Sukkur IBA University
Department of Computer Science

DATA STRUCTURES
Lab01 – Arrays, LinkesLists

Instructor: Saif Hassan

READ IT FIRST

Prior to start solving the problems in this assignments, please give full concentration on following points.

1. WORKING – This is individual lab. If you are stuck in a problem contact your teacher, but, in mean time start doing next question (don't waste time).
2. DEADLINE – 11th March, 2022
3. SUBMISSION – This assignment needs to be submitted in a soft copy.
4. WHERE TO SUBMIT – Please visit your LMS.
5. WHAT TO SUBMIT – Submit this docx and pdf file.

KEEP IT WITH YOU!

1. Indent your code inside the classes and functions. It's a good practice!
2. It is not bad if you keep your code indented inside the loops, if and else blocks as well.
3. Comment your code, where it is necessary.
4. Read the entire question. Don't jump to the formula directly.

I_**Amjad Ali**_ with student ID _**191-21-0001**_

Section ____**"A"**_____hereby declare that I do understand the instructions above and follow them. This is

my own work.

Exercises

Task1 Description

Stack

Note: Keep this code with you till the course ends.

Task 01: (Stack using array)

Understand provided code and implement all required methods in Stack. Stack Code is given below:

```
1.  import java.util.*;
2.
3.  class Stack
4.  {
5.      private int arr[];
6.      private int top;
7.      private int capacity;
8.
9.      // Constructor to initialize stack
10.     Stack(int size)
11.     {
12.         arr = new int[size];
13.         capacity = size;
14.         top = -1;
15.     }
16.
17.     // Utility function to add an element x in the stack and check for stack overflow
18.     public void push(int x)
19.     {
20.         // Write your code here
21.     }
22.
23.     // Utility function to pop top element from the stack and check for stack underflow

24.     public int pop()
25.     {
26.         // Write your code here
27.     }
28.
```

```
29.     // Utility function to return top element in a stack
30.     public int peek()
31.     {
32.         // Write your code here
33.     }
34.
35.     // Utility function to return the size of the stack
36.     public int size()
37.     {
38.         // Write your code here
39.     }
40.
41.     // Utility function to check if the stack is empty or not
42.     public Boolean isEmpty()
43.     {
44.         // Write your code here
45.     }
46.
47.     // Utility function to check if the stack is full or not
48.     public Boolean isFull()
49.     {
50.         // Write your code here
51.     }
52.
53.     public static void main (String[] args)
54.     {
55.         Stack stack = new Stack(3);
56.
57.         stack.push(1);      // Inserting 1 in the stack
58.         stack.push(2);      // Inserting 2 in the stack
59.
60.         stack.pop();        // removing the top 2
61.         stack.pop();        // removing the top 1
62.
63.         stack.push(3);      // Inserting 3 in the stack
65.         System.out.println("Top element is: " + stack.peek());
66.         System.out.println("Stack size is " + stack.size());
67.
68.         stack.pop();        // removing the top 3
69.
70.         // check if stack is empty
71.         if (stack.isEmpty())
72.             System.out.println("Stack Is Empty");
73.         else
74.             System.out.println("Stack Is Not Empty");
75.     }
76. }
```

After implementing all the methods, run the code. Your output should be like as follows:

Output

```
Inserting 1
Inserting 2
Removing 2
Removing 1
Inserting 3
Top element is: 3
Stack size is 1
Removing 3
Stack Is Empty
```

Solution:

