ΑΤΕΙ ΘΕΣΣΑΛΟΝΙΚΗΣ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
Αντικειμενοστρεφής Προγραμματισμός Η/Υ

Οδηγίες Συγγραφής και Αξιολόγησης Εργασιών του μαθήματος

Το κείμενο αυτό γράφεται με σκοπό να βοηθήσει τους φοιτητές του μαθήματος Αντικειμενοστρεφής Προγραμματισμός του τμήματος Πληροφορικής του ΑΤΕΙΘ στην αξιολόγηση των εργασιών που τους δίνονται στα εργαστήρια του μαθήματος. Τα σχόλια και οι υποδείξεις είναι προσαρμοσμένα στους στόχους του μαθήματος καθώς επίσης και την μορφή των εργασιών και **δεν** προτείνονται ως γενικότερη μεθοδολογία αξιολόγησης λογισμικού.
Για την αξιολόγηση των εργασιών θα πρέπει να αξιολογούνται τα παρακάτω:

- Απόδοση - Ορθότητα και σταθερότητα κώδικα
- Σωστή τεκμηρίωση
- Αντικειμενοστραφής προγραμματισμός
- Δομημένος προγραμματισμός
- Αλγοριθμική

1. Ορθότητα και Σταθερότητα κώδικα

Φυσικά είναι ιδιαίτερα σημαντικό ο κώδικας του προγράμματος να είναι σωστός και σταθερός (robust). Αυτό σημαίνει ότι ο κώδικας θα πρέπει να υλοποιείται σύμφωνα με τις προδιαγραφές που υπάρχουν, θα πρέπει να εκτελείται σωστά και να χειρίζεται με επιτυχία ακραίες καταστάσεις και τιμές χωρίς να τερματίζεται βίαια και πρόωρα. Σε αυτή την κατεύθυνση θα βοηθήσουν και τα σωστά μηνύματα. Μηνύματα που κυρίως θα ενημερώνουν τον χρήστη για το είδος και των εύρος των τιμών που θα πρέπει να εισάγει κάθε φορά.

Εισάγετε λογικούς ελέγχους σε σημαντικά σημεία τού κώδικά σας, όπως η αρχή των "`public`" μεθόδων/συναρτήσεων ή το τέλος του κώδικα ο οποίος πρέπει οπωσδήποτε να επιστρέψει επιτυχώς κάτι και οπουδήποτε το εύρος των τιμών είναι σημαντικό.

Ένας καλός έλεγχος της ορθότητας και σταθερότητας του κώδικά σας μπορεί να γίνει από οποιονδήποτε «άσχετο/η» με το πρόγραμμά σας (ακόμη και με την Πληροφορική). Βάλτε τον/την να δοκιμάσει το πρόγραμμά σας. Αν για ένα σχετικά μεγάλο έργο το κάνετε με πέντε διαφορετικούς ανθρώπους θα έχετε ανακαλύψει πάνω από 95% των πιθανών προβλημάτων χρήσης του κώδικά που γράψατε. Αν για τις εργασίες σας βρείτε δύο τέτοια άτομα να κάνουν αυτό τον έλεγχο να είστε σίγουροι ότι θα ανακαλύψουν το σύνολο των προβλημάτων που υπάρχουν στον κώδικά σας.

Η σωστή χρήση των βιβλιοθηκών της γλώσσας είναι κάτι που επίσης επηρεάζει την απόδοση του προγράμματος. Στα πλαίσια του μαθήματος Αντικειμενοστρεφής

