CS201
Data Structures and Algorithms
Revision Session 4

queue as array
queue as linked list

Queue

implementation:
queue as array
queue as linked list

algorithms:
isFull
isEmpty
enqueue
dequeue

queue as array

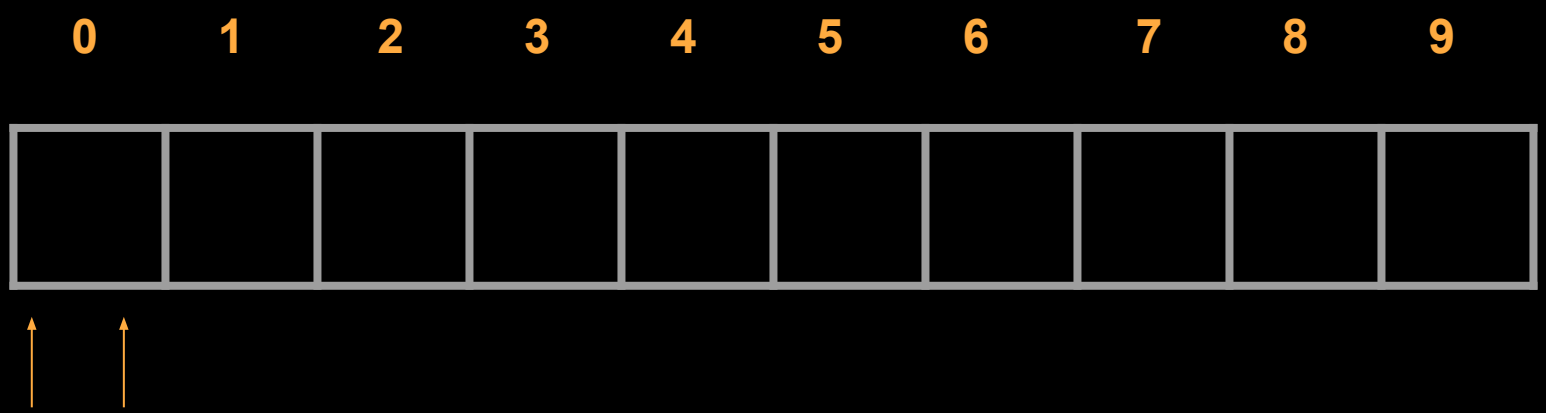
implementation

```java
public class QueueArray {
    3 usages
    Element[] array;
    6 usages
    int first;
    6 usages
    int last;
    4 usages
    int N;
```

```java
public QueueArray(int N){
    array = new Element[N];
    first = 0;
    last = 0;
    this.N = N;
}
```

