Topic 4: Queue

Topic 4: Queue
Learning Outcomes

At the end of this topic lesson, student should be able to:

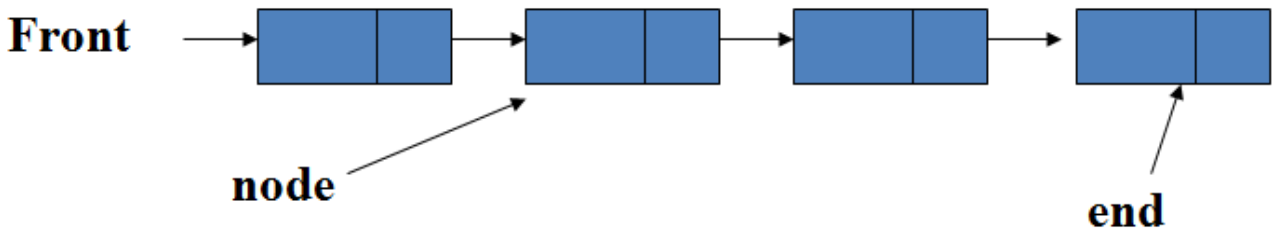
- understand basic concept of Queue
- applications of Queue
- use the ADT of Queue and its operations
- apply and implement the Queue

Topic 4: Queue

- Basic Queue Concept
- Suitable Type of Problems Requiring the use of Queue

Concept of Queue

- A queue is a linear data structure that operates on a **FIFO (First-In, First-Out)** concept.
- This means that the first element added to the queue is the first one to be removed.
- Elements are inserted at the rear/end and removed from the front of the queue.
- **Dynamic Size -** Queues can grow or shrink dynamically.
- In this study, the implementation of Queue will be using linked-list structured.



Applications of Queue

Customer Service – Managing customer calls, placed in a queue until the next available agent can assist them.

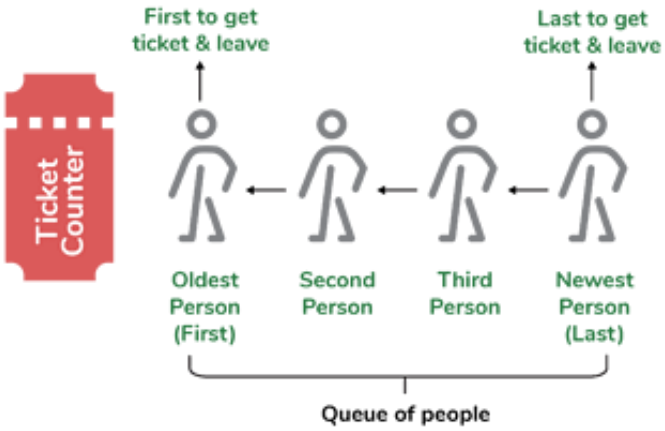
Waiting line at ticket counter - the person at the front is serviced first and then leaves, others join at the end and wait until they move to the front.

Operating Systems - Handling system processes and interrupts

Print Spooler - Managing print jobs in a queue.

Network Routing - Managing data packets in network queues

Web Servers - Handling incoming requests from clients.



Queue Implementation

- The implementation of `Queue` is done by creating composite object of `LinkedList` class.
- These operations (methods) of `LinkedList` are incorporated in `Queue`'s operation:

```
public void addLast(E e);
public E removeFirst();
public E getFirst();
public boolean isEmpty();
```

Queue Implementation

- Implementation of `Queue` class requires these operations:

Operation	Description
`public Queue()`	Default constructor
`public void enqueue (E e)`	Adds an element to the end of queue
`public E dequeue()`	Removes an element from front of queue
`public E getFront()`	Accessing the element at the front of the queue
`public boolean isEmpty()`	Return true if the queue has no element and return false otherwise. This method will be used to control loop for processing

