Programming Languages and Techniques (CIS120)

Lecture 26

Mar 21, 2012

Encapsulation & Queues

Encapsulation

Object encapsulation

- Encapsulation preserves invariants about the state of the object value.
- Basic idea of OO programming: represent data structures by objects with protected state, encapsulated by methods.
- All modification to the state of the data structure must be done using methods defined in the class that created the object.
- Enforce encapsulation by not returning aliases. Make a COPY of internal data structures if necessary.

Mutable Queues

```
module type QUEUE =
sig
  type 'a queue
  val create : unit -> 'a queue
  val is_empty :
      ‘a queue -> bool
  val enq :
      'a -> 'a queue -> unit
  val deq : 'a queue -> ‘a

end
```

```
public interface Queue<E> {




    public boolean is_empty ();

    public void enq (E elt);

    public E deq ();

}
```

Queue Implementation

```
public class QNode<E> {
   public final E  v;
   public QNode<E> next;
   public QNode (E v0, QNode<E> next0) {
     v = v0;
     next = next0;
   }
}

public class QueueImpl<E> implements Queue<E> {
   private QNode<E> head;
   private QNode<E> tail;

   /** Determine if the queue is empty. */
   public boolean is_empty() { … }
   …
 }
```

What are the invariants for QueueImpl? How can QueueImpl preserve those inbvariants?

Code Walk Through

Queue.java, QueueImpl.java

Constructing an Object

Workspace | Stack | Heap

```
Queue<String> q =
  new QueueImpl<String>();
q.enq("a");
q.enq("b");
```

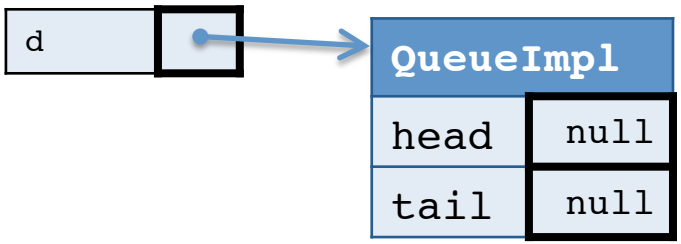
Adding to the queue

Workspace

```
q.enq("a");
q.enq("b");
```

Stack

Heap



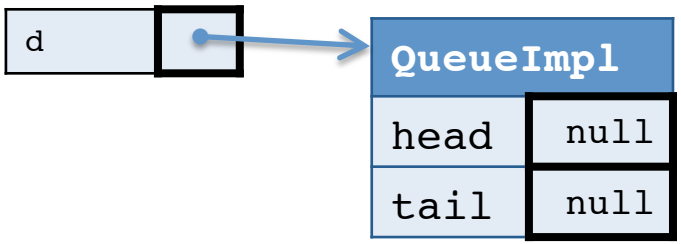
Adding to the queue

Workspace

```
q.enq("a");
q.enq("b");
```

Stack

Heap



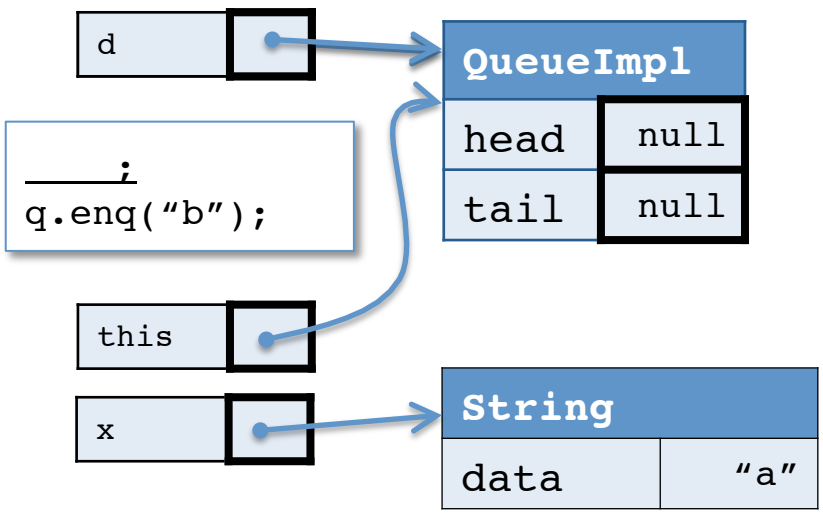
Adding to the queue

Workspace

```
QNode<E> newnode =
  new QNode<E>(x, null);
if (this.tail == null) {
   this.head = newnode;
   this.tail = newnode;
} else {
   this.tail.next
     = newnode;
   this.tail = newnode;
}
```

Stack

Heap



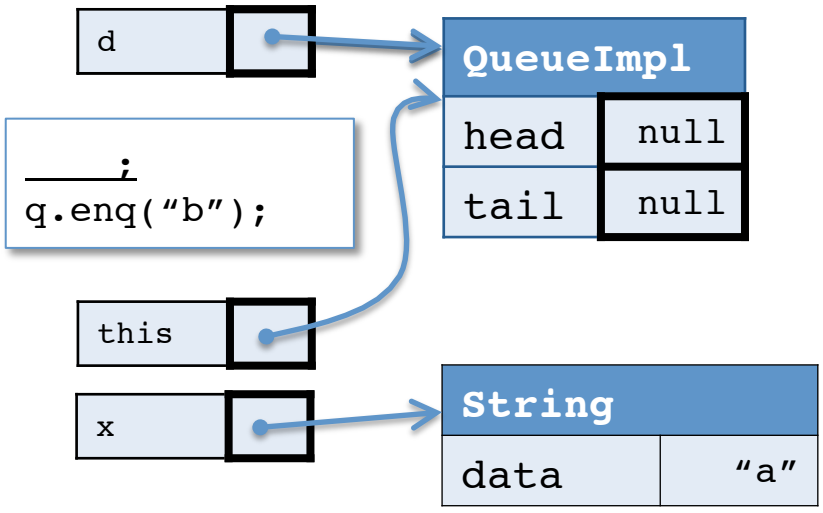
Adding to the queue

Workspace

```
QNode<E> newnode =
  new QNode<E>(x, null);
if (this.tail == null) {
   this.head = newnode;
   this.tail = newnode;
} else {
   this.tail.next
     = newnode;
   this.tail = newnode;
}
```