isEmpty

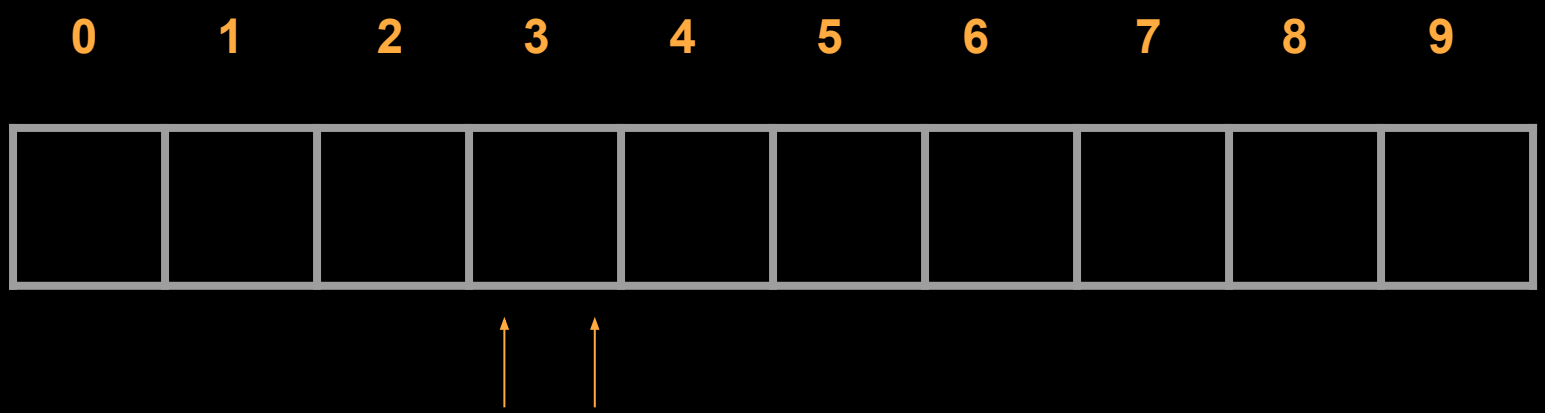
N = 10



If the fist and last fields point to the same index, the queue is empty.

```
public boolean isEmpty(){
    return first == last;
}
```

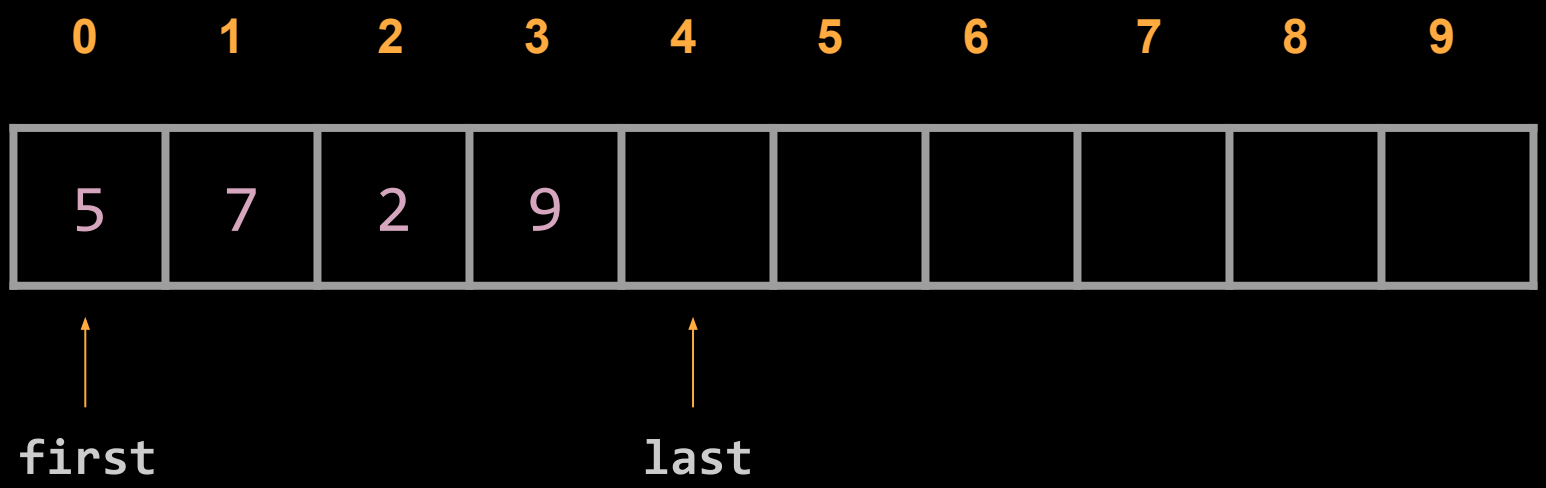
N = 10



If the fist and last fields point to the same index, the queue is empty.

```
public boolean isEmpty(){
    return first == last;
}
```