Προγραμματισμός αυτό δεν είναι ιδιαίτερα δύσκολο αφού οι βιβλιοθήκες που χρησιμοποιούνται είναι ελάχιστες. Δεν πρόκειται να αποφασίσετε αν θα χρησιμοποιήσετε πίνακα ή vector ή ακόμη πως θα υλοποιήσετε μία συλλογή αντικειμένων μεταβλητού μεγέθους. Θα πρέπει όμως να γνωρίζετε καλά τις μεθόδους της κλάσης "`String`" όπως και τις μεθόδους της "`Object`" τις οποίες μπορείτε να υπερβείτε και να τις δώσετε την λειτουργία που επιθυμείτε, πχ. Η μέθοδος "`toString()`". Σε κάποιες εργασίες, θα πρέπει να αποφασίσετε αν θα χρησιμοποιήσετε την "`Math.random()`" ή κάποια από τις μεθόδους της κλάσης "`Random`". Γενικά όταν θέλετε περισσότερους και όχι μόνο 1-2 τυχαίους αριθμούς είναι καλύτερα να χρησιμοποιείται τις μεθόδους της κλάσης "`Random`".

Για εκπαιδευτικούς λόγους, στις εργασίες πολλές φορές σας ζητείται να δημιουργήσετε κάποια μέθοδο ή κλάση η οποία υπάρχει ήδη στην βιβλιοθήκη της Java. Γενικά όταν λειτουργείτε ως προγραμματιστές, πρέπει να έχετε στο νου σας ότι είναι καλύτερα να χρησιμοποιήσετε κάποια έτοιμη βιβλιοθήκη η οποία έχει ελεγχθεί και δοκιμαστεί παρά να γράψετε τον δικό σας κώδικα από την αρχή.

2. Στυλ Κωδικοποίησης

2.1. Σχολιασμός

Η εισαγωγή σχολίων είναι όχι μόνο χρήσιμη αλλά απαραίτητη εάν θέλουμε να έχουμε την δυνατότητα συντήρησης του κώδικα που αναπτύσσουμε. Πολλές φορές η εισαγωγή κατάλληλων σχολίων που θα μας βοηθήσουν (εμάς, τους συνεργάτες μας ή άλλους που θα ασχοληθούν) στην κατανόηση του κώδικα, είναι δυσκολότερη από την συγγραφή ακόμη και του ίδιου του κώδικα. Προσοχή η εισαγωγή σχολίων θα πρέπει να γίνεται όταν τελειώνει η υλοποίηση και όχι όσο μεταβάλλεται ο κώδικας.

Ο επαρκής σχολιασμός του κώδικα δεν περιλαμβάνει μόνο την χρήση και την λειτουργία των συνολικών (global) μεταβλητών αλλά και των μεταβλητών δομών και κλάσεων. Κάποιες φορές χρειάζονται σχόλια ακόμη και σε τοπικές μεταβλητές. (Οι βοηθητικές μεταβλητές θα πρέπει πάντα να δηλώνονται ως τοπικές). Προσοχή, αυτό δεν σημαίνει ότι θα πρέπει να έχουμε σχόλια παντού γιατί αυτό θα οδηγήσει στην απαξίωση των σχολίων. Δεν εισάγουμε σχόλια σε κώδικα ο οποίος είναι προφανής. Στις τοπικές μεταβλητές συνήθως θα πρέπει να αφήνουμε το όνομα της μεταβλητής να εξηγεί την χρήση της. Επίσης σχόλια δεν χρειάζονται σε μεθόδους πρόσβασης και μεταβολής (get/set).

Οι μη συνήθεις συναρτήσεις/μέθοδοι που ορίζονται θα πρέπει να έχουν σχόλια τα οποία εξηγούν τι κάνουν, τι παραμέτρους δέχονται και τι επιστρέφουν. Κάποιες φορές χρειάζεται να αναφέρονται κάποιες τιμές των παραμέτρων οι οποίες μπορούν να οδηγήσουν σε μη αναμενόμενη συμπεριφορά και αποτελέσματα. Σχόλια είναι απαραίτητα και για τμήματα κώδικα τα οποία υλοποιούν ένα πολύπλοκο αλγόριθμο.

