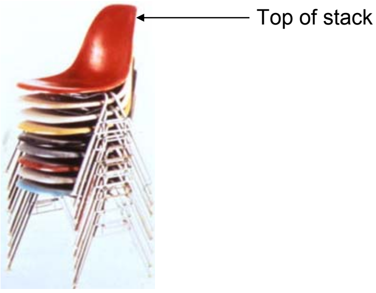
Stacks and Queues

- In this section of notes you will learn about two additional data structures as well as the consequences of different implementations

Stacks

- A list where additions and deletions are made at only one end of the list.
- The last element to be added is the first element to be removed (LIFO).

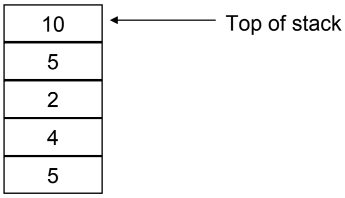


Common Stack Operations

- Push
- Pop
- Peek
- Check if empty
- Clear

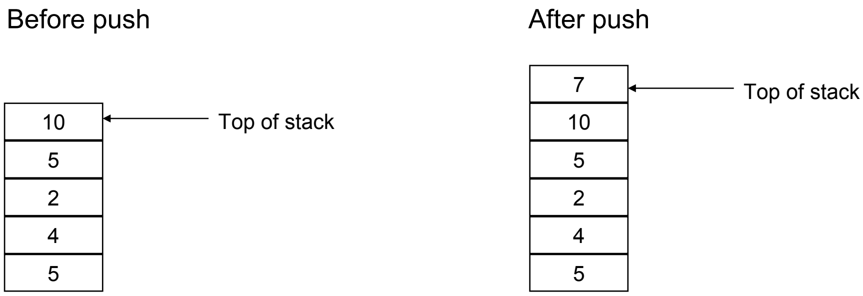
Push Operation

- Adding an item to the top of the stack


10
5
2
4
5
Top of stack

Push Operation

•“7” has been added to the stack and this new item becomes the top of the stack.

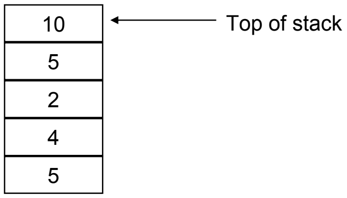


Before push

After push

Pop Operation

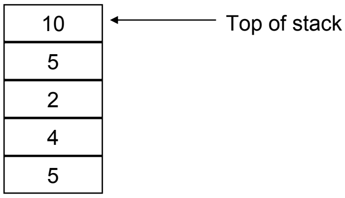
•Removing an item from the top of the stack



Pop Operation

•"10" has been removed and "5" becomes the new top of the stack.

Before pop



10

5

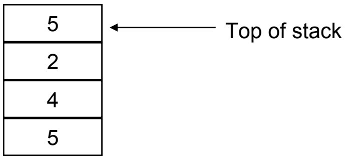
2

4

5

Top of stack

After pop



5

2

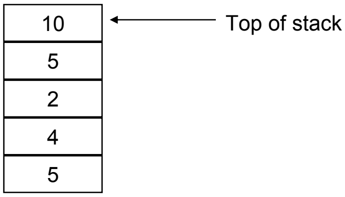
4

5

Top of stack

Peek Operation

•Examine the item at the top of the stack without removing it



10

5

2

4

5

Top of stack

Implementing A Stack As An Array: Class Driver

The full example can be found in the directory:
/home/331/tamj/examples/stacksQueues/arrayStack

```
class Driver
{
   public static void main (String [] args)
   {
      MyStack tamjStack = new MyStack (4);
      System.out.println(ms.peek());
      tamjStack.push(4);
      tamjStack.push(3);
      tamjStack.push(2);
      tamjStack.push(1);
      tamjStack.push(0);
      while (tamjStack.isEmpty() == false)
         System.out.println(tamjStack.pop());
```

