ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Design Patterns & Anti-Patterns

ΠΡΟΤΥΠΑ ΚΑΙ ΑΝΤΙ-ΠΡΟΤΥΠΑ ΣΧΕΔΙΑΣΗΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ
ΠΑΡΟΥΣΙΑΣΗ, ΑΝΑΛΥΣΗ, ΥΛΟΠΟΙΗΣΗ

Πτυχιακή Εργασία

Χαράλαμπος Πουρνάρης (03/2336)

Επιβλέπον καθηγητής: Δεληγιάννης Ιγνάτιος

ΠΕΡΙΕΧΟΜΕΝΑ

Πρόλογος

Η εργασία αυτή ασχολείται με την ανάλυση μεθόδων που σχετίζονται με την ανάπτυξη λογισμικού σε περιβάλλον αντικειμενοστραφούς προγραμματισμού. Συγκεκριμένα παρουσιάζει τα πρότυπα σχεδίασης (Design Patterns) που αποτελούν δοκιμασμένες, αξιόπιστες, επεκτάσιμες και επαναχρησιμοποιήσιμες μεθόδους σχεδίασης συστημάτων. Τα πρότυπα αυτά έχουν αναπτυχθεί από ειδήμονες πάνω στον τομέα του αντικειμενοστραφούς προγραμματισμού και από την εμπειρία που έχουν αποκομίσει από την συνεχή ενασχόλησή τους με τον σχεδιασμό επεκτάσιμων συστημάτων και την αντιμετώπιση συνηθισμένων δυσκολιών που παρουσιάζονται σε αυτήν την διαδικασία, της συγγραφής δηλαδή ποιοτικού κώδικα και κατ επέκταση ποιοτικών και επεκτάσιμων συστημάτων λογισμικού.

Επιπρόσθετα με τα πρότυπα σχεδίασης που αποτελούν τον ενδεδειγμένο και αποτελεσματικό τρόπο σχεδίασης, γίνεται προσπάθεια παρουσίασης των **μη** ενδεδειγμένων τρόπων σχεδίασης (Anti-Patterns) έτσι ώστε ο αναγνώστης να είναι σε θέση να εντοπίζει ευκολότερα τα προβληματικά μέρη στην σχεδίασή του και να αποφεύγει την συγγραφή μονολιθικών συστημάτων που παρουσιάζουν δυσκολίες στην συντήρηση, επέκταση, κατανόηση και αποσφαλμάτωσή τους.

Η παρουσίαση των προτύπων συνοδεύεται από κώδικα για το καθένα με σκοπό την καλύτερη αφομοίωση τους από τον αναγνώστη όπως επίσης και από διαγράμματα UML που απεικονίζουν κλάσεις, αντικείμενα και τις συσχετίσεις που έχουν μεταξύ τους. Αυτό φυσικά προϋποθέτει την γνώση βασικών εννοιών και συμβολισμών της UML (**U**nified **M**odeling **L**anguage) από τον αναγνώστη για την ορθή κατανόησή τους. Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε για την συγγραφή του κώδικα είναι η JAVA, χωρίς αυτό να σημαίνει ότι τα πρότυπα σχεδίασης περιορίζονται και μπορούν να υλοποιηθούν σε μία συγκεκριμένη μόνο γλώσσα προγραμματισμού. Αντιθέτως μπορούν να εφαρμοστούν σε όλες τις γλώσσες που παρέχουν δυνατότητες συγγραφής αντικειμενοστραφούς κώδικα και ανεξάρτητα από το αν η γλώσσα απαιτεί τον ορισμό του τύπου των μεταβλητών πριν την χρήση τους (static typing) η όχι (dynamic typing). Για παράδειγμα τα πρότυπα μπορούν να υλοποιηθούν εξίσου αποτελεσματικά σε γλώσσες προγραμματισμού όπως η C++, Smalltalk, PHP, Python, C# .. η λίστα είναι πολύ μεγάλη για να αναφερθούν όλες εδώ. Κάποιες από αυτές παρέχουν ενσωματωμένες δυνατότητες στην ίδια την γλώσσα που διευκολύνουν την υλοποίηση ορισμένων προτύπων σχεδίασης σε σχέση με άλλες ή μπορεί να παρέχουν ακόμα και έτοιμες υλοποιήσεις (out of the box) για κάποια από αυτά.

Κλείνοντας τον πρόλογο αξίζει να σημειωθεί ότι οι ποιο έμπειροι στον τομέα της σχεδίασης αντικειμενοστραφών συστημάτων, όταν καλούνται να αξιολογήσουν την ποιότητα ενός τέτοιου συστήματος, αναζητούν σε ποιο βαθμό γίνεται χρήση των προτύπων σχεδίασης σε αυτό. Η γνώση των προτύπων σχεδίασης αποτελεί έναν πολύ σημαντικό και κρίσιμο παράγοντα για την σωστή και αποτελεσματική ανάπτυξη απλούστερων ή πολυπλοκότερων συστημάτων λογισμικού για όποιον επιθυμεί να ασχοληθεί σοβαρά με αυτόν τον τομέα και για την εις βάθος εκμάθηση και σωστή χρήση τους απαιτείται συνεχής και προσεκτική ενασχόληση.

Abstract

This text describes Design Patterns and how one can use them to create flexible, extensible and reusable object oriented software. Design patterns can solve commonly occurring problems placing particular emphasis on the context and forces surrounding the problem, the consequences, trade-offs and impact of the solution. Seven widely used design patterns are described, analyzed and implemented: Composite, Adapter, Bridge, Abstract Factory, Strategy, Template Method, Decorator. For each design pattern there is a description with a simple example and usage guidance, a real example with sample code implemented in the Java programming language and UML class diagrams which shows the Structure of a particular Pattern.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω όλους τους καθηγητές μου στο ΤΕΙ Θεσσαλονίκης για τις γνώσεις που μου προσέφεραν και μου προσφέρουν όλα αυτά τα χρόνια και ιδιαίτερα τους Δεληγιάννη Ιγνάτιο, Αμπατζόγλου Απόστολο και Αδαμίδη Παναγιώτη για τις γνώσεις που απέκτησα στον τομέα του προγραμματισμού και την μηχανική λογισμικού/αντικειμενοστραφή σχεδίαση.

1. Εισαγωγή

1.1 Τι είναι τα πρότυπα σχεδίασης και σε τι μας χρησιμεύουν

Τα πρότυπα σχεδίασης παρουσιάζουν μεθόδους και τεχνικές επίλυσης κάποιον επαναλαμβανόμενων προβλημάτων που παρουσιάζονται κατά την σχεδίαση συστημάτων λογισμικού, με συνεπή και συστηματικό τρόπο. Τα πρότυπα παρουσιάζουν συνηθισμένες λύσεις σε συχνά προβλήματα σχεδίασης και έχουν αναπτυχθεί τις τελευταίες δεκαετίες με βάση την εμπειρία και την προσπάθεια ανάπτυξης ποιοτικού λογισμικού από την κοινότητα που ασχολείται με τον αντικειμενοστραφή προγραμματισμό.

Τα πρότυπα σχεδίασης αποτελούνται από τα εξής στοιχεία:

- Κάθε πρότυπο έχει ένα **όνομα** (pattern name) που σχετίζεται άμεσα με την λειτουργικότητα που παρέχει και εμπλουτίζει το λεξιλόγιο μας αναφορικά με την σχεδίαση, δίνοντας μας την δυνατότητα να μιλάμε για αυτά με τους συναδέλφους μας και με άλλους προγραμματιστές και να «καταλαβαινόμαστε», καθώς επίσης και να τα χρησιμοποιούμαι σε εγχειρίδια για την ευκολότερη και γρηγορότερη κατανόησή τους από άλλους. Εν ολίγοις με την χρήση του ονόματος ενός προτύπου σχεδίασης περιγράφουμε ένα πρόβλημα, την λύση του και τις επιπτώσεις που επιφέρει η χρήση του σε ένα σύστημα.

- Το **πρόβλημα** (design problem) που περιγράφει τις περιπτώσεις στις οποίες θα πρέπει να γίνεται η χρήση του προτύπου. Πολλές μορφές σχεδίασης, κυρίως από τους νεότερους προγραμματιστές παρουσιάζουν αδυναμίες που μπορεί να προκαλέσουν ανεπιθύμητες συνέπειες στην λειτουργία και συντήρηση ενός συστήματος. Τα πρότυπα μας βοηθούν στον εντοπισμό αυτών των προβληματικών σημείων της σχεδίασης και μας παρέχουν αποτελεσματικές και δοκιμασμένες λύσεις.

- Τα πρότυπα παρέχουν μια **λύση** (solution) σε ένα σχεδιαστικό πρόβλημα περιγράφοντας τα στοιχεία που αποτελούν το σχέδιο, τις αρμοδιότητες τους, και τον τρόπο επικοινωνίας και συνεργασίας μεταξύ τους (τις κλάσεις, τα αντικείμενα και τις συσχετίσεις μεταξύ τους στην περίπτωσή μας). Οι λύσεις που προτείνουν είναι γενικές και αφηρημένες, όπως ακριβώς είναι και τα προβλήματα που επιχειρούν να επιλύσουν έτσι ώστε να καλύπτουν ένα ευρύτερο φάσμα παρόμοιων περιπτώσεων, που όμως κάθε φορά οι λεπτομέρειες τους διαφέρουν.

- Οι **επιπτώσεις** (consequences) από την χρήση τους, που σχετίζονται με το κόστος αλλά και τα ωφέλει που επιφέρουν στην σχεδίαση του συστήματος. Συγκεκριμένα στην ανάπτυξη λογισμικού οι επιπτώσεις αφορούν την ανταλλαγή που γίνεται μεταξύ ταχύτητας εκτέλεσης και του απαιτούμενου χώρου. Λόγω του ότι η επαναχρησιμοποίηση μονάδων λογισμικού αποτελεί σημαντικό παράγοντα στην αντικειμενοστραφή σχεδίαση, οι επιπτώσεις της χρήσης των προτύπων εστιάζουν στο κατά πόσο επηρεάζεται η επεκτασιμότητα, ευελιξία και μεταφερσιμότητα του συστήματος.

Διαφορετικοί προγραμματιστές ενδέχεται να αντιλαμβάνονται διαφορετικά την χρήση των προτύπων σχεδίασης. Για παράδειγμα κάποιος λιγότερο έμπειρος προγραμματιστής μπορεί να δυσκολεύεται στην κατανόηση και την χρήση ενός προτύπου ενώ για κάποιον ποιο έμπειρο να είναι αυτονόητη και εύκολα προσαρμόσιμη (στο συγκεκριμένο πλαίσιο σχεδίασης) η χρήση και λειτουργία του.

Είναι πολύ συνηθισμένο οι νέοι προγραμματιστές να καταφεύγουν σε εύκολες σχεδιάστηκες λύσεις που ναι μεν ικανοποιούν μια συγκεκριμένη ανάγκη εκείνη την χρονική στιγμή, αλλά μελλοντικά καταστούν την συντήρηση, την επεκτασιμότητα του συστήματος αλλά και την ευελιξία στην ικανοποίηση νέων απαιτήσεων δυσκολότερη. Τα πρότυπα σχεδίασης καθοδηγούν τον προγραμματιστή με την δομή που προτείνουν έτσι ώστε η τελική σχεδίαση να είναι ευέλικτη, επεκτάσιμη και κυρίως επαναχρησιμοποιήσιμη.

Ένας από τους σημαντικότερους παράγοντες που επηρέασαν και έδωσαν το έναυσμα για την δημιουργία των προτύπων σχεδίασης στον αντικειμενοστραφή προγραμματισμό ήταν η μελέτη του Christopher Alexander πάνω στην αρχιτεκτονική των κτηρίων. Μελέτησε τις διαφορές που εμφάνιζαν οι «καλές» αρχιτεκτονικές σε σχέση με τις «κακές» και δημιούργησε τα πρότυπα που μπορούσαν να εφαρμοστούν και να δώσουν λύσεις σε επαναλαμβανόμενα προβλήματα. Η σημαντικότερη προσπάθεια αναφορικά με τα πρότυπα σχεδίασης

για την κατασκευή λογισμικού έγινε από τους Erich Gamma, Richard Helm, Ralph Johnson και John Vlissides με την συγγραφή του βιβλίου Design Patterns: Elements of Reusable Object-Oriented Software όπου καταγράφουν και επεξηγούν λεπτομερειακά 23 πρότυπα σχεδίασης. Πολλοί τους κατονομάζουν σαν «η συμμορία των τεσσάρων» (Gang Of Four ή GoF).

Τα πρότυπα σχεδίασης βρίσκουν μεγάλη εφαρμογή σε αρχιτεκτονικές όπως είναι το Μοντέλο – Όψη - Ελεγκτής (Model View Controller ή MVC). Το μοντέλο αυτό χρησιμοποιείται κυρίως για την ανάπτυξη γραφικών διεπαφών χρήστη και επιτρέπει τον σαφή διαχωρισμό των δεδομένων από την εμφάνισή τους. Κάθε φορά που γίνεται τροποποίηση των δεδομένων αυτόματα ενημερώνεται και η διεπαφή, επίσης τα ***ίδια*** δεδομένα μπορούν να εμφανίζονται με διαφορετικό τρόπο στον χρήστη και ο τρόπος με τον οποίο γίνεται η επικοινωνία μεταξύ διεπαφής και χρήστη μπορεί να μεταβάλλεται δυναμικά αλλάζοντας απλός τα αντικείμενα που υλοποιούν τους ελεγκτές. Όλα αυτά επιτυγχάνονται με την χρήση αρκετών συνήθως προτύπων σχεδίασης. Ορισμένα παραδείγματα τέτοιων αρχιτεκτονικών στα οποία γίνεται εκτενής χρήση των προτύπων είναι το MVC της Smalltalk και το SWING της JAVA.

Η περιγραφή των προτύπων σχεδίασης σε αυτήν την εργασία παρουσιάζει μία συγκεκριμένη δομή. Αρχικά γίνεται μια περιγραφή του προτύπου και του προβλήματος που καλείτε να επιλύσει, ακολουθεί ένα παράδειγμα που δείχνει σε ποιές περιπτώσεις θα μπορούσε να χρησιμοποιηθεί καθώς και η υλοποίηση του παραδείγματος σε κώδικα JAVA, που μπορεί να μεταγλωττιστεί και να εκτελεστεί. Επιπρόσθετα σε κάθε πρότυπο γίνεται αναφορά στον λανθασμένο τρόπο επίλυσης του προβλήματος που καταφεύγουν συνήθως οι προγραμματιστές που δεν γνωρίζουν τα πρότυπα σχεδίασης και παρουσιάζεται το διάγραμμα κλάσεων και συσχετίσεων που αποτελεί την οπτική απεικόνιση του σχεδίου και των αρμοδιοτήτων που έχει κάθε στοιχείο του.

Τα πρότυπα που αναλύονται και παρουσιάζονται είναι τα εξής:

Πίνακας προτύπων σχεδίασης με σύντομη περιγραφή

Σύνθετο (**Composite**)	Σύνθεση αντικειμένων σε δενδρικές δομές για την αναπαράσταση ιεραρχιών μέρος-όλου. Επιτρέπει στα αντικείμενα πελάτες να χειρίζονται μεμονωμένα αντικείμενα και συνθέσεις αντικειμένων με ενιαίο τρόπο.
Προσαρμογέας (**Adapter**)	Μετατροπή της διεπαφής μίας κλάσης στην διεπαφή που αναμένουν οι κλάσεις πελατών. Ο προσαρμογέας επιτρέπει την ανταλλαγή μηνυμάτων μεταξύ κλάσεων που σε διαφορετική περίπτωση θα ήταν αδύνατη λόγο ασυμβατότητας των διεπαφών τους.

Γέφυρα (**Bridge**)	Αποσύνδεση μιας αφαίρεσης από την υλοποίησή της έτσι ώστε να μπορούν να μεταβάλλονται ανεξάρτητα.
Αφηρημένο Εργοστάσιο (**Abstract Factory**)	Παρέχει μια διεπαφή για την δημιουργία οικογενειών αντικειμένων που σχετίζονται η έχουν εξάρτηση μεταξύ τους.
Στρατηγική (**Strategy**)	Ορίζει μια «οικογένεια» αλγορίθμων, ενθυλακώνει τον καθένα από αυτούς και επιτρέπει την εναλλαγή μεταξύ τους. Το πρότυπο Στρατηγική επιτρέπει στον αλγόριθμο να μεταβάλλεται ανεξάρτητα από τις κλάσεις των πελατών που τον χρησιμοποιούν.
Μέθοδος υπόδειγμα (**Template Method**)	Ορίζει τον «σκελετό» ενός αλγορίθμου σε μία λειτουργία, μεταβιβάζοντας ορισμένα βήματα στις υποκλάσεις. Η μέθοδος υπόδειγμα επιτρέπει στις υποκλάσεις να επαναπροσδιορίσουν κάποια βήματα ενός αλγορίθμου χωρίς να αλλάξουν την δομή του.
Διακοσμητής (**Decorator**)	Προσθέτει επιπλέον αρμοδιότητες σε ένα αντικείμενο δυναμικά κατά τον χρόνο εκτέλεσης. Οι Διακοσμητές παρέχουν μια ευέλικτη εναλλακτική λύση αντί των υποκλάσεων για την επέκταση της λειτουργικότητας.

1.2 Κατηγοριοποίηση προτύπων σχεδίασης

Στο βιβλίο της «συμμορίας των τεσσάρων» τα πρότυπα σχεδίασης ομαδοποιούνται με βάση την λειτουργία, την γενικότητα και τον σκοπό τους σε κατηγορίες έτσι ώστε αφενός να είναι ευκολότερη η εκμάθηση τους και αφετέρου να κατευθύνουν τις προσπάθειες για την εύρεση νέων προτύπων.

Η κατηγοριοποίηση τους γίνεται σύμφωνα με δύο κριτήρια:

- Με βάση στον σκοπό (**purpose**) τους
- Ανάλογα με το πεδίο εφαρμογής (**scope**) τους

Για το πρώτο κριτήριο, ο σκοπός του προτύπου αντικατοπτρίζει το τι κάνει και μπορεί να σχετίζεται είτε με την δημιουργία (**creational**), την δομή (**structural**) ή την συμπεριφορά (**behavioral**) του. Τα «δημιουργικά» πρότυπα ασχολούνται με την διαδικασία της δημιουργίας αντικειμένων. Τα «δομικά» πρότυπα αντιμετωπίζουν το πρόβλημα της σύνθεσης των αντικειμένων, ενώ τα πρότυπα «συμπεριφοράς» καθορίζουν τον τρόπο με τον οποίο γίνεται η επικοινωνία μεταξύ τους και η διανομή των εργασιών που καλούνται να επιτελέσουν.

Κάθε πρότυπο μπορεί να εφαρμόζεται (κυρίως) είτε σε κλάσεις, ή σε αντικείμενα και το δεύτερο κριτήριο αναφέρεται σε αυτόν ακριβώς τον διαχωρισμό. Τα πρότυπα που εφαρμόζονται κυρίως σε κλάσεις ασχολούνται με τις σχέσεις μεταξύ των κλάσεων και των υποκλάσεων, ενώ οι σχέσεις αυτές είναι στατικές και δεν αλλάζουν κατά τον χρόνο εκτέλεσης. Σε αντίθεση, τα πρότυπα που εφαρμόζονται κυρίως σε αντικείμενα αφορούν τις σχέσεις και την αλληλεπίδραση των αντικειμένων κατά τον χρόνο εκτέλεσης.

1.3 Πώς τα πρότυπα σχεδίασης λύνουν προβλήματα σχεδίασης

Η χρήση των προτύπων κατά την σχεδίαση μεγάλων και πολύπλοκων (κυρίως) συστημάτων μας βοηθά στο να το σχεδιάσουμε με τέτοιο τρόπο ώστε να είναι ευέλικτο, επεκτάσιμο και οι μελλοντικές αλλαγές στο σύστημα να μην απαιτούν τον επανασχεδιασμό μεγάλων τμημάτων του ή άλλων μονάδων λογισμικού που εξαρτώνται άμεσα από αυτό.

Τα πρότυπα μας βοηθούν να εντοπίσουμε τα αντικείμενα του συστήματος και τον τρόπο με τον οποίο πρέπει να επικοινωνούν μεταξύ τους και να ανταλλάσουν μηνύματα. Επιπλέον μας διευκολύνουν να αποφασίσουμε για το μέγεθος και τις αρμοδιότητες που θα έχει το καθένα.

Σε προγραμματιστικό επίπεδο, λαμβάνοντας υπόψη μας τα πρότυπα σχεδίασης μπορούμε να προσδιορίσουμε ευκολότερα τις διεπαφές των αντικειμένων και τον τρόπο με τον οποίο θα πρέπει να υλοποιηθούν για να είναι αποτελεσματική η επικοινωνία μεταξύ τους καθώς και με τα αντικείμενα των πελατών. Ο όρος (αντικείμενα) πελάτες αφορά τις κλάσεις και τα αντικείμενα που δεν ανήκουν στο ίδιο το πρότυπο και δεν συντελούν άμεσα στην λειτουργία του, στις περισσότερες τουλάχιστον περιπτώσεις, αλλά το χρησιμοποιούν και έχουν άμεση εξάρτηση από αυτό και από τις διεπαφές των αντικειμένων του. Για τον λόγο αυτό ο προσδιορισμός τον διεπαφών θα πρέπει να γίνεται πολύ προσεκτικά και με τέτοιο τρόπο ώστε οι μελλοντικές αλλαγές να μην απαιτούν τροποποιήσεις σε αυτές.

1.4 Βασικές αρχές αντικειμενοστραφούς προγραμματισμού

Παρακάτω γίνεται περιγραφή ορισμένων αρχών που βρίσκουν εφαρμογή στον αντικειμενοστραφή προγραμματισμό και αποτελούν χρήσιμες συμβουλές κατά την ανάπτυξη επεκτάσιμων συστημάτων που είναι εύκολα στην συντήρηση «ανοικτά» σε αλλαγές.

Προγραμματίστε στην διεπαφή, όχι στην υλοποίηση

Η παραπάνω αρχή αναφέρεται στην χρήση διεπαφών (**interface**) ή αφηρημένων κλάσεων (**abstract class**) σε κάποιες γλώσσες προγραμματισμού (που δεν διαχωρίζουν μια διεπαφή από την υλοποίησή της), τις οποίες και θα κληρονομήσουν οι κλάσεις που θα υλοποιήσουν τις αντίστοιχες λειτουργίες. Αυτό εξασφαλίζει ότι οι κλάσεις των πελατών θα είναι συμβατές με όλα τα αντικείμενα

που υλοποιούν την διεπαφή (ή την αφηρημένη κλάση), χωρίς να γνωρίζουν ποια ακριβώς υλοποίηση έχει κάθε αντικείμενο.

Για παράδειγμα έστω μια κλάση πελάτης (EncryptExample σε αυτήν την περίπτωση) η οποία κωδικοποιεί δεδομένα (συμβολοσειρές) έτσι ώστε να μπορεί να τα μεταφέρει αργότερα σε κάποιο άλλο σύστημα με ασφαλή τρόπο. Η διεπαφή Crypt δηλώνει τις λειτουργίες που θα πρέπει να είναι σε θέση να εκτελέσουν τα αντικείμενα (αλγόριθμοι) που θα την υλοποιήσουν. Δηλώνει δηλαδή έναν τύπο (**Type**) με την ονομασία Crypt. Η κλάση Blowfish υλοποιεί την διεπαφή Crypt χρησιμοποιώντας έναν συγκεκριμένο αλγόριθμο για την κωδικοποίηση και αποκωδικοποίηση. Η κλάση πελάτης (EncryptExample) διατηρεί μια αναφορά τύπου Crypt, την οποία και χρησιμοποιεί για να εκτελέσει τις διαδικασίες της κωδικοποίησης και αποκωδικοποίησης.

