Τρίτη Σειρά ασκήσεων
Ημερομηνία Παράδοσης: Δευτέρα 12/5 3:00 μ.μ.

Η τρίτη σειρά ασκήσεων περιέχει δύο τύπους ασκήσεων. Τις βασικές ασκήσεις και τις bonus ασκήσεις. Οι βασικές ασκήσεις πρέπει να παραδοθούν μέχρι τις 2/6 πριν τις 3:00 μ.μ. Για τις bonus ασκήσεις η προθεσμία είναι στις 17/6 πριν τις 6:00 μ.μ. Θα χρησιμοποιήσετε διαφορετικό turnin για τις διαφορετικές ασκήσεις.
Στην υλοποίηση των κλάσεων σας δεν θα πρέπει να έχετε public μεταβλητές. Βαθμοί θα αφαιρεθούν για προγράμματα που δεν είναι καλά γραμμένα, δηλαδή δεν είναι σωστά στοιχισμένα ή δεν έχουν καλά επιλεγμένα ονόματα μεταβλητών ώστε να διαβάζονται εύκολα.

Άσκηση 1

Για την άσκηση αυτή θα φτιάξετε ένα πρόγραμμα που υλοποιεί μια προσομοίωση της εξέλιξης δύο ειδών ζώων σε διαφορετικά περιβάλλοντα. Η επιβίωση ενός ζώου εξαρτάται από δύο πράγματα: Από την εξυπνάδα του (intelligence) και από την ταχύτητα του (speed). Έχουμε ζώα που είναι έξυπνα, και ζώα που είναι γρήγορα. Η συνολική ικανότητα (ability) του ζώου να επιβιώνει υπολογίζεται ως 2×speed+intelligence για τα γρήγορα ζώα και 2×intelligence+speed για τα έξυπνα ζώα.

Ο χώρος στον οποίο κινούνται τα ζώα είναι ένα πλέγμα (grid) από κελιά (cells). Ένα κελί χαρακτηρίζεται από δύο συνθήκες: αν είναι άγονη έκταση ή όχι, και αν είναι βαλτώδες περιβάλλον ή όχι. Η αξία της ταχύτητας μειώνεται σε βαλτώδες περιβάλλον, και η αξία της εξυπνάδας μειώνεται σε άγονο χώρο που δεν μπορείς να κρυφτείς. Το πλέγμα χωρίζεται σε δύο τμήματα: κάτω από τη διαγώνιο είναι ένας άγονος χώρος, ενώ πάνω από την διαγώνιο είναι βάλτος. Η διαγώνιος είναι στεγνό έδαφος με βλάστηση.

Η προσομοίωση προχωρά σε βήματα όπου σε κάθε βήμα συμβαίνουν τα εξής. Σε κάθε κελί υπάρχει ένας αριθμός από ζώα. Τα ζώα συναντιούνται ανά δύο και επιβιώνει αυτό με τη μεγαλύτερη ικανότητα. Τα ζώα που επιβίωσαν αναπαράγονται. Το κάθε ζώο παράγει ένα απόγονο με περίπου την ίδια ταχύτητα και εξυπνάδα. Στη συνέχεια μετακινούνται τυχαία σε ένα γειτονικό κελί και η διαδικασία συνεχίζεται. Σε κάθε βήμα θα τυπώνετε την (ακέραια) μέση τιμή της εξυπνάδας και της ταχύτητας των ζώων που είναι στο κελί. Μετά από μερικά βήματα προσομοίωσης βλέπουμε ότι έχουμε μεγαλύτερη ταχύτητα κάτω από τη διαγώνιο και μεγαλύτερη εξυπνάδα πάνω από την διαγώνιο.

Παρακάτω σας δίνονται μερικές οδηγίες και υποδείξεις για την υλοποίηση, οι οποίες θα ξεκαθαρίσουν και κάποιες λεπτομέρειες της προσομοίωσης. Υπάρχουν τρείς βασικές κλάσεις: η κλάση Animal, η κλάση Cell και η κλάση Simulation, τις οποίες περιγράφουμε παρακάτω.

