CS2400 - Data Structures and Advanced Programming
Module 6: Stacks

Hao Ji

Computer Science Department

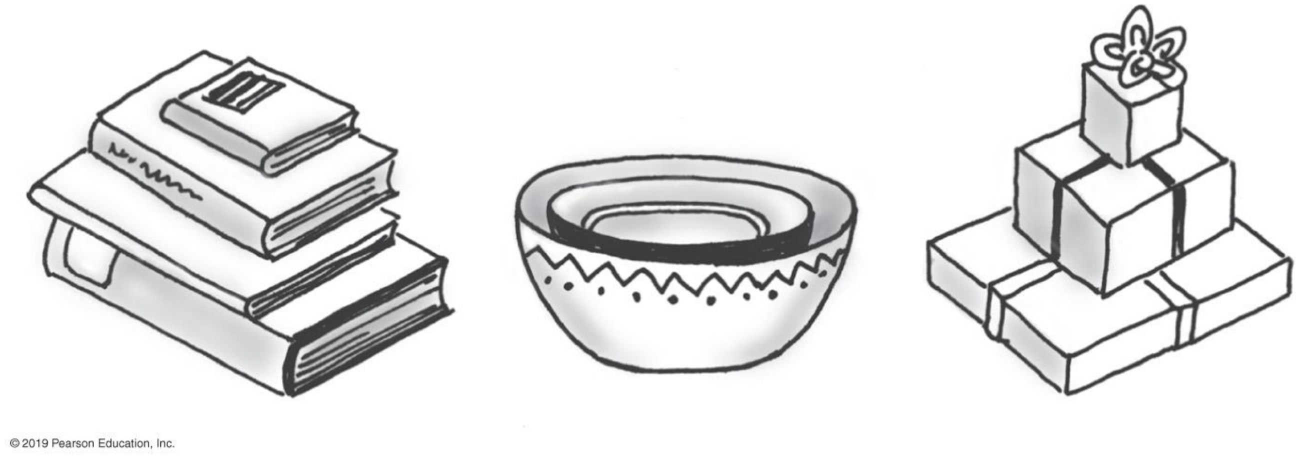
Cal Poly Pomona

Stack

- A way to organize data
    - A collection of objects in reverse chronological order and having the same data type

Stack

- A way to organize data
  - A collection of objects in reverse chronological order and having the same data type



- When you add an item to a stack, you place it on top of the stack.
- When you remove an item, you take the topmost one. This topmost item is the last one that was added to the stack.
  - Last In, First Out … LIFO

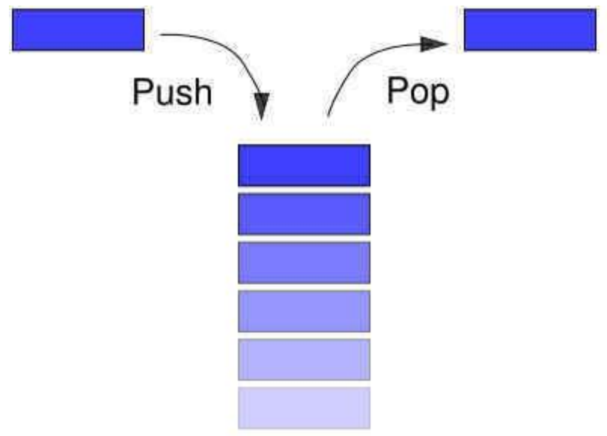
Stack

- A way to organize data
  - A collection of objects in reverse chronological order and having the same data type

