμα δεδομένα). Σε αυτά τοποθετούμε τις πολιτικές επεξεργασίας και διάθεσης των δεδομένων, ώστε οι όποιες αλλαγές να μπορούν να γίνουν τοπικά και να μην έχουν επίδραση στη συνολική λειτουργία του συστήματος. Οι ελεγκτές χρησιμοποιούνται ακριβώς

επειδή δεν έχουμε ακόμη αρκετή πληροφορία για να αποδώσουμε τη συμπεριφορά που αναπαριστούν σε συγκεκριμένες κλάσεις.

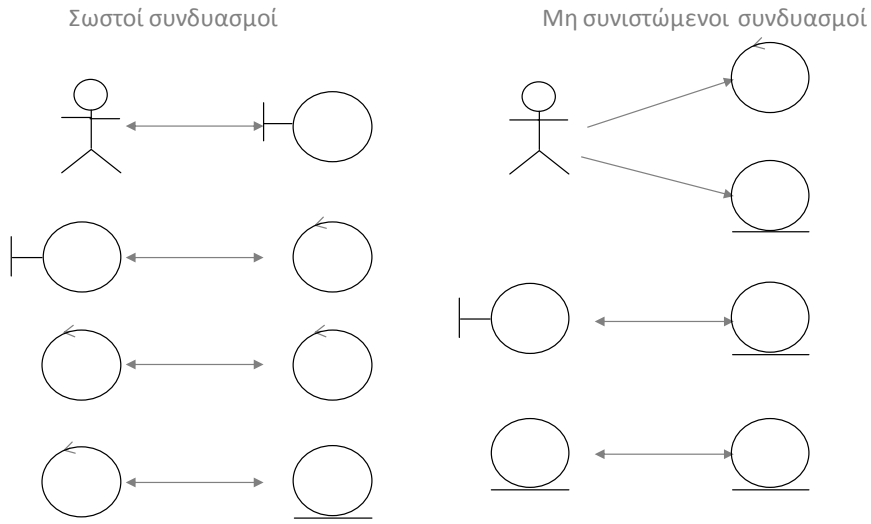
Η γραφική απεικόνιση των αντικειμένων παρουσιάζεται στην Εικόνα 26.

Εικόνα 26: Γραφική απεικόνιση αντικειμένων κατά την ανάλυση ευρωστίας

Για την κατασκευή του διαγράμματος ευρωστίας αναλύουμε τα βήματα των ΠΧ και σε κάθε βήμα της βασικής ροής τοποθετούμε το κατάλληλο αντικείμενο. Για κάθε σενάριο θα πρέπει να δημιουργήσουμε ένα διάγραμμα ευρωστίας, δηλαδή ένα για τη βασική ροή και ένα για κάθε εναλλακτική ροή. Αν ακολουθήσουμε την προσέγγιση αυτή ο συνολικός αριθμός διαγραμμάτων ευρωστίας είναι πολύ μεγάλος αφού για κάθε ΠΧ θα πρέπει να δημιουργήσουμε δύο, τρία ή και περισσότερα διαγράμματα ευρωστίας. Για το λόγο αυτό θα πρέπει να επιλέξουμε πόσα και ποια διαγράμματα ευρωστίας θα φτιάξουμε ώστε αφενός να έχουμε μια πλήρη περιγραφή του συστήματος αφετέρου την οικονομία της προσπάθειας στην ανάπτυξη του συστήματος. Προτιμότερο είναι να τοποθετήσουμε τόσο τη βασική ροή καθώς και τις εναλλακτικές ροές στο ίδιο διάγραμμα. Αυτό όμως θα πρέπει να γίνεται μόνο στις απλές ΠΧ. Σε αντίθετη περίπτωση το διάγραμμα ευρωστίας γίνεται δυσανάγνωστο και δύσχρηστο.

Επιπλέον, θα πρέπει να έχουμε υπόψη τους επόμενους κανόνες:

1. Οι χειριστές επικοινωνούν μόνο με συνοριακά αντικείμενα
2. Τα συνοριακά αντικείμενα επικοινωνούν μόνο με χειριστές και αντικείμενα ελέγχου
3. Τα αντικείμενα οντοτήτων επικοινωνούν μόνο με αντικείμενα ελέγχου
4. Τα αντικείμενα ελέγχου επικοινωνούν με συνοριακά αντικείμενα, με αντικείμενα ελέγχου και με άλλα αντικείμενα ελέγχου αλλά όχι με χειριστές.

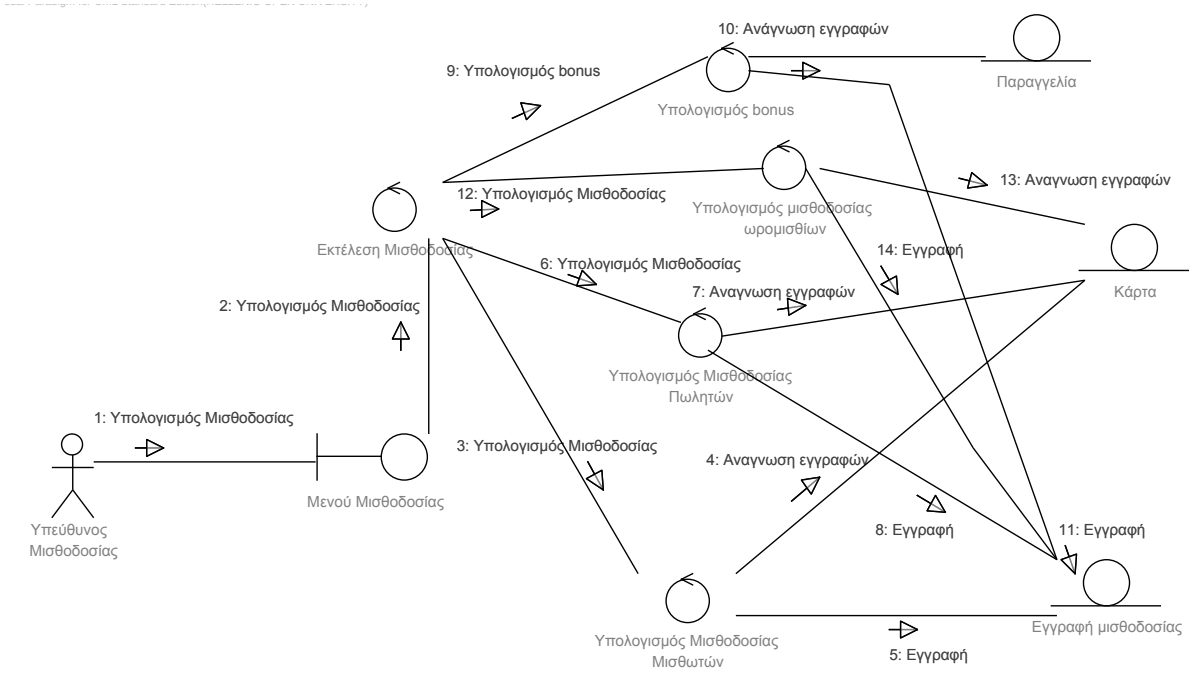


Εικόνα 27: Συνδυασμοί αντικειμένων ανάλυσης ευρωστίας

Δε θα πρέπει να ξεχνάμε ότι τα συνοριακά αντικείμενα και τα αντικείμενα οντοτήτων περιγράφονται συνήθως με ουσιαστικά, ενώ τα αντικείμενα ελέγχου με ρήματα. Ένας εμπειρικός κανόνας για να θυμόμαστε τους κανόνες επικοινωνίας είναι:

- τα ουσιαστικά δεν επικοινωνούν με ουσιαστικά και
- τα ρήματα επικοινωνούν με ουσιαστικά ή με ρήματα.

Στην Εικόνα 28 παρουσιάζεται το διάγραμμα ευρωστίας για την ΠΧ «Υπολογισμός μισθοδοσίας» και για τη βασική της ροή, ενώ στη συνέχεια παρουσιάζεται η αντιστοίχηση των βημάτων της βασικής ροής της ΠΧ με τα μηνύματα που ανταλλάσσονται στο διάγραμμα ευρωστίας. Παρατηρούμε ότι υπάρχει άμεση αντιστοίχιση μεταξύ της ροής της ΠΧ και του διαγράμματος ευρωστίας.



Εικόνα 28: Διάγραμμα ευρωστίας για την ΠΧ «Υπολογισμός Μισθοδοσίας»

Βήμα κύριας ροής ΠΧ «Υπολογισμός Μισθοδοσίας»	**Μηνύματα διαγράμματος ευρωστίας**
1. Ο διαχειριστής μισθοδοσίας επιλέγει «Εκτέλεση Μισθοδοσίας» από την κεντρική οθόνη του συστήματος	Α. Ο χειριστής *Υπεύθυνος μισθοδοσίας* (Payroll Administrator) ενεργοποιεί ή επιλέγει την λειτουργία 1. «Υπολογισμός Μισθοδοσίας» στο συνοριακό αντικείμενο *Μενού Μισθοδοσίας.*
2. Επιλέγει την περίοδο για την οποία θα εκτελέσει τη μισθοδοσία	Β. Αφού γίνει η επιλογή περιόδου υπολογισμού μισθοδοσίας, το συνοριακό αντικείμενο *Μενού Μισθοδοσίας* στέλνει το μήνυμα 2. «Υπολογισμός Μισθοδοσίας» στο αντικείμενο ελέγχου *Εκτέλεση Μισθοδοσίας.*
3. Γίνεται υπολογισμός της μισθοδοσίας	Γ. Το αντικείμενο ελέγχου *Εκτέλεση*

Βήμα κύριας ροής ΠΧ «Υπολογισμός Μισθοδοσίας»	**Μηνύματα διαγράμματος ευρωστίας**
εργαζομένων	*Μισθοδοσίας* στέλνει το μήνυμα 3. «Υπολογισμός Μισθοδοσίας» στο αντικείμενο ελέγχου *Εκτέλεση Μισθοδοσίας Μισθωτών.* Δ. Το αντικείμενο ελέγχου *Εκτέλεση Μισθοδοσίας Μισθωτών* στέλνει το μήνυμα 4. «Ανάγνωση Εγγραφών» στο αντικείμενο οντότητας *Κάρτα.* Ε. Αφού γίνει ο υπολογισμός της μισθοδοσίας, το αντικείμενο ελέγχου *Εκτέλεση Μισθοδοσίας Μισθωτών* στέλνει το μήνυμα 5. «Εγγραφή» στο αντικείμενο οντότητας *Εγγραφή Μισθοδοσίας.*
4. Γίνεται υπολογισμός της μισθοδοσίας πωλητών	ΣΤ. Το αντικείμενο ελέγχου *Εκτέλεση Μισθοδοσίας* στέλνει το μήνυμα 6. «Υπολογισμός Μισθοδοσίας» στο αντικείμενο ελέγχου *Εκτέλεση Μισθοδοσίας Πωλητών.* Ζ. Το αντικείμενο ελέγχου *Εκτέλεση Μισθοδοσίας Πωλητών* στέλνει το μήνυμα 7. «Ανάγνωση Εγγραφών» στο αντικείμενο οντότητας *Κάρτα.* Η. Αφού γίνει ο υπολογισμός της μισθοδοσίας, το αντικείμενο ελέγχου *Εκτέλεση Μισθοδοσίας Πωλητών* στέλνει το μήνυμα 8. «Εγγραφή» στο αντικείμενο

Βήμα κύριας ροής ΠΧ «Υπολογισμός Μισθοδοσίας»	**Μηνύματα διαγράμματος ευρωστίας**
	οντότητας *Εγγραφή Μισθοδοσίας.*
5. Γίνεται υπολογισμός των bonus πωλητών	Θ. Το αντικείμενο ελέγχου *Εκτέλεση Μισθοδοσίας* στέλνει το μήνυμα 9. «Υπολογισμός Bonus» στο αντικείμενο ελέγχου *Υπολογισμός Bonus.* Ι. Το αντικείμενο ελέγχου *Υπολογισμός Bonus* στέλνει το μήνυμα 10. «Ανάγνωση Εγγραφών» στο αντικείμενο οντότητας *Παραγγελία.* ΙΑ. Αφού γίνει ο υπολογισμός των bonus, το αντικείμενο ελέγχου *Υπολογισμός Bonus* στέλνει το μήνυμα 11. «Εγγραφή» στο αντικείμενο οντότητας *Εγγραφή Μισθοδοσίας.*
6. Γίνεται υπολογισμός της μισθοδοσίας ωρομισθίων	ΙΒ. Το αντικείμενο ελέγχου *Εκτέλεση Μισθοδοσίας* στέλνει το μήνυμα 12. «Υπολογισμός Μισθοδοσίας» στο αντικείμενο ελέγχου *Εκτέλεση Μισθοδοσίας Ωρομισθίων.* ΙΓ. Το αντικείμενο ελέγχου *Εκτέλεση Μισθοδοσίας Ωρομισθίων* στέλνει το μήνυμα 13. «Ανάγνωση Εγγραφών» στο αντικείμενο οντότητας *Κάρτα.* ΙΔ. Αφού γίνει ο υπολογισμός της μισθοδοσίας, το αντικείμενο ελέγχου *Εκτέλεση Μισθοδοσίας Ωρομισθίων*

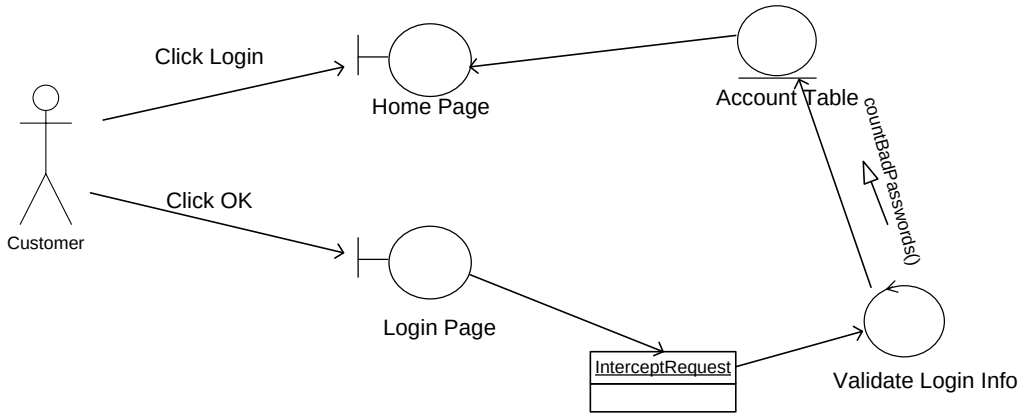
Βήμα κύριας ροής ΠΧ «Υπολογισμός Μισθοδοσίας»	Μηνύματα διαγράμματος ευρωστίας
	στέλνει το μήνυμα 14. «Εγγραφή» στο αντικείμενο οντότητας *Εγγραφή Μισθοδοσίας.*
7. Επιστροφή στην κεντρική οθόνη του συστήματος	Δεν αντιστοιχεί σε μήνυμα

Δραστηριότητα 4

Δίνεται το παρακάτω διάγραμμα ευρωστίας.

1) Βρείτε τα σφάλματα.

2) Κατασκευάστε νέα διορθωμένη έκδοση του διαγράμματος



Εικόνα 29: Λανθασμένο διάγραμμα ευρωστίας

4.6 Φάση 3 - Βήμα 4. Λεπτομερής σχεδιασμός

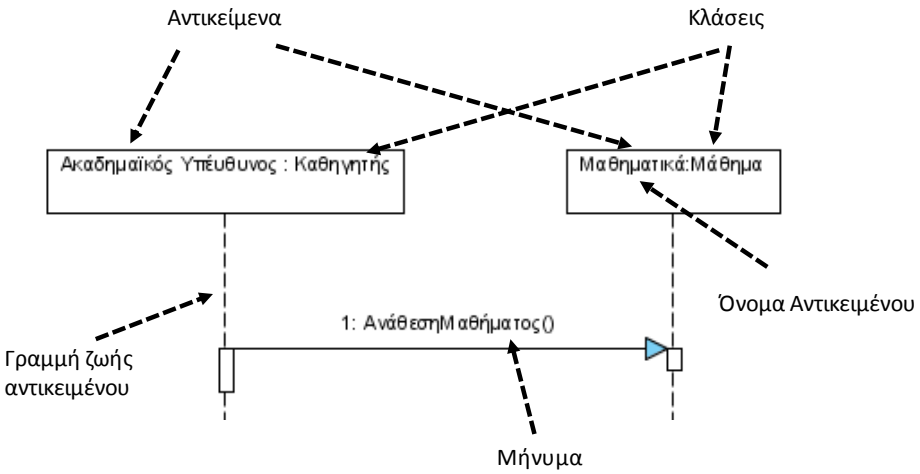
Ο λεπτομερής σχεδιασμός του συστήματος έχει ως στόχο τον πλήρη ορισμό της συμπεριφοράς των αντικειμένων που έχουμε εντοπίσει, μέσα από την αντιστοίχιση μεθόδων στις κλάσεις του συστήματος. Ο λεπτομερής σχεδιασμός γίνεται με δύο βασικούς

τρόπους:

1) Αναπτύσσουμε διαγράμματα ακολουθίας (sequence diagrams) και
2) Εκλεπτύνουμε το διάγραμμα κλάσεων του συστήματος

4.6.1 Διαγράμματα ακολουθίας

Τα διαγράμματα ακολουθίας έχουν ως βασικό στόχο να δείξουν την αλληλεπίδραση των αντικειμένων κατά τη διάρκεια λειτουργίας του συστήματος. Τα αντικείμενα συμβολίζονται με παραλληλόγραμμα στα οποία αναγράφεται το όνομα του αντικειμένου ακολουθούμενο από μία άνω-κάτω τελεία και στη συνέχεια το όνομα της κλάσης. Κάτω από κάθε αντικείμενο εκτείνεται μία διακεκομμένη γραμμή που ονομάζεται γραμμή ζωής (lifeline) του αντικειμένου. Ένα σύντομο και απλό παράδειγμα διαγράμματος ακολουθίας δίνεται στην Εικόνα 30.



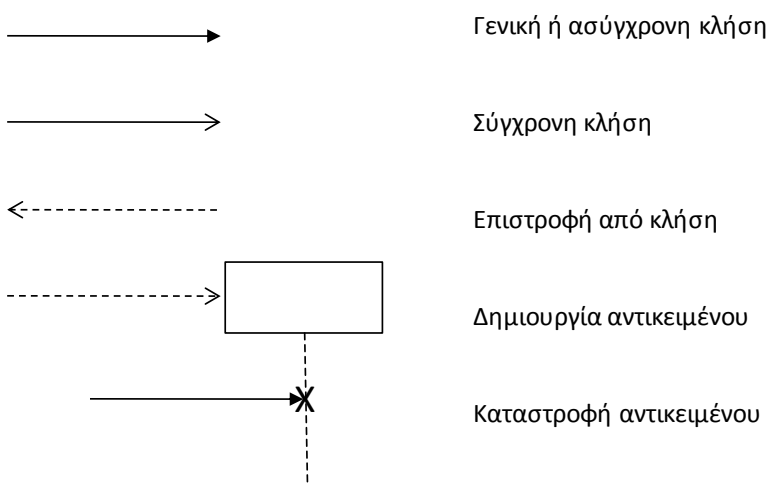
Εικόνα 30: Παράδειγμα διαγράμματος ακολουθίας

Αν πρέπει να δείξουμε τον τερματισμό της ζωής ενός αντικειμένου, μπορούμε να βάλουμε ένα X στο τέλος της γραμμής ζωής του. Τα αντικείμενα γράφονται από αριστερά προς τα δεξιά και στον οριζόντιο άξονα. Τα αντικείμενα ανταλλάσσουν μηνύματα, τα οποία στην επίσημη ορολογία της UML ονομάζονται ερεθίσματα (stimuli). Ερέθισμα μπορεί να είναι οτιδήποτε από τα παρακάτω:

- Κλήση μιας λειτουργίας: όταν ένα αντικείμενο καλεί μία λειτουργία ενός άλλου αντικειμένου. Πρόκειται για σύγχρονο μήνυμα, δηλαδή ο αποστολέας του

μηνύματος θα πρέπει να περιμένει την ολοκλήρωση της λειτουργίας για να συνεχίσει. Συμβολίζεται με ένα βέλος από τον αποστολέα προς τον παραλήπτη. Η κεφαλή του βέλους είναι γεμισμένη με μαύρο χρώμα. Πάνω από το βέλος αναγράφεται το όνομα της λειτουργίας που καλείται, με τις ενδεχόμενες παραμέτρους σε παρενθέσεις.

- Γενικό ή ασύγχρονο μήνυμα. Ασύγχρονα μηνύματα συναντάμε σε πολυνηματικές (multithreading) εφαρμογές, όπου ένα μήνυμα τοποθετείται σε κάποια ουρά ενός νήματος εκτέλεσης ενώ το ενεργό αντικείμενο-παραλήπτης θα επεξεργαστεί το μήνυμα σε κάποια επόμενη χρονική στιγμή. Η διαφορά με την κλήση στο συμβολισμό είναι πως η κατάληξη είναι ένα ανοιχτό βέλος. Η σημασιολογική διάκριση είναι πως το αντικείμενο-αποστολέας του ασύγχρονου μηνύματος μπορεί να συνεχίσει την επεξεργασία χωρίς να περιμένει την ολοκλήρωση της επεξεργασίας του μηνύματος που απέστειλε.
- Δημιουργία αντικειμένου: είναι ένα ερέθισμα που καταλήγει στη δημιουργία ενός νέου αντικειμένου και όχι στη γραμμή ζωής κάποιου υπάρχοντος αντικειμένου. Ένα τυπικό παράδειγμα είναι η απεικόνιση της κλήσης της μεθόδου κατασκευής ενός αντικειμένου.



Εικόνα 31: Είδη μηνυμάτων στα διαγράμματα ακολουθίας

Η κατασκευή του διαγράμματος ακολουθίας έχοντας ως δεδομένο το διάγραμμα ευρωστίας γίνεται σε τρία βασικά βήματα:

- Προσθέτουμε τα αντικείμενα οντοτήτων του μοντέλου ευρωστίας στον οριζόντιο επάνω άξονα του διαγράμματος ακολουθίας. Καθένα από αυτά αντιστοιχεί σ'ένα στιγμιότυπο μιας κλάσης του στατικού μοντέλου. Κάθε νέο πεδίο που ανακαλύπτουμε κατά τη δραστηριότητα αυτή το προσθέτουμε χωρίς καθυστέρηση στο στατικό μοντέλο (διάγραμμα κλάσεων)

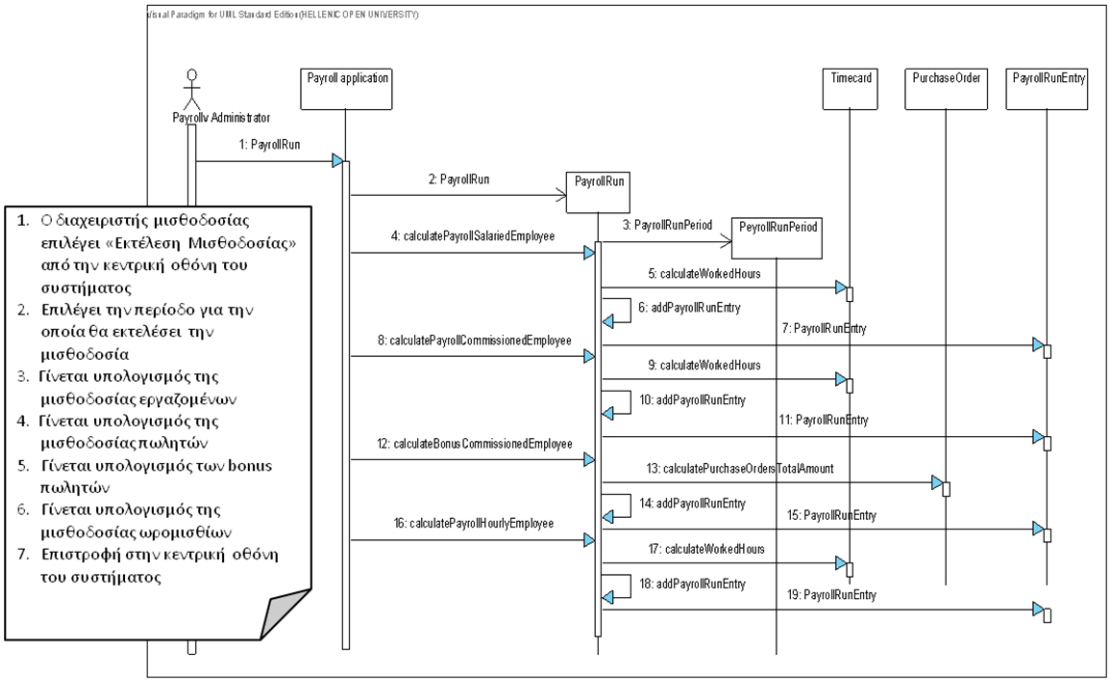
- Προσθέτουμε στον ίδιο άξονα τα συνοριακά αντικείμενα και τους χειριστές.

- Καταχωρίζουμε μεθόδους σε κλάσεις. Αποτελεί το πιο σημαντικό και δύσκολο βήμα αφού μετατρέπουμε τα αντικειμένα ελέγχου (από το διάγραμμα ευρωστίας) σε ένα σύνολο μεθόδων και μηνυμάτων που υλοποιούν τη συμπεριφορά του.

Ταυτόχρονα με τη δημιουργία των διαγραμμάτων ακολουθίας θα πρέπει να εμπλουτίζουμε το διάγραμμα κλάσεων ώστε να συμπεριλαμβάνει τις σχεδιαστικές αποφάσεις που λάβαμε.

Στην Εικόνα 32 παρουσιάζεται το διάγραμμα ακολουθίας για την ΠΧ «Υπολογισμός Μισθοδοσίας». Τα μηνύματα 7,11,15,19 θα έπρεπε να δημιουργούν νέα αντικείμενα, αλλά για λόγους καλύτερης παρουσίασης οδηγούνται στο ίδιο lifeline.

Προφανώς θα μπορούσαμε να παρουσιάσουμε ένα επίπεδο λεπτομέρειας ακόμη παρουσιάζοντας και τα αντικείμενα των κλάσεων LogEntry.



Εικόνα 32: Το διάγραμμα ακολουθίας για την ΠΧ «Υπολογισμός Μισθοδοσίας»

4.6.2 Διάγραμμα κλάσεων λεπτομερούς σχεδιασμού

Η ολοκλήρωση του λεπτομερούς σχεδιασμού γίνεται με την επικαιροποίηση και επέκταση του στατικού μοντέλου του συστήματος, δηλαδή του διαγράμματος κλάσεων.

Για την ολοκλήρωση του διαγράμματος κλάσεων θα πρέπει:

- να γίνει αναλυτικός ορισμός των πεδίων των κλάσεων. Ο αναλυτικός ορισμός των πεδίων περιλαμβάνει ορισμό του τύπου, της ορατότητας, της αρχικής τιμής, των περιορισμών, κ.α.
- να γίνει αναλυτικός ορισμός των μεθόδων των κλάσεων. Ο ορισμός των μεθόδων περιλαμβάνει ορισμό παραμέτρων, του τύπου επιστροφής, της ορατότητας, τους τύπους των δεδομένων που επιστρέφουν οι μέθοδοι, κ.α.
- να αποφασίσουμε για τον τρόπο που θα υλοποιηθούν οι σχέσεις που υπάρχουν στο μοντέλο μας,
- να αποφασίσουμε για τον τρόπο αποθήκευσης των δεδομένων στη βάση των δεδομένων
- να αποφασίσουμε για τις κλάσεις που θα χρησιμοποιήσουμε από τις υπάρχουσες βιβλιοθήκες της java.

Στην Εικόνα 33 παρουσιάζονται οι κλάσεις του πακέτου Employees. Για κάθε πεδίο των κλάσεων έχουμε ορίσει τον τύπο καθώς και την ορατότητα (visibility) του πεδίου. Η ορατότητα προσδιορίζει το ποιος μπορεί να διαβάσει ή να τροποποιήσει τα περιεχόμενα μιας μεταβλητής. Στόχος μας είναι όλα τα πεδία να είναι εσωτερικά σε κάθε κλάση ώστε να επιτυγχάνουμε με τον καλύτερο δυνατό τρόπο την αρχή της κελυφοποίησης (encapsulation). Μπορούμε να ορίσουμε την ορατότητα με τέσσερεις διαφορετικούς τρόπους τόσο στη UML όσο και στη Java:

- **public**, όταν όλες οι κλάσεις μπορούν να προσπελάσουν τη μεταβλητή. Συμβολίζεται με το (+).
- **protected**, όταν μπορεί να προσπελασθεί μόνο από τις μεθόδους της ίδιας κλάσης, από οποιαδήποτε κλάση παιδί (subclass) και από οποιαδήποτε άλλη κλάση που ανήκει στο ίδιο πακέτο. Συμβολίζεται με το (#).
- **private**, όταν είναι προσπελάσιμη μόνο από τις μεθόδους της ίδιας κλάσης. Συμβολίζεται με το (-).
- χωρίς χαρακτηρισμό, που είναι και ο εξ ορισμού χαρακτηρισμός όπου η μεταβλητή μπορεί να προσπελασθεί μόνο από τις μεθόδους της ίδιας κλάσης, και από οποιαδήποτε άλλη κλάση που ανήκει στο ίδιο πακέτο. Η μη ύπαρξη χαρακτηρισμού ισοδυναμεί με το χαρακτηρισμό friend στη γλώσσα C++ ή protected package στη γλώσσα Java. Στη UML συμβολίζεται με το (~).

