Stack Implementations

Chapter 6

Data Structures and Abstractions with Java, 4e
Frank Carrano

Linked Implementation

- Each operation involves top of stack
  - `push`
  - `pop`
  - `peek`
- Head of linked list easiest, fastest to access
  - Let this be the top of the stack

Linked Implementation

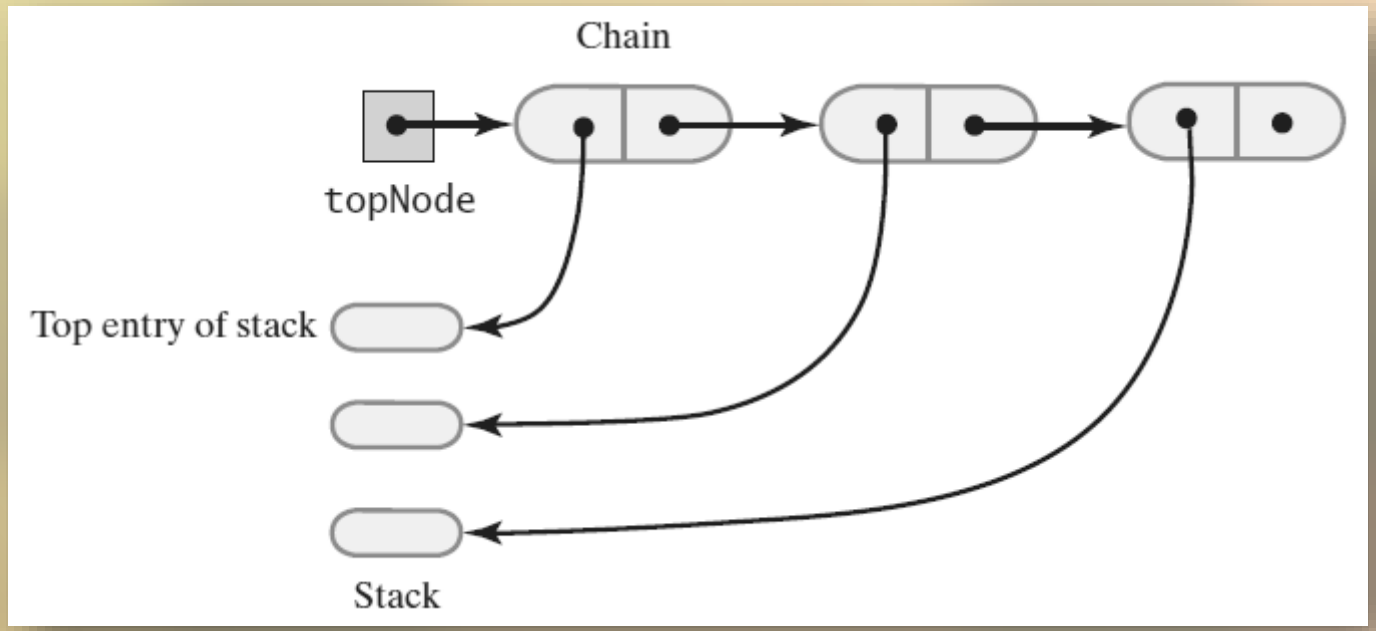

FIGURE 6-1 A chain of linked nodes that implements a stack

Linked Implementation

```
1  /**
2      A class of stacks whose entries are stored in a chain of nodes.
3      @author Frank M. Carrano
4  */
5  public final class LinkedStack<T> implements StackInterface<T>
6  {
7     private Node topNode; // References the first node in the chain
8
9     public LinkedStack()
10    {
11       topNode = null;
12    } // end default constructor
13    < Implementations of the stack operations go here. >
14    . . .
15
16    private class Node
17    {
18
```

LISTING 6-1 An outline of a linked implementation of the ADT stack

Linked Implementation

```
19        private T    data; // Entry in stack
20        private Node next; // Link to next node
21
22        < Constructors and the methods getData, setData, getNextNode, and setNextNode
23          are here. >
24    } // end Node
25 } // end LinkedStack
```