Η κλάση Animal και οι παράγωγες της:

Δημιουργήστε μια **αφηρημένη** κλάση **Animal** η οποία κρατάει πληροφορία για ένα ζώο. Οι πληροφορίες που χρειαζόμαστε είναι η ταχύτητα και η εξυπνάδα του ζώου και το κελί στο οποίο βρίσκεται. Η κλάση έχει μια αφηρημένη μέθοδο computeAbility η οποία υπολογίζει την ικανότητα επιβίωσης του ζώου ανάλογα με τον τύπο του. Έχει επίσης μια αφηρημένη μέθοδο clone η οποία παράγει ένα νέο ζώο με την ίδια ταχύτητα και εξυπνάδα. Έχουμε δύο ενυπόστατες (concrete) κλάσεις **FastAnimal** και **SmartAnimal** που κληρονομούν από την Animal και υλοποιούν ένα γρήγορο και ένα έξυπνο ζώο αντίστοιχα.

Σχετικά με τις μεθόδους της Animal:

- Ο constructor αρχικοποιεί την ταχύτητα και την εξυπνάδα του ζώου σε μια τυχαία τιμή στο διάστημα [0,50). Η τιμή της ταχύτητας διπλασιάζεται στον constructor της FastAnimal ενώ η τιμή της εξυπνάδας διπλασιάζεται στον constructor της SmartAnimal.
- Ορίστε μια μέθοδο survive η οποία παίρνει σαν όρισμα ένα Animal other και επιστρέφει true αν η ικανότητα του ζώου που καλεί την μέθοδο είναι μεγαλύτερη από αυτή του other, ή false αλλιώς.

- Ορίστε μία μέθοδο reproduce η οποία παράγει ένα ζώου ίδιου τύπου και το τοποθετεί στο ίδιο κελί με το παρόν ζώο. Η ταχύτητα του νέου ζώου υπολογίζεται προσθέτοντας ένα τυχαίο αριθμό μεταξύ [-5,+5] στην ταχύτητα του παρόντος ζώου. Η εξυπνάδα του νέου ζώου υπολογίζεται προσθέτοντας ένα τυχαίο αριθμό μεταξύ [-5,+5] στην εξυπνάδα του παρόντος ζώου.
- Ορίστε μια μέθοδο move η οποία μετακινεί το ζώο σε ένα τυχαίο κελί από τα γειτονικά κελιά του πλέγματος.
- Ορίστε accessor και mutator μεθόδους όπως χρειάζονται, καθώς και όποια άλλη μέθοδο κρίνετε απαραίτητη.

Η κλάση Cell και οι παράγωγες της:
Δημιουργήστε την κλάση **Cell** η οποία κρατάει πληροφορία για ένα κελί του πλέγματος. Η βασική πληροφορία που χρειάζεται θα κρατάει η Cell είναι μια λίστα με τα ζώα που είναι στο κελί και μία λίστα με τους γείτονες του κελιού. (Σημείωση: θα θεωρηθεί λάθος αν κάνετε πολλαπλές λίστες για διαφορετικούς τύπους ζώων/κελιών). Ένα αντικείμενο τύπου Cell αντιστοιχεί σε ένα κελί που έχει βλάστηση και δεν είναι βάλτος. Για να ορίσουμε ένα άγονο και ένα βαλτώδες κελί ορίζουμε τις κλάσεις **BarrenCell** και **SwampCell** αντίστοιχα οι οποίες κληρονομούν από την Cell.

