Language Processors Lab 2 - Week 2 (SOLUTION)

The Solution to Lab2 can be found in Moodle under the Lab 2/Solution space. It is made of this document, the complete LexTest.jj file and an updated version of the `StackMachine.java` file (the order of the operators for some operations was wrong).

Most of the implementation of the RPN calculator is done by `StackMachine.java`. The JavaCC file contains the definition of the grammar of RPN expressions i.e. a sequence of numbers and operators, and Java code (semantic actions) that , after parsing the input string into tokens, 'feeds' the stack machine.

You were given code that implemented most of the JavaCC side of the calculator, with the exception of the handling floating points. Key to the latter is the understanding of how the RPN class handles integers, as shown by the following code extract:

```
switch(kind) {
  case INTEGER_LITERAL:
     machine.pushInteger(Integer.parseInt(t.image)); break;
...
```

We can see that the parsed string (e.g. `"12"`) contained in `t.image` must be converted to its integer value before it is pushed into the stack. This is done by the static `parseInt` method, which is part of the `Integer` class.

Now, we need to do something similar with `FLOAT_LITERAL`: convert the string into an integer (our stack only handles integers) and push it on top of the stack machine.

We might be tempted to use `parseInt` again (after adding a new case option):

```
  case FLOAT_LITERAL:
     machine.pushInteger(Integer.parseInt(t.image)); break;
```

Unfortunately this won't work. Run javacc, re-compile and execute the program. If you enter a floating point number e.g. 3.2, you will get:

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "3.2"
at java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
at java.lang.Integer.parseInt(Integer.java:458)
at java.lang.Integer.parseInt(Integer.java:499)
at LexTest.TokenList(LexTest.java:66)
at LexTest.main(LexTest.java:8)
```

Method `parseInt` expects a string that conforms to integer syntax. If this is not the case, it throws a `NumberFormatException`. If there is a method (in the Java library) that parses integers then surely there must be something similar for floating points, or doubles. If you look hard enough you will find that `parseDouble`, from the `Double` class, does the jod. Then, the following should work:

```
  case FLOAT_LITERAL:
     machine.pushInteger(Double.parseDouble(t.image)); break;
```

Almost! If you compile the generated files you will get a compilation (type) error:

```
LexTest.java:66: pushInteger(int) in StackMachine cannot be applied to (double)
        machine.pushInteger(Double.parseDouble(t.image));
```

This is because `pushInteger` expects an `int` as argument, and `parseDouble` returns, logically, a double. This is solved by casting the result:

```
  case FLOAT_LITERAL:
     machine.pushInteger((int) Double.parseDouble(t.image)); break;
```

Done!! Below, we show the sequence of operations, and stack snapshots, performed during the evaluation of "20 3.2 5 + *".

```
STACK              NEXT STRING     OPERATION
---------------------------------------------------
[stack: empty]        "20"        pushInteger(20)
[stack: 20]           "3.2"       pushInteger(3)  // 3.2 is truncated by the cast
[stack: 20 : 3]       "5"         pushInteger(5)
[stack: 20 : 3 : 5]   "+"         plus()
[stack: 20 : 8]       "*"         multiply()
[stack: 160]
```

The result can be found on top of the stack.