JAVA NOTES

DATA STRUCTURES AND ALGORITHMS

Terry Marris July 2001

12 DYNAMIC QUEUES

12.1 LEARNING OUTCOMES

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

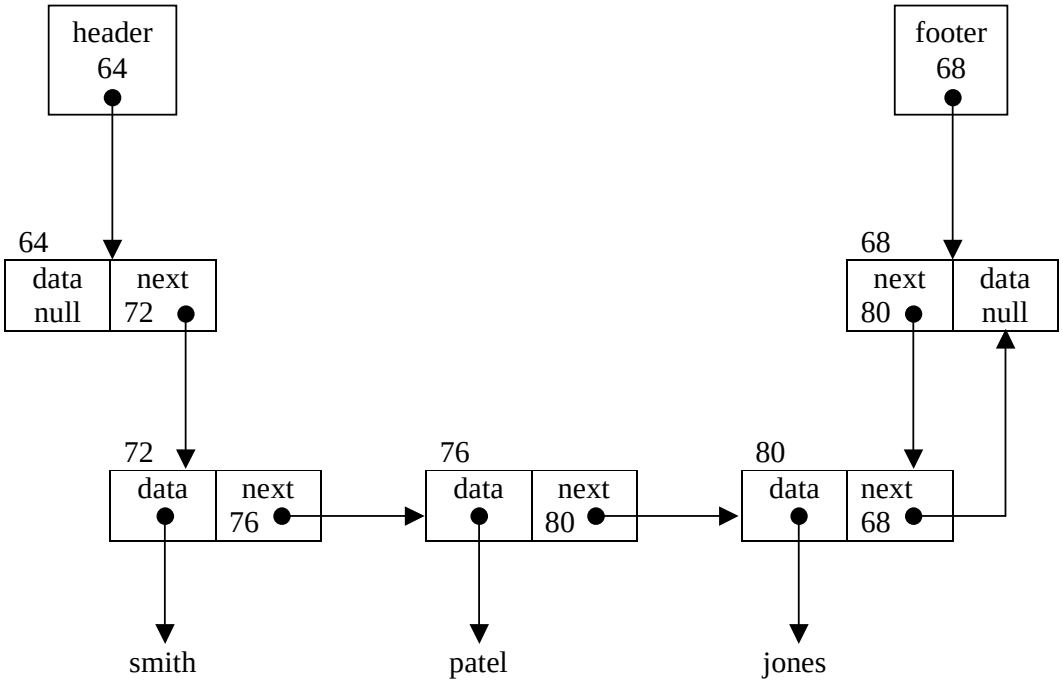
- draw diagrams to show how queue operations can be implemented on a linked list of nodes
- implement the standard queue operations on a linked list of nodes

12.2 PRE-REQUISITES

The student should be comfortable with §9 Dynamic Stacks

12.3 QUEUE

We picture a queue as a linear sequence of nodes. The following diagram represents a queue with three elements referring to strings *smith, patel* and *jones*.



header refers to the front of the queue; items are removed from this end (node 64).

footer refers to the rear of the queue; items are added at this end (node 68).

Node 72 has been in the queue for the longest. Node 80 is the most recent addition to the queue.

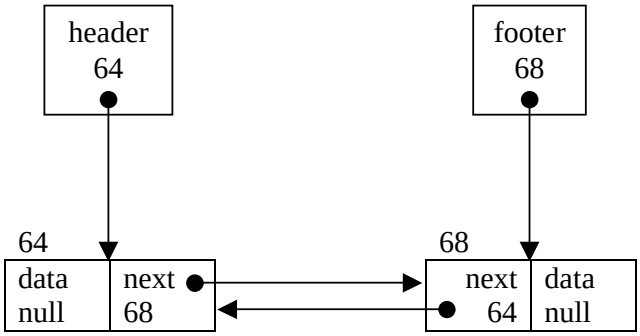
The *DynamicQueue* class has three fields.

```
public class DynamicQueue implements Queue {
    ...
  private Node header;
  private Node footer;
  private int size;
```

size represents the number of data objects stored in the queue. The *Node* class is defined in §9.3.