Code:

```
1.  import java.util.*;
2.
3. class Stack {
4.     private int arr[];
5.     private int top;
6.     private int capacity;
7.
8.     // Constructor to initialize stack
9.     Stack(int size) {
10.         arr = new int[size];
11.         capacity = size;
12.         top = -1;
13.     }
14.
15.     // Utility function to add an element x in the stack and
   check for stack overflow
16.     public void push(int x) {
17.         // Write your code here
18.         if (isFull()) {
19.             System.out.println("Stack Overflow");
20.         } else {
21.             System.out.println("Inserting "+x);
22.             arr[++top] = x;
```

```
23.         }
24.     }
25.     // Utility function to pop top element from the stack and
   check for stack underflow
26.
27.     public int pop() {
28.         // Write your code here
29.         if(isEmpty())
30.         {
31.             System.out.println("Stack underflow");
32.             return -1;
33.         }
34.         else{
35.             System.out.println("Removing "+arr[top]);
36.             return arr[top--];
37.         }
38.     }
39.
40.     // Utility function to return top element in a stack
41.     public int peek() {
42.         // Write your code here
43.         if(isEmpty())
44.         {
45.             System.out.println("Stack underflow");
46.             return -1;
47.         }
48.         else{
49.             return arr[top];
50.         }
51.     }
52.
53.     // Utility function to return the size of the stack
54.     public int size() {// Write your code here
55.         return top + 1;
56.     }
57.
58.     // Utility function to check if the stack is empty or not
59.     public Boolean isEmpty() {// Write your code here
60.         return top == -1;
61.     }
62.
63.     // Utility function to check if the stack is full or not
64.     public Boolean isFull() {
65.         // Write your code here
66.         return top + 1 == capacity;
```

```
67.     }
68.
69.     public static void main(String[] args) {
70.         Stack stack = new Stack(3);
71.         stack.push(1); // Inserting 1 in the stack
72.         stack.push(2); // Inserting 2 in the stack
73.
74.         stack.pop(); // removing the top 2
75.         stack.pop(); // removing the top 1
76.
77.         stack.push(3); // Inserting 3 in the stack
78.
79.         System.out.println("Top element is: " + stack.peek());
80.         System.out.println("Stack size is " + stack.size());
81.
82.         stack.pop(); // removing the top 3
83.         // check if stack is empty
84.         if (stack.isEmpty())
85.             System.out.println("Stack Is Empty");
86.         else
87.             System.out.println("Stack Is Not Empty");
88.     }
89. }
```

Sample Input:

```
Stack stack = new Stack( size: 3);
stack.push( x: 1); // Inserting 1 in the stack
stack.push( x: 2); // Inserting 2 in the stack

stack.pop(); // removing the top 2
stack.pop(); // removing the top 1

stack.push( x: 3); // Inserting 3 in the stack

System.out.println("Top element is: " + stack.peek());
System.out.println("Stack size is " + stack.size());

stack.pop(); // removing the top 3
// check if stack is empty
if (stack.isEmpty())
    System.out.println("Stack Is Empty");
else
    System.out.println("Stack Is Not Empty");
```

Sample Output

```
Inserting 1
Inserting 2
Removing 2
Removing 1
Inserting 3
Top element is: 3
Stack size is 1
Removing 3
Stack Is Empty

Process finished with exit code 0
```

Task2 Description

```
Stack Is Empty
```

Task 02:(Stack using Linked list)

Understand provided code and implement all required methods in Stack. Stack Code is given below:

```
1.  import java.util.*;
2.
3.  // A linked list node
4.  class Node
5.  {
6.      int data;       // integer data
7.      Node next;      // pointer to the next node
8.  };
9.
10. class Stack
11. {
12.     private Node top;
13.
14.     public Stack() {
15.         this.top = null;
16.     }
17.
18.     // Utility function to add an element x in the stack
19.     public void push(int x) // insert at the beginning
20.     {
21.         // Write your code here
22.     }
23.
24.     // Utility function to check if the stack is empty or not
25.     public boolean isEmpty()
26.     {
27.         // Write your code here
28.     }
29.
30.     // Utility function to return top element in a stack
31.     public int peek()
32.     {
33.         // Write your code here
34.     }
35.
36.     // Utility function to pop top element from the stack and check for Stack underflow
37.     public void pop() // remove at the beginning
38.     {
39.         // Write your code here
40.     }
```

```
41. }
42.
43. class StackImpl
44. {
45.     public static void main(String[] args)
46.     {
47.         Stack stack = new Stack();
48.
49.         stack.push(1);
50.         stack.push(2);
51.         stack.push(3);
52.
53.         System.out.println("Top element is " +  stack.peek());
54.
55.         stack.pop();
56.         stack.pop();
57.         stack.pop();
58.
59.         if (stack.isEmpty()) {
60.             System.out.print("Stack is empty");
61.         } else {
62.             System.out.print("Stack is not empty");
63.         }
64.     }
65. }
```