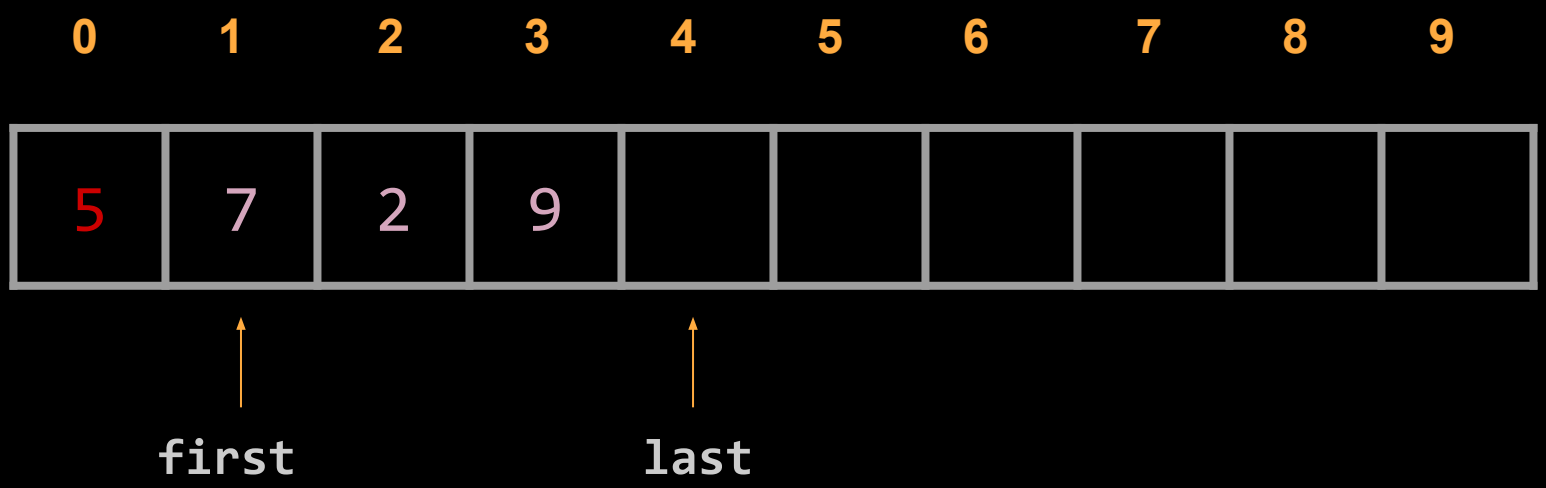
N = 10



If the fist and last fields point to the same index, the queue is empty.

```
public boolean isEmpty(){
    return first == last;
}
```

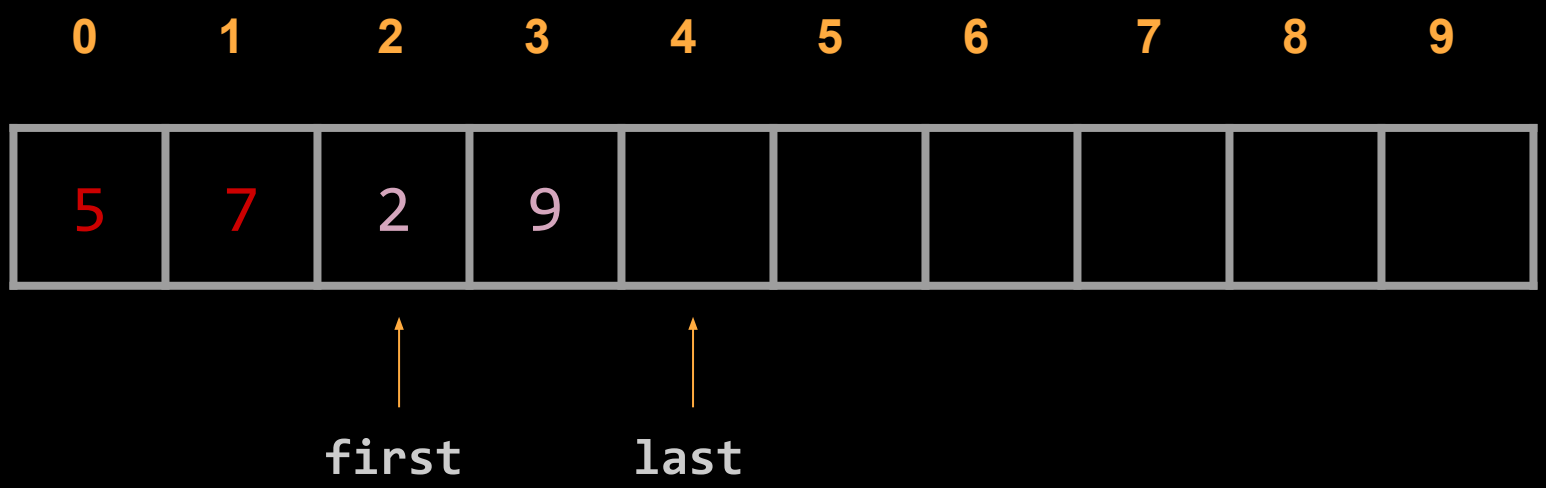
N = 10



If the fist and last fields point to the same index, the queue is empty.

```
public boolean isEmpty(){
    return first == last;
}
```