Implementing A Stack As An Array: Class MyStack

```
public class MyStack
{
   private int [] stack;
   private int topOfStack;
   private static final int DEFAULT_MAX_SIZE = 100;
   public MyStack ()
   {
      stack = new int[DEFAULT_MAX_SIZE];
      topOfStack = -1;
   }
   public MyStack (int maxSize)
   {
      stack = new int[maxSize];
      topOfStack = -1;
   }
```

Implementing A Stack As An Array: Class MyStack (2)

```
public boolean isFull ()
{
   if (topOfStack >= (stack.length-1))
      return true;
   else
      return false;
}

public boolean isEmpty ()
{
   if (topOfStack < 0)
      return true;
   else
      return false;
}
```

Implementing A Stack As An Array: Class MyStack (3)

```
public void push (int newValue)
{
   if (isFull() == true)
   {
      System.out.println("Stack is full, unable to add element");
      return;
   }
   topOfStack++;
   stack[topOfStack] = newValue;
}
```

Implementing A Stack As An Array: Class MyStack (4)

```
public int peek ()
{
   if (isEmpty() == false)
      return stack[topOfStack];
   else
      return -1;
}
```

Implementing A Stack As An Array: Class MyStack (5)

```
public int pop ()
{
   int top;
   if (isEmpty() == false)
   {
      top = stack[topOfStack];
      stack[topOfStack] = -1;
      topOfStack--;
   }
   else
   {
      top = -1;
   }
   return top;
}
```

Implementing A Stack As An Array: Class MyStack (6)

```
     public void clear ()
    {
       while (topOfStack >= 0)
       {
          stack[topOfStack] = -1;
          topOfStack--;
       }
    }
}
```

Implementing A Stack As A Linked List: Class Driver

The full example can be found in the directory:
/home/331/tamj/examples/stacksQueues/linkedStack

```
class Driver
{
   public static void main (String [] args)
   {
      int temp;
      MyStack tamjStack = new MyStack ();
      System.out.println(tamjStack.peek());
      tamjStack.push();
      tamjStack.push();
      while (tamjStack.isEmpty() == false)
         System.out.println(tamjStack.pop());
   }
}
```

Implementing A Stack As A Linked List: Class MyStack

```
public class MyStack
{
   private Node topNode;
   private int currentDataValue = 10;
   public MyStack ()
   {
      topNode = null;
   }
   public boolean isEmpty ()
   {
      if (topNode == null)
         return true;
      else
         return false;
   }
```

Implementing A Stack As A Linked List: Class MyStack (2)

```
   public void push ()
   {
      Node temp = new Node (currentDataValue, topNode);
      currentDataValue += 10;
      topNode = temp;
   }
   public int peek ()
   {
      Node top;
      if (isEmpty() == false)
      {
         top = topNode;
         return top.data;
      }
      return -1;
   }
```

Implementing A Stack As A Linked List: Class MyStack (3)

```
public int pop ()
{
   Node top;
   if (isEmpty() == false)
   {
      top = topNode;
      topNode = topNode.next;
      return top.data;
   }
   return -1;
}
```

Implementing A Stack As A Linked List: Class MyStack (4)

```
 public void clear ()
{
   while (topNode != null)
        topNode = topNode.next;
}
```

Implementing A Stack As A Linked List: (Inner) Class Node

```
private class Node
{
   private int data;
   private Node next;
   private Node ()
   {
      data = 0;
      next = null;
   }
  private Node (int startingData)
   {
      data = startingData;
      next = null;
   }
```

Implementing A Stack As A Linked List: (Inner) Class Node (2)

