



Week 13: Queues

4009 DATA STRUCTURES & ALGORITHM (DSA)

Learning Outcomes

- ❖ Create, manipulate and understand the data structure of Queues
- ❖ Define and understand the queue data structure
- ❖ Manipulate queues coding in Java and use of Java's inbuilt methods
- ❖ Understand and implement a generic stack code using Linked List in Java
- ❖ Understand and implement basic non generic stacks using Linked List in Java

Content Overview

- ❖ Queues Introduction
- ❖ The queue abstract data type
- ❖ Example of a Queue data structure operation or method manipulation
- ❖ The `java.util.Queue` interface in Java
- ❖ Singly Linked List-based Queue implementation
- ❖ Implementing a queue using a generic linked-list

Data Structure Trade-offs

Data structures	Advantages	Disadvantages
Array	Quick insertion, very fast access to index known	Slow search, slow deletion, fixed size
Ordered Array	Quicker Search than unsorted array	Slow insertion and deletion, fixed size
Stack	Provide Last-in First-out access	Slow access to other items
Queue	Provide First-in First out access	Slow access to other items
Linked List	Quick Insertion, Quick deletion	Slow search
Binary Tree	Quick search, insertion, deletion (if tree balanced)	Deletion algorithm is complex
Hash Table	Very fast access if key known, fast insertion	Slow deletion, access slow if key not known, inefficient memory usage
Heap	Fast insertion, deletion, access to large item	Slow access to other items
Graphs	Model real world situations	Some algorithms are slow and complex

Queues introduction

- Queues data structure is a close “cousin” of the stack
- A queue is a collection of objects that are inserted and removed according to the first-in, first-out (FIFO) principle. That is, elements can be inserted at any time, but only the element that has been in the queue the longest can be next removed. Therefore, an element enters a queue at the back and are removed from the front.
- Think of the metaphor for this terminology is a line of people waiting to get on an ATM line. People waiting to use the ATM join or enter at the back of the line and use the ATM at the front of the line.

Queues introduction

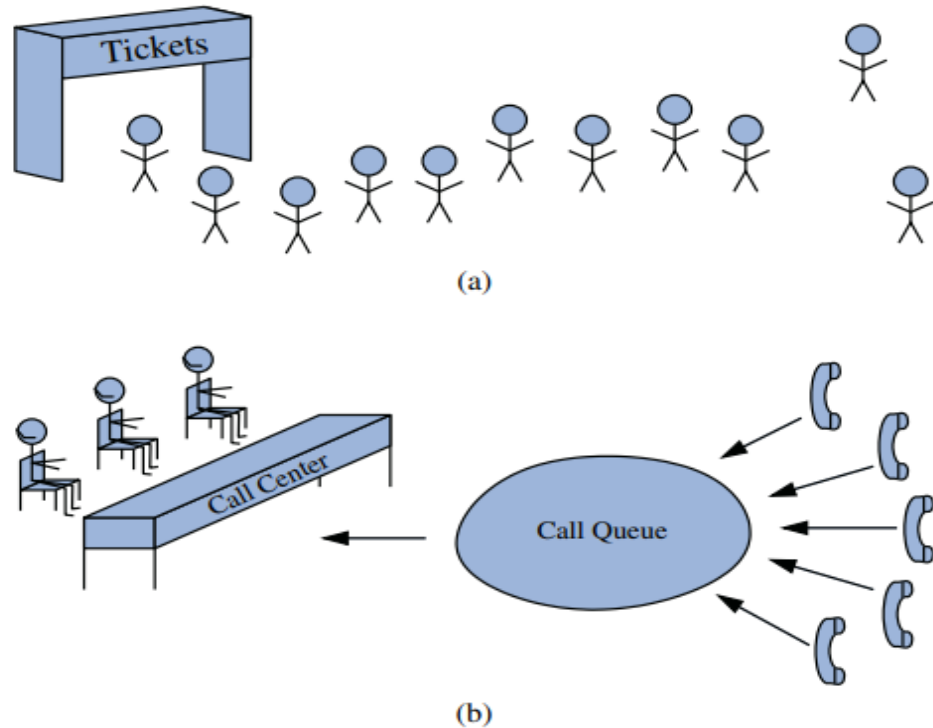


Figure 6.4: Real-world examples of a first-in, first-out queue. (a) People waiting in line to purchase tickets; (b) phone calls being routed to a customer service center.