After implementing all the methods, run the code. Your output should be like as follows:

Output

```
Inserting 1
Inserting 2
Inserting 3
Top element is 3
Removing 3
Removing 2
Removing 1
Stack is empty
```

Solution:

Code:

Node and Stack Classes

```
1. package com.company;
2.
3. import java.util.*;
4.
5. // A linked list node
6. class Node {
7.     int data; // integer data
8.     Node next; // pointer to the next node
9.
10.     Node(int data) {
11.         this.data = data;
12.         this.next = null;
13.     }
14. }
15.
16. class Stack {
17.     private Node top, tail;
18.
19.     Stack() {
20.         this.top = null;
21.         this.tail = null;
22.     }
23.
24.     // Utility function to add an element x in the stack
25.     public void push(int x) // insert at the beginning
26.     {// Write your code here
27.         Node newNode = new Node(x);
28.         if (isEmpty()) {
29.             top = tail = newNode;
30.         } else {
31.             System.out.println("Inserting " + x);
32.             newNode.next = top;
33.             top = newNode;
34.         }
35.     }
36.
```

```
37.
38.     // Utility function to check if the stack is empty or not
39.     public boolean isEmpty() {
40.         // Write your code here
41.         return top == null;
42.     }
43.
44.     // Utility function to return top element in a stack
45.     public int peek() {
46.         // Write your code here
47.         if (isEmpty()) {
48.             System.out.println("Stack underflow");
49.             return -1;
50.         } else {
51.             return top.data;
52.         }
53.
54.     }
55.
56.     // Utility function to pop top element from the stack and
   check for Stack underflow
57.
58.     public int pop() // remove at the beginning
59.     {// Write your code here
60.         if (isEmpty()) {
61.             System.out.println("Stack underflow");
62.             return -1;
63.         } else {
64.
65.             int temp = top.data;
66.             System.out.println("Removing " + temp);
67.             top = top.next;
68.             return temp;
69.
70.         }
71.
72.     }
73. }
```

Stack Implementation Class

```
1. package com.company;
2.
3. class StackImpl {
```

```
4.      public static void main(String[] args) {
5.          Stack stack = new Stack();
6.
7.          stack.push(1);
8.          stack.push(2);
9.          stack.push(3);
10.
11.           System.out.println("Top element is " + stack.peek());
12.
13.           stack.pop();
14.           stack.pop();
15.           stack.pop();
16.
17.           if (stack.isEmpty()) {
18.               System.out.print("Stack is empty");
19.           } else {
20.               System.out.print("Stack is not empty");
21.           }
22.       }
23.  }
```

Sample Input:

```
stack.push( x: 1);
stack.push( x: 2);
stack.push( x: 3);

System.out.println("Top element is " + stack.peek());

stack.pop();
stack.pop();
stack.pop();

if (stack.isEmpty()) {
    System.out.print("Stack is empty");
} else {
    System.out.print("Stack is not empty");
}
```

Sample Output

```
Inserting 2
Inserting 3
Top element is 3
Removing 3
Removing 2
Removing 1
Stack is empty
Process finished with exit code 0
```

Task3 Description

Queue

Note: Keep this code with you till the course ends.

