CSPP 511-01:
Introduction to Object-Oriented Programming

Harri Hakula

Ryerson 256, tel. 773-702-8584
hhakula@cs.uchicago.edu

July 17, 2000
Outline

- Reverse Polish Notation
- Dynamic vs Static Data Structures
- List
- Stack Implemented with List
- Encapsulation
Reverse Polish Notation

In standard math notation we use the infix notation:

\((2 + 3) \times 3 = 15\)

where the operators are between the operands, and parentheses are used to order the calculations. We can even nest these operations:

\(((2 + 3) \times 4 - (4 + 3))/2\).

Inner parentheses are evaluated first, of course.

Consider: How can a programmer detect, what is an inner statement?

Answer: By reading the whole statement first and then finding the right order of evaluation.
Reverse Polish Notation Cntd.

There is a better way: Let us use postfix notation instead. Now, 

\[(2 + 3) \times 3\] becomes \[2 \ 3 \ + \ 3 \times\], and \[((2 + 3) \times 4 - (4 + 3))/2\] becomes \[2 \ 3 \ + \ 4 \times \ 3 \ 4 \ + \ -2/\]. In other words, every operator acts on the last two calculated or given values. Yes, every value is simply put to a stack, and operators operate on that stack.

\[2 \ 3 \ + \ 3 \times\] is push(2), push(3), push(pop() + pop()), push(3), push(pop() * pop()).

This notation which is far better than the standard infix, is called Reverse Polish Notation (RPN).
Command-Line Calculator

Let us write a calculator which uses the postfix notation. Just to make everything simple we consider only a command-line version with integers here.

We want to call the calculator like this: java RPN 2 3 + 3 *
It should then print the solution.

We already have a stack of ints, we must add the logic to identify ints and operators.
Command-Line Calculator Cntd.

The arguments are given as Strings. Our task should then be to distinguish between integers and operators. However, there is an additional problem: minus sign can be either unary (sign) or binary.

What is a number? A number is a sequence of digits which may be preceded by a sign.

What is an operator? A single character.
public static char analyze(String arg) {
    int i, start = 0, len = arg.length();
    if((arg.charAt(0) == '-') && (len > 1))
        start = 1;
    for(i = start; (i < len) &&
        Character.isDigit(arg.charAt(i)); ++i)
    {
        if(i == len) return NUMBER;
        if(len > 1) return UNKNOWN;
        return arg.charAt(0);
    }
}
Command-Line Calculator Cntd.

for(int i = 0; i < args.length; ++i) {
    int left = 0, right = 0;
    switch( analyze(args[i]) ) {
        case '+': right = stack.pop();
                  left = stack.pop();
                  stack.push(left + right);
                  break;
        ...
        case NUMBER:
                  stack.push(Integer.parseInt(args[i]));
                  break;
        default: /* Error */ break; } }
Dynamic vs Static Data Structures

Note that our calculator is still has a limited capacity, since our Stack implementation is limited in this respect. We could, of course, allocate a new array, copy the contents of the old one, and let the new array be the storage array. However, the maximum capacity is then only one half of the actual capacity.

Since the size of the array is fixed once it has been allocated, array is a static data structure. A dynamic data structure does not have a fixed size.

This flexibility comes at a cost, however.
List

List is the basic dynamic data structure. Unlike in the case of array, the data is not stored in single contiguous memory location, but is organized in such a way that if the starting point is known it is possible to visit every item in a list.

This is accomplished by letting every item in the list to point to the next one in the list. The last item points to a special object, null, which is a universal "ground object" or null object.

Implementation of this data structure is relatively simple. Here we present a version which is not an ADT.
List Implementation (Not ADT!)

public class Link {
    public Link(int value, Link next) {
        value_ = value;
        next_ = next;
    }
    public int value() { return value_; }
    public Link next() { return next_; }
    private int value_;  
    private Link next_;  
}
List Implementation (Not ADT!) Cntd.

Link root = new Link(1, null);
Link second = new Link(2, root);
root = second;

Our List is not an ADT, since the user must know that it is implemented in terms of Link. Remember, that the last assignment root = second; simply sets the pointer. root.next().value() is 1!
List Implementation Cntd.

Note, that in JAVA an object is not instanciated before the operator new has been called. Therefore, private Link next_; defines a pointer to a memory location holding exactly one object of type Link. (The name of the variable can be thought of as a pointer.)

When the constructor is called, the pointer is set to refer to some particular object of type Link. Universal null object is of any type.
List: Analysis

**Size** Unlimited. (The operator `new` will eventually fail.)

**Organization** The first item is special. Access to elements depend on their distance from the first one.

**Deletion** Let the items be $A \rightarrow B \rightarrow C$, where a connection via `next_` is denoted by $\rightarrow$. Setting $A.\text{next}_ = C$; removes $B$. Note that our implementation does not support this since we did not include the assignment above to our API. However, our implementation makes it easy to remove the first element.
Implementing Stack with List

```java
public class IntStack {
    public IntStack() {
        first_ = null;
    }
    public void push(int value) {
        Link lnk = new Link(value, first_);
        first_ = lnk;
    }
    public boolean isEmpty() {
        return (first_ == null);
    }
}
```
public int pop() {
    if (first_ == null) {
        System.out.println("Error: stack is empty, cannot pop.");
        return 0;
    }
    int value = first_.value();
    first_ = first_.next();
    return value;
}

private Link first_;
Encapsulation

Notice, that since our Stack is an ADT, every object or program (well, aren’t they the same in JAVA . . . ) using our Stack gets the benefits for free.

There is still one annoyance though, we must always specify the type of the objects we want to store in our Stack (or Link for that matter).

Everything we have covered so far can be implemented in relative ease with just about any language. It is the solution to the type problem which is the foundation of the object-oriented programming. But, we are not ready yet . . .