Northeastern
University

Lecture 19: Stack Implementations - 2

Prof. Chen-Hsiang (Jones) Yu, Ph.D.

College of Engineering

Materials are edited by Prof. Jones Yu from

Data Structures and Abstractions with Java, 5th edition. By Frank M. Carrano and Timothy M. Henry.
ISBN-13 978-0-13-483169-5 © 2019 Pearson Education, Inc.

An Array-Based Implementation

Array-Based Implementation

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

Array-Based Implementation

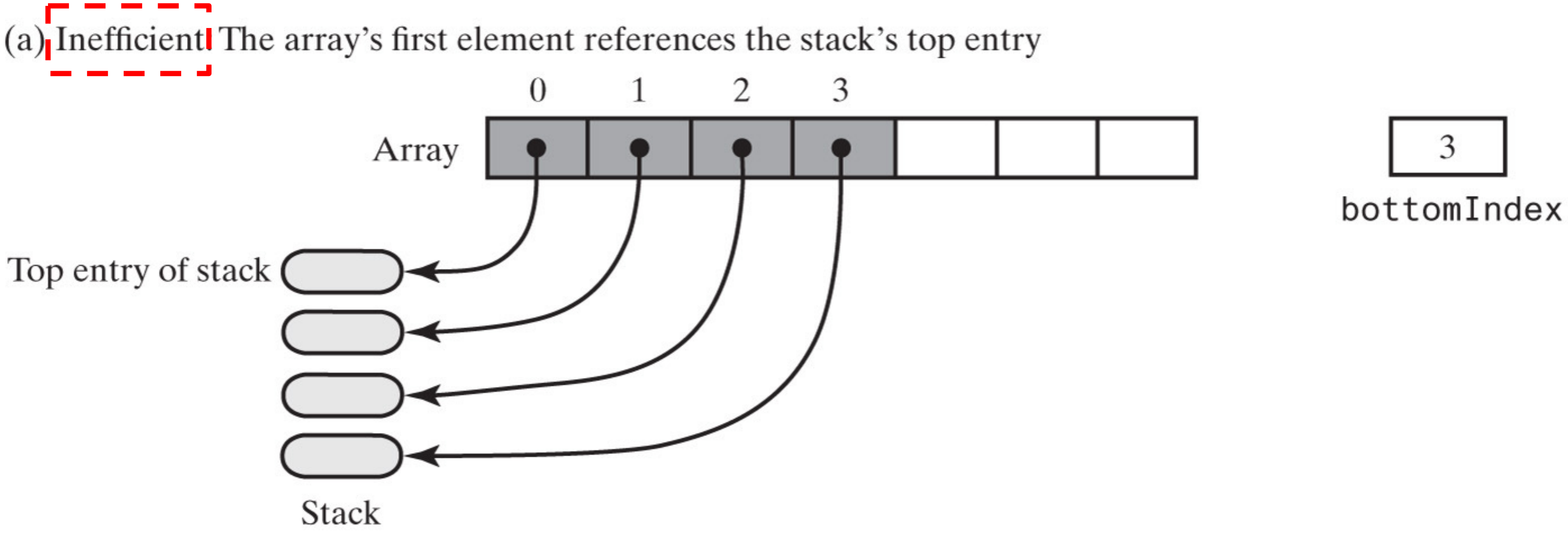

(a) Inefficient The array's first element references the stack's top entry
0
1
2
3
Array
3
bottomIndex
Top entry of stack
Stack

