JAVA NOTES

DATA STRUCTURES AND ALGORITHMS

Terry Marris July 2001

11 QUEUE IMPLEMENTATION - ARRAYS

11.1 LEARNING OUTCOMES

By the end of this lesson the student should be able to

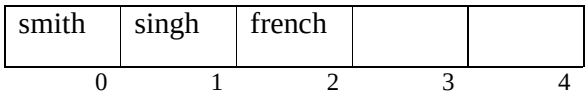
- describe how an array may be used to implement the standard queue operations
- explain the concept of wrap-around
- explain additional queue methods
- test queue methods
- use queues to solve simple problems

11.2 INTRODUCTION

In the last lesson on the queue interface we saw that a queue is a linear data storage structure where items are added at one end (the rear) and removed from the other end (the front). We described the properties of a queue. We described what the standard queue methods *join(Object)*, *leave()*, *retrieve()* and *isEmpty()* do. Now we go on to describe *how* they do it. We shall implement the queue interface using an array as the underlying data storage structure.

11.3 ARRAY IMPLEMENTATION

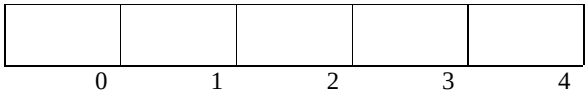
A queue containing three objects might look like this



front	rear	size	capacity
0	2	3	5

The array is named *data*. Variables *front* and *rear* refer to the objects at the front and at the rear of the queue. *size* represents the number of objects in the queue. *capacity* defines the maximum number of objects the queue can contain.

Initially, the queue is empty.



front	rear	size	capacity
-1	-1	0	5

A queue is empty if its size is zero.

```
boolean isEmpty()
{
  return size <= 0;
}
```

To add an item to the queue we increment *rear* and then place the object in *data[rear]*.

```
rear++;
data[rear] = obj;
```

And we increment *size*. If the size of the queue is one, then the object in the queue is both the first and the last object.

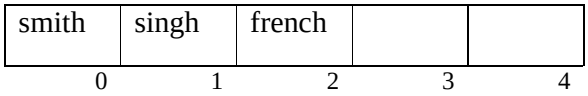
```
size++;
if (size == 1)
  front = rear;
```

data

smith				
0	1	2	3	4

front	rear	size	capacity
0	0	1	5

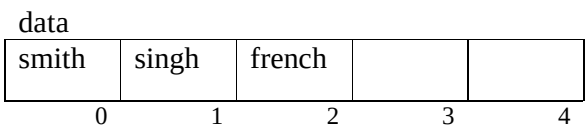
To remove an object from this queue



front	rear	size	capacity
0	2	3	5

we increment *front* and decrement *size*.

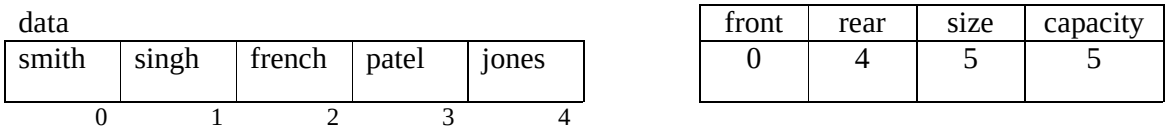
```
front++;
size--;
```



front	rear	size	capacity
1	2	2	5

Although *smith* physically remains in the array, it is logically removed (since the front of the queue is specified by the variable *front*).

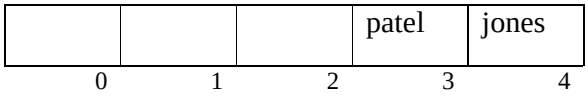
A full queue might look like this.



A queue is full if its size matches its capacity.

```
boolean isFull()
{
  return size >= capacity;
}
```

Now let's remove three items from the queue shown above.



front	rear	size	capacity
3	4	2	5

Do we shuffle *patel* and *jones* along the queue towards location zero? No. To do so is a pain. Here is what we do to add another item.

We increment *rear*.

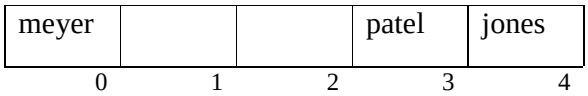
```
rear++;
```

Its value is now five - outside the bounds of the array.

Then we divide it by *capacity* to get the *remainder*: 5 ÷ 5 = 1 remainder 0

```
rear = rear % capacity;
```

rear is now zero. Now we can insert the next new item (meyer) .



front	rear	size	capacity
3	0	3	5

The process of using an array in a circular fashion is known as wrap around.