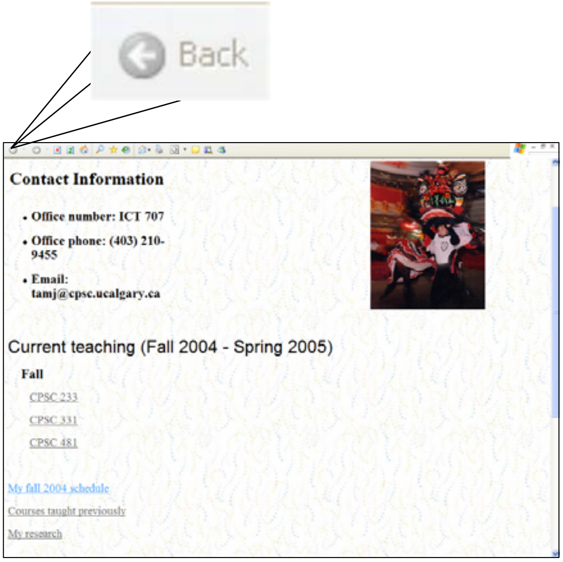
```
   private Node (Node startingNext)
   {
      data = 0;
      next = startingNext;
   }
   private Node (int startingData, Node nextNode)
   {
      data = startingData;
      next = nextNode;
   }
   public String toString ()
   {
      String s = new String ();
      s = s + data;
      return s;
   }
```

A Real Life Application Of Stacks

- Web navigation using the “back” button.

Question: You start out at my CPSC home page and then you go to my 233 course page. Next you press back to go to my home page and then follow the link to my 331 course page. How many clicks of the back button are needed to get back to my 233 course page?

Start At My CPSC Home Page



Contact Information

- **Office number: ICT 707**
- **Office phone: (403) 210-9455**
- **Email: tamj@cpsc.ucalgary.ca**

Current teaching (Fall 2004 - Spring 2005)

Fall

CPSC 233

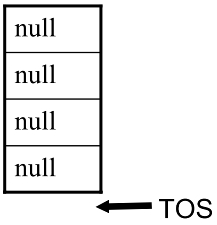
CPSC 331

CPSC 481

My fall 2004 schedule

Courses taught previously

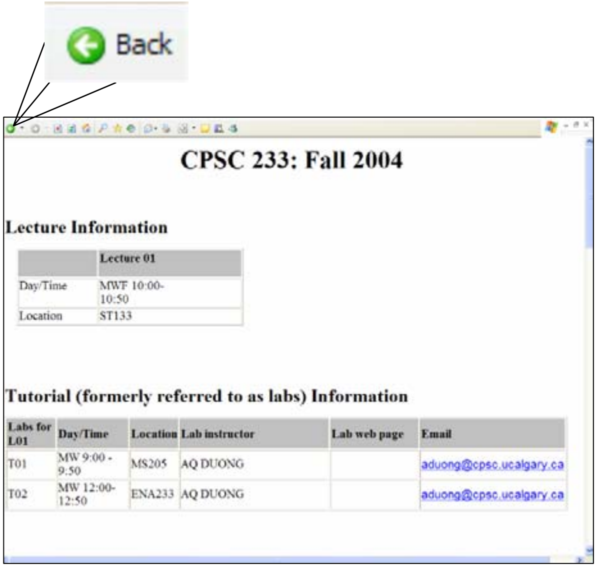
My research



null
null
null
null

TOS

Click On The Link To My 233 Course Page



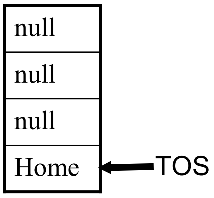
CPSC 233: Fall 2004

Lecture Information

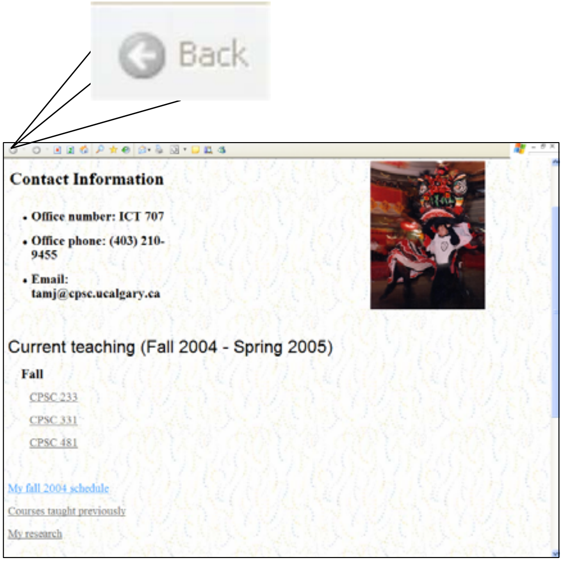
	Lecture 01
Day/Time	MWF 10:00-10:50
Location	ST133