12.4 EMPTY QUEUE

A newly created queue is an empty queue.



The constructor initialises both *header* and *footer* with a reference to new nodes, sets *header.next* to refer to the *footer*, sets *footer.next* to refer to the *header*, and sets *size* to zero.

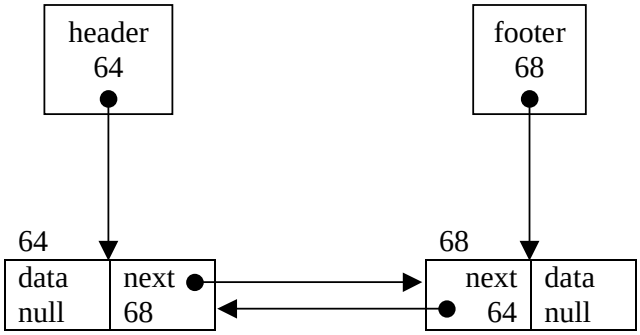
```
public DynamicQueue()
{
  header = new Node(null, null);
  footer = new Node(null, null);
  header.next = footer;
  footer.next = header;
  size = 0;
}
```

The *isEmpty()* method returns true if *size* is zero.

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

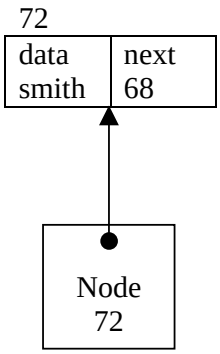
12.5 JOIN

We start with an empty queue.



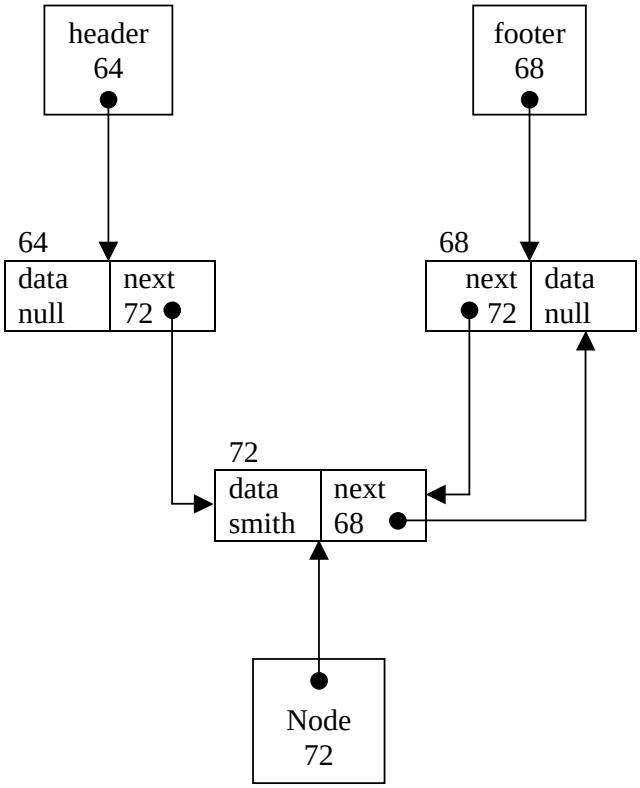
We create a new node with the given object (*smith*) and the contents of *footer* for its link field.

```
Node node = new Node(obj, footer);
```