The only way to look at an entry that is not at the top of the stack

- repeatedly remove items from the stack until the desired item reaches the top

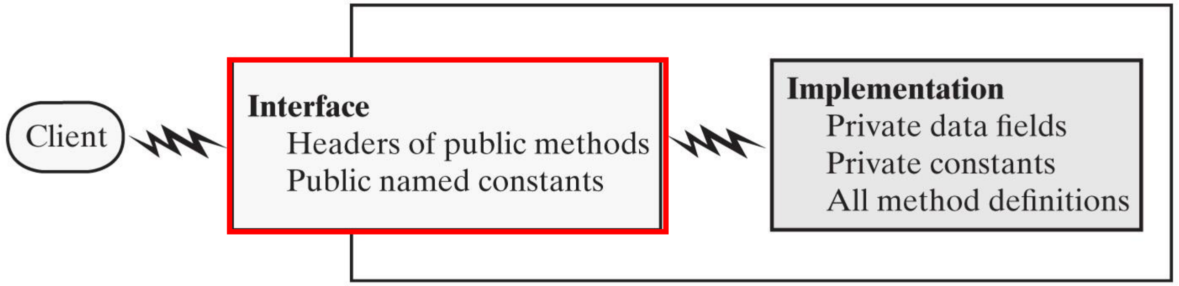
Using UML Notation to Specify a Class

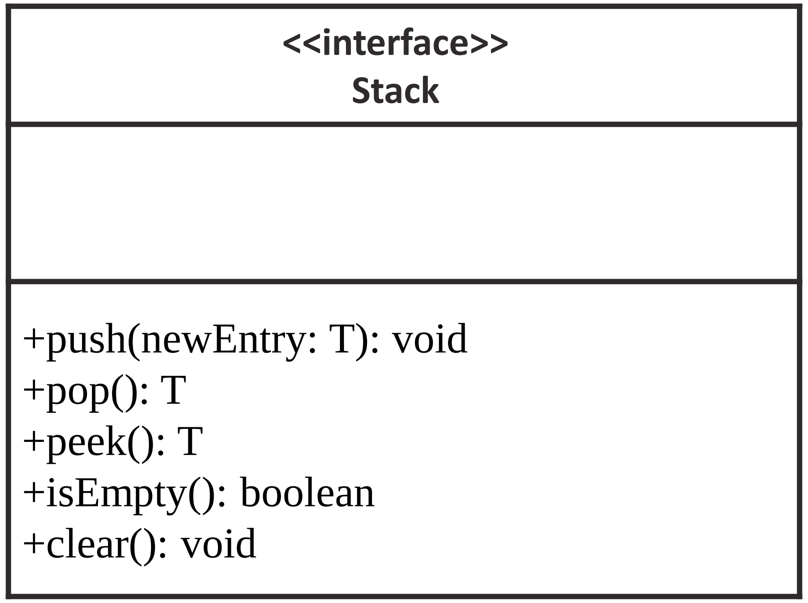


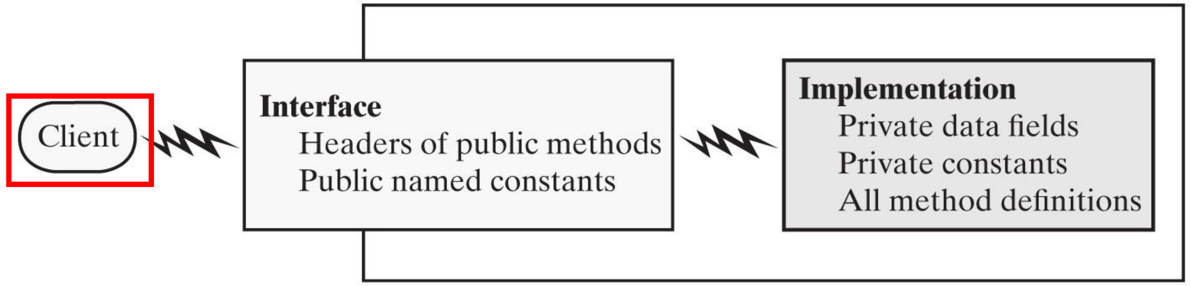
<<interface>>
Stack

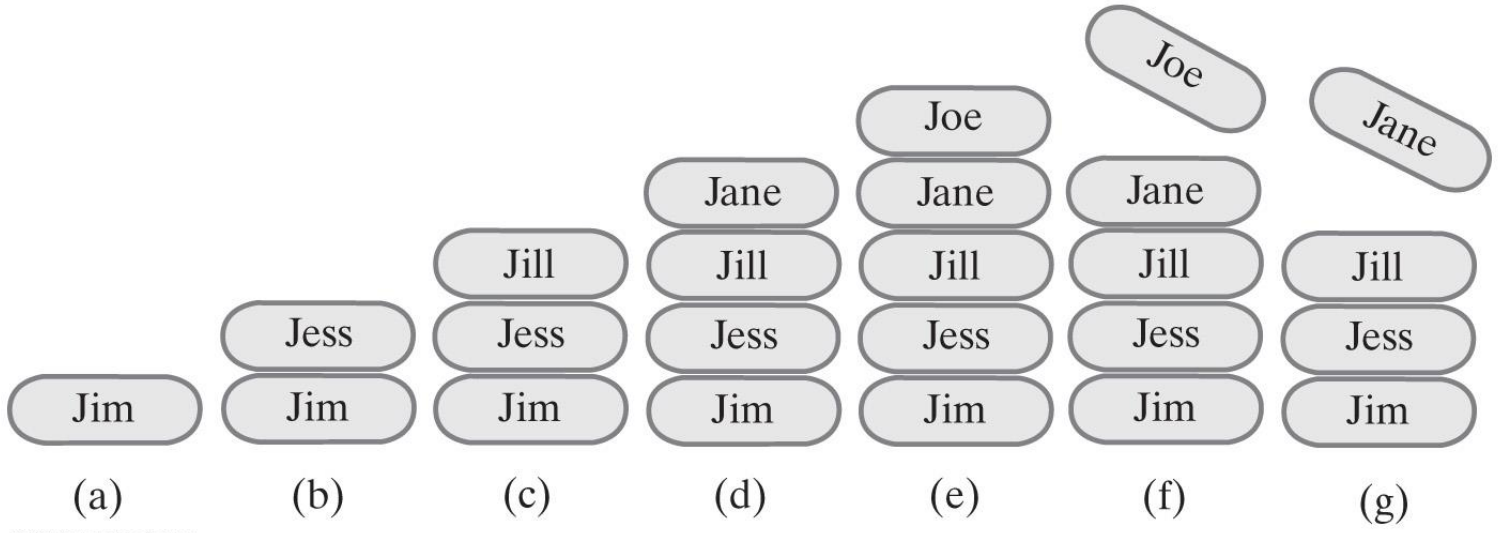
+push(newEntry: T): void	// Adds a new entry to the top of the stack
+pop(): T	// Removes and returns the stack's top entry
+peek(): T	// Retrieves the stack's top entry without changing the stack in any way
+isEmpty(): boolean	// Detects whether the stack is empty.
+clear(): void	// Removes all entries from the stack.

```java
/** An interface for the ADT stack. */
public interface StackInterface<T>
{
  /** Adds a new entry to the top of this stack.
     @param newEntry  An object to be added to the stack. */
  public void push(T newEntry);

  /** Removes and returns this stack's top entry.
     @return  The object at the top of the stack.
     @throws  EmptyStackException if the stack is empty before the operation. */
  public T pop();

  /** Retrieves this stack's top entry.
     @return  The object at the top of the stack.
     @throws  EmptyStackException if the stack is empty. */
  public T peek();

  /** Detects whether this stack is empty.
     @return  True if the stack is empty. */
  public boolean isEmpty();

  /** Removes all entries from this stack. */
  public void clear();
} // end StackInterface
```









```
    StackInterface<String> stringStack = new OurStack<>();
(a) stringStack.push(“Jim”);
(b) stringStack.push("Jess");
(c) stringStack.push("Jill");
(d) stringStack.push("Jane");
(e) stringStack.push("Joe");
(f) stringStack.pop();
(g) stringStack.pop();
```

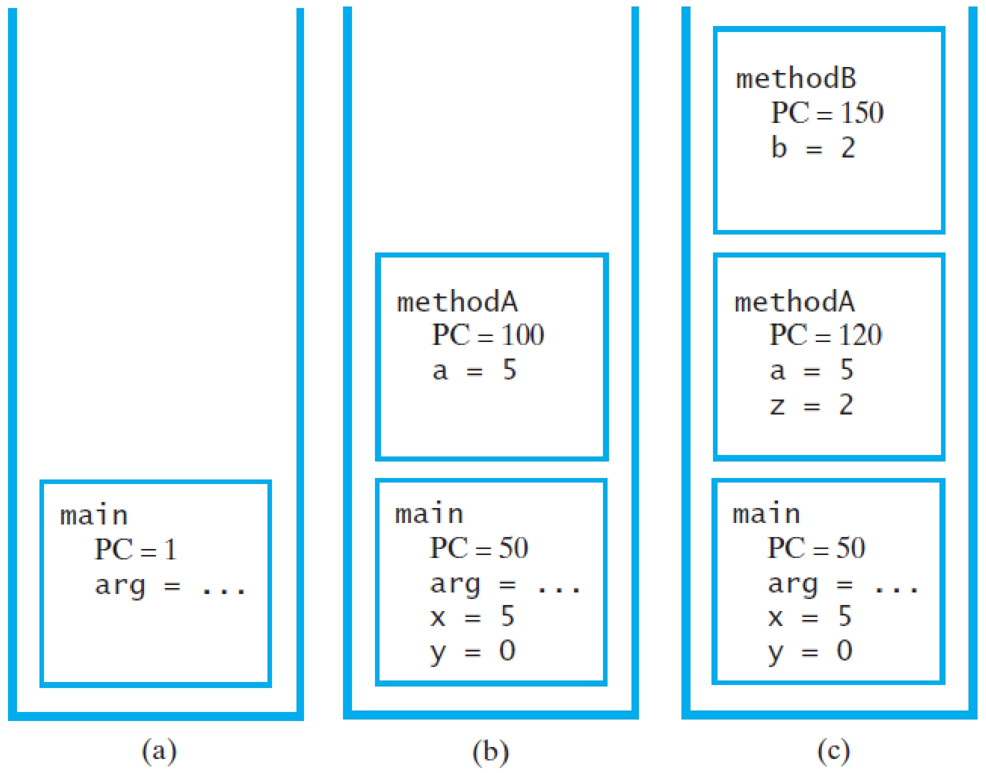
Program Stack

- When a method is called, the program’s run-time environment creates an object called an **activation record**, or **frame**, for the method.
- The activation record shows the method’s state during its execution, which contains the method’s arguments, local variables, and a reference to the current instruction—that is, a copy of the program counter.
- At the time the method is called, the activation record is pushed onto a stack called the **program stack** or, in Java, the **Java stack**.

Program Stack

```
1      public static
       void main(string[] arg)
       {
          . . .
          int x = 5;
50        int y = methodA(x);
          . . .
       } // end main

100    public static
       int methodA(int a)
       {
          . . .
          int z = 2;
120        methodB(z);
          . . .
          return z;
       } // end methodA

150    public static
       void methodB(int b)
       {
          . . .
       } // end methodB
```

Program

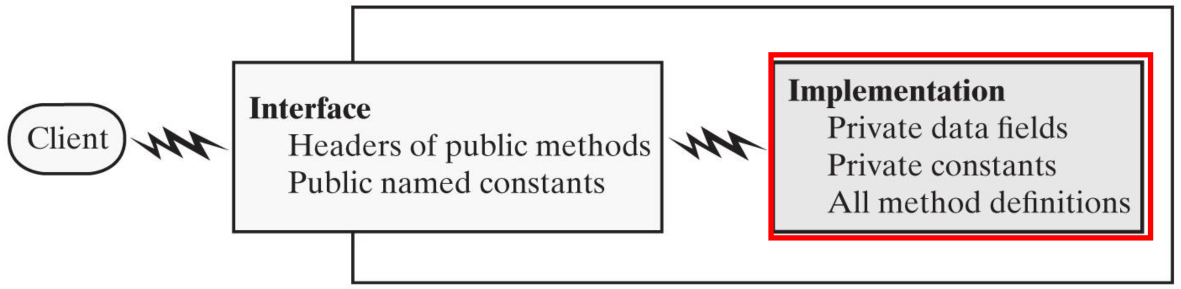


Program stack at three points in time (PC is the program counter)

Java Class Library: The Class Stack

- Found in **`java.util`**
- Methods
  - A constructor – creates an empty stack
  - **`public T push(T item);`**
  - **`public T pop();`**
  - **`public T peek();`**
  - **`public boolean empty();`**

Implementations of a Stack

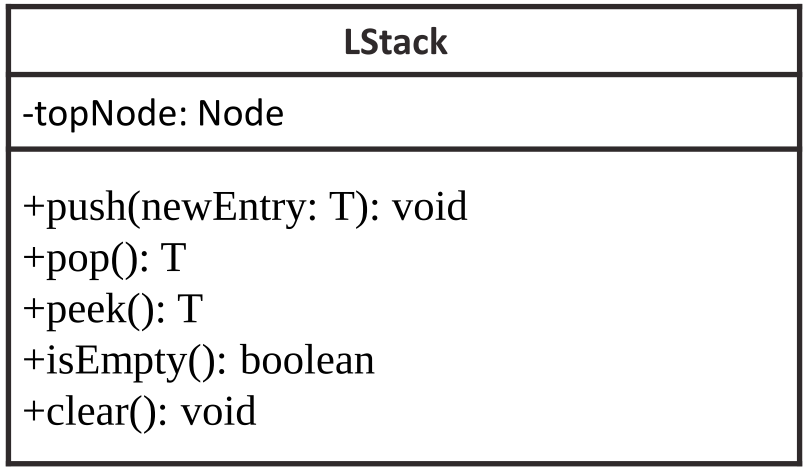


Implementations of a Stack

- A Linked Implementation
- An Array-Based Implementation
- A Vector-Based Implementation