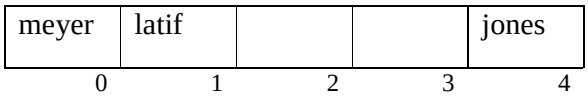
```
public String join(Object obj)
{
  if (isFull())
    return "failure - queue full";

  rear++;
  rear = rear % capacity;  // rear in 0..capacity-1
  data[rear] = obj;

  size++;
  if (size == 1)
    front = rear;

  return "success";
}
```

Let's remove *jones* from the queue shown below.



front	rear	size	capacity
4	1	3	5

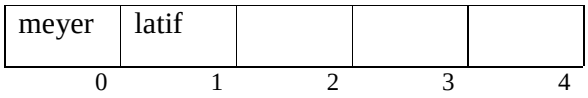
We increment *front*.

```
front++;
```

front is now five, outside the bounds of the array. We find the remainder after dividing by *capacity*.

```
front = front % capacity;
```

front is now zero.



front	rear	size	capacity
0	1	2	5

```
public String leave()
{
  if (isEmpty())
    return "failure - queue empty";

  front++;
  front = front % capacity;  // front in 0..capacity-1

  size--;
  return "success";
}
```

```
/* ArrayQueue.java
   Terry Marris  26 July 2001
*/

public class ArrayQueue implements Queue {
  /* Implements the standard queue operations join(Object),
     leave(), retrieve() and isEmpty().
     In addition implements constructors and
     isFull(), size() and getCapacity().
  */

  private Object[] data;
  private int capacity;
  private int size;
  private int front, rear;


  public ArrayQueue(int capacity)
  /* Initialises a new queue with the given capacity.
  */
  {
    capacity = Math.abs(capacity);
    this.capacity = capacity;
    data = new Object[capacity];  // indexed 0..capacity-1
    size = 0;
    front = rear = -1;
  }


  public ArrayQueue()
  /* Initialises a new queue with a capacity of 10 objects.
  */
  {
    this(10);
  }


  public boolean isEmpty()
  /*  Returns true if this queue contains no objects,
      false otherwise.
  */
  {
    return size <= 0;
  }
```

```
public boolean isFull()
/* Returns true if this queue cannot contain
   an additional object, false otherwise.
*/
{
  return (size >= capacity);
}


public String join(Object obj)
/* Adds the given object to the rear of this queue,
   if there is room, and returns success,
   otherwise returns failure.
*/
{
  if (isFull())
    return "failure - queue full";

  rear++;
  rear = rear % capacity;  // rear in 0..capacity-1
  data[rear] = obj;

  size++;
  if (size == 1)
    front = rear;

  return "success";
}


public String leave()
/* Removes the item at the front of this queue,
   if there is one, and returns success,
   otherwise returns failure.
*/
{
  if (isEmpty())
    return "failure - queue empty";

  front++;
  front = front % capacity;  // front in 0..capacity-1

  size--;
  return "success";
}
```

```
  public Object retrieve()
  /* Returns the item at the front of this queue,
     if there is one, otherwise returns null.
  */
  {
    if (isEmpty())
      return null;
    return data[front];
  }


  public int getCapacity()
  /* Returns the maximum number of objects this queue
     can contain.
  */
  {
    return capacity;
  }


  public int size()
  /* Returns the number of objects currently held
     in this queue.
  */
  {
    return size;
  }
}
```

11.4 TEST ARRAY QUEUE

The test program shown below creates a queue with capacity five, fills it with string objects, and then empties the queue item by item.

```
/* TestArrayQueue.java
   Terry Marris  26 July 2001
*/

public class TestArrayQueue {
  public static void main(String[] s)
  {
    ArrayQueue queue = new ArrayQueue(5);
    System.out.println("A new queue is an empty queue ... " +
                       queue.isEmpty());

    System.out.println(
                    "Adding five strings to the queue ...");
    System.out.println("smith: " +
                       queue.join(new String("smith")));
    System.out.println("singh: " +
                       queue.join(new String("singh")));
    System.out.println("french: " +
                       queue.join(new String("french")));
    System.out.println("patel: " +
                       queue.join(new String("patel")));
    System.out.println("jones: " +
                       queue.join(new String("jones")));

    System.out.println("The queue is now full ..." +
                       queue.isFull());

    System.out.println("Adding a string to a full queue ...");
    System.out.println("tagore: " +
                       queue.join(new String("tagore")));

    System.out.println("Emptying the queue item by item ...");
    while (!queue.isEmpty()) {
      System.out.println(queue.retrieve());
      queue.leave();
    }
  }
}
```

Output

```
A new queue is an empty queue ... true
Adding five strings to the queue ...
smith: success
singh: success
french: success
patel: success
jones: success
The queue is now full ...true
Adding a string to a full queue ...
tagore: failure - queue full
Emptying the queue item by item ...
smith
singh
french
patel
jones
```

11.5 REVIEW

11.6 FURTHER READING

In the next lesson we see how to implement a dynamic queue.

11.7 EXERCISES

1 By using a sequence of diagrams, supported by descriptive prose and snippets of Java code, illustrate the following sequence of events

start with an empty queue capacity 6
hume joins the queue
wilson, *heath* and *thatcher* join the queue
hume leaves the queue
wilson and heath *leave* the queue
major, *blair* and *ghandi* join the queue
nehru and *chastri* join the queue
ghandi leaves the queue
nehru and *chastri* leave the queue

2 Test the array queue implementation with the sequence of events described in question 1 above.