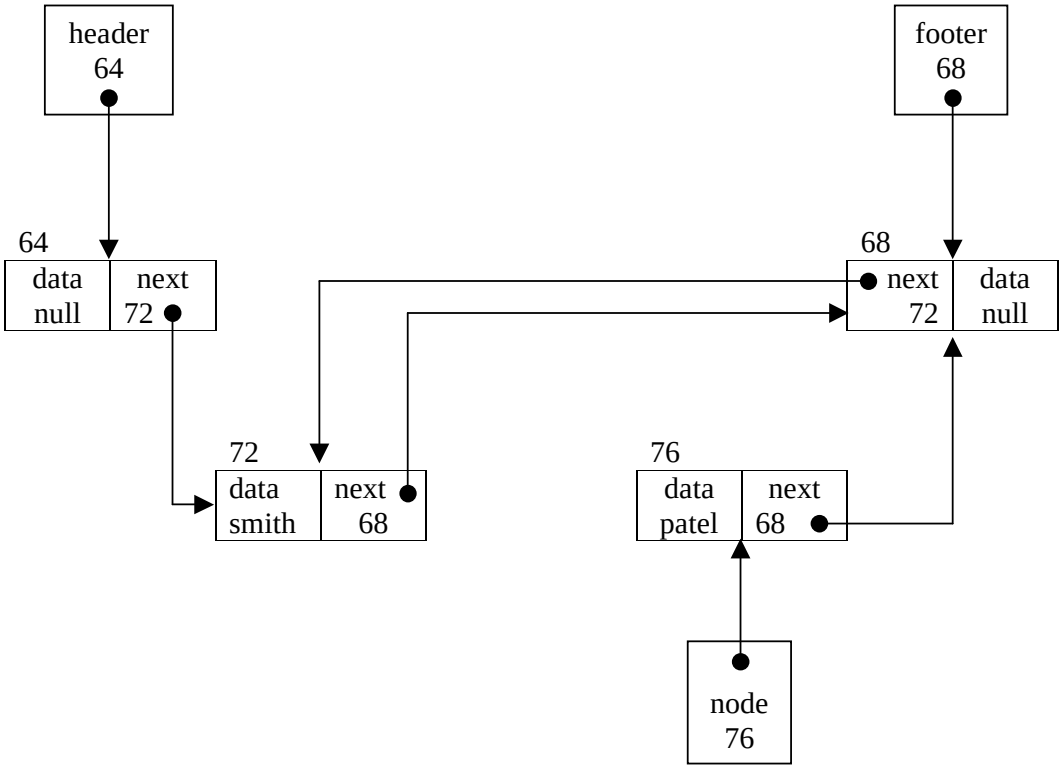
Then we update *footer.next.next* and then *footer.next* to contain the address of the new node.

```
footer.next.next = node;
footer.next = node;
```



