Stack Array-Based Implementation

Chapter 6

Data Structures and Abstractions with Java, 4e, Global Edition
Frank Carrano

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

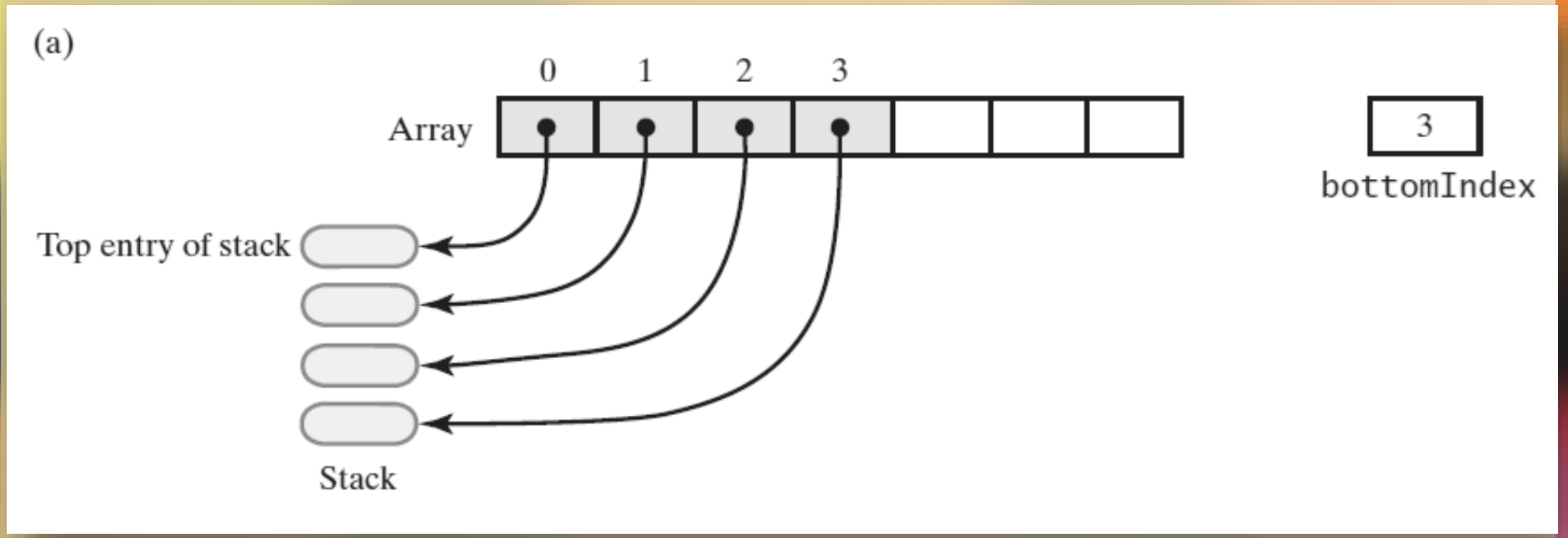

FIGURE 6-4 An array that implements a stack; its first location references (a) the top entry in the stack;

