Αρχείο CharDoubleEndedQueueImpl.java (μέρος Α):
Στο αρχείο πηγαίου κώδικα Java CharDoubleEndedQueueImpl, η διεπαφή (που χρησιμοποιεί Generics) υλοποιήθηκε ως εξής:
Δημιουργούμε την κλάση CharDoubleEndedQueueImpl<T> η οποία υλοποιεί την διεπαφή CharDoubleEndedQueue<T>.
Αρχικοποιούμε τις ιδιωτικές μεταβλητές της κλάσης:
name, size, ListNode<T> firstElement και ListNode<T> lastElement, που είναι αντίστοιχα το όνομα, το μέγεθος, ο πρώτος και ο τελευταίος κόμβος αντίστοιχα της λίστας (που βρίσκεται ουσιαστικά πίσω από την δομή της ουράς).

Δημιουργούμε τον default constructor CharDoubleEndedQueueImpl(), που δεν παίρνει ορίσματα και καλεί τον δεύτερο constructor, τον CharDoubleEndedQueueImpl(String name) με όρισμα την μεταβλητή με όνομα name τύπου String, ο οποίος δημιουργεί μια άδεια ουρά, ενώ αρχικοποιεί την μεταβλητή size με την τιμή 0.

Ύστερα η μέθοδος isEmpty() επιστρέφει true αν η ουρά είναι κενή ή false σε αντίθετη περίπτωση.

Μετά ακολουθεί η μέθοδος addFirst η οποία δημιουργεί ενα νέο αντικείμενο με όνομα node τύπου ListNode<T> και μετά ελέγχει αν η ουρά είναι κενή. Σε αυτή την περίπτωση (δηλαδή αν όντως η ουρά είναι κενή), αρχικοποιούμε το πρώτο και το τελευταίο στοιχείο της ουράς και τα θέτουμε ίσα με το αντικείμενο node. Διαφορετικά, το θέτουμε το firstElement ως τον επόμενο του νέου κόμβου node, το προηγούμενο node του στοιχείου firstElement γίνεται το νέο στοιχείο node και μετά θέτουμε το firstElement να να γίνει ίσο με το node. Μετά το μέγεθος της ουράς αυξάνεται κατά 1.

Η addLast είναι παρόμοια με την παραπάνω, δηλαδή ελέγχει αν η ουρά είναι κενή. Αν είναι, απλά προσθέτει στην ουρά το στοιχείο όπως παραπάνω. Αν δεν είναι κενή θέτουμε τον προηγούμενο κόμβο του νέου κόμβου ως το τελευταίο στοιχείο, τον επόμενο κόμβο του τελευταίου στοιχείου ως τον νέο κόμβο και τον τελευταίο κόμβο ως τον νέο κόμβο.
Το μέγεθος της ουράς αυξάνεται μετά κατά 1.

Η removeFirst() ελέγχει αν η ουρά είναι κενή. Αν είναι, πετάει ένα Exception (The queue is empty. You can't remove first element.) Άν όχι, αφαιρεί το πρώτο στοιχείο της ουράς και επιστρέφει τα δεδομένα του, αφού πρώτα θέσει το προηγούμενο κόμβο του επόμενου κόμβου του πρώτου στοιχείου ως null, και κάνει το δεύτερο στοιχείο (δηλαδή αυτό που βρίσκεται μετά από αυτό που αφαιρέθηκε) πρώτο.

Η removeLast() ελέγχει αν η ουρά είναι κενή. Αν είναι, πετάει ένα Exception (The queue is empty. You can't remove last element.) Άν όχι, αφαιρεί το τελευταίο στοιχείο απο την ουρά και επιστρέφει τα δεδομένα του, αφού πρώτα θέσει το επόμενο κόμβο του προηγούμενου κόμβου του τελευταίου στοιχείου ως null, και κάνει το προτελευταίο στοιχείο (δηλαδή αυτό που βρίσκεται πριν από αυτό που αφαιρέθηκε) τελευταίο.

Η getFirst() επιστρέφει τα δεδομένα του πρώτου στοιχείου της ουράς, χωρίς όμως να το αφαιρεί από αυτή. Αν η ουρά είναι κενή, πετάει ένα Exception(The queue is empty.)

Η getLast() επιστρέφει τα δεδομένα του τελευταίου στοιχείου της ουράς, χωρίς όμως να το αφαιρεί από αυτή. Αν η ουρά είναι κενή, πετάει ένα Exception(The queue is empty.)

Η size() επιστρέφει το μέγεθος της ουράς.

Η printQueue(PrintStream stream) εκτυπώνει στην οθόνη κάθε στοιχείο της ουράς, όσο αυτή δεν είναι κενή. Αν εξαρχής κενή τότε εκτυπώνει "The queue is empty."

Αρχείο DNAPalindrome.java (μέρος Β):

Για το μέρος Β, χρησιμοποιήσαμε την υλοποίηση από το μέρος Α για να φτιάξουμε το πρόγραμμα που ζητείται, ως εξής:
Δημιουργήσαμε αρχικά την κλάση DNAPalindrome και την main μας μέθοδο.

