ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ ΠΛΥ212/ΜΥΥ205
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ & ΠΛΗΡΟΦΟΡΙΚΗΣ Τεχνικές Αντικειμενοστρεφούς Προγραμματισμού

Πρώτη Σειρά ασκήσεων
Ημερομηνία Παράδοσης: 12 Απριλίου 2016, 3 μ.μ.

Οι παρακάτω ασκήσεις πρέπει να παραδοθούν μέχρι τις 12/4/2016, 3 μ.μ. . Στην υλοποίηση των κλάσεων σας δεν θα πρέπει να έχετε public πεδία. 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 υλοποιήσετε σε Java τον Αφηρημένο Τύπο Δεδομένων **Ουρά Προτεραιότητας (PriorityQueue)**. Η ουρά αποθηκεύει ακεραίους και μας δίνει εύκολη πρόσβαση στο μεγαλύτερο στοιχείο. Η ουρά που θα υλοποιήσετε θα έχει σταθερή χωρητικότητα (**capacity**), που είναι ο μέγιστος αριθμός στοιχείων που μπορεί να αποθηκεύσει. Για την υλοποίηση σας θα χρησιμοποιήσετε ένα **πίνακα** (όχι ArrayList). Το μέγιστο στοιχείο θα αποθηκεύεται στην πρώτη θέση του πίνακα. Μπορείτε να ορίσετε και όποιο άλλο πεδίο χρειάζεστε.

Κατασκευάσετε την κλάση **PriorityQueue** που υλοποιεί μια ουρά που κρατάει θετικούς ακεραίους. Η κλάση θα πρέπει να έχει τις εξής μεθόδους:

- Ένα **constructor** χωρίς ορίσματα που αρχικοποιεί την ουρά με χωρητικότητα capacity = 100.
- Ένα **constructor** που αρχικοποιεί την ουρά με όρισμα την χωρητικότητα του πίνακα.
- Μία μέθοδο **insert** η οποία παίρνει σαν όρισμα ένα ακέραιο και τον προσθέτει στην ουρά. Επιστρέφει μία boolean τιμή αν έγινε σωστά η προσθήκη του στοιχείου. Η προσθήκη θα πρέπει να γίνεται σε σταθερό χρόνο, δηλαδή ανεξάρτητο από το πόσα στοιχεία υπάρχουν στην ουρά.
- Μία μέθοδο **getMax** η οποία επιστρέφει την τιμή του μεγαλύτερου ακέραιου από την ουρά. Αν η ουρά είναι άδεια επιστρέφει -1.
- Μία μέθοδο **extractMax** η οποία αφαιρεί και επιστρέφει τον μεγαλύτερο ακέραιο από την ουρά. Αν η ουρά είναι άδεια επιστρέφει -1.
- Μία μέθοδο **merge** η οποία παίρνει σαν όρισμα μία ουρά προτεραιότητας και την συνενώνει με την ουρά που καλεί την μέθοδο. Το αποτέλεσμα αποθηκεύεται στην ουρά που καλεί τη μέθοδο. Η ουρά που περνάμε σαν όρισμα στο τέλος θα είναι άδεια. Επιστρέφει μια boolean τιμή ανάλογα με το αν ήταν δυνατή η συνένωση ή όχι.
- Την μέθοδο **toString** η οποία επιστρέφει ένα αλφαριθμητικό με τα στοιχεία της ουράς με σειρά που είναι τοποθετημένα στον πίνακα, χωρισμένα με κενό.
- Την μέθοδο **equals** η οποία ελέγχει αν η ουρά είναι ίδια με κάποια άλλη. Ισότητα σημαίνει ότι οι δύο ουρές περιέχουν ακριβώς τα ίδια στοιχεία.

Ορίστε μια μέθοδο main για να τεστάρετε την κλάση σας. Για να σας βαθμολογήσουμε θα φτιάξουμε μια κλάση **PriorityQueueTest** που θα χρησιμοποιεί την PriorityQueue που θα παραδώσετε. Ένα παράδειγμα δίνεται στην σελίδα του μαθήματος.

Bonus: Στην σελίδα του μαθήματος υπάρχει μια κλάση FindTopK η οποία στην main καλεί μια μέθοδο printTopK η οποία παίρνει σαν όρισμα ένα πίνακα και μια τιμή K και τυπώνει τα K μικρότερα στοιχεία του πίνακα. Υλοποιήστε την μέθοδο printTopK χρησιμοποιώντας (υποχρεωτικά) την κλάση PriorityQueue. Παραδώστε το αρχείο FindTopK.java.

