Ουρά με δύο άκρα με χρήση Κυκλικού πίνακα

1. Interface DeQueue:

Ξεκινήσαμε με την δημιουργία του Interface φτιάχνοντας για αρχή ένα Interface το οποίο το ονομάσαμε DeQueue<E> όπου E οποιοδήποτε Element. Στην συνέχεια αντιγράψαμε τις μεθόδους που μας είχαν δοθεί.

```java
public interface DeQueue<E> {
    /**
     * Push a new element at the front of the queue
     *
     * @param elem the element
     */
    void pushFirst(E elem);
    /**
     * Push a new element at the end of the queue
     *
     * @param elem the element
     */
    void pushLast(E elem);
    /**
     * Pop an element from the front of the queue
     */
    E popFirst();
    /**
     * Pop an element from the end of the queue
     */
    E popLast();
    /**
     * Return the first element of the queue
     *
     * @return the first element of the queue
     */
    E first();
    /**
     * Return the last element of the queue
     *
     * @return the last element of the queue
     */
    E last();
    /**
     * Check if a queue is empty
     *
     * @return true if empty, false otherwise
     */
    boolean isEmpty();

    /**
     * Get the size of the queue
     *
     * @return the size of the queue
     */
    int size();
    /**
     * Clear the queue
     */
    void clear();
    /**
     * Returns an iterator over the elements of the queue
     */
    Iterator<E> iterator();
    /**
     * Returns an iterator over the elements of the queue in reverse
     * sequential order.
     */
    Iterator<E> descendingIterator();
```

2. Class DeQueueImpl:

Στην συνέχεια φτιάξαμε μία κλάση η οποία θα έκανε implement το interface που φτιάξαμε

```
public class DequeueImpl<E> implements DeQueue<E> {
```

Και επίσης φτιάξαμε 6 μεταβλητές απαραίτητες για την υλοποίηση αυτής της δομής δεδομένων.

```
private Node<E>[] array;
private int f;
private int r;
private int size;
private volatile int modCount = 0;
private static final int DEFAULT_CAPACITY = 8;
```

Όπου το array είναι τύπου Node<E> που χρησιμοποιείται για την αποθήκευση των δεδομένων :

```
private static class Node<E> { //create a class node for the array
    public E data;

    public Node(E data) { //for the push function this is needed to pass a value
        this.data = data;
    }
}
```

Το f και r είναι οι δύο δείκτες μέσα στον πίνακα front και rear αντίστοιχα.
Το size δείχνει κατά πόσο έχει γεμίσει ο πίνακας.
Το modCount χρησιμοποιείται στον Iterator για προστασία από παράλληλες αλλαγές.
Το DEFAULT_CAPACITY είναι το αρχικό μέγεθος του πίνακα.

<u>Δημιουργία Constructor:</u>

```
public DequeueImpl(){
    this.array = (Node<E>[]) new Node[DEFAULT_CAPACITY];
    this.f = 0;
    this.r = -1;
    this.size = 0;
}
```

<u>Υλοποίηση συναρτήσεων:</u>

isEmpty():

```java
@Override
public boolean isEmpty() {
    return size == 0; //if size = 0 then its empty
}
```

Επιστρέφει false or true αναλόγως για αν το size είναι 0.

Size():

```java
@Override
public int size() { //returns the size
    return size;
}
```

Επιστρέφει την μεταβλητή size.

Clear():

```java
@Override
public void clear() { //takes the values of the constructor
    array = (Node<E>[]) new Node[DEFAULT_CAPACITY];
    this.f = 0;
    this.r = -1;
    this.size = 0;
}
```

Αρχικοποίησή τα πάντα σαν τον constructor:

Last():

```java
@Override
public E last() {
    if (isEmpty()) {
        throw new NoSuchElementException();
    }
    return array[(r - 1 + array.length) % array.length].data;   //returns the last data, usung modulo math because pointer is always infront of value
}
```

Επιστρέφει την πληροφορία του πίνακα 1 θέση πριν τον δείκτη rear, διότι δείχνει πάντα 1 μπροστά. Χρησιμοποιήσαμε modulo επειδή αν υπάρχει περίπτωση να χρειαστεί να πάει σε edge του πίνακα.

