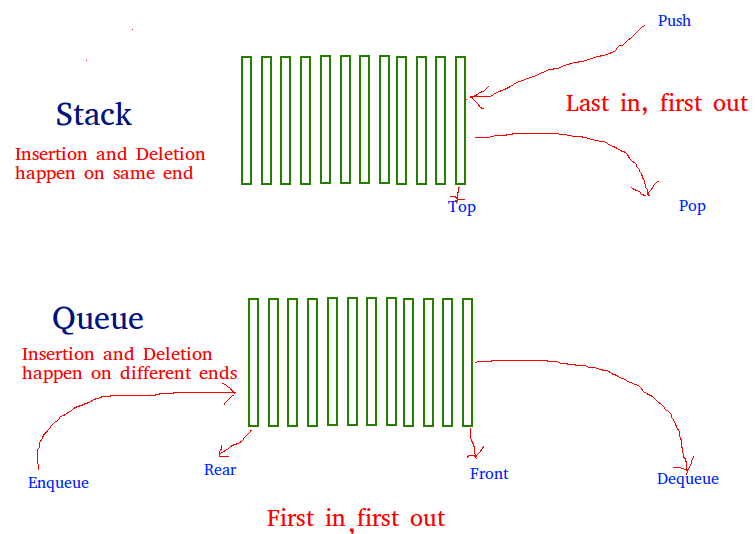
Implement Queue using Stacks

[**2.5**](http://www.geeksforgeeks.org/easy/)

The problem is opposite of [this](http://www.geeksforgeeks.org/implement-stack-using-queue/) post. We are given a stack data structure with push and pop operations, the task is to implement a queue using instances of stack data structure and operations on them.



A queue can be implemented using two stacks. Let queue ` to be implemented be q and stacks used to implement q be stack1 and stack2. q can be implemented in two ways:

[**Recommended: Please solve it on “*PRACTICE*” first, before moving on to the solution.**](http://practice.geeksforgeeks.org/problems/queue-using-two-stacks/1)

**Method 1 (By making enQueue operation costly)** This method makes sure that oldest entered element is always at the top of stack 1, so that deQueue operation just pops from stack1. To put the element at top of stack1, stack2 is used.

enQueue(q, x)

1) While stack1 is not empty, push everything from satck1 to stack2.

2) Push x to stack1 (assuming size of stacks is unlimited).

3) Push everything back to stack1.

dnQueue(q)

1) If stack1 is empty then error

2) Pop an item from stack1 and return it

**Method 2 (By making deQueue operation costly)**In this method, in en-queue operation, the new element is entered at the top of stack1. In de-queue operation, if stack2 is empty then all the elements are moved to stack2 and finally top of stack2 is returned.

enQueue(q, x)

1) Push x to stack1 (assuming size of stacks is unlimited).

deQueue(q)

1) If both stacks are empty then error.

2) If stack2 is empty

While stack1 is not empty, push everything from stack1 to stack2.

3) Pop the element from stack2 and return it.

Method 2 is definitely better than method 1.  
Method 1 moves all the elements twice in enQueue operation, while method 2 (in deQueue operation) moves the elements once and moves elements only if stack2 empty.  
Implementation of method 2:

* C
* Java

|  |
| --- |
| /\* Java Program to implement a queue using two stacks \*/  // Note that Stack class is used for Stack implementation    import java.util.Stack;    public class GFG  {      /\* class of queue having two stacks \*/      static class Queue      {          Stack<Integer> stack1 ;          Stack<Integer> stack2 ;      }        /\* Function to push an item to stack\*/      static void push(Stack<Integer> top\_ref, int new\_data)      {          //Push the data onto the stack          top\_ref.push(new\_data);      }        /\* Function to pop an item from stack\*/      static int pop(Stack<Integer> top\_ref)      {          /\*If stack is empty then error \*/          if(top\_ref.isEmpty())          {              System.out.println("Stack Overflow");              System.exit(0);          }          //pop the data from the stack          return top\_ref.pop();      }      //Function to enqueue an item to the queue      static void enQueue(Queue q, int x)      {          push(q.stack1, x);      }      /\* Function to dequeue an item from queue \*/      static int deQueue(Queue q)      {          int x;          /\* If both stacks are empty then error \*/          if(q.stack1.isEmpty() && q.stack2.isEmpty() )          {              System.out.println("Q is empty");              System.exit(0);          }            /\* Move elements from stack1 to stack 2 only if          stack2 is empty \*/          if(q.stack2.isEmpty())          {              while(!q.stack1.isEmpty())              {              x = pop(q.stack1);              push(q.stack2, x);                }          }          x = pop(q.stack2);          return x;      }        /\* Driver function to test anove functions \*/      public static void main(String args[])      {          /\* Create a queue with items 1 2 3\*/          Queue q= new Queue();          q.stack1 = new Stack<>();          q.stack2 = new Stack<>();          enQueue(q, 1);          enQueue(q, 2);          enQueue(q, 3);            /\* Dequeue items \*/          System.out.print(deQueue(q)+" ");          System.out.print(deQueue(q)+" ");          System.out.println(deQueue(q)+" ");      }  }  //This code is contributed by Sumit Ghosh |