Επιπλέον, κάποια πεδία των κλάσεων εμφανίζονται ως υπογραμμισμένα (π.χ. ~Weeklypaid : boolean = false). Τα πεδία αυτά είναι πεδία που ανήκουν στην κλάση και όχι στα στιγμιότυπα της κλάσης (instances). Αυτό σημαίνει ότι υπάρχει μόνο ένα τέτοιο πεδίο για κάθε κλάση το οποίο είναι κοινό για όλα τα αντικείμενα της κλάσης. Στην Java τα πεδία αυτά θα έχουν το χαρακτηρισμό static.

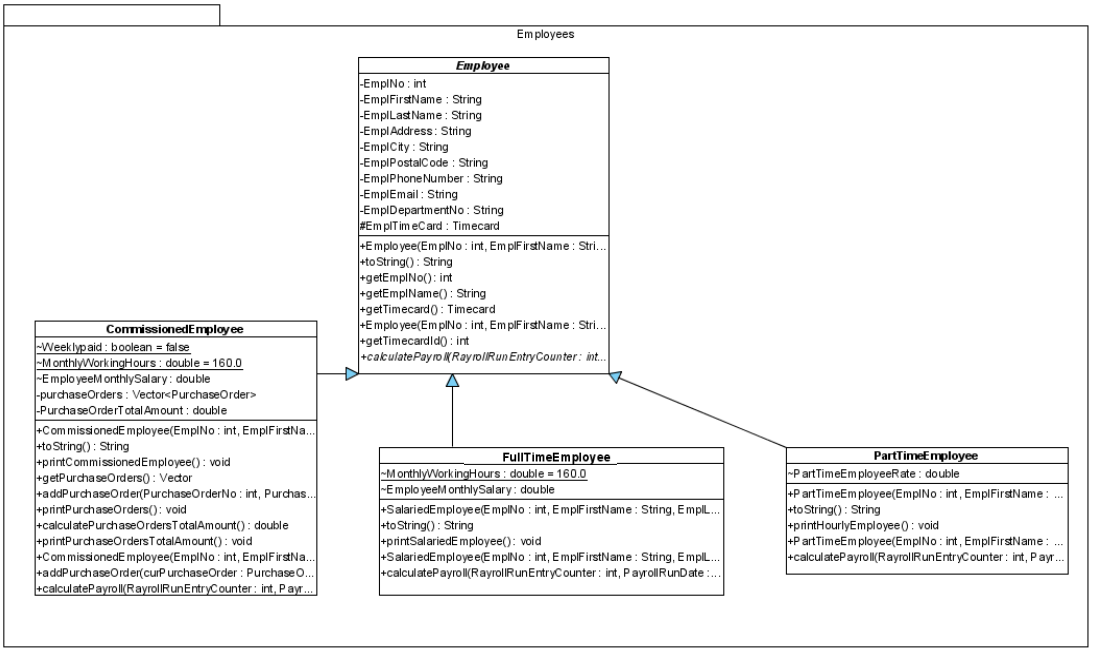
Για την υλοποίηση της σχέσης μεταξύ της κλάσης Employee και TimeCard δηλώνουμε το πεδίο #EmplTimeCard : Timecard. Στην περίπτωση αυτή δηλώνουμε ότι το πεδίο αυτό είναι τύπου Timecard, δηλαδή είναι και το ίδιο αντικείμενο.

Αντίστοιχα, για την υλοποίηση της σχέσης μεταξύ της κλάσης CommissionedEmployee και PurchaseOrder δηλώνουμε το πεδίο με όνομα purchaseOrders

το οποίο είναι τύπου Vector.

Το Vector είναι μια κλάση του πακέτου κλάσεων java.util, η οποία υλοποιεί ένα δυναμικά αυξανόμενο πίνακα αντικειμένων. Αποτελεί ένα εύκολο τρόπο για να υλοποιήσουμε λίστες αντικειμένων, ειδικά όταν δε γνωρίζουμε τον ακριβή τους αριθμό.

Επιπλέον, στην Εικόνα 33 θα πρέπει να προσέξουμε τη μέθοδο calculatePayroll, η οποία εμφανίζεται στο διάγραμμα κλάσεων με italics. Ο τρόπος γραφής με italics δηλώνει ότι η μέθοδος είναι abstract (αφηρημένη). Οι abstract μέθοδοι χρησιμοποιούνται για τη μοντελοποίηση κοινών λειτουργιών σε ιεραρχίες κλάσεων και δεν έχουν υλοποίηση. Στη συνέχεια, ορίζουμε σε κάθε μια από τις υποκλάσεις μια μέθοδο calculatePayroll, κάθε μια εκ των οποίων έχει διαφορετική υλοποίηση. Ο τρόπος σχεδίασης της συγκεκριμένης μεθόδου υλοποιεί την τεχνική του πολυμορφισμού, σύμφωνα με την οποία διαφορετικά αντικείμενα (ή κλάσεις) αντιδρούν με διαφορετικό τρόπο στο ίδιο μήνυμα. Επιπλέον, όταν δηλώσουμε μια μέθοδο ως abstract, η κλάση που την περιλαμβάνει είναι και αυτή *abstract.* Αυτό σημαίνει ότι η κλάση *Employee* είναι abstract, δηλαδή δεν υπάρχουν αντικείμενα τύπου Employee.

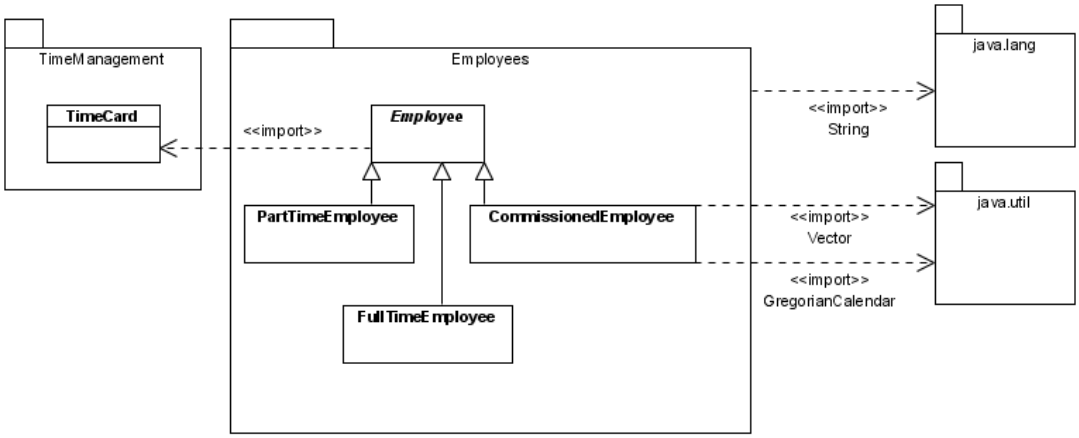


Εικόνα 33: Κλάσεις του πακέτου Employees

Δραστηριότητα 5

Δώστε ένα παράδειγμα πολυμορφισμού. Επεξηγήστε το σκεπτικό σας.

Η χρήση κλάσεων από τις βιβλιοθήκες της java αποτελεί μια εξάρτηση τύπου import, η οποία θα πρέπει να αποτυπωθεί στο λεπτομερή σχεδιασμό του συστήματος.



Εικόνα 34: Οι εξαρτήσεις των πακέτων

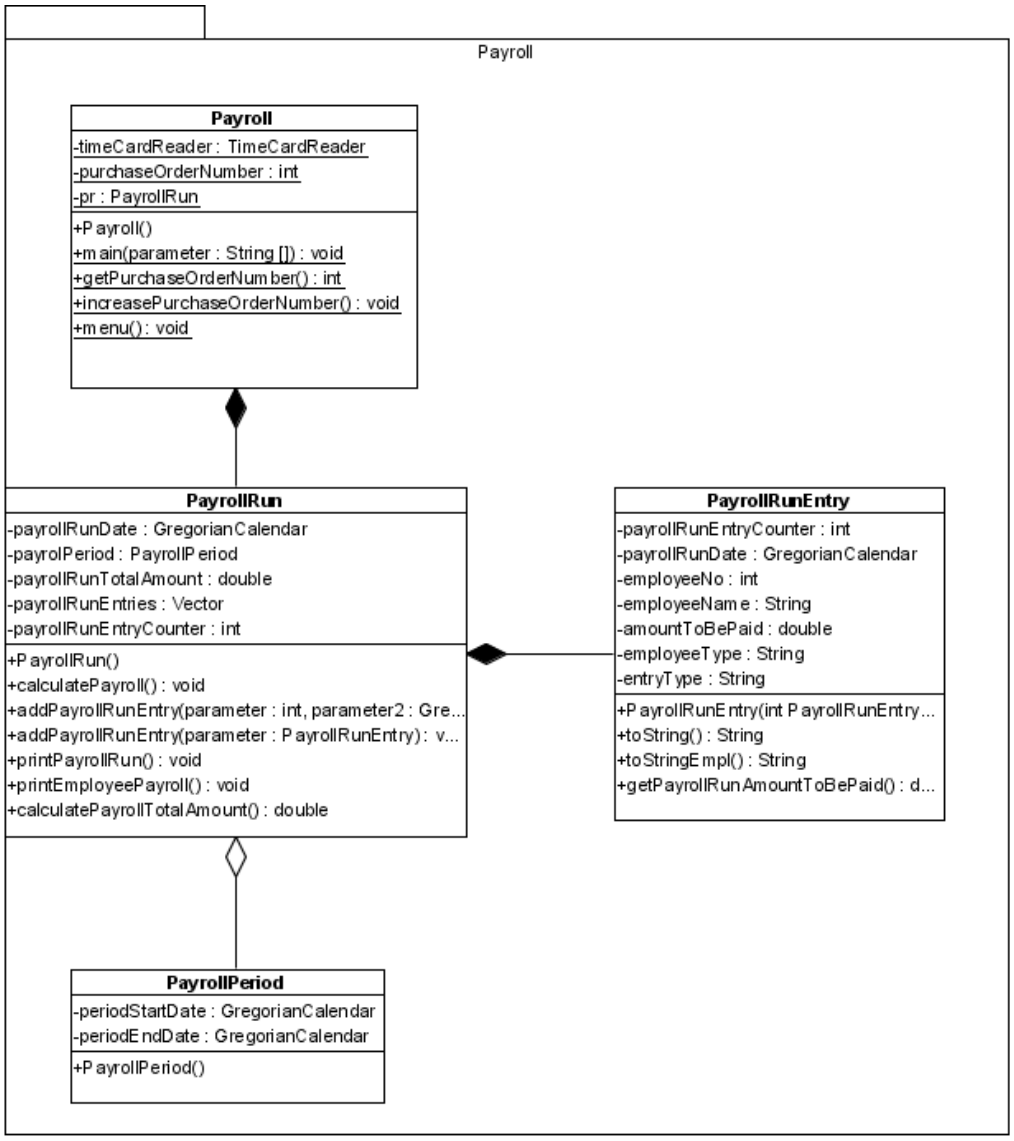
Προχωρούμε με τον ορισμό του πακέτου Payroll. Οι κλάσεις του πακέτου Payroll παρουσιάζονται στην Εικόνα 35.

Στην κλάση PayrollRun έχουμε ορίσει δύο μεθόδους με το ίδιο όνομα addPayrollRunEntry αλλά με διαφορετικό αριθμό παραμέτρων. Η τεχνική αυτή ονομάζεται *υπερφόρτωση μεθόδων* (method overloading) και χρησιμοποιείται για την απλοποίηση της σχεδίασης. Στη συγκεκριμένη περίπτωση έχουμε δύο μεθόδους που δημιουργούν μια PayrollRunEntry περνώντας ως παράμετρο είτε ένα αντικείμενο του τύπου PayrollRunEntry είτε αναλυτικά όλες τις παραμέτρους που απαρτίζουν ένα αντικείμενο PayrollRunEntry. Όταν καλείται η μέθοδος από άλλα αντικείμενα, το ποια ακριβώς μέθοδος θα κληθεί εξαρτάται από τον τύπο και τον αριθμό των παραμέτρων. Και στις δυο περιπτώσεις και ανεξάρτητα από το ποια μέθοδος θα κληθεί το αποτέλεσμα θα είναι να προστεθεί ένα ακόμη αντικείμενο PayrollRunEntry στη «λίστα» των εγγραφών μισθοδοσίας για το συγκεκριμένο PayrollRun.

[83]

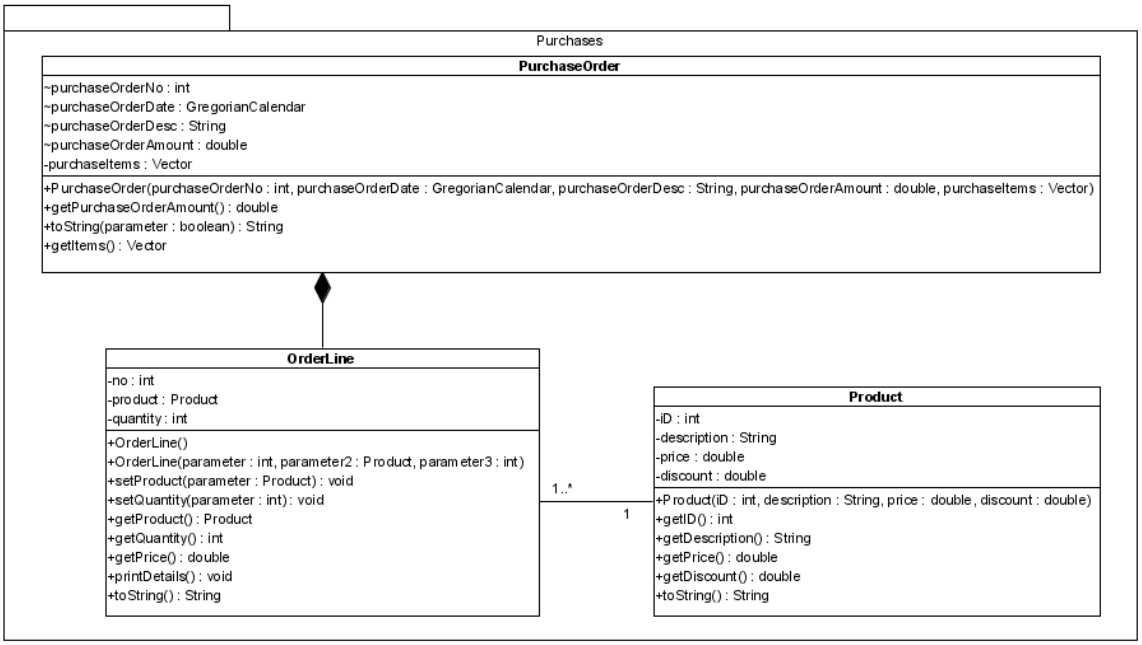
Δραστηριότητα 6

Δώστε ένα παράδειγμα υπερφόρτωσης μεθόδων από μια κλάση της Java.



Εικόνα 35: Ο αναλυτικός σχεδιασμός των κλάσεων του πακέτου Payroll

Αντίστοιχα, οι κλάσεις του πακέτου Purchases, δηλαδή οι κλάσεις PurchaseOrder, Product παρουσιάζονται στην Εικόνα 36.



Εικόνα 36: Αναλυτικός σχεδιασμός των κλάσεων του πακέτου Purchases

Δραστηριότητα 7

Δώστε τον αναλυτικό σχεδιασμό των κλάσεων του πακέτου TimeManagement, δηλαδή των κλάσεων TimeCard, CardReader, LogEntry.

Θα συνεχίσουμε παρουσιάζοντας την κλάση PersistentStorage η οποία χρησιμοποιείται για την αποθήκευση των δεδομένων στη βάση δεδομένων. Σύμφωνα με την εκφώνηση:

> *Τα δεδομένα της μισθοδοσίας όσο και τα στοιχεία των υπαλλήλων θα αποθηκεύονται σε ΒΔ. Όταν ξεκινά η εφαρμογή θα πρέπει να διαβάζεται αυτόματα η πληροφορία των υπαλλήλων από τη ΒΔ και να δημιουργούνται τα κατάλληλα αντικείμενα (υπαλλήλων, κάρτας εισόδου, κ.ο.κ).*

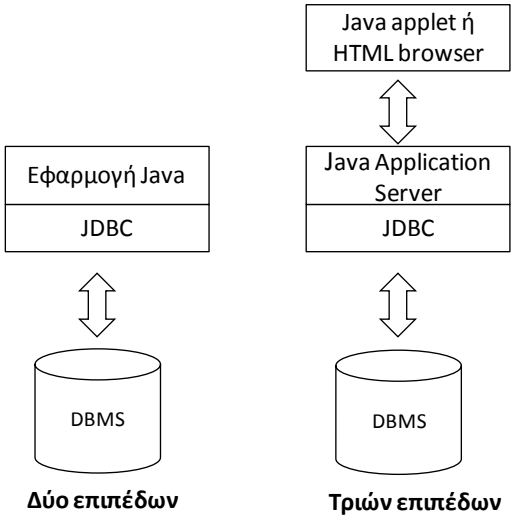
Η παραπάνω εκφώνηση είναι περιοριστική αφού έγινε προσπάθεια να κρατηθεί η εφαρμογή όσο το δυνατόν πιο απλή. Για την αποθήκευση δεδομένων σε μια σχεσιακή ΒΔ ο απλούστερος τρόπος είναι να χρησιμοποιήσουμε τη τεχνολογία JDBC.

Το JDBC API είναι η βιβλιοθήκη της Java που μας επιτρέπει να προσπελάσουμε με

εύκολο τρόπο δεδομένα που είναι αποθηκευμένα σε σχεσιακές βάσεις δεδομένων. Το JDBC μας βοηθά να γράψουμε εφαρμογές, οι οποίες μας επιτρέπουν:

1) Να συνδεθούμε σε μια πηγή δεδομένων, όπως μια βάση δεδομένων.
2) Να εκτελέσουμε ερωτήσεις (queries) ή να ενημερώσουμε τα δεδομένα που είναι αποθηκευμένα στη βάση δεδομένων.
3) Να ανακτήσουμε τα δεδομένα που είναι αποθηκευμένα στη Βάση δεδομένων και απαντούν στις ερωτήσεις μας.

Το JDBC υποστηρίζει αρχιτεκτονικές δύο ή τριών επιπέδων για τη σύνδεση με τη βάση δεδομένων. Η πιο απλή περίπτωση είναι αυτή της αρχιτεκτονικής των δυο επιπέδων, η οποία και παρουσιάζεται στην **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.**.



Εικόνα 37: Αρχιτεκτονική JDBC

Στην περίπτωση που εξετάζουμε η αρχιτεκτονική που ικανοποιεί τις απαιτήσεις είναι αυτή των δύο επιπέδων, αφού η εφαρμογή είναι παραθυρική και η λογική της εφαρμογής (business logic) είναι ενοποιημένη με το GUI (presentation logic).

Η δομή του JDBC, αποτελείται κατά κύριο λόγο από δύο τμήματα:

1. Το ***JDBC API***, ένα σύνολο κλάσεων που επιτρέπουν τη διεπαφή του χρήστη με το JDBC και
2. Τον ***JDBC driver manager*** ο οποίος επικοινωνεί με άλλους εξειδικευμένους drivers, που εξαρτώνται από τον κατασκευαστή της βάσης δεδομένων, και οι

οποίοι εκτελούν την πραγματική επικοινωνία με τη ΒΔ. Η μετάφραση στη μορφή που ορίζει ο κατασκευαστής της κάθε ΒΔ γίνεται στον client. Έτσι δε χρειάζονται αλλαγές στον server, ενώ χρειάζεται ένας driver που να κάνει τη μετάφραση στον client.

Το JDBC API είναι μέρος του περιβάλλοντος της Java, τόσο της *Java Standard Edition* (Java SE) όσο και της *Java Enterprise Edition* (Java EE). Το JDBC API αποτελείται από δύο πακέτα: το `java.sql` και το `javax.sql`.

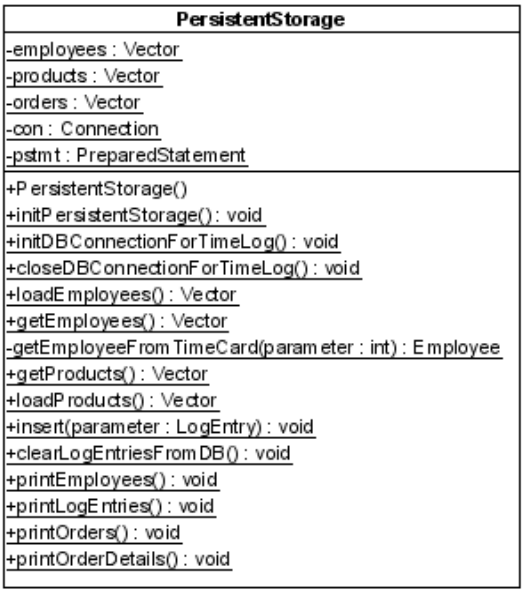
Στο σενάριο που επιλέξαμε να υλοποιήσουμε τα στοιχεία των υπαλλήλων υπάρχουν ήδη αποθηκευμένα στον πίνακα EMPLOYEE μιας ΒΔ και θα πρέπει αρχικά να ανακτηθούν ώστε να δημιουργηθούν τα αντίστοιχα αντικείμενα, και μετά να εκτυπωθούν.

Η ανάκτηση και δημιουργία των αντικειμένων όλων των κατηγοριών υπαλλήλων, γίνεται άπαξ με την εκκίνηση της εφαρμογής καλώντας τη μέθοδο loadEmployees() της κλάσης PersistentStorage.

Εδώ τίθεται το ερώτημα, γιατί θα πρέπει να δημιουργηθεί μια νέα κλάση (δε δημιουργεί αντικείμενα) στην οποία βρίσκονται οι πίνακες αντικειμένων (Vectors) αλλά και γενικά η διαχείριση της βάσης δεδομένων;

Η πρακτική αυτή βασίζεται σε συγκεκριμένο σχεδιαστικό πρότυπο, που προτάθηκε από τον Larman, το Pure Fabrication, το οποίο συνιστά το διαχωρισμό της διαχείρισης των πινάκων της βάσης δεδομένων από τις υπόλοιπες κλάσεις του συστήματος, με σκοπό τη μείωση της πολυπλοκότητας του συστήματος όπως και της επαναχρησιμοποίησης των κλάσεων. Ετσι, θεωρήθηκε αναγκαίο να προστεθεί στο υπάρχον διάγραμμα κλάσεων η κλάση PersistentStorage, στην οποία θα ανατεθεί ο έλεγχος της διαχείρισης των πινάκων της βάσης δεδομένων.

Η μέθοδος loadEmployees() της κλάσης PersistentStorage ανακτά τις εγγραφές των Υπαλλήλων από τη ΒΔ και δημιουργεί τα αντίστοιχα αντικείμενα σε Vector (Vector<Employee>), με βάση τον τύπο του εργαζομένου. Τα αντικείμενα αυτά, αν και τριών διαφορετικών κατηγοριών (PartTimeEmployee, CommissionedEmployee, FullTimeEmployee) αποθηκεύονται στη συνέχεια στον πίνακα employees επειδή είναι υπερ-τύπου Employee εκμεταλλευόμενοι το μηχανισμό του πολυμορφισμού.



Εικόνα 38: Η κλάση PersistentStorage

5. Ο κώδικας Java

Στο κεφάλαιο αυτό θα παρουσιάσουμε βασικά σημεία της υλοποίησης σε java, δίνοντας έμφαση σε σημεία που χρειάζονται επεξήγηση. Ο πλήρης κώδικας δίνεται στο παράρτημα Γ.

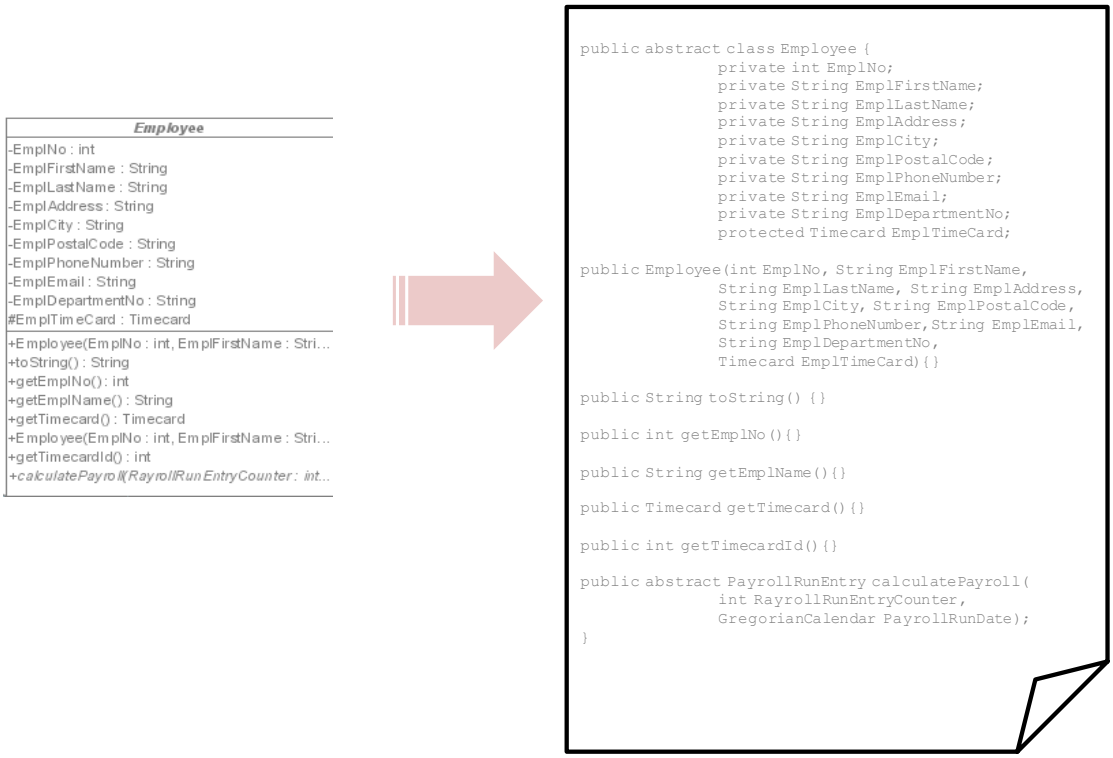
5.1 Γενικοί κανόνες για το μετασχηματισμό του μοντέλου UML σε κώδικα

Το μοντέλο σχεδιασμού που έχουμε δημιουργήσει μας επιτρέπει να δημιουργήσουμε με εύκολο τρόπο τον κώδικα σε Java. Πιο συγκεκριμένα, το διάγραμμα κλάσεων μας επιτρέπει να δημιουργήσουμε άμεσα το σκελετό των κλάσεων, αφού οι κανόνες που ισχύουν είναι πολύ απλοί. Δηλαδή:

- Κάθε κλάση UML αντιστοιχεί σε μια κλάση Java
- Κάθε πεδίο της κλάσης UML αντιστοιχεί σε μια μεταβλητή της κλάσης Java
- Κάθε μέθοδος της κλάσης UML αντιστοιχεί σε μια μέθοδο της κλάσης Java

Σε πολλές περιπτώσεις είναι δυνατόν να παράγουμε αυτόματα τον σκελετό του κώδικα της Java ειδικά όταν έχουμε χρησιμοποιήσει για την ανάπτυξη του μοντέλου UML ένα ολοκληρωμένο εργαλείο CASE (Computer Aided Software Engineering).

