Programming Assignment 1 (P1)

Edit: grading rubric

This assignment has two parts. Part 1 is due at 11 PM on Sep 12 2016. Part 1 will not be accepted past this time (the regular late policy does not apply to Part 1). Part 2 is due at 11 PM on Sep 15 2016. For Part 2, see the late policy for information on late submissions. Make sure to follow the submission instructions.

Each student must do this assignment alone. You may work in pairs from Programming Assignment 2 (not Part 2) onwards. You may discuss this assignment with the instructor, TA, and other students, but you may not share code.

Overview

In this assignment, you will write several Java classes to be used later in the semester to represent a symbol table. This is a simple assignment to get you up to speed with your computing environment, Java, and our programming and testing conventions.

Make sure you read through everything carefully. The assignment appears long but it is less work than it might initially seem. If it seems difficult, ask for help; you may be misunderstanding something.

Specifications

For this assignment you will implement four Java classes: SymTable, Sym, DuplicateSymException, and EmptySymTableException. You will also write a program called P1.java to test your implementations.

The SymTable class will be used by the compiler you write later in the semester to represent a symbol table: a data structure that stores the identifiers declared in the program being compiled (e.g., function and variable names) and information about each identifier (e.g., its type, where it will be stored at runtime). The symbol table will be implemented as a List of HashMaps. Eventually, each HashMap will store the identifiers declared in one scope in the program being compiled.

The HashMap keys will be Strings (the declared identifier names) and the associated information will be Syms (you will also implement the Sym class). For now, the only information in a Sym will be the type of the identifier, represented using a String (e.g., “int”, “double”, etc.).

The DuplicateSymException and EmptySymTableException classes will define exceptions that can be thrown by methods of the SymTable class.

In addition to defining the four classes, you will write a main program to test your implementation. You will be graded on the correctness of your Sym and SymTable classes, on how thoroughly you test the classes that you implement, on the efficiency of your code, and on your programming style.

The Sym Class

The Sym class must be in a file named Sym.java. You must implement the following Sym constructor and public methods (and no other public or protected methods):

Sym(String type)  This is the constructor; it should initialize the Sym to have the given type.
String getType()  Return this Sym's type.
String toString()  
Return this Sym's type. (This method will be changed later in a future project when more information is stored in a Sym.)

The SymTable Class

The SymTable class must be in a file named SymTable.java. It must be implemented using a List of HashMaps. (Think about the operations that will be done on a SymTable to decide whether to use an ArrayList or a LinkedList.) The HashMaps must map a String to a Sym. This means that the SymTable class will have a (private) field of type List<HashMap<String,Sym>>.

List and HashMap are defined in the java.util package. This means that you will need to have the line

```java
import java.util.*;
```
at the top of SymTable.java.

You must implement the following SymTable constructor and public methods (and no other public or protected methods):

```java
SymTable()  
This is the constructor; it should initialize the SymTable's List field to contain a single, empty HashMap.

void addDecl(String name, Sym sym) throws DuplicateSymException, EmptySymTableException  
If this SymTable's list is empty, throw an EmptySymTableException. If either name or sym (or both) is null, throw a NullPointerException. If the first HashMap in the list already contains the given name as a key, throw a DuplicateSymException. Otherwise, add the given name and sym to the first HashMap in the list.

void addScope()  
Add a new, empty HashMap to the front of the list.

Sym lookupLocal(String name)  
If this SymTable's list is empty, throw an EmptySymTableException. Otherwise, if the first HashMap in the list contains name as a key, return the associated Sym; otherwise, return null.

Sym lookupGlobal(String name)  
If this SymTable's list is empty, throw an EmptySymTableException. If any HashMap in the list contains name as a key, return the first associated Sym (i.e., the one from the HashMap that is closest to the front of the list); otherwise, return null.

void removeScope() throws EmptySymTableException  
If this SymTable's list is empty, throw an EmptySymTableException; otherwise, remove the HashMap from the front of the list. To clarify, throw an exception only if before attempting to remove, the list is empty (i.e. there are no HashMaps to remove).

void print()  
This method is for debugging. First, print “
Sym Table
”. Then, for each HashMap M in the list, print M.toString() followed by a newline. Finally, print one more newline. All output should go to System.out.
```

The DuplicateSymException and EmptySymTableException Classes

These two classes (which must be in files named DuplicateSymException.java and EmptySymTableException.java) will simply define the two checked exceptions that can be thrown by the SymTable class. Each exception must be able to be created using a constructor that takes no arguments.

