Abstract Data Types

Stacks

Curriculum Connections / Objectives

5.1.6 Describe the characteristics and applications of a stack.
5.1.7 Construct algorithms using the access methods of a stack.

Vermont

Massachusetts

Rhode Island

New Hampshire

Maine

Helpful Vocabulary

Abstract Data Type

A mathematical model or logical description of how we view the data and the operations that are allowed on it, without regard to how they will be implemented.

Data Structure

A data structure is a collection of data items, in addition, a number of operations are provided by the software to manipulate the data structure.

Static Data Structure

Arrays, such as the ones we have seen in java and that we saw last year with C, are allocated with a fixed and known storage requirement.

Dynamic Data Structure

Using linked lists to implement a data structure such as a stack enables the program to allocate memory as needed, so the maximum number of items in the structure need not be known.

Compare and contrast dynamic and static data structures.

Dynamic and Static data structures

DYNAMIC	STATIC
Memory is allocated to the data structure dynamically i.e. as the program executes.	Memory is allocated at compile time. Fixed size.
Disadvantage: Because the memory allocation is dynamic, it is possible for the structure to 'overflow' should it exceed its allowed limit. It can also 'underflow' should it become empty.	Advantage: The memory allocation is fixed and so there will be no problem with adding and removing data items.
Advantage: Makes the most efficient use of memory as the data structure only uses as much memory as it needs	Disadvantage: Can be very inefficient as the memory for the data structure has been set aside regardless of whether it is needed or not whilst the program is executing.
Disadvantage: Harder to program as the software needs to keep track of its size and data item locations at all times	Advantage: Easier to program as there is no need to check on data structure size at any point.

Source: http://www.teach-ict.com

Characteristics of a stack

The 'STACK' is a Last-In First-Out (LIFO) List.
Which means also that the first item in is the last item out. (FILO)
Only the last item in the stack can be accessed directly.

A **stack** is an ordered collection of items into which new items may be inserted and from which items may be deleted at one end, called the **top** of the stack.
-Aaron M. Tenenbaum and Moche J. Augenstein

Typical operations (methods) to support a stack

push:	add an item to the top of the stack
pop:	remove the top item of the stack (and return its value)
isEmpty:	returns the truth if whether the stack contains any items
peek (also known as **top**)	returns the top item of the stack, without altering the stack. This is often implemented as a pop followed by a push of the "popped" item.

Example:

Create a stack of state names, inserting the following state names:

Maine New Hampshire Vermont Rhode Island Massachusetts

1) To begin, suppose the stack is currently empty: The first item in the stack will be Maine

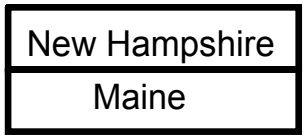
push("Maine")

The Stack

2) Next:

push("New Hampshire")

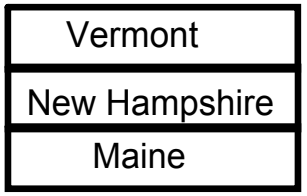
The Stack



3) Next:

push("Vermont")

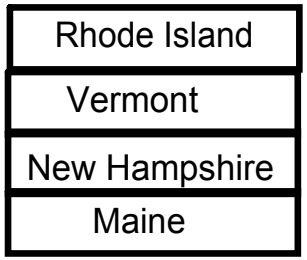
The Stack



4) Next:

push("Rhode Island")

The Stack



5) Next:

push("Massachusetts")

The Stack

```
//File: StateNode.java
//Purpose: Support a linked list of state names

public class StateNode {
   public String          state;
   public StateNode    next;

   StateNode() {
      state = null;
      next  = null;
      }

   StateNode(String s) {
      state = s;
      next  = null;
      }

   public String toString() {
     return state;
  }

} // end StateNode
```

The StateNode class contains a typical structure of a linked list. Namely, there are fields to contain data - such as

```
String          state;
```

and there are fields to "link" from one data item to another. In this case the

```
StateNode    next;
```

will link from one node to another.

```
//File: StateNode.java
//Purpose: Support a linked list of state names

public class StateNode {
   public String          state;
   public StateNode    next;

   StateNode() {
      state = null;
      next  = null;
      }

   StateNode(String s) {
      state = s;
      next  = null;
      }

   public String toString() {
     return state;
   }

} // end StateNode
```