Tutorial (formerly referred to as labs) Information

Labs for L01	Day/Time	Location	Lab instructor	Lab web page	Email
T01	MW 9:00 - 9:50	MS205	AQ DUONG		aduong@cpsc.ucalgary.ca
T02	MW 12:00-12:50	ENA233	AQ DUONG		aduong@cpsc.ucalgary.ca



Click “Back”: Return To My Home Pages



Contact Information

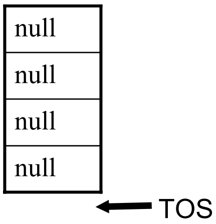
- Office number: ICT 707
- Office phone: (403) 210-9455
- Email: tamj@cpsc.ucalgary.ca

Current teaching (Fall 2004 - Spring 2005)

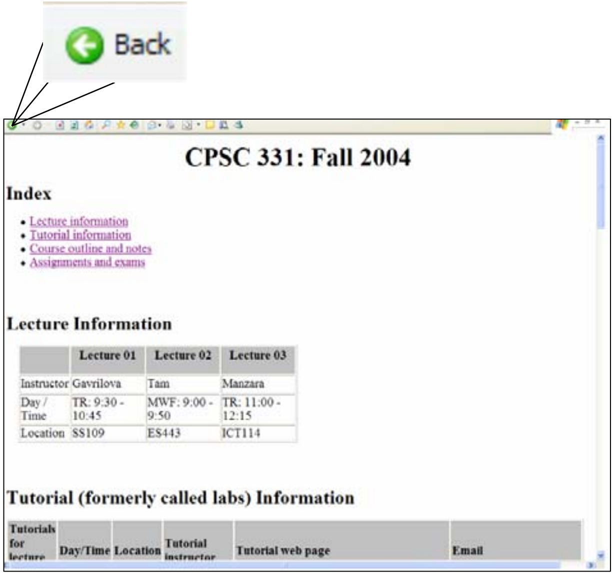
Fall

CPSC 233
CPSC 331
CPSC 481

My fall 2004 schedule
Courses taught previously
My research



Click On The Link To My 331 Course Page



CPSC 331: Fall 2004

Index

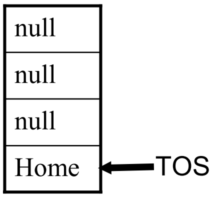
- Lecture information
- Tutorial information
- Course outline and notes
- Assignments and exams

Lecture Information

	Lecture 01	Lecture 02	Lecture 03
Instructor	Gavrilova	Tam	Manzara
Day / Time	TR: 9:30 - 10:45	MWF: 9:00 - 9:50	TR: 11:00 - 12:15
Location	SS109	ES443	ICT114

Tutorial (formerly called labs) Information

Tutorials for lecture	Day/Time	Location	Tutorial instructor	Tutorial web page	Email


null

null

null

Home

TOS

Queues

- A list where additions occur only at one end of the list and deletions occur only at the other end.
- The first element that was added to the queue is the first element to be removed (FIFO).