- A queue would therefore be a logical choice for a data structure to handle calls to a customer service center, or a wait-list at a restaurant. FIFO queues are also used by many computing devices, such as a networked printer, or a Web server responding to requests.

Application of Queues in the real world

- Using of task scheduling use queues to prior and order to receive tasks and complete them.
- Operation system use queues to manage processes, requests and resources.
- Computer networks use queues in networking protocols like TCP to manage packets transmitted over the network. Ensure delivery order and appropriate rate of network communication.

Basic operations of queues

1. **enqueue:** Adds a new element to the back of the queue
2. **dequeue:** Removes and returns the front element from the queue
3. **first:** returns the first element of the queue
4. **isEmpty:** Checks if the queue is empty.
5. **Size:** Finds the number of elements in the queue.

The queue abstract data type (ADT) (1/2)

- The queue abstract data type defines a collection that keeps objects in a sequence, where element access and deletion are restricted to the first element in the queue, and element insertion is restricted to the back of the sequence. This restriction enforces the rule that items are inserted and deleted in a queue according to the first-in, first-out (FIFO) principle.
- The queue abstract data type (ADT) supports the following two update methods:
 - `enqueue(e)`: Adds element `e` to the back of queue.
 - `dequeue()`: Removes and returns the first element from the queue (or null if the queue is empty).
- The queue ADT also includes the following accessor methods (with `first` being analogous to the stack's `top` method):
 - `first()`: Returns the first element of the queue, without removing it (or null if the queue is empty).
 - `size()`: Returns the number of elements in the queue.
 - `isEmpty()`: Returns a boolean indicating whether the queue is empty. By convention, we assume that elements added to the queue can have arbitrary type and that a newly created queue is empty. We formalize the queue ADT with the Java interface shown in Code Fragment 6.9.

The queue abstract data type (ADT) (2/2)

```
1  public interface Queue<E> {  
2      /** Returns the number of elements in the queue. */  
3      int size();  
4      /** Tests whether the queue is empty. */  
5      boolean isEmpty();  
6      /** Inserts an element at the rear of the queue. */  
7      void enqueue(E e);  
8      /** Returns, but does not remove, the first element of the queue (null if empty). */  
9      E first();  
10     /** Removes and returns the first element of the queue (null if empty). */  
11     E dequeue();  
12 }
```

Code Fragment 6.9: A Queue interface defining the queue ADT, with a standard FIFO protocol for insertions and removals.

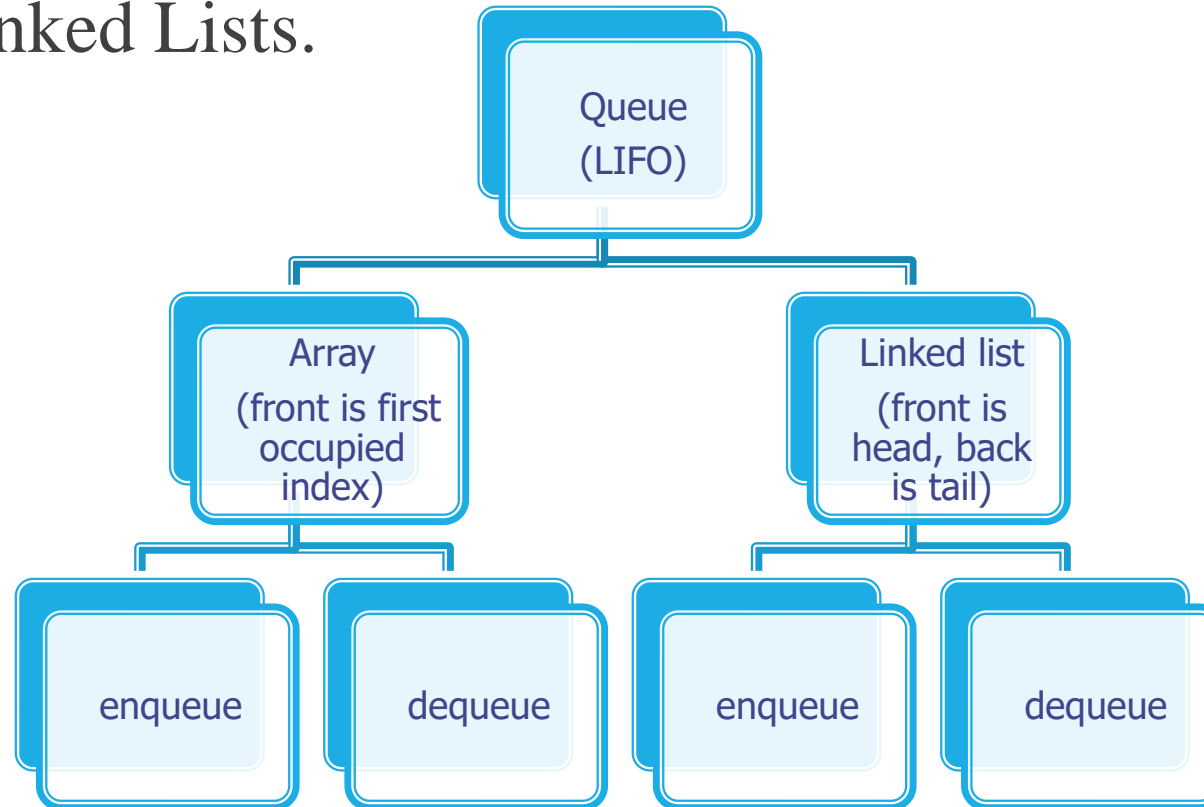
Example of a Queue data structure operation or method manipulation

