

CS 541 — Fall 2020

Programming Assignment 1 Symbol Table

Introduction

You are to write a set of Java classes that implement a **block-structured symbol table**. You must also write a test driver and create test data that thoroughly test your symbol table implementation.

You should implement the following six Java classes: `Symb`, `SymbolTable`, `TestSymb`, `DuplicateException`, `EmptySTException` and `P1`.

- Subclasses of the `Symb` class will eventually be used in your compiler to store information about each identifier that appears in a program (such as the variable and function names). The only information stored in a `Symb` is the name of the identifier (a `String`); more information appears in subclasses of `Symb`. Java's subclassing rules allow any subclass of `Symb` to be used where a `Symb` object is expected. The symbol table methods we develop in this project accept all subclasses of `Symb`. `TestSymb` is a subclass of `Symb` that contains a single `String` field. It is used to test the operation of the `SymbolTable` class.
- The `SymbolTable` class implements a block-structured symbol table. It can be built using a `List` of `Java HashMap` objects, one for each open scope.
- The `DuplicateException` and `EmptySTException` classes are exceptions that can be thrown by methods of the `SymbolTable` class.
- Class `P1` (for `Program1`) implements an interactive test driver used to test your `SymbolTable` class.

Class Specifications

You don't have to follow these specifications precisely, but you should have a good reason to diverge from them.

class Symb

<code>Symb(String s)</code>	The class constructor; initialize <code>Symb</code> to have name <code>s</code> .
<code>String name()</code>	Return the name of this <code>Symb</code> object.
<code>String toString()</code>	Return a string representation of this <code>Symb</code> object.

class TestSym

TestSym(String s, String i)	The class constructor; initialize TestSym to have name s and value i.
String value()	Return the value of this TestSym object.
String toString()	Return a string representation of this TestSym object.

class SymbolTable

SymbolTable()	The class constructor; initialize SymbolTable to contain no scopes.
void openScope()	Add a new, initially empty scope to the list of scopes contained in this SymbolTable.
void closeScope()	If the list of scopes in this SymbolTable is empty, throw an EmptySTException. Otherwise, remove the current (front) scope from the list of scopes contained in this SymbolTable.
void insert(Symb s)	If the list of scopes in this SymbolTable is empty, throw an EmptySTException. If the current (first) scope contains a Symb whose name is the same as that of s (including case), throw a DuplicateException. Otherwise, insert s into the current (front) scope of this SymbolTable.
Symb localLookup(String n)	If the list of scopes in this SymbolTable is empty, return null. If the current (first) scope contains a Symb whose name is n (including case), return that Symb. Otherwise, return null.
Symb globalLookup(String n)	If any scope contains a Symb whose name is n (including case), return the first matching Symb found (in the scope nearest to the front of the scope list). Otherwise, return null.
void dump(PrintStream p)	This method is for debugging. The contents of this SymbolTable are written to Printstream p (System.out is a Printstream).
String toString()	Return a string representation of this SymbolTable.

class P1

void main(String[] args)	The test driver used to test your SymbolTable implementation.
--------------------------	---

class DuplicateException and **class** EmptySTException

These two classes, which extend `java.lang.Exception`, are empty. They are used to signal duplicate-insertion and empty symbol-table errors.

Getting Started

The MultiLab supports the `javac` Java compiler, OpenJDK version 11, which compiles a recent version of Java, including generic classes. You may also use other quality Java compilers or integrated development environments. (*Eclipse* is free and highly regarded).

You can find partial implementations of the required classes, along with a Makefile and sample test data for all five projects, in `~raphael/courses/cs541/public` (or use the tarball <http://www.cs.uky.edu/~raphael/courses/-CS541/public.tar.gz>). You will certainly need to edit and extend the `SymbolTable` and `P1` classes. You may leave the other classes (which are quite simple) mostly as they are. The Makefile allows you to easily compile and test your solution to this assignment. You should use `make` to speed and simplify program development. The command

```
make
```

recompiles classes as needed after any changes you make. The command

```
make test
```

recompiles as necessary and then tests your solution by calling `P1.main` with the commands in `testInput`. (You should edit this file to more thoroughly test your implementation). This command invokes the compiler with warnings enabled; your code should generate no warnings. The command

```
make clean
```

removes all class files created by the compiler. All class files reside in the `classes` subdirectory to avoid cluttering your top-level project directory. The command

```
make style
```

runs a style checker on your code and suggests improvements. Take them seriously!