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

Array-Based Implementation

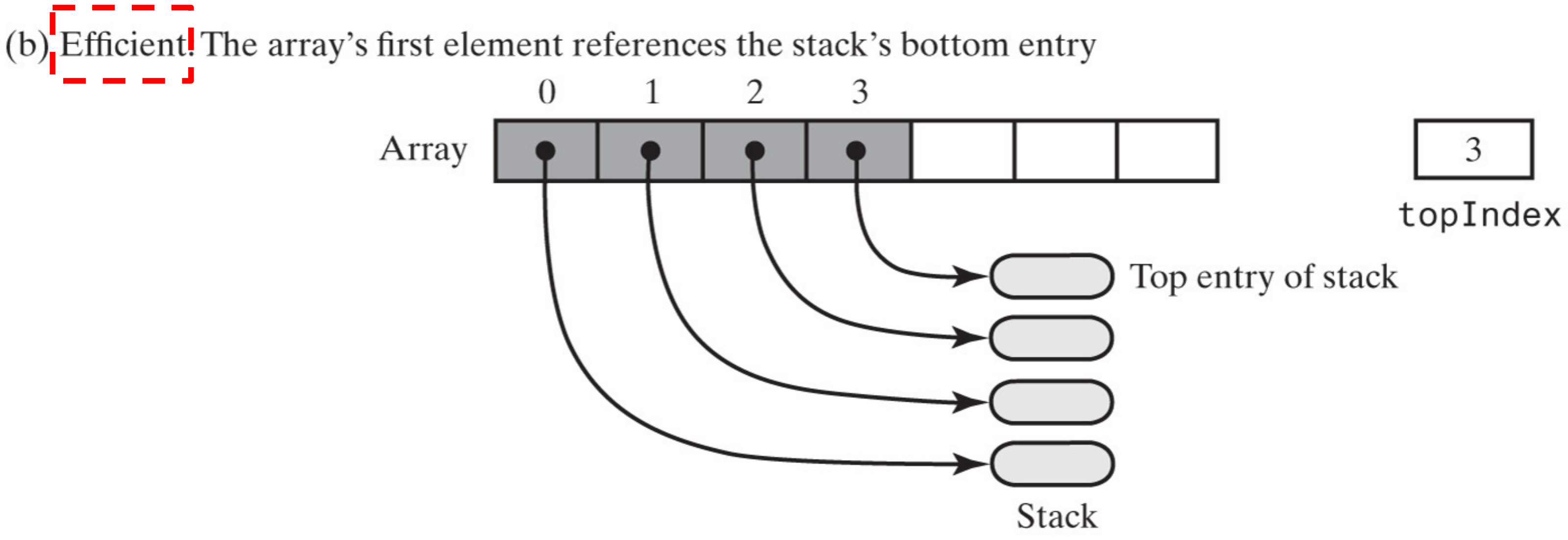

(b) Efficient! The array's first element references the stack's bottom entry
0
1
2
3
Array
3
topIndex
Top entry of stack
Stack

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

Array-Based Implementation

```java
/** 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
```