Σημείωση: Στο μάθημα των Δομών θα μάθετε μια πολύ καλύτερη υλοποίηση για την Ουρά Προτεραιότητας η οποία υποστηρίζει τις ίδιες λειτουργίες αλλά πιο γρήγορα.

Άσκηση 2

Για την άσκηση αυτή θα υλοποιήσετε σε Java ένα πρόγραμμα που θα υλοποιεί το παιχνίδι Σκορ 4 (Connect 4). Το παιχνίδι παίζεται με δύο παίκτες. Οι παίκτες έχουν μάρκες διαφορετικού χρώματος. Υπάρχει ένα πλέγμα (συνήθως 7 × 6), στο οποίο οι παίκτες μπορούν να ρίξουν τις μάρκες από σχισμές στην κορυφή του πλέγματος. Η μάρκα πέφτει στην αντίστοιχη στήλη, στο πρώτο ελεύθερο κελί. Ο πρώτος παίκτης που θα δημιουργήσει μια σειρά με 4 συνεχόμενες μάρκες του ίδιου χρώματος κερδίζει. Η σειρά μπορεί να είναι είτε οριζόντια, είτε κάθετα, είτε διαγώνια.

Για την υλοποίηση σας θα πρέπει να δημιουργήσετε τρεις κλάσεις. Την κλάση **Grid** η οποία κρατάει πληροφορίες για το πλέγμα. Την κλάση **Player** που υλοποιεί το παιχνίδι ενός παίκτη. Την κλάση **Connect4** που υλοποιεί την ροή του παιχνιδιού.

Grid: Η πιο σημαντική κλάση που κρατάει πληροφορίες για την κατάσταση του πλέγματος. Το πλέγμα με τις μάρκες θα το υλοποιήσετε ως ένα δισδιάστατο πίνακα width × height (με default τιμές 7 και 6 αντίστοιχα) από ακεραίους. Τα δύο διαφορετικά χρώματα θα αντιστοιχούν στους αριθμούς 1 και 2, ενώ η τιμή μηδέν σημαίνει ότι το κελί είναι άδειο. Επίσης θα κρατάει ένα μονοδιάστατο πίνακα μήκους width οπού για κάθε θέση, κρατάει το παρόν ύψος που έχουν οι μάρκες σε εκείνη την θέση.

Η κλάση θα πρέπει να έχει τις εξής μεθόδους:

- Εκτός από τον **default constructor** χωρίς ορίσματα που θα αρχικοποιεί στις default τιμές, ορίστε και ένα **constructor** ο οποίος παίρνει σαν όρισμα το πλάτος και ύψος του πλέγματος.
- Την μέθοδο **printGrid** η οποία θα εκτυπώνει το πλέγμα. Προσέξτε ότι οι γραμμές του πλέγματος θα πρέπει να τυπώνονται από πάνω προς τα κάτω.
- Μια μέθοδο **isValidMove** η οποία παίρνει σαν όρισμα μια θέση και επιστρέφει μια Boolean μεταβλητή αν μπορούμε να ρίξουμε μάρκα σε αυτή τη θέση. Αν βρείτε τρόπο τυπώστε τις διαφορετικές μάρκες με διαφορετικό χρώμα.
- Μία μέθοδο **addPiece** η οποία παίρνει σαν όρισμα μια θέση και το χρώμα της μάρκας και την προσθέτει στο πλέγμα.
- Μια μέθοδο **isWinningPiece** η οποία θα παίρνει σαν όρισμα μία θέση και επιστρέφει μια boolean τιμή αν η μάρκα που προστέθηκε σε εκείνη την θέση δημιούργησε μια τετράδα. Αυτή είναι η πιο δύσκολη μέθοδος, παρακάτω υπάρχουν κάποιες υποδείξεις για την υλοποίηση της.
- Μία μέθοδο **checkPosition** η οποία παίρνει σαν όρισμα μια θέση και ένα χρώμα μάρκας και επιστρέφει μια Boolean τιμή, αν η προσθήκη της μάρκας σε εκείνη την θέση θα έχει ως αποτέλεσμα την δημιουργία τετράδας, χωρίς όμως να προσθέσει την μάρκα στο πλέγμα.
- Μία μέθοδο **isFull** η οποία επιστρέφει true αν όλες το πλέγμα έχει γεμίσει.

Μπορείτε να ορίσετε και άλλες public ή private κλάσεις εφόσον το θεωρείτε απαραίτητο.