Implementations of a Stack

- **A Linked Implementation**
- An Array-Based Implementation
- A Vector-Based Implementation

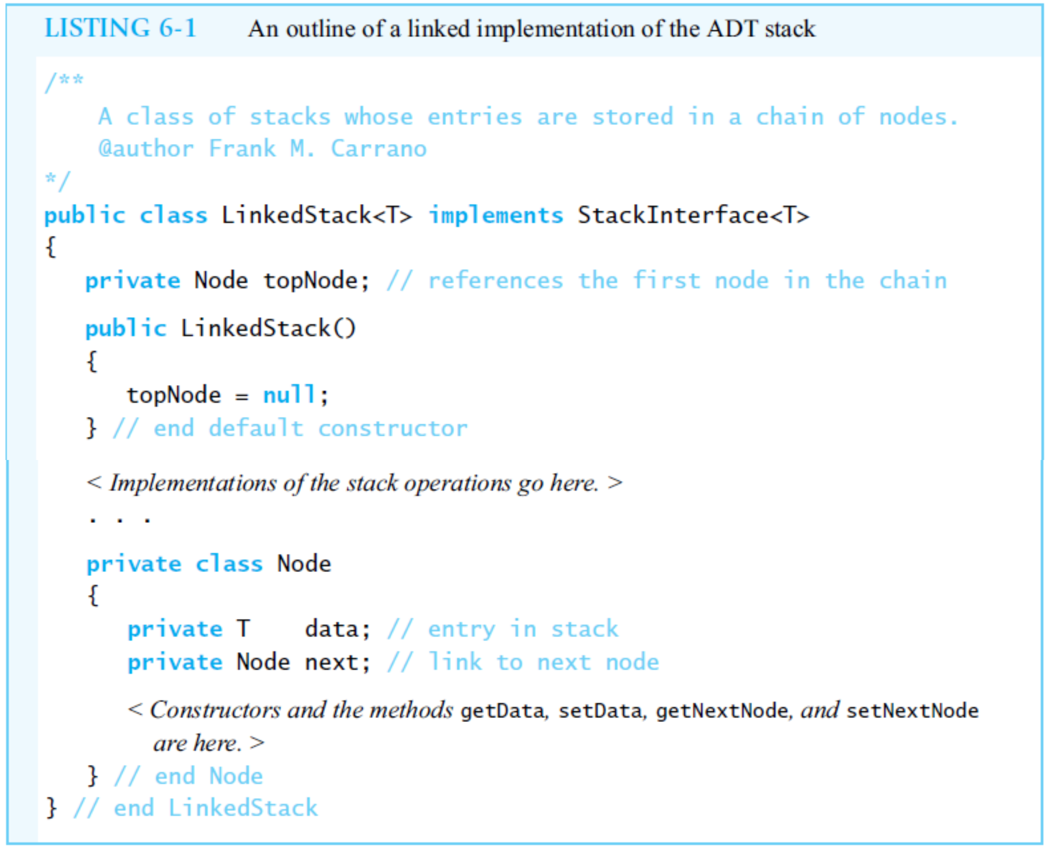


LStack

-topNode: Node

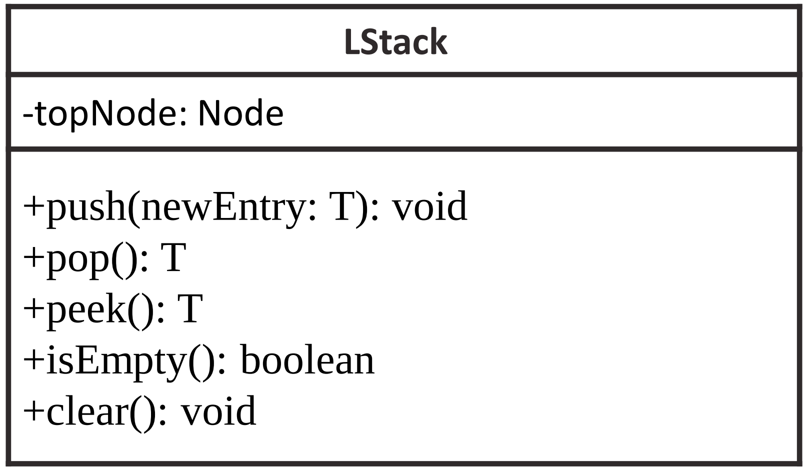
+push(newEntry: T): void
+pop(): T
+peek(): T
+isEmpty(): boolean
+clear(): void

Implementations of a Stack



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

```
/**
    A class of stacks whose entries are stored in a chain of nodes.
    @author Frank M. Carrano
*/
public 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

      < Constructors and the methods getData, setData, getNextNode, and setNextNode
        are here. >
   } // end Node
} // end LinkedStack
```



LStack

-topNode: Node

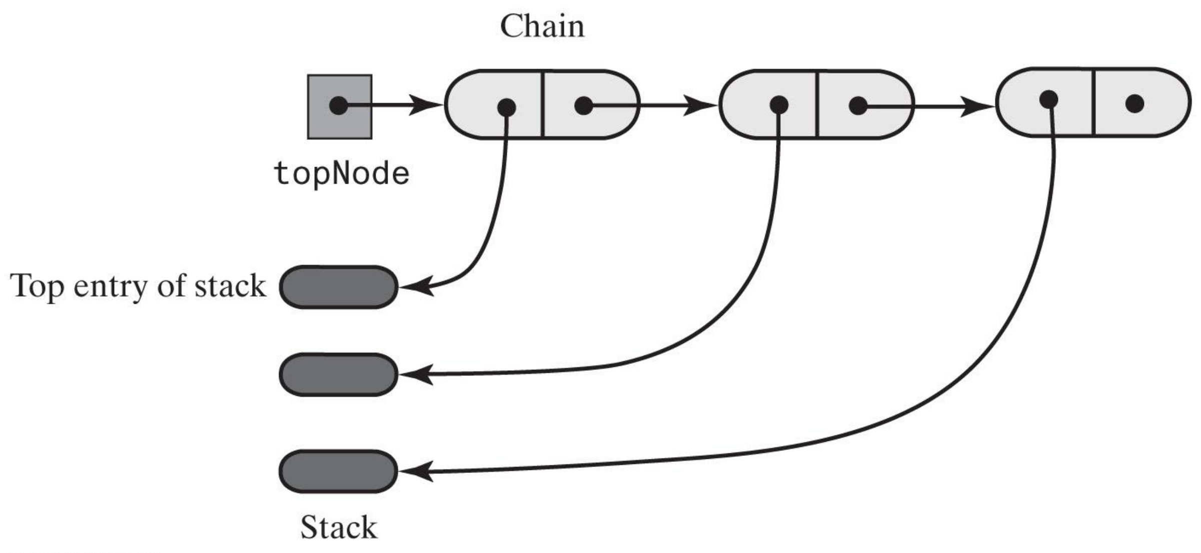
+push(newEntry: T): void
+pop(): T
+peek(): T
+isEmpty(): boolean
+clear(): void

Implementations of a Stack

- Each operation involves top of stack
  - **`push`**
  - **`pop`**
  - **`peek`**

- Head of linked list easiest, fastest to access
  - <u>Let the head of linked list</u> be the <u>top of the stack</u>

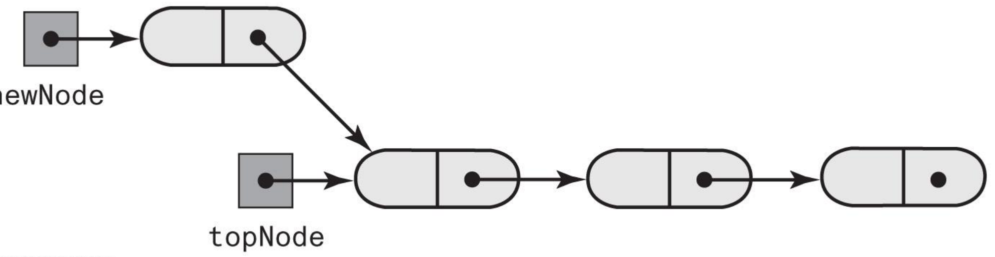
LStack
-topNode: Node
+push(newEntry: T): void +pop(): T +peek(): T +isEmpty(): boolean +clear(): void