Array-Based Implementation

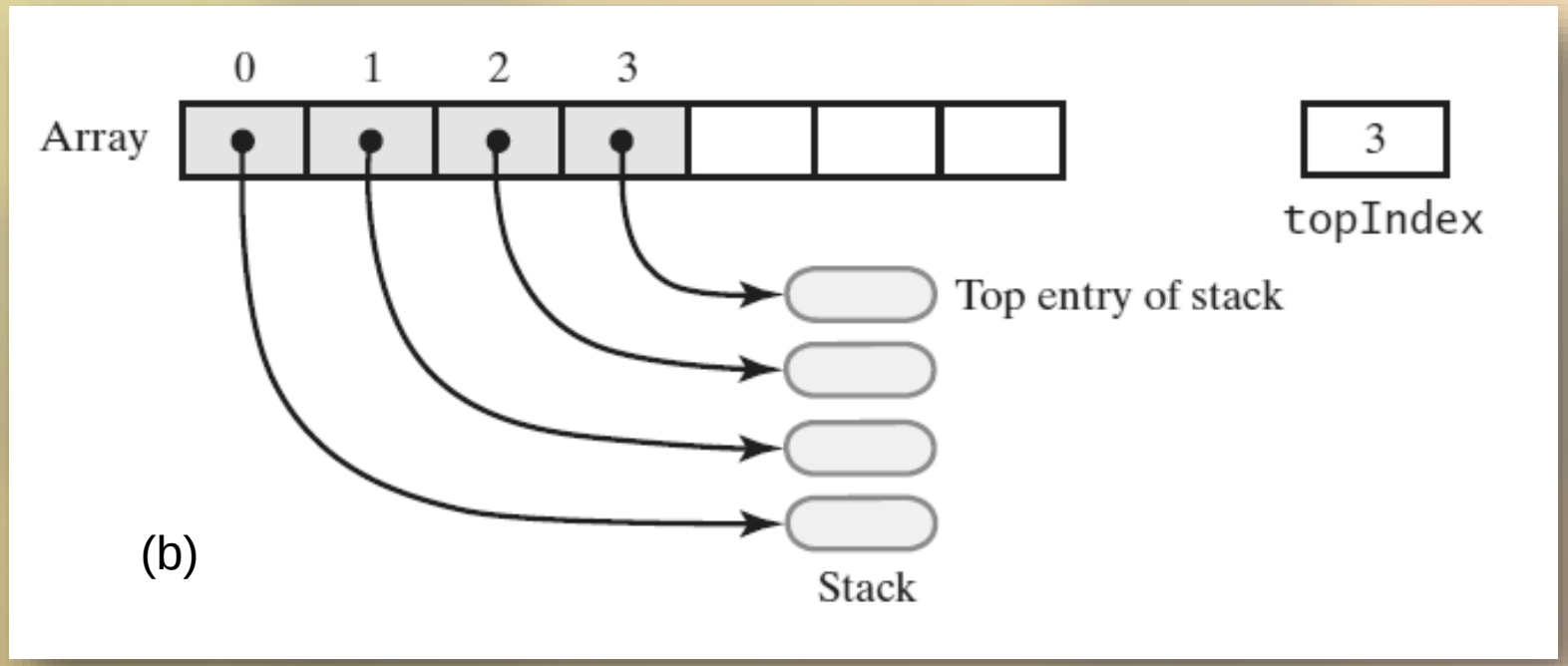

FIGURE 6-4 An array that implements a stack; its first location references (b) the bottom entry in the stack

Array-Based Implementation

```
/**
    A class of stacks whose entries are stored in an array.
    @author Frank M. Carrano
*/
public final class ArrayStack<T> implements StackInterface<T>
{
   private T[] stack;    // Array of stack entries
   private int topIndex; // Index of top entry
   private boolean initialized = 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)
```

Array-Based Implementation

```
public ArrayStack(int initialCapacity)
{
    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;
    initialized = true;
} // end constructor

< Implementations of the stack operations go here. >
< Implementations of the private methods go here; checkCapacity and checkInitialization
  are analogous to those in Chapter 2. >
. . .
} // 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
```

