

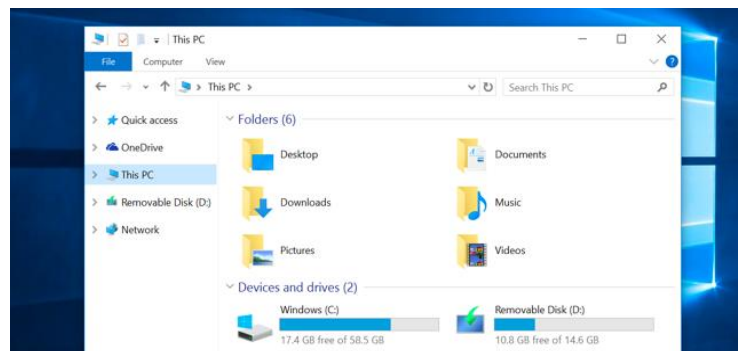
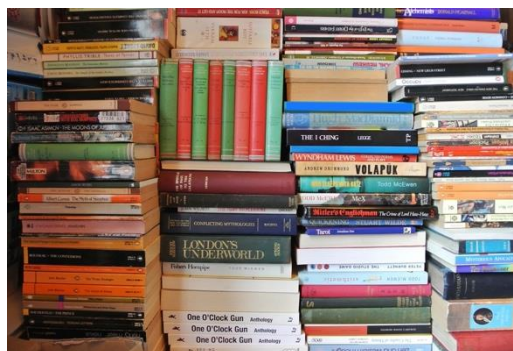


COM410 Programming in Practice

A5.2 Queues




Recall Examples of Data Organisation



- Queue – First in, first out – another common data organisation technique in everyday life

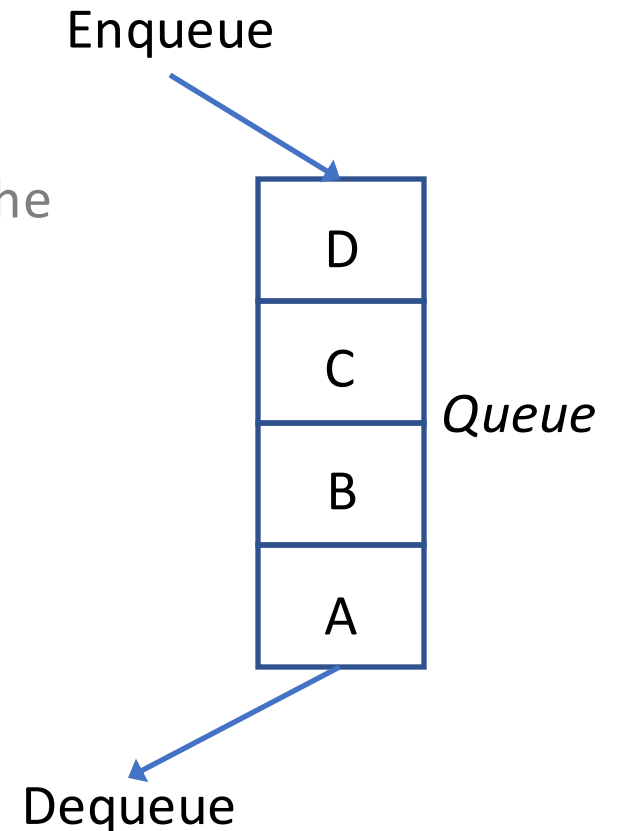
Queues

- A **queue** is another name for a waiting line
 - Used within operating systems and to simulate real-world events
 - Come into play whenever processes or events must wait
 - Entries organized first-in, first-out (chronologically)
- 
- A photograph showing a line of eight diverse people standing in a queue, separated by a red stanchion rope. They are standing in a single file line, waiting for service.
- Sometimes more flexibility is needed:
 - A double-ended queue permits operations on both oldest and newest entries (known as a **deque**)
 - When the importance of an object depends on criteria other than its arrival time, you can assign it a priority (known as a **priority queue**)