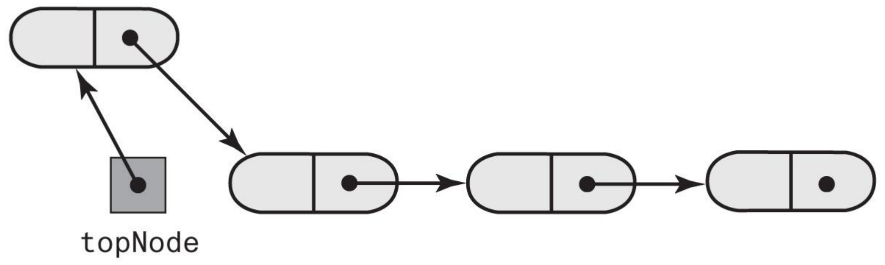
Implementations of a Stack

- Adding to the top

(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

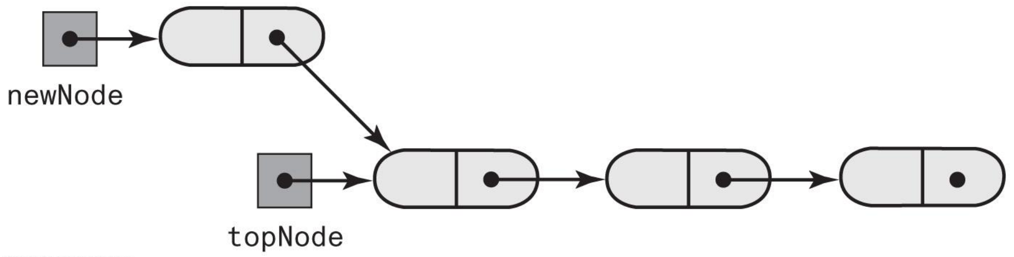


LStack
-topNode: Node
+push(newEntry: T): void +pop(): T +peek(): T +isEmpty(): boolean +clear(): void

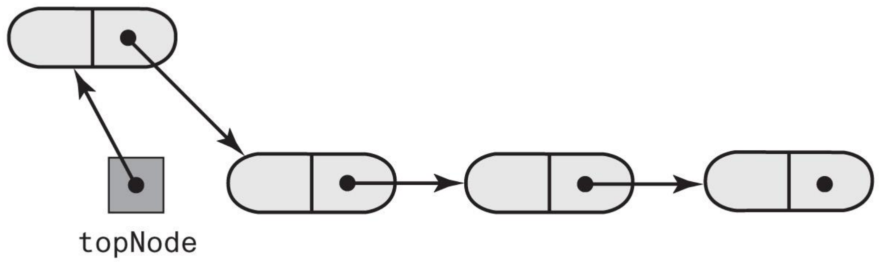
Implementations of a Stack

- Adding to the top

(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

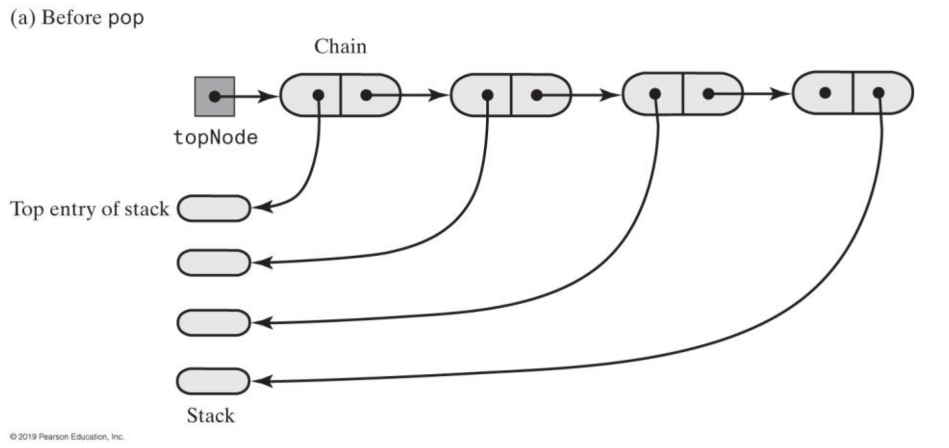


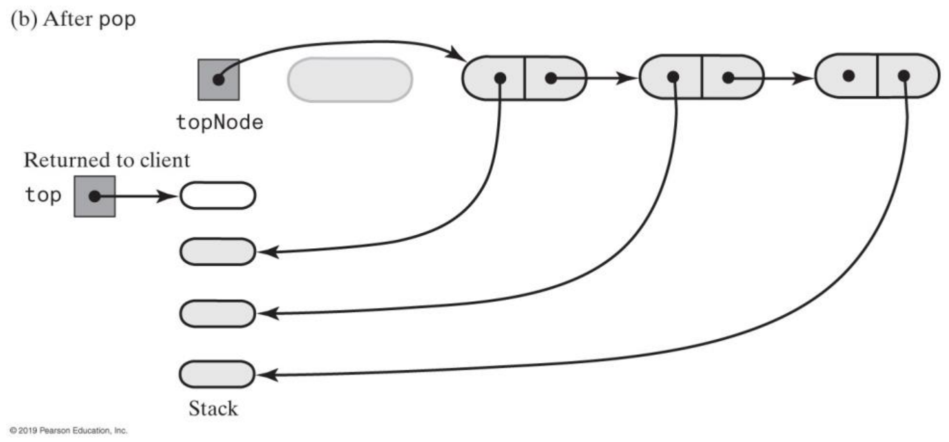
LStack
-topNode: Node
+push(newEntry: T): void +pop(): T +peek(): T +isEmpty(): boolean +clear(): void

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

Implementations of a Stack

- Adding to the top





LStack
-topNode: Node
+push(newEntry: T): void **+pop(): T** +peek(): T +isEmpty(): boolean +clear(): void

```
public T pop()
{
   T top = peek();  // Might throw EmptyStackException
   // Assertion: topNode != null
   topNode = topNode.getNextNode();

   return top;
} // end pop
```