Προσοχή, δεν είναι απαραίτητο να εισάγουμε σχόλια για τα πάντα αλλά μόνο για ότι δεν είναι προφανές. Ο υπερβολικός σχολιασμός αποτελεί επίσης πρόβλημα ειδικά κατά την συντήρηση του κώδικα όπου λησμονούμε να ενημερώσουμε τα σχόλια (ειδικά όταν είναι ιδιαίτερα λεπτομερή) όταν μετατρέπουμε τον κώδικα. Έτσι τα σχόλια του κώδικα παραπλανούν αντί να βοηθούν. Προτείνεται τα σχόλια να γράφονται με λατινικούς χαρακτήρες.

Τελειώνοντας να υπενθυμίσω και να τονίσω ότι τα σχόλια υπάρχουν για την κατανόηση του κώδικα του προγράμματος.

2.2. Ονοματοδοσία

Τα ονόματα των αναγνωριστικών (identifiers) πρέπει να αντιστοιχούν κατά το δυνατό σε αυτό που αντιπροσωπεύουν. Τα ονόματα των αναγνωριστικών είναι καλύτερα να είναι περιγραφικά. Αντί να χρησιμοποιήσετε το όνομα "`gtotd()`" είναι καλύτερα να χρησιμοποιήσετε το όνομα "`getTimeOfTheDay()`" ή το "`get_time_of_the_day()`" (Συνηθίζεται στη C/C++). Έτσι ο κώδικας γίνεται πιο ευανάγνωστος και σχεδόν αυτό-τεκμηριώνεται.

Ταυτόχρονα όμως θα πρέπει να προσέχουμε να μην γίνεται κατάχρηση στο πλήθος των χαρακτήρων ενός αναγνωριστικού. Το πλήθος των χαρακτήρων πρέπει να είναι όσο το δυνατό μικρότερο. Δεν υπάρχει λόγος να χρησιμοποιούμε ένα μακρύ και περιγραφικό όνομα στο δείκτη μιας δομής επανάληψης for, όπου το "`i`" είναι αρκετό. Αντιθέτως το "`i`" δεν είναι αρκετό όταν αντιπροσωπεύει μία μεταβλητή η οποία αντιπροσωπεύει το έτος εισαγωγής ενός σπουδαστή σε κάποια σχολή.

Για τα αναγνωριστικά που χρησιμοποιούνται ως ονόματα μεταβλητών και μεθόδων, προτείνεται η χρήση πεζών χαρακτήρων. Εάν το αναγνωριστικό αποτελείται από περισσότερες από μία λέξεις τότε κάθε λέξη εκτός από την πρώτη θα πρέπει να αρχίζει από κεφαλαίο χαρακτήρα. Σε ονόματα αναγνωριστικών κλάσεων όλες οι λέξεις αρχίζουν από κεφαλαίο. Τα ονόματα καθολικών σταθερών (public static final) προτείνεται να γράφονται με κεφαλαία γράμματα.

Σε οποιαδήποτε περίπτωση θα πρέπει να αποφεύγεται η χρήση διαφορετικών τρόπων ονοματοδοσίας σε ένα πρόγραμμα όπως πχ η δήλωση των μεθόδων setEtosEisagogis και get_etos_eisagogis. Δείχνει την έλλειψη συγκέντρωσης του προγραμματιστή. Ακόμη και αν το πρόγραμμα γράφεται από διαφορετικά άτομα, δείχνει την κακή συνεργασία μεταξύ τους και μας προϊδεάζει να περιμένουμε λάθη στην σύνδεση των τμημάτων κώδικα που έγραψαν διαφορετικοί προγραμματιστές.

2.3. Στοίχιση και κενά (indentation - spacing)

Η κατάλληλη στοίχιση του κώδικα τα κενά και οι κενές γραμμές βοηθούν να διαχωριστούν τα στοιχεία του προγράμματος και κάνουν τον κώδικα ευανάγνωστο.

Υπάρχουν διάφορα είδη στοίχισης που μπορείτε να χρησιμοποιήσετε και προτείνεται η χρήση κάποιου από αυτούς. Χειρότερα ακόμη είναι να χρησιμοποιήσετε περισσότερους από έναν στο ίδιο αρχείο ή στην ίδια κλάση. Οπωσδήποτε δεν θα ήθελα να βρεθώ στη θέση κάποιου που θα πρέπει να κατανοήσει κώδικα που μπορεί να συνδυάζει περισσότερα είδη στοίχισης ακόμη και σε τμήματα κώδικα.