Task 03: (Queue using array)

Understand provided code and implement all required methods in Queue. Queue Code is given below:

```java
import java.util.*;

// Class for queue
class Queue
{
    private int arr[];
    private int front;
    private int rear;
    private int capacity;
    private int count;

    // Constructor to initialize queue
    Queue(int size)
    {
        arr = new int[size];
        capacity = size;
        front = 0;
        rear = -1;
        count = 0;
    }

    // Utility function to remove front element from the queue and check for Queue Underflow
    public void dequeue()
    {
        // Write your code here
    }

    // Utility function to add an item to the queue and check for queue overflow
    public void enqueue(int item)
    {
        // Write your code here
    }

    // Utility function to return front element in the queue and check for Queue Underflow
    public int peek()
    {
        // Write your code here
    }

    // Utility function to return the size of the queue
    public int size()
    {
        // Write your code here
    }

    // Utility function to check if the queue is empty or not
    public Boolean isEmpty()
    {
        // Write your code here
```

```
50.     }
51.
52.     // Utility function to check if the queue is empty or not
53.     public Boolean isFull()
54.     {
55.         // Write your code here
56.     }
57. }
58.
59. class Main
60. {
61.     // main function
62.     public static void main (String[] args)
63.     {
64.         // create a queue of capacity 5
65.         Queue q = new Queue(5);
66.
67.         q.enqueue(1);
68.         q.enqueue(2);
69.         q.enqueue(3);
70.
71.         System.out.println("Front element is: " + q.peek());
72.         q.dequeue();
73.         System.out.println("Front element is: " + q.peek());
74.
75.         System.out.println("Queue size is " + q.size());
76.
77.         q.dequeue();
78.         q.dequeue();
79.
80.         if (q.isEmpty())
81.             System.out.println("Queue Is Empty");
82.         else
83.             System.out.println("Queue Is Not Empty");
84.     }
85. }
```

After implementing all the methods, run the code. Your output should be like as follows:

Output

```
Inserting 1
Inserting 2
Inserting 3
Front element is: 1
Removing 1
Front element is: 2
Queue size is 2
Removing 2
Removing 3
Queue Is Empty
```

Task 04:(Queue using Linked list)

Solution:

Code:-

Queue Class

```
1. package com.company;
2.
3. import java.util.*;
4.
5. // Class for queue
6.
7. public class Queue {
8.     private int arr[];
9.     private int front;
10.     private int rear;
11.     private int capacity;
12.     private int count;
13.
14.     // Constructor to initialize queue
15.     public Queue(int size) {
16.         arr = new int[size];
17.         capacity = size;
18.         front = 0;
19.         rear = -1;
20.         count = 0;
21.     }
22.
23.     // Utility function to remove front element from
  the queue and check for Queue Underflow
24.     public void dequeue() {
25.         // Write your code here
26.         if (isEmpty()) {
27.             System.out.println("Stack underflow");
28.         } else {
29.             System.out.println("Removing " +
  arr[front]);
30.             for (int i = 0; i < rear + 1; i++) {
31.                 arr[i] = arr[i + 1];
```

```
32.
33.             }
34.             rear = rear - 1;
35.
36.         }
37.
38.     }
39.
40.     // Utility function to add an item to the queue
  and check for queue overflow
41.     public void enqueue(int item) {
42.         // Write your code here
43.         if (isFull()) {
44.             System.out.println("Queue overflow");
45.         } else {
46.             System.out.println("Inserting " + item);
47.             arr[++rear] = item;
48.         }
49.     }
50.
51.     // Utility function to return front element in the
  queue and check for Queue Underflow
52.     public int peek() {
53.         // Write your code here
54.         if (isFull()) {
55.             System.out.println("Queue overflow");
56.             return -1;
57.         } else {
58.             int temp = arr[front];
59.             return temp;
60.         }
61.
62.     }
63.
64.     // Utility function to return the size of the
  queue
65.     public int size() {
66.         // Write your code here
67.         return rear + 1;
```

```
68.     }
69.
70.     // Utility function to check if the queue is empty
  or not
71.     public Boolean isEmpty() {
72.         // Write your code here
73.         return rear == -1;
74.     }
75.
76.     // Utility function to check if the queue is empty
  or not
77.     public Boolean isFull() {
78.         // Write your code here
79.         return rear + 1 == capacity;
80.     }
81.
82. }
```

