ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Αξιολόγηση των Σχεδιαστικών Προτύπων και της Ποιότητας του Λογισμικού μέσω Μετρικών, στις Περιπτώσεις Προσθήκης Λειτουργικότητας και Αναδόμησης του Κώδικα

Του φοιτητή

Αντωνιάδη Χρήστου

Αρ. Μητρώου: 04/2517

Επιβλέπων καθηγητής

Σφέτσος Παναγιώτης

Θεσσαλονίκη 2012

ΠΡΟΛΟΓΟΣ

Αντικείμενο της παρούσης πτυχιακής είναι η ερευνητική αξιολόγηση των σχεδιαστικών προτύπων και της ποιότητας του λογισμικού μέσω μετρικών στις περιπτώσεις προσθήκης λειτουργικότητας και αναδόμησης του κώδικα.

Η πτυχιακή περιλαμβάνει τα παρακάτω:

1. Αναλυτική παρουσίαση των σχεδιαστικών και GRASP προτύπων-απεικονίσεις

ΠΕΡΙΛΗΨΗ

Η παρούσα εργασία αφορά στην τεχνολογία λογισμικού.

Συγκεκριμένα θα αναλυθούν διεξοδικά τα πρότυπα GRASP τα οποία είναι μία συλλογή από αρχές και κανόνες με τις οποίες μπορεί ένας μηχανικός λογισμικού να αντιληφθεί την λειτουργικότητα των αντικειμένων που παράγονται μέσα στο έργο, τις αρχές του αντικειμενοστρεφούς προγραμματισμού και τις εφαρμογές του, καθώς και την χρήση προτύπων σχεδίασης ως μέρος της σωστής εφαρμογής των θεωρητικών αρχών της τεχνολογίας λογισμικού στην πράξη.

ΠΕΡΙΕΧΟΜΕΝΑ

Ευρετήριο σχημάτων

ΕΙΣΑΓΩΓΗ

Η εργασία αυτή έχει ως σκοπό την λεπτομερή καταγραφή των προτύπων σχεδίασης:

Συγκεκριμένα θα παρουσιαστούν τέσσερεις κατηγορίες προτύπων λογισμικού

- Κατασκευαστικά (Creational)
- Δομικά (Structural)
- Συμπεριφορικά (Behavioral)
- Πρότυπα GRASP[1].

Η τελευταία κατηγορία (GRASP) θεωρείται από πολλούς και η πλέον σημαντική και άμεσα εφαρμόσιμη, για αυτό και αναλύονται πιο διεξοδικά. Τα εννέα αυτά πρότυπα περιγράφουν μία σειρά από κανόνες και προτροπές ορθής συγγραφής λογισμικού, σωστής τοποθέτησης και χρήσης των διάφορων τμημάτων αυτού του λογισμικού και τρόποι αποδοτικότερης συγγραφής κώδικα αλλά και επαναχρησιμοποίησης του ήδη υπάρχοντος.

Στο πρώτο κεφάλαιο γίνεται μία ανασκόπηση της κατάστασης στην τεχνολογία λογισμικού και μία περιγραφή της γλώσσας UML.

Στο δεύτερο κεφάλαιο αναλύεται η λογική των προτύπων γενικότερα, περιγράφονται τα πρότυπα GRASP ως ένα είδος προτύπων σχεδίασης, και τέλος αναλύονται διεξοδικά και τα εννέα πρότυπα GRASP.

[1] **General Responsibility Assignment Software Patterns** (or **Principles**) (Wikipedia, 2011) Πρότυπα (ή Αρχές) Ανάθεσης Γενικών Αρμοδιοτήτων στο Λογισμικό

ΚΕΦΑΛΑΙΟ 1

<Η ΜΗΧΑΝΙΚΗ ΛΟΓΙΣΜΙΚΟΥ ΣΗΜΕΡΑ>

ΕΙΣΑΓΩΓΗ

”Designing object-oriented software is hard, and designing reusable object-oriented software is even harder” (Gamma, Helm, Johnson, & Vlissides, 1995).

Ο μηχανικός λογισμικού πρέπει να κατασκευάσει τα αντικείμενα έτσι ώστε να σχετίζονται μεταξύ τους, να σχεδιάσει και να αναδείξει σωστά την ιεραρχία μεταξύ τους και φυσικά να περιγράψει και να δημιουργήσει ένα πολύπλοκο σύστημα μεταξύ των σχέσεων. Ενώ το λογισμικό που θα κατασκευάσει θα πρέπει προφανώς να λύνει σωστά και αξιόπιστα το ζητούμενο για το οποίο κατασκευάζεται, θα πρέπει την ίδια στιγμή να είναι αρκετά ευέλικτο και παραμετροποιήσιμο ώστε να μπορεί να χρησιμοποιηθεί και για μελλοντικές παραλλαγές του σημερινού προβλήματος, ή σε αντίστοιχα και παρεμφερή με το υπάρχον, προβλήματα.

Συνήθως όλα αυτά δεν μπορούν να επιτευχθούν στην πρώτη έκδοση ενός λογισμικού. Χρειάζεται κόπος, χρόνος και πολύ μεράκι και γνώση για να φτάσουν κάποιες μελλοντικές εκδόσεις του ώριμου πια λογισμικού για να προσεγγίσουν στοιχειωδώς κάποιες από αυτές τις απαιτήσεις.

ΥΠΟΚΕΦΑΛΑΙΟ 1.1 Τι είναι η UML;

Η γλώσσα UML, (Unified Modeling Language), Ενοποιημένη Γλώσσα Μοντελοποίησης, είναι μία γραφική γλώσσα γενικού σκοπού, η οποία χρησιμοποιείται για τον προσδιορισμό, την οπτικοποίηση, ανάπτυξη και τεκμηρίωση των κατασκευασμάτων ενός συστήματος λογισμικού, παρέχοντας μία σημειογραφική προσέγγιση για την περιγραφή αντικειμενοστρεφών λύσεων. Μπορεί να προσαρμοστεί ώστε να ταιριάζει σε διάφορες καταστάσεις ανάπτυξης και κύκλους ζωής λογισμικού (Βούλγαρης, 2009)

Η UML θεωρείται πλέον ως δεδομένη, δηλαδή ως πρότυπη γλώσσα αναπαράστασης στην αντικειμενοστρεφή σημειογραφία.

Υπάρχουν τρείς βασικοί τρόποι χρήσης/εφαρμογής της UML:

- **Η χρήση σχεδίων χωρίς ιδιαίτερο φορμαλισμό.** Συνήθως αυτά τα σχήματα είναι ζωγραφισμένα στο χέρι, χωρίς να είναι ολοκληρωμένα και συνεπή. Σκοπό έχουν να καταδείξουν τα ενδιαφέροντα και «δύσκολα» κομμάτια ενός προβλήματος υπό συζήτηση, οπτικοποιώντας το.
- **Σχεδιαστικές αναπαραστάσεις:** Αρκετά λεπτομερή σχεδιαγράμματα. Συνήθως γίνονται για τις ανάγκες reverse engineering ή καλύτερης κατανόησης υπάρχοντος κώδικα σε άλλα UML διαγράμματα, ή για την κατευθείαν παραγωγή κώδικα.
- **Η UML ως πλήρης προγραμματιστική γλώσσα:** Πλήρης αναπαράσταση του κώδικα και του συνεπακόλουθου εκτελέσιμου προγράμματος σε UML. Ο κώδικας μάλιστα παράγεται αυτόματα αφού πρώτα ο προγραμματιστής έχει ολοκληρώσει την συγγραφή του προγράμματος σε UML. Αρκετά δύσκολη και επίπονη μέθοδος, η οποία μάλιστα είναι ακόμη ανοιχτή ερευνητικά όσον αφορά στις δυνατότητες, τα εργαλεία, την αξιοπιστία και την ευχρηστία (Craig Larman, 2004)

Η χρήση Agile μεθόδων προτρέπει την χρήση UML ως την πρώτη περίπτωση, δηλαδή ως σχήματα συνήθως χειρόγραφα γιατί συνήθως χρειάζεται ελάχιστος

κόπος και χρόνος για την κατασκευή τους και η ωφέλεια είναι πολύ υψηλή.

ΥΠΟΚΕΦΑΛΑΙΟ 1.1 Τι είναι το Πρότυπο Σχεδίασης;

Ο όρος «Πρότυπο Σχεδίασης» έρχεται από την επιστήμη της αρχιτεκτονικής. Συγκεκριμένα στο διάσημο πλέον βιβλίο του, ο Christopher Alexander περιγράφει το Πρότυπο Σχεδίασης, ως εξής:

«Ένα πρότυπο (σχεδίασης) είναι μία προσεκτική περιγραφή μίας επιτυχημένης (στον χρόνο) λύσης ενός συχνά εμφανιζόμενου προβλήματος στην κατασκευή ενός δομικού περιβάλλοντος, η οποία περιγράφει επιτυχώς μία από τις συντεταγμένες προσπάθειες να ολοκληρωθεί το οικοδόμημα στο οποίο αναφέρεται».

Κάθε πρότυπο περιγράφει ένα πρόβλημα το οποίο εμφανίζεται ξανά και ξανά στο συγκεκριμένο περιβάλλον και μετά περιγράφει τον βασικό τρόπο λύσης αυτού του προβλήματος, με τέτοιο τρόπο ώστε να μπορεί να χρησιμοποιηθεί ξανά χιλιάδες φορές χωρίς να εφαρμοστεί με τον ίδιο τρόπο δύο φορές (Alexander κ.ά., 1977).

Όσον αφορά στο λογισμικό, τα πρότυπα σχεδίασης είναι αυτά που μας επιτρέπουν να περιγράψουμε κομμάτια σχεδίασης και επαναχρησιμοποίηση κάποιων ιδεών σχεδίασης, βοηθώντας έτσι τους μηχανικούς λογισμικού να επωφεληθούν από την υπάρχουσα γνώση άλλων προγραμματιστών. Τα πρότυπα σχεδίασης λογισμικού δίνουν όνομα και υπόσταση σε αφαιρετικά τμήματα κώδικα που λύνουν συγκεκριμένα προβλήματα, κανόνες και καλές πρακτικές (best practices) στον αντικειμενοστρεφή προγραμματισμό. Κανείς σώφρων μηχανικός λογισμικού δεν θέλει να ξεκινήσει από το μηδέν, όταν μπορεί εύκολα να οικειοποιηθεί την ήδη υπάρχουσα και ελεγμένη ως προς το αποτέλεσμα γνώση (C. Larman, 2002).

Υπάρχουν τέσσερεις κατηγορίες προτύπων σχεδίασης λογισμικού**:**

- **Κατασκευαστικά Πρότυπα (Creational Patterns)** είναι αυτά τα σχεδιαστικά πρότυπα που προσπαθούν να δημιουργήσουν αντικείμενα με έναν τρόπο που να αρμόζει στην περίσταση που εξετάζεται. Η βασική δομή δημιουργίας αντικειμένων συναντά πολλά προβλήματα σε επιμέρους περιπτώσεις. Τα κατασκευαστικά

πρότυπα προσπαθούν να λύσουν αυτό το πρόβλημα, προσπαθώντας να ελέγξουν τους τρόπους δημιουργίας των αντικειμένων. Υποκατηγορίες τους είναι το πρότυπο δημιουργίας αντικειμένων, και το πρότυπο δημιουργίας κλάσεων

- **Δομικά Πρότυπα (Structural Patterns)** τα οποία έχουν σκοπό να απλοποιήσουν και να ομογενοποιήσουν τα σχεδιαστικά πρότυπα προσφέροντας έναν εύκολο τρόπο δημιουργίας σχέσεων μεταξύ οντοτήτων
- **Συμπεριφορικά Πρότυπα (Behavioral Patterns)** τα οποία αναγνωρίζουν κοινούς τρόπους επικοινωνίας μεταξύ αντικειμένων και τους μετατρέπουν σε σχεδιαστικό πρότυπο (Gamma κ.ά., 1995).
- **Πρότυπα Ανάθεσης Γενικών Αρμοδιοτήτων Λογισμικού ή GRASP (General Responsibility Assignment Software Pattern)** τα οποία ουσιαστικά είναι οδηγίες ανάθεσης ευθυνών σε κλάσεις και αντικείμενα στον αντικειμενοστρεφή προγραμματισμό

Στην παρούσα πτυχιακή, ιδιαίτερο βάρος δίνεται στην ανάπτυξη των προτύπων GRASP λόγω της σπουδαιότητας που έχουν στην ανάπτυξη έργων αντικειμενοστρεφούς προγραμματισμού σήμερα.

ΥΠΟΚΕΦΑΛΑΙΟ 1.2 Περιγραφή των Προτύπων Σχεδίασης

Γενικά, ένα πρότυπο σχεδίασης λογισμικού αποτελείται από τέσσερα βασικά τμήματα:

1. **Το όνομα του προτύπου (pattern name),** το οποίο είναι προφανώς το χαρακτηριστικό που μας βοηθάει να περιγράψουμε το πρόβλημα, τις λύσεις και τις επιπτώσεις τους, σε μία-δύο λέξεις. Ονοματίζοντας το πρότυπο μας επιτρέπει να εργαστούμε σε ένα μεγαλύτερο επίπεδο αφαιρετικότητας, μεγαλώνοντας ταυτόχρονα το διαθέσιμο λεξιλόγιο. Όταν το πρότυπο διαθέτει ένα χαρακτηριστικό όνομα, μας επιτρέπει να συζητούμε για αυτό με συναδέλφους αλλά και να αναφερόμαστε σε αυτό. Για αυτό και η εύρεση ενός σωστού ονόματος είναι και δύσκολη αλλά και ουσιώδης.
2. **Το πρόβλημα (the problem),** το οποίο περιγράφει επακριβώς τις συνθήκες στις οποίες θα εφαρμοστεί το συγκεκριμένο πρότυπο. Μπορεί να περιγράφει συγκεκριμένα προβλήματα σχεδίασης όπως τον τρόπο αναπαράστασης αλγορίθμων ως αντικείμενα. Μπορεί να καταδεικνύει κλάσεις ή αντικείμενα που είναι αποτελέσματα μίας ανελαστικής σχεδίασης. Πολλές φορές το πρόβλημα μπορεί να περιγράφει μία σειρά από συνθήκες ή υποθέσεις που θα πρέπει να ικανοποιηθούν πριν χρειαστεί να εφαρμοστεί το συγκεκριμένο πρότυπο.
3. **Η λύση (solution),** περιγράφει τα στοιχεία που εμπεριέχονται στο πρότυπο, τις μεταξύ τους σχέσεις, τις αρμοδιότητες τους και τις μεταξύ τους συνέργιες. Η λύση δεν περιγράφει ένα συγκεκριμένο μονολιθικό σχεδιασμό ή εφαρμογή, επειδή το πρότυπο είναι ένα περίγραμμα το οποίο μπορεί να χρησιμοποιηθεί σε πολλές διαφορετικές περιστάσεις. Η λύση αντίθετα προσφέρει μία αφαιρετική και γενική περιγραφή ενός προβλήματος σχεδίασης και πως μπορεί μία διάταξη κλάσεων και αντικειμένων να οδηγήσει στην λύση αυτού του προβλήματος.
4. **Τα επακόλουθα (Consequences),** είναι τα αποτελέσματα της αλληλεπίδρασης της εφαρμογής του προτύπου. Αν και συχνά τα επακόλουθα παραλείπονται κατά την περιγραφή αποφάσεων

εφαρμογής προτύπων, είναι κρίσιμα για την αποτίμηση των εναλλακτικών προτύπων και για την κατανόηση των επιπτώσεων και των ωφελειών εφαρμογής του προτύπου. Τα επακόλουθα της εφαρμογής προτύπων σχεδίασης στο λογισμικό συνήθως αφορούν στον χώρο, στο μέγεθος δηλαδή του λογισμικού και στον χρόνο, δηλαδή σε ταχύτητες εκτέλεσης. Μπορεί να αφορούν και σε γλώσσες ή τρόπους υλοποίησης και εφαρμογής. Επειδή η επαναχρησιμοποίηση είναι σημαντικός παράγοντας στον αντικειμενοστρεφή προγραμματισμό, τα επακόλουθα εφαρμογής ενός προτύπου αφορούν στην επίδραση που έχουν στο σύστημα όσον αφορά στην προσαρμοστικότητα, αναβαθμησιμότητα, μεταφερσιμότητα. Η παράθεση αυτών των αποτελεσμάτων συγκριτικά, βοηθά στην κατανόηση και συνολική αποτίμηση του προτύπου.

Τα πρότυπα σχεδίασης σίγουρα δεν είναι κομμάτια λογισμικού όπως για παράδειγμα οι συνδεδεμένες λίστες (linked lists) και πίνακες κατακερματισμού (hash tables) που μπορούν να μετατραπούν άμεσα σε κλάσεις και να επαναχρησιμοποιηθούν παντού όπου είναι απαραίτητα. Ούτε όμως είναι πολύπλοκες και ειδικές σχεδιαστικές δομές που αφορούν σε ένα συγκεκριμένο κομμάτι μίας πολύ συγκεκριμένης εφαρμογής.

Τα πρότυπα σχεδίασης είναι περιγραφές αντικειμένων και κλάσεων γενικευμένων με τέτοιο τρόπο ώστε να λύνουν ένα γενικό πρόβλημα σχεδιασμού σε ένα συγκεκριμένο περιβάλλον (Gamma κ.ά., 1995).

Η σαφής και λεπτομερής περιγραφή των προτύπων σχεδίασης θεωρείται πολύ σημαντική. Για να μπορεί να επαναχρησιμοποιηθεί το πρότυπο, κάτι που αποτελεί και το ουσιαστικό στόχο της δημιουργίας των προτύπων σχεδίασης, θα πρέπει τα πρότυπα να περιγράφονται με τέτοιο τρόπο ώστε να απεικονίζονται οι αποφάσεις που πρέπει να παρθούν, οι αλληλεπιδράσεις τους με άλλα πρότυπα, αλλά και με τον υπόλοιπο κώδικα. Επίσης πρέπει να απεικονίζονται και κάποια βασικά παραδείγματα χρήσης ώστε να μπορεί κάποιος να τα χρησιμοποιήσει άμεσα στην πράξη.

Η περιγραφή των προτύπων σχεδίασης γίνεται χρησιμοποιώντας ένα

συγκεκριμένο πλαίσιο (consistent format). Κάθε πρότυπο χωρίζεται σε τομείς ανάλογα με το περίγραμμα που περιγράφεται παρακάτω. Το περίγραμμα αυτό παρέχει ένα ενιαίο πλαίσιο στην παρεχόμενη πληροφορία κάνοντας έτσι τα πρότυπα σχεδίασης ευκολότερα στην εκμάθηση, σύγκριση και χρήση :

- **Όνομα Προτύπου και Ταξινόμηση (pattern name & classification)** Το όνομα του προτύπου όπως προαναφέρθηκε το περιγράφει συνοπτικά. Ένα σωστό όνομα είναι κρίσιμο αφού θα γίνει συνώνυμο του προτύπου και θα επιτρέπει την αναφορά σε αυτό.
- **Σκοπός (intent)** Μία συνοπτική φράση που απαντάει στην ερώτηση: τι κάνει αυτό το πρότυπο σχεδίασης; Ποιος είναι ο στόχος και ο σκοπός του; Ποια συγκεκριμένα θέματα ή προβλήματα σχεδίασης επιλύει;
- **Άλλη ονομασία (Also known as)**, Αν υπάρχει άλλη γνωστή ονομασία του προτύπου
- **Κίνητρο (Motivation),** ένα σενάριο που υποδεικνύει ένα πρόβλημα σχεδίασης και πως οι συγκεκριμένες κλάσεις και δομές αντικειμένων αυτού του προτύπου μπορούν να το επιλύσουν. Ένα τέτοιο σενάριο βοηθάει στην αφαιρετική κατανόηση του προτύπου που ακολουθεί
- **Πεδίο Εφαρμογής (Applicability)** Ποιες είναι οι περιπτώσεις κατά τις οποίες το συγκεκριμένο πρότυπο μπορεί να εφαρμοστεί; Υπάρχουν παραδείγματα ή περιπτώσεις κακής σχεδίασης στα οποία μπορεί να εφαρμοστεί το πρότυπο; Πως μπορούν να αναγνωριστούν αυτές οι περιπτώσεις;
- **Δομή (Structure)** Μία γραφική αναπαράσταση των κλάσεων του προτύπου χρησιμοποιώντας αναπαράσταση βασισμένη στο Object Modeling Technique (OMT) (Rumbaugh, Blaha, Premerlani, Eddy, & Lorenson, 1991). Επίσης μπορούν να χρησιμοποιηθούν διαγράμματα αλληλεπίδρασης (interaction diagrams) (Jacobson, Christerson, Jonsson, & Overgaard, 1992), (Booch, 1990) για να αναδειχθούν οι αλληλεπιδράσεις και οι εξαρτήσεις μεταξύ των αντικειμένων
- **Συμμετέχοντες (Participants),** οι κλάσεις και/ή τα αντικείμενα που

συμμετέχουν σε αυτό το πρότυπο σχεδίασης και οι αρμοδιότητες τους

- **Συνέργιες (Collaborations)** Πως οι συμμετέχοντες συνεργάζονται για να εκτελέσουν αρμονικά τα καθήκοντα τους
- **Επιπτώσεις (Consequences)** Πως φτάνει στους στόχους του το συγκεκριμένο πρότυπο; Ποιες είναι οι ανταλλαγές και τα αποτελέσματα της χρήσης αυτού του προτύπου; Ποιες όψεις του συστήματος μπορούν να διαφοροποιηθούν ανεξάρτητα από τις άλλες;
- **Υλοποίηση (implementation)** Ποια είναι τα δύσκολα σημεία, τα πράγματα που θα πρέπει να θεωρούνται γνωστά, οι τεχνικές που θα χρησιμοποιηθούν στην υλοποίηση του προτύπου; Υπάρχουν θέματα που άπτονται συγκεκριμένης γλώσσας προγραμματισμού ή/και φραγμοί;
- **Δείγμα Κώδικα (Sample Code)** Κομμάτια κώδικα που δείχνουν τον τρόπο υλοποίησης σε μία γλώσσα ή γλώσσες προγραμματισμού
- **Γνωστά θέματα/προβλήματα (Known Issues)** Παραδείγματα σχεδίασης που υπάρχουν σε πραγματικά συστήματα.
- **Σχετικά πρότυπα (Related Patterns)** Ποιά άλλα σχετικά πρότυπα σχετίζονται με το παρόν πρότυπο; Ποιες είναι οι κύριες διαφορές τους; Με ποια άλλα πρότυπα μαζί πρέπει να χρησιμοποιηθεί το παρόν πρότυπο; (Gamma κ.ά., 1995)

Επειδή τα πρότυπα σχεδίασης ακολουθούν τις τάσεις και τις επεκτάσεις των γλωσσών προγραμματισμού, είναι προφανές ότι στην πορεία της εξέλιξης των γλωσσών αυτών και φυσικά της αντικειμενοστρέφειας, κάποια πρότυπα πέφτουν σιγά-σιγά σε αχρηστία, ενώ κάποια καινούρια πρέπει να οριστούν.

Όπως σαφώς αναφέρεται εδώ (Schmidt, Stal, Rohnert, & Buschmann, 2000) σελίδα 444, η σχετική βιβλιογραφία με την μεγαλύτερη επίπτωση στο παρελθόν έχει επικεντρωθεί σε συμπαγή πρότυπα και γλώσσες, συχνά προερχόμενη από τα μεγάλα πλαίσια (frameworks) αντικειμενοστρεφών εφαρμογών. Η επόμενη γενιά

αντικειμενοστρεφών εφαρμογών θα εμπεριέχει τα πρότυπα σχεδίασης.

Σε αυτή την μελλοντική πορεία, κάποια πρότυπα θα πέσουν ή έχουν πέσει ήδη σε αχρηστία και για αυτό το λόγο κρίθηκε σκόπιμο να παραλειφθούν από αυτή την εργασία.

ΕΠΙΛΟΓΟΣ

Σε αυτό το κεφάλαιο έγινε μία αναφορά στην γλώσσα UML και στα πρότυπα σχεδίασης. Στα επόμενα κεφάλαια θα αναπτυχθούν όλες οι οικογένειες προτύπων λογισμικού. Θα γίνει λεπτομερής αναφορά στα πρότυπα σχεδίασης GRASP, αφού θεωρούνται τα σημαντικότερα ίσως πρότυπα στον αντικειμενοστρεφή προγραμματισμό.

ΚΕΦΑΛΑΙΟ 2

<ΚΑΤΑΣΚΕΥΑΣΤΙΚΑ ΠΡΟΤΥΠΑ ΣΧΕΔΙΑΣΗΣ CREATIONAL PATTERNS>

ΕΙΣΑΓΩΓΗ

Τα Κατασκευαστικά Πρότυπα συνδέονται στο αφαιρετικό επίπεδο με την διαδικασία «αποστασιοποίησης», με τον τρόπο δηλαδή που ένα σύστημα είναι ανεξάρτητο από το πώς καλούνται τα αντικείμενα, καθώς και από την δημιουργία τους, σύνθεση τους και απεικόνιση τους.

Σε αυτά τα πρότυπα υπάρχουν δύο επανεμφανιζόμενα συνέχεια θέματα. Πρώτον εμπεριέχουν την γνώση των βασικών δομικών κλάσεων που χρησιμοποιεί το σύστημα. Δεύτερον αποκρύπτουν τον τρόπο με τον οποίο τα στιγμιότυπα αυτών των κλάσεων δημιουργούνται και αλληλεπιδρούν. Το σύστημα γνωρίζει για τα αντικείμενα μόνο τις διεπαφές τους όπως προσδιορίζονται στις αφαιρετικές (abstract) κλάσεις του.

Συνεπακόλουθα, τα δημιουργικά πρότυπα δίνουν μεγάλη ελαστικότητα στο τι δημιουργείται, ποιος το δημιουργεί, πως δημιουργείται, και πότε.

Πολλές φορές τα δημιουργικά πρότυπα είναι ανταγωνιστικά. Υπάρχουν περιπτώσεις όπου θα μπορούσε να χρησιμοποιηθεί είτε το Πρωτότυπο πρότυπο, είτε το αφηρημένο εργοστάσιο. Άλλες φορές πάλι είναι συμπληρωματικά. Υπάρχουν περιπτώσεις που το πρότυπο κατασκευαστής χρησιμοποιεί κάποιο από τα υπόλοιπα μέσω των οποίων υλοποιεί την δημιουργία συγκεκριμένων μερών του συστήματος (Gamma κ.ά., 1995).

ΥΠΟΚΕΦΑΛΑΙΟ 2.1 Πρότυπο Νοητό Εργοστάσιο, Creational Pattern – Abstract Factory

Το πρόβλημα:

Παροχή μίας διεπαφής για την δημιουργία αντικειμένων που ανήκουν στην ίδια οικογένεια, ή συνδέονται στενά μεταξύ τους, χωρίς να πρέπει να καθοριστούν οι αποκλειστικές κλάσεις τους.

Επίσης γνωστό ως:

Kit

Σκοπός:

Ένα κιτ διεπαφής χρήστη που υποστηρίζει πολλαπλά στάνταρτ «επίδρασης μέσω χρήσης» “look-and-feel”. Διαφορετικές χρήσεις του κιτ, υποστηρίζουν διαφορετικές εμφανίσεις και συμπεριφορές των πλάγιων μπαρών (scroll bars) κουμπιών κτλ. Για να είναι μεταφέρσιμο ένα τέτοιο κιτ, η εφαρμογή δεν πρέπει να ορίζει εκ των προτέρων τον τρόπο απεικόνισης αυτών των αντικειμένων. Αν λοιπόν δημιουργηθούν μέσα στο αντικείμενο συγκεκριμένες κλάσεις για αυτά τα αντικείμενα, δεν θα μπορούν να τροποποιηθούν αργότερα.

Αν δημιουργηθεί μία αφηρημένη κλάση που να ορίζει αυτά τα αντικείμενα μπορεί να παρακαμφτεί αυτό το πρόβλημα.

Οι συγκεκριμένες υλοποιήσεις αυτών των αντικειμένων θα καλούν αργότερα τις συγκεκριμένες κλάσεις του κάθε επιμέρους αντικειμένου.

Εφαρμογή:

Το πρότυπο Νοητικού Εργοστασίου χρησιμοποιείται όταν:

- Το υπό κατασκευή σύστημα πρέπει να είναι ανεξάρτητο από το πώς τα επιμέρους δομικά στοιχεία του δημιουργούνται, συντίθενται και απεικονίζονται.
- Το σύστημα πρέπει να παραμετροποιηθεί με κάποιο από τα προϊόντα μίας

ομάδας (οικογένειας) προϊόντων.

- Μία ομάδα σχετιζόμενων αντικειμένων έχει σχεδιαστεί έτσι ώστε να χρησιμοποιούνται όλα μαζί, και πρέπει αυτός ο περιορισμός (της ολικής χρήσης) να παραμείνει εν ενεργεία.
- Θέλουμε να δημιουργήσουμε μία βιβλιοθήκη προϊόντων για τα οποία θέλουμε να δώσουμε προς τα έξω μόνο τις διεπαφές τους, και όχι τις υλοποιήσεις τους.

Συνέργειες

Συνήθως ένα και μοναδικό στιγμιότυπο μίας τέτοιας κλάσης δημιουργείται την στιγμή της εκτέλεσης του λογισμικού (run-time). Η δημιουργία διαφορετικών αντικειμένων της ίδιας κλάσης περνάει στις υποκλάσεις που έχουν δημιουργηθεί για να χειρίζονται ακριβώς αυτά τα διαφορετικά ομοειδή αντικείμενα.

Ωφέλειες

Το πρότυπο νοητού εργοστασίου (abstract factory) έχει τις κάτωθι ωφέλειες και αποτελέσματα:

Απομονώνει τις καλά δομημένες κλάσεις: ελέγχει τις κλάσεις αντικειμένων που δημιουργούν οι εφαρμογές. Επειδή ενσωματώνει την ευθύνη δημιουργίας αντικειμένων, ελέγχει κάθε εφαρμογή πελάτη που προσπαθεί να δημιουργήσει τέτοια αντικείμενα. Αν χρειάζονται πολλαπλά στιγμιότυπα, αυτό γίνεται μέσω της μητρικής αφαιρετικής (abstract) κλάσης.

Κάνει ευκολότερη την ανταλλαγή αντικειμένων ομοειδών κλάσεων: κάθε κλάση του νοητού εργοστασίου, εμφανίζεται ως στιγμιότυπο μόνο μία φορά σε κάθε εφαρμογή, εκεί που δημιουργήθηκε αυτό το στιγμιότυπο. Μέσω της αρχικής αφαιρετικής (abstract) μία εφαρμογή έχει άμεση πρόσβαση σε όλα τα αντικείμενα αυτής της κλάσης. Μπορεί έτσι εύκολα να αλλάξει ή να ανταλλάξει αντικείμενα που ανήκουν στην κλάση.

Ευνοεί την συνέπεια μεταξύ προϊόντων: επειδή τα αντικείμενα μίας οικογένειας αντικειμένων είναι σχεδιασμένα ώστε να συνεργάζονται, πρέπει να υπάρχει μόνο ένα στιγμιότυπο δημιουργημένο αυτής της κλάσης για λόγους συνέπειας.

Αντιλογίες:

Η υποστήριξη νέων προϊόντων είναι αρκετά δύσκολη. Η επέκταση της αρχικής νοητής κλάσης ώστε να συμπεριλάβει καινούριες τεχνικές ή μεθόδους δεν είναι εύκολη, επειδή ακριβώς το νοητό εργοστάσιο περιορίζει και ορίζει σαφώς το είδος των προϊόντων και κλάσεων που θα δημιουργηθούν.

Στην περίπτωση δημιουργίας καινούριου προϊόντος, πρέπει να επεκταθεί η αρχική αφηρημένη κλάση και όλες οι υποκλάσεις ώστε να υποστηρίζουν το νέο προϊόν.

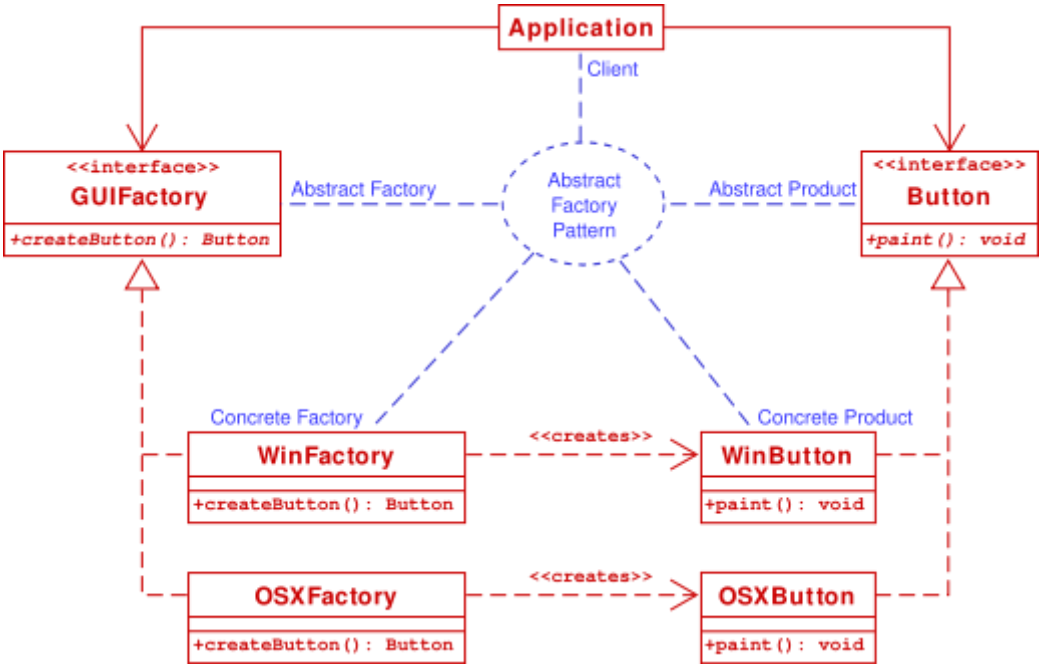
Υλοποίηση:

Χρήσιμες τεχνικές υλοποίησης του προτύπου είναι οι εξής:

1. Μοναδικό αντίγραφο για κάθε εφαρμογή για κάθε οικογένεια προϊόντων.
2. Δημιουργία των προϊόντων: Το νοητό εργοστάσιο μόνο περιγράφει τις διεπαφές. Εναπόκειται στις υποκλάσεις να τις υλοποιήσουν. Ο εύκολος τρόπος είναι η «δημοσίευση» ενός μοναδικού τρόπου κατασκευής αυτών των υποκλάσεων. Το μειονέκτημα της είναι ότι απαιτεί διαφορετικό κατασκευαστή για κάθε υποκλάση. Εάν είναι δυνατόν να υπάρξουν πολλές οικογένειες προϊόντων, μπορεί να χρησιμοποιηθεί το κατασκευαστικό πρότυπο «Πρωτότυπου» (Prototype). Μέσω του αρχικού κοινού προϊόντος για όλες τις οικογένειες, μπορεί να δημιουργείται η κλάση για το καινούριο προϊόν, «κλωνοποιώντας» την αρχική.
3. Ορίζοντας τις πιθανές επεκτάσεις της αρχικής αφηρημένης κλάσης, το πρότυπο συνήθως ορίζει έτσι τις διαφορετικές λειτουργίες που μπορεί και επιτελεί κάθε μέλος της αρχικής οικογένειας προϊόντων. Κάθε προσθήκη όμως νέου μέλους, απαιτεί την αλλαγή όλων των διεπαφών και όλων των

κλάσεων που εξαρτώνται από αυτό. Για αυτό τον λόγο καλό είναι να χρησιμοποιείται μία παράμετρος καθορισμού του αντικειμένου προς δημιουργία. Μία απλή λειτουργία “Make” με αυτό τον τρόπο θα μπορούσε να δημιουργεί διαφορετικού είδους αντικείμενο, αναλόγως με την παράμετρο όρισμα που θα δεχθεί.

UML Διάγραμμα:



Σχήμα 1: Διάγραμμα UML προτύπου Abstract

Σχετικά Πρότυπα:

Το «αφηρημένο εργοστάσιο» (“Abstract Factory”) συχνά υλοποιείται με την «Μέθοδο εργοστασίου» (“Factory Method”) αλλά και με το «Πρωτότυπο» (“Prototype”).

