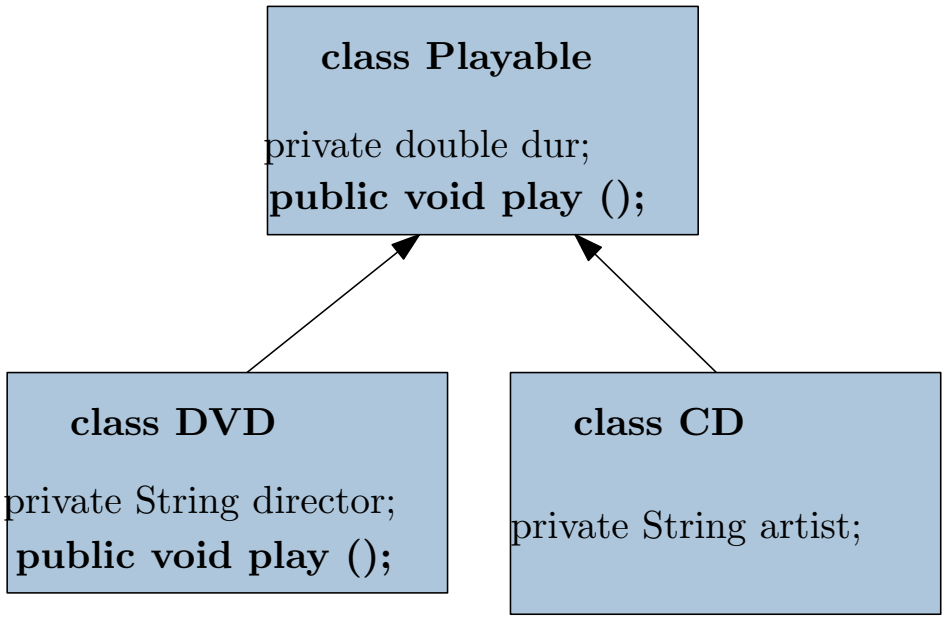
Linked List and Queue

Daniel Archambault

Previously in CS-115



Inheritance and Generics

Previously in CS-115

- What is the main advantage of using a generic?

Previously in CS-115

- What is the main advantage of using a generic?
- What is a type parameter?

Previously in CS-115

- What is the main advantage of using a generic?
- What is a type parameter?
- Can we use simple types as a type parameter?

Previously in CS-115

- What is the main advantage of using a generic?
- What is a type parameter?
- Can we use simple types as a type parameter?
- Is `T t = new T();` valid?

Previously in CS-115

- What is the main advantage of using a generic?
- What is a type parameter?
- Can we use simple types as a type parameter?
- Is `T t = new T();` valid?
- Can we have type parameters on specific methods?

Previously in CS-115

- It's time to build towards data structures

Linked Lists and Queue

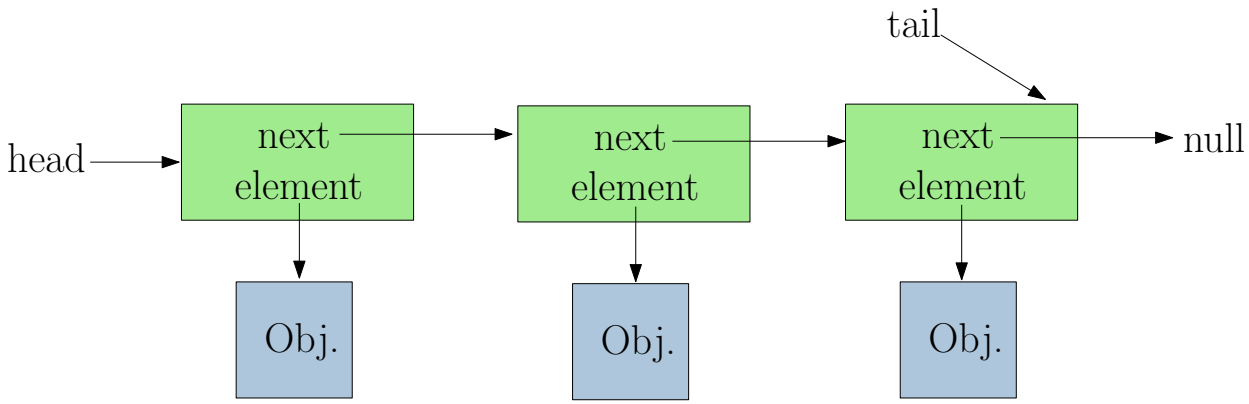
List ADT

- Types of operations on data, typically:
  - Add data to the list
  - Remove elements from the list
  - Query the list
- Can use an array for this but a linked list has advantages

Link List as a Chain...

