# CSCI 2120:
## Software Design & Development II

*UNIT2: Data management*

*Collections Framework & Generics*

**Queue**

# Overview

1. Introduction
2. Realtime Example of Queue
3. Hierarchy of Queue interface
4. Features of Queue interface
5. How to create a Queue in Java?
6. Methods of Queue interface
7. Queue Examples

# Introduction

A **Queue in Java** is a collection of elements that implements the “**First-In-First-Out**” order.

In simple words, a queue represents the arrangement of elements in the first in first out (FIFO) fashion.

That means an element that is stored as a first element into the queue will be removed first from the queue. That is, the first element is removed first and last element is removed at last.

Java Collection Framework added `Queue` interface in Java 5.0. It is present in `java.util.Queue`

# Realtime Example of Queue

1. The most common realtime example of Queue is a waiting line in the supermarket. The cashier services the person that is at the beginning of the line first. Other customers enter the line only at the other end and wait for service.

**A queue in the super market**

# Realtime Example of Queue

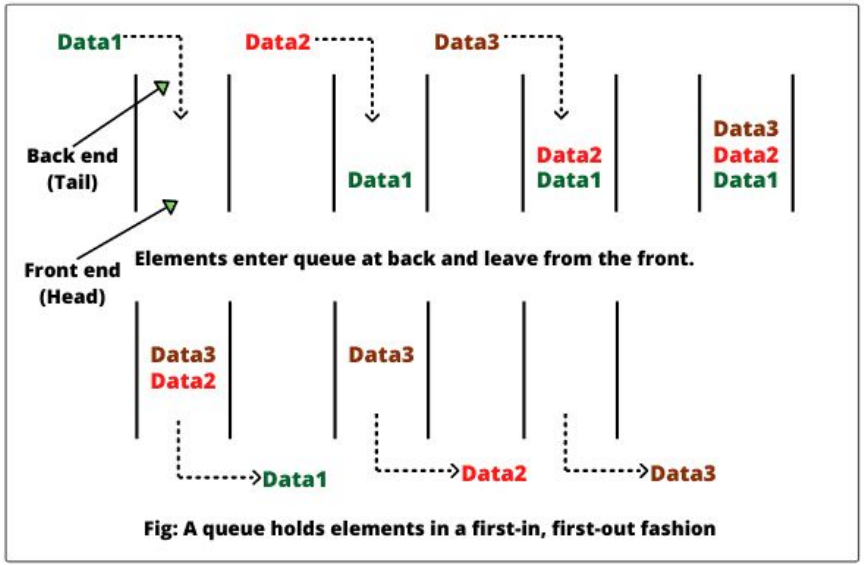

Fig: A queue holds elements in a first-in, first-out fashion

Similarly, a queue in java is a common data structure in which there are two ends: front (head) and back (tail) of the queue. Elements are always inserted into the end (tail) of the queue and are accessed and deleted from the beginning (head) of the queue, as shown in the below figure.

For this reason, a queue is a **first-in, first-out (FIFO) data structure**. The operations of inserting and removing elements are called **enqueue** and **dequeue**.

An element that is added to the end of the queue will remain on the queue until all the elements in front of it have been removed.

This process is similar to the “line” or “queue” of customers waiting for service. Customers are serviced in the order in which they arrive in the queue.