Πολύ συχνά, αν είναι ένα και μοναδικό αντίγραφο, ικανοποιεί και το πρότυπο «Μοναδικότητας» (“Singleton”) .

ΥΠΟΚΕΦΑΛΑΙΟ 2.2 Πρότυπο Δημιουργός, Creational Pattern – Builder

Το πρόβλημα:

Διαχωρισμός της δημιουργίας ενός πολύπλοκου αντικειμένου από την απεικόνιση του, ώστε η ίδια διαδικασία κατασκευής (αυτού του αντικειμένου) να παράγει διαφορετικές απεικονίσεις του.

Σκοπός:

Ένας αναγνώστης ενός μορφοποιημένου κειμένου RTF (Rich Text Format), θα πρέπει να έχει την δυνατότητα να μετατρέψει αυτό το ίδιο το κείμενο σε πολλές μορφές αναπαράστασης. Προφανώς η μετατροπή θα πρέπει να υποστηρίζει και μελλοντικά πρότυπα, άγνωστα ακόμη στον δημιουργό του αναγνώστη.

Μία λύση θα ήταν η δημιουργία μίας κλάσης αναγνώρισης κειμένου και ενός μετατροπέα κειμένου.

Η κλάση αναγνώστης κάθε φορά που αναγνωρίζει μία είσοδο ως κείμενο, παραδίδει αυτό το κομμάτι στον μετατροπέα ο οποίος φροντίζει να το αναπαραστήσει στο ζητούμενο φορματ.

Οι όποιες υποκλάσεις είναι αρμόδιες για τα επιμέρους προβλήματα και θέματα κάθε διαφορετικού φορμάτ, και φυσικά μπορεί να προστεθεί οικογένεια υποκλάσεων στο μέλλον που υποστηρίζει κάποιο καινούριο, άγνωστο σήμερα πρότυπο μορφοποίησης κειμένου.

Εφαρμογή:

Η χρήση του προτύπου ενδείκνυται όταν,

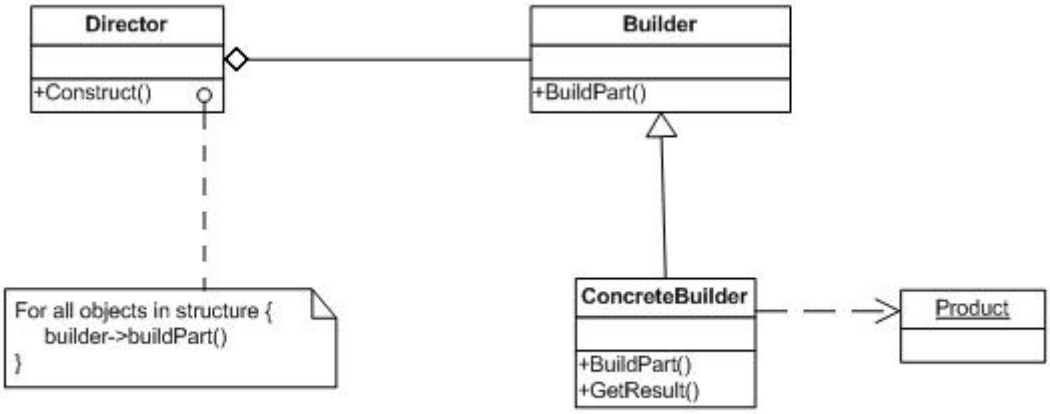
Ο αλγόριθμος δημιουργίας ενός πολύπλοκου αντικειμένου πρέπει να είναι ανεξάρτητος από τα τμήματα που αποτελούν αλλά και δημιουργούν το αντικείμενο.

Η διαδικασία δημιουργίας πρέπει να επιτρέπει διαφορετικές απεικονίσεις για το αντικείμενο υπό κατασκευή.

Επιπτώσεις:

1. Επιτρέπει μεγάλο βαθμό ελευθερίας στην εσωτερική απεικόνιση του προϊόντος. Επειδή παρέχει μία γενική και αφηρημένη διεπαφή, η αναπαράσταση και η εσωτερική δομή των αντικειμένων δεν εμφανίζονται προς τα έξω. Αποκρύπτει επίσης τον τρόπο κατασκευής των αντικειμένων.
2. Απομονώνει τον κώδικα αρμόδιο για δημιουργία και κατασκευή αντικειμένων. Έτσι βελτιώνει τον αρθρωτό προγραμματισμό αφού εξωτερικές κλάσεις δεν ενδιαφέρονται για την εσωτερική δομή των αντικειμένων.
3. Παρέχει καλύτερο έλεγχο πάνω στην διαδικασία κατασκευής αντικειμένων. Κατασκευάζει αντικείμενα βήμα-βήμα και μόνο κατά την ολοκλήρωση του το παραδίδει διαθέσιμο σε εξωτερικούς χρήστες (ή κλάσεις).

Διάγραμμα UML:



Σχήμα 2: Διάγραμμα UML Προτύπου Builder

Σχετικά Πρότυπα:

Το Αφηρημένο Εργοστάσιο μπορεί επίσης πολλές φορές να ασχολείται με την δημιουργία πολύπλοκων αντικειμένων, αν και το πρότυπο αυτό εστιάζει σε οικογένειες αντικειμένων και τα χαρακτηριστικά τους, ενώ το πρότυπο «Δημιουργός» παραδίδει έτοιμο, «δημιουργημένο» τελικό προϊόν.

Το πρότυπο «Δημιουργός» συχνά δημιουργεί ένα προϊόν που ανήκει στο πρότυπο «Σύνθεσης» (Composite)

ΥΠΟΚΕΦΑΛΑΙΟ 2.3 Πρότυπο Εργοστάσιο, Creational Pattern – Factory

Το πρόβλημα:

Ορισμός μίας διεπαφής που να δημιουργεί ένα αντικείμενο, αλλά να επιτρέπει στις υποκλάσεις να αποφασίσουν ποια κλάση να ενεργοποιήσουν. Το πρότυπο επιτρέπει στις υποκλάσεις να δημιουργούν στιγμιότυπα μίας κλάσης.

Γνωστό ως:

Ιδεατός Δημιουργός (Virtual Constructor)

Σκοπός

Τα διάφορα «πλαίσια» (Framework) χρησιμοποιούν τις ιδεατές κλάσεις για να αποκαθιστούν σχέσεις μεταξύ αντικειμένων. Πολλές φορές ένα τέτοιο πλαίσιο είναι υπεύθυνο και για την δημιουργία αυτών των αντικειμένων.

Έστω ένα τέτοιο πλαίσιο το οποίο εμφανίζει διαφόρων ειδών έγγραφα (documents) στον χρήστη. Προφανώς αυτό το πλαίσιο χρησιμοποιεί τις ιδεατές κλάσεις «Έγγραφο» και «Εφαρμογή», για τις οποίες ο κάθε προγραμματιστής υλοποιεί συγκεκριμένες υποκλάσεις που απευθύνονται σε συγκεκριμένο ζεύγος εφαρμογή-έγγραφο. Οι γενικές κλήσεις της αρχικής ιδεατής εφαρμογής, «άνοιγμα» και «δημιουργία» θα κληθούν με τα ορίσματα των υποκλάσεων κάθε φορά, υλοποιώντας ή ανοίγοντας το ιδιαίτερο τύπο εγγράφου για κάθε εφαρμογή για την οποία κλήθηκαν.

Το πρότυπο Εργοστάσιο προσφέρει μία εύκολη λύση σε αυτά τα θέματα, αφού ενθυλακώνει την γνώση του συγκεκριμένου εγγράφου προς δημιουργία στην υποκλάση και έτσι μεταφέρει την πληροφορία αυτή, έξω από το αρχικό πλαίσιο.

Εφαρμογές

Χρησιμοποιούμε το πρότυπο «Εργοστάσιο» όταν:

- Μία κλάση δεν μπορεί να προβλέπει την κλάση που θα καλέσει για τα αντικείμενα που πρέπει να δημιουργήσει
- Μία κλάση ζητά από τις υποκλάσεις της να ορίσουν τα αντικείμενα που αυτή δημιουργεί
- Κάποιες κλάσεις παραδίδουν τον έλεγχο σε μία από τις πολλές βοηθητικές κλάσεις και πρέπει να ανακοινωθεί στις υπόλοιπες, για το ποια κλάση παρέλαβε τον έλεγχο

Συνέργιες:

Το πρότυπο Δημιουργός (Creator) αναθέτει στις υποκλάσεις να ορίσουν την μέθοδο προτύπου Εργοστασίου (Factory) που θα επιστρέψει ένα στιγμιότυπο του αντικειμένου.

Συνέπειες:

Οι μέθοδοι που υλοποιούνται με το πρότυπο Εργοστασίου εξαλείφουν την ανάγκη σύνδεσης της εφαρμογής με συγκεκριμένες κλάσεις μέσα στον κώδικα. Ο κώδικας αλληλεπιδρά μόνο με την διεπαφή του προϊόντος.

Ένα πιθανό μειονέκτημα του προτύπου είναι ότι οι προγραμματιστές μπορεί να πρέπει να δημιουργήσουν υποκλάσεις του δημιουργού μόνο και μόνο για να δημιουργήσουν κάποιο συγκεκριμένο αντικείμενο.

Επιπλέον επιπτώσεις του προτύπου είναι οι εξής:

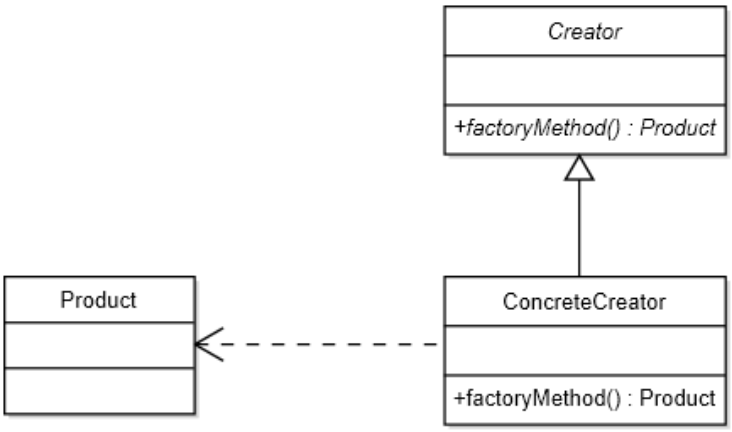
1. Παρέχει σημείο σύνδεσης για τις υποκλάσεις: η δημιουργία αντικειμένων μέσα σε μία κλάση είναι πάντα πιο εύκολη μετά την χρήση του προτύπου, αφού οι υποκλάσεις μπορούν εύκολα να δημιουργήσουν εκτεταμένες εκδοχές των αρχικών αντικειμένων.

Στο παράδειγμα με την εφαρμογή ανάγνωσης κειμένου, η κλάση «έγγραφο» μπορεί να περιγράψει μία μέθοδο “CreateFileDialog” η οποία προφανώς εκκινεί τον οδηγό δημιουργίας καινούριου εγγράφου. Η υποκλάση για κάθε συγκεκριμένη εφαρμογή μπορεί να υπερκαλύπτει (override) με μία πιο συγκεκριμένη μέθοδο την αρχική γενική μέθοδο.

2. Συνδέει ιεραρχίες παράλληλων κλάσεων: Οι κλάσεις του προτύπου μπορούν να κληθούν και από μία εφαρμογή-πελάτη.

   Σε αυτή την περίπτωση μία κλάση παραδίδει κάποιες αρμοδιότητες (ή ευθύνες) σε κάποια άλλη κλάση. Π.χ. όταν η κλάση ανοίγματος υπάρχοντος εγγράφου, παραδίδει τον έλεγχο στην αναζήτηση του Λειτουργικού Συστήματος.

Διάγραμμα UML:



Σχήμα 3: Διάγραμμα UML προτύπου Factory

Σχετικά Πρότυπα:

Συχνά το πρότυπο Αφηρημένο Εργοστάσιο υλοποιείται μέσω του προτύπου «Εργοστασίου».

Οι μέθοδοι που υλοποιούν το πρότυπο Εργοστάσιο, συχνά καλούνται από μεθόδους που υλοποιούν το πρότυπο «Προτύπου»

Οι μέθοδοι του προτύπου Εργοστάσιο δεν χρειάζεται να δημιουργήσουν έναν δημιουργό αντικειμένου μέσα στην κλάση του προϊόντος, όπως απαιτεί το πρότυπο «Πρωτοτύπου»

ΥΠΟΚΕΦΑΛΑΙΟ 2.4 Πρότυπο Πρωτότυπο, Creational Pattern – Prototype

Το πρόβλημα:

Περιγραφή του είδους των αντικειμένων προς δημιουργία με την χρήση ενός πρωτότυπου στιγμιότυπου και δημιουργία νέων αντικειμένων με την αντιγραφή αυτού του πρωτότυπου

Σκοπός:

Μπορούμε να δημιουργήσουμε ένα πρόγραμμα επεξεργασίας μουσικής, προσαρμόζοντας ένα γενικό πλαίσιο από προγράμματα γραφικής επεξεργασίας και προσθέτοντας νέα αντικείμενα που αναπαριστούν τις νότες, οκτάβες κτλ.

Προφανώς υπάρχει ήδη μία παλέτα εργαλείων στο πρόγραμμα για την προσθήκη νέων αντικειμένων, όπως και γενικότερα για την χρήση και επεξεργασία αντικειμένων. Το πλαίσιο για αυτό τον σκοπό παρέχει ιδεατές κλάσεις για την υλοποίηση των διάφορων εργαλείων της παλέτας. Το πλαίσιο επίσης καθορίζει μία υποκλάση γραφικών αντικειμένων την GraphicTool η οποία αναλαμβάνει να δημιουργεί στιγμιότυπα γραφικών αντικειμένων και να τα ενσωματώνει στο αρχικό έγγραφο.

Εδώ έγκειται το πρόβλημα. Η κλάση GraphicTool δεν γνωρίζει πώς να χειριστεί αντικείμενα μουσικής με νότες, οκτάβες κτλ.

Η δημιουργία μέσω της GraphicTool ενός κλώνου ή αντιγράφου του στιγμιότυπου του αντικειμένου που δημιουργεί αυτή η κλάση , λύνει αυτό το πρόβλημα. Αυτό το στιγμιότυπο ονομάζεται, «**πρωτότυπο**». Εάν όλες οι υποκλάσεις υποστηρίζουν την δημιουργία «κλώνου», τότε η κλάση GraphicTool μπορεί να «κλωνοποιήσει» κάθε είδους γραφικό αντικείμενο.

Χρήσεις

Η χρήση του προτύπου «Πρωτότυπο» ενδείκνυται όταν το σύστημα πρέπει να είναι ανεξάρτητο από τον τρόπο δημιουργίας και απεικόνισης των αντικειμένων, και όταν οι κλάσεις που πρέπει να ενεργοποιηθούν ορίζονται την στιγμή της εκτέλεσης (run-time), πχ με δυναμική σύνδεση, ή

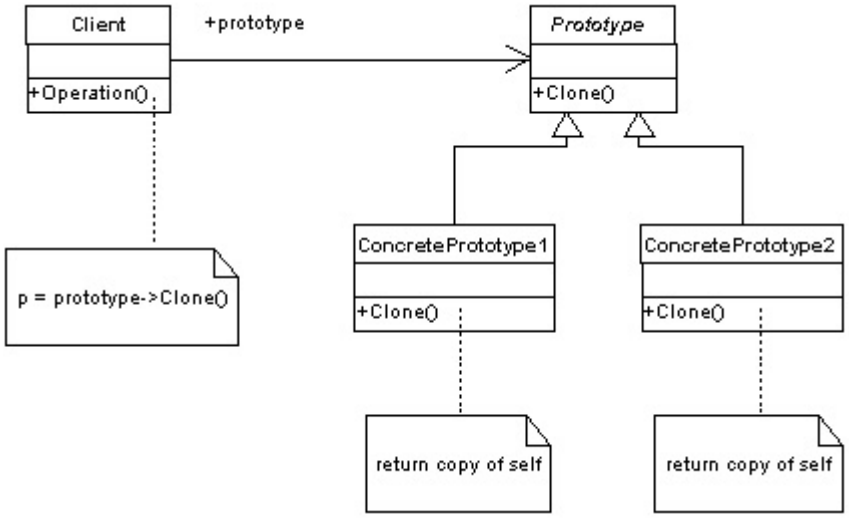
- Για να αποφευχθεί η οικοδόμηση μίας ιεραρχίας κλάσεων προτύπου εργοστασίου που γίνεται παράλληλα με την δημιουργία ιεραρχίας κλάσεων διαφορετικών προϊόντων, ή
- Όταν τα στιγμιότυπα μία κλάση μπορούν να βρίσκονται σε μία μόνο, από αρκετές διαθέσιμες καταστάσεις, ή
- Όταν τα στιγμιότυπα μία κλάσης μπορούν να βρίσκονται σε μία μόνο από αρκετές διαθέσιμες καταστάσεις. Σε αυτή την περίπτωση ίσως να χρειάζεται να εγκατασταθούν πολλαπλοί «κλώνοι» της κλάσης, ο καθένας σε διαφορετική κατάσταση.

Σχετικά Πρότυπα

Το πρότυπο «Πρωτότυπο» και το «Αφηρημένο Εργοστάσιο» είναι μερικές φορές ανταγωνιστικά, ενώ κάποιες άλλες φορές μπορούν να χρησιμοποιηθούν συμπληρωματικά.

Σχεδιασμοί λογισμικού που κάνουν χρήση του προτύπου «Σύνθεσης» (Composite) και του προτύπου «Διακόσμησης» (Decorator) μπορούν να επωφεληθούν και από το πρότυπο «Πρωτότυπο».

Παράδειγμα UML του προτύπου



Σχήμα 4: UML Διάγραμμα Προτύπου Πρωτότυπο

ΥΠΟΚΕΦΑΛΑΙΟ 2.5 Πρότυπο Μοναδικότητας, Creational Pattern – Singleton

Το πρόβλημα:

Η διασφάλιση ότι μία κλάση έχει μόνο ένα στιγμιότυπο και παροχή ενός καθολικού σημείου πρόσβασης σε αυτήν.

Σκοπός:

Για αρκετές κλάσεις είναι σημαντικό να έχουν μόνο ένα (ενεργό) στιγμιότυπο. Π.χ. αν και μπορεί να υπάρχουν πολλοί εκτυπωτές σε ένα σύστημα, μόνο μία μνήμη ουράς εκτύπωσης (Printer spooler) υπάρχει. Υπάρχει μόνο ένα Σύστημα Αρχείων (File System) και μόνο ένας Window Manager.

Ο τρόπος με τον οποίο διασφαλίζεται η μοναδικότητα των στιγμιότυπων τέτοιων κλάσεων μαζί με την εύκολη πρόσβαση σε αυτό, διασφαλίζεται με την ανάθεση στην ίδια την κλάση της παρακολούθησης του πλήθους των στιγμιότυπων της.

Χρήσεις:

Ακολουθούν κάποιες περιπτώσεις χαρακτηριστικής χρήσης του προτύπου:

- Διασφάλιση ενός μοναδικού στιγμιότυπου: Το πρότυπο Μοναδικότητας αντιμετωπίζει το μοναδικό στιγμιότυπο, ως κανονικό στιγμιότυπο της κλάσης του, αλλά η κλάση αυτή είναι γραμμένη έτσι ώστε να μπορεί να δημιουργήσει μόνο ένα στιγμιότυπο. Ένα συνηθισμένος τρόπος επίτευξης αυτού, είναι η απόκρυψη του στιγμιότυπου πίσω από μια λειτουργία της κλάσης(είτε δηλαδή ως στατικό μέλος (Static member function) ή ως δημιουργό της κλάσης). Μέσω μίας μεταβλητής που αρχικοποιείται πριν την δημιουργία της κλάσης και επιστρέφει την τιμής της, εξασφαλίζεται η μοναδικότητα του προτύπου.
- Υποκλάσεις της αρχικής μοναδιαίας κλάσης: το κυριότερο θέμα εδώ είναι η εγκατάσταση του (μοναδικού) στιγμιότυπου της υποκλάσης, ώστε οι

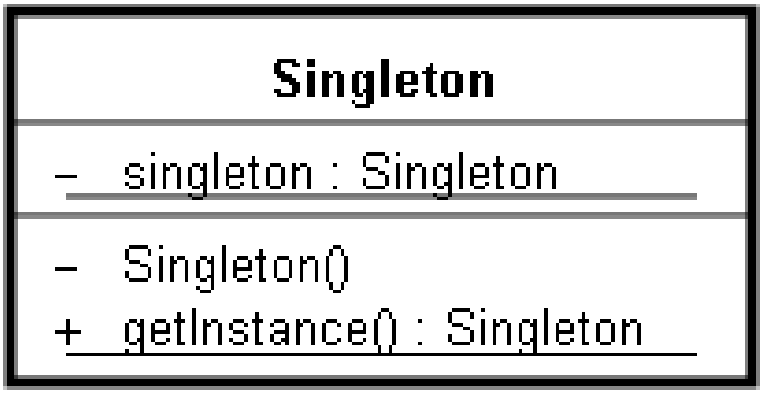
εφαρμογές πελάτες να έχουν πρόσβαση μόνο σε αυτό. Αυτό γίνεται μέσω της καθολικότητας της μεταβλητής που ελέγχει το πλήθος της αρχικής μοναδιαίας κλάσης.

Εφαρμογές

Η χρήση του προτύπου Μοναδικότητας επιβάλλεται όταν:

- Πρέπει να υπάρχει ακριβώς ένα στιγμιότυπο μίας κλάσης και πρέπει να είναι προσβάσιμο σε όλες τις εξωτερικές εφαρμογές από ένα γνωστό και καλά καθορισμένο σημείο εισόδου.
- Όταν το μοναδικό στιγμιότυπο πρέπει να είναι προσβάσιμο μέσω υποκλάσεων, και οι εφαρμογές πελάτες πρέπει να μπορούν να χρησιμοποιήσουν ένα εκτεταμένο στιγμιότυπο χωρίς να τροποποιηθεί ο κώδικας τους,

Διάγραμμα UML:



Σχήμα 5: Διάγραμμα UML προτύπου Singleton

Σχετικά Πρότυπα:

Πολλά πρότυπα μπορούν να υλοποιηθούν σύμφωνα με το πρότυπο Μοναδικότητας, όπως το Νοητό Εργοστάσιο (abstract Factory),το πρότυπο Δημιουργού (builder) και το Πρωτότυπο (prototype).

ΚΕΦΑΛΑΙΟ 3

<ΔΟΜΙΚΑ ΠΡΟΤΥΠΑ ΣΧΕΔΙΑΣΗΣ STRUCTURAL PATTERNS>

ΕΙΣΑΓΩΓΗ

Τα δομικά πρότυπα ασχολούνται με το πώς οι κλάσεις και να αντικείμενα συντίθενται μεταξύ τους ώστε να δημιουργήσουν μεγαλύτερες και πολυπλοκότερες δομές. Οι κλάσεις του προτύπου χρησιμοποιούν την κληρονομικότητα (inheritance) για να δημιουργήσουν διεπαφές ή αντικείμενα.

Αν σκεφτούμε τον τρόπο με τον οποίο η πολλαπλή κληρονομικότητα ενώνει δύο κλάσεις σε μία, το αποτέλεσμα είναι μία κλάση που συνδυάζει τις ιδιότητες των κληρονομούμενων κλάσεων. Αυτή η δυνατότητα είναι χρήσιμη στην περίπτωση συνεργασίας ανεξάρτητων μεταξύ τους βιβλιοθηκών (library classes).

Αντί να δημιουργούν επαφές και αντικείμενα, τα αντικείμενα που δημιουργούν τα δομικά πρότυπα περιγράφουν τρόπους δημιουργίας αντικειμένων ώστε να εμπεριέχουν καινούριες δυνατότητες. Αυτές οι δυνατότητες υπάρχουν ακριβώς επειδή τα περιεχόμενα του αντικειμένου μπορούν να αλλάξουν κατά την εκτέλεση (run-time), το οποίο είναι προφανώς αδύνατον σε υλοποιήσεις στατικών (Static) αντικειμένων.

Τα περισσότερα από τα δομικά πρότυπα συνδέονται μεταξύ τους σε κάποιο βαθμό.

ΥΠΟΚΕΦΑΛΑΙΟ 3.1 Πρότυπο Προσαρμογής,Structural Pattern – Adapter

Το πρόβλημα:

Μετατροπή της διεπαφής μίας κλάσης σε μία άλλη διεπαφή κάποιας εφαρμογής πελάτη. Το πρότυπο επιτρέπει σε κλάσεις με ασύμβατες διεπαφές να συνεργαστούν μεταξύ τους.

Επίσης γνωστό ως:

Wrapper

Σκοπός:

Μερικές φορές μία κλάση-εργαλείο η οποία έχει σχεδιαστεί για επαναχρησιμοποίηση, δεν γίνεται να επαναχρησιμοποιηθεί μόνο και μόνο γιατί η διεπαφή της δεν είναι συμβατή με την συγκεκριμένη διεπαφή που απαιτεί (περιμένει) μία συγκεκριμένη εφαρμογή.

Έστω ένα πρόγραμμα ζωγραφικής που επιτρέπει την δημιουργία γραφικών αντικειμένων (γραμμές, πολύγωνα, κείμενο κτλ) και δημιουργεί εικόνες και διαγράμματα από αυτά. Το κλειδί στην αφηρημένη κλάση ενός τέτοιου προγράμματος έγκειται στο γραφικό αντικείμενο το οποίο έχει μία μορφή που επιδέχεται τροποποίησης (editing) και μπορεί να ζωγραφίσει και τον εαυτό της. Η διεπαφή για ένα τέτοιο αντικείμενο υλοποιείται σε μία ιδεατή κλάση, την Shape. Το πρόγραμμα ορίζει μία υποκλάση για κάθε είδος γραφικού αντικειμένου. Πχ το LineShape για τις γραμμές, το PolygonShape για τα πολύγωνα κτλ.

Το πρόγραμμα TextView που εμφανίζει κείμενο, δεν μπορεί προφανώς να συνεργαστεί με το ασύμβατο πρόγραμμα Shape αφού διαχειρίζονται διαφορετικά αντικείμενα. Η πρόταση αλλαγής της διεπαφής του TextView δεν είναι δυνατή να υλοποιηθεί, αφού και αυτό ορίζει τις όποιες ιδεατές κλάσεις για το γενικό

αντικείμενο κειμένου text, τις οποίες εξειδικεύει στο εσωτερικό του προγράμματος μέσω υποκλάσεων.

Μπορούμε να δημιουργήσουμε μία κλάση TextShape η οποία προσαρμόζεται από την μία μεριά με την διεπαφή του TextView και από την άλλη στην διεπαφή του Shape με δύο τρόπους:

Α. Κληρονομώντας την διεπαφή του Shape και υλοποιώντας μία κλάση του TextView

Β. Δημιουργώντας ένα στιγμιότυπο του TextView μέσα σε ένα αντικείμενο TextShape και υλοποιώντας το TextShape σύμφωνα με την διεπαφή του TextView.

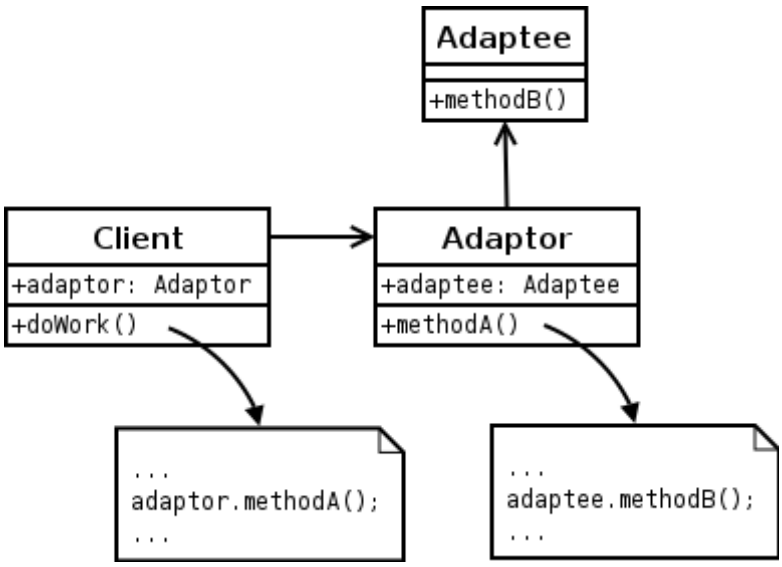
Και στις δύο περιπτώσεις θα έχει δημιουργηθεί ένα ενδιάμεσο αντικείμενο, ένας προσαρμοστής (Adapter).

Εφαρμογές

Επιβάλλεται η χρήση του προτύπου όταν:

- Θέλουμε να χρησιμοποιήσουμε μία υπάρχουσα κλάση και η διεπαφή της δεν συμπίπτει με αυτή που χρειαζόμαστε.
- Θέλουμε να χρησιμοποιήσουμε μία επαναχρησιμοποιούμενη κλάση η οποία συνεργάζεται με άσχετες ή μη αναμενόμενες κλάσεις, κλάσεις δηλαδή που δεν διαθέτουν συμβατές διεπαφές.
- Θέλουμε να χρησιμοποιήσουμε πολλαπλές υπάρχουσες υποκλάσεις αλλά δεν είναι πρακτικό να τροποποιήσουμε όλες τις διεπαφές τους με το να δημιουργήσουμε καινούριες υποκλάσεις για το καθένα. Ένα αντικείμενο που θα είναι συμβατό και με το πρότυπο Αντικειμένου (Object) μπορεί να προσαρμόσει μία τέτοια διεπαφή στην μητρική κλάση.

Διάγραμμα UML:



Σχήμα 6: Διάγραμμα UML Προτύπου (Object) Adapter

Σχετικά Πρότυπα:

Το πρότυπο Γέφυρα (Bridge) έχει παραπλήσια δομή με το παρόν πρότυπο αλλά διαφορετικό σκοπό: χρησιμοποιείται για να διαχωρίζει μία διεπαφή από την υλοποίηση της, έτσι ώστε το καθένα να μπορεί να επεξεργαστεί ανεξάρτητα από το άλλο. Το παρόν πρότυπο χρησιμοποιείται για να διαφοροποιεί την διεπαφή από ένα υπάρχον αντικείμενο.

Το πρότυπο Διακοσμητής (Decorator) βελτιώνει ένα αντικείμενο χωρίς να παρεμβαίνει στην διεπαφή του. Άρα είναι και περισσότερο εύκολο στην σύνδεση από το παρόν πρότυπο. Επίσης υποστηρίζει και αναδρομική σύνθεση, κάτι που δεν γίνεται με το πρότυπο Προσαρμογής.

Το πρότυπο Πληρεξούσιος (Proxy) ορίζει έναν αντιπρόσωπο (αναπληρωτή) για κάποιο αντικείμενο και δεν αλλάζει την διεπαφή του.

ΥΠΟΚΕΦΑΛΑΙΟ 3.2 Πρότυπο Γέφυρας,Structural Pattern – Bridge

Το πρόβλημα:

Αποσύνδεση μίας ιδεατής κλάσης από την εφαρμογή της (implementation) έτσι ώστε τα δύο τμήματα να μπορούν να διαφοροποιηθούν ανεξάρτητα από το άλλο.

Επίσης γνωστό ως:

Χειριστής/Κορμός (Handle/Body)

Σκοπός:

Όταν ένα ιδεατό αντικείμενο (abstraction) μπορεί να έχει πέραν της μίας πιθανής υλοποίησης, η συνηθισμένη διαδικασία που ακολουθείται για να συνδεθούν με το ιδεατό αντικείμενο είναι μέσω της κληρονομικότητας (inheritance). Μία ιδεατή κλάση αναθέτει την διεπαφή στο ιδεατό αντικείμενο και οι υποκλάσεις αναλαμβάνουν να το υλοποιήσουν με τους διαφορετικούς τρόπους, αν και αυτή η προσέγγιση δεν είναι πάντα το ίδιο ευέλικτη αφού «δένει» την κάθε υλοποίηση με το ιδεατό αντικείμενο, κάνοντας έτσι αδύνατες τις αλλαγές σε ένα από τα δύο.

Εφαρμογή:

Χρησιμοποιούμε το πρότυπο Γέφυρας, όταν:

- Θέλουμε να αποφύγουμε έναν μόνιμο δεσμό μεταξύ ενός ιδεατού αντικειμένου και της υλοποίησης του. Όταν για παράδειγμα η υλοποίηση πρέπει να επιλεγεί την στιγμή της εκτέλεσης (run-time).

- Και το ιδεατό αντικείμενο και η υλοποίηση του πρέπει να μπορούν να επεκταθούν μέσω υποκλάσεων. Σε αυτή την περίπτωση, το πρότυπο Γέφυρας επιτρέπει να συνδυαστούν τα διάφορα ιδεατά αντικείμενα και οι διεπαφές τους και να επεκταθούν ανεξάρτητα.
- Οι αλλαγές σε κάποια διεπαφή δεν πρέπει να έχουν αντίκτυπο στις εφαρμογές πελάτες.
- Στην C++ αν θέλουμε να αποκρύψουμε πλήρως την υλοποίηση ενός ιδεατού αντικειμένου από τις εφαρμογές πελάτες, γιατί στην C++ η αναπαράσταση της κλάσης είναι ορατή στην διεπαφή της.
- Αν υπάρχει μία τέτοια διάδοση (πολλαπλασιασμός) κλάσεων, ώστε να παρίσταται η ανάγκη για τον διαχωρισμό τους στα δύο.
- Θέλουμε να δώσουμε πρόσβαση σε μία διεπαφή σε πολλαπλά αντικείμενα και αυτό πρέπει να κρύβεται από τις εφαρμογές πελάτες.

Επιπτώσεις:

Το πρότυπο Γέφυρας έχει τις εξής βασικές επιπτώσεις:

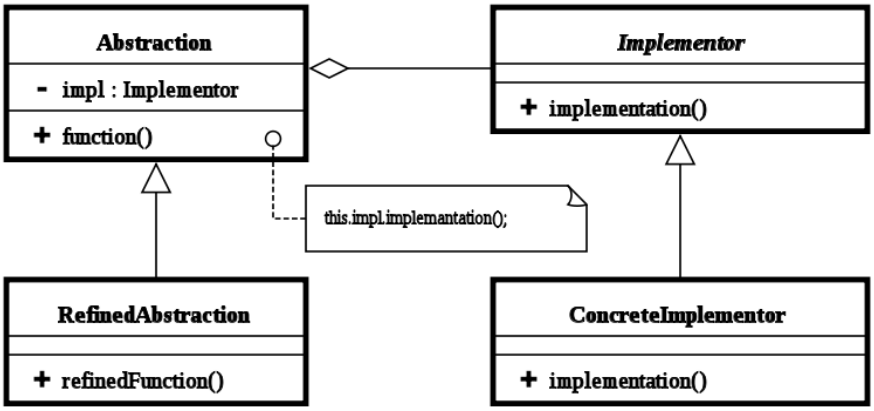
1. Αποσυνδέει την διεπαφή με την υλοποίηση: Η υλοποίηση δεν είναι πλέον στατικά δεμένη με την διεπαφή. Έτσι μπορεί να παραμετροποιηθεί την στιγμή της εκτέλεσης (run-time). Είναι δυνατόν έτσι να αλλάξει ακόμη και η υλοποίηση ενός αντικειμένου την στιγμή της εκτέλεσης.

   Η αποσύνδεση του ιδεατού αντικειμένου από την υλοποίηση επίσης εξαλείφει την απαίτηση διασυνδέσεων την στιγμή της σύνταξης (compile) του κώδικα. Η αλλαγή μίας κλάσης υλοποίησης δεν απαιτεί ανασύνταξη (recompile) της ιδεατής κλάσης και των εφαρμογών πελατών.

   Η αποσύνδεση αυτή οδηγεί και σε ένα καλύτερα δομημένο σύστημα. Το υψηλού επιπέδου τμήμα του συστήματος γνωρίζει μόνο την ιδεατή κλάση και την υλοποίηση της.

2. Αυξημένη δυνατότητα επέκτασης, αφού το ιδεατό αντικείμενο και η υλοποίηση του μπορούν να τροποποιηθούν ανεξάρτητα
3. Απόκρυψη των λεπτομερειών υλοποίησης από τις εφαρμογές πελάτες, όπως τα διάφορα αντικείμενα που διαμοιράζουν οι εσωτερικές κλάσεις.