Queue Operations

- Like a stack the queue ADT organizes entries in the order in which they were added
- The behaviour of a queue is described as **FIFO** (First In, First Out)

- In the queue all additions are to the **back** of the queue (the entry at the back is the latest item added to the queue)
- The item that was added earliest is at the **front** of the queue
- The operation that adds an entry to the queue is traditionally called **enqueue**
- The operation that removes an entry from the queue is traditionally called **dequeue**



Queue Operations

- Like the stack, the queue also restricts access to its entries (can only look at or retrieve the earliest entry)
- In addition to enqueue and dequeue, the operation to retrieve the front entry without removing it is called **getFront**
- Typically you cannot search a queue for a specific entry
- The only way to look at an entry not at the front of the queue is to repeatedly remove items from the queue until the desired item reaches the front

Queue Operations

Interface for the Queue ADT

```
public interface QueueInterface<T> {  
    public void enqueue (T newEntry);  
    public T dequeue();  
    public T getFront();  
    public boolean empty();  
    public void clear();  
}
```

→ Add a new entry to the back

→ Remove entry from the front

→ Return entry from the front

→ Check for no entries in queue

→ Remove all entries

Java Interface for the Queue ADT

```
public interface QueueInterface<T> {
```

```
    public void add (T newEntry);
```

→ Add a new entry to the back

```
    public T remove();
```

→ Remove entry from the front

```
    public T peek();
```

→ Return entry from the front

```
    public boolean isEmpty();
```

→ Check for no entries in queue

```
    public void clear();
```

→ Remove all entries

```
}
```


Java Queue Class

- As for stack, Java provides a built-in **Queue** class in the library `java.util.Queue`

```
1  import java.util.LinkedList;
2  import java.util.Queue;
3
4  public class UsingQueues {
5
6      public static void main(String[] args) {
7
8          Queue<String> myQueue = new LinkedList<String>();
```

- We need to import the **Queue** object from the library `java.util` at the top of the class in which we want to use it.
- Note that Java provides **Queue** as an implementation of the **LinkedList** class, hence we also need to include **LinkedList** in the imports and the queue object calls the **LinkedList** constructor

Java Queue Class

- Using the `isEmpty()` method to check for an empty `Queue` and the `size()` method to report the number of elements contained
- Initially a newly-created `Queue` is empty.

```
10 System.out.println("\nShowing the initial queue");
11 System.out.println(myQueue);
12 if (myQueue.isEmpty()) System.out.println("Queue is empty");
13 else System.out.printf("Queue contains %d elements \n", myQueue.size());
```

```
Showing the initial queue
[]
Queue is empty
```

Java Queue Class

- Using the `add()` method (note – not `enqueue`) to add an element to the queue
- New elements are added to the back of the queue

```
15 System.out.println("\nAdding Adrian, Belle and Charles");
16 myQueue.add("Adrian");
17 myQueue.add("Belle");
18 myQueue.add("Charles");
19 System.out.println(myQueue);
20 if (myQueue.isEmpty()) System.out.println("Queue is empty");
21 else System.out.printf("Queue contains %d elements \n", myQueue.size());
```

```
Adding Adrian, Belle and Charles
[Adrian, Belle, Charles]
Queue contains 3 elements
```

Java Queue Class

- Using the `remove()` method (note – not `dequeue`) to remove an element from the queue
- Elements are removed from the front of the queue. The element that has been in the queue for the longest time is the first to be removed

```
23 System.out.println("\nRemoving from the front of the queue");
24 System.out.println(myQueue.remove());
25 System.out.println(myQueue);
26 if (myQueue.isEmpty()) System.out.println("Queue is empty");
27 else System.out.printf("Queue contains %d elements \n", myQueue.size());
```

```
Removing from the front of the queue
Adrian
[Belle, Charles]
Queue contains 2 elements
```

