Lecture 11: Stack Implementations

CS 0445: Data Structures

Constantinos Costa

http://db.cs.pitt.edu/courses/cs0445/current.term/

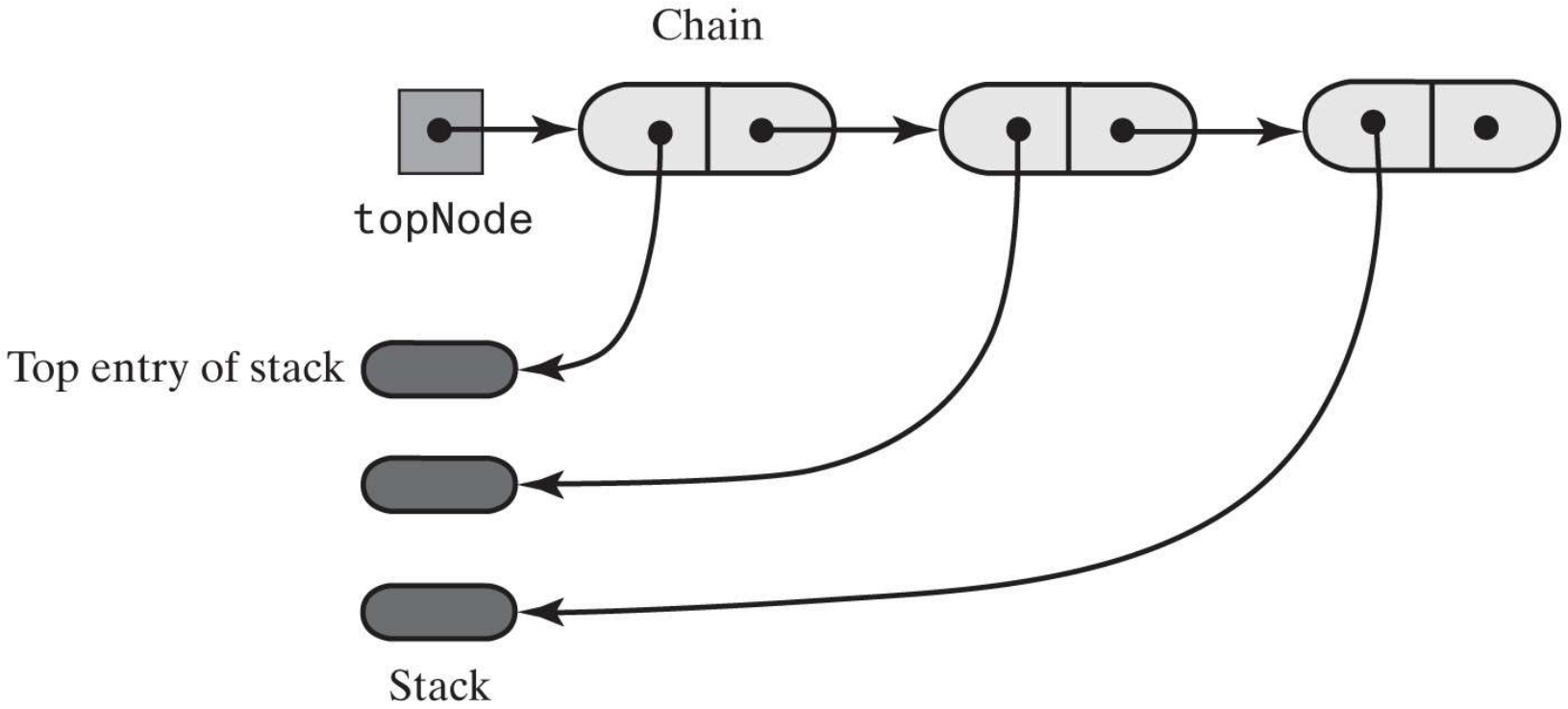
Sep 25, 2019, 8:00-9:15
University of Pittsburgh, Pittsburgh, PA