Stack

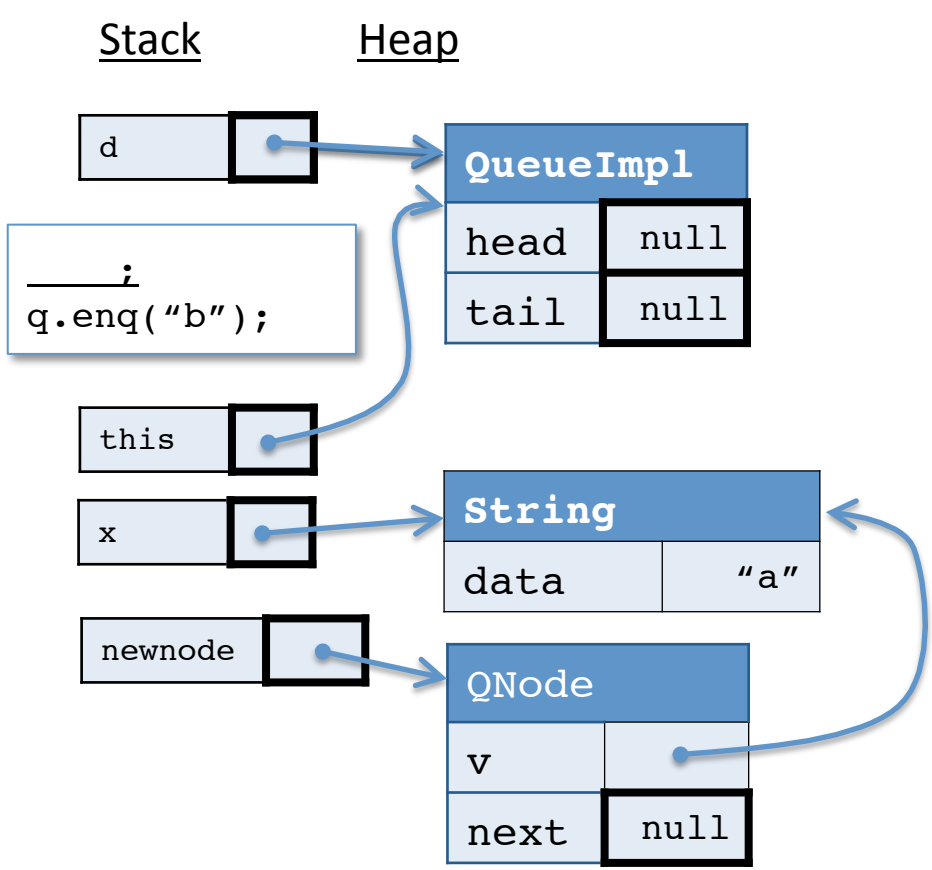
Heap



Adding to the queue

Workspace

```
if (this.tail == null) {
    this.head = newnode;
    this.tail = newnode;
} else {
    this.tail.next
      = newnode;
    this.tail = newnode;
}
```



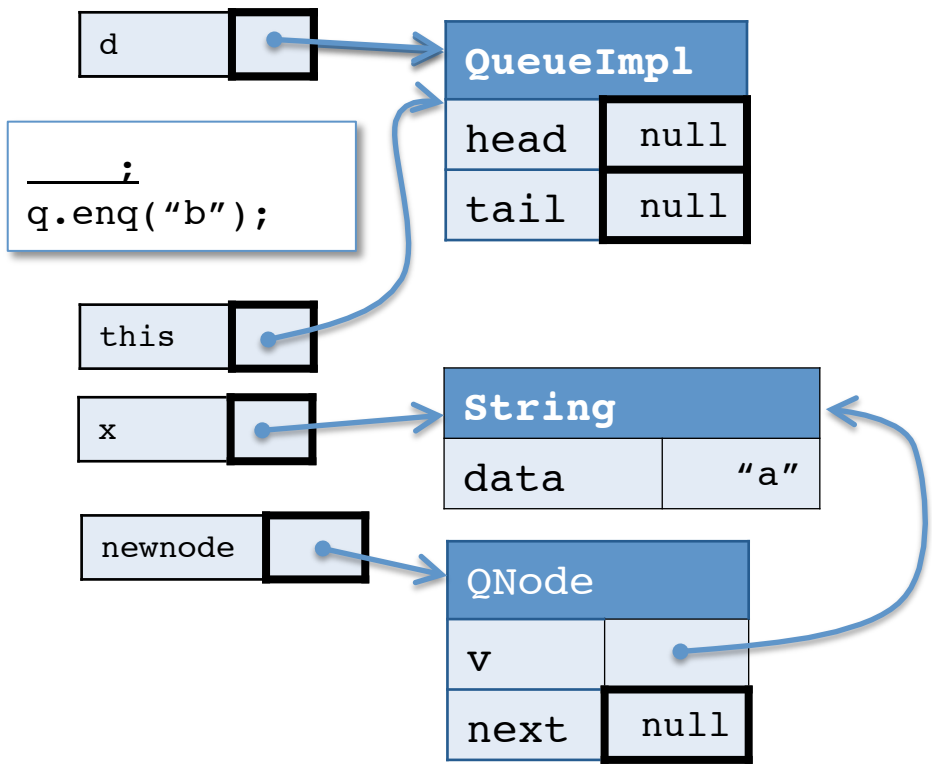
Adding to the queue

Workspace

```
if (true) {
    this.head = newnode;
    this.tail = newnode;
} else {
    this.tail.next
      = newnode;
    this.tail = newnode;
}
```