- Links are nodes
  - a reference to the rest of the chain (join links)
  - a value at the current link
- Think of a chain with a direction
- Alternative to an array

Linked List Data Structure




```
public class Link
{
   private Link next;
   private Object element;
}

public class LinkedList
{
   private Link head;
   private Link tail;
}
```

Linked List Navigation

- Navigation is entirely based on references
- Indexes have less meaning
- Objects in list only make sense in terms of relative position

How do we look something up?

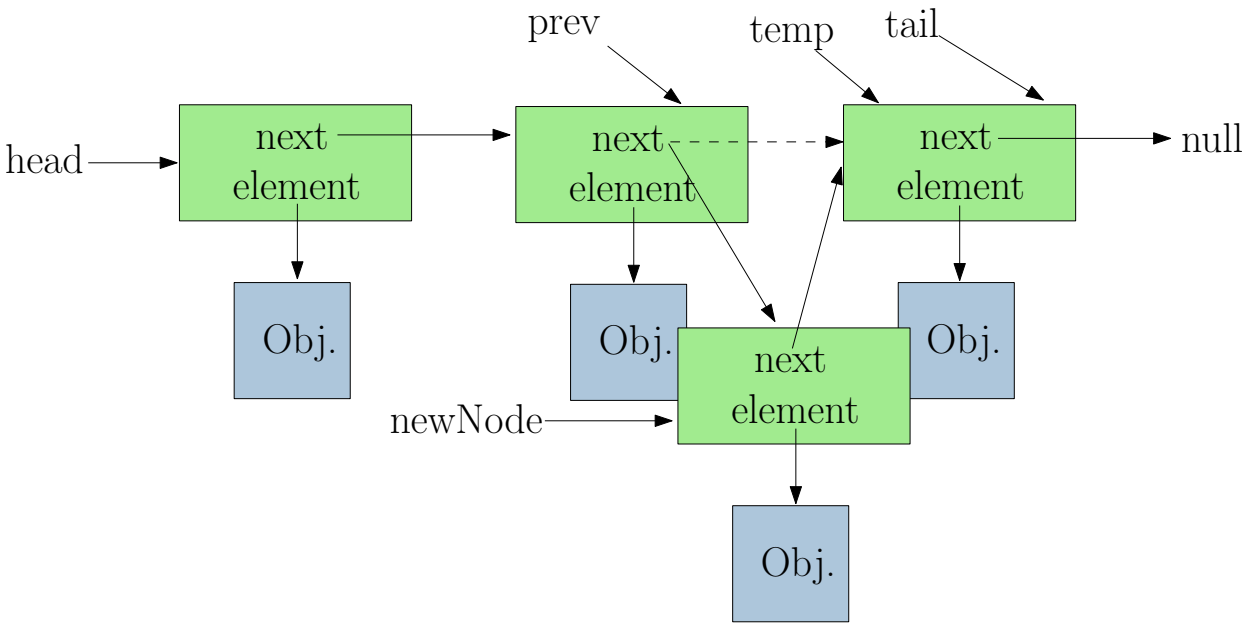
```
public class LinkedList {
   ...
    public Object getItem (int index)
   {
      Link curItem = head;
      for (int i = 0; i < index; ++i)
      {
         curItem = curItem.next;
      }
      return curItem.element;
   }
   ...
}
```

- Follow the chain from the head!
- We are slower than an array for random access
  - What do we do in the array case?

Where are link lists useful?

- We are slower for lookups in the general case
- However, we are better for inserts and deletes
- Why?
  - the position of an element only depends on its neighbours
  - the position is not absolute
  - the size of the list is not absolute