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

- A chain of linked nodes that implements a stack



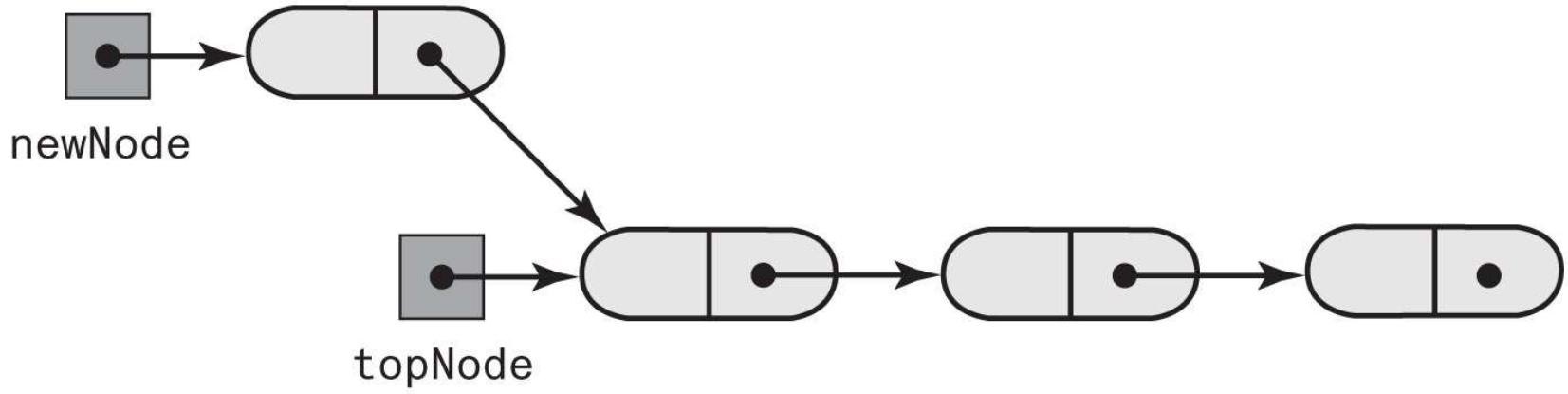
Linked Implementation of a Stack

```
/** A class of stacks whose entries are stored in a chain of nodes. */
public final class LinkedStack<T> implements StackInterface<T>
{
  private Node topNode; // References the first node in the chain

  public LinkedStack()
  {
    topNode = null;
  } // end default constructor

//  < Implementations of the stack operations go here. >
//  . . .

     private class Node
     {
    private T    data; // Entry in stack
    private Node next; // Link to next node

                                  //  < Implementations of the node operations go here. >

     } // end Node
} // end LinkedStack
```