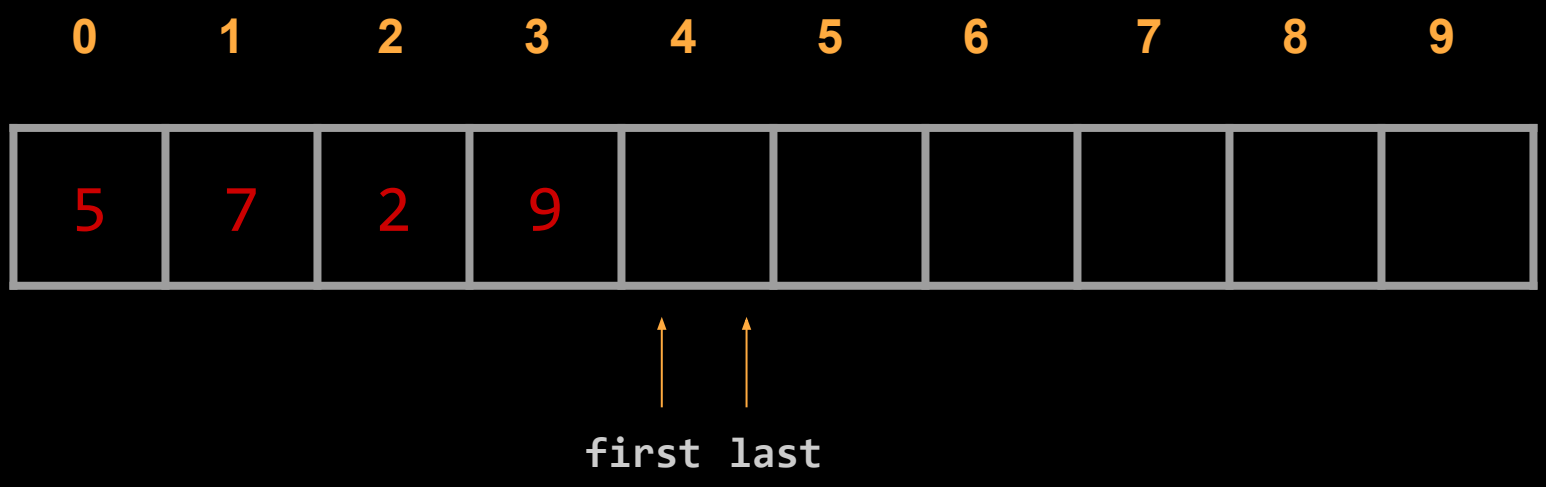
N = 10



If the fist and last fields point to the same index, the queue is empty.

```
public boolean isEmpty(){
    return first == last;
}
```