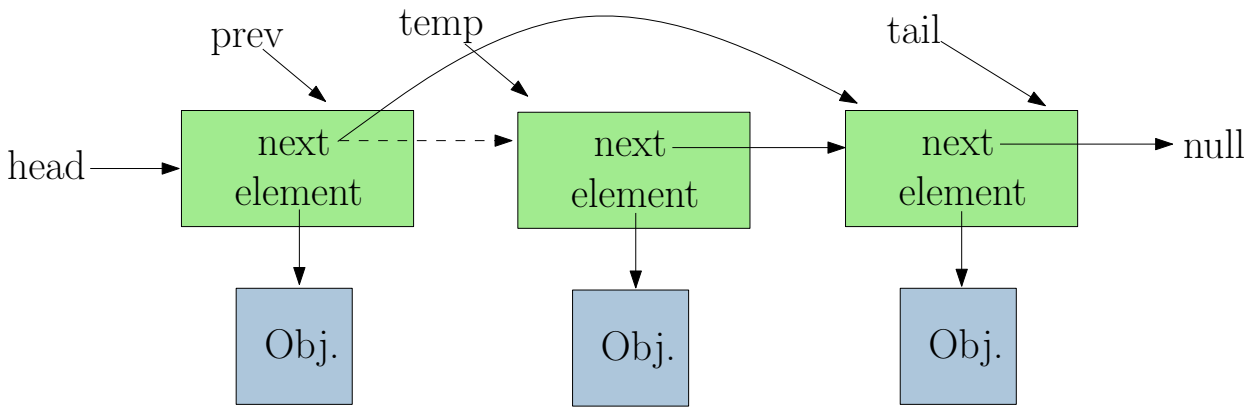
Insertion in Linked List




```
...
//prev is the reference before and next is the reference after
Link temp = prev.next;
prev.next = newNode;
newNode.next = temp;
...
```

- Once have position, insertion is low cost
- Array implementation? Must move everyone over

Deletion in Linked List



...
//prev is the node before and next is the node after
Link temp = prev.next;
prev.next = temp.next;
...

- Once have the position, deletion is low cost
- Array implementation?
  - What happens when the list gets small?

Advantages/Disadvantages

- Linked list advantage/disadvantage
  - slower accesses in the general case
  - cheap insertion/deletion in the general case
  - store only what we need
- Array advantage/disadvantage
  - fast accesses in the general case
  - slower insertion or deletion in the general case
  - sometimes require more storage than we need

Queue ADT

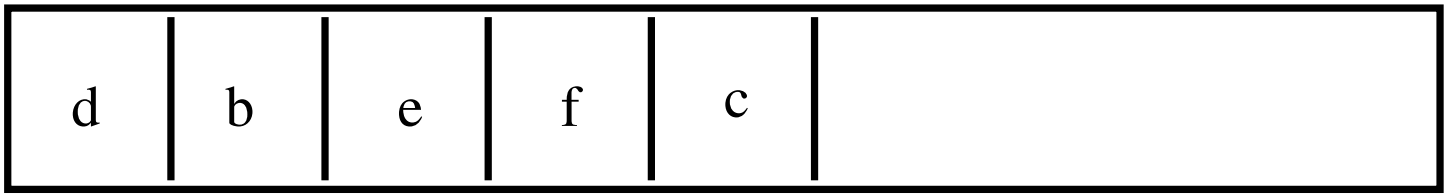
- *First in first out*

```
public interface Queue {
   public boolean isEmpty ();
   public void enqueue (Object newItem);
   public void dequeue ();
   public Object peek ();
}
```