Σχετικά με τις μεθόδους της Cell:
- Είπαμε ότι σε βαλτώδεις περιοχές η ταχύτητα παύει να είναι σημαντική και το ίδιο και για την εξυπνάδα σε άγονες περιοχές χωρίς βλάστηση. Για να ενημερώσουμε την ταχύτητα και την εξυπνάδα ανάλογα με τον τύπο του κελιού θα ορίσετε τις μεθόδους updateSpeed και updateIntelligence η οποίες παίρνουν σαν όρισμα την τιμή της ταχύτητας και της εξυπνάδας αντίστοιχα και επιστρέφουν την ενημερωμένη τιμή. Αν έχουμε ένα απλό κελί τότε η τιμή δεν αλλάζει. Αν το κελί είναι SwampCell τότε η ταχύτητα μειώνεται στο μισό, ενώ για το κελί BarrenCell η εξυπνάδα μειώνεται στο μισό. Ορίστε και υπερβείτε κατάλληλα τις μεθόδους updateSpeed και updateIntelligence για τις κλάσεις Cell, SwampCell και BarrenCell.
- Ορίστε την μέθοδο survival η οποία πραγματοποιεί τις συναντήσεις ανά δύο των ζώων (αν είναι περιττός ο αριθμός των ζώων ένα ζώο δεν συναντιέται με κανένα). Στο τέλος η λίστα με τα ζώα πρέπει να κρατάει μόνο αυτά που επιβίωσαν.
- Ορίστε μεθόδους addAnimal και removeAnimal που προσθέτουν και να αφαιρούν ένα ζώο στο κελί αντίστοιχα. Μπορείτε κατά την πρόσθεση να ενημερώνετε και το κελί του ζώου.
- Ορίσετε μεθόδους που επιστρέφουν την μέση τιμή σε ταχύτητα και εξυπνάδα.
- Ορίστε μια μέθοδο getRandomNeighbor που επιστρέφει ένα τυχαίο γείτονα.
- Ορίστε accessor και mutator μεθόδους όπως και όποτε τις χρειάζεστε, καθώς και όποια άλλη μέθοδο θεωρείτε απαραίτητη.

Υποδείξεις:
- Για απλότητα κάνετε τις διαιρέσεις να είναι ακέραιες.
- Δεν μπορείτε να προσθέσετε ή να αφαιρέσετε στοιχεία από ένα ArrayList ενώ το διατρέχετε.

Η κλάση Simulation:
Δημιουργήστε την κλάση **Simulation** η οποία θα υλοποιεί την προσομοίωση. Η κλάση πρέπει να κρατάει πληροφορία για το πλέγμα και τα ζώα που είναι στο πλέγμα. Ο constructor θα παίρνει σαν ορίσματα το μέγεθος του πλέγματος και τον αριθμό των ζώων που θα προστεθούν. Αν ο αριθμός των ζώων είναι N, προσθέτουμε 2N ζώα, ένα γρήγορο για κάθε έξυπνο. Τα ζώα αρχικά τοποθετούνται σε τυχαίες θέσεις.

Η κλάση θα πρέπει να έχει μεθόδους ώστε να κάνει τα εξής:
- Δημιουργεί το πλέγμα (grid) με τα κελιά (και τους γείτονες τους) όπως τα περιγράψαμε στην αρχή (άγονο έδαφος κάτω από την διαγώνιο, βαλτώδες πάνω από την διαγώνιο, κανονικό στην διαγώνιο). Μπορείτε να υλοποιήσετε το πλέγμα σαν ένα δισδιάστατο πίνακα, αλλά οποιαδήποτε άλλη υλοποίηση είναι αποδεκτή.
- Δημιουργεί τα ζώα και τα τοποθετεί τυχαία στο πλέγμα.
- Γίνονται όλες τις συναντήσεις μεταξύ των ζώων σε όλα τα κελιά.
- Τα ζώα που έχουν επιβιώσει αναπαράγονται.

- Γίνονται όλες οι μετακινήσεις.
- Τυπώνεται η κατάσταση του πλέγματος (η μέση τιμή της ταχύτητας και εξυπνάδας σε κάθε κελί).

Υπόδειξη: Για τις κάποιες από τις παραπάνω λειτουργίες είναι πιο βολικό να διατρέχετε μια λίστα από όλα τα ζώα και να καλείτε κάποια μέθοδο για το καθένα. Στην περίπτωση αυτή θα πρέπει να δημιουργήσετε μια μέθοδο που κατασκευάζει αυτή τη λίστα από το πλέγμα.

Η κλάση RunSimulation:
Δημιουργήστε την κλάση **RunSimulation** η οποία θα έχει την μέθοδο main και θα υλοποιεί τις επαναλήψεις της προσομοίωσης. Προτεινόμενες τιμές για την προσομοίωση: 5 για το μέγεθος του πλέγματος, 100 για τον αριθμό των ζώων, 20-50 για τον αριθμό των επαναλήψεων.