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

Array-Based Implementation

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

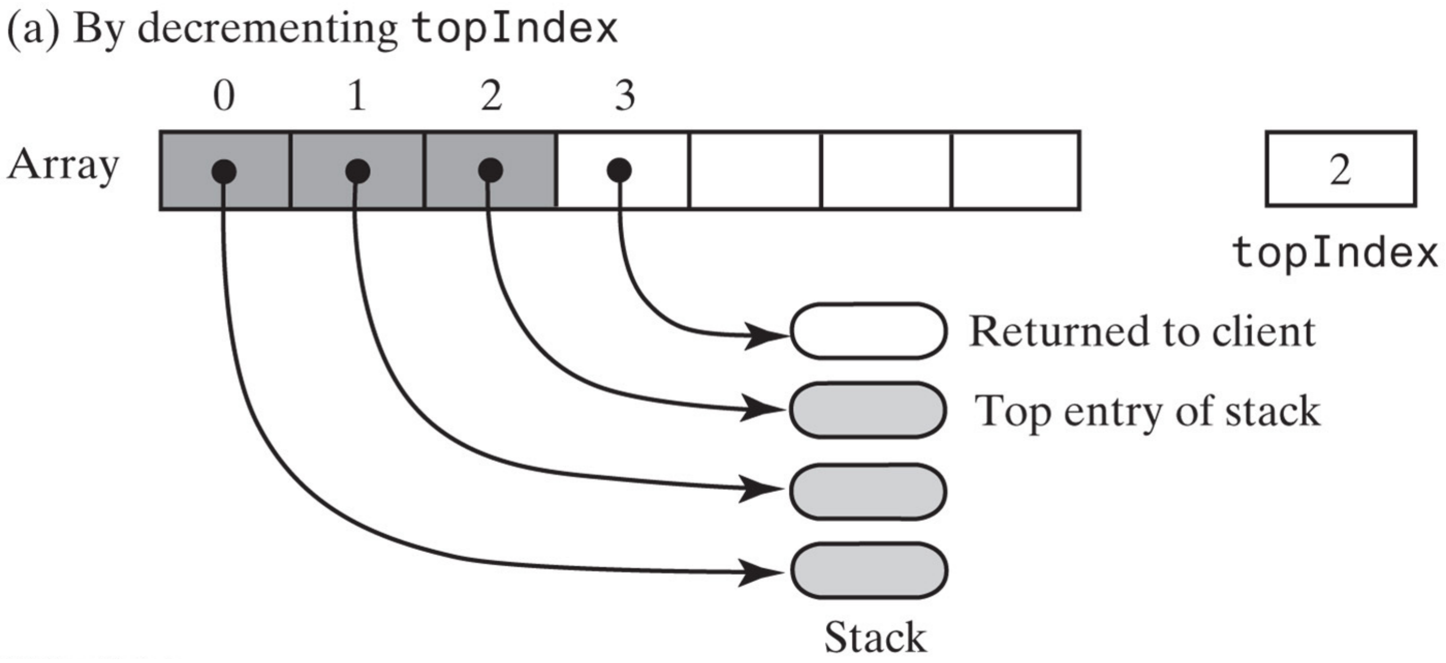


Figure 6-5: An array-based stack after its top entry is removed by (a) decrementing `topIndex`;