Ύστερα, δημιουργούμε μια νέα ουρά με δύο άκρα (από το ερώτημα Α) με όνομα queue και αρχικοποιούμε ορισμένες μεταβλητές του προγράμματος,

Α) Την valid, η οποία δείχνει αν το String που δίνεται από το χρήστη περιέχει μόνο τους τέσσερις έγκυρους χαρακτήρες.

Β) Την WCCompPalindrome η οποία δείχνει αν το String που δίνεται από το χρήστη είναι συμπληρωματικά παλίνδρομο κατά Watson - Crick.

C) Την size, που δείχνει το μέγεθος της queue (αρχικοποιείται αργότερα).
Μετά δημιουργούμε ένα νέο αντικείμενο in τύπου Scanner για να διαβάσουμε δεδομένα από το πληκτρολόγιο και έπειτα εμφανίζουμε στην μονάδα εξόδου ένα κατάλληλο prompt message.
Μετά, ο χρήστης πληκτρολογεί το String, και έπειτα ελέγχεται η εγκυρότητα κάθε χαρακτήρα με την βοήθεια της μεθόδου isInvalid.
Αν το String είναι έγκυρο, τότε κάθε χαρακτήρας εισάγεται στην ουρά που δημιουργήσαμε πιο πάνω με την μέθοδο addFirst. Έπειτα, το μέγεθος της ουράς μπαίνει στην μεταβλητή size.
Ύστερα, ελέγχεται αν η το περιεχόμενο της ουράς είναι συμπληρωματικά παλίνδρομο κατά Watson - Crick, ελέγχοντας κάθε φορά αν το πρώτο στοιχείο είναι ίσο με το συμπλήρωμα του τελευταίου στοιχείου (με την βοήθεια της μεθόδου getSupplement), το δεύτερο με το συμπλήρωμα του προτελευταίου κ.α.κ.
Αν μετά τον έλεγχο προκύψει κάποια αναντιστοιχία, τότε η μεταβλητή WCCompPalindrome γίνεται false.
Διαφορετικά, αν πρόκειται για στοιχεία που είναι συμπληρωματικά παλίνδρομα κατά Watson - Crick, τότε εμφανίζεται το μήνυμα: "It is a Watson-Crick complemented palindrome!" και μετά εμφανίζονται το ένα μετά το άλλο τα στοιχεία της ουράς αντεστραμένα.

Η μέθοδος isInvalid(char c) επιστρέφει true αν ο χαρακτήρας που δίνεται ως όρισμα είναι ένας από τους κεφαλαίους αγγλικούς χαρακτήρες A, T, C ή G.

Τέλος, η getSupplement(char c) επιστρέφει το συμπληρωματικό στοιχείο κάθε χαρακτήρα-νουκλεοτιδίου, δηλαδή: 'T' για το 'A', 'A' για το 'T', 'C' για το 'G' και 'G' σε κάθε άλλη περίπτωση (πάντα πρώτα εκτελείται η isInvalid, οπότε εκτελεστεί το else ο χαρακτήρας δεν μπορεί παρά να είναι ο C.

Αρχείο CharQueueImpl .java (μέρος Γ):
Στο αρχείο πηγαίου κώδικα Java CharQueueImpl, η διεπαφή υλοποιήθηκε ως εξής:
Δημιουργούμε την κλάση CharQueueImpl<T> η οποία υλοποιεί την διεπαφή CharQueue<T> (που χρησιμοποιεί Generics).
Αρχικοποιούμε τις ιδιωτικές μεταβλητές της κλάσης:
name, size, ListNode<T> firstElement και ListNode<T> lastElement, που είναι αντίστοιχα το όνομα, το μέγεθος, ο πρώτος και ο τελευταίος κόμβος αντίστοιχα της λίστας (που βρίσκεται ουσιαστικά πίσω από την δομή της ουράς).

Δημιουργούμε τον default constructor CharQueueImpl(), που δεν παίρνει ορίσματα και καλεί τον δεύτερο constructor, τον CharQueueImpl(String name) με όρισμα την μεταβλητή με όνομα name τύπου String, ο οποίος δημιουργεί μια άδεια ουρά, ενώ αρχικοποιεί την μεταβλητή size με την τιμή 0.

Ύστερα η μέθοδος isEmpty() επιστρέφει true αν η ουρά είναι κενή ή false σε αντίθετη περίπτωση.

Η μέθοδος put(T item) αρχικά δημιουργεί ένα νέο αντικείμενο τύπου ListNode<T> με όνομα temp. Αν η ουρά είναι κενή αρχικοποιούμε το πρώτο και το τελευταίο στοιχείο της ουράς και τα θέτουμε ίσα με το αντικείμενο temp. Διαφορετικά, τοποθετούμε το νέο στοιχείο στο node που βρίσκεται δεξιά του στοιχείου που δείχνει η tail και μετά παίρνουμε την μεταβλητή tail και την βάζουμε να δείχνει στο στοιχείο που βρίσκεται δεξιά του πρώτου στοιχείου της ουράς.
Έπειτα αυξάνουμε το μέγεθος της ουράς κατά 1.