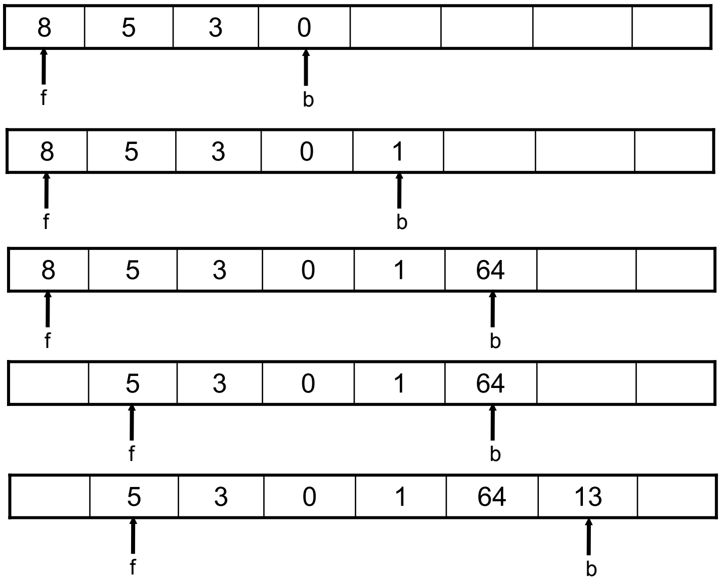
Front: Exit queue

Back: Enter queue

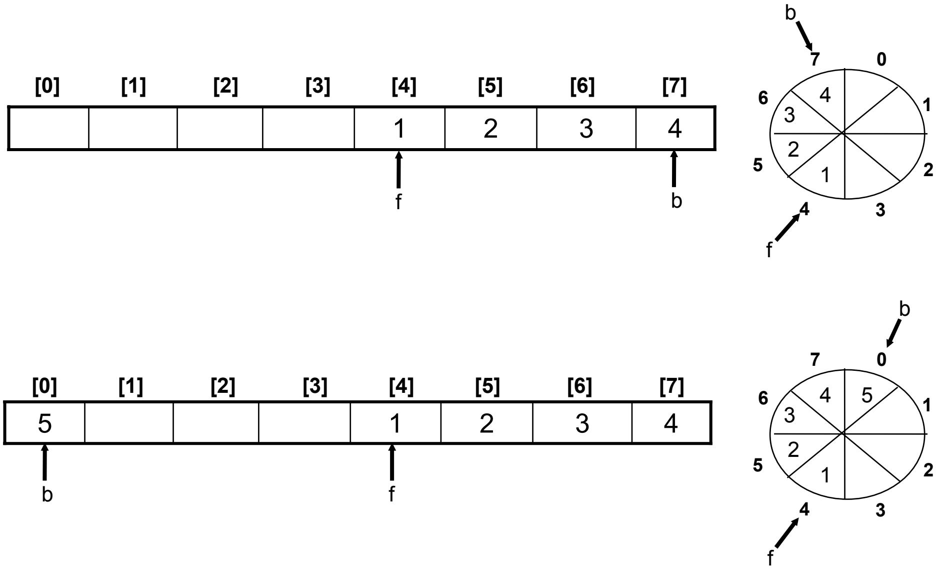
Common Operations On Queues

- Enqueue (add)
- Dequeue (remove)
- Peek
- Check if empty
- Clear

Array Implementation Of A Queue



The Array Implementation Is "Circular"

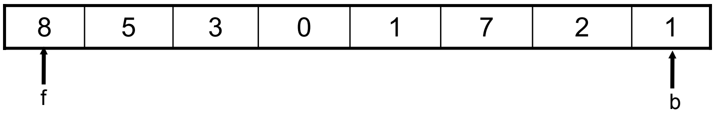


Array Implementation Of A Queue

- The challenges:
  - Detecting when the queue is empty.
  - Detecting when the queue is full.