Για παράδειγμα, ο σκελετός του κώδικα java που παράγεται για την κλάση Employee με βάση το διάγραμμα κλάσεων είναι:



Εικόνα 39: Δημιουργία κλάσεων java από διάγραμμα κλάσεων

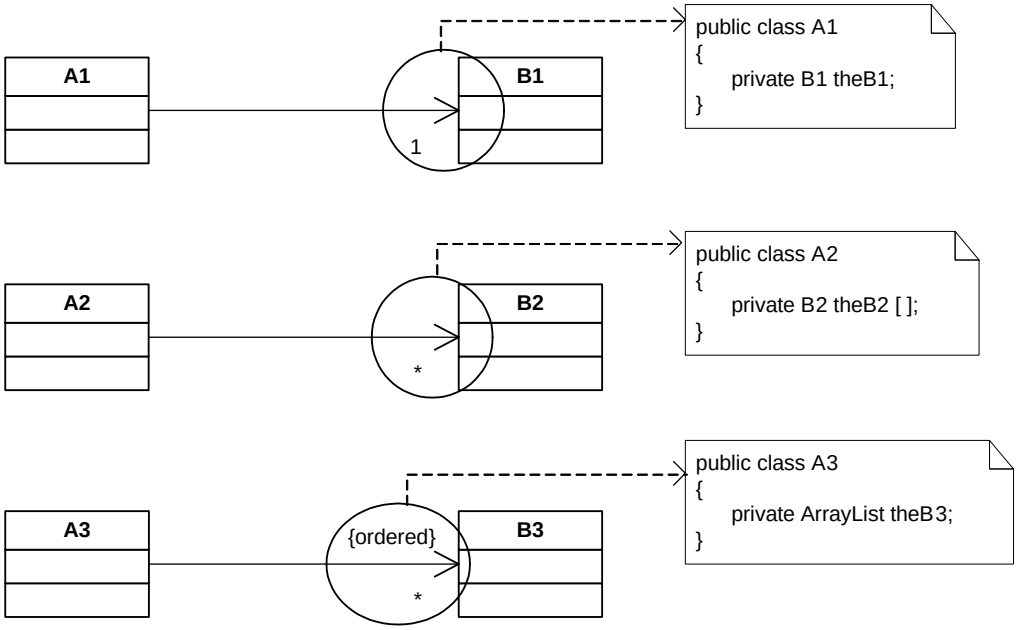
Δραστηριότητα 8

Δώστε το σκελετό κώδικα java για την κλάση PurchaseOrders.

Ένα ακόμη σημαντικό θέμα που θα μας απασχολήσει στη δημιουργία του κώδικα είναι ο μετασχηματισμός σχέσεων συσχέτισης, συναρμολόγησης και σύνδεσης σε κώδικα java. Η Εικόνα 40 παρουσιάζει τρεις περιπτώσεις μετασχηματισμού σχέσεων σε κώδικα java. Έτσι έχουμε:

- Για μια σχέση με πολλαπλότητα 1-1, ορίζουμε ένα αντικείμενο τύπου B1 (theB1) μέσα στην κλάση A1.
- Για μια σχέση 1 προς *, ορίζουμε έναν πίνακα αντικειμένων (theB2s []) μέσα στην κλάση A2.
- Για μια διατεταγμένη σχέση (ordered) 1 προς *, ορίζουμε μια συλλογή τύπου ArrayList ή Vector μέσα στην κλάση A2. Η διάταξη, στην περίπτωση που έχουμε πολλές τιμές (π.χ. έναν πίνακα τιμών) σημαίνει πως έχει σημασία η σειρά των τιμών

(ποια είναι πρώτη, ποια δεύτερη κ.λπ.) και επομένως, κατά την υλοποίηση της συγκεκριμένης κλάσης με κάποια γλώσσα προγραμματισμού, θα πρέπει να χρησιμοποιηθεί μία συλλογή που σέβεται τη διάταξη (π.χ. μία λίστα) και όχι κάποια που δεν εγγυάται τη διάταξη (π.χ. ένα σύνολο).

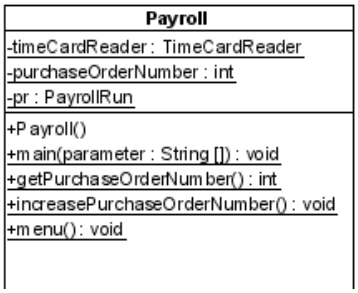


Εικόνα 40: Μετασχηματισμός σχέσεων συσχέτισης σε κώδικα java

Στις επόμενες παραγράφους θα προχωρήσουμε στην αναλυτική παρουσίαση και σχολιασμό των κλάσεων που απαρτίζουν το σύστημα.

5.2 Η κλάση Payroll

Η κλάση Payroll ανήκει στο πακέτο payroll και αποτελεί το σημείο εκκίνησης της εφαρμογής του συστήματος της μισθοδοσίας.



Εικόνα 41: Η κλάση Payroll

[91]

5.2.1 Η δομή ενός αρχείου java

Κάθε αρχείο πηγαίου κώδικα java (*.java) έχει μια συγκεκριμένη δομή η οποία η ακόλουθη.

- Ορισμός πακέτου. Κάθε κλάση μπορεί να ανήκει μόνο σε ένα πακέτο. Για την κλάση payroll έχουμε

```
package payroll;
```

- Δηλώσεις των import. Σε κάθε αρχείο μπορούν να υπάρχουν πολλά import. Το ποια είναι αυτά εξαρτάται από τις κλάσεις που χρησιμοποιούνται στην κλάση

```
import java.util.*;
import java.io.*;
import java.text.*;
import java.sql.*;
```

- Δήλωση του σώματος της κλάσης

```
public class Test {
       // Δηλώσεις πεδίων
       ….
       // Δηλώσεις μεθόδων
       // Κατασκευαστής
       Public Test() {}
       // Ορισμός άλλων μεθόδων
       …..
}
```

Πιο συγκεκριμένα για την κλάση Payroll έχουμε

```
public class Payroll {

// Δηλώσεις πεδίων

// Ορίζουμε τον αναγνώστη καρτών
// Θεωρούμε ότι υπάρχει μόνο ένας και γι' αυτό έχει δηλωθεί ως
// static. Θυμίζουμε ότι η δεσμευμένη λέξη static ορίζει μια
// μεταβλητή η οποία είναι κοινή για όλα τα αντικείμενα της
// κλάσης
private static TimeCardReader timeCardReader;

// Αριθμός Παραγγελίας. Αυξάνεται σε κάθε νέα παραγγελία
private static int purchaseOrderNumber = 0;

// Ορίζουμε ένα αντικείμενο με όνομα pr που είναι τύπου
// PayrollRun
private PayrollRun pr;

// Ορίζουμε τις μεθόδους της κλάσης Payroll
// Ο κατασκευαστής της κλάσης PayrollRun
public PayrollRun () {}

// Η κλάση main αποτελεί το σημείο εκκίνησης της εφαρμογής
public static void main(String[] args)
```

```
{

        System.out.println("PAYROLL APPLICATION STARTING …");
        // Φόρτωση των εγγραφών από τη βάση δεδομένων
        // και δημιουργία αντικειμένων. Καλούμε τη μέθοδο
        // initPersistentStorage της κλάσης PersistentStorage
        PersistentStorage.initPersistentStorage();

        // Δημιουργούμε το αντικείμενο pr με τη χρήση της
        // δεσμευμένης λέξης new
        pr = new PayrollRun();

        // Καλούμε το μενού της εφαρμογής.
        // Το menu() αποτελεί μέθοδο της κλάσης Payroll
        menu();
}


// Η μέθοδος getPurchaseOrderNumber μας επιστέφει έναν ακέραιο
// τον τρέχοντα αριθμό της παραγγελίας.
// Η μεταβλητή purchaseOrderNumber έχει δηλωθεί ως static ώστε
// ο αριθμός παραγγελίας να είναι μοναδικός
public static int getPurchaseOrderNumber()
{
        return purchaseOrderNumber;
}

// Η μέθοδος getPurchaseOrderNumber αυξάνει κατά ένα
// τον τρέχοντα αριθμό της παραγγελίας.
public static void increasePurchaseOrderNumber()
{
        purchaseOrderNumber++;
}
```

5.2.2 Η μέθοδος menu()

Η μέθοδος menu() παρουσιάζει το μενού των επιλογών που είναι διαθέσιμες στο χρήστη. Η δομή της μεθόδου menu() είναι απλή:

1. Παρουσιάζονται οι διαθέσιμες επιλογές
2. Ο χρήστης κάνει την επιλογή του
3. Καλείται η μέθοδος που αντιστοιχεί στην επιλογή του χρήστη

Τα τρία παραπάνω βήματα βρίσκονται μέσα σε μια εντολή επανάληψης while. Η εντολή while εκτελεί συνεχώς ένα σύνολο εντολών, όσο η συνθήκη ελέγχου είναι αληθής. Το συντακτικό της εντολής είναι το ακόλουθο:

```
while (expression) {
        statement(s)
}
```

Στην περίπτωση της μεθόδου menu() η συνθήκη ελέγχου είναι η μεταβλητή choice

να είναι διαφορετική του μηδενός. Η μεταβλητή choice παίρνει τιμή με τη βοήθεια της κλάσης Scanner. Η κλάση Scanner χρησιμοποιείται για την ανάλυση κειμένου και την αναγνώριση απλών τύπων όπως ακέραιοι και string. Συνήθως, η κλάση Scanner χρησιμοποιείται για την ανάγνωση δεδομένων εισόδου από το πληκτρολόγιο. Έτσι, το παρακάτω τμήμα κώδικα

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
```

διαβάζει τον επόμενο ακέραιο από το System.in. Προφανώς, η χρησιμότητα της κλάσης Scanner δεν περιορίζεται στην αναγνώριση ακεραίων αλλά μπορεί να χρησιμοποιηθεί για την αναγνώριση άλλων τύπων.

Η μέθοδος menu συνεχίζει με μια εντολή switch. Μια κοινή πρακτική στις γλώσσες προγραμματισμού είναι να συγκρίνουμε μια μεταβλητή με κάποια τιμή και σε περίπτωση που δεν ταιριάζουν να τη συγκρίνουμε με άλλη τιμή κ.ο.κ. Η χρήση προτάσεων if-else δεν ενδείκνυται σε τέτοιες περιπτώσεις διότι ενδέχεται να καταλήξει σε πολύπλοκο κώδικα. Για τέτοιες περιπτώσεις χρησιμοποιείται μια εξειδικευμένη πρόταση switch, η οποία παρέχει έναν εύχρηστο μηχανισμό που ομαδοποιεί τους ελέγχους και τις δράσεις μαζί σε μια και μόνο πρόταση.

Η switch μερικές φορές αναφέρεται και σαν πρόταση επιλογής διότι επιλέγει για εκτέλεση ένα από πολλά τμήματα κώδικα με βάση την τιμή μιας έκφρασης ελέγχου. Η μορφή της πρότασης switch φαίνεται παρακάτω:

```
switch (expression) {
       case condition_1: code_block_1; break;
       case condition_2: code_block_2; break;
       ...
       case condition_n: code_block_n; break;
       default: code_block_default;
}
```

Στη συνέχεια παραθέτουμε τον κώδικα της μεθόδου menu()

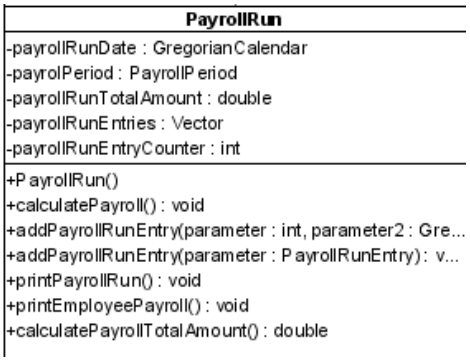
```
public static void menu()
{

     Scanner keyboard = new Scanner(System.in);
     int choice = 999;          //Αρχικοποίηση της επιλογής

     while (choice != 0) {
            // Εμφανίζει το ακόλουθο μενού μέχρι,
            // να επιλέξουμε την επιλογή 0 για Έξοδο
```

```
 System.out.println("\n\n\t\t ΣΥΣΤΗΜΑ ΜΙΣΘΟΔΟΣΙΑΣ ");
 System.out.println("\t------------------------------ ");
 System.out.println("\t 1. Εκτύπωση Στοιχείων Εργαζομένων");
 System.out.println("\t 2. Δημιουργία και Καταγραφή Ωρών
                          Εργασίας");
 System.out.println("\t 3. Εκτύπωση Ωρών Εργασίας");
 System.out.println("\t 4. Εισαγωγή Παραγγελιών");
 System.out.println("\t 5. Εκτύπωση Παραγγελιών");
 System.out.println("\t 6. Αναλυτική Εκτύπωση Παραγγελιών");
 System.out.println("\t 7. Μισθοδοσία");
 System.out.println("\t 8. Συγκεντρωτική Κατάσταση
                          Μισθοδοσίας");
 System.out.println("\t 9. Κατάσταση Μισθοδοσίας Εργαζομένου");
 System.out.println("\t 0. Έξοδος ");
 System.out.println("\t ------------------------------ ");
 System.out.print("\t\t Επιλογή --> ");

//Περιμένει να πατήσουμε ένα πλήκτρο
 choice = keyboard.nextInt();
 switch (choice) {

 case 1: // Εκτύπωση στοιχείων εργαζομένων
        PersistentStorage.printEmployees();
        break;

 case 2: // Δημιουργία και Καταγραφή Ωρών Εργασίας
        PersistentStorage.clearLogEntriesFromDB();
        timeCardReader = new TimeCardReader();

        timeCardReader.recordLogEntries(PersistentStorage.getEmp
        loyees());
        break;

 case 3: // Εκτύπωση Ωρών Εργασίας
        PersistentStorage.printLogEntries();
        break;

 case 4: // Δημιουργία Φόρμας Παραγγελιών
        showOrderFrame();
        break;

 case 5: // Εκτύπωση Παραγγελιών
        PersistentStorage.printOrders();
        break;

 case 6: // Αναλυτική Εκτύπωση Παραγγελιών
        PersistentStorage.printOrderDetails();
        break;

 case 7: // Υπολογισμός Μισθοδοσίας
        pr.misthodosia();
        break;

 case 8: // Δημιουργία συγκεντρωτικής κατάστασης μισθοδοσίας
        pr.printPayrollRun();
        pr.calculatePayrollTotalAmount();
        break;

case 9: // Δημιουργία κατάστασης μισθοδοσίας εργαζομένου
        pr.printEmployeePayroll();
        break;
```

```
            case 0:
                    System.exit(0);
                    break;

             default:
                    System.out.print("\tΕπιλογή --> ");
             }
      }
}
```

5.3 Κλάση PayrollRun

Η κλάση PayrollRun ανήκει στο πακέτο payroll και αποτελεί τη βασική κλάση που χρησιμοποιείται για τον υπολογισμό της μισθοδοσίας. Όπως παρουσιάσαμε και στην Εικόνα 35 η κλάση PayrollRun συνδέεται με σχέση σύνθεσης με την κλάση PayrollRunEntry.



Εικόνα 42: Η κλάση PayrollRun

Για την υλοποίηση αυτής της σχέσης χρησιμοποιούμε την κλάση Vector και τη δήλωση

```
private Vector<PayrollRunEntry> payrollRunEntries;
```

5.3.1 Οι συλλογές Java

Στο σημείο αυτό θα πρέπει να πούμε λίγα λόγια για τις συλλογές της Java. Στο πακέτο java.util της βασικής βιβλιοθήκης της Java παρέχονται κλάσεις για την υλοποίηση συχνά χρησιμοποιούμενων δομών δεδομένων που είναι αναγκαίες σε πολλές εφαρμογές. Μια συλλογή είναι ένας αριθμός πραγμάτων που ομαδοποιούνται με κάποιο τρόπο.

Η Java προσφέρει ειδικές κλάσεις για να υποστηρίξει την έννοια της συλλογής αντικειμένων. Οι πιο βασικές από αυτές είναι:

- Η ***Set*** στην οποία τα στοιχεία είναι αντικείμενα οποιουδήποτε είδους ενώ δεν επιτρέπονται δύο ίδια αντικείμενα να ανήκουν στο ίδιο set,
- Η ***List*** όπου τα στοιχεία είναι πάλι αντικείμενα οποιουδήποτε είδους αλλά επιτρέπονται ίδια αντικείμενα,
- Η ***Map*** όπου τα στοιχεία είναι ζεύγη αντικειμένων <κλειδί, τιμή> και δεν επιτρέπονται ίδια κλειδιά.
- Η ***ArrayList*** όπου τα στοιχεία της έχουν συγκεκριμένη διάταξη (index). Μια arraylist είναι παρόμοια με έναν πίνακα αλλά είναι δυναμική, δηλαδή δε χρειάζεται να ανησυχούμε μη βγούμε εκτός των ορίων της. Όταν γεμίσει δημιουργεί αυτόματα μια νέα λίστα 10% μεγαλύτερη και αντιγράφει τα δεδομένα από την παλιά λίστα στη νέα.
- Η ***Vector*** που είναι ίδια με την ArrayList αλλά σχεδιασμένη για ασφαλή χρήση σε multithreaded περιβάλλοντα (είναι synchronized αλλά και πιο αργή).
- Η ***HashMap*** που είναι μια συλλογή ζευγών (κλειδί/τιμή), χρησιμοποιεί τη μέθοδο του hashcode για την αποθήκευση των στοιχείων στη συλλογή ενώ προσφέρει αποδοτική ανάκτηση των δεδομένων. Το HashCode είναι ένας τρόπος για να υπολογίζουμε ένα μικρό (32 bit) αριθμητικό κλειδί είτε από ένα αλφαριθμητικό είτε από μια ομάδα bytes και με βάση αυτό αναζητούμε πολύ πιο αποδοτικά τα δεδομένα που έχουμε αποθηκευμένα.

Στη συγκεκριμένη υλοποίηση χρησιμοποιούμε την κλάση Vector. Η κλάση Vector, όπως και η HashMap, ανήκουν σε μια ειδική κατηγορία τύπων της Java που ονομάζονται γενικές (generics). Ένα generic δέχεται έναν ή περισσότερους τύπους (κλάσεις ή διεπαφές) ως παραμέτρους:

```
payrollRunEntries = new Vector<PayrollRunEntry>();
```

Στο παραπάνω τμήμα κώδικα ορίζεται το Vector payrollRunEntries με παράμετρο τον τύπο PayrollRunEntry. Με αυτό τον τρόπο δηλώνεται ότι τα στοιχεία που προστίθενται στο συγκεκριμένο Vector πρέπει να είναι του τύπου που δηλώνεται ως παράμετρος. Έτσι η επόμενη εντολή προσθέτει ένα ακόμη PayrollRunEntry στο Vector payrollRunEntries.

```
payrollRunEntries.add(curPayrollRunEntry);
```

Επιπλέον, για την προσπέλαση των στοιχείων του Vector, η εντολή for της Java έχει μια ειδική σύνταξη:

```
        for (PayrollRunEntry element : payrollRunEntries)
              System.out.printf("%s ", element.toStringEmpl());
```

5.3.2 Ο κώδικας της κλάσης PayrollRun

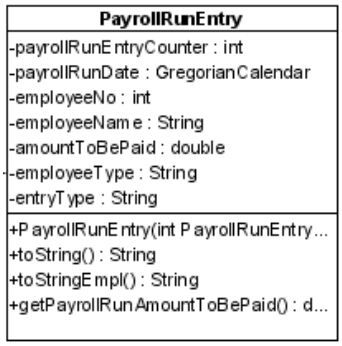
Στη συνέχεια παραθέτουμε τον κώδικα της κλάσης PayrollRun με αναλυτικά σχόλια:

```
public class PayrollRun {
       // η ημερομηνία εκτέλεσης μισθοδοσίας
       private GregorianCalendar PayrollRunDate;
       // η περίοδος μισθοδοσίας
       private PayrollPeriod payrolPeriod;
       // το συνολικό ποσό μισθοδοσίας
       private double PayrollRunTotalAmount;
       // ο Vector με τις εγγραφές μισθοδοσίας
       private Vector<PayrollRunEntry> payrollRunEntries;
       // ο μετρητής εγγραφών
       private int RayrollRunEntryCounter;

       // ο κατασκευαστής της κλάσης PayrollRun
       public PayrollRun() {
              // αρχικοποίηση των μεταβλητών
              PayrollRunDate = new GregorianCalendar();
              payrollRunEntries = new Vector<PayrollRunEntry>();
              RayrollRunEntryCounter = 0;
              PayrollRunTotalAmount = 0;
       }

       // Υπολογισμός μισθοδοσίας εργαζομένων
       public void calculatePayroll() {

              System.out.println("ΥΠΟΛΟΓΙΣΜΟΣ ΜΙΣΘΟΔΟΣΙΑΣ ΕΡΓΑΖΟΜΕΝΩΝ ....");

              // αρχικοποίηση του vector
              payrollRunEntries.clear();

              // φορτώνουμε τα δεδομένα από τη βάση δεδομένων στο
              // Vector employees.
              // Χρησιμοποιούμε την κλάση PersistentStorage
              Vector<Employee> employees = PersistentStorage.getEmployees();

              for (int i = 0; i < employees.size(); i++)
              {

              // Σύνθετη κλήση
              // α) Παίρνουμε τον επόμενο εργαζόμενο employees.get(i)
              // β) Υπολογίζουμε το μισθό employees.get(i).calculatePayroll
              // γ) Προσθέτουμε μια εγγραφή στο Vector payrollRunEntries
                     payrollRunEntries.add(
                            employees.get(i).calculatePayroll(
                                   RayrollRunEntryCounter,PayrollRunDate));

              // Ελέγχουμε εάν η κλάση του εργαζομένου που εξετάζουμε
              // είναι CommissionedEmployee
              if (employees.get(i).getClass().getName().equals(
                     "employees.CommissionedEmployee"))

              // Υπολογίζουμε το bonus
              payrollRunEntries.add(((CommissionedEmployee)employees.get(i)).
                     calculateBonus(RayrollRunEntryCounter,PayrollRunDate));
              RayrollRunEntryCounter++;
```

```
        }
    }

    // Προσθέτουμε μια εγγραφή μισθοδοσίας
     public void addPayrollRunEntry( int RayrollRunEntryCounter,
                                     GregorianCalendar PayrollRunDate,
                                     int EmployeeNo, String EmployeeName,
                                     double amountToBePaid,
                                     String EmployeeType,
                                     String EntryType) {

          PayrollRunEntry curPayrollRunEntry;
          // Δημιουργούμε ένα PayrollRunEntry
          // και το αποθηκεύουμε
          curPayrollRunEntry = new PayrollRunEntry(
                 RayrollRunEntryCounter, PayrollRunDate,
                 EmployeeNo, EmployeeName, amountToBePaid,
                 EmployeeType, EntryType);

          payrollRunEntries.add(curPayrollRunEntry);
     }


     // Πρόσθεση εγγραφής μισθοδοσίας
     public void addPayrollRunEntry(PayrollRunEntry curPayrollRunEntry) {
         payrollRunEntries.add(curPayrollRunEntry);
     }


    // Εκτύπωση συνολικής μισθοδοσίας
    public void printPayrollRun()
    {
     System.out.println("ΣΥΓΚΕΝΤΡΩΤΙΚΗ ΚΑΤΑΣΤΑΣΗ ΜΙΣΘΟΔΟΣΙΑΣ ΕΡΓΑΖΟΜΕΝΩΝ
....");
          if (payrollRunEntries.isEmpty())
              System.out.println("ΔΕΝ ΥΠΑΡΧΟΥΝ ΠΛΗΡΩΜΕΣ ΜΙΣΘΟΔΟΣΙΑΣ");

          else { // διατρέχουμε όλα τα elements του vector
              System.out.println("ΕΓΓΡΑΦΕΣ ΜΙΣΘΟΔΟΣΙΑΣ");
              // output elements
              for (PayrollRunEntry element : payrollRunEntries)
                  System.out.printf("%s ", element.toString());
          } // end else

          System.out.println("\n");
    }

    // Εκτύπωση  μισθοδοσίας εργαζομένου
    public void printEmployeePayroll() {
           System.out.println("ΜΙΣΘΟΔΟΣΙΑ ΕΡΓΑΖΟΜΕΝΟΥ");
           if (payrollRunEntries.isEmpty())
              System.out.println("ΔΕΝ ΥΠΑΡΧΟΥΝ ΠΛΗΡΩΜΕΣ ΜΙΣΘΟΔΟΣΙΑΣ");

           else { // διατρέχουμε όλα τα elements του vector
                  System.out.println("ΕΓΓΡΑΦΕΣ ΜΙΣΘΟΔΟΣΙΑΣ");
                  // output elements
                  for (PayrollRunEntry element : payrollRunEntries)
                         System.out.printf("%s ", element.toStringEmpl());
           } // end else
           System.out.println("\n");
    }
```

```
        // Συνολικό ποσο μισθοδοσίας
        public double calculatePayrollTotalAmount() {
                PayrollRunTotalAmount=0;

                if ( payrollRunEntries.isEmpty() )
                {
                        System.out.println(
                                "ΔΕ ΜΠΟΡΩ ΝΑ ΥΠΟΛΟΓΙΣΩ ΤΟ ΣΥΝΟΛΙΚΟ ΠΟΣΟ");
                        System.out.println("ΔΕΝ ΥΠΑΡΧΟΥΝ ΕΓΓΡΑΦΕΣ ΜΙΣΘΟΔΟΣΙΑΣ");
                }
                else  // διατρέχουμε όλα τα elements του vector
                {
                        for ( PayrollRunEntry element : payrollRunEntries )
                                PayrollRunTotalAmount=PayrollRunTotalAmount+
                                        element.getPayrollRunAmountToBePaid();
                } // end else
                System.out.println( "Συνολικό ποσό για μισθοδοσία: " +
                        PayrollRunTotalAmount );
                return(PayrollRunTotalAmount);
        }
}
```

5.4 Κλάση PayrollRunEntry



Εικόνα 43: Η κλάση PayrollRunEntry

Στη συνέχεια παρουσιάζεται ο κώδικας της κλάσης PayrollRunEntry. Στην κλάση αυτή, ο αναγνώστης θα πρέπει να κατανοήσει

1. τη χρήση της δεσμευμένης λέξης this,
2. τη διαχείριση των ημερομηνιών, καθώς και
3. το ρόλο των μεθόδων toString().

5.4.1 Η δεσμευμένη λέξη this

Η λέξη this χρησιμοποιείται για να αναφέρουμε τις μεταβλητές ή τις μεθόδους του

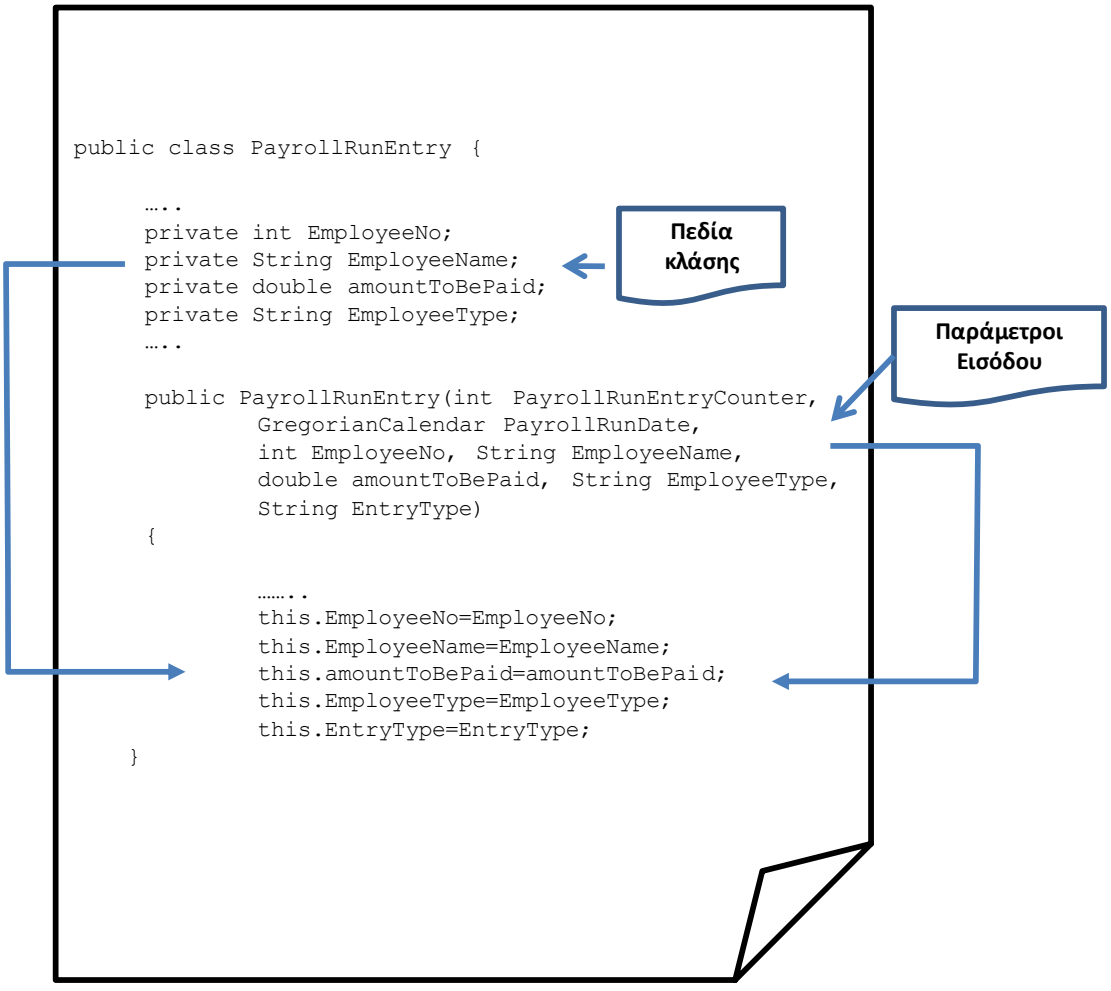
ίδιου του αντικείμενου. Για να κατανοήσουμε τη χρήση του this θα πρέπει να δούμε τις δηλώσεις των πεδίων μέσα στην κλάση σε συνδυασμό με τον τρόπο που αυτά χρησιμοποιούνται μέσα στις μεθόδους. Έτσι για παράδειγμα, δηλώνουμε τη μεταβλητή