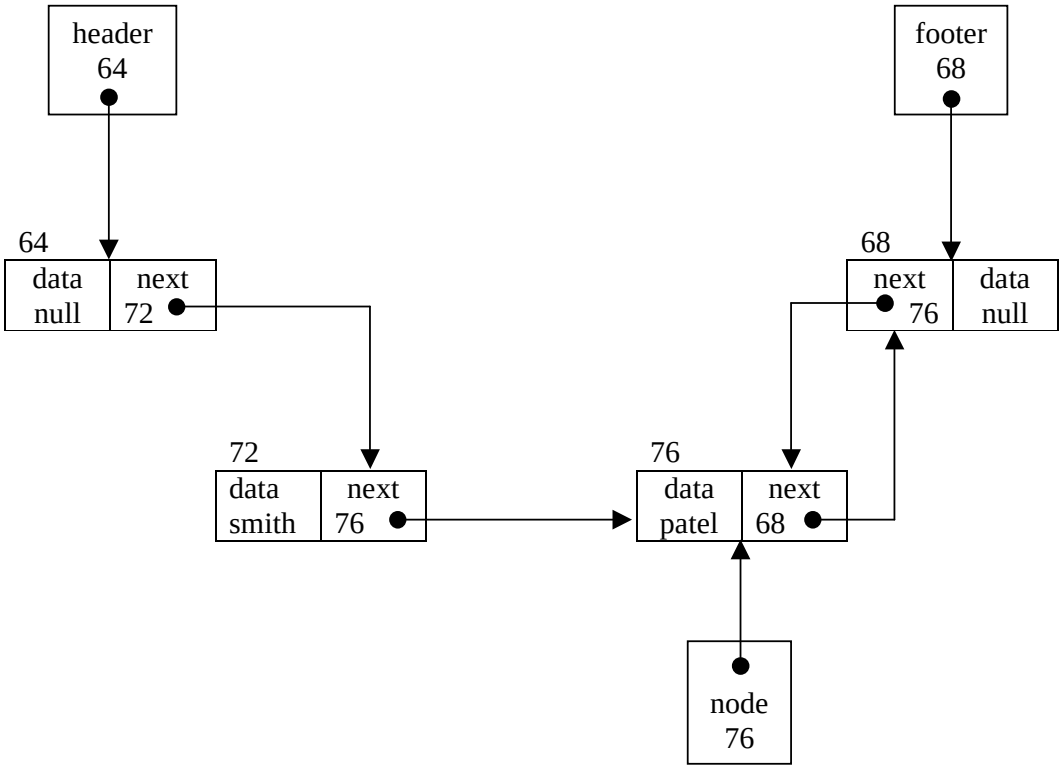
We check out the steps by adding another new node to the queue. We start with a queue containing just one item. We create a new node with the given object (*patel*) and the contents of *footer* for its link field.

```
Node node = new Node(obj, footer);
```



Then we update *footer.next.next* and then *footer.next* to contain the address of the new node.

```
footer.next.next = node;
footer.next = node;
```



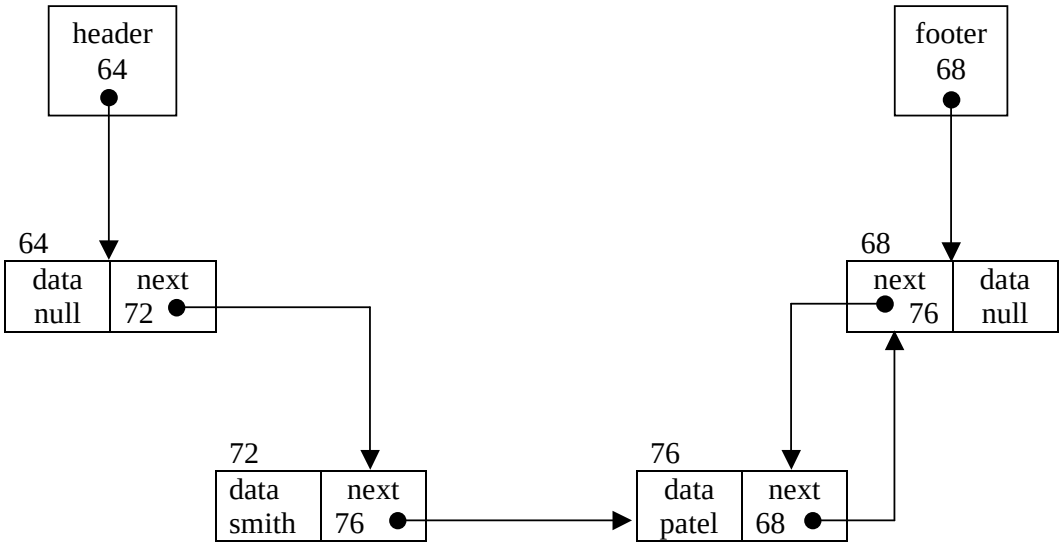

```
public String join(Object obj)
{
  Node node = new Node(obj, footer);

  footer.next.next = node;
  footer.next = node;
  size++;

  return "success";
}
```