Η υλοποίηση της **isWinningPiece** είναι το πιο δύσκολο κομμάτι της υλοποίησης. Για τον έλεγχο θα πρέπει να ελέγξετε αν δημιουργείται τετράδα σε τέσσερεις διαφορετικές κατευθύνσεις: κάθετα, οριζόντια, διαγωνίως αριστερά προς δεξιά και διαγωνίως δεξιά προς αριστερά. Μια πρόταση είναι να κάνετε μια μέθοδο για καθένα από αυτούς τους ελέγχους. Ο κάθετος έλεγχος είναι εύκολος. Για τον οριζόντιο έλεγχο θεωρείστε ένα παράθυρο μεγέθους 4 το οποίο κινείται οριζόντια και περιέχει την θέση που μας ενδιαφέρει. Αυτό που θέλετε να ελέγξετε είναι αν για κάποια θέση του παραθύρου όλες οι θέσεις του πλέγματος μέσα στο παράθυρο έχουν το ίδιο χρώμα όπως η μάρκα. Παρόμοιος είναι ο έλεγχος και για τις διαγωνίους, αλλά στην περίπτωση αυτή το παράθυρο κινείται όχι μόνο οριζόντια αλλά ταυτόχρονα και κάθετα.

Player: Η κλάση αυτή θα υλοποιεί το παιχνίδι ενός παίκτη. Έχει δύο βασικά πεδία: το χρώμα του παίκτη (int), και μία μεταβλητή String που καθορίζει αν ο παίκτης ελέγχεται από άνθρωπο ή υπολογιστή (το default είναι άνθρωπος).

Θα έχει τις εξής μεθόδους:

- Δύο **constructor**: έναν που παίρνει μόνο το χρώμα, και έναν που παίρνει το χρώμα και τον τύπο του παίκτη.
- Την private μέθοδο **humanPlay** η οποία υλοποιεί το παιχνίδι του παίκτη-ανθρώπου. Η μέθοδος θα παίρνει σαν όρισμα το πλέγμα του παιχνιδιού. Θα ζητάει από τον παίκτη να δώσει την θέση στην οποία θα ρίξει την μάρκα μέχρι να πάρει μια αποδεκτή επιλογή, και θα προσθέτει την μάρκα στο πλέγμα.
- Την private μέθοδο **computerPlay** η οποία υλοποιεί το παιχνίδι του παίκτη-υπολογιστή. Η μέθοδος θα παίρνει σαν όρισμα το πλέγμα του παιχνιδιού. Ο υπολογιστής θα επιλέγει την θέση στην οποία θα ρίξει την μάρκα ως εξής: (α) Πρώτα ελέγχει αν υπάρχει μια θέση με την οποία μπορεί να κερδίσει το παιχνίδι, και αν ναι την επιλέγει. (β) Αν δεν βρει μια τέτοια θέση ελέγχει αν υπάρχει μία θέση με την οποία ο αντίπαλος κερδίζει το παιχνίδι και αν ναι την επιλέγει. (γ) Αν δεν ισχύει τίποτα από τα παραπάνω διαλέγει μία από τις διαθέσιμες θέσεις τυχαία.
- Την μέθοδο **play** η οποία παίρνει σαν όρισμα το πλέγμα του παιχνιδιού και ανάλογα με τον τύπο του παίκτη καλεί την **humanPlay** ή την **computerPlay**.
- Μία **μέθοδο προσπέλασης (accessor method)** για το χρώμα του παίκτη.

Connect4: Η κλάση που περιέχει την μέθοδο **main**, η οποία υλοποιεί τη βασική ροή του παιχνιδιού. Δημιουργεί το πλέγμα και τους παίκτες. Ο πρώτος παίκτης θα ελέγχεται από τον χρήστη και θα τον ρωτάει αν θα θέλει να παίξει με άνθρωπο ή υπολογιστή και δημιουργεί τον δεύτερο παίκτη ανάλογα. Οι παίκτες παίζουν εναλλάξ. Μετά την κίνηση του κάθε παίχτη γίνεται έλεγχος αν κέρδισε και αν ναι τυπώνεται ένα μήνυμα και σταματάει το παιχνίδι. Το παιχνίδι συνεχίζεται μέχρι είτε κάποιος παίκτης να κερδίσει, είτε να γεμίσει το πλέγμα.

Υποδείξη:

- Η αρχικοποίηση και η ενημέρωση της εντολής for μπορεί να περιλαμβάνει πολλαπλές μεταβλητές. Π.χ., ο παρακάτω κώδικας διατρέχει την διαγώνιο του πίνακα array.

```
for (int i = 0,j = 0; i < n && j<n; i ++, j ++){
      System.out.println(array[i][j]+ " ");
}
```