Array-Based Implementation

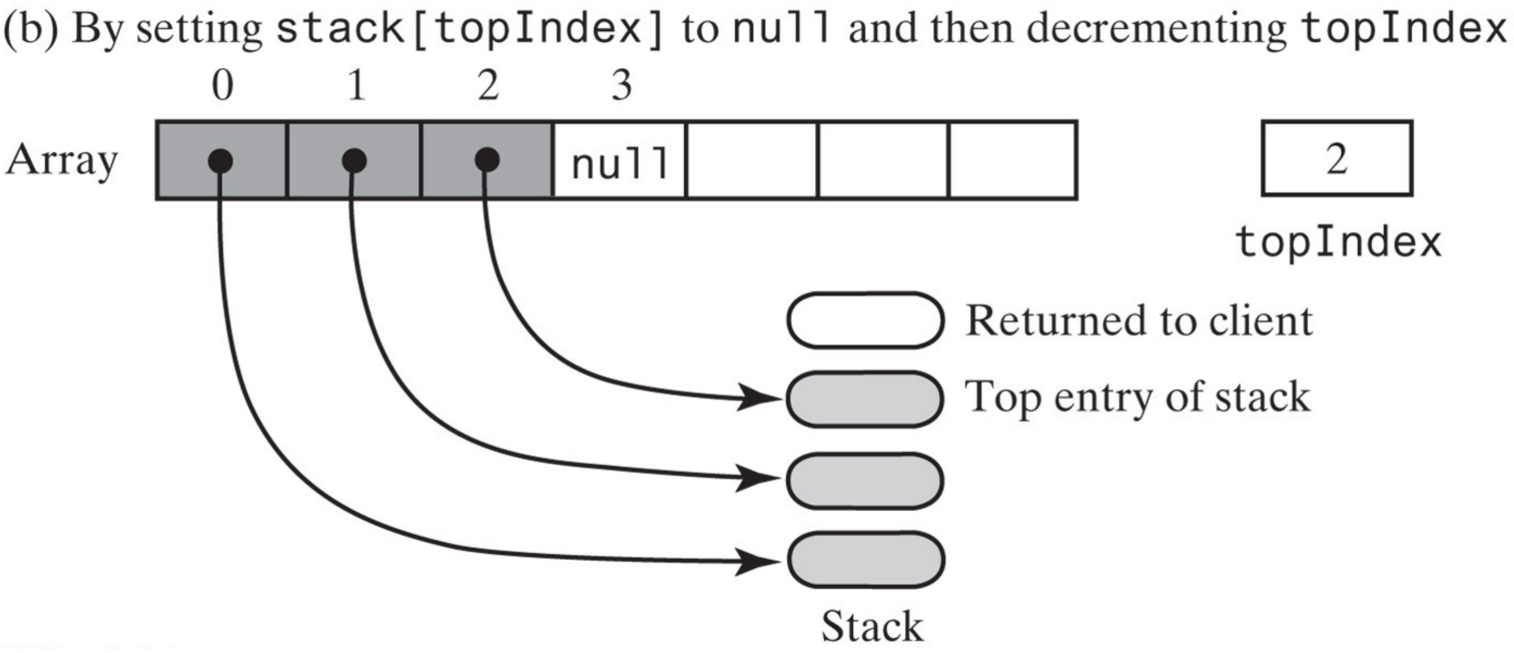

(b) By setting stack[topIndex] to null and then decrementing topIndex
0
1
2
3
Array
null
2
topIndex
Returned to client
Top entry of stack
Stack