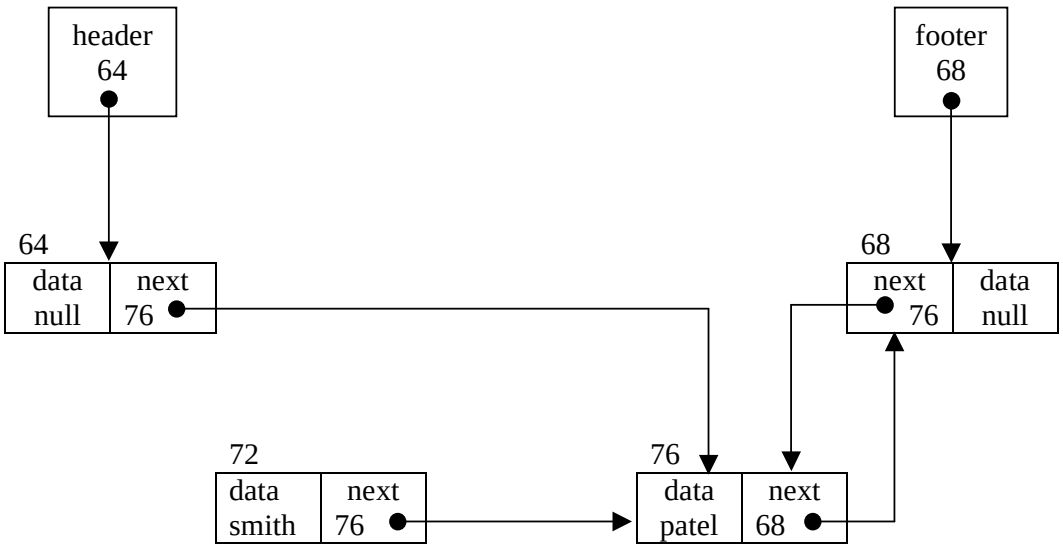
12.6 LEAVE

We start with a queue containing two items.



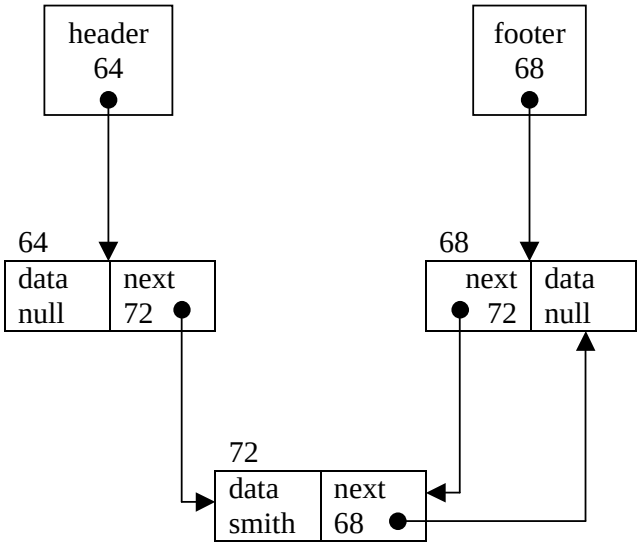
In order to remove an item we merely adjust the value contained by *header.next* so that it bypasses the first node in the queue.

```
header.next = header.next.next;
```

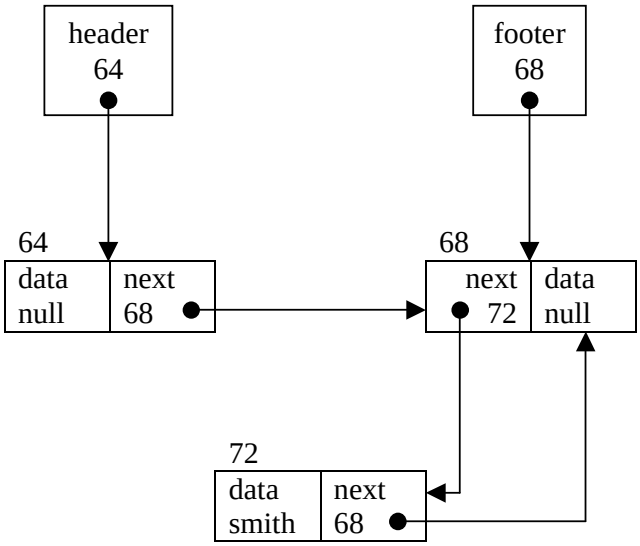


We rely on the Java garbage collection system to dispose of the dangling node (72).