Stack

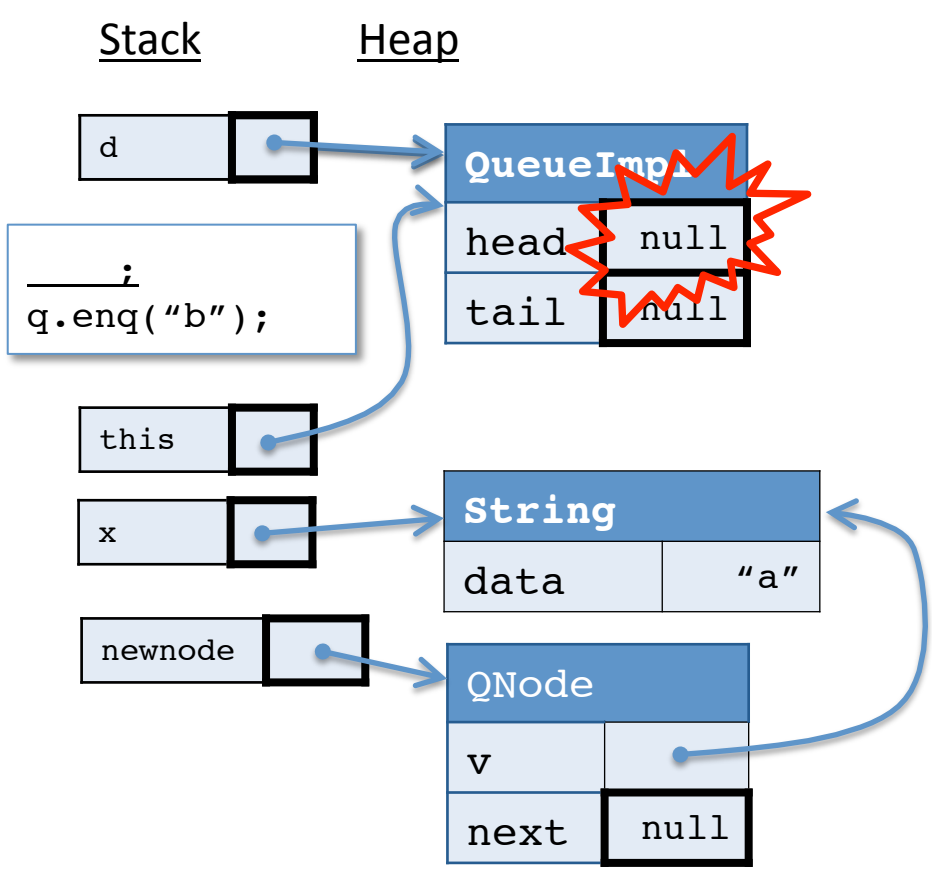
Heap



Adding to the queue

Workspace

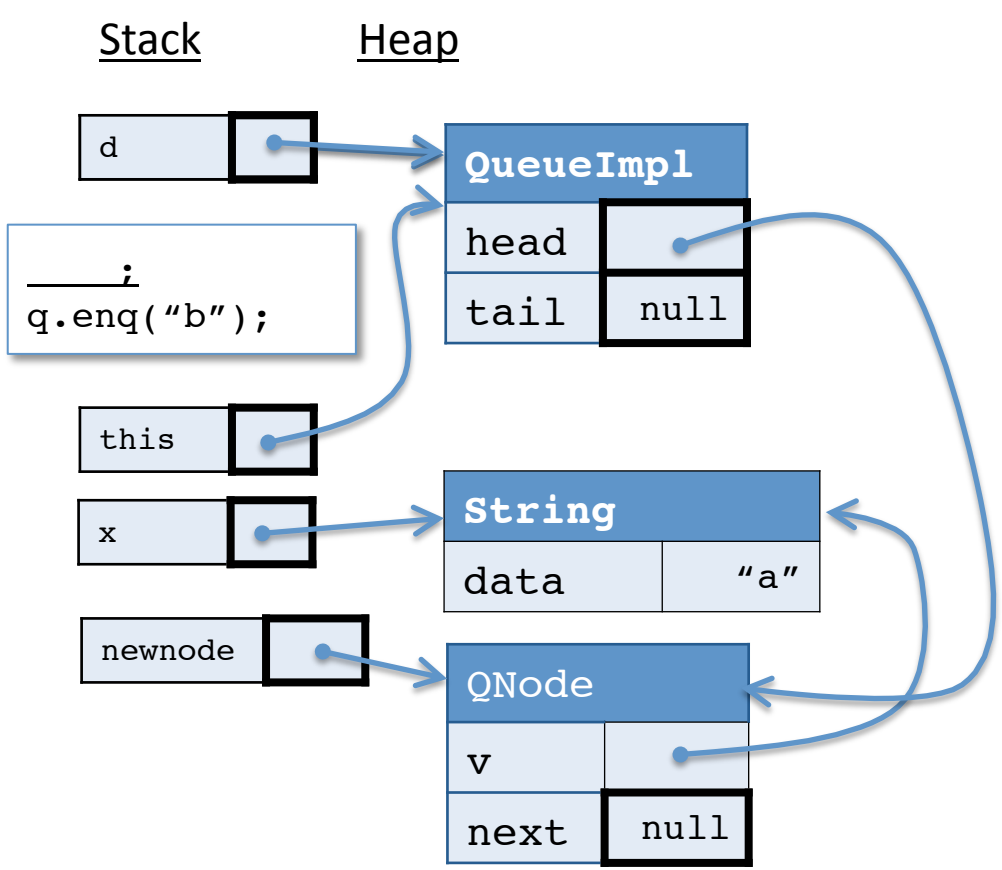
```
this.head = newnode;
this.tail = newnode;
```