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

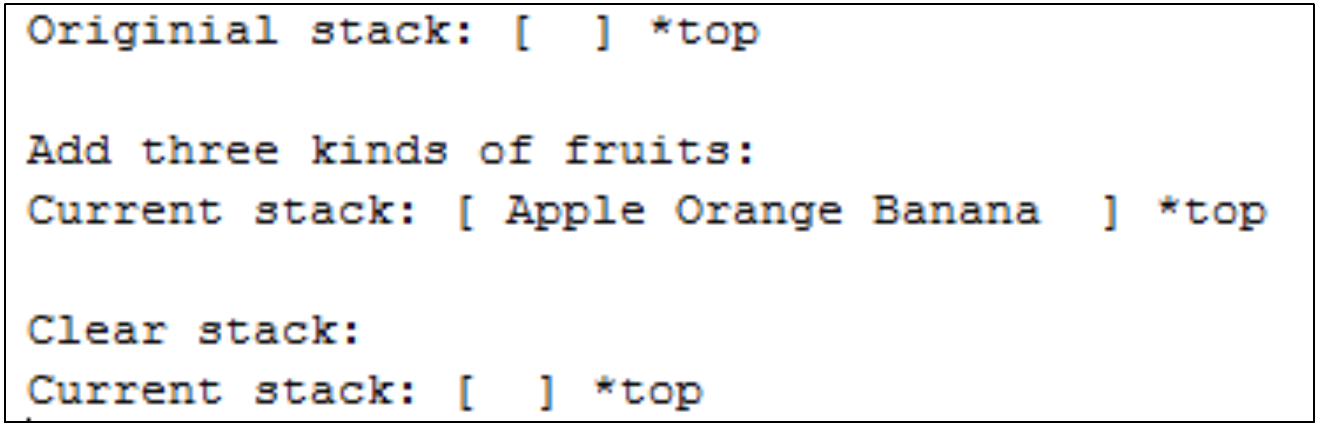
```
public T peek()
{
   checkIntegrity();
   if (isEmpty())
      throw new EmptyStackException();
   else
      return stack[topIndex];
} // end peek

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

Retrieving the top, operation is O(1)

Exercise (10 - 15 mins)

- Download “L19_E1” from the Canvas
- Implement `clear()`, `toString()` and `main()` such that you can see following results:

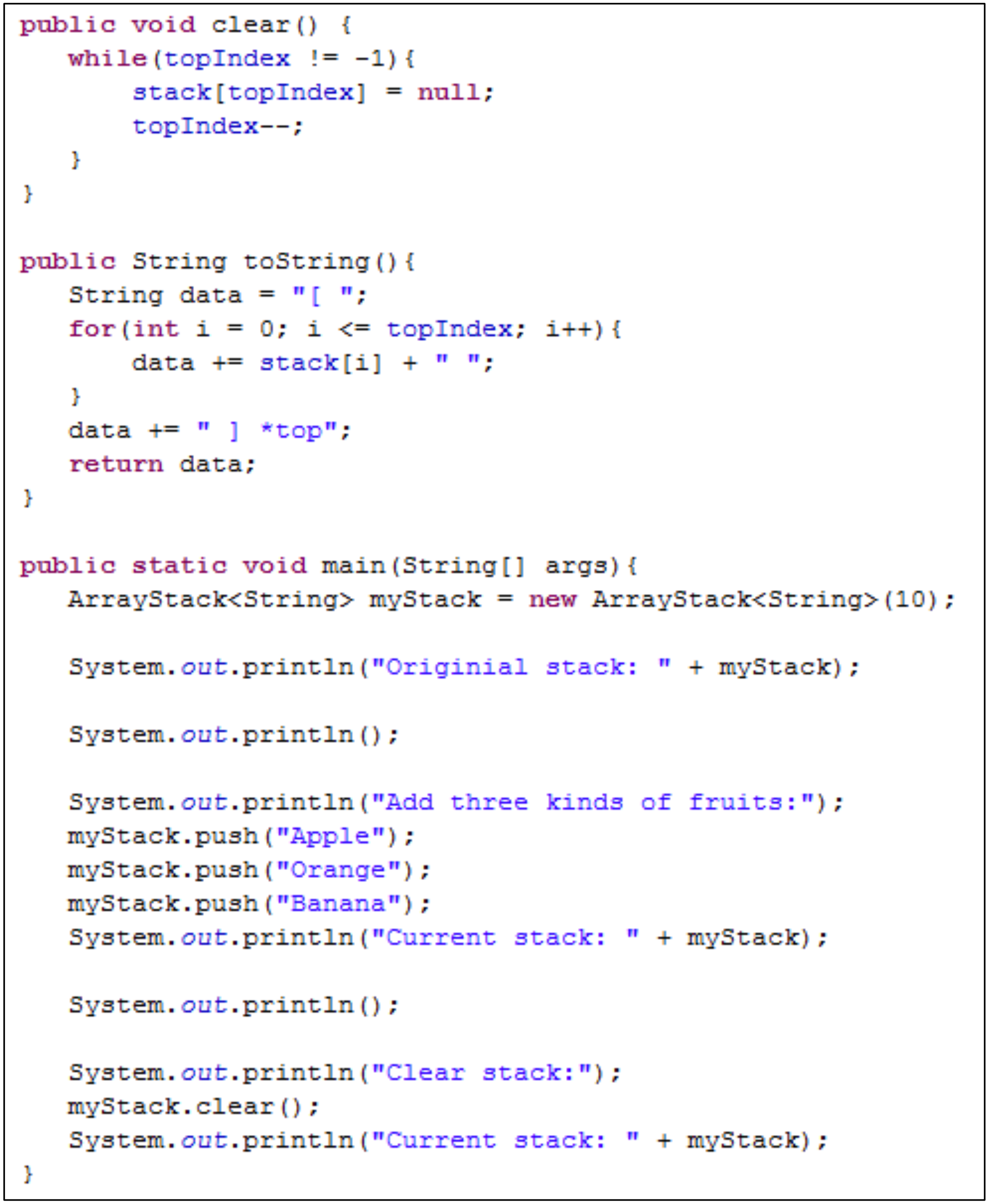