===============Implement Queue===============

Java Program to Implement Queue using Linked List

This is a Java Program to implement a queue using linked list. Queue is a particular kind of abstract data type or collection in which the entities in the collection are kept in order and the principal (or only) operations on the collection are the addition of entities to the rear terminal position and removal of entities from the front terminal position. This makes queue a First-In-First-Out (FIFO) data structure. A linked list is an ordered set of data elements, each containing a link to its successor. Here we need to apply the application of linked list to perform basic operations of a queue.

Here is the source code of the Java program to implement a queue using linked list. The Java program is successfully compiled and run on a Windows system. The program output is also shown below.

*/\**

*\* Java Program to Implement Queue using Linked List*

*\*/*

**import** java.util.\*;

*/\* Class Node \*/*

**class** Node

{

**protected** **int** data;

**protected** Node link;

*/\* Constructor \*/*

**public** Node()

{

link = **null**;

data = 0;

}

*/\* Constructor \*/*

**public** Node(**int** d,Node n)

{

data = d;

link = n;

}

*/\* Function to set link to next Node \*/*

**public** **void** setLink(Node n)

{

link = n;

}

*/\* Function to set data to current Node \*/*

**public** **void** setData(**int** d)

{

data = d;

}

*/\* Function to get link to next node \*/*

**public** Node getLink()

{

**return** link;

}

*/\* Function to get data from current Node \*/*

**public** **int** getData()

{

**return** data;

}

}

*/\* Class linkedQueue \*/*

**class** linkedQueue

{

**protected** Node front, rear;

**public** **int** size;

*/\* Constructor \*/*

**public** linkedQueue()

{

front = **null**;

rear = **null**;

size = 0;

}

*/\* Function to check if queue is empty \*/*

**public** **boolean** isEmpty()

{

**return** front == **null**;

}

*/\* Function to get the size of the queue \*/*

**public** **int** getSize()

{

**return** size;

}

*/\* Function to insert an element to the queue \*/*

**public** **void** insert(**int** data)

{

Node nptr = **new** Node(data, **null**);

**if** (rear == **null**)

{

front = nptr;

rear = nptr;

}

**else**

{

rear.setLink(nptr);

rear = rear.getLink();

}

size++ ;

}

*/\* Function to remove front element from the queue \*/*

**public** **int** remove()

{

**if** (isEmpty() )

**throw** **new** NoSuchElementException("Underflow Exception");

Node ptr = front;

front = ptr.getLink();

**if** (front == **null**)

rear = **null**;

size-- ;

**return** ptr.getData();

}

*/\* Function to check the front element of the queue \*/*

**public** **int** peek()

{

**if** (isEmpty() )

**throw** **new** NoSuchElementException("Underflow Exception");

**return** front.getData();

}

*/\* Function to display the status of the queue \*/*

**public** **void** display()

{

System.out.print("**\n**Queue = ");

**if** (size == 0)

{

System.out.print("Empty**\n**");

**return** ;

}

Node ptr = front;

**while** (ptr != rear.getLink() )

{

System.out.print(ptr.getData()+" ");

ptr = ptr.getLink();

}

System.out.println();

}

}

*/\* Class LinkedQueueImplement \*/*

**public** **class** LinkedQueueImplement

{

**public** **static** **void** main(String[] args)

{

Scanner scan = **new** Scanner(System.in);

*/\* Creating object of class linkedQueue \*/*

linkedQueue lq = **new** linkedQueue();

*/\* Perform Queue Operations \*/*

System.out.println("Linked Queue Test**\n**");

**char** ch;

**do**

{

System.out.println("**\n**Queue Operations");

System.out.println("1. insert");

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

System.out.println("3. peek");

System.out.println("4. check empty");

System.out.println("5. size");

**int** choice = scan.nextInt();

**switch** (choice)

{

**case** 1 :

System.out.println("Enter integer element to insert");

lq.insert( scan.nextInt() );

**break**;

**case** 2 :

**try**

{

System.out.println("Removed Element = "+ lq.remove());

}

**catch** (Exception e)

{

System.out.println("Error : " + e.getMessage());

}

**break**;

**case** 3 :

**try**

{

System.out.println("Peek Element = "+ lq.peek());

}

**catch** (Exception e)

{

System.out.println("Error : " + e.getMessage());

}

**break**;

**case** 4 :

System.out.println("Empty status = "+ lq.isEmpty());