```
private int EmployeeNo;  // Κωδικός εργαζομένου
```

και στη συνέχεια μέσα στον κατασκευαστή της κλάσης έχουμε

```
...
this.EmployeeNo=EmployeeNo;
...
```

όπου με this.EmployeeNo αναφερόμαστε στη μεταβλητή της κλάσης ενώ με EmployeeNo αναφερόμαστε στην παράμετρο εισόδου της μεθόδου.



Εικόνα 44: Η δεσμευμένη λέξη this

5.4.2 Διαχείριση των ημερομηνιών

Η γλώσσα Java, για τη διαχείριση του χρόνου και των ημερομηνιών, μετρά τον αριθμό των milliseconds που πέρασαν από την 1η Ιανουαρίου 1970. Αυτό σημαίνει ότι, η 2α Ιανουαρίου 1970, ξεκίνησε 86,400,000 milliseconds αργότερα. Η αναπαράσταση του αριθμού των milliseconds που πέρασαν από αυτή την ημερομηνία μπορεί να γίνει με τη χρήση ενός long αριθμού. Αυτός είναι ακριβώς, ο τρόπος με τον οποίο, η βασική κλάση για τη διαχείριση των ημερομηνιών Date, διαχειρίζεται τις ημερομηνίες. Αξίζει να σημειωθεί ότι εάν η ημερομηνία που θέλουμε να αναπαραστήσουμε είναι πριν την 1/1/1970, τότε ο αριθμός long είναι αρνητικός, ενώ ένας long αριθμός μπορεί να παράγει ημερομηνίες μέχρι και 290.000.000 χρόνια πριν ή μετά την 1/1/1970.

Επομένως, η κλάση Date που βρίσκεται στο πακέτο java.util ενθυλακώνει ένα long αριθμό, ο οποίος αναπαριστά μια συγκεκριμένη χρονική στιγμή. Όταν δημιουργούμε ένα αντικείμενο της κλάσης Date, με τη χρήση του κατασκευαστή Date(), ουσιαστικά δημιουργούμε ένα long αριθμό, ο οποίος αναπαριστά τη στιγμή δημιουργίας του αντικειμένου. Στο παρακάτω παράδειγμα χρησιμοποιούμε τον κατασκευαστή Date() για να δημιουργήσουμε ένα αντικείμενο, ενώ με τη μέθοδο getTime() βρίσκουμε τον αριθμό των milliseconds που αναπαριστούν αυτή την ημερομηνία:

```
import java.util.*;
public class Now
{
	public static void main(String[] args)
	{
		Date now = new Date();
		long nowLong = now.getTime();
		System.out.println("Value is " + nowLong);
	}
}
```

Το αποτέλεσμα του παραπάνω προγράμματος εξαρτάται από τη στιγμή που θα τρέξει κανείς το πρόγραμμα. Για τη μετατροπή του αριθμού που προκύπτει σε String χρησιμοποιούμε την κλάση DateFormat. Ο τρόπος απεικόνισης της πληροφορίας εξαρτάται από το format που έχει επιλεγεί. Για τη χρήση του default format και τη μετατροπή της ημερομηνίας σε string θα πρέπει να εκτελέσουμε:

```
DateFormat df = DateFormat.getDateInstance();
String s = df.format(now);
```

όπου now είναι τύπου Date.

Η μέθοδος getDateInstance δημιουργεί ένα αντικείμενο με default format. Εναλλακτικά μπορούμε να χρησιμοποιήσουμε τη μέθοδο με παραμέτρους SHORT, MEDIUM, LONG, ή FULL, ώστε να εμφανισθεί η ημερομηνία με διαφορετικό τρόπο. Έτσι ανάλογα με τον τρόπο μορφοποίησης που επιλέγουμε η ημερομηνία εμφανίζεται με διαφορετικό τρόπο. Έτσι οι εντολές:

```
DateFormat df =  DateFormat.getDateInstance();
DateFormat df1 = DateFormat.getDateInstance(DateFormat.SHORT);
DateFormat df2 = DateFormat.getDateInstance(DateFormat.MEDIUM);
DateFormat df3 = DateFormat.getDateInstance(DateFormat.LONG);
DateFormat df4 = DateFormat.getDateInstance(DateFormat.FULL);
```

παράγουν αντίστοιχα

```
(Default) Nov 8, 2006
(SHORT)   11/8/06
(MEDIUM)  Nov 8, 2006
(LONG)    November 8, 2006
(FULL)    Wednesday, November 8, 2006
```

Το ίδιο πρόγραμμα με διαφορετικές ρυθμίσεις (regional settings) θα έδινε:

```
(Default) 2006-Νοε-08
(SHORT)   2006-11-08
(MEDIUM)  2006-Νοε-08
(LONG)    8 Νοεμβρίου 2006
(FULL)    Τετάρτη, 8 Νοεμβρίου 2006
```

Για τη μετατροπή ενός string σε ημερομηνία θα πρέπει να εκτελέσουμε μια εντολή της μορφής

```
String ds = "November 1, 2006";
DateFormat df = DateFormat.getDateInstance();
Date d = df.parse(ds);
```

Η μέθοδος parse() μετατρέπει το string ds σε ημερομηνία και είναι ένας τρόπος για να δημιουργούμε συγκεκριμένες ημερομηνίες.

Την ίδια λειτουργικότητα και με πιο απλό τρόπο μπορούμε να επιτύχουμε με την χρήση της κλάσης GregorianCalendar, η οποία υλοποιεί το Γρηγοριανό Ημερολόγιο το οποίο είναι σε χρήση σήμερα. Η παρακάτω εντολή δημιουργεί ένα αντικείμενο με ημερομηνία 20 Σεπτεμβρίου 2007:

```
GregorianCalendar testDate = new GregorianCalendar(2007, 8, 20);
```

Στην παραπάνω εντολή, η μέτρηση του μήνα ξεκινάει από τον αριθμό 0. Επομένως, ο Σεπτέμβριος εμφανίζεται με τον αριθμό 8 και όχι με τον αριθμό 9 όπως θα περίμενε κανείς.

Αντίστοιχα, θα εργασθούμε για τη μετατροπή του GregorianCalendar σε string. Ο παρακάτω κώδικας παρουσιάζει τον τρόπο εργασίας

```
GregorianCalendar testDate = new GregorianCalendar(2007, 8, 20);
Date d = testDate.getTime();
DateFormat df = DateFormat.getDateInstance();
String s = df.format(d);
System.out.println("Test date is" + s);
```

Για να αρχικοποιήσουμε ένα αντικείμενο τύπου GregorianCalendar στη σημερινή ημερομηνία θα πρέπει

```
GregorianCalendar thisday = new GregorianCalendar();
```

5.4.3 Η μέθοδος toString

Η μέθοδος toString είναι μια μέθοδος που συναντάμε πολύ συχνά. Ο λόγος ύπαρξής της είναι ότι πολύ συχνά υπάρχει ανάγκη να μετατρέψουμε τα περιεχόμενα ενός αντικειμένου σε string για εκτύπωση, για επεξεργασία ή για οποιοδήποτε άλλο τρόπο. Η μέθοδος αυτή χρησιμοποιείται ευρέως στο παράδειγμά μας.

```
// Για παρουσίαση στοιχείων ανά εργαζόμενο
public String toStringEmpl() {
       // Προσωρινές μεταβλητές για την  επεξεργασία των ημερομηνιών
       Date tempDate1;
       DateFormat df1;
       String s1;
       // Μετατρέπω το GregorianCalendar object σε Date
       // Η μέθοδος getTime επιστρέφει Date
       tempDate1 = PayrollRunDate.getTime();
       df1 = DateFormat.getDateTimeInstance(DateFormat.SHORT,
DateFormat.SHORT);
       // Μετατρέπω τις ημερομηνίες σε String
       s1 = df1.format(tempDate1);
       return  "        ΜΙΣΘΟΔΟΣΙΑ ΕΡΓΑΖΟΜΕΝΟΥ \n" +
               " _____ \n" +
               " ΚΩΔΙΚΟΣ ΕΡΓ.:" + EmployeeNo +  "\n"+
               " ΟΝΟΜΑ ΕΡΓ.:" + EmployeeName +  "\n" +
               " ΗΜΕΡΟΜΗΝΙΑ:" + s1 + "\n" +
               " ΚΑΤΗΓΟΡΙΑ ΕΡΓ.:" + EmployeeType +  "\n" +
               " ΕΙΔΟΣ ΠΛ.:"+ EntryType +  "\n" +
               " ΠΟΣΟ ΠΛΗΡΩΜΗΣ:" + amountToBePaid + "\n" +
               " _____ \n\n\n";
    }
```

Το παραγόμενο string μπορεί πολύ εύκολα να εκτυπωθεί. Με τον τρόπο αυτό βελτιώνουμε σημαντικά τη δομή του κώδικά μας.

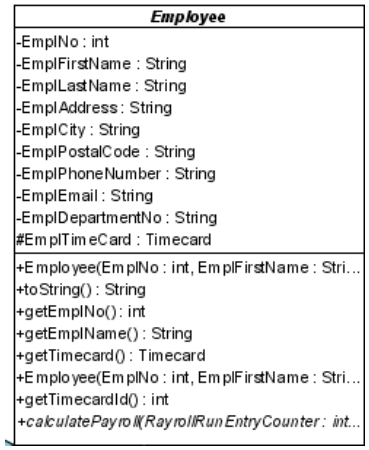
5.4.4 Ο κώδικας της κλάσης PayrollRunEntry

```
public class PayrollRunEntry
{

       private int PayrollRunEntryCounter;      // Μετρητής εγγραφών
```

```
// Ημερομηνία δημιουργίας εγγραφής

private GregorianCalendar PayrollRunDate;
private int EmployeeNo;           // Κωδικός εργαζομένου
private String EmployeeName;      // Ονοματεπώνυμο εργαζομένου
private double amountToBePaid;    // Ποσό πληρωμής
private String EmployeeType;      // Μισθωτός, Ωρομίσθιος
private String EntryType;         // Μισθός , Bonus


// Ο κατασκευαστής
public PayrollRunEntry(int PayrollRunEntryCounter,
                       GregorianCalendar PayrollRunDate,
                       int EmployeeNo, String EmployeeName,
                       double amountToBePaid, String EmployeeType,
                       String EntryType)
{
      this.PayrollRunEntryCounter= PayrollRunEntryCounter;
      this.PayrollRunDate=PayrollRunDate;
      this.EmployeeNo=EmployeeNo;
      this.EmployeeName=EmployeeName;
      this.amountToBePaid=amountToBePaid;
      this.EmployeeType=EmployeeType;
      this.EntryType=EntryType;
}

public String toString()
{
 // Προσωρινές μεταβλητές για την  επεξεργασία των ημερομηνιών
      Date tempDate1;
      DateFormat df1;
      String s1;
      // Μετατρέπω το GregorianCalendar object σε Date
      // Η μέθοδος getTime επιστρέφει Date
      tempDate1 = PayrollRunDate.getTime();
      df1 = DateFormat.getDateTimeInstance(
                  DateFormat.SHORT, DateFormat.SHORT);
      // Μετατρέπω τις ημερομηνίες σε String
      s1 = df1.format(tempDate1);
      return  "ΑΡ. ΕΓΓΡΑΦΗΣ: " + PayrollRunEntryCounter + ", "
      +      " ΗΜΕΡΟΜΗΝΙΑ:" + s1 + ", " +
             " ΚΩΔΙΚΟΣ ΕΡΓ.:" + EmployeeNo +  ", "+
             " ΟΝΟΜΑ ΕΡΓ.:" + EmployeeName +  ", " +
             " ΚΑΤΗΓΟΡΙΑ ΕΡΓ.:" + EmployeeType +  ", " +
             " ΕΙΔΟΣ ΠΛ.:"+ EntryType +  ", " +
             " ΠΟΣΟ ΠΛΗΡΩΜΗΣ:" + amountToBePaid + "\n";
}

// Για παρουσίαση στοιχείων ανά εργαζόμενο
public String toStringEmpl()
{
 // Προσωρινές μεταβλητές για την  επεξεργασία των ημερομηνιών
      Date tempDate1;
      DateFormat df1;
      String s1;
      // Μετατρέπω το GregorianCalendar object σε Date
      // Η μέθοδος getTime επιστρέφει Date
      tempDate1 = PayrollRunDate.getTime();
      df1 = DateFormat.getDateTimeInstance(DateFormat.SHORT,
                  DateFormat.SHORT);
      // Μετατρέπω τις ημερομηνίες σε String
      s1 = df1.format(tempDate1);
```

```
            return  "        ΜΙΣΘΟΔΟΣΙΑ ΕΡΓΑΖΟΜΕΝΟΥ \n" +
               "  _____ \n" +
               " ΚΩΔΙΚΟΣ ΕΡΓ.:" + EmployeeNo +  "\n"+
               " ΟΝΟΜΑ ΕΡΓ.:" + EmployeeName +  "\n" +
               " ΗΜΕΡΟΜΗΝΙΑ:" + s1 + "\n" +
               " ΚΑΤΗΓΟΡΙΑ ΕΡΓ.:" + EmployeeType +  "\n" +
               " ΕΙΔΟΣ ΠΛ.:"+ EntryType +  "\n" +
               " ΠΟΣΟ ΠΛΗΡΩΜΗΣ:" + amountToBePaid + "\n" +
               "  _____ \n\n\n";
    }


    // Συνολικό ποσό μισθοδοσίας
    public double getPayrollRunAmountToBePaid()
    {
            return(amountToBePaid);
    }
}
```

5.5 Κλάση Employee



Εικόνα 45:Η κλάση Employee

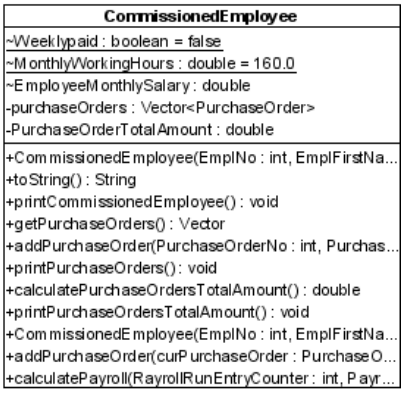
5.5.1 Ο κώδικας της κλάσης Employee

```
public abstract class Employee {

    static double PEROVERTIME = 1.5;     // Ποσοστό υπερωριών

    private int EmplNo;
    private String EmplFirstName;
    private String EmplLastName;
    private String EmplAddress;
    private String EmplCity;
    private String EmplPostalCode;
    private String EmplPhoneNumber;
    private String EmplEmail;
    private String EmplDepartmentNo;
    private Timecard EmplTimeCard;

 // Η Timecard χρειάζεται να είναι protected για να προσπελαύνεται
 // από το payroll.
 // Χρησιμοποιώντας protected μεταβλητές επιτρέπουμε προσπέλαση της
 // μεταβλητής μέσα στο ίδιο πακέτο.
```

```
// Κατασκευαστής της κλάσης Employee
public Employee(int EmplNo, String EmplFirstName,
              String EmplLastName, String EmplAddress,
              String EmplCity, String EmplPostalCode,
              String EmplPhoneNumber, String EmplEmail,
              String EmplDepartmentNo, Timecard EmplTimeCard)
{
       this.EmplNo=EmplNo;
       this.EmplFirstName=EmplFirstName;
       this.EmplLastName=EmplLastName;
       this.EmplAddress=EmplAddress;
       this.EmplCity = EmplCity;
       this.EmplPostalCode=EmplPostalCode;
       this.EmplPhoneNumber=EmplPhoneNumber;
       this.EmplEmail=EmplEmail;
       this.EmplDepartmentNo=EmplDepartmentNo;
       this.EmplTimeCard= EmplTimeCard;
}

// Η συνάρτηση toString είναι απαραίτητη
// για την εκτύπωση των δεδομένων
public String toString()
{
       return  "Κωδικός εργαζομένου :" + EmplNo + "\n" +
          "Όνομα :" + EmplFirstName + "\n" +
          "Επίθετο :" + EmplLastName + "\n" +
          "Διεύθυνση :" + EmplAddress + "\n" +
          "Πόλη :" + EmplCity + "\n" +
          "Τ.Κ. :" + EmplPostalCode + "\n" +
          "Τηλέφωνο :" + EmplPhoneNumber + "\n" +
          "Email :" + EmplEmail + "\n" +
          "Τμήμα :" + EmplDepartmentNo + "\n" +
          "Κωδικός κάρτας :" +
          EmplTimeCard.getTimecardId()+"\n";
}

// Επιστρέφει τον κωδικό του εργαζομένου
public int getEmplNo()
{
       return EmplNo;
}

// Επιστρέφει το πλήρες όνομα του εργαζομένου
// (Όνομα + Επίθετο)
public String getEmplName()
{
       return EmplFirstName + " " + EmplLastName;
}

// Επιστρέφει την κάρτα του εργαζομένου
public Timecard getTimecard()
{
       return EmplTimeCard;
}

// Επιστρέφει τον αριθμό της κάρτας εργαζομένου
public int getTimecardId()
{
       return EmplTimeCard.getTimecardId();
}
```

```
        // Με την εισαγωγή abstract μεθόδου η κλάση γίνεται abstract
        public abstract PayrollRunEntry calculatePayroll (int
                        RayrollRunEntryCounter,
                        GregorianCalendar PayrollRunDate);
    }
```

5.6 Κλάση CommissionedEmployee



Εικόνα 46: Η κλάση CommissionedEmployee

Η κλάση CommissionedEmployee συνδέεται με σχέση κληρονομικότητας με την κλάση Employee. Το γεγονός ότι υπάρχει σχέση κληρονομικότητας σημαίνει ότι κληρονομούνται όλα τα πεδία, οι μέθοδοι και συσχετίσεις της μητρικής κλάσης. Σε περίπτωση διαφορετικής υλοποίησης μιας κληρονομούμενης μεθόδου, τότε αυτή υπερφορτώνεται στην κλάση παιδί. Δεν επιτρέπεται ωστόσο κατάχρηση της επαναϋλοποίησης με κενή από κώδικα μέθοδο, η οποία ουσιαστικά ακυρώνει τη λειτουργικότητα της υποκλάσης.

Θα πρέπει να κατανοηθεί η χρήση της δεσμευμένης λέξης super η οποία χρησιμοποιείται στον κατασκευαστή της κλάσης CommissionedEmployee.

Γενικά, η super χρησιμοποιείται σε δύο περιπτώσεις:

- όταν καλούμε τον κατασκευαστή της μητρικής κλάσης (superclass) και
- όταν θέλουμε να έχουμε πρόσβαση σε κάποιο μέλος της μητρικής κλάσης που έχει αποκρυφτεί από την κλάση παιδί.

Στη συγκεκριμένη περίπτωση της κλάσης CommissionedEmployee η λέξη super χρησιμοποιείται α) για να καλέσουμε τον κατασκευαστή της μητρικής κλάσης μέσα στη

μέθοδο του κατασκευαστή και β) μέσα στη μέθοδο toString για να καλέσουμε την αντίστοιχη μέθοδο της μητρικής κλάσης.

Επιπλέον, η κλάση CommissionedEmployee υλοποιεί τη μέθοδο calculatePayroll, η οποία είχε δηλωθεί ως abstract στην κλάση Employee.

5.6.1 Ο κώδικας της κλάσης CommissionedEmployee

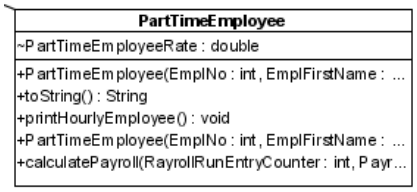
```
package employees;
import java.util.Vector;
import java.util.GregorianCalendar;

public class CommissionedEmployee extends Employee{
	static boolean Weeklypaid = false;
	static double MonthlyWorkingHours = 160.0;
	static double BONUSLOW = 10000;     // Bonus ελάχιστο ποσό
	static double BONUSMEDIUM = 20000;  // Bonus μεσαίο ποσό
	static double PERLOW = 0.05;         // Μικρό ποσοστό
	static double PERMEDIUM = 0,07;      // Μεσαίο ποσοστό
	static double PERHIGH = 0.1;        // Υψηλό ποσοστό


	double EmployeeMonthlySalary;

	// Χρησιμοποιούμε την κλάση Vector για να υλοποιήσουμε
	// έναν πίνακα παραγγελιών (PurchaseOrder)
	private Vector<PurchaseOrder> purchaseOrders;
	private double PurchaseOrderTotalAmount;

	// Κατασκευαστής
	public CommissionedEmployee(int EmplNo, String
		EmplFirstName, String EmplLastName,
		String EmplAddress, String EmplCity, String
		EmplPostalCode, String EmplPhoneNumber,
		String EmplEmail,  String EmplDepartmentNo,
		double EmployeeMonthlySalary, Timecard EmplTimeCard)
	{
	// Καλεί τον κατασκευαστή της μητρικής κλάσης Employee
		super(EmplNo, EmplFirstName, EmplLastName,
			EmplAddress, EmplCity, EmplPostalCode,
			EmplPhoneNumber, EmplEmail, EmplDepartmentNo,
			EmplTimeCard);
		this.EmployeeMonthlySalary = EmployeeMonthlySalary;
		purchaseOrders = new Vector<PurchaseOrder>();
	}

	// Προετοιμάζει τα στοιχεία του εργαζομένου
	// και τα αποθηκεύει σε ένα string
	public String toString()
	{
		return
			"ΕΡΓΑΖΟΜΕΝΟΣ ΠΩΛΗΤΗΣ \n"+
			// Χρησιμοποιεί την αντίστοιχη μέθοδο του πατέρα
			super.toString() + "Ο μισθός είναι :" +
			EmployeeMonthlySalary+"\n"+
			"-----------------------------------";
	}

	// Τυπώνει τα στοιχεία του εργαζομένου
```

```
public void printCommissionedEmployee()
{
	System.out.println(this.toString());
}

// Επιστρέφει τον πίνακα (vector) με τις παραγγελίες του πωλητή
public Vector getPurchaseOrders()
{
	return purchaseOrders;
}

// Προσθέτουμε μια νέα παραγγελία στον πωλητή
public void addPurchaseOrder(int PurchaseOrderNo,
                    GregorianCalendar PurchaseOrderDate,
                     String PurchaseOrderDesc,
                     int PurchaseOrderAmount)
{
	// Δημιουργούμε ένα PurchaseOrder
	PurchaseOrder curPurchaseOrder = new
		PurchaseOrder(PurchaseOrderNo,
		this, PurchaseOrderDate, PurchaseOrderDesc);
	// Το προσθέτουμε στις εγγραφές του συγκεκριμένου πωλητή
	purchaseOrders.add(curPurchaseOrder);
}

// Προσθέτουμε μια νέα παραγγελία στον πωλητή
public void addPurchaseOrder(PurchaseOrder curPurchaseOrder)
{
	// Το προσθέτουμε στις εγγραφές του συγκεκριμένου πωλητή
	purchaseOrders.add(curPurchaseOrder);
	printPurchaseOrders();
}


// Τυπώνουμε τις παραγγελίες του πωλητή
public void printPurchaseOrders()
{
	// Έλεγχος αν το vector είναι άδειο
	if ( purchaseOrders.isEmpty() )
		System.out.println( "Ο ΕΡΓΑΖΟΜΕΝΟΣ ΔΕΝ ΕΧΕΙ
			ΠΑΡΑΓΓΕΛΙΕΣ \n\n" ); // είναι άδειο
	else  // διατρέχουμε όλα τα elements του vector
	{
		//System.out.println( "Παραγγελίες ανά πωλητή" );
		// output elements
		for ( PurchaseOrder element : purchaseOrders )
			element.printOrderLines();
	} // end else
	System.out.println( "\n" );
}

// Υπολογίζουμε το συνολικό ποσό παραγγελιών του πωλητή
public double calculatePurchaseOrdersTotalAmount()
{
	PurchaseOrderTotalAmount=0;

	if ( purchaseOrders.isEmpty() )
		System.out.println( "ΔΕ ΜΠΟΡΕΙ ΝΑ ΓΙΝΕΙ
	ΥΠΟΛΟΓΙΣΜΟΣ: Ο ΕΡΓΑΖΟΜΕΝΟΣ ΔΕΝ ΕΧΕΙ ΠΑΡΑΓΓΕΛΙΕΣ " );
	else  // διατρέχουμε όλα τα elements του vector
	{
		for ( PurchaseOrder element : purchaseOrders )
```

```
PurchaseOrderTotalAmount = PurchaseOrderTotalAmount+
                        element.getPurchaseOrderAmount();
                } // end else
                return(PurchaseOrderTotalAmount);
        }

        // Εκτύπωση του συνολικού ποσού παραγγελιών του πωλητή
        public void printPurchaseOrdersTotalAmount()
        {
                System.out.println( "Συνολικό ποσό παραγγελιών για
εργαζόμενο: " + PurchaseOrderTotalAmount );
        }


        // Υπολογισμός μισθού πωλητών
        public PayrollRunEntry calculatePayroll(
                        int RayrollRunEntryCounter,
                        GregorianCalendar PayrollRunDate)
        {
                // Αρχικοποίηση των προσωρινών μεταβλητών της μεθόδου
                double amountToBePaid = 0;
                double hourlyWage = 0;
                double moreHours = 0;
                double lessHours = 0;

                if (EmplTimeCard.calculateWorkedHours() >
                                        MonthlyWorkingHours)
                // Οι ώρες εργασίας εργαζομένου περισσότερες κανονικού.
                // Χρειάζεται να υπολογίσουμε υπερωρίες.
                // Κάνουμε την παραδοχή ότι οι
                // υπερωρίες πληρώνονται με 50% επιπλέον
                {
                        hourlyWage = EmployeeMonthlySalary /
                                MonthlyWorkingHours;
                        moreHours = EmplTimeCard.calculateWorkedHours() -
                                MonthlyWorkingHours;
                        amountToBePaid = EmployeeMonthlySalary +
                                 moreHours * hourlyWage * PEROVERTIME;
                }
                else if (EmplTimeCard.calculateWorkedHours() = =
                                        MonthlyWorkingHours)
                // Οι ώρες εργασίας είναι όσες οι προβλεπόμενες
                // Πλήρωσε μόνο τον μισθό
                {
                        amountToBePaid = EmployeeMonthlySalary;
                }

                else
                // Οι ώρες εργασίας λιγότερες από τις κανονικές.
                // Θα πρέπει να μειωθεί ο μισθός.
                {
                        hourlyWage = EmployeeMonthlySalary /
                                MonthlyWorkingHours;
                        lessHours = MonthlyWorkingHours –
                                EmplTimeCard.calculateWorkedHours();
                        amountToBePaid = EmployeeMonthlySalary –
                                lessHours * hourlyWage;
                }

                // Δημιουργία εγγραφής μισθοδοσίας

                return new PayrollRunEntry(RayrollRunEntryCounter,
```

```
                                        PayrollRunDate,
                                        getEmplNo(),
                                        getEmplName(),
                                        amountToBePaid,
                                        "ΠΩΛΗΤΗΣ",
                                        "ΜΙΣΘΟΣ");
        }

        // Υπολογισμός bonus πωλητών

        public PayrollRunEntry calculateBonus(
                            int RayrollRunEntryCounter,
                            GregorianCalendar PayrollRunDate)
        {
               // Αρχικοποίηση των προσωρινών μεταβλητών της μεθόδου
               double weeklySalary = 0;
               double amountToBePaid = 0;
               double sales = calculatePurchaseOrdersTotalAmount();

               weeklySalary = EmployeeMonthlySalary;

               if (sales < BONUSLOW)
                      amountToBePaid = weeklySalary * PERLOW;
                             // 5% επί του εβδομαδιαίου μισθού
               else if ((sales >= BONUSLOW) && (sales < BONUSMEDIUM))
                      amountToBePaid = weeklySalary * PERMEDIUM;
                             // 7% επί του εβδομαδιαίου μισθού
               else
                      amountToBePaid = weeklySalary * PERHIGH;
                             // 10% επί του εβδομαδιαίου μισθού

          // Δημιουργία εγγραφής bonus

               return new PayrollRunEntry(RayrollRunEntryCounter,
                                        PayrollRunDate,
                                        getEmplNo(),
                                        getEmplName(),
                                        amountToBePaid,
                                        "ΠΩΛΗΤΗΣ",
                                        "BONUS");
        }