Queue Implementation Class

```
1. import com.company.Queue;
2.
3. public class QueueImpl {
4.     // main function
5.     public static void main(String[] args) {
6.         // create a queue of capacity 5
7.         Queue q = new Queue(5);
8.
9.         q.enqueue(1);
10.          q.enqueue(2);
11.          q.enqueue(3);
12.
13.          System.out.println("Front element is: " + q.peek());
14.          q.dequeue();
15.          System.out.println("Front element is: " + q.peek());
16.
17.          System.out.println("Queue size is " + q.size());
18.
19.          q.dequeue();
20.          q.dequeue();
21.
```

```
22.          if (q.isEmpty())
23.              System.out.println("Queue Is Empty");
24.          else
25.              System.out.println("Queue Is Not Empty");
26.      }
27.  }
```

Sample Input:

```
q.enqueue( item: 1);
q.enqueue( item: 2);
q.enqueue( item: 3);
System.out.println("Front element is: " + q.peek());
q.dequeue();
System.out.println("Front element is: " + q.peek());
System.out.println("Queue size is " + q.size());
q.dequeue();
q.dequeue();
if (q.isEmpty())
    System.out.println("Queue Is Empty");
else
    System.out.println("Queue Is Not Empty");
```

Sample Output

```
Inserting 1
Inserting 2
Inserting 3
Front element is: 1
Removing 1
Front element is: 2
Queue size is 2
Removing 2
Removing 3
Queue Is Empty
```

Task4 Description

Task 04:(Queue using Linked list)

Understand provided code and implement all required methods in Queue. Queue Code is given below:

```
1.  // A linked list node
2.  class Node
3.  {
4.      int data;       // integer data
5.      Node next;      // pointer to the next node
6.
7.      public Node(int data)
8.      {
9.          // set the data in allocated node and return the node
10.         this.data = data;
11.         this.next = null;
12.     }
13. }
14.
15. class Queue
16. {
17.     private static Node rear = null, front = null;
18.
19.     // Utility function to remove front element from the queue and check for Queue Unde
    rflow
20.     public static int dequeue()     // delete at the beginning
21.     {
22.         // Write your code here
23.     }
24.
25.     // Utility function to add an item in the queue
26.     public static void enqueue(int item)    // insertion at the end
27.     {
28.         // Write your code here
29.     }
30.
31.     // Utility function to return top element in a queue
32.     public static int peek()
33.     {
34.         // Write your code here
35.      }
36.
37.     // Utility function to check if the queue is empty or not
38.     public static boolean isEmpty()
39.     {
40.         // Write your code here
41.     }
42. }
43.
44. class Main {
45.     public static void main(String[] args)
46.     {
47.         Queue q = new Queue();
48.         q.enqueue(1);
49.         q.enqueue(2);
50.         q.enqueue(3);
51.         q.enqueue(4);
52.
53.         System.out.printf("Front element is %d\n", q.peek());
54.
55.         q.dequeue();
56.         q.dequeue();
```

```
57.         q.dequeue();
58.         q.dequeue();
59.
60.         if (q.isEmpty()) {
61.             System.out.print("Queue is empty");
62.         } else {
63.             System.out.print("Queue is not empty");
64.         }
65.     }
66. }
```

After implementing all the methods, run the code. Your output should be like as follows:

Output

```
Inserting 1
Inserting 2
Inserting 3
Inserting 4
Front element is 1
Removing 1
Removing 2
Removing 3
Removing 4
Queue is empty
```

Solution:

