Εργαστηριακή άσκηση #11

Θέμα: Εξοικείωση με την χρήση μεταβλητών τύπου αναφοράς για δημιουργία συνδεδεμένων δομών. Εξοικείωση με τη χρήση κλάσεων (σε ρόλο «κλάσης βιβλιοθήκης») ως συστατικών μεγαλύτερων εφαρμογών.

Η εργαστηριακή αυτή άσκηση αποσκοπεί στην εξοικείωση με τη χρήση των μεταβλητών τύπου αναφοράς για την δημιουργία και τον χειρισμό μιας συνδεδεμένης δομής αποθήκευσης δεδομένων, καθώς και την χρήση των δομών αυτών ως συστατικών μεγαλύτερων (και πιο σύνθετων εφαρμογών). Θα δημιουργήσουμε το έργο **PriorityQueues** το οποίο περιέχει τις κλάσεις **PQItem, PQNode, DLULL_PQ, RandomArrayGenerator** και **SortingMaching**. Οι τρεις πρώτες κλάσεις, υποστηρίζουν τη δομή δεδομένων που είναι γνωστή ως ουρά προτεραιότητας (PriorityQueue). Σύμφωνα με το σενάριο που θα υλοποιήσουμε στο εργαστήριο, θα δημιουργήσουμε ένα τυχαίο μονοδιάστατο διάνυσμα (με χρήση της κλάσης **RandomArrayGenerator)** το οποίο στη συνεχεία θα ταξινομήσουμε με τη βοήθεια της κλάσης **SortingMaching.** Η υλοποίηση της κλάσης **SortingMaching** χρησιμοποιεί ως κύριο μηχανισμό ταξινόμησης την ουρά προτεραιότητας **DLULL_PQ.**

Η κλάση **DLULL_PQ** υλοποιεί μια διπλά συνδεδεμένη μη διατεταγμένη λίστα (**D**oubly-**L**inked **U**nsorted **L**inked **L**ist) από **PQItem**s. Κάθε αντικείμενο τύπου **PQItem** αποθηκεύει 2 στοιχεία: τα δεδομένα του (τύπου Object) και την προτεραιότητα (priority) του. Η «θέση του» μέσα στην ουρά προτεραιότητας εξαρτάται από την προτεραιότητα του.

Λειτουργίες που παρέχονται από την ουρά προτεραιότητας περιλαμβάνουν την προσθήκη **PQItem**s (insert), τη διαγραφή του στοιχείου με την «υψηλότερη» προτεραιότητα (deleteMin), την πρόσβαση (χωρίς ταυτόχρονη διαγραφή) στο στοιχείο με την «υψηλότερη» προτεραιότητα (findMin), και τον έλεγχο του εάν η ουρά είναι άδεια (isEmpty).

Αντικείμενα της κλάσης **PQNode** αποτελούν τα δομικά υλικά για την κατασκευή της λίστας **DLULL_PQ.** Κάθε αντικείμενο τύπου **PQNode** περιέχει τρία πεδία: ένα για να αποθηκεύσει τα δεδομένα (τυπου **PQItem**) , ένα για να υποδείξει το επόμενο στοιχείο (τύπου **PQNode**) στη λίστα και ενα για να υποδείξει το προηγούμενο στοιχείο (επίσης τυπου **PQNode**) .

1. Δημιουργήστε ένα νέο έργο με την ονομασία **PriorityQueues**. Για οικονομία χρόνου, στην εργαστηριακή αυτή άσκηση δεν θα εισάγουμε σχόλια στον κώδικα που δημιουργούμε.

2. Δημιουργήστε την κλάση **PQItem**. Η κλάση **PQItem** αποτελείται από τα πεδία

```
private int priority;
private  Object data;
```

και παρέχει τις παρακάτω μεθόδους:

- **public PQItem(int pr, Object d)** — Κατασκευαστής. Ως παράμετροι δίνονται η προτεραιότητα του αντικειμένου που θα δημιουργηθεί και τα δεδομένα που αποθηκεύει.
- public void setPriority(int newPr)
- public int getPriority()
- public Object getData()

3. Δημιουργήστε την κλάση **PQNode**. Η κλάση **PQNode** αποτελείται από τα πεδία

```
private PQNode previous;
private PQNode next;
private PQItem nodeData;
```

και παρέχει τις παρακάτω μεθόδους:

- **public PQNode(PQItem d)** — Κατασκευαστής.
- public PQItem getNodeData()
- public void setPrevious(PQNode p)
- public void setNext(PQNode n)

- public PQNode getPrevious()
- public PQNode getNext()

4. Υλοποιήστε την κλάση **PQNode** και βεβαιωθείτε ότι καταλαβαίνετε τη λειτουργία της. Ιδιαίτερη προσοχή να δοθεί στο ρόλο των πεδίων next και previous και στη χρήση τους για τη δημιουργία διπλά συνδεδεμένων δομών δεδομένων. Μεταφράστε την κλάση, και δημιουργείστε τουλάχιστον 3 αντικείμενα. Κατόπιν κάνοντας χρήση της μεθόδου **setNext()** και **setPreviou()** συνδέστε τα σε μία λίστα.