First Approach: Queue Is When "Back" Refers To The Last Element

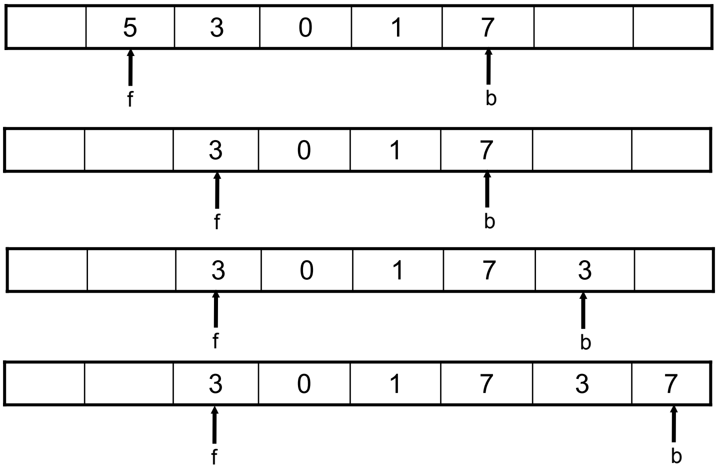
- Queue full when back = last element (length - 1)



8	5	3	0	1	7	2	1
f							b

Problem With The First Approach

- Queue is full?



	5	3	0	1	7		
	f				b		

		3	0	1	7		
		f			b		

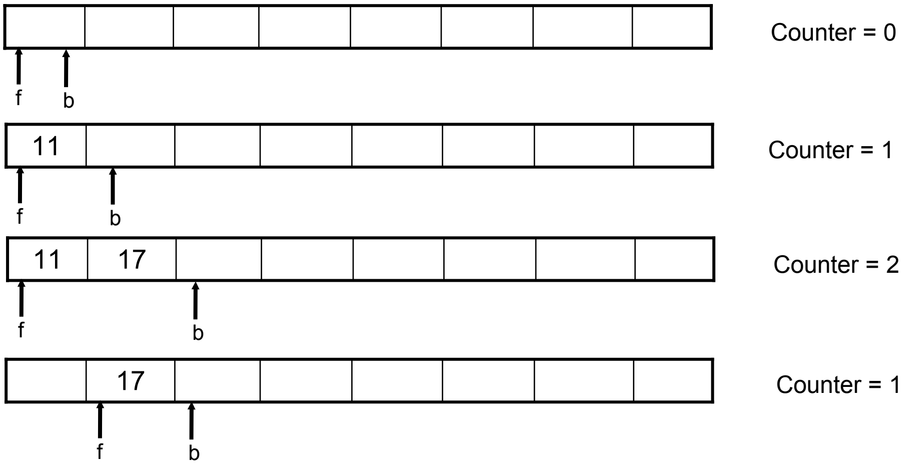
		3	0	1	7	3	
		f				b	

		3	0	1	7	3	7
		f					b

Second Approach: Employ A Counter

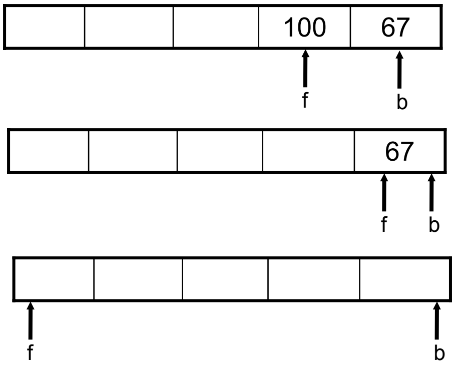
- Enqueue operation: increment the counter
- Dequeue operation: decrement the counter
- When the counter equals the zero the queue is empty
- When the counter equals the length of the list the queue is full

Second Approach: Employ A Counter



General Problem With Circular Queues

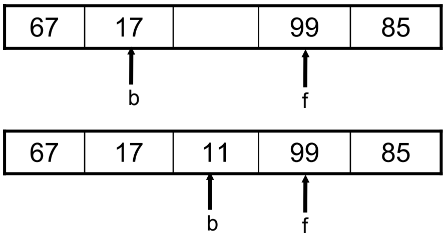
- Confusing the “queue empty” and “queue full” conditions.



Aha! When “f” is just ahead of “b” then the queue is empty.

General Problem With Circular Queues

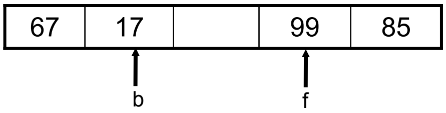
- Confusing the “queue empty” and “queue full” conditions.