From the StateNode class we can implement a stack using linked lists. Suppose we define a class called StateStack which implements a stack using nodes of class StateNode:

```
public class StateStack {
  private StateNode stack;  // this is the top
                            // of the stack
   StateStack() {
      stack = null;
   }

 //push : Add a new node to the stack
 public void push(String s) {
       StateNode newState = new StateNode(s);
       newState.next = stack;
       stack = newState;
       }
```

```
//File: StateStack
//Purpose: Implement a stack of StateNodes
//               using a linked list

public class StateStack {

    private StateNode stack;

    StateStack() {
        stack = null;
    }

   // let someone test to see if the stack is empty
   public boolean isEmpty() {
       return (stack == null);
   }

   //pop : return the top node of the stack, and remove the top node
   //      from the stack.
   public StateNode pop() {
            StateNode t = stack;
            if (stack != null) stack = stack.next;
            return t;
        }

   //push : Add a new node to the stack
   public void push(String s) {
           StateNode newState = new StateNode(s);
           newState.next = stack;
           stack = newState;
          }

    //peek : return the top node of the stack.
    public StateNode peek() {
          return stack;
        }

    public void printStates() {
        StateNode   tPtr = stack;
        while (tPtr != null) {
            System.out.println(tPtr.toString());
            tPtr = tPtr.next;
            }
    } // end printStates


} // end StateStack
```

```
//File: States
//Purpose: Create a stack with 5 state nodes
//         This is a demonstration of using stack operations
public class States {

    public static void main(String[] Args) {
        StateStack my_stack = new StateStack();

        my_stack.push("Maine");
        my_stack.push("New Hampshire");
        my_stack.push("Vermont");
        my_stack.push("Rhode Island");
        my_stack.push("Massachusetts");
        my_stack.printStates();
        System.out.println("\n----------------\n");

        StateNode j = my_stack.pop();
        my_stack.push("Maryland");
        my_stack.push("Connecticut");
        my_stack.push("Arkansas");
        my_stack.push("Delaware");
        my_stack.printStates();

        System.out.println("\n----------------\n");
        for (int i = 0; i < 4; i++)
            j = my_stack.pop();
        my_stack.printStates();


    } // end main


} // end States
```

A sample States class to test the functionality of the StatesStack.

Begin with an empty stack,

add some nodes using push

print out what we have so far,

pop one node,
push several more states

print what we have now,

pop a few nodes off the stack

and print the nodes that remain.

StateNode

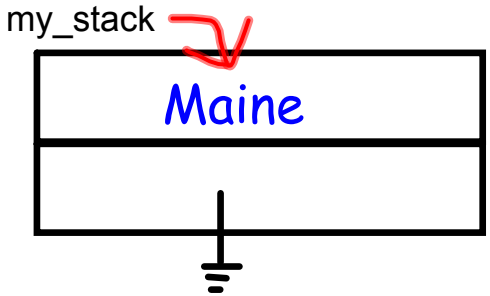
String state;

StateNode next;

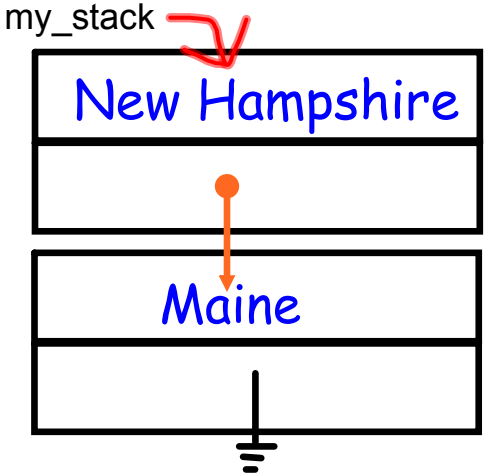
Trace the stack created by the following sequence:

```
my_stack.push("Maine");
my_stack.push("New Hampshire");
my_stack.push("Vermont");
my_stack.push("Rhode Island");
my_stack.push("Massachusetts");
my_stack.printStates();
```

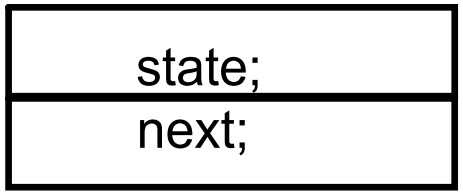
1) my_stack.push("Maine");



2) my_stack.push("New Hampshire");