**break**;

**case** 5 :

System.out.println("Size = "+ lq.getSize());

**break**;

**default** :

System.out.println("Wrong Entry **\n** ");

**break**;

}

*/\* display queue \*/*

lq.display();

System.out.println("**\n**Do you want to continue (Type y or n) **\n**");

ch = scan.next().charAt(0);

} **while** (ch == 'Y'|| ch == 'y');

}

}

Linked Queue Test

Queue Operations

1. insert

2. remove

3. peek

4. check empty

5. **size**

4

Empty status = **true**

Queue = Empty

Do you want to **continue** **(**Type y or n**)**

y

Queue Operations

1. insert

2. remove

3. peek

4. check empty

5. **size**

1

Enter integer element to insert

24

Queue = 24

Do you want to **continue** **(**Type y or n**)**

y

Queue Operations

1. insert

2. remove

3. peek

4. check empty

5. **size**

1

Enter integer element to insert

5

Queue = 24 5

Do you want to **continue** **(**Type y or n**)**

y

Queue Operations

1. insert

2. remove

3. peek

4. check empty

5. **size**

1

Enter integer element to insert

9

Queue = 24 5 9

Do you want to **continue** **(**Type y or n**)**

y

Queue Operations

1. insert

2. remove

3. peek

4. check empty

5. **size**

1

Enter integer element to insert

72

Queue = 24 5 9 72

Do you want to **continue** **(**Type y or n**)**

y

Queue Operations

1. insert

2. remove

3. peek

4. check empty

5. **size**

1

Enter integer element to insert

14

Queue = 24 5 9 72 14

Do you want to **continue** **(**Type y or n**)**

y

Queue Operations

1. insert

2. remove

3. peek

4. check empty

5. **size**

1

Enter integer element to insert

1

Queue = 24 5 9 72 14 1

Do you want to **continue** **(**Type y or n**)**

y

Queue Operations

1. insert

2. remove

3. peek

4. check empty

5. **size**

5

Size = 6

Queue = 24 5 9 72 14 1

Do you want to **continue** **(**Type y or n**)**

y

Queue Operations

1. insert

2. remove

3. peek

4. check empty

5. **size**

3

Peek Element = 24

Queue = 24 5 9 72 14 1

Do you want to **continue** **(**Type y or n**)**

y

Queue Operations

1. insert

2. remove

3. peek

4. check empty

5. **size**

2

Removed Element = 24

Queue = 5 9 72 14 1

Do you want to **continue** **(**Type y or n**)**

y

Queue Operations

1. insert

2. remove

3. peek

4. check empty

5. **size**

2

Removed Element = 5

Queue = 9 72 14 1

Do you want to **continue** **(**Type y or n**)**

y

Queue Operations

1. insert

2. remove

3. peek

4. check empty

5. **size**

2

Removed Element = 9

Queue = 72 14 1

Do you want to **continue** **(**Type y or n**)**

y

Queue Operations

1. insert

2. remove

3. peek

4. check empty

5. **size**

5

Size = 3

Queue = 72 14 1

Do you want to **continue** **(**Type y or n**)**

y

Queue Operations

1. insert

2. remove

3. peek

4. check empty

5. **size**

3

Peek Element = 72

Queue = 72 14 1

Do you want to **continue** **(**Type y or n**)**

y

Queue Operations

1. insert

2. remove

3. peek

4. check empty

5. **size**

2

Removed Element = 72

Queue = 14 1

Do you want to **continue** **(**Type y or n**)**

y

Queue Operations

1. insert

2. remove

3. peek

4. check empty

5. **size**

2

Removed Element = 14

Queue = 1

Do you want to **continue** **(**Type y or n**)**

y

Queue Operations

1. insert

2. remove

3. peek

4. check empty

5. **size**

3

Peek Element = 1

Queue = 1

Do you want to **continue** **(**Type y or n**)**

y

Queue Operations

1. insert

2. remove

3. peek

4. check empty

5. **size**

2

Removed Element = 1

Queue = Empty

Do you want to **continue** **(**Type y or n**)**

y

Queue Operations

1. insert

2. remove

3. peek

4. check empty

5. **size**

2

Error : Underflow Exception

Queue = Empty

Do you want to **continue** **(**Type y or n**)**

y

Queue Operations

1. insert

2. remove

3. peek

4. check empty

5. **size**

4

Empty status = **true**

Queue = Empty

Do you want to **continue** **(**Type y or n**)**

n

Implement Queue Using Stacks

## Discuss how to implement queue using stack.

A queue can be implemented by using 2 stacks:-  
1. An element is inserted in the queue by pushing it into stack 1  
2. An element is extracted from the queue by popping it from the stack 2  
3. If the stack 2 is empty then all elements currently in stack 1 are transferred to stack 2 but in the reverse order  
4. If the stack 2 is not empty just pop the value from stack 2.

