Γενικά:
Τους κώδικες μου, τους εκτέλεσα στο InteliJ

Ερώτημα Α:
Για να μπορέσω να εκτελέσω την διεπαφή του πρώτου ερωτήματος χρειάστηκε να κάνω αρχικά “implement” στην “StringDoubleEndedQueueImpl” την “StringDoubleEndedQueue” και στην συνέχεια με την χρήση του “@override” να υλοποιήσω τις μεθόδους μου.
Από εκεί και πέρα, προκειμένου να δημιουργήσω την Doubly linked list μου, δημιούργησα τη κλάση Node, στην οποιά αρχικοποιώ τους κόμβους next και previous, καθώς και τη συμβολοσειρά item, που είναι το όρισμα που δέχεται κάθε φορά η κλαση.
‘Οσον αφορά τώρα την “StringDoubleEndedQueue” ξεκίνησα δηλώνοντας τους ιδιωτικούς κόμβους header και lastNode, οι οποίοι αποτελούν το πρώτο και τον τελευταίο κόμβο της λίστας αντίστοιχα .Επίσης δηλώνω την ακέραια μεταβλητή size, η οποία είναι πολύτιμη στο να μπορώ να σημειώνω πόσα στοιχεία έχει κάθε φορά η ουρά μου. Στην συνέχεια έχω τον constructor για την αρχικοποίηση των κόμβων. Έπειτα ξεκινάει η υλοποίηση των μεθόδων.
Η “isEmpty” είναι ένα αντικείμενο, το οποίο επιστρέφει μία λογική τιμή, η οποία προκύπτει από το αν ο πρώτος κόμβος (header) περιέχει μέσα του την τιμή null ή όχι.
Στη συνέχεια υπάρχουν οι “addFirst(item)” && “addLast(item)” , οι οποίες δεν επιστρέφουν τίποτα, διότι είναι void αλλά είναι πολύτιμες για την εισαγωγή νέων στοιχείων στην αρχή και το τέλος της ουράς αντίστοιχα. Βασική προϋπόθεση για να δημιουργήσω έναν νέο κόμβο είναι να καλέσω τον constructor της Node, δίνοντας σε αυτόν το όρισμα που θέλουμε να προσθέσει στην αρχή ή στο τέλος της ουράς. Από εκεί και πέρα, ελέγχουμε το μέγεθος της ουράς μέσα από την “isEmpty” , ώστε να κάνουμε τις ανάλογες ενέργειες. Οι ενέργειες αυτές έχουν να κάνουν με το περιεχόμενο των κόμβων next και previous. Ο next πρέπει να περιέχει την τιμή του επόμενου κόμβου και ο previous την τιμή του προηγούμενου κόμβου. Αν δεν υπάρχει προηγούμενος και επόμενος κόμβος, τότε οι κόμβοι αυτοί περιέχουν την τιμή null. Τέλος αυξάνουμε το size κατά 1, αφού πλέον έχουμε προσθέσει ένα νέο String στην ουρά μας.
Ύστερα έχουμε την “removeLast()” η οποία αφαιρεί το τελευταίο στοιχείο της ουράς, όταν της το ζητήσει ο χρήστης. Πάλι χρειαζόμαστε την “isEmpty” για να ελέγξουμε αν υπάρχει άλλο στοιχείο στην ουρά, διότι αν δεν υπάρχει και επιχειρήσουμε να αφαιρέσουμε κάποιο στοιχείο θα γίνει error. Ωστόσο για αυτόν το λόγο χρησιμοποιούμε την NoSuchElementException ώστε να εντοπίζει την περίπτωση αυτή και να εμφανίζει το ανάλογο μήνυμα.
Επίσης η "removeLast()" δεν αφαιρεί μόνο το τελευταίο στοιχείο αλλά και το επιστρέφει κιόλας. Όλοι οι μέθοδοι εισαγωγής και αφαίρεσης στοιχείων είναι πολυπλοκότητας O(1), διότι δεν υπάρχει κανένα loop n επαναλήψεων και γενικά ο compiler μεταγλωττίζει το πρόγραμμα αμέσως χωρίς να χρειάζεται να ελέγξει κάτι, παραπάνω απο μία φορά.
Στη συνέχεια έχουμε τις getFirst() και getLast(), που επιστρέφουν το πρώτο και το τελεύταιο στοιχείο της ουράς, εφόσον αυτή δεν είναι άδεια.

Τέλος, η size επιστρέφει το μέγεθος της ούρας και η printQueue(PrintStream stream) είναι πολύ σημαντική για την εκτύπωση των στοιχείων της ουράς.
Όλες οι μέθοδοι έχουν πολυπλοκότητα O(1), αφου δεν περιέχουν καμία εντολή επανάληψης και όλες οι εντολές εκτελούνται μια φορά.

