

COSC 2006 –Data Structures I

Assignment #4

Due: Friday, Nov 22th, 2013

ADT Stack

Objectives

- To gain understanding of how an implementation of an ADT is used by an application program.
- To learn how expressions can be evaluated at run-time.
- To become familiar with how infix expressions can be converted to postfix expressions.

Introduction

Your task is to design a program to implement a calculator. The calculator will take an infix expression convert it to a postfix expression and then evaluate it. In this draft, the infix expression will consist of operands that are single digits and the operators {+, -, *, /}. An example of such an expression is $2+3*4/6$. This will translate to $234*6/+$ and will then be evaluated as 4.0. To get a grade of **100** in this draft, your program must evaluate infix expressions containing parentheses such as $(2+4)*(8-2)/(11-9)$.

Your project can be divided into two parts:

- A Converter class that will convert the input string to postfix.
- A Calculator class that will evaluate the postfix expression.

Normally we would use a stack of char to do the conversion and a stack of double to do the evaluation. Java, however, allows us to use a stack of Object for both (please check API). Unfortunately the primitive types, in our case double and char are not classes. However, we can use wrapper classes Character and Double to accomplish our goal. Since both of these are subclasses of Object, instances of both these types can be pushed on the stack. Its like assigning an instance of a smaller class to an instance of a wider class. For instance, when pushing ans make it a field of a Double instance variable and push that instance variable on the stack by using:

```
prim = new Double(ans);  
stack.push(prim );
```

Here the Double instance variable is prim.

Special consideration must be given to removing items from the stack. Since the Object class is wider than the Double class, writing

```
prim = stack.pop();
```

will cause an error since you cannot assign an object of a wider class to an object of a narrower class without type casting. So you must write:

```
prim = (Double)stack.pop();
```

To get the primitive double value, use

```
System.out.println(  
prim.doubleValue() );
```

where `doubleValue()` is the wrapper class method that extracts the double value from the object `prim`. For the Character wrapper class don't use the `charValue()`. Use the `toString()` method implicitly.

In both the Converter and Calculator classes, use a StackADT that checks for an empty stack when popping and a full stack when pushing. You should write your own `MTstackException` and `FullStackException` classes to do this. As a consequence, you should do no stack checking in the driver program except when popping the stack converting from infix to postfix.

The Converter class

This uses the following methods

1. The boolean method `isOperator(char ch)`
2. The boolean method `isOperand(char ch)`
3. The boolean method `precedence(String op1, char op2)` which determines if the incoming token `op2` has precedence over the top of the stack `op1`.

The Calculator class

This uses the following methods

1. The boolean method `isOperator(char ch)`
2. The boolean method `isOperand(char ch)`
3. The double method `result(double op1, double op2, char op)` which determines the value of the binary operator `op` operating on the operands `op1` and `op2`.

Strategy for writing the program

Here is a recommended way of writing the program. If you feel comfortable skipping some steps to get to the final result, then by all means do.

1. Write the Calculator class using a stack of Object.
2. Write the Converter class using a stack of Object and instead of it printing the final postfix string, it should return a String.
3. Have the Calculator class call the Converter class in order to get a postfix expression as input.

For this draft, write your program so that there are three files. That way it is easier to compile (one file at a time). Here are the steps:

1. A Converter class written so that the main method is rewritten as a method that returns a string. Let's call it toPostfix() with the signature: public String toPostfix()
2. A Calculator class that instantiates the Converter class so that it can access method toPostfix().
3. A StackADT class and the exception classes in one file.

Sample input

- $3+4*5/6$
- $(3+2)*(4-2)/(8+7)$
- $(4+8)*(6-5)/((3-2)*(2+2))$

Your output should show the converted postfix string and the result of the calculation. For instance the last sample input will produce the following in the output window:

```
type your infix expression
(4+8)*(6-5)/((3-2)*(2+2))
```

```
converted string is 48+65-*32-22+*/
```

```
answer is 3.0
```

Submission:

- Hand in your complete Java source code; and a copy of the results produced
- Upload your source code to CMS
- Demonstrate your program to TA