PURDUE UNIVERSITY

CS18000: Problem Solving And Object-Oriented Programming

Linked Data Structures

21 March 2011

Prof. Chris Clifton

COMPUTER SCIENCES

Multiple Items: Beyond Arrays

```
interface Set<E> {
    boolean contains(E item);
      /* true iff ∃x s.t.
      item.equals(x) */
    void add(E item) throws
      SetFull;
      /* ensure contains(item);
      */
    void remove(E item);
      /* ensure !contains(item);
      */
}
```

```
abstract class ArraySet<T>
    implements Set<E> {
    E[ ] items;
    public boolean contains(E item)
      {
        for (int i=0; i<items.length; ++i) {
            if ( item.equals(items[i]) ) return
              true;
        }
        return false;
    }
    abstract public void add(E item)
      throws SetFull;
    abstract public void remove(E
      item);
}
```

Sidebar: What is this <E>?

- Generics: When you want to handle multiple types
  - Set<Person> s = new ArraySet<Person>(11);
  - ArraySet<String> ss = new ArraySet<String>(7);
  - ss.add("A String"); okay; s.add("A String") not
- A powerful data abstraction tool
  - We'll see more of it later

Sidebar: What is an *Abstract* Class?

- Interface on Steroids
  - Declares methods, constants
    - Just like an interface
  - Implements some methods
  - Defines some fields
- Can't instantiate
  - Since you can't call the unimplemented methods
  - You can define a constructor method!
- Extend it just like any other class
  - Must implement abstract methods

Why do we need Interfaces?

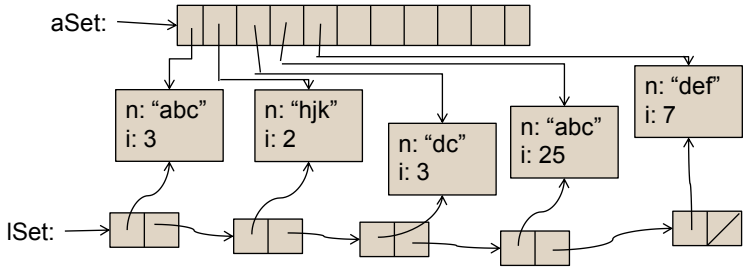
- Isn't an interface just an abstract class?
  - With every method abstract?
  - Yes … and no
- A class can only *extend* a single class
  - True for abstract classes as well
- Can *implement* multiple interfaces
- Suppose I want a concurrent Set?
  - ~~class CSet extends thread, ArraySet<Recipe> { … }~~
  - class CSet extends thread implements Set<Recipe> { … }

Problems with Arrays

- Fixed size
  - Is it big enough?
  - Is it more than we need?
  - Can extend, but adds complexity
- Deletion
  - Removing items leaves "holes"
  - Can manage (e.g., null values), but again adds complexity

Solution: Linked Structures

- Array: List of pointers to contained objects



- Alternative: Object has pointer to next contained object!

What is the best reason to use an Array over a Linked List?

A. We have plenty of memory – we can always just make it really big, so it doesn't get full

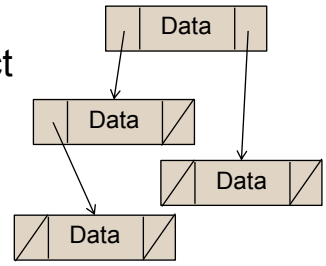
B. We can go straight to the *i*th item – a[i]

C. It uses less space since we don't have to keep a pointer to the next item

D. It is always better to use a Linked List

What goes in a Linked Data Structure?

- Contents of one "cell"
  - Instance variables for object
  - Pointer to an object
- Pointer to next object
  - Instance of your own class
  - Null value for this variable means "end"
  - Can even have multiple "next objects"



Accessing Linked Data Structures

- Iterator interface
  - next()
  - hasNext()
- Loop
  - while (hasNext()) next().processAnItem();
- Recursive
  - if (hasNext()) {
    - item = next();
    - item.processAnItem();
    - processRestOfItems();

Linked List Set

```
Class LinkedSet<E> implements
    Set<E> {
    private E current;
    private LinkedSet<E> next;

    boolean contains(E item) {
        if ( item.equals(current) ) return true;
        else {
            if (next == null) return false;
            else return next.contains(item);
    }
    void add(E item) {
        if (current == null) current = item;
        else {
            LinkedSet<E> newItem = new
                LinkedSet<E>(item, next);
            next = newItem;
        }
    public LinkedSet() {
        // Creates an empty set.
        current = null;
        next = null;
    }
    private LinkedSet(E item,
        LinkedSet<E> next) {
        this.current = item;
        this.next = next;
    }
    void remove(E item) {
        if (item.equals(current)) current =
            null;
        if (next != null) next.remove(item);
    }
}
```


What is wrong?