First():

```java
@Override
public E first() {
    if(isEmpty()){
        throw new NoSuchElementException();
    }
    return array[f].data; //returns the first data
}
```

Επιστρέφει την πληροφορία του πίνακα στον δείκτη front.

popLast():

```java
@Override
public E popLast() {
    if (isEmpty()) { //cant pop if empty
        throw new NoSuchElementException();
    }
    if(size <= array.length/4 && array.length != 4){//cant shrink lower than 4
        shrink();
    }
    E data = array[(r - 1 + array.length) % array.length].data; //saves data to be popped
    array[(r - 1 + array.length) % array.length] = null; //removes the value
    r = (r-1 + array.length) % array.length; //using modulo math rear goes to its previous position
    size--;

    return data;
}
```

Πριν διαγράψει πληροφορία, ελέγχει αν ο πίνακας μπορεί να μικρύνει και καλεί την συνάρτηση shrink(). Μετά αποθηκεύει την πληροφορία που πάει να διαγραφτεί στο data για να το επιστρέψει στο τέλος. Τέλος διαγράφει την πληροφορία και το rear πάει μία θέση πίσω και το size μειώνεται κατά 1.

popFirst():

```java
@Override
public E popFirst() {
    if (isEmpty()) { //cant pop if empty
        throw new NoSuchElementException();
    }
    if(size <= array.length/4 && array.length != 4){ //cant shrink lower than 4
        shrink();
    }
    E data = array[f].data; //saves data to be popped
    array[f] = null; //removes the value
    f = (f + 1) % array.length; //using modulo math front goes to its previous position
    size--; //size gets reduces

    return data; //returns data
}
```

Πριν διαγράψει πληροφορία, ελέγχει αν ο πίνακας μπορεί να μικρύνει και καλεί την συνάρτηση shrink(). Μετά αποθηκεύει την πληροφορία που πάει να διαγραφτεί στο data για να το επιστρέψει στο τέλος. Τέλος διαγράφει την πληροφορία και το front πάει μία θέση μπροστά και το size μειώνεται κατά 1.

pushLast(E elem):

```
@Override
public void pushLast(E elem) {
        if (isFull()) { //checks if its full if it is then resize
           doubleSize();
        }
        Node<E> newNode = new Node<>(elem); //element to be pushed gets inserted in new node

          if (isEmpty()) { //if array is empty then the front index gets the value
             array[f] = newNode;
             r = f + 1; //rear is enabled and goes to where front is + 1
             size++;//incriment size
          }
          else { //if its not empty
                array[r] = newNode;//new data is inserted
                size++;//incriment size
                if (isFull()) { //checks if its full if it is then resize
                    doubleSize();
                }
                r = (r + 1) % array.length; //using modulo math rear goes one forward if not on edge, if on edge then goes to start
            }

}
```

Πριν προσθέσει πληροφορία ελέγχει αν ο πίνακας έχει γεμίσει, και καλεί την συνάρτηση doubleSize(). Στην συνέχεια φτιάχνετε ένα προσωρινό newNode που αποθηκεύεται η πληροφορία που δόθηκε με την κλήση της συνάρτησης. Αν είναι άδειος ο πίνακας τότε η πληροφορία μπαίνει στην θέση f δηλαδή την 0 το rear γίνεται f+1 κ το size αυξάνεται κατά 1. Αν δεν είναι άδειος τότε η πληροφορία μπαίνει στην θέση r αυξάνεται το size και μετά αυξάνεται και το r μία θέση μπροστά και αν είναι στο τέλος του πίνακα τότε πάει στην αρχή.

pushFirst():

```
@Override
public void pushFirst(E elem) {
     if (isFull()) { //checks if its full if it is then resize
        doubleSize();
     }
     Node<E> newNode = new Node<>(elem); //element to be pushed gets inserted in new node

      if (isEmpty()) {//if array is empty then the front index gets the value
        array[f] = newNode;
        r = f + 1; //rear is enabled and goes to where front is
      }
      else { //if its not empty
        f = (f - 1 + array.length) % array.length; //using modulo math front goes one back if not on edge, if on edge then goes to end
        array[f] = newNode; //new data is inserted
     }
     size++; //incriment size
}
```