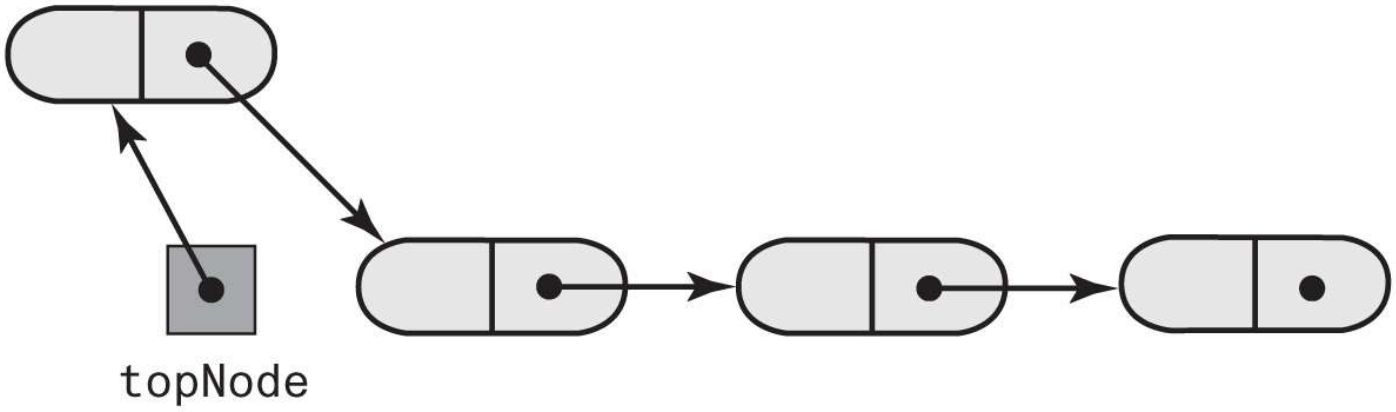
Linked Implementation of a Stack

Adding a new node to the top of a linked stack

(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 of a Stack

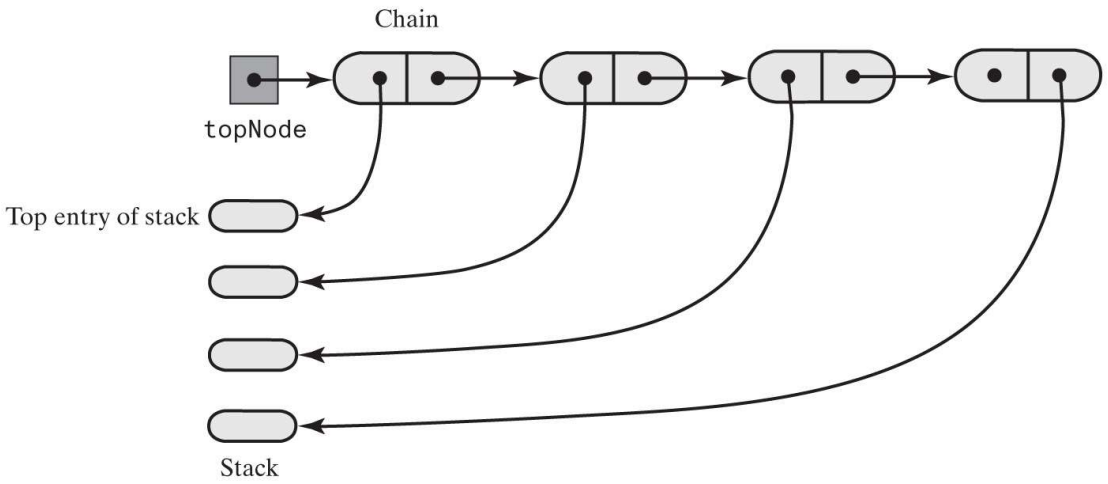
- Definition of `push`

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

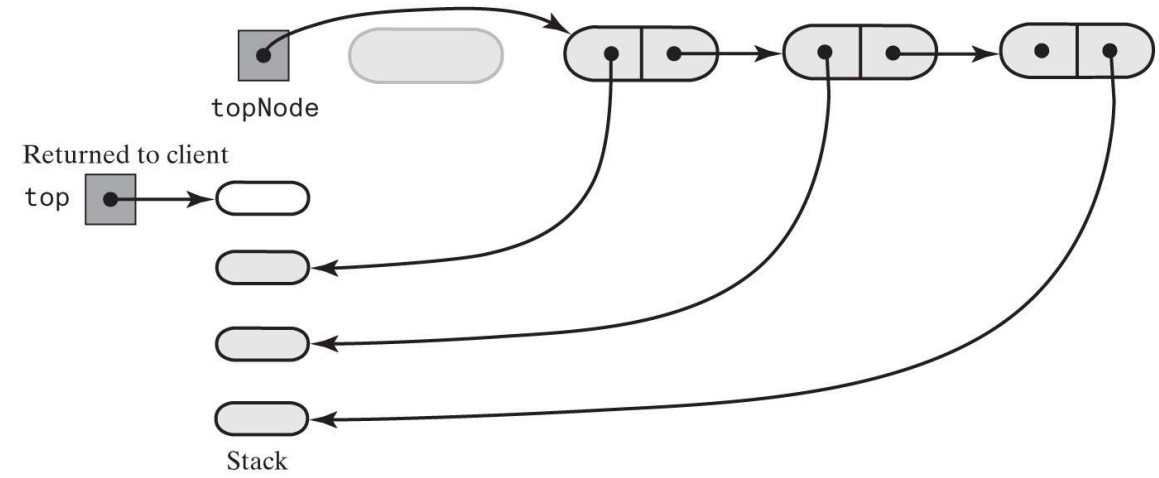
Linked Implementation of a Stack

- The stack before and after pop deletes the first node in the chain

(a) Before pop



(b) After pop



Linked Implementation

- Definition of `peek` and `pop`

```
public T peek()
{
  if (isEmpty())
    throw new EmptyStackException();
  else
    return topNode.getData();
} // end peek


public T pop()
{
  T top = peek();  // Might throw EmptyStackException

  // Assertion: topNode != null
  topNode = topNode.getNextNode();

  return top;
} // end pop
```

Linked Implementation

- Definition of `isEmpty` and `clear`.