Διάγραμμα UML:



Σχήμα 7: Διάγραμμα UML Προτύπου Bridge

Σχετικά Πρότυπα:

Μία κλάση Νοητού Εργοστασίου μπορεί να δημιουργήσει και να παραμετροποιήσει μια κλάση Γέφυρας.

Το πρότυπο Προσαρμογέας είναι στραμμένο προς το να βοηθάει μη σχετιζόμενες κλάσεις να συνεργαστούν. Συνήθως εφαρμόζεται σε συστήματα αφού αυτά έχουν σχεδιαστεί. Το πρότυπο Γέφυρας από την άλλη μεριά χρησιμοποιείται από την αρχή σε έναν σχεδιασμό όπου επιτρέπεται τα ιδεατά αντικείμενα και οι υλοποιήσεις τους να διαφοροποιούνται ανεξάρτητα το ένα από το άλλο.

ΥΠΟΚΕΦΑΛΑΙΟ 3.3 Πρότυπο Σύνθεσης,Structural Pattern – Composite

Το πρόβλημα:

Η σύνθεση αντικειμένων σε μία δενδροειδή δομή για την αναπαράσταση τμημάτων της ιεραρχικής αναπαράστασης των κλάσεων. Το πρότυπο Σύνθεσης θα πρέπει να επιτρέπει στις εφαρμογές πελάτες να διαχειρίζονται ανεξάρτητα αντικείμενα και ομάδες αντικειμένων με τον ίδιο τρόπο.

Σκοπός:

Οι γραφικές εφαρμογές όπως αυτές που δημιουργούν σχήματα, επιτρέπουν στον χρήστη να δημιουργήσει πολύπλοκα αντικείμενα (διαγράμματα) με την χρήση απλών πρωτογενών στοιχείων (γραμμές, κύκλους, κτλ).ο χρήστης μπορεί να ομαδοποιήσει τέτοια στοιχεία για να δημιουργήσει πολύπλοκες δομές, οι οποίες με την σειρά τους μπορούν να δημιουργήσουν ακόμη πολυπλοκότερες δομές. Μία απλή υλοποίηση θα μπορούσε να ορίσει κλάσεις για τα βασικά στοιχεία όπως κείμενο (Text) και γραμμή (lines) και άλλες κλάσεις-υποδοχείς αυτών των στοιχείων.

Αυτή η προσέγγιση εμπεριέχει ένα πρόβλημα: ο κώδικας που χρησιμοποιεί τις παραπάνω κλάσεις θα πρέπει να τις χειρίζεται με διαφορετικό τρόπο (τις κλάσεις των βασικών στοιχείων με τις κλάσεις-υποδοχείς αυτών), αν και τις περισσότερες φορές ο χρήστης τις χρησιμοποιεί και τις αντιμετωπίζει με τον ίδιο τρόπο. Η εφαρμογή γίνεται πολύπλοκη επειδή πρέπει να διαχωρίζει αυτές τις δύο κατηγορίες.

Το πρότυπο Σύνθεσης λύνει το παραπάνω πρόβλημα. Χρησιμοποιώντας αναδρομική σύνθεση των αντικειμένων δεν χρειάζεται πλέον να υφίσταται αυτός ο διαχωρισμός. Αυτό γίνεται με την δημιουργία μίας ιδεατής κλάσης η οποία αντιπροσωπεύει και τα βασικά στοιχεία και τους υποδοχείς τους. Στο παραπάνω παράδειγμα, αυτή είναι η κλάση Graphic η οποία υποστηρίζει την λειτουργία Draw,

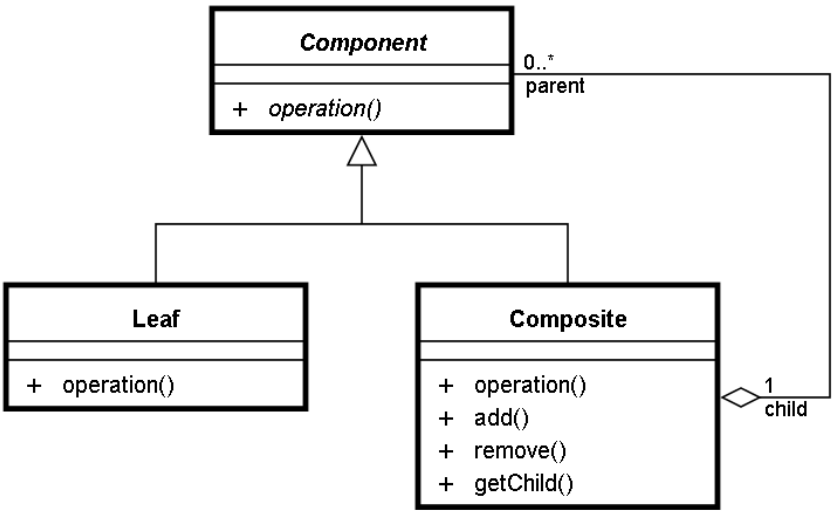
και έχει υποκλάσεις για το κάθε βασικό στοιχείο (Line, Circle), οι οποίες υλοποιούν διαφορετικά (override) την Draw για τα αντικείμενα τους.

Εφαρμογές:

Η χρήση του προτύπου Σύνθεσης γίνεται όταν:

- Θέλουμε να απεικονίσουμε τμήματα της ιεραρχίας αντικειμένων
- Θέλουμε να μπορούν οι εφαρμογές πελάτες να αγνοούν τις διαφορές μεταξύ συνθέσεων αντικειμένων και των αντικειμένων αυτών. Οι εφαρμογές πελάτες θα χειρίζονται όλα τα αντικείμενα της δομής με τον ίδιο τρόπο.

Διάγραμμα UML:



Σχήμα 8: Διάγραμμα UML προτύπου Composite

Σχετικά Πρότυπα:

- Συχνά η σύνδεση μεταξύ αντικειμένων και της σύνθεσης τους χρησιμοποιείται και στο πρότυπο Αρμοδιότητας.
- Το πρότυπο Διακοσμητής χρησιμοποιείται συχνά με το πρότυπο Σύνθεσης. Όταν χρησιμοποιούνται μαζί τότε συνήθως έχουν μία κοινή μητρική κλάση. Τότε η κλάση του προτύπου Διακόσμησης πρέπει να υποστηρίζει την

διεπαφή του προτύπου Σύνθεσης μέσω λειτουργιών, πρόσθεσης (Add), αφαίρεσης (remove), και GetChild.

- Το πρότυπο Επανάληψης μπορεί να χρησιμοποιηθεί για να διατρέξει τα αντικείμενα Σύνθεσης
- Το πρότυπο Επισκέπτης περιχαρακώνει λειτουργίες και συμπεριφορές οι οποίες διαφορετικά θα εξαπλωθούν σε όλες της κλάσεις των προτύπων Σύνθεσης και Φύλλου.

ΥΠΟΚΕΦΑΛΑΙΟ 3.4 Πρότυπο Διακοσμητής,Structural Pattern – Decorator

Το πρόβλημα:

Επισύναψη επιπλέον αρμοδιοτήτων σε ένα αντικείμενο με δυναμικό τρόπο. Οι κλάσεις του προτύπου Διακοσμητής παρέχουν ένα ευέλικτο εναλλακτικό τρόπο δημιουργίας (ή επέκτασης) υποκλάσεων ώστε να επεκταθούν και οι δυνατότητες.

Σκοπός:

Μερικές φορές θέλουμε να αναθέσουμε αρμοδιότητες σε συγκεκριμένα αντικείμενα, και όχι σε όλη την κλάση τους.

Ένας τρόπος για να συμβεί αυτό είναι μέσω της κληρονομικότητας. Η «κληρονομιά» ενός γραφικού για ένα κουμπί από μία άλλη κλάση, θα βάλει αυτό το γραφικό σε όλα τα αντίστοιχα κουμπιά, το οποίο είναι προφανώς προβληματική λειτουργία αφού η επιλογή έγινε στατικά.

Μία πιο ευέλικτη προσέγγιση θα ήταν να ενσωματωθεί το γραφικό σε ένα άλλο αντικείμενο το οποίο κάνει αυτή την δουλειά. Το αντικείμενο αυτό που περιέχει το γραφικό, ονομάζεται ο Διακοσμητής. Αυτός αναλαμβάνει να πάρει το αρχικό γραφικό και να το επισυνάψει εκεί που θα του οριστεί, έστω και αναδρομικά, έτσι ώστε να μπορούν να χρησιμοποιηθούν πολλοί Διακοσμητές ταυτόχρονα για πολλαπλές λειτουργίες και σε διαφορετικό ανεξάρτητο επίπεδο από τα υπόλοιπα αντικείμενα.

Εφαρμογές:

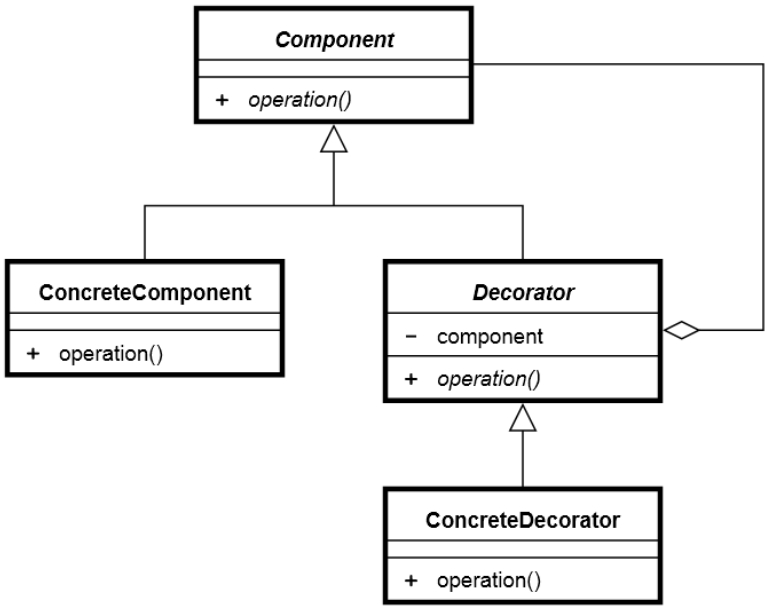
- Για να προσθέσουμε αρμοδιότητες σε μεμονωμένα αντικείμενα δυναμικά και χωρίς να επηρεάζονται άλλα αντικείμενα
- Για την ανάθεση αρμοδιοτήτων που μπορεί να ανακληθούν

- Όταν η επέκταση μέσω υποκλάσεων δεν είναι πρακτική. Μερικές φορές ένας μεγάλος αριθμός από παραλλαγές είναι δυνατός, έτσι ώστε να παράγουν μεγάλο πλήθος υποκλάσεων για να υποστηρίξουμε κάθε δυνατή παραλλαγή. Μπορεί επίσης η δήλωση της κλάσης να είναι κρυφή ή μη διαθέσιμη για δημιουργία υποκλάσεων.

Συνέπειες:

1. Μεγαλύτερη ευελιξία από την στατική κληρονομικότητα: Παρέχει μεγαλύτερη ευελιξία ανάθεσης αρμοδιοτήτων στα αντικείμενα επειδή οι αρμοδιότητες μπορούν να αποδοθούν και ανακληθούν άμεσα την στιγμή της εκτέλεσης (run-time).
   Επίσης είναι εύκολο να ανατεθεί μία ιδιότητα δύο φορές. Για παράδειγμα η απόδοση σε ένα αντικείμενο TextView διπλού περιγράμματος, απλώς αναθέτουμε την ιδιότητα border δύο φορές.
2. Αποτρέπει την συνεχή ανάθεση αρμοδιοτήτων (και πολυπλοκότητας) σε ήδη βεβαρυμμένες κλάσεις. Επειδή μεγαλώνει μόνο στιγμιαία τα αντικείμενα και τις κλάσεις δεν επιφέρει μόνιμο πρόβλημα μεγέθυνσης σε αυτά. Αντί να προσπαθούμε να «στριμώξουμε» σε μία κλάση όλες τις πιθανές δυνατότητες, αυτές ανατίθενται σε μία κλάση Διακοσμητή.
3. Ο Διακοσμητής και το περιεχόμενο του δεν είναι ταυτόσημα. Για αυτό το λόγο θα πρέπει να δίνεται ανάλογη προσοχή στην διάκριση μεταξύ της κλήσης του Διακοσμητή και της χρήσης του αντικειμένου που περιέχει.
4. Δημιουργία πολλών μικρών αντικειμένων: Σε ένα σύστημα που χρησιμοποιείται ο Διακοσμητής πολλές φορές καταλήγουμε με πολλά μικρά ταυτόσημα αντικείμενα. Αν και το σύστημα φαίνεται σχετικά εύκολο στον αρχικό δημιουργό του, είναι αρκετά δύσκολο στην κατανόηση και στην αποσφαλμάτωση.

UML Διάγραμμα



Σχετικά Πρότυπα:

- Πρότυπο Προσαρμογέας: Ένας Διακοσμητής διαφέρει από τον Προσαρμογέα μόνο στο ότι ο Διακοσμητής αλλάζεις τις αρμοδιότητες του αντικειμένου και όχι την διεπαφή του. Ο Προσαρμογέας δίνει στο αντικείμενο μία τελείως νέα διεπαφή.
- Πρότυπο Σύνθεσης: Ο Διακοσμητής μπορεί να ειδωθεί ως μία εκφυλισμένη κλάση προτύπου Σύνθεσης με μόνο ένα συστατικό στοιχείο. Αν και ο Διακοσμητής προσθέτει επιπλέον αρμοδιότητες και δεν χρησιμοποιείται για συσσώρευση αντικειμένων.
- Πρότυπο Στρατηγικής: Ένας Διακοσμητής επιτρέπει την αλλαγή της εμφάνισης ενός αντικειμένου. Ένα αντικείμενο Στρατηγικής επιτρέπει την αλλαγή του εσωτερικού. Αποτελούν και τα δύο εναλλακτικούς τρόπους αλλαγής ενός αντικειμένου.

ΥΠΟΚΕΦΑΛΑΙΟ 3.5 Πρότυπο Πρόσοψης,Structural Pattern – Façade

Το πρόβλημα:

Η παροχή μίας ενιαίας διεπαφής σε μία ομάδα διεπαφών ενός υποσυστήματος. Το πρότυπο Πρόσοψης παρέχει μία διεπαφή υψηλότερου επιπέδου ώστε το υποσύστημα να είναι ευκολότερο να χρησιμοποιηθεί

Σκοπός

Η ανάλυση ενός συστήματος σε υποσυστήματα είναι προφανές ότι κατεβάζει την πολυπλοκότητα του. ένα κοινός σχεδιαστικός στόχος είναι η ελαχιστοποίηση της επικοινωνίας και των εξαρτήσεων μεταξύ υποσυστημάτων. Ένας τρόπος να επιτευχθεί αυτό, είναι με την εισαγωγή μίας **Πρόσοψης** η οποία παρέχει μία μοναδική απλοποιημένη διεπαφή με τις γενικές λειτουργίες του υποσυστήματος.

Χρήσεις:

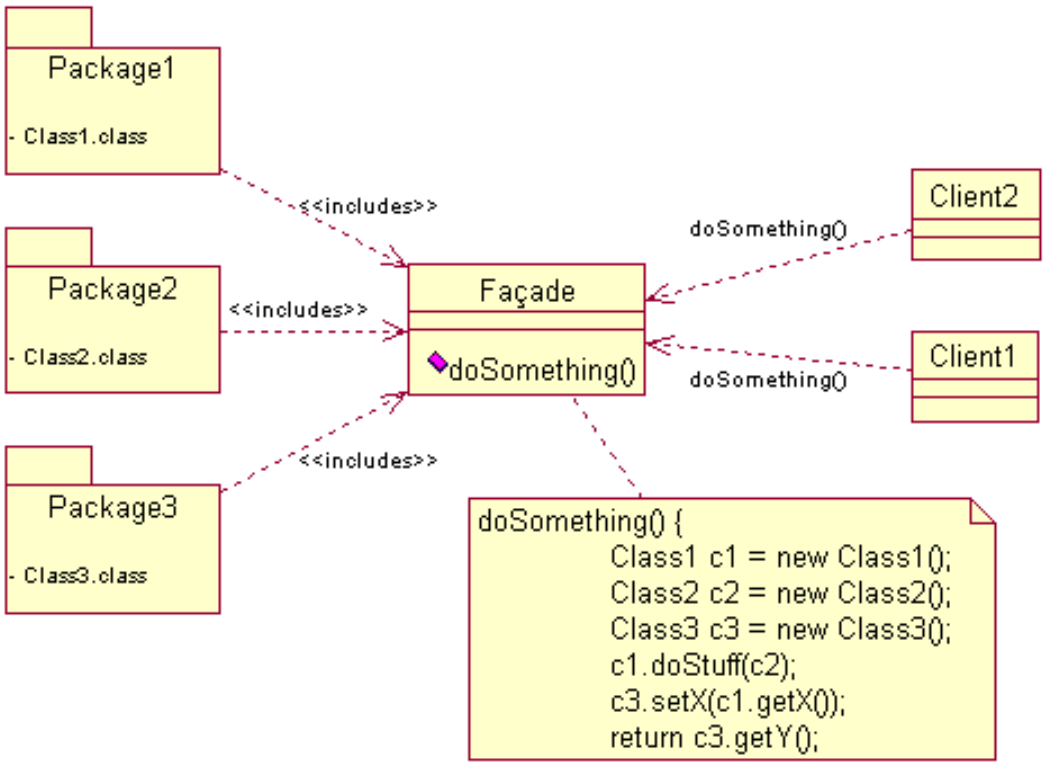
Χρησιμοποιούμε το πρότυπο Πρόσοψης όταν:

- Θέλουμε να παρέχουμε μία μοναδική διεπαφή σε ένα πολύπλοκο υποσύστημα. Τα υποσυστήματα πολλές φορές γίνονται όλο και πιο πολύπλοκα, όσο εξελίσσονται. Τα περισσότερα πρότυπα όταν εφαρμόζονται, οδηγούν σε όλο και μικρότερες κλάσεις. Έτσι τα όποια υποσυστήματα γίνονται όλο και πιο επαναχρησιμοποιήσιμα και ευκολότερα στην παραμετροποίηση, αλλά γίνεται όλο και πιο δύσκολο να χρησιμοποιηθούν από εφαρμογές πελάτες χωρίς να παραμετροποιηθούν ανάλογα. Μία Πρόσοψη μπορεί να παρέχει μία απλή μοναδική όψη του υποσυστήματος η οποία είναι αρκετά καλή για τις περισσότερες εφαρμογές πελάτες. Μόνο αυτές οι εφαρμογές που θα χρειαστούν πιο λεπτομερή

παραμετροποιήση και αλληλεπίδραση θα χρειαστεί να «δουν» πιο πίσω από την Πρόσοψη.

- Υπάρχουν πολλές εξαρτήσεις μεταξύ εφαρμογών πελατών και των κλάσεων υλοποιήσης (των αρχικών ιδεατών κλάσεων). Η εισαγωγή μίας Πρόσοψης αποσυμπλέκει το σύστημα από τους πελάτες και τα άλλα υποσυστήματα και έτσι προάγει την ανεξαρτησία του υποσυστήματος και την μεταφερσιμότητα.
- Θέλουμε να δημιουργήσουμε επίπεδα σε ένα υποσύστημα. Χρησιοποιούμε μία Πρόσοψη για να ορίσουμε ένα σημείο εισόδου σε κάθε επίπεδο του υποσυστήματος. Εάν τα υποσυστήματα είναι εξαρτώμενα τότε μπορούμε να απλοποιήσουμε τις εξαρτήσεις μεταξύ τους διατηρώντας την επικοινωνία μεταξύ τους μόνο μέσω της Πρόσοψης.

Διάγραμμα UML:



Σχήμα 9: Διάγραμμα UML προτύπου Facade

Σχετικά Πρότυπα:

Το Νοητό Εργοστάσιο μπορεί να χρησιμοποιηθεί μαζί με την Πρόσοψη για να παρέχουν μία διεπαφή για να δημιουργήσουν ένα υποσύστημα αντικειμένων με έναν ανεξάρτητο των υπολοίπων υποσυστημάτων τρόπο. Το Νοητό Εργοστάσιο μπορεί επίσης να χρησιμοποιηθεί ως εναλλακτικό της Πρόσοψης για να κρύψει κλάσεις που είναι σχετικές με συγκεκριμένη πλατφόρμα.

Το πρότυπο Μεσολαβητής (Mediator) είναι παρόμοιο με την Πρόσοψη στον τρόπο που αντιμετωπίζει ιδεατά τις λειτουργικότητες υπαρχόντων κλάσεων. Παρόλα αυτά, ο σκοπός του Μεσολαβητή είναι να περιορίσει τις αυθαίρετες επικοινωνίες μεταξύ συνεργαζόμενων αντικειμένων συχνά με το να κάνει αυτή την επικοινωνία κεντρικά. Τα συνεργαζόμενα με τον Μεσολαβητή αντικείμενα, έχουν γνώση της ύπαρξης του και ξέρουν ότι επικοινωνούν με αυτόν και όχι απευθείας με τα άλλα αντικείμενα. Αντίθετα μία Πρόσοψη κυρίως δημιουργεί ένα νέο

αφαιρετικό τρόπο σύνδεσης με το υποσύστημα κάνοντας ευκολότερη την επικοινωνία μαζί του. δεν ορίζει καινούριες λειτουργικότητες και οι κλάσεις των υποσυστημάτων δεν γνωρίζουν την ύπαρξη του.

Συνήθως μόνο ένα αντικείμενο Πρόσοψης είναι απαραίτητο. Έτσι τα αντικείμενο Πρόσοψης είναι συχνά και Μοναδιαία (singleton).

ΥΠΟΚΕΦΑΛΑΙΟ 3.6 Πρότυπο Πληρεξούσιος,Structural Pattern – Proxy

Το πρόβλημα:

Παροχή ενός υποκατάστατου ή σημείου πρόσβασης σε ένα άλλο αντικείμενο ώστε να κάνει έλεγχο πρόσβασης σε αυτό.

Επίσης γνωστό ως

Υποκατάστατο

Σκοπός:

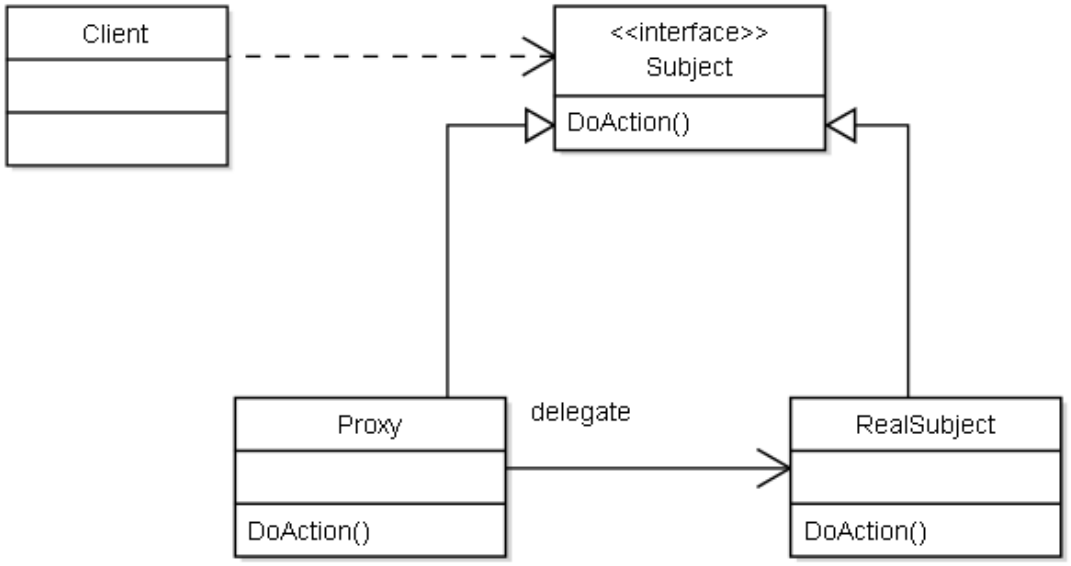
Ένας λόγος για τον οποίο χρειάζεται ο έλεγχος πρόσβασης σε ένα αντικείμενο είναι να αναβάλλουμε την δημιουργία και ενεργοποίηση του, μέχρι την στιγμή που θα τον χρειαστούμε πραγματικά. Έστω ένας επεξεργαστής εγγράφων ο οποίος μπορεί επίσης να δεχθεί (εισάγει) γραφικά αντικείμενα σε ένα αρχείο εγγράφου. Κάποια τέτοια γραφικά αντικείμενα όπως πολύπλοκα γραφήματα ράστερ μπορεί να έχουν μεγάλο κόστος υλοποίησης και δημιουργίας. Το άνοιγμα όμως ενός εγγράφου πρέπει να είναι άμεσο και έτσι αποφεύγουμε να δημιουργήσουμε όλα τα «βαριά» αντικείμενα που περιέχει αμέσως μόλις ανοιχτεί το έγγραφο. Ούτως ή άλλως όλα αυτά τα «βαριά» αντικείμενα δεν μπορεί να είναι ορατά ταυτόχρονα (αφού προφανώς μοιράζονται στις πολλαπλές σελίδες και σημεία του εγγράφου).

Αυτοί οι περιορισμοί προτείνουν την δημιουργία κάθε τέτοιου «ακριβού» (σε κόστος) αντικειμένου κατ’ απαίτηση, η οποία σε αυτή την περίπτωση είναι προφανώς όταν το αντικείμενο γίνεται ορατό. Αλλά το πρόβλημα εδώ είναι τι θα εμφανίζεται στην θέση του αντικειμένου που δεν έχει δημιουργηθεί ακόμη χωρίς να επιφέρουμε πολυπλοκότητα στον Επεξεργαστή Εγγράφων.

Η λύση είναι να χρησιμοποιήσουμε ένα άλλο αντικείμενο, έναν **Πληρεξούσιο**, ο

οποίος θα καταλαμβάνει την θέση του πραγματικού γραφικού. Ο Πληρεξούσιος δρα ακριβώς όπως το ίδιο το αντικείμενο γραφικού, και φυσικά το ενεργοποιεί, όταν και αν χρειαστεί, παραχωρώντας την θέση του.

Διάγραμμα UML:



Σχήμα 10: Διάγραμμα UML προτύπου Proxy

Σχετικά Πρότυπα:

Το πρότυπο Προσαρμογέας: Ένας προσαρμογέας παρέχει μία διαφορετική διεπαφή στο αντικείμενο στο οποίο προσαρμόζεται. Σε αντίθεση ένας Πληρεξούσιος παρέχει την ίδια διεπαφή όπως και το αντικείμενο το οποίο αντιπροσωπεύει. Παρόλα αυτά ένας Πληρεξούσιος που χρησιμοποιείται για την προστασία πρόσβασης μπορεί να αρνηθεί να εκτελέσει κάποια λειτουργία την οποία θα εκτελούσε το αντικείμενο που αντιπροσωπεύει και έτσι η διεπαφή του μπορεί σταδιακά να γίνει ένας μερικός αντιπρόσωπος του αντικειμένου.

Το πρότυπο Διακοσμητής: Αν και οι Διακοσμητές μπορούν να έχουν παρόμοιες υλοποιήσεις με τους Πληρεξούσιους, έχουν διαφορετικό σκοπό. Ένας Διακοσμητής προσθέτει μία ή περισσότερες αρμοδιότητες σε ένα αντικείμενο, ενώ αντίθετα ο Πληρεξούσιος ελέγχει την πρόσβαση στο αντικείμενο.

Οι Μεσολαβητές μπορούν να έχουν διαφορετικούς βαθμούς υλοποίησης όπως και οι Διακοσμητές. Ένας Μεσολαβητής προστασίας μπορεί να υλοποιηθεί ακριβώς όπως και ένας Διακοσμητής. Από την άλλη μεριά, ένας απομακρυσμένος (remote) Μεσολαβητής δεν θα περιέχει μία ευθεία αναφορά στο πραγματικό του υποκείμενο αλλά μία αναφορά που θα περιγράφει τη διεύθυνση του εξυπηρέτη και την περιγραφή του (host address, host ID).

Το πρότυπο Επανάληψης μπορεί να ιδωθεί ως ένα είδος Μεσολαβητή.

ΚΕΦΑΛΑΙΟ 4

<ΣΥΜΠΕΡΙΦΟΡΙΚΑ ΠΡΟΤΥΠΑ ΣΧΕΔΙΑΣΗΣ BEHAVIORAL PATTERNS>

ΕΙΣΑΓΩΓΗ

Τα συμπεριφορικά πρότυπα ασχολούνται με αλγόριθμούς και την ανάθεση αρμοδιοτήτων μεταξύ αντικειμένων. Περιγράφουν επίσης όχι απλώς πρότυπα κλάσεων και αντικειμένων, αλλά και πρότυπα επικοινωνίας μεταξύ αυτών. Αυτά τα πρότυπα χαρακτηρίζουν πολύπλοκους τρόπους ελέγχου, οι οποίες είναι δύσκολο να παρακολουθηθούν την στιγμή της εκτέλεσης (run-time). Μετατοπίζουν τον έλεγχο από τον τρόπο ελέγχου των αντικειμένων στον τρόπο με τον οποίο αυτά συνδέονται μεταξύ τους.

Οι συμπεριφορικές κλάσεις χρησιμοποιούν την κληρονομικότητα για να διαχύσουν αυτές ακριβώς τις συμπεριφορές μεταξύ των κλάσεων. Κάποια συμπεριφορικά πρότυπα χρησιμοποιούν την ενθυλάκωση (encapsulation) για να γίνουν οι «αντιπρόσωποι» ενός αντικειμένου και να χειριστούν τις κλήσεις σε αυτό.

ΥΠΟΚΕΦΑΛΑΙΟ 4.1 Πρότυπο Προσταγής, Behavioral Pattern – Command

Το πρόβλημα:

Η συσκευασία μίας εντολής ως ένα αντικείμενο έτσι ώστε να να επιτρέπει την παραμετροποίηση των πελατών (Clients) με διαφορετικές απαιτήσεις, καθώς και υποστήριξη λειτουργιών που δεν γινόντουσαν (δεν υποστηριζόντουσαν) προηγουμένως.

Επίσης γνωστό ως:

Δράση, Συναλλαγή (Action, Transaction)

Σκοπός:

Πολλές φορές είναι απαραίτητο να αποσταλούν αιτήματα σε αντικείμενα χωρίς να υπάρχει γνώση αυτής καθεαυτής της διαδικασίας, η του παραλήπτη του αιτήματος. Για παράδειγμα ένα κουμπί στην διεπαφή χρήστη μπορεί να κάνει κάτι το οποίο ο δημιουργός του κουμπιού δεν γνωρίζει. Δεν γνωρίζει καν ποια εφαρμογή θα καλέσει, ή τι δεδομένα θα στείλει ένα τέτοιο κουμπί.

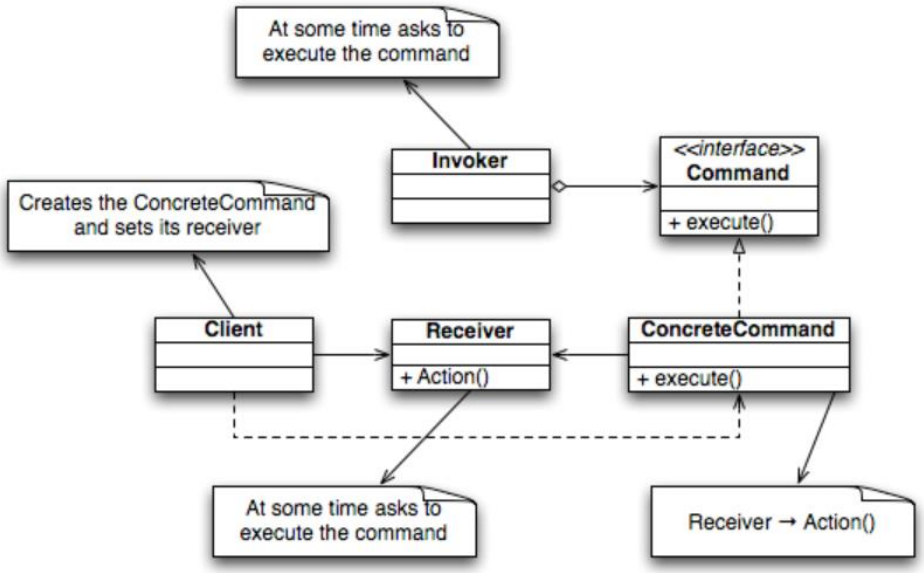
Το παρόν πρότυπο επιτρέπει σε τέτοια κουμπιά να στέλνουν άγνωστα μηνύματα σε άγνωστους παραλήπτες μεταμορφώνοντας το μήνυμα σε αντικείμενο. Αυτό το αντικείμενο μπορεί να αποθηκευτεί και να χειριστεί όπως όλα τα υπόλοιπα αντικείμενα. Αυτό επιτυγχάνεται με μία ιδεατή κλάση Προσταγής (Command) η οποία δηλώνει μία διεπαφή η οποία θα εκτελεί αυτές τις λειτουργίες.

Με το πρότυπο αυτό τα μενού μπορούν να κατασκευαστούν πολύ εύκολα. Κάθε επιλογή σε ένα μενού είναι ένα στιγμιότυπο της κλάσης MenuItem. Μία εφαρμογή δημιουργεί αυτά τα μενού μαζί με την υπόλοιπη διεπαφή χρήστη (user interface).

Η εφαρμογή αυτή επίσης παραμετροποιεί κάθε MenuItem με ένα στιγμιότυπο μίας υποκλάσης Προσταγής. Όταν ο χρήστης επιλέγει ένα MenuItem αυτό καλεί την μέθοδο εκτέλεση (execute) πάνω σε αυτό το συγκεκριμένο στιγμιότυπο της κλάσης. Τα ίδια τα MenuItems δεν γνωρίζουν ποια υποκλάση της εκτέλεσης έχουν καλέσει.

Για παράδειγμα η Επικόλληση (Paste) όταν καλείται θα φέρει μέσα στο ενεργό έγγραφο τα περιεχόμενα του πρόχειρου. Αλλά η ίδια η εφαρμογή που κάλεσε (εκτέλεσε) την εντολή «επικόλληση» δεν γνωρίζει τι θα φέρει μέσα στο ενεργό έγγραφο.

Διάγραμμα UML:



Σχήμα 11: Διάγραμμα UML προτύπου command

Σχετικά Πρότυπα:

Το πρότυπο Σύνθεσης μπορεί να χρησιμοποιηθεί για να υλοποιηθούν οι μακροεντολές

Το πρότυπο Αναμνηστικού μπορεί να διατηρεί το ιστορικό της κατάστασης της εντολής που απαιτείται για να αρθεί το τελευταίο εφέ που χρησιμοποιήθηκε

Μία εντολή που πρέπει να αντιγραφεί (copy) πριν να αρχειοθετηθεί στην λίστα εγγραφών, λειτουργεί ως πρότυπο Πρωτότυπου.

ΥΠΟΚΕΦΑΛΑΙΟ 4.2 Πρότυπο Ακολουθία Ευθύνης, Behavioral Pattern – Chain of Responsibility

Το πρόβλημα:

Αποφυγή σύζευξης του αποστολέα μίας αίτησης με τον παραλήπτη, παρέχοντας σε περισσότερα του ενός αντικείμενα την δυνατότητα να χειριστούν την αίτηση. Δημιουργία μίας ακολουθίας με τα παραλαμβάνοντα αντικείμενα και πέρασμα της αίτησης δια μήκους της ακολουθίας μέχρι κάποιο αντικείμενο της ακολουθίας να την παραλάβει.

Σκοπός:

Έστω μία αλληλεπιδραστική εφαρμογή παροχής βοήθειας για ένα γραφικό περιβάλλον χρήστη. Ο χρήστης μπορεί να ανατρέξει στην βοήθεια σε κάθε σημείο του γραφικού περιβάλλοντος, κάνοντας κλικ επάνω σε εκείνο το σημείο. Είναι προφανές ότι η βοήθεια θα επιστρέψει το σχετικό με το συγκεκριμένο σημείο (ή αντικείμενο) απόσπασμα. Αν δεν υπάρχει συγκεκριμένο απόσπασμα για αυτό το συγκεκριμένο σημείο, πρέπει να επιστρέψει το γενικό τμήμα της βοήθειας ή κάποιο εισαγωγικό κείμενο.