Παρακάτω παρουσιάζεται ο κώδικας (Οι μέθοδοι της κωδικοποίησης και αποκωδικοποίησης δεν έχουν υλοποιηθεί αλλά προσομοιωθεί) :

```
interface Crypt {
    byte[] encrypt(String plain_text);
    String decrypt(byte[] bytes);
}

class Blowfish implements Crypt {
    public byte[] encrypt(String plain_text) {
        byte[] encrypted = null;
        System.out.println("Encrypting using Blowfish: "+plain_text);
        // Implementation below..

        return encrypted;
    }

    public String decrypt(byte[] bytes) {
        String decrypted = null;

        System.out.println("Decrypting bytes using Blowfish.");
        // Implementation below..

        return decrypted;
    }
}

public class EncryptExample {
    public static void main(String[] args) {
        Crypt cr = new Blowfish();

        cr.encrypt("Test");
        cr.decrypt(new byte[0]);
    }

}
```

Με αυτόν τον τρόπο το αντικείμενο πελάτης δεν είναι απαραίτητο να γνωρίζει κάθε φορά τον συγκεκριμένο αλγόριθμο που χρησιμοποιείτε παρά μόνον να «κρατάει» μια αναφορά στον τύπο Crypt. Εάν αργότερα παρουσιάζονταν η ανάγκη να προστεθεί ένας επιπλέον αλγόριθμος (πχ. RC4) δεν θα χρειαζόταν ιδιαίτερες αλλαγές στο σύστημά μας καθώς θα μπορούσε να γίνει με την προσθήκη μίας νέας κλάσης που υλοποιεί την διεπαφή Crypt:

```
class RC4 implements Crypt {
    public byte[] encrypt(String plain_text) {
        byte[] encrypted = null;
        System.out.println("Encrypting using RC4: "+plain_text);
        // Implementation below..

        return encrypted;
    }

    public String decrypt(byte[] bytes) {
        String decrypted = null;

        System.out.println("Decrypting bytes using RC4.");
        // Implementation below..

        return decrypted;
    }
}
```

Επιπλέον στο αντικείμενο πελάτης δεν χρειάζεται καμία απολύτως αλλαγή αφού η αναφορά cr (τύπου Crypt) μπορεί να δείχνει σε οποιοδήποτε αντικείμενο υλοποιεί την διεπαφή. Εάν χρησιμοποιούσαμε απευθείας δύο διαφορετικές υλοποιήσεις (Blowfish, RC4) χωρίς την χρήση διεπαφής τότε το αντικείμενο πελάτη θα έπρεπε κάθε φορά να τροποποιεί την αναφορά προς το αντικείμενο που κάνει την κωδικοποίηση/αποκωδικοποίηση ώστε να έχει τον ίδιο τύπο. Επίσης θα ήταν αδύνατο να εντοπιστούν λάθη από τον μεταγλωττιστή όπως η ασυμφωνία στην ονομασία των μεθόδων.

Χρησιμοποιείστε σύνθεση αντικειμένων αντί της κληρονομικότητας

Υπάρχουν δύο βασικοί τρόποι με τους οποίους επιτυγχάνετε η επαναχρησιμοποίηση μονάδων λογισμικού:

Ο πρώτος τρόπος αφορά την χρήση υποκλάσεων (**subclass**) οι οποίες κληρονομούν την λειτουργικότητα τους από τις γονικές κλάσεις (**super class** ή **parent class**). Ο τρόπος αυτός ονομάζεται επαναχρησιμοποίηση λευκού κουτιού (**white-box reuse**) καθώς η εσωτερική κατάσταση των γονικών κλάσεων εκτίθεται στις υποκλάσεις. Ένα μειονέκτημα που παρουσιάζει ο τρόπος αυτός είναι ότι δεν μπορεί να υλοποιηθεί κατά την διάρκεια εκτέλεσης της εφαρμογής καθώς ορίζεται στατικά κατά την μεταγλώττιση του κώδικα. Ένα άλλο μειονέκτημα είναι ότι οι υποκλάσεις είναι πολύ στενά συνδεδεμένες και εξαρτημένες από τις γονικές κλάσεις με αποτέλεσμα να «σπάζει» η ενθυλάκωση.

Ο δεύτερος τρόπος αφορά την χρήση σύνθεσης αντικειμένων έτσι ώστε ένα αντικείμενο να προωθεί (**delegate**) τις αιτήσεις (ή αλλιώς τα μηνύματα) που φτάνουν σε αυτό σε κάποιο άλλο στο οποίο διατηρεί μια αναφορά. Ο τρόπος αυτός αναφέρεται και ως επαναχρησιμοποίηση μαύρου κουτιού (**black-box reuse**) διότι η εσωτερική κατάσταση των αντικειμένων δεν είναι ορατή όπως στην περίπτωση της επαναχρησιμοποίησης λευκού κουτιού. Η χρήση της σύνθεσης αντικειμένων επιτρέπει την εναλλαγή τους κατά τον χρόνο εκτέλεσης αρκεί να υλοποιούν την ίδια διεπαφή, έτσι, η ενθυλάκωση των αντικειμένων παραμένει αναλλοίωτη. Τα αντικείμενα παραμένουν μικρά σε μέγεθος και εστιάζουν σε μια συγκεκριμένη λειτουργία χωρίς να υπάρχει ο φόβος να εξελιχθούν σε τεράστια και ανεξέλεγκτα «τέρατα». Για τους λόγους αυτούς θα πρέπει να προτιμάται η σύνθεση αντικειμένων σε σχέση με την κληρονομικότητα για την ανάπτυξη επαναχρησιμοποιήσιμων μονάδων λογισμικού.

Σχεδιάστε προβλέποντας τις μελλοντικές αλλαγές

Για την επίτευξη του μέγιστου βαθμού επαναχρησιμοποίησης θα πρέπει το σύστημα να είναι έτοιμο να δεχθεί και να αναπροσαρμοστεί σε αλλαγές, καθώς οι απαιτήσεις μεταβάλλονται κατά την διάρκεια ζωής του. Εάν αποτύχει να το κάνει αυτό ενδέχεται να χρειαστούν σημαντικές τροποποιήσεις σε κλάσεις του συστήματος και των πελατών για να ανταπεξέλθει στις νέες απαιτήσεις και το κόστος σε αυτή την περίπτωση είναι πολύ μεγάλο. Θα πρέπει λοιπόν κατά τον σχεδιασμό του συστήματος να λαμβάνονται σοβαρά υπόψη τυχόν αλλαγές που ενδέχεται να χρειαστούν στο μέλλον. Τα πρότυπα σχεδίασης βοηθούν έτσι ώστε το σύστημα να μπορεί να μεταβάλλεται με συγκεκριμένους τρόπους καθιστώντας το περισσότερο ευέλικτο σε κάποια συγκεκριμένη αλλαγή.

Ορισμένοι λόγοι που μπορεί να προκαλέσουν τον επανασχεδιασμό μονάδων του συστήματος είναι οι εξής:

- Δημιουργία αντικειμένων από μια συγκεκριμένη κλάση (concrete class), καθώς δεσμεύεται το αντικείμενο αυτό σε μια συγκεκριμένη υλοποίηση αντί μιας διεπαφής
- Εξάρτηση από το υλικό του συστήματος ή του λειτουργικού συστήματος που «τρέχει»
- Εξάρτηση σε συγκεκριμένους αλγορίθμους
- Μεγάλη σύζευξη μεταξύ των κλάσεων (**tight coupling**)
- Αδυναμία πραγματοποίησης αλλαγών σε κάποια κλάση με τον κατάλληλο τρόπο (πχ. Έλλειψη του πηγαίου κώδικα)
- Επέκταση της λειτουργικότητας με την χρήση της κληρονομικότητας καθώς απαιτείται η σε βάθος γνώση της γονικής κλάσης

1.5 Παραδείγματα κατηγοριών λογισμικού με ευρεία χρήση των προτύπων σχεδίασης

Υπάρχουν τρείς κατηγορίες λογισμικού στις οποίες τα πρότυπα σχεδίασης παίζουν πολύ σημαντικό ρόλο. Οι κατηγορίες αυτές είναι: Προγράμματα εφαρμογών (**Application programs**), Εργαλειοθήκες (**Toolkits**), Πλαίσια λογισμικού (**Frameworks**).

Προγράμματα Εφαρμογών

Η κατηγορία αυτή αναφέρεται κυρίως σε εφαρμογές όπως είναι ένας συντάκτης κειμένου ή μια εφαρμογή λογιστικών φύλλων. Η χρήση των προτύπων σχεδίασης έχει ως αποτέλεσμα την καλύτερη επαναχρησιμοποίηση των μονάδων του λογισμικού, ευκολία στην επέκταση και προσθήκη νέων λειτουργιών καθώς και την αποτελεσματικότερη συντήρηση του. Αυτό συμβαίνει διότι με την χρήση των προτύπων μειώνονται οι εξαρτήσεις που έχουν μεταξύ τους τα αντικείμενα υποστηρίζοντας την μικρότερη δυνατή σύζευξη.

Εργαλειοθήκες

Οι εργαλειοθήκες αποτελούνται από ένα σύνολο κλάσεων και αντικειμένων που σχετίζονται και συνεργάζονται μεταξύ τους για την επίτευξη μιας συγκεκριμένης λειτουργίας έτσι ώστε ο προγραμματιστής να μην ανακαλύπτει τον τροχό από την αρχή γράφοντας κώδικα για βασικές λειτουργίες. Για παράδειγμα κλάσεις που υλοποιούν στοίβες, λίστες ή πίνακες συσχετίσεων ανήκουν σε αυτήν την κατηγορία. Ένα κλασικό παράδειγμα εργαλειοθήκης αποτελούν οι κλάσεις I/O Steam της C++. Η συγγραφή αυτού του είδους λογισμικού είναι δυσκολότερη από τις απλές εφαρμογές καθώς ο προγραμματιστής που το σχεδιάζει δεν είναι σε θέση να γνωρίζει εξ αρχής τι είδους θα είναι οι εφαρμογές στις οποίες θα χρησιμοποιηθεί καθώς είναι γενικού χαρακτήρα.

Πλαίσια Λογισμικού

Τα πλαίσια είναι σύνολα κλάσεων που συνεργάζονται μεταξύ τους δημιουργώντας επαναχρησιμοποιήσιμα σχέδια για ένα συγκεκριμένο είδος λογισμικού. Οι προγραμματιστές που χρησιμοποιούν τα σχέδια αυτά αναλαμβάνουν να υλοποιήσουν κάποιες αφηρημένες κλάσεις, παρέχοντας έτσι υλοποίηση για ένα συγκεκριμένο πεδίο εφαρμογής. Η αρχιτεκτονική της εφαρμογής είναι προσχεδιασμένη όπως επίσης οι ευθύνες και οι σχέσεις μεταξύ των αντικειμένων και η ροή εκτέλεσης της. Σε αντίθεση δηλαδή με τις εργαλειοθήκες όπου ο προγραμματιστής «γράφει» το κυρίως σώμα της εφαρμογής και χρησιμοποιεί ορισμένες από τις κλάσεις της για συγκεκριμένες λειτουργίες, τα πλαίσια λογισμικού καθορίζουν την δομή που θα έχει ολόκληρο το σύστημα και ο προγραμματιστής κληρονομώντας από τις έτοιμες κλάσεις, προσαρμόζει την εφαρμογή κατάλληλα για την επίτευξη των στόχων του. Για αυτούς τους λόγους η

συγγραφή πλαισίων λογισμικού θεωρείτε δυσκολότερη σε σχέση με τις δύο προηγούμενες κατηγορίες.

2. Κατάλογος προτύπων σχεδίασης

2.1 Σύνθετο (Composite)

Περιγραφή

Το πρότυπο σχεδίασης σύνθετο επιτρέπει την *σύνθεση αντικειμένων σε δενδρικές δομές για την αναπαράσταση ιεραρχιών μέρος-όλου. Επιτρέπει στα αντικείμενα πελάτες να χειρίζονται μεμονωμένα αντικείμενα και συνθέσεις αντικειμένων με ενιαίο τρόπο*.

Πολλές φορές κατά την σχεδίαση ενός συστήματος παρατηρούμε ότι μερικά από τα αντικείμενα από τα οποία απαρτίζεται παρουσιάζουν μια σχέση «μέρος-όλο» μεταξύ τους, όπου κάποιο από αυτά περιέχει ή έχει αναφορές προς άλλα αντικείμενα. Για παράδειγμα ένα αντικείμενο τύπου Βιβλιοθήκη θα μπορούσε να περιέχει τις κλάσεις Βιβλίο και Περιοδικό. Η σχέση μεταξύ του αντικειμένου Βιβλιοθήκη και των αντικειμένων που περιλαμβάνει μπορεί να είναι είτε συσσωμάτωση όπου το πρώτο διατηρεί αναφορές προς τα υπόλοιπα αντικείμενα χωρίς όμως να είναι υπεύθυνο για την δημιουργία και καταστροφή τους (στις γλώσσες προγραμματισμού που επιτρέπουν την άμεση διαγραφή των αντικειμένων από την μνήμη) ή σύνθεση όπου το αντικείμενο Βιβλιοθήκη έχει αυτήν την ευθύνη, να δημιουργήσει δηλαδή και να καταστρέψει τα αντικείμενα που περιέχει.

Παρατηρούμε επίσης ότι σε πολλές τέτοιες περιπτώσεις τα αντικείμενα πελάτες χειρίζονται με παρόμοιο τρόπο τόσο τα αντικείμενα που αναπαριστούν το «όλο» όσο και τα αντικείμενα που αναπαριστούν το «μέρος». Οι πελάτες όμως σε αυτήν την περίπτωση θα πρέπει να διατηρούν αναφορές σε όλα τα αντικείμενα για τα οποία ενδιαφέρονται παρόλο που τα χειρίζονται με παρόμοιο τρόπο, για παράδειγμα θα πρέπει να έχουν αναφορές και στις τρείς κλάσεις (Βιβλιοθήκη, Βιβλίο και Περιοδικό). Εάν αργότερα θελήσουμε να προσθέσουμε μια ακόμα κλάση Κόμικ θα πρέπει να τροποποιήσουμε την κλάση Βιβλιοθήκη και τις κλάσεις των πελατών, πράγμα που μπορεί να είναι αδύνατο να γίνει ή να έχει πολύ μεγάλο κόστος. Αυτού του είδους τα προβλήματα καλείτε να επιλύσει το πρότυπο σχεδίασης «Σύνθετο».

Η χρήση του προτύπου θα πρέπει να γίνεται στις εξής περιπτώσεις:

- Όταν θέλουμε να αναπαραστήσουμε ιεραρχίες αντικειμένων που έχουν την μορφή «μέρος-όλο»
- Όταν θέλουμε οι κλάσεις των πελατών να αγνοούν τις διαφορές μεταξύ των πρωταρχικών αντικειμένων και των αντικειμένων που τα περικλείουν και να τα χειρίζονται όλα με παρόμοιο τρόπο

Το πρότυπο «Σύνθετο» ανήκει στην κατηγορία των «δομικών» προτύπων.

Κίνητρο – Παράδειγμα

Ένα σύστημα αρχείων (File System) περιλαμβάνει ένα σύνολο από κόμβους που έχουν ορισμένες ιδιότητες, όπως ένα όνομα, μέγεθος κτλ. και είναι οργανωμένοι σε δενδροειδείς ιεραρχίες. Η αναπαράσταση ενός τέτοιου συστήματος μπορεί να γίνει με φυσικό τρόπο χρησιμοποιώντας το πρότυπο σχεδίασης «Σύνθετο».

Ένας Κόμβος (Node) στο σύστημα μπορεί να αναπαριστά ένα Αρχείο (File) που αποτελεί έναν πρωταρχικό κόμβο ή έναν Κατάλογο (Directory) που μπορεί να περιλαμβάνει πρωταρχικούς κόμβους (αρχεία) ή σύνθετους κόμβους (άλλους καταλόγους). Εκτός από τις ιδιότητες που έχουν οι κόμβοι, περιλαμβάνουν και μία σειρά από λειτουργίες όπως είναι ο υπολογισμός του μεγέθους τους, είτε πρόκειται για κόμβο-αρχείο ή για κόμβο-κατάλογο, η εκτύπωση τους με την μορφή που είναι οργανωμένοι στην μνήμη του υπολογιστή κατά τον χρόνο εκτέλεσης και η διαχείριση (προσθήκη, αφαίρεση) κόμβων, για τους σύνθετους κόμβους.

Σκοπός του συστήματος είναι να παρέχει διαφάνεια στον τρόπο με τον οποίο οι κλάσεις των πελατών αναφέρονται, χειρίζονται και καλούν λειτουργίες στους κόμβους, χωρίς να χρειάζεται να γνωρίζουν τι είδους είναι και το συγκεκριμένο αντικείμενο από το οποίο υλοποιείτε ο καθένας, να μπορούν δηλαδή να χειριστούν όλους τους κόμβους ομοιόμορφα. Τα πλεονεκτήματα μίας τέτοιας σχεδίασης γίνονται άμεσα ορατά καθώς οι κλάσεις-πελάτες δεν χρειάζεται να γνωρίζουν τις λεπτομέρειες υλοποίησης ενός αρχείου ή καταλόγου για να εκτελέσουν κάποιες κοινές λειτουργίες τους. Για να το καταφέρουμε αυτό θα πρέπει να ορίσουμε μία αφηρημένη κλάση (ή μια διεπαφή) Κόμβος, στιγμιότυπα της οποίας δεν θα μπορούν να δημιουργηθούν, αλλά θα παρέχει μια προκαθορισμένη (default) υλοποίηση κάποιον μεθόδων. Οι κλάσεις Αρχείο και Κατάλογος κληρονομούν τις ιδιότητες και λειτουργίες ενός κόμβου, παρέχοντας συγκεκριμένη υλοποίηση για τις μεθόδους. Επιπρόσθετα, η κλάση Κατάλογος υλοποιεί μεθόδους που αφορούν την διαχείριση άλλων κόμβων, σε αντίθεση με την κλάση Αρχείο που κάτι τέτοιο δεν θα είχε νόημα και για τον λόγο αυτό εκτελεί την προκαθορισμένη υλοποίηση που ορίζει η αφηρημένη κλάση Κόμβος.

Παρακάτω γίνεται παρουσίαση με κώδικα (γραμμένο σε JAVA) του προτύπου για το πρόβλημα του συστήματος αρχείων.

Κώδικας

Η αφηρημένη κλάση Κόμβος (Node) περιλαμβάνει τις ιδιότητες που είναι κοινές σε όλους τους κόμβους, όπως είναι το όνομα του και η ημερομηνία δημιουργίας και τελευταίας τροποποίησης του.

```
/*
 * Node.java
 */
```

```java
package gr.teithe.it.dp;

import java.util.Date;

/**
 * Abstract class to represent a system node.
 *
 * @author Charalampos Pournaris
 */
public abstract class Node {
   protected String name;
   protected Date created, modified;

   public Node(String name) {
      Date now = new Date();

      this.name = name;
      this.created = now;
      this.modified = now;
   }

   public String getName() {
      return name;
   }

   public abstract long getSize();

   public void print(int depth) {
      for (int i = depth; i > 0; i--) {
         System.out.append("    ");
      }
   }

   public boolean addNode(Node n) {
      System.out.print("[BUG] Trying to add Node in a non-composite object.");
      System.exit(1);

      return false;
   }

   public boolean removeNode(Node n) {
      System.out.print("[BUG] Trying to remove Node from non-composite
object.");
      System.exit(1);

      return false;
   }
}
```

Η προκαθορισμένη υλοποίηση για τις μεθόδους που αφορούν την διαχείριση κόμβων (addNode(), removeNode()) είναι να τερματίζεται η εφαρμογή εμφανίζοντας στην οθόνη ένα μήνυμα λάθους. Η επιλογή αυτή έγινε διότι κάθε κλήση σε μία λειτουργία διαχείρισης κόμβων σε ένα πρωταρχικό κόμβο θεωρείτε σφάλμα (bug) και μπορεί να οδηγήσει σε άλλα πρόβλημα που είναι να δύσκολο να βρεθούν στην πορεία κατά την διάρκεια εκτέλεσης. Οι σύνθετοι κόμβοι παρέχουν

ξεχωριστή υλοποίηση για αυτές τις μεθόδους. Η μέθοδος getSize() επιστρέφει το συνολικό μέγεθος του κόμβου και πρέπει να υλοποιηθεί από όλες τις κλάσεις που κληρονομούν έναν Κόμβο. Η μέθοδος print() εμφανίζει την δομή που έχει ο κόμβος (και οι κόμβοι που περιλαμβάνει σε περίπτωση που είναι σύνθετος) κατά τον χρόνο εκτέλεσης. Η προκαθορισμένη υλοποίηση καλείτε από τις υποκλάσεις και εμφανίζει έναν αριθμό από κενά για την καλύτερη μορφοποίηση της εξόδου.

Η κλάση Αρχείο (File) υλοποιεί έναν πρωταρχικό κόμβο που δεν μπορεί να περιλαμβάνει άλλους κόμβους-παιδιά. Μία επιπλέον ιδιότητα που προσθέτει η κλάση αυτή σε σχέση με την αφηρημένη κλάση που κληρονομεί, είναι το μέγεθος του κόμβου.

```
/*
 * File.java
 */

package gr.teithe.it.dp;

/**
 * File concrete class extending functionality from Node.
 *
 * @author Charalampos Pournaris
 */
public class File extends Node {
   private long filesize;

   public File(String name, long filesize) {
      super(name);

      this.filesize = filesize;
   }

   public long getSize() {
      return filesize;
   }

   public void print(int depth) {
      super.print(depth);
      System.out.println(name + " ( " + filesize + " bytes) \t\t{Created: " +
created + " Last modified: " + modified + "}");
   }

}
```

Η κλάση Κατάλογος (Directory) μπορεί να περιλαμβάνει άλλους κόμβους, αρκεί να κληρονομούν την αφηρημένη κλάση Κόμβος. Μπορεί δηλαδή να περιέχει Αρχεία ή άλλους Καταλόγους που μπορεί επίσης να περιέχουν άλλα αρχεία ή καταλόγους κοκ.

```
/*
 * Directory.java
 */
```

```java
package gr.teithe.it.dp;

import java.util.ArrayList;
import java.util.List;

/**
 * Directory concrete class extending functionality from Node.
 *
 * @author Charalampos Pournaris
 */
public class Directory extends Node {
   private List<Node> nodes;

   public Directory(String name) {
      super(name);

      nodes = new ArrayList<Node>(0);
   }

   public void print(int depth) {
      super.print(depth);
      depth++;
      System.out.println(name + " ( " + getSize() + " bytes) \t\t{Created: " +
created + " Last modified: " + modified + "}");
      for (Node n : nodes) {
         n.print(depth);
      }
   }

   public long getSize() {
      long totalsize = 0;

      for (Node n : nodes) {
         totalsize += n.getSize();
      }

      return totalsize;
   }

   public boolean addNode(Node n) {
      return nodes.add(n);
   }

   public boolean removeNode(Node n) {
      return nodes.remove(n);
   }
}
```

Για την διαχείριση των κόμβων-παιδιών γίνεται χρήσης μίας λίστας (ArrayList συγκεκριμένα) από τις έτοιμες συλλογές (collections) που παρέχει η JAVA. Αντί για λίστα θα μπορούσε να χρησιμοποιηθεί οποιαδήποτε δομή δεδομένων (ουρά, πίνακας κτλ.). Για τον υπολογισμό του μεγέθους ενός κόμβου και της εκτύπωσής του γίνεται σάρωση όλης της λίστας. Μια καλύτερη υλοποίηση θα μπορούσε ενδεχόμενος να αποθηκεύει τις ενδιάμεσες τιμές (**caching**), όπως για παράδειγμα

το συνολικό μέγεθος του κόμβου (εάν πρόκειται για σύνθετο κόμβο) έτσι ώστε να μην χρειάζεται να το υπολογίζει κάθε φορά που καλείτε η μέθοδος. Σε μια τέτοια υλοποίηση βέβαια θα πρέπει να ληφθεί υπόψη ότι σε κάθε αλλαγή των κόμβων-παιδιών η τιμή αυτή θα πρέπει να υπολογιστεί ξανά.