Για παράδειγμα στον κώδικα του πυρήνα του Linux χρησιμοποιούνται στηλοθέτες μεγέθους 8 κενών θέσεων. Είναι αρκετό για να φαίνεται η αρχή και των τέλος τμημάτων κώδικα και από την άλλη σας βοηθάει να γράφεται πιο δομημένο κώδικα αναγκάζοντας σας να χωρίζεται τον κώδικα σε περισσότερες μεθόδους. Αν η στοίχιση σας πάει πολύ δεξιά, αυτό είναι ένδειξη κακής σχεδίασης της μεθόδου και σας παροτρύνει να την χωρίσετε σε περισσότερες μεθόδους/συναρτήσεις.

Προτείνεται η χρήση κενών χαρακτήρων κατά τον σχηματισμό των εντολών όπως επίσης και κατά τον διαχωρισμό των τελεστών από τους τελεστέους όπως επίσης και η χρήση κενών γραμμών μεταξύ των μεθόδων και πολλές φορές ανάμεσα και σε τμήματα κώδικα της ίδιας μεθόδου.

3. Δομημένος Προγραμματισμός

Ο δομημένος προγραμματισμός παρουσιάστηκε στα μέσα του 60 και ήταν η επικρατούσα μεθοδολογία πριν τον αντικειμενοστραφή προγραμματισμό ο οποίος είναι υπερσύνολό του. Δεν είναι απλώς ένα είδος προγραμματισμού, είναι μια μεθοδολογία σύνταξης προγραμμάτων που έχει σκοπό να βοηθήσει τον προγραμματιστή στην ανάπτυξη σύνθετων προγραμμάτων, να μειώσει τα λάθη, να εξασφαλίσει την εύκολη κατανόηση των προγραμμάτων και να διευκολύνει τις διορθώσεις και τις αλλαγές σ' αυτά.

Ο δομημένος προγραμματισμός στηρίζεται στη χρήση τριών και μόνο στοιχειωδών λογικών δομών, τη δομή της ακολουθίας (οι εντολές εκτελούνται με τη σειρά), τη δομή της επιλογής και τη δομή της επανάληψης. Όλα τα προγράμματα μπορούν να γραφούν χρησιμοποιώντας μόνο αυτές τις τρεις δομές καθώς και συνδυασμό τους. Κάθε πρόγραμμα όπως και κάθε ενότητα προγράμματος έχει μόνο μια είσοδο και μόνο μια έξοδο.

Ο δομημένος προγραμματισμός ως μεθοδολογία εμπεριέχει τις λογικές του ιεραρχικού (**Top down** - είναι η διάσπαση του προβλήματος σε μια σειρά από απλούστερα υπο-προβλήματα) και του τμηματικού προγραμματισμού (υλοποίηση ιεραρχικής σχεδίασης και ανάπτυξης προγραμμάτων ως σύνολο από απλούστερα τμήματα προγραμμάτων).

Όπως ειπώθηκε, ο δομημένος προγραμματισμός ενθαρρύνει και βοηθάει την ανάλυση του προβλήματος σε επί μέρους προβλήματα. Η αντιμετώπιση μικρότερων προβλημάτων είναι ευκολότερη. Στόχος είναι κάθε υπο-πρόβλημα να αντιμετωπίζεται κατά το δυνατό ανεξάρτητα. Εάν κάποιο τμήμα είναι αρκετά πολύπλοκο είναι καλύτερα να διαιρεθεί σε ακόμη μικρότερα υπο-προβλήματα τα οποία θα επιλυθούν ευκολότερα. Κάθε υπο-πρόβλημα αντιμετωπίζεται και υλοποιείται από διαφορετική μέθοδο. Όταν ο κώδικας της μεθόδου αρχίζει να ξεπερνάει την μία οθόνη θα πρέπει να αρχίζουμε να σκεφτόμαστε μήπως πρέπει να την διαιρέσουμε σε περισσότερες μεθόδους.