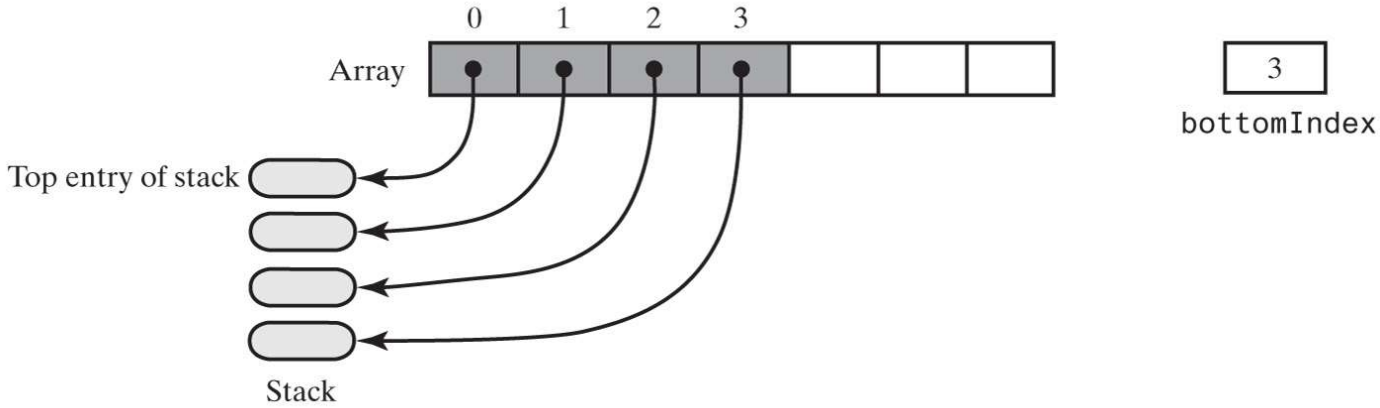
```
public boolean isEmpty()
{
  return topNode == null;
} // end isEmpty


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

Array-Based Stack Implementation

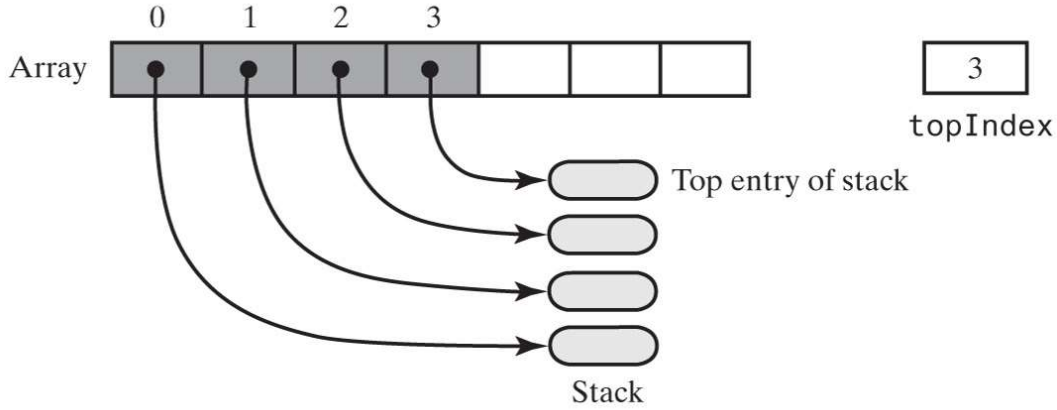
- Each operation involves top of stack
  - **`push`**
  - **`pop`**
  - **`peek`**
- End of the array easiest to access
  - Let this be top of stack
  - Let first entry be bottom of stack

Array-Based Stack Implementation

(a) Inefficient: The array's first element references the stack's top entry



(b) Efficient: The array's first element references the stack's bottom entry



Array-Based Stack Implementation

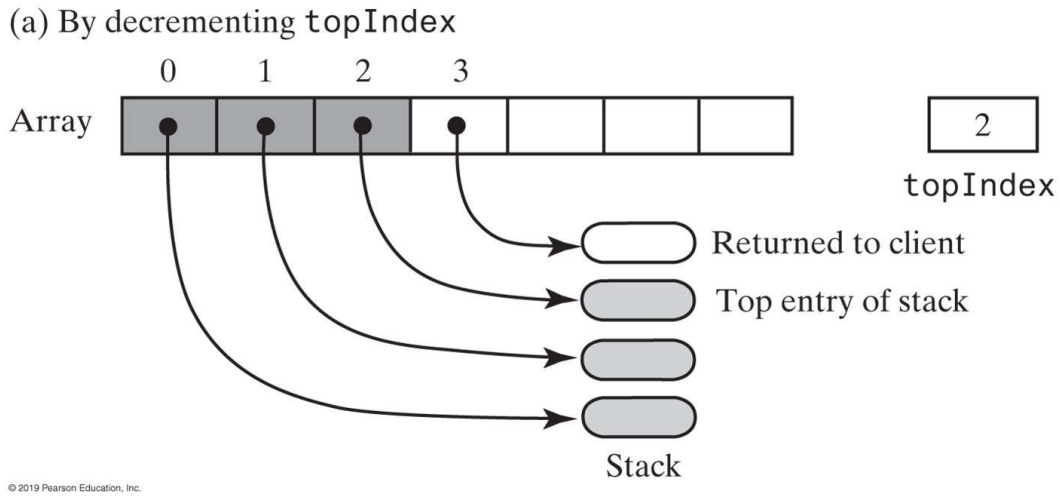
```
/** A class of stacks whose entries are stored in an array. */
public final class ArrayStack<T> implements StackInterface<T>
{
        private T[] stack;    // Array of stack entries
        private int topIndex; // Index of top entry
  private boolean integrityOK = false;
        private static final int DEFAULT_CAPACITY = 50;
        private static final int MAX_CAPACITY = 10000;