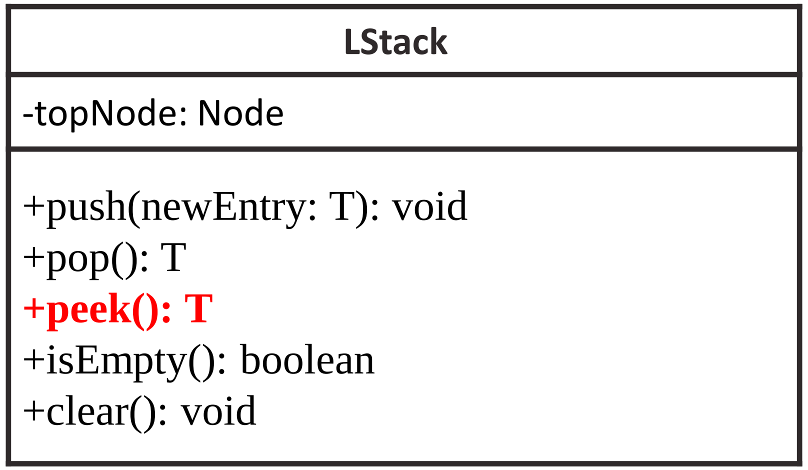
Implementations of a Stack

- Retrieving the top

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

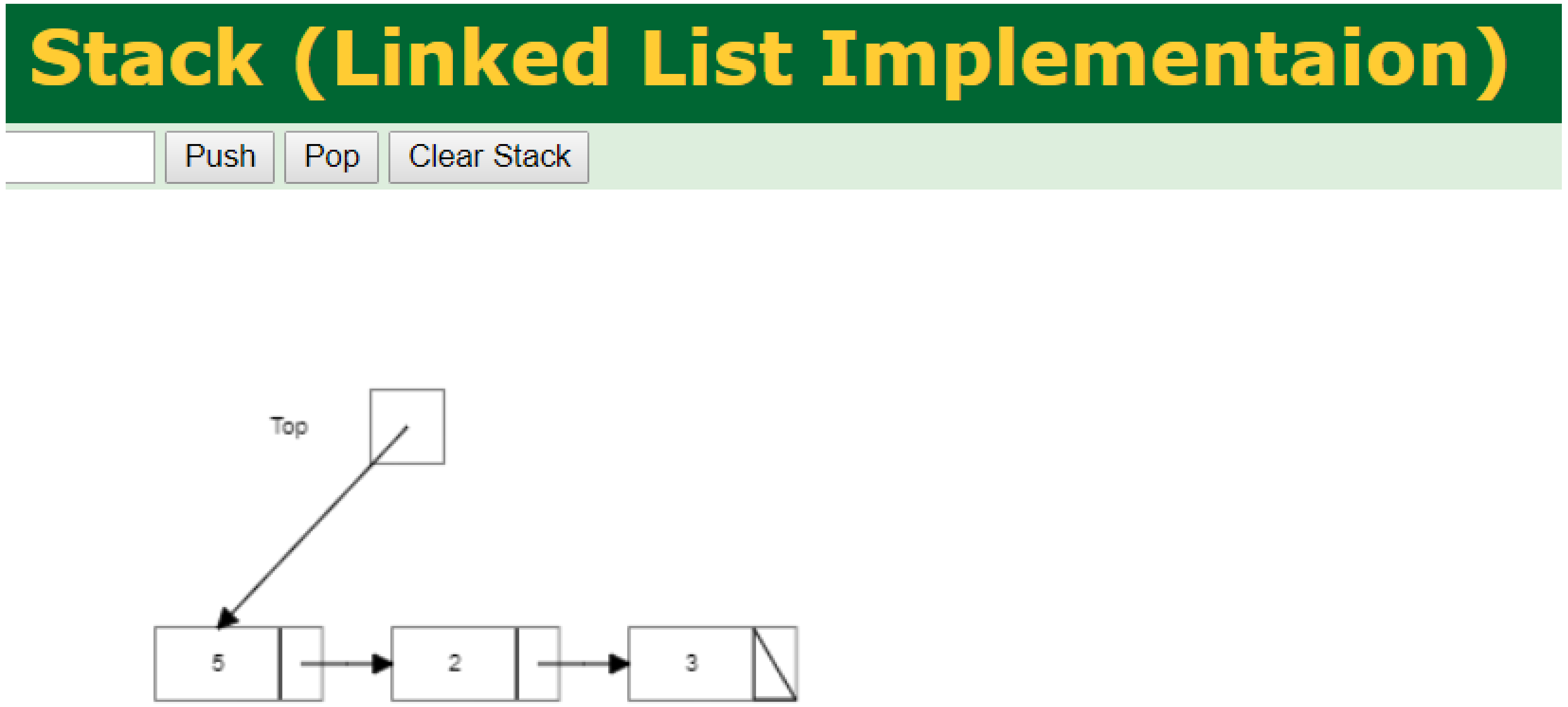
- The methods isEmpty and clear

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


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



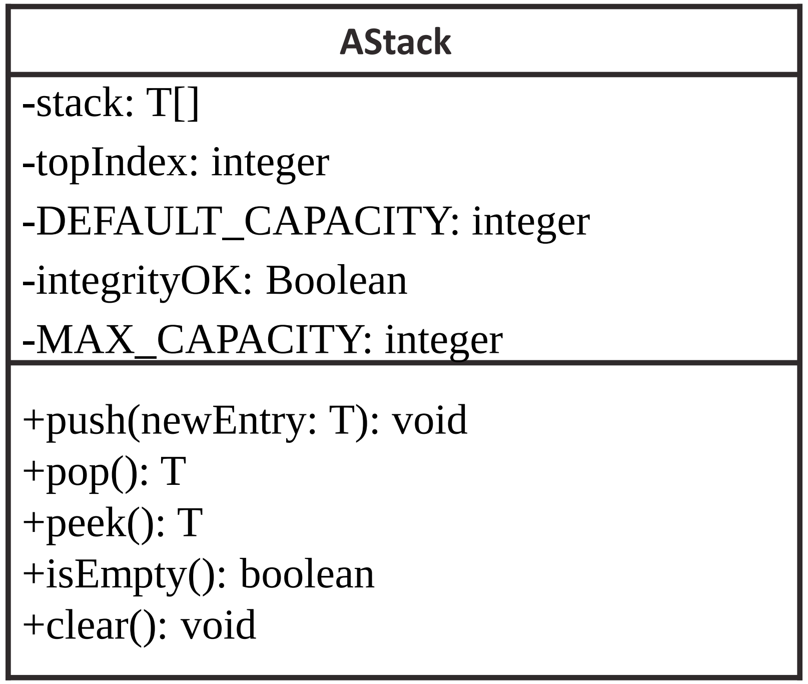
Interactive and Visualization Demo

- https://www.cs.usfca.edu/~galles/visualization/StackLL.html



Implementations of a Stack

- A Linked Implementation
- **An Array-Based Implementation**
- A Vector-Based Implementation



```
AStack
-stack: T[]
-topIndex: integer
-DEFAULT_CAPACITY: integer
-integrityOK: Boolean
-MAX_CAPACITY: integer

+push(newEntry: T): void
+pop(): T
+peek(): T
+isEmpty(): boolean
+clear(): void
```

AStack

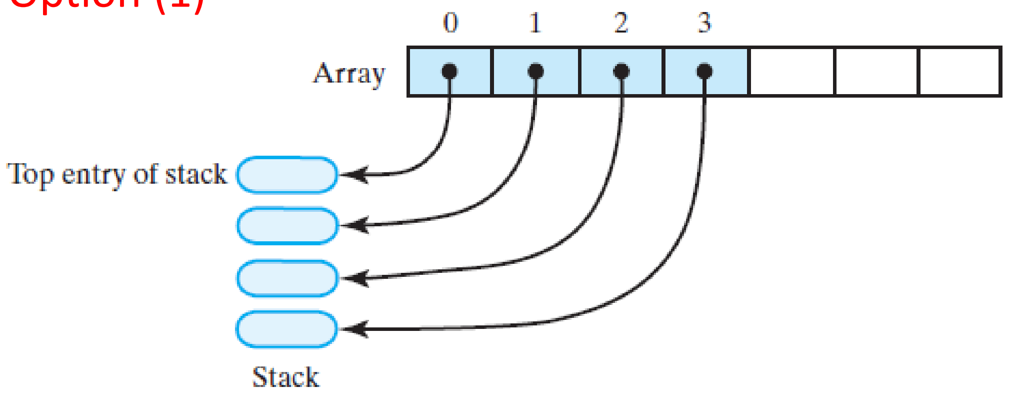
-stack: T[]
-topIndex: integer
-DEFAULT_CAPACITY: integer
-integrityOK: Boolean
-MAX_CAPACITY: integer

+push(newEntry: T): void
+pop(): T
+peek(): T
+isEmpty(): boolean
+clear(): void

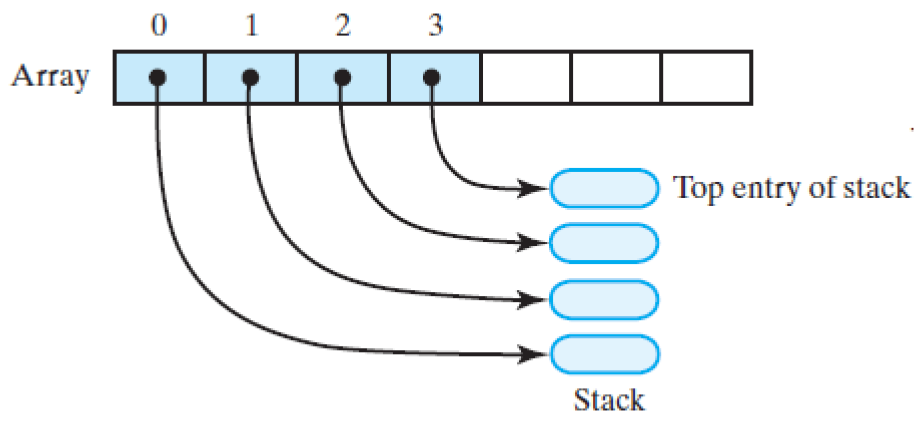
Implementations of a Stack

- Where should we place the stack's top entry?

Option (1)



Option (2)

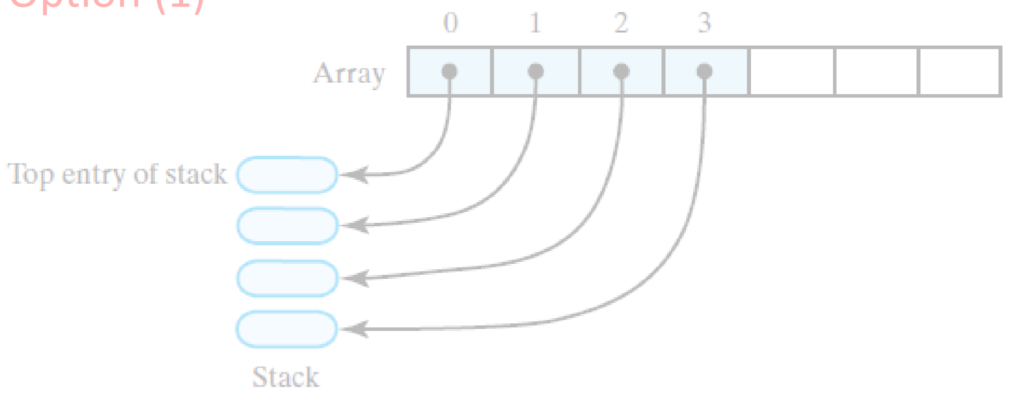


Implementations of a Stack

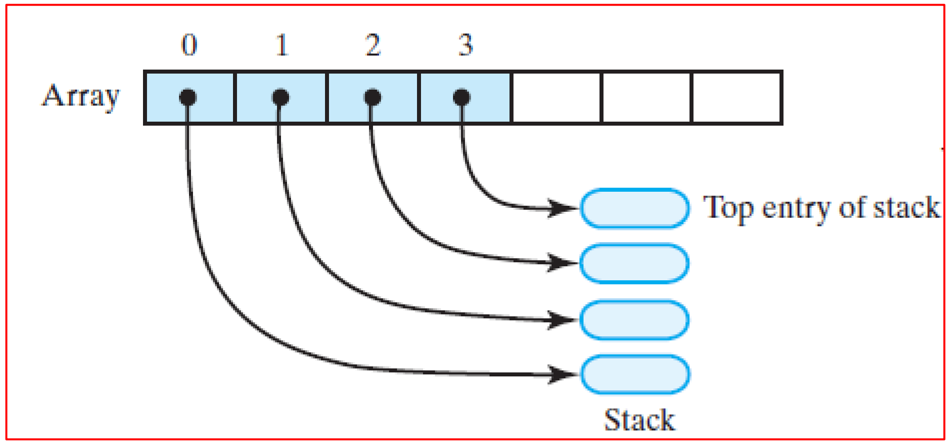
- Where should we place the stack's top entry?