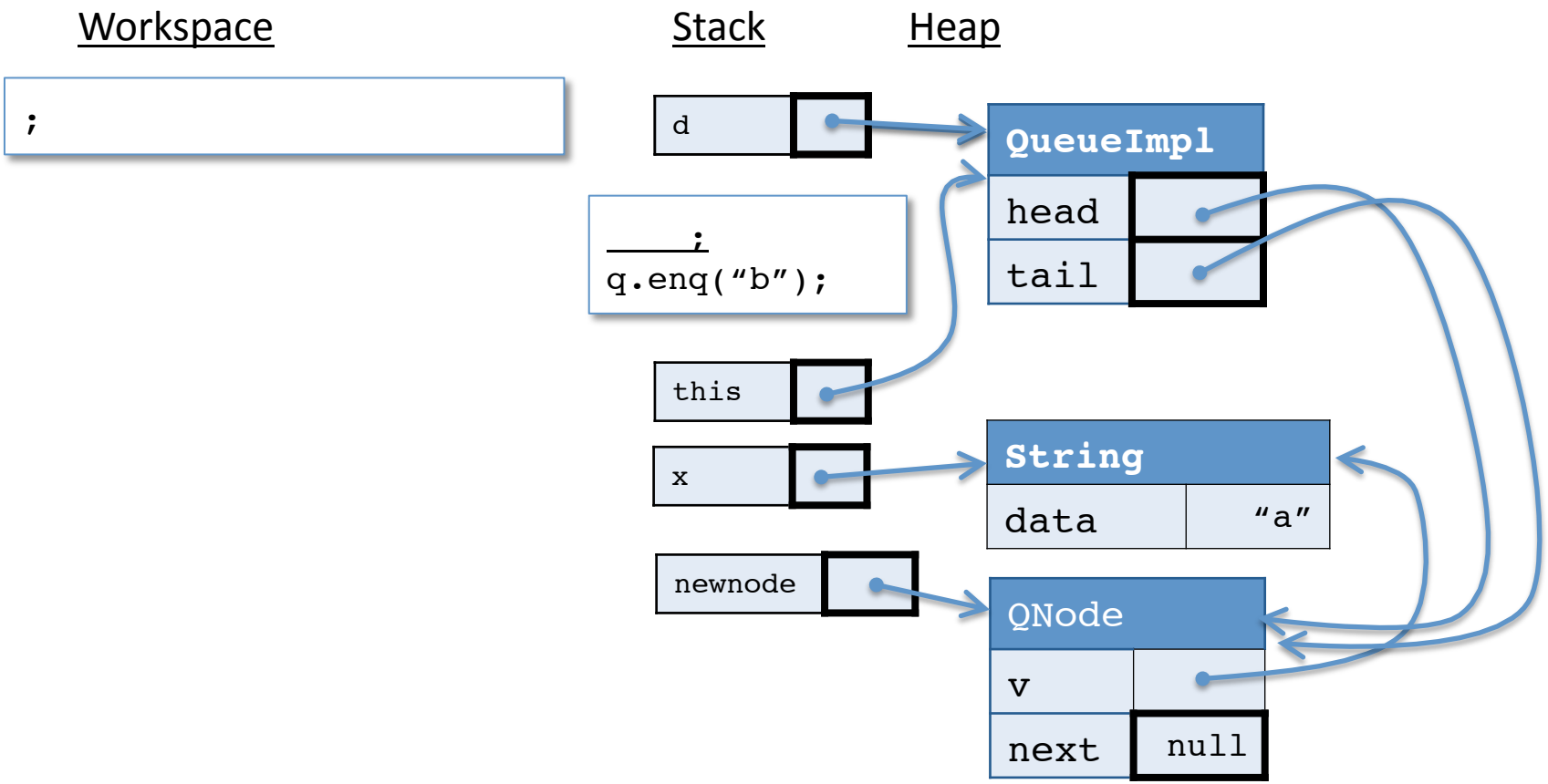
Adding to the queue

Workspace

```
this.tail = newnode;
```



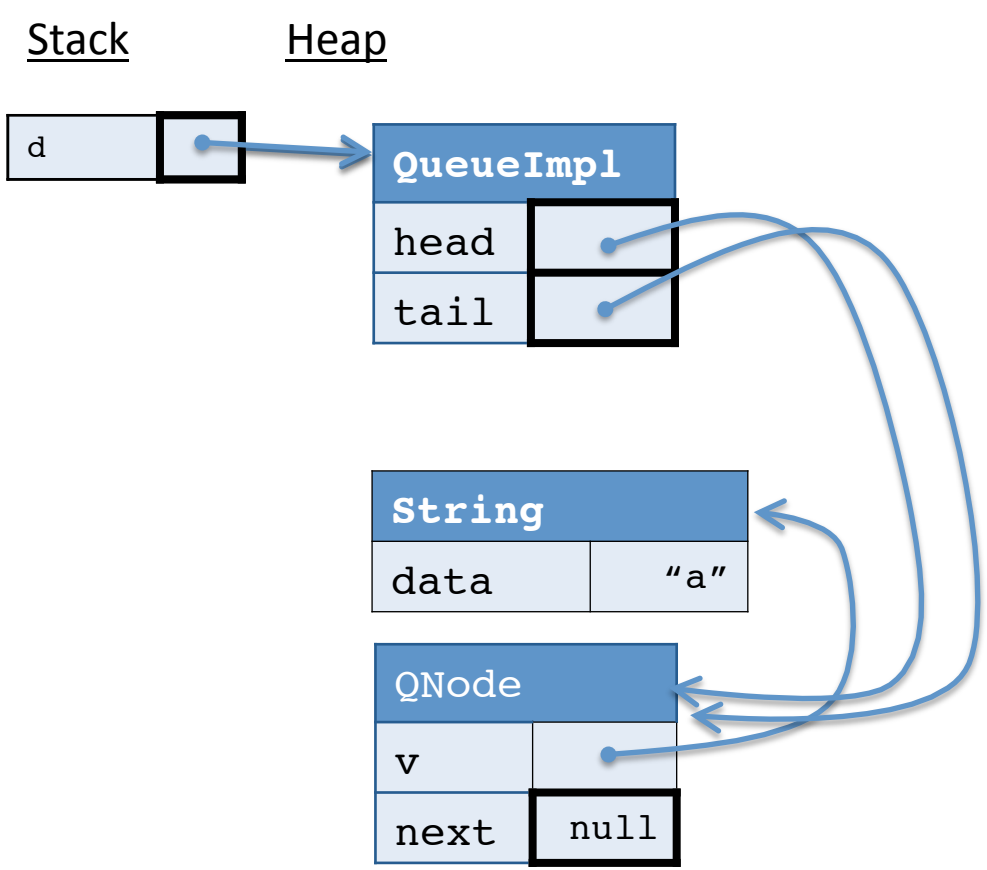
Adding to the queue



Adding to the queue

Workspace

```
q.enq("b");
```

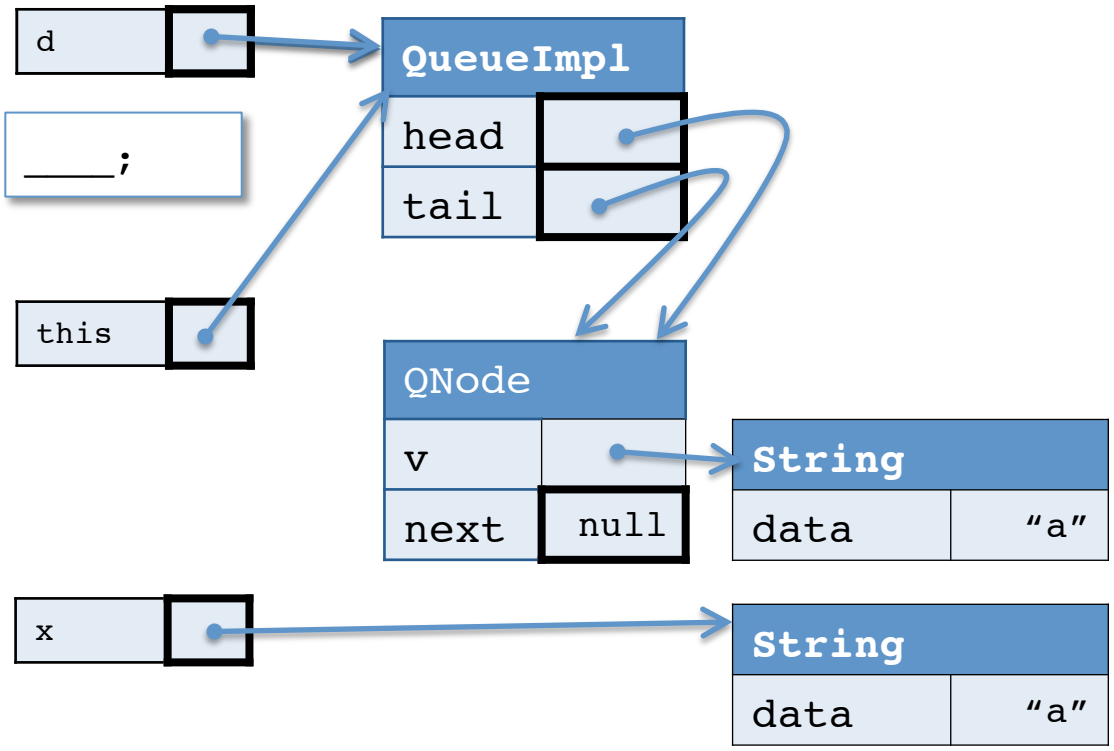


Untangle the heap

Workspace

```
QNode<E> newnode =
  new QNode<E>(x, null);
if (this.tail == null) {
   this.head = newnode;
   this.tail = newnode;
} else {
   this.tail.next
     = newnode;
   this.tail = newnode;
}
```

