- Each of our stack implementations was a generic stack of objects, but the implementations can be changed to any other type without trouble to obtain a new implementation, such as a stack of `int`.
- Stacks have many uses in computer science. The evaluation and translation of arithmetic expressions are two common uses.

## Solutions to Self-Test Exercises

1. Push a D; push an A; push an H; push an L; pop an L; pop an H; pop an A; pop a D; the output is LHAD.

2. Pop one item, storing it in a local variable called `top`. Then peek at the next item, storing it in another local variable called `result`. Push the `top` back on the stack and return the `result`.

3. Add a new case to the switch statements in both `evaluate` and `evaluateStackTops`. In `evaluateStackTops` the new case calculates `operand1` raised to the power of `operand2` and pushes the result back onto the numbers stack.

4. The modification is most easily accomplished in the method `evaluate`. Within this method, you can add a bit of code in the section that reads an operator symbol. The new code should look ahead to see whether the very next character of the Scanner is a slash (as described in Appendix B). If so, read and discard the rest of the line (instead of pushing the operator onto the stack).

5. Here are some illegal expressions that are caught: too many right parentheses (3+4)), a missing operand (3 4), a missing right parenthesis (3+4.

6. Here are some illegal expressions that are not caught: too many left parentheses ((3+4), missing left parentheses 3+4).

7. The solution is the same as the example that starts on page 316, but the numbers are changed. The final value is 24.

8. The numbers stack grows large if the input expression has parentheses that are nested deeply. For example, consider the input expression `(1 + (2 + (3 + (4 + 5))))`. By the time the 5 is read and pushed on the stack, the 1, 2, 3, and 4 are already on the numbers stack. In this example, there are four nested subexpressions, and we need to push five numbers onto the numbers stack. So, the general stack size will be one more than the depth of the nesting.

9. The evaluation method is linear. If $n$ is the length of the input expression, then the main loop cannot execute more than $n$ times (since each iteration of the loop reads and processes at least one character). Moreover, each iteration performs no more than a constant number of operations, so $n$ iterations will do no more than a constant times $n$ operations.

10. The array version: `data[manyItems-1]`. The linked list version has the top at the head of the list.

11. The function should return `data.length` minus `manyItems`.



12. Here is one of the two solutions (for the linked list):

```
public E second( )
{
   if (top == null)
      throw
      new EmptyStackException( );
   if (top.link == null)
      throw
      new NoSuchElementException( );

   return top.link.data;
}
```

13. We maintain only a head reference.

14. The constructor isn't needed because any instance variable that is a reference is automatically initialized to `null`.

15. Our `size` implementation uses `listSize`, which is linear time. For a constant-time implementation, you could maintain another private instance variable to continually keep track of the list length.

16. An `EmptyStackException`, which is defined in `java.util.EmptyStackException`.

17. 42

18. There is no need for a second stack of operation symbols, because each operation is used as soon as it is read.

19. Prefix: `* + 7 3 2` Postfix: `7 3 + 2 *`

20. The trace is the same as the computation at the bottom of Figure 6.8 on page 336, except that the numbers are three times as large.

21. The trace is much the same as the computation in Figure 6.11 on page 341, except that the operations are different.

## PROGRAMMING PROJECTS

**1** Write an applet that is a simple stack-based calculator. The calculator should have a text field where you can enter a double number and a text area that displays a stack of numbers. There should be an `enter` button that reads the number from the text field and pushes this number onto the stack. There are also several buttons for operations such as +, –, *, and /. When an operation button is pressed, the top two numbers are popped off the stack, and the operation is applied. The first number popped is always the second operand.

**2** Choose one of the stack implementations and implement a method with this specification:

```
Object itemAt(int n)
// Precondition: 0 <= n and n < size( ).
// Postcondition: The return value is the
// item that is n from the top (with the top at
// n = 0, the next at n = 1, and so on). The
// stack is not changed.
```

Throw a `NoSuchElementException` if the precondition is violated (from `java.util.NoSuchElementException`).

**3** In this exercise, you will need the `itemAt` method from the previous programming project. Write a program that prints all strings with at most $n$ letters, where the letters are chosen from a range `first...last` of characters. The following is an outline for an algorithm to do this using a stack. Your program should use a stack to implement this algorithm: