# Άσκηση 1

Μιχαήλ Μυλωνάκης

4 Νοεμβρίου 2015

## 1 Visitor

**1.** *Γιατί ταίριαζε για τη χρήση αυτή το πρότυπο που διαλέξατε;*

Επιθυμούμε να μπορούμε να προσθέτουμε συνεχώς νέες λειτουργίες στις κλάσεις μας, χωρίς να αλλάζουμε το object structure. Θέλουμε να υλοποιήσουμε ουσιαστικά διαφορετικούς αλγορίθμους (`Schedulers`), που δρούν πάνω στο ίδιο data set (OperatingSystem).Επίσης θέλουμε να προσομοιώσουμε το double dispatch: Θέλουμε τα `OperatingSystem (LinuxOS, MacOS, WindowsOS)`, να καλούν την αντίστοιχη `schedule()` ανάλογα με τον δυναμικό τύπο του `Scheduler`, τον οποίο θα γνωρίζουν μόνο κατά την εκτέλεση του προγράμματος. Το πρότυπο `Visitor` μας δίνει την δυνατότητα να ορίσουμε νέες δυνατότητες στις κλάσεις τύπου `OperatingSystem`, χωρίς να χρειάζεται να αλλάξουμε τον κώδικά τους, και να υλοποιήσουμε το double dispatch στην C++.

**2.** *Ποιά είναι τα πλεονεκτήματα / μειονεκτήματα της υλοποίησής σας;*

Το βασικότερο πλεονέκτημα, είναι αυτό που αναφέρθηκε στην προηγούμενη παράγραφο, και αφορά το double dispatch. Υλοποιώντας το `Visitor`, ο δυναμικός του τύπος χρησιμοποιείται, και οδηγούμαστε στην λειτουργικότητα του κατάλληλου `Scheduler`. Επίσης μπορούμε πολύ εύκολα να προσθέτουμε νέες λειτουργίες στις κλάσεις μας (`OperatingSystem`), απλά προσθέτωντας έναν καινούριο `Visitor`, χωρίς να χρειάζεται να αλλάξουμε τον κώδικά τους. Αυτό μας βοηθάει στο να κάνουμε κατά κάποιο τρόπο δεσουπλε τα αντικείμενα μας , από τις λειτουργίες πάνω σε αυτά (φυσικά με περιορισμούς).

Ένα σημαντικό μειονέκτημα της υλοποίησης είναι το overhead που επιφέρει αυτό το διπλό dispatch που προβλέπει το πρότυπο. Αν η συγκεκριμένη λειτουργικότητα πρόκειται να εκτελείται

συχνά (κάτι που μάλλον ισχύει στην περίπτωση του `SchedulerVisitor`, το overhead των virtual κλήσεων είναι σημαντικό.

**3.** *Πέρα από τις πολύ βασικές αντικειμενοστραφείς λειτουργίες, υπάρχουν στοιχεία στη γλώσα που διαλέξατε που σας βοήθησαν να υλοποιήσετε το πρότυπο πιο εύκολα; Υπάρχουν στοιχεία σεάλλες γλώσσες που θα σας βοηθούσαν;*

Το Overloading είναι ο βασικότερος παράγοντας που μας επιτρέπει να υλοποιήσουμε το Visitor Pattern. Το Overloading είναι αυτό που βοηθάει τον κάθε `Visitor` να αποφασίσει ποια μέθοδος τελικά θα κληθεί.

Στοιχεία σε άλλες γλώσσες που θα μπορούσαν να βοηθήσουν στην υλοποίηση του συγκεκριμένου προτύπου, είναι το build-in Pattern Matching που παρέχουν οι περισσότερες συναρτησιακές γλώσσες. Σε αυτήν την περίπτωση, ο `Vistor` Θα μπορούσε να ελέγχει όλους τους δυνατούς υποτύπους των visitable κλάσεων, και η ίδια η γλώσσα θα έκανε σε compile time τον έλεγχο, του εαν έχουν εξεταστεί όλες οι δυνατές πετιπτώσεις.

# 2 Observer

**1.** *Γιατί ταίριαζε για τη χρήση αυτή το πρότυπο που διαλέξατε;*

Το πρότυπο `Observer` ταιριάζει σε αυτό το use case, γιατί θέλουμε όλα αντικείμενα που εξαρτώνται από τον κεντρικό πίνακα του συναγερμού `Alarm`, που είναι η σειρήνα `Siren`, πληκτρολόγιο (με ένδειξη) , `Mailer`, να είναι ανά πάσα στιγμή ενήμερα για την κατάσταση του συναγερμού σε πραγματικό χρόνο. Θέλουμε να το επτύχουμε αυτό χωρίς να είναι coupled αυτές οι κλάσεις με την κλάση του κεντρικού συναγερμού, έτσι ώστε να έχουμε ανεξάρτητο development και reusability. Έτσι, ο συναγερμός `Alarm`, θα μπορεί να δεχτεί οποιονδήποτε νέο παρατηρητή (π.χ ένα sms modem), χωρίς να χρειάζεται να ξέρει κάτι για την υλοποίησή του.

**2.** *Ποιά είναι τα πλεονεκτήματα / μειονεκτήματα της υλοποίησής σας;*

Το προφανές πλεονέκτημα της υλοποίησης αυτής, είναι ότι επιτυγχάνεται εύκολα το consistency μεταξύ των Observers. Περνώντας το `this` στην `publish`, ένας `observer` μπορεί να παρατηρεί περισσότερους από έναν `Publishers`. Τα updates γίνονται fire στην αλλαγή του state. Έτσι δεν χρειάζεται ο `Observer` να θυμάται ότι πρέπει να ενημερώσει `(Publish)` όλους τους υπόλοιπους. Θα μπορούσε βέβαια σε κάποιο άλλο σενάριο, να μην θέλουμε αυτήν την λειτουρ-

γικότητα, σε περίπτωση που είχαμε πολλές συνεχόμενες αλλαγές στο `State` του `Publisher` από κάποιον client, και μας ενδιέφερε μόνο το τελευταίο state. Με την τρέχουσα υλοποίηση, θα γινόντουσαν fire οι ενημερώσεις για όλα τα ενδιάμεσα States.

**3.** *Πέρα από τις πολύ βασικές αντικειμενοστραφείς λειτουργίες, υπάρχουν στοιχεία στη γλώσα που διαλέξατε που σας βοήθησαν να υλοποιήσετε το πρότυπο πιο εύκολα; Υπάρχουν στοιχεία σεάλλες γλώσσες που θα σας βοηθούσαν;*

Δεδομένου ότι οι `Observers`, το μόνο που οφείλουν είναι να υλοποιήσουν την `update` για να ορίσουν το τι θα κάνουν όταν ο `Publisher` αλλάξει state, θα βοηθούσε να μπορούμε να έχουμε κάτι σαν τις Anonymous classes της java, ή τα Delegates της C++ έτσι ώστε να μην χρειάζεται δημιουργήσουμε κλάσεις που να κάνουν Extend την `Observer`. Ουσιαστικά το μόνο συμβόλαιο που έχουμε με τον `Observer` είναι αυτή η `update()`, συνεπώς θα βόλευε να μπορούσαμε να δώσουμε μια on-the-fly υλοποίησή της.