  public ArrayStack()
  {
    this(DEFAULT_CAPACITY);
  } // end default constructor

  public ArrayStack(int initialCapacity)
  {
    integrityOK = false;
    checkCapacity(initialCapacity);

    // The cast is safe because the new array contains null entries
    @SuppressWarnings("unchecked")
    T[] tempStack = (T[])new Object[initialCapacity];
    stack = tempStack;
       topIndex = -1;
    integrityOK = true;
  } // end constructor

//  < Implementations of the stack operations go here. >
//  . . .
} // end ArrayStack
```

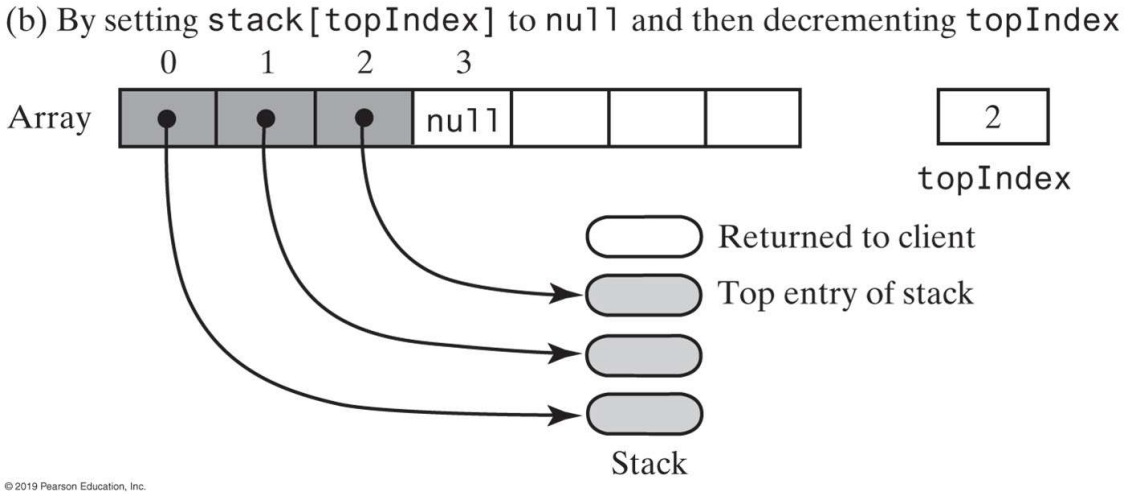
Array-Based Stack Implementation

