Stack Implementations

Chapter 6
Contents

• A Linked Implementation
• An Array-Based Implementation
• A Vector-Based Implementation
  ▪ Java Class Library: The Class `Vector`
  ▪ Using a Vector to Implement the ADT Stack
Objectives

• Implement ADT stack by using either
  ▪ Linked chain
  ▪ Array
  ▪ Vector
• Compare and contrast various implementations and their performance
Linked Implementation

• Consider push, pop, peek
  ▪ Each involves top of stack
• Best to put top of stack at head node
  ▪ Fastest, easiest to access
• Java will manage reclaiming memory without instruction from programmer
• Note source code for linked implementation
  ▪ Listing 6-1

Note: Code listing files must be in same folder as PowerPoint files for links to work
Figure 6-1 A chain of linked nodes that implements a stack
FIGURE 6-2 (a) A new node that references the node at the top of the stack; (b) the new node is now at the top of the stack
**Linked Implementation**

```java
public void push(T newEntry)
{
    Node newNode = new Node(newEntry, topNode);
    topNode = newNode;
} // end push

public T peek()
{
    T top = null;
    if (topNode != null)
    { // topNode is not null
        top = topNode.getData();
    }
    return top;
} // end peek

public T pop()
{
    T top = peek();
    if (topNode != null)
    { // topNode is not null
        topNode = topNode.getNextNode();
    }
    return top;
} // end pop
```
Figure 6-3 The stack (a) before the first node in the chain is deleted
Figure 6-3 The stack (b) after the first node in the chain is deleted
Array Based Implementation

• Again the question:
  ▪ Were to place the top entry?

• More efficient operations with bottom of stack at beginning of array
  ▪ Top of stack at last occupied entry

• Must consider memory wastage of unused array elements
Figure 6-4 An array that implements a stack; its first location references (a) the top entry in the stack;
Figure 6-4 An array that implements a stack; its first location references (b) the bottom entry in the stack;
Adding to the Top

```java
public void push(T newEntry)
{
    ensureCapacity();
    topIndex++;
    stack[topIndex] = newEntry;
} // end push

private void ensureCapacity()
{
    if (topIndex == stack.length - 1) // if array is full,
        // double size of array
        stack = Arrays.copyOf(stack, 2 * stack.length);
} // end ensureCapacity
```
public T peek()
{
    T top = null;
    if (!isEmpty())
        top = stack[topIndex];
    return top;
} // end peek
Figure 6-5 An array-based stack after its top entry is removed by (a) decrementing `topIndex`;
Figure 6-5 An array-based stack after its top entry is removed by (b) setting \texttt{stack[topIndex]} to \texttt{null} and then decrementing \texttt{topIndex}.
Method `Pop`

```java
public T pop()
{
    T top = null;
    if (!isEmpty())
    {
        top = stack[topIndex];
        stack[topIndex] = null;
        topIndex--;
    } // end if

    return top;
} // end pop
```
End

Chapter 6