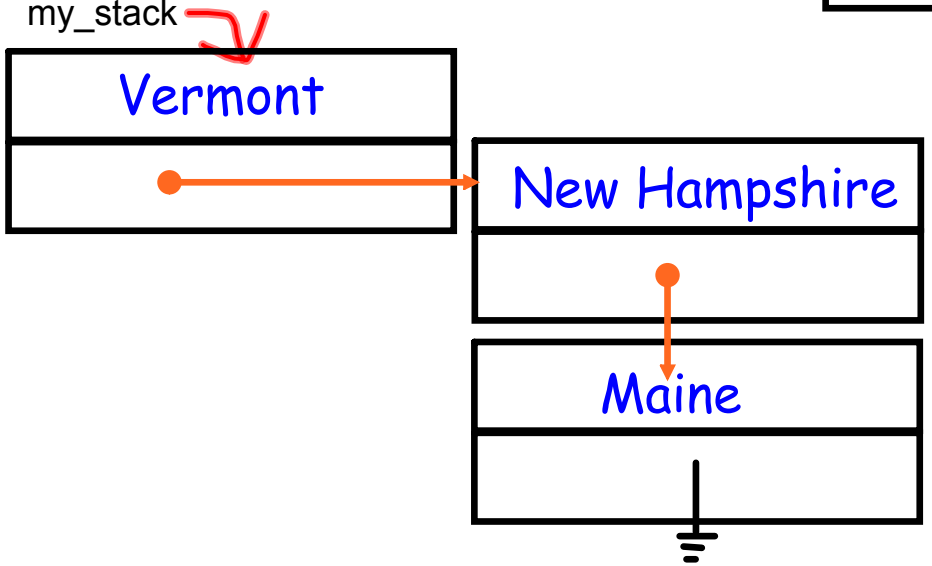
StateNode
String
StateNode



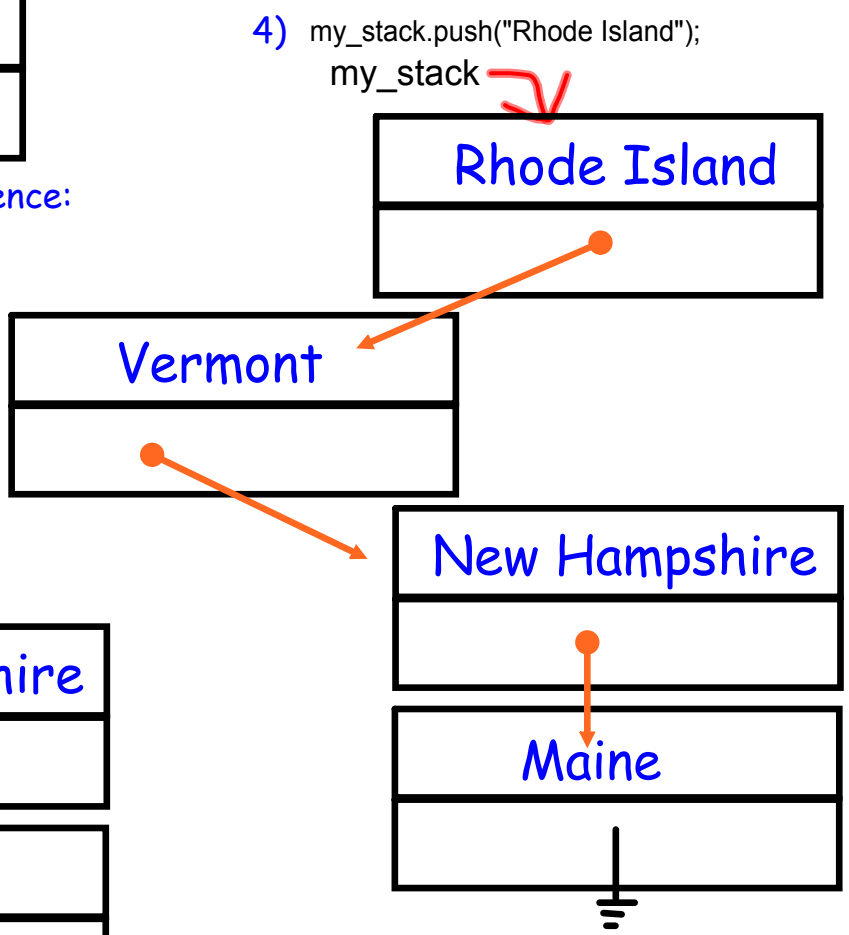
Tracing the stack created by the following sequence:

```
my_stack.push("Maine");
my_stack.push("New Hampshire");
my_stack.push("Vermont");
my_stack.push("Rhode Island");
my_stack.push("Massachusetts");
my_stack.printStates();
```

3) my_stack.push("Vermont");



4) my_stack.push("Rhode Island");