Πριν προσθέσει πληροφορία ελέγχει αν ο πίνακας έχει γεμίσει, και καλεί την συνάρτηση doubleSize(). Στην συνέχεια φτιάχνετε ένα προσωρινό newNode που αποθηκεύεται η πληροφορία που δόθηκε με την κλήση της συνάρτησης. Αν είναι άδειος ο πίνακας τότε η πληροφορία μπαίνει στην θέση f δηλαδή την 0 το rear γίνεται f+1 κ το size αυξάνεται κατά 1. Αν δεν είναι άδειος τότε πηγαίνει το f μία θέση πίσω και αν είναι στην αρχή του πίνακα τότε πάει στο τέλος. Η πληροφορία μπαίνει στην θέση f και αυξάνεται το size.

Iterator():

```
@Override
public Iterator<E> iterator() {
    int expectedModCount = modCount; //record the modCount when creating the iterator
    return new Iterator<E>() {

        private int currentIndex = f;
        private int visited = 0;

        @Override
        public boolean hasNext() {
            checkForConcurrentModification(); //check for concurrent modifications
            return visited < size;
        }

        @Override
        public E next() {
            checkForConcurrentModification(); //check for concurrent modifications
            if (!hasNext()) {
                throw new NoSuchElementException();
            }

            E data = array[currentIndex % array.length].data;
            currentIndex = (currentIndex + 1) % array.length;
            visited++;
            return data;
        }

        private void checkForConcurrentModification() { //function that checks concurent modifications
            if (expectedModCount != modCount) {
                throw new ConcurrentModificationException("Concurrent modification detected");
            }
        }
    };
}
```

Επιστρέφει έναν νέο Iterator<E> όπου έχει δύο μεταβλητές currentIndex και visited. Ο current Index παίρνει την τιμή f επειδή θέλουμε να ξεκινήσουμε να μετράμε από το first. Η visited δείχνει πόσα δεδομένα έχει προσπελάσει. Στην hasNext() ελέγχει αν υπάρχει ταυτόχρονη χρήση iterator και επιστρέφει true αν ο visited < size. Στην next() ελέγχει αν έχει υπάρχει ταυτόχρονη χρήση iterator και αν υπάρχει άλλο στοιχείο και επιστρέφει το επόμενο στοιχείο και αλλάζει το currentIndex και το visited αυξάνεται κατά 1.

descendingIterator():

```java
@Override
public Iterator<E> descendingIterator() {
    int expectedModCount = modCount; //record the modCount when creating the iterator
    return new Iterator<E>() {
        private int currentIndex = r - 1;
        private int visited = 0;

        @Override
        public boolean hasNext() {
            checkForConcurrentModification(); //check for concurrent modifications
            return visited < size;
        }

        @Override
        public E next() {
            checkForConcurrentModification(); //check for concurrent modifications
            if (!hasNext()) {
                throw new NoSuchElementException();
            }

            E data = null;
            if (currentIndex != -1) {
                data = array[currentIndex % array.length].data;
            } else {
                data = array[array.length - 1].data;
            }

            currentIndex = (currentIndex - 1 + array.length) % array.length;
            visited++;
            return data;
        }

        private void checkForConcurrentModification() { //function that checks concurent modifications
            if (expectedModCount != modCount) {
                throw new ConcurrentModificationException("Concurrent modification detected");
            }
        }
    };
}
```

Επιστρέφει έναν νέο Iterator<E> όπου έχει δύο μεταβλητές currentIndex και visited. Ο current Index παίρνει την τιμή r-1 επειδή θέλουμε να ξεκινήσουμε να μετράμε από το last. Η visited δείχνει πόσα δεδομένα έχει προσπελάσει. Στην hasNext() ελέγχει αν υπάρχει ταυτόχρονη χρήση iterator και επιστρέφει true αν ο visited < size. Στην next() ελέγχει αν έχει υπάρχει ταυτόχρονη χρήση iterator και αν υπάρχει άλλο στοιχείο και επιστρέφει το επόμενο στοιχείο και αλλάζει το currentIndex προς τα πίσω και το visited αυξάνεται κατά 1. Επίσης κάνει έλεγχο για το αν το rear είναι 0 δηλαδή το currentIndex είναι -1.