Η μέθοδος get() ελέγχει αν η ουρά είναι κενή. Αν είναι, πετάει ένα NoSuchElementException (The queue is empty. You can't remove oldest element.) Αν όχι, αποθηκεύει τα δεδομένα που περιέχει το node στο οποίο δείχνει το head σε μια μεταβλητή με όνομα tempData. Μετά, αν οι μεταβλητές head και tail ταυτίζονται, τότε τις θέτουμε ως null. Αλλιώς, η head δείχνει στο αμέσως επόμενο node από αυτή. Μετά μειώνουμε το μέγεθος της ουράς κατά 1 και επιστρέφουμε το tempData.

Η peek() ελέγχει αν η ουρά είναι κενή. Αν είναι, πετάει ένα NoSuchElementException (The queue is empty.) Αν όχι, επιστρέφει τα δεδομένα που περιέχει το στοιχείο της ουράς που δείχνει η head.

Όσον αφορά την size(), αυτή απλά επιστρέφει το μέγεθος της ουράς.

Τέλος έχουμε την printQueue(PrintStream stream) η οποία αρχικά ελέγχει αν η δεν ουρά είναι κενή. Αν δεν είναι κενή, τότε αποθηκεύει την head στην currentnode (η οποία είναι τύπου ListNode<T>), εμφανίζει "Print the queue:", και τυπώνει διαδοχικά σε διαφορετική σειρά τα περιεχόμενα της ουράς. Αν η ουρά είναι κενή, εμφανίζει "The queue is empty.".

Αρχείο CharQueueWithMin.java (μέρος Δ):

Για την υλοποίηση του Μέρους Δ χρησιμοποιήσαμε την υλοποίηση της ουράς FIFO από το Μέρος Γ (εφεξής θα αναφέρεται ως F) και την ουρά με διπλά άκρα από το Μέρος Α (εφεξής θα αναφέρεται ως D) ως εξής: Αρχικά, δημιουργούμε μια κλάση CharQueueWithMin<Character> η οποία επεκτείνει την CharQueueImpl<Character>, την οποία αναπτύξαμε στο Μέρος Γ. Αρχικοποιούμε τις ιδιωτικές μεταβλητές της κλάσης:
name, size, ListNode<Character> tail και ListNode<Character> head, που είναι αντίστοιχα το όνομα, το μέγεθος, ο τελευταίος και ο πρώτος κόμβος αντίστοιχα της λίστας (που βρίσκεται ουσιαστικά πίσω από την δομή της ουράς F).

Δημιουργούμε τον default constructor CharQueueWithMin(), που δεν παίρνει ορίσματα και καλεί τον δεύτερο constructor, τον CharQueueWithMin(String name) με όρισμα την μεταβλητή με όνομα name τύπου String, ο οποίος δημιουργεί μια άδεια ουρά F.

Ύστερα η μέθοδος isEmpty() επιστρέφει true αν η ουρά F είναι κενή ή false σε αντίθετη περίπτωση.

Στη συνέχεια ακολουθεί η put(Character item) μέθοδος, η οποία τοποθετεί ένα item στην αρχή της ουράς F, χρησιμοποιώντας την αντίστοιχη υλοποίηση της υπερκλάσης με την εντολή super.put(item). Επίσης τοποθετούμε το στοιχείο και στην ουρά D, εκτός αν το πρώτο στοιχείο της D είναι μεγαλύτερο από το νέο στοιχείο.

Μετά ακολουθεί η get() μέθοδος η οποία αφαιρεί ένα στοιχείο από το πίσω μέρος της ουράς F, χρησιμοποιώντας την αντίστοιχη υλοποίηση της υπερκλάσης με την εντολή super.get(). Αφαιρείται επίσης και το τελευταίο στοιχείο της ουράς D. Έπειτα επιστρέφει τα δεδομένα αυτού του στοιχείου της ουράς D. Αν δεν υπάρχει στοιχείο στην ουρά F η μέθοδος πετάει NoSuchElementException.

Η μέθοδος peek() απλά επιστρέφει τα δεδομένα του πρώτου στοιχείου της ουράς χωρίς να το διαγράφει, χρησιμοποιώντας την αντίστοιχη υλοποίηση της υπερκλάσης με την εντολή super.peek().

Η μέθοδος size() απλά επιστρέφει το μέγεθος της ουράς.

Η μέθοδος min() επιστρέφει το πρώτο στοιχείο της ουράς D, το οποίο λόγω της παραπάνω υλοποίησης θα είναι και το μικρότερο στοιχείο της ουράς.

Τέλος, η μέθοδος printQueue(PrintStream stream), εκτυπώνει την ουρά χρησιμοποιόντας την αντίστοιχη υλοποίηση της υπερκλάσης με την εντολή super.printQueue(stream).