BY [SJ](http://algorithms.tutorialhorizon.com/author/sumitjain/) · JANUARY 31, 2015

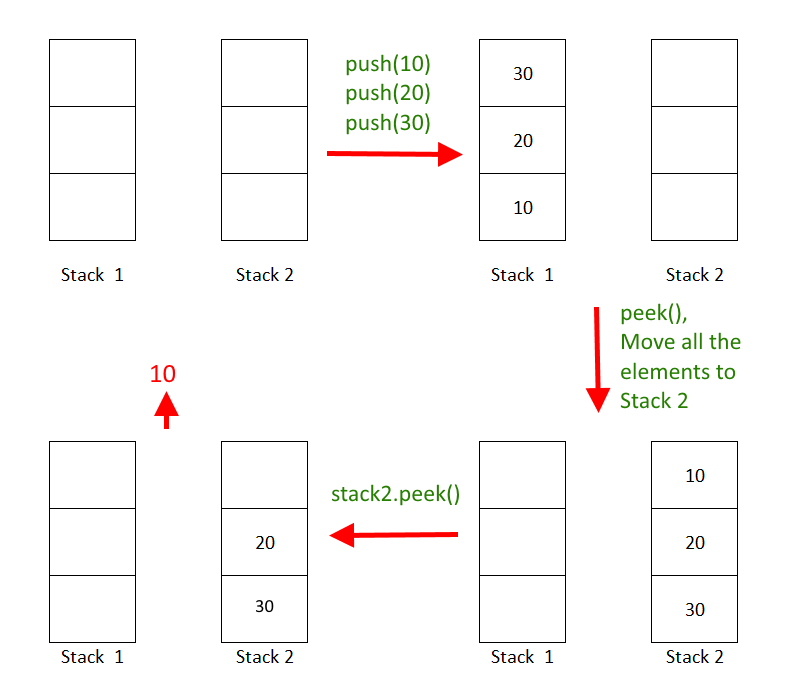
**Objec­tive:**We know that ***Queue*** is ***FIFO (First-in-First-Out)*** and ***Stack*** is ***LIFO ( Last-in-First-Out).***

Here our objec­tive is to imple­ment queue using stacks.

**Approach:**

* Take 2 Stacks, stack1 and stack2.
* stack1 will be used a back of the Queue and stack2 will be used as front of the Queue.
* Push() oper­a­tion will be done on stack1, and peek() and pop() oper­a­tions will be done on stack2.
* When peek() and pop() are called, check is stack2 is empty, if yes then move all the ele­ments from stack1 and push them into stack2.

**Exam­ple**:

[](http://algorithms.tutorialhorizon.com/files/2015/01/Implement-Queue-Using-Stacks-1.png)

*Implement-Queue-Using-Stacks*

**Com­plete Code**:

|  |  |
| --- | --- |
|  | import java.util.Stack; |
|  |  |
|  | public class QueueUsingStacks { |
|  |  |
|  | Stack<Integer> stack1 = new Stack<>(); //act as back of the Queue |
|  | Stack<Integer> stack2 = new Stack<>(); // act as the front of the Queue |
|  |  |
|  | public void push(int x) { // push into stack 1 |
|  | stack1.push(x); |
|  | } |
|  |  |
|  | public int peek() { |
|  | if (stack2.isEmpty()) { |
|  | moveItems(stack1, stack2); |
|  | } |
|  | return stack2.peek(); // return the top element in stack2 |
|  | } |
|  |  |
|  | public int pop() { |
|  | if (stack2.isEmpty()) { |
|  | moveItems(stack1, stack2); |
|  | } |
|  | return stack2.pop(); // return the top element in stack2 |
|  | } |
|  |  |
|  | public void moveItems(Stack<Integer> s1, Stack<Integer> s2) { |
|  | while (!stack1.isEmpty()) { |
|  | s2.push(s1.pop()); // move all the elements from stack 1 to stack 2 |
|  | } |
|  | } |
|  |  |
|  | public static void main(String[] args) { |
|  | QueueUsingStacks q = new QueueUsingStacks(); |
|  | q.push(10); |
|  | q.push(20); |
|  | q.push(30); |
|  | System.out.println("POP from Queue " + q.pop()); |
|  |  |
|  | } |
|  |  |
|  | } |

**[view raw](https://gist.github.com/thmain/240988ef0e0e1d151a04/raw/57d19c2d5b3793680ed149abdee4d4d84624c98a/QueueUsingStacks.java)[QueueUsingStacks.java](https://gist.github.com/thmain/240988ef0e0e1d151a04" \l "file-queueusingstacks-java)** hosted with  by **[GitHub](https://github.com/)**

**Out­put**:

POP from Queue 10