Άλλες συναρτήσεις:

Length():

```java
public int Length(){ //returns the array length just to show the functionality of shrink and doubleSize (used only in main)
    return array.length;
}
```

Χρησιμοποιείται μόνο στην main για να μας δείχνει την σωστή λειτουργία των shrink() and doubleSize()

isFull():

```
private boolean isFull(){
    return size == array.length; // if size is the same as the array size then its full
}
```

doubleSize():

```
private void doubleSize(){
   int newCapacity = array.length * 2; //new size is double the original size
   int oldCapacity = array.length; //keeps the old size for later meazures
    Node<E>[] newArray = (Node<E>[]) new Node[newCapacity]; //new array

    for(int i = f; i < array.length; i++){
        newArray[newArray.length - array.length + i] = array[i]; //everything from the end of the original array goes to the end of the new array
    }
     for(int i = r; i >= 0; i--){ //eveything before the rear goes to the begining of the new array
        newArray[i] = array[i];
    }


    array = newArray; //old  array takes all the data from the new array
    f = array.length - oldCapacity + f; //front changes
    //rear stays as it is
}
```

Δημιουργείται ένας πίνακας διπλάσιος από τον προηγούμενο όπου όλα τα στοιχεία μετά του f του αρχικού πίνακα πάνε στο τέλος του νέου πχ. (Αν ο παλιός πίνακας μεγέθους 4 έιχε το f στο 2 τότε το αντίστοιχο στον νέο πίνακα θα ήταν 2*4 – 4 + 2 = 6, άρα όλες οι πληροφορίες του παλιού f θα ξεκινάνε από το 6).
Όλα τα στοιχεία πριν το r πάνε στην αρχή του νέου πίνακα και τέλος ο παλιός πίνακας παίρνει την τιμή του νέου και αλλάζει το f.

Shrink():

```
private void shrink(){
    int newCapacity = array.length / 2;
    int oldCapacity = array.length;
    Node<E>[] newArray = (Node<E>[]) new Node[newCapacity];

    if(r < newCapacity){ //checks for potential errors

    if(f != 0){ //checks for potential errors
    for(int i = f; i < array.length; i++){
       newArray[Math.abs(i -( oldCapacity - newCapacity))] = array[i];
    }

    }
     for(int i = r; i >= 0; i--){ //eveything before the rear goes to the begining of the new array
        newArray[i] = array[i];
    }

     array = newArray; //old  array takes all the data from the new array
     f = Math.abs(f - ( oldCapacity - newCapacity));
    }
}
```

Δημιουργείται ένας πίνακας μικρότερος κατά 2 από τον αρχικό και η λογική είναι παρόμοια με το doubleSize() μόνο που αλλάζουν τα μαθηματικά.

3. Tests:

Μετά την υλοποίηση του Dequeue φτιάξαμε 7 test.

testFirstQueue():

```
@Test
  public void testFirstQueue() {
      DequeueImpl<Integer> q = new DequeueImpl<>();
      assertTrue(q.isEmpty());
      int count = 100000;
      for (int i = 0; i < count; i++) {
          q.pushFirst(i);
          assertTrue(q.size() == i + 1);
      }
      assertTrue(q.first() == count -1);
      assertTrue(q.size() == count);
     int current = count - 1;
      while (!q.isEmpty()) {
          assertTrue(q.popFirst()== current);
          current--;
      }
      assertTrue(q.isEmpty());
  }
```

testLastQueue():

```
@Test
public void testLastQueue() {
    DequeueImpl<Integer> q = new DequeueImpl<>();
    assertTrue(q.isEmpty());
    int count = 100000;
    for (int i = 0; i < count; i++) {
        q.pushLast(i);
        assertTrue(q.size() == i + 1);
    }
    assertTrue(q.last()== count -1);
    assertTrue(q.size() == count);
   int current = count - 1;
    while (!q.isEmpty()) {
        assertTrue(q.popLast()== current);
        current--;
    }
    assertTrue(q.isEmpty());
}
```