Queue Definition

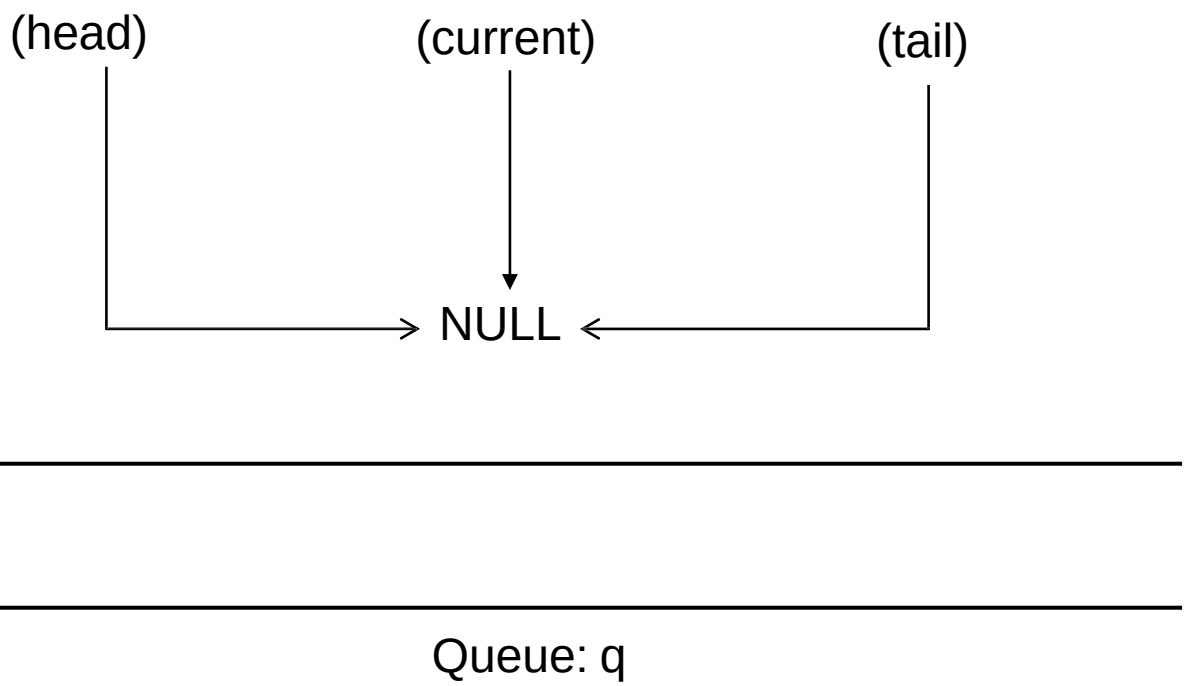
```
public class Queue<E> {
      private LinkedList <E> list;

      public Queue()
      {list = new LinkedList<E>()}

      public void enqueue(E data)
      {list.addLast(data);}

      public E dequeue()
      {return list.removeFirst();}

      public E getFront()
      {return list.getFirst();}

      public boolean isEmpty()
      {return list.isEmpty();}
}
```