Έστω τώρα ότι θέλουμε να προσθέσουμε ένα νέο πρωταρχικό κόμβο Σύνδεση (Link) στο παραπάνω σύστημα που το μόνο που εξυπηρετεί είναι να δείχνει σε άλλους κόμβους. Δεν χρειάζεται καμία απολύτως αλλαγή στην κλάση του πελάτη (παρουσιάζεται παρακάτω) αλλά ούτε και στο υπόλοιπο σύστημα, παρά μόνο η δημιουργία μιας νέας κλάσης που κληρονομεί έναν Κόμβο.

```
/*
 * Link.java
 */

package gr.teithe.it.dp;

/**
 * Link concrete class extending functionality from Node.
 *
 * @author Charalampos Pournaris
 */
public class Link extends Node {
   Node original;

   public Link(String name, Node n) {
      super(name);

      original = n;
   }

   public void print(int depth) {
      super.print(depth);
      System.out.println(name + " -> " + original.getName() + " \t\t{Created: "
+ created + " Last modified: " + modified + "}");
   }

   public long getSize() {
      return 0;
   }
}
```

Η κλάση πελάτη (Client) διατηρεί αναφορές μόνο προς την αφηρημένη κλάση Κόμβος χωρίς να ενδιαφέρεται για τις λεπτομέρειες της υλοποίησης κάθε αντικειμένου, αρκεί φυσικά τα αντικείμενα αυτά να κληρονομούν από έναν Κόμβο.

```
/*
 * Client.java
 */

package gr.teithe.it.dp;

/**
```

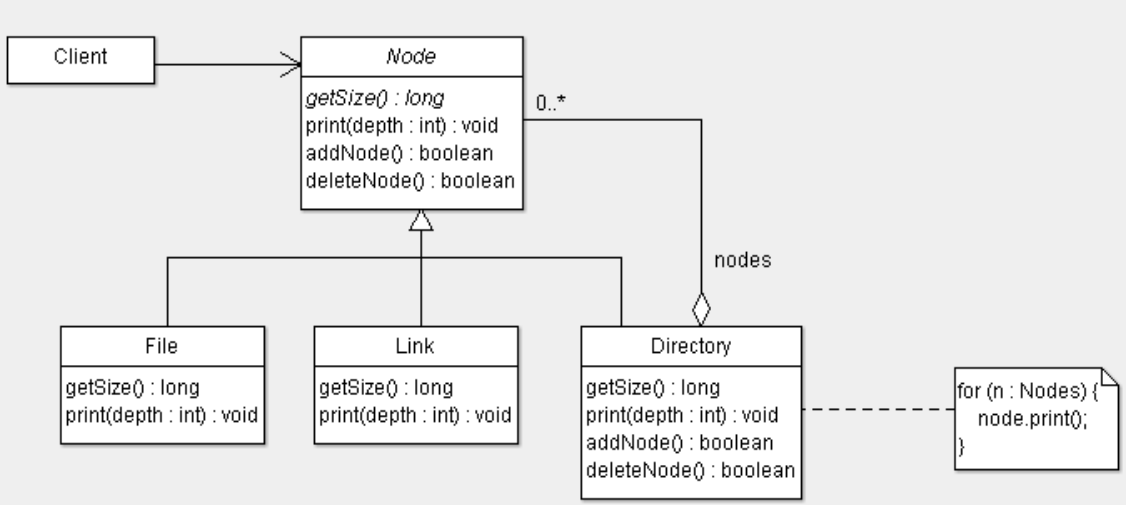
```java
/* Composite Design Pattern Client Usage Demonstration.
*
* @author Charalampos Pournaris
*/
public class Client {
   public static void main(String[] args) {
      Node f1 = new File("Client.java", 221);
      Node f2 = new File("Node.java", 300);
      Node f3 = new File("File.java", 612);
      Node f4 = new File("Directory.java", 147);
      Node d1 = new Directory("/");
      Node d2 = new Directory("src/");
      Node d3 = new Directory("it/");
      Node d4 = new Directory("empty/");
      Node d5 = new Directory("empty_deeper/");
      Node l1 = new Link("node.java_pointer", f2);
      Node l2 = new Link("src_pointer", d2);

      d1.addNode(d2);
      d2.addNode(d3);
      d2.addNode(d4);

      d2.addNode(f1);
      d2.addNode(f2);
      d3.addNode(f3);
      d3.addNode(f4);
      d4.addNode(d5);

      d4.addNode(l1);
      d1.addNode(l2);

      System.out.println("Printing root /:");
      d1.print(0);

      System.out.println("\nPrinting /src/it/:");
      d3.print(0);
   }
}
```

Η έξοδος από την εκτέλεση της εφαρμογής φαίνεται παρακάτω (κάποιες γραμμές και στήλες έχουν παραληφθεί):

```
Printing root /:
/ ( 1280 bytes)
    src/ ( 1280 bytes)
        it/ ( 759 bytes)
            File.java ( 612 bytes)
            Directory.java ( 147 bytes)
        empty/ ( 0 bytes)
            empty_deeper/ ( 0 bytes)
            node.java_pointer -> Node.java
        Client.java ( 221 bytes)
        Node.java ( 300 bytes)
    src_pointer -> src/
```

Δομή

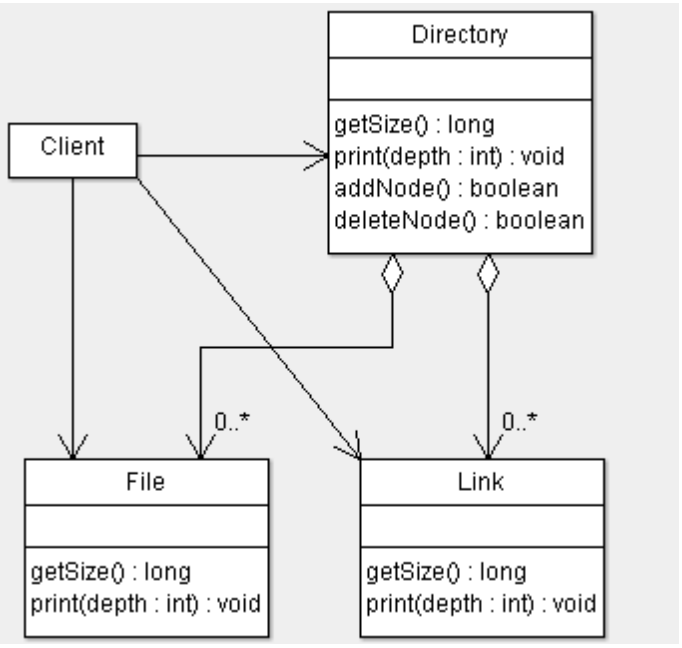
Παρακάτω φαίνεται το διάγραμμα του συστήματος σε UML.



2.1.1 Διάγραμμα κλάσης προτύπου σχεδίασης «Σύνθετο»

Εναλλακτικός τρόπος σχεδίασης χωρίς την χρήση προτύπου

Σε περίπτωση που δεν χρησιμοποιούσαμε το πρότυπο σχεδίασης και δεν υπήρχε η αφηρημένη κλάση Κόμβος (Node) το διάγραμμα θα έπαιρνε την παρακάτω μορφή.



2.1.2 Διάγραμμα για το αντί-πρότυπο σχεδίασης «Σύθετο»

Παρατηρούμε ότι η κλάση του πελάτη πρέπει να διατηρεί αναφορές προς όλα τα άλλα αντικείμενα όπως επίσης και οι σύνθετες κλάσεις προς τις πρωταρχικές. Σε περίπτωση που προσθέσουμε έναν νέο κόμβο θα πρέπει να τροποποιηθεί ολόκληρο το σύστημα.

2.2 Προσαρμογέας (Adapter)

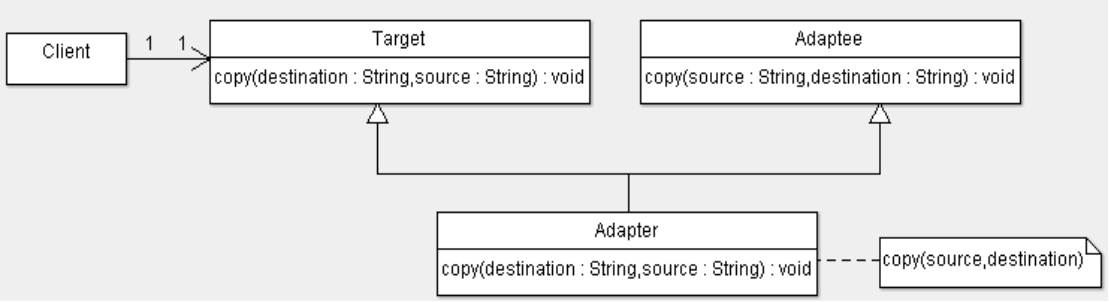
Περιγραφή

Το πρότυπο σχεδίασης «Προσαρμογέας» έχει ως στόχο την *μετατροπή της διεπαφής μίας κλάσης στην διεπαφή που αναμένουν οι κλάσεις πελατών επιτρέποντας με αυτόν τον τρόπο την ανταλλαγή μηνυμάτων μεταξύ κλάσεων που σε διαφορετική περίπτωση θα ήταν αδύνατη λόγο ασυμβατότητας των διεπαφών τους*.

Υπάρχουν περιπτώσεις που ενώ θέλουμε να χρησιμοποιήσουμε μια έτοιμη κλάση στο σύστημα μας, αδυνατούμε λόγω ασύμβατων διεπαφών μεταξύ της κλάσης που θέλουμε να συμπεριλάβουμε και των κλάσεων που υπάρχουν ήδη στο σύστημα. Έστω για παράδειγμα ότι θέλουμε να χρησιμοποιήσουμε μια υπάρχουσα κλάση η οποία υλοποιεί μία μέθοδο copy(source, destination) που λαμβάνει δύο παραμέτρους. Η πρώτη παράμετρος είναι το όνομα ενός αρχείου που αποτελεί την πηγή (source) και η δεύτερη ένα όνομα αρχείου προορισμού στον οποίο θέλουμε να αντιγράψουμε τα περιεχόμενα του πρώτου. Οι κλάσεις πελατών όμως έχουν σχεδιαστεί έτσι ώστε να παρέχουν τις δύο παραμέτρους με αντίθετη σειρά copy (destination, source) και δεν έχουμε την επιλογή να τις τροποποιήσουμε για να μην «σπάσουν» άλλες λειτουργίες ή λόγω υψηλού κόστους (οι κλάσεις των πελατών είναι ήδη εγκατεστημένες σε πολλά συστήματα). Ένα άλλο πρόβλημα είναι η αδυναμία τροποποίησης της νέας κλάσης για να μπορεί να λειτουργήσει με τον τρόπο που αναμένουν οι πελάτες καθώς δεν είναι διαθέσιμος ο πηγαίος κώδικας. Εκτός από την διαφορετική σειρά που δέχεται τα ορίσματα η μέθοδος της νέας κλάσης, θα μπορούσε να διαφέρει και το όνομα της (πχ. copyFile). Σε τέτοιες περιπτώσεις βρίσκει εφαρμογή το πρότυπο σχεδίασης «Προσαρμογέας».

Χρησιμοποιώντας μία «ενδιάμεση» κλάση μπορούμε να προσαρμόσουμε την διεπαφή της νέας κλάσης στην διεπαφή που αναμένουν οι πελάτες, επιτρέποντας έτσι την ανταλλαγή μηνυμάτων μεταξύ τους. Στον πραγματικό κόσμο κάτι ανάλογο είναι η χρήση προσαρμογέων ώστε να μπορούμε να συνδέουμε μεταξύ τους συσκευές που έχουν διαφορετικές διεπαφές.

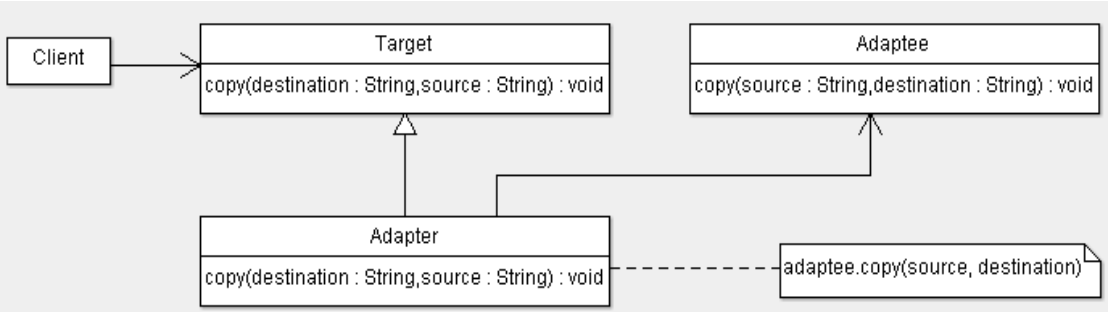
Το πρότυπο μπορεί να χρησιμοποιηθεί με δύο τρόπους. Στην πρώτη περίπτωση γίνεται χρήση ενός προσαρμογέα κλάσης όπως φαίνεται στο παρακάτω διάγραμμα.



2.2.1 Προσαρμογέας κλάσης με πολλαπλή κληρονομικότητα

Προσαρμόζουμε την διεπαφή της κλάσης Adaptee στην διεπαφή του Στόχου (Target). Για να το πετύχουμε αυτό η κλάση του Προσαρμογέα (Adapter) χρησιμοποιεί πολλαπλή κληρονομικότητα (**multiple inheritance**) και συγκεκριμένα κληρονομεί την διεπαφή Target και την υλοποίηση από την Adaptee (αξίζει να σημειωθεί ότι η γλώσσα προγραμματισμού JAVA δεν υποστηρίζει πολλαπλή κληρονομικότητα αλλά επιτρέπει την υλοποίηση πολλαπλών διεπαφών, θα πρέπει δηλαδή στο παραπάνω διάγραμμα η Target να είναι διεπαφή και όχι αφηρημένη κλάση).

Μια δεύτερη λύση αποτελεί η χρήση ενός προσαρμογέα αντικειμένου όπως φαίνεται παρακάτω.



2.2.2 Προσαρμογέας κλάσης με την χρήση αντικειμένου

Σε αυτή την περίπτωση ο Προσαρμογέας (Adapter) κληρονομεί την διεπαφή Target και διατηρεί μία αναφορά στο αντικείμενο της κλάσης Adaptee. Όταν ο προσαρμογέας λάβει ένα μήνυμα για την εκτέλεση κάποιας λειτουργίας, την προωθεί στο αντικείμενο adaptee.

Γενικά το πρότυπο σχεδίασης «Προσαρμογέας» θα πρέπει να χρησιμοποιείται στις εξής περιπτώσεις:

- Όταν θέλουμε να χρησιμοποιήσουμε μία υπάρχουσα κλάση, η διεπαφή της οποίας διαφέρει από αυτήν που θέλουμε
- Όταν θέλουμε να δημιουργήσουμε μια επαναχρησιμοποιήσιμη κλάση που θα μπορεί να συνεργάζεται με άλλες ανεξάρτητες κλάσεις που δεν έχουν απαραίτητα συμβατές διεπαφές
- (*Μόνο για τον προσαρμογέα αντικειμένου*) Όταν θέλουμε να χρησιμοποιήσουμε αρκετές υπάρχουσες υποκλάσεις αλλά δεν είναι πρακτικό να προσαρμόσουμε τις διεπαφές τους κληρονομώντας κάθε μία από αυτές. Ο προσαρμογέας αντικειμένου μπορεί να προσαρμόσει την διεπαφή της γονικής τους κλάσης

Κίνητρο – Παράδειγμα

Θεωρούμε μια εφαρμογή η οποία επιτρέπει στους πελάτες να μεταφέρουν αρχεία (αποστολή/λήψη) αφού πρώτα δημιουργήσουν μια σύνδεση σε έναν απομακρυσμένο διακομιστή. Η μεταφορά των αρχείων επιτρέπεται μόνον εφόσον ο χρήστης έχει δώσει σωστά στοιχεία (ψευδώνυμο και συνθηματικό) και αυτά επιβεβαιωθούν από τον διακομιστή αμέσως μετά την ολοκλήρωση της σύνδεσης. Η μεταφορά των αρχείων μπορεί να πραγματοποιηθεί με την χρήση διάφορων πρωτοκόλλων, όπως για παράδειγμα με HTTP (Hypertext Transfer Protocol) που χρησιμοποιείται κυρίως για την περιήγηση στον παγκόσμιο ιστό (**W**orld **W**ide **W**eb) ή με FTP (File Transfer Protocol) που έχει δημιουργηθεί αποκλειστικά για την μεταφορά αρχείων.

Το σύστημα παρέχει την δυνατότητα ενσωμάτωσης νέων πρωτοκόλλων, αρκεί οι κλάσεις που θα τα υλοποιήσουν να κληρονομούν από μια αφηρημένη κλάση «Μεταφορά Αρχείων» (FileTransfer). Οι λειτουργίες για το πρωτόκολλο HTTP ενσωματώνονται στην κλάση HTTPFileTransfer ενώ για το FTP στην FTPFileTransfer. Οι δύο αυτές κλάσεις όπως είναι φυσικό κληρονομούν από την αφηρημένη κλάση FileTransfer.

Ας υποθέσουμε ότι θέλουμε το σύστημα μας να μπορεί να μεταφέρει αρχεία και με την χρήση ενός άλλου πρωτοκόλλου που φέρει την ονομασία Rsync (**R**emote **S**ynchronization). Το πρωτόκολλο αυτό είναι αρκετά περίπλοκο και δύσκολο στην υλοποίηση του και για αυτόν τον λόγο δεν θέλουμε να το «γράψουμε» εμείς από την αρχή. Αυτό που μπορούμε να κάνουμε είναι να χρησιμοποιήσουμε μια έτοιμη κλάση με την ονομασία RsyncLib. Το πρόβλημα που παρουσιάζεται με αυτήν την προσέγγιση είναι ότι η νέα κλάση που θέλουμε να ενσωματώσουμε στο σύστημα δεν είναι συμβατή με την αφηρημένη κλάση FileTransfer και για τους λόγους που αναφέραμε παραπάνω αδυνατούμε να τροποποιήσουμε τις κλάσεις των πελατών (υψηλό κόστος) ή την ίδια την RsyncLib (έλλειψη πηγαίου κώδικα). Για την αντιμετώπιση αυτού του προβλήματος εφαρμόζουμε το πρότυπο σχεδίασης «Προσαρμογέας» δημιουργώντας μια νέα κλάση RsyncFileTransfer που προσαρμόζει την διαπεφή της κλάσης RsyncLib με αυτήν της FileTransfer.

Παρακάτω παρουσιάζεται ο κώδικας και η περιγραφή για τις προαναφερθείσες κλάσεις.

Κώδικας

Η αφηρημένη κλάση FileTransfer διατηρεί ορισμένα χαρακτηριστικά που είναι κοινά σε όλα τα πρωτόκολλα που την κληρονομούν, όπως είναι το όνομα του χρήστη, ο κωδικός του και η διεύθυνση του διακομιστή που θα πρέπει να συνδεθεί η εφαρμογή. Επίσης η κλάση παρέχει κοινή υλοποίηση για την προσθήκη addFile() και αφαίρεση αρχείων removeFile() από μια μία λίστα (files), πριν αυτά σταλούν (σε περίπτωση που θέλουμε να «ανεβάσουμε» αρχεία) ή προτού γίνει η λήψη τους («κατέβασμα» αρχείων). Οι αφηρημένες μέθοδοι που θα πρέπει να υλοποιήσουν οι υποκλάσεις, αφορούν την πραγματοποίηση της σύνδεσης openConnection() , της

αποσύνδεσης closeConnection(), της αποστολής αρχείων sendFiles() και τέλος της λήψης αρχείων retrieveFiles().

```
/*
 * FileTransfer.java
 */

package gr.teithe.it.dp;

import java.io.File;
import java.net.InetAddress;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author Charalampos Pournaris
 */
public abstract class FileTransfer {
    protected String host, username, password;
    protected int port;
    protected InetAddress address;
    protected List<File> files;

    public FileTransfer(String host, String username, String password)
            throws Exception {
        this.host = host;
        this.username = username;
        this.password = password;
        this.address = InetAddress.getByName(host);
        files = new ArrayList<File>(0);
    }

    public boolean addFile(File f) {
        return files.add(f);
    }

    public boolean removeFile(File f) {
        return files.remove(f);
    }

    public abstract boolean openConnection();
    public abstract void closeConnection();
    public abstract boolean sendFiles();
    public abstract boolean retrieveFiles();
}
```

Η κλάση HTTPFileTransfer κληρονομεί από την FileTransfer και υλοποιεί όλες τις αφηρημένες μεθόδους για το συγκεκριμένο πρωτόκολλο.

```
/*
 * HTTPFileTransfer.java
 */

package gr.teithe.it.dp;
```

```java
import java.io.File;

/**
 *
 * @author Charalampos Pournaris
 */
public class HTTPFileTransfer extends FileTransfer {

   public HTTPFileTransfer(String host, String username, String password)
         throws Exception {
      super(host, username, password);
      port = 80;
   }

   public boolean openConnection() {
      System.out.println("Opening HTTP Connection to " + host + ":" + port);

      return true;
   }
   public void closeConnection() {
      System.out.println("HTTP Connection to " + host + ":" + port + " closed");
   }
   public boolean sendFiles() {
      for (File f : files) {
         System.out.println("Sending file: " + f.getName());
      }

      return true;
   }
   public boolean retrieveFiles() {
      for (File f : files) {
         System.out.println("Retrieving file: " + f.getName());
      }

      return true;
   }
}
```

Στην υλοποίηση των μεθόδων sendFiles() και retrieveFiles() γίνεται σάρωση της λίστας έτσι ώστε η αποστολή/λήψη να γίνει μαζικά για όλα τα αρχεία.
Η κλάση FTPFileTransfer είναι παρόμοια με την HTTPFileTransfer με την διαφορά ότι υλοποιεί τις λειτουργίες της αποστολής/λήψης για διαφορετικό πρωτόκολλο (υποθετικά βέβαια καθώς δεν υπάρχει ο κώδικας που υλοποιεί τις λειτουργίες με τις λεπτομέρειες του κάθε πρωτοκόλλου αλλά γίνεται προσομοίωση).

```java
/*
 * FTPFileTransfer.java
 */

package gr.teithe.it.dp;

import java.io.File;

/**
```

```
 *
 * @author Charalampos Pournaris
 */
public class FTPFileTransfer extends FileTransfer {

    public FTPFileTransfer(String host, String username, String password)
            throws Exception {
        super(host, username, password);
        port = 21;
    }

    public boolean openConnection() {
        System.out.println("Opening FTP Connection to " + host + ":" + port);

        return true;
    }
    public void closeConnection() {
        System.out.println("FTP Connection to " + host + ":" + port + " closed");
    }
    public boolean sendFiles() {
        for (File f : files) {
            System.out.println("Sending file: " + f.getName());
        }

        return true;
    }
    public boolean retrieveFiles() {
        for (File f : files) {
            System.out.println("Retrieving file: " + f.getName());
        }

        return true;
    }

}
```

Η κλάση RsyncLib αποτελεί την νέα κλάση που θέλουμε να προσθέσουμε στο σύστημά μας. Όπως είναι αναμενόμενο, η διεπαφή της κλάσης είναι διαφορετική από αυτήν που θέλουμε, οι λειτουργίες όμως που μας παρέχει είναι παρόμοιες με αυτές που χρειαζόμαστε.

```
/*
 * RsyncLib.java
 */

package gr.teithe.it.dp;

import java.io.File;

/**
 *
 * @author Charalampos Pournaris
 */
public class RsyncLib {
    String connectionString;
    private int defaultPort = 873;
```

```
    public RsyncLib(String connectionString) {
        this.connectionString = connectionString;
    }

    public int connect() {
        System.out.println("Opening Rsync Connection to " +
                connectionString.substring(connectionString.lastIndexOf('@') + 1));

        return 1;
    }

    public void disconnect() {
        System.out.println("Rsync connection to " +
                connectionString.substring(connectionString.lastIndexOf('@') + 1)
                + " closed");
    }

    public boolean uploadOne(File file) {
        System.out.println("Uploading file: " + file.getName());

        return true;
    }

    public boolean downloadOne(File file) {
        System.out.println("Downloading file: " + file.getName());

        return true;
    }

    public int getDefaultPort() {
        return defaultPort;
    }
}
```

Παρατηρούμε ότι τα ονόματα των μεθόδων καθώς και κάποιες άλλες ιδιότητες διαφέρουν από αυτές που αναμένει η κλάση του πελάτη (παρουσιάζεται παρακάτω). Για παράδειγμα η πραγματοποίηση της σύνδεσης με τον διακομιστή γίνεται και με την κλήση της μεθόδου connect(), η αποστολή ενός αρχείου με την uploadOne() κτλ.
Για να μπορέσουμε να χρησιμοποιήσουμε την κλάση RsyncLib, προσαρμόζουμε την διεπαφή της, δημιουργώντας μια νέα κλάση RsyncFileTransfer που κληρονομεί από την FileTransfer και αναλαμβάνει να μεταβιβάσει τις αιτήσεις που γίνονται σε αυτήν στην RsyncLib.

```
/*
 * RsyncFileTransfer.java
 */
package gr.teithe.it.dp;

import java.io.File;

/**
 *
```

```
 * @author Charalampos Pournaris
 */
public class RsyncFileTransfer extends FileTransfer {

    private RsyncLib rlib;

    public RsyncFileTransfer(String host, String username, String password)
            throws Exception {
        super(host, username, password);

        rlib = new RsyncLib(username + ':' + password + '@' + host);
        port = rlib.getDefaultPort();
    }

    public boolean openConnection() {
        int ret = rlib.connect();

        switch (ret) {
            case 1:
                return true;
            default:
                return false;
        }
    }

    public void closeConnection() {
        rlib.disconnect();
    }

    public boolean sendFiles() {
        for (File f : files) {
            if (!rlib.uploadOne(f)) {
                return false;
            }
        }

        return true;
    }

    public boolean retrieveFiles() {
        for (File f : files) {
            if (!rlib.downloadOne(f)) {
                return false;
            }
        }

        return true;
    }
}
```

Η κλάση RsyncFileTransfer διατηρεί μια αναφορά προς την RsyncLib έτσι ώστε κάθε φορά που γίνεται μια κλήση σε κάποια μέθοδο της να καλεί την αντίστοιχη της στο αντικείμενο rlib. Πλέον η νέα κλάση έχει ενσωματωθεί στο σύστημα και οι πελάτες μπορούν να την χρησιμοποιούν με τον ίδιο τρόπο που χρησιμοποιούν και τις ήδη υπάρχουσες.