Ερώτημα Β:
Το βασικό ζητούμενο, κατά τη γνώμη μου, για να μπορέσει κάποιος να γράψει κάποιος τον κώδικα μετατροπής μιας παράστασης από μεταθεματική σε ενθεματική μορφή είναι να έχει καταλάβει τον συγκεκριμένο αλγόριθμο. Η δική μου οπτική, μου δείχνει ότι μόλις δεχθούμε την παράσταση από το χρήστη, ελέγχουμε ένα-ένα τα γράμματα της συμβολοσειράς(για αυτή το λόγο χρησιμοποίησα το .charAt()) και πράττουμε ως εξής:
1. Αν δεχθούμε αριθμό τον κάνουμε addLast() στην ουρά
2. Αν δεχθούμε τελεστή τότε αποθηκεύουμε, με τη χρήση της getLast()τα δύο προηγούμενα στοιχεία της ουράς σε δύο προσωρινές μεταβλητές και με τη χρήση της removeLast() το αφαιρούμε από αυτήν . Στη συνέχεια αποθηκεύουμε την πράξη μέσα σε παρένθεση σε μια νέα μεταβλητή temp την οποία στην συνέχεια κάνουμε addLast(temp) στην ουρά.
Παράδειγμα:
Αν εισάγουμε το: 12* το αποτέλεσμα που θα προκύψει στην οθόνη του υπολογιστή είναι (1*2).Το μειονέκτημα που έχει ο συγκεκριμένος αλγόριθμος είναι ότι παρόλο που βρίσκει το αποτέλεσμα σωστά , εμφανίζει στη λύση περιττές παρενθέσεις που ωστόσο δεν δημιουργούν κάποιο πρόβλημα.
Πιο συγκεκριμένα ξεκινάω το πρόγραμμα μου, διαβάζοντας μια συμβολοσειρά (η οποία θα αποτελείται από αριθμούς και κάποιους συγκεκριμένους τελεστές) και αποθηκεύοντας την, σε μια μεταβλητή τύπου String .Στη συνέχεια δημιουργώ ένα αντικείμενο list και χρησιμοποιώ μια λογική μεταβλητή “ντετέκτιβ” προκειμένου να εντοπίσω αν κάποια στιγμή θα συμβεί κάποιο σφάλμα. Εφόσον όλα κυλήσουν ομαλά και ο χρήστης δώσει από το πληκτρολόγιο μια παράσταση με σωστή μεταθεματική μορφή, η οθόνη θα του εμφανίσει το σωστό αποτέλεσμα. Σε διαφορετική περίπτωση η οθόνη εμφανίζει μήνυμα λάθους.

Ερώτημα Γ:
Το ερώτημα αυτό απαντήθηκε με την ζητούμενη πολυπλοκότητα που ζητούσε η άσκηση(O(n)).Χρησιμοποίησα μόνο 1 loop και μόνο 1 αντικείμενο ουράς. Αρχικά ζητάω από τον χρήστη να πληκτρολογήσει τα στοιχεία που αποτελούν το DNA, δηλαδή ένα εκ των ‘A’, ‘T’,’C’,’G’. Όταν ο χρήστης επιχειρήσει να δώσει κάποιο λανθασμένο string, το πρόγραμμα θα του εμφανίσει το κατάλληλο μήνυμα. Αυτό συμβαίνει μέσα από το if που βρίσκεται στην do…while. Η do….while χρησιμοποιείται προκειμένου να δώσει ο χρήστης τα στοιχεία του DNA, τα οποία αποθηκεύονται μέσα στην ουρά μέσω της μεθόδου addLast(). Η εισαγωγή των στοιχείων τερματίζεται μόλις πληκτρολογήσει ο χρήστης “stop”. Από εκεί και πέρα ελέγχω μέσω της size() το πλήθος των στοιχείων της ουράς, διότι σε περίπτωση που το πλήθος είναι περιττό ή 0, τότε σίγουρα η ουρά δεν είναι της μορφής DNA που επιθυμούμε. Αυτό συμβαίνει διότι μια συμβολοσειρά DNA με περιττό πλήθος στοιχείων θα περιέχει πάντα ένα στοιχείο που δεν θα μπορεί να «ντουμπλάρει» με κάποιο άλλο και συνεπώς σίγουρα δεν θα ισχύει η μορφή Watson-Crick.

Εφόσον, το πλήθος της ουράς αποτελείται από ζυγό αριθμό στοιχείων τότε ακολουθούμε την εξής διαδικασία:

Δεσμεύουμε δύο προσωρινές μεταβλητές τύπου string , οι οποίες περιέχουν το πρώτο και το τελευταίο στοιχείο της ουράς. Αυτό επιτυγχάνεται μέσω των μεθόδων getFirst() && getLast(). Ύστερα ελέγχουμε αν αυτά τα strings είναι συμπληρωματικά μεταξύ τους και εφόσον συμβαίνει αυτό, τα 'διώχνουμε' από την ουρά μέσω των removeFirst() && removeLast(). Έφόσον η διαδικασία αυτή ολοκληρωθεί με απόλυτη επιτυχία μέχρι να αδειάσει η ούρα , το DNA που έχουμε εισάγει στην αρχή είναι της μορφής Watson-Crick complemented palindrome. Στην περίπτωση που εισάγουμε ένα DNA, το οποίο δεν είναι της μορφής που ζητάει η άσκηση, η οθόνη εμφανίζει το κατάλληλο μήνυμα.