Queue Example

Create a queue

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

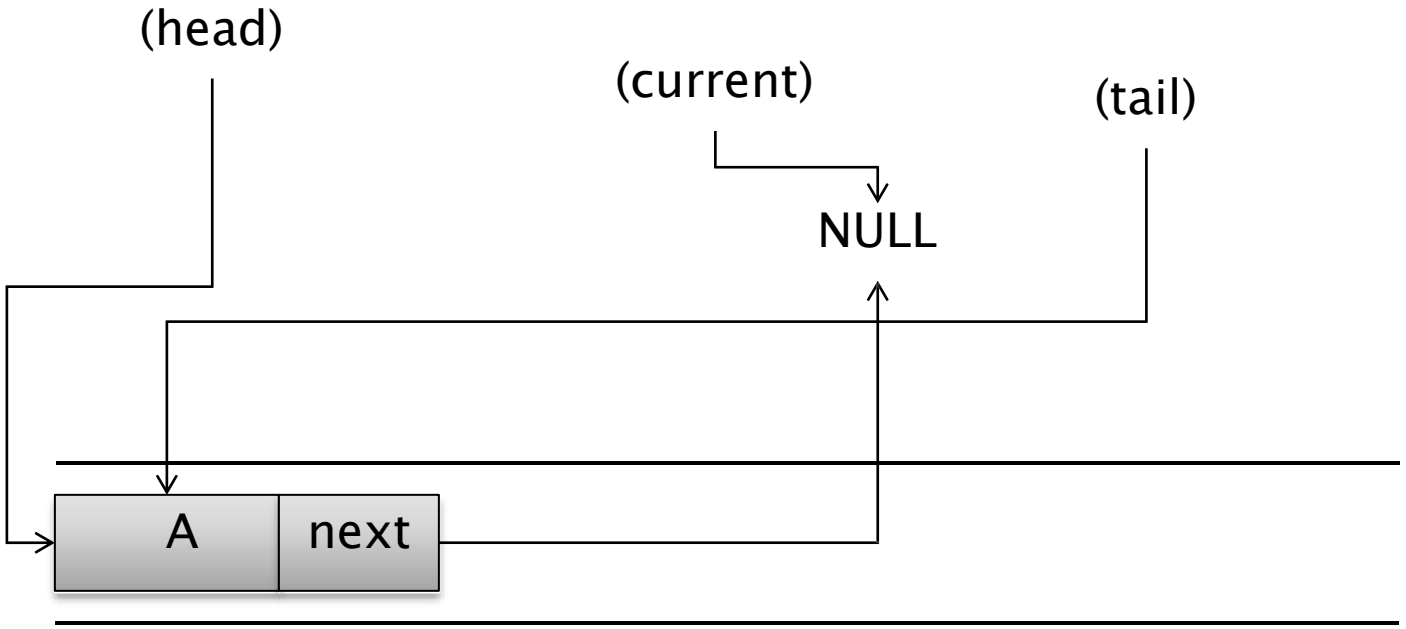
// this statement will create a Queue (extends of LinkedList) named q



Queue Example (cont.)

Add an element to the end of queue

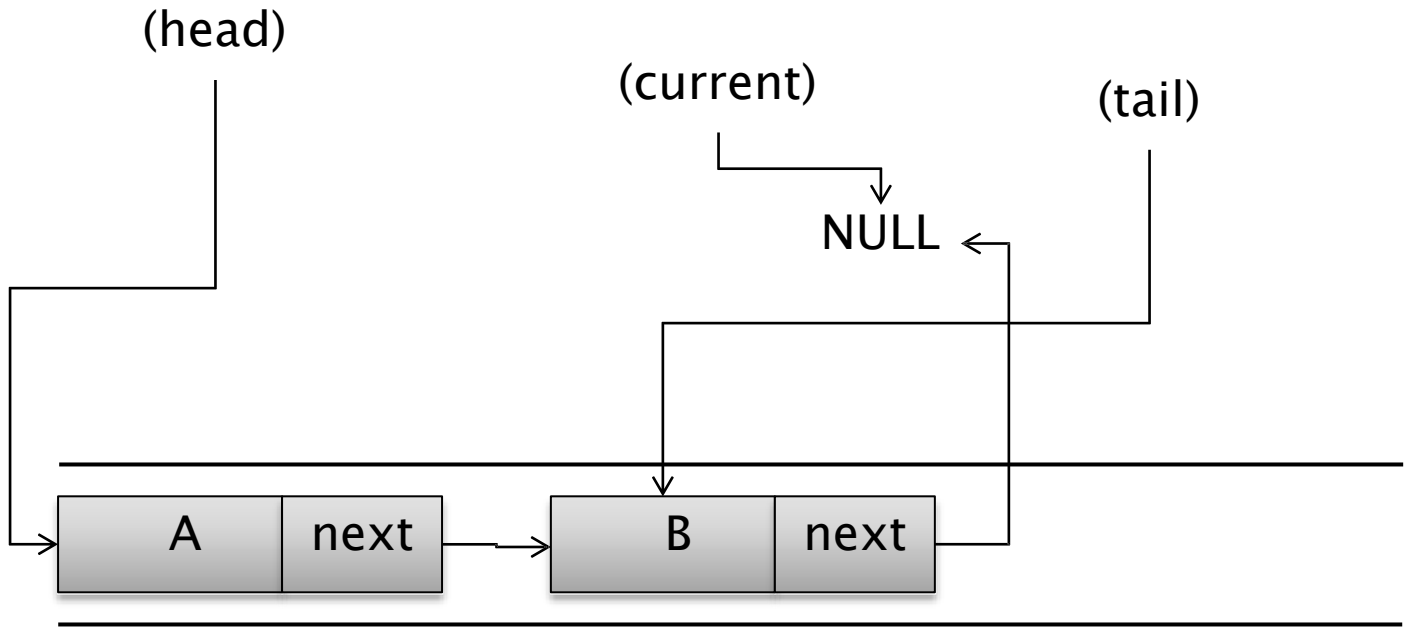
```
q.enqueue("A");
// this statement invokes addLast("A") of LinkedList
```