Είναι προφανές ότι η βοήθεια πρέπει να οργανωθεί από το ειδικότερο προς το γενικότερο. Είναι επίσης προφανές ότι το αίτημα για παροχή βοήθειας θα το χειριστούν πολλαπλά αντικείμενα του γραφικού περιβάλλοντος. Το ποιο ακριβώς τελικά θα την παραλάβει, εξαρτάται από το συγκεκριμένο σημείο που έκανε κλικ ο χρήστης.

Το πρόβλημα στην παραπάνω περιγραφή είναι ότι το αντικείμενο το οποίο θα παρέχει την βοήθεια δεν είναι γνωστό στο αντικείμενο το οποίο εκκινεί την παροχή βοήθειας. Εδώ χρειάζεται ένας μηχανισμός ο οποίος να αποσυμπλέκει το κουμπί που εκκίνησε την διαδικασία παροχής βοήθειας από το αντικείμενο που τελικά θα παράσχει την βοήθεια. Το πρότυπο Ακολουθίας Ευθύνης υποδεικνύει τον τρόπο που θα γίνει αυτό.

Εφαρμογές:

- Χρησιμοποιούμε την Ακολουθία Ευθύνης όταν
- Περισσότερα του ενός αντικείμενα πρέπει να χειριστούν μία αίτηση και ο τελικός χειριστής (παραλήπτης) δεν είναι γνωστός εκ των προτέρων. Ο τελικός χειριστής πρέπει να μπορεί να βρεθεί μέσα στην ακολουθία αυτόματα.
- Θέλουμε να εκκινήσουμε μία αίτηση σε ένα από μία ομάδα αντικειμένων χωρίς να συγκεκριμενοποιήσουμε τον παραλήπτη.
- Η ακολουθία αντικειμένων που μπορούν να χειριστούν μία αίτηση πρέπει να οριστεί δυναμικά.

Παράδειγμα:

Οι παρακάτω κώδικες σε Java δείχνουν την εφαρμογή του προτύπου μέσω ενός παραδείγματος μίας κλάσης καταγραφής συμβάντων (logging class). Κάθε χειριστής συμβάντος αποφασίζει αν πρέπει να κάνει κάτι (action taken) σε αυτό το επίπεδο, και μετά περνάει το μήνυμα (δηλαδή το συμβάν) στον επόμενο χειριστή συμβάντων. Η έξοδος θα μπορούσε να περιγραφεί ως εξής:

```
Writing to stdout:   Entering function y.
Writing to stdout:   Step1 completed.
Sending via e-mail:  Step1 completed.
Writing to stdout:   An error has occurred.
Sending via e-mail:  An error has occurred.
Writing to stderr:   An error has occurred.
```

Οι πρότυπες κλάσεις που περιγράφονται παρακάτω ΔΕΝ πρέπει να χρησιμοποιηθούν για την υλοποίηση ενός πραγματικού logger αλλά παρουσιάζονται μόνο για να επιδειχθεί η λογική του περάσματος του μηνύματος σε διαφορετικά επίπεδα χειριστών συμβάντων. Στην πραγματικότητα ένας logger λαμβάνει το μήνυμα και το αποθηκεύει ΧΩΡΙΣ να περνάει το μήνυμα στους επόμενους χειριστές.

```
abstract class Logger {
    public static int ERR = 3;
```

```
    public static int NOTICE = 5;
    public static int DEBUG = 7;
    protected int mask;

    // The next element in the chain of responsibility
    protected Logger next;

    public Logger setNext(Logger log) {
        next = log;
        return log;
    }

    public void message(String msg, int priority) {
        if (priority <= mask) {
            writeMessage(msg);
        }
        if (next != null) {
            next.message(msg, priority);
        }
    }

    abstract protected void writeMessage(String msg);
}

class StdoutLogger extends Logger {
    public StdoutLogger(int mask) {
        this.mask = mask;
    }

    protected void writeMessage(String msg) {
        System.out.println("Writing to stdout: " + msg);
    }
}


class EmailLogger extends Logger {
    public EmailLogger(int mask) {
        this.mask = mask;
    }

    protected void writeMessage(String msg) {
        System.out.println("Sending via email: " + msg);
    }
}

class StderrLogger extends Logger {
    public StderrLogger(int mask) {
        this.mask = mask;
    }

    protected void writeMessage(String msg) {
        System.err.println("Sending to stderr: " + msg);
    }
}

public class ChainOfResponsibilityExample {
    public static void main(String[] args) {
        // Build the chain of responsibility
```

```
        Logger logger, logger1;
        logger1 = logger = new StdoutLogger(Logger.DEBUG);
        logger1 = logger1.setNext(new EmailLogger(Logger.NOTICE));
        logger1 = logger1.setNext(new StderrLogger(Logger.ERR));

        // Handled by StdoutLogger
        logger.message("Entering function y.", Logger.DEBUG);

        // Handled by StdoutLogger and EmailLogger
        logger.message("Step1 completed.", Logger.NOTICE);

        // Handled by all three loggers
        logger.message("An error has occurred.", Logger.ERR);
    }
}
```

Σχήμα 12: Παράδειγμα κώδικα Java για το πρότυπο Chain of Responsibility

Σχετικά Πρότυπα:

Το πρότυπο Ακολουθία Ευθύνης συχνά εφαρμόζεται σε συνδυασμό με το Πρότυπο Σύνθεσης.

ΥΠΟΚΕΦΑΛΑΙΟ 4.3 Πρότυπο Διερμηνέας, Behavioral Pattern – Interpreter

Το πρόβλημα:

Δοθείσης μίας γλώσσας, να οριστεί μία αναπαράσταση της γραμματικής της με ένα διερμηνευτή (interpreter) που χρησιμοποιεί την αναπαράσταση αυτή για να ερμηνεύσει φράσεις αυτής της γλώσσας.

Σκοπός:

Εάν ένα συγκεκριμένο είδος προβλήματος εμφανίζεται αρκετά συχνά, τότε ίσως είναι προτιμότερο να αναπαραστήσουμε στιγμιότυπα αυτού του προβλήματος σε μία απλή γλώσσα. Μετά μπορούμε να δημιουργήσουμε έναν διερμηνευτή που να λύνει το πρόβλημα διερμηνεύοντας τα στιγμιότυπα που δημιουργήσαμε.

Έστω ότι ψάχνουμε συγκεκριμένη συμβολοσειρά που αντιστοιχεί σε συγκεκριμένη φόρμα (pattern) π.χ. όλες οι συμβολοσειρές που έχουν 8 γράμματα και περιέχουν τουλάχιστον 3 ‘α’ και μία παύλα ‘-‘ . Οι κανονικές εκφράσεις (regular expressions) χρησιμοποιούνται για να ορίσουν φόρμες σε συμβολοσειρές. Αντί να ξεκινήσουμε να φτιάξουμε αλγόριθμο που αναγνωρίζει την παραπάνω φράση, μπορούμε να χρησιμοποιήσουμε τις ιδιότητες των κανονικών εκφράσεων για να διατρέξουμε τις συμβολοσειρές που μας ενδιαφέρουν.

Το πρότυπο Διερμηνευτής περιγράφει το πώς να ορίζουμε μία γραμματική για απλές γλώσσες, πώς να αναπαραστήσουμε προτάσεις σε αυτές τις γλώσσες, και πως τελικά να διατρέξουμε συγκεκριμένες προτάσεις (συμβολοσειρές) σε αυτές τις γλώσσες.

Εφαρμογές:

Χρησιμοποιούμε το πρότυπο Διερμηνευτής όταν υπάρχει μία γλώσσα προς ανάλυση και μπορούμε να αναπαραστήσουμε τα κατηγορήματα της γλώσσας σαν ιδεατά συντακτικά δέντρα. Το πρότυπο εφαρμόζεται καλύτερα όταν:

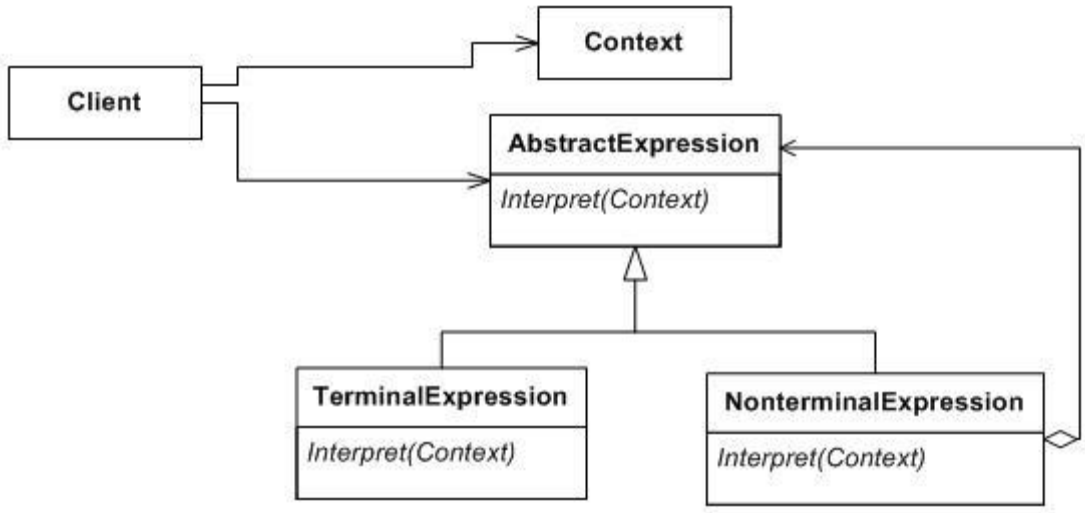
- Η γραμματική είναι απλή. Για πολύπλοκες γραμματικές η ιεραρχία των κλάσεων της γραμματικής γίνεται μεγάλη και ξεφεύγει από τον έλεγχο.
- Η αποτελεσματικότητα δεν είναι το κύριο ζητούμενο. Οι πιο αποτελεσματικοί διερμηνευτές πρέπει πρώτα να μετατραπούν σε άλλη μορφή πριν χρησιμοποιηθούν.

Χρήσεις:

Το πρότυπο Διερμηνευτής χρησιμοποιείται ευρύτατα σε Συντάκτες (Compilers) αντικειμενοστρεφών γλωσσών χρήσεις η SmallTalk.

Στην πιο γενική περίπτωση, σχεδόν κάθε χρήση του προτύπου Σύνθεσης θα περιέχει χρήσεις και το πρότυπο Διερμηνευτή. Αλλά η χρήση του Διερμηνευτή θα πρέπει να γίνεται μόνο χρήσεις περιπτώσεις που η ιεραρχία κλάσεων μπορεί (και πρέπει) να ιδωθεί ως γλώσσα.

Διάγραμμα UML:



Σχήμα 13: Διάγραμμα UML Προτύπου Intepreter

Σχετικά Πρότυπα:

- Πρότυπο Σύνθεσης: το ιδεατό δέντρο σύνταξης είναι ένα στιγμιότυπο του προτύπου Σύνθεσης
- Πρότυπο Επανάληψης: Ο Διερμηνευτής μπορεί να χρησιμοποιήσεις τον Επαναλήπτη για να διατρέξει την δομή υπό εξέταση
- Επισκέπτης: Μπορεί να χρησιμοποιηθεί για να διατηρήσει την συμπεριφορά κάθε κόμβου χρήσεις δομής δέντρου σε μία κλάση

ΥΠΟΚΕΦΑΛΑΙΟ 4.4 Πρότυπο Επαναλήπτης, Behavioral Pattern – Iterator

Το πρόβλημα:

Παροχή ενός τρόπου πρόσβασης στα στοιχεία ενός συλλογικού αντικειμένου διαδοχικά, χωρίς να γίνει γνωστή η εσωτερική αναπαράσταση των στοιχείων αυτών.

Επίσης γνωστό ως:

Δρομέας

Σκοπός:

Ένα συλλογικό αντικείμενο, όπως πχ μία λίστα πρέπει να παρέχει έναν τρόπο πρόσβασης στα στοιχεία της χωρίς να αποκαλύπτει την εσωτερική δομή της. Ακόμη είναι πιθανόν να χρειάζεται να διατρέξουμε την λίστα με διαφορετικούς τρόπους αναλόγως από το ζητούμενο κάθε φορά. Δεν θέλουμε όμως να «φορτώσουμε» την διεπαφή της λίστας με όλους αυτούς τους τρόπους πρόσβασης σε αυτή. Επίσης ίσως χρειάζεται να διατρέχουν περισσότερες της μίας (Διαφορετικές) αναζητήσεις στην λίστα κάθε δεδομένη στιγμή.

Το πρότυπο Επαναλήπτης αναλαμβάνει όλο αυτό το έργο. Αναλαμβάνει την ευθύνη πρόσβασης και να διατρέχει την λίστα καθώς και την εξαγωγή του στοιχείου υπό αναζήτηση και την ανάθεση του σε ένα αντικείμενο **επανάληψης**. Η κλάση Επαναλήπτης είναι υπεύθυνη να διατηρεί την θέση του τρέχοντος στοιχείου, γνωρίζει δηλαδή ποια στοιχεία έχει ήδη διατρέξει.

Εφαρμογές:

Χρησιμοποιούμε τον Επαναλήπτη:

- Για να προσπελάσουμε ένα αντικείμενο που περιέχει σύνολο στοιχείων χωρίς να αποκαλύψουμε την εσωτερική δομή του αντικειμένου
- Για να μπορούμε να διατρέξουμε αυτό το αντικείμενο πολλαπλές και ίσως ταυτόχρονες φορές
- Για να παρέχουμε μία ομοιόμορφη διεπαφή που να διατρέχει διαφορετικές δομές (να υποστηρίζει δηλαδή πολυμορφισμό μέσω Επαναλήπτη)

Χρήσεις:

Οι Επαναλήπτες βρίσκονται συχνά σε αντικειμενοστρεφή συστήματα.

Παραδείγματα Υλοποίησης:

Μερικές γλώσσες υλοποιούν το συντακτικό τους με στάνταρ/συγκεκριμένο τρόπο. Η C++ και η Python είναι χαρακτηριστικά παραδείγματα.

C++

Η C + + υλοποιεί τις επαναλήψεις με την χρήση των δεικτών. Στην C + +, μια τάξη μπορεί να υπερφορτώσει το σύνολο των απαραίτητων εργασιών δείκτη, έτσι ώστε μία επανάληψη μπορεί να υλοποιηθεί ώστε να επιδρα πάνω σε ένα δείκτη, και μέσω της δεικτοδότησης να υλοποιεί την αύξηση και μείωση του. Αυτό έχει το πλεονέκτημα ότι η C + + υλοποιεί εύκολα αλγόριθμούς όπως ο std::sort μπορεί άμεσα να εφαρμοστεί σε παλιά buffers μνήμης, και ότι δεν υπάρχει καθόλου νέο συντακτικό. Ωστόσο, αυτό απαιτεί έναν επαναλήπτη "τέλους" για τη δοκιμή για την ισότητα, αντί να επιτρέπει σε έναν επαναλήπτη να γνωρίζει ότι έχει φτάσει στο τέλος.

Java

Στην Java υπάρχει μία διεπαφή Iterator την οποία πρέπει να υλοποιήσουν οι Συλλογές ώστε να μπορούν από έξω να διατρέξουν τα στοιχεία της συλλογής.

Σχετικά Πρότυπα:

Πρότυπο Σύνθεσης: Οι Επαναλήπτες συχνά εφαρμόζονται σε αναδρομικές δομές όπως αυτές που παράγει το πρότυπο Σύνθεσης

Πρότυπο Εργοστασίου: Οι πολυμορφικοί Επαναλήπτες βασίζονται σε μεθόδους του προτύπου Εργοστάσιο για να δημιουργήσουν ένα στιγμιότυπο της κατάλληλης κλάσης Επαναλήπτη

Πρότυπο Ενθύμιο: Το πρότυπο Ενθύμιο πολλές φορές χρησιμοποιείται σε συνδυασμό με το πρότυπο Επανάληψης. Ένας Επαναλήπτης μπορεί να χρησιμοποιήσει ένα Ενθύμιο για να διατηρήσει την κατάσταση της επανάληψης. Ο Επαναλήπτης αποθηκεύει εσωτερικά το Ενθύμιο.

ΥΠΟΚΕΦΑΛΑΙΟ 4.5 Πρότυπο Μεσολαβητής, Behavioral Pattern – Mediator

Το πρόβλημα:

Καθορισμός ενός αντικειμένου που ενσωματώνει τους τρόπους που μία ομάδα αντικειμένων αλληλεπιδρά. Το πρότυπο προκρίνει το πρότυπο χαμηλής σύμπλεξης (low coupling) με το να αποτρέπει τα αντικείμενα από το να αναφέρονται το ένα για το άλλο αποκλειστικά και επιτρέπει στον χρήστη να τροποποιήσει τον βαθμό αλληλεπίδρασης ανεξάρτητα από τα υπόλοιπα.

Σκοπός:

Ο αντικειμενοστρεφής σχεδιασμός ευνοεί την διανομή γνώσης μεταξύ αντικειμένων. Αυτό προφανώς μπορεί να οδηγήσει σε αντικείμενα στενά συνδεδεμένα με όλα τα υπόλοιπα. Μπορεί να φτάσει στο σημείο, κάθε αντικείμενο να γνωρίζει «τα πάντα» για όλα τα άλλα αντικείμενα.

Αν και η τμηματοποίηση ενός συστήματος ευνοεί την επαναχρησιμοποίηση, οι στενοί δεσμοί μεταξύ αυτών των τμημάτων, άρουν αυτό το πλεονέκτημα. Επίσης έτσι είναι πολύ δύσκολο να αλλάξει η συμπεριφορά του συστήματος αφού όλα τα τμήματα του αλληλεπιδρούν.

Αυτά τα προβλήματα μπορούν να αποφευχθούν με την ενθυλάκωση πολλών συμπεριφορών που διατρέχουν πολλά τμήματα σε ένα ξεχωριστό αντικείμενο **Μεσολαβητή**. Ο Μεσολαβητής είναι υπεύθυνος για τον έλεγχο και συντονισμό των αλληλεπιδράσεων σε μία ομάδα αντικειμένων. Ο Μεσολαβητής δρα ως ο ενδιάμεσος που αποτρέπει τα αντικείμενα μέσα σε μία ομάδα από το να αναφέρονται το ένα στο άλλο. Αντ’ αυτού τα αντικείμενα γνωρίζουν μόνο τον Μεσολαβητή και έτσι μειώνονται οι απευθείας «επαφές» μεταξύ αντικειμένων.

Εφαρμογές:

Χρησιμοποιούμε τον Μεσολαβητή όταν:

- Μία ομάδα αντικειμένων επικοινωνούν με καλά ορισμένους αλλά και πολύπλοκους τρόπους. Οι αλληλεξαρτήσεις που προκύπτουν είναι αδόμητες και δύσκολες στην κατανόηση.
- Η επαναχρησιμοποίηση ενός αντικειμένου είναι δύσκολη επειδή αυτό αναφέρεται και επικοινωνεί με πολλά άλλα αντικείμενα
- Η συμπεριφορά που διαχέεται μεταξύ κλάσεων πρέπει να μπορεί να παραμετροποιηθεί χωρίς την δημιουργία πολλών υποκλάσεων

Συνέργειες

Οι κλάσεις του προτύπου Συνεργάτη (Colleuages) στέλνουν και λαμβάνουν αιτήσεις από ένα αντικείμενο του προτύπου Μεσολαβητή. Ο Μεσολαβητής υλοποιεί την συνεργατική συμπεριφορά με το να δρομολογεί αιτήματα μεταξύ των κατάλληλων Συνεργατών.

Επιπτώσεις:

Το πρότυπο Συνεργάτης έχει τα ακόλουθα πλεονεκτήματα & μειονεκτήματα:

1. Περιορίζει την δημιουργία υποκλάσεων: Ο Μεσολαβητής επιβάλει συγκεκριμένες συμπεριφορές να παραμείνουν σε τοπικό επίπεδο οι οποίες διαφορετικά θα διέτρεχαν πολλαπλά αντικείμενα. Η αλλαγή αυτή με άλλο τρόπο θα απαιτούσε εκτεταμένο αριθμό υποκλάσεων, ενώ με αυτό τον τρόπο οι κλάσεις των συμμετεχόντων μπορούν να χρησιμοποιηθούν ως έχουν.
2. Αποσυμπλέκει Συνεργάτες-Αντικείμενα: Ο Μεσολαβητής προάγει το πρότυπο Χαμηλής Σύμπλεξης (Low Coupling) μεταξύ συνεργαζόμενων αντικειμένων. Έτσι οι κλάσεις των αντικειμένων αυτών μπορούν να χρησιμοποιηθούν αυτόνομα.

3. Απλοποιεί τα πρωτόκολλα επικοινωνίας μεταξύ αντικειμένων: Ο Μεσολαβητής αντικαθιστά αλληλεπιδράσεις πολλά-προς-πολλά με ένα-προς-πολλά, δηλαδή μεταξύ του Μεσολαβητή και των συνεργαζόμενων με αυτόν αντικειμένων. Οι σχέσεις ένα-προς-πολλά είναι πιο εύκολες στην κατανόηση, επέκταση και συντήρηση.
4. Μεταφέρει την αλληλεπίδραση των αντικειμένων σε αφηρημένο επίπεδο: Επειδή ενσωματώνει το αντικείμενο της μεταφοράς επιτρέπει να συγκεντρωθούμε στην αλληλεπίδραση των αντικειμένων και όχι στην συμπεριφορά αυτού του αντικειμένου.
5. Μεταφέρει σε κεντρικό επίπεδο τον έλεγχο: Το πρότυπο μεσολάβησης ανταλλάσει πολυπλοκότητα της συναλλαγής με την πολυπλοκότητα του ίδιου του Μεσολαβητή. Επειδή ο Μεσολαβητής ενσωματώνει πρωτόκολλα επικοινωνίας μπορεί να γίνει πολυπλοκότερος από τα αντικείμενα στα οποία μεσολαβεί. Έτσι μπορεί μερικές φορές να γίνει τόσο μονολιθικός που να είναι δύσκολος στην συντήρηση.

Παράδειγμα Υλοποίησης σε Java:

Η λογική του προτύπου είναι να «Ορίσει ένα αντικείμενο το οποίο εμπερικλείει τον τρόπο με τον οποίο ένα σετ αντικειμένων αλληλεπιδρά. Το πρότυπο προωθεί το «Loose coupling» με το να διατηρεί τα αντικείμενα από το να αναφέρονται το ένα στο άλλο αποκλειστικά και επιτρέπει στον χρήστη να διαφοροποιεί τις αλληλεπιδράσεις τους ανεξάρτητα το ένα από το άλλο (Gamma κ.ά., 1995).

```
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

//Colleague interface
interface Command {
    void execute();
```

```
}

//Concrete mediator
class Mediator {

    BtnView btnView;
    BtnSearch btnSearch;
    BtnBook btnBook;
    LblDisplay show;

    //....
    void registerView(BtnView v) {
        btnView = v;
    }

    void registerSearch(BtnSearch s) {
        btnSearch = s;
    }

    void registerBook(BtnBook b) {
        btnBook = b;
    }

    void registerDisplay(LblDisplay d) {
        show = d;
    }

    void book() {
        btnBook.setEnabled(false);
        btnView.setEnabled(true);
        btnSearch.setEnabled(true);
        show.setText("booking...");
    }

    void view() {
        btnView.setEnabled(false);
        btnSearch.setEnabled(true);
        btnBook.setEnabled(true);
        show.setText("viewing...");
    }

    void search() {
        btnSearch.setEnabled(false);
        btnView.setEnabled(true);
        btnBook.setEnabled(true);
        show.setText("searching...");
    }

}

//A concrete colleague
class BtnView extends JButton implements Command {

    Mediator med;

    BtnView(ActionListener al, Mediator m) {
        super("View");
        addActionListener(al);
```

```
        med = m;
        med.registerView(this);
    }

    public void execute() {
        med.view();
    }

}

//A concrete colleague
class BtnSearch extends JButton implements Command {

    Mediator med;

    BtnSearch(ActionListener al, Mediator m) {
        super("Search");
        addActionListener(al);
        med = m;
        med.registerSearch(this);
    }

    public void execute() {
        med.search();
    }

}

//A concrete colleague
class BtnBook extends JButton implements Command {

    Mediator med;

    BtnBook(ActionListener al, Mediator m) {
        super("Book");
        addActionListener(al);
        med = m;
        med.registerBook(this);
    }

    public void execute() {
        med.book();
    }

}

class LblDisplay extends JLabel {

    Mediator med;

    LblDisplay(Mediator m) {
        super("Just start...");
        med = m;
        med.registerDisplay(this);
        setFont(new Font("Arial", Font.BOLD, 24));
    }

}
```

```
class MediatorDemo extends JFrame implements ActionListener {

    Mediator med = new Mediator();

    MediatorDemo() {
        JPanel p = new JPanel();
        p.add(new BtnView(this, med));
        p.add(new BtnBook(this, med));
        p.add(new BtnSearch(this, med));
        getContentPane().add(new LblDisplay(med), "North");
        getContentPane().add(p, "South");
        setSize(400, 200);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent ae) {
        Command comd = (Command) ae.getSource();
        comd.execute();
    }

    public static void main(String[] args) {
        new MediatorDemo();
    }

}
```

Σχετικά Πρότυπα:

Πρότυπο Πρόσοψη (Façade) : Διαφέρει από τον Μεσολαβητή στο ότι προάγει σε αφηρημένο επίπεδο ένα υποσύστημα αντικειμένων για να παρέχει μία πιο εύχρηστη διεπαφή. Η επικοινωνία της είναι μονόδρομη, δηλαδή η Πρόσοψη στέλνει αιτήσεις προς το υποσύστημα που αντιπροσωπεύει, αλλά όχι το αντίθετο. Αντίθετα ο Μεσολαβητής ενεργοποιεί την αμφίδρομη συνεργασία μεταξύ αντικειμένων που είτε δεν είχαν, είτε δεν μπορούσαν να έχουν και ως εκ τούτου η επικοινωνία είναι αμφίδρομη.

Αντικείμενα του προτύπου Συνεργάτη (Colleagues) μπορούν να επικοινωνήσουν με τον Μεσολαβητή μέσω αντικειμένων του προτύπου Παρατηρητή (Observer).

ΥΠΟΚΕΦΑΛΑΙΟ 4.6 Πρότυπο Ενθύμιο, Behavioral Pattern – Memento

Το πρόβλημα:

Χωρίς να παραβιάζεται η ενθυλάκωση (Encapsulation) να μπορεί να καταγράφεται και να δημοσιοποιείται η εσωτερική κατάσταση ενός αντικειμένου, έτσι ώστε αυτό το αντικείμενο να μπορεί να επανέλθει σε αυτή την κατάσταση αργότερα.

Επίσης γνωστό ως:

Σύμβολο (Token)

Σκοπός:

Πολλές φορές είναι απαραίτητο να γνωρίζουμε την εσωτερική κατάσταση ενός αντικειμένου. Αυτό μπορεί να απαιτηθεί όταν υλοποιούνται σημεία ελέγχου και μηχανισμούς αναίρεσης (undo) οι οποίοι επιτρέπουν στον χρήστη να επιστρέψει σε πρότερη κατάσταση, ή να επανακάμψει μετά από λάθη ή αστοχίες. Αυτή η πληροφορία πρέπει κάπου να αποθηκευτεί ώστε να είναι διαθέσιμη για την αναίρεση. Συνήθως τα αντικείμενα διατηρούν κάποιο ιστορικό σχετικά με την κατάσταση τους, όχι όμως πλήρες. Επιπροσθέτως δεν είναι δυνατόν να γίνει αναφορά σε αυτή την πληροφορία εξωτερικά. Η δημοσιοποιήση αυτών των πεδίων αποτελεί παραβίαση της ενθυλάκωσης και μπορεί να οδηγήσει την εφαρμογή σε αστάθειες και αναξιοπιστία.

Αυτό το πρόβλημα μπορεί να επιλυθεί με το πρότυπο Ενθύμιο. Ένα **Ενθύμιο** είναι ένα αντικείμενο το οποίο αποθηκεύει (διατηρεί) την δεδομένη κατάσταση ενός άλλου αντικειμένου, του Δημιουργού (originator) του Ενθυμίου. Ο μηχανισμός αναίρεσης (undo mechanism) θα αιτηθεί ένα Ενθύμιο από τον Δημιουργό, όταν χρειαστεί ένα σημείο ελέγχου για την κατάσταση του Δημιουργού. Μόνο ο Δημιουργός μπορεί να αποθηκεύσει και να ανακαλέσει πληροφορία από

το Ενθύμιο του. Το Ενθύμιο είναι αδιαφανές (απροσπέλαστο) από άλλα αντικείμενα.

Εφαρμογές:

Χρησιμοποιούμε το πρότυπο Ενθύμιο όταν:

- Ένα στιγμιότυπο της κατάστασης ενός αντικειμένου (ή ένα μέρος αυτού), πρέπει να καταχωρηθεί ώστε να μπορεί να ανακληθεί αργότερα, και
- Η απευθείας έκθεση της κατάστασης του αντικειμένου αυτού στην διεπαφή θα μπορούσε να εκθέσει ευαίσθητες πληροφορίες και να παραβιάσει την ενθυλάκωση.

Παράδειγμα Υλοποίησης σε Java:

Ένα κλασσικό παράδειγμα του προτύπου είναι η υλοποίηση του seed ενός ψευδο-τυχαίου αλγόριθμου, το οποίο παράγει την ίδια ακολουθία αριθμών όταν αρχικοποιείται με το ίδιο seed. Άλλο κλασσικό παράδειγμα είναι η κατάσταση μίας finite state machine.

Παράδειγμα Υλοποίησης σε Java ενός “undo” μέσω του προτύπου Memento.

```
 import java.util.List;
import java.util.ArrayList;
class Originator {
    private String state;
    // The class could also contain additional data that is not part of
the
    // state saved in the memento.

    public void set(String state) {
        System.out.println("Originator: Setting state to " + state);
        this.state = state;
    }

    public Memento saveToMemento() {
        System.out.println("Originator: Saving to Memento.");
        return new Memento(state);
    }

    public void restoreFromMemento(Memento memento) {
```

```
        state = memento.getSavedState();
        System.out.println("Originator: State after restoring from
Memento: " + state);
    }

    public static class Memento {
        private final String state;

        private Memento(String stateToSave) {
            state = stateToSave;
        }

        private String getSavedState() {
            return state;
        }
    }
}

class Caretaker {
    public static void main(String[] args) {
        List<Originator.Memento> savedStates = new
ArrayList<Originator.Memento>();

        Originator originator = new Originator();
        originator.set("State1");
        originator.set("State2");
        savedStates.add(originator.saveToMemento());
        originator.set("State3");
        // We can request multiple mementos, and choose which one to roll
back to.
        savedStates.add(originator.saveToMemento());
        originator.set("State4");

        originator.restoreFromMemento(savedStates.get(1));
    }
}
```

Η έξοδος του προγράμματος είναι όπως παρακάτω:

```
Originator: Setting state to State1
Originator: Setting state to State2
Originator: Saving to Memento.
Originator: Setting state to State3
Originator: Saving to Memento.
Originator: Setting state to State4
Originator: State after restoring from Memento: State3
```

Σχετικά Πρότυπα:

- Πρότυπο Προσταγής (Command): Οι Προσταγές μπορούν να χρησιμοποιήσουν τα Ενθύμια για να διατηρήσουν ιστορικό ενεργειών που μπορούν να αρθούν.

- Πρότυπο Επαναλήπτης: Ενθύμια μπορούν να χρησιμοποιηθούν για επανάληψη αναίρεσης, όπως αναφέρθηκε.

ΥΠΟΚΕΦΑΛΑΙΟ 4.7 Πρότυπο Παρατηρητής, Behavioral Pattern – Observer

Το πρόβλημα:

Ορισμός μίας ένα-προς-πολλά εξάρτησης μεταξύ αντικειμένων έτσι ώστε όταν ένα αντικείμενο αλλάζει κατάσταση όλες οι εξαρτήσεις του να ενημερώνονται αυτόματα.

Επίσης γνωστό ως:

Εξαρτήσεις (Dependents), Έκδοση-Συνδρομή (Publish-Subscribe)

Σκοπός:

Μία γνωστή παρενέργεια της κατάτμησης ενός συστήματος σε συλλογή από συνεργαζόμενες κλάσεις είναι η ανάγκη της διατήρησης της συνέπειας και ακεραιότητας μεταξύ συσχετιζόμενων αντικειμένων. Ο στόχος της συνέπειας δεν θα πρέπει να επιτυγχάνεται με την αύξηση της σύμπλεξης (High coupling) μεταξύ τους, γιατί έτσι μειώνεται η δυνατότητα επαναχρησιμοποίησης τους.

Το πρότυπο Παρατηρητής περιγράφει πως μπορούν να αποκατασταθούν τέτοιες σχέσεις. Το κύριο εργαλείο σε αυτή την κατεύθυνση είναι το ζεύγος **Υποκειμένου-Παρατηρητή**. Το υποκείμενο είναι ο εκδότης των ειδοποιητηρίων. Στέλνει τα ειδοποιητήρια χωρίς να χρειάζεται να γνωρίζει ποιοι είναι οι παραλήπτες (παρατηρητές). Οποιοσδήποτε αριθμός Παρατηρητών μπορεί να «εγγραφεί» ως παραλήπτης συγκεκριμένων ειδοποιητηρίων.

Χρήσεις:

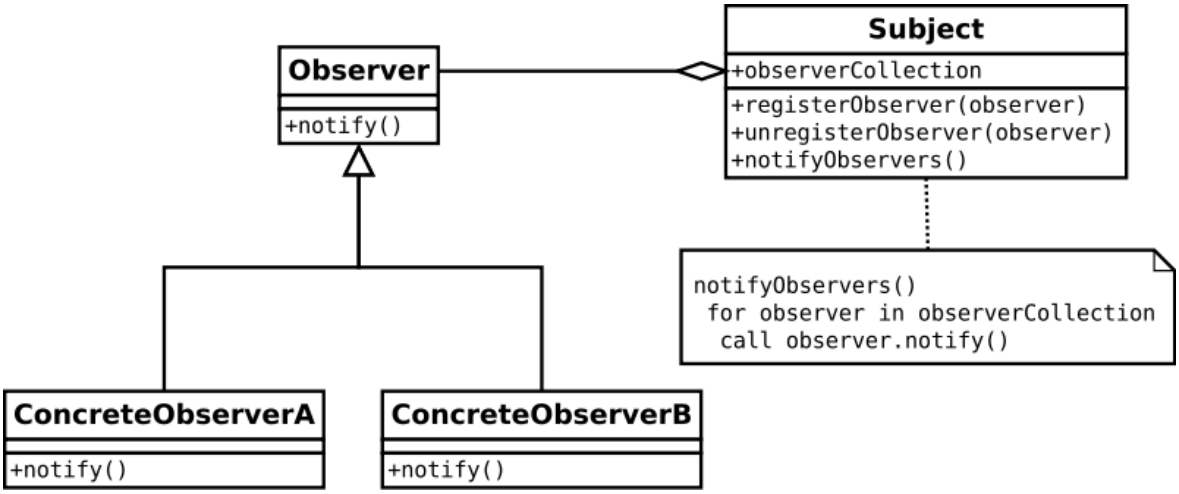
Χρησιμοποιούμε το Πρότυπο Παρατηρητής στις παρακάτω περιπτώσεις:

- Όταν μία ιδεατή κλάση έχει δύο όψεις, οι οποίες η μία εξαρτάται από την

άλλη. Η ενθυλάκωση αυτών των όψεων σε ξεχωριστά αντικείμενα επιτρέπει την διαφοροποίηση τους και χρήση της μίας ανεξάρτητα από την άλλη.

- Όταν μία αλλαγή σε ένα αντικείμενο απαιτεί αλλαγές σε κάποια άλλα και δεν είναι γνωστό πόσα αντικείμενα πρέπει να αλλάξουν
- Όταν ένα αντικείμενο πρέπει να είναι ικανό να ενημερώσει άλλα αντικείμενα χωρίς να κάνει υποθέσεις για το ποια είναι αυτά τα αντικείμενα. Δεν πρέπει δηλαδή αυτά τα αντικείμενα να έχουν υψηλή σύμπλεξη (high couple).