Example 6.4: *The following table shows a series of queue operations and their effects on an initially empty queue Q of integers.*

Method	Return Value	first $\leftarrow Q \leftarrow$ last
enqueue(5)	—	(5)
enqueue(3)	—	(5, 3)
size()	2	(5, 3)
dequeue()	5	(3)
isEmpty()	false	(3)
dequeue()	3	()
isEmpty()	true	()
dequeue()	null	()
enqueue(7)	—	(7)
enqueue(9)	—	(7, 9)
first()	7	(7, 9)
enqueue(4)	—	(7, 9, 4)

Queues (LIFO) implemented through Array and Linked List

The use of queues [First-in-first-out (FIFO)] are implemented through Arrays and Linked Lists.



Singly Linked List-Based Queue Implementation

Similar to Stacks, arrays and linked list can be implemented using queue data structure.

To implement, we can easily adapt a singly linked list to implement the queue ADT while supporting limited capacity and operation problems. Here, we align the front of the queue with the front of the linked list and back of the queue with the tail of the linked list, because the only update operation that singly linked list supports at the back end is an insertion.

The following code supports this implementation.

```
1 import java.util.LinkedList;
2 import java.util.Queue;
3
4 public class Week9Lab1 {
5
6     public static void main(String[] args) {
7         // Create a Queue object called itiCourses
8         Queue<String> itiCourses = new LinkedList<>();
9
10        // Enqueue elements into the queue
11        itiCourses.add("DIT");
12        itiCourses.add("DHRM");
13        itiCourses.add("DACC");
14        itiCourses.add("DICT");
15
16        // Peek or see the front element of the queue
17        System.out.println("Front element: " + itiCourses.peek());
18
19        // If queue is empty, print message; else print elements in the queue
20        if(itiCourses.isEmpty()) {
21            System.out.println("Queue is empty");
22        } else {
23            // Print the queue (it will display elements in FIFO order)
24            System.out.println("Queue contents: " + itiCourses);
25        }
26    }
27 }
```

Stack Vs Queue

```
1 /* DSA Week 8 Lab 1 */
2
3 import java.util.Stack;
4
5 public class Week8Lab1 {
6
7     public static void main(String[] args) {
8         // Create a stack object called itiCourses
9         Stack<String> itiCourses = new Stack<>();
10
11         // Push elements onto the stack
12         itiCourses.push("DIT");
13         itiCourses.push("DHRM");
14         itiCourses.push("DACC");
15         itiCourses.push("DICT");
16
17         // Peek or see the top element of the stack
18         System.out.println("Top element: " + itiCourses.peek());
19
20         //if stack is empty print message stack is empty, else print elements in the stack
21         if(itiCourses.isEmpty()){
22             System.out.print("Stack is empty");
23         }else{
24             // Print the stack (it will display elements in LIFO order)
25             System.out.print("Stack contents: " + itiCourses);
26         }
27     }
28 }
```