LISTING 6-1 An outline of a linked implementation of the ADT stack

Linked Implementation

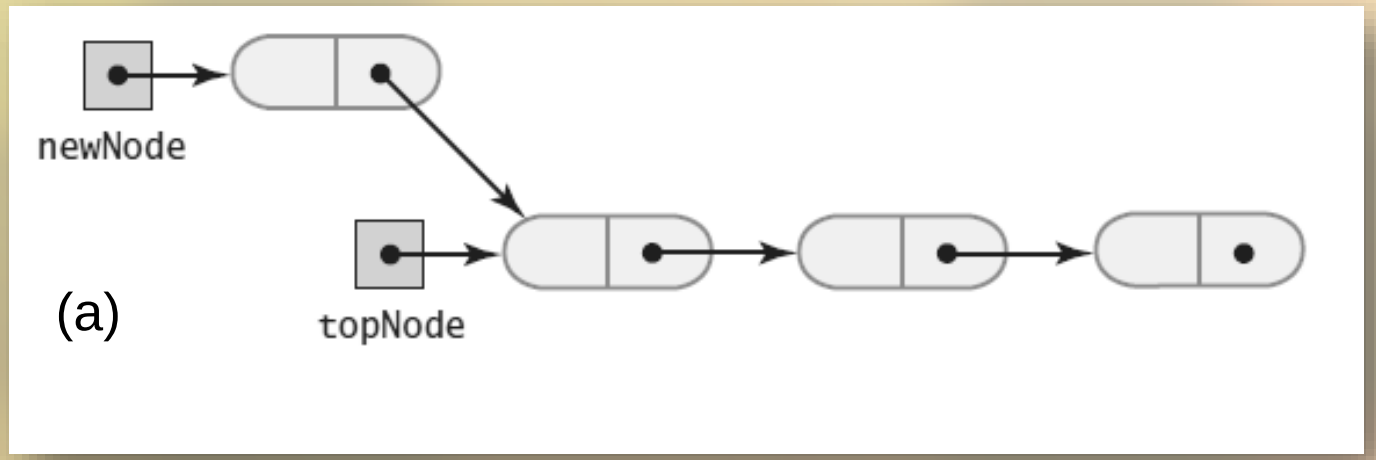

FIGURE 6-2 (a) A new node that references the node at the top of the stack;