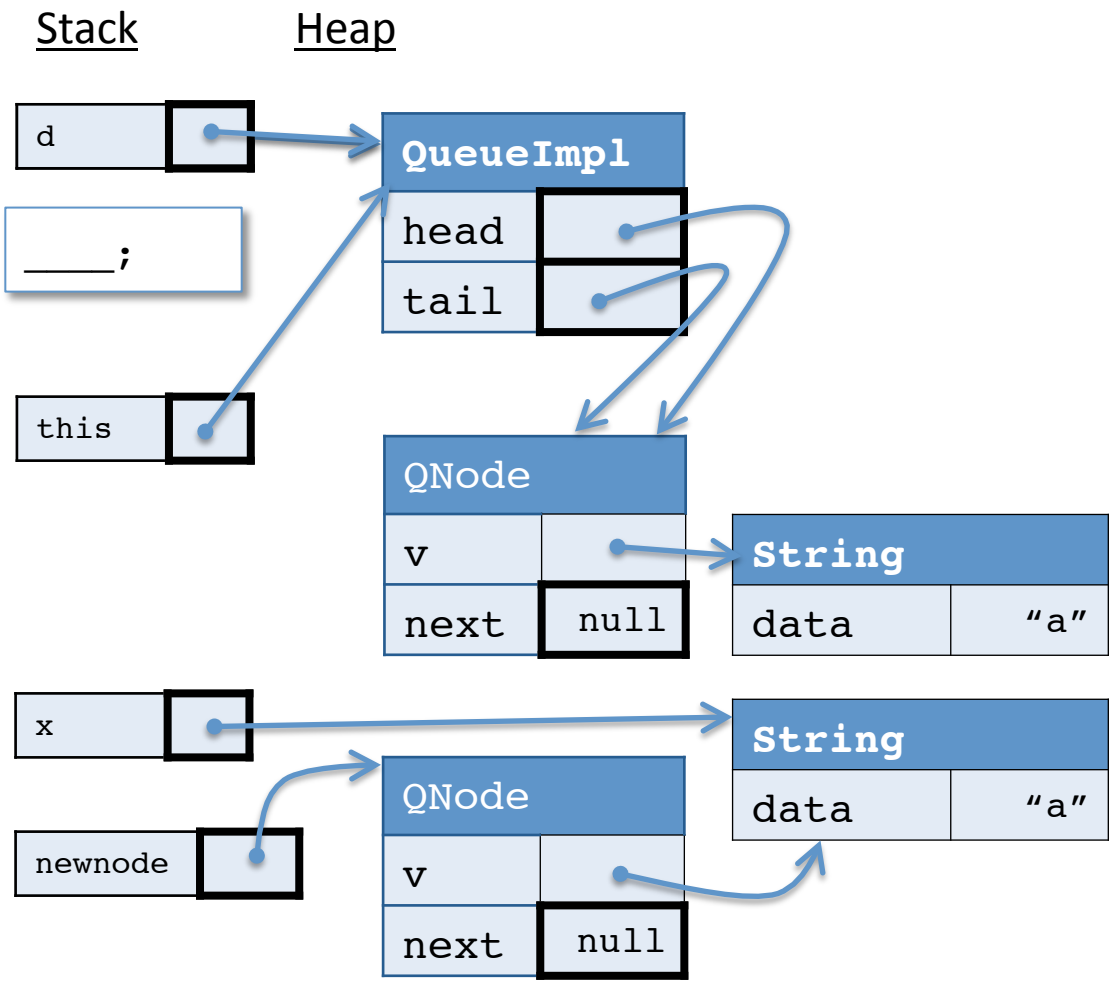
Stack Heap



Add the second value

Workspace

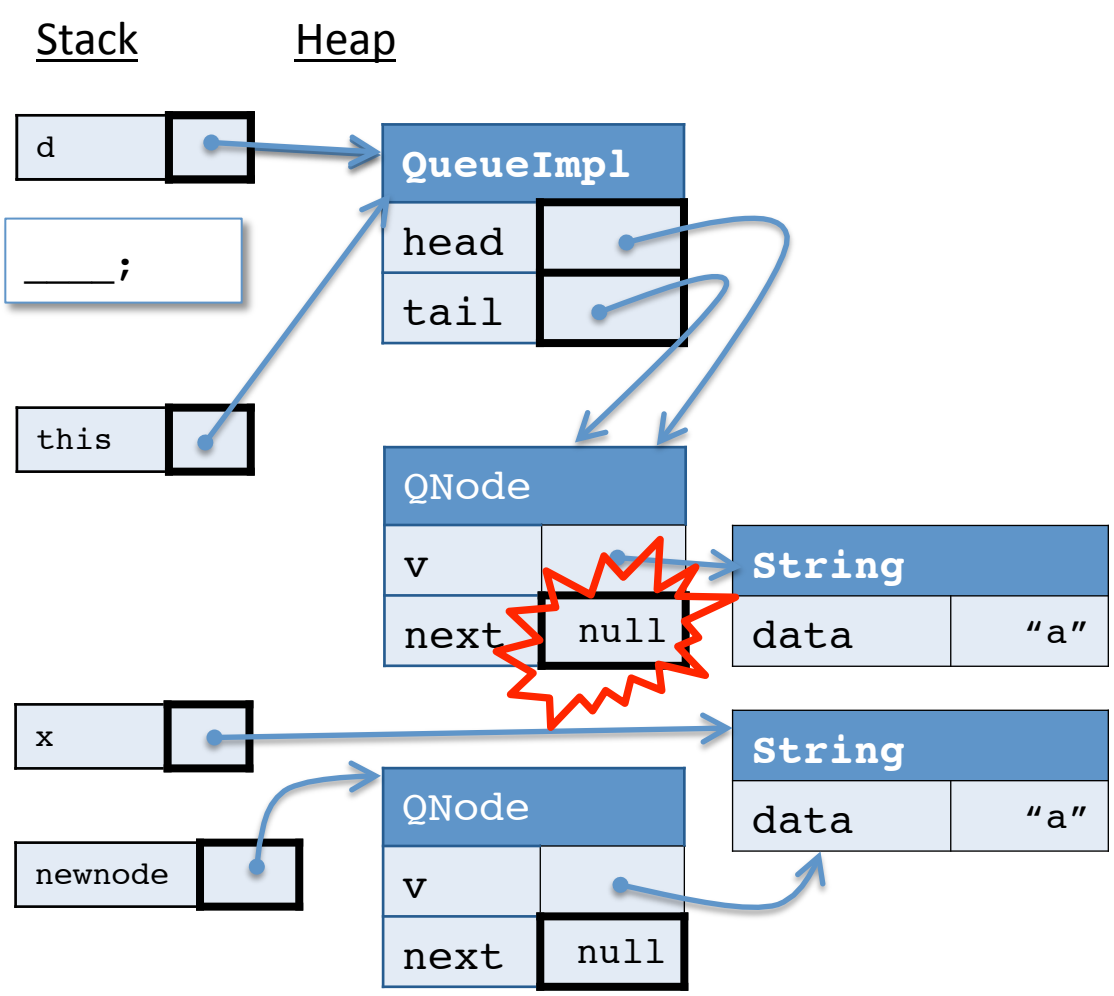
```
if (this.tail == null) {
    this.head = newnode;
    this.tail = newnode;
} else {
    this.tail.next
      = newnode;
    this.tail = newnode;
}
```