Διάγραμμα UML:



Σχήμα 14: Διάγραμμα UML προτύπου observer

Σχετικά Πρότυπα:

- Πρότυπο Μεσολαβητής (Mediator): Μέσω της ε3νθυλάκωσης πολύπλοκων μηνυμάτων κατάστασης μεταξύ αντικειμένων ο αποστολέας τους ενεργεί ως Μεσολαβητής.
- Πρότυπο Μοναδικότητας (Singleton): Ο αποστολέας των μηνυμάτων μπορεί να χρησιμοποιήσει το πρότυπο μοναδικότητας για να γίνει μοναδικός και προσβάσιμος από όλο το σύστημα.

ΥΠΟΚΕΦΑΛΑΙΟ 4.8 Πρότυπο Κατάστασης, Behavioral Pattern – State

Το πρόβλημα:

Να επιτραπεί σε ένα αντικείμενο να τροποποιήσει την συμπεριφορά του όταν μία εσωτερική του κατάσταση αλλάξει. Το αντικείμενο πρέπει να εμφανίζει ότι άλλαξε την κλάση του.

Επίσης γνωστό ως:

Αντικείμενα για Καταστάσεις (Objects for States)

Σκοπός:

Έστω μια κλάση TCPConnection που αναπαριστά μια σύνδεση δικτύου. Το αντικείμενο της κλάσης TCPConnection μπορεί να έχει μία από πολλές διαφορετικές καταστάσεις: Αποκαταστημένη (σύνδεση), Αναμονή, Ολοκληρωμένη. Όταν ένα τέτοιο αντικείμενο λάβει μία αίτηση από ένα άλλο αντικείμενο απαντάει διαφορετικά και ανάλογα με την κατάσταση την οποία βρίσκεται. Για παράδειγμα η απάντηση σε μία αίτηση Open θα είναι διαφορετική αν το αντικείμενο βρίσκεται σε κατάσταση Αποκαταστημένη ή Ολοκληρωμένη.

Το πρότυπο Κατάστασης περιγράφει πως η TCPConnection μπορεί να επιδεικνύει διαφορετική συμπεριφορά κάθε φορά.

Η κεντρική ιδέα του προτύπου είναι να εισάγει μία ιδεατή κλάση έστω την TCPState, η οποία αναπαριστά τις καταστάσεις της σύνδεσης δικτύου. Η κλάση TCPState παρέχει μία διεπαφή κοινή σε όλες τις κλάσεις οι οποίες αναπαριστούν διαφορετικές καταστάσεις. Οι υποκλάσεις της TCPSTate υλοποιούν εξαρτώμενες από την κατάσταση λειτουργίες.

Χρήσεις:

Χρησιμοποιούμε το πρότυπο Κατάστασης σε μία από τις παρακάτω περιπτώσεις

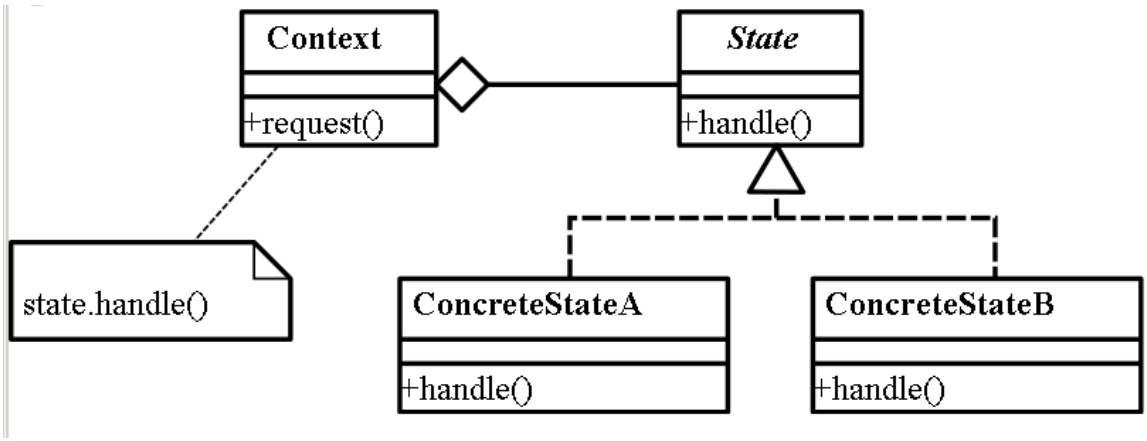
Η συμπεριφορά ενός αντικειμένου εξαρτάται από την κατάσταση του και πρέπει να αλλάξει την συμπεριφορά του την στιγμή της εκτέλεσης (run-time) πάντα εξαρτώμενη από την κατάσταση του.

Κάποιες λειτουργίες έχουν μεγάλες πολυτμηματικές καταστάσεις που εξαρτώνται από την κατάσταση του αντικειμένου. Αυτή η κατάσταση συνήθως αντιπροσωπεύεται από μία ή περισσότερες αριθμητικές σταθερές. Συχνά πολλές λειτουργίες θα περιέχουν αυτή την ίδια δομή κατάστασης. Το πρότυπο Κατάστασης βάζει την κάθε καινούρια διακλάδωση σε μία ξεχωριστή κλάση. Έτσι μπορούμε να δούμε την κατάσταση του αντικειμένου ως ξεχωριστή, τελείως αποκομμένη από άλλα αντικείμενα.

Γνωστές Χρήσεις:

Τα πιο δημοφιλή αλληλεπιδραστικά προγράμματα σχεδίασης παρέχουν «εργαλεία» για άμεση χρήση. Για παράδειγμα ένα εργαλείο επιτρέπει στον χρήστη να «σύρει και να γράψει» μία καινούρια γραμμή. Ένα κουμπί επιλογής επιτρέπει στον χρήστη να διαλέξει σχήμα. Συνήθως υπάρχει μία παλέτα από την οποία ο χρήστης μπορεί να επιλέξει. Το πρόγραμμα αλλάζει κατάσταση κάθε φορά που επιλέγεται διαφορετικό εργαλείο. Όταν έχει επιλεγεί το κουμπί γραμμής μπορούμε να γράψουμε γραμμή, όταν έχει επιλεγεί η παλέτα, επιλέγουμε εργαλείο (δεν γράφουμε δηλαδή γραμμή) κτλ κτλ.

Διάγραμμα UML:



Σχήμα 15: Διάγραμμα UML προτύπου State

Σχετικά Πρότυπα:

Τα αντικείμενα Κατάστασης είναι συχνά και Μοναδικά (Singleton).

ΥΠΟΚΕΦΑΛΑΙΟ 4.9 Πρότυπο Στρατηγικής, Behavioral Pattern – Strategy

Το πρόβλημα:

Ο ορισμός μίας οικογένειας αλγόριθμων, που αφού ενθυλακωθούν να είναι δυνατή η αλλαγή του ενός από τον άλλον. ΤΟ πρότυπο Στρατηγικής επιτρέπει στους αλγόριθμους να διαφέρουν ανεξάρτητα από τους πελάτες που τους χρησιμοποιούν.

Επίσης γνωστό ως:

Πολιτική (Policy)

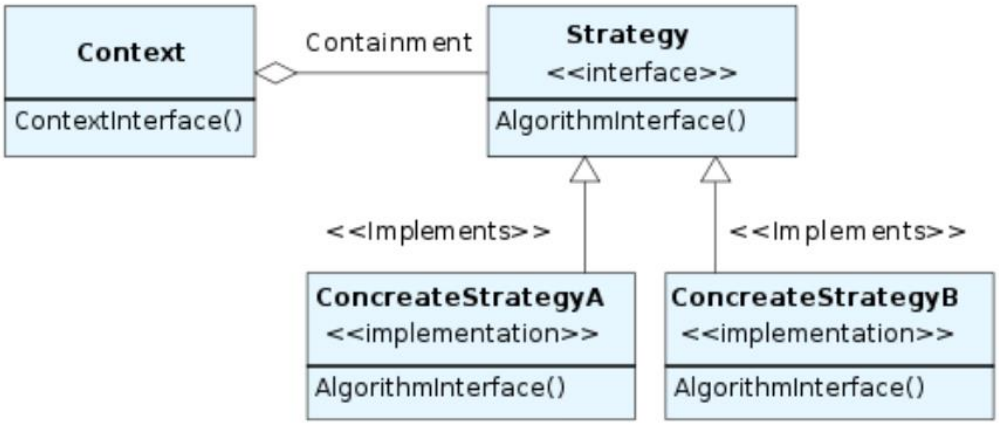
Σκοπός:

Υπάρχουν πολλοί αλγόριθμοι για την μορφοποίηση ενός τρέχοντος κειμένου σε κείμενο σε γραμμές. Η ενσωμάτωση όλων αυτών των αλγόριθμων σε κλάσεις που πρέπει να τους έχουν (χρησιμοποιούν) δεν είναι επιθυμητό για πολλούς λόγους:

- Οι πελάτες εφαρμογές που χρειάζονται την μορφοποίηση με γραμμές στο κείμενο τους θα γίνουν πολύ πιο πολύπλοκες και δυσκολοσυντήρητες αν ενσωματώσουν τον κώδικα αυτών των αλγόριθμων, ειδικά αν αυτοί οι αλγόριθμοι είναι αρκετοί.
- Διαφορετικοί αλγόριθμοι θα είναι οι κατάλληλοι κάθε φορά. Προφανώς δεν χρειάζεται να τους «φορτώνουμε» όλους αν δεν τους χρησιμοποιούμε.
- Είναι δύσκολο να προστεθούν νέοι αλγόριθμοι, ή να τροποποιηθούν οι υπάρχοντες όταν αυτοί θα είναι ενσωματωμένοι σε μία μεγαλύτερη δομή.

Αυτά τα προβλήματα μπορούν να αποφευχθούν με τον ορισμό κλάσεων οι οποίες ενθυλακώνουν διαφορετικούς τέτοιους αλγόριθμους. Οι αλγόριθμοι αυτοί που ενθυλακώθηκαν, ονομάζονται **Στρατηγικές**.

Διάγραμμα UML:



Σχήμα 16: Διάγραμμα UML προτύπου Strategy

Χρήσεις:

Χρησιμοποιούμε το πρότυπο Στρατηγικής όταν:

- Πολλές σχετιζόμενες μεταξύ τους κλάσεις διαφέρουν μόνο ως προς την συμπεριφορά τους. Οι Στρατηγικές προσφέρουν έναν τρόπο να παραμετροποιήσουμε μία κλάση με μια ή παραπάνω συμπεριφορές
- Χρειάζονται διάφορες παραλλαγές ενός αλγόριθμου.
- Ένας αλγόριθμος χρησιμοποιεί δεδομένα τα οποία οι πελάτες-εφαρμογές δεν χρειάζεται (πρέπει) να γνωρίζουν. Χρησιμοποιούμε το πρότυπο Στρατηγικής για να αποφύγουμε έκθεση αυτών των εσωτερικών δεδομένων των αλγόριθμων προς τα έξω
- Μία κλάση καθορίζει πολλαπλές συμπεριφορές και αυτές εμφανίζονται ως πολλαπλές επιλογές υπό συνθήκη στις λειτουργίες της. Αντί για πολλαπλές επιλογές, μπορούν αυτές να μετατεθούν στις δικές τους κλάσεις Στρατηγικής

ΥΠΟΚΕΦΑΛΑΙΟ 4.10 Πρότυπο Επισκέπτης, Behavioral Pattern – Visitor

Το πρόβλημα:

Η αναπαράσταση μίας λειτουργίας που πρέπει να εκτελεστεί πάνω στα στοιχεία μίας δομής. Το πρότυπο Επισκέπτης επιτρέπει να ορίσουμε μία νέα λειτουργία χωρίς να αλλάξουμε τις κλάσεις των στοιχείων πάνω στα οποία θα ενεργήσει.

Σκοπός:

Έστω ένας συντάκτης (Compiler) ο οποίος αναπαριστά προγράμματα ως αφηρημένες δενδροειδείς δομές. Θα χρειαστεί να εκτελέσει λειτουργίες πάνω σε αυτές τις δενδροειδείς δομές για ανάλυση όπως ο έλεγχος ότι όλες οι μεταβλητές είναι ορισμένες. Επίσης θα χρειαστεί να δημιουργήσει κώδικα για αυτό πρέπει να οριστούν λειτουργίες για έλεγχο τύπων, βελτιστοποιήση κώδικα, ανάλυση ροής, έλεγχο για ανάθεση τιμών σε μεταβλητές πριν μπορέσουν να χρησιμοποιηθούν κτλ κτλ. Ακόμη η αρχική δενδροειδής δομή μπορεί να χρησιμοποιηθεί για μορφοποιημένη εκτύπωση, δόμηση προγράμματος, και μέτρηση διαφόρων παραμέτρων στον κώδικα.

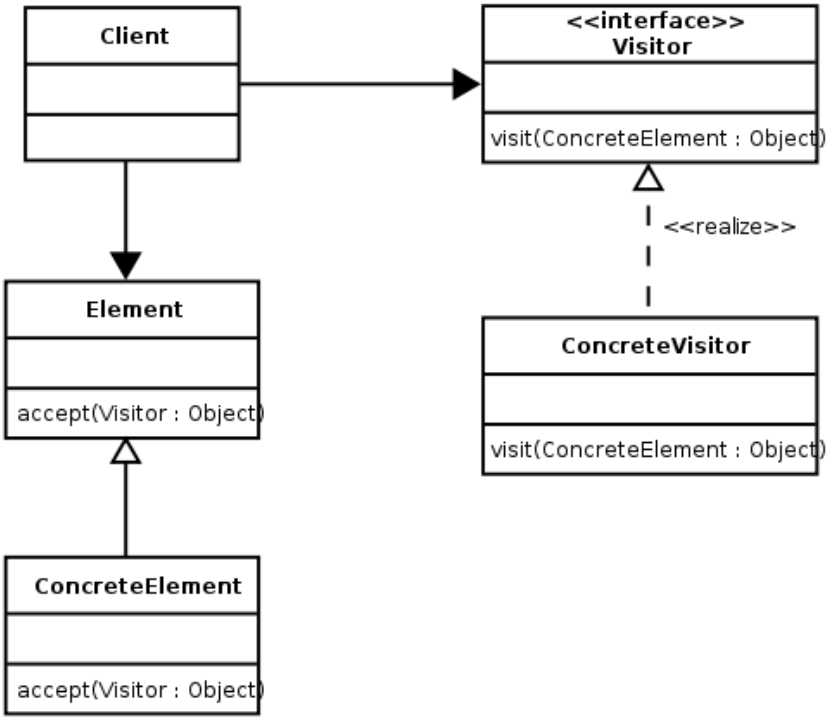
Οι πιο πολλές από αυτές τις λειτουργίες θα χρειαστεί να χειριστούν κόμβους που αναπαριστούν μεταβλητές διαφορετικά από κόμβους που αναπαριστούν φράσεις ολόκληρες ως εκφράσεις (expressions). Έτσι θα υπάρχει μία κλάση για την ανάθεση δηλώσεων, μία κλάση για ανάθεση των μεταβλητών κτλ κτλ. Το τελικό σύνολο των κλάσεων εξαρτάται από την γλώσσα προγραμματισμού και τις δυνατότητες της αλλά δεν αλλάζει σχεδόν καθόλου για μία δεδομένη γλώσσα.

Εφαρμογές

Χρησιμοποιούμε το πρότυπο Επισκέπτη όταν:

- Η δομή ενός αντικειμένου περιέχει πολλές κλάσεις αντικειμένου με διαφορετικές διεπαφές και θέλουμε να επιτελέσουμε λειτουργίες σε αυτά τα αντικείμενα οι οποίες εξαρτώνται από κλάσεις παγιωμένες.
- Πολλές ξεχωριστές και άσχετες μεταξύ τους λειτουργίες χρειάζεται να γίνουν σε αντικείμενα μέσα σε μία δομή αντικειμένων και θέλουμε να αποφύγουμε την «μόλυνση» των κλάσεων του με αυτές τις λειτουργίες. Το πρότυπο επιτρέπει την ομαδοποίηση τέτοιων λειτουργιών σε μία κλάση.
- Οι κλάσεις οι οποίες ορίζουν την δομή του αντικειμένου αλλάζουν σπάνια, αλλά θέλουμε συχνά να ορίσουμε νέες λειτουργίες πάνω σε αυτή την δομή. Αλλάζοντας τις κλάσεις της δομής του αντικειμένου απαιτεί επαναπροσδιορισμό της διεπαφής για όλα τα αντικείμενα Επισκέπτη το οποίο είναι προφανώς κοστοβόρο. Εάν οι κλάσεις της δομής του αντικειμένου αλλάζουν συχνά τότε ίσως είναι καλύτερα να υλοποιηθούν αυτές οι λειτουργίες μέσα σε αυτές τις κλάσεις.

Διάγραμμα UML:



Σχήμα 17: Διάγραμμα UML προτύπου Visitor

Σχετικά Πρότυπα:

Πρότυπο Σύνθεσης: Οι Επισκέπτες μπορούν να χρησιμοποιηθούν για να εφαρμόσουν μία λειτουργία πάνω σε μία δομή αντικειμένων ορισμένη από το πρότυπο Σύνθεσης

Πρότυπο Διερμηνευτής: ο Επισκέπτης μπορεί να πρέπει να κάνει (Αναλάβει) την διερμηνεία

ΚΕΦΑΛΑΙΟ 5

<ΠΡΟΤΥΠΑ ΣΧΕΔΙΑΣΗΣ GRASP>

ΕΙΣΑΓΩΓΗ

Τι πρέπει να περιμένει κανείς από τα πρότυπα σχεδίασης;

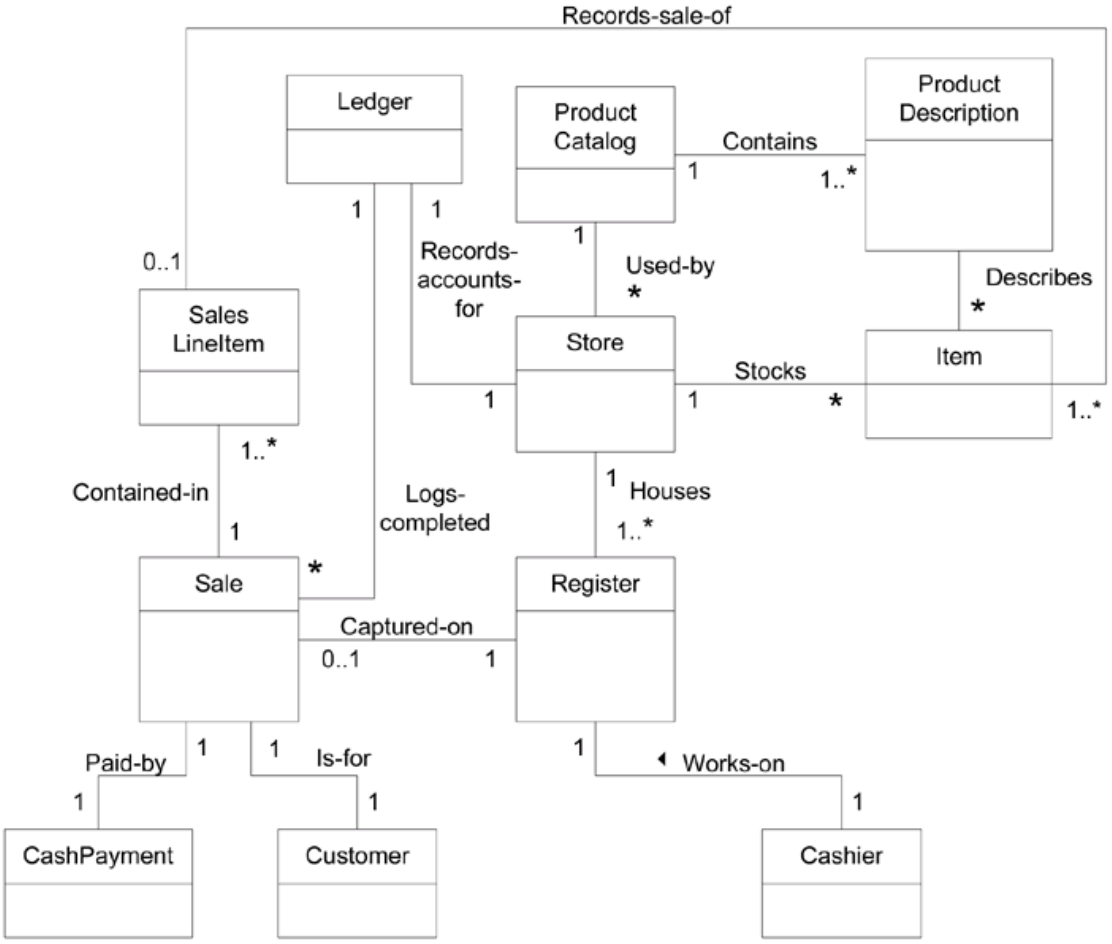
Έρευνες μεταξύ ειδικών και προγραμματιστών για τις γλώσσες προγραμματισμού, έδειξαν ότι η γνώση και η εμπειρία σε αυτές δεν οργανώνεται απλά γύρω από την σύνταξη τους και την γνώση των κανόνων τους, αλλά σε μεγαλύτερες βασικές δομές όπως αλγόριθμοι, δομές δεδομένων και ιδιαίτερες δομές προσαρμοσμένες σε κάθε γλώσσα, καθώς και συγκεκριμένους τρόπους επίτευξης του στόχου. Ένας μηχανικός λογισμικού δεν σκέφτεται τις περισσότερες φορές καινούριους τρόπους να αναπαραστήσει ένα συγκεκριμένο θέμα ή ζητούμενο, αλλά μάλλον ανακαλεί κάθε φορά γνωστούς ήδη σε αυτόν τρόπους και διαδικασίες για να το επιτύχει (Curtis, 1989), (Spohrer & Soloway, 1986).

Έχοντας αυτό κατά νου, μπορούμε να πούμε ότι τα πρότυπα GRASP που θα αναλυθούν διεξοδικά σε αυτό το κεφάλαιο, συμβάλλουν τα μέγιστα προς αυτή την κατεύθυνση με την ευκολία χρήσης τους και την δυνατότητα άμεσης επαναχρησιμοποίησης αλλά και την προσαρμοστικότητα που διαθέτουν .

Εδώ πρέπει να σημειώσουμε, ότι αν και έρευνες αποτελεσματικότητας συγκεκριμένων τέτοιων μεθόδων πολλές φορές δεν αποδεικνύουν το ζητούμενο (το ότι δηλαδή όντως αποτελούν εργαλεία αποτελεσματικότητας και λειτουργικότητας) όπως στην εργασία αυτή (Deligiannis, Sfetsos, & Chatzigeorgiou, 2008), παρόλα αυτά είναι γενικώς αποδεκτό ότι οι μέθοδοι αυτές προσφέρουν ωφέλιμα εργαλεία στον μηχανικό λογισμικού αλλά σίγουρα δεν είναι η πανάκεια, ή το φοβερό όπλο που θα αυξήσει κατακόρυφα την παραγωγικότητα ενός προγραμματιστή, κάτι που ούτως ή άλλως αμφισβητείται έντονα ότι μπορεί να υπάρξει στο γνωστό βιβλίο του Brooks (Brooks Jr, 1995).

Στην λεπτομερή ανάλυση των σχεδιαστικών προτύπων GRASP που θα ακολουθήσει θα χρησιμοποιηθούν ως οπτικό βοήθημα τα παραδείγματα από την δημιουργία του λογισμικού για ένα POS (Point Of Sales) το NextGen POS, όπως ακριβώς περιγράφεται στο βιβλίο (Craig Larman, 2004). Εδώ παρατίθεται το

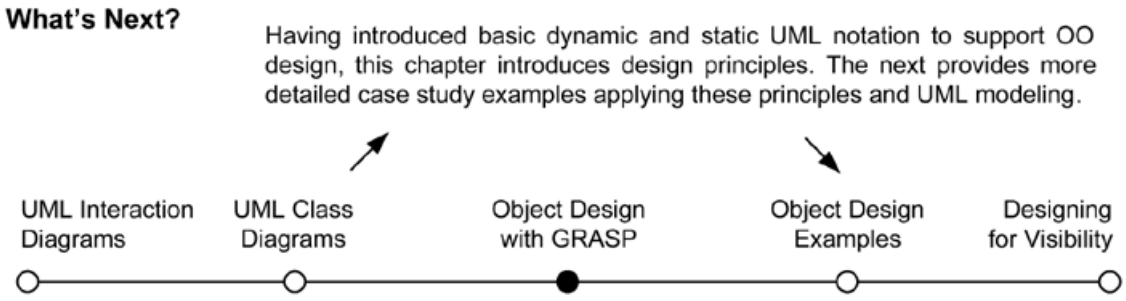
αρχικό λογικό σχεδιάγραμμα του έργου (Project):



Σχήμα 18 "Point Of Sales Example Main UML"

Σε όλα τα υποκεφάλαια που ακολουθούν θα χρησιμοποιηθούν παραδείγματα που αναφέρονται σε αυτό το αρχικό UML γράφημα.

ΥΠΟΚΕΦΑΛΑΙΟ 5.1 Τα Σχεδιαστικά Πρότυπα GRASP



Σχήμα 19 "Η σειρά χρήσης προτύπων GRASP"

Ένας αρκετά δημοφιλής τρόπος σκέψης σχετικά με την δημιουργία και περιγραφή αντικειμένων στο λογισμικό, είναι με όρους αρμοδιοτήτων, ρόλων και συνεργιών (responsibilities, roles, and collaborations) (Craig Larman, 2004).

Αυτό ουσιαστικά αποτελεί μέρος μίας ευρύτερης προσέγγισης στον προγραμματισμό, ονομαζόμενης «Σχεδιασμός οδηγούμενος από τις ευθύνες» responsibility-driven design, ή απλά RDD (Wirfs-Brock & McKean, 2003).

Στην RDD σχεδίαση τα αντικείμενα του λογισμικού έχουν αρμοδιότητες ως προς το τι κάνουν. Η UML ορίζει μία αρμοδιότητα (responsibility) ως «ένα συμβόλαιο ή μία υποχρέωση ενός καταχωριστή (classifier) . Οι αρμοδιότητες αυτές σχετίζονται με τις υποχρεώσεις ή την συμπεριφορά ενός αντικειμένου σχετικά με τον ρόλο του. Ουσιαστικά οι αρμοδιότητες αυτές είναι δύο ειδών: Πράξης (Doing) και Γνώσης (Knowing) (Craig Larman, 2004).

Οι **Αρμοδιότητες Πράξης (Doing)** ενός αντικειμένου περιλαμβάνουν:

- Κάτι που κάνει το ίδιο, όπως να δημιουργεί ένα αντικείμενο, ή να κάνει έναν υπολογισμό
- Να εκκινεί ενέργειες σε άλλα αντικείμενα
- Να ελέγχει και να συντονίζει δράσεις άλλων αντικειμένων

Οι **Αρμοδιότητες Γνώσης (Knowing)** ενός αντικειμένου περιλαμβάνουν:

- Γνώση των ενθυλακωμένων ιδιωτικών δεδομένων (encapsulated private data)
- Γνώση σχετικών (σχετιζόμενων) αντικειμένων

- Γνώση πραγμάτων τα οποία μπορεί να εξάγει ή να υπολογίσει

Οι Αρμοδιότητες ανατίθενται στις κλάσεις των αντικειμένων κατά την διάρκεια της σχεδίασης αυτών των αντικειμένων.

Η μετάφραση αυτών των Αρμοδιοτήτων σε κλάσεις και μεθόδους επηρεάζεται από την διασπορά των ευθυνών σε αυτές τις κλάσεις και τις μεθόδους. Μεγάλες (πολύπλοκες) Αρμοδιότητες απαιτούν εκατοντάδες κλάσεις και μεθόδους ενώ μικρές (πολύ συγκεκριμένες) Αρμοδιότητες μπορεί να απαιτούν μόνο μία κλάση ή μέθοδο (Schmidt, 1995)..

Η Αρμοδιότητα δεν είναι το ίδιο πράγμα με την μέθοδο. Είναι ένα αφαιρετικό κατασκεύασμα, το οποίο όμως η μέθοδος οφείλει να ικανοποιεί.

Η RDD επίσης εμπεριέχει την έννοια της συνέργιας (collaboration). Οι Αρμοδιότητες υλοποιούνται ως μέθοδοι οι οποίες είτε ενεργούν αυτόνομα, είτε σε συνεργασία με άλλες μεθόδους και αντικείμενα.

Το πρότυπο GRASP ονοματίζει και περιγράφει αρκετές βασικές αρχές ανάθεσης αρμοδιοτήτων και για αυτό είναι χρήσιμη η πρότερη γνώση της RDD, σύμφωνα με τον Larman (Craig Larman, 2004).

ΥΠΟΚΕΦΑΛΑΙΟ 5.2 Είδη GRASP Προτύπων

GRASP, General Responsibility Assignment Software Pattern (or Principles)

Τα πρότυπα GRASP είναι ουσιαστικά ένα βοήθημα εκμάθησης για την αντικειμενοστρεφή προγραμματισμό με Αρμοδιότητες όπως περιγράφηκαν στο προηγούμενο κεφάλαιο.

Το GRASP αποτελείται από οδηγίες ανάθεσης ευθυνών σε κλάσεις και αντικείμενα στον αντικειμενοστρεφή προγραμματισμό (Craig Larman, 2004).

Μπορούμε να ονοματίσουμε και να επεξηγήσουμε τις λεπτομερείς αρχές και λογική που διέπουν τις βασικές αρχές δημιουργίας αντικειμένων βασισμένα στο GRASP, απλώς με το να αναθέσουμε Αρμοδιότητες στα αντικείμενα. Οι αρχές και τα πρότυπα GRASP που θα αναλυθούν παρακάτω, είναι ένα βοήθημα εκμάθησης και κατανόησης των βασικών τεχνικών δημιουργίας αντικειμένων στο λογισμικό καθώς και εφαρμογής αυτών των τεχνικών δημιουργίας αντικειμένων με έναν μεθοδικό, λογικό και εξηγήσιμο τρόπο. Αυτή η προσέγγιση είναι βασισμένη στην λογική της δημιουργίας αντικειμένων μέσω της χρήσης προτύπων που αναθέτουν αρμοδιότητες στα αντικείμενα (Craig Larman, 2004).

Πρέπει να τονιστεί επίσης ότι ενώ το GRASP ως γενική σχεδιαστική αρχή πρέπει να γίνει κατανοητή ευθύς εξαρχής από τον προγραμματιστή, τα επιμέρους πρότυπα που περιγράφει το GRASP για κάθε ιδιαίτερη περίπτωση χρήσης δεν είναι απαραίτητο να είναι γνωστά στον προγραμματιστή εκ των προτέρων, αφού εύκολα μπορεί να αποκτήσει πρόσβαση σε αυτά όταν γνωρίζει την ύπαρξη τους

Υπάρχουν **εννέα πρότυπα GRASP** (Craig Larman, 2004)**:**

1. **Δημιουργικό Πρότυπο, Creator**
2. **Πληροφοριακό Πρότυπο, (Information) Expert**
3. **Χαμηλής Σύζευξης, Low Coupling**
4. **Ελέγχου (Ή Πρότυπο Ελεγκτής)Controller**

5. **Υψηλής Συνοχής, High Cohesion**
6. **Πολυμορφισμού, Polymorphism**
7. **Κατασκευής, Pure Fabrication**
8. **Έμμεσο Πρότυπο Indirection**
9. **Προστατευμένων Παραλλαγών, Protected Variations (Don’t Talk to Strangers)**

Αυτά τα πρότυπα (ιδίως τα πρώτα πέντε) απαντούν σε κάποιο πρόβλημα λογισμικού και σχεδόν σε κάθε περίπτωση αυτά ακριβώς τα προβλήματα είναι συνηθισμένα και απατούνται σε κάθε μεγάλο ή μικρό Project λογισμικού σήμερα όπου εφαρμόζεται αντικειμενοστρεφής προγραμματισμός.

Αυτές οι τεχνικές και μέθοδοι δεν δημιουργήθηκαν για να προσφέρουν νέους τρόπους εργασία και προσέγγισης, αλλά για να θέσουν μέσα σε πλαίσια και συγκεκριμένες κατευθύνσεις (στάνταρτς) παλιές και δοκιμασμένες επιτυχώς προγραμματιστικές αρχές στον αντικειμενοστρεφή σχεδιασμό (C. Larman, 2002).

Ο Larman στο ίδιο βιβλίο που αναφέρεται στην προηγούμενη παράγραφο, συνεχίζει:

«…Το ουσιώδες σχεδιαστικό εργαλείο στον σχεδιασμό λογισμικού είναι ένα μυαλό καλά εκπαιδευμένο στις προγραμματιστικές αρχές σχεδιασμού. Δεν είναι ούτε η UML, ούτε κανενός άλλου είδους τεχνολογία…».

Για αυτό και το GRASP θεωρείται ένα θεωρητικό εργαλείο εξάσκησης του νου, ένα μαθησιακό βοήθημα στον αντικειμενοστρεφή προγραμματισμό.

Στα υποκεφάλαια που ακολουθούν θα αναλυθούν λεπτομερώς και τα εννέα επιμέρους πρότυπα GRASP.

ΥΠΟΚΕΦΑΛΑΙΟ 5.3 Πρότυπο Δημιουργός GRASP Pattern –Creator

Το πρόβλημα:

Ποιος είναι υπεύθυνος για την δημιουργία ενός καινούριου στιγμιότυπου μίας κλάσης;

Η δημιουργία αντικειμένων είναι ίσως η συνηθέστερη διαδικασία σε ένα αντικειμενοστρεφές σύστημα. Συνεπώς είναι χρήσιμο να υπάρχει μία γενική αρχή ανάθεσης αρμοδιοτήτων δημιουργίας αντικειμένων. Εάν αυτή η αρμοδιότητα ανατεθεί σωστά, η σχεδίαση του συστήματος μπορεί να οδηγήσει σε μη σύγχυση αρμοδιοτήτων, αυξημένη διαύγεια, ενθυλάκωση και αυξημένη επαναχρησιμοποίηση.

Η λύση:

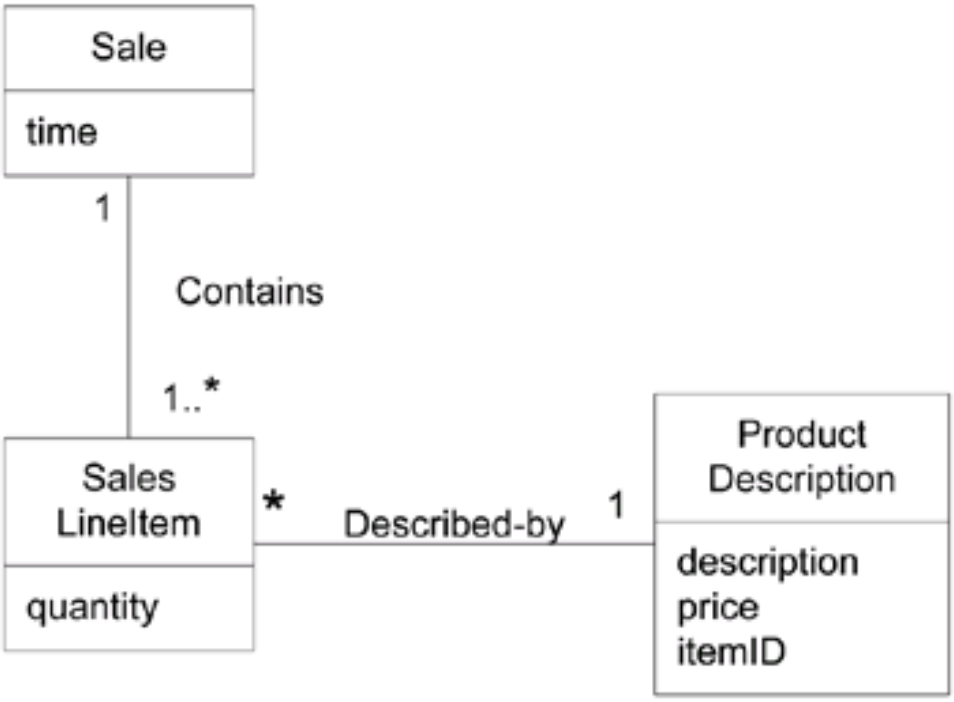
Ανατίθεται στην κλάση Β η αρμοδιότητα δημιουργίας στιγμιότυπων της κλάσης Α, εάν μία από τις παρακάτω συνθήκες είναι αληθής (όσο περισσότερες, τόσο το καλύτερο):

- Η Β εμπεριέχει ή συνενώνεται με την Α
- Η Β παρακολουθεί την Α
- Η Β χρησιμοποιεί στενά (επανειλημμένα) την Α
- Η Β περιέχει τα δεδομένα αρχικοποίησης της Α, τα οποία περνά ως παραμέτρους στην Α όταν την δημιουργεί (το στιγμιότυπο). Τότε η Β ανήκει στο πρότυπο Expert (θα εξεταστεί σε επόμενο κεφάλαιο) με αρμοδιότητα την δημιουργία της Α
- Η Β είναι δημιουργός των αντικειμένων Α

Εάν μία ή παραπάνω συνθήκες από τις παραπάνω ισχύουν, τότε συνήθως η Β θα πρέπει ή να συμπληρώνει, ή να περιέχει την Α.