Παρακάτω παρουσιάζεται μια απλή υλοποίηση της κλάσης ενός πελάτη και ακολουθεί η έξοδος ύστερα από την εκτέλεσή της.

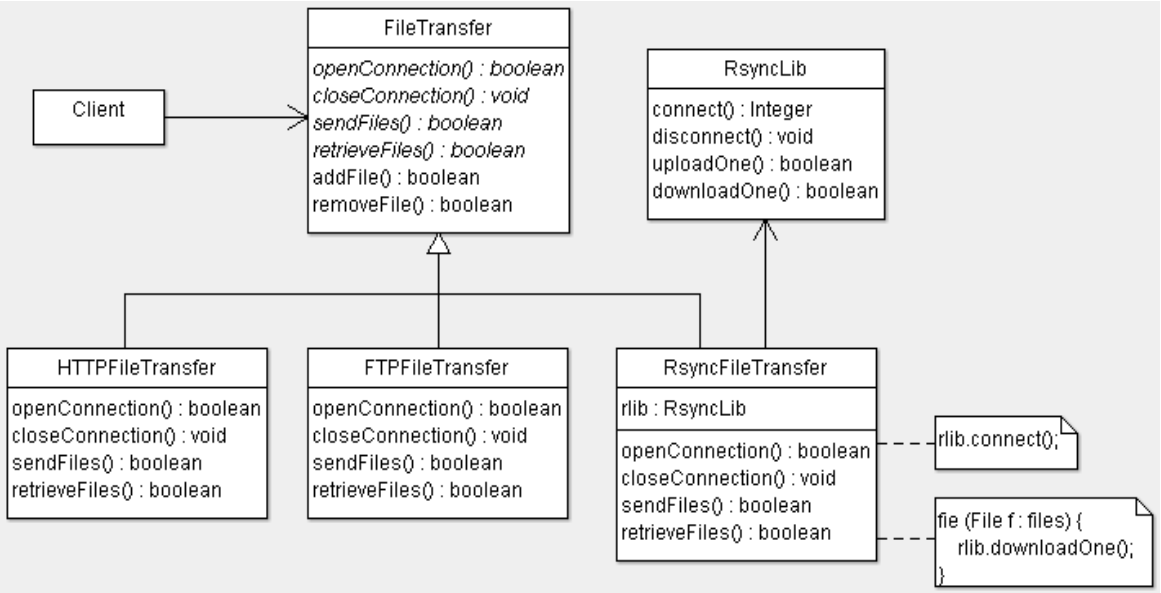
```
/*
 * Client.java
 */

package gr.teithe.it.dp;

import java.io.File;

/**
 * Demonstrate Adapater Design Pattern.
 *
 * @author Charalampos Pournaris
 */
public class Client {
   public static void main(String[] args) throws Exception {
      final String host = "www.example.com";
      final String username = "babis";
      final String password = "godsecret";

      File[] files = new File[] { new File("doc1.txt"), new File("important.lib")
};

      FileTransfer ft = new HTTPFileTransfer(host, username, password);

      ft.openConnection();
      ft.addFile(files[0]);
      ft.addFile(files[1]);
      ft.sendFiles();
      ft.closeConnection();

      System.out.print('\n');

      ft = new RsyncFileTransfer(host, username, password);

      ft.openConnection();
      ft.addFile(new File("dp_are_great.doc"));
      ft.addFile(files[0]);
      ft.addFile(files[1]);
      ft.retrieveFiles();
      ft.closeConnection();
   }
}
```

Έξοδος:

```
Opening HTTP Connection to www.example.com:80
Sending file: doc1.txt
Sending file: important.lib
HTTP Connection to www.example.com:80 closed

Opening Rsync Connection to www.example.com
Downloading file: dp_are_great.doc
```

```
Downloading file: doc1.txt
Downloading file: important.lib
Rsync connection to www.example.com closed
```

Δομή



2.2.3 Διάγραμμα κλάσης προτύπου σχεδίασης «Προσαρμογέας»

Εναλλακτικός τρόπος σχεδίασης χωρίς την χρήση προτύπου

Για το πρότυπο σχεδίασης «Προσαρμογέας» δεν υπάρχει εναλλακτικός τρόπος σχεδίασης καθώς χωρίς την χρήση του οι επιμέρους ασύμβατες κλάσεις δεν μπορούν να ανταλλάξουν μηνύματα.

2.3 Γέφυρα (Bridge)

Περιγραφή

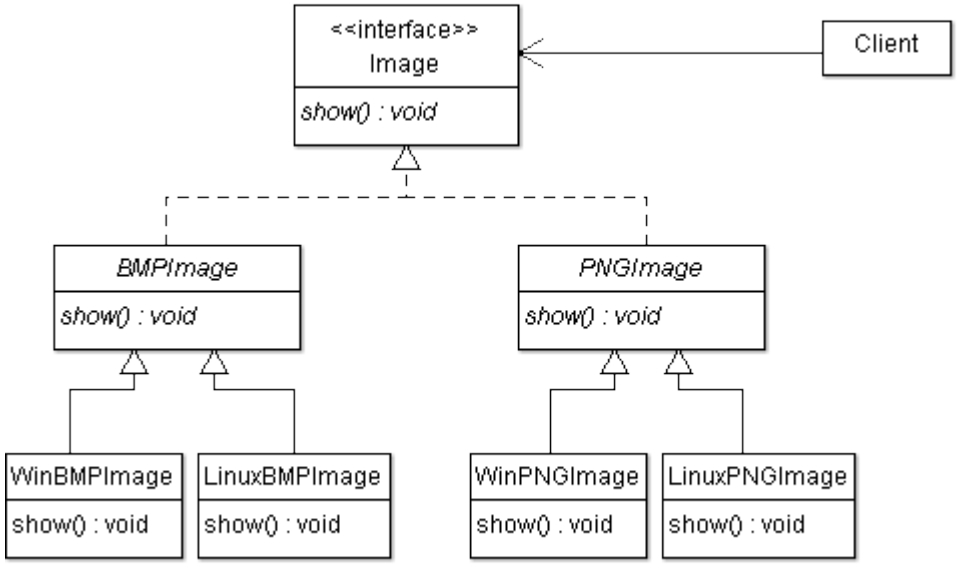
Το πρότυπο σχεδίασης «Γέφυρα» έχει ως στόχο *την αποσύνδεση μιας αφαίρεσης από την υλοποίησή της έτσι ώστε να μπορούν να μεταβάλλονται ανεξάρτητα*.

Όταν μια αφαίρεση μπορεί να έχει διαφορετικές υλοποιήσεις, ο συνηθισμένος τρόπος αντιμετώπισης είναι με την χρήση κληρονομικότητας. Σαν αφαίρεση σε αυτή την περίπτωση θεωρούμε μία διεπαφή, ενώ σαν υλοποίηση τις συγκεκριμένες κλάσεις που υλοποιούν με διαφορετικούς τρόπους τις αφηρημένες λειτουργίες που ορίζονται στην διεπαφή. Με αυτόν τον τρόπο όμως η σχεδίασή μας δεν είναι ευέλικτη καθώς η κληρονομικότητα «δένει» μόνιμα μία συγκεκριμένη

υλοποίηση με την αφαίρεση της, δυσκολεύοντας την τροποποίηση, επέκταση και επαναχρησιμοποίηση της αφαίρεσης και της υλοποίησης με τρόπο ανεξάρτητο.

Ας υποθέσουμε ότι θέλουμε να υλοποιήσουμε μία αφαίρεση «Εικόνα» που θα επιτρέπει στις εφαρμογές μας να απεικονίζουν εικόνες διαφορετικών ειδών στα λειτουργικά συστήματα Windows και Linux, θα υποστηρίζει δηλαδή την μεταφερσιμότητα (**portability**). Χρησιμοποιώντας κληρονομικότητα, ορίζουμε αρχικά μία αφηρημένη κλάση «Εικόνα» (Image) την οποία κληρονομούν συγκεκριμένες εικόνες όπως μια εικόνα τύπου BMP (BMPImage) ή PNG (PNGImage). Για κάθε μια συγκεκριμένη κλάση εικόνας θα πρέπει να παρέχουμε δύο διαφορετικές υλοποιήσεις που αντιστοιχούν στα δύο διαφορετικά λειτουργικά συστήματα που θέλουμε να υποστηρίζει η εφαρμογή μας. Θα πρέπει λοιπόν να προσθέσουμε τέσσερις νέες κλάσεις, δύο που κληρονομούν την BMPImage με ονόματα WinBMPImage και LinuxBMPImage και άλλες δύο που κληρονομούν από την PNGImage με αντίστοιχες ονομασίες WinPNGImage και LinuxPNGImage.

Παρακάτω φαίνεται το διάγραμμα κλάσεων για το συγκεκριμένο σύστημα.



2.3.1 Διάγραμμα κλάσης για το σύστημα απεικόνισης εικόνων

Παρατηρούμε ότι κάθε φορά που θέλουμε να προσθέσουμε μία νέα μορφή εικόνας θα πρέπει να προστεθούν δύο επιπλέον κλάσεις που να την υλοποιούν, μία για κάθε λειτουργικό σύστημα. Ακόμα χειρότερα, εάν θελήσουμε να προσθέσουμε υποστήριξη για ένα νέο λειτουργικό σύστημα, θα πρέπει να προσθέσουμε μια νέα κλάση για κάθε είδος εικόνας προκαλώντας εκρηκτική αύξηση του αριθμού των κλάσεων του συστήματος.

Ένα ακόμα μειονέκτημα αυτής της προσέγγισης είναι ότι οι κλάσεις των πελατών δεσμεύονται σε μια συγκεκριμένη υλοποίηση δυσκολεύοντας έτσι την μεταφερσιμότητα σε άλλα λειτουργικά συστήματα. Οι πελάτες θα πρέπει να

μπορούν να απεικονίσουν εικόνες χωρίς όμως να δεσμευτούν κατά την μεταγλώττιση του προγράμματος σε μία συγκεκριμένη υλοποίηση.

Το πρότυπο «Γέφυρα» μας επιτρέπει να αποσυνδέσουμε την αφαίρεση (Εικόνα) από την υλοποίησή της (WinXXXImage, LinuxXXXImage) έτσι ώστε να μπορούν να μεταβάλλονται ανεξάρτητα. Θα πρέπει δηλαδή κάθε φορά που προσθέτουμε ένα νέο είδος εικόνας να μην επηρεάζεται η υλοποίηση και το αντίστροφο. Για να το πετύχουμε αυτό πρέπει να διαχωρίσουμε την αφαίρεση από την υλοποίηση της σε δύο ξεχωριστές ιεραρχίες κλάσεων και να τις «γεφυρώσουμε».

Το πρότυπο σχεδίασης «Γέφυρα» θα πρέπει να χρησιμοποιείτε στις παρακάτω περιπτώσεις:

- Όταν θέλουμε να αποφύγουμε το μόνιμο «δέσιμο» μεταξύ μίας αφαίρεσης και της υλοποίησής της. Τέτοια περίπτωση μπορεί να προκύψει όταν θέλουμε να επιλέξουμε η να αντικαταστήσουμε μία υλοποίηση κατά τον χρόνο εκτέλεσης

- Όταν η αφαίρεση και η υλοποίηση της μπορούν να επεκταθούν με την χρήση υποκλάσεων. Στην περίπτωση αυτή το πρότυπο επιτρέπει τον συνδυασμό διαφορετικών αφαιρέσεων και υλοποιήσεων και της επέκτασής τους με τρόπο ανεξάρτητο

- Όταν οι αλλαγές στην υλοποίηση μίας αφαίρεσης δεν πρέπει να έχουν κάποιο αντίκτυπο στις κλάσεις των πελατών. Να μην χρειάζεται δηλαδή εκ νέου μεταγλώττιση του πηγαίου κώδικα των πελατών

- (Για την γλώσσα προγραμματισμού C++) Όταν θέλουμε να αποκρύψουμε τελείως την υλοποίηση μίας αφαίρεσης από τους πελάτες

- Όταν οι κλάσεις πολλαπλασιάζονται μαζικά όπως φάνηκε στο παράδειγμα με τις Εικόνες. Στην περίπτωση αυτή θα πρέπει να γίνει διαχωρισμός του αντικειμένου σε δύο μέρη

- Σε περιπτώσεις που θέλουμε να αποκρύψουμε τον διαμοιρασμό (**sharing**) μίας υλοποίησης μεταξύ πολλών αντικειμένων από τους πελάτες

Κίνητρο – Παράδειγμα

Θεωρούμε ένα σύστημα στο οποίο μπορούμε να αναπαραστήσουμε μηνύματα με διαφορετικούς τρόπους, για παράδειγμα μπορούμε να έχουμε απλό κείμενο, κρυπτογραφημένο κείμενο κτλ. Επιπλέον θέλουμε τα μηνύματα αυτά να μπορούμε είτε να τα εμφανίσουμε στην οθόνη ή να τα αποθηκεύσουμε μόνιμα στον αποθηκευτικό χώρο του συστήματος (Πχ. Στον σκληρό δίσκο).

Αρχικά δημιουργούμε μια αφηρημένη κλάση Μήνυμα (Message) που αποτελεί την αφαίρεση σύμφωνα με το πρότυπο σχεδίασης «Γέφυρα» και μπορεί να μεταβάλλεται (ιεραρχία κλάσεων της αφαίρεσης). Μια μεταβολή της είναι η συγκεκριμένη κλάση που αναπαριστά απλό κείμενο (TextMessage) και κληρονομεί από την κλάση Μήνυμα. Μια άλλη συγκεκριμένη κλάση ενός μηνύματος αναπαριστά ένα κρυπτογραφημένο μήνυμα (EncryptedMessage). Για την εμφάνιση και αποθήκευση των μηνυμάτων, δηλαδή σαν υλοποίηση της αφαίρεσης Μύνημα ορίζουμε μια νέα ιεραρχία κλάσεων με βασική κλάση την LoggerImp. Η αφηρημένη αυτή κλάση περιλαμβάνει κάποια χαρακτηριστικά και λειτουργίες που είναι κοινά σε όλες τις υλοποιήσεις (είτε πρόκειται για εμφάνιση μηνυμάτων στην οθόνη ή την αποθήκευσή τους) καθώς και αφηρημένες μεθόδους που πρέπει να υλοποιηθούν από τις υποκλάσεις. Η υποκλάση της LoggerImp που υλοποιεί την λειτουργία της εμφάνισης ενός μηνύματος σε μια κονσόλα (συνήθως στην οθόνη του υπολογιστή) ονομάζεται ConsoleLoggerImp. Μια δεύτερη υλοποίηση με όνομα FileLoggerImp που επίσης κληρονομεί από την LoggerImp, αποθηκεύει τα μηνύματα σε ένα αρχείο. Για την «γεφύρωση» των δύο ιεραρχιών κλάσεων, δηλαδή της αφαίρεσης Μήνυμα (Message) και της υλοποίησης LoggerImp, η πρώτη διατηρεί μια αναφορά στην δεύτερη και καλεί το σύνολο των λειτουργιών της όπως ορίζεται στην διεπαφή της.

Εάν τώρα θελήσουμε να προσθέσουμε μια νέα αφαίρεση ενός μηνύματος όπως είναι ένα συμπιεσμένο μήνυμα (CompressedMessage), το μόνο που χρειάζεται να κάνουμε είναι να κληρονομήσουμε από την αφηρημένη κλάση Μήνυμα και να υλοποιήσουμε τις αφηρημένες μεθόδους που ορίζει, χωρίς να απαιτείται καμία απολύτως αλλαγή στην ιεραρχία LoggerImp που αφορά την υλοποίηση. Αντίστοιχα, για την προσθήκη μίας νέας υλοποίησης που θα μπορεί να αποθηκεύει τα μηνύματα σε μία σχεσιακή βάση δεδομένων (RDMS) με όνομα DatabaseLoggerImp, αρκεί να κληρονομήσουμε από την LoggerImp χωρίς να κάνουμε καμία απολύτως άλλη αλλαγή στην άλλη ιεραρχία.

Κώδικας

Η αφηρημένη κλάση Μήνυμα (Message) περιλαμβάνει λειτουργίες για την επεξεργασία του περιεχομένου του μηνύματος (Αλλαγή, Προσθήκη) και διατηρεί μία αναφορά στην αφηρημένη κλάση LoggerImp και όχι σε κάποια συγκεκριμένη υλοποίηση.

```
/*
 * Message.java
 */

package gr.teithe.it.dp;

/**
 * Message Abstraction hierarchy
 *
 * @author Charalampos Pournaris
 */
public abstract class Message {
```

```
    private LoggerImp imp;
    private String contents;

    public Message(LoggerImp imp) {
        this.imp = imp;
    }

    public void setLogger(LoggerImp imp) {
        if (imp == null) {
            // Default implementation
            imp = new ConsoleLoggerImp("CRITICAL");
        }

        this.imp = imp;
    }

    public LoggerImp getLogger() {
        return imp;
    }

    public String getContents() {
        return contents;
    }

    public void appendContents(String newContent) {
        this.contents += newContent;
    }

    public void setContents(String contents) {
        this.contents = contents;
    }

    public abstract boolean logMessage(String level);
}
```

Η προκαθορισμένη (default) υλοποίηση που χρησιμοποιείται σε περίπτωση που δεν δοθεί κάποια άλλη στον κατασκευαστή κατά την δημιουργία ενός συγκεκριμένου αντικειμένου μηνύματος, είναι η εμφάνιση των μηνυμάτων στην οθόνη του υπολογιστή (αντικείμενο τύπου ConsoleLoggerImp).

Η κλάση TextMessage που υλοποιεί μια συγκεκριμένη μορφή μηνύματος, απλός προωθεί τα μηνύματα στο αντικείμενο της υλοποίησης που «δείχνει» η αναφορά imp την συγκεκριμένη χρονική στιγμή, χωρίς να τα επεξεργαστεί.

```
/*
 * TextMessage.java
 */

package gr.teithe.it.dp;

/**
 * Concrete Message implementation
 *
 * @author Charalampos Pournaris
 */
```

```
public class TextMessage extends Message {
   public TextMessage(LoggerImp imp) {
      super(imp);
   }

   public boolean logMessage(String level) {
      return getLogger().logImpl(getContents(), level);
   }
}
```

Παρόμοια είναι και η κλάση EncryptedMessage με την διαφορά ότι κρυπτογραφεί το μήνυμα προτού το προωθήσει.

```
/*
 * EncryptedMessage.java
 */

package gr.teithe.it.dp;

import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;

/**
 *
 * @author Charalampos Pournaris
 */
public class EncryptedMessage extends Message {
   public EncryptedMessage(LoggerImp imp) {
      super(imp);
   }

   private String encryptMessage() {
      byte input[] = getContents().getBytes();
      byte[] encoded = null;

      try {
         SecretKey key = KeyGenerator.getInstance("DES").generateKey();
         Cipher cipher = Cipher.getInstance("DES");

         cipher.init(Cipher.ENCRYPT_MODE, key);

         encoded = cipher.doFinal(input);
      } catch (Exception e) {
         System.out.println(e.getMessage());
         System.exit(1);
      }

      return new sun.misc.BASE64Encoder().encode(encoded);
   }

   public boolean logMessage(String level) {
      return getLogger().logImpl(encryptMessage(), level);
   }
}
```

Μία τρίτη κλάση που συμπιέζει το μήνυμα πριν αυτό αποσταλεί είναι η CompressedMessage.

```
/*
 * CompressedMessage.java
 */

package gr.teithe.it.dp;

/**
 *
 * @author Charalampos Pournaris
 */
public class CompressedMessage extends Message {
    String compressionType;

    public CompressedMessage(LoggerImp imp, String type) {
        super(imp);

        compressionType = type;
    }

    private String compressMessage() {
        // Compress using the apropriate method
        System.out.println("Compressing String \"" + getContents() + "\" using "
                + compressionType + " method.");

        return compressionType + " compressed message";
    }

    public boolean logMessage(String level) {
        return getLogger().logImpl(compressMessage(), level);
    }
}
```

Η αφηρημένη κλάση LoggerImp αποτελεί την βάση στην ιεραρχία της υλοποίησης και παρέχει τα κοινά χαρακτηριστικά και μεθόδους που κληρονομούν οι υποκλάσεις (συγκεκριμένες υλοποιήσεις). Προτού εμφανιστεί ή αποθηκευτεί ένα μήνυμα, προστίθενται σε αυτό ένα πρόθεμα που περιέχει την ημερομηνία που έγινε η λήψη του καθώς και η κατηγορία στην οποία ανήκει (DEBUG, INFO, WARNING, CRITICAL) με βάση την σημαντικότητα του. Κατά την δημιουργία ενός συγκεκριμένου αντικειμένου της υλοποίησης ορίζουμε ποιες κατηγορίες μηνυμάτων θέλουμε να εμφανίζονται/αποθηκεύονται και ποιες όχι.

```
/*
 * LoggerImp.java
 */

package gr.teithe.it.dp;

import java.util.Date;

/**
```

```
* Logger message implementation hierarchy
* @author Charalampos Pournaris
*/
public abstract class LoggerImp {
   private static final String[] levels = new String[] { "DEBUG", "INFO",
   "WARNING", "CRITICAL" };
   private int log_level;

   public LoggerImp(String level) {
      log_level = getLogLevelIndex(level);
   }

   public int getLogLevel() {
      return log_level;
   }

   public int getLogLevelIndex(String level) {
      int idx = -1;

      for (int i = 0; i < levels.length; i++) {
         if (levels[i].equals(level)) {
            idx = i;
            break;
         }
      }

      return idx;
   }

   public void setLogLevel(int level) {
      log_level = level;
   }

   public String getLogPrefix(String level) {
      return "[" + levels[getLogLevelIndex(level)] + "]";
   }

   public Date getDate() {
      return new Date();
   }

   public abstract boolean logImpl(String contents, String level);
}
```

Η συγκεκριμένη υλοποίηση ConsoleLoggerImp κληρονομεί από την LoggerImp και η λειτουργία που επιτελεί είναι να εμφανίζει τα μηνύματα στην κονσόλα του συστήματος.

```
/*
 * ConsoleLoggerImp.java
 */

package gr.teithe.it.dp;

/**
 *
```

```
 * @author Charalampos Pournaris
 */
public class ConsoleLoggerImp extends LoggerImp {
   public ConsoleLoggerImp(String level) {
      super(level);
   }

   public boolean logImpl(String contents, String level) {
      if (getLogLevelIndex(level) >= getLogLevel()) {
         String prefix = "(" + getDate() + ") " + getLogPrefix(level) + " ";
         System.out.println(prefix + contents);
      }

      return true;
   }
}
```

