Swinburne University Of Technology

Faculty of Science, Engineering and Technology

ASSIGNMENT COVER SHEET

Subject Code:	COS30023
Subject Title:	Languages in Software Development
Assignment number and title:	6, JavaCC – RPN & Stack Machine
Due date:	**October 6, 2014, 10:30, on paper**
Lecturer:	Dr. Markus Lumpe

Your name:____________________

Marker's comments:

Problem	Marks	Obtained
1	63	
Total	63	

Extension certification:

This assignment has been given an extension and is now due on __________________

Signature of Convener:________________________

Problem Set 6: JavaCC – RPN & Stack Machine

Start with the solution of Lab 7 – Reverse Polish Notation & Stack Machine.

The goal of this assignment is to implement a virtual RPN stack machine for the RPN language. In particular, we want to use the Visitor pattern and implement the RPN stack machine as a visitor for the abstract syntax tree nodes.

The visitor methods have to implement the semantics of the RPN instructions. There are two sets of visitor methods: one for the instructions and one for the arguments:

```
public interface PCodeVisitor
{
  public void visit( Add aInstruction );
  public void visit( Sub aInstruction );
  public void visit( Mul aInstruction );
  public void visit( Div aInstruction );
  public void visit( Dup aInstruction );
  public void visit( Print aInstruction );
  public void visit( Load aInstruction );
  public void visit( Store aInstruction );

  public Double visit( PCodeVariable aArgument );
  public Double visit( PCodeNumber aArgument );
}
```

The instruction methods do not return a result, whereas the argument methods return a `Double` value. Remember, a visitor has to traverse the object structure.

The visitor (i.e., the virtual RPN stack machine) should be implemented in the package `machine` to separate it from the rest of the front-end. The RPN stack machine requires two instance variables:

- `fStack` of type `Stack< Double >` to provide the value stack for the RPN machine, and
- `fMemory` of type `Hashtable< String, Double >` to emulate the RPN machine's memory.

The utility classes are defined in `java.util.*`.

In addition, the RPN machine should also implement two auxiliary methods:

- **`void`** `printStackTrace()` to print the content of the value stack to the system console, and
- **`void`** `printMemoryTrace()` to print the content of the RPN machine's memory to the system console.

Naturally, the RPN machine needs a constructor to properly initialize the instance variables.

You will need to update the classes developed in Lab 7.

Revised abstract syntax:

The abstract syntax for the PCode instructions has to be derived from:

- class Position:

```
package ast;

public class Position
{
  protected int fBeginLine;
  protected int fBeginColumn;
  protected int fEndLine;
  protected int fEndColumn;
}
```

- class PCode:

```
package ast;

import parser.Token;

public abstract class PCode extends Position
{
   public PCode( Token aInstruction )
   {
     fBeginLine = aInstruction.beginLine;
     fBeginColumn = aInstruction.beginColumn;
     fEndLine = aInstruction.endLine;
     fEndColumn = aInstruction.endColumn;
   }

   public abstract String toString();

   public abstract void accept( PCodeVisitor aVisitor );
}
```

  for PCode instructions

- class PCodeArgument:

```
package ast;

public abstract class PCodeArgument extends Position
{
  public abstract String toString();

  public abstract Double accept( PCodeVisitor aVisitor );
}
```

for IEEE floating point and variable arguments.

Revised PCodeParser skeleton:

```
PARSER_BEGIN(PCodeParser)

package parser;

import java.io.*;
import java.util.*;
import ast.*;
import machine.*;

public class PCodeParser
{
  public static void main( String[] args )
  {
    try
    {
      PCodeParser lParser = new PCodeParser( new FileInputStream( args[0] ) );

      ArrayList< PCode > lInstructions = lParser.Program();

      System.out.println( "PCode accepted:" );
      for ( PCode pc : lInstructions )
      {
        System.out.println( pc );
      }

      System.out.println( "Running program: " );
      PCodeMachine lMachine = new PCodeMachine();

      for ( PCode inst : lInstructions )
      {
        inst.accept( lMachine );
      }

      lMachine.printStackTrace();
      lMachine.printMemoryTrace();
    }
    catch (ParseException e)
    {
      System.out.println( "Syntax Error : \n"+ e.toString() );
    }
    catch( java.io.FileNotFoundException e )
    {
      System.err.println( e.toString() );
    }
  }
}

PARSER_END(PCodeParser)
```

Sample program and output:

```
load  20
dup
store $a
load  4
load  2
mul
dup
print "4 * 2 = "
load  1
add
dup
print "4 * 2 + 1 = "
store $x
```

produces

```
PCode accepted:
load  20.0
dup
store $a
load  4.0
load  2.0
mul
dup
print "4 * 2 = "
load  1.0
add
dup
print "4 * 2 + 1 = "
store $x
Running program:
4 * 2 = 8.0
4 * 2 + 1 = 9.0
Stack:
1:    20.0
Memory:
$x:   9.0
$a:   20.0
```

Submission deadline: Monday, October 6, 2014, 10:30.

Submission procedure: on paper.