}
```

5.7 Κλάση PartTimeEmployee



Εικόνα 47: Η κλάση PartTimeEmployee

Η υλοποίηση της κλάσης PartTimeEmployee έχει σημαντικές ομοιότητες με την υλοποίηση της κλάσης CommissionedEmployee. Ενδεικτικά:

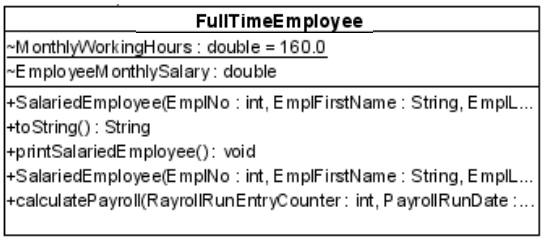
1. Η κλάση συνδέεται με σχέση κληρονομικότητας με την κλάση Employee.
2. Γίνεται χρήση μεθόδων της μητρικής κλάσης με τη χρήση της δεσμευμένης λέξης super.
3. Γίνεται υλοποίηση της αφηρημένης μεθόδου calculatePayroll.

5.7.1 Ο κώδικας της κλάσης PartTimeEmployee

```
public class HourlyEmployee extends Employee
{
	double HourlyEmployeeRate;

	// Ο κατασκευαστής
	public HourlyEmployee(int EmplNo, String EmplFirstName,
			String EmplLastName,
			String EmplAddress, String EmplCity,
			String EmplPostalCode,
			String EmplPhoneNumber,
			String EmplEmail, String EmplDepartmentNo,
			double HourlyEmployeeRate,
			Timecard EmplTimeCard)
	{
		// Καλεί τον κατασκευαστή της μητρικής κλάσης employee
		super(EmplNo, EmplFirstName, EmplLastName,
			EmplAddress, EmplCity, EmplPostalCode,
			EmplPhoneNumber, EmplEmail, EmplDepartmentNo,
			EmplTimeCard);
		this.HourlyEmployeeRate = HourlyEmployeeRate;
	}

	// Προετοιμάζει τα στοιχεία του εργαζομένου
	// και τα αποθηκεύει σε ένα string
	public String toString()
	{
		return
			"ΕΡΓΑΖΟΜΕΝΟΣ ΜΕ ΩΡΟΜΙΣΘΙΟ \n" +
			super.toString() + "Το ωρομίσθιο είναι :" +
			HourlyEmployeeRate +
			"\n" +
			"-------------------------------------";
	}

	// Τυπώνει τα στοιχεία του εργαζομένου
	public void printHourlyEmployee()
	{
		System.out.println(this.toString());
	}

	 // Υπολογισμός μισθού ωρομισθίου
	public PayrollRunEntry calculatePayroll(
			int RayrollRunEntryCounter,
			GregorianCalendar PayrollRunDate)
	{

		double amountToBePaid = 0;
		double hours = 0;

		// Υπολογίζουμε την αμοιβή κάθε ημέρα
		// Εάν η διάρκεια εργασίας <=8 ώρες τότε
```

```
                // το ποσό υπολογίζεται από αριθμό ωρών * ωρομίσθιο
                // Εαν διάρκεια εργασίας > 8 ώρες τότε
                // το ποσό υπολογίζεται ως εξής
                // 8 ώρες * ωρομίσθιο+υπερωρίες*ωρομίσθιο*1,5

                Vector<LogEntry> tempLogEntries;
                amountToBePaid = 0;
                // Προσπελαύνουμε τις εγγραφές του εργαζομένου

                tempLogEntries = EmplTimeCard.getTimecardVector();
                if (tempLogEntries.isEmpty())
                       System.out.println("ΔΕΝ ΥΠΑΡΧΟΥΝ ΕΓΓΡΑΦΕΣ ΓΙΑ
                                    ΕΡΓΑΖΟΜΕΝΟ");

                else
                {
                       // διατρέχουμε όλα τα elements του vector
                       for (LogEntry element : tempLogEntries)
                       {
                              hours = element.CalculateWorkTime();
                              if (hours <= 8)
                                     amountToBePaid = amountToBePaid +
                                         hours * HourlyEmployeeRate;
                              else
amountToBePaid =      amountToBePaid + 8 * HourlyEmployeeRate +
                       (hours - 8) * HourlyEmployeeRate * PEROVERTIME;
                       }  // end for
                } // end else

                // Δημιουργία εγγραφής μισθοδοσίας
                 return new PayrollRunEntry(RayrollRunEntryCounter,
                                        PayrollRunDate,
                                        getEmplNo(),
                                        getEmplName(),
                                        amountToBePaid,
                                        "ΩΡΟΜΙΣΘΙΟΣ",
                                        "ΜΙΣΘΟΣ");
    }
}
```

5.8 Κλάση FullTimeEmployee



Εικόνα 48: Η κλάση FullTimeEmployee

Η υλοποίηση της κλάσης FullTimeEmployee έχει σημαντικές ομοιότητες με την υλοποίηση τις κλάσεις CommissionedEmployee και PartTimeEmployee. Για παράδειγμα,

1. η κλάση συνδέεται με σχέση κληρονομικότητας με την κλάση Employee,

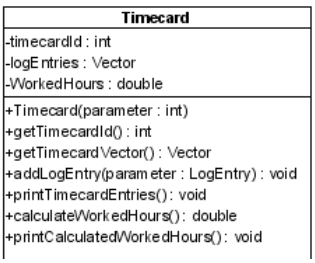
2. γίνεται χρήση μεθόδων της μητρικής κλάσης με τη χρήση της δεσμευμένης λέξης super, και
3. γίνεται υλοποίηση της αφηρημένης μεθόδου calculatePayroll.

5.8.1 Ο κώδικας της κλάσης FullTimeEmployee

```
public class SalariedEmployee extends Employee
{
	static double MonthlyWorkingHours = 160.0;
	double EmployeeMonthlySalary;

	public SalariedEmployee(int EmplNo, String EmplFirstName,
			String EmplLastName, String EmplAddress, String
			EmplCity, String EmplPostalCode, String
			EmplPhoneNumber,
			String EmplEmail, String EmplDepartmentNo,
			double EmployeeMonthlySalary,
			Timecard EmplTimeCard)
	{
	// Καλεί τον κατασκευαστή της μητρικής κλάσης employee
	   super(EmplNo, EmplFirstName, EmplLastName,
	      EmplAddress, EmplCity, EmplPostalCode,
	      EmplPhoneNumber,EmplEmail, EmplDepartmentNo,
	      EmplTimeCard);
	   this.EmployeeMonthlySalary=EmployeeMonthlySalary;
	}

	// Προετοιμάζει τα στοιχεία του εργαζομένου
	// και τα αποθηκεύει σε ένα string
	public String toString()
	{
		return
			"ΕΡΓΑΖΟΜΕΝΟΣ ΜΕ ΜΗΝΙΑΙΟ ΜΙΣΘΟ \n"+
			super.toString()+ "Ο μισθός είναι :"
			+ EmployeeMonthlySalary+"\n"+
			"-------------------------------------";
	}

	// Τυπώνει τα στοιχεία του εργαζομένου
	public void printSalariedEmployee()
	{
		System.out.println(this.toString());
	}

	// Υπολογισμός μισθού εργαζομένων
	public PayrollRunEntry calculatePayroll(
			int RayrollRunEntryCounter,
			GregorianCalendar PayrollRunDate)
	{

		// Αρχικοποίηση προσωρινών μεταβλητών
		double amountToBePaid = 0;
		double hourlyWage = 0;
		double moreHours = 0;
		double lessHours = 0;

		if (EmplTimeCard.calculateWorkedHours() >
					MonthlyWorkingHours)
		// Οι ώρες εργασίας εργαζομένου περισσότερες κανονικού.
```

```
                // Χρειάζεται να υπολογίσουμε υπερωρίες.
                // Κάνουμε την παραδοχή ότι οι υπερωρίες πληρώνονται
                // με 50% επιπλέον
                {
                        hourlyWage = EmployeeMonthlySalary /
                                                MonthlyWorkingHours;
                        moreHours = EmplTimeCard.calculateWorkedHours() -
                                                MonthlyWorkingHours;
                        amountToBePaid = EmployeeMonthlySalary + moreHours
                                                * hourlyWage * PEROVERTIME;
                }
                else if (EmplTimeCard.calculateWorkedHours() ==
                                                MonthlyWorkingHours)
                // Ώρες εργασίας ίδιες με κανονικές.
                // Θα πρέπει να υπολογίσουμε μόνο το μισθό
                {
                        amountToBePaid = EmployeeMonthlySalary;
                }
                else
                // Οι ώρες εργασίας είναι λιγότερες από τις κανονικές.
                // Θα πρέπει να μειωθεί ο μισθός.
                {
                        hourlyWage = EmployeeMonthlySalary /
                                        MonthlyWorkingHours;
                        lessHours = MonthlyWorkingHours –
                                EmplTimeCard.calculateWorkedHours();
                        amountToBePaid = EmployeeMonthlySalary - lessHours
                                * hourlyWage;
                }

                // Δημιουργία εγγραφής μισθοδοσίας
                return new PayrollRunEntry(RayrollRunEntryCounter,
                                                PayrollRunDate,
                                                getEmplNo(),
                                                getEmplName(),
                                                amountToBePaid,
                                                "ΜΙΣΘΩΤΟΣ",
                                                "ΜΙΣΘΟΣ");
        }
}
```

5.9 Κλάση Timecard



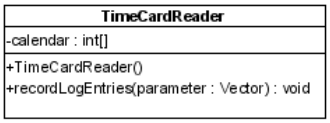
Εικόνα 49: Η κλάση TimeCard

5.9.1 Ο κώδικας της κλάσης TimeCard

```
public class Timecard {

	private int timecardId;
	private Vector<LogEntry> logEntries;
	private double WorkedHours;

	// Ο κατασκευαστής
	public Timecard(int timecardId)
	{
		this.timecardId=timecardId;
		// δημιουργία του Vector
		logEntries = new Vector<LogEntry>();
	}

	// Επιστρέφει των κωδικό της κάρτας
	public int getTimecardId()
	{
		return timecardId;
	}

	// Επιτρέφει τις εγγραφές χρόνου που σχετίζονται με την κάρτα
	public Vector getTimecardVector()
	{
		 return logEntries;
	}

	// Δημιουργούμε μια εγγραφή χρόνου
	// την οποία συνδέουμε με τη συγκεκριμένη κάρτα
	public void addLogEntry(LogEntry curLogEntry)
	{
		logEntries.add(curLogEntry);
	}

	// Εκτύπωση εγγραφών χρόνου εργαζομένου (LogEntry)
	public void printTimecardEntries()
	{
		if ( logEntries.isEmpty() )
			System.out.println( "Η ΚΑΡΤΑ ΔΕΝ ΣΥΝΔΕΕΤΑΙ ΜΕ
					ΕΓΓΡΑΦΕΣ ΕΡΓΑΖΟΜΕΝΟΥ" );
		else  // διατρέχουμε όλα τα elements του vector
		{
			System.out.println( "Εγγραφές χρόνου
					ανά εργαζόμενο" );
			for ( LogEntry element : logEntries )
				System.out.printf( "%s ",
					element.toString() );
		} // end else
		System.out.println( "\n" );
	}

	// Υπολογισμός ωρών εργασίας εργαζομένου
	public double calculateWorkedHours()
	{
		this.WorkedHours=0;

		if ( logEntries.isEmpty() ) // έχει το vector εγγραφές
			System.out.println( "ΔΕΝ ΜΠΟΡΩ ΝΑ ΥΠΟΛΟΓΙΣΩ:
				Η ΚΑΡΤΑ ΔΕΝ ΣΥΝΔΕΕΤΑΙ ΜΕ ΕΓΓΡΑΦΕΣ" );
		else  // διατρέχουμε όλα τα elements του vector
```

```
                {
                        for ( LogEntry element : logEntries )
                WorkedHours=WorkedHours+element.CalculateWorkTime();
                } // end else

                return(WorkedHours);
        }

        // Εκτύπωση ωρών εργασίας
        public void printCalculatedWorkedHours()
        {
                System.out.println( "Συνολικές ώρες εργασίας για
                εργαζόμενο: " + WorkedHours );
        }
}
```

5.10 Κλάση TimeCardReader



Εικόνα 50: Η κλάση TimeCardReader

Η υλοποίηση της κλάση TimeCardReader και ιδιαίτερα της μεθόδου recordLogEntries παρουσιάζει αρκετά ενδιαφέροντα σημεία τα οποία χρειάζονται επεξήγηση και σχολιασμό. Τα σημεία αυτά είναι:

- Η κλάση Random
- Οι εξαιρέσεις (exceptions)
- Εμβέλεια των μεταβλητών (scope)
- Δημιουργία αντικειμένων ως παράμετροι σε κλήσεις μεθόδων

Στη συνέχεια θα σχολιάσουμε τα παραπάνω σημεία και τέλος θα παρουσιάζουμε τον κώδικα της κλάσης TimeCardReader.

5.10.1 Η κλάση Random

Η κλάση Random ορίζεται στο πακέτο java.util και έχει ως σκοπό την παραγωγή τυχαίων αριθμών. Η κλάση χρησιμοποιεί ένα seed (σπόρο-αρχική τιμή) 48 δυαδικών ψηφίων η οποία τροποποιείται χρησιμοποιώντας έναν αλγόριθμο παραγωγής τυχαίων αριθμών. Έτσι, αν δύο περιπτώσεις Random δημιουργούνται από το ίδιο seed και γίνεται η ίδια ακολουθία κλήσεων μεθόδου τότε θα παράγουν και τις ίδιες ακολουθίες αριθμών.

Η αρχικοποίηση της κλάσης Random γίνεται απλά με την παρακάτω εντολή:

```
Random generator = new Random();
```

Ο παραπάνω κατασκευαστής χρησιμοποιεί την ώρα του συστήματος ώστε να παράγει τους τυχαίους αριθμούς. Αυτό σημαίνει, ότι αν δημιουργήσουμε δύο γεννήτριες τυχαίων αριθμών, θα πρέπει η δημιουργία τους να απέχει κατά ένα millisecond η μια από την άλλη ώστε να παράγονται διαφορετικοί τυχαίοι αριθμοί.

Εναλλακτικά, μπορούμε να αρχικοποιήσουμε την κλάση δίνοντας ένα τυχαίο δικό μας seed. Δηλαδή,

```
Random generator = new Random( 19580427 );
```

Μετά την αρχικοποίηση της γεννήτριας τυχαίων αριθμών, μπορούμε να αρχίσουμε τη δημιουργία τυχαίων αριθμών ακέραιων ή πραγματικών ή άλλων τύπων:

```
int r = generator.nextInt();
```

ή

```
double r = generator.nextDouble();
```

Πολλές φορές χρειάζεται να παράγουμε τυχαίους αριθμούς μέσα σε μια συγκεκριμένη περιοχή αριθμών. Έτσι, αν χρειαζόμαστε τυχαίους αριθμούς μεταξύ του 0 και 14, όπως χρειαζόμαστε στο παράδειγμά μας, για να μοντελοποιήσουμε το χρόνο άφιξης έχουμε:

```
int minArrival = (random.nextInt(14) + 1);
```

Αντίστοιχα, για να μοντελοποιήσουμε το χρόνο εξόδου από 0 έως 29 έχουμε:

```
int minDepart = (random.nextInt(29) + 1);
```

Θα πρέπει να διευκρινίσουμε ότι η κλήση random.nextInt(29) επιστρέφει αριθμούς από 0 έως 28.

5.10.2 Οι εξαιρέσεις (exceptions)

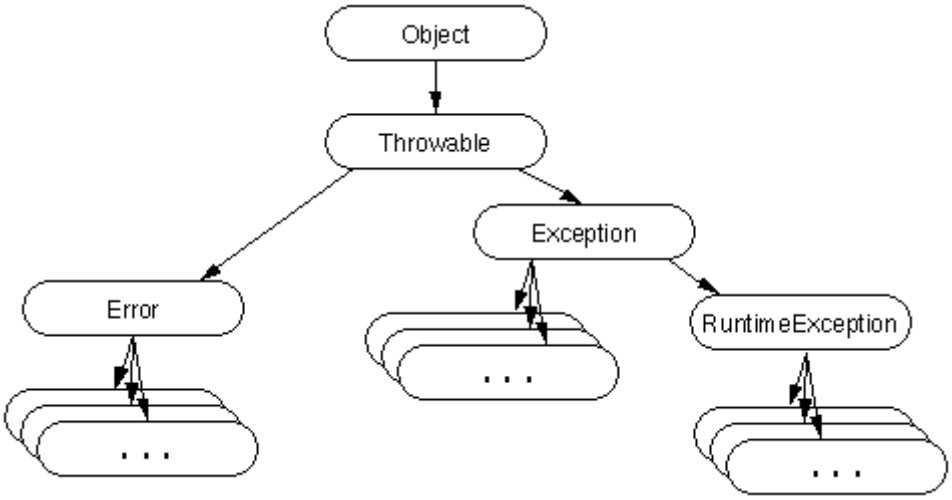
Η java παρέχει ένα μηχανισμό, τις εξαιρέσεις (exceptions), που βοηθά ένα πρόγραμμα java να χειριστεί τα σφάλματα. Εξαίρεση (exception) λέγεται λοιπόν ένα συμβάν, το οποίο κατά το χρόνο εκτέλεσης (π.χ. αδυναμία ανοίγματος αρχείου ή ανάγνωσης πληροφορίας, μη ύπαρξη αντικειμένου στη στοίβα) απαιτεί ειδικό χειρισμό. Στις πιο πολλές φορές, η εξαίρεση είναι ένα λάθος, που διακόπτει τη ροή εκτέλεσης ενός

προγράμματος.

Η κλασική διαχείριση λαθών περιπλέκει τον κώδικα, ώστε να χάνεται η «διαύγειά» του, διότι δεν υπάρχει διάκριση μεταξύ του κώδικα που αφορά την κυρίως κανονική λειτουργία και αυτού που αφορά στο χειρισμό των λαθών-εξαιρέσεων.

Ο ξεχωριστός μηχανισμός χειρισμού εξαιρέσεων επιτρέπει τη συγγραφή καθαρού, εύρωστου και ανεκτικού σε λάθη κώδικα.

Η εξαίρεση είναι αντικείμενο που προέρχεται άμεσα ή έμμεσα από την κλάση Throwable. Η κλάση αυτή περιέχει δύο υποκλάσεις, τις Error και Exception.



Εικόνα 51: Η ιεραρχία των εξαιρέσεων

Εν συντομία η διαχείριση εξαιρέσεων περιλαμβάνει τρία σκέλη:

1. Η δημιουργία εξαιρέσεων. Το πώς ένα κομμάτι κώδικα δημιουργεί μια εξαίρεση.
2. Ο χειρισμός των εξαιρέσεων. Το τι γίνεται και πως συμπεριφέρεται η εφαρμογή μας όταν δημιουργηθεί μια εξαίρεση.
3. Δημιουργία των δικών μας κατηγοριών εξαιρέσεων.

Δημιουργία εξαιρέσεων

Η δημιουργία μιας εξαίρεσης γίνεται με τη χρήση της εντολής throw. Για παράδειγμα η εντολή:

```
if (amount < 0)
```

```
throw new NegativeAmmountException();
```

ή το πιο συνηθισμένο παράδειγμα όταν βγαίνουμε εκτός ορίων ενός πίνακα

```
if (index >= elementCount)
   throw new ArrayIndexOutOfBoundsException
          (index + " >= " + elementCount);
```

Όταν δημιουργηθεί μια εξαίρεση, με την εντολή throw, τότε αυτή θα διαδοθεί – μεταφερθεί στη μέθοδο που έκανε την κλήση. Στη συνέχεια, αυτή η μέθοδος είτε θα χειριστεί η ίδια την εξαίρεση είτε θα διαδώσει με τη σειρά της την εξαίρεση σε παραπάνω επίπεδο χρησιμοποιώντας ξανά τη φράση throws στη δήλωση της.

Χειρισμός εξαιρέσεων

Ο χειρισμός εξαιρέσεων γίνεται με τη χρήση τριών εντολών: try, catch και finally. Η δομή του χειρισμού των εξαιρέσεων είναι η ακόλουθη:

```
try
{
      // οι εντολές που ελέγχονται;
}
catch (ExceptionName e1)
{
      // οι εντολές που εκτελούνται όταν συμβεί η εξαίρεση;
}
catch (ExceptionName e2)  // any number of catch statements
{
      // display exception to screen
      System.out.println("Exception: " + e2);
}
finally
{
      // εκτελείται πάντα
}
```

Στο ακόλουθο παράδειγμα, μέσα στο block εντολών try, έχουμε την εντολή myThread.sleep(10). Η εντολή αυτή είναι δυνατό να προκαλέσει εξαιρέσεις και σφάλματα. Σε περίπτωση λάθους, ο έλεγχος μεταφέρεται και εκτελείται το block εντολών catch που αντιστοιχεί στο είδος της συγκεκριμένης εξαίρεσης. Στη συγκεκριμένη περίπτωση έχουμε ένα είδος εξαίρεσης, την InterruptedException.

```
try
{
      myThread.sleep(10);// καθυστέρηση χρόνου 1”
}
catch (InterruptedException e)
{
      System.out.println("error sleep");
}
```

Δημιουργία των δικών μας κατηγοριών εξαιρέσεων

Για τη δημιουργία δικών μας εξαιρέσεων θα πρέπει να δημιουργήσουμε μια νέα κλάση η οποία θα κληρονομεί από την κλάση Exception ή από την πιο υψηλού επιπέδου κλάση RuntimeException.

Ένα σύντομο παράδειγμα μια τέτοιας κλάσης ακολουθεί:

```
public class ParseError extends Exception {
       // Κατασκευαστής
       public ParseError(String message) {
              // Στο σημείο αυτό μπορούμε να δημιουργήσουμε
              // το κατάλληλο μήνυμα λάθους ή απλά
              // να περάσουμε το μήνυμα στον κατασκευαστή της μητρικής
              // κλάσης Exception
                     super(message);
                 }
              }
```

Στη συνέχεια, στο σημείο του προγράμματος που θέλουμε να παράγουμε την εξαίρεση θα πρέπει να εισάγουμε τον κώδικα

```
throw new ParseError("Encountered an illegal negative number.");
```

ή

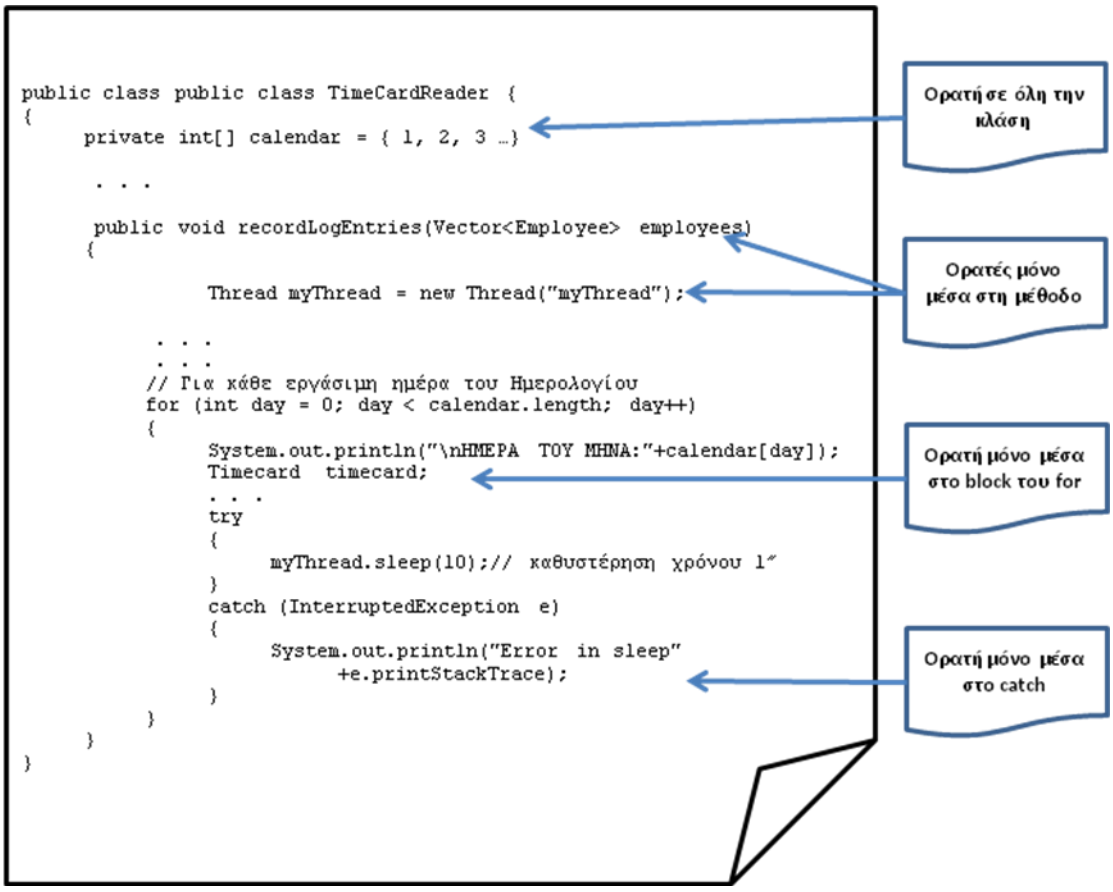
```
throw new ParseError("The word '" + word
              + "' is not a valid file name.");
```

5.10.3 Εμβέλεια των μεταβλητών (scope)

Στην κλάση TimeCardReader έχουμε αρκετές περιπτώσεις δήλωσης μεταβλητών με πεπερασμένη εμβέλεια. Η εμβέλεια της κάθε μεταβλητής είναι ιδιαίτερα σημαντική για τη δομή του προγράμματος.

Έτσι εκτός από τις περιπτώσεις εμβέλειας που ορίσαμε σε επίπεδο εφαρμογής (public, protected, private, undefined) έχουμε και την εμβέλεια των μεταβλητών όπως αυτή ορίζεται μέσα σε μια κλάση.

Στην Εικόνα 52 παρουσιάζεται με γραφικό τρόπο η εμβέλεια της κάθε μεταβλητής μέσα στην κλάση TimeCardReader.



Εικόνα 52: Η εμβέλεια των μεταβλητών μέσα στην κλάση TimeCardReader

5.10.4 Δημιουργία αντικειμένων ως παράμετροι σε κλήσεις μεθόδων

Μια βασική ιδιότητα της γλώσσας java είναι ο συνοπτικός και περιεκτικός τρόπος συγγραφής του κώδικα. Ο τρόπος αυτός μειώνει κατά πολύ το μέγεθος του κώδικα ενώ ταυτόχρονα δεν αυξάνει τη δυσκολία κατανόησης του προγράμματος. Θα δώσουμε ως παράδειγμα, την εντολή της μεθόδου recordLogEntries που έχει ως σκοπό τη δημιουργία μιας νέας εγγραφής.