testPushFirstAndPopFirst():

```
@Test
public void testPushFirstAndPopFirst() {
    DequeueImpl<Integer> deque = new DequeueImpl<>();
    deque.pushFirst(1);
    deque.pushFirst(2);
    deque.pushFirst(3);

    assertEquals(3, (int) deque.popFirst());
    assertEquals(2, (int) deque.popFirst());
    assertEquals(1, (int) deque.popFirst());
    assertTrue(deque.isEmpty());
}
```

testPushLastAndPopLast():

```
@Test
public void testPushLastAndPopLast() {
    DequeueImpl<String> deque = new DequeueImpl<>();
    deque.pushLast("A");
    deque.pushLast("B");
    deque.pushLast("C");

    assertEquals("C", deque.popLast());
    assertEquals("B", deque.popLast());
    assertEquals("A", deque.popLast());
    assertTrue(deque.isEmpty());
}
```

testClear():

```
@Test
public void testClear() {
    DequeueImpl<Integer> deque = new DequeueImpl<>();
    deque.pushFirst(1);
    deque.pushLast(2);
    deque.pushFirst(3);

    assertFalse(deque.isEmpty());
    deque.clear();
    assertTrue(deque.isEmpty());
    assertEquals(0, deque.size());
}
```

testSizeAndIsEmpty():

```
@Test
public void testSizeAndIsEmpty() {
    DequeueImpl<Double> deque = new DequeueImpl<>();
    assertTrue(deque.isEmpty());
    assertEquals(0, deque.size());

    deque.pushFirst(1.1);
    deque.pushLast(2.2);
    deque.pushFirst(3.3);

    assertFalse(deque.isEmpty());
    assertEquals(3, deque.size());

    deque.popFirst();
    deque.popLast();

    assertFalse(deque.isEmpty());
    assertEquals(1, deque.size());
}
```

testFirstAndLast():

```
@Test
public void testFirstAndLast() {
    DequeueImpl<Character> deque = new DequeueImpl<>();
    deque.pushFirst('A');
    deque.pushLast('B');
    deque.pushLast('C');

    assertEquals('A', (char) deque.first());
    assertEquals('C', (char) deque.last());
}
```

Όλα αυτά τα tests έχουν φτιαχτεί για να ελέγχουν ότι όλες οι λειτουργίες που υλοποιήσαμε δουλεύουν έτσι όπως θα έπρεπε.

```
[INFO] skip non existing resourceDirectory C:\Users\John\Documents\NetBeansProjects\DataStructure\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ DataStructure ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\Users\John\Documents\NetBeansProjects\DataStructure\target\test-classes
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ DataStructure ---
[INFO] Surefire report directory: C:\Users\John\Documents\NetBeansProjects\DataStructure\target\surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running Test1
Tests run: 7, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.029 sec

Results :

Tests run: 7, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- maven-jar-plugin:3.1.0:jar (default-jar) @ DataStructure ---
[INFO] Building jar: C:\Users\John\Documents\NetBeansProjects\DataStructure\target\DataStructure-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.758 s
[INFO] Finished at: 2023-12-31T15:19:31+02:00
[INFO] ------------------------------------------------------------------------
```

4. Main:

Για την main υλοποιήσαμε παρόμοιο κώδικα με αυτόν του εργαστηρίου 2, αλλά για λόγους πληρότητας εκτελέσαμε διαφορετικές παραλλαγές του κώδικα.