Μια δεύτερη υλοποίηση με όνομα FileLoggerImp αποθηκεύει τα μηνύματα στο αρχείο που δηλώνουμε κατά την αρχικοποίηση του αντικειμένου.

```
/*
 * FileLoggerImp.java
 */

package gr.teithe.it.dp;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

/**
 *
 * @author Charalampos Pournaris
 */
public class FileLoggerImp extends LoggerImp {
   String logfile;

   public FileLoggerImp(String level, String filename) {
      super(level);

      logfile = filename;
   }

   public boolean logImpl(String contents, String level) {
      if (getLogLevelIndex(level) >= getLogLevel()) {
         String prefix = "(" + getDate() + ") " + getLogPrefix(level) + " ";
         try {
            BufferedWriter out = new BufferedWriter(new FileWriter(logfile));
            out.write(prefix + contents);
            out.close();
         } catch (IOException e) {
            System.out.println(e.getMessage());
            System.exit(1);
         }
         System.out.println("Writing \"" + prefix + contents + "\" to file '"
               + logfile + "'");
```

```
        }

        return true;
    }
}
```

Η κλάση DatabaseLoggerImp αποτελεί την τρίτη υλοποίηση και έχει την δυνατότητα να συνδέεται σε μία βάση δεδομένων και να αποθηκεύει εκεί τα μηνύματα. Τα στοιχεία της σύνδεσης τα λαμβάνει κατά την αρχικοποίηση του αντικειμένου.

```
/*
 * DatabaseLoggerImp.java
 */

package gr.teithe.it.dp;

/**
 *
 * @author Charalampos Pournaris
 */
public class DatabaseLoggerImp extends LoggerImp {
    String host, username, password;
    int port;

    public DatabaseLoggerImp(String level, String host, int port,
            String username, String password) {
        super(level);

        this.host = host;
        this.port = port;
        this.username = username;
        this.password = password;
    }

    private boolean connect() {
        System.out.println("Connecting to database " + host + ":" + port);

        return true;
    }

    private void disconnect() {
        System.out.println("Disconnecting from database " + host + ":" + port);
    }

    public boolean logImpl(String contents, String level) {
        if (getLogLevelIndex(level) >= getLogLevel()) {
            if (connect()) {
                String prefix = "(" + getDate() + ") " + getLogPrefix(level) + " ";
                System.out.println("Writing \"" + prefix + contents +
                        "\" to the database");
                disconnect();
            }
        }

        return true;
    }
```

```
}
```

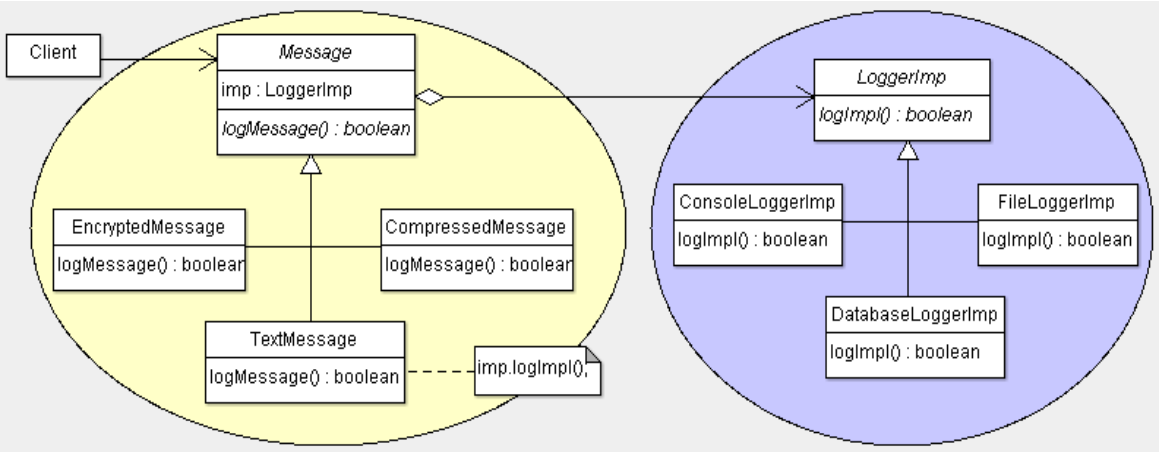
Παρακάτω παρουσιάζεται μια απλή υλοποίηση της κλάσης ενός πελάτη και ακολουθεί η έξοδος μετά την εκτέλεσή της.

```
/*
 * Client.java
 */

package gr.teithe.it.dp;

/**
 * Bridge Design Pattern Client Usage Demonstration.
 *
 * @author Charalampos Pournaris
 */
public class Client {
   public static void main(String[] args) {
      LoggerImp logger = new ConsoleLoggerImp("WARNING");

      Message msg = new TextMessage(logger);

      msg.setContents("This is a test WARNING message");
      msg.logMessage("WARNING");

      msg = new EncryptedMessage(new FileLoggerImp("DEBUG",
"temp.log"));

      msg.setContents("This is a test INFO message");
      msg.logMessage("INFO");

      msg.setLogger(new DatabaseLoggerImp("CRITICAL", "localhost", 3306,
"root", "godsecret"));

      msg.logMessage("CRITICAL");
   }
}
```

Έξοδος:

```
(Fri Feb 26 07:34:12 EET 2010) [WARNING] This is a test WARNING
message
Writing "(Fri Feb 26 07:34:12 EET 2010) [INFO]
KM6wFsPkgnri9hh0euSoBvy4uH6KLP7NIvM4QW8WkCw=" to file 'temp.log'
Connecting to database localhost:3306
Writing "(Fri Feb 26 07:34:12 EET 2010) [CRITICAL]
UW3KMpPfezpkZxqZr0wkUGNbM3xv5gHhgNkODaTHieM=" to the database
Disconnecting from database localhost:3306
```

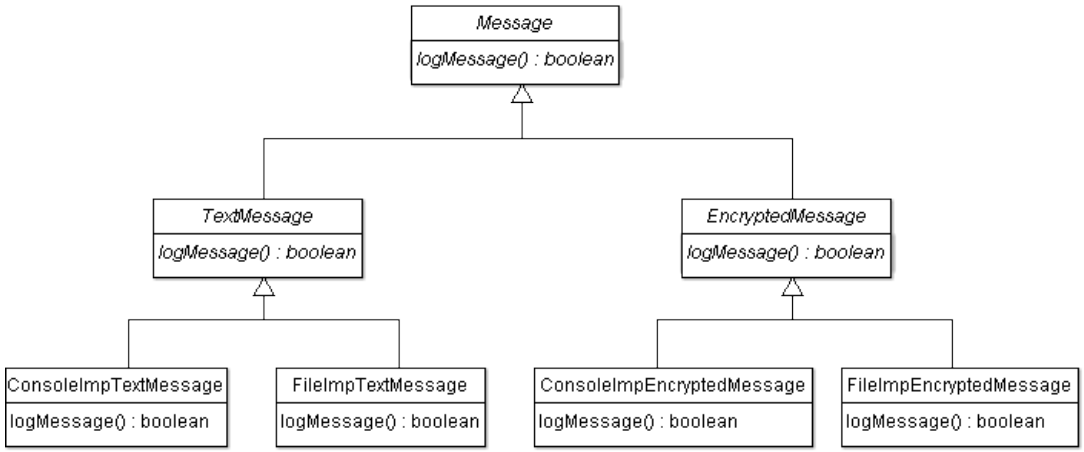
Δομή



2.3.2 Διάγραμμα κλάσης προτύπου σχεδίασης «Γέφυρα»

Εναλλακτικός τρόπος σχεδίασης χωρίς την χρήση προτύπου

Εάν δεν διαχωρίζαμε την αφαίρεση από την υλοποίησή της σε δύο διαφορετικές ιεραρχίες κλάσεων στο το παραπάνω σύστημα, το διάγραμμα κλάσεων θα έπαιρνε την παρακάτω μορφή (Η αφαίρεση CompressedMessage και η υλοποίηση DatabaseLoggerImp δεν έχουν προστεθεί στο διάγραμμα για λόγους συντομίας).



2.3.3 Διάγραμμα για το αντί-πρότυπο σχεδίασης «Γέφυρα»»

Για κάθε νέα αφαίρεση ενός μηνύματος χρειαζόμαστε δύο επιπλέον κλάσεις υλοποίησης, ενώ για την προσθήκη νέας υλοποίησης ο αριθμός των υποκλάσεων που πρέπει να προστεθούν στο σύστημα ισούται με τον αριθμό των αφαιρέσεων. Καταλαβαίνουμε ότι σε ένα μεγαλύτερο σύστημα με δεκάδες αφαιρέσεις και υλοποιήσεις τέτοιου είδους αλλαγές θα προκαλούσαν τεράστιο πολλαπλασιασμό των κλάσεων με καταστροφικά αποτελέσματα για την επεκτασιμότητα, ευελιξία και επαναχρησιμοποίηση του.

2.4 Αφηρημένο Εργοστάσιο (Abstract Factory)

Περιγραφή

Το πρότυπο σχεδίασης «Αφηρημένο Εργοστάσιο» *παρέχει μια διεπαφή για την δημιουργία οικογενειών αντικειμένων που σχετίζονται η έχουν εξάρτηση μεταξύ τους.*

Σύμφωνα με την αρχή «Προγραμματίστε στην διεπαφή, όχι στην υλοποίηση» δεν θα πρέπει τα προγράμματά μας να δεσμεύονται σε κάποια συγκεκριμένη υλοποίηση, όταν φυσικά έχουμε σκοπό να παρέχουμε εναλλακτικές υλοποιήσεις (προβλέποντας και τις μελλοντικές αλλαγές). Όταν οι κλάσεις των πελατών έχουν αναφορές σε συγκεκριμένες κλάσεις που δεν είναι αφηρημένες ή δεν αποτελούν διεπαφές, κάθε φορά που θέλουμε να αντικαταστήσουμε την υλοποίηση με κάποια άλλη ή να τροποποιήσουμε την ήδη υπάρχουσα θα πρέπει να αναζητούμε τα σημεία που γίνεται η χρήση της και να κάνουμε τις αλλαγές, πράγμα που συνεπάγεται φυσικά και την επαναμεταγλώττιση του κώδικα για την συγκεκριμένη κλάση και για όσες εξαρτώνται από αυτήν .

Ας υποθέσουμε ότι έχουμε μια κλάση πελάτη που μπορεί να ζωγραφίζει σχήματα (Painter) και δύο συγκεκριμένες υλοποιήσεις σχημάτων, μια για κύκλους (Circle) και μία για ορθογώνια (Rectangle). Για να μπορέσει η κλάση του πελάτη να εμφανίσει στην οθόνη κάποιο από τα σχήματα, θα πρέπει να διατηρεί δύο αναφορές προς τις δύο συγκεκριμένες κλάσεις των σχημάτων και με την χρήση διακλαδώσεων στον κώδικα (if-then-else) να αποφασίζει κάθε φορά ποιο σχήμα πρέπει να εμφανιστεί. Εάν αργότερα θελήσουμε να δώσουμε την δυνατότητα στο σύστημα να εμφανίζει και τρίγωνα (Triangle) θα πρέπει να προσθέσουμε μία ακόμα αναφορά στην κλάση Paint για την καινούργια υλοποίηση, μία ακόμα διακλάδωση και να μεταγλωττίσουμε ξανά τον κώδικα. Παρομοίως σε περίπτωση που τροποποιήσουμε κάποια από τις υλοποιήσεις θα χρειαστεί επαναμεταγλώττιση.

Σε αυτές τις περιπτώσεις βρίσκει εφαρμογή το πρότυπο «Αφηρημένο Εργοστάσιο» καθώς επιτρέπει στις κλάσεις των πελατών να διατηρούν μία ή περισσότερες αναφορές σε κάποια αφηρημένη κλάση ή διασύνδεση (θα μπορούσε να είναι μια διασύνδεση Σχήμα – **Shape** – στο παραπάνω παράδειγμα) και να δημιουργούν συγκεκριμένα αντικείμενα με ελεγχόμενο τρόπο μέσο της διεπαφής που παρέχει το Αφηρημένο Εργοστάσιο, χωρίς να χρειάζεται να γνωρίζουν και να εξαρτώνται από τις λεπτομέρειες της κάθε υλοποίησης.

Το πρότυπο μπορεί να εφαρμοστεί στις παρακάτω περιπτώσεις:

- Όταν ένα σύστημα δεν πρέπει να εξαρτάται από τον τρόπο που δημιουργούνται, αναπαρίστανται και συντίθενται τα αντικείμενα του
- Ένα σύστημα πρέπει να ρυθμιστεί έτσι ώστε να χρησιμοποιεί μία συγκεκριμένη οικογένεια αντικειμένων, από ένα σύνολο τέτοιων οικογενειών

- Μία οικογένεια σχετιζόμενων αντικειμένων είναι σχεδιασμένα έτσι ώστε να χρησιμοποιούνται μαζί και πρέπει να επιβληθεί αυτός ο περιορισμός

- Όταν θέλουμε να παρέχουμε μια βιβλιοθήκη κλάσεων «αποκαλύπτοντας» μόνο τις διασυνδέσεις τους και όχι την υλοποίηση τους

Κίνητρο – Παράδειγμα

Θεωρούμε μία εφαρμογή που παρέχει στους πελάτες την δυνατότητα σύνδεσης σε σχεσιακές βάσεις δεδομένων, χωρίς να περιορίζεται σε κάποια συγκεκριμένη υλοποίηση. Αυτό επιτυγχάνεται αποφεύγοντας την χρήση αναφορών σε συγκεκριμένες κλάσεις αλλά αποκλειστικά σε αφηρημένες κλάσεις ή διεπαφές μέσα στην κλάση του πελάτη.

Η εφαρμογή παρέχει υλοποίηση για την επικοινωνία με δύο διαφορετικές βάσεις δεδομένων, την MySQL και την PostgreSQL. Η διαχείριση της σύνδεσης με την βάση γίνεται από τους πελάτες μέσο της αφηρημένης κλάσης Σύνδεση (Connection) από την οποία κληρονομούν οι δύο συγκεκριμένες υλοποιήσεις, η μία για την επικοινωνία με βάση MySQL (MysqlConnection) και η άλλη για την PostgreSQL (PosgreSQLConnection). Για την επεξεργασία των αποτελεσμάτων μετά από κάποιο SQL ερώτημα στην βάση, χρησιμοποιείται η αφηρημένη κλάση ResultSet. Οι συγκεκριμένες υλοποιήσεις της είναι MysqlResultSet και PostgreSQLResultSet για τις βάσεις MySQL και PostgreSQL αντίστοιχα.

Όπως είναι φυσικό κάποια στιγμή θα πρέπει να δημιουργηθούν αντικείμενα συγκεκριμένων κλάσεων (είτε της MySQL ή της PostgreSQL) και αυτήν ακριβώς την αρμοδιότητα αναλαμβάνει η διεπαφή «Αφηρημένο εργοστάσιο βάσεων» (DatabaseFactory) και ποιο συγκεκριμένα οι κλάσεις που την υλοποιούν. Τα αντικείμενα που επιστρέφει μπορεί να είναι στιγμιότυπα συγκεκριμένων κλάσεων, όμως ο τύπος επιστροφής που ορίζεται στις μεθόδους της δεν είναι προς κάποια συγκεκριμένη κλάση αλλά προς μια αφηρημένη κλάση. Η DatabaseFactory παρέχει μεθόδους για την δημιουργία αντικειμένων σύνδεσης: makeConnection() με τύπο επιστροφής Connection και αντικειμένων Αποτελεσμάτων: makeResultSet() με τύπο επιστροφής ResultSet. Παρατηρούμε ότι σε καμία από τις δύο μεθόδους δεν γίνεται αναφορά σε κάποια συγκεκριμένη κλάση (πχ. MysqlConnection ή PostgreSQLResultSet) αποκρύπτοντας έτσι από τους πελάτες τις λεπτομέρειες της κάθε υλοποίησης. Την διεπαφή DatabaseFactory υλοποιούν οι δύο κλάσεις-εργοστάσια MysqlDatabaseFactory και PostgreSQLDatabaseFactory που «παράγουν» συγκεκριμένα αντικείμενα για την βάση δεδομένων που αντιπροσωπεύουν (οικογένειες αντικειμένων). Το εργοστάσιο MysqlDatabaseFactory για παράδειγμα, επιστρέφει στιγμιότυπα των κλάσεων MysqlConnection και MysqlResultSet στις κλήσεις των μεθόδων makeConnection() και makeResultSet() αντίστοιχα. Με παρόμοιο τρόπο το εργοστάσιο PostgreSQLDatabaseFactory επιστρέφει κάθε φορά συγκεκριμένα στιγμιότυπα της οικογένειας αντικειμένων της.

Κατά την δημιουργία του αφηρημένου εργοστασίου (DatabaseFactory) ο πελάτης θα πρέπει να δηλώσει συγκεκριμένα ποιο εργοστάσιο επιθυμεί να

χρησιμοποιήσει, επιτρέποντας έτσι την δημιουργία αντικειμένων μόνο από αυτή την «οικογένεια». Επιβάλλοντας αυτόν τον περιορισμό αποτρέπουμε περιπτώσεις όπως για παράδειγμα την χρήση ενός αντικειμένου PostgreSQLConnection για την σύνδεση στην βάση και αντικειμένου MysqlResultSet για την επεξεργασία των αποτελεσμάτων που είναι ασύμβατα.

Παρακάτω παρουσιάζεται ο κώδικας της εφαρμογής σε JAVA (γίνεται προσομοίωση πολλών μεθόδων με την εμφάνιση ενός σχετικού μηνύματος στην οθόνη).

Κώδικας

Η αφηρημένη κλάση Σύνδεση (Connection) περιλαμβάνει ορισμένα κοινά χαρακτηριστικά και λειτουργίες που εμφανίζονται σε όλες τις υλοποιήσεις όπως είναι η διεύθυνση που βρίσκεται η βάση δεδομένων (host, port), πληροφορίες για την αναγνώριση του χρήστη (username, password) και ρυθμίσεις της σύνδεσης (options). Επιπλέον περιλαμβάνει ένα σύνολο από αφηρημένες μεθόδους που θα πρέπει να υλοποιήσουν οι υποκλάσεις όπως η σύνδεση, αποσύνδεση, έλεγχος συνδεσιμότητας και αλλαγή της βάσης που θα εκτελούνται τα SQL ερωτήματα.

```
/*
 * Connection.java
 */

package gr.teithe.it.dp;

/**
 * @author Charalampos Pournaris
 */
public abstract class Connection {
   String host, username, password, database;
   int port, options;

   public Connection(String host, int port, String username, String password) {
      this.host = host;
      this.port = port;
      this.username = username;
      this.password = password;
   }

   public void setHost(String host) {
      this.host = host;
   }

   public void SetOptions(int opt) {
      options = opt;
   }

   public abstract boolean open();
   public abstract void close();
   public abstract int ping();
   public abstract boolean changeDatabase(String db);
}
```

Παρακάτω παρουσιάζεται η συγκεκριμένη υλοποίηση σύνδεσης σε βάση δεδομένων MySQL (MysqlConnection).

```
/*
 * MysqlConnection.java
 */

package gr.teithe.it.dp;

/**
 *
 * @author Charalampos Pournaris
 */
public class MysqlConnection extends Connection {
   public MysqlConnection(String host, int port, String username, String
password) {
      super(host, port, username, password);
   }

   public boolean open() {
      System.out.println("[MySQL] Connecting to database @ " + host
            + ":" + port);

      return true;
   }

   public void close() {
      System.out.println("[MySQL] Disconnecting from database @ " + host + ":"
+ port);
   }

   public int ping() {
      System.out.println("[MySQL] Checking database connection");

      return 0;
   }

   public boolean changeDatabase(String db) {
      database = db;
      System.out.println("[MySQL] Database changed to " + database);

      return true;
   }
}
```

Για την σύνδεση σε βάση PostgreSQL παρέχεται ή συγκεκριμένη κλάση PostgreSQLConnection.

```
/*
 * PostgreSQLConnection.java
 */

package gr.teithe.it.dp;
```

```
/**
 *
 * @author Charalampos Pournaris
 */
public class PostgreSQLConnection extends Connection {
    public PostgreSQLConnection(String host, int port, String username, String
password) {
        super(host, port, username, password);
    }

    public boolean open() {
        System.out.println("[PostgreSQL] Connecting to database @ " + host + ":"
+ port);

        return true;
    }

    public void close() {
        System.out.println("[PostgreSQL] Disconnecting from database @ " + host
+ ":" + port);
    }

    public int ping() {
        System.out.println("[PostgreSQL] Checking database connection");

        return 0;
    }

    public boolean changeDatabase(String db) {
        database = db;
        System.out.println("[PostgreSQL] Database changed to " + database);

        return true;
    }
}
```

Η διαχείριση των αποτελεσμάτων από ένα SQL ερώτημα γίνεται με την χρήση της αφηρημένης κλάσης ResultSet από τον πελάτη. Όλες οι γραμμές αποθηκεύονται σε μία λίστα (ArrayList συγκεκριμένα) και οι τιμές κάθε γραμμής λαμβάνονται από τις μεθόδους getString() και getInt() που είναι αφηρημένες και υλοποιούνται με συγκεκριμένο τρόπο για κάθε βάση από τις υποκλάσεις (παρουσιάζονται παρακάτω). Οι δύο αφηρημένες μέθοδοι first() και next() επιτρέπουν την μετακίνηση μεταξύ των γραμμών.

```
/*
 * ResultSet.java
 */

package gr.teithe.it.dp;

import java.util.ArrayList;
import java.util.List;
```

```java
/**
 *
 * @author Charalampos Pournaris
 */
public abstract class ResultSet {
   List<String> rows = new ArrayList<String>(0);
   int total_rows, current_row;

   public ResultSet() {
      // Sample data
      rows.add("row1");
      rows.add("row2");
      rows.add("row3");
      rows.add("row4");

      total_rows = rows.size();
      current_row = -1;
   }

   public abstract boolean next();
   public abstract void first();
   public abstract String getString();
   public abstract int getInt();
}
```

Η κλάση MysqlResultSet αποτελεί την συγκεκριμένη υλοποίηση για την διαχείριση των αποτελεσμάτων μίας MySQL βάσης.

```java
/*
 * MysqlResultSet.java
 */

package gr.teithe.it.dp;

/**
 *
 * @author Charalampos Pournaris
 */
public class MysqlResultSet extends ResultSet {
   public MysqlResultSet() {
      super();
   }

   public boolean next() {
      if (current_row == total_rows - 1) {
         return false;
      }

      current_row++;
      return true;
   }

   public void first() {
      current_row = 0;
   }
```

```
    public String getString() {
        return rows.get(current_row);
    }

    public int getInt() {
      return Integer.parseInt(rows.get(current_row));
    }
}
```

Για την διαχείριση των αποτελεσμάτων μίας βάσης PostgreSQL παρέχεται η υλοποίηση PostgreSQLResultSet.

```
/*
 * PostgresqlResultSet.java
 */

package gr.teithe.it.dp;

/**
 *
 * @author Charalampos Pournaris
 */
public class PostgreSQLResultSet extends ResultSet {
    public PostgreSQLResultSet() {
        super();
    }

    public boolean next() {
        if (current_row == total_rows - 1) {
            return false;
        }

        current_row++;
        return true;
    }

    public void first() {
        current_row = 0;
    }

    public String getString() {
        return rows.get(current_row);
    }

    public int getInt() {
        return Integer.parseInt(rows.get(current_row));
    }
}
```

Η διεπαφή DatabaseFactory χρησιμοποιείται από τις κλάσεις των πελατών όπως προαναφέρθηκε για την δημιουργία συγκεκριμένων αντικειμένων (υλοποιήσεων) ανάλογα με την οικογένεια αντικειμένων που έχει επιλεγεί. Επιστρέφει ένα αντικείμενο σύνδεσης (Connection) κατά την κλήση της μεθόδου makeConnection()

και ένα αντικείμενο διαχείρισης αποτελεσμάτων (ResultSet) με την κλήση στην makeResultSet().

```
/*
 * DatabaseFactory.java
 */

package gr.teithe.it.dp;

/**
 *
 * @author Charalampos Pournaris
 */
public interface DatabaseFactory {
    public Connection makeConnection(String host, int port, String username,
        String password);
    public ResultSet makeResultSet();
}
```

Μια εναλλακτική υλοποίηση θα ήταν η χρήση μίας μόνο μεθόδου make() που θα λάμβανε με παράμετρο (αλφαριθμητικό, αριθμός, όνομα κλάσης κτλ.) το είδος του αντικειμένου που επιθυμεί ο πελάτης να δημιουργήσει χωρίς να απαιτείται έτσι η τροποποίηση της διασύνδεσης κάθε φορά που προστίθεται νέα λειτουργικότητα. Αυτή η προσέγγιση όμως έχει και τα μειονεκτήματά της καθώς θα μπορούσαν να προκύψουν σφάλματα κατά την εκτέλεση λόγο λανθασμένων παραμέτρων.