Linked Implementation

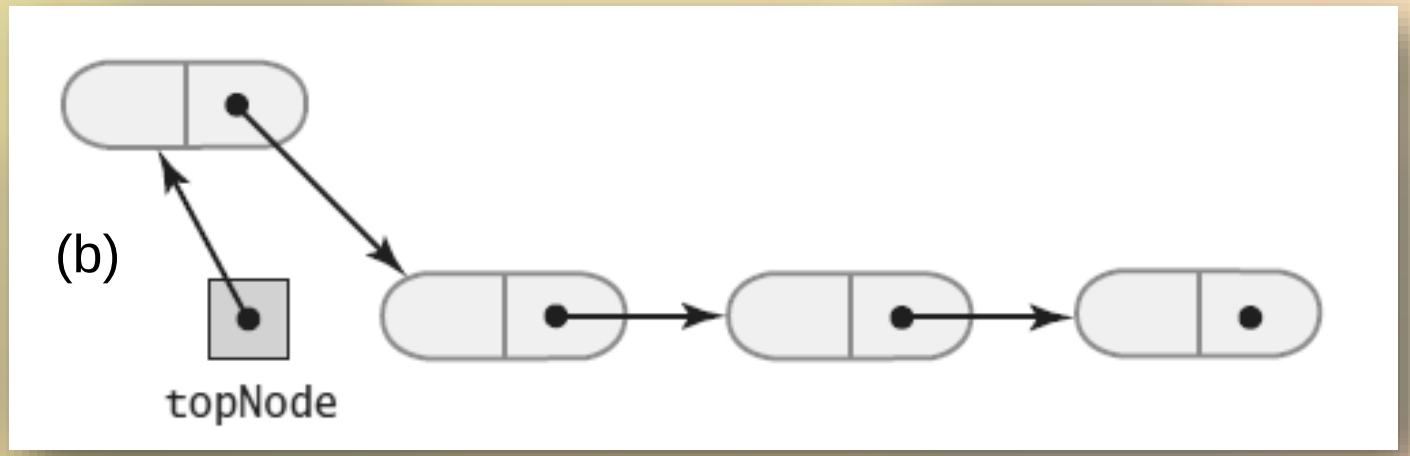

FIGURE 6-2 (b) the new node is now at the top of the stack

Linked Implementation

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

Definition of **`push`**

Linked Implementation

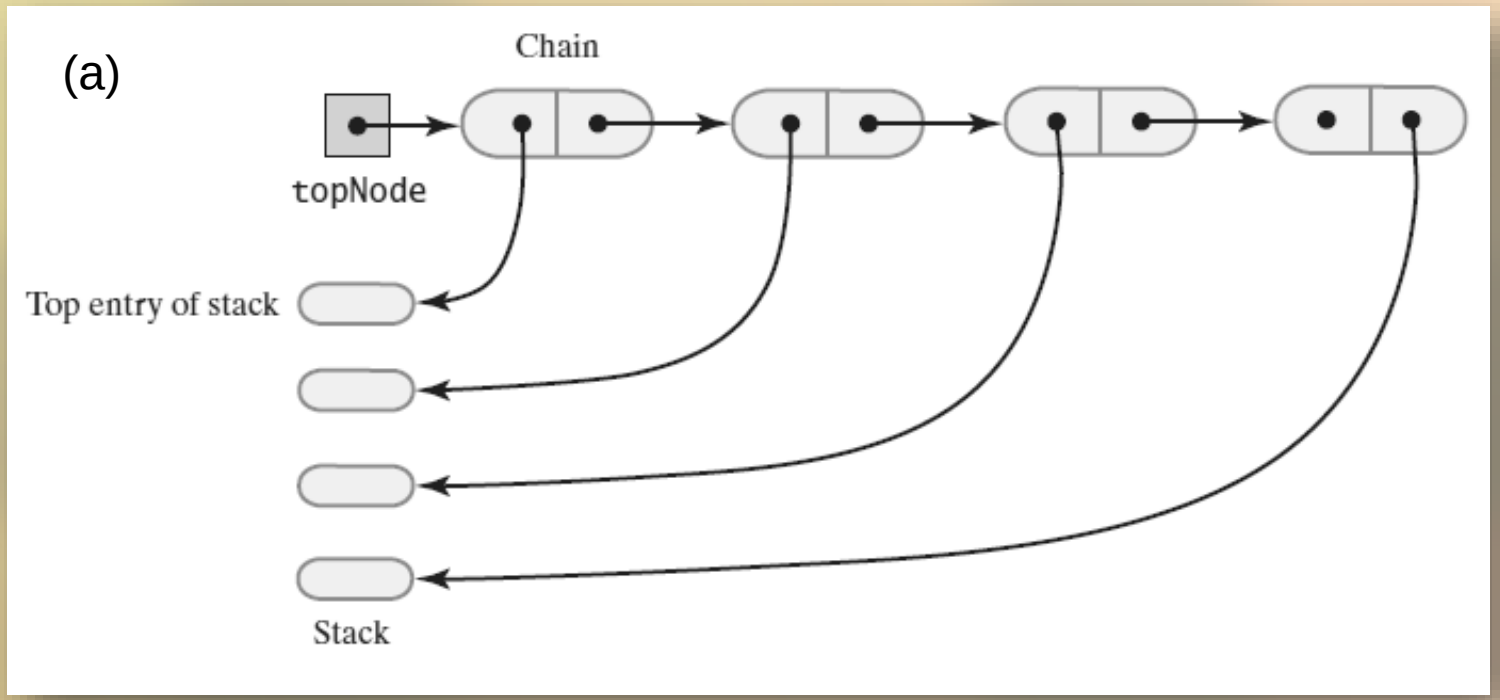

FIGURE 6-3 The stack (a) before the first node in the chain is deleted

Linked Implementation

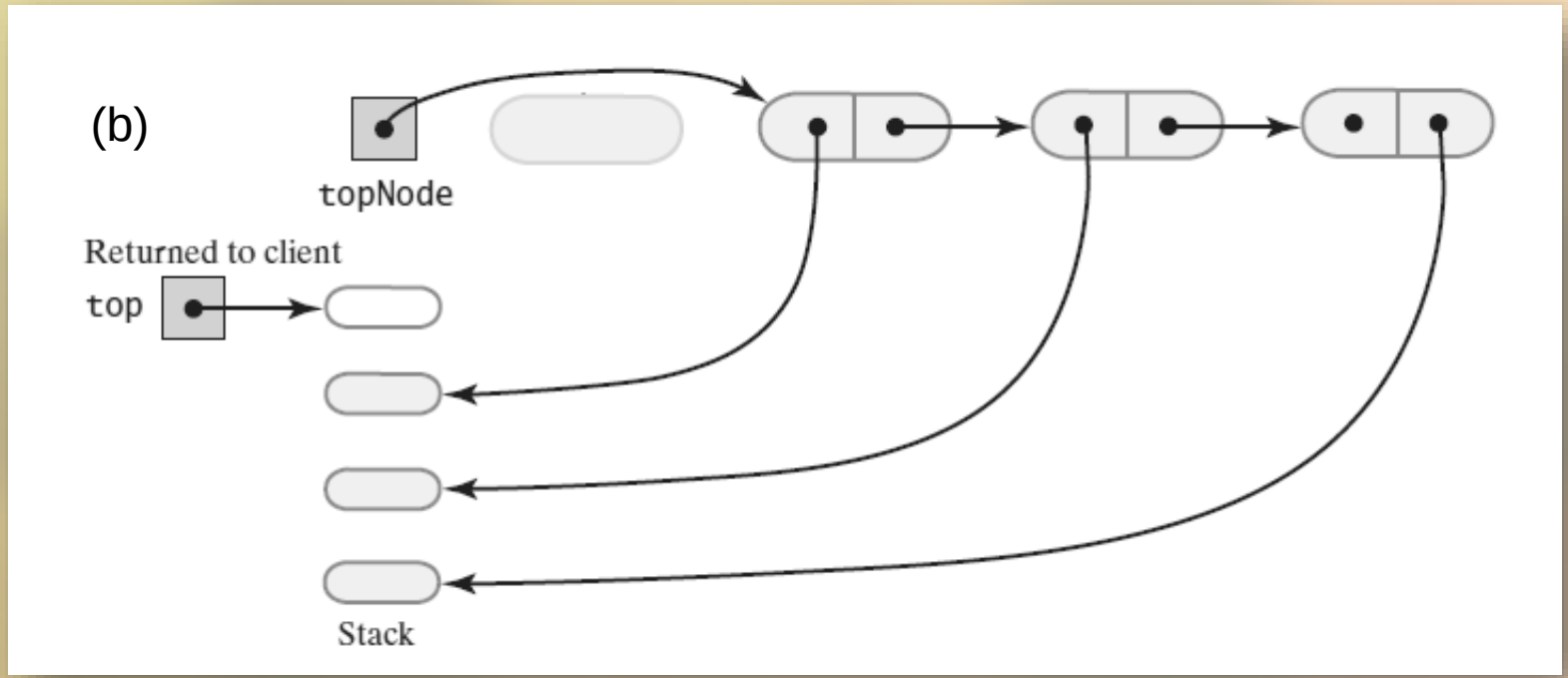

FIGURE 6-3 The stack (b) after the first node in the chain is deleted

Linked Implementation

```
public T pop()
{
   T top = peek(); // Might throw EmptyStackException
   assert topNode != null;
   topNode = topNode.getNextNode();
   return top;
} // end pop
```

Definition of `pop`

Linked Implementation

```
public boolean isEmpty()
{
   return topNode == null;
} // end isEmpty

public void clear()
{
   topNode = null;
} // end clear
```

Definition of rest of class.

Array-Based Implementation

- Each operation involves top of stack
  - `push`
  - `pop`
  - `peek`
- End of the array easiest to access
  - Let this be top of stack
  - Let first entry be bottom of stack