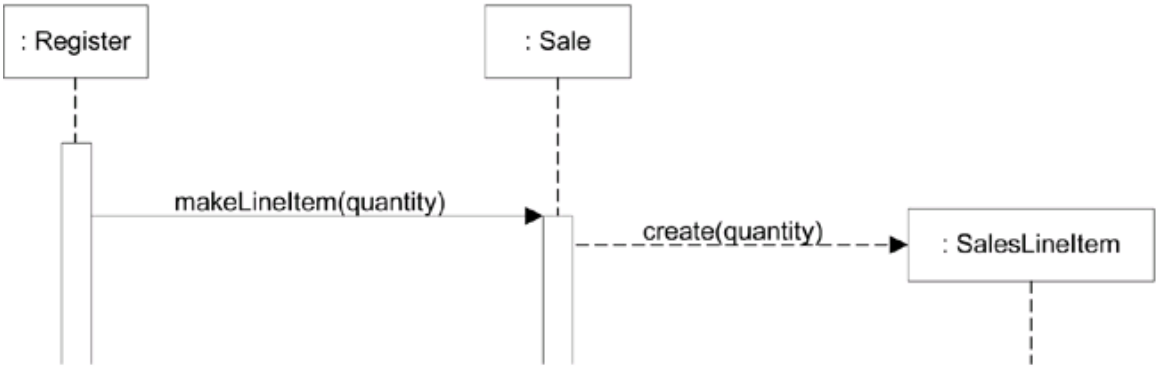
Παράδειγμα:

Στην ήδη γνωστή από το σχήμα 2 ανάλυση UML ενός POS, επικεντρώνουμε το παράδειγμα στο ερώτημα «ποιος είναι υπεύθυνος για την δημιουργία του στιγμιότυπου της κλάσης SalesLineItem ; Σύμφωνα με το πρότυπο Creator θα πρέπει να ψάξουμε για μία κλάση η οποία εμπεριέχει, παρακολουθεί κτλ τα στιγμιότυπα της SalesLineItem.



Σχήμα 20: «Παράδειγμα προτύπου Creator στο POS

Σύμφωνα με το σχήμα 20, επειδή μία Sale περιέχει (ή καλύτερα ακόμη συναθροίζει) πολλά αντικείμενα SalesLineItem, το πρότυπο Creator υποδεικνύει ότι η Sale είναι ένας καλός υποψήφιος να έχει την αρμοδιότητα της δημιουργίας των στιγμιότυπων της SalesLineItem. Αυτό οδηγεί στον σχεδιασμό του σχήματος 21:



Σχήμα 21: Σχεδίαση του POS Sales βάσει του προτύπου Creator

Αυτή η ανάθεση αρμοδιοτήτων απαιτεί τον ορισμό μίας μεθόδου makeLineItem μέσα στην Sale. Η απόφαση αυτή πάρθηκε αφού πρώτα έγινε ένα πρόχειρο διάγραμμα καταγραφής αρμοδιοτήτων (δηλαδή διάγραμμα UML). Το κομμάτι της κλάσης όπου υλοποιείται η μέθοδος μπορεί να συγκεντρώσει τα αποτελέσματα της ανάθεσης αυτής ως μία μέθοδος.

Συζήτηση

Το δημιουργικό πρότυπο Creator ορίζει την ανάθεση των αρμοδιοτήτων σχετικά με την δημιουργία αντικειμένων, μία πολύ συχνή διαδικασία. Ο βασικός σκοπός-στόχος του Creator είναι να βρει έναν δημιουργό ο οποίος μπορεί εύκολα να συνδεθεί με το δημιουργημένο (στιγμιότυπο ενός) αντικείμενο κάθε φορά. Η σωστή επιλογή οδηγεί σε χαμηλές ζεύξεις μεταξύ κλάσεων.

Τελικά το δημιουργικό πρότυπο Creator υποδεικνύει ότι η κλάση που εμπεριέχει το αντικείμενο είναι ένας πολύ καλός υποψήφιος δημιουργός αυτού του αντικειμένου. Προφανώς και αυτό μπορεί να χρησιμοποιηθεί ως βασική αρχή.

Μερικές φορές ο δημιουργός μπορεί εύκολα να αναγνωριστεί απλώς με την ανάγνωση της κλάσης που εμπεριέχει τα δεδομένα αρχικοποίησης που θα δεχθεί το αντικείμενο υπό δημιουργία. Στο σχήμα 21 είναι εύκολο να αντιληφθούμε ότι ένα στιγμιότυπο Payment όταν δημιουργηθεί χρειάζεται το Sale total, το οποίο προφανώς το γνωρίζει η Sale, η οποία πολύ εύκολα μπορεί να το περάσει ως παράμετρο στον constructor του αντικειμένου σε μία υλοποίηση Java.

Αντιλογίες

Συχνά, η δημιουργία απαιτεί αυξημένη πολυπλοκότητα όπως όταν χρησιμοποιούνται ανακυκλωμένα στιγμιότυπα όσον αφορά στις επιδόσεις, ειδικά όταν ένα στιγμιότυπο βασίζει την δημιουργία του στην είσοδο εξωτερικών δεδομένων. Σε τέτοιες περιπτώσεις συνίσταται η μεταβίβαση της δημιουργίας αντικειμένων σε μία βοηθητική κλάση η οποία ονομάζεται Concrete Factory ή Abstract Factory (Gamma κ.ά., 1995).

Ωφέλειες

Με την χρήση του προτύπου το υπό κατασκευή πρόγραμμα συμμορφώνεται και με το πρότυπο Χαμηλής Σύζευξης Low Coupling, δηλαδή χαμηλές αλληλεξαρτήσεις, κάτι που σημαίνει ανεξαρτησία και μεγάλη δυνατότητα επαναχρησιμοποίησης με ελάχιστο κόπο.

Σχετικά Πρότυπα

- Χαμηλής Σύζευξης , Low Coupling

ΥΠΟΚΕΦΑΛΑΙΟ 5.4 Πρότυπο Πληροφορίας GRASP Pattern – (Information) Expert

Το πρόβλημα:

Ποιά είναι η γενική αρχή ανάθεσης αρμοδιοτήτων στα αντικείμενα;

Ένα μοντέλο σχεδίασης μπορεί να ορίζει εκατοντάδες ή χιλιάδες κλάσεις και μία εφαρμογή (λογισμικού) μπορεί να απαιτεί εκατοντάδες από προϋποθέσεις πριν εκτελεστεί. Κατά την διάρκεια του σχεδιασμού αντικειμένων, όταν ορίζονται οι αλληλεπιδράσεις μεταξύ τους, λαμβάνονται και οι απαραίτητες αποφάσεις ανάθεσης αρμοδιοτήτων στις κλάσεις. Εάν η ανάθεση είναι σωστή, ένα σύστημα είναι ευκολότερο να γίνει κατανοητό, να συντηρηθεί και να επεκταθεί, και συνεπακόλουθα γίνεται ευκολότερη επαναχρησιμοποίηση των επιμέρους συστατικών σε όποιες μελλοντικές εφαρμογές.

Η λύση:

Ανάθεση αρμοδιοτήτων στον «ειδικό», στην κλάση δηλαδή η οποία κατέχει την απαραίτητη πληροφορία για την εκτέλεση της συγκεκριμένης αρμοδιότητας.

Παράδειγμα:

Στο γνωστό παράδειγμα του NextGen POS, κάποια κλάση χρειάζεται να γνωρίζει το τελικό (γενικό) σύνολο πωλήσεων.

Η ερώτηση αυτή μπορεί να μετατραπεί ως εξής:

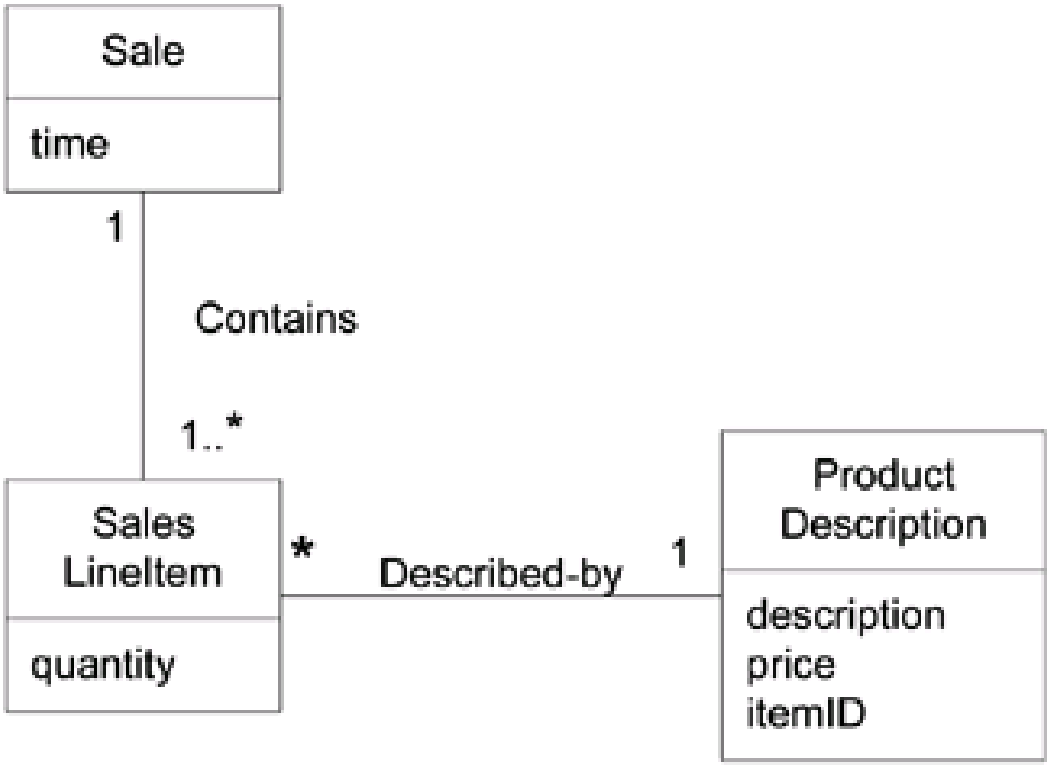
«Ποιος (ποιο κομμάτι του λογισμικού) θα πρέπει να είναι αρμόδιος να γνωρίζει το τελικό σύνολο των πωλήσεων;»

Σύμφωνα με το πρότυπο αυτό, πρέπει να αναζητήσουμε την κλάση η οποία (ήδη) κατέχει την πληροφορία αυτή.

Βέβαια σε μία τέτοια ερώτηση κάποιος θα αναρωτηθεί αν η απάντηση βρίσκεται στο σχεδιάγραμμα UML ή στην υλοποίηση του λογισμικού;

Η απάντηση είναι εύκολη: Αν υπάρχουν (αν φαίνονται) κάποιες σχετικές κλάσεις στο σχεδιάγραμμα UML, τότε εκεί βρίσκεται και η απάντηση. Αλλιώς συνήθως μέσα στο διάγραμμα συσχετίσεων των πραγματικών οντοτήτων (του πραγματικού κόσμου) υπάρχει η βάση στην οποία μπορεί να στηριχθεί η δημιουργία των απαραίτητων νέων κλάσεων.

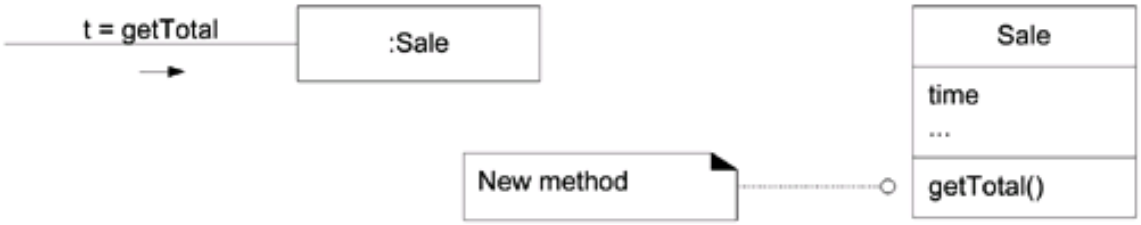
Για παράδειγμα εφόσον προφανώς υπάρχει μία οντότητα Sale στον πραγματικό κόσμο, είναι εύκολο να δημιουργήσουμε στο διάγραμμα UML μία κλάση Sale και να της αναθέσουμε την αρμοδιότητα γνώσης του συνόλου πωλήσεων μέσω μίας μεθόδου που θα ονομάζεται getTotal. Αυτή η προσέγγιση παρέχει ένα μικρό χάσμα απεικόνισης (ή καλύτερα αντιστοίχησης) μεταξύ των οντοτήτων σε ένα έργο λογισμικού και στον πραγματικό κόσμο.



Σχήμα 22: «Οι σχέσεις του Sale»

Στο παράδειγμα του σχήματος 22 , το ερώτημα είναι αν χρειάζεται να ορίσουμε το τελικό σύνολο. Πρέπει να γνωρίζουμε την πληροφορία για κάθε στιγμιότυπο του SalesLineItem και το σύνολο των επιμέρους subtotals. Ένα στιγμιότυπο Sale εμπεριέχει αυτή την πληροφορία και ως εκ τούτου σύμφωνα με το πρότυπο Expert το Sale είναι μία κλάση η οποία μπορεί να αναλάβει αυτή την ευθύνη. Είναι ένας expert (ειδικός) για αυτήν την δουλειά.

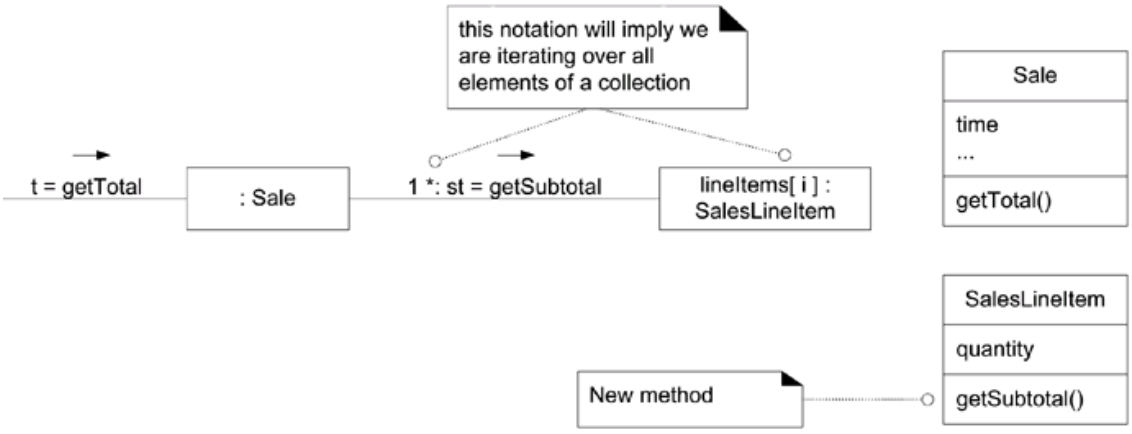
Συνήθως αυτές οι ερωτήσεις ανακύπτουν αλλά και απαντιούνται κατά την διάρκεια της δημιουργίας των διαγραμμάτων UML. Η απάντηση στην παραπάνω ερώτηση, μπορεί εν μέρει να απαντηθεί με το διάγραμμα του σχήματος 23:



Σχήμα 23: «Αρχικές Επιδράσεις του Sale και εύρεση του Total»

Η επόμενη ερώτηση είναι, ποια ακόμη πληροφορία χρειαζόμαστε για να βρούμε το υποσύνολο για το Line item subtotal ; Η πληροφορία αυτή περιέχεται στα SalesLineItem.quantity και στο ProductDescription.price. Το SalesLineItem γνωρίζει την ποσότητα που συνδέεται με το κάθε ProductDescription. Άρα το SalesLineItem πρέπει να ορίζει το μερικό σύνολο (υποσύνολο). Είναι ο «ειδικός» (expert).

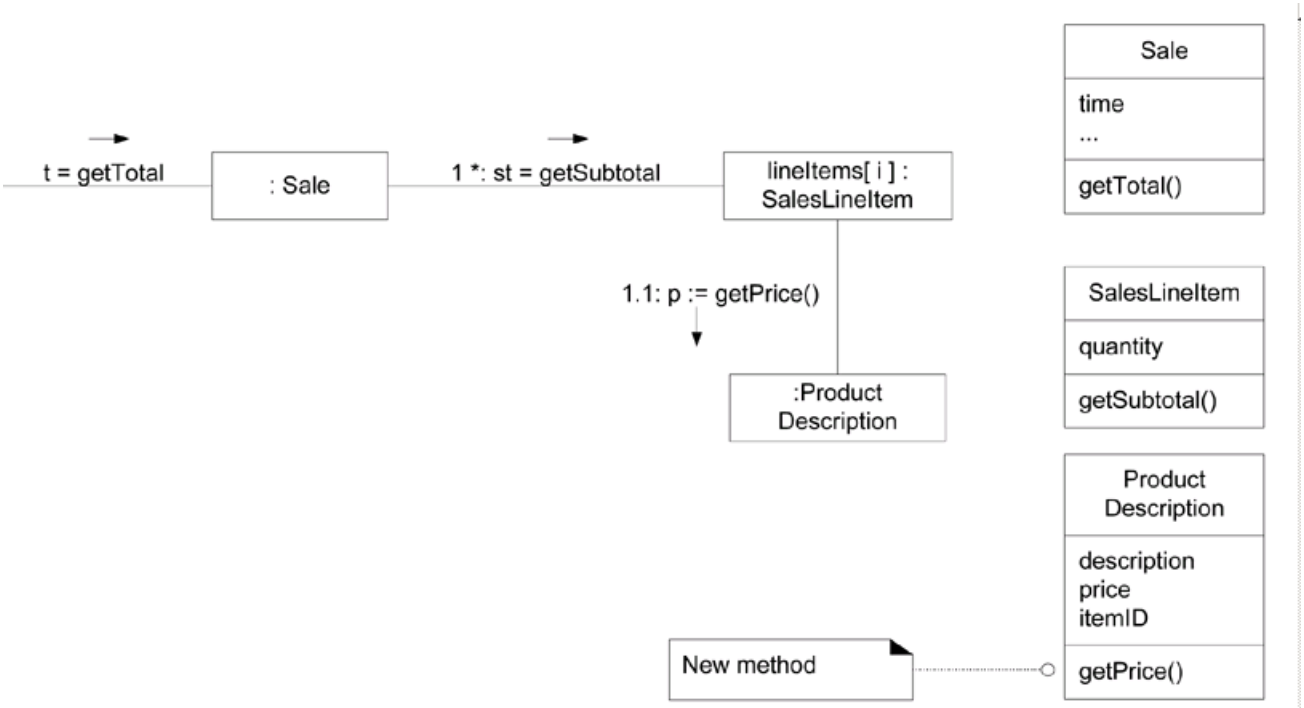
Εάν αυτό πρέπει να εκφραστεί με ένα διάγραμμα αλληλεπίδρασης, σημαίνει ότι το Sale θα πρέπει να στέλνει μηνύματα getSubtotal σε κάθε ένα από τα SalesLineItems και να αθροίσει τα αποτελέσματα. Η σχεδίαση αυτή φαίνεται στο επόμενο σχήμα:



Σχήμα 24: «Υπολογισμός του συνόλου πωλήσεων (Sale Total)»

Το αντικείμενο SalesLineItem για να υπολογίσει το subtotal πρέπει να γνωρίζει την τιμή του προϊόντος (product price).

Εφόσον το ProductDescription είναι ο information expert όσον αφορά την τιμή (price) θα πρέπει το SalesLineItem να του στείλει ένα ερώτημα για την τιμή (product price).



Σχήμα 25: Υπολογισμός της συνολικής τιμής (Sale Total)

Τελικά για να γίνει γνωστό το τελικό σύνολο πωλήσεων ανατέθηκαν τρείς αρμοδιότητες σε τρείς κλάσεις αντικείμενων όπως ο πίνακας που ακολουθεί:

Σχεδιαστική Κλάση	Αρμοδιότητα
Sale	Γνωρίζει το σύνολο πωλήσεων (sale total)
SalesLineItem	Γνωρίζει το υποσύνολο για κάθε στιγμιότυπο (line item subtotal)
ProductDescription	Γνωρίζει την τιμή του κάθε προϊόντος (product price)

Σχήμα 26: Ανάθεση Αρμοδιοτήτων Εύρεσης του Τελικού Συνόλου Πωλήσεων

Η αρχή ανάθεσης αρμοδιοτήτων έγινε βάσει της αρχής του προτύπου Information Expert, αναθέτοντας δηλαδή την αρμοδιότητα στο αντικείμενο που είχε την πληροφορία.

Συζήτηση

Το Πληροφοριακό Πρότυπο Expert χρησιμοποιείται συχνά στην ανάθεση αρμοδιοτήτων. Είναι βασική κατευθυντήρια αρχή η οποία χρησιμοποιείται συνεχώς στην σχεδίαση λογισμικού. Το συγκεκριμένο πρότυπο δεν είναι κάτι συγκεχυμένο ή θεωρητικό. Ουσιαστικά εκφράζει την προφανή άποψη ότι τα αντικείμενα κάνουν πράγματα σχετικά με την πληροφορία που κατέχουν ή γνωρίζουν.

Επειδή η εκτέλεση μίας αρμοδιότητας προϋποθέτει την γνώση πληροφορίας που συνήθως είναι διάσπαρτη ανάμεσα σε διάφορες κλάσεις και αντικείμενα, αυτό σημαίνει ότι πολλοί experts που κατέχουν τμήματα αυτής της γνώσης θα πρέπει να συνεργαστούν σε ένα κοινό στόχο.

Το πρότυπο αυτό συνήθως οδηγεί σε σχεδιασμό ο οποίος αντιστοιχεί στον αντίστοιχο του πραγματικού κόσμου, παρόλο που για παράδειγμα στον

πραγματικό κόσμο χωρίς την χρήση ηλεκτρομαγνητικών βοηθημάτων μία πώληση δεν μπορεί μόνη της να πει το τελικό σύνολο πωλήσεων, ενώ αντίθετα σε ένα λογισμικό η κλάση Sale μπορεί να βρει και να επιστρέψει το σύνολο των πωλήσεων.

Το Πληροφοριακό Πρότυπο Expert όπως πολλά πράγματα στην τεχνολογία λογισμικού έχει ευθεία αναλογία και στο πραγματικό κόσμο. Συχνά αναθέτουμε αρμοδιότητες σε άτομα που έχουν την απαραίτητη πληροφορία για να διεκπεραιώσουν την εργασία. Για παράδειγμα σε μία πραγματική επιχείρηση, το άτομο που θα συντάξει έναν ισολογισμό, είναι προφανές ότι πρέπει να έχει πλήρη πρόσβαση στα συνολικά δεδομένα πωλήσεων. Με την ίδια λογική της σύμπραξης πολλών επιμέρους κλάσεων για την συγκέντρωση των απαραίτητων ποσών, στον πραγματικό κόσμο, αρκετοί άνθρωποι πρέπει να συνεργαστούν για να συγκεντρωθούν τα απαραίτητα οικονομικά στοιχεία. Τελικά ο αρχιλογιστής της εταιρίας μπορεί να συγκεντρώσει τα απαραίτητα στοιχεία ρωτώντας τους υπαλλήλους του λογιστηρίου που είναι αρμόδιοι για κάθε οικονομικό τομέα.

Αντιλογίες

Σε κάποιες περιπτώσεις η λύση που προτείνει το πρότυπο Expert είναι ανεπιθύμητη, συνήθως επειδή φέρνει προβλήματα συνέργιας και συνοχής (τα οποία αντιστοιχούν σε πρότυπα που θα συζητηθούν παρακάτω). Για παράδειγμα ποιος θα πρέπει να είναι υπεύθυνος για την αποστολή του ποσού μίας πώλησης στην βάση δεδομένων; Το πρότυπο εύκολα υποδεικνύει την κλάση Sale ως κατάλληλη. Αυτό όμως οδηγεί σε προβλήματα που έχουν να κάνουν με συνέργιες, υψηλή διάχυση και αλληλοεπικάλυψη. Για παράδειγμα η κλάση Sale πρέπει να περιέχει απαραίτητα τμήματα κώδικα σχετικά με βάσεις δεδομένων όμως SQL και αναπόφευκτα συναρτήσεις από την JDBC (Java Database Connectivity) αν φυσικά το πρόγραμμα είναι γραμμένο σε Java. Τώρα πια η κλάση που δεν κάνει μόνο sales (πωλήσεις) αλλά επιφορτίζεται και με άλλες αρμοδιότητες. Έτσι οι συσχετίσεις (coupling) της κλάσης αυξάνονται αφού πρέπει να συσχετίζεται όχι μόνο με τις «όμορες» κλάσεις σχετικές με πωλήσεις, αλλά και με κλάσεις σχετικές

με βάσεις δεδομένων.

Φυσικά αυτό οδηγεί σε μία ευθεία αντίφαση με την γενική προγραμματιστική αρχή που λέει ότι θα πρέπει να ξεχωρίζουμε τις συναλλαγές (transactions) με την βάση δεδομένων σε κλάσεις αφιερωμένες σε αυτόν τον σκοπό και μόνο.

Στο τέλος λοιπόν, αν και το πρότυπο υποδεικνύει την ευθεία σύζευξη της κλάσης Sale με την βάση δεδομένων, τελικά λόγω της αντίθεσης με άλλα πρότυπα (cohesion, coupling) ίσως αυτό να μην είναι εφικτό.

Ωφέλειες

- Η ενθυλάκωση της πληροφορίας διατηρείται αφού τα αντικείμενα χρησιμοποιούν τις δικές τους πληροφορίες για να εκτελέσουν τις αποστολές τους. Αυτό συνήθως οδηγεί σε χαμηλή σύζευξη (low coupling), το οποίο με την σειρά του οδηγεί σε συμπαγή και ευκολοσυντήρητα συστήματα. Άλλωστε το πρότυπο Χαμηλής Σύζευξης (Coupling) είναι ένα πρότυπο που θα συζητηθεί διεξοδικά παρακάτω.
- Η συμπεριφορά αυτή επειδή διαχέεται σε όλες τις κλάσεις του συστήματος οδηγεί σε «ελαφρύτερες» και έτσι ευκολότερο να κατανοηθούν κλάσεις. Συνήθως οδηγεί και σε υψηλή συνοχή (cohesion) το οποίο είναι άλλο ένα πρότυπο που θα συζητηθεί παρακάτω.

Σχετικά Πρότυπα

- Χαμηλής Σύζευξης , Low Coupling
- Υψηλής Συνοχής , High Cohesion

Επίσης γνωστό ως (Συναφές με το)

Παρατίθενται αμετάφραστα τα σχετικά πεδία για να είναι σε ευθεία σύγκριση με τα ονόματα των προτύπων που δεν μεταφράστηκαν:

"Place responsibilities with data," "That which knows, does," "Do It Myself,"

"Put Services with the Attributes They Work On."

ΥΠΟΚΕΦΑΛΑΙΟ 5.5 Πρότυπο Χαμηλής Σύζευξης GRASP Pattern – Low Coupling

Το πρόβλημα:

Πώς να επιτευχθεί η χαμηλή αλληλεξάρτηση, χαμηλός αντίκτυπος αλλαγών και αυξημένη επαναχρησιμοποίηση;

Το Πρότυπο Χαμηλής Σύζευξης (Low Coupling) ή σκέτο Σύζευξης (Coupling) είναι μία μέτρηση του πόσο ένα στοιχείο είναι συνδεδεμένο με, γνωρίζει, ή βασίζεται σε άλλα στοιχεία. Ένα στοιχείο με χαμηλή (ή αδύναμη) σύζευξη coupling δεν εξαρτάται (η συνδέεται) με πολλά άλλα στοιχεία. Το «πολλά» στην συγκεκριμένη περίπτωση είναι σχετικό κάθε φορά. Τα στοιχεία υπό εξέταση περιλαμβάνουν κλάσεις, υποσυστήματα, ολόκληρα συστήματα λογισμικού κτλ.

Μία κλάση με υψηλή σύζευξη coupling εξαρτάται από πολλές άλλες κλάσεις. Τέτοιες κλάσεις πρέπει να αποφεύγονται. Κάποιες από αυτές θα υποφέρουν από τα παρακάτω προβλήματα:

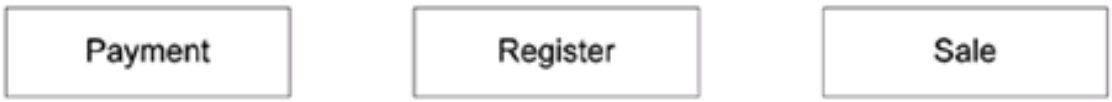
- Επιβολή τοπικών αλλαγών λόγω αλλαγών σε συνδεόμενες (εξαρτώμενες) κλάσεις.
- Δυσκολία κατανόησης της λειτουργίας τους χωρίς την ανάγνωση και των εξαρτώμενων κλάσεων
- Δυσκολία επαναχρησιμοποίησης λόγω της ανάγκης ύπαρξης και των εξαρτώμενων κλάσεων

Η λύση:

Ανάθεση αρμοδιοτήτων ώστε η σύζευξη coupling να παραμένει χαμηλή. Η αρχή αυτή θα πρέπει να χρησιμοποιείται για την αξιολόγηση όλων των εναλλακτικών λύσεων.

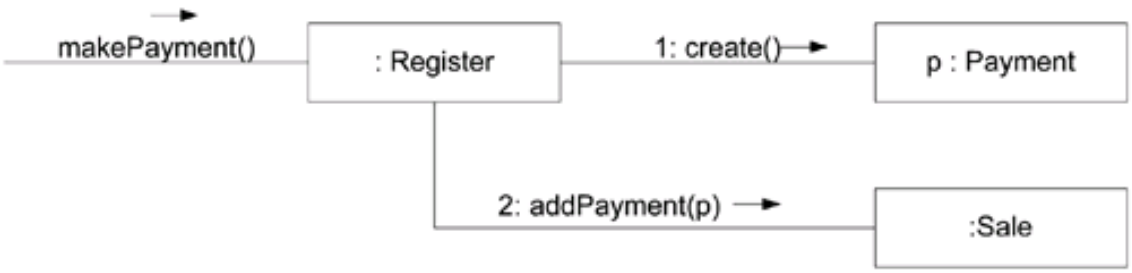
Παράδειγμα:

Έστω το παρακάτω μερικό διάγραμμα κλάσεων από το γνωστό παράδειγμα του NextGen POS:



Σχήμα 27: «Μερικό διάγραμμα κλάσεων Προτύπου Coupling»

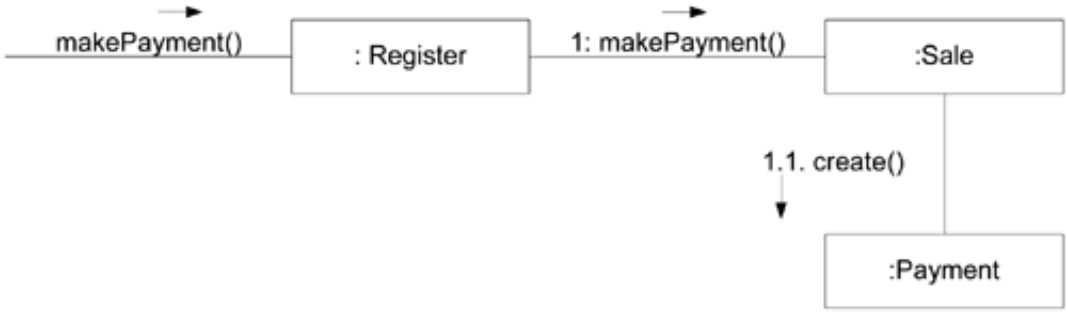
Έστω ότι χρειάζεται να δημιουργηθεί το στιγμιότυπο του Payment το οποίο πρέπει να συνδεθεί με το Sale. Σε ποια κλάση πρέπει να ανατεθεί η αρμοδιότητα αυτή; Επειδή το Register καταγράφει μία πώληση (Payment) στον πραγματικό κόσμο, το τρέχον πρότυπο υποδεικνύει το Register ως υποψήφιο δημιουργό του Payment. Το στιγμιότυπο Register μπορεί να στείλει ένα addPayment μήνυμα στο Sale, περνώντας το new Payment ως παράμετρο. Ένα πιθανό μερικό αλληλεπιδραστικό διάγραμμα φαίνεται στο επόμενο σχήμα:



Σχήμα 28: Πρότυπο Coupling: Το Register δημιουργεί ένα Payment

Ποιος σχεδιασμός βασισμένος σε μία ανάθεση αρμοδιοτήτων υποστηρίζει χαμηλή σύζευξη Coupling; Και στις δύο περιπτώσεις υποθέτουμε ότι η Sale πρέπει προοδευτικά να μάθει την πληροφορία ενός Payment. Ο σχεδιασμός στο σχήμα 28 όπου το Register δημιουργεί το Payment προσθέτει coupling του Register στο Payment. Ο σχεδιασμός στο σχήμα 29 όπου το Sale δημιουργεί ένα Payment, δεν αυξάνει το coupling. Άρα όσον αφορά στο τρέχον πρότυπο, ο σχεδιασμός του σχήματος 29 είναι καλύτερος. Εδώ φαίνεται καθαρά ότι δύο

διαφορετικά πρότυπα, το δημιουργικό πρότυπο Creator Και το Χαμηλής Σύζευξης Coupling προτείνουν δύο τελείως διαφορετικές προσεγγίσεις σχεδίασης.



Σχήμα 29: Πρότυπο Coupling: Το Sale δημιουργεί ένα Payment

Στην πράξη, το επίπεδο του προτύπου Χαμηλής Σύζευξης Coupling δεν μπορεί να εξεταστεί μόνο του, αλλά σε συνάρτηση με το πληροφοριακό πρότυπο Expert και το πρότυπο Υψηλής Συνοχής High Cohesion. Παρόλα αυτά, είναι προφανές ότι θα πρέπει να λαμβάνεται υπόψιν στον σχεδιασμό του λογισμικού.

Συζήτηση

Η χαμηλή σύζευξη είναι μια αρχή που θα πρέπει να τηρείται κατά το σχεδιασμό του συστήματος λογισμικού. Είναι ένας στόχος που πρέπει να λαμβάνεται συνεχώς υπόψη. Είναι μία **αρχή αξιολόγησης (evaluative principle)** που θα πρέπει να εφαρμόζεται σε όλες τις αποφάσεις σχεδίασης.

Σε αντικειμενοστρεφείς γλώσσες προγραμματισμού όπως η C++, Java και C#, συνηθισμένες φόρμες μετάβασης χαμηλής σύζευξης Coupling από ένα αντικείμενο TypeX σε ένα αντικείμενο TypeY περιέχουν τα ακόλουθα:

- Το TypeX εχεί μία ιδιότητα (είτε data member, είτε μεταβλητή) η οποία αφορά σε ένα στυγιμότυπο του TypeY, ή στο TypeY καθεαυτό.
- Ένα TypeX αντικείμενο ζητάει υπηρεσίες από ένα αντικείμενο TypeY.
- Το TypeX έχει μία μέθοδο η οποία αναφέρεται σε ένα στιγμιότυπο

του TypeY, ή στο TypeY καθεαυτό. Αυτό συνήθως σημάινει ότι εμπεριέχει μία παράμετρο ή τοπική μεταβλητή τύπου TypeY, ή το αντικείμενο που επιστρέφεται από ένα μήνυμα, είναι ένα στιγμιότυπο του TypeY.

- Το TypeX είναι μία ευθεία ή όχι υποκλάση του TypeY.
- Το TypeY είναι ένα Interface και το TypeX το υλοποιεί

Η χαμηλή σύζευξη Coupling υποστηρίζει τον σχεδιασμό κλάσεων που είναι ανεξάρτητες, το οποίο τελικά οδηγεί τις αλλαγές στο σύστημα να έχουν χαμηλό αντίκτυπο. Αν και δεν μπορεί να εξεταστεί ξεχωριστά από άλλα πρότυπα, και ειδικά από το δημιουργικό πρότυπο και το πρότυπο υψηλής συνονχής, θα πρέπει μονίμως να εξετάζεται ως σοβαρή παράμετρος του συστήματος.