N = 10

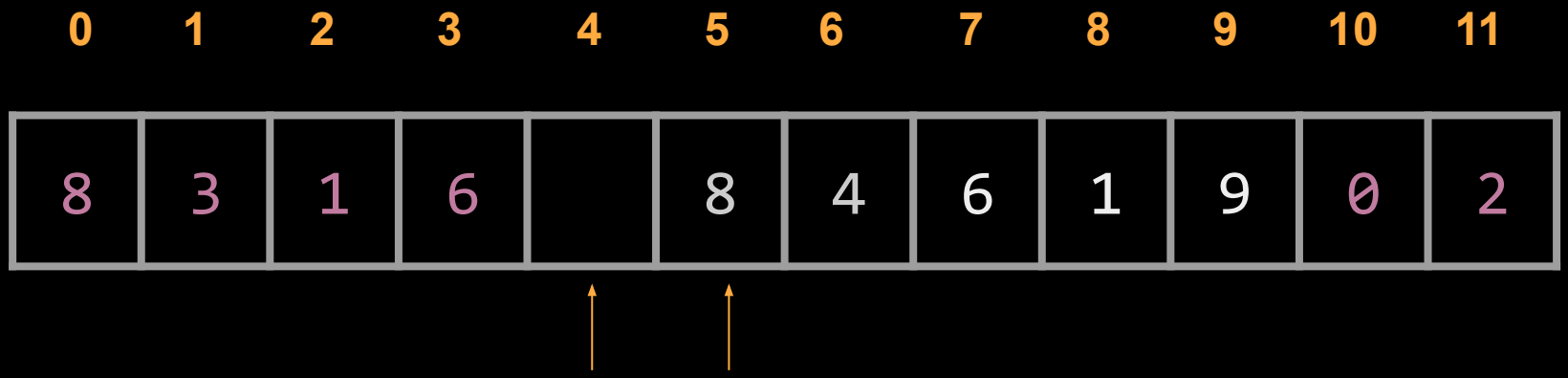


If the fist and last fields point to the same index, the queue is empty.

```
public boolean isEmpty(){
    return first == last;
}
```

isFull

N = 12



5 = (4 + 1) % 12

5 = 5 % 12

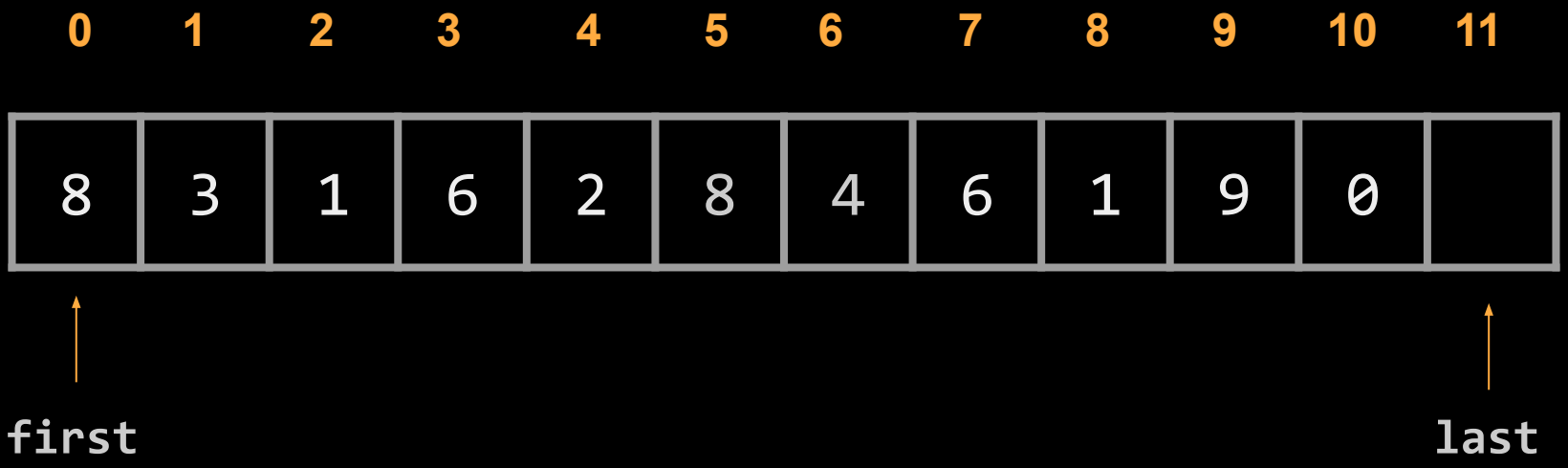
5 = 5

```
public boolean isFull(){
    return (first == (last + 1) % N);
}
```

If the first field comes right after the last field, the queue is full.

Why do we need the modulus operator?

N = 12




```
public boolean isFull(){
    return (first == (last + 1) % N);
}
```

If the first field comes right after the last field, the queue is full.

0 = (11 + 1) % 12

0 = 12 % 12

0 = 0

enqueue

N = 10

N = 10

N = 10

How do we pass from index 9 to index 0?