Η αντιγραφή ενός τμήματος κώδικα από ένα μέρος σε άλλο με μικρές μόνο αλλαγές και προσαρμογές είναι πολύ εύκολο να γίνει, αλλά πρέπει να αποφεύγεται. Η επανάληψη τμημάτων κώδικα δείχνει ότι δεν χρησιμοποιούνται σωστά μηχανισμοί αφαίρεσης όπως μέθοδοι/συναρτήσεις, υποκλάσεις, και γενικευμένοι τύποι. Ως αποτέλεσμα ο κώδικας έχει πολύ περισσότερες γραμμές από αυτό που θα'πρεπε και σίγουρα έχει περισσότερα λάθη. Η επανάληψη παραλλαγμένων τμημάτων κώδικα δυσχεραίνει την διόρθωση και συντήρησή του αφού σας αναγκάζει να ελέγξετε διεξοδικά κάθε ένα τμήμα κώδικα.

Ένα άλλο σημείο που προκαλεί προβλήματα είναι οι κρυμμένες προϋποθέσεις ή αποδοχές όπως συγκεκριμένο μέγεθος ενός πίνακα ή κάποιοι «μαγικοί αριθμοί», δηλ. αριθμοί που εμφανίζονται ξαφνικά μέσα στον κώδικα και η φυσική τους έννοια είναι πολύ δύσκολο να βρεθεί. Συνήθως η χρήση τους δεν μπορεί να αποφευχθεί. Για παράδειγμα για να οριστεί ότι το ΦΠΑ είναι 0.19 ή ότι το πλήθος των πακέτων του TCP είναι 6. Τέτοιοι αριθμοί θα πρέπει να δηλώνονται ως καθολικές σταθερές (public static final) και φυσικά αυτοί οι αριθμοί σε καμία περίπτωση δεν πρέπει να χρησιμοποιούνται άμεσα στον κώδικα.

4. Αντικειμενοστραφής Προγραμματισμός

Γενικά, στον αντικειμενοστραφή προγραμματισμό δίνεται έμφαση στον κύκλο ζωής του λογισμικού. Η ουσία της αντικειμενοστραφούς σχεδίασης είναι ότι το όλο σύστημα αποτελείται από αντικείμενα, τα οποία είναι τα βασικά δομικά συστατικά της αντικειμενοστραφούς προσέγγισης.

Τα αντικείμενα συγκεντρώνουν τα δεδομένα (χαρακτηριστικά/ιδιότητες – attributes) και τις λειτουργίες (μέθοδοι) που εφαρμόζονται σε αυτά τα δεδομένα. Έτσι τα μετρίσιμα στοιχεία ενός αντικειμένου είναι το πλήθος των ιδιοτήτων/χαρακτηριστικών (attributes), το πλήθος των μεθόδων που καλούνται από άλλα αντικείμενα, το πλήθος των μεθόδων εκτός του καλούμενου αντικειμένου και η θέση του αντικειμένου στην ιεραρχία των κλάσεων.

Μοναδικά στοιχεία της αντικειμενοστραφούς σχεδίασης και προγραμματισμού όπως πέρασμα μηνυμάτων, κληρονομικότητα και πολυμορφισμός, εισάγουν μεγαλύτερη πολυπλοκότητα και απαιτούν ιδιαίτερους τρόπους μέτρησης για τον χειρισμό τους.

