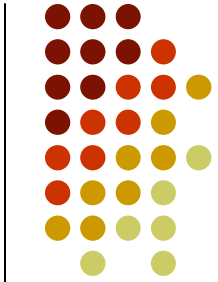
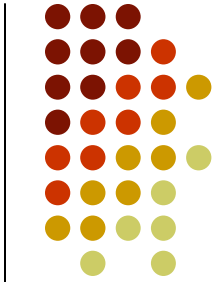
Stacks



- A stack is a data structure that only allows items to be inserted and removed at ***one end***
  - We call this end the ***top*** of the stack
  - The other end is called the bottom
- Access to other items in the stack is not allowed
- A **LIFO** (Last In First Out) data structure

Using a Stack

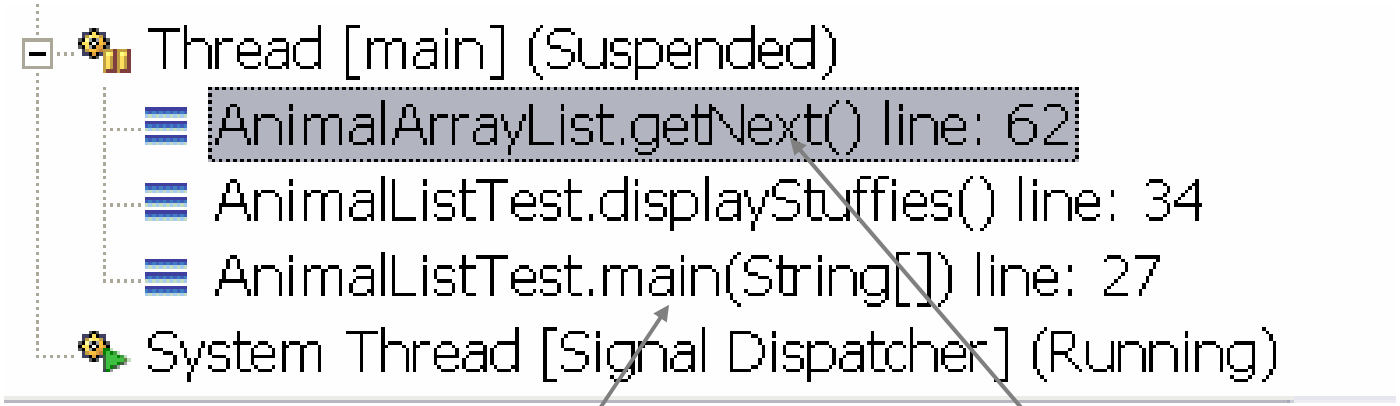
What Are Stacks Used For?



- Most programming languages use a “call stack” to implement function calling
  - When a method is called, its line number and other useful information are pushed (inserted) on the call stack
  - When a method ends, it is popped (removed) from the call stack and execution restarts at the indicated line number in the method that is now at the top of the stack

The Call Stack

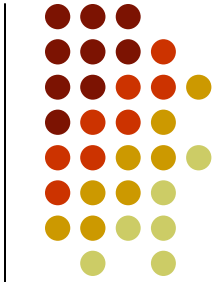
This is a display of the call stack (from the Eclipse Debug window)



Bottom of the stack: least recently called method

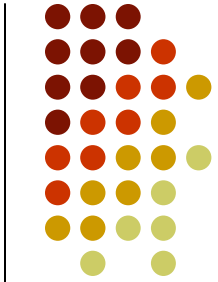
Top of the stack: most recently called method.

Call Stacks and Recursion



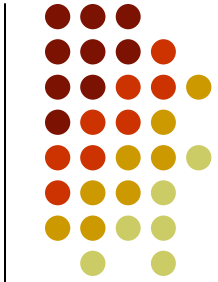
- A call stack is what makes recursion possible
- Stacks are also important when traversing tree data structures
  - They enable us to “backtrack” through the tree
  - We’ll see more about this later in the course

Stack Operations



- A stack should implement (at least) these operations:
  - **push** – insert an item at the top of the stack
  - **pop** – remove and return the top item
  - **peek** – return the top item (without removing it)
- These operations should be performed efficiently – in O(1) time

Axioms

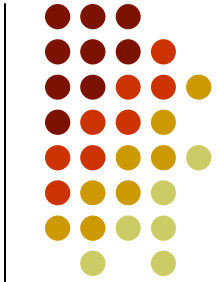


- Axioms are used to define an ADT formally
  - Example
    - Axiom to specify that the last item inserted into `stack` is the first item to be removed

      `(stack.push(newItem)).pop() = stack`

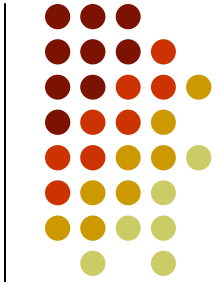
      `(stack.push(newItem)).peek() = newItem`

Stack Implementation



- The stack ADT can be implemented using a variety of data structures. We will look at two:
  - Arrays
  - Linked Lists
- Both implementations must implement all the stack operations

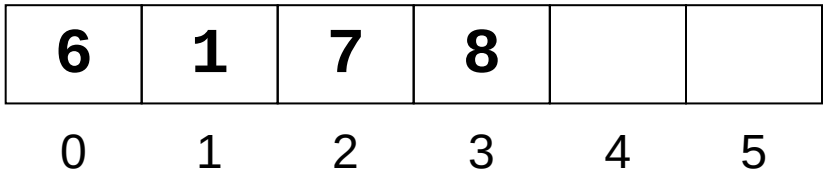
Stack: Array Implementation



- If an array is used to implement a stack what is a good index for the top item?
  - Is it position 0?
  - Is it position n-1?

- Note that push and pop must both work in O(1) time as stacks are usually assumed to be extremely fast
- The index of the top of the stack is the number of items in the stack - 1

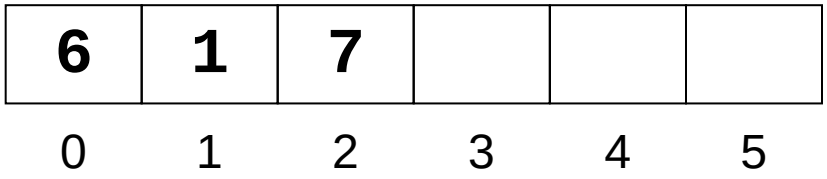
Array Stack Example



index of **top** is
current size – 1

```
//Java Code
Stack st = new Stack();
st.push(6); //top = 0
st.push(1); //top = 1
st.push(7); //top = 2
st.push(8); //top = 3
```

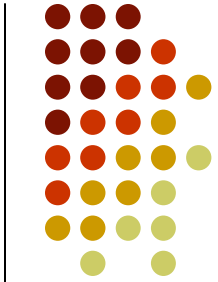
Array Stack Example



index of **top** is
current size – 1

```
//Java Code
Stack st = new Stack();
st.push(6); //top = 0
st.push(1); //top = 1
st.push(7); //top = 2
st.push(8); //top = 3
st.pop(); //top = 2
```

Array Implementation Summary



- Easy to implement a stack with an array
- `push` and `pop` can be performed in O(1) time
- But the implementation is subject to the limitation of arrays that the size must be initially specified because
  - The array size must be known when the array is created and is fixed, so that the right amount of memory can be reserved
  - Once the array is full no new items can be inserted
- If the maximum size of the stack is not known (or is much larger than the expected size) a dynamic array can be used
  - But occasionally push will take O(*n*) time