N = 10

N = 10

(last + 1) % N

(N-1 + 1) % N

N % N

Until it reaches N-1, it will always give the value itself.

6 % 10 = 6

N = 10

```java
public void enqueue(Element newElement){

    if (!isFull()){
        array[last] = newElement;
        last = (last + 1) % N;
    } else
        System.out.println("Queue is full!");
}
```

If the queue is not full:

put the new element in last

shift the last field by one (it should wrap around when it reaches the end)

dequeue

```java
public Element dequeue(){

    Element dequeued = null;

    if (isEmpty()){
        System.out.println("Queue is empty!");
    } else {
        dequeued = array[first];
        first = (first + 1) % N;
    }
    return dequeued;
}
```

If the list is not empty, store the first element in another variable. Then, shift the first field to the left. If the old first was at N-1, the modulus will shift it to index 0.

N = 10

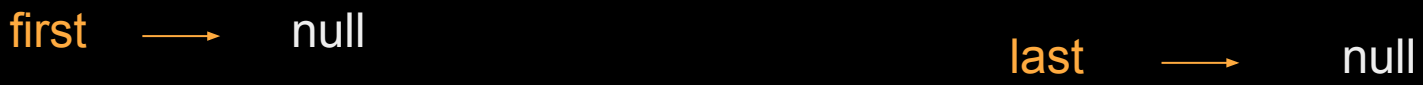
N = 10

queue as linked list

implementation

```java
public class QueueLinked {
    7 usages
    Node first;
    4 usages
    Node last;

    no usages  neslihancesurr
    public QueueLinked(){
        first = null;
        last = null;
    }
```

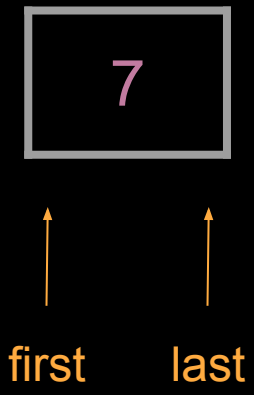

first
null
last
null

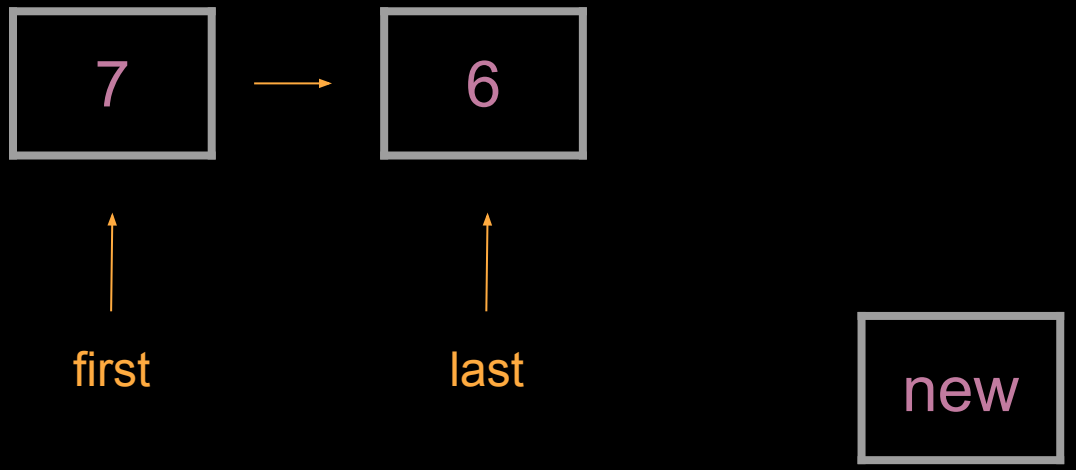
isEmpty?