```
LogEntry curLogEntry = new LogEntry(employees.get(emp).getTimecardId(),
       new GregorianCalendar(2007, 3, calendar[day]),
       new GregorianCalendar(2007, 3, calendar[day], 8, minArrival),
       new GregorianCalendar(2007, 3, calendar[day], 16, minDepart));
```

Αναλύοντας την παραπάνω εντολή παρατηρούμε ότι ο κατασκευαστής της μεθόδου LogEntry απαιτεί τέσσερα ορίσματα:

1. int timecardId,
2. GregorianCalendar LogDate,
3. GregorianCalendar EntryTime, και
4. GregorianCalendar DepartTime

Ως πρώτο όρισμα στην κλάση της μεθόδου περνάμε την παράμετρο

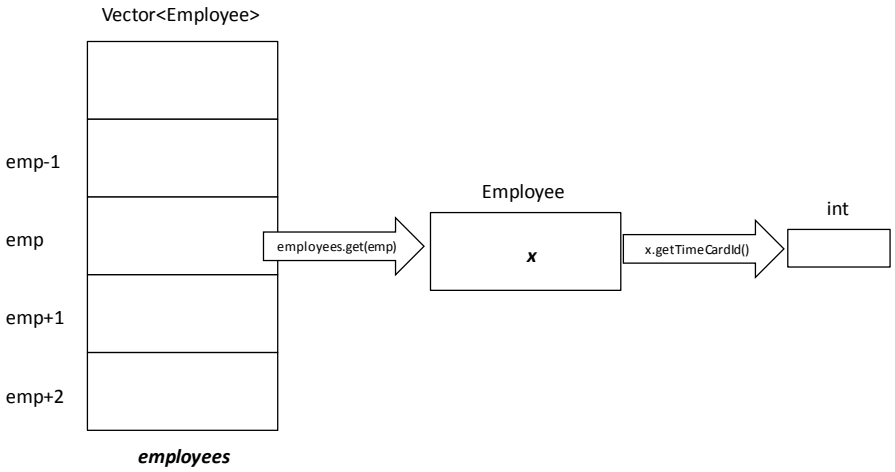
```
employees.get(emp).getTimecardId()
```

Για την κατανόηση αυτής της παραμέτρου θα πρέπει να εργασθούμε βήμα-βήμα. Αρχικά, η μεταβλητή employees είναι τύπου Vector<employee> και έχει περάσει ως παράμετρος της μεθόδου recordLogEntries. Επομένως η κλήση της μεθόδου get(emp) αντιστοιχεί στην κλήση της μεθόδου get(emp) της κλάσης Vector και θα μας επιστρέψει έναν εργαζόμενο. Ο εργαζόμενος που επιστρέφεται είναι αυτός που βρίσκεται στην emp θέση του Vector. Αν κοιτάξουμε τον κώδικα, θα δούμε ότι η μεταβλητή emp είναι ακέραια και είναι αυτή που διατρέχει όλα τα αντικείμενα του Vector. Έστω, ότι εργαζόμενος που επιστρέφεται από την κλήση είναι ο x. Τότε, η αρχική εντολή έχει μετασχηματισθεί σε

```
x.getTimecardId()
```

Η μέθοδος getTimecardId() είναι μέθοδος της κλάσης Employee και επιστρέφει τον αριθμό της κάρτας εργαζομένου, ο οποίος είναι ένας ακέραιος.

Επομένως, στο πρώτο όρισμα της μεθόδου LogEntry έχουμε έναν ακέραιο, όπως θα έπρεπε, ο οποίος αναπαριστά τον αριθμό της κάρτας του τρέχοντα εργαζομένου.



Εικόνα 53: Ανάλυση σύνθετων κλήσεων μεθόδων

Το δεύτερο όρισμα της μεθόδου LogEntry προκύπτει από τη δημιουργία ενός αντικειμένου της κλάσης GregorianCalendar. Για τη δημιουργία αυτού του αντικειμένου περνάμε τρία ορίσματα: χρόνο, μήνα, ημέρα. Επομένως η ημερομηνία που θα δημιουργηθεί θα είναι της μορφής dd/mm/yyyy.

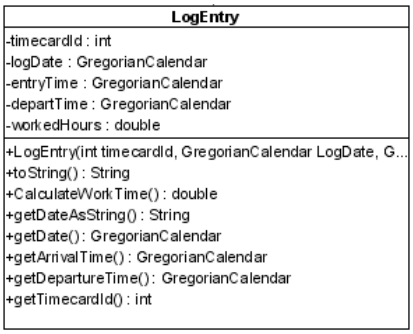
Στο τρίτο και τέταρτο όρισμα δημιουργούμε ξανά ημερομηνίες τύπου GregorianCalendar με τη διαφορά ότι έχουμε επιπλέον και χρόνο εισόδου και εξόδου.

5.10.5 Ο κώδικας της κλάσης TimeCardReader

```
public class TimeCardReader {


        // Κάνουμε την παραδοχή ότι η μελέτη περίπτωσης
        // θα τρέξει μόνο για ένα μήνα (με 30 ημέρες)
        // Έστω τον Απρίλιο του 2007
        private int[] calendar = { 1, 2, 3,
                                   5, 6, 7, 8, 9,
                                   12, 13, 14, 15, 16,
                                   19, 20, 21, 22, 23,
                                   26, 27, 28, 29, 30};

        public TimeCardReader()
        {
        }

        public void recordLogEntries(Vector<Employee> employees)
        {

                // Δημιουργούμε την κλάση Thread ώστε να προσομοιώσουμε
                // το πέρασμα του χρόνου.
                Thread myThread = new Thread("myThread");

                // Αρχικοποιούμε μια τυχαία μεταβλητή
                Random random = new Random();

                GregorianCalendar c = new GregorianCalendar();

                // Αρχικοποιούμε τη σύνδεση με τη βάση δεδομένων
                PersistentStorage.initDBConnectionForTimeLog();

                // Για κάθε εργάσιμη ημέρα του Ημερολογίου
                for (int day = 0; day < calendar.length; day++)
                {

                        System.out.println("\nHMEPA TOY MHNA:" +
                                calendar[day]);
                        Timecard timecard;

                        // Για κάθε εργαζόμενο παίρνει την κάρτα
                        // του για επικύρωση

                        for (int emp = 0; emp < employees.size(); emp++)
                        {
                                timecard =
                                        employees.get(emp).getTimecard();
                                int minArrival = (random.nextInt(14)
                                        + 1);
                                int minDepart = (random.nextInt(29) + 1);
LogEntry curLogEntry = new LogEntry(employees.get(emp).getTimecardId(),
        new GregorianCalendar(2007, 3, calendar[day]),
        new GregorianCalendar(2007, 3, calendar[day], 8, minArrival),
        new GregorianCalendar(2007, 3, calendar[day], 16, minDepart));
                                timecard.addLogEntry(curLogEntry);
```

```
                        System.out.println(
                                employees.get(emp).getEmplName() +
                                "  " + curLogEntry);

                    // Στέλνει το curLogEntry στην ΒΔ για καταχώρηση
                        PersistentStorage.insert(curLogEntry);
                    } //for για κάθε εργαζόμενο
                    try
                    {
                        myThread.sleep(10);// καθυστέρηση χρόνου 1"
                    }
                    catch (InterruptedException e)
                    {
                        System.out.println("Error in sleep"
                                +e.printStackTrace);
                    }
                } // for για κάθε ημέρα του ημερολογίου

            // Κλείνουμε τη σύνδεση μετά την εισαγωγή όλων των εγγραφών
              PersistentStorage.closeDBConnectionForTimeLog();
        }  // method recordLogEntries
    } // TimeCardReader
```

5.11 Κλάση LogEntry



Εικόνα 54: Η κλάση LogEntry

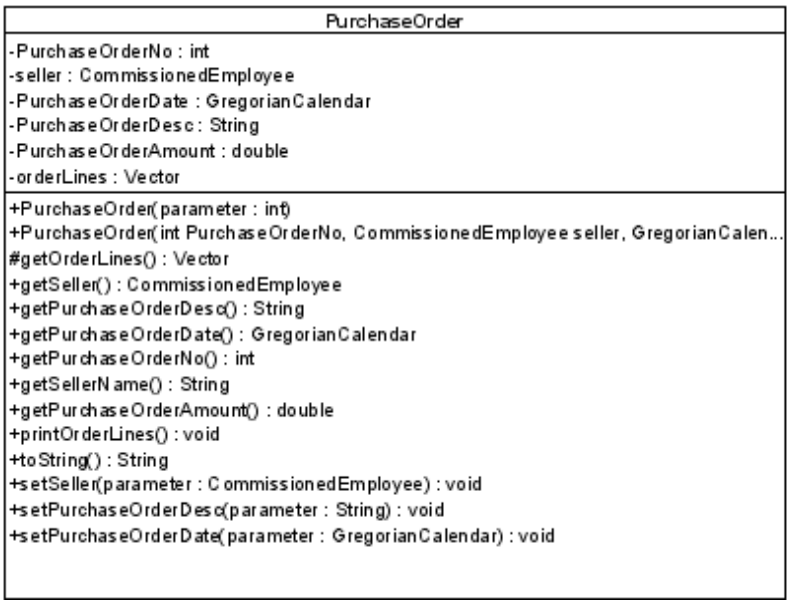
5.11.1 Ο κώδικας της κλάσης LogEntry

```
public class LogEntry {

        private int timecardId;           // Ο κωδικός της κάρτας
        private GregorianCalendar LogDate;// Η ημερομηνία που έγινε η εγγραφή
        private GregorianCalendar EntryTime;    // Η ώρα εισόδου
        private GregorianCalendar DepartTime;   // Η ώρα αποχώρησης
        private double WorkedHours; // Οι ώρες που εργάσθηκε ο εργαζόμενος


        // Ο κατασκευαστής
        public LogEntry(int timecardId, GregorianCalendar LogDate,
                        GregorianCalendar EntryTime,
                        GregorianCalendar DepartTime)
        {
                this.timecardId = timecardId;
```

```
        this.LogDate = LogDate;
        this.EntryTime = EntryTime;
        this.DepartTime = DepartTime;
        CalculateWorkTime();
}

public String toString()
{
        // Προσωρινές μεταβλητές για την  επεξεργασία των ημερομηνιών
        Date tempDate1, tempDate2, tempDate3;
         DateFormat df1;
        String s1, s2, s3;

        // Μετατρέπω το GregorianCalendar object σε Date
        // Η μέθοδος getTime επιστρέφει την ημερομηνία (Date)
        tempDate1 = LogDate.getTime();
        tempDate2 = EntryTime.getTime();
        tempDate3 = DepartTime.getTime();

        // Ορίζω το format εμφάνισης της ημερομηνίας και της ώρας
        // SHORT εμφανίζεται ως 12.13.52 ή 3:30pm
        // MEDIUM εμφανίζεται ως Jan 12, 1952
        // LONG εμφανίζεται ως January 12, 1952 ή 3:30:32pm
        // FULL είναι πλήρες Tuesday, April 12, 1952 AD or 3:30:42pm.

        // Ορίζω το Ελληνικό τρόπο παρουσίασης τον ημερομηνιών
        Locale l = new Locale("el");
        df1 = DateFormat.getDateTimeInstance(DateFormat.SHORT,
                    DateFormat.SHORT, l);

        // Μετατρέπω τις ημερομηνίες σε String
        // Ο χειρισμός των ημερομηνιών στην Java είναι δύσκολος.

        s1 = df1.format(tempDate1);
        s2 = df1.format(tempDate2);
        s3 = df1.format(tempDate3);

        return " ΗΜΕΡΟΜΗΝΙΑ ΚΑΤΑΓΡΑΦΗΣ : " + s1 +
               " ΩΡΑ ΕΙΣΟΔΟΥ :" + s2 +
               " ΩΡΑ ΕΞΟΔΟΥ :" + s3 +
               " ΩΡΕΣ ΕΡΓΑΣΙΑΣ :" + WorkedHours;
}

// Υπολογισμός χρόνου εργασίας εργαζομένου
// Είναι ένας σύνθετος υπολογισμός μια και ο υπολογισμός μπορεί
// να λάβει υπόψη δευτερόλεπτα, λεπτά ή ώρες.
// Η επιλογή μας είναι ο υπολογισμός να γίνεται σε ώρες

public double CalculateWorkTime()
{
        Date tempDate1, tempDate2;
        DateFormat df1;
        Long l1, l2, difference;

        // Μετατρέπω το αντικείμενο GregorianCalendar σε Date
        tempDate1 = EntryTime.getTime();
        tempDate2 = DepartTime.getTime();

        // Η κλάση Date μετράει το χρόνο που πέρασε από την 01/01/1970
        // χρησιμοποιώντας ένα long αριθμό.
        l1 = tempDate1.getTime();
        l2 = tempDate2.getTime();
```

```
		difference = (l2 - l1) / 1000;    // σε δευτερόλεπτα
		WorkedHours = difference / 3600;  // σε ώρες
		return (WorkedHours);
	}


	// Μετατροπή ημερομηνίας σε string
	public String getDateAsString()
	{
		// Προσωρινές μεταβλητές για την  επεξεργασία των ημερομηνιών
		Date tempDate1;
		DateFormat df1;
		String s1;

		// Μετατρέπω το GregorianCalendar object σε Date
		tempDate1 = LogDate.getTime();
		df1 = DateFormat.getDateTimeInstance(DateFormat.SHORT,
			DateFormat.SHORT,new Locale("el"));
		s1 = df1.format(tempDate1);

		System.out.println(" ΗΜΕΡΟΜΗΝΙΑ ΚΑΤΑΓΡΑΦΗΣ : " + s1);
		return s1;
	}

	// Επιστρέφει την ημερομηνία καταγραφής
	public GregorianCalendar getDate()
	{
		return this.LogDate;
	}

	// Επιστρέφει την ώρα προσέλευσης
	public GregorianCalendar getArrivalTime()
	{
		return this.EntryTime;
	}

	// Επιστρέφει την ώρα αναχώρησης
	public GregorianCalendar getDepartureTime()
	{
		return this.DepartTime;
	}

	// Επιστρέφει τον κωδικό της κάρτας
	public int getTimecardId()
	{
		return timecardId;
	}
}
```

5.12 Κλάση PurchaseOrder



Εικόνα 55: Η κλάση PurchaseOrder

5.12.1 Ο κώδικας της κλάσης PurchaseOrder

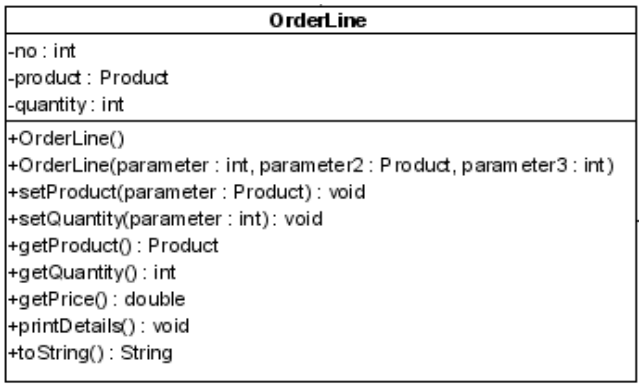
```
public class PurchaseOrder {

	private int PurchaseOrderNo;                  // Αριθμός παραγγελίας
	private CommissionedEmployee seller;          // Πωλητής
	private GregorianCalendar PurchaseOrderDate; // Ημερομηνία παραγγελίας
	private String PurchaseOrderDesc;             // Περιγραφή παραγγελίας
	private double PurchaseOrderAmount;           // Ποσό παραγγελίας
	private Vector<OrderLine> orderLines;

	// Πρώτος Κατασκευαστής
	// Ο πρώτος κατασκευαστής ορίζει μια παράμετρο και δύο μεταβλητές
	// 1. Κωδικό PurchaseOrderNo που περνάει παραμετρικά
	// 2. Ημερομηνία
	// 3. Προϊόντα που περιέχει η παραγγελία
	public PurchaseOrder(int PurchaseOrderNo)
	{
		this.PurchaseOrderNo=PurchaseOrderNo;
		this.PurchaseOrderDate= new
			GregorianCalendar(TimeZone.getTimeZone("GMT+2"));
		orderLines = new Vector<OrderLine>();
	}

	// Δεύτερος Κατασκευαστής
	// Ο πρώτος κατασκευαστής ορίζει τέσσερεις παραμέτρους
	// 1. Κωδικό PurchaseOrderNo που περνάει παραμετρικά
	// 2. Πωλητή seller
	// 3. Ημερομηνία παραγγελίας
	// 4. Περιγραφή παραγγελίας

	public PurchaseOrder(int PurchaseOrderNo,
```

```
                CommissionedEmployee seller, GregorianCalendar
                PurchaseOrderDate, String PurchaseOrderDesc)
{
        this.PurchaseOrderNo=PurchaseOrderNo;    // Ο Κωδικός
        this.seller=seller;                 // Ο πωλητής
        this.PurchaseOrderDate=PurchaseOrderDate; // Η ημερομηνία
        this.PurchaseOrderDesc=PurchaseOrderDesc; // Η περιγραφή

        orderLines = new Vector<OrderLine>(); // Η λίστα με τα προϊόντα
}

// Επιτρέφει τον πίνακα με τις παραγγελίες
public Vector<OrderLine> getOrderLines()
{
        return orderLines;
}


// Επιτρέφει τον πωλητή
public CommissionedEmployee getSeller()
{
        return seller;
}

// Επιτρέφει την περιγραφή της παραγγελίας
public String getPurchaseOrderDesc()
{
        return PurchaseOrderDesc;
}

// Επιτρέφει την ημερομηνία της παραγγελίας
public GregorianCalendar getPurchaseOrderDate()
{
        return PurchaseOrderDate;
}

// Επιστρέφει κωδικό παραγγελίας
public int getPurchaseOrderNo()
{
        return(PurchaseOrderNo);
}

// Επιστρέφει το όνομα του πωλητή
// Η μέθοδος δεν χρησιμοποιείται.
// Στην επόμενη έκδοση του συστήματος θα πρέπει να αφαιρεθεί
public String getSellerName()
{
        return("KENO");
}

// Επιστρέφει ποσό παραγγελίας
public double getPurchaseOrderAmount()
{
// Ο Iterator είναι μια κλάση που μπορεί να χρησιμοποιηθεί
// σε συνδυασμό με οποιαδήποτε συλλογή (Collection) της Java
// Ο Iterator διατρέχει τα στοιχεία μιας συλλογής
// με τη χρήση της μεθόδου hasNext() και next()

        Iterator<OrderLine> i = orderLines.iterator();
        PurchaseOrderAmount=0.0;

        // Η μέθοδος hasNext() ελέγχει εάν υπάρχει επόμενο στοιχείο
```

```
        // στη συλλογή.
        // Αν υπάρχει επιστρέφει true
        // Αν δεν υπάρχει επιστρέφει false
        while(i.hasNext())
        {
                // Η μέθοδος i.next() επιστρέφει το επόμενο στοιχείο

                PurchaseOrderAmount+=i.next().getPrice();
        }
        return(PurchaseOrderAmount);
}

// Εκτυπώνει τις γραμμές παραγγελίας
public void printOrderLines()
{
        System.out.println(this.toString());
        System.out.println("A.A. \t
                Κωδ.\tΠεριγραφή\tΤιμή\tΈκπτωση\tΠοσότητα\tΤιμή");
        Iterator<OrderLine> i = orderLines.iterator();
        PurchaseOrderAmount=0.0;
        String orderLinesOut = new String();
        while(i.hasNext())
        {
                i.next().printDetails();
        }
}

// Προετοιμάζει δεδομένα για εκτύπωση
public String toString()
{
        // Προσωρινές μεταβλητές για την  επεξεργασία των ημερομηνιών
        Date tempDate1;
        DateFormat df1;
        String s1;

        // Μετατρέπω το GregorianCalendar object σε Date
        tempDate1 = PurchaseOrderDate.getTime();

        // Ορίζω το format εμφάνισης της ημερομηνίας και της ώρας
        df1 = DateFormat.getDateTimeInstance(DateFormat.SHORT,
                        DateFormat.SHORT);

        // Μετατρέπω τις ημερομηνίες σε String
        s1 = df1.format(tempDate1);

        return "ΑΡΙΘΜΟΣ ΠΑΡΑΓΓΕΛΙΑΣ: " + PurchaseOrderNo +
                        //"ΠΩΛΗΤΗΣ: " + seller.getEmplName() +
                        " ΗΜΕΡΟΜΗΝΙΑ ΠΑΡΑΓΓΕΛΙΑΣ: " + s1 +
                        " ΠΕΡΙΓΡΑΦΗ: " + PurchaseOrderDesc +
                        " ΠΟΣΟ ΠΑΡΑΓΓΕΛΙΑΣ: " + PurchaseOrderAmount +
                        "\n";
}

// Δίνει τιμή στον πωλητή
public void setSeller(CommissionedEmployee seller)
{
        this.seller = seller;
}

// Δίνουμε τιμή στην περιγραφή
public void setPurchaseOrderDesc(String PurchaseOrderDesc)
{
```

```
		this.PurchaseOrderDesc = PurchaseOrderDesc;
	}

	// Δίνουμε τιμή στην ημερομηνία παραγγελίας
	public void setPurchaseOrderDate(GregorianCalendar PurchaseOrderDate)
	{
		this.PurchaseOrderDate = PurchaseOrderDate;
	}
}
```

5.13 Κλάση OrderLine



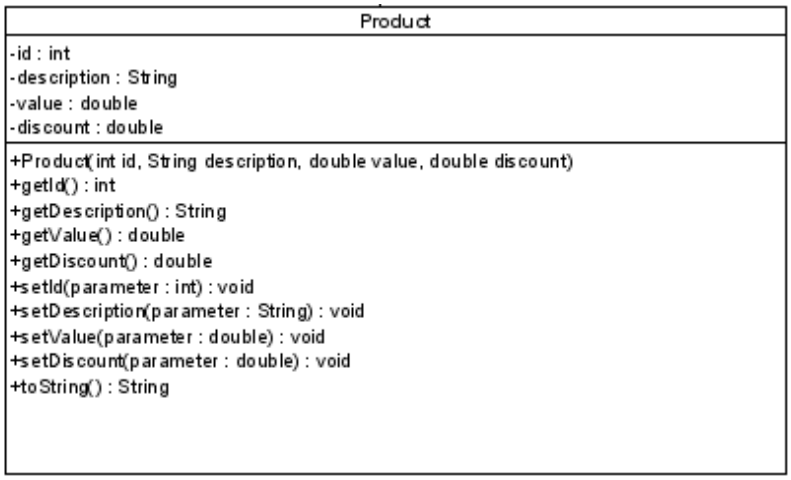
Εικόνα 56: Η κλάση OrderLine

5.13.1 Ο κώδικας της κλάσης OrderLine

```
// Για κάθε PurchaseOrder υπάρχουν πολλές γραμμές παραγγελίας
// Η κάθε γραμμή παραγγελίας σχετίζεται με ένα προϊόν
// Κάθε προϊόν μπορεί να εμφανίζεται σε πολλές γραμμές παραγγελίας
public class OrderLine {

	private int no = 0;		// Η γραμμή της παραγγελίας
	private Product product;	// Το προϊόν της παραγγελίας
	private int quantity=0;		// Η ποσότητα του προϊόντος


	// Ο κατασκευαστής
	public OrderLine(int no, Product product, int quantity)
	{
		this.no = no;
		this.product = product;
		this.quantity = quantity;
	}

	public void setProduct(Product product)
	{
		this.product = product;
	}

	public void setQuantity(int volume)
	{
		this.quantity = volume;
	}
```

```
        public Product getProduct()
        {
               return product;
        }

        public int getQuantity()
        {
               return quantity;
        }

        public double getPrice()
        {
               return product.getValue()*quantity*(1-product.getDiscount());
        }

        // Προετοιμάζει δεδομένα για εκτύπωση
        public void printDetails()
        {
               System.out.println(no + "\t" + product + "\t\t" +
                      quantity + "\t\t" + getPrice());
        }

        // Προετοιμάζει δεδομένα για εκτύπωση
        public String toString()
        {
               return (no + "   " + product + "   " +
                      quantity + "   " + getPrice());
        }
}
```

5.14 Κλάση Product



Εικόνα 57: Η κλάση Product

5.14.1 Ο κώδικας της κλάσης Product

```
public class Product
{

        private int id;                    // Ο κωδικός προϊόντος
        private String description;        // Η περιγραφή
```

```
private double value;             // Η τιμή
private double discount;          // Η έκπτωση


// Ο κατασκευαστής
public Product(int id, String description,
                    double value, double discount)
{
       this.id = id;
       this.description = description;
       this.value = value;
       this.discount = discount;
}

// Επιστρέφει τον κωδικό προϊόντος
public int getId()
{
       return id;
}

// Επιστρέφει την περιγραφή του προϊόντος
public String getDescription()
{
       return description;
}

// Επιστρέφει την τιμή του προϊόντος
public double getValue()
{
       return value;
}

// Επιστρέφει την έκπτωση που υπάρχει στο προϊόν
public double getDiscount()
{
       return discount;
}

// Θέτουμε τον κωδικό του προϊόντος
public void setId(int id)
{
       this.id = id;
}

// Θέτουμε την περιγραφή του προϊόντος
public void setDescription(String description)
{
       this.description = description;
}

// Θέτουμε την τιμή του προϊόντος
public void setValue(double value)
{
       this.value = value;
}

// Θέτουμε την έκπτωση του προϊόντος
public void setDiscount(double value)
{
       this.discount = discount;
}
```

```
        // Προετοιμάζει δεδομένα για εκτύπωση
        public String toString()
        {
                return (" " + id + "\t" + description +
                        "\t" + value + "\t" + discount);
        }
}
```

5.15 Κλάση PersistentStorage

Εικόνα 58: Η κλάση PersistentStorage

5.15.1 Χρήση της βιβλιοθήκης JDBC

Όπως ήδη αναφέραμε, το JDBC API είναι η βιβλιοθήκη της Java που μας επιτρέπει να προσπελάσουμε με εύκολο τρόπο δεδομένα που είναι αποθηκευμένα σε σχεσιακές βάσεις δεδομένων. Η χρήση του JDBC είναι εξαιρετικά απλή. Το παρακάτω απόσπασμα κώδικα μας δίνει το απλούστερο δυνατό παράδειγμα χρήσης JDBC.

// Σύνδεση με τη ΒΣ

```
Connection con = DriverManager.getConnection
             ("jdbc:odbc:Payroll", "myLogin","myPassword");
```

// Δημιουργία της ερώτησης προς τη ΒΔ

```
Statement stmt = con.createStatement();
       ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");

// Ανάκτηση των δεδομένων
while (rs.next())
       {
```

```
            int x = rs.getInt("a");
            String s = rs.getString("b");
            float f = rs.getFloat("c");
        }
```

Πιο αναλυτικά, τα βήματα που απαιτούνται για την ανάπτυξη μια εφαρμογής java που περιλαμβάνει σύνδεση με τη βάση δεδομένων είναι έξι. Αυτά είναι:

1. Κάνουμε import τα απαραίτητα packages που περιέχουν τις απαιτούμενες jdbc κλάσεις. Στις περισσότερες περιπτώσεις επαρκεί το παρακάτω πακέτο:

   ```
   import java.sql.*
   ```

2. Αρχικοποίηση του JDBC driver ώστε να δημιουργήσουμε το κανάλι επικοινωνίας με τη βάση δεδομένων. Για την αρχικοποίηση του JDBC driver χρειάζεται αφενός να βρούμε το όνομα και το μονοπάτι που οδηγεί στο driver και στη συνέχεια να φορτώσουμε την κλάση αυτή. Αυτό μπορεί να γίνει με την παρακάτω εντολή

   ```
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver")
   ```

   η οποία φορτώνει την κλάση και δημιουργεί ένα στιγμιότυπο του driver. Ο συγκεκριμένος driver είναι τύπου bridge μια και μεταφράζει τις jdbc κλήσεις σε odbc κλήσεις και στη συνέχεια στέλνονται στη βάση δεδομένων.

3. Ανοίγουμε τη σύνδεση με τη βάση δεδομένων με τη χρήση της μεθόδου DriverManager.getConnection(), με σκοπό τη δημιουργία ενός αντικειμένου της κλάσης Connection, η οποία ανοίγει μια φυσική σύνδεση με τη βάση δεδομένων. Για τη δημιουργία του αντικειμένου Connection χρειάζεται να προσδιορίσουμε απαραίτητα το URL της βάσης δεδομένων και προαιρετικά το login και password.

   ```
   // Δημιουργούμε το URL με το όνομα της βάσης δεδομένων
   // Για το συγκεκριμένο παράδειγμα το όνομα της βάσης δεδομένων
   // είναι Payroll και δεν υπάρχει login και password
   String url = "jdbc:odbc:Payroll";

   // Δημιουργούμε τη σύνδεση με τη βάση δεδομένων
   Connection con = DriverManager.getConnection(url);
   ```

4. Δημιουργούμε την εντολή SQL που θέλουμε να εκτελέσουμε και την εκτελούμε

   ```
   // Δημιουργούμε την sql statement με τη βάση δεδομένων
   Statement stmt = con.createStatement();

   // Εκτελούμε το query
   ResultSet rs = stmt.executeQuery("SELECT * FROM Employee");
   ```

Στη συγκεκριμένη περίπτωση εκτελούμε μια πολύ απλή ερώτηση επιλέγοντας όλα τα πεδία, όλων των εγγραφών του πίνακα Employee.

5. Το αποτέλεσμα της εκτέλεσης της ερώτησης είναι όλες οι εγγραφές του πίνακα Employee, δηλαδή το σύνολο των εγγραφών. Το σύνολο αυτό μοντελοποιείται με την κλάση ResultSet.

   Στη συγκεκριμένη εφαρμογή το rs είναι αντικείμενο της κλάσης ResultSet. Η μέθοδος rs.next() μας δίνει κάθε φορά την επόμενη εγγραφή του πίνακα Employee. Στη συνέχεια, για κάθε πεδίο του πίνακα, μεταφέρουμε τα περιεχόμενά του σε μια τοπική μεταβλητή του ιδίου τύπου.

```
while (rs.next())
{
        int TimecardId =    rs.getInt("TimecardID");
        String JobType =    rs.getString("JobType");
        int EmpNo =   rs.getInt("EmpNo");
        …
```

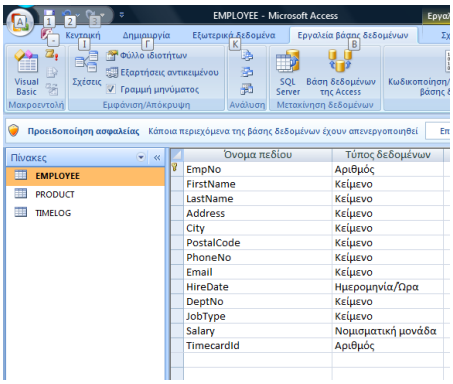
6. Κλείνουμε το statement και το connection ώστε να απελευθερώσουμε τους μη χρησιμοποιούμενους πόρους.