First:

```
public class App {

    public static void main(String[] args) {
       DequeueImpl<Integer> q = new DequeueImpl<>();
       for (int i = 0; i < 65; i++) {
                System.out.println("Adding element " + i + " to Last with array length: "+ q.Length());
                q.pushFirst(i);
            }

            while (!q.isEmpty()) {
                System.out.println("Next element served from queue: " + q.popFirst()+" With array length: "+ q.Length());
            }


    }
}
```

```
Adding element 1 to Last with array length: 8
Adding element 2 to Last with array length: 8
Adding element 3 to Last with array length: 8
Adding element 4 to Last with array length: 8
Adding element 5 to Last with array length: 8
Adding element 6 to Last with array length: 8
Adding element 7 to Last with array length: 8
Adding element 8 to Last with array length: 8
Adding element 9 to Last with array length: 16
Adding element 10 to Last with array length: 16
Adding element 11 to Last with array length: 16
Adding element 12 to Last with array length: 16
Adding element 13 to Last with array length: 16
Adding element 14 to Last with array length: 16
Adding element 15 to Last with array length: 16
Adding element 16 to Last with array length: 16
Adding element 17 to Last with array length: 32
Adding element 18 to Last with array length: 32
Adding element 19 to Last with array length: 32
Adding element 20 to Last with array length: 32
Adding element 21 to Last with array length: 32
Adding element 22 to Last with array length: 32
Adding element 23 to Last with array length: 32
Adding element 24 to Last with array length: 32
Adding element 25 to Last with array length: 32
Adding element 26 to Last with array length: 32
Adding element 27 to Last with array length: 32
Adding element 28 to Last with array length: 32
Adding element 29 to Last with array length: 32
Adding element 30 to Last with array length: 32
Adding element 31 to Last with array length: 32
Adding element 32 to Last with array length: 32
Adding element 33 to Last with array length: 64
Adding element 34 to Last with array length: 64
Adding element 35 to Last with array length: 64
Adding element 36 to Last with array length: 64
Adding element 37 to Last with array length: 64
Adding element 38 to Last with array length: 64
Adding element 39 to Last with array length: 64
Adding element 40 to Last with array length: 64
Adding element 41 to Last with array length: 64
Adding element 42 to Last with array length: 64
Adding element 43 to Last with array length: 64
Adding element 44 to Last with array length: 64
Adding element 45 to Last with array length: 64
Adding element 46 to Last with array length: 64
Adding element 47 to Last with array length: 64
Adding element 48 to Last with array length: 64
Adding element 49 to Last with array length: 64
Adding element 50 to Last with array length: 64
Adding element 51 to Last with array length: 64
Adding element 52 to Last with array length: 64
Adding element 53 to Last with array length: 64
Adding element 54 to Last with array length: 64
Adding element 55 to Last with array length: 64
Adding element 56 to Last with array length: 64
Adding element 57 to Last with array length: 64
Adding element 58 to Last with array length: 64
Adding element 59 to Last with array length: 64
Adding element 60 to Last with array length: 64
Adding element 61 to Last with array length: 64
Adding element 62 to Last with array length: 64
Adding element 63 to Last with array length: 64
```

```
Next element served from queue: 62 With array length: 128
Next element served from queue: 61 With array length: 128
Next element served from queue: 60 With array length: 128
Next element served from queue: 59 With array length: 128
Next element served from queue: 58 With array length: 128
Next element served from queue: 57 With array length: 128
Next element served from queue: 56 With array length: 128
Next element served from queue: 55 With array length: 128
Next element served from queue: 54 With array length: 128
Next element served from queue: 53 With array length: 128
Next element served from queue: 52 With array length: 128
Next element served from queue: 51 With array length: 128
Next element served from queue: 50 With array length: 128
Next element served from queue: 49 With array length: 128
Next element served from queue: 48 With array length: 128
Next element served from queue: 47 With array length: 128
Next element served from queue: 46 With array length: 128
Next element served from queue: 45 With array length: 128
Next element served from queue: 44 With array length: 128
Next element served from queue: 43 With array length: 128
Next element served from queue: 42 With array length: 128
Next element served from queue: 41 With array length: 128
Next element served from queue: 40 With array length: 128
Next element served from queue: 39 With array length: 128
Next element served from queue: 38 With array length: 128
Next element served from queue: 37 With array length: 128
Next element served from queue: 36 With array length: 128
Next element served from queue: 35 With array length: 128
Next element served from queue: 34 With array length: 128
Next element served from queue: 33 With array length: 128
Next element served from queue: 32 With array length: 128
Next element served from queue: 31 With array length: 64
Next element served from queue: 30 With array length: 64
Next element served from queue: 29 With array length: 64
Next element served from queue: 28 With array length: 64
Next element served from queue: 27 With array length: 64
Next element served from queue: 26 With array length: 64
Next element served from queue: 25 With array length: 64
Next element served from queue: 24 With array length: 64
Next element served from queue: 23 With array length: 64
Next element served from queue: 22 With array length: 64
Next element served from queue: 21 With array length: 64
Next element served from queue: 20 With array length: 64
Next element served from queue: 19 With array length: 64
Next element served from queue: 18 With array length: 64
Next element served from queue: 17 With array length: 64
Next element served from queue: 16 With array length: 64
Next element served from queue: 15 With array length: 32
Next element served from queue: 14 With array length: 32
Next element served from queue: 13 With array length: 32
Next element served from queue: 12 With array length: 32
Next element served from queue: 11 With array length: 32
Next element served from queue: 10 With array length: 32
Next element served from queue: 9 With array length: 32
Next element served from queue: 8 With array length: 32
Next element served from queue: 7 With array length: 16
Next element served from queue: 6 With array length: 16
Next element served from queue: 5 With array length: 16
Next element served from queue: 4 With array length: 16
Next element served from queue: 3 With array length: 8
Next element served from queue: 2 With array length: 8
Next element served from queue: 1 With array length: 4
Next element served from queue: 0 With array length: 4
```