Σημείωση: Μπορείτε να παρεκκλίνετε αν θέλετε από κάποιες από τις παραπάνω οδηγίες και υποδείξεις, αλλά πρέπει να ορίσετε την Animal ως αφηρημένη κλάση, με τις αφηρημένες μεθόδους που ζητούνται, καθώς και τις παράγωγες concrete κλάσεις FastAnimal και SmartAnimal. Επίσης πρέπει να ορίσετε την Cell και τις BarrenCell και SwampCell ως παράγωγες.

Άσκηση 2

Ο στόχος αυτής της άσκησης είναι να εξασκηθείτε με την δομή HashMap της Java. Δεδομένης μια συλλογής από κείμενα, μια δημοφιλής μετρική για να μετράμε πόσο σημαντική είναι μια λέξη μέσα σε ένα κείμενο είναι να υπολογίζουμε το tf-idf score της λέξης μέσα στο κείμενο. Για μία λέξη w το tf-idf score για ένα κείμενο d υπολογίζεται ως TF(w,d)*IDF(w). TF(w,d) είναι ο αριθμός των εμφανίσεων της λέξης w μέσα στο κείμενο d. IDF(w) είναι μία τιμή που έχει να κάνει με το σε πόσα κείμενα εμφανίζεται η λέξη w. Μία λέξη που εμφανίζεται σε πολλά διαφορετικά κείμενα είναι λιγότερο χαρακτηριστική για το κείμενο d και συνεπώς λιγότερο σημαντική. Αν DF(w) είναι ο αριθμός των κειμένων στα οποία εμφανίζεται η λέξη w, και N είναι ο συνολικός αριθμός των κειμένων τότε IDF(w) = log(N/DF(w)). Αν μια λέξη εμφανίζεται σε όλα τα κείμενα (π.χ. ένα άρθρο) τότε έχει IDF(w) = 0, και συνεπώς tf-idf score μηδέν.

Στη σελίδα του μαθήματος υπάρχει ένα αρχείο reviews.txt το οποίο περιέχει 15 (μεγάλες) γραμμές. Η κάθε γραμμή είναι ένα κείμενο, συγκεκριμένα ένα online review για ένα προϊόν. Τα reviews 1-3 είναι για laptops, τα 4-6 για φωτογραφικές μηχανές, τα 7-9 για κινητά τηλέφωνα, τα 10-12 για mp3 players, και τα 13-15 για τηλεοράσεις. Ο στόχος της άσκησης είναι να υπολογίζουμε το tf-idf score για τις λέξεις σε αυτά τα κείμενα για την συλλογή αυτών των 15 κειμένων.

Δημιουργήστε μια κλάση **Document** που κρατάει πληροφορίες για ένα κείμενο και κάνει τη βασική επεξεργασία του. Αρχικοποιείται με ένα String που είναι το περιεχόμενο του κειμένου. Θα έχει τις εξής μεθόδους:

- **process**: Σπάει το κείμενο σε λέξεις, αφαιρεί όλα τα σημεία στίξεως και μετατρέπει τα κεφαλαία σε μικρά (lowercase). Χρησιμοποιείστε μεθόδους της κλάσης String για να κάνετε αυτή την επεξεργασία. Επίσης υπολογίζει το TF(w,d) για κάθε λέξη μέσα στο κείμενο (χρησιμοποιώντας ένα HashMap). Η μέθοδος πρέπει να επίσης να αυξήσει το DF(w) για κάθε λέξη στο κείμενο, οπότε παίρνει σαν όρισμα ένα HashMap που κρατάει το DF(w) για κάθε λέξη μέχρι τη στιγμή της επεξεργασίας αυτού του κειμένου.
- **computeTFIDF**: Υπολογίζει το tf-idf score για κάθε λέξη και το αποθηκεύει σε ένα νέο HashMap. Παίρνει σαν όρισμα ένα HashMap που κρατάει το DF(w).
- **toString**: Επιστρέφει ένα String με όλες τις λέξεις του κειμένου και το tf-idf score τους. Το String θα έχει ζευγάρια της μορφής word:tf-idf για κάθε λέξη, χωρισμένα με κενά.

