Project #3: 
Straight-Line Interpreter*

1 Objectives

There are several objectives to this project. You will gain experience implementing a non-trivial class hierarchy involving multiple levels of interface inheritance. This will reinforce your understanding of the interface and inheritance mechanisms in Java and generally improve your object-oriented design skills. You will additionally reinforce your understanding of the Singleton pattern. It is also likely that the project will improve your ability to debug Java programs.

2 Overview

For this project, you will implement the execution core of an interpreter for a simple programming language. You will not be responsible for implementing the scanner, the parser, or the generator. Instead, you will provide a class library that could be used as the compilation target of a complete interpreter implementation. Or stated another way, you will provide the classes that provide the interpreter’s execution logic.

3 Requirements

Consider the language grammar shown in Listing 1. The grammar describes a simple straight-line programming language; there are no loops, conditionals, or functions. Programs written in this language consist of assignment statements, arithmetic expressions, and print statements — nothing else. All expressions, including variable references, are of integer type. Variables are declared implicitly; a variable is declared at the point it is first used (i.e., read or written).

Your task is to implement the execution core for this language using the grammar as the basis for structuring your design. Specifically, you are required to implement a Java class library consistent with the UML class diagram shown in Figure 1. You may choose to add additional methods and classes, but your design must at least include those specified in the diagram. We will discuss the diagram in more detail in class.

Your library is intended to serve as the compilation target for a complete interpreter implementation. Consider the simple program shown in Listing 2. The output of this program should be 2306. Trace this code to be sure that you understand the semantics of the language.

To execute this program, the interpreter would generate Java source code that uses your library implementation. Specifically, the interpreter would generate the source fragment shown in Listing 3. (Note that the symbol table required to store variable values is not shown in the listing.) The output of this program should be 2306.

*This project is based on an exercise described in Andrew W. Appel’s excellent textbook, Modern Compiler Implementation in Java.
4 Submission Instructions

This project is due in class on April 14th. Absolutely no late assignments will be accepted.

When your project is complete, archive your source materials and use the `handin` command to submit your work. The command for this project is: `handin.215.your_section 3 !filename!`. You may use any standard archive format you like.

5 Grading

Your project will be graded based on your adherence to the specified requirements and the programming style guidelines discussed in class.

This is an intermediate course in software development. Your source materials should be properly documented. Your source must compile. Your application must not crash. A violation of any of these requirements will result in an automatic zero. **Test your application thoroughly.**

6 Collaboration

You must work independently on this assignment. You must *not* discuss the problem or the solution with *any* classmates. **Any form of collaboration will be considered academic misconduct.**

---

1 \( \langle Stm \rangle \rightarrow \langle Stm \rangle ; \langle Stm \rangle \) \hspace{2cm} (CompoundStm)
2 \( \langle Stm \rangle \rightarrow \text{id} := \langle Exp \rangle \) \hspace{2cm} (AssignStm)
3 \( \langle Stm \rangle \rightarrow \text{print}(\langle Exp \rangle) \) \hspace{2cm} (PrintStm)
4 \( \langle Exp \rangle \rightarrow \text{id} \) \hspace{2cm} (IdExp)
5 \( \langle Exp \rangle \rightarrow \text{num} \) \hspace{2cm} (NumExp)
6 \( \langle Exp \rangle \rightarrow \langle Exp \rangle \; (+|-|*|/\; \langle Exp \rangle \) \hspace{2cm} (OpExp)

Listing 1: Language Grammar

1 a := 2;
2 b := 3;
3 d := a * b + c / 1;
4 print(a);
5 print(b);
6 print(c);
7 print(d)

Listing 2: Program Example
Figure 1: UML Class Diagram for Interpreter Core
Stm stm = new CompoundStm(
    new AssignStm("a", new NumExp(2)),
    new CompoundStm(
        new AssignStm("b", new NumExp(3)),
        new CompoundStm(
            new AssignStm("d",
                new OpExp(
                    new OpExp(new IdExp("a"), Op.TIMES, new IdExp("b")),
                    Op.PLUS,
                    new OpExp(new IdExp("c"), Op.DIV, new NumExp(1)))))
    )
);  
stm.execute();

Listing 3: Generated Java Code for Program Example