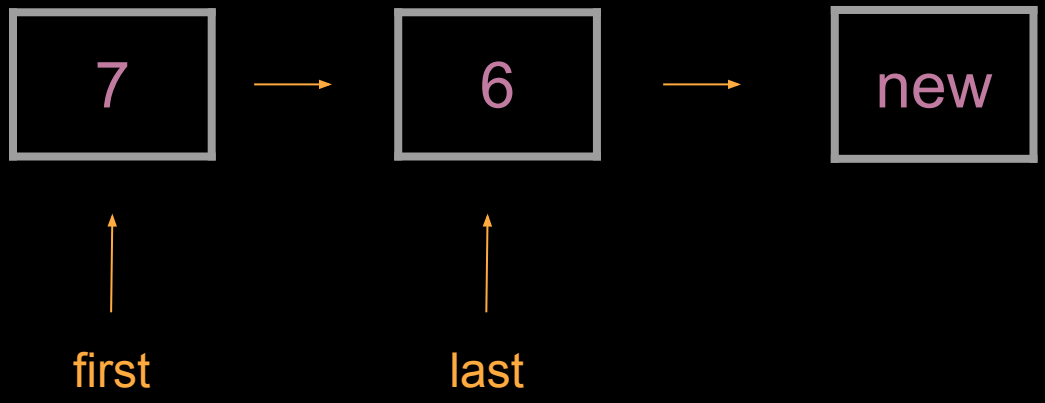
```
public boolean isEmpty(){
    return first == null;
}
```

If the first element is null,
the queue is empty.