Array-Based Implementation

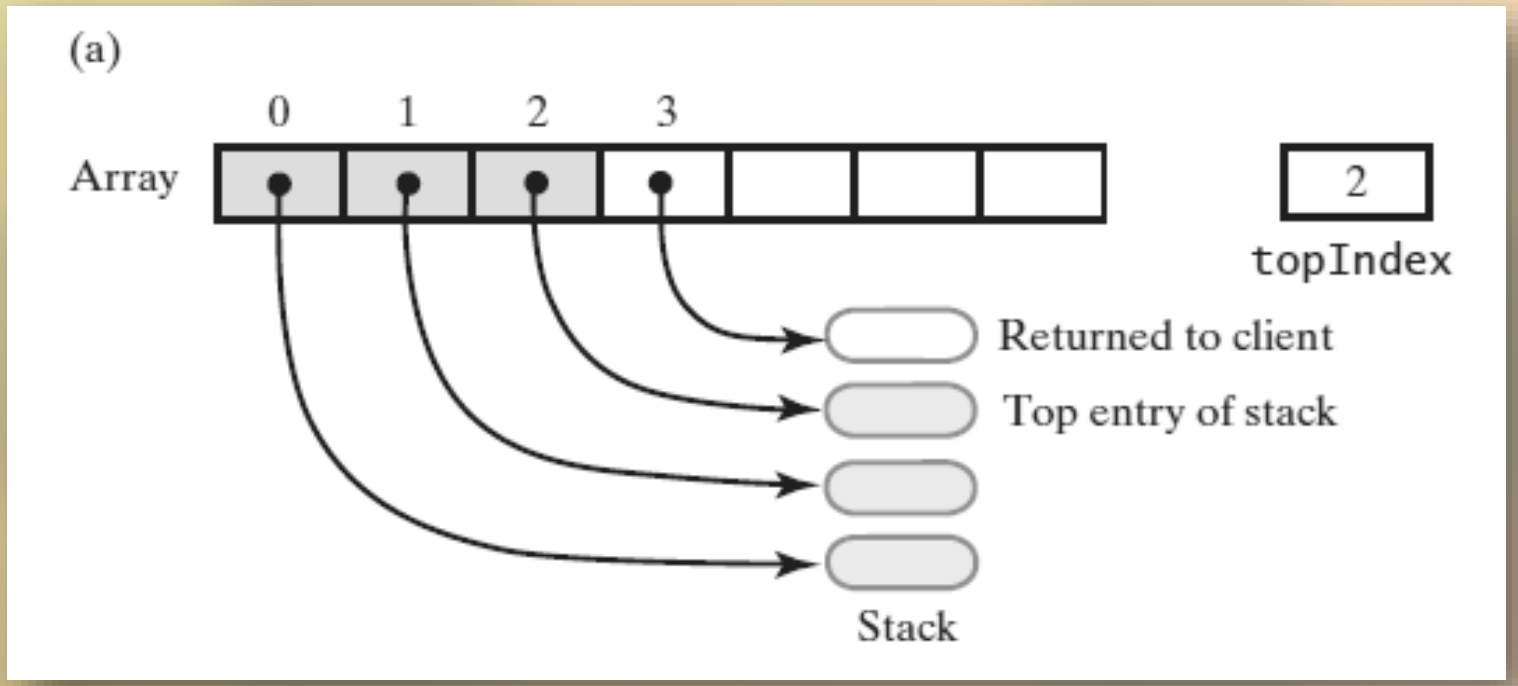

FIGURE 6-5 An array-based stack after its top entry is removed by (a) decrementing **`topIndex`**;