Array-Based Implementation

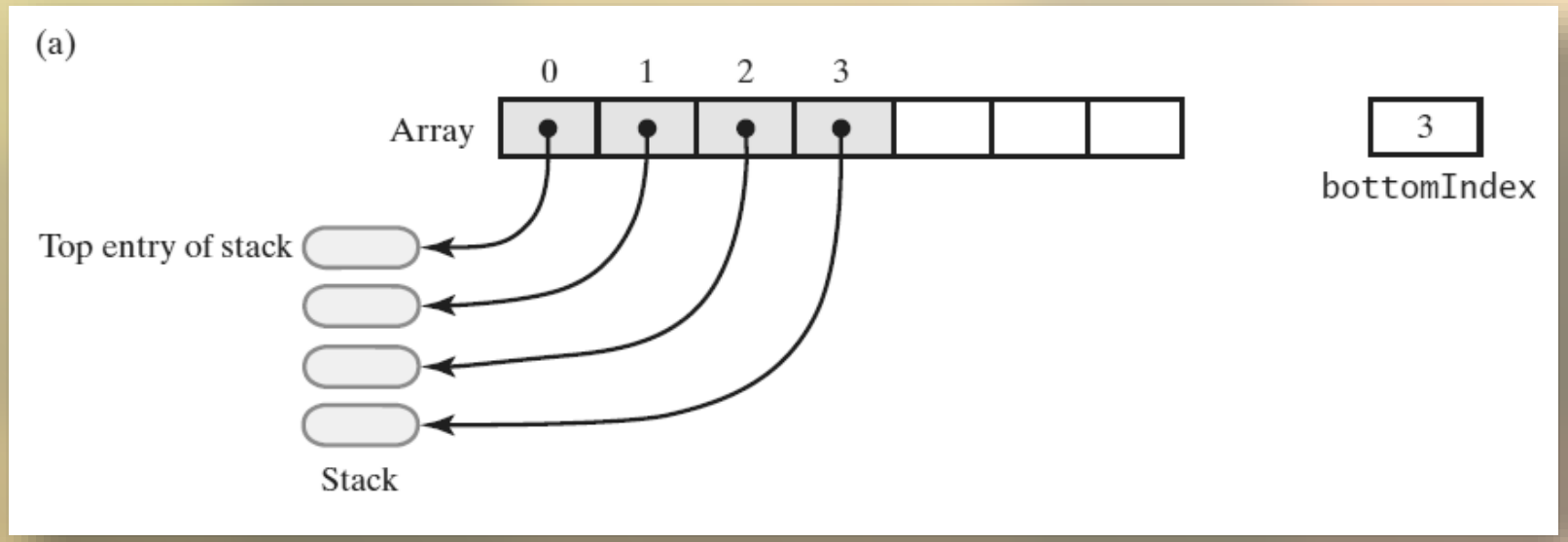

FIGURE 6-4 An array that implements a stack; its first location references (a) the top entry in the stack;

Array-Based Implementation

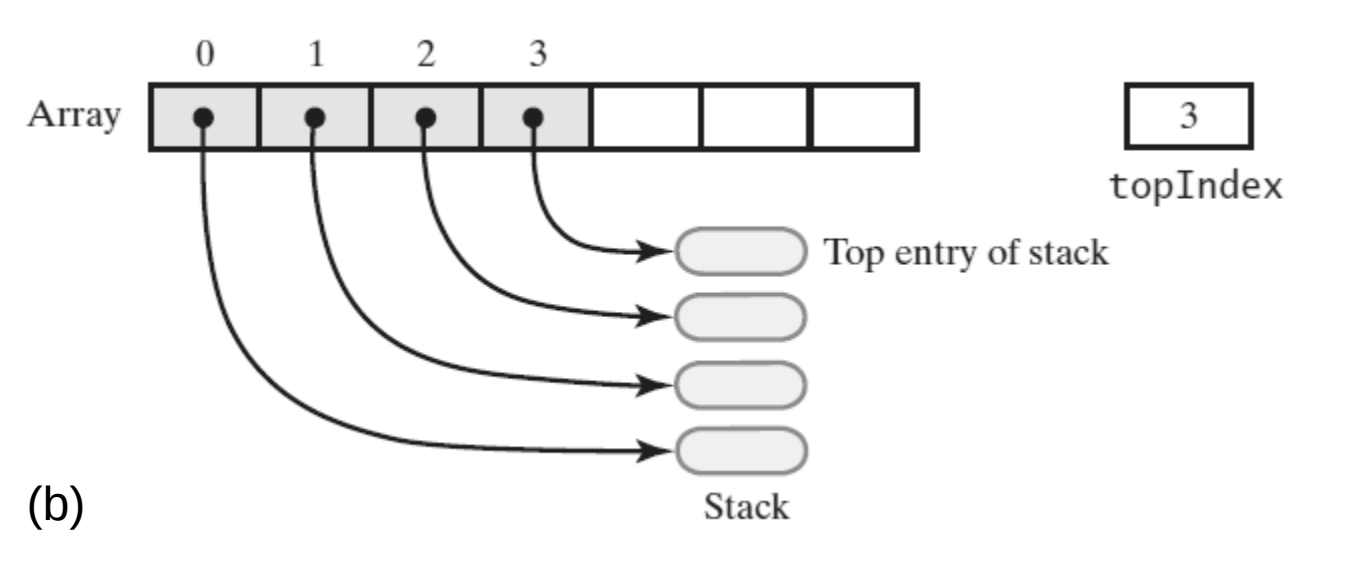

FIGURE 6-4 An array that implements a stack; its first location references (b) the bottom entry in the stack