```
public void push(T newEntry)
{
  checkInyegrity();
  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
```

Array-Based Stack Implementation

- An array-based stack after its top entry is removed in two different ways





Array-Based Stack Implementation

- Retrieving the top, operation is O(1)

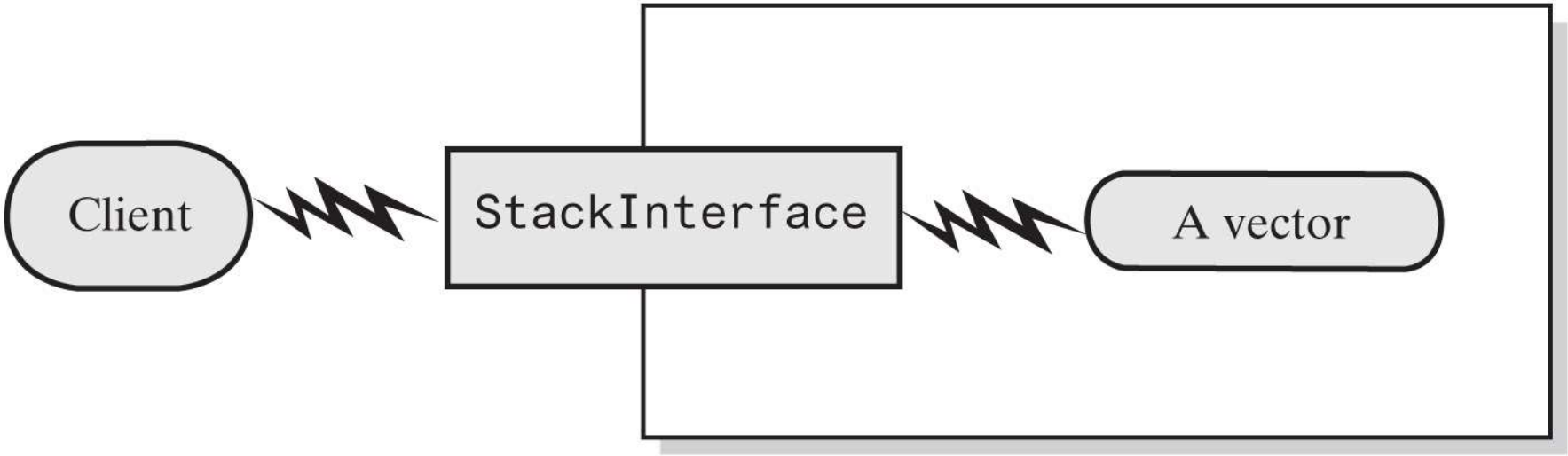
```
public T peek()
{
  checkIntegrity();
  if (isEmpty())
    throw new EmptyStackException();
  else
    return stack[topIndex];
} // end peek

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

Vector-Based Stack Implementation

- The class **`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
  - Has methods to add, remove, clear
    - Also methods to determine
      - Last element
      - Is the vector empty
      - Number of entries
- Use vector's methods to manipulate stack

Vector-Based Stack Implementation

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



Implementation of a stack

Vector-Based Stack Implementation

```
import java.util.Vector;
/** A class of stacks whose entries are stored in a vector. */
public final class VectorStack<T> implements StackInterface<T>
{
  private Vector<T> stack;   // Last element is the top entry in stack
  private boolean integrityOK;
      private static final int DEFAULT_CAPACITY = 50;
      private static final int MAX_CAPACITY = 10000;

  public VectorStack()
  {
    this(DEFAULT_CAPACITY);
  } // end default constructor

  public VectorStack(int initialCapacity)
  {
    integrityOK = false;
    checkCapacity(initialCapacity);
    stack = new Vector<>(initialCapacity); // Size doubles as needed
    integrityOK = true;
  } // end constructor

//  < Implementations of checkIntegrity, checkCapacity, and the stack
//    operations go here. >
//  . . .
} // end VectorStack
```

Vector-Based Stack Implementation

- Adding to the top

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

Vector-Based Stack Implementation

- Retrieving the top

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

Vector-Based Stack Implementation

- Removing the top

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

Vector-Based Stack Implementation

- The rest of the class.

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

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