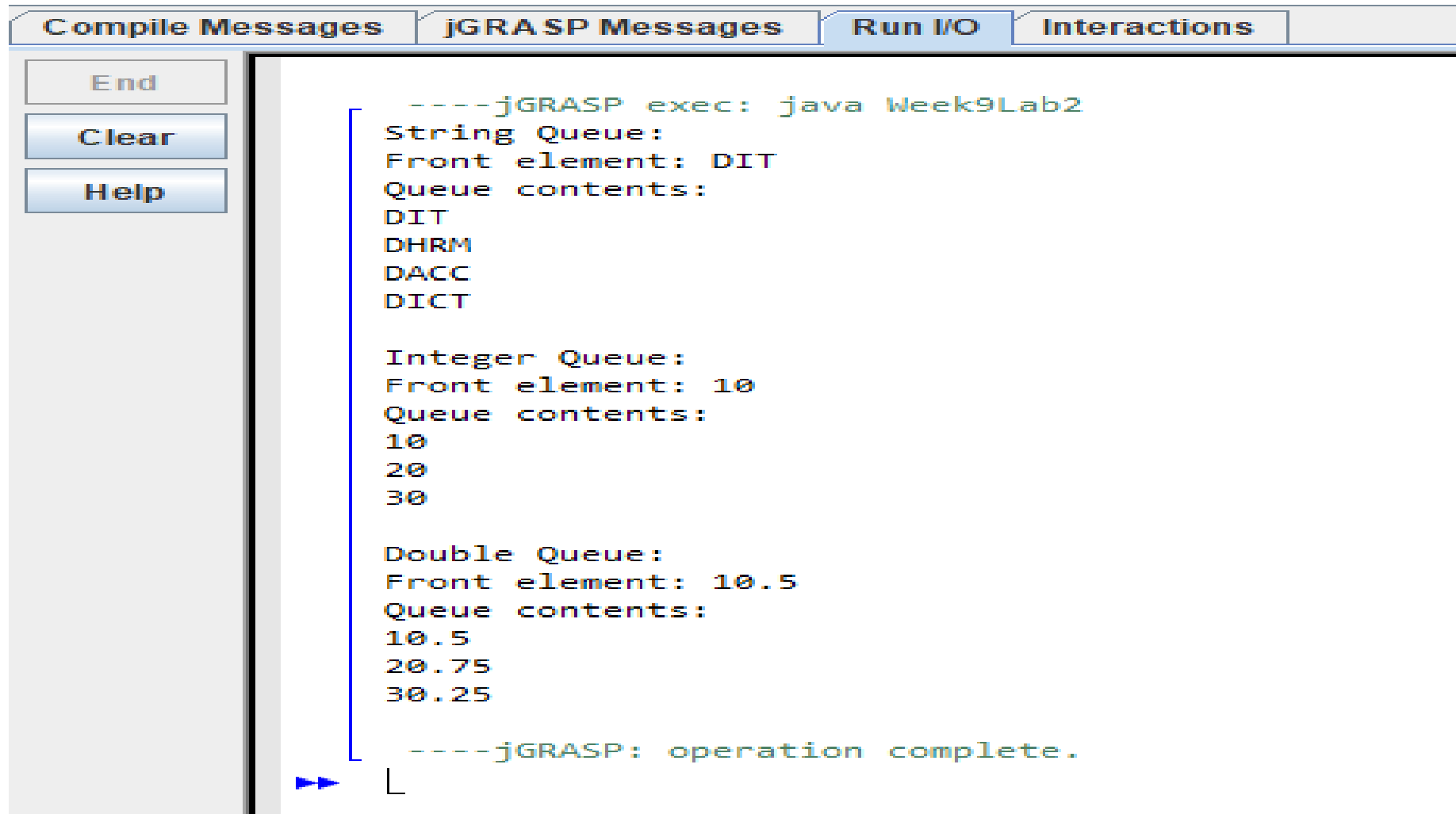
```
1 import java.util.LinkedList;
2 import java.util.Queue;
3
4 public class Week9Lab1 {
5
6     public static void main(String[] args) {
7         // Create a Queue object called itiCourses
8         Queue<String> itiCourses = new LinkedList<>();
9
10        // Enqueue elements into the queue
11        itiCourses.add("DIT");
12        itiCourses.add("DHRM");
13        itiCourses.add("DACC");
14        itiCourses.add("DICT");
15
16        // Peek or see the front element of the queue
17        System.out.println("Front element: " + itiCourses.peek());
18
19        // If queue is empty, print message; else print elements in the queue
20        if(itiCourses.isEmpty()) {
21            System.out.println("Queue is empty");
22        } else {
23            // Print the queue (it will display elements in FIFO order)
24            System.out.println("Queue contents: " + itiCourses);
25        }
26    }
27 }
```