Queue Example (cont.)

Add an element to the end of queue

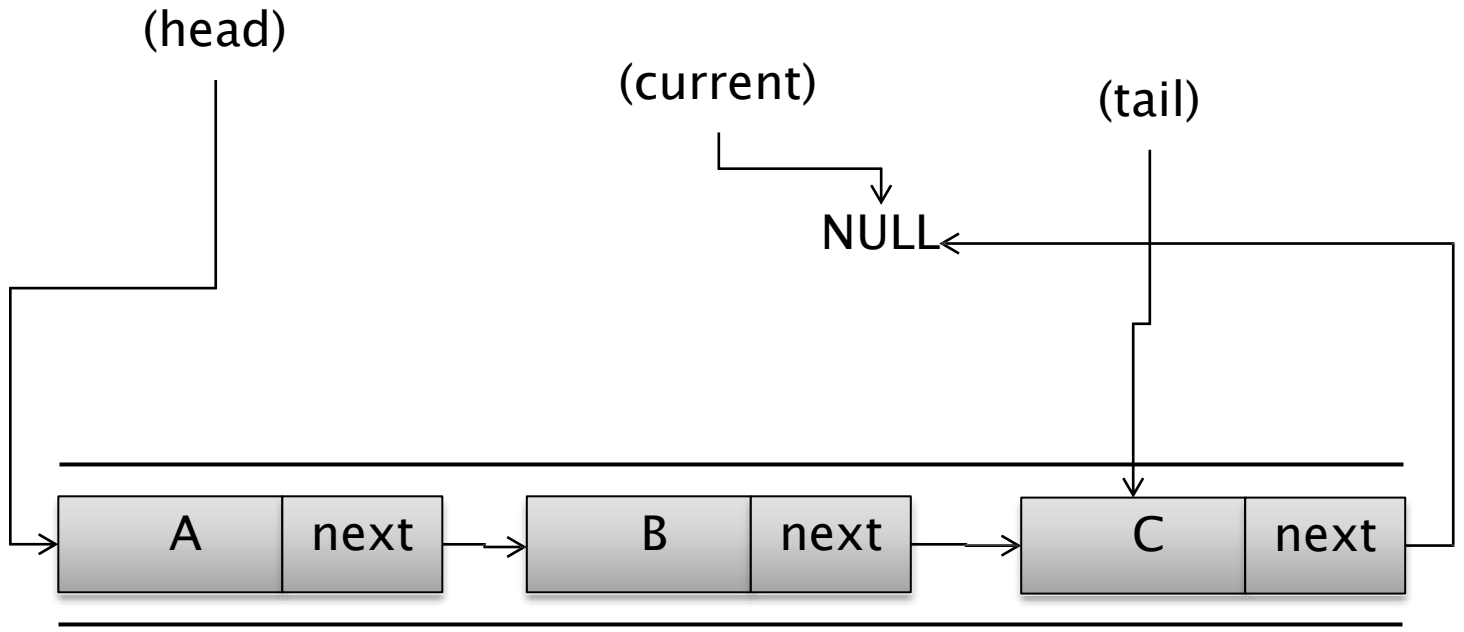
```
q.enqueue("B");
// this statement invokes addLast("B") of LinkedList
```



Queue Example (cont.)