Το εργοστάσιο MysqlDatabaseFactory υλοποιεί την διεπαφή DatabaseFactory και επιστρέφει αντικείμενα της οικογένειας MySQL.

```
/*
 * MysqlDatabaseFactory.java
 */

package gr.teithe.it.dp;

/**
 *
 * @author Charalampos Pournaris
 */
public class MysqlDatabaseFactory implements DatabaseFactory {
   public Connection makeConnection(String host, int port, String username,
        String password) {
      return new MysqlConnection(host, port, username, password);
   }

   public ResultSet makeResultSet() {
      return new MysqlResultSet();
   }
}
```

Η δεύτερη υλοποίηση της DatabaseFactory αφορά την οικογένεια αντικειμένων PostgreSQL (PostgreSQLDatabaseFactory).

```
/*
 * PostgresqlDatabaseFactory.java
 */

package gr.teithe.it.dp;

/**
 *
 * @author Charalampos Pournaris
 */
public class PostgreSQLDatabaseFactory implements DatabaseFactory {
    public Connection makeConnection(String host, int port, String username,
            String password) {
        return new PostgreSQLConnection(host, port, username, password);
    }

    public ResultSet makeResultSet() {
        return new PostgreSQLResultSet();
    }
}
```

Παρακάτω παρουσιάζεται μια απλή υλοποίηση της κλάσης ενός πελάτη και ακολουθεί η έξοδος μετά την εκτέλεσή της.

```
/*
 * Client.java
 */

package gr.teithe.it.dp;

/**
 * Abstract Factory Pattern Demonstration
 *
 * @author Charalampos Pournaris
 */
public class Client {
    public static void main(String[] args) {
        Connection con = null;
        ResultSet res = null;
        DatabaseFactory factory = new MysqlDatabaseFactory();

        System.out.println("Using MySQL Factory:\n--------------------");

        con = factory.makeConnection("127.0.0.1", 3306, "mysql_user",
"secretpass");
        res = factory.makeResultSet();

        con.open();
        con.changeDatabase("newdb");
        while (res.next()) {
            System.out.println(res.getString());
```

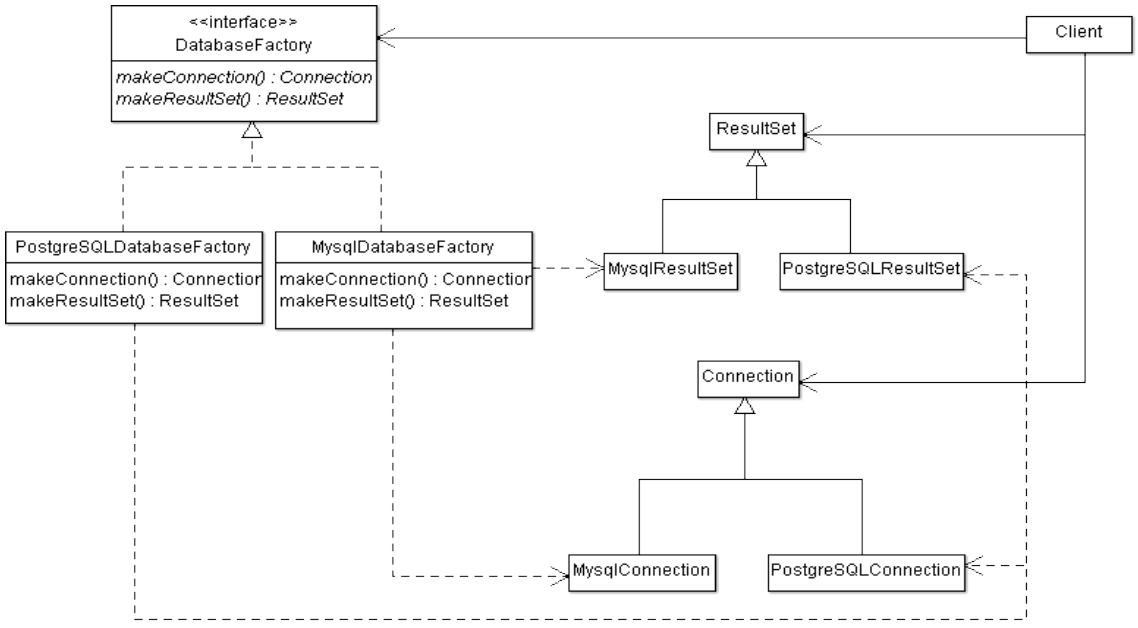
```
        }
        con.close();

        System.out.println("Using PostgreSQL Factory:\n----------------------");

        factory = new PostgreSQLDatabaseFactory();

        con = factory.makeConnection("127.0.0.1", 5432, "postgresql_user",
"secretpass");
        res = factory.makeResultSet();

        con.open();
        con.changeDatabase("testdb");
        while (res.next()) {
           System.out.println(res.getString());
        }
        con.close();
    }
}
```

Έξοδος:

```
Using MySQL Factory:
--------------------
[MySQL] Connecting to database @ 127.0.0.1:3306
[MySQL] Database changed to newdb
row1
row2
row3
row4
[MySQL] Disconnecting from database @ 127.0.0.1:3306
Using PostgreSQL Factory:
-------------------------
[PostgreSQL] Connecting to database @ 127.0.0.1:5432
[PostgreSQL] Database changed to testdb
row1
row2
row3
row4
[PostgreSQL] Disconnecting from database @ 127.0.0.1:5432
```

Παρατηρούμε ότι η μόνη αλλαγή που χρειάζεται στην κλάση του πελάτη (Client) για την σύνδεση σε διαφορετική βάση δεδομένων είναι η αντικατάσταση του εργοστασίου παραγωγής αντικειμένων με μία συγκεκριμένη υλοποίηση.

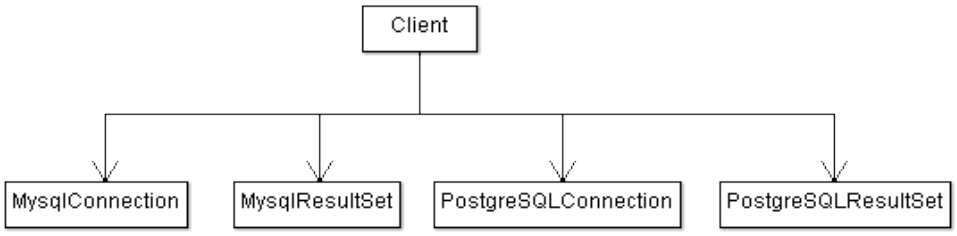
Δομή



2.4.1 Διάγραμμα κλάσης προτύπου σχεδίασης «Αφηρημένο Εργοστάσιο»

Εναλλακτικός τρόπος σχεδίασης χωρίς την χρήση προτύπου

Χωρίς την εφαρμογή του προτύπου σχεδίασης «Αφηρημένο Εργοστάσιο» η κλάση του πελάτη θα έπρεπε να διατηρεί αναφορές σε συγκεκριμένες κλάσεις, όπως φαίνεται στο παρακάτω διάγραμμα.



2.4.2 Διάγραμμα κλάσης για το αντί-πρότυπο σχεδίασης «Αφηρημένο Εργοστάσιο»

Επιπλέον σε διάφορα σημεία του κώδικα θα χρειαζόταν η χρήση διακλαδώσεων για την επιλογή της κατάλληλης κάθε φορά υλοποίησης:

```
if (use_mysql) {
      mconnection = new MysqlConnection(...);
} else if (use_postgresql) {
      pconnection = new PostgreSQLConnection(...); }
```

2.5 Στρατηγική (Strategy)

Περιγραφή

Το πρότυπο σχεδίασης «Στρατηγική» *ορίζει μια «οικογένεια» αλγορίθμων, ενθυλακώνει τον καθένα από αυτούς και επιτρέπει την εναλλαγή μεταξύ τους. Επιπλέον, επιτρέπει στον αλγόριθμο να μεταβάλλεται ανεξάρτητα από τις κλάσεις των πελατών που τον χρησιμοποιούν*.

Πολύ συχνά σχεδιάζοντας ένα σύστημα διαπιστώνουμε ότι κάποιες λειτουργίες του μπορούν να υλοποιηθούν με διαφορετικούς τρόπους και πως θα ήταν χρήσιμο να υπάρχει η δυνατότητα επιλογής της καταλληλότερης υλοποίησης (για την εκάστοτε περίπτωση) από τα προγράμματα των πελατών, σε περίπτωση που η προκαθορισμένη επιλογή δεν είναι επαρκής. Αναγνωρίζουμε δηλαδή μια γενική στρατηγική στο σύστημα μας που είναι κοινή σε πολλές περιπτώσεις αλλά διαφοροποιείται κάθε φορά η υλοποίησή της.

Στην περίπτωση ενός αλγορίθμου ταξινόμησης για παράδειγμα όπου η στρατηγική είναι η διαδικασία της ταξινόμησης ορισμένων στοιχείων, η υλοποίηση του αλγορίθμου μπορεί κάθε φορά να διαφέρει (πχ. ταξινόμηση φυσαλίδας, γρήγορη ταξινόμηση) το αποτέλεσμα όμως σε όλες τις περιπτώσεις είναι το ίδιο. Θεωρώντας μία κλάση Λίστα (List) που περιέχει μη ταξινομημένα στοιχεία και μια μέθοδο sort() για την ταξινόμηση των στοιχείων της, το παραπάνω πρόβλημα θα μπορούσε να λυθεί με την χρήστη κληρονομικότητας έτσι ώστε οι υποκλάσεις της List να υλοποιούν τον αλγόριθμο ταξινόμησης (μέθοδος sort) με διαφορετικό τρόπο. Το πρόβλημα με αυτή την προσέγγιση είναι ότι η Λίστα περιέχει δεδομένα και μεθόδους που δεν σχετίζονται με την διαδικασία της ταξινόμησης και ως εκ τούτου η χρήση της κληρονομικότητας δένει έναν συγκεκριμένο αλγόριθμο με την υλοποίηση της λίστας, δυσκολεύοντας την επέκταση, κατανόηση και συντήρηση της. Ένα άλλο μειονέκτημα αυτής της μεθόδου είναι ότι η υλοποίηση του αλγορίθμου δεν μπορεί να αλλάξει κατά τον χρόνο εκτέλεσης του προγράμματος.

Το πρότυπο σχεδίασης «Στρατηγική» λύνει το πρόβλημα με έναν κομψό και ευέλικτο τρόπο κάνοντας χρήση σύνθεσης αντικειμένων έναντι της κληρονομικότητας. Στην συγκεκριμένη περίπτωση εφαρμόζοντας το πρότυπο θα μπορούσαμε να χρησιμοποιήσουμε μία αφηρημένη κλάση Ταξινόμηση (Sort) την οποία κληρονομούν οι συγκεκριμένες υλοποιήσεις του αλγορίθμου (BubbleSort, QuickSort) . Η λίστα (List) διατηρεί μία αναφορά προς την αφηρημένη κλάση Sort και κάθε φορά που πρέπει να πραγματοποιηθεί η ταξινόμηση καλεί την μέθοδο sort() στο αντικείμενο που «δείχνει» η αναφορά.

Το πρότυπο σχεδίασης «Στρατηγική» θα πρέπει να χρησιμοποιείται στις παρακάτω περιπτώσεις:

- Όταν έχουμε ένα σύνολο από συσχετιζόμενες κλάσεις που διαφέρουν μόνο στην συμπεριφορά τους. Το πρότυπο επιτρέπει την επιλογή μίας συγκεκριμένης συμπεριφοράς για κάποια κλάση

- Όταν απαιτούνται διαφορετικές παραλλαγές ενός αλγορίθμου. Οι παραλλαγές του αλγορίθμου για παράδειγμα μπορεί να αφορούν την ταχύτητα εκτέλεσής του ή του χώρου που απαιτεί. Οι στρατηγικές μπορούν να εφαρμοστούν όταν αυτές οι παραλλαγές υλοποιούνται σε μια ιεραρχία κλάσεων αλγορίθμων

- Όταν ένας αλγόριθμος χρησιμοποιεί δεδομένα που δεν πρέπει να γνωρίζουν οι πελάτες. Το πρότυπο επιτρέπει την απόκρυψη πολύπλοκων δομών δεδομένων που σχετίζονται με κάποιον συγκεκριμένο αλγόριθμο

- Όταν μια κλάση ορίζει διαφορετικές συμπεριφορές και αυτές εμφανίζονται σαν διακλαδώσεις στις μεθόδους της. Αντί των διακλαδώσεων οι διάφορες συμπεριφορές μπορούν να μεταφερθούν σε ξεχωριστές κλάσεις στρατηγικής

Κίνητρο – Παράδειγμα

Θεωρούμαι ένα σύστημα που δίνει την δυνατότητα στους πελάτες να συμπιέζουν αρχεία σε διαφορετικές μορφές (zip, gzip) με την χρήση διαφορετικών αλγορίθμων συμπίεσης. Το σύστημα επιτρέπει στους πελάτες να επιλέξουν τον αλγόριθμο που θεωρούν καταλληλότερο και την δυνατότητα εναλλαγής του κατά τον χρόνο εκτέλεσης.

Για να μπορέσει η εφαρμογή να υποστηρίξει μελλοντικά διαφορετικούς αλγορίθμους, χρησιμοποιείται μια στρατηγική (ιεραρχία κλάσεων) όπου η κάθε κλάση ενθυλακώνει έναν συγκεκριμένο αλγόριθμο συμπίεσης. Στην κορυφή της ιεραρχίας υπάρχει η αφηρημένη κλάση Συμπιεστής (Compressor) που ορίζει τα κοινά χαρακτηριστικά και τις λειτουργίες (διεπαφή) για όλους τους αλγορίθμους. Στο σύστημα υπάρχουν δύο συγκεκριμένες κλάσεις που υλοποιούν την αφηρημένη κλάση Compressor, η ZipCompressor που υλοποιεί τον αλγόριθμο συμπίεσης ZIP και η GzipCompressor που υλοποιεί τον αλγόριθμο GZIP.

Για την διαχείριση των αρχείων (προσθήκη/αφαίρεση από λίστα αρχείων), την επιλογή του αλγορίθμου συμπίεσης και την εκκίνηση της διαδικασίας συμπίεσης στο σύνολο όλων των αρχείων που έχουν προστεθεί στην λίστα μέχρι εκείνη την χρονική στιγμή, το σύστημα παρέχει την κλάση «Διαχειριστής Αρχείων» (ArchiveManager). Κατά την δημιουργία ενός αντικειμένου ArchiveManager ο πελάτης έχει την δυνατότητα να επιλέξει τον αλγόριθμο συμπίεσης που επιθυμεί, δημιουργώντας ένα αντικείμενο τύπου Compressor (είτε ZipCompressor ή GzipCompressor) και περνώντας την αναφορά στον κατασκευαστή της κλάσης ArchiveManager. Εάν δεν έχει κάποια ιδιαίτερη προτίμηση μπορεί να περάσει την τιμή *null* στον κατασκευαστή για να χρησιμοποιηθεί ο προεπιλεγμένος αλγόριθμος.

Παρακάτω παρουσιάζεται ο κώδικας του συστήματος σε JAVA.

Κώδικας

Η κλάση ArchiveManager χρησιμοποιείτε από τις κλάσεις των πελατών και επιτρέπει την προσθήκη/αφαίρεση αρχείων προς συμπίεση. Επιπλέον, διατηρεί μία αναφορά στην αφηρημένη κλάση Compressor καλώντας τις λειτουργίες του αντικειμένου στο οποίο δείχνει κάθε φορά η αναφορά. Η αρχικοποίηση της αναφοράς γίνεται στον κατασκευαστή της κλάσης σε περίπτωση που θέλει να επιλέξει έναν συγκεκριμένο αλγόριθμο ο πελάτης, διαφορετικά γίνεται χρήση του προκαθορισμένου αλγορίθμου (ZIP σε αυτήν την περίπτωση).

```
/*
 * ArchiveManager.java
 */

package gr.teithe.it.dp;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author Charalampos Pournaris
 */
public class ArchiveManager {
    List<File> filelist;
    Compressor comp = null;

    public ArchiveManager(Compressor comp) {
        // Default behavior in case no Compressor is specified
        this.comp = comp == null ? new ZipCompressor() : comp;

        filelist = new ArrayList<File>(0);
    }

    public boolean addFile(String filename) {
        File f = new File(filename);

        if (f.exists()) {
            filelist.add(f);
        } else {
            return false;
        }

        return true;
    }

    public boolean removeFile(String filename) {
        for (File f : filelist) {
            if (f.getName().equals(filename)) {
                filelist.remove(f);
                return true;
            }
        }
```

```
        return false;
    }

    public void setCompressor(Compressor c) {
        comp = c;
    }

    public boolean compressFiles() {
        for (File f : filelist) {
            if (!comp.compressFile(f)) {
                return false;
            }
        }

        clearFileList();

        return true;
    }

    public void clearFileList() {
        filelist.clear();
    }

}
```

Η αφηρημένη κλάση Compressor αποτελεί την κορυφή της ιεραρχίας των κλάσεων για την στρατηγική της συμπίεσης. Περιλαμβάνει κοινά χαρακτηριστικά και λειτουργίες για όλες τις υλοποιήσεις.

```
/*
 * Compressor.java
 */

package gr.teithe.it.dp;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;

/**
 *
 * @author Charalampos Pournaris
 */
public abstract class Compressor {
    protected String name, extension;

    public Compressor() {
        name = "No algorithm specified";
        extension = ".unk";
    }

    public String getName() {
        return name;
```

```
    }

    protected byte[] fileContents(File f) throws IOException {
        InputStream is = new FileInputStream(f);
        long fsize = f.length();
        byte[] bytes = new byte[(int) fsize];

        if (fsize > Integer.MAX_VALUE) {
            return null;
        }

        int offset = 0;
        int numRead = 0;
        while (offset < bytes.length &&
               (numRead = is.read(bytes, offset, bytes.length - offset)) >= 0) {
            offset += numRead;
        }

        if (offset < bytes.length) {
            throw new IOException("Could not completely read file " +
f.getName());
        }

        is.close();
        return bytes;
    }

    public abstract boolean compressFile(File f);
}
```

Η μέθοδος fileContents() διαβάζει τα περιεχόμενα ενός αρχείου στην μνήμη πριν πραγματοποιηθεί η συμπίεση όταν το μέγεθος του αρχείου είναι σχετικά μικρό. Η αφηρημένη μέθοδος compressFile() υλοποιείτε με διαφορετικό τρόπο από κάθε αλγόριθμο και αφορά την συμπίεση ενός αρχείου.

Η κλάση ZipCompressor κληρονομεί από την Compressor και υλοποιεί τον αλγόριθμο συμπίεσης ZIP.

```
/*
 * ZipCompressor.java
 */

package gr.teithe.it.dp;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.zip.ZipEntry;
import java.util.zip.ZipOutputStream;

/**
 * Compression using ZIP.
```

```
 *
 * @author Charalampos Pournaris
 */
public class ZipCompressor extends Compressor {
   public ZipCompressor() {
      name = "Zip Compression";
      extension = ".zip";
   }

   public boolean compressFile(File f) {
      byte[] b;
      String outfile = f.getName() + extension;

      System.out.print("Compressing File \"" + f.getName() + "\" to \""
            + f.getName() + extension + "\" using: " + name);

      try {
         ZipOutputStream out = new ZipOutputStream(new
FileOutputStream(outfile));
         out.putNextEntry(new ZipEntry(f.getName()));

         if (f.length() < (2 * (1024 * 1024))) {
            System.out.println(" (Small File scheme).");
            b = fileContents(f);
            out.write(b, 0, (int) f.length());
         } else {
            System.out.println(" (Big File scheme).");
            InputStream is = new FileInputStream(f);
            b = new byte[4096];

            while (is.read(b) != -1) {
               out.write(b);
            }
         }

         out.close();
      } catch (IOException e) {
         System.out.println("Error occured: " + e.getMessage());
         return false;
      }

      return true;
   }
}
```

Εάν το μέγεθος του αρχείου προς συμπίεση είναι μικρότερο από 2 Megabytes (2 * 1024 * 1024 bytes) γίνεται πρώτα ανάγνωση όλου του αρχείου στην μνήμη πριν την συμπίεση διαφορετικά η ανάγνωση γίνεται σταδιακά σε μικρότερα τμήματα.

Η κλάση GzipCompressor υλοποιεί τον αλγόριθμο συμπίεσης GZIP.

```
/*
 * GzipCompressor.java
 */
```

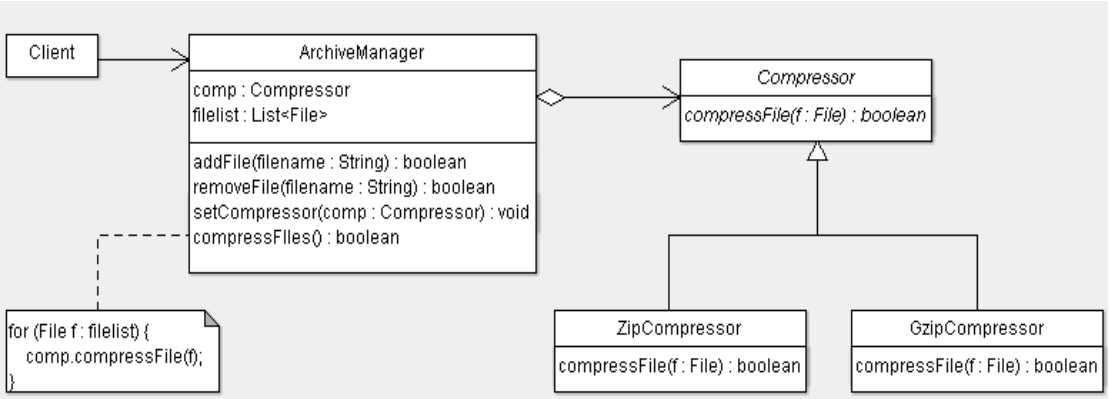
```
package gr.teithe.it.dp;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.zip.GZIPOutputStream;

/**
 *  Compression using GZIP.
 *
 * @author Charalampos Pournaris
 */
public class GzipCompressor extends Compressor {
   public GzipCompressor() {
      name = "Gzip Compression";
      extension = ".gzip";
   }

   public boolean compressFile(File f) {
      byte[] b;
      String outfile = f.getName() + extension;

      System.out.print("Compressing File \"" + f.getName() + "\" to \""
            + f.getName() + extension + "\" using: " + name);

      try {
         GZIPOutputStream out = new GZIPOutputStream(new
FileOutputStream(outfile));

         if (f.length() < (2 * (1024 * 1024))) {
            System.out.println(" (Small File scheme).");
            b = fileContents(f);
            out.write(b, 0, (int) f.length());
         } else {
            System.out.println(" (Big File scheme).");
            InputStream is = new FileInputStream(f);
            b = new byte[4096];

            while (is.read(b) != -1) {
               out.write(b);
            }
         }

         out.close();
      } catch (IOException e) {
         System.out.println("Error occured: " + e.getMessage());
         return false;
      }

      return true;
   }
}
```

Παρακάτω παρουσιάζεται η κλάση ενός πελάτη.

```java
/*
 * Client.java
 */

package gr.teithe.it.dp;

/**
 * Strategy Pattern Demonstration
 *
 * @author Charalampos Pournaris
 */
public class Client {
   public static void main(String[] args) {
      Compressor algo = new GzipCompressor();
      ArchiveManager fmanager = new ArchiveManager(algo);

      fmanager.addFile("C:\\Design_Patterns_CP032336.docx");
      fmanager.addFile("D:\\ISO\\Ubuntu\\ubuntu-9.10-desktop-i386.iso");

      fmanager.compressFiles();

      fmanager.setCompressor(new ZipCompressor());

      fmanager.addFile("C:\\Design_Patterns_CP032336.docx");
      fmanager.addFile("D:\\ISO\\Ubuntu\\ubuntu-9.10-desktop-i386.iso");

      fmanager.compressFiles();
   }
}
```

Κατά την δημιουργία του αντικειμένου ArchiveManager επιλέγεται ο αλγόριθμος GZIP για την συμπίεση ενώ αργότερα γίνεται εναλλαγή με τον αλγόριθμο ZIP (μέθοδος setCompressor). Η έξοδος από την εκτέλεση του προγράμματος είναι η δημιουργία τεσσάρων συμπιεσμένων αρχείων, δύο σε μορφή ZIP και δύο σε μορφή GZIP.