It is always important to check that the boundary cases present no problems. We remove the item from a list that contains just one item.

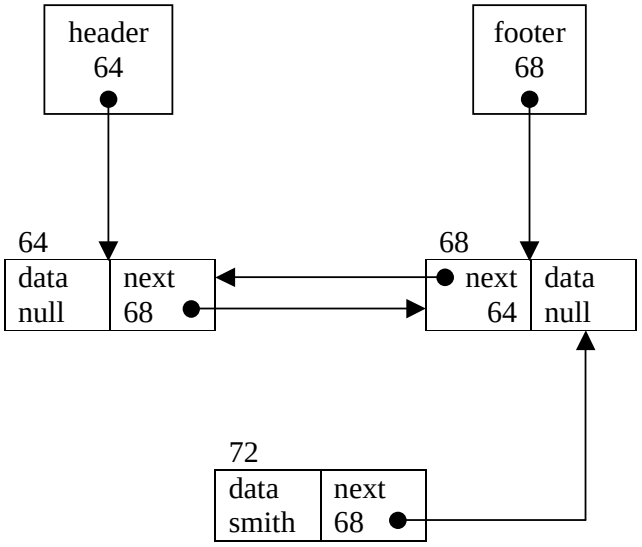


Updating *header.next* with the contents of *header.next.next* we get



This leaves *footer.next* referring to a deleted item. So

```
if (size == 0)
  footer.next = header;
```



As before, we rely on the Java garbage collection system to dispose of node 72.

```
public String leave()
{
  if (isEmpty())
    return "failure - queue empty";
  header.next = header.next.next;
  size--;
  if (size == 0)
    footer.next = header;
  return "success";
}
```

```
/* DynamicQueue.java
   Terry Marris 28 July 2001
*/

public class DynamicQueue implements Queue {
  private static class Node {
    Object data;
    Node next;

    Node(Object data, Node next)
    {
      this.data = data;
      this.next = next;
    }
  }


  private Node header;
  private Node footer;
  private int size;


  public DynamicQueue()
  {
    header = new Node(null, null);
    footer = new Node(null, null);
    header.next = footer;
    footer.next = header;
    size = 0;
  }


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


  public String join(Object obj)
  {
    Node node = new Node(obj, footer);

    footer.next.next = node;
    footer.next = node;
    size++;

    return "success";
  }
```

```
  public String leave()
  {
    if (isEmpty())
      return "failure - queue empty";
    header.next = header.next.next;
    size--;
    if (size == 0)
      footer.next = header;
    return "success";
  }


  public Object retrieve()
  {
    if (isEmpty())
      return null;
    else
      return header.next.data;
  }


  public int size()
  {
    return size;
  }
}
```

```
/* TestDynamicQueue.java
   Terry Marris  28 July 2001
*/

public class TestDynamicQueue {
  public static void main(String[] s)
  {
    DynamicQueue queue = new DynamicQueue();
    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("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
Emptying the queue item by item ...
smith
singh
french
patel
jones
```

12.7 REVIEW

12.8 FURTHER READING

In the next lesson we look at lists.

12.9 EXERCISES

1(a) Draw a diagram to represent a queue as a linked list of nodes. Your list should contain two data items.

(b) Draw a sequence of diagrams, supported by descriptive prose and snippets of Java code, to show how a new data item may be added to the queue.

(c) Draw a sequence of diagrams, supported by descriptive prose and snippets of Java code, to show how an item may be removed from your queue.

2 Test all the methods defined in the *DynamicQueue* class.