Μία υποκλάση έχει προφανώς υψηλή σύζευξη με την περιέχουσα κλάση της. Άρα η δημιουργία τους θα πρέπει να εξετάζεται προσεκτικά. Για παράδειγμα, αν κάποια αντικείμενα πρέπει να αποθηκεύονται μόνιμα σε μία βάση δεδομένων, θα πρέπει να δημιουργηθεί μία αφηρημένη (abstract) υπερκλάση, έστω η PersistentObject από την οποία θα παράγονται οι άλλες κλάσεις. Το μειονέκτημα μίας τέτοιας σχεδίασης είναι η μεγάλη σύζευξη των υποκλάσεων με την υπερκλάση, ενώ το πλεονέκτημα είναι η κληρονομικότητα των μόνιμων χαρακτηριστικών αποθήκευσης που θα έχουν εξαρχής.

Η απόλυτη μέτρηση της τιμής της σύζευξης στερείται νοήματος. Αντί για αυτό θα πρέπει να εξετάζεται αν η αύξηση του οδηγεί σε προβλήματα. Συνήθως κλάσεις οι οποίες είναι εκ φύσεως γενικές και με υψηλή πιθανότητα επαναχρησιμοποίησης θα πρέπει να έχουν ιδιαίτερα χαμηλή σύζευξη.

Στην ακραία περίπτωση που δεν υπάρχει σύζευξη μεταξύ κλάσεων, υπάρχει μια απόλυτη περιγραφή αυτού του συστήματος: Είναι ένα σύστημα αντικειμένων που επικοινωνούν μόνο μέσω μηνυμάτων. Αυτό ουσιαστικά σημαίνει ότι οι κλάσεις του συστήματος είναι ανεξάρτητες και δεν υπάρχει λόγος να συνυπάρχουν. Άρα ένας λογικός βαθμός σύζευξης είναι αναμενόμενος σε κάθε σύστημα λογισμικού.

Αντιλογίες

Η υψηλή τιμή σύζευξης σε σταθερά και συχνοεμφανιζόμενα στοιχεία είναι συνήθως πρόβλημα. Για παράδειγμα μία εφαρμογή J2EE μπορεί να έχει σύζευξη με τις βιβλιοθήκες της Java (π.χ. java.util κτλ) επειδή είναι σταθερές και συχνά εμφανιζόμενες.

Πότε να χρησιμοποιηθεί

Η υψηλή σύζευξη αυτή καθεαυτή δεν είναι πρόβλημα. Γίνεται πρόβλημα όταν έχει υψηλές τιμές σε αντικείμενα που θα μπορούσαν να έχουν πρόβλημα με αυτό, όπως τα γραφικά περιβάλλοντα (interfaces), οι διάφορες εφαρμογές (implementation) κτλ

Για να προσδώσουμε ευελιξία στο σύστημα, θα πρέπει να ενσωματώνουμε τις λεπτομέρειες και τις τελικές υλοποιήσεις σε πολλά σημεία του συστήματος. Αλλά η επιδίωξη του χαμηλού Coupling ως γενική αρχή, δεν έχει κανένα νόημα στην δημιουργία ενός ολόκληρου συστήματος λογισμικού.

Αντίθετα, θα πρέπει να γίνεται προσεκτική επιλογή των σημείων του συστήματος που θα πρέπει να έχουμε χαμηλή σύζευξη και ενθυλάκωση πληροφορίας. Στο παράδειγμα του NextGEn POS γνωρίζοντας ότι εξωτερικά συστήματα υπολογισμού φόρων και κρατήσεων τρίτου κατασκευαστή θα χρειαστεί να ενσωματωθούν με το σύστημα, σε αυτά ακριβώς τα σημεία σύνδεσης, η σύζευξη θα πρέπει να είναι εξαιρετικά χαμηλή.

Ωφέλειες

- Το σύστημα δεν επηρεάζεται από αλλαγές σε άλλα σημεία
- Ευκολία κατανόησης των επιμέρους συστημάτων , κλάσεων αντικειμένων, όταν εξετάζονται αυτόνομα
- Εύκολη επαναχρησιμοποίηση

Υπόβαθρο

Η (χαμηλή) σύζευξη (Coupling), μαζί με την (υψηλή) συνοχή (Cohesion) είναι πράγματι βασικές αρχές σχεδιασμού και θα πρέπει να τυγχάνουν αυξημένης βαρύτητας και να εφαρμόζονται από όλους τους μηχανικούς λογισμικού. Ένας από τους πρωτεργάτες της τεχνολογίας λογισμικού, ο Larry Constantine θεωρεί αυτές τις δύο ως τις βασικότερες αρχές που θα πρέπει να εφαρμόζονται σε κάθε έργο λογισμικού (Constantine, 1994), (Constantine & Lockwood, 1999)

Σχετικά Πρότυπα

- Προστατευμένων Παραλλαγών Protected Variation

ΥΠΟΚΕΦΑΛΑΙΟ 5.6 Πρότυπο Ελέγχου GRASP Pattern - Controller

Το πρόβλημα:

Ποιο είναι το πρώτο αντικείμενο εκτός της διεπαφής του χρήστη που υποδέχεται και διευθύνει (ελέγχει = control) τις λειτουργίες του συστήματος;

Οι **λειτουργίες συστήματος** είναι οι σημαντικές λειτουργίες εισόδου/εισαγωγής του συστήματος. Για παράδειγμα, όταν ένας ταμίας χρησιμοποιώντας το POS πατάει το κουμπί “End Sale”, «Ολοκλήρωση Πώλησης», δημιουργεί μία κλήση συστήματος που δίνει την οδηγία ότι η πώληση ολοκληρώθηκε. Αντίστοιχα όταν κάποιος χρησιμοποιώντας έναν επεξεργαστή κειμένου πατάει το κουμπί. “spell check” δημιουργεί μία κλήση συστήματος που εκκινεί την διαδικασία ελέγχου του κειμένου για γραμματικά λάθη.

Ο **Ελεγκτής (Controller)** είναι το πρώτο αντικείμενο πέραν της διεπαφής του χρήστη που θα είναι αρμόδιος για την υποδοχή και χειρισμό των μηνυμάτων λειτουργιών συστήματος.

Η λύση:

Ανάθεση της αρμοδιότητας σε μία κλάση που αναπαριστά κάποιο από τα παρακάτω:

- Αντιπροσωπεύει το πλήρες σύστημα, Είναι δηλαδή μία βασική κλάση, ένα πρωταρχικό αντικείμενο, μία συσκευή μέσα στην οποία εκτελείται το λογισμικό, ένα κυρίαρχο υποσύστημα, ότι τέλος πάντων μπορεί να αντιπροσωπεύει ένας **πρωταρχικός ελεγκτής (facade controller).**
- Αντιπροσωπεύει ένα σενάριο χρήσης μέσα στο οποίο συμβαίνει αυτή η κλήση συστήματος, συχνά ονομαζόμενη ως <UseCaseName>Handler, <UseCaseName>Coordinator, ή <UseCaseName>Session και χρησιμοποιεί
    - Την ίδια κλάση ελεγκτή για όλα τα συμβάντα συστήματος στο ίδιο

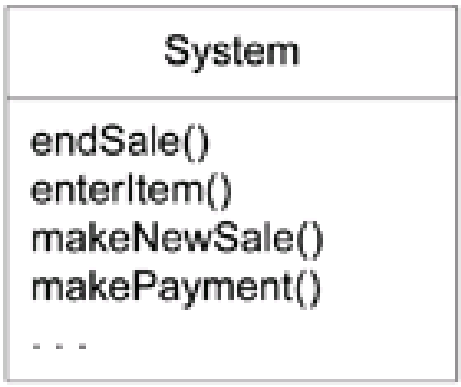
σενάριο χρήσης

- Ανεπίσημα μία σύνοδο σε ένα στιγμιότυπο αποστολής/παραλαβής μηνυμάτων με εξωτερικούς χρήστες (συστήματα ή πραγματικούς χρήστες).

Σημείωση: Κλάσεις όπως η “window”, “view”, “document” δεν ανήκουν σε αυτήν την κατηγορία. Τέτοιες κλάσεις δεν θα πρέπει να εκτελούν διεργασίες συστήματος. Συνήθως απλώς υποδέχονται τέτοια γεγονότα και τα παραπέμπουν σε έναν ελεγκτή (controller).

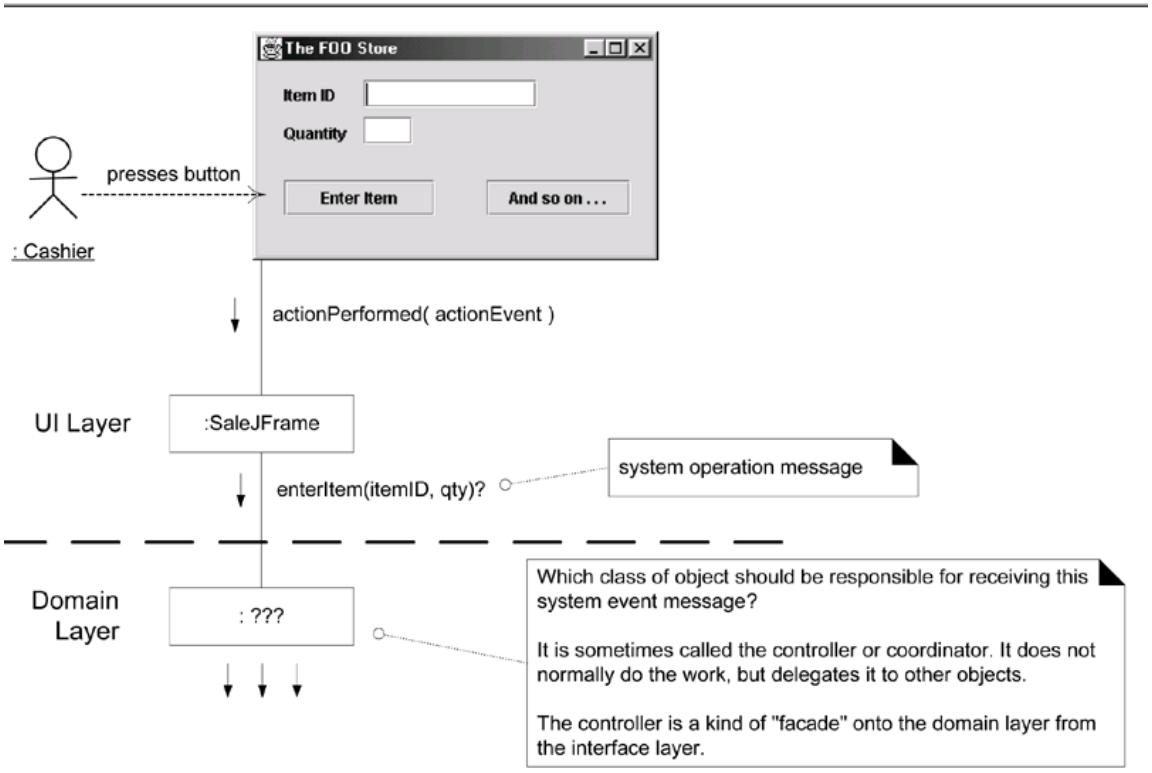
Παράδειγμα:

Το μοντέλο στο σχήμα 30 δείχνει το σύστημα ως μία κλάση, το οποίο κάποιες φορές είναι χρήσιμο όταν λαμβάνονται αποφάσεις αναπαράστασης



Σχήμα 30: «Παράδειγμα προτύπου Controller, Λειτουργίες Συστήματος του NextHGen POS

Κατά την διάρκεια της ανάλυσης οι λειτουργίες συστήματος μπορούν να ανατεθούν στην κλάση System ώστε να φαίνεται καθαρά ότι είναι λειτουργίες συστήματος. Αυτό δεν σημαίνει ότι μία κλάση ονομαζόμενη System μπορεί και πρέπει να τις εκτελεί. Αντί για αυτό, θα πρέπει μία κλάση-ελεγκτής (controller class) να έχει τέτοιες αρμοδιότητες όπως φαίνεται στο σχήμα 31:



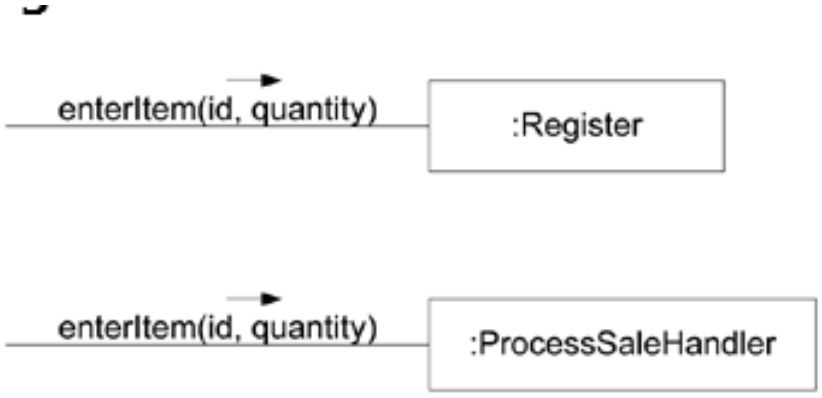
Σχήμα 31: Ποιο αντικείμενο θα πρέπει να είναι ο Ελεγκτής του enterItem;

Ποιος θα πρέπει να είναι ο Ελεγκτής των συμβάντων συστήματος όπως το enterItem και το endSale;

Σύμφωνα με το παρόν πρότυπο, ακολουθούν μερικές επιλογές:

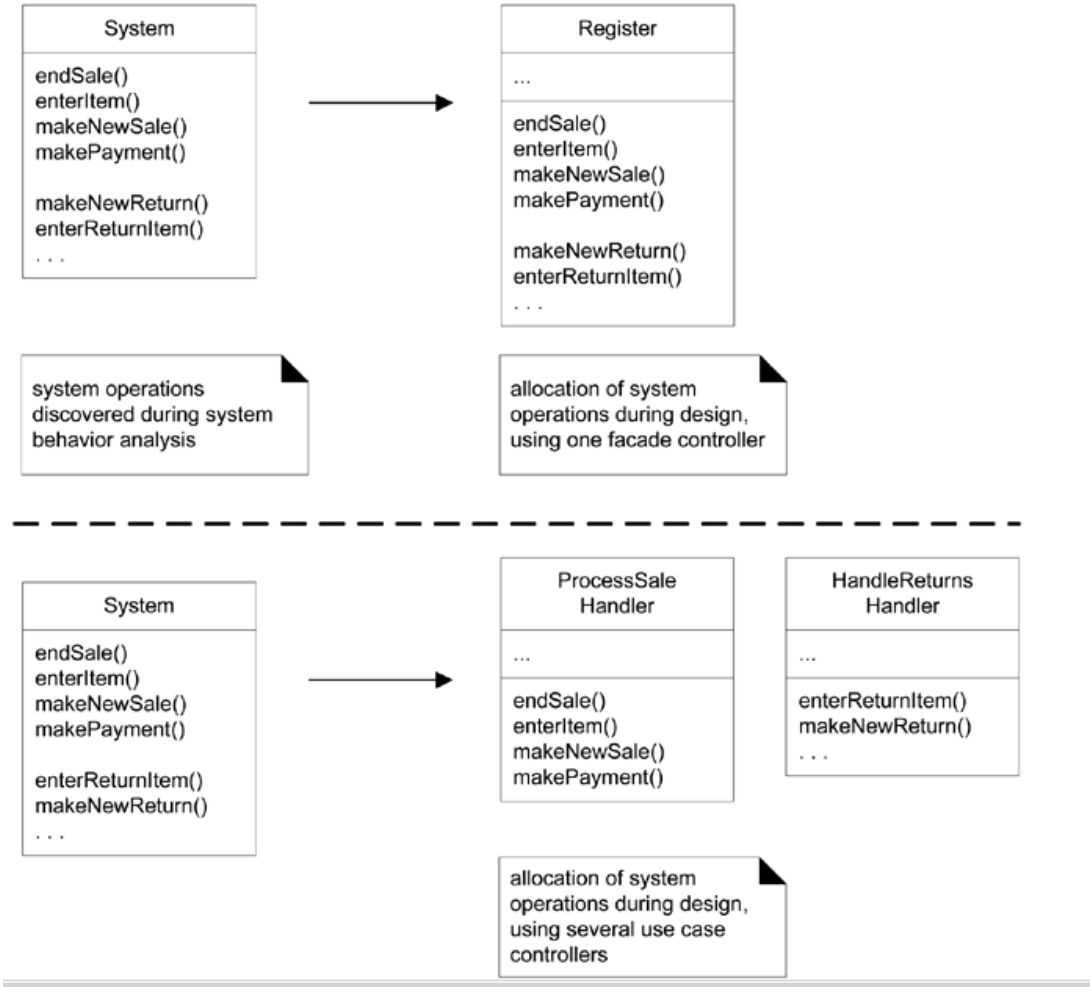
Register, POS System Αντιπροσωπεύει το όλο «σύστημα», το αρχικό αντικείμενο, συσκευή, ή υποσύστημα

ProcessSaleHandler, ProcessSaleSession Αντιπροσωπεύει έναν υποδοχέα ή χειριστή όλων των συμβάντων συστήματος σε κάποιο σενάριο χρήσης



Σχήμα 32: Επιλογές Ελεγκτή (Controller)

Στο σχήμα 32 η επιλογή Ελεγκτή εξαρτάται από άλλους παράγοντες που όπως φαίνεται στο σχήμα 33 μπορούν να ανατεθούν στην κλάση Register:



Σχήμα 33: Εντοπισμός των λειτουργιών Συστήματος

Συζήτηση

Το πρότυπο αυτό είναι ένα πρότυπο που ορίζει τους τρόπους αποστολής μηνυμάτων. Ορίζει τον τρόπο αποστολής μηνυμάτων συνήθως μεταξύ της διεπαφής χρήστη και των επιπέδων πίσω και κάτω από αυτή. Όταν αυτό το επίπεδο είναι το επίπεδο συστήματος, τότε το πρότυπο ουσιαστικά συγκεντρώνει όλες τις αρχικές αποφάσεις του αντικειμενοστρεφούς προγραμματισμού που θα πρέπει να παρθούν σε επίπεδο συστήματος.

Είτε ένα σύστημα λαμβάνει μηνύματα κατευθείαν από τον άνθρωπο-χειριστή, είτε από άλλα συμβάντα (π.χ. από το χτύπημα της γραμμής του τηλεφώνου, την έναρξη λήψης φαξ) θα πρέπει να οριστεί ένας χειριστής στο λογισμικό αυτών των συμβάντων, όπως στο σχήμα 31.

Συχνά είναι επιθυμητό ένας Ελεγκτής να χειρίζεται πολλά συμβάντα του συστήματος, αν είναι δυνατόν για μία ολόκληρη συγκεκριμένη περίπτωση χρήσης, έτσι ώστε να υπάρχει και η απαραίτητη ιεράρχηση στην αλληλουχία των γεγονότων, π.χ. η λειτουργία MakePayment θα πρέπει πάντα να προηγείται μίας λειτουργίας EndSale. Διαφορετικοί Ελεγκτές προφανώς θα χειρίζονται διαφορετικά συμβάντα ή/και περιπτώσεις χρήσης.

Ένα συνηθισμένο σύμπτωμα σε αυτές τις περιπτώσεις είναι η υπερσυγκέντρωση αρμοδιοτήτων σε έναν μοναδικό Ελεγκτή. Σε τέτοιες περιπτώσεις ο Ελεγκτής «υποφέρει» από χαμηλό cohesion, παραβιάζοντας το σχετικό πρότυπο που ορίζει πως το cohesion πρέπει να είναι υψηλό.

Συνήθως **ένας Ελεγκτής απλώς ορίζει, κατευθύνει και αναθέτει αρμοδιότητες** σε υποκείμενα κομμάτια του συστήματος. Ο ίδιος ο Ελεγκτής δεν κάνει κάποιες από αυτές τις εργασίες.

Οι λεγόμενοι πρωταρχικοί Ελεγκτές (facade controllers) είναι χρήσιμοι όταν δεν υπάρχουν πολλά συμβάντα συστήματος, ή όταν η διεπαφή χρήστη δεν μπορεί να ανακατευθύνει την ροή των μηνυμάτων του συστήματος σε εναλλακτικούς Ελεγκτές.

Η χρήση των Ελεγκτών Περίπτωσης (Case Controllers) προϋποθέτει την ύπαρξη ενός από αυτούς για κάθε περίπτωση χρήσης. Σε αυτή την περίπτωση ο Ελεγκτής είναι ένας κατασκευασμένος μηχανισμός για αυτόν τον σκοπό. Στο γνωστό παράδειγμα του NextGen POS, θα πρέπει να δημιουργηθεί η κλάση ProccessSaleHandler που θα χειρίζεται την πορεία της πώλησης (Sale), μία αντίστοιχη κλάση HandleReturns για τον χειρισμό των επιστροφών, κτλ.

Η απόφαση της προτίμησης ενός Ελεγκτή Περίπτωσης έναντι ενός Πρωταρχικού Ελεγκτή, λαμβάνεται αν η υλοποίηση του δεύτερου θα οδηγήσει σε σχεδίαση ενός συστήματος με χαμηλή συνοχή (cohesion) ή υψηλή σύζευξη (coupling), συνήθως δηλαδή όταν ο πρωταρχικός ελεγκτής «ασφυκτιά» κάτω από το βάρος πολλαπλών αρμοδιοτήτων.

Ένα σημαντικό αποτέλεσμα του τρέχοντος προτύπου είναι το ότι τα αντικείμενα της διεπαφής χρήστη (π.χ. τα παράθυρα, ή τα κουμπιά) αλλά και αυτή καθεαυτή η διεπαφή χρήστη δεν θα έχουν αρμοδιότητες χειρισμού συμβάντων του συστήματος. Δηλαδή **οι χειρισμοί του συστήματος πρέπει να γίνονται από εφαρμογές και όχι στην διεπιφάνεια χρήστη**.

Ωφέλειες

- Αυξημένη δυνητική επαναχρησιμοποίηση και μεταφερσιμότητα των διεπαφών. Αυτό εξασφαλίζεται από το ότι οι διεπαφές δεν χειρίζονται συμβάντα του συστήματος. Αν και οι αρμοδιότητες ενός Ελεγκτή θα μπορούσαν άνετα να ανατεθούν σε ένα συστατικό στοιχείο της διεπαφής χρήστη (π.χ. ένα κουμπί), αυτό θα απέτρεπε τόσο την επαναχρησιμοποίηση αυτής της διεπαφής, αλλά και θα δυσκόλευε υπερβολικά την χρήση της σε μελλοντικές εκδόσεις και αναβαθμίσεις.
- Δυνατότητα αντίληψης του τρέχοντος σταδίου της περίπτωσης χρήσης. Σε πολλές περιπτώσεις χρειάζεται να είναι γνωστή η ακριβής κατάσταση μίας περίπτωσης χρήσης Π.χ. θα πρέπει να υπάρξει μία δικλείδα ότι το makePayment, συμβαίνει πάντα αφού ολοκληρώσει επιτυχώς το endSale. Η πληροφορία αυτή πρέπει να αποθηκεύεται κάπου και ένας Ελεγκτής είναι

το σημείο όπου αυτή η πληροφορία μπορεί να οδηγήσει αυτόν τον Ελεγκτή στην αποστολή μηνύματος εκτέλεσης επόμενου σταδίου ή εντολής.

Υλοποίηση

Το παρακάτω παράδειγμα χρησιμοποιεί Java σε έναν rich client σε Java Swing.

Άξια Προσοχής Τμήματα:

Στο (1) το ProcessSaleJFrame παράθυρο έχει μία αναφορά στο Register. Στο (2) ο χειριστής ορίζεται για το κλικ του κουμπιού. Στο (3) φαίνεται το κυρίως μήνυμα που τερματίζει το enterItem.

```
package com.craiglarman.nextgen.ui.swing;

       // imports…


       // in Java, a JFrame is a typical window
    public class ProcessSaleJFrame extends JFrame
    {

       // the window has a reference to the 'controller' domain object

(1)  private Register register;


       // the window is passed the register, on creation
    public ProcessSaleJFrame(Register _register)
    {
       register = _register;
    }
// this button is clicked to perform the
       // system operation "enterItem"
    private JButton BTN_ENTER_ITEM;


       // this is the important method!
       // here i show the message from the UI layer to domain layer
    private JButton getBTN_ENTER_ITEM()
    {
          // does the button exist?
       if (BTN_ENTER_ITEM != null)
          return BTN_ENTER_ITEM;


          // ELSE button needs to be initialized...
       BTN_ENTER_ITEM = new JButton();
       BTN_ENTER_ITEM.setText("Enter Item");
```

```
          // THIS IS THE KEY SECTION!
          // in Java, this is how you define
          // a click handler for a button
(2) BTN_ENTER_ITEM.addActionListener(new ActionListener()
        {
        public void actionPerformed(ActionEvent e)
        {
               // Transformer is a utility class to
               // transform Strings to other data types
               // because the JTextField GUI widgets have Strings
           ItemID id = Transformer.toItemID(getTXT_ID().getText());
           int qty = Transformer.toInt(getTXT_QTY().getText());


               // here we cross the boundary from the
               // UI layer to the domain layer
               // delegate to the 'controller'
               // > > > THIS IS THE KEY STATEMENT < < <

(3)          register.enterItem(id, qty);
        }
        } ); // end of the addActionListener call

     return BTN_ENTER_ITEM;
     } // end of method

  // …
  } // end of class
```

Υπερφορτωμένοι Ελεγκτές

Αν μία κλάση ενός Ελεγκτή δεν σχεδιαστεί σωστά, τότε θα υπάρχει χαμηλό cohesion επειδή θα υπάρχουν πολλές αρμοδιότητες σε πολλούς τομείς ταυτόχρονα. Σημάδια της υπερφόρτωσης είναι:

- Υπάρχει μόνο ένας Ελεγκτής που λαμβάνει όλα τα (υπεραρκετά) μηνύματα του συστήματος, χωρίς να τα μεταβιβάζει σε άλλους ελεγκτές. Αυτό πολλές φορές συμβαίνει όταν έχει επιλεγεί ένας facade controller.
- Ο Ελεγκτής διεκπεραιώνει μόνος του πολλές από τις απαραίτητες ενέργειες του συστήματος χωρίς να μεταβιβάζει σε άλλα τμήματα αυτές τις εργασίες. Αυτό συνήθως εναντιώνεται στα πρότυπα Information Expert και High Cohesion.
- Ένας Ελεγκτής έχει πολλές ιδιότητες (παραμέτρους) και διατηρεί σημαντικές πληροφορίες τόσο για το λογισμικό , όσο και για ολόκληρο το σύστημα, ενώ θα έπρεπε να μεταφέρονται σε άλλα αντικείμενα, ή διατηρεί

εις διπλούν πληροφορία που υπάρχει και σε άλλα σημεία του συστήματος.

Για να διορθωθούν τα παραπάνω, αρκεί συνήθως να προστεθούν Ελεγκτές. Αντί για πρωταρχικούς Ελεγκτές, ίσως πρέπει να προστεθούν Ελεγκτές Περίπτωσης. Ίσως επίσης να πρέπει ο Ελεγκτής να διαχέει τις αρμοδιότητες σε υπάρχοντα ή ακόμη και δημιουργημένα για αυτόν τον σκοπό αντικείμενα (ή κλάσεις).

Σχετικά Πρότυπα

- Κατασκευής Pure Fabrication

ΥΠΟΚΕΦΑΛΑΙΟ 5.7 Πρότυπο Υψηλής Συνοχής GRASP Pattern – High Cohesion

Το πρόβλημα:

Πως διατηρούμε τα αντικείμενα επικεντρωμένα στο στόχο τους, κατανοητά και διαχειρήσιμα και ως παρενέργεια όλο αυτό να υποστηρίζει και το πρότυπο χαμηλό Coupling;

Με όρους αντικειμενοστρέφειας, το cohesion (συνάφεια, συνοχή) είναι ένα μέτρο το πόσο ισχυρά συνδεδεμένες μεταξύ τους και συνεπείς στον στόχο τους είναι οι αρμοδιότητες ενός αντικειμένου. Ένα στοιχείο με πολύ συγγενείς μεταξύ τους αρμοδιότητες, το οποίο δεν κάνει έναν τεράστιο «κύκλο εργασιών» έχει υψηλή συνοχή. Αυτά τα αντικείμενα μπορεί να είναι κλάσεις, υποσυστήματα κτλ

Η λύση:

Ανάθεση αρμοδιοτήτων έτσι ώστε η συνοχή να παραμένει υψηλή. Σύμφωνα με αυτό αποτιμώνται όλες οι πιθανές λύσεις.

Μία κλάση με χαμηλή συνοχή κάνει πολλές μη συγγενείς μεταξύ τους εργασίες ή κάνει πάρα πολλά πράγματα. Τέτοιες κλάσεις δεν είναι επιθυμητές, και σίγουρα «υποφέρουν» από ένα τουλάχιστον από τα παρακάτω:

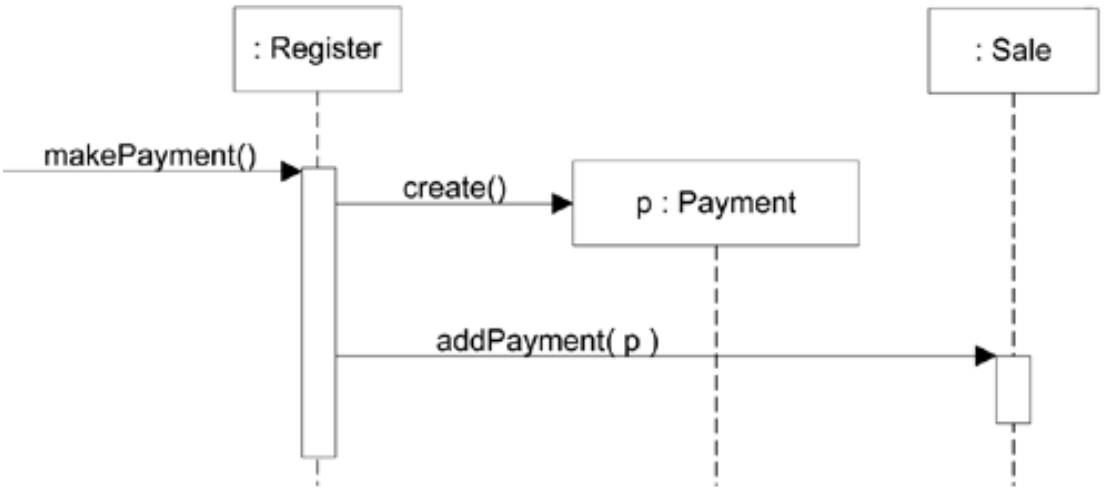
- Δυσκολία κατανόησης
- Δυσκολία επαναχρησιμοποιήσης
- Δυσκολία συντήρησης
- (Υπερβολική) Ευαισθησία στις αλλαγές που έχουν ισχυρό αντίκτυπο πάνω τους

Οι κλάσεις με χαμηλή συνοχή συχνά αντιπροσωπεύουν ένα πολύ ευρύ πεδίο αφαιρετικότητας, ή έχουν αναλάβει αρμοδιότητες που θα έπρεπε να έχουν διαχυθεί σε άλλες οντότητες του συστήματος.

Παράδειγμα:

Θα εξεταστεί το παράδειγμα που δόθηκε για το χαμηλή σύζευξη στο αντίστοιχο κεφάλαιο:

Έστω ότι χρειάζεται να υλοποιηθεί ένα στιγμιότυπο Payment και να αντιστοιχισθεί με το Sale. Ποια κλάση θα έπρεπε να είναι αρμόδια για αυτό; Αφού το Register καταχωρεί ένα Payment στον πραγματικό κόσμο, το πρότυπο Pattern υποδεικνύει το Register ως υποψήφιο δημιουργό του Payment. Το στιγμιότυπο Register μπορεί τότε ένα μήνυμα addPayment στο Sale περνώντας αυτή την καινούρια Payment ως παράμετρο, όπως φαίνεται στο παρακάτω σχήμα:

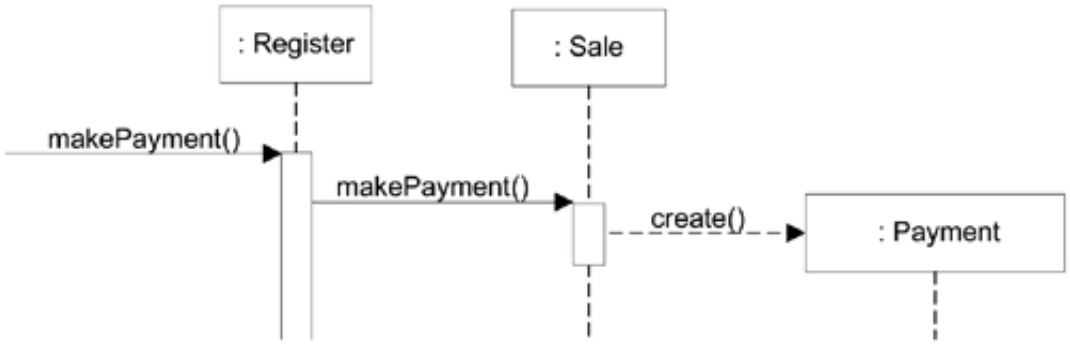


Σχήμα 34: «Πέρασμα του addPayment ως παραμέτρου στο Sale από το Register

Σε αυτό το παράδειγμα ανατίθεται στο Register η αρμοδιότητα του να κάνει μία Payment, μέσω της ανάθεσης του makePayment. Αν συνεχίσουμε να αναθέτουμε όμως τέτοιες αρμοδιότητες στην κλάση Register, δηλαδή συμβάντα συστήματος, προοδευτικά θα «βουλιάξει» κάτω από το βάρος των ευθυνών αυτών, οδηγώντας σε χαμηλή συνοχή, cohesive, ή ασυνέχεια (incohesive).

Εδώ να σημειωθεί ότι η ανάθεση της Payment από μόνη της, προφανώς δεν οδηγεί σε χαμηλή συνέχεια, cohesive αλλά η συγκέντρωση τέτοιων αντίστοιχων αρμοδιοτήτων, μπορεί εύκολα να οδηγήσει εκεί.

Αντιθέτως όπως φαίνεται στο επόμενο σχήμα, η ανάθεση της δημιουργίας ενός Payment στην κλάση Sale υποστηρίζει υψηλότερη συνοχή cohesion στην Register.



Σχήμα 35: Η Sale δημιουργεί ένα Payment

Αυτός ο δεύτερος τρόπος σχεδιασμού, υποστηρίζει και υψηλή συνέχεια, cohesion και χαμηλή σύζευξη coupling.

Να σημειωθεί ότι στην πράξη η συνοχή δεν μπορεί να αποτιμάται μόνη της. Πρέπει πάντα να γίνεται σε συνδυασμό με άλλα πρότυπα όμως το Πληροφοριακό Expert και το Χαμηλής Σύζευξης Coupling.

.

Συζήτηση

Όπως και τη χαμηλή Σύζευξη Coupling, η Υψηλή Συνοχή Cohesion είναι μία σχεδιαστική αρχή που πρέπει να λαμβάνεται υπόψη σε κάθε σχεδιαστική απόφαση. Είναι μία αρχή που θα πρέπει να αποτιμάται συνεχώς, σε κάθε στάδιο της δημιουργίας ενός έργου λογισμικού.

Ακολουθούν μερικά συνοπτικά παραδείγματα περιπτώσεων εξέτασης του προτύπου:

- Μία κλάση Α με χαμηλή συνοχή cohesion είναι αποκλειστικά αρμόδια για πολλά πράγματα σε διαφορετικά πεδία του συστήματος

- Μία κλάση Α με χαμηλή συνοχή έχει αποκλειστική αρμοδιότητα ενός πολύπλοκου έργου σε μία μονοθεματική περιοχή του συστήματος
- Μία κλάση Α με υψηλή συνοχή cohesion έχει αποδεκτού πλήθους αρμοδιότητες σε μία συγκεκριμένη περιοχή του συστήματος και συνεργάζεται με άλλες κλάσεις για να φέρει σε πέρας τις εργασίες που της ανατέθηκαν
- Μία κλάση Α με μέτρια συνοχή cohesion έχει κάποιες ελάχιστες αρμοδιότητες σε λίγα σχετικά πεδία του συστήματος τα οποία συνδέονται λογικά με την κλάση αυτή, αλλά όχι μεταξύ τους

Ως γενικός κανόνας, μία κλάση με υψηλή συνοχή cohesion έχει έναν σχετικά μικρό αριθμό μεθόδων, με υψηλού βαθμού συγγένεια στις λειτουργίες τους και δεν κάνει πάρα πολλά πράγματα. Συνεργάζεται με άλλα κομμάτια του συστήματος για να εκτελέσει τις πολύπλοκες λειτουργίες.