# Hierarchy of Queue Interface

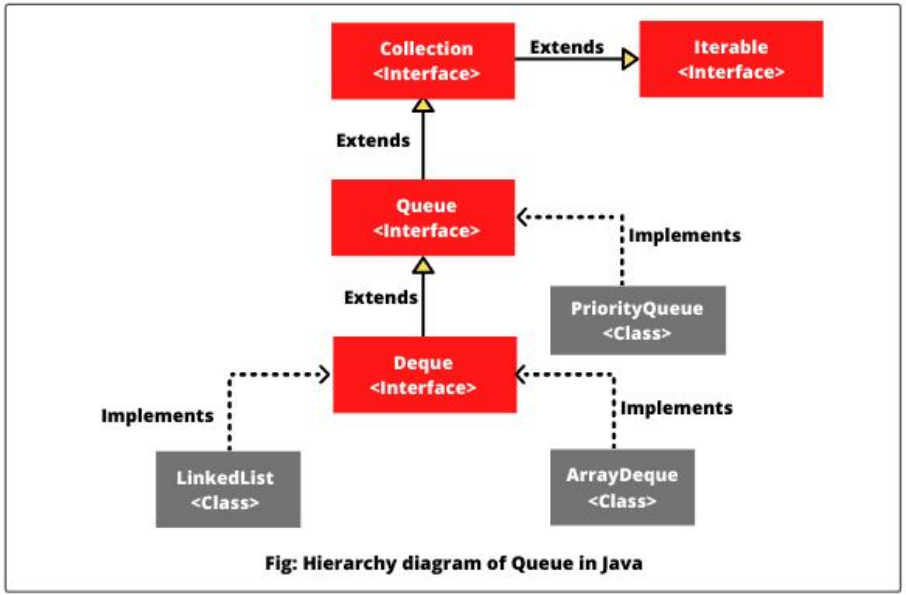

Fig: Hierarchy diagram of Queue in Java

In the Java Collections Framework, a queue is represented by java.util.queue interface. Java `Queue` interface extends the `Collection` interface. It is the super interface of `BlockingDeque<E>, BlockingQueue<E>, Deque<E>, TransferQueue<E>`.

A `Queue` interface is implemented by several classes such as `LinkedList, AbstractQueue, ArrayBlockingQueue, ArrayDeque, PriorityQueue.`

Out of these, `ArrayDeque` class is the best choice for simple FIFO queues. It provides all the normal functionalities of a queue.

# Queue interface declaration

Queue is a generic interface that is declared in a general form as below:

```
public interface Queue<E> extends Collection<E>
```

*In the above syntax, E represents the type of objects that the set will hold.*

# Features of Queue interface

There are several interesting features of the queue in Java that is as follows:

1. `Queue` interface orders elements in First In First Out policy.
2. Elements can be accessed and removed only from the front (head) of the queue.
3. Elements can be added only from the back (tail) of the queue.
4. Queue does not allow to add the null object.

# How to create a Queue in Java?

A `Queue` interface can be implemented in java by using either any one of four classes: `LinkedList class, AbstractQueue class, PriorityQueue class`, and `ArrayDeque` class. For simple FIFO queues, `LinkedList` and `ArrayDeque` classes are the best choices to create a queue.

When you need to create a queue, simply initialize a `Queue` variable with a `LinkedList` object as given below:

```
Queue<String> q = new LinkedList<>();

Queue<Integer> q = new ArrayDeque<>();
```

# Methods of Queue Interface

| Method | Description |
|---|---|
| boolean add(E e) | This method is used to insert the specified element into the queue if the space is available without violating capacity restrictions. On successfully added, it returns true. It throws an `IllegalStateException` if no space is currently available.<br><br>A `ClassCastException` is thrown when an object is not compatible with elements in the queue. A `NullPointerException` is thrown when we attempt to store a null object because null elements are not allowed in the queue. `IllegalArgumentException` is thrown when an invalid argument is used. `IllegalStateException` is thrown when we try to add an element to a fixed-length queue that is full. |
| E element() | It is used to retrieve the element at the head of queue, but the element is not removed from the head of the queue. It throws an exception named `NoSuchElementException` if the queue is empty. |

# Methods of Queue Interface

| Method | Description |
|---|---|
| boolean offer(E e) | It is used to insert the specified element e to the queue. This method returns true if e was added and false otherwise. |
| E peek() | The peek() method is used to retrieve the element at the head of queue, but the element is not removed. It returns null if the queue is empty. |
| E poll() | The poll() method is used to retrieve the element at the head of queue and removes the element in the process. This method returns null if this queue is empty. |
| E remove() | It is used to retrieve and remove the element at head of the queue. It throws an exception named NoSuchElementException if the queue is empty. |

# Methods of Queue Interface

**Methods inherited from Collection interface:**

`addAll, remove, removeAll, isEmpty, clear, contains, containsAll, equals, hashCode, iterator,  retainAll, size, toArray.`

# Methods of Queue Interface

**Note:**

1. The `poll()` and `remove()` methods are similar, except that `poll()` returns `null` object if the queue is empty, whereas `remove()` throws an exception named `NoSuchElementException`.

2. The `peek()` and `element()` methods are similar, except that `peek()` returns `null` object if the queue is empty, whereas `element()` throws an exception named `NoSuchElementException`.

3. The `offer()` method is used to insert an element to the queue. This method is similar to the `add()` method inherited from the `Collection` interface, but the `offer()` method is more preferred for queues.

# Queue Examples

Let’s take example programs to perform various operations based on the above methods defined by queue interface.

