JAVA Lab Assignment 3

Name: Om Ashish Mishra

Registration Number: 16BCE0789

Slot: G2

The Question:

8. A double-ended queue is a data structure in which the enqueue and dequeue operations can be done at both the ends of the queue. Implement a doubleended queue using a suitable Java Collection object. Develop a Java application to test the same.

The Answer:

The Code:

package DoublyLinkedList;

/\*\*

\*

\* @author OM MISHRA

\*/

public class DoublyLinkedList<T> {

private Node<T> head;

private Node<T> tail;

/\*\*

\* Inserts the value at the first position (head) of LinkedList.

\*

\* @param value

\* to be inserted

\*/

public void insertFirst(final T value) {

final Node<T> node = new Node<>(value);

node.next = head;

if (head != null) {

head.previous = node;

}

head = node;

if (tail == null) {

tail = node;

}

}

/\*\*

\* Inserts the value at the last position (tail) of LinkedList.

\*

\* @param value

\*/

public void insertLast(final T value) {

final Node<T> node = new Node<>(value);

if (tail != null) {

tail.next = node;

node.previous = tail;

}

tail = node;

if (head == null) {

head = node;

}

}

/\*\*

\* Removes the node from first position (head) of LinkedList.

\*

\* @return the value of node deleted. Null if no nodes are present

\*/

public T removeFirst() {

T value = null;

if (head != null) {

value = head.value;

if (head == tail) {

tail = null;

}

head = head.next;

head.previous = null;

}

return value;

}

/\*\*

\* Removes the node from last position (tail) of LinkedList.

\*

\* @return the value of node deleted. Null if no nodes are present

\*/

public T removeLast() {

T value = null;

if (tail != null) {

value = tail.value;

if (tail == head) {

head = tail = null;

} else {

tail = tail.previous;

tail.next = null;

}

}

return value;

}

/\*\*

\* Removes the first occurance of node having the value same as input value.

\*

\* @param value

\* to be removed

\* @return deleted node's value if node is found else null;

\*/

public T remove(final T value) {

T deletedObj = null;

if (head != null) {

if (head == tail) {

if (head.value.equals(value)) {

deletedObj = head.value;

head = tail = null;

}

} else {

Node<T> node = head;

do {

if (node.value.equals(value)) {

deletedObj = node.value;

if (node.previous != null) {

node.previous.next = node.next;

}

node.next.previous = node.previous;

break;

}

node = node.next;

} while (node != null);

}

}

return deletedObj;

}

/\*\*

\* Implementation of a Node of a Doubly Linked List.

\*

\* @author Sain Technology Solutions

\*

\* @param <T>

\*/

private static class Node<T> {

T value;

Node<T> next;

Node<T> previous;

private Node(T value) {

this.value = value;

}

@Override

public String toString() {

return "Node [value=" + value + "]";

}

}

/\*\*

\* Entry point for testing LinkedList.

\*/

public static void main(String[] args) {

final DoublyLinkedList<Integer> doublyLinkedList = new DoublyLinkedList<>();

// Inserts the node with value 5 at the head position

doublyLinkedList.insertFirst(5);

// Inserts the node with value 1 at the head position, pushing the

// previously inserted node to second position

doublyLinkedList.insertFirst(1);

// Inserts the node with value 2 at the head position, pushing the

// previously inserted node to second position

doublyLinkedList.insertFirst(2);

// Inserts the node with value 3 at the tail position

doublyLinkedList.insertLast(3);

// Inserts the node with value 4 at the tail position, pushing the

// previously inserted node to second position from last

doublyLinkedList.insertLast(4);

// At this point, LinkedList will look like: 2 <=> 1 <=> 5 <=> 3 <=> 4

// Removes the node with value 2 since it is head node. This operation

// will also make node with value 1 as head node

System.out.println(doublyLinkedList.removeFirst());

// Removes the node with value 1 since it is head node. This operation

// will also make node with value 5 as head node

System.out.println(doublyLinkedList.removeFirst());

// Removes the node with value 4 since it is tail node. This operation

// will also make node with value 3 as tail node

System.out.println(doublyLinkedList.removeLast());

// Removes the node with value 3 since it is tail node. This operation

// will also make node with value 5 as tail node

System.out.println(doublyLinkedList.removeLast());

// Removes the node with value 5

System.out.println(doublyLinkedList.remove(5));

// Returns null since there is no node with value 2 as it was removed

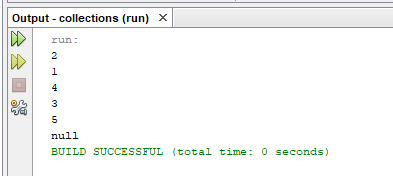
// due to earlier removeXXX method calls

System.out.println(doublyLinkedList.remove(2));

}

}

The Output:



9. Write a Java application to store the different comments given by the users on the surgical attack by our Indian army on terrorist camps in Pakistan held on 26-Feb-2019, 3.30am. The comment should be of ONE word and the duplicates need not to be stored. Use suitable Java Collection object to implement this program and test it.

The Answer:

The Code:

package Collections;

import java.util.\*;

/\*\*

\*

\* @author OM MISHRA

\*/

public class hashset {

public static void main(String arg[]){

HashSet hs=new HashSet();

hs.add("Pakisthan");

hs.add("3:30 AM");

hs.add("26-Feb-2019");

hs.add("Army");

hs.add("Attack");

hs.add("Army");

hs.add("Army");

hs.add("Firing");

hs.add(200);

Iterator it1=hs.iterator();

System.out.println("Elements of HashSet");

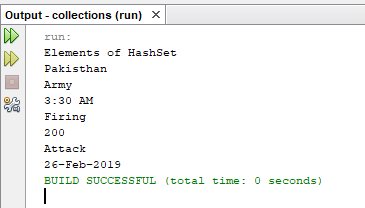
while(it1.hasNext())

System.out.println(it1.next());

}

}

The Output:



10. Implement a Lucky Draw game in which the user has to input an integer ‘input’ and multiply it with a random integer to get the product ‘p’. Depending upon the value of ‘p’ display the prize amount. Use a suitable Java Collection object to store the prize amount for each value of ‘p’. [Hint: Use modulo operation to get final value of ‘p’]

The Answer:

The Code:

package DoublyLinkedList;

import java.util.\*;

/\*\*

\*

\* @author OM MISHRA

\*/

public class Linkedhashmap {

public static void main(String args[]){

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

Random rn = new Random();

int n1 = rn.nextInt(50);

System.out.println("Random Number Generated : "+n1);

int val = n\*n1;

System.out.println("Product Formed : "+val);

int p = val%5;

System.out.println("Value of p : "+p);

LinkedHashMap<Integer,String> hm=new LinkedHashMap();

hm.put(0,"Bicycle");

hm.put(1,"Car");

hm.put(2,"Bike");

hm.put(3,"Game Set");

hm.put(4, "Dog");

Set set=hm.entrySet();

Iterator itr=set.iterator();

while(itr.hasNext()){

//Converting to Map.Entry so that we can get key and value separately

Map.Entry entry=(Map.Entry)itr.next();

if(p == (int)(entry.getKey()))

{

System.out.println("The prize recieved is "+entry.getValue());

}

}

}

}

The Output:

