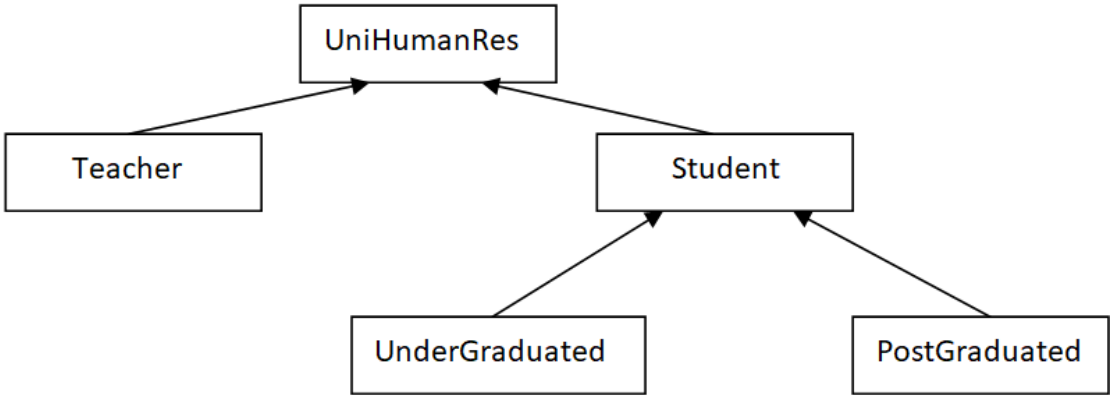
Προηγμένες Μέθοδοι Προγραμματισμού

Άσκηση1

Ονοματεπώνυμο: Πέτρος Πετσίνης
A.M. cs2200014

1) Visitor (Dynamic Proxy):

Η ιεραρχία των κλάσεων που υλοποιήθηκε φαίνεται στο παρακάτω σχήμα:



Φάκελος: **Visitor**

Αρχεία: **“MyHw1.java”, “MyVisitor.java”, “GetInfo.java” , “UniHandler.java”, “UniHumanRes.java”, “Teacher.java”, “Student.java”, “UnderGraduated.java”, “PostGraduated.java”**

Αρχικά **cd** στο path του φάκελου **Visitor**

Compile**: javac ./*.java**

Run: **java MyHw1**

2) Binary-Tree (AspectJ):

Φάκελος: **BinaryTree**

Αρχεία: **“ajc”, “aspectjrt.jar”, “BinaryTree.java”, “ReadersWritersLock.java”, “AspectSynchronized.aj”, “sources.lst”**

Αρχικά **cd** στο path του φάκελου **BinaryTree**

Compile: **ajc -argfile sources.lst -cp aspectjrt.jar -source 13**

Run: **java -cp aspectjrt.jar:. BinaryTree**

Παρατήρηση: Το ajc ενδέχεται να μην έχει περασμένο execute bit και να θέλει ένα **chmod u+x ajc**

3) Dynamic Proxy – AspectJ:

Το Βήμα 1, δηλαδή αντί να γράψουμε με το χέρι την PrintVisitor, μπορεί να γίνει με χρήση της AspectJ. Υλοποιώντας μία μέθοδο **around** σε **pointcut execution** της **accept**, ενός αντικειμένου κλάσης **UniHumanRes**. Ελέγχοντας με **Reflection** την κλάση τον δυναμικό τύπο του αντικειμένου που κάλεσε την **accept**, καθώς και του ορίσματος (Visitor), εκτελεί την αντίστοιχη λειτουργία, ο κώδικας της οποίας βρίσκεται μέσα στην **around**.

Το Βήμα 2, δηλαδή η υλοποίηση με εξωτερικό τρόπο ενός συγχρονισμό των μεθόδων της κλάσης **BinaryTree**, μπορεί επίσης να γίνει με χρήση Dynamic Proxy. Πιο συγκεκριμένα, μπορεί να οριστεί ένα Interface **BinaryTreeInterface** με μεθόδους (insert, remove και lookup) όμοιες με αυτής της κλάσης **BinaryTree** και μία κλάση **BinaryTreeSynchronizedHandler** που να κάνει implement την **InvocationHandler**. Θα διαθέτει ένα πεδίο **locktree** της κλάσης **ReadWritersLock** που θα λειτουργεί ως ο schreduler των διαδικασιών. Κάνοντας την **BinaryTree implements BinaryTreeInterface** , μπορεί το δέντρο να έχει πρόσβαση στο εξωτερικό **locktree** διασφαλίζοντας τον συγχρονισμό των διαδικασιών (μέσω δημιουργίας με BinaryTreeSynchronizedHandler).

Η ευελιξία σαφώς είναι πιο δυναμική στην περίπτωση της χρήσης Dynamic Proxy, αφού το αντικείμενο δημιουργείται την ώρα της εκτέλεσης και η χρήση της reflection μπορεί να προσφέρει διαφορετική λειτουργικότητα για διαφορετικούς συνδυασμούς κλήσης συναρτήσεων. Υπάρχει κίνδυνος, λοιπόν, runtime errors που δεν μπορούν να εντοπιστούν κατά το στάδιο της μεταγλώττισης. Αντίθετα με την AspectJ υπάρχει μεγαλύτερη στατικότητα, γεγονός που οδηγεί σε μεγαλύτερη ασφάλεια και ταχύτητα εκτέλεσης έναντι του προηγούμενου μηχανισμού. Μέσω pointcuts και advices μπορούμε με χρήση AspectJ να επέμβουμε στην λειτουργικότητα των διαδικασιών και κατ’ επέκταση στου ίδιου του προγράμματος.