AStack
-stack: T[] -topIndex: integer -DEFAULT_CAPACITY: integer -integrityOK: Boolean -MAX_CAPACITY: integer
+push(newEntry: T): void +pop(): T +peek(): T +isEmpty(): boolean +clear(): void

Option (1)



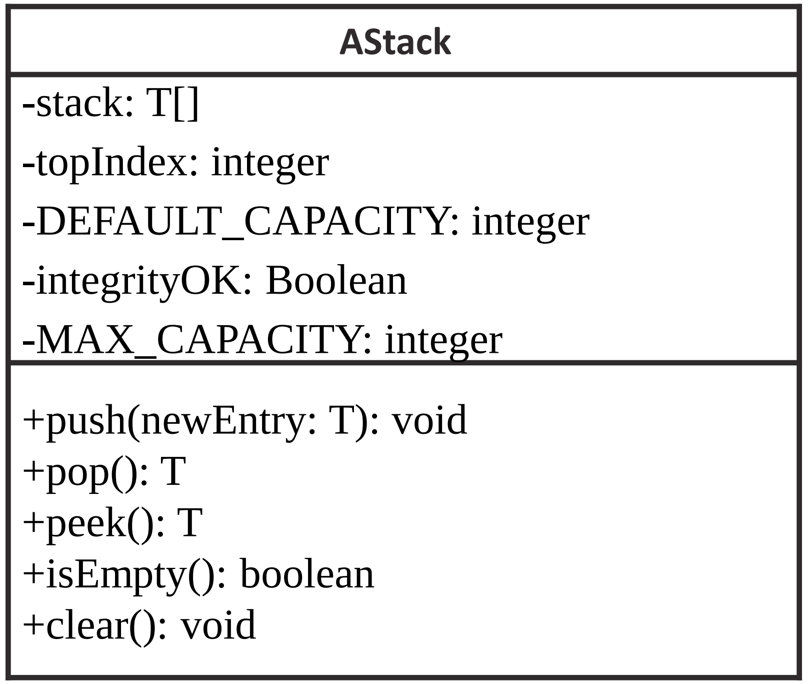
Option (2)



Efficient: the array's first element references the stack's bottom entry

Implementations of a Stack

```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
```



Implementations of a Stack

- Adding to the top

```
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
```

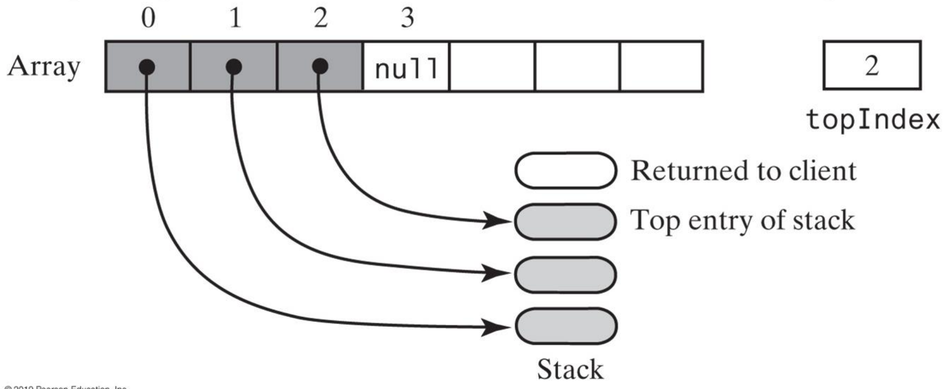
AStack
-stack: T[]
-topIndex: integer
-DEFAULT_CAPACITY: integer
-integrityOK: Boolean
-MAX_CAPACITY: integer
+push(newEntry: T): void
+pop(): T
+peek(): T
+isEmpty(): boolean
+clear(): void
-ensureCapacity(): void

Implementations of a Stack

- Removing the top

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

By setting `stack[topIndex]` to `null` and then decrementing `topIndex`

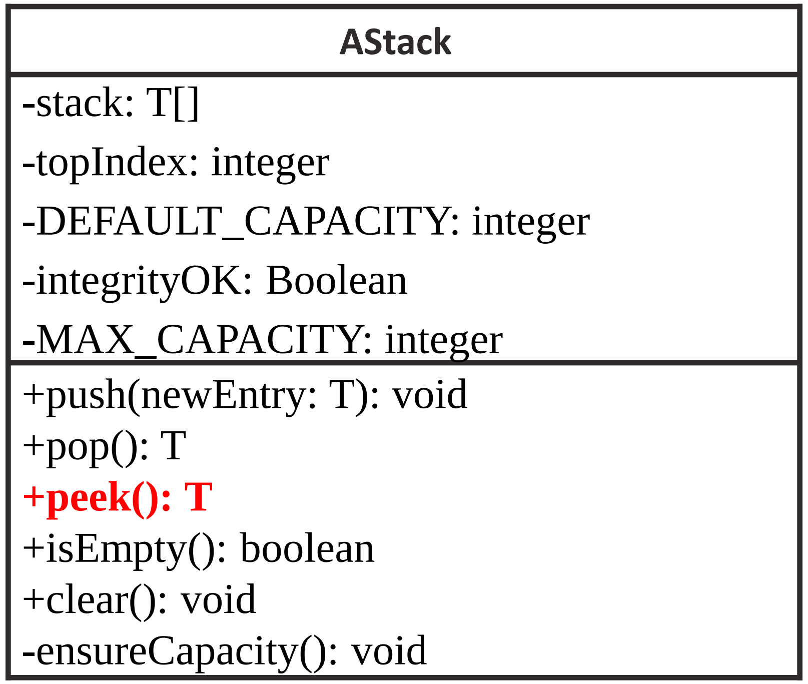


AStack
-stack: T[]
-topIndex: integer
-DEFAULT_CAPACITY: integer
-integrityOK: Boolean
-MAX_CAPACITY: integer
+push(newEntry: T): void
+pop(): T
+peek(): T
+isEmpty(): boolean
+clear(): void
-ensureCapacity(): void

Implementations of a Stack

- Retrieving the top

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



AStack

-stack: T[]
-topIndex: integer
-DEFAULT_CAPACITY: integer
-integrityOK: Boolean
-MAX_CAPACITY: integer

+push(newEntry: T): void
+pop(): T
+peek(): T
+isEmpty(): boolean
+clear(): void
-ensureCapacity(): void