Last:

```java
public class App {

    public static void main(String[] args) {
        DequeueImpl<Integer> q = new DequeueImpl<>();
        for (int i = 0; i < 65; i++) {
                System.out.println("Adding element " + i + " to Last with array length: "+ q.Length());
                q.pushLast(i);
            }

            while (!q.isEmpty()) {
                System.out.println("Next element served from queue: " + q.popLast()+" With array length: "+ q.Length());
            }


    }
}
```

```
Adding element 0 to Last with array length: 8
Adding element 1 to Last with array length: 8
Adding element 2 to Last with array length: 8
Adding element 3 to Last with array length: 8
Adding element 4 to Last with array length: 8
Adding element 5 to Last with array length: 8
Adding element 6 to Last with array length: 8
Adding element 7 to Last with array length: 8
Adding element 8 to Last with array length: 16
Adding element 9 to Last with array length: 16
Adding element 10 to Last with array length: 16
Adding element 11 to Last with array length: 16
Adding element 12 to Last with array length: 16
Adding element 13 to Last with array length: 16
Adding element 14 to Last with array length: 16
Adding element 15 to Last with array length: 16
Adding element 16 to Last with array length: 32
Adding element 17 to Last with array length: 32
Adding element 18 to Last with array length: 32
Adding element 19 to Last with array length: 32
Adding element 20 to Last with array length: 32
Adding element 21 to Last with array length: 32
Adding element 22 to Last with array length: 32
Adding element 23 to Last with array length: 32
Adding element 24 to Last with array length: 32
Adding element 25 to Last with array length: 32
Adding element 26 to Last with array length: 32
Adding element 27 to Last with array length: 32
Adding element 28 to Last with array length: 32
Adding element 29 to Last with array length: 32
Adding element 30 to Last with array length: 32
Adding element 31 to Last with array length: 32
Adding element 32 to Last with array length: 64
Adding element 33 to Last with array length: 64
Adding element 34 to Last with array length: 64
Adding element 35 to Last with array length: 64
Adding element 36 to Last with array length: 64
Adding element 37 to Last with array length: 64
Adding element 38 to Last with array length: 64
Adding element 39 to Last with array length: 64
Adding element 40 to Last with array length: 64
Adding element 41 to Last with array length: 64
Adding element 42 to Last with array length: 64
Adding element 43 to Last with array length: 64
Adding element 44 to Last with array length: 64
Adding element 45 to Last with array length: 64
Adding element 46 to Last with array length: 64
Adding element 47 to Last with array length: 64
Adding element 48 to Last with array length: 64
Adding element 49 to Last with array length: 64
Adding element 50 to Last with array length: 64
Adding element 51 to Last with array length: 64
Adding element 52 to Last with array length: 64
Adding element 53 to Last with array length: 64
Adding element 54 to Last with array length: 64
Adding element 55 to Last with array length: 64
Adding element 56 to Last with array length: 64
Adding element 57 to Last with array length: 64
Adding element 58 to Last with array length: 64
Adding element 59 to Last with array length: 64
Adding element 60 to Last with array length: 64
Adding element 61 to Last with array length: 64
Adding element 62 to Last with array length: 64
```

```
Next element served from queue: 63 With array length: 128
Next element served from queue: 62 With array length: 128
Next element served from queue: 61 With array length: 128
Next element served from queue: 60 With array length: 128
Next element served from queue: 59 With array length: 128
Next element served from queue: 58 With array length: 128
Next element served from queue: 57 With array length: 128
Next element served from queue: 56 With array length: 128
Next element served from queue: 55 With array length: 128