Add the second value

Workspace

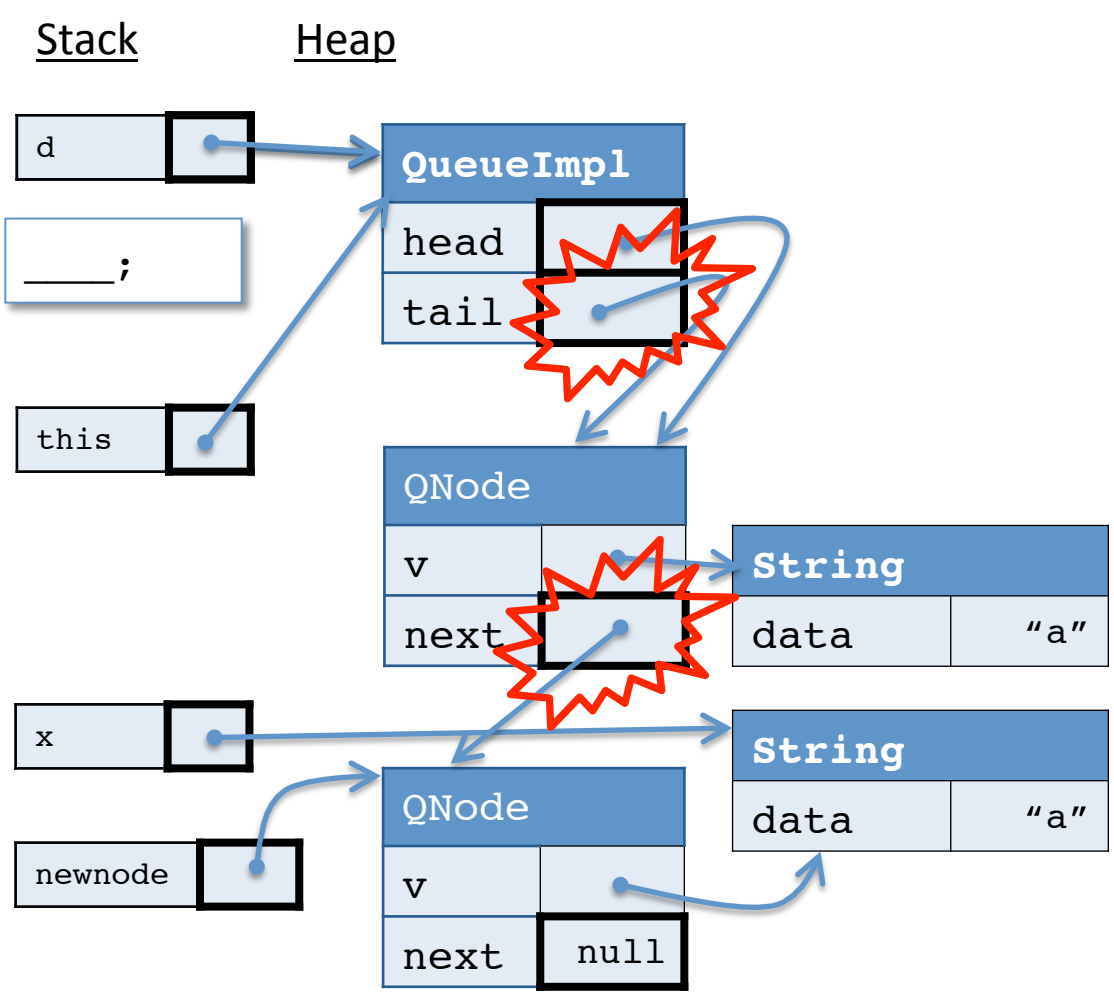
```
this.tail.next = newnode;
this.tail = newnode;
```