Implementing a queue using a generic Linked List (1/2)

```
1 import java.util.LinkedList;
2 import java.util.Queue;
3
4 public class Week9Lab2<E> {
5
6     private Queue<E> queue; // Change Stack to Queue
7
8     // Constructor
9     public Week9Lab2() {
10         this.queue = new LinkedList<>();
11     }
12
13     // Enqueue method (add elements to the queue)
14     public void enqueue(E item) {
15         queue.add(item);
16     }
17
18     // Dequeue method (remove elements from the front of the queue)
19     public E dequeue() {
20         if (!queue.isEmpty()) {
21             return queue.poll();
22         } else {
23             System.out.println("Queue is empty.");
24             return null;
25         }
26     }
27
28     // Peek method (check front element)
29     public E peek() {
30         if (!queue.isEmpty()) {
31             return queue.peek();
32         } else {
33             System.out.println("Queue is empty.");
34             return null;
35         }
36     }
37
38     // Check if queue is empty
39     public boolean isEmpty() {
40         return queue.isEmpty();
41     }
42 }
```

```
43 // Print queue elements
44 public void printQueue() {
45     if (queue.isEmpty()) {
46         System.out.println("Queue is empty.");
47     } else {
48         System.out.println("Queue contents:");
49         for (E item : queue) {
50             System.out.println(item);
51         }
52     }
53 }
54
55 public static void main(String[] args) {
56     // Create a generic queue for Strings
57     Week9Lab2<String> stringQueue = new Week9Lab2<>();
58     stringQueue.enqueue("DIT");
59     stringQueue.enqueue("DHRM");
60     stringQueue.enqueue("DACC");
61     stringQueue.enqueue("DICT");
62
63     System.out.println("String Queue:");
64     System.out.println("Front element: " + stringQueue.peek());
65     stringQueue.printQueue();
66
67     // Create a generic queue for Integers
68     Week9Lab2<Integer> intQueue = new Week9Lab2<>();
69     intQueue.enqueue(10);
70     intQueue.enqueue(20);
71     intQueue.enqueue(30);
72
73     System.out.println("\nInteger Queue:");
74     System.out.println("Front element: " + intQueue.peek());
75     intQueue.printQueue();
76
77     // Create a generic queue for Doubles
78     Week9Lab2<Double> doubleQueue = new Week9Lab2<>();
79     doubleQueue.enqueue(10.5);
80     doubleQueue.enqueue(20.75);
81     doubleQueue.enqueue(30.25);
82
83     System.out.println("\nDouble Queue:");
84     System.out.println("Front element: " + doubleQueue.peek());
85     doubleQueue.printQueue();
86 }
87 }
```

Implementing a queue using a generic Linked List (2/2)



The screenshot shows a Java IDE window with four tabs: "Compile Messages", "jGRASP Messages", "Run I/O", and "Interactions". The "Run I/O" tab is active, displaying the output of a Java program. On the left side of the IDE, there are three buttons: "End", "Clear", and "Help". The output text is as follows:

```
----jGRASP exec: java Week9Lab2
String Queue:
Front element: DIT
Queue contents:
DIT
DHRM
DACC
DICT

Integer Queue:
Front element: 10
Queue contents:
10
20
30

Double Queue:
Front element: 10.5
Queue contents:
10.5
20.75
30.25

----jGRASP: operation complete.
```

A blue arrow points to the end of the output text.