Next element served from queue: 54 With array length: 128
Next element served from queue: 53 With array length: 128
Next element served from queue: 52 With array length: 128
Next element served from queue: 51 With array length: 128
Next element served from queue: 50 With array length: 128
Next element served from queue: 49 With array length: 128
Next element served from queue: 48 With array length: 128
Next element served from queue: 47 With array length: 128
Next element served from queue: 46 With array length: 128
Next element served from queue: 45 With array length: 128
Next element served from queue: 44 With array length: 128
Next element served from queue: 43 With array length: 128
Next element served from queue: 42 With array length: 128
Next element served from queue: 41 With array length: 128
Next element served from queue: 40 With array length: 128
Next element served from queue: 39 With array length: 128
Next element served from queue: 38 With array length: 128
Next element served from queue: 37 With array length: 128
Next element served from queue: 36 With array length: 128
Next element served from queue: 35 With array length: 128
Next element served from queue: 34 With array length: 128
Next element served from queue: 33 With array length: 128
Next element served from queue: 32 With array length: 128
Next element served from queue: 31 With array length: 64
Next element served from queue: 30 With array length: 64
Next element served from queue: 29 With array length: 64
Next element served from queue: 28 With array length: 64
Next element served from queue: 27 With array length: 64
Next element served from queue: 26 With array length: 64
Next element served from queue: 25 With array length: 64
Next element served from queue: 24 With array length: 64
Next element served from queue: 23 With array length: 64
Next element served from queue: 22 With array length: 64
Next element served from queue: 21 With array length: 64
Next element served from queue: 20 With array length: 64
Next element served from queue: 19 With array length: 64
Next element served from queue: 18 With array length: 64
Next element served from queue: 17 With array length: 64
Next element served from queue: 16 With array length: 64
Next element served from queue: 15 With array length: 32
Next element served from queue: 14 With array length: 32
Next element served from queue: 13 With array length: 32
Next element served from queue: 12 With array length: 32
Next element served from queue: 11 With array length: 32
Next element served from queue: 10 With array length: 32
Next element served from queue: 9 With array length: 32
Next element served from queue: 8 With array length: 32
Next element served from queue: 7 With array length: 16
Next element served from queue: 6 With array length: 16
Next element served from queue: 5 With array length: 16
Next element served from queue: 4 With array length: 16
Next element served from queue: 3 With array length: 8
Next element served from queue: 2 With array length: 8
Next element served from queue: 1 With array length: 4
```

Iterator:

```java
public class App {

    public static void main(String[] args) {
        DequeueImpl<Integer> q = new DequeueImpl<>();
        q.pushFirst(4);
        q.pushLast(5);
        q.pushLast(6);
        q.pushLast(7);
        q.pushFirst(3);
        q.pushFirst(2);
        q.pushFirst(1);

        Iterator<Integer> iterator = q.iterator();
         while (iterator.hasNext()) {
             System.out.println(iterator.next());
         }


    }
}
```

```
1
2
3
4
5
6
7
```

Descending Iterator:

```java
public class App {

    public static void main(String[] args) {
        DequeueImpl<Integer> q = new DequeueImpl<>();
        q.pushFirst(4);
        q.pushLast(5);
        q.pushLast(6);
        q.pushLast(7);
        q.pushFirst(3);
        q.pushFirst(2);
        q.pushFirst(1);

        Iterator<Integer> iterator = q.descendingIterator();
         while (iterator.hasNext()) {
             System.out.println(iterator.next());
         }


    }
}
```

```
7
6
5
4
3
2
1
```