Java Queue Class

- Using the `peek()` method to return an element from the queue
- Elements are returned from the front of the queue but are NOT removed. The `peek()` operation does not change the queue contents

```
29 System.out.println("\nReading the element at the the front of the queue");
30 System.out.println(myQueue.peek());
31 System.out.println(myQueue);
32 if (myQueue.isEmpty()) System.out.println("Queue is empty");
33 else System.out.printf("Queue contains %d elements \n", myQueue.size());
```

```
Reading the element at the the front of the queue
Belle
[Belle, Charles]
Queue contains 2 elements
```

Java Queue Class

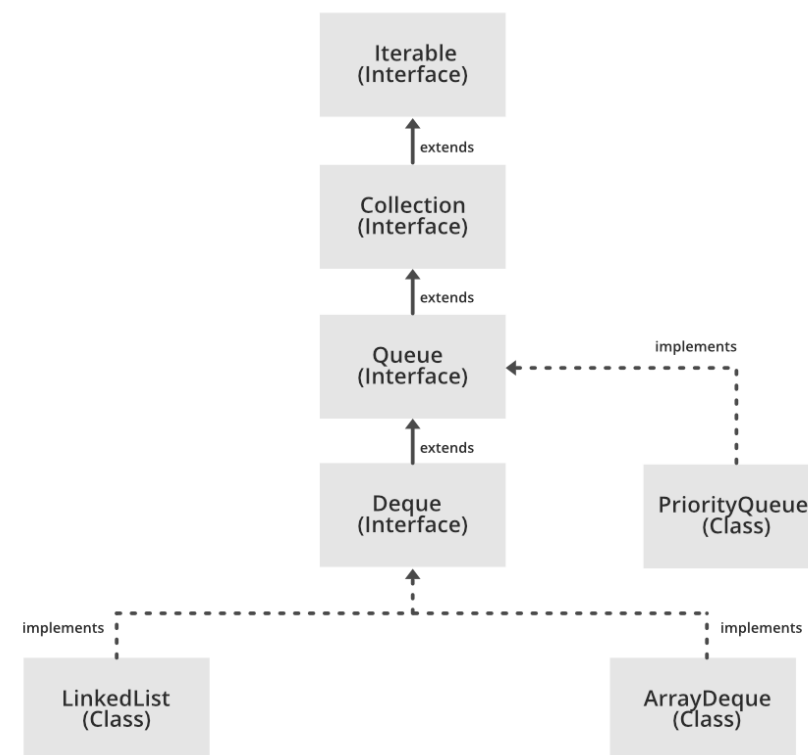
- Using the `clear()` method to empty the queue
- No elements are returned, but all are removed. The `clear()` operation returns the queue to its initial empty state

```
35 System.out.println("\nClearing the queue");
36 myQueue.clear();
37 System.out.println(myQueue);
38 if (myQueue.isEmpty()) System.out.println("Queue is empty");
39 else System.out.printf("Stack contains %d entries \n", myQueue.size());
```

```
Clearing the queue
[]
Queue is empty
```

Note on the Java Class Libraries

- In the Java Class Library, **Queue** is defined as an interface class. Since Interfaces cannot be instantiated directly, we also need to import a class that implements Queue - therefore class **LinkedList**
- Even though the traditional queue methods **add()**, **remove()** and **peek()** are the only means of adding, accessing, and removing data, the Java **Queue** class inherits a very wide range of methods from its parent classes – which include non-traditional possibilities such as searching, retrieving from the middle, etc.
- We will not use these inherited methods here.



Scenario

- Add a new class to your **DataStructures** project called **Queues** and implement an application to determine the first non-repeating character in a string as follows.
 - The user should be prompted to enter a string from the keyboard.
 - The individual characters should be retrieved one at a time, beginning from the 1st character in the string, and entered into a queue.
 - As the characters are being entered into the queue, a count should be maintained for the number of times each character has been seen.
 - Remove the characters one at a time from the queue, stopping when a character with a count of exactly one has been found, and reporting this to be the first non-repeating character.
 - If no non-repeating characters are found, report this to the user.