Week 07

Announcements
- Program 2 Spec Posted this week
- Quiz on Recursion

Material
- Stack class
- Choice of data structures
Class Stack

- Definition
- Restricted list structure
- Dynamic LIFO Storage Structure
  - Size and Contents can change during execution
  - Last In First Out

![Diagram of a stack with insertion order I1, I2, I3 and top and bottom markers]

Only the top element may be accessed.

Insertion Order: I1, I2, I3
## Stack Operations

### Methods

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean empty()</td>
</tr>
<tr>
<td>Tests if this stack is empty.</td>
</tr>
<tr>
<td>Object peek()</td>
</tr>
<tr>
<td>Looks at the object at the top of this stack without removing it from the stack.</td>
</tr>
<tr>
<td>Object pop()</td>
</tr>
<tr>
<td>Removes the object at the top of this stack and returns that object as the value of this function.</td>
</tr>
<tr>
<td>Object push(Object item)</td>
</tr>
<tr>
<td>Pushes an item onto the top of this stack.</td>
</tr>
<tr>
<td>int search(Object o)</td>
</tr>
<tr>
<td>Returns the 1-based position where an object is on this stack.</td>
</tr>
</tbody>
</table>

Search returns the distance from the top of the stack of the occurrence nearest the top of the stack; the topmost item on the stack is considered to be at distance 1. The equals method is used to compare o to the items in this stack.
Iterative KnapSack

• Non-recursive knapsack solution

```java
final String NONE = "none";
final String INCLUDED = "included";
final String EXCLUDED = "excluded";

bool Knap (int ray[], int total, int start, int end)
{
    bool found = false;
    Stack StatusStack = new Stack(); // Stack of statuses
    String dummy = null;

    StatusStack.push( NONE );

    while ( ! StatusStack.empty() )
    {
        if ( (found) || (total == 0) )
        {
            // soln found
            found = true ;
            --start;
            dummy = (String) StatusStack.pop();
        }
    }
```
else if ( ((total < 0) && ((String) StatusStack.peek() == NONE))
    || (start > end) )
    { // no possible solution with current selections
        --start;
        dummy = StatusStack.pop();
    }
else // no soln yet
{ // consider status of current element
    if ((String) StatusStack.peek() == NONE)
    { //try including current array element
        //try including current array element
        total -= ray[start];
        dummy = (String) StatusStack.pop();
        StatusStack.push( INCLUDED );
        ++start;
        StatusStack.push( NONE );
    }
else if ((String) StatusStack.peek() == INCLUDED) {
    // try excluding current array element
    total += ray[start];
    dummy = (String) StatusStack.pop();
    StatusStack.push( EXCLUDED );
    ++start;
    StatusStack.push( NONE );
}
else {
    // give up on current element & current sum
    --start;
    dummy = (String) StatusStack.pop();
}

} // end Knap
Choice of Data Structures

- Arrays, Vectors, Lists, Bags, Sets, Stacks, and soon Queues

- Which one do you use?
  - Depends on needed functionality
  - Types of access: direct (or random) vs sequential
  - Size of collection: fixed or flexible

- Which one do you use to IMPLEMENT?
  - Depends on above
  - Keep in mind the separation of interface/implementation
public interface StackADT //L & C
{
    // Adds one element to the top of this stack
    public void push (Object element);

    // Removes and returns the top element from this stack
    public Object pop();

    // Returns without removing the top element of this stack
    public Object peek();

    // Returns true if this stack contains no elements
    public boolean isEmpty();

    // Returns the number of elements in this stack
    public int size();

    // Returns a string representation of this stack
    public String toString();
}
Stack

- One implementation (e.g. in Java API)
- Stack is defined as a subclass of Vector
- What does it mean to be a subclass?
- Is this appropriate?
  - think encapsulation
  - think contract-based programming
  - think of relation between stack and vector
Options

• Using inheritance
  ```java
class Stack extends Vector implements StackADT {
    public void push (Object element) { add(element); }  
    ...
}
• Using vector for implementation
  ```java
class Stack implements StackADT {
  private Vector vector;
  
  public void push (Object element) { vector.add(element); }  
    ...
}
• Which is more appropriate? Is this legal?
Stack p;
p.push("spring");
p.add("what?");
```