Array-Based Implementation

```
1  /**
2      A class of stacks whose entries are stored in an array.
3      @author Frank M. Carrano
4  */
5  public final class ArrayStack<T> implements StackInterface<T>
6  {
7     private T[] stack;     // Array of stack entries
8     private int topIndex; // Index of top entry
9     private boolean initialized = false;
10    private static final int DEFAULT_CAPACITY = 50;
11    private static final int MAX_CAPACITY = 10000;
12
13    public ArrayStack()
14    {
15       this(DEFAULT_CAPACITY);
16    } // end default constructor
17
18    public ArrayStack(int initialCapacity)
```

LISTING 6-2 An outline of an array-based implementation of the ADT stack

Array-Based Implementation

```
18    public ArrayStack(int initialCapacity)
19    {
20        checkCapacity(initialCapacity);
21
22        // The cast is safe because the new array contains null entries
23        @SuppressWarnings("unchecked")
24        T[] tempStack = (T[])new Object[initialCapacity];
25        stack = tempStack;
26        topIndex = -1;
27        initialized = true;
28    } // end constructor
29
30    < Implementations of the stack operations go here. >
31    < Implementations of the private methods go here; checkCapacity and checkInitialization
32      are analogous to those in Chapter 2. >
33    . . .
34 } // end ArrayStack
```

LISTING 6-2 An outline of an array-based implementation of the ADT stack

Array-Based Implementation

```
public void push(T newEntry)
{
   checkInitialization();
   ensureCapacity();
   stack[topIndex + 1] = newEntry;
   topIndex++;
} // end push

private void ensureCapacity()
{
   if (topIndex == stack.length - 1) // If array is full, double its size
   {
      int newLength = 2 * stack.length;
      checkCapacity(newLength);
      stack = Arrays.copyOf(stack, newLength);
   } // end if
} // end ensureCapacity
```

Adding to the top.

Array-Based Implementation

```
public T peek()
{
   checkInitialization();
   if (isEmpty())
      throw new EmptyStackException();
   else
      return stack[topIndex];
} // end peek
```

Retrieving the top, operation is O(1)