# Example 1: LinkedList, add, remove, offer elements

```
import java.util.LinkedList;
import java.util.Queue;
public class QueueTester1 {
   public static void main(String[] args) {
       // Create a Queue.
       Queue<String> q = new LinkedList<>();

       // Adds elements to the tail of queue.
       q.add("ABC");
       q.add("DEF");
       q.add("GHI");
       q.add("JKL");
       q.add("MNO");

       System.out.println("Elements in queue: " +q);
       System.out.println("Head element of queue: " +q.element());
       System.out.println("Removed element: " +q.remove());
       System.out.println("Elements in queue after removed: " +q);

       boolean addElement = q.offer("PQR");
       System.out.println("Is new element added to the tail of queue: " +addElement);
       System.out.println("Elements in queue after adding new element: " +q);
   }
}
```

# Example 1: LinkedList, add, remove, offer elements

**Output:**

```
Elements in queue: [ABC, DEF, GHI, JKL, MNO]
Head element of queue: ABC
Removed element: ABC
Elements in queue after removed: [DEF, GHI, JKL, MNO]
Is new element added to the tail of queue: true
Elements in queue after adding new element: [DEF, GHI, JKL, MNO, PQR]
```

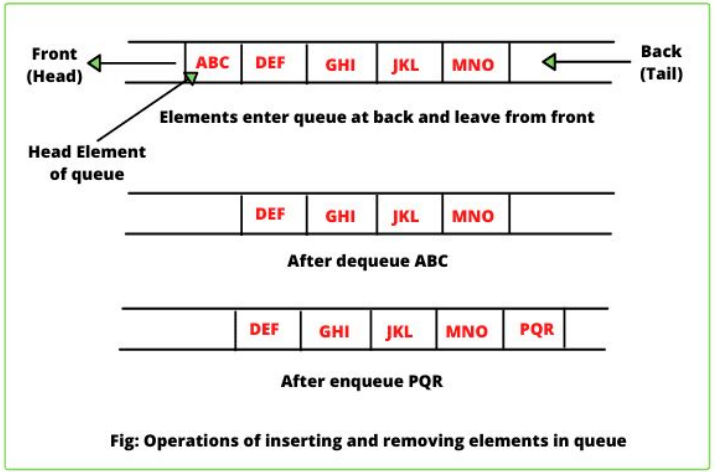

Fig: Operations of inserting and removing elements in queue

**Explanation:**

As you can see in the diagram, the adding operation takes place at one end of the queue called tail. This operation is known as enqueue. The removal operation takes place at the other end called head. This operation is known as dequeue.

# Example 2: LinkedList, peak, poll, remove

```
import java.util.LinkedList;
import java.util.Queue;
public class QueueTester2 {
   public static void main(String[] args) {
       Queue<Integer> q = new LinkedList<>();              // Create a Queue.
       boolean isEmpty = q.isEmpty();                      // Check queue is empty or not.
       System.out.println("Is queue empty: " +isEmpty);
       System.out.println("q.peak(): " +q.peek());         // Returns null because queue is empty.
       System.out.println("q.poll(): " +q.poll());         // Returns null because queue is empty.

       // Adds elements to the tail of queue.
       q.add(10);
       q.add(20);
       q.add(30);
       q.add(25);
       q.add(50);
       System.out.println("Size of queue: " +q.size());
       System.out.println("Original elements in queue: " +q);

       int head = q.remove();
       System.out.println("Removed element at the head of queue: " +head); // Removes the head of queue.
       System.out.println("Elements in queue: " +q);

       int peek = q.peek();
       System.out.println("Head element of queue: " +peek); // Retrieves the head of queue without removing.
       System.out.println("Elements in queue: " +q);
   }
}
```

# Example 2: LinkedList, peak, poll, remove

**Output:**

```
Is queue empty: true
q.peak(): null
q.poll(): null
Size of queue: 5
Original elements in queue: [10, 20, 30, 25, 50]
Removed element at the head of queue: 10
Elements in queue: [20, 30, 25, 50]
Head element of queue: 20
Elements in queue: [20, 30, 25, 50]
```

# Example 3: ArrayDeque

```
import java.util.ArrayDeque;
import java.util.Queue;
public class QueueTester3 {
   public static void main(String[] args) {
       Queue<Integer> q = new ArrayDeque<>();   // Create a Queue.
       q.offer(50);
       q.offer(50);
       q.offer(60);
       q.offer(20);
       q.offer(10);

       System.out.println(q);
       System.out.println("q.element(): " + q.element());

       System.out.println("q.remove(): " + q.remove());
       System.out.println(q);

       System.out.println("q.remove(): " + q.remove());
       System.out.println(q);

       System.out.println("q.offer(100): ");
       q.offer(100);
       System.out.println(q);

       System.out.println("q.remove(): " + q.remove());
       System.out.println(q);
   }
}
```

# Example 3: ArrayDeque

**Output:**

```
[50, 50, 60, 20, 10]
q.element(): 50
q.remove(): 50
[50, 60, 20, 10]
q.remove(): 50
[60, 20, 10]
q.offer(100):
[60, 20, 10, 100]
q.remove(): 60
[20, 10, 100]
```

# END