Figure 1: Program output

Generic queue code (syntax) Stacks Vs Queue (1/2)

```
1 import java.util.LinkedList;
2 import java.util.Queue;
3
4 public class Week9Lab2<E> {
5
6     private Queue<E> queue; // Change Stack to Queue
7
8     // Constructor
9     public Week9Lab2() {
10         this.queue = new LinkedList<>();
11     }
12
13     // Enqueue method (add elements to the queue)
14     public void enqueue(E item) {
15         queue.add(item);
16     }
17
18     // Dequeue method (remove elements from the front of the queue)
19     public E dequeue() {
20         if (!queue.isEmpty()) {
21             return queue.poll();
22         } else {
23             System.out.println("Queue is empty.");
24             return null;
25         }
26     }
27
28     // Peek method (check front element)
29     public E peek() {
30         if (!queue.isEmpty()) {
31             return queue.peek();
32         } else {
33             System.out.println("Queue is empty.");
34             return null;
35         }
36     }
37
38     // Check if queue is empty
39     public boolean isEmpty() {
40         return queue.isEmpty();
41     }
42 }
```

```
1 import java.util.Stack;
2
3 public class GenericStackExample<E> {
4
5     private Stack<E> stack;
6
7     // Constructor
8     public GenericStackExample() {
9         this.stack = new Stack<>();
10    }
11
12    // Push method
13    public void push(E item) {
14        stack.push(item);
15    }
16
17    // Peek method
18    public E peek() {
19        if (!stack.isEmpty()) {
20            return stack.peek();
21        } else {
22            System.out.println("Stack is empty.");
23            return null;
24        }
25    }
26
27    // Check if stack is empty
28    public boolean isEmpty() {
29        return stack.isEmpty();
30    }
31
32    // Print stack elements
33    public void printStack() {
34        if (stack.isEmpty()) {
35            System.out.println("Stack is empty.");
36        } else {
37            System.out.println("Stack contents:");
38            for (E item : stack) {
39                System.out.println(item);
40            }
41        }
42    }
43 }
```

Generic queue code (syntax) Stacks Vs Queue (2/2)

```
43 // Print queue elements
44 public void printQueue() {
45     if (queue.isEmpty()) {
46         System.out.println("Queue is empty.");
47     } else {
48         System.out.println("Queue contents:");
49         for (E item : queue) {
50             System.out.println(item);
51         }
52     }
53 }
54
55 public static void main(String[] args) {
56     // Create a generic queue for Strings
57     Week9Lab2<String> stringQueue = new Week9Lab2<>();
58     stringQueue.enqueue("DIT");
59     stringQueue.enqueue("DHRM");
60     stringQueue.enqueue("DACC");
61     stringQueue.enqueue("DICT");
62
63     System.out.println("String Queue:");
64     System.out.println("Front element: " + stringQueue.peek());
65     stringQueue.printQueue();
66
67     // Create a generic queue for Integers
68     Week9Lab2<Integer> intQueue = new Week9Lab2<>();
69     intQueue.enqueue(10);
70     intQueue.enqueue(20);
71     intQueue.enqueue(30);
72
73     System.out.println("\nInteger Queue:");
74     System.out.println("Front element: " + intQueue.peek());
75     intQueue.printQueue();
76
77     // Create a generic queue for Doubles
78     Week9Lab2<Double> doubleQueue = new Week9Lab2<>();
79     doubleQueue.enqueue(10.5);
80     doubleQueue.enqueue(20.75);
81     doubleQueue.enqueue(30.25);
82
83     System.out.println("\nDouble Queue:");
84     System.out.println("Front element: " + doubleQueue.peek());
85     doubleQueue.printQueue();
86 }
87 }
```

```
44 public static void main(String[] args) {
45     // Create a generic stack for Strings
46     GenericStackExample<String> stringStack = new GenericStackExample<>();
47     stringStack.push("DIT");
48     stringStack.push("DHRM");
49     stringStack.push("DACC");
50     stringStack.push("DICT");
51
52     System.out.println("String Stack:");
53     System.out.println("Top element: " + stringStack.peek());
54     stringStack.printStack();
55
56     // Create a generic stack for Integers
57     GenericStackExample<Integer> intStack = new GenericStackExample<>();
58     intStack.push(10);
59     intStack.push(20);
60     intStack.push(30);
61
62     System.out.println("\nInteger Stack:");
63     System.out.println("Top element: " + intStack.peek());
64     intStack.printStack();
65
66     // Create a generic stack for Doubles
67     GenericStackExample<Double> doubleStack = new GenericStackExample<>();
68     doubleStack.push(10.5);
69     doubleStack.push(20.75);
70     doubleStack.push(30.25);
71
72     System.out.println("\nDouble Stack:");
73     System.out.println("Top element: " + doubleStack.peek());
74     doubleStack.printStack();
75 }
76 }
```

Topic next week

Week 14 Lists and Iterators. Array lists, node lists & iterators

DSA Week 13 activities

Refer to print out materials.

- This week, you are required to complete the questions and two labs.
 - Refer to the print out, answer all week 13 questions.
 - Refer to the print out, complete the two labs activities using the lab computers.

Note: You can complete the activities in any order, however, make afford to complete and understand everything which prepares you for well for test 2 & Final Exam.