Δομή

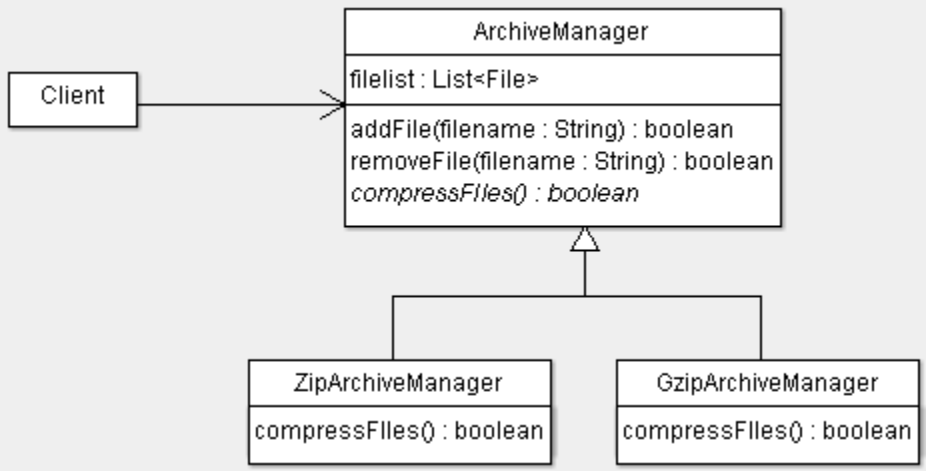


2.5.1 Διάγραμμα κλάσης για το πρότυπο σχεδίασης «Στρατηγική»

Εναλλακτικός τρόπος σχεδίασης χωρίς την χρήση προτύπου

Υπάρχουν δύο εναλλακτικοί τρόποι σχεδίασης για το παραπάνω σύστημα:

- **Με την χρήση κληρονομικότητας**



2.5.2 Διάγραμμα κλάσης για το αντί-πρότυπο σχεδίασης «Στρατηγική»

Το μειονέκτημα αυτής της προσέγγισης είναι ότι η υλοποίηση του αλγορίθμου «δένεται» με αυτήν του διαχειριστή αρχείων (ArchiveManager) και τυχόν αλλαγές στην γονική κλάση μπορεί να έχουν ανεπιθύμητες επιπτώσεις στις υποκλάσεις της. Η συντήρηση γίνεται δυσκολότερη και δεν υπάρχει η δυνατότητα εναλλαγής των αλγορίθμων κατά τον χρόνο εκτέλεσης.

- **Με την χρήση δομών ελέγχου (if-then-else)**

Σε αυτήν την περίπτωση η κλάση ArchiveManager περιλαμβάνει δύο επιπλέον λειτουργίες zipCompressFiles() και gzipCompressFiles(). Με την χρήση διακλαδώσεων επιλέγει κάθε φορά τον αλγόριθμο που πρέπει να χρησιμοποιηθεί.

```
switch (selectedCompressor) {
    case 1:
        zipCompressFiles();
        break;
    case 2:
        gzipCompressFiles();
        break;
    default:
        // Choose default behavior..
}
```

Κάθε φορά που χρειάζεται η προσθήκη ενός νέου αλγορίθμου συμπίεσης απαιτείται τροποποίηση της δομής ελέγχου δυσκολεύοντας την συντήρηση, επέκταση και κατανόηση του συστήματος.

2.6 Μέθοδος Υπόδειγμα (Template Method)

Περιγραφή

Το πρότυπο σχεδίασης «Μέθοδος υπόδειγμα» *ορίζει τον «σκελετό» ενός αλγορίθμου σε μία λειτουργία, μεταβιβάζοντας ορισμένα βήματα στις υποκλάσεις. Η μέθοδος υπόδειγμα επιτρέπει στις υποκλάσεις να επαναπροσδιορίσουν κάποια βήματα ενός αλγορίθμου χωρίς να αλλάξουν την δομή του*.

Το πρότυπο μέθοδος υπόδειγμα χρησιμοποιείται πάρα πολύ συχνά από τους σχεδιαστές αντικειμενοστραφών συστημάτων ακόμα και αν σε ορισμένες περιπτώσεις αυτό δεν γίνεται αντιληπτό από τους ίδιους. Στόχος του προτύπου είναι η επίλυση ενός παρόμοιου προβλήματος με αυτό που επιλύει το πρότυπο «Στρατηγική», τον διαχωρισμό δηλαδή ενός αλγορίθμου από μια συγκεκριμένη υλοποίηση. Η διαφορά τους έγκειται στην χρήση της κληρονομικότητας στο πρότυπο «μέθοδος υπόδειγμα» σε σχέση με την διαβίβαση μηνυμάτων (**delegation**) που πραγματοποιείται στο πρότυπο «στρατηγική».

Για παράδειγμα η διαδικασία της ταξινόμησης των στοιχείων μίας λίστας μπορεί να διαιρεθεί σε δύο λειτουργίες:

1. Σύγκριση δύο διαδοχικών στοιχείων της λίστας (compare)

2. Εναλλαγή τον στοιχείων μεταξύ τους σύμφωνα με τον αλγόριθμο ταξινόμησης (swap)

Θεωρούμε μια αφηρημένη κλάση Sorter που υλοποιεί έναν αλγόριθμο ταξινόμησης και περιλαμβάνει τις αφηρημένες λειτουργίες compare() και swap(). Η κλήση των λειτουργιών αυτών γίνεται από την μέθοδο sort() η οποία αφού αρχικά καλέσει την compare() για να συγκρίνει δύο στοιχεία της λίστας και ανάλογα με το αποτέλεσμα της σύγκρισης, καλεί την swap() για την εναλλαγή των δύο στοιχείων. Η μέθοδος sort() δηλαδή καθορίζει την σειρά με την οποία θα εκτελεστούν οι υπόλοιπες λειτουργίες ορίζοντας έτσι το περίγραμμα του αλγορίθμου και ονομάζεται **μέθοδος υπόδειγμα**. Συγκεκριμένες υποκλάσεις της βασικής κλάσης Sorter υλοποιούν τις μεθόδους (compare, swap) για διαφορετικούς τύπους δεδομένων που μπορεί να είναι είτε βασικοί τύποι (int, float, double) είτε πολυπλοκότεροι (Currency, Date, BigInt). Με την σειρά τους αυτοί οι τύποι δεδομένων μπορεί να είναι αποθηκευμένοι σε διαφορετικές δομές δεδομένων (Array, List, Hash Table). Κάθε κλάση λοιπόν που κληρονομεί την Sorter υλοποιεί τις λειτουργίες για τις διάφορες περιπτώσεις που μπορεί να προκύψουν.

Η μέθοδος υπόδειγμα υλοποιείται πάντα στην βασική κλάση (για τον λόγο αυτό δεν μπορεί να δηλωθεί σαν «καθαρά» διεπαφή αλλά σαν αφηρημένη κλάση στις γλώσσες προγραμματισμού που υπάρχει αυτός ο διαχωρισμός, όπως για παράδειγμα στην JAVA) και μπορεί να καλεί μεθόδους που εν γένει είναι χρήσιμες σε όλες τις υποκλάσεις και υλοποιούνται στην *ίδια* την αφηρημένη κλάση, άλλες *αφηρημένες μεθόδους* που η συγκεκριμένη υλοποίηση τους πραγματοποιείται από τις υποκλάσεις (όπως αναφέρθηκε στο παραπάνω παράδειγμα), καθώς και

μεθόδους «άγκιστρα» (**hook operations**) που παρέχουν μια προκαθορισμένη σειρά ενεργειών και μπορούν να τις υπερβούν οι υποκλάσεις (προαιρετικά).

Το πρότυπο «Μέθοδος Υπόδειγμα» θα πρέπει να χρησιμοποιείται στις εξής περιπτώσεις:

- Για την υλοποίηση του αμετάβλητου μέρους ενός αλγορίθμου μόνο μία φορά στην βασική κλάση, αφήνοντας τις υποκλάσεις να υλοποιήσουν την συμπεριφορά που μπορεί να μεταβάλλεται

- Όταν εντοπίζεται κοινή συμπεριφορά στις υποκλάσεις και πρέπει αυτή η συμπεριφορά να υλοποιηθεί σε μία κοινή κλάση για την αποφυγή της επανάληψης του ίδιου κώδικα

- Για τον έλεγχο της επεκτασιμότητας που προσδίδουν οι υποκλάσεις. Μπορούμε να ορίσουμε την μέθοδο υπόδειγμα έτσι ώστε να καλεί μεθόδους «άγκιστρα» σε συγκεκριμένα σημεία επιτρέποντας την επέκταση της λειτουργικότητας μόνο σε αυτά τα σημεία

Κίνητρο – Παράδειγμα

Θεωρούμε ένα σύστημα συλλογής πληροφοριών (ειδήσεων) που έχει την δυνατότητα να συνδέεται σε διαφορετικές τοποθεσίες με την χρήση του πρωτοκόλλου HTTP και αφού γίνει ταυτοποίηση του χρήστη στον απομακρυσμένο διακομιστή να συλλέγει τα τελευταία νέα και να τα αποθηκεύει τοπικά σε μία σχεσιακή βάση δεδομένων ή σε ένα αρχείο. Η συλλογή των ειδήσεων γίνεται από δύο διαφορετικές πηγές, από RSS Feeds και από ένα Portal.

Για την επίτευξη αυτής της λειτουργικότητας θα πρέπει να γίνουν διαδοχικά οι παρακάτω ενέργειες:

1. Σύνδεση στον απομακρυσμένο διακομιστή
2. Εάν η σύνδεση είναι επιτυχής, ταυτοποίηση του χρήστη
3. Επεξεργασία του κειμένου για την απομόνωση των ειδήσεων από τα στοιχεία μορφοποίησης
4. Αποθήκευση των ειδήσεων σε μία βάση δεδομένων
5. Αποσύνδεση από τον διακομιστή

Τα βήματα αυτά αποτελούν τον αλγόριθμο για την πραγματοποίηση της επιθυμητής λειτουργικότητας και θα πρέπει να γίνονται με αυτήν την συγκεκριμένη σειρά. Σύμφωνα με το πρότυπο σχεδίασης μπορούμε να ενσωματώσουμε αυτές τις λειτουργίες σε μία μέθοδο υπόδειγμα. Ορίζουμε μια αφηρημένη κλάση «συλλογής ειδήσεων» (NewsCollector) στην οποία ορίζουμε την μέθοδο υπόδειγμα με την ονομασία process(). Αυτή η μέθοδος καλεί όλες τις υπόλοιπες λειτουργίες που είναι απαραίτητες για την εκτέλεση του αλγορίθμου. Ορισμένες από αυτές τις λειτουργίες υλοποιούνται στην αφηρημένη κλάση καθώς είναι χρήσιμες και στις υποκλάσεις (connect, disconnect) ενώ άλλες δηλώνονται ως αφηρημένες και υλοποιούνται με διαφορετικό τρόπο από τις υποκλάσεις. Επειδή η ταυτοποίηση, η

επεξεργασία και η αποθήκευση των ειδήσεων στις δύο τοποθεσίες (Portal, RSS Feeds) γίνεται με διαφορετικό τρόπο, η υλοποίηση των αντίστοιχων λειτουργιών (authenticate, parse, save) γίνεται από τις υποκλάσεις PortalNewsCollector και RSSNewsCollector αντίστοιχα. Στην περίπτωση του Portal, η ταυτοποίηση γίνεται με την μέθοδο HTTP Authentication και η επεξεργασία των ειδήσεων γίνεται σε κείμενο HTML, ενώ για τα RSS Feeds ο χρήστης ταυτοποιείται με την αποστολή των στοιχείων σε μία HTML φόρμα, ενώ το κείμενο προς επεξεργασία είναι σε μορφή XML.

Για την διαδικασία της ταυτοποίησης παρέχεται μια ξεχωριστή ιεραρχία κλάσεων με βασική αφηρημένη κλάση την Authentication και τις συγκεκριμένες υλοποιήσεις HTTPAuthentication και FormAuthentication που κληρονομούν από αυτήν.

Κώδικας

Η αφηρημένη κλάση NewsCollector ορίζει όλες τις μεθόδους που αντιστοιχούν στα απαραίτητα βήματα για την εκτέλεση του αλγορίθμου χωρίς όμως να παρέχει υλοποίηση για κάθε μία από αυτές. Συγκριμένα, παρέχει υλοποίηση για τις μεθόδους που πραγματοποιούν την σύνδεση/αποσύνδεση (connect/disconnect) με έναν απομακρυσμένο διακομιστή και είναι κοινές σε όλες τις υποκλάσεις, ενώ αυτές που πρέπει να υλοποιηθούν αποκλειστικά από τις υποκλάσεις δηλώνονται ως αφηρημένες (authenticate, parse, save). Η μέθοδος υπόδειγμα process() καλεί όλες αυτές τις μεθόδους με μία προκαθορισμένη σειρά που δεν αλλάζει και αποτελεί το σταθερό μέρος του αλγορίθμου. Επιπλέον, διατηρεί μια αναφορά στην αφηρημένη κλάση Authentication έτσι ώστε να διαβιβάζει τα μηνύματα που αφορούν την ταυτοποίηση χρήστη στο αντίστοιχο αντικείμενο ταυτοποίησης.

```
/*
 * NewsCollector.java
 */

package gr.teithe.it.dp;

/**
 *
 * @author Charalampos Pournaris
 */
public abstract class NewsCollector {
    private String host;
    private int port;
    private Authentication auth;

    public NewsCollector(String host, int port) {
        this.host = host;
        this.port = port;
    }

    /**
    * process represents the template method
    */
```

```
    public void process(String username, String password) {
        if (!connect()) {
            System.out.println("Unable to connect to the remote server");
        }

        if (authenticate(username, password)) {
            // Authentication is OK
            parse();
            save();
        } else {
            System.out.println("Authentication failed");
        }

        disconnect();
    }

    protected boolean connect() {
        System.out.println("Connected to the remote host " + host + ":" + port);

        return true;
    }

    protected void disconnect() {
        System.out.println("Disconnected from the remote host " + host + ":" +
port);
    }

    protected void setAuthentication(Authentication auth) {
        this.auth = auth;
    }

    protected Authentication getAuthentication() {
        return auth;
    }

    protected abstract boolean authenticate(String username, String
password);
    protected abstract void parse();
    protected abstract boolean save();
}
```

Η κλάση PortalNewsCollector κληρονομεί από την NewsCollector και υλοποιεί τις μεθόδους για την ταυτοποίηση (authenticate), επεξεργασία (parse) και αποθήκευση (save) των ειδήσεων από ένα Portal. Για την ταυτοποίηση δημιουργεί ένα νέο αντικείμενο τύπου HTTPAuthentication το οποίο εκχωρείτε στην μεταβλητή auth (που κληρονομείτε από την γονική κλάση και είναι τύπου Authentication) και στο οποίο μεταβιβάζονται τα μηνύματα που αφορούν την ταυτοποίηση. Τα αποτελέσματα της επεξεργασίας αποθηκεύονται σε μία βάση δεδομένων.

```
/*
 * PortalNewsCollector.java
 */

package gr.teithe.it.dp;
```

```
/**
 *
 * @author Charalampos Pournaris
 */
public class PortalNewsCollector extends NewsCollector {
   public PortalNewsCollector(String host, int port) {
      super(host, port);
   }

   public boolean authenticate(String username, String password) {
      // Authenticate using simple HTTP authentication
      setAuthentication(new HTTPAuthentication(username, password));
      getAuthentication().authenticate();

      return true;
   }

   protected void parse() {
      // Initialize an HTML Parser
      System.out.println("Using HTML parser to grab news");
   }

   protected boolean save() {
      /* Initialize a database connection (possibly using a Factory Method)
       and insert news into a table. */
      System.out.println("Inserting news data into the database tables");

      return true;
   }
}
```

Η κλάση RSSNewsCollector είναι παρόμοια με την PortalNewsCollector αλλά υλοποιεί με διαφορετικό τρόπο τις αφηρημένες λειτουργίες που κληρονομεί από την κλάση NewsCollector. Για την ταυτοποίηση χρησιμοποιείτε ένα αντικείμενο τύπου FormAuthentication ενώ η αποθήκευση των ειδήσεων γίνεται σε ένα αρχείο του συστήματος αντί για βάση δεδομένων.

Η αφηρημένη κλάση Authentication περιλαμβάνει ορισμένα χαρακτηριστικά που είναι κοινά σε όλες τις υποκλάσεις όπως είναι το όνομα χρήστη (username) και ο κωδικός χρήστη (password). Επιπλέον ορίζει την αφηρημένη μέθοδο authenticate().

```
/*
 * Authentication.java
 */

package gr.teithe.it.dp;

/**
 *
 * @author Charalampos Pournaris
 */
public abstract class Authentication {
   String username, password;
```

```
    public Authentication(String username, String password) {
        this.username = username;
        this.password = password;
    }

    public abstract boolean authenticate();
}
```

Η κλάση HTTPAuthentication κληρονομεί από την Authentication υλοποιώντας έναν συγκεκριμένο μηχανισμό ταυτοποίησης.

```
/*
 * HTTPAuthentication.java
 */

package gr.teithe.it.dp;

/**
 *
 * @author Charalampos Pournaris
 */
public class HTTPAuthentication extends Authentication {
    public HTTPAuthentication(String username, String password) {
        super(username, password);
    }

     public boolean authenticate() {
         // Do the actual http authentication..
         System.out.println("Authenticated using HTTP Authentication");

         return true;
     }
}
```

Παρόμοια είναι και η κλάση FormAuthentication.

```
/*
 * FormAuthentication.java
 */

package gr.teithe.it.dp;

/**
 *
 * @author Charalampos Pournaris
 */
public class FormAuthentication extends Authentication {
    public FormAuthentication(String username, String password) {
        super(username, password);
    }

    public boolean authenticate() {
        // Do the actual form authentication..
        System.out.println("Authenticated using POST on HTML FORM");
```

```
        return true;
    }
}
```

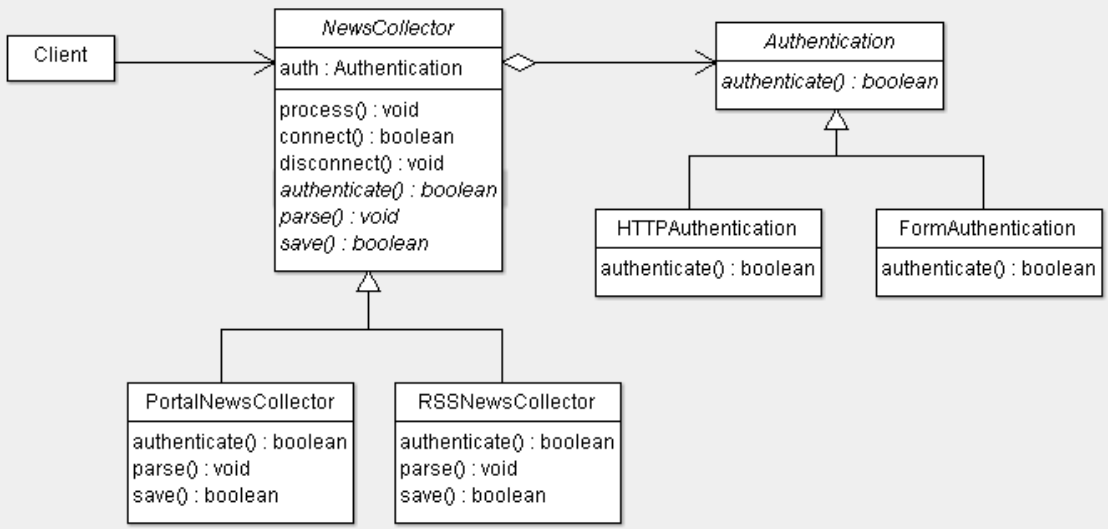
Παρακάτω παρουσιάζεται μια απλή υλοποίηση της κλάσης ενός πελάτη και ακολουθεί η έξοδος ύστερα από την εκτέλεσή της.

```
/*
 * Client.java
 */

package gr.teithe.it.dp;

/**
 * Template Method Demonstration
 *
 * @author Charalampos Pournaris
 */
public class Client {
    public static void main(String[] args) {
        // Create news object and process news from the portal
        NewsCollector news = new PortalNewsCollector("portal.example.com", 80);

        news.process("test", "secret");

        System.out.print('\n');

        // Create new instance of the news object and process news from rss feeds
        news = new RSSNewsCollector("rss.example.com", 8080);

        news.process("rssmaster", "n3wz");
    }
}
```

Έξοδος:

```
Connected to the remote host portal.example.com:80
Authenticated using HTTP Authentication
Using HTML parser to grab news
Inserting news data into the database tables
Disconnected from the remote host portal.example.com:80

Connected to the remote host rss.example.com:8080
Authenticated using POST on HTML FORM
Using XML parser to grab news from RSS
Appending news to a regular file
Disconnected from the remote host rss.example.com:8080
```

Δομή



2.6.1 Διάγραμμα κλάσης για το πρότυπο σχεδίασης «Μέθοδος Υπόδειγμα»

Εναλλακτικός τρόπος σχεδίασης χωρίς την χρήση προτύπου

Χωρίς την χρήση του προτύπου θα έπρεπε σε κάθε λειτουργία του αλγορίθμου (ταυτοποίηση, επεξεργασία, αποθήκευση) να προσθέσουμε δομές ελέγχου έτσι ώστε να εκτελούμε κάθε φορά τις σωστές ενέργειας ανάλογα με την πηγή από την οποία θέλουμε να συλλέξουμε ειδήσεις.

Για παράδειγμα η μέθοδος authenticate θα ήταν κάπως έτσι:

```
private boolean authenticate(String username, String password);
    switch (newsSource) {
        case PORTAL_NEWS:
            auth = new HTTPAuthentication(username, password);
            break;
        case RSS_NEWS:
            auth = new FormAuthentication(username, password);
            break;
    }

    auth.authenticate();
}
```

Με παρόμοιο τρόπο θα έπρεπε και στις υπόλοιπες μεθόδους να γίνεται αντίστοιχος έλεγχος. Αυτή η προσέγγιση δυσκολεύει πολύ την συντήρηση της εφαρμογής καθώς για την προσθήκη μιας νέας πηγής ειδήσεων απαιτείτε η τροποποίηση της κλάσης NewsCollector (η οποία πλέον περιλαμβάνει την υλοποίηση για όλες τις μεθόδους του αλγορίθμου και δεν είναι αφηρημένη) και κατ'επέκταση και τροποποίηση της κλάσης του πελάτη.

2.7 Διακοσμητής (Decorator)

Περιγραφή

Το πρότυπο σχεδίασης διακοσμητής *επιτρέπει την προσθήκη επιπλέον αρμοδιοτήτων σε ένα αντικείμενο δυναμικά κατά τον χρόνο εκτέλεσης. Παρέχει μια εναλλακτική και ευέλικτη λύση έναντι της κληρονομικότητας για την επέκταση της λειτουργικότητας ενός αντικειμένου*.