Use the standard Java utility class `java.util.HashMap` in implementing your block-structured symbol table. `HashMap<K, V>` defines a hash table in which all keys have class `K` and all table entries have class `V`. Explicit casting is not required. You might also find `java.util.Scanner` useful. You can find details of all Java library routines at <http://download.oracle.com/javase/8/docs/api/>.

The Test Driver

You'll need to create an interactive test driver, in method `main()` of class `P1`, to test the operation of your block structured symbol table. Your test driver should accept the following commands from standard input (case-insensitive). Initially no scope is open.

Command	Operation
Open	Open a new scope
Close	Close the top (innermost) scope.
Dump	Dump the contents of symbol table.
Insert	Read two strings (each line-end delimited). The first is the key, the second is the value. Insert the (key, value) pair into the innermost scope. Case is significant.
Lookup	Read a string and look it up as a key in the top (currently open) scope. Case is significant. Print the value in the symbol table entry found or display an error message if it is not found.
Global	Read a string and look it up as a key in the nearest scope that contains an entry. Case is significant. Print the integer in the symbol table entry found or display an error message if it is not found.
Quit	Exit the test driver.

One-letter abbreviations of the commands should be allowed.

The following illustrates the operation of the test driver (text entered by the user is printed in **bold face**). This example is only meant to illustrate our testing interface; it does not by itself represent an exhaustive test set. To facilitate automatic grading, please make your wording of responses to commands similar to that shown below.

open

New scope opened.

insert

Enter symbol: **kentucky**

Enter associated value: **1848**

(kentucky:1848) entered into symbol table.

insert

Enter symbol: **Florida**

Enter associated value: **wet**

(florida:wet) entered into symbol table.

lookup

Enter symbol: **kentucky**

(kentucky:1848) found in top scope.

lookup

Enter symbol: **Florida**

(Florida:wet) found in top scope.

lookup

Enter symbol: **Kentucky**

Kentucky not found in top scope.

insert

Enter symbol: **kentucky**

Enter associated value: **bluegrass**

kentucky already entered into top scope.

open

New scope opened.

insert

Enter symbol: **kentucky**

Enter associated value: **palisades**

(kentucky:palisades) entered into symbol table.

lookup

Enter symbol: **kentucky**

(kentucky:palisades) found in top scope.

dump

Contents of symbol table:

{(kentucky:palisades)}

{(Florida:wet), kentucky:1848)}

lookup

Enter symbol: **Florida**

Florida not found in top scope.

global

Enter symbol: **Florida**

(Florida:wet) found in symbol table.

close

Top scope closed.

lookup

Enter symbol: **kentucky**

(kentucky:1848) found in top scope.

lookup

Enter symbol: **Florida**

(Florida:wet) found in top scope.

close

Top scope closed.

lookup

Enter symbol: **kentucky**

Kentucky not found in top scope.

quit

Testing done

What To Hand In

Submit your project electronically as a single file (tar.gz, rar, zip, or equivalent) via Canvas. Please run `make clean` first to remove all `class` files. Include your version of `testInput` that comprises the tests you used to verify the operation of your symbol table routines. Include `testOutput`, which is the output generated by your program in response to your `testInput` file. Include a `README` file to hold external documentation. We'll run your program on our own test data.

We will grade your program on the basis of the **completeness of your testing** (as shown in the `testInput` and `testOutput` files) as well as the **error-free compilation** and **correct operation** of your symbol table routines.

The **quality of your documentation** is also important. Make sure that you provide both external documentation (in the `README` file) and internal documentation (in the source files). Check your spelling and grammar. It should be easy to understand the organization and structure of your program. We may exact significant penalties if we find your program poorly documented or difficult to understand.