```
stmt.close();
con.close();
```

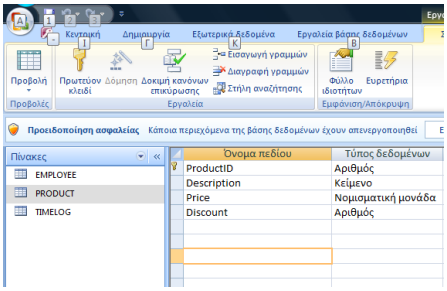
5.15.2 Η βάση δεδομένων Payroll

Η βάση δεδομένων Payroll είναι φτιαγμένη σε MS-Access και αποτελείται από τρεις πίνακες. Οι πίνακες αυτοί είναι:

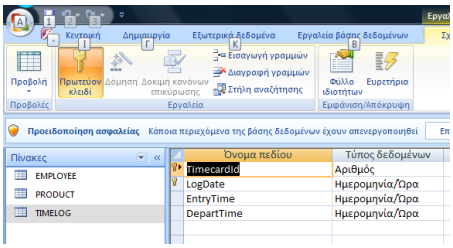
- Employee
- Product
- Timelog



Εικόνα 59: Ο πίνακας Employee



Εικόνα 60: Ο πίνακας Product



Εικόνα 61: Ο πίνακας Product

5.15.3 Ο κώδικας της κλάσης PersistentStorage

```
public class PersistentStorage {

    private static Vector<Employee> employees;
    private static Vector<Product> products;
    private static Vector<PurchaseOrder> orders;
    private static Connection con = null;
    private static PreparedStatement pstmt;


    // Αρχικοποίηση των Vectors
    public static void initPersistentStorage()
    {
```

```
        // Αρχικοποίηση δομών δεδομένων που θα αποθηκευτούν
        // τα δεδομένα
        employees = new Vector<Employee>(10);
        products = new Vector<Product>(10);
        orders = new Vector<PurchaseOrder>(10);

        // Αρχικοποίηση της σύνδεσης με τη βάση δεδομένων
        initDBConnectionForTimeLog();

        // Φόρτωση των δεδομένων εργαζομένων
        // στο Vector employees
        loadEmployees();

        // Φόρτωση των δεδομένων των προϊόντων
        // στο Vector products
        loadProducts();
    }


    public static void initDBConnectionForTimeLog()
    {
        // Εάν υπάρχει ήδη σύνδεση με τη βάση δεδομένων
        // κλείνουμε τη σύνδεση
        if (con != null)
             closeDBConnectionForTimeLog();
        try {
            // Βρίσκουμε το όνομα του driver της βάσης δεδομένων
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            // Δίνουμε το όνομα της βάσης δεδομένων
            String url = "jdbc:odbc:Payroll";

            // Δημιουργούμε τη σύνδεση με τη βάση δεδομένων
            con = DriverManager.getConnection(url);

            // Ορίζουμε το query
            String sql = "INSERT INTO TIMELOG" +
                         "(TimecardId,LogDate,EntryTime,DepartTime)" +
                         "VALUES(?,?,?,?)";

            // Κάνουμε το prepare statement
            pstmt = con.prepareStatement(sql);
        }
        catch (Exception ex) {
           System.err.println("Exception: " + ex.getMessage());
       }
    }

    public static void closeDBConnectionForTimeLog()
    {
        // Εάν υπάρχει σύνδεση
        // 1. Κλείνουμε το prepare statement
        // 2. Κλείνουμε τη σύνδεση με τη βάση δεδομένων

        if (con != null)
            try
            {
                   pstmt.close();
                   con.close();
            }
            catch (SQLException ex)
            {
```

```
                System.err.println("SQLException: " +
                        ex.getMessage());
            }
    }

    // Φόρτωμα δεδομένων από τη βάση
    public static Vector<Employee> loadEmployees()
    {
        try
        {

        // 1. Βρίσκουμε και φορτώνουμε το driver της βάσης δεδομένων
              Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

        // 2. Δημιουργούμε το URL με το όνομα της βάσης δεδομένων
              String url = "jdbc:odbc:Payroll";

        // 3.  Δημιουργούμε τη σύνδεση με τη βάση δεδομένων
              Connection con = DriverManager.getConnection(url);

        // 4. Δημιουργούμε την sql statement με τη βάση δεδομένων
              Statement stmt = con.createStatement();

        //    Εκτελούμε το query
           ResultSet rs = stmt.executeQuery(
                    "SELECT * FROM Employee");

        // 5. Διατρέχουμε όλες τις εγγραφές

            while (rs.next())
            {
                int TimecardId =  rs.getInt("TimecardID");
                String JobType =  rs.getString("JobType");
                int EmpNo =       rs.getInt("EmpNo");
                String FirstName= rs.getString("FirstName");
                String LastName = rs.getString("LastName");
                String Address =  rs.getString("Address");
                String City =     rs.getString("City");
                String PostalCode =
                    String.valueOf(rs.getString("PostalCode"));
                String PhoneNo =  rs.getString("PhoneNo");
                String Email =    rs.getString("Email");
                String DeptNo =   rs.getString("DeptNo");
                double Salary =
                    Double.valueOf(rs.getString("Salary"));

                System.out.println(EmpNo + "  " + FirstName +
                    "  " + LastName +
                    "  " + Address + "  " + City);

                // Καλούμε τους κατασκευαστές ώστε
                //να αρχικοποιήσουμε τα αντικείμενα

                // ΕΡΓΑΖΟΜΕΝΟΙ ΜΕ ΜΙΣΘΟ
                if (JobType.equals("FULL TIME"))
                    employees.add(new SalariedEmployee(EmpNo,
                            FirstName,
                            LastName, Address, City,
                            PostalCode, PhoneNo, Email, DeptNo, Salary,
                            new Timecard(TimecardId)));

                // ΕΡΓΑΖΟΜΕΝΟΙ ΜΕΡΙΚΗΣ ΑΠΑΣΧΟΛΗΣΗΣ
```

```
                else if (JobType.equals("PART TIME"))
                    employees.add(new HourlyEmployee(EmpNo, FirstName,
                            LastName,
                            Address, City,
                            PostalCode, PhoneNo,
                            Email, DeptNo, Salary,
                            new Timecard(TimecardId)));

             // ΠΩΛΗΤΕΣ
                else
                    employees.add(new CommissionedEmployee(EmpNo,
                            FirstName,
                            LastName, Address, City,
                            PostalCode, PhoneNo, Email,
                            DeptNo, Salary,
                            new Timecard(TimecardId)));
            }
            // 6. Κλείνουμε το statement κα την connection

            stmt.close();
            con.close();
        }
        catch (java.lang.Exception ex)
        {
                    ex.printStackTrace();
        }
        return employees;
}

// Επιστρέφει τον πίνακα των Εργαζομένων
public static Vector<Employee> getEmployees()
{
    return employees;
}

// Βρίσκουμε τον εργαζόμενο από το timeCardId
private static Employee getEmployeeFromTimeCard(int timeCardId)
{
   for(Employee em : employees)
       if(em.getTimecardId()==timeCardId)
           return em;
   return null;
}

// Επιστρέφει τον πίνακα των Προϊόντων
public static Vector<Product> getProducts()
{
    return products;
}


public static Vector<Product> loadProducts()
{
   try
   {
    // 1. Βρίσκουμε και φορτώνουμε το driver της βάσης δεδομένων
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    // 2. Δημιουργούμε το URL με το όνομα της βάσης δεδομένων
        String url = "jdbc:odbc:Payroll";

    // 3. Δημιουργούμε τη σύνδεση με τη βάση δεδομένων
```

```
            Connection con = DriverManager.getConnection(url);

        // 4. Δημιουργούμε την sql statement με τη βάση δεδομένων
            Statement stmt = con.createStatement();

        //  Εκτελούμε το query
            ResultSet rs = stmt.executeQuery("SELECT * FROM Product");

        // 5. Διατρέχουμε όλες τις εγγραφές
            while (rs.next())
             {
                int productID = rs.getInt("ProductID");
                String description = rs.getString("Description");
                double price = Double.valueOf(rs.getString("Price"));
                double discount =
                          Double.valueOf(rs.getString("Discount"));

                System.out.println(productID + "  " +
                          description + "  " +
                          price + "  " + discount);
                products.add(new Product(productID,
                          description, price, discount));
             }

             // 6. Κλείνουμε το statement κα την connection
             stmt.close();
             con.close();
         }
       catch (java.lang.Exception ex)
       {
             ex.printStackTrace();
        }
        return products;
}


// Εισαγωγή εγγραφών στη βάση δεδομένων
public static void insert(LogEntry logEntry)
{
    try
    {
          // Κάνουμε prepare το statement
          // για την εισαγωγή εγγραφής
          pstmt.setInt(1, logEntry.getTimecardId());
          pstmt.setDate(2, new
   java.sql.Date(logEntry.getHmeromhnia().getTimeInMillis()));
          pstmt.setTime(3, new Time(logEntry.getProseleysisTime().
                                    getTimeInMillis()));
          pstmt.setTime(4, new Time(logEntry.getApoxwrisisTime().
                                    getTimeInMillis()));
         // Εισαγωγή της εγγραφής
          pstmt.executeUpdate();
    }
    catch (SQLException ex)
    {
         System.err.println("SQLException: " + ex.getMessage());
    }
}

// Διαγραφή εγγραφών χρόνου στη βάση δεδομένων
public static void clearLogEntriesFromDB()
{
```

```
        try
        {
            // Load Sun's jdbc-odbc driver
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            String url = "jdbc:odbc:Payroll";
            Connection con = DriverManager.getConnection(url);
            Statement stmt = con.createStatement();
            String sql = "DELETE FROM TIMELOG";
            // Διαγραφή των εγγραφών του πίνακα TIMELOG
            // πριν από κάθε τρέξιμο
            stmt.executeUpdate(sql);
            stmt.close();
            con.close();
        }
        catch (Exception ex)
        {
            System.err.println("Exception on clear TIMELOG: " +
                    ex.getMessage());
         }
    }


    // ΕΡΓΑΣΙΕΣ ΜΕΝΟΥ
    //   Εκτύπωση στοιχείων εργαζομένων
    public static void printEmployees()
    {
        for (int i = 0; i < employees.size(); i++)
            System.out.println(employees.get(i));
    }

    //   Εκτύπωση Ωρών Εργασίας
    public static void printLogEntries()
    {
        try
        {
            SimpleDateFormat df1 = new
                    SimpleDateFormat("dd/MM/yyyy",new Locale("el"));
            SimpleDateFormat tf1 = new SimpleDateFormat("h:mm a",
                    new Locale("el"));

            // Load Sun's jdbc-odbc driver
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            String url = "jdbc:odbc:Payroll";
            Connection con = DriverManager.getConnection(url);
            // create and execute a SELECT
            Statement stmt = con.createStatement();

            // LogEntries loaded
            ResultSet rs = stmt.executeQuery("SELECT * FROM TIMELOG
                    ORDER BY TimecardId,LogDate,EntryTime");
            System.out.println("Εκτύπωση Χρόνων Προσέλευσης και
                    Αποχώρησης Εργαζομένων:");
            int timecardId=-1; Employee em=null;
            while (rs.next())
            {
                int tmp = rs.getInt("TimecardID");
                if(tmp!=timecardId)
                {
                    timecardId = tmp;
                    em=getEmployeeFromTimeCard(timecardId);
                    if(em!=null)
                        System.out.println("\nΕργαζόμενος:"+
```

```
                        em.getEmplName());
                }
                Date logdate = rs.getDate("LogDate");
                Time entrytime = rs.getTime("EntryTime");
                Time departtime = rs.getTime("DepartTime");
                System.out.println("\t" + df1.format(logdate) +
                        " Είσοδος:" + tf1.format(entrytime) +
                        " Αποχώρηση:"+
                        tf1.format(departtime));
            }
            stmt.close();
            con.close();
        }
        catch (java.lang.Exception ex) {
            ex.printStackTrace();
        }
    }

    //   Εκτύπωση παραγγελιών
    public static void printOrders()
    {
        System.out.println("\n5. ΠΩΛΗΣΕΙΣ ΠΩΛΗΤΩΝ");
        System.out.println("--------------------");
        for (int i = 0; i < employees.size(); i++)
        {
            String s = getEmployees().get(i).getClass().getName();
            if (s.equals("payrollp.CommissionedEmployee"))
            {
                CommissionedEmployee ce =
                    (CommissionedEmployee) employees.get(i);
                System.out.println(ce.getEmplName() +
                    "  Συνολικό ποσό παραγγελιών: " +
                    ce.calculatePurchaseOrdersTotalAmount());
            }
        }
    }

    //   Αναλυτική Εκτύπωση παραγγελιών
    public static void printOrderDetails()
    {
        System.out.println("\n6. ΑΝΑΛΥΤΙΚΕΣ ΠΩΛΗΣΕΙΣ ΠΩΛΗΤΩΝ\n");
        System.out.println("----------------------------\n");
        for (int i = 0; i < employees.size(); i++)
        {
            String s = getEmployees().get(i).getClass().getName();
            if (s.equals("payrollp.CommissionedEmployee"))
            {
                CommissionedEmployee ce =
                    (CommissionedEmployee) employees.get(i);
                System.out.println(ce.getEmplName() +
                    "  Συνολικό ποσό παραγγελιών: " +
                ce.calculatePurchaseOrdersTotalAmount());
                ce.printPurchaseOrders();
            }
        }
    }