Υλοποιήστε την κλάση **Collection** με την οποία χειριζόμαστε μία συλλογή από κείμενα. Ο constructor παίρνει σαν όρισμα το όνομα του αρχείου εισόδου και το όνομα του αρχείου εξόδου. Η κλάση θα κρατάει ένα ArrayList με τα κείμενα και ένα HashMap με τα document frequencies για κάθε λέξη. Έχει μια βασική μέθοδο **process** η οποία διαβάζει ένα-ένα τα κείμενα από το αρχείο, και δημιουργεί ένα αντικείμενο Document για το καθένα. Κάνει την επεξεργασία και υπολογίζει τα tf-idf scores για κάθε κείμενο. Στη συνέχει γράφει το tf-idf String για κάθε αρχείο στο αρχείο εξόδου.

Δημιουργήστε μια κλάση **TextProcessing** η οποία έχει την main. Παίρνει σαν ορίσματα εκτέλεσης (command-line arguments) το όνομα του αρχείου εισόδου και του αρχείου εξόδου. Δημιουργεί ένα αντικείμενο Collection που κάνει την επεξεργασία.

Bonus Άσκηση 1

Στις προηγούμενες σειρές ασκήσεων δώσατε δύο διαφορετικές υλοποιήσεις για τον Αφηρημένο Τύπο Δεδομένων ταξινομημένος πίνακας/λίστα. Σε αυτή την άσκηση θα τροποποιήσετε την υλοποίηση σας ώστε να ορίσετε μία γενικευμένη κλάση η οποία παίρνει σαν παράμετρο τον τύπο των δεδομένων T που αποθηκεύει ο πίνακας/λίστα σας. Ο τύπος T θα πρέπει να υλοποιεί το interface Comparable<T> ώστε να σας επιτρέπει να κάνετε σύγκριση μεταξύ στοιχείων του τύπου T.

Κατασκευάσετε μια κλάση **SortedCollection** που υλοποιεί την ταξινομημένη συλλογή σας. Εκτός από τον constructor, η κλάση θα πρέπει να έχει τις μεθόδους **insert**, **delete**, και **get**.

Bonus Άσκηση 2

Για την άσκηση αυτή θα υλοποιήσετε ένα πρόγραμμα που διαχειρίζεται την λειτουργία μιας εταιρίας λογισμικού. Η εταιρία έχει εργαζόμενους και για κάθε εργαζόμενο έχουμε το όνομα του και το ΑΦΜ του. Οι εργαζόμενοι χωρίζονται σε managers και engineers. Ο κάθε manager έχει υπό την επίβλεψη του κάποιους εργαζόμενους (μπορεί να είναι και managers και engineers). Όλοι οι εργαζόμενοι έχουν ένα manager (εκτός από τον manager στην κορυφή της ιεραρχίας).

Η δουλειά στην εταιρία οργανώνεται σε projects. Κάθε project έχει ένα κωδικό όνομα και μία αξία που είναι μια τιμή μεταξύ 1 και 10, και απασχολεί ένα αριθμό από engineers. Ένας engineer μπορεί να συμμετέχει σε πάνω από ένα projects. Όταν το project ολοκληρωθεί οι engineers μοιράζονται την αξία του project ως bonus points. Ο manager παίρνει το άθροισμα των bonus points αυτών που επιβλέπει. Η ταξινόμηση όλων των υπαλλήλων γίνεται με βάσει τα bonus points.

Υλοποιήστε ένα πρόγραμμα που προσομοιώνει την λειτουργία μιας εταιρίας που έχει 3 managers και 8 engineers Ο ένας manager επιβλέπει τους άλλους δύο, και οι άλλοι έχουν από 4 engineers ο καθένας. Η εταιρία έχει 5 projects που το καθένα απασχολεί από 1 έως 4 engineers. Τον αριθμό των engineers και την αξία των projects θα τα αποφασίσετε εσείς. Από αυτά τα 4 ολοκληρώνονται. Υπολογίστε τα bonus points του κάθε εργαζόμενου και τυπώστε ταξινομημένη τη λίστα των εργαζομένων και τα bonus points τους.

Για την άσκηση αυτή εκτός από τον κώδικα σας θα παραδώσετε και ένα αρχείο κειμένου στο οποίο θα περιγράφετε πως σχεδιάσατε το πρόγραμμα σας: τι κλάσεις ορίσατε, τι πεδία και μεθόδους έχουν και πως συνδέονται μεταξύ τους.