To define a checked exception named xxx, you can use code like this:

```java
public class XXX extends Exception {
}
```

Note that the class has an empty body (it will have a no-argument constructor by default).
The main program

To test your SymTable implementation, you will write a main program in a file named P1.java. The program must not expect any command-line arguments or user input. It can read from one or more files; if you set it up to do that, be sure to hand in the file(s) along with P1.java.

Be sure that your P1.java tests all of the Sym and SymTable operations and all situations under which exceptions are thrown. Also think about testing both “boundary” and “non-boundary” cases.

It is up to you how your program works. A suggested approach is to write your program so that output is only produced if one of the methods that it is testing does not work as expected (e.g., if the lookupLocal method of the SymTable class returns null when you expect it to return a non-null value). This will make it much easier to determine whether your test succeeds or fails. The one exception to this approach is that P1.java will need to test the print method of the SymTable class and that will cause output to be produced.

To help you understand better the kind of code you would write using this suggested approach, look at TestList.java. This file contains a main program designed to test a (fictional) List class whose methods are documented in TestList.java. You are being asked to write something similar (in a file called P1.java) to test the Sym and SymTable classes. You should be able to write P1.java before you write the classes that it's designed to test.

Test Code

After the Part 1 deadline, download our P1.java file and test it against the expected output. Make sure that your actual output matches this.

On a Linux machine you can see whether two files match by using the diff utility. For example, typing diff file1 file2 compares the two files file1 and file2. Typing diff -b -B file1 file2 does the same comparison, but ignores differences in whitespace.

If you send the output of P1.java to a file, you can use diff to make sure that it matches the expected output. To send the output of P1.java to a file named out.txt (on a Linux machine) type java P1 > out.txt.

Handing In

Deadlines are at the top of the page. See instructions for submitting assignments.

By the Part 1 deadline, submit your P1.java file (and the files that it reads, if any).
By the Part 2 deadline, submit the rest of your .java files. This should include your Sym.java, SymTable.java, DuplicateSymException.java, and EmptySymTable.java.

You may work in the environment of your choice, but be aware that your submitted code must run on the department lab Linux machines.

Do not turn in any .class files and do not create any subdirectories in your submission. If you accidentally turn in (or create) extra files or subdirectories, make a new submission that does not include them.

Remember, your P1.java is worth 15% of the grade for this assignment and will not be accepted past the deadline.

Grading Criteria

For this program, extra emphasis will be placed on style. In particular,
Every class, method, and field must have a comment that describes its purpose. Comments should also be used to explain anything that would not be obvious to an experienced Java programmer who has read this assignment.

Identifiers must conform to standard Java conventions. `UPPER_CASE` with underscores for named constants, `CamelCase` starting with a capital letter for classes, and `camelCase` starting with a lower-case letter for other identifiers. Names should help a reader to understand the code.

Indentation must be consistent and clear. Use either one tab character or four spaces for each level of indentation. Do not mix spaces and tabs for indentation; either always use tabs or never use them.

Avoid lines that are longer than 80 characters (including indentation).

Each field or method must be declared public, protected, or private. If you have good reason to give a field or method “package” (default) access – which is highly unlikely – you must include a comment explaining why.

The goal is to make your code readable to an experienced Java programmer who is used to the conventions. The goal is not to develop your own personal style, even if it’s “better” than the standard. For more advice on Java programming style, see [Code Conventions for the Java Programming Language](http://www.oracle.com/technetwork/java/javase/codestyle-132301.html). See also the [style](http://www.oracle.com/technetwork/java/javase/codestyle-132301.html) and [commenting](http://www.oracle.com/technetwork/java/javase/codestyle-132301.html) standards used in CS 302 and CS 367.

Also be very sure that you use the specified file names (being careful about the upper- and lower-case letters in those names). And be sure that the output that is produced when we run our `P1.java` using your implementations of the `Sym`, `SymTable`, `DuplicateSymException`, and `EmptySymTableException` classes matches the expected output that we provide. We will test that output by automatically comparing it to the expected output and you will lose points for even minor differences.