}
```

6. Αναφορές

Alhir, S.S. (2003). Learning UML. O'Reilly.

Beck, K. (2004). Extreme Programming Explained: Embrace Change. Addison Wesley, 2nd edition.

Beedle, M. και Schwaber, K. (2002). Agile Software Development with SCRUM. Prentice Hall.

Carmichael, A. και Haywood, D. (2002). Better Software Faster, Prentice-Hall NJ.

Charvat, J. (2003) Project Management Methodologies—Selecting, Implementing, and Supporting Methodologies and Processes for Projects, New Jersey: John Wiley & Sons.

Cockburn, A. (2000). Writing Effective Use Cases, Addison-Wesley Professional.

Cockburn, A. (2006). Agile Software Development: The Cooperative Game. Addison-Wesley, 2nd edition.

Constantine, L. (2001). Cutting Corners: Shortcuts in Model-Driven Web Design. The Management Forum, Software Development, February 2000. Corrected reprint in L. Constantine, ed., Beyond Chaos: The Expert Edge in Managing Software Development. Addison-Wesley, 2001.

Karner, G. (1993). Metrics for Objectory. Diploma thesis, University of Linko, Sweden.

Highsmith, J. (2002). Agile Software Development Ecosystems, Addison Wesley.

Smith, J. (1999). The estimation of effort based on use cases. Rational Software.

Knudsen, J. και Niemeyer, P. (2002). Learning Java™, 2nd Edition. O'Reilly.

Laird, L. και Brennan, C. (2006). Software Measurement and Estimation A Practical Approach. Wiley-InterScience.

Larman, G. (2001). Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design (2nd Edition), C. Larman, Prentice Hall.

Rosenberg, D., Stephens, M. και Scott K. (2005). Use Case Driven Object Modeling: Theory and Practice. New York: Addison-Wesley.

Rosenberg, D. και Scott, K. (1999). Use Case Driven Object Modeling with UML: A Practical Approach. Addison Wesley.

Rosenberg, D. και Scott, K. (2001). Applying Use Case Driven Object Modeling with UML. Addison Wesley.

Rosenberg, D., Stephens, M. και Collins-Cope, M. Agile Development with ICONIX Process—People, Process, and Pragmatism. Addison Wesley.

Schwaber, K. (2004). Agile Project Management with Scrum. Microsoft Press.

Takeuchi, H. και Nonaka, I. (1986). The New New Product Development Game. Harvard Business Review.

Todd, T., (2002), Java Data Access—JDBC, JNDI, and JAXP, M&T Books. Hungry Minds, Inc.

Γερογιάννης, Β., Κακαρόντζας, Γ., Καμέας, Α., Σταμέλος, Γ., και Φιτσιλής, Π. (2006). Αντικειμενοστρεφής Ανάπτυξη Λογισμικού με τη UML. Εκδόσεις Κλειδάριθμος.

Κακαρόντζας, Γ., (2003). Αντικειμενοστρεφής Προγραμματισμός ΙΙ (Java) — Τμήμα Τεχνολογίας Πληροφορικής ΤΕΙ Λάρισας. http://www.cs.teilar.gr/gkakaron/index.html. Προσπελάσθηκε 15/9/2005.

Σταμέλος, Ι. και Σφέτσος, Π. (2003) Agile methods and extreme programming (XP) Ευέλικτες μέθοδοι και Ακραίος Προγραμματισμός. Θεσσαλονίκη.

Σφέτσος, Π., και Σταμέλος, Ι., (2003). Προγραμματισμός με Java, Αριστοτέλειο Πανεπιστήμιο. http://sweng.csd.auth.gr/eclass/GRAD-GAL100/document/SHMEIWSEIS_JAVA.pdf. Προσπελάσθηκε 15/9/2005.

7. Παράρτημα Α – Ευέλικτες μέθοδοι

Ευελιξία (agility) στην ανάπτυξη συστημάτων λογισμικού είναι η ικανότητα της προσαρμογής και του επαναπροσδιορισμού ενός αναπτυσσόμενου και συνεχώς εξελισσόμενου συστήματος στην περίπτωση που εμφανίζονται αλλαγές στις αρχικές απαιτήσεις και παραδοχές. Οι οργανισμοί που χρησιμοποιούν ευέλικτες μεθόδους (agile methods) βλέπουν την αλλαγή σαν ευκαιρία βελτίωσης και προόδου και όχι σαν απειλή.

Το Agile Software Development Manifesto εκδόθηκε από μία ομάδα προγραμματιστών και συμβούλων επιχειρήσεων το 2001 (http://agilemanifesto.org/) (Cockburn, 2006; Anderson, 2003; Highsmith, 2002) και εστιάζεται στην αναγνώριση του ανθρώπινου παράγοντα ως τον πρωταρχικό παράγοντα επιτυχίας ενός έργου λογισμικού, τη συνεχή έμφαση στην αποτελεσματικότητα, την προσαρμοστικότητα και τη διαχείριση της αλλαγής (http://c2.com/cgi/wiki?ExtremeProgramming).

Οι ευέλικτες μέθοδοι είναι:

- *Επαναληπτικές* (iterative): Αρχικά παραδίδεται μια περιορισμένη έκδοση του συστήματος σύστημα και στη συνέχεια γίνονται επεκτάσεις σε κάθε νέα έκδοση.
- *Επαυξητικές* (incremental): Το σύστημα διαιρείται σε υποσυστήματα με βάση τη λειτουργία τους. Νέες λειτουργίες προστίθενται σε κάθε νέα έκδοση.
- *Αυτο-διοργανούμενες* (self-organising): Η ομάδα έχει την αυτονομία να οργανωθεί κατά βούληση.
- *Προκύπτουσες* (emergent): Οι απαιτήσεις και η τεχνολογία που θα χρησιμοποιηθεί προκύπτουν κατά τη διάρκεια του κύκλου ανάπτυξης.

Επιπλέον, όλες οι ευέλικτες μέθοδοι βασίζονται σε 12 αρχές που περιγράφονται στη διακηρυκτική συνεδρίαση για τις ευέλικτες μεθόδους και είναι οι εξής:

- Ικανοποίηση του πελάτη.
- Συχνή παράδοση λογισμικού.
- Η αλλαγή είναι ευπρόσδεκτη.
- Καθημερινή συνεργασία με τον πελάτη.

- Ικανό προσωπικό και περιβάλλον εμπιστοσύνης στην ομάδα.
- Διαπροσωπική συζήτηση για την ανταλλαγή πληροφοριών.
- Σωστή λειτουργία του λογισμικού που κατασκευάζεται.
- Εξασφάλιση σταθερού ρυθμού ανάπτυξης.
- Τεχνική αρτιότητα και καλός σχεδιασμός.
- Υλοποίηση στόχων με σύντομο και αποτελεσματικό τρόπο.
- Αυτό-διοργανούμενες ομάδες.
- Επαναπροσδιορισμός της συμπεριφοράς της ομάδας.

Συνοπτικά τα σημεία στα οποία οι ευέλικτες μέθοδοι δίνουν αξία είναι τα εξής:

- Άτομα και αλληλεπιδράσεις αντί διαδικασίες και εργαλεία.
- Δυναμικός κώδικας αντί γραπτής τεκμηρίωσης.
- Συνεργασία με τον πελάτη αντί αυστηρών συμβολαίων.
- Ανταπόκριση σε αλλαγές αντί ακολουθούμενου σχεδίου.

Οι ευέλικτες μέθοδοι που έχουν προταθεί στη βιβλιογραφία είναι αρκετές. Οι πιο σημαντικές είναι οι παρακάτω (Σταμέλος και Σφέτσος, 2003):

- Ακραίος προγραμματισμός (XP)(Beck, 2004)
- Scrum (Schwaber, 2004)
- Iconix (Rosenberg κ.α, 2005)
- Crystal methods
- Adaptive Software Development (ASD)
- Feature Driven Development (FDD) (Carmichael και Haywood, 2002)

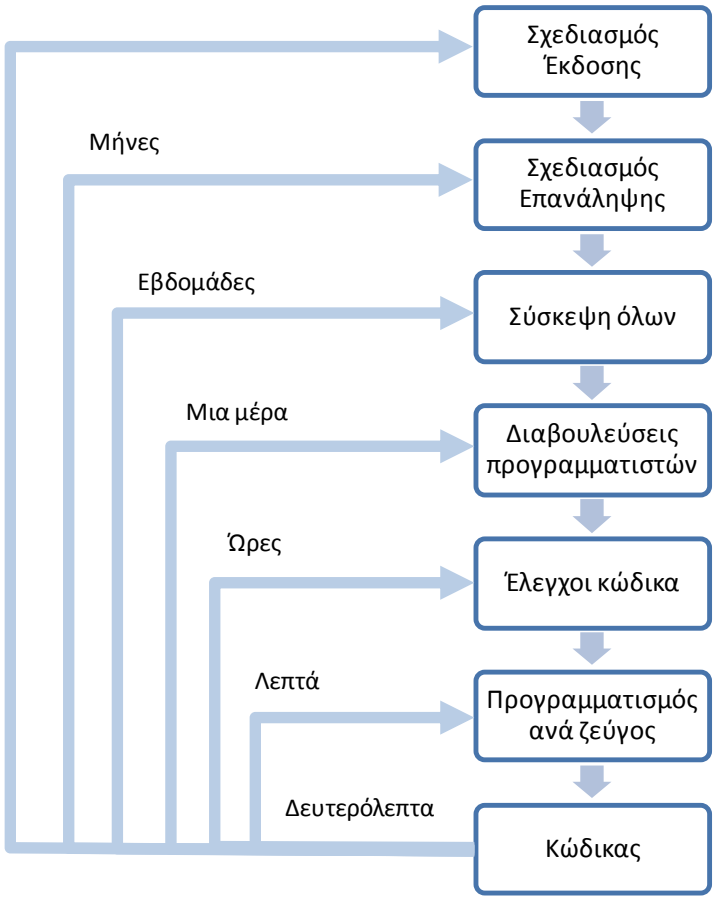
Στη συνέχεια μελετώνται κάποιες από τις πιο διαδεδομένες ευέλικτες μεθόδους.

Ακραίος προγραμματισμός (XP)

Η μεθοδολογία Extreme Programming (XP) είναι μια από τις πλέον διαδεδομένες μεθοδολογίες ανάπτυξης λογισμικού (Beck, 2004). Τα βασικά χαρακτηριστικά της XP είναι:

- Περιορισμός του αριθμού των συμμετεχόντων σε κάθε ομάδα ανάπτυξης
- Μικρούς και διακριτούς κύκλους ανάπτυξης με σαφώς καθορισμένα και απόλυτα λειτουργικά παραγόμενα
- Διαρκής έλεγχος του λογισμικού

- Μεγαλύτερη δυνατή τμηματοποίηση του έργου σε επίπεδο διαδικασίας ανάπτυξης.
- Αφαίρεση όλων των περιττών ενεργειών της διαδικασίας ανάπτυξης
- Ανάπτυξη λογισμικού με βάση τη συνεργασία και την επικοινωνία
- Ανάπτυξη μιας πρώτης έκδοσης σε διάστημα μερικών εβδομάδων για τη λήψη άμεσης και γρήγορης ανατροφοδότησης
- Εύρεση και χρήση των απλούστερων λύσεων
- Μείωση της απαραίτητης τεκμηρίωσης του έργου
- Συνεχής επαφή, διαθεσιμότητα και συμμετοχή του πελάτη



Εικόνα 62: Ανατροφοδότηση κατά τη διάρκεια του κύκλου ζωής της μεθοδολογίας XP

Ο ακραίος προγραμματισμός ορίζει ένα ενοποιημένο σύνολο από 12 πρακτικές, το οποίο μπορεί να οδηγήσει στην επιτυχία ενός έργου. Επιπρόσθετα η XP μπορεί να

χρησιμοποιηθεί από κοινού με άλλα πλαίσια ανάπτυξης, όπως είναι η μεθοδολογία RUP. Ο συνδυασμός αυτός έχει ονομαστεί διαδικασία dX (Charvat, 2003). Στην Εικόνα 63 που ακολουθεί εμφανίζονται εικονογραφημένα οι 12 πρακτικές της XP:

Εικόνα 63: Οι δώδεκα πρακτικές της μεθοδολογίας XP

Ολοκληρώνοντας, ο ακραίος προγραμματισμός επικεντρώνεται στην κατασκευή του κώδικα ενθαρρύνοντας τη γραφή πολλών τεστ πριν ακόμη από τον κώδικα, τον προγραμματισμό σε ζεύγη, την ανακατασκευή του κώδικα (refactoring), τη συνεχή επικοινωνία με τους πελάτες και την προσαρμογή στις τρέχουσες αλλαγές. Τέλος, εισάγει τη χρήση των CRC (Class-Responsibility-Collaborator) καρτών όπου γίνεται αποσύνθεση των απαιτήσεων σε κλάσεις (class), αναγράφονται οι υπευθυνότητες (responsibilities) και οι συνεργαζόμενες κλάσεις (collaborators) (Σταμέλος και Σφέτσος, 2003). Στη συνέχεια αναπαριστάται διαγραμματικά ένας κύκλος ζωής έργου.

Εικόνα 64: Κύκλος ζωής έργου με βάση τη μεθοδολογία XP

Ωστόσο, αν και η XP κατατάσσεται στις πολλά υποσχόμενες μεθοδολογίες που αξίζει να μελετηθεί, δε θα πρέπει να χρησιμοποιείται σε μεγάλα έργα. Η XP είναι μια περιορισμένη διαδικασία που χρειάζεται προσθήκες για να ταιριάξει σε ένα ολοκληρωμένο έργο ανάπτυξης. Για μια μικρή ομάδα ανάπτυξης έργου που εργάζεται σε ένα περιβάλλον σχετικά υψηλής εμπιστοσύνης όπου ο χρήστης είναι αναπόσπαστο κομμάτι της ομάδας ανάπτυξης, η XP μπορεί να αποδώσει τα βέλτιστα (Charvat, 2003).

Scrum

Η Scrum είναι μια ευέλικτη μεθοδολογία διοίκησης έργου που απευθύνεται κυρίως σε έργα λογισμικού. Δανείζεται το όνομά της από ένα είδος παιχνιδιού ράγκμπι και υιοθετεί αρκετά στοιχεία της φιλοσοφίας του που αφορούν το μηχανισμό επαναφοράς στο παιχνίδι της μπάλας που έχει βγει από αυτό. Η μεθοδολογία έχει τη θεωρητική της βάση σε μια εμπειρική διαδικασία ελέγχου, ενώ οι πρακτικές της προέρχονται από την εκτεταμένη χρήση της στην πράξη. Οι δύο πυλώνες της μεθοδολογίας Scrum στην ανάπτυξη έργου λογισμικού είναι:

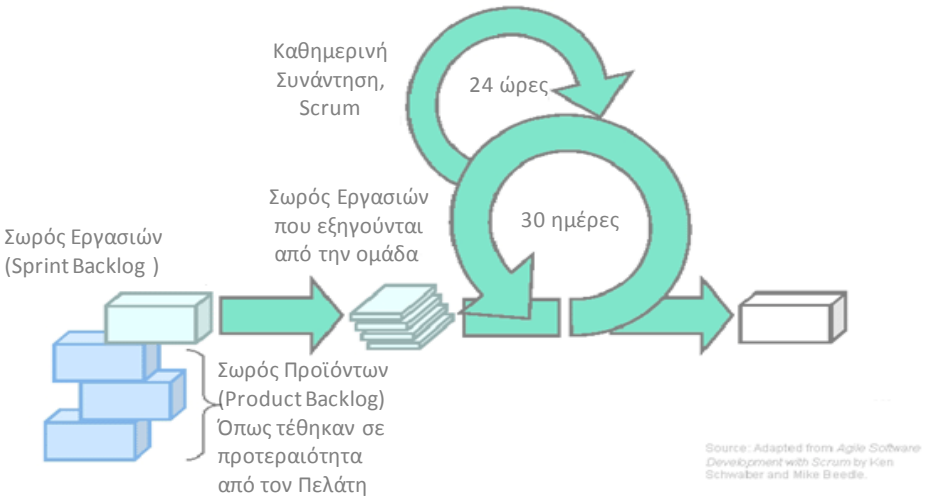
- *Η ενδυνάμωση της ομάδας* (team empowerment): Η μεγιστοποίηση της ικανότητας της ομάδας να ανταποκριθεί με ευέλικτο τρόπο στις προκύπτουσες προκλήσεις αποτελεί λύση στη μερική κατανόηση των προβλημάτων που πάντοτε υπάρχει στα πρώτα στάδια.
- *Η προσαρμοστικότητα* (adaptability): Η ομάδα διατηρεί τις ισορροπίες με τη δημιουργία επαυξητικών φάσεων, και την απομόνωση τυχόν διαταραχών από

το εξωτερικό περιβάλλον σε κάθε μια από αυτές. Οι επαυξητικές φάσεις διακόπτονται κάθε τριάντα μέρες έτσι ώστε η ομάδα αλλά και η διοίκηση να μπορέσει να αξιολογήσει το τι πρέπει να γίνει κατά τη διάρκεια της επόμενης φάσης. Η απόφαση αυτή βασίζεται στο τι έχει πετύχει η ομάδα μέχρι εκείνη τη στιγμή και τι προστάζει το περιβάλλον πως είναι το επόμενο σημαντικό πράγμα που πρέπει να γίνει.

Απώτερος στόχος της μεθοδολογίας Scrum είναι η παραγωγή ποιοτικού λογισμικού μέσα σε μια αλληλουχία τριών έως οχτώ χρονικών διαστημάτων (sprints) που συνήθως διαρκούν τριάντα ημέρες. Πριν ξεκινήσει ένα sprint, ορίζεται η απαιτούμενη λειτουργικότητά του και στη συνέχεια δίνεται η αρμοδιότητα στην ομάδα έργου να το αναπτύξει και να το εκτελέσει. Στο τέλος των τριάντα ημερών, του sprint, ο διοικητής έργου επιθεωρεί τα αποτελέσματα, αποτιμά τις αλλαγές του επιχειρησιακού περιβάλλοντος και εμπειρικά καθορίζει τι θα γίνει στη συνέχεια. Ο στόχος είναι η σταθεροποίηση των απαιτήσεων κατά τη διάρκεια του sprint. Η ομάδα έργου διενεργεί καθημερινές συναντήσεις, το αποκαλούμενο scrum, κατά τις οποίες ανατρέχει γρήγορα στις δραστηριότητες της επόμενης ημέρας. Στο τέλος κάθε sprint:

- Παρουσιάζεται στον πελάτη η εξέλιξη του έργου.
- Ενοποιείται και ελέγχεται ένα σημαντικό τμήμα του συστήματος που αναπτύσσεται.
- Η ομάδα δεσμεύεται για το επόμενο sprint.
- Επιβεβαιώνεται η πρόοδος του έργου.

Η διοίκηση έργου στη μεθοδολογία Scrum εστιάζεται περισσότερο στην επανάληψη και τη διοίκηση των διανεμηθέντων τμημάτων του έργου, την επίλυση λαθών, και την προετοιμασία της ομάδας για την επόμενη διανομή του έργου.



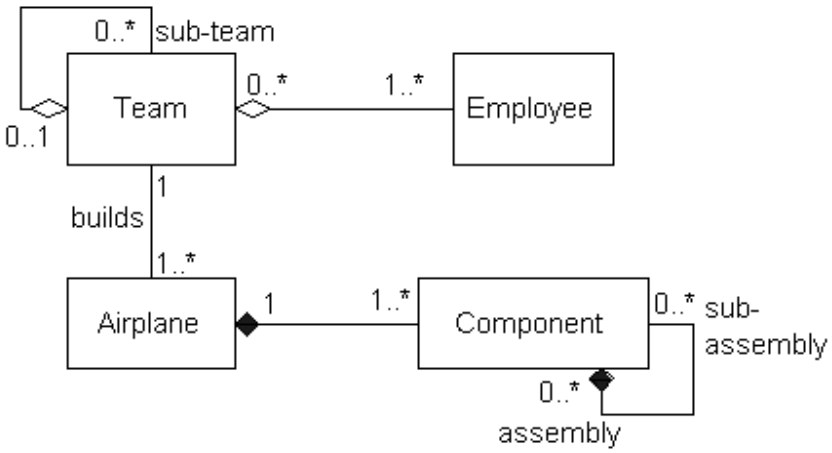
Εικόνα 65: Κύκλος ζωής έργου με βάση τη μεθοδολογία Scrum

Τελειώνοντας την περιγραφή της Scrum στα πλεονεκτήματά της συγκαταλέγονται τα εξής (Charvat, 2003):

- Αυξάνει την εμπλοκή του τελικού χρήστη στη διαδικασία ανάπτυξης.
- Επιτρέπει στους χρήστες να αλλάξουν ή να δημιουργήσουν νέες απαιτήσεις ενώ το έργο προχωράει.
- Εστιάζει την προσοχή της ομάδας στην ανάπτυξη της πιο σημαντικής λειτουργίας του συστήματος κάθε φορά.
- Στοχεύει στη δημιουργία νέας έκδοσης του συστήματος κάθε 30 ημέρες.
- Μεγιστοποιεί το αποτέλεσμα του έργου.
- Διασφαλίζει την ανάπτυξη της λειτουργικότητας που πραγματικά θέλουν οι χρήστες, ελαχιστοποιώντας περιττή λειτουργικότητα του συστήματος και συνεπώς και το επιπλέον κόστος.

8. Παράρτημα Β - Ενδεικτικές απαντήσεις στις δραστηριότητες

Δραστηριότητα 1



Ένα αεροπλάνο αποτελείται από εξαρτήματα (components) τα οποία με τη σειρά τους περιέχουν άλλα εξαρτήματα, δημιουργώντας μια ιεραρχία εξαρτημάτων. Επιπλέον, κάθε εξάρτημα ανήκει αποκλειστικά σε ένα αεροπλάνο, σε ένα υποσύστημα κ.λπ.

Αντίστοιχα, μια ομάδα αποτελείται από εργαζομένους. Κάθε εργαζόμενος μπορεί να ανήκει σε περισσότερες από μια μονάδα, πολλαπλότητα 0..*, και συνεπώς η σχέση είναι σχέση συναρμολόγησης.

[154]

Δραστηριότητα 2

1) Το σύστημα διαχείρισης έργων αλληλεπιδρά με τους παρακάτω χειριστές:

   a. Ανθρώπινος πόρος[8]. Οι ανθρώπινοι πόροι μπορεί να είναι διαχειριστές έργου (project managers), διαχειριστές πόρων (resource managers), και διαχειριστές συστήματος (system administrators). Στη γενική περίπτωση, η διαχείριση ενός έργου περιλαμβάνει περισσότερους ρόλους αλλά με σκοπό την απλοποίηση της δραστηριότητας παρουσιάζουμε μόνο μερικούς από αυτούς.

   b. Εξωτερικά συστήματα όπως τα εφεδρικά συστήματα (backup systems), και οι εξυπηρετητές δικτύου (project web servers).

   c. Συσκευές όπως οι εκτυπωτές (printers).

   Οι ΠΧ που αντιστοιχούν ανά χειριστή είναι:

   a. Ανθρώπινος πόρος.

      - Login
      - Logout

   b. Διαχειριστής έργου

      - Manage Project η οποία εξειδικεύεται από τις ΠΧ Maintain Task, Maintain Activity, Maintain Project και περιλαμβάνει την ΠΧ Log Activity
      - Publish Status η οποία συνδέεται με σχέση κληρονομικότητα με τις ΠΧ Generate Report και Generate Web Site. Η ΠΧ Generate Report (Παραγωγή Αναφοράς) έχει ως δευτερεύοντα χειριστή τον Printer, ενώ η ΠΧ Generate Web Site έχει ως δευτερεύοντα χειριστή το Project Web Server.

   c. Διαχειριστής συστήματος

[8] Χρησιμοποιούμε τον όρο ανθρώπινος πόρος (human resource) αντί για τον απλούστερο εργαζόμενος ώστε να δώσουμε έμφαση αφενός στη διαχείριση αυτών (χρόνος, κόστος) και επιπλέον να διαφοροποιήσουμε τους ανθρώπινους πόρους από άλλα είδη πόρων που χρησιμοποιούνται στο έργο.

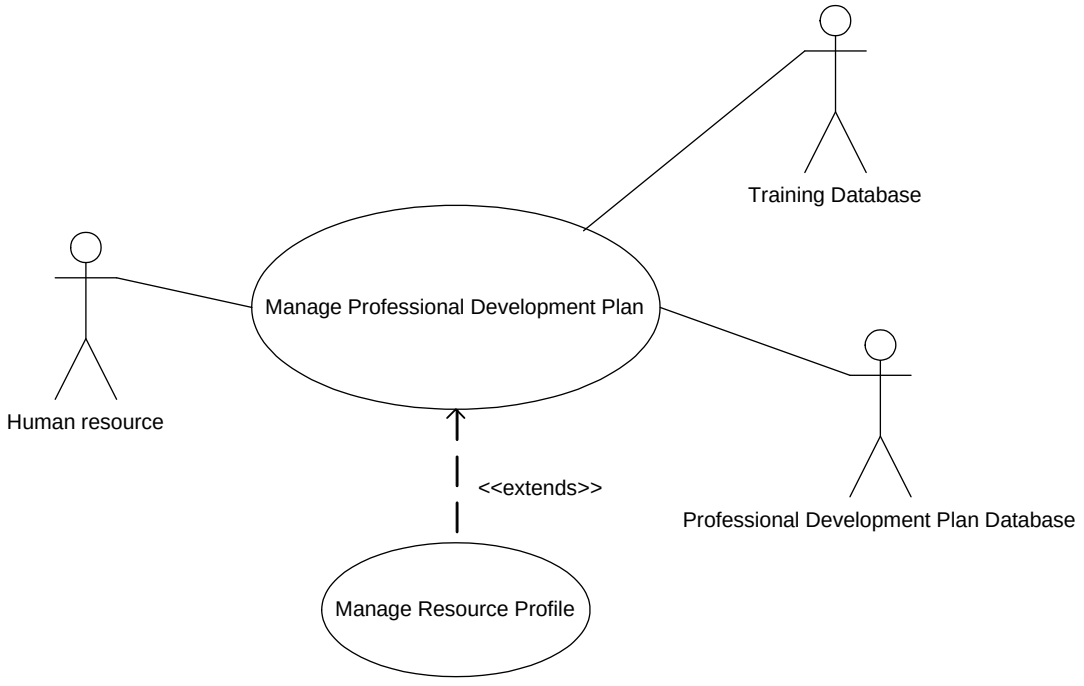
- Administer system. Η ΠΧ Administer system εξειδικεύεται από τις ΠΧ Startup System και Shutdown System. Τα σημεία επέκτασης (extention points) αυτών των ΠΧ είναι Before Startup και After Startup και αντίστοιχα Before Shotdown και After Shotdown.

d. Διαχειριστής πόρων (resource manager)

- Manage Resources

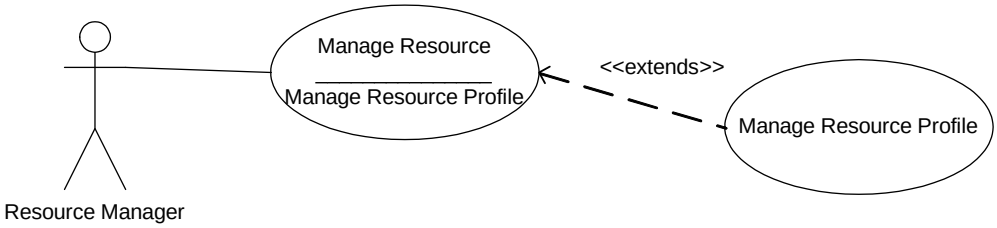
2)

a. Οι εργαζόμενοι (human resource) διαχειρίζονται το πλάνο επαγγελματικής ανάπτυξης (professional development plan). Η διαχείριση του πλάνου περιλαμβάνει τη διαχείριση του προφίλ του εργαζομένου, την πρόσβαση στο πρόγραμμα εκπαιδεύσεων εργαζομένων που είναι αποθηκευμένο σε βάση δεδομένων. Το πλάνο επαγγελματικής ανάπτυξης εργαζομένου θα πρέπει να αποθηκεύεται σε βάση δεδομένων, η οποία δεν αποτελεί τμήμα του συστήματος διαχείρισης έργων.

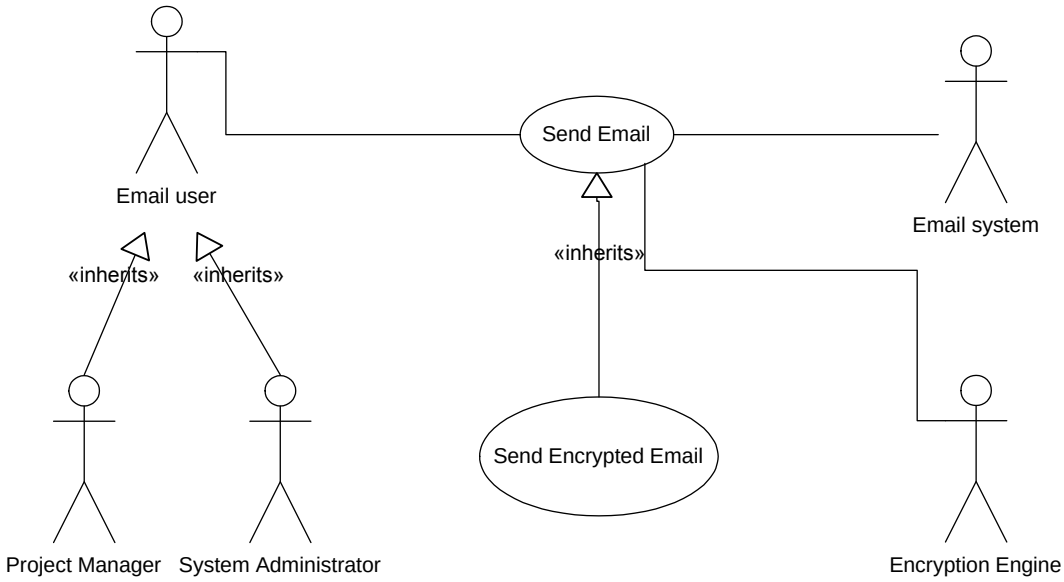


b. Ο διαχειριστής πόρων (resource manager) θα πρέπει να έχει πρόσβαση σε λειτουργικότητα που θα επιτρέπει τη διαχείριση πόρων. Μια

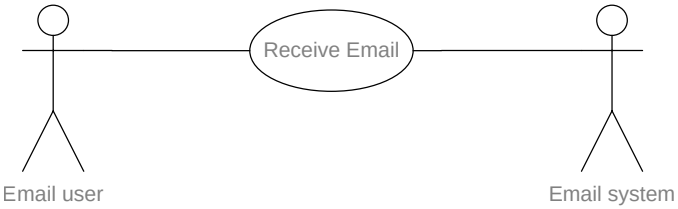
επιλογή είναι η διαχείριση του προφίλ του πόρου (resource profile). Η ίδια λειτουργικότητα θα πρέπει να είναι διαθέσιμη και στο διαχειριστή ανθρωπίνων πόρων.



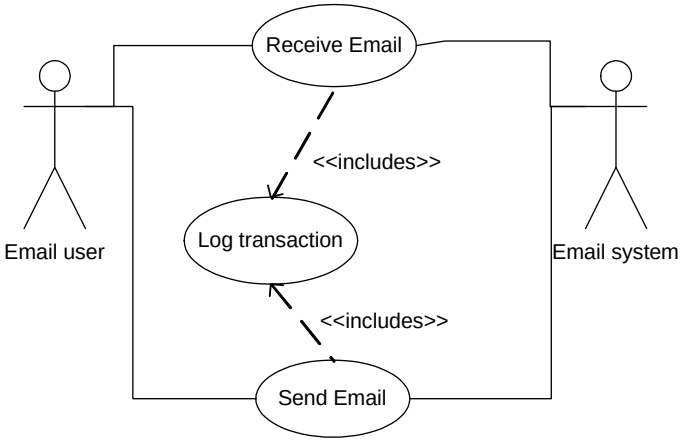
c. Ο διαχειριστής έργου (project manager) ή ο διαχειριστής του συστήματος (administrator), θα πρέπει να είναι σε θέση να αποστέλλει e-mail. Το σύστημα e-mail δεν είναι μέρος του συστήματος διαχείρισης έργων. Όταν αποστέλλουμε e-mail θα πρέπει να έχουμε τη δυνατότητα της ασφαλούς αποστολής, με τη χρήση κρυπτογραφικού μηχανισμού, μηχανισμό τον οποίο θα προμηθευτούμε από εξειδικευμένη εταιρεία.



d. Το σύστημα e-mail όταν παραλαμβάνει e-mail ενημερώνει τους χρήστες. Ο χρήστης του συστήματος μπορεί να ελέγχει αν έχει παραληφθεί καινούργιο e-mail.



e. Όταν αποστέλλουμε ή παραλαμβάνουμε e-mail, το σύστημα θα καταγράφει τη συναλλαγή.



3) Η προτεινόμενη σειρά ανάπτυξης των ΠΧ είναι η ακόλουθη:

a. Η ΠΧ Log Activity θα πρέπει να αναπτυχθεί πριν από τις ΠΧ Manage Project, Manage Resource και Administer System.

b. Η ΠΧ Manage Project θα πρέπει να αναπτυχθεί πριν από τις ΠΧ Maintain Project, Maintain Activity και Maintain Task.

c. Η ΠΧ Publish Status θα πρέπει να αναπτυχθεί πριν από τις ΠΧ Generate Report και Generate Website.

d. Η ΠΧ Administer System θα πρέπει να αναπτυχθεί πριν από τις ΠΧ Startup System και Shutdown System.

e. Η ΠΧ Startup System θα πρέπει να αναπτυχθεί πριν από την ΠΧ Restore Data .

f. Η ΠΧ Shutdown System θα πρέπει να αναπτυχθεί πριν από την ΠΧ Backup System.

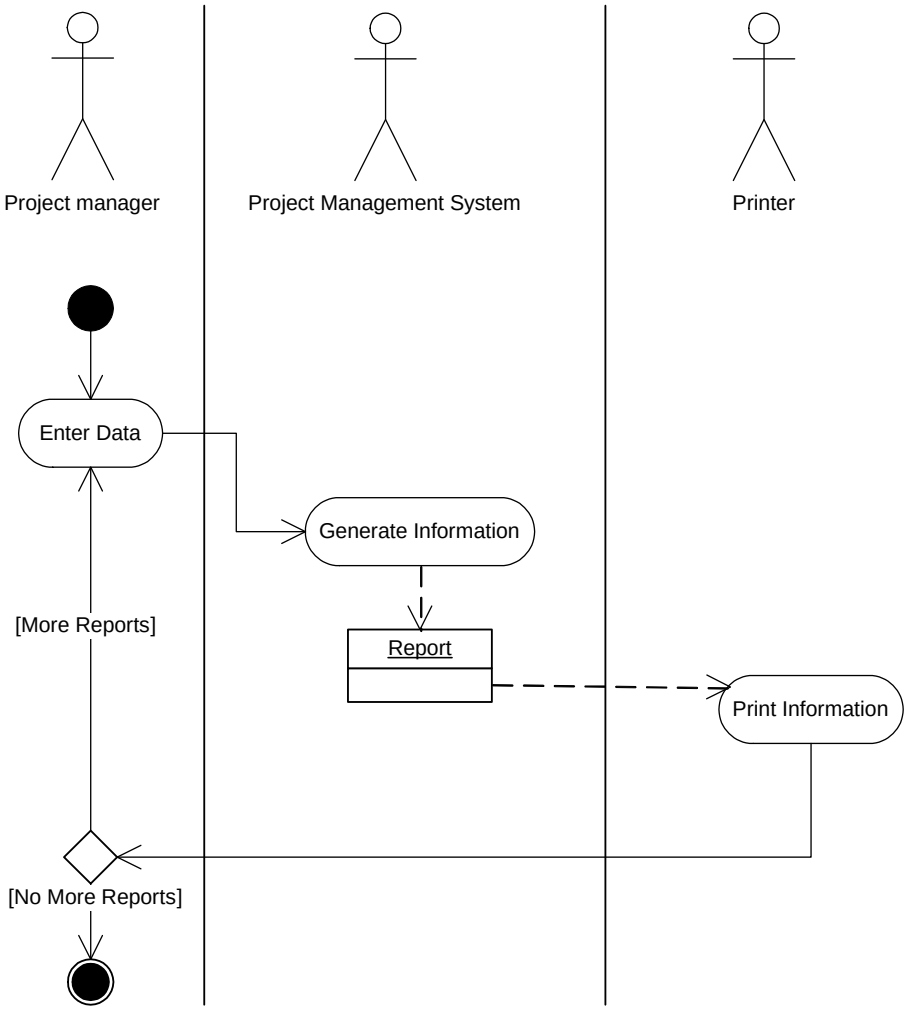
g. Η ΠΧ Manage Professional Development Plan θα πρέπει να αναπτυχθεί πριν από την ΠΧ Manage Resource Profile.

h. Η ΠΧ Manage Resource θα πρέπει να αναπτυχθεί πριν από την ΠΧ Manage Resource Profile.

i. Η ΠΧ Log Transaction θα πρέπει να αναπτυχθεί πριν από τις ΠΧ Send Email και Receive Email .

j. Η ΠΧ Send Email θα πρέπει να αναπτυχθεί πριν από την ΠΧ Send Encrypted Email .

Δραστηριότητα 3



Έχουμε τρεις διαδρόμους (swimlanes) δράσεων: του Διαχειριστή Έργου (Project Manager), του Συστήματος Διαχείρισης Έργων (Project Management System) και του Εκτυπωτή (Printer).

Αρχικά, ο Διαχειριστής Έργου εισάγει τις παραμέτρους για την παραγωγή αναφοράς στην κατάσταση δράσης (action state) Enter Data. Στη συνέχεια, το Σύστημα Διαχείρισης Έργων παράγει την αναφορά χρησιμοποιώντας την κατάσταση δράσης, Generate Information.

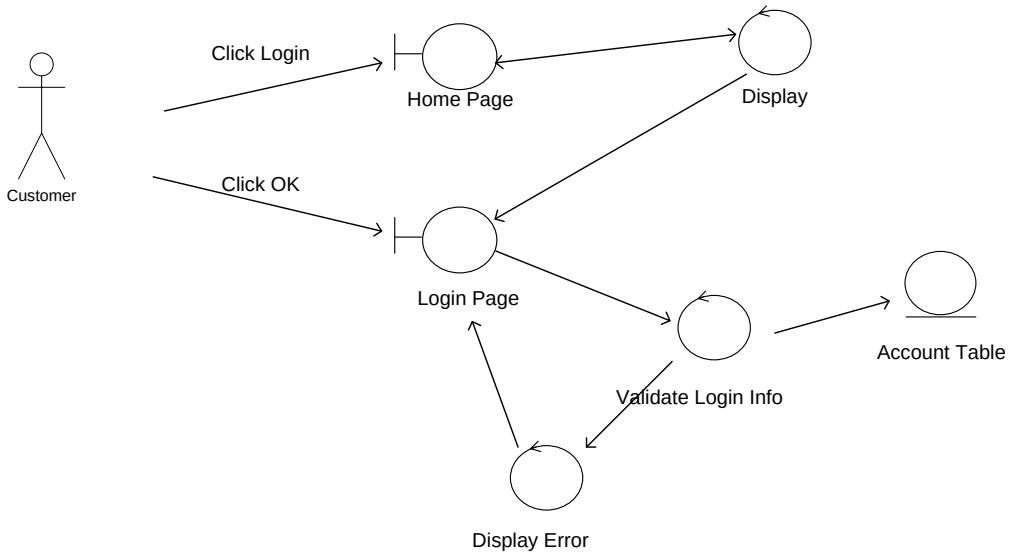
Η κατάσταση δράσης Generate Information δημιουργεί ένα Report αντικείμενο το οποίο στέλνεται στην κατάσταση δράσης Print Information.

Δραστηριότητα 4

Τα λάθη που έχει το διάγραμμα είναι τα ακόλουθα:

1. Το συνοριακό αντικείμενο Home Page επικοινωνεί με το αντικείμενο οντοτήτων Account Table
2. Το αντικείμενο intercept request δε θα έπρεπε να υπάρχει αφού θα έπρεπε να εμφανίζεται στο λεπτομερή σχεδιασμό
3. Με την ίδια λογική δε θα έπρεπε να εμφανίζεται η μέθοδος countBadPasswords

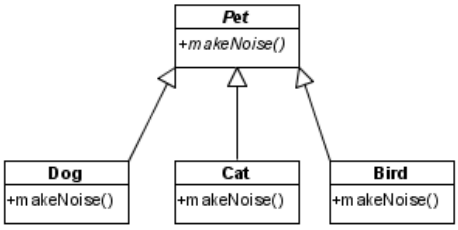
Μια διορθωμένη έκδοση του διαγράμματος εμφανίζεται παρακάτω.



Δραστηριότητα 5

Όπως είπαμε ο πολυμορφισμός είναι χρήσιμος όταν υπάρχουν αντικείμενα που μοιράζονται μια συμπεριφορά αλλά με διαφορές.

Στο παράδειγμά μας έχουμε κατοικίδια ζώα το καθένα από τα οποία έχει διαφορετική φωνή και συνεπώς κάνει διαφορετικό θόρυβο (make noise). Αν προστεθεί ένα καινούργιο κατοικίδιο, μια καινούργια κλάση, το σχήμα θα μείνει το ίδιο χωρίς ουσιαστικές αλλαγές.



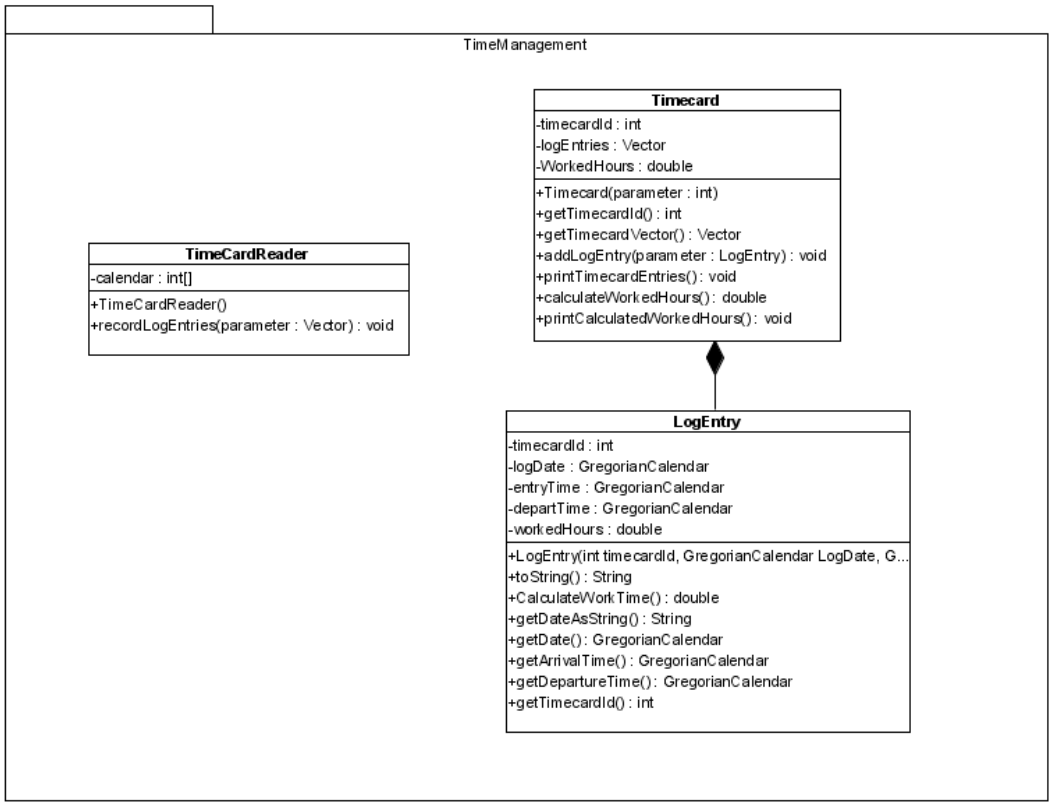
Δραστηριότητα 6

Σχεδόν όλες οι κλάσεις της java υλοποιούν υπερφόρτωση μεθόδων. Για παράδειγμα, η κλάση Vector που έχουμε χρησιμοποιήσει έχει τέσσερεις διαφορετικούς κατασκευαστές

1. Vector(), κατασκευάζει ένα κενό Vector με αρχική χωρητικότητα 10.
2. Vector(Collection c), κατασκευάζει ένα κενό Vector για τη συγκεκριμένη συλλογή και επιστρέφει τον iterator.
3. Vector(int initialCapacity), κατασκευάζει ένα κενό Vector με αρχική χωρητικότητα initialCapacity.
4. Vector(int initialCapacity, int capacityIncrement), κατασκευάζει ένα κενό Vector με αρχική χωρητικότητα initialCapacity, του οποίου η χωρητικότητα αυξάνεται κατά capacityIncrement.

Δραστηριότητα 7

Στην παρακάτω εικόνα παρουσιάζεται ο αναλυτικός σχεδιασμός του πακέτου TimeManagement.



Δραστηριότητα 8