Node and Queue Class

```
1. // A linked list node
2. class Node2 {
3.     int data; // integer data
4.     Node2 next; // pointer to the next node
5.
6.     public Node2(int data) {
7.         // set the data in allocated node and return the node
8.         this.data = data;
9.         this.next = null;
10.     }
```

```
11. }
12.
13. class Queue {
14.     private static Node2 rear = null;
15.     private static Node2 front = null;
16.
17.     // Utility function to remove front element from the queue
    and check for Queue Underflow
18.     public static int dequeue() // delete at the beginning
19.     {
20.         // Write your code here
21.         if (isEmpty()) {
22.             System.out.println("Queue underflow");
23.             return -1;
24.         } else {
25.             int temp = front.data;
26.             System.out.println("Removing " + temp);
27.             front = front.next;
28.             return temp;
29.         }
30.     }
31.
32.     // Utility function to add an item in the queue
33.     public static void enqueue(int item) // insertion at the
    end
34.     {// Write your code here
35.         Node2 newNode = new Node2(item);
36.         if (isEmpty()) {
37.             front = rear = newNode;
38.         } else {
39.             System.out.println("Inserting " + item);
40.             rear.next = newNode;
41.             rear = newNode;
42.         }
43.     }
44.
45.     // Utility function to return top element in a queue
46.     public static int peek() {// Write your code here
47.         if (isEmpty()) {
48.             System.out.println("Queue underflow");
49.             return -1;
50.         } else {
51.             int temp = front.data;
52.             return temp;
53.         }
```

```
54.     }
55.
56.     // Utility function to check if the queue is empty or not
57.     public static boolean isEmpty() {
58.         // Write your code here
59.         return rear == null;
60.     }
61. }
```

Queue Class Implementation

```
1. public class Queue2Imple {
2.
3.
4.     public static void main(String[] args) {
5.         Queue q = new Queue();
6.         q.enqueue(1);
7.         q.enqueue(2);
8.         q.enqueue(3);
9.         q.enqueue(4);
10.
11.          System.out.printf("Front element is %d\n",
   q.peek());
12.
13.         q.dequeue();
14.         q.dequeue();
15.         q.dequeue();
16.         q.dequeue();
17.
18.         if (q.isEmpty()) {
19.             System.out.print("Queue is empty");
20.         } else {
21.             System.out.print("Queue is not empty");
22.         }
23.     }}
```

Sample Input:

```
Queue q = new Queue();
q.enqueue( item: 1);
q.enqueue( item: 2);
q.enqueue( item: 3);
q.enqueue( item: 4);
System.out.printf("Front element is %d\n", q.peek());
q.dequeue();
q.dequeue();
q.dequeue();
q.dequeue();
if (q.isEmpty()) {
    System.out.print("Queue is empty");
} else {
    System.out.print("Queue is not empty");
}
```

Sample Output

```
Inserting 2
Inserting 3
Inserting 4
Front element is 1
Removing 1
Removing 2
Removing 3
Removing 4
Queue is not empty
Process finished with exit code 0
```

Task5 Description

Queue using two Stacks

Question 5: Understand provided code and implement all required methods in Queue Class. Sample Code is given below:

```
1.  import java.util.*;
2.
3.  // Implement Queue using two stacks
4.  class Queue {
5.      private Stack s1, s2;
6.
7.      // Constructor
8.      Queue() {
9.          s1 = new Stack();
10.         s2 = new Stack();
11.     }
12.
13.     // Enqueue an item to the queue
14.     public void enqueue(int data)
15.     {
16.         // Write your code here
17.     }
18.
19.     // Dequeue an item from the queue
20.     public int dequeue()
21.     {
22.         // Write your code here
23.     }
24.
25.     public static void main(String[] args) {
26.         int[] keys = { 1, 2, 3, 4, 5 };
27.         Queue q = new Queue();
28.
29.         // insert above keys
30.         for (int key : keys) {
31.             q.enqueue(key);
32.         }
33.
34.         System.out.println(q.dequeue());    // print 1
35.         System.out.println(q.dequeue());    // print 2
36.     }
37. }
```

After implementing all the methods, run the code.