Μία κλάση με υψηλή συνοχή cohesion είναι επιθυμητή γιατί είναι σχετικά εύκολο να συντηρηθεί, κατανοηθεί και επαναχρησιμοποιηθεί. Η υψηλού βαθμού λειτουργικότητα σε συνδυασμό με τον μικρό αριθμό εργασιών (συναρτήσεων) απλοποιούν και την συντήρηση αλλά και τον εμπλουτισμό δυνατοτήτων.

Cohesion & Coupling, αλληλοσυμπληρώνονται

Η «κακή» συνοχή συνήθως γεννά και «κακή» Σύζευξη Coupling, και το αντίθετο. Αυτά τα δύο συνήθως ικανοποιούνται μαζί. Για παράδειγμα σε μία GUI widget κλάση που αντιπροσωπεύει και χρωματίζει το widget, σώζει δεδομένα στην βάση δεδομένων και εμπλέκει και απομακρυσμένα αντικείμενα, όχι μόνο αυτή η κλάση έχει προφανώς χαμηλή συνοχή αλλά έχει επίσης και υψηλή σύζευξη.

Αντιλογίες

Σε κάποιες εξαιρετικές περιπτώσεις η χαμηλή συνοχή είναι ζητούμενο.

Μία τέτοια περίπτωση είναι η συνένωση αρμοδιοτήτων ή κώδικα σε μία

κλάση για να απλουστευτεί η συντήρηση από έναν άνθρωπο, αν και πολλές φορές κάτι τέτοιο δυσκολεύει ακόμη και την συντήρηση.

Κλασσικό παράδειγμα μίας τέτοιας περίπτωσης είναι η ανάπτυξη λογισμικού με βάση δεδομένων, όπου προφανώς υπάρχουν εντολές SQL. Για να μπορεί ο SQL συντηρητής να ασχολείται με τα ερωτήματα SQL χωρίς να χρειάζεται να ανατρέχει σε όλο το έργο λογισμικού, είναι προτιμότερο να υπάρχει μία κλάση RDBOperations, μέσα στην οποία συγκεντρώνονται όλα τα ερωτήματα αυτά.

Οφέλη

- Η απλούστευση και ευκολία κατανόησης αυξάνονται
- Η συντήρηση και οι προσθήκες δυνατοτήτων απλοποιούνται
- Υποστηρίζει και το πρότυπο Χαμηλής Σύζευξης Coupling
- Επαναχρησιμοποίηση των κλάσεων αφού επιτελούν λίγους και καθορισμένους σκοπούς

ΥΠΟΚΕΦΑΛΑΙΟ 5.8 Πρότυπο Πολυμορφισμού GRASP Pattern – Polymorphism

Το πρόβλημα:

Πως γίνεται ο χειρισμός εναλλακτικών μεθόδων βασισμένων στον αρχικό τύπο; Πώς να δημιουργηθούν κομμάτια λογισμικού που να μπορούν να εμφυτευθούν σε μεγαλύτερα μέρη του συστήματος;

Εναλλακτικά τμήματα βασισμένα στον αρχικό τύπο είναι βασικά θέματα στην συγγραφή λογισμικού. Εάν ένα πρόγραμμα είναι σχεδιασμένο να χρησιμοποιεί το if-then-else, ή την δομή case, εάν εμφανιστεί μία καινούρια εναλλακτική πιθανότητα (π.χ. εάν υπάρξει ακόμη ένα if), χρειάζονται τροποποιήσεις σε πολλά σημεία του λογισμικού. Αυτό καθιστά το πρόγραμμα ανελαστικό και δύσκολο στις αλλαγές και τις προσθήκες.

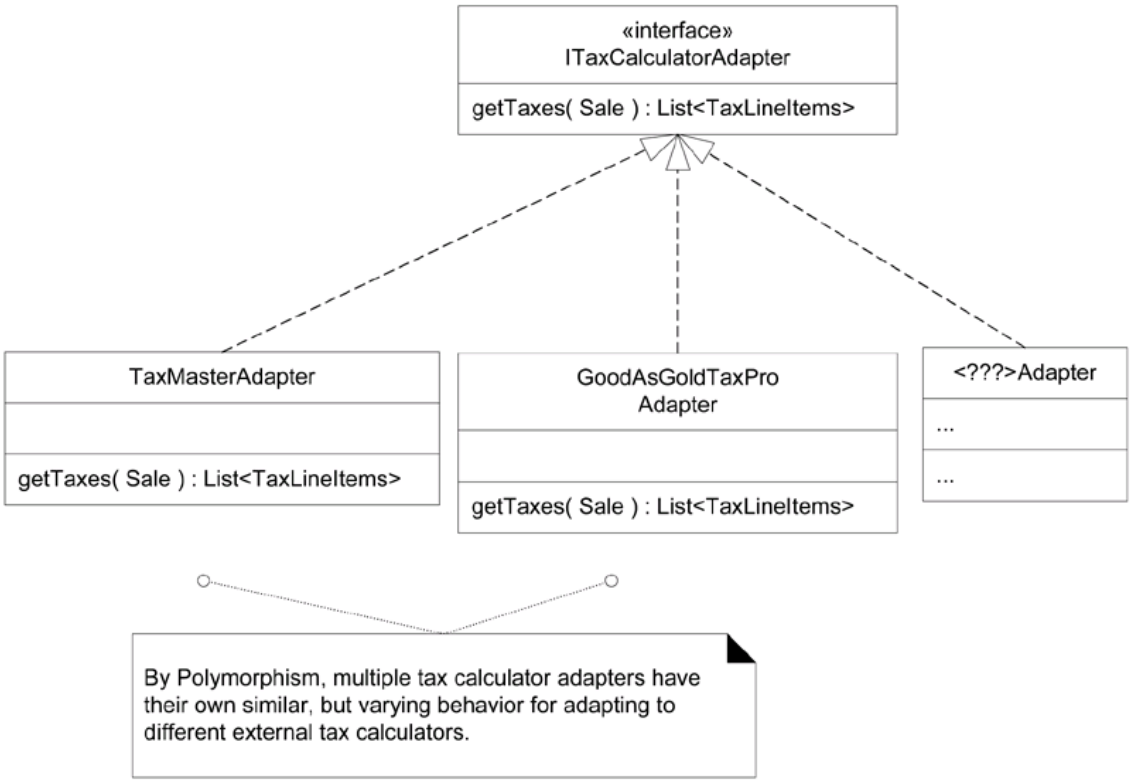
Σε ένα μοντέλο Εξυπηρετητή-Πελάτη, πως δημιουργούμε τέτοια κομμάτια λογισμικού ώστε όταν αλλάζουμε ένα δομικό κομμάτι του εξυπηρετητή να μην επηρεάζεται ο πελάτης;

Παράδειγμα:

Στην γνωστή εφαρμογή NextGen POS υπάρχουν αρκετά προγράμματα υπολογισμού κρατήσεων ανεξάρτητου κατασκευαστή που πρέπει να υποστηρίζονται. Το σύστημα θα πρέπει να μπορεί να συνεργάζεται αρμονικά με όλα. Κάθε ένα από αυτά τα προγράμματα (έστω το TaxMaster και το GoodAsTaxGoldPro) έχει διαφορετική διεπαφή, άρα θα πρέπει να υιοθετηθεί παρόμοια μεν, διαφορετική δε διεπαφή για το καθένα από αυτά.

Κάποιο από αυτά μπορεί να επικοινωνεί μέσω TCP, κάποιο άλλο με SOAP και κάποιο τρίτο με Java RMI. Όλα αυτά θα πρέπει να ανατεθούν σε μία πολυμορφική κλάση getTaxes η οποία προφανώς θα δέχεται διαφορετικές παραμέτρους και θα εκτελεί διαφορετικές λειτουργίες αναλόγως της διεπαφής με

την οποία θα συνεργαστεί, όπως διατυπώνεται στο παρακάτω σχήμα:



Σχήμα 36: «Πολυμορφισμός σε διεπαφές υπολογστιών κρατήσεων στο NextGen POS

Η μέθοδος getTaxes δέχεται το αντικείμενο Sale ως παράμετρο. Η υλοποίηση της κάθε μεθόδου getTaxes θα είναι διαφορετική, και θα προσαρμόσει την κλήση και τις παραμέτρους στο API του συγκεκριμένου προγράμματος.

Συζήτηση

Ο πολυμορφισμός είναι μία βασική αρχή σχεδιασμού για το πώς ένα σύστημα μπορεί και πρέπει να χειρίζεται τις παραλλαγές. Ένας αρχικός σχεδιασμός που υποστηρίζει πολυμορφισμό, μπορεί εύκολα να επεκταθεί ώστε να υποδεχθεί νέες παραλλαγές. Για παράδειγμα, στο NextGen POS μπορεί εύκολα να προστεθεί μία νέα getTaxes που θα αναλάβει να συνδέεται με κάποιο καινούριο πρόγραμμα υπολογισμού κρατήσεων του οποίου το API είναι γραμμένο σε UDP πρωτόκολλο.

Πότε σχεδιάζουμε και τις διεπαφές;

Ο πολυμορφισμός υπονοεί την παρουσία αφηρημένων (abstract) κλάσεων ή διεπαφών στις περισσότερες αντικειμενοστρεφείς γλώσσες. Πότε όμως πρέπει πραγματικά να χρησιμοποιηθεί μία διεπαφή; Η γενική απάντηση είναι, όταν χρειάζεται πολυμορφισμός χωρίς να πρέπει να είναι περιορισμένος σε μία συγκεκριμένη ιεραρχία κλάσεων. Εάν μία υπερκλάση AC χρησιμοποιείται χωρίς διεπαφή, τότε κάθε καινούρια πολυμορφική λύση πρέπει να είναι υποκλάση αυτής της AC το οποίο είναι πολύ περιοριστικό σε γλώσσες που δεν υποστηρίζουν πολλαπλή κληρονομικότητα όπως η Java και η C#.

Αντιλογίες

Μερικές φορές οι προγραμματιστές σχεδιάζουν συστήματα με διεπαφές και πολυμορφισμό έχοντας κατά νου κάποια μελλοντικά, άγνωστα προς το παρόν, προβλήματα ή θέματα που θα κληθούν να αντιμετωπίσουν. Εάν αυτή η πιθανότητα είναι σίγουρο ότι θα γίνει βεβαιότητα στο μέλλον, τότε σωστά πράττουν. Αλλά μία ψύχραιμη αποτίμηση είναι απαραίτητη γιατί πολλές φορές βρίσκουμε πολυμορφισμό και τις ανάλογες φυσικά πολυπλοκότητες σε σημεία που δεν θα χρειαστεί ποτέ.

Ωφέλειες

- Οι όποιες επεκτάσεις χρειάζονται για καινούριες παραλλαγές είναι πολύ εύκολο να υλοποιηθούν
- Καινούριες υλοποιήσεις μπορούν να εισαχθούν χωρίς να επηρεαστούν οι εφαρμογές-πελάτες.

Σχετικά Πρότυπα

- Προστατευμένων Παραλλαγών Protected Variations

Επίσης γνωστό ως:

- Choosing Message, Don’t Ask “What Kind?”

ΥΠΟΚΕΦΑΛΑΙΟ 5.9 Πρότυπο Κατασκευής GRASP Pattern – Pure Fabrication

Το πρόβλημα:

Ποιο αντικείμενο θα έπρεπε να έχει την αρμοδιότητα, όταν δεν πρέπει να παραβιάζονται η υψηλή Συνοχή Cohesion, η χαμηλή Σύζευξη Coupling ή και άλλα πρότυπα αλλά και επιπλέον λύσεις που υποδεικνύει το Πληροφοριακό πρότυπο (Expert) για παράδειγμα, δεν είναι εφαρμόσιμες.

Πολύ συχνά στον αντικειμενοστρεφή προγραμματισμό οι κλάσεις που δημιουργούνται έχουν μία ευθεία αναλογία με αντικείμενα και λειτουργίες του πραγματικού κόσμου, ώστε να μειωθεί το λεγόμενο κενό αναπαράστασης. Για παράδειγμα οι κλάσεις Sale, Costumer, είναι προφανές το τι κάνουν. Πολλές φορές όμως τέτοιες αναπαραστάσεις οδηγούν σε προβλήματα χαμηλής συνοχής ή χαμηλής σύζευξης, ή χαμηλή δυνατότητα επαναχρησιμοποίησης.

Η λύση:

Ανάθεση ενός σετ από αρμοδιότητες συγγενείς μεταξύ τους (με υψηλή συνοχή) σε μία τεχνητή κλάση, η οποία προφανώς δεν αντιπροσωπεύει κάτι αντίστοιχο στον πραγματικό κόσμο, μπορεί όμως να υποστηρίξει υψηλή συνοχή, χαμηλή σύζευξη και υψηλή δυνατότητα επαναχρησιμοποίησης.

Παράδειγμα:

Αποθηκεύοντας ένα αντικείμενο Sale στην βάση δεδομένων:

Έστω ότι παρίσταται η ανάγκη να αποθηκεύονται τα στιγμιότυπα του Sale στην βάση δεδομένων. Βάσει του πληροφοριακού προτύπου Expert, υπάρχει μία σοβαρή δικαιολογία να ανατεθεί αυτή η λειτουργία στην ίδια την κλάση Sale, επειδή προφανώς η κλάση η ίδια έχει (γνωρίζει) τα δεδομένα που πρέπει να αποθηκευθούν. Αλλά μπορεί να υπάρχουν οι κάτωθι επιπλοκές:

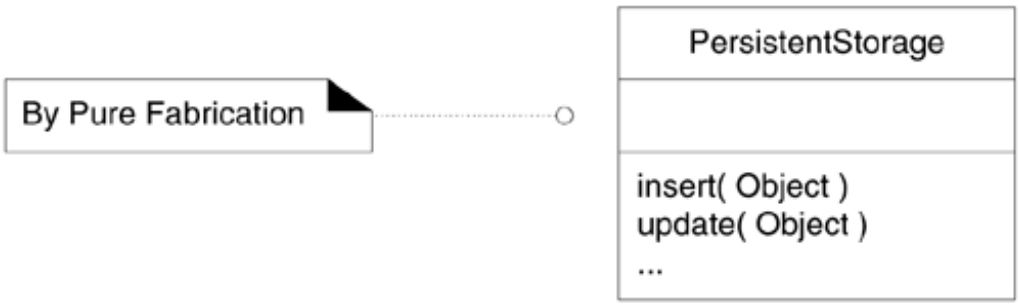
- Το έργο αυτό προϋποθέτει έναν αρκετά μεγάλο αριθμό από λειτουργίες και

λειτουργικότητες σχετικές με την βάση δεδομένων, οι οποίες προφανώς δεν συνδέονται με κανέναν τρόπο με τις «Πωλήσεις» με τις οποίες ασχολείται η Sale. Άρα με την προσθήκη αυτών των λειτουργιών χαμηλώνει η συνοχή.

- Η κλάση Sale πρέπει να είναι συνδεδεμένη με την βάση δεδομένων (π.χ. μέσω JDBC αν πρόκειται για Java) άρα το coupling ανεβαίνει.
- Η αποθήκευση δεδομένων σε βάση δεδομένων είναι μία πολύ γενική εργασία, την οποία προφανώς και επιτελούν αρκετές κλάσεις του συστήματος. Αν εμφωλεύσουμε μέσα στην Sale τις σχετικές με την αποθήκευση λειτουργίες, σημαίνει ότι αποκλείουμε όλες τις άλλες κλάσεις από την χρήση τους, ή πρέπει να τις ξαναγράψουμε μέσα σε κάθε κλάση.

Άρα, αν και σύμφωνα με το πληροφοριακό πρότυπο Expert η Sale θα μπορούσε να σώζει τον «εαυτό» της στην βάση δεδομένων, ένας τέτοιος σχεδιασμός οδηγεί σε χαμηλή συνοχή, υψηλή σύζευξη και χαμηλή πιθανότητα επαναχρησιμοποίησης.

Μία λογική λύση είναι η δημιουργία μίας νέας κλάσης που θα έχει αποκλειστική αρμοδιότητα την αποθήκευση δεδομένων σε βάσεις δεδομένων, έστω η κλάση PeristentStorage, η οποία προφανώς είναι ένα τεχνητό κατασκεύασμα, χωρίς αντιστοιχία στον πραγματικό κόσμο και αποθηκεύει δεδομένα σε σταθερά μέσα αποθήκευσης.



Σχήμα 37: «Δημιουργία τεχνητής κλάσης αποθήκευσης σε βάση δεδομένων

Είναι προφανές ότι η κλάση του σχήματος 37 είναι ένα τεχνητό (fabricated) τμήμα του λογισμικού, του οποίου το όνομα προδίδει τον σκοπό του, συνήθως μόνο σε μηχανικούς λογισμικού που θα ασχοληθούν με το συγκεκριμένο κομμάτι

και όχι με το εμπορικό τμήμα/προσωπικό της εταιρίας.

Το παρόν πρότυπο λύνει (ή τουλάχιστον προσπαθεί) τα κάτωθι προβλήματα:

- Η κλάση Sale παραμένει καλοσχεδιασμένη με υψηλή συνοχή και χαμηλή σύζευξη,
- Η καινούρια κλάση (PeristentStorage) έχει σχετικά καλή συνοχή αφού έχει ως μοναδικό σκοπό την αποθήκευση και εισαγωγή αντικειμένων σε μία βάση δεδομένων (ή άλλο σταθερό μέσο αποθήκευσης)
- Η κλάση PeristentStorage είναι ένα πολύ γενικό και εύκολα επαναχρησιμοποιούμενο αντικείμενο

Συζήτηση

Ο σχεδιασμός των αντικειμένων σε ένα έργο λογισμικού μπορεί να διαχωριστεί σε δύο πολύ βασικές κατηγορίες:

1. Αυτά που επιλέγονται βάσει της απεικονιστικής αναδόμησης
2. Αυτά που επιλέγονται βάσει της συμπεριφοριστικής αναδόμησης

Για παράδειγμα η δημιουργία μίας κλάσης Sale γίνεται βάσει της απεικονιστικής αναδόμησης. Το συστατικό αυτό του λογισμικού αναπαριστά ή συγγενεύει με ένα αντίστοιχο πράγμα στον πραγματικό κόσμο. Αυτού του είδους η αναδόμηση είναι μία κοινή στρατηγική στον σχεδιασμό αντικειμένων και υποστηρίζει τον στόχο του μικρού διαστήματος (κενού) με τον πραγματικό κόσμο.

Ένα καλό παράδειγμα είναι ένας αλγόριθμος, έστω ο ονομαζόμενος TableOfContentGenerator ο οποίος προφανώς δημιουργεί τον πίνακα περιεχομένων και δημιουργήθηκε ως βοήθημα για τους προγραμματιστές, χωρίς το όνομα του να αντιστοιχεί σε μία οντότητα του πραγματικού κόσμου. Δημιουργήθηκε ως βοηθητικό εργαλείο για να ομαδοποιήσει ομοειδή αντικείμενα ή συμπεριφορές και ως εκ τούτου είναι παρακινούμενο από την συμπεριφοριστική

αναδόμηση.

Αντιθέτως μία κλάση TableOfCOntent, είναι προφανώς μία οντότητα που συναντά την αντίστοιχη της στον πραγματικό κόσμο και περιέχει πληροφορίες ανάλογες με αυτές του πραγματικού κόσμου. Αυτή η κλάση είναι προφανώς αποτέλεσμα της απεικονιστικής αναδόμησης.

Η ταυτοποίηση μίας κλάσης ως αποτέλεσμα ή ανήκουσα στο τρέχον πρότυπο δεν είναι μία κρίσιμη διαδικασία. Είναι ένα εκπαιδευτικό ζήτημα που απλώς προσπαθεί να μεταδώσει πιο ολοκληρωμένα της γενική ιδέα του ότι κάποιες κλάσεις λογισμικού έχουν αφετηρία από οντότητες του πραγματικού κόσμου, ενώ κάποιες άλλες είναι «εφευρέσεις» των μηχανικών λογισμικού για την δική τους ευκολία και εξυπηρέτηση συγκεκριμένων στόχων.

Πρέπει να τονιστεί ότι μερικές φορές η λύση που προτείνει το πληροφοριακό πρότυπο Expert δεν είναι επιθυμητή. Αν και το αντικείμενο είναι υποψήφιο για να αναλάβει την ευθύνη ελέγχου, επειδή κατέχει μεγάλο ποσοστό της απαιτούμενης πληροφορίας, παρόλα αυτά, ο σχεδιασμός αυτός οδηγεί σε προβλήματα με το cohesion και το coupling.

Ωφέλειες

- Το πρότυπο οδηγεί και σε υψηλή συνοχή cohesion επειδή οι αρμοδιότητες κατανέμονται σε κλάσεις που ασχολούνται με συγκεκριμένο πλήθος συγγενών μεταξύ τους λειτουργιών
- Η πιθανότητα επαναχρησιμοποίησης αυξάνεται γιατί η ύπαρξη των κλάσεων που δημιουργήθηκαν από το τρέχον πρότυπο έχουν τέτοιες αρμοδιότητες που θα φανούν χρήσιμες σε πολλαπλές περιστάσεις.

Αντιλογίες

Μερικές φορές γίνεται κατάχρηση της συμπεριφοριστικής αναδόμησης από μηχανικούς άπειρους στον σχεδιασμό λογισμικού που προσπαθούν να

αντιστοιχήσουν πιστά τον πραγματικό κόσμο. Πολλές φορές οι λειτουργίες υποβιβάζονται σε αντικείμενα, το οποίο δεν είναι καταρχήν κακό, αρκεί να υπάρχει μία ισορροπία με την δημιουργία αντικειμένων βάσει της αναπαραστατικής αναδόμησης, δηλαδή για παράδειγμα η κλάση Sale πρέπει να έχει αρμοδιότητες εκτός από αυτή καθεαυτή την καταγραφή μίας πώλησης.

Εάν γίνει κατάχρηση του προτύπου, θα υπάρχουν πολλά συμπεριφοριστικά αντικείμενα που έχουν αρμοδιότητες που δεν συνδέονται με την πληροφορία που κατέχουν, το οποίο με την σειρά του θα επηρεάσει την σύζευξη.

Σχετικά Πρότυπα

- Χαμηλής Σύζευξης Low Coupling
- Υψηλής Συνοχής High Cohesion

ΥΠΟΚΕΦΑΛΑΙΟ 5.10 Έμμεσο Πρότυπο GRASP Pattern – Indirection

Το πρόβλημα:

Που πρέπει να ανατεθεί η αρμοδιότητα ώστε να αποφευχθεί η χαμηλή σύζευξη μεταξύ δύο (ή περισσοτέρων) οντοτήτων; Πώς να χαμηλώσει η σύζευξη των αντικειμένων και παράλληλα να υποστηρίζεται η υψηλή δυνατότητα επαναχρησιμοποίησης;

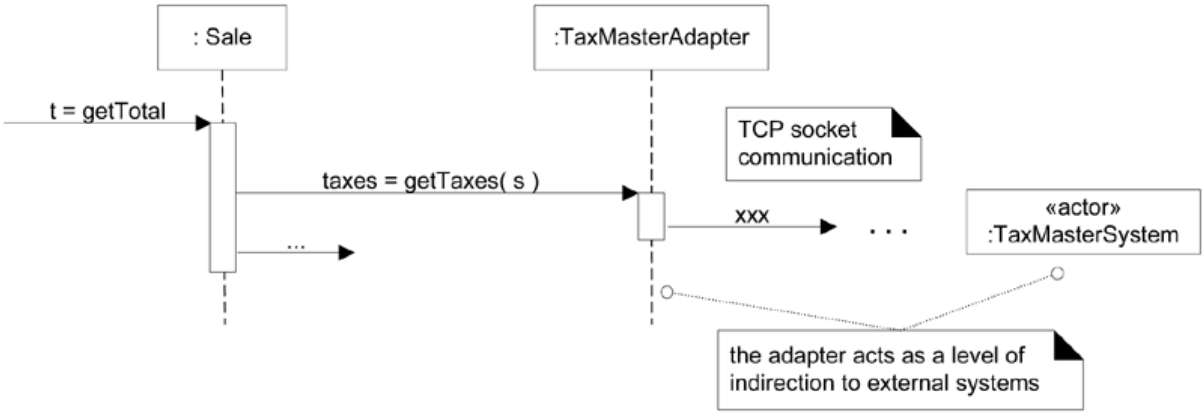
Η λύση:

Ανάθεση της αρμοδιότητας σε ένα ενδιάμεσο αντικείμενο ως μεσολαβητή μεταξύ αντικειμένων ή υπηρεσιών ώστε να μην έχουν υψηλή σύζευξη.

Αυτό το αντικείμενο είναι ένας ενδιάμεσος (intermediate).

Παράδειγμα:

Όπως διακρίνεται στο σχήμα 38, μέσω του πολυμορφισμού και του επιπλέον επιπέδου του ενδιάμεσου, αποκρύπτεται και προστατεύεται ο εσωτερικός σχεδιασμός απέναντι σε πιθανές παραλλαγές των εξωτερικών διεπαφών.



Σχήμα 38: «Παράδειγμα προτύπου Creator στο POS

Στο σχήμα 38, το σημαντικό είναι η εφαρμογή TaxMaster που εξυπηρετείται απομακρυσμένα, μέσω του ενδιάμεσου, χωρίς να ενδιαφέρεται κανείς να γνωρίζει τις λεπτομέρειες υλοποίησης των επιμέρους τμημάτων της διεπαφής.

Ωφέλειες

Χαμηλότερη Σϋζευξη μεταξύ των συστατικών μερών

Σχετικά Πρότυπα

- Προστατευμένων Παραλλαγών Protected Variations
- Χαμηλής Σύζευξης Low Coupling

ΥΠΟΚΕΦΑΛΑΙΟ 5.11 Πρότυπο Προστατευμένων Παραλλαγών GRASP Pattern – Protected Variation

Το πρόβλημα:

Πώς να σχεδιάζονται αντικείμενα, υποσυστήματα και συστήματα ώστε οι όποιες οι παραλλαγές, ή οι όποιες αστάθειες σε αυτά να μην επηρεάζουν άλλα στοιχεία του λογισμικού;

Η λύση:

Ταυτοποίηση των σημείων που επιφέρουν παραλλαγές ή αστάθεια. Ανάθεση της αρμοδιότητας δημιουργίας μίας σταθερής διεπαφής γύρω τους.

Εδώ ο όρος διεπαφή έχει γενική έννοια. Είναι το μέσον, ο μοχλός σύνδεσης με το αντικείμενο και δεν ταυτίζεται με την γραφική διεπαφή για παράδειγμα της Java.

Παράδειγμα:

Στο προηγούμενο σύστημα όπου αναφέρθηκε το εξωτερικό πρόγραμμα υπολογισμού κρατήσεων στο σχήμα 33, το σημείο που επιφέρει αστάθεια ή παραλλαγές, είναι οι πολλές διαφορετικές διεπαφές με τα API των εξωτερικών εφαρμογών. Το POS χρειάζεται να μπορεί να συνδεθεί απρόσκοπτα με πολλά υπάρχοντα προγράμματα υπολογισμού κρατήσεων και προφανώς και με μελλοντικά τέτοια προγράμματα.

Προσθέτοντας ένα επίπεδο indirection, μία διεπαφή, και χρησιμοποιώντας πολυμορφισμό με τις διάφορες υλοποιήσεις του ITaxCalculatorAdapter, υπάρχει προστασία μέσα στο σύστημα από τις παραλλαγές των εξωτερικών API. Τα εσωτερικά αντικείμενα συνεργάζονται με μία σταθερή διεπαφή. Οι διάφορες υλοποιήσεις σύνδεσης (adapters) κρύβουν τις παραλλαγές τους από τα εξωτερικά

συστήματα.

Συζήτηση

Το παρόν πρότυπο είναι μία πολύ σημαντική, θεμελιακή αρχή του σχεδιασμού λογισμικού. Σχεδόν κάθε έργο λογισμικού ή αρχιτεκτονική αρχή όπως η ενθυλάκωση, ο πολυμορφισμός, ο σχεδιασμός οδηγούμενος από τα δεδομένα, οι διεπαφές, οι εικονικές μηχανές, τα αρχεία παραμετροποίησης (configuration files), τα λειτουργικά συστήματα και αναρίθμητα άλλα παραδείγματα είναι μία εξειδίκευση του παρόντος προτύπου.

Αντιλογίες

Δύο σημεία αλλαγών είναι άξια επισήμανσης:

Variation Point: Παραλλαγές σε υπάρχοντα συστήματα ή απαιτήσεις, όπως οι πολλαπλές διεπαφές που απαιτούνται για την υποστήριξη των εξωτερικών προγραμμάτων υπολογισμού κρατήσεων

Evolution Point: Θεωρητικά σημεία παραλλαγών τα οποία μπορεί να ανακύψουν στο μέλλον, και δεν υπάρχουν στις τρέχουσες απαιτήσεις-προδιαγραφές

Το τρέχον πρότυπο εφαρμόζεται και στις δύο παραπάνω περιπτώσεις.

Μερικές φορές η σχεδίαση του συστήματος ώστε να αντιμετωπίζει μελλοντικά προβλήματα όταν και αν παρουσιαστούν, είναι πολύπλοκη και επιφέρει πολυπλοκότητα που ίσως να μην χρειαστεί ποτέ. Ίσως τελικά να είναι καλύτερα το σύστημα να αντιμετωπίζει τα σημερινά προβλήματα σε έναν καθαρογραμμένο και συμπαγές σύνολο, και να τροποποιηθεί στο μέλλον, όταν και αν παρουσιαστεί σχετική ανάγκη.

Ωφέλειες

    - Οι όποιες επεκτάσεις του συστήματος για καινούριες παραλλαγές γίνονται εύκολα
- Καινούριες υλοποιήσεις μπορούν να εισαχθούν χωρίς να επηρεάσουν τις εφαρμογές – «πελάτες»
- Επηρεάζει την Σύζευξη προς τα κάτω (δηλαδή θετικά)
- Ο αντίκτυπος του κόστους των αλλαγών μειώνεται

Σχετικά Πρότυπα

- Τα περισσότερα πρότυπα του GRASP, αλλά και γενικά άλλα πρότυπα σχεδίασης.

ΕΠΙΛΟΓΟΣ

Στο παρόν κεφάλαιο παρουσιάστηκαν τα εννέα πρότυπα σχεδίασης GRASP με

όλες τις παραμέτρους τους. Για το καθένα έγινε λεπτομερής περιγραφή, αναλύθηκαν κάποια χαρακτηριστικά παραδείγματα και μέσω αυτών περιγράφηκαν κάποιες χαρακτηριστικές περιπτώσεις «καλού» και «κακού» σχεδιασμού σε ένα έργο λογισμικού.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Τα σχεδιαστικά πρότυπα γενικώς είναι μία γενική επαναχρησιμοποιούμενη λύση σε συχνά εμφανιζόμενα προβλήματα στην συγγραφή λογισμικού. Ένα πρότυπο δεν είναι μία ολοκληρωμένη και λεπτομερής πρόταση σχεδιασμού λογισμικού, αλλά μία περιγραφή ή αλλιώς ένας οδηγός επίλυσης ενός συγκεκριμένου προβλήματος ή μίας οικογένειας προβλημάτων που μπορούν να εμφανιστούν κάτω από διαφορετικές συνθήκες. Τα σχεδιαστικά πρότυπα στον αντικειμενοστρεφή σχεδιασμό συνήθως δείχνουν σχέσεις και αλληλεπιδράσεις μεταξύ κλάσεων και αντικειμένων χωρίς να ορίζουν σαφώς τις τελικές κλάσεις ή αντικείμενα πάνω στα οποία επιδρούν. Υπάρχουν αρκετά σχεδιαστικά πρότυπα τα οποία κατά βάση ανήκουν σε τέσσερεις οικογένειες: Κατασκευαστικά (Creational), Δομικά (Structural) , Συμπεριφορικά (Behavioral) και GRASP.

Τα σχεδιαστικά πρότυπα GRASP αποτελούν ένα εξαιρετικό και αυτόνομα εργαλείο για κάθε μηχανικό λογισμικού, ο οποίος μπορεί να βασιστεί πάνω τους αναπτύσσοντας ένα οποιοδήποτε έργο λογισμικού, μικρό ή μεγάλο.

Τα εννέα αυτά πρότυπα, αν ληφθούν υπόψη κατά την διάρκεια όλων των φάσεων της ανάπτυξης του λογισμικού, εξασφαλίζουν μία σειρά από ωφέλειες τόσο για τους προγραμματιστές, αλλά και για τους χειριστές και τους τελικούς αποδέκτες του λογισμικού.

Επίσης κάνοντας χρήση των προτύπων, αναδεικνύονται μία σειρά από κέρδη, τόσο στην ταχύτητα, σταθερότητα και αρτιότητα της υλοποίησης, αλλά επίσης το λογισμικό γίνεται ευεπίφορο σε αλλαγές, προσθήκες, μελλοντικές τροποποιήσεις λόγων αλλαγών, απαιτήσεων συμμόρφωσης, όπως επίσης και αυξάνουν κατά πολύ την ανοχή του λογισμικού σε κακή χρήση, λάθη. Λιγοστεύουν τις αστάθειες, τα σημεία όπου μπορούν να συμβεί μη ορθή χρήση, ή μη αναμενόμενη συμπεριφορά.

ΒΙΒΛΙΟΓΡΑΦΙΑ

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., & Angel, S. (Συγγρ.). (1977). *A pattern language*. Oxford Univ. Pr.
Booch, G. (Συγγρ.). (1990). *Object oriented design with applications*. Benjamin-Cummings Publishing Co., Inc.
Brooks Jr, F. P. (Συγγρ.). (1995). *The mythical man-month*. Pearson Education India.
Constantine, L. L. (Συγγρ.). (1994). Essentially Speaking. *Software Development*, *2*(11), 95–96.
Constantine, L. L., & Lockwood, L. A. . (Συγγρ.). (1999). *Software for use: a practical guide to the models and methods of usage-centered design*. Addison-Wesley.
Curtis, B. (Συγγρ.). (1989). *Cognitive issues in reusing software artifacts, Software reusability: vol. 2, applications and experience*. ACM Press, New York, NY.
Deligiannis, I., Sfetsos, P., & Chatzigeorgiou, A. (Συγγρ.). (2008). A controlled experiment investigation of Behavior allocation in two Object-Oriented methods.
Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (Συγγρ.). (1995). *Design patterns: elements of reusable object-oriented software* (T. 206). Addison-wesley Reading, MA.
Jacobson, I., Christerson, M., Jonsson, P., & Overgaard, G. (Συγγρ.). (1992). *Object-oriented software engineering: a use case driven approach*. Addison-Wesley.
Larman, C. (Συγγρ.). (2002). *Applying UML and patterns: an introduction to object-oriented analysis and design and the unified process* (2ος έκδ.). Prentice Hall.
Larman, Craig (Συγγρ.). (2004). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)* (3ος έκδ.). Prentice Hall PTR.
Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorenson, W. (Συγγρ.). (1991). Object-oriented modeling and design.
Schmidt, D. C. (Συγγρ.). (1995). Using design patterns to develop reusable object-oriented communication software. *Communications of the ACM*, *38*(10), 65–74.
Schmidt, D. C., Stal, M., Rohnert, H., & Buschmann, F. (Συγγρ.). (2000). *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2*. Wiley.
Spohrer, J. C., & Soloway, E. (Συγγρ.). (1986). Novice mistakes: Are the folk wisdoms correct? *Communications of the ACM*, *29*(7), 624–632.
Wikipedia (Συγγρ.). (2011). GRASP (object-oriented design) - Wikipedia, the free encyclopedia. Ανακτήθηκε Οκτώβριος 4, 2011, από http://en.wikipedia.org/wiki/GRASP_(object-oriented_design)
Wirfs-Brock, R., & McKean, A. (Συγγρ.). (2003). *Object design: roles, responsibilities, and collaborations*. Addison-Wesley Professional.