Add an element to the end of queue

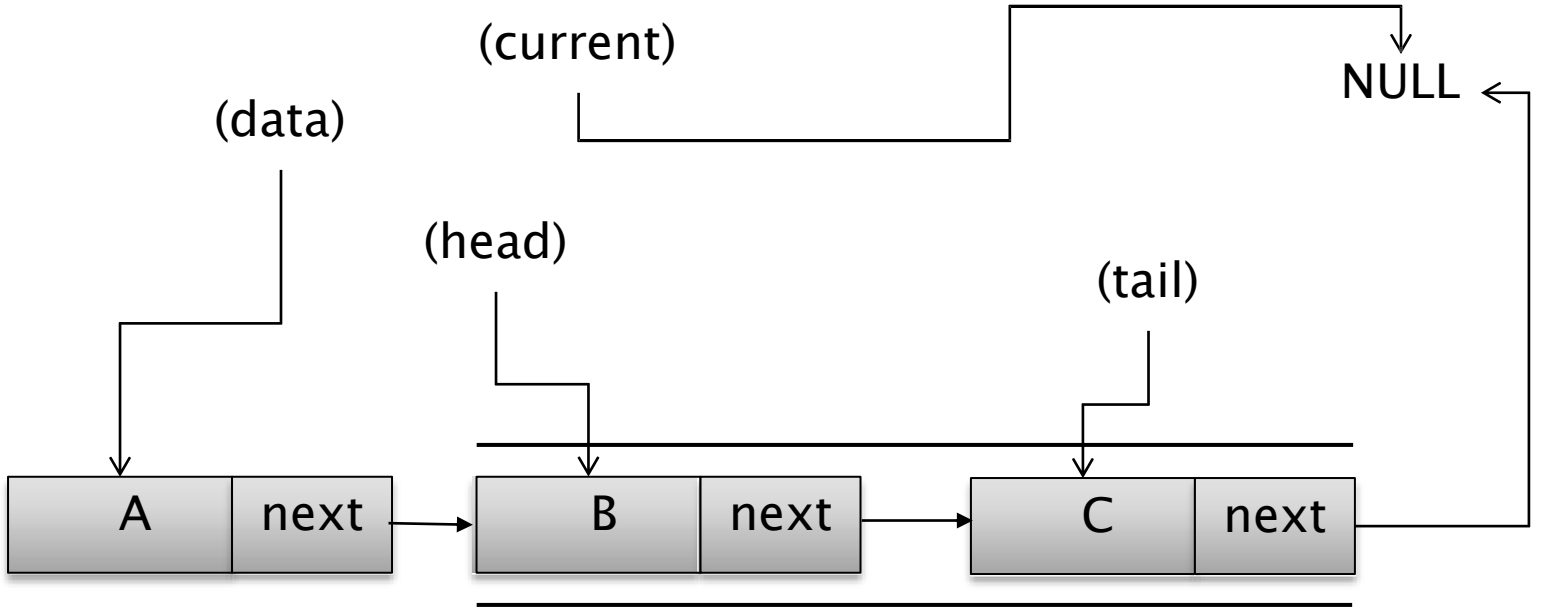
```
q.enqueue("C");
// this statement invokes addLast("C") of LinkedList
```



Queue Example (cont.)