isEmpty behaviour

- Returns true if there are no elements in the queue
- Otherwise, returns false

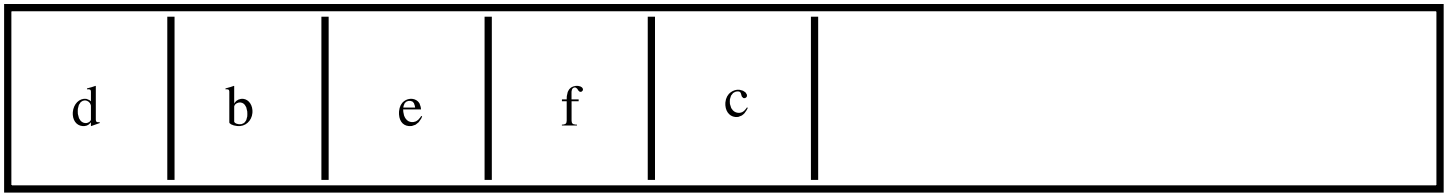
(a) true



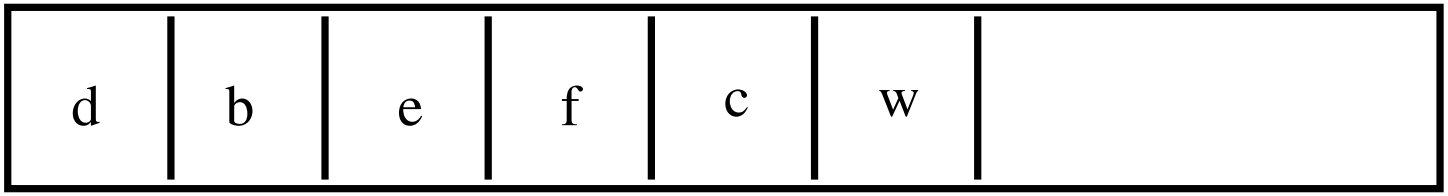
(b) false

enqueue behaviour

- Adds an item to the back of the queue



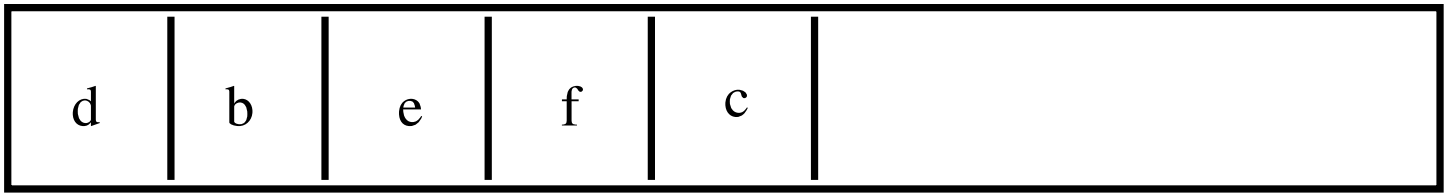
(c) before enqueue of w



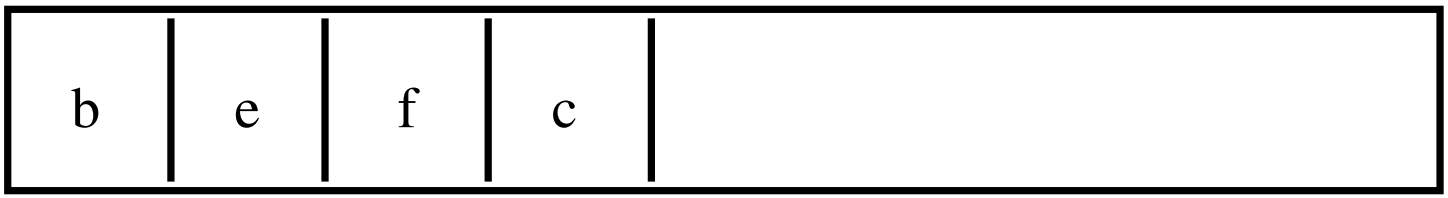
(d) after enqueue

dequeue behaviour

- Removes an item from the front of the queue



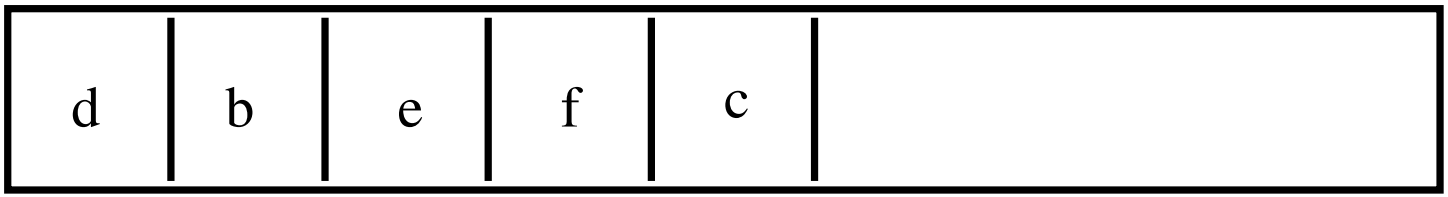
(e) before dequeue d



(f) after dequeue

peek behaviour

- Returns the front of the queue



(g) returns d

Implementation of Queue

- To turn a ADT Queue into a Queue data structure we can choose either an array or linked list
  - most natural implementation is linked list
- I'll explain this implementation with a linked list

Attributes of Queue Implemented with Linked List

```
public class Queue {
   private Link head;
   private Link tail;

    public Queue () {
      head = null;
      tail = null;
   }
}
```

isEmpty implementation

- Simply check if the queue is empty
- Really, could just check head, but this prevents bugs

```
...
return ((head == null) && (tail == null));
...
```

enqueue implementation

- Adds an item to the back of the queue
- Simply move tail back one
- If tail is null, you need to set head and tail...

```
...
Link newNode = new Link (element, null);
tail.next = newNode;
tail = newNode;
...
```

dequeue implementation

- Removes an item from the front of the queue
- Simply remove the link in front

...
head = head.next;
...

- Check if head is null. If so, set tail null (queue empty).

peek implementation

- Returns the front of the queue
- If the queue is empty, you need to throw an exception

...
return head.element;
...