Solution:

```
1. import java.util.Stack;
2.
3. public class Queue {
4.     Stack<Integer> s1, s2;
5.
6.     Queue() {
```

```
7.          s1 = new Stack();
8.          s2 = new Stack();
9.      }
10.
11.     // Enqueue an item to the queue
12.     public void enqueue(int data) {
13.         // Write your code here
14.         s1.push(data);
15.     }
16.
17.     // Dequeue an item from the queue
18.     public int dequeue() {
19.         // Write your code here
20.         if (s2.isEmpty()) {
21.             while (!s1.isEmpty()) {
22.                 s2.push(s1.pop());
23.             }
24.         }
25.         return s2.pop();
26.     }
27.
28.     public static void main(String[] args) {
29.         int[] keys = {1, 2, 3, 4, 5};
30.         Queue q = new Queue();
31.
32.         // insert above keys
33.         for (int key : keys) {
34.             q.enqueue(key);
35.         }
36.
37.         System.out.println(q.dequeue()); // print 1
38.         System.out.println(q.dequeue()); // print 2
39.     }
40.
41. }
```

Sample Input:

```
    int[] keys = {1, 2, 3, 4, 5};
    Queue q = new Queue();

    // insert above keys
    for (int key : keys) {
        q.enqueue(key);
    }

    System.out.println(q.dequeue()); // print 1
    System.out.println(q.dequeue()); // print 2
}
```

Sample Output

```
"C:\Program Files\Java\jdk-16.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Comm
1
2

Process finished with exit code 0
```

Task6 Description

Bonus Task: Think about the inverse of task 05 (Stack using queue)

Solution:

```
1. import java.util.*;
2. import java.util.Queue;
3.
4. class Stack {
5.     // Two inbuilt queues
6.     Queue<Integer> q1 = new LinkedList<Integer>();
7.     Queue<Integer> q2 = new LinkedList<Integer>();
8.
9.
10.     // To maintain current number of
11.     // elements
12.     int curr_size;
13.
14.     Stack() {
15.         curr_size = 0;
16.     }
17.
18.     void push(int x) {
19.         curr_size++;
20.
21.         // Push x first in empty q2
22.         q2.add(x);
23.
24.         // Push all the remaining
25.         // elements in q1 to q2.
26.         while (!q1.isEmpty()) {
27.             q2.add(q1.peek());
28.             q1.remove();
29.         }
30.
31.         // swap the names of two queues
32.         Queue<Integer> q = q1;
33.         q1 = q2;
34.         q2 = q;
35.     }
36.
37.     void pop() {
```

```
38.
39.         // if no elements are there in q1
40.         if (q1.isEmpty())
41.             return;
42.         q1.remove();
43.         curr_size--;
44.     }
45.
46.     int top() {
47.         if (q1.isEmpty())
48.             return -1;
49.         return q1.peek();
50.     }
51.
52.     int size() {
53.         return curr_size;
54.     }
55.
56.
57.     // driver code
58.     public static void main(String[] args) {
59.         Stack s = new Stack();
60.         s.push(1);
61.         s.push(2);
62.         s.push(3);
63.
64.         System.out.println("current size: " + s.size());
65.         System.out.println(s.top());
66.         s.pop();
67.         System.out.println(s.top());
68.         s.pop();
69.         System.out.println(s.top());
70.
71.         System.out.println("current size: " + s.size());
72.     }
73. }
```

Sample Input:

```
Stack s = new Stack();
s.push( x: 1);
s.push( x: 2);
s.push( x: 3);
System.out.println("current size: " + s.size());
System.out.println(s.top());
s.pop();
System.out.println(s.top());
s.pop();
System.out.println(s.top());

System.out.println("current size: " + s.size());
```

Sample Output

```
current size: 3
3
2
1
current size: 1

Process finished with exit code 0
```