

Project 8 - 20 points

Write a Java program to demonstrate using stacks. This project will use two stacks to evaluate infix arithmetic expressions. This will act similar to the way the Java compiler may evaluate simple expressions. The `GenericStack` object being created will be a generic object.

Given:

A file of arithmetic expressions passed on the command line.

You will need for `d8.java`:

A method to completely evaluate expressions:

```
public static int eval(BufferedReader src);
```

OR

```
public static int eval(String s);
```

A function to apply a single binary operation (ie, apply):

```
public static void apply(GenericStack<Integer> v, GenericStack<Character> o);
```

A function to determine operator precedence. (ie, prec):

```
public static int prec(char op);
```

You will need for `GenericStack.java`:

A method for pushing and popping a stack:

```
public void push(T item);  
public T pop() throws EmptyStackException;
```

Stack checking tools:

```
public T top();  
public boolean isEmpty();
```

Generic Node:

```
private class Node<T>
```

If you choose the `BufferedReader` class (as defined in the `java.io` package) to solve the problem, a following useful set of methods includes:

```
public int read();  
public void mark(int);  
public void reset();
```

- Expressions will not cross line boundaries.
- All expression evaluation will occur in the `eval()` method. There will be two stacks: an operator stack and a value stack.
- The `apply()` method should take both stacks as parameters and pop one operator from the operator stack and two operands from the value stack. Once the operation is applied the result

- should be pushed back onto the value stack.
- Expressions will have the following components:
 - Integers
 - Binary Operators {*, /, %, +, -}.
 - Parentheses for altering precedence.
 - Whitespace which can be ignored.
- All expressions are considered to be correct in syntax.
- Precedence rules are the same as for Java. Parentheses { (,) } first, then { *, /, % } and lastly { +, - }. Operators of like precedence are evaluated from left to right.
- You should pass `prec ()` the operator and have it return an integer. This is so you can compare the precedence of the operator on the stack with the one from the input stream.
- Be certain to apply whatever is left on the value and operator stacks once the expression in its string form has been exhausted. The result will be the only value left on the value stack.

Consider the following evaluation rules when building your program:

The rules for evaluation of expressions in `d8` are as follows:

| Category | What to do |
|----------|---|
| ----- | ----- |
| (| push onto operator stack |
| val | push onto value stack |
|) | while (top operator stack != '(') apply pop '(' from operator stack |
| op | if (! empty operator stack) while (!empty operator stack && (prec(currop) <= prec(top operator stack))) apply push currop |

When you've run out of input on the current expression:

```
while ( ! emptystack( opstk ) )
    apply
```

This project should be called `d8.java` and submitted using:

```
$ submit jojo d8.java
$ submit jojo GenericStack.java
```