Το τμήμα του Aspectj γράφτηκε με την σύνταξη της γλώσσας καθ'αυτής, χωρίς Java annotations, και μεταγλωττίστηκε με τον ajc compiler. Χρησιμοποίησα την έκδοση AspectJ 1.8.9, η οποία προς το παρών και η τελευταία stable και φυσικά Java 8 για τη μεταγγλώτιση της Java. Ο κώδικας σε AspectJ επεξεργάστηκε με Notepad++ σε περιβάλλον Windows 8.1, αφού έγινε η τροποποίηση του CLASSPATH environment variable, ώστε να συμπεριλάβει τη βιβλιοθήκη aspectjrt.jar. Χρησιμοποίησα το makefile που επισυνάπτω στο παρών αρχείο για να κάνω τη μεταγγλώτιση.

Αρχικώς έφτιαξα τον κώδικα της λίστας Thread Safe σε Java (βρίσκεται στο ThreadSafeNLList.java αρχείο το οποίο παραδίδω για λόγους πληρότητας) και έπειτα ασχολήθηκα με τον Aspect Oriented προγραμματισμό σε AspectJ. Δεν είχα ξαναχρησιμοποιήσει AOP και ήταν μια καινούρια εμπειρία για εμένα που μου άνοιξε νέους ορίζοντες στις μεθόδους προγραμματισμού. Έπρεπε να διαβάσω και να ασχοληθώ με αρκετά κομμάτια του AOP σε AspectJ ώστε να μπορέσω να τα συνδυάσω όλα μαζί για να υλοποιήσω ολόκληρο το πρόγραμμα. Στην αρχή αυτή η διαδικασία ήταν ομολογουμένως αρκετά επίπονη (όπως και κάθε αρχή) αλλά σταδιακά μυήθηκα στην ορολογία των Aspects, τα join points, τα advices κοκ. και κατανόησα πως αλληλεπιδρούν με τον κώδικα των .java αρχείων πηγαίου κώδικα.

Χρησιμοποίησα τη σύνταξη .aj διότι μου φάνηκε πιο αυθεντικός τρόπος προγραμματισμού. Η σύνταξη .aj επίσης μου φάνηκε πιο γνώριμη. Δηλαδή ήταν κυρίως αισθητικός λόγος η επιλογή μεταξύ των δύο. Η γλώσσα AspectJ έχει κυριολεκτικά τεράστιο δυναμικό για ανάπτυξη εφαρμογών. Κάνει τον κώδικα φανερώς πιο παραμετροποιήσιμο και απλό. Το σχετικά «δύσκολο» κομμάτι είναι η εκμάθηση της σύνταξης, των αρχών και των νέων keywords της AspectJ και του Aspect Oriented programming paradigm γενικότερα - το οποίο ίσως αρχικά να αποθαρύνει κάποιους προγραμματιστές, αλλά στην πορεία αποδεικνύεται πολύτιμη γνώση και είμαι ευγνώμων που απέκτησα ένα μικρό κομμάτι της.

Για παραγωγή ψευδοτυχαίων ακολουθιών εντολών δε χρησιμοποιήσα τη κλάση java.util.Random. Απλώς τροφοδοτούσα σε ένα loop μια αλληλουχία εντολών που επινόησα εξ'αρχής. Αυτά βρίσκονται φυσικά στη run() μέθοδο της κλάσης MyThread που έκανα χρήση για τη δημιουργία των Threads. Εν ολίγοις, αφού έφτιαξα τη συνδεδεμένη λίστα, έφτιαξα 21 νήματα. Θεώρησα ότι το καθένα φτιάχνει τη δικιά του λίστα. Ο aspect κώδικας καθόριζε το synchronized - multithreading της εφαρμογής. Για synchronized δεν ασχολήθηκα με τις low level ρουτίνες wait(), notify() και notifyAll(), αλλά χρησιμοποίησα το έξοχο Readers-Writers lock που βρίσκεται έτοιμο στη Java εισάγοντας τις Lock, ReadLock & ReentrantReadWriteLock στον .aj κώδικα. Έχω και επιπλέον σχόλια στο πηγαίο κώδικα.

Όσον αφορά το ζήτημα του timeout, το υλοποίησα μόνο στην prepend() μέθοδο αλλά αντιστοίχως θα μπορούσε να γίνει και για τις άλλες. Το aspect, με το before() advice ελέγχει το ενεργό thread εκείνη τη στιγμή πριν μπεί στο critical section και έπειτα με around() advice κλειδώνει τη μέθοδο και ελέγχει το thread που τελικά εισήλθε. Αν τελικώς είναι κάποιο άλλο από αυτό που ανακαλύφθηκε στην before() τότε μειώνεται η τιμή timeout. Εάν φτάσει στο 0 τότε σημαίνει ότι το αρχικό thread διακόπηκε 5 (ή παραπάνω) φορές οπότε του θέτω τη σημαία markForTermination, το οποίο θα το σταματήσει την επόμενη φορά που θα τρέξει, εφόσον θα ολοκληρώσει την run().

Ίσως χρειαστεί να το τρέξετε πάνω από μια φορά για να δείτε threads να σταματούν. Όταν συμβαίνει αυτό εκτυπώνεται στο τερματικό “Child#_ quits.”. Εναλλακτικά μπορείτε να αυξήσετε τον αριθμό τον threads, ή να μειώσετε την αρχική τιμή του timeout για να παρατηρήσετε threads να αποτυγχάνουν συχνότερα.

Η λίστα ονομάζεται NikLList.java, και η main() βρίσκεται στο Ask2TestList.java. Ο aspect κώδικας βρίσκεται στο NikLList_aspect_Ts.aj και η κλάση των threads βρίσκεται στο MyThread.java. Σας δίνω και το Makefile που χρησιμοποίησα για την εκτέλεση.