Array-Based Implementation

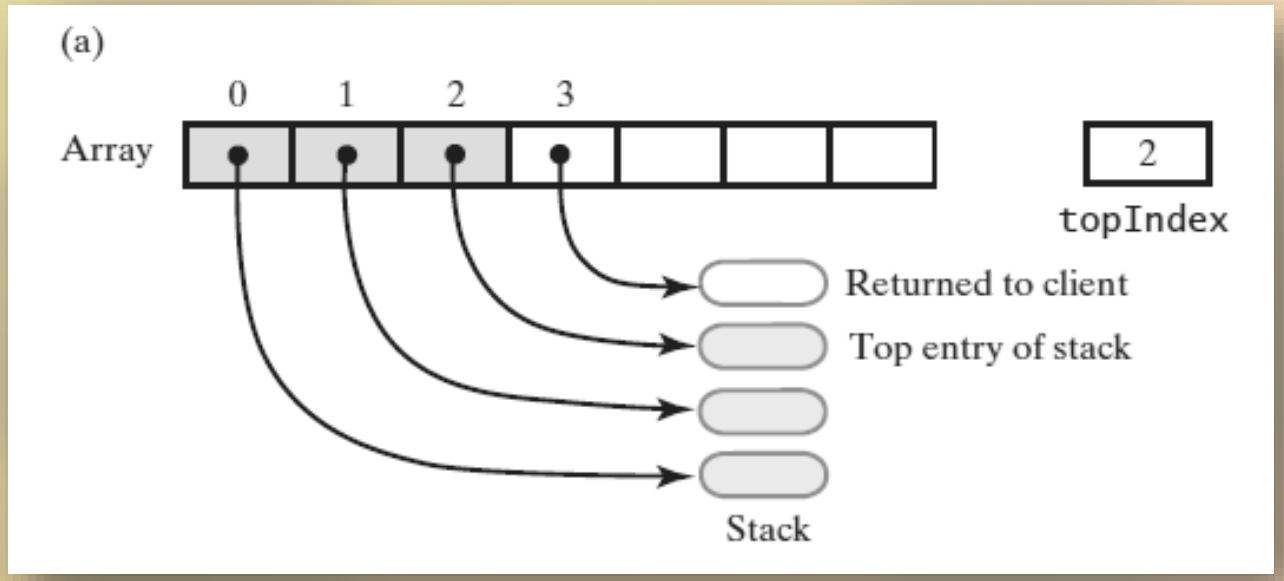

FIGURE 6-5 An array-based stack after its top entry is removed by (a) decrementing **`topIndex`**;

Array-Based Implementation

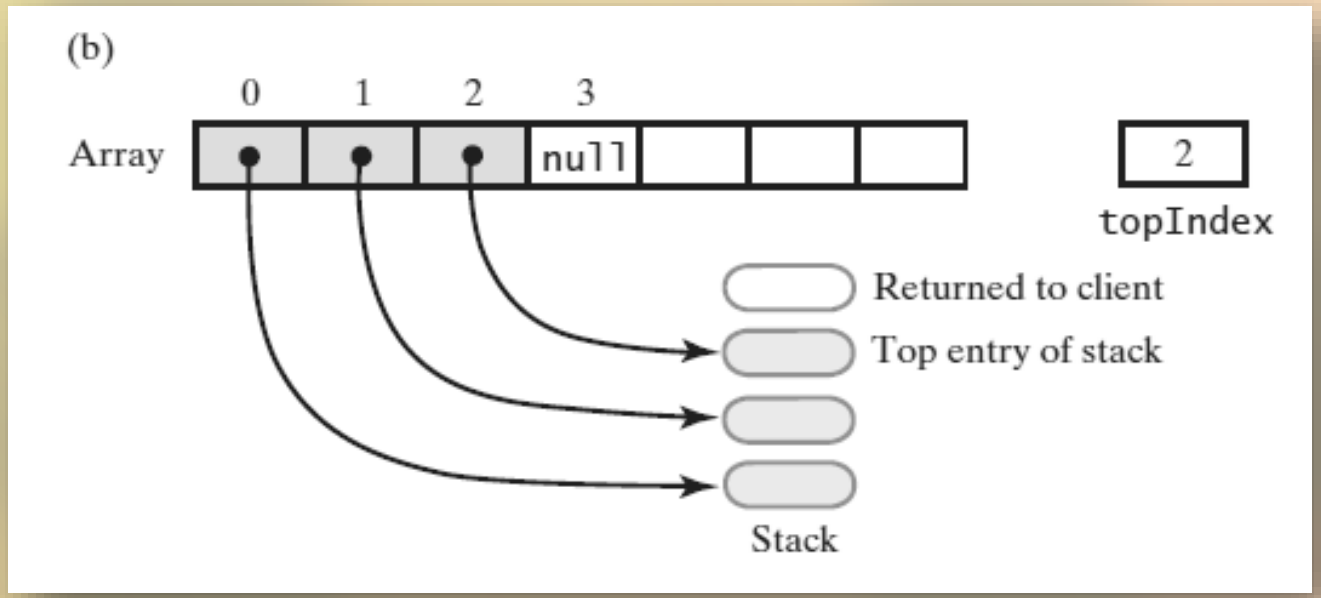

FIGURE 6-5 An array-based stack after its top entry is removed by (b) setting `stack[topIndex]` to null and then decrementing `topIndex`

Array-Based Implementation

```
public T pop()
{
  checkInitialization();
  if (isEmpty())
     throw new EmptyStackException();
  else
  {
     T top = stack[topIndex];
     stack[topIndex] = null;
     topIndex--;
     return top;
  } // end if
} // end pop
```

Removing the top

Vector Based Implementation

- Vector: an object that behaves like a high-level array
  - Index begins with 0
  - Methods to access or set entries
  - Size will grow as needed
- Use vector's methods to manipulate stack