5. Δημιουργήστε την κλάση **DLULL_PQ**. Η κλάση **DLULL_PQ** αποτελείται από το πεδίο

   private PQNode head; //αναφορά προς το πρώτο στοιχείο της διπλά συνδεδεμένης λίστας που υλοποιεί την ουρά προτεραιότητας

6. Η κλάση **DLULL_PQ** παρέχει τις παρακάτω μεθόδους:

▪ **DLULL_PQ()**	Κατασκευάζει μία άδεια ουρά προτεραιότητας.
▪ **boolean isEmpty()**	Ελέγχει το εάν η ουρά προτεραιότητας περιέχει στοιχεία
▪ **void insert(PQItem item)**	Εισάγει ένα νέο στοιχείο στην ουρά προτεραιότητας
▪ **PQItem deleteMin()**	Διαγράφει το στοιχείο από την ουρά προτεραιότητας με την μικρότερη «αριθμητικά» προτεραιότητα. Το στοιχείο αυτό θεωρείται ως το στοιχείο με την μεγαλύτερη προτεραιότητα.
▪ **public PQItem findMin()**	Όμοια λειτουργία με την **deleteMin()**, χωρίς όμως να διαγράφει το στοιχείο. Απλώς μας δίνει πρόσβαση σε αυτό έτσι ώστε να μπορούμε να δούμε την προτεραιότητα και τα δεδομένα που περιέχει.

7. Υλοποιήστε τις μεθόδους της κλάσης **DLULL_PQ.** Να δώσετε ιδιαίτερη προσοχή στις οριακές περιπτώσεις, δηλ, όταν εισάγουμε σε άδεια λίστα, όταν διαγράφουμε το πρώτο ή το τελευταίο στοιχείο της, και όταν διαγράφουμε το μοναδικό στοιχείο της ουράς. Μετά από την υλοποίηση κάθε μεθόδου να πειραματίζεστε δημιουργώντας μια νέα ουρά προτεραιότητας και εκτελώντας τη μέθοδο. Να ελέγχετε πάντα τις οριακές περιπτώσεις.

8. Να υλοποιηθεί η κλάση RandomIntArrayGenerator η οποία περιέχει την μέθοδο

   ```
   public static int[] createRandomIntArray(int len)
   ```

   η οποία δημιουργεί και επιστρέφει ένα τυχαίο διάνυσμα από **len** ακεραίους.

9. Να υλοποιηθεί η κλάση SortingMachine η οποία περιέχει την μέθοδο

   ```
   public void integerSort(int data[])
   ```

   η οποία ταξινομεί το διάνυσμα των ακεραίων που της δίνεται ως παράμετρος. Η ταξινόμηση γίνεται με χρήση μίας ουράς προτεραιότητας. Πρώτα εισάγονται όλα τα στοιχεία του διανύσματος στην ουρά και κατόπιν εξάγονται και τοποθετούνται στη σωστή θέση τους μέσω της μεθόδου **deleteMin()**.

10. Η διαγραφή του στοιχείου με την ελάχιστη «αριθμητικά» προτεραιότητα απαιτεί την διαπέραση όλης της λίστας για την εύρεση του, και κατά συνέπεια, ο χρόνος εκτέλεσης της μεθόδου **deleteMin()** είναι ανάλογος με τον αριθμό των στοιχείων που περιέχει η ουρά προτεραιότητας. Το ίδιο ισχύει και για μια απλοϊκή υλοποίηση της μεθόδου **findMin()** όπου είναι σχεδόν ταυτόσημη με την υλοποίηση της μεθόδου **deleteMin()**. Να υλοποιήσετε τη μέθοδο **findMin()** έτσι ώστε σε περίπτωση διαδοχικών κλήσεων της (χωρίς δηλαδή να παρεμβάλλονται κλήσης της μεθόδου **deleteMin()**) μόνο ο χρόνος εκτέλεσης της πρώτης κλήσης της να είναι ανάλογος με τον αριθμό των στοιχείων που περιέχει η ουρά προτεραιότητας. Ο χρόνος εκτέλεσης της κλήσεων που ακολουθούν να είναι σταθερός, ανεξάρτητος δηλαδή από τον αριθμό των στοιχείων που περιέχει η ουρά προτεραιότητας.

11. Να εμπλουτιστεί ο κώδικας κλάσεων του έργου **PriorityQueues** με σχόλια.