Add the second value

Workspace

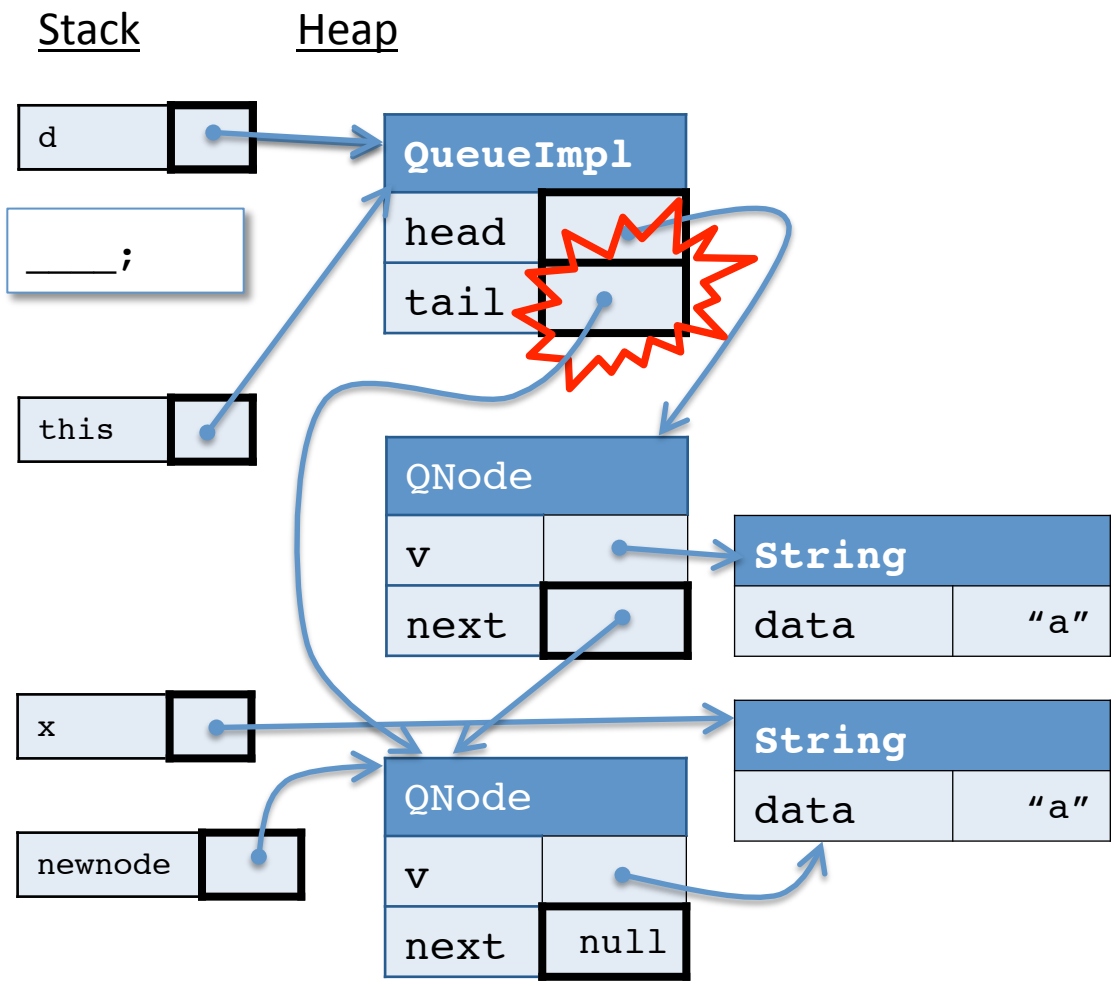
```
this.tail = newnode;
```



Add the second value

Workspace

```
;
```

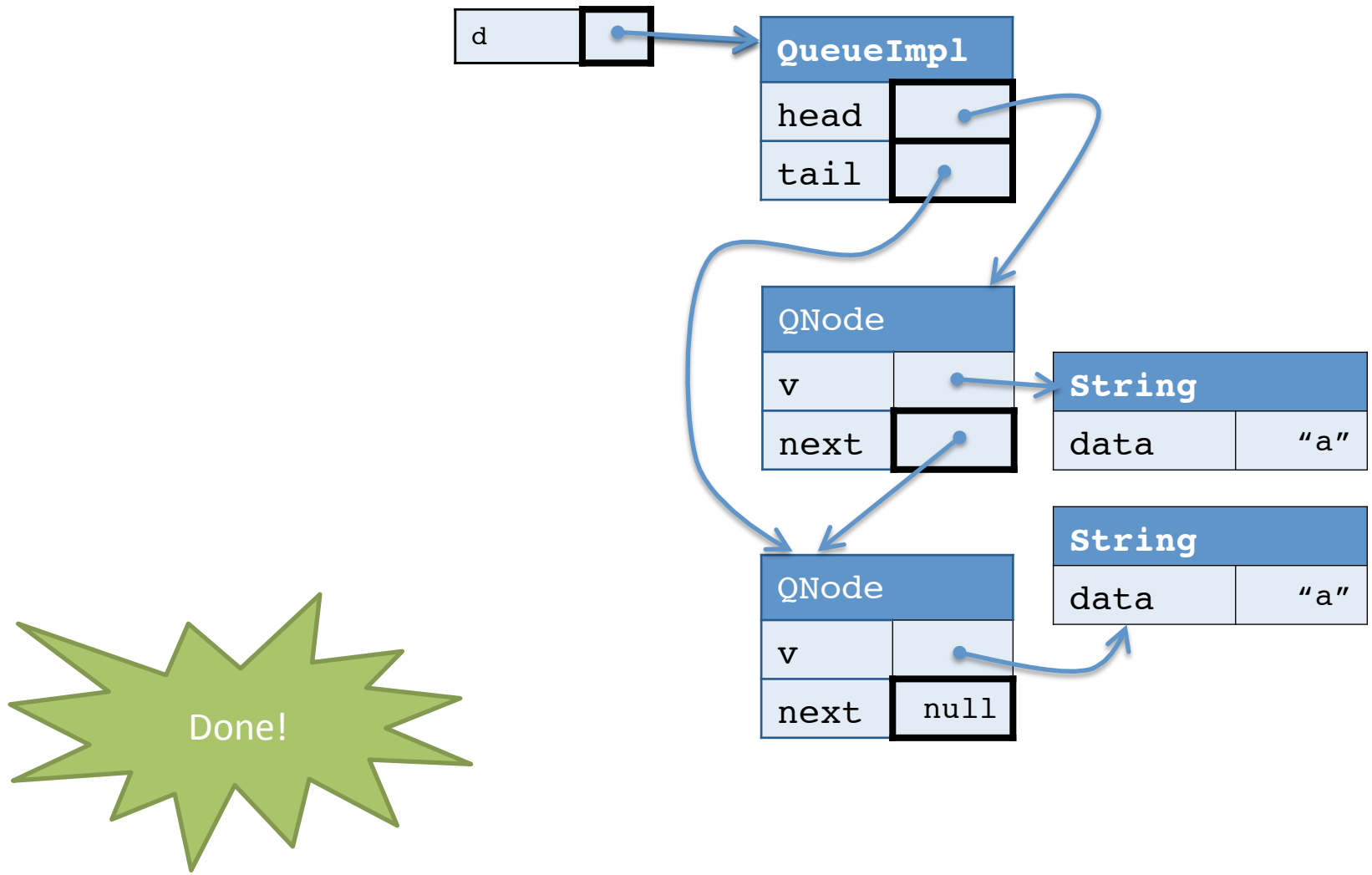


Add the second value

Workspace

Stack

Heap



What could go wrong?

- The QNode class has no protection for its state.

- It is the responsibility of the QueueImpl class to encapsulate the state of the queue.

- The QueueImpl class would fail to encapsulate this state if:
  - The head and tail fields were not private

    q.tail.next = q.head;
  - *Any* method in this class returned a reference a queue node in the linked list

    QNode<String> x = q.m();

    x.next = x;