Removes an element from front of queue

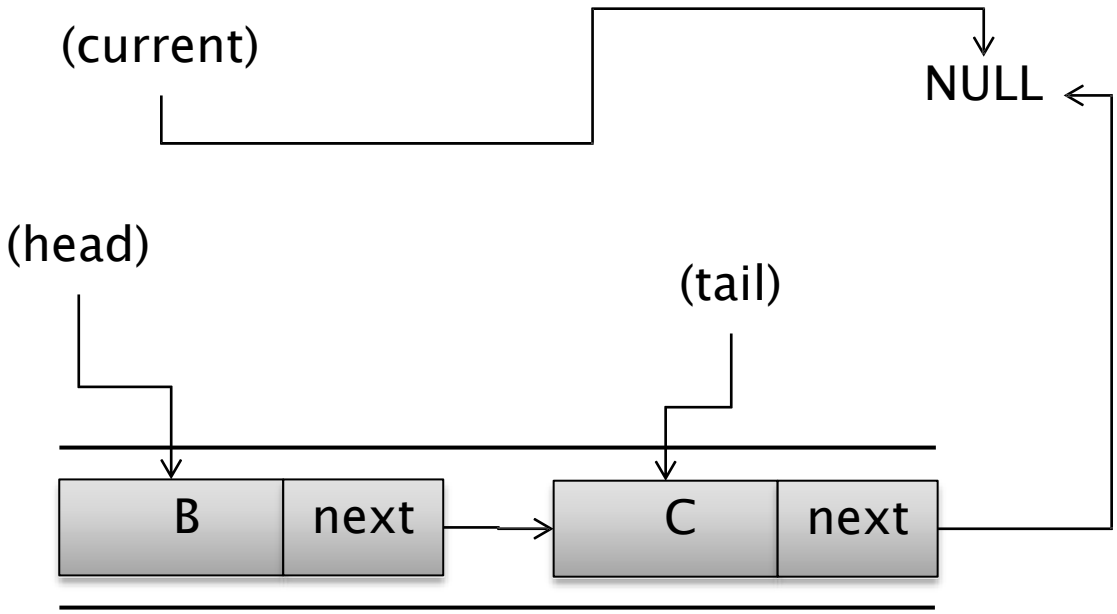
```
String data = q.dequeue();
//invoke removeFirst() of LinkedList
```



Queue Example (cont.)

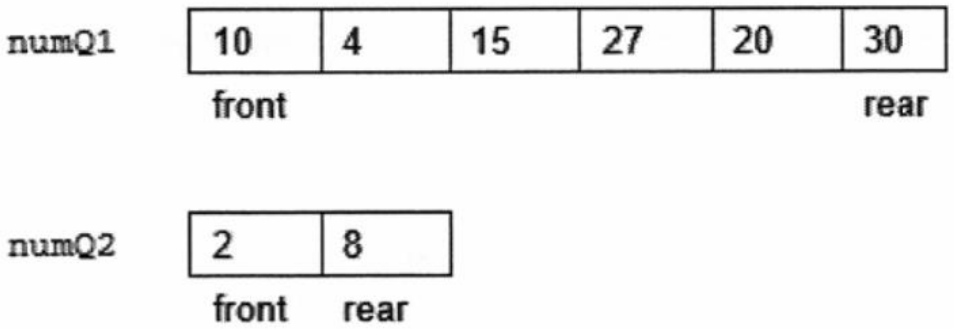
Removes an element from front of queue

```
String data = q.dequeue();
//invoke removeFirst() of LinkedList
```



Exercise

Given 2 integer queues namely `numQ1` and `numQ2` as shown in the following diagrams:



Illustrate the latest diagram of `numQ1` and `numQ2` after the execution of the following program fragment.

```
int num;
for (int i=0; i < 4; i++)
{
    num = Integer.parseInt(numQ1.dequeue().toString());
    if ((num * 4) % 2 == 0)
       numQ2.enqueue (num);
 }
```

Exercise

Given the following `CarWash` and `Queue` class

```
public class CarWash{
 private String carType;        //e.g kelisa,honda
 private String washSelection; // e.g body wash, engine wash
 private double amountCharge ;

 public CarWash(String a, String b, double c){..}
 //accessor and mutator method
}
public class Queue{
 public Queue(){..}
 public void enqueue(Object elem){..}
 public Object dequeue(){..}
 //definition other method
}
```

- Create a queue name `qCarWash`. (1 mark)
- Get input for twenty (20) cars washed object into `qCarWash`. (3 marks)
- Calculate the total amount received from the twenty cars washed. (4 Marks)

References

1. Y. Daniel Liang (2019). Introduction to Java Programming and Data Structures, Comprehensive Version (11th Edition). United Kingdom: Pearson Education Limited.
2. Peter Drake (2014). Data Structures and Algorithm in Java (1st Edition). USA: Pearson Education Limited.

Approved by:
Zuriati Ismail
Resource Person CSC248