StateNode

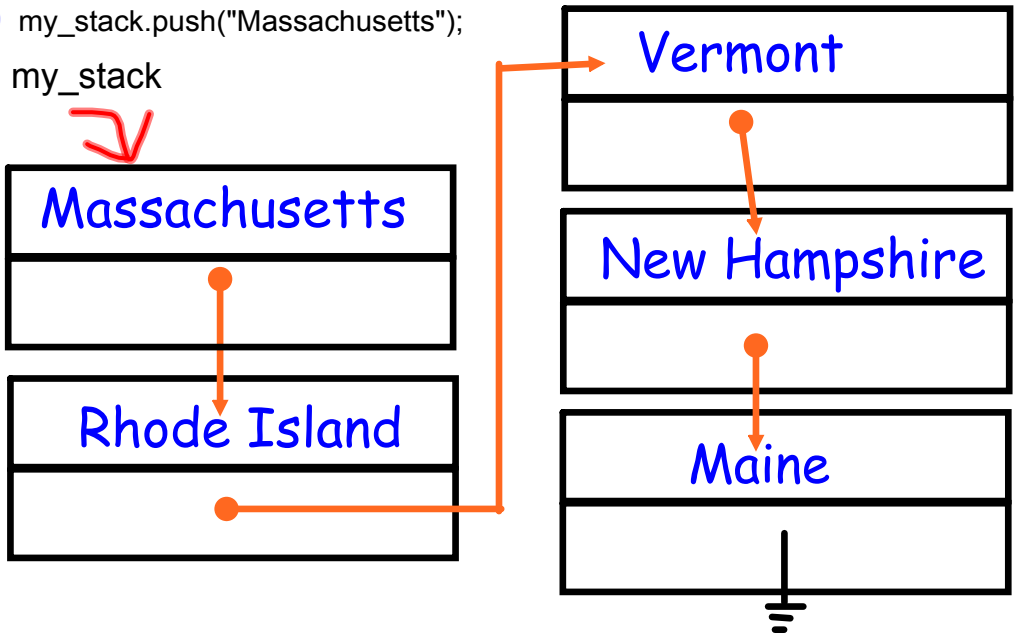
String state;

StateNode next;

Tracing the stack created by the following sequence:

```
my_stack.push("Maine");
my_stack.push("New Hampshire");
my_stack.push("Vermont");
my_stack.push("Rhode Island");
my_stack.push("Massachusetts");
my_stack.printStates();
```

5) my_stack.push("Massachusetts");



6) my_stack.printStates();

Massachusetts
Rhode Island
Vermont
New Hampshire
Maine

Exercise

Trace the following program segment that makes calls to a functioning stack:

```
public static void main(String[] Args) {
  StateStack my_stack = new StateStack();

  my_stack.push("Maine");
  my_stack.push("New Hampshire");
  my_stack.push("Vermont");
  my_stack.push("Rhode Island");
  my_stack.push("Massachusetts");
  my_stack.printStates();
  System.out.println("\n-----------------\n");

  StateNode j = my_stack.pop();
  my_stack.push("Maryland");
  my_stack.push("Connecticut");
  my_stack.push("Arkansas");
  my_stack.push("Delaware");
  my_stack.printStates();

  System.out.println("\n-----------------\n");
  for (int i = 0; i < 4; i++)
        j = my_stack.pop();
  my_stack.printStates();


 } // end main
```

Program Output:

Work area:

Exercise

Solution

Program Statements	Trace: Contents of the Stack	Program Output jbrennan:~/workspace $ java States

my_stack.push("Maine");
my_stack.push("New Hampshire");
my_stack.push("Vermont");
my_stack.push("Rhode Island");
my_stack.push("Massachusetts");
my_stack.printStates();
System.out.println("\n----------\n");
	Massachusetts
	Rhode Island
	Vermont
	New Hampshire
	Maine
	
	---------- 	
		Massachusetts
		Rhode Island
		Vermont
		New Hampshire
		Maine
		
		---------- 	

StateNode j = my_stack.pop();	Rhode Island Vermont New Hampshire Maine	Delaware Arkansas Connecticut Maryland Rhode Island Vermont New Hampshire Maine

my_stack.push("Maryland");
my_stack.push("Connecticut");
my_stack.push("Arkansas");
my_stack.push("Delaware");
my_stack.printStates();
System.out.println("\n----------\n");
	Delaware
	Arkansas
	Connecticut
	Maryland
	Rhode Island
	Vermont
	New Hampshire
	Maine
	---------- 	
		---------- 	
		
		Rhode Island
		Vermont
		New Hampshire
		Maine

for (int i = 0; i < 4; i++)
j = my_stack.pop();
my_stack.printStates();
	Rhode Island
	Vermont
	New Hampshire
	Maine	

Zoom level. Click to open the Zoom dialog box.