Για τις ανάγκες του μαθήματος θα χρησιμοποιείται μόνο ένα μικρό υποσύνολο των στοιχείων αξιολόγησης αντικειμενοστραφούς λογισμικού. Κατά την αξιολόγηση του αντικειμενοστραφούς στοιχείου ενός προγράμματος θα πρέπει να ελέγχονται:

- Η συνολική δομή του συστήματος με το σωστό πλήθος κλάσεων
- Σωστή χρήση της κληρονομικότητας
- Στις κλάσεις να υπάρχουν όλες οι απαραίτητες μέθοδοι αλλά να μην υπάρχουν μέθοδοι οι οποίοι δεν χρειάζονται ή είναι εξαιρετικά πολύπλοκοι να χρησιμοποιηθούν.
- Σωστή χρήση προσδιοριστικών πρόσβασης τόσο στις μεταβλητές (κατά κανόνα “private”) όσο και στις μεθόδους (κατά κανόνα “public”).
- Abstraction: Τα βασικά χαρακτηριστικά μιας κατηγορίας αντικειμένων τα οποία το διαχωρίζουν από αντικείμενα άλλου τύπου και έτσι παρέχουν ξεκάθαρα εννοιολογικά όρια μεταξύ των αντικειμένων. Συνήθως αυτά θα πρέπει να ορίζονται σε ξεχωριστή κλάση η οποία συνήθως θα λειτουργεί ως διασύνδεση μεταξύ των υποκλάσεών της και δεν θα δημιουργούνται αντικείμενα αυτής της κλάσης. Η έλλειψη δημιουργίας αντικειμένων μιας κλάσης μας παροτρύνει να την δηλώσουμε και ως “abstract” ή ως διασύνδεση (interface).

Η υπερβολική χρήση μηχανισμών όπως η κληρονομικότητα και ο πολυμορφισμός (δεν αναφέρομαι σε μηχανισμούς που δεν συμπεριλαμβάνονται στην ύλη του μαθήματος) οδηγούν σε πολύπλοκη σχεδίαση η οποία πρέπει να αποφεύγεται. Υπάρχει διαφορά ανάμεσα σε μια πολύπλοκη σχεδίαση η οποία είναι απαραίτητη και σε μια πολύπλοκη σχεδίαση που στόχο έχει να αντιμετωπίσει πιθανές μελλοντικές απαιτήσεις (η οποία συνήθως οδηγεί σε σπατάλη της παραγωγικότητας των προγραμματιστών και πρέπει να αποφεύγεται) ή στην επίδειξη γνώσεων η οποία τελικά δείχνει μόνο ανωριμότητα.

5. Αλγοριθμική

Θα πρέπει να κρίνεται αν η υλοποίηση του αλγόριθμου ακολουθεί σωστά τα βήματα υλοποίησής του.

Πολλές φορές θα πρέπει να κρίνεται αν χρησιμοποιείται η σωστή αλγοριθμική δομή όπως η σωστή δομή επανάληψης (for, while, do … while) ή η κατάλληλη δομής επιλογής (if, if … else, switch).

Θα πρέπει να ελέγχεται αν ο αλγόριθμος αντιμετωπίζει οριακές καταστάσεις όπως και αν τερματίζεται σωστά.

6. Πρόταση Βαθμολόγησης Εργασιών Εαρινού 2006-07

Επειδή οι δύο πρώτες εργασίες έχουν διαφορετικούς στόχους από τις υπόλοιπες η πρόταση για τις δύο πρώτες εργασία διαφέρει από τις υπόλοιπες.

Για τις δύο πρώτες εργασίες όπου το βάρος δίνεται κυρίως στην αλγοριθμική και στον δομημένο προγραμματισμό (κυρίως όσον αφορά το μενού) προτείνονται τα παρακάτω ποσοστά:

Στυλ κωδικοποίησης (σχόλια κλπ.): 20%

ΟΟ και δομημένος προγραμματισμός: 10%

Ολοκληρωμένη εκτέλεση – ορθότητα - Αλγόριθμοι: 50%

Μενού κλπ της main (δομημένος προγραμματισμός/αλγοριθμική): 20%

Για τις υπόλοιπες εργασίες προτείνεται:

Ολοκληρωμένη εκτέλεση - ορθότητα: 20%

Στυλ κωδικοποίησης (σχόλια κλπ.): 20%

Αντικειμενοστραφής προγραμματισμός: 40%

Δομημένος προγραμματισμός: 10%

Αλγόριθμοι: 10%