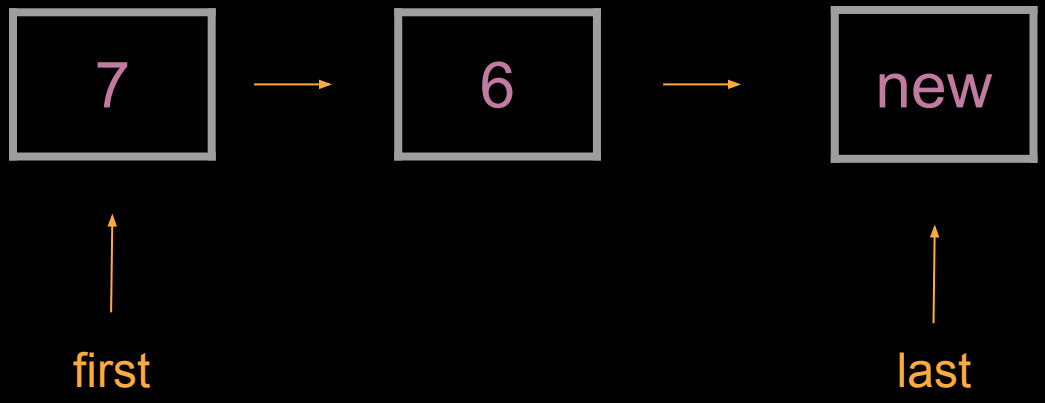
enqueue


7
first
last


7
6
first
last
new




```
last.next = newNode;
```

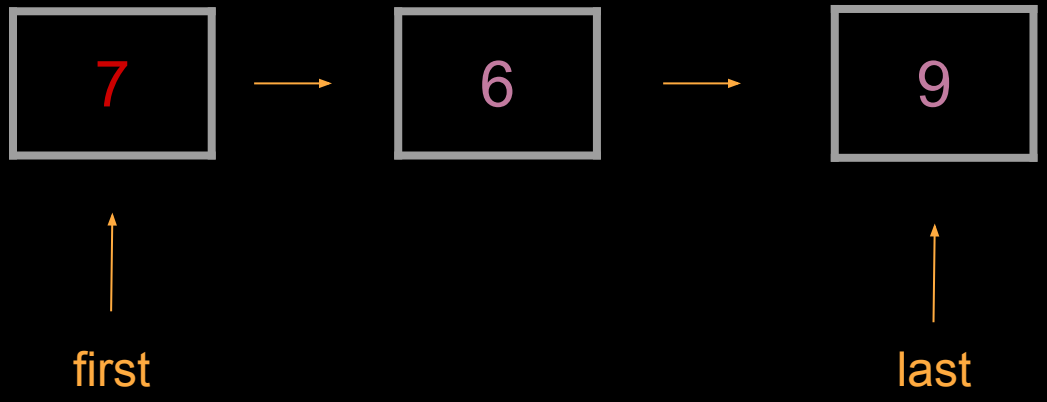



```
last = newNode;
```

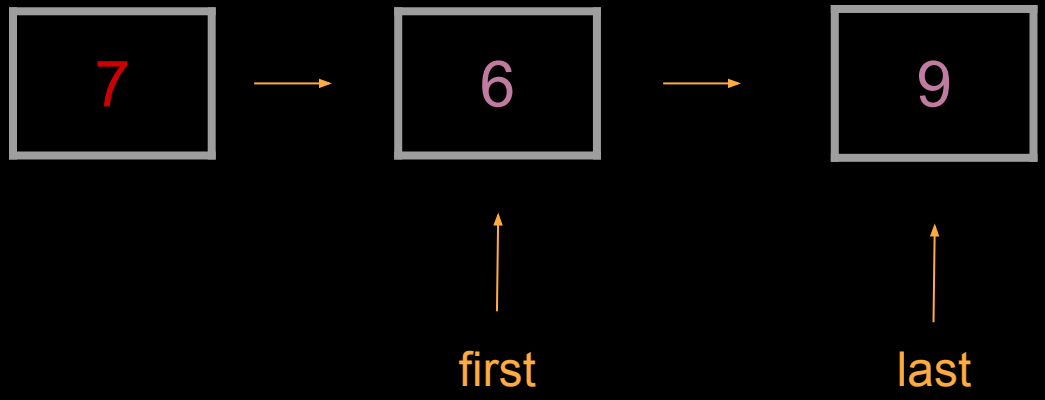
```
public void enqueue(Node newNode){
    if (isEmpty()){
        first = newNode;
    } else {
        last.next = newNode;
    }
    last = newNode;
}
```

If queue is empty, the new node will be first and last. If it is not empty next field of last will point to the new node and the new lat will be the new node.

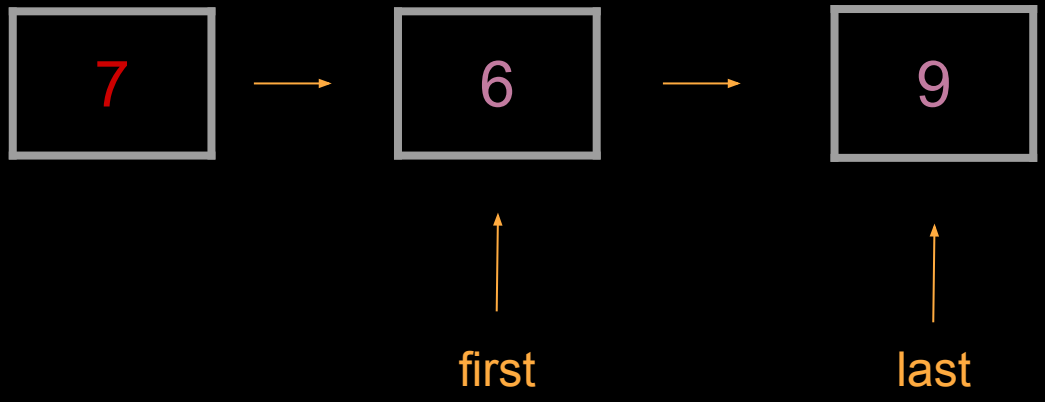
dequeue



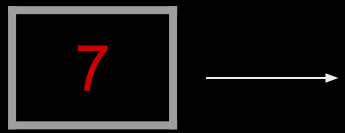

```
Node dequeued = first;
```




```
first = first.next;
```




```
return dequeued;
```



Keep the first node to return it later.

If the link is not empty shift first to its next.

If its next is full (only one element), make the only element last.

Return the old first.

```
public Node dequeue(){

    Node dequeued = first;

    if (!isEmpty()){
        first = first.next;
        if (first == null){
            last = null;
        }
    } else {
        System.out.println("Queue is empty!");
    }

    return dequeued;
}
```

Queue exercise

Write a function that moves the element currently at the front of the queue to the rear of the queue. Write the function for both array and linked list implementations.

```
public void moveToRear()
```

before moveToRear(): first 5 6 8 3 4 last

after moveToRear(): first 6 8 10 3 4 5 last