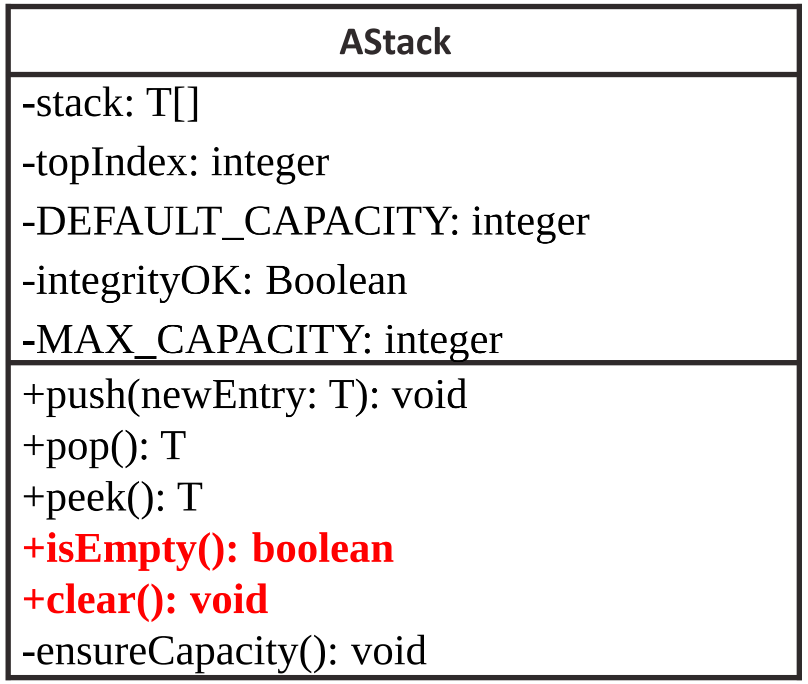
Implementations of a Stack

- Retrieving the top

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

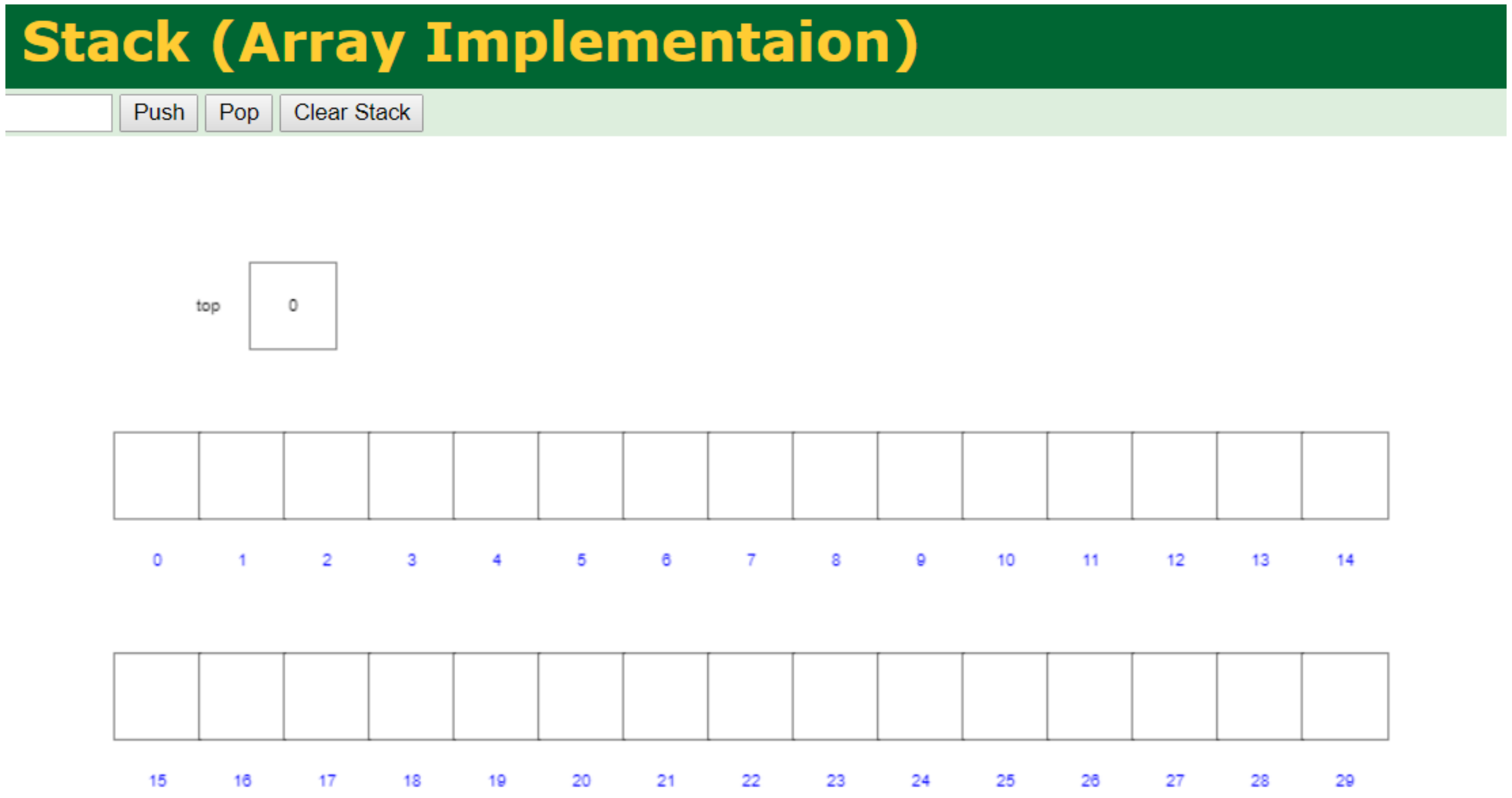
- isEmpty and clear

```
public boolean isEmpty()
{
      return topIndex < 0;
} // end isEmpty

public void clear()
{
      checkIntegrity();

      // Remove references to the objects in the stack,
      // but do not deallocate the array
      while (topIndex > -1)
      {
          stack[topIndex] = null;
          topIndex--;
      } // end while
      //Assertion: topIndex is -1
} // end clear
```



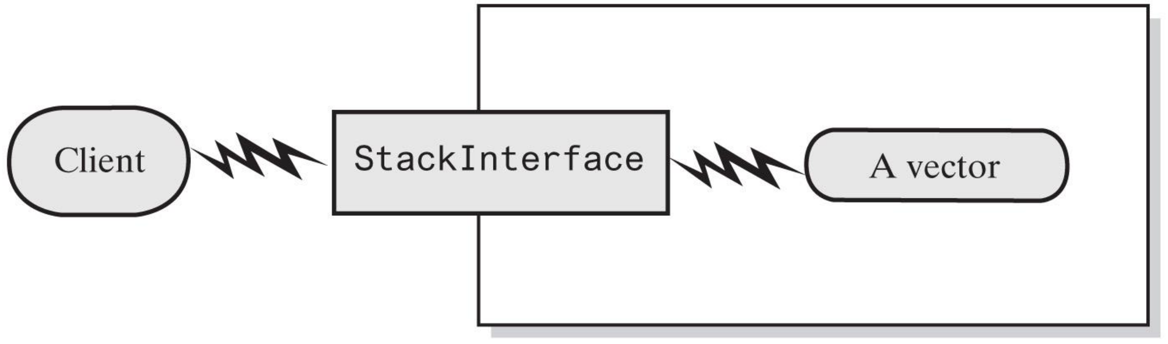
Interactive and Visualization Demo

- https://www.cs.usfca.edu/~galles/visualization/StackArray.html

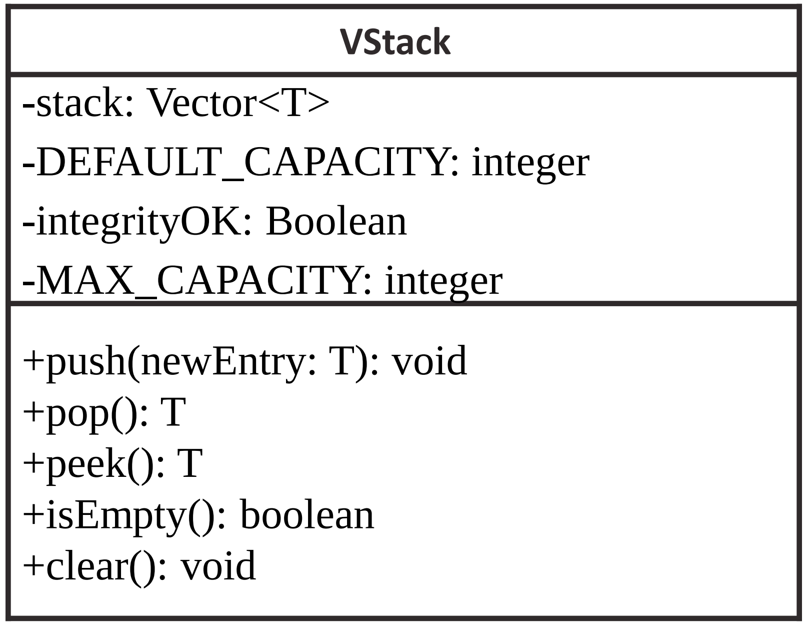


Implementations of a Stack

- A Linked Implementation
- An Array-Based Implementation
- **A Vector-Based Implementation**



Implementation of a stack



VStack

-stack: Vector<T>
-DEFAULT_CAPACITY: integer
-integrityOK: Boolean
-MAX_CAPACITY: integer

