CSC 402-01 Assignment #4

```
Original Due: 3:00 PM, Monday, April 11
    Extended: 3:00 PM, Monday, April 18
```

You must complete this assignment by yourself. You cannot work with anyone else in the class or with someone outside of the class. You are not allowed to copy solutions from the world wide web. The code you write must be your own.

Provided Files:

- Main.java – a shell file that contains the main method.
- Prog.java – a shell file.
- Stm.java – a shell file.
- Visitor.java – a shell file.

Description: In order to check that a program is parsed correctly into a parse tree, we build a so-called "pretty-printer." The idea behind pretty-printing is to take a parse tree and write it back to the screen as source code again. This sounds meaningless but it is actually a useful way to see that one has indeed parsed the source file correctly into a parse tree.

For this assignment, your task is to make a Visitor design pattern for all `prettyprint` methods in Stm.java following each of the **steps #1-5** below. Your refactored code (after each step) must produce the original output:

```
a := 5 + 3 ; b := ( print( a , a - 1 ) , 10 * a ) ; print( b )
```

Step #1

1. Nothing.

Step #2

1. Retrofit a Singleton design pattern into class `Visitor` (in Visitor.java).
2. Make sure that your refactored code produces the original output above.
3. Create a zip file (**two.zip**) containing your current Main.java, Prog.java, Stm.java and Visitor.java

Step #3

1. Add a `Visitor`-type parameter to all `prettyprint` methods and use the `Visitor`-type singleton instance as a default value.
2. Make sure that your refactored code produces the original output above.
3. Create a zip file (**three.zip**) containing your current Main.java, Prog.java, Stm.java and Visitor.java

Step #4

1. Perform the Move-Instance-Method-with-Leaving-a-Delegate-Behind refactoring on all concrete (not abstract/interface) `prettyprint` methods.
2. Make sure that your refactored code produces the original output above.
3. Create a zip file (**four.zip**) containing your current Main.java, Prog.java, Stm.java and Visitor.java

Step #5

1. Rename all delegate methods (created by **Step #4**) to "`accept`" without breaking run-time polymorphism.
2. Rename all `prettyprint` methods in class `Visitor` to "`visit`"
3. Make sure that your refactored code produces the original output above.
4. Create a zip file (**five.zip**) containing your current Main.java, Prog.java, Stm.java and Visitor.java

You must use the shell files for this assignment.

Submission: your files named **two.zip**, **three.zip**, **four.zip**, **five.zip**

General Programming Assignment Requirements:

- Classes must be in the default (no package statement) unless otherwise specified. You will lose all points if you put a package statement in your program.
- If your program does not compile or does not run, you will lose all points.
- If you submit the wrong file, you will lose all points.
- You must fill in the header for every file you submit. Otherwise, you will lose all points.

Checklist: Did you remember to:

- work on the programming assignment individually?
- fill in the header in your Main.java, Prog.java, Stm.java and Visitor.java?
- ensure your program does not suffer a compile error or runtime error?
- ensure your program creates the correct output and that it matches the expected output exactly?
- properly indent your source code so that your indenting is readable and consistent?
- use good names for variables to make your program easy to understand?
- turn in your Java source code in files named **two.zip**, **three.zip**, **four.zip**, **five.zip** through D2L?