Vector Based Implementation

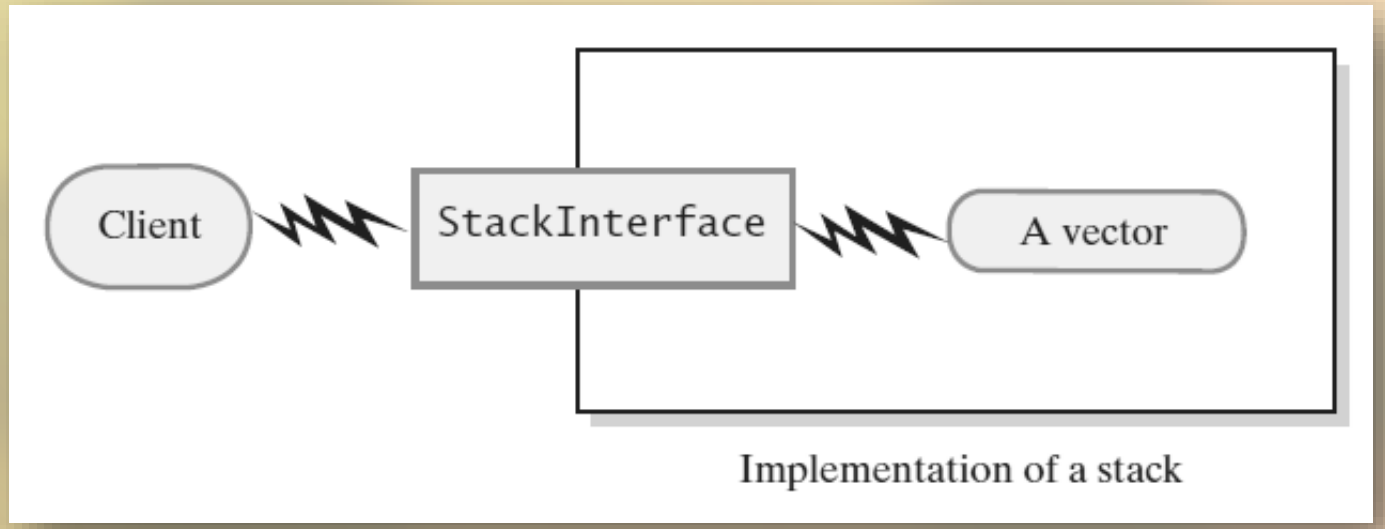

FIGURE 6-6 A client using the methods given in **`StackInterface`**; these methods interact with a vector's methods to perform stack operations

The Class `Vector`

- Constructors
- Has methods to add, remove, clear
- Also methods to determine
  - Last element
  - Is the vector empty
  - Number of entries

Vector Based Implementation

```
1  /**
2      A class of stacks whose entries are stored in a vector.
3      @author Frank M. Carrano
4  */
5  public final class VectorStack<T> implements StackInterface<T>
6  {
7     private Vector<T> stack; // Last element is the top entry in stack
8     private boolean initialized = false;
9     private static final int DEFAULT_CAPACITY = 50;
10    private static final int MAX_CAPACITY = 10000;
11
12    public VectorStack()
13    {
14       this(DEFAULT_CAPACITY);
15    } // end default constructor
16
17    public VectorStack(int initialCapacity)
```

LISTING 6-3 An outline of a vector-based implementation of the ADT stack

Vector Based Implementation

```
12    public VectorStack()
13    {
14       this(DEFAULT_CAPACITY);
15    } // end default constructor
16
17    public VectorStack(int initialCapacity)
18    {
19       checkCapacity(initialCapacity);
20       stack = new Vector<>(initialCapacity); // Size doubles as needed
21       initialized = true;
22    } // end constructor
23
24    < Implementations of checkInitialization, checkCapacity, and the stack operations go here. >
25    . . .
26 } // end VectorStack
```

LISTING 6-3 An outline of a vector-based implementation of the ADT stack

Vector Based Implementation

```
public void push(T newEntry)
{
   stack.add(newEntry);
} // end push
```

Adding to the top

Vector Based Implementation

```
public T peek()
{
   checkInitialization();
   if (isEmpty())
      throw new EmptyStackException();
   else
      return stack.lastElement();
} // end peek
```

Retrieving the top

Vector Based Implementation

```
public T pop()
{
   checkInitialization();
   if (isEmpty())
      throw new EmptyStackException();
   else
      return stack.remove(stack.size() - 1);
} // end pop
```

Removing the top

Vector Based Implementation

```
public boolean isEmpty()
{
   return stack.isEmpty();
} // end isEmpty

public void clear()
{
   stack.clear();
} // end clear
```

The rest of the class.

Stack Implementations

Chapter 6