Αρκετές φορές παρουσιάζεται η ανάγκη προσθήκης επιπλέον λειτουργιών σε ορισμένα αντικείμενα μιας κλάσης, όχι όμως σε όλα τα αντικείμενα της κλάσης αυτής. Μία εργαλειοθήκη για την κατασκευή γραφικών διεπαφών για παράδειγμα θα πρέπει να επιτρέπει την προσθήκη ιδιοτήτων όπως είναι το περίγραμμα και η γραμμή κύλισης σε όλα τα συστατικά που περιλαμβάνει (Κουμπιά, Εικόνες, Πλαίσια κτλ.). Ένας τρόπος για την προσθήκη των λειτουργιών αυτών θα ήταν με την χρήση της κληρονομικότητας. Μία κλάση Περίγραμμα (Border) θα μπορούσε να κληρονομήσει από μια κλάση ενός συστατικού (ButtonBorder για παράδειγμα) και να προσθέσει το περίγραμμα, αυτή η λύση όμως δεν είναι ευέλικτη καθώς κάθε κλάση που επεκτείνει η Border Θα πρέπει να έχει περίγραμμα και η επιλογή του περιγράμματος γίνεται στατικά κατά την μεταγλώττιση του κώδικα. Ο πελάτης δηλαδή δεν μπορεί να επιλέξει πότε και πώς θα «διακοσμηθεί» ένα συστατικό.

Μια περισσότερο ευέλικτη προσέγγιση είναι η ενθυλάκωση ενός συστατικού σε ένα άλλο αντικείμενο **διακοσμητή** που προσθέτει το περίγραμμα. Η διεπαφή του διακοσμητή θα πρέπει να είναι συμβατή με την διεπαφή που ορίζει το συστατικό που «διακοσμεί» έτσι ώστε να υπάρχει διαφάνεια στις κλάσεις των πελατών. Το αντικείμενο διακοσμητής προωθεί τις αιτήσεις (μηνύματα) που λαμβάνει στο συστατικό που ενθυλακώνει και μπορεί να εκτελεί επιπλέον λειτουργίες (όπως για παράδειγμα την σχεδίαση περιγράμματος) πριν η μετά την προώθηση του μηνύματος. Επιπλέον η διαφάνεια (εξαιτίας της κοινής διεπαφής συστατικών-διακοσμητών) επιτρέπει την σύνδεση πολλών αντικειμένων διακοσμητών αναδρομικά για την προσθήκη ενός απεριόριστου αριθμού επιπλέον αρμοδιοτήτων σε ένα αντικείμενο.

Το πρότυπο «Διακοσμητής» θα πρέπει να χρησιμοποιείται στις παρακάτω περιπτώσεις:

- Για την προθήκη αρμοδιοτήτων σε συγκεκριμένα αντικείμενα δυναμικά και με διαφάνεια, χωρίς να επηρεάζονται δηλαδή άλλα αντικείμενα
- Για αρμοδιότητες που μπορούν να «αφαιρεθούν» από κάποια αντικείμενα
- Όταν δεν είναι πρακτική η επέκταση λειτουργικότητας με την χρήση κληρονομικότητας. Σε ορισμένες περιπτώσεις είναι δυνατό να υπάρχει ένας μεγάλος αριθμός από ανεξάρτητες επεκτάσεις που μπορεί προκαλέσει εκρηκτικό αριθμό υποκλάσεων για την υποστήριξη όλων των δυνατών συνδυασμών αυτών των επεκτάσεων

Κίνητρο – Παράδειγμα

Θεωρούμε ένα σύστημα που επιτρέπει στον πελάτη την σύνδεση σε έναν απομακρυσμένο διακομιστή από τον οποίο μπορεί να λάβει ηλεκτρονική αλληλογραφία. Η λήψη των ηλεκτρονικών μηνυμάτων μπορεί να γίνει είτε με την χρήση του πρωτοκόλλου POP3 ή του πρωτοκόλλου IMAP.

Μία αφηρημένη κλάση «Ανάγνωσης μηνυμάτων» (MailReader) περιλαμβάνει τα κοινά χαρακτηριστικά και λειτουργίες των υποκλάσεων καθώς και ορισμένες αφηρημένες μεθόδους που πρέπει να υλοποιηθούν ξεχωριστά από τις υποκλάσεις για κάθε πρωτόκολλο. Οι συγκεκριμένες κλάσεις POP3MailReader και IMAPMailReader κληρονομούν από την MailReader και υλοποιούν τις λειτουργίες για τα πρωτόκολλα POP3 και IMAP αντίστοιχα.

Εξαιτίας του μεγάλου αριθμού των διαφημιστικών/μολυσμένων με ιούς μηνυμάτων που λαμβάνουν οι πελάτες αποφασίζουμε να προσθέσουμε λειτουργικότητα (με την χρήση αντικειμένων διακοσμητών) για το «φιλτράρισμα» τέτοιων ενοχλητικών μηνυμάτων. Προσθέτουμε στο σύστημα την αφηρημένη κλάση Φίλτρο (Filter) η οποία κληρονομεί από την MailReader έτσι ώστε να έχουν συμβατές διεπαφές, παρέχοντας διαφάνεια στους πελάτες. Η κλάση Filter διατηρεί μία αναφορά σε ένα αντικείμενο τύπου MailReader και προωθεί σε αυτό τα μηνύματα που λαμβάνει. Δύο υποκλάσεις της αφηρημένης κλάσης Filter με ονομασίες SpamFilter και VirusFilter αναλαμβάνουν την διαγραφή των διαφημιστικών και μολυσμένων ηλεκτρονικών μηνυμάτων αντίστοιχα, προσθέτοντας επιπλέον αρμοδιότητες στα αντικείμενα POP3MailReader και IMAPMailReader.

Παρακάτω παρουσιάζεται ο κώδικας του συστήματος σε JAVA.

Κώδικας

Η κλάση Mail αναπαριστά ένα ηλεκτρονικό μήνυμα και περιλαμβάνει μία σειρά από χαρακτηριστικά όπως είναι το θέμα, το κείμενο του μηνύματος καθώς και αρχεία που πιθανός να έχουν αποσταλεί μαζί με το μήνυμα και μεθόδους για την πρόσβαση στα στοιχεία αυτά.

```
/*
 * Mail.java
 */

package gr.teithe.it.dp;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

/**
 *
 * @author Charalampos Pournaris
```

```
*/
public class Mail {
   private String subject, from, contents;
   List<byte[]> attachments = null; // Raw bytes for each attachment (base64 decoded)

   public Mail(String subject, String from, String contents) {
      this.subject = subject;
      this.from = from;
      this.contents = contents;
   }

   public void addSampleAttachments() {
      // Suppose we have attachments
      attachments = new ArrayList<byte[]>(2);
      attachments.add("Virus infected attachment 1!".getBytes());
      attachments.add("Exam results in XLS file".getBytes());
   }

   public String getFrom() {
      return from;
   }

   public String getContents() {
      return contents;
   }

   public void appendContent(String s) {
      contents += s;
   }

   public String getSubject() {
      return subject;
   }

   public List<byte[]> getAttachments() {
      return attachments;
   }
}
```

Η αφηρημένη κλάση ανάγνωσης μηνυμάτων MailReader που χρησιμοποιείτε από τους πελάτες για την λήψη των ηλεκτρονικών μηνυμάτων, περιλαμβάνει μία λίστα όπου αποθηκεύονται τα μηνύματα, πληροφορίες για την σύνδεση στον απομακρυσμένο διακομιστή και κάποιες άλλες λειτουργίες που είναι κοινές για όλες τις υποκλάσεις. Επιπλέον ορίζει αφηρημένους μεθόδους για την σύνδεση και λήψη των μηνυμάτων που πρέπει να υλοποιηθούν από τις υποκλάσεις για κάθε πρωτόκολλο.

```
/*
 * MailReader.java
 */

package gr.teithe.it.dp;

import java.util.ArrayList;
import java.util.List;
```

```java
/**
 *
 * @author Charalampos Pournaris
 */
public abstract class MailReader {
   protected String host, username, password;
   protected int port;
   protected List<Mail> mails;

   public MailReader(String host, int port, String username, String password) {
      this.host = host;
      this.port = port;
      this.username = username;
      this.password = password;

      mails = new ArrayList<Mail>(0);
   }

   public void saveMails() {
      for (Mail m : mails) {
         System.out.println("Saving message with subject \"" + m.getSubject()
+ "\" to the mail file.");
      }
   }

   public List<Mail> getMails() {
      return mails;
   }

   public void printMails() {
      int i = 1;
      List <byte[]> attachments;

      for (Mail m : mails) {
         System.out.println("\n******** MAIL #" + (i++) + " ********");
         System.out.println("Subject: " + m.getSubject());
         System.out.println("From: " + m.getFrom());
         System.out.println("Contents: " + m.getContents());
         System.out.println("\nAttachments\n----------");
         attachments = m.getAttachments();
         int aid = 1;
         if (attachments != null) {
            for (byte[] b : attachments) {
               System.out.println("File #" + (aid++) + " :" + new String(b));
            }
         } else {
            System.out.println("No attachments.");
         }
         System.out.println("**************************\n");
      }
   }

   public abstract boolean connect();
   public abstract boolean authenticate();
   public abstract void disconnect();
   public abstract void retrieveMails(int num_mails);
```

```
}
```

Η κλάση POP3MailReader επεκτείνει την MailReader υλοποιώντας τις αφηρημένες λειτουργίες για το πρωτόκολλο POP3.

```
/*
 * POP3MailReader.java
 */

package gr.teithe.it.dp;

/**
 *
 * @author Charalampos Pournaris
 */
public class POP3MailReader extends MailReader {
   public POP3MailReader(String host, int port, String username, String
password) {
      super(host, port, username, password);
   }

   public boolean connect() {
      System.out.println("[POP3] Connected to the remote server @ " + host +
            ":" + port);

      return true;
   }

   public boolean authenticate() {
       System.out.println("[POP3] Authenticated");

       return true;
   }

   public void disconnect() {
      System.out.println("[POP3] Disconnected from the remote server");
   }

   public void retrieveMails(int num_mails) {
      // Suppose we retrieved 2 messages from the remote pop3 server
      Mail m1 = new Mail("Bad news for today", "bignet@zorg.ch",
            "We have some bad news for you.. you are fired!");
      m1.addSampleAttachments();
      mails.add(m1);
      mails.add(new Mail("*SPAM* You have won", "unknown@spam.org",
            "You are very lucky today. You won 2.000.000$ Congratulations"));
   }
}
```

Παρομοίως η κλάση IMAPMailReader παρέχει την υλοποίηση για το πρωτόκολλο IMAP.

```
/*
 * IMAPMailReader.java
 */
```

```
package gr.teithe.it.dp;

/**
 *
 * @author Charalampos Pournaris
 */
public class IMAPMailReader extends MailReader {
   public IMAPMailReader(String host, int port, String username, String
password) {
      super(host, port, username, password);
   }

   public boolean connect() {
      System.out.println("[IMAP] Connected to the remote server @ " + host +
            ":" + port);

      return true;
   }

   public boolean authenticate() {
      System.out.println("[IMAP] Authenticated");

      return true;
   }

   public void disconnect() {
      System.out.println("[IMAP] Disconnected from the remote server");
   }

   public void retrieveMails(int num_mails) {
      // Suppose we retrieved 2 messages from the remote imap server
      mails.add(new Mail("*SPAM* You have won", "unknown@spam.org",
            "You are very lucky today. You won 2.000.000$ Congratulations"));
      mails.add(new Mail("Bad news for today", "bignet@zorg.ch",
            "We have some bad news for you.. you are fired!"));
   }
}
```

Η αφηρημένη κλάση Φίλτρο (Filter) παίζει τον ρόλο του διακοσμητή στο σύστημά μας για την επέκταση της λειτουργικότητας, επιτρέποντας στις συγκεκριμένες υποκλάσεις της να εκτελούν κάποιες επιπλέον λειτουργίες πριν η μετά την κλήση της αντίστοιχης μεθόδου του αντικειμένου που διακοσμεί. Διατηρεί μία αναφορά τύπου MailReader και προωθεί τα μηνύματα που λαμβάνει στο αντικείμενο που δείχνει κάθε φορά η αναφορά.

```
/*
 * Filter.java
 */

package gr.teithe.it.dp;

/**
 *
 * @author Charalampos Pournaris
```

```
 */
public abstract class Filter extends MailReader {
   protected MailReader reader;

   public Filter(MailReader r) {
      super(null, -1, null, null);
      reader = r;
   }

   public boolean connect() {
      return reader.connect();
   }

   public boolean authenticate() {
      return reader.authenticate();
   }

   public void disconnect() {
      reader.disconnect();
   }

   public void retrieveMails(int num_mails) {
      reader.retrieveMails(num_mails);
   }
}
```

Ο συγκεκριμένος διακοσμητής SpamFilter «φιλτράρει» τα μηνύματα αμέσως μετά την λήψη τους σύμφωνα με τις ρυθμίσεις που έχει δώσει ο χρήστης κατά την δημιουργία του αντικειμένου, διαγράφοντας τα ενοχλητικά μηνύματα. Η μοναδική μέθοδος που επικαλύπτει είναι αυτή της λήψης μηνυμάτων retrieveMails(), καθώς όλες οι υπόλοιπες κλήσεις κληρονομούνται από την γονική κλάση Filter που τις προωθεί στο αντικείμενο που δείχνει η αναφορά reader.

```
/*
 * SpamFilter.java
 */
package gr.teithe.it.dp;

import java.util.Iterator;

/**
 *
 * @author Charalampos Pournaris
 */
public class SpamFilter extends Filter {

   private String[] bad_addresses;

   public SpamFilter(MailReader r, String[] bad_addresses) {
      super(r);

      this.bad_addresses = bad_addresses;
   }
```

```
    @Override
    public void retrieveMails(int num_mails) {
        super.retrieveMails(num_mails);

        mails = reader.getMails();

        Iterator<Mail> it = mails.iterator();

        while (it.hasNext()) {
            Mail m = it.next();

            for (int i = 0; i < bad_addresses.length; i++) {
                if (m.getFrom().trim().equalsIgnoreCase(bad_addresses[i].trim()))
{
                    it.remove();
                    break;
                }
            }
        }
    }
}
```

Ένα άλλο φίλτρο (συγκεκριμένος διακοσμητής) με ονομασία VirusFilter αναλαμβάνει τον έλεγχο των μηνυμάτων αμέσως μετά την λήψη τους για ιούς και για κακόβουλα αρχεία, διαγράφοντάς τα.

```
/*
 * VirusFilter.java
 */
package gr.teithe.it.dp;

import java.util.Iterator;

/**
 *
 * @author Charalampos Pournaris
 */
public class VirusFilter extends Filter {

    String external_scanner;

    public VirusFilter(MailReader r, String external_scanner) {
        super(r);

        this.external_scanner = external_scanner;
    }

    @Override
    public void retrieveMails(int num_mails) {
        super.retrieveMails(num_mails);

        boolean notice_appended = false;

        mails = reader.getMails();

        // Suppose we scan attachments with an external virus scanner
```

```
        Iterator<Mail> it = mails.iterator();

        while (it.hasNext()) {
           Mail m = it.next();

           Iterator<byte[]> ait = m.getAttachments().iterator();

           while (ait.hasNext()) {
              byte[] b = ait.next();

              if ((new String(b)).indexOf("Virus") != -1) {
                 if (!notice_appended) {
                    m.appendContent(" [NOD32] Removed virus infected
attachment(s)");
                    notice_appended = true;
                 }
                 ait.remove();
                 break;
              }
           }
        }
     }
}
```

Παρακάτω παρουσιάζεται μια απλή υλοποίηση ενός πελάτη που χρησιμοποιεί το σύστημα.

```
/*
 * Client.java
 */

package gr.teithe.it.dp;

/**
 * Decorator pattern code demonstration
 *
 * @author Charalampos Pournaris
 */
public class Client {
   public static void main(String[] args) {

        MailReader reader = new VirusFilter(
              new SpamFilter(
                 new POP3MailReader("mail.example.net", 110,
"mail@example.net", "s3cr3tmai|"),
                 new String[] { "unknown@spam.org" } // Addresses that are threated
as spam
              ),
              "nod32.exe" // External application to check for viruses
        );

        reader.connect();
        reader.retrieveMails(2);
        reader.printMails();
        reader.saveMails();
```

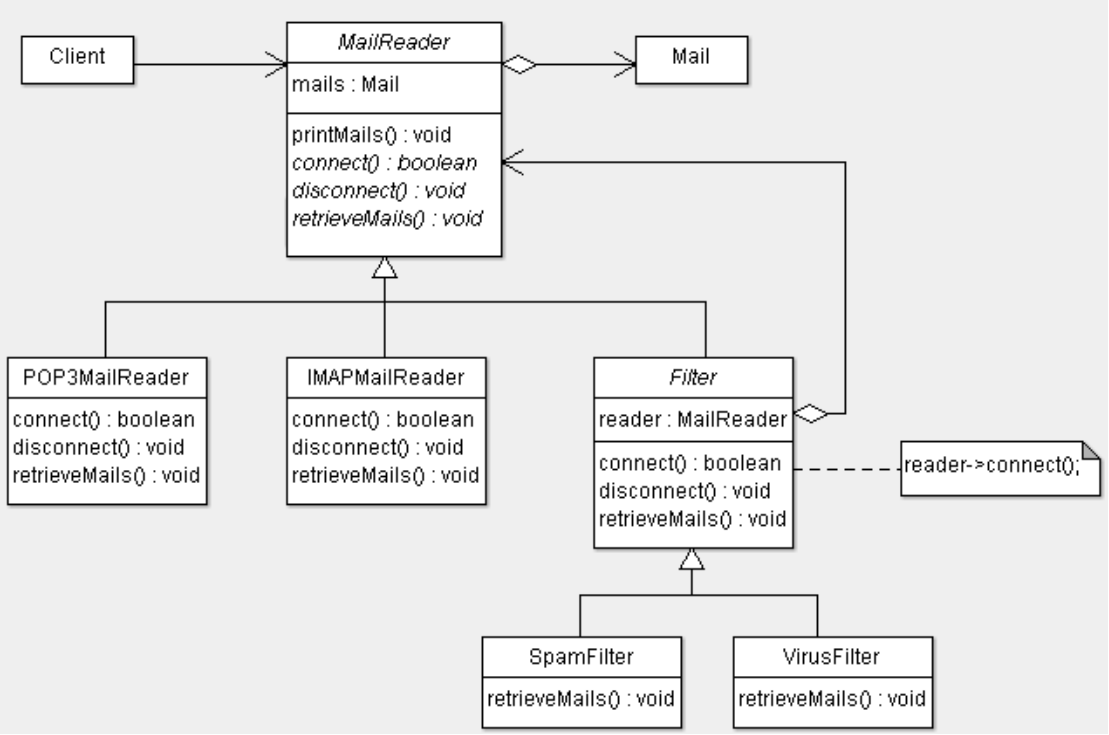
```
      reader.disconnect();
    }
}
```

Παρατηρούμε ότι για την δημιουργία ενός ή περισσότερων φίλτρων το μόνο που πρέπει να κάνουμε είναι να περάσουμε το αντικείμενο τύπου POP3MailReader/IMAPMailReader σαν παράμετρο στον κατασκευαστή του αντικειμένου-διακοσμητή μαζί με όποιες άλλες παραμέτρους χρειάζεται το συγκεκριμένο φίλτρο, καλώντας όμως τις ίδιες μεθόδους που θα καλούσαμε ακόμα και εάν δεν υπήρχε κάποιος διακοσμητής. Με αυτόν τον τρόπο ο πελάτης μπορεί να συνδυάζει με όποιον τρόπο επιθυμεί τα φίλτρα χωρίς να αλλάζει η διεπαφή που γνωρίζει.

Η έξοδος της εφαρμογής φαίνεται παρακάτω:

```
[POP3] Connected to the remote server @ mail.example.net:110

******** MAIL #1 ********
Subject: Bad news for today
From: bignet@zorg.ch
Contents: We have some bad news for you.. you are fired! [NOD32]
Removed virus infected attachment(s)

Attachments
-----------
File #1 :Exam results in XLS file
*************************

Saving message with subject "Bad news for today" to the mail file.
[POP3] Disconnected from the remote server
```

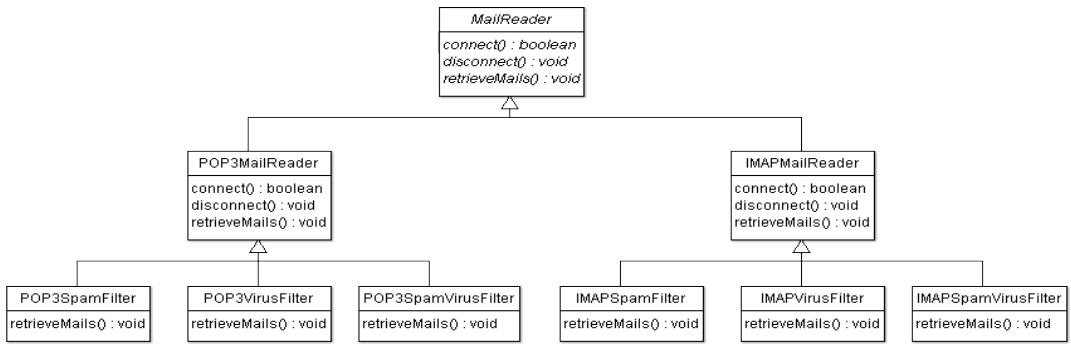
Δομή



2.7.1 Διάγραμμα κλάσης για το πρότυπο σχεδίασης «Διακοσμητής»

Εναλλακτικός τρόπος σχεδίασης χωρίς την χρήση προτύπου

Χωρίς την χρήση του προτύπου σχεδίασης θα πρέπει κάθε φίλτρο να κληρονομεί από την κλάση που διακοσμεί για την επέκταση της λειτουργικότητας και για κάθε δυνατό συνδυασμό φίλτρων χρειάζεται μια ξεχωριστή υποκλάση όπως φαίνεται στο παρακάτω διάγραμμα.



2.7.2 Διάγραμμα κλάσης για το αντί-πρότυπο σχεδίασης «Διακοσμητής»

Μια τέτοια προσέγγιση φυσικά καθιστά την συντήρηση και επέκταση του συστήματος πολύ δύσκολη και το κόστος μεγάλο για ενδεχόμενες αλλαγές.

Συμπεράσματα

Τα πρότυπα σχεδίασης παίζουν πολύ σημαντικό ρόλο στην σχεδίαση και υλοποίηση επεκτάσιμων και εύκολα συντηρήσιμων συστημάτων αντικειμενοστραφούς προγραμματισμού και πολλές φορές οι προγραμματιστές τα χρησιμοποιούν χωρίς να το αντιλαμβάνονται. Επιπλέον, δίνουν ευέλικτες και κομψές λύσεις σε επαναλαμβανόμενα προβλήματα που προκύπτουν κατά την σχεδίαση νέων συστημάτων. Στην εργασία αυτή παρουσιάστηκαν, αναλύθηκαν και υλοποιήθηκαν (σε κώδικα JAVA) τα σημαντικότερα και ποιο ευρέως χρησιμοποιούμενα πρότυπα με την χρήση πραγματικών παραδειγμάτων για την καλύτερη κατανόηση καθώς και με διαγράμματα UML που παρουσιάζουν σχηματικά όλα τα στοιχεία μίας σχεδίασης και τις σχέσεις που έχουν μεταξύ τους.

Αναφορές

http://en.wikipedia.org/wiki/ (Wikipedia)

http://java.sun.com/j2se/ (Java Api)

http://www.java2s.com/Code/Java/Design-Pattern/BridgePattern1.htm

http://www.oodesign.com/factory-pattern.html

Βιβλιογραφία

Αλέξανδρος Χατζηγεωργίου (2005) , *Αντικειμενοστρεφής σχεδίαση UML, Αρχές, Πρότυπα και Ευρετικοί κανόνες*.

Herbert Schildt, *Java2 Guide.*

John Zukowski (2005), *The definitive guide to Java Swing.*

Craig Larman (2004), Applying UML and Patterns (3rd).

Brian W. Kernighan, Dennis M. Ritchie (1978), *The C Programming Language (2nd).*

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (1994), *Elements of Resusable Object-Oriented Software.*