- Preceding code works, but could be better
  - Remove leaves empty cells in list
  - Why not just take them out?
- Need to expose abstraction
  - Class aware of it's own structure
  - Access next directly

Linked List Set

```
public void remove(E item) {
    if (item.equals(current)) {
        if (next == null) current = null;
        else {
            current = next.current; // move next to self
            next = next.next; // "forget" next
            remove(item); // Try again
        }
    if (next != null) next.remove(item);
}
```


PURDUE UNIVERSITY

CS18000: Problem Solving And Object-Oriented Programming

Linked Data Structures

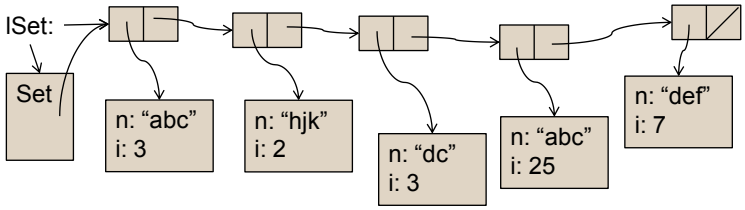
23 March 2011

Prof. Chris Clifton

COMPUTER SCIENCES

Linked Structures

- Problem: Can't remove first element
  - Would have to reveal "next"
  - Makes abstraction "dirty"



- Solution: Contain first element in abstraction

Linked List Set

```
class LinkedSet<E>
   implements Set<E> {

   private class
      LinkedSetElem<E> {
   // see next slide
   }

   private LinkedSetElem<E> ll
      = null;

   public boolean contains(E
      item) {
      if (ll == null) return false;
      else return ll.contains(item);
   }
   public void add(E item)  {
      ll = new
         LinkedSetElem<E>(item, ll);
   }
   public void remove(E item) {
      if (ll != null)
         ll = ll.remove(item);
   }
}
```

Linked List Set

```
private class
    LinkedSetElem<E> {
     private E value;
     private LinkedSetElem<E>
        next;

     public LinkedSetElem(E item,
        LinkedSetElem<E> next) {
          value = item;
          this.next = next;
     }
```

```
     public boolean contains(E item)
        {
          if ( item.equals(value) )
             return true;
          else if (next == null) return false;
          else return next.contains(item);
     }

     public LinkedSetElem<E>
        remove(E item) {
          if ( next != null )
             next = next.remove(item);
          if ( item.equals(value) )
             return next;
          else return this;
     }
}
```


Using the Set

- s = new LinkedSet<Integer>();
- s.add(1);
- s.add(2);
- s.add(3);
- shout(s.contains(3));
- shout(s.contains(4));
- s.add(2);
- s.remove(2);
- shout(s.contains(2));

- After s.add(2), 2 will be
  A. in front of 1
  B. behind 1
  C. in place of 1
- s.remove(2) is done
  A. when it finds the first 2
  B. when it finds the last 2
  C. when it gets to the end of the list
  D. never

Using the Set *concurrently*

Thread A

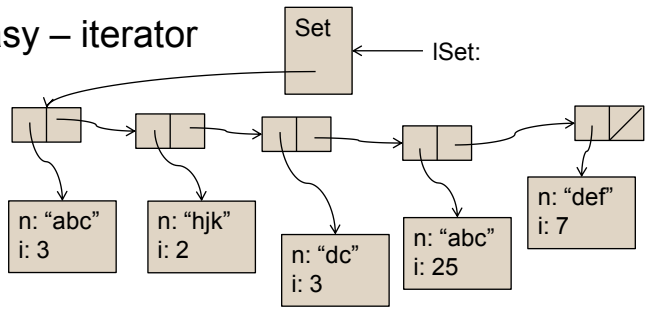
- s.add(4);
- s.remove(5);

Thread B

- s.add(5);
- s.add(6);

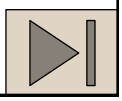
Iterators

- What if we want to traverse in order?
  - Easy – iterator



Iterator

- What is an Iterator?
- Interface supporting next() operation
  - Code can operate on collections by using next()
  - Can operate on any Iterator<E>
- To be used by such code, implement Iterator<E>



Iterator

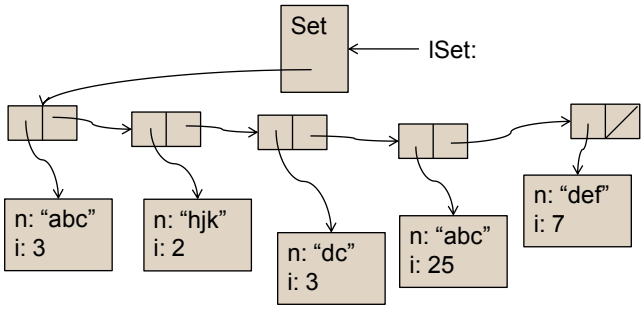
```
public class LinkedSetIterate<E> extends LinkedSet<E>
    implements java.util.Iterator<E> {
     private LinkedSetElem<E> current = this;

     public boolean hasNext() {
          return  current != null;
     }

     public E next() throws java.util.NoSuchElementException {
          if (! hasNext()) throw new java.util.NoSuchElementException();
          E result = current.value;
          current = currenr.next;
          return result;
     }
}
```

Other Data Abstractions

- How hard would a stack be?



- How about a queue?

Doubly-Linked List