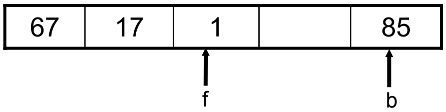
Is the queue empty or full???

Third Approach: Reserve An Array Element

- The index of the unused element lies between the front and the back indices (one greater than the back index and one less than the front index).
- The full condition occurs when all elements but one are occupied or when:
  - The front index is two elements ahead of the back index or



  - The back index is two ahead of the front index (front is two behind the back)



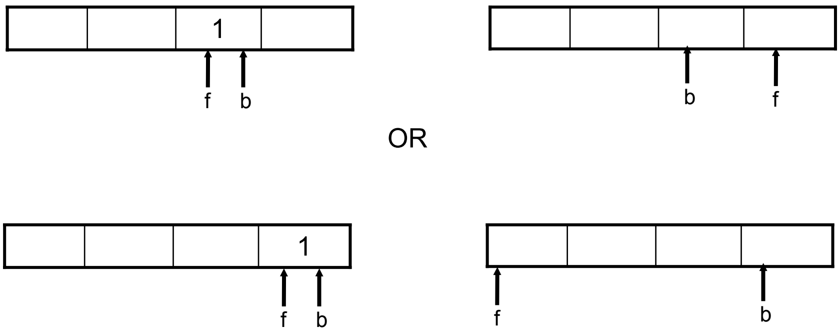
Third Approach: Reserve An Array Element

- General formula to detect when the queue is full:

  Front index = (back index + 2) % queue.length

Third Approach: Reserve An Array Element

- The empty condition occurs when all array elements are unoccupied or when:
  - The front index is one element “ahead” of the back index.



Third Approach: Reserve An Array Element

- General formula to detect when the queue is empty:

Front index = (back index + 1) % queue.length

Implementing A Stack As An Array: Class Driver

The full example can be found in the directory:
/home/331/tamj/examples/stacksQueues/arrayQueue

```
class Driver
{
   public static void main (String [] args)
   {
      MyQueue tamjQueue = new MyQueue (4);
      System.out.println(tamjQueue.peek());
      tamjQueue.enqueue(4);
      tamjQueue.enqueue(3);
      tamjQueue.enqueue(2);
      tamjQueue.dequeue();
      tamjQueue.enqueue(4);
```

Implementing A Stack As An Array: Class Driver (2)

```
      tamjQueue.enqueue(4);
      while (tamjQueue.isEmpty() == false)
      {
         System.out.println(tamjQueue.dequeue());
      }
   }
}
```

Implementing A Stack As An Array: Class MyQueue

```
public class MyQueue
{
    private int [] queue;
    private int front;
    private int back;
    private static final int DEFAULT_MAX_SIZE = 100;

    public MyQueue ()
    {
        queue = new int [DEFAULT_MAX_SIZE];
        front = 0;
        back = queue.length - 1;
    }
```

Implementing A Stack As An Array: Class MyQueue (2)

```
    public MyQueue (int maxSize)
    {
        queue = new int[maxSize];
        front = 0;
        back = queue.length - 1;
    }

    public boolean isFull ()
    {
        if (front == (back+2) % queue.length)
            return true;
        else
            return false;
    }
```

Implementing A Stack As An Array: Class MyQueue (3)

```
public boolean isEmpty ()
{
    if (front == ((back + 1) % queue.length))
        return true;
    else
        return false;
}
```

Implementing A Stack As An Array: Class MyQueue (4)

```
public void enqueue (int newValue)
{
    if (isFull() == true)
    {
        System.out.println("Queue is full, unable to add element");
        return;
    }
    back = (back + 1) % queue.length;
    queue[back] = newValue;
}
```