```
Originial stack: [  ] *top

Add three kinds of fruits:
Current stack: [ Apple Orange Banana  ] *top

Clear stack:
Current stack: [  ] *top
```

Answer



```
public void clear() {
   while(topIndex != -1){
       stack[topIndex] = null;
       topIndex--;
   }
}

public String toString(){
   String data = "[ ";
   for(int i = 0; i <= topIndex; i++){
       data += stack[i] + " ";
   }
   data += " ] *top";
   return data;
}

public static void main(String[] args){
   ArrayStack<String> myStack = new ArrayStack<String>(10);

   System.out.println("Originial stack: " + myStack);

   System.out.println();

   System.out.println("Add three kinds of fruits:");
   myStack.push("Apple");
   myStack.push("Orange");
   myStack.push("Banana");
   System.out.println("Current stack: " + myStack);

   System.out.println();

   System.out.println("Clear stack:");
   myStack.clear();
   System.out.println("Current stack: " + myStack);
}
```

A Vector-Based Implementation

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

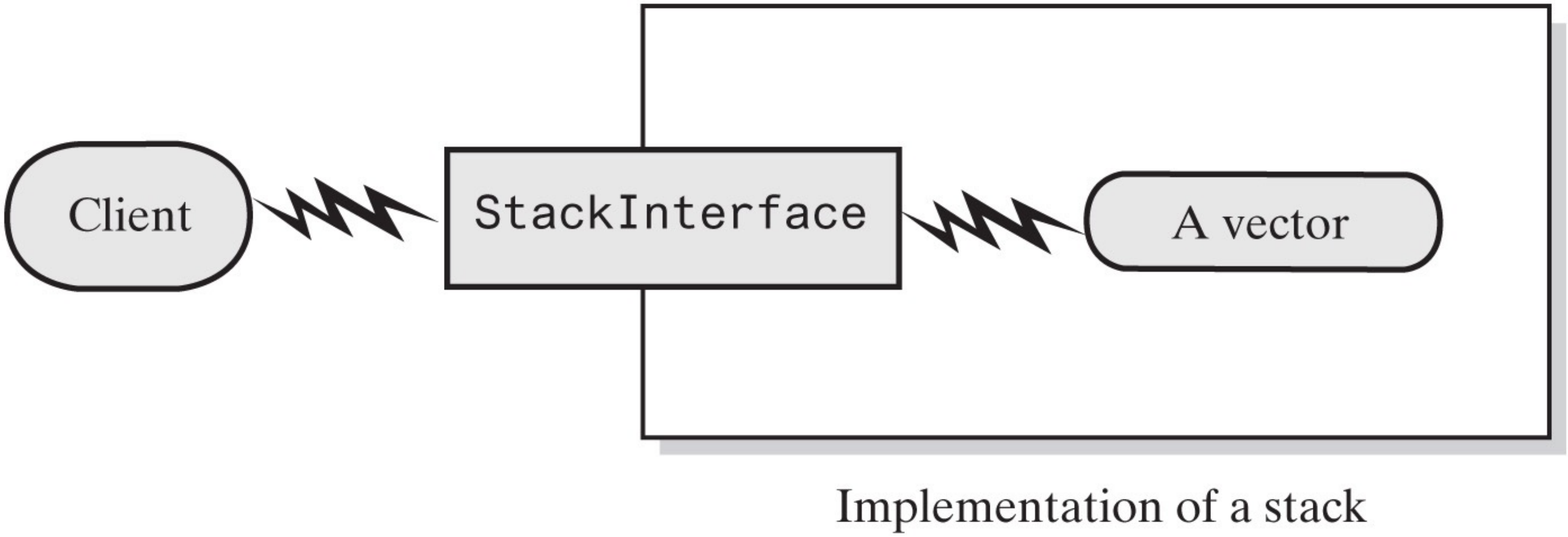


Figure 6-6: A client using the methods given in `StackInterface`; these methods interact with a vector's methods to perform stack operations

The Class `Vector`

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

Vector-Based 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
```

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

Vector-Based Implementation

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

Adding to the top

Vector-Based Implementation

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

Retrieving the top

Vector-Based Implementation

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

Removing the top

Vector-Based Implementation

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

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

The rest of the class.