Array-Based Implementation

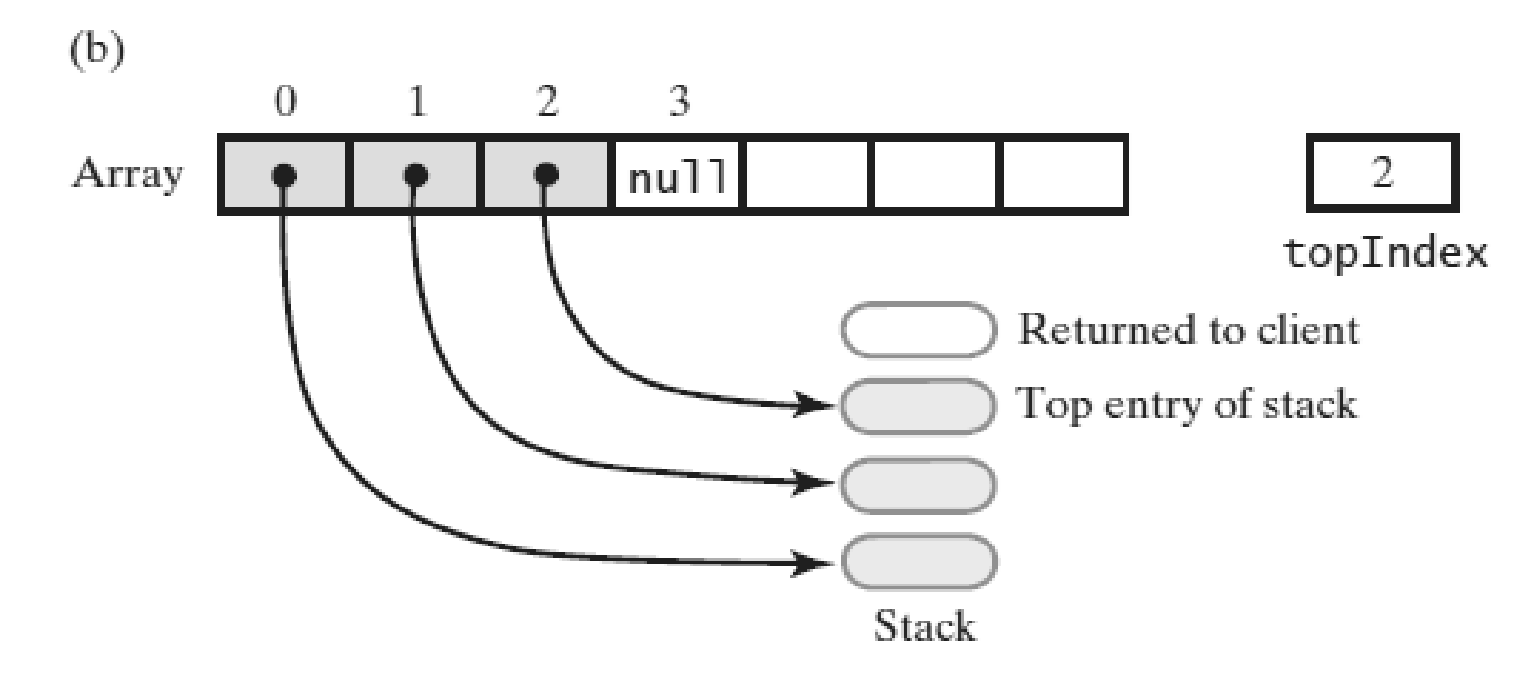

FIGURE 6-5 An array-based stack after its top entry is removed by (b) setting `stack[topIndex]` to null and then decrementing `topIndex`

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

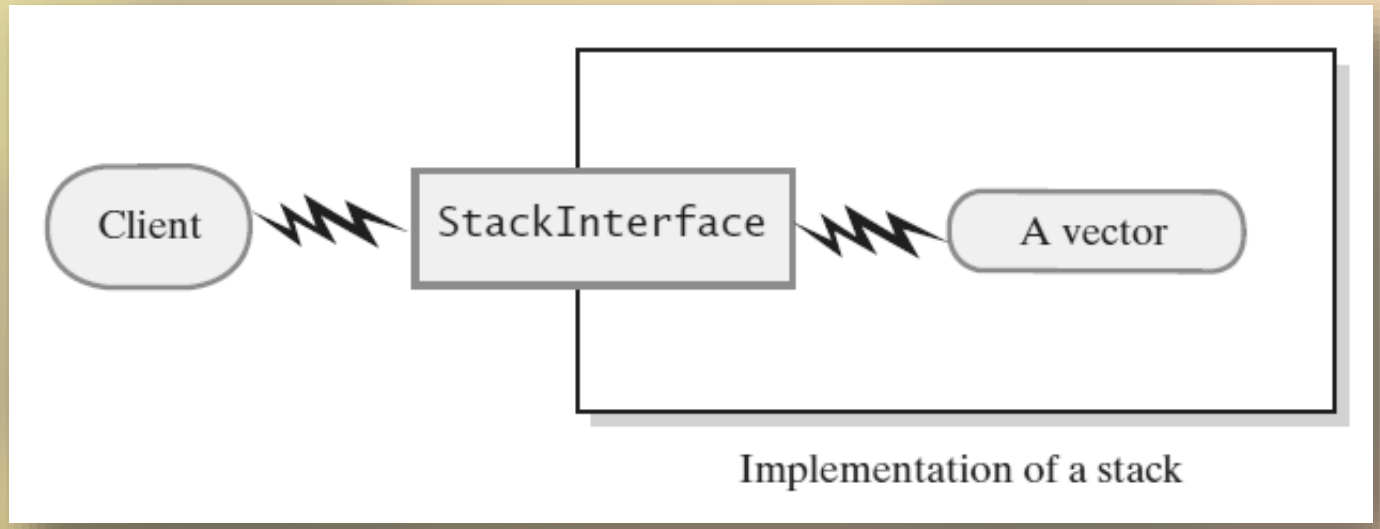

FIGURE 6-6 A client using the methods given in `StackInterface`; these methods interact with a vector's methods to perform stack operations

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

```
/**
    A class of stacks whose entries are stored in a vector.
    @author Frank M. Carrano
*/
public final class VectorStack<T> implements StackInterface<T>
{
   private Vector<T> stack; // Last element is the top entry in stack
   private boolean initialized = false;
   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)
```

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

Vector Based Implementation

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

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

    < Implementations of checkInitialization, checkCapacity, and the stack operations
    . . .
} // 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