Implementing A Stack As An Array: Class MyQueue (5)

```
public int dequeue ()
{
   int first;
   if (isEmpty() == false)
   {
      first = queue[front];
      queue[front] = -1;
      front = (front + 1) % queue.length;
   }
   else
   {
      first = -1;
   }
   return first;
}
```

Implementing A Stack As An Array: Class MyQueue (6)

```
public int peek ()
{
   if (isEmpty() == false)
      return queue[front];
   else
      return -1;
}

public void clear ()
{
   while (isEmpty() == false)
   {
      queue[front] = -1;
      front = (front + 1) % queue.length;
   }
```

Implementing A Stack As An Array: Class MyQueue (7)

```
        front = 0;
        back = queue.length;
    }
}
```

Implementing A Queue As A Linked List: Class Driver

The full example can be found in the directory:
/home/331/tamj/examples/stacksQueues/linkedQueue

```
class Driver
{
    public static void main (String [] args)
    {
        int temp;
        MyQueue tamjQueue = new MyQueue ();
        System.out.println(tamjQueue.peek());
        tamjQueue.enqueue();
        tamjQueue.enqueue();
```

Implementing A Queue As A Linked List: Class Driver (2)

```
        tamjQueue.enqueue();
        tamjQueue.dequeue();
        System.out.println(tamjQueue.peek());
        tamjQueue.dequeue();
        System.out.println(tamjQueue.peek());
        tamjQueue.clear();
        System.out.println(tamjQueue.peek());
    }
}
```

Implementing A Queue As A Linked List: Class MyQueue

```
public class MyQueue
{
    private Node front;
    private Node back;
    private int currentDataValue = 10;

    public MyQueue ()
    {
        front = null;
        back = null;
    }
```

Implementing A Queue As A Linked List: Class MyQueue (2)

```
public boolean isEmpty ()
{
   if (front == null)
      return true;
   else
      return false;
}
```

Implementing A Queue As A Linked List: Class MyQueue (3)

```
public void enqueue ()
{
   Node temp = new Node (currentDataValue);
   if (isEmpty() == false)
      back.next = temp;
   else
      front = temp;
   back = temp;
   currentDataValue += 10;
}
```

Implementing A Queue As A Linked List: Class MyQueue (5)

```
public int peek ()
{
   Node temp;
   if (isEmpty() == false)
   {
      temp = front;
      return temp.data;
   }
   return -1;
}
```

Implementing A Queue As A Linked List: Class MyQueue (6)

```
public int dequeue ()
{
   Node temp;
   if (isEmpty() == false)
   {
      temp = front;
      front = front.next;
      return temp.data;
   }
   return -1;
}
```

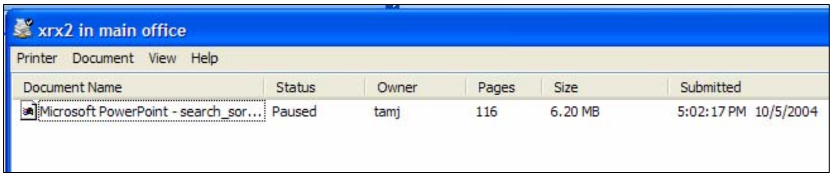
Implementing A Queue As A Linked List: Class MyQueue (7)

```
public void clear ()
{
   front = null;
   back = null;
}


// Plus class Node must also be defined as an inner class of class MyQueue.
```

A Real Life Application Of Queues

- Printing on a printer



Document Name	Status	Owner	Pages	Size	Submitted
Microsoft PowerPoint - search_sor...	Paused	tamj	116	6.20 MB	5:02:17 PM 10/5/2004

You Should Now Know

- What is a stack?
- What is a queue?
- What are the implications of array vs. a linked list implementations?

Sources Of Lecture Material

- *Data Structures and Abstractions with Java* by Frank M. Carrano and Walter Savitch
- *Data Abstraction and Problem Solving with Java* by Frank M. Carrano and Janet J. Prichard
- CPSC 331 course notes by Marina L. Gavrilova http://pages.cpsc.ucalgary.ca/~marina/331/