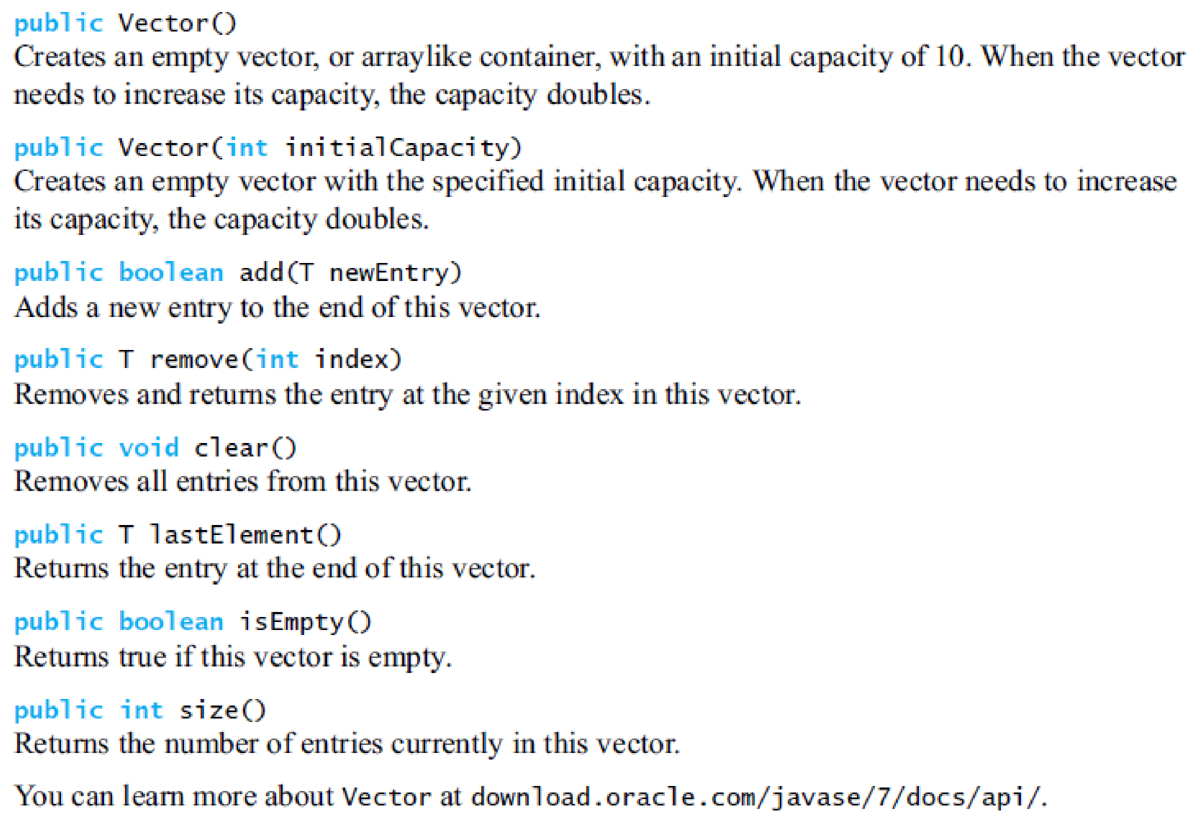
+push(newEntry: T): void
+pop(): T
+peek(): T
+isEmpty(): boolean
+clear(): void

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

- 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**



`public Vector()`
Creates an empty vector, or arraylike container, with an initial capacity of 10. When the vector needs to increase its capacity, the capacity doubles.

`public Vector(int initialCapacity)`
Creates an empty vector with the specified initial capacity. When the vector needs to increase its capacity, the capacity doubles.

`public boolean add(T newEntry)`
Adds a new entry to the end of this vector.

`public T remove(int index)`
Removes and returns the entry at the given index in this vector.

`public void clear()`
Removes all entries from this vector.

`public T lastElement()`
Returns the entry at the end of this vector.

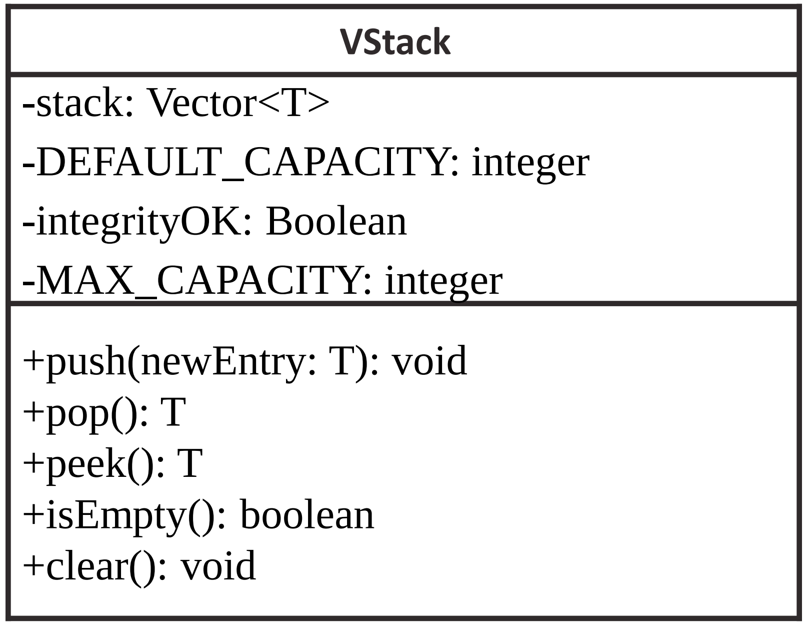
`public boolean isEmpty()`
Returns true if this vector is empty.

`public int size()`
Returns the number of entries currently in this vector.

You can learn more about `Vector` at `download.oracle.com/javase/7/docs/api/`.

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
```



VStack
-stack: Vector<T> -DEFAULT_CAPACITY: integer -integrityOK: Boolean -MAX_CAPACITY: integer
+push(newEntry: T): void +pop(): T +peek(): T +isEmpty(): boolean +clear(): void

Vector-Based Stack Implementation

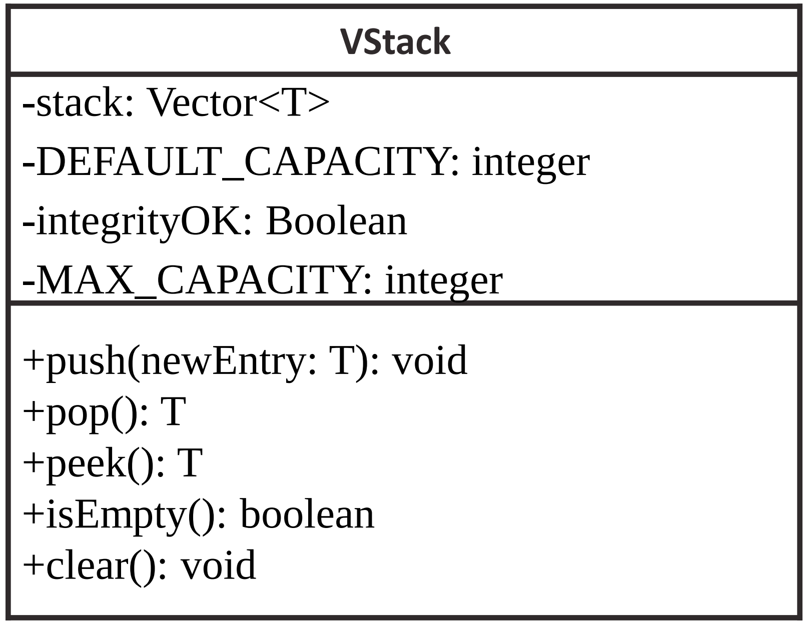
```java
public void push(T newEntry)
{
   checkIntegrity();
   stack.add(newEntry);
} // end push
```

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

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

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

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



```
VStack
-stack: Vector<T>
-DEFAULT_CAPACITY: integer
-integrityOK: Boolean
-MAX_CAPACITY: integer
+push(newEntry: T): void
+pop(): T
+peek(): T
+isEmpty(): boolean
+clear(): void
```

In-Class Exercises: Algorithm Analysis

- What is the **Big Oh** of each stack method in the best case and the worst case?

LStack
-topNode: Node
+push(newEntry: T): void +pop(): T +peek(): T +isEmpty(): boolean +clear(): void

AStack
-stack: T[] -topIndex: integer -DEFAULT_CAPACITY: integer -integrityOK: Boolean -MAX_CAPACITY: integer
+push(newEntry: T): void +pop(): T +peek(): T +isEmpty(): boolean +clear(): void -ensureCapacity(): void

VStack
-stack: Vector<T> -DEFAULT_CAPACITY: integer -integrityOK: Boolean -MAX_CAPACITY: integer
+push(newEntry: T): void +pop(): T +peek(): T +isEmpty(): boolean +clear(): void

Summary

- Stacks
- Implementations of a Stack

What I Want You to Do

- Review class slides
- Review Chapters 5 and 6

- Next Topics
  - ADT Queues, Deques, and Priority Queues