1 Abstract Data Types

- A model of a data structure
- Specifies what is stored, the operations that can be performed on the data, but not how it is stored

2 Stacks

- Represents a stack of items (just like a stack of books)
- Support two data modification operations: push and pop
- First in, last out (the first item to be pushed is at the bottom of the stack, and will be the last to be popped)
  - push places an item on the top of the stack
  - pop removes an item from the top of the stack (and returns it)
- May want to do other operations, such as finding the size, knowing whether the stack isEmpty, and looking at the top of the stack without removing it.

3 Stacks (with a Linked List)

```java
public class Stack {
    private IntList stack; // the first element is the top of the stack
    private int size;

    public Stack() {
        stack = null;
        size = 0;
    }

    public void push(int value) {
        stack = new IntList(value, stack);
        size++;
    }

    public int pop() throws EmptyStackException {
        if (size == 0)
            throw new EmptyStackException();
        int value = stack.data;
        stack = stack.next;
        size--;
        return value;
    }
}
```
public int size() {
    return size;
}

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

public int top() throws EmptyStackException {
    if (size == 0)
        throw new EmptyStackException();
    return stack.data;
}

4 Queue ADT

• Support two operations: enqueue and dequeue
• First in, first out (the first item to be inserted will be the first to be removed)
• Otherwise like a Stack

5 Announcements

• Homework 2 – submit only from one account for each partnership
• Feedback (send some in!) – also let me know about these lecture notes
• Reading – Catch up with past assigned reading
• Exam – Tuesday, July 9, 6:40 PM in 155 Dwinelle – if you need accommodations, speak to me today

6 Sequence ADT

• Represents an ordered group of items
• Items should be directly accessible by position (rank)

7 Vectors

• Implements the sequence ADT
• Idea: Direct access to elements like an array (fast), but resizable
• Approach: Use an array, and when it’s full, create a larger array, and copy the elements from the old one
• Fast, but some space wasted (where?)
• Available as java.util.Vector
Example

```java
public class Vector {
    private int size; // current num of elements in the Vector
    private Object[] elements; // the array of elements

    public Vector() {
        elements = new Object[2];
        size = 0;
    }

    public void add(Object o) {
        if (size >= elements.length)
            expand();
        elements[size] = o;
        size++;
    }

    public int size() {
        return size;
    }

    public Object get(int k) {
        return elements[k];
    }

    protected void expand() {
        Object[] big = new Object[elements.length*2];
        System.arraycopy(elements, 0, big, 0, size);
        elements = big;
    }
}
```

8 Question

How can we use a Vector to implement the Stack ADT (most efficiently)?

White: Place pushed items in the front of the Vector, pushing all other items to the right, and pop items from the end of the Vector.

Blue: Place pushed items at the end of the Vector, and pop items from the end of the Vector.

Yellow: Place pushed items at the end of the Vector, and pop items from the front of the Vector.

Green: A Vector cannot support any operations of the Stack ADT.

Pink: A Vector can only support pop, but not push of the Stack ADT.

9 Linked-lists

- Can be singly-linked or doubly-linked
- Do not allow direct (fast) access to elements – must iterate through the list to retrieve an element given an index
- Only needs space for the elements that are currently stored (unlike a Vector)
- Can be used to implement a Stack, Queue, or Sequence ADT

Example

```java
public class DIntList {
    private int data;
    private DIntList next, prev;

    public DIntList(int data, DIntList next, DIntList prev) {
        this.data = data;
        this.next = next;
        this.prev = prev;
    }

    public insertBack(int data) {
        if (next == null) {
            next = new DIntList(data, null, this);
        } else {
            next.add(data);
        }
    }

    /* other methods can be placed here */
}
```