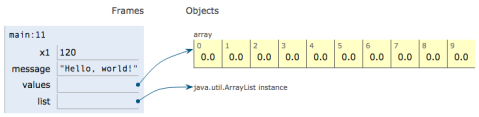
Intermediate Milestone 2

Tao Wang
CS 6460: Educational Technology
Spring 2016

Week #	***Due Date***	***Milestone or Task***
9	3/13/16	Work on incorporating conditional and loop structures into the trace.
10	3/20/16	*Intermediate Milestone 2* Deliverables: (1) command-line program that traces execution of the main method of a Java class. Trace output still formatted as text to standard out. (2) Documentation containing examples.

Overview



This is the second milestone of a project to develop a Java interpreter that can step through code segments and produce line-by-line code traces and visualizations. The motivation for this work is to provide a tool that can assist new computer science students in doing code traces, debugging, and problem solving. The tool can also be used by instructors in preparing example code walk throughs for lectures or tutors and teaching assistants when demonstrating code for students in one-on-one sessions. The main inspiration comes from Java Visualizer (http://cscircles.cemc.uwaterloo.ca/java_visualize/#).

Progress

In the past two weeks, I've been working on expanding the capabilities of the Java interpreter that will become the backbone of this project. In short, `Trace` is a program that uses BeanShell (http://www.beanshell.org/) to evaluate Java code line by line. The Java code need not be a part of a class and can be executed on its own. The goal is to create a sort of Java "playground" where short code segments can be tested and debugged.

In addition to being able to interpret Java at runtime, `Trace` outputs "snapshots" of variables and objects after each line is evaluated as a debugger would. Currently, `Trace` supports most of the basic programming structures in Java. An example code segment and its output are given at the end of this document.

Format

The interpreter outputs snapshots in JSON format after the evaluation of each Java statement. The format of the output looks like this:

{
    "code" : *the line that was just evaluated*,
    "pc" : *the 'program counter' - the line number where the code appears*,
    "step" : *an increasing counter of the number of evaluations*,
    "state" : { *a list of all variables that are active within the current scope and their values* }
}

Code

Repository: https://github.com/tao-wang/JavaCodeTrace

Usage: `java Trace filepath`

`filepath` refers to a text file that contains Java statements. The program does not accept compiled bytecode and the file does not have to define a Java class with a main method. Partial Java programs and code segments can be used (and required at this point, actually).

No bytecode is produced during runtime. BeanShell allows for the runtime evaluation of Java statements and provides access to most standard library classes. This does have some limitations. For instance, BeanShell is based off an older JRE (1.3, I think).

Remaining Goals

1. Create program to read the output of `Trace` and produce visualizations.
2. Create graphical user interface for that program.
3. *Extend the interpreter to keep track of stack vs. heap memory.*

Now that `Trace` is effectively working, the next step is to create a visual interface to view all the code snapshots. This will give users the ability to walk through code to see how the contents of memory change after each Java statement.

The next couple of weeks will be spent designing a visual language in order to display variables, objects, and references. That will make up most of the work shown for milestone 3. Milestone 4 will be the graphical user interface for the program itself.

Requirement #3 is an optional "would-be-nice" requirement.

Example: Bubble Sort

Here's a small example of bubble sort applied to a four-element array. The whole evaluation process takes 46 steps and demonstrates how the trace handles loops and conditional statements.

```
1     int[] values = {4, 1, 2, 3};
2     for (int i = values.length; i > 0; i--)
3     {
4           for (int j = 0; j < i - 1; j++)
5           {
6                 if (values[j] > values[j+1])
7                 {
8                       int temp = values[j+1];
9                       values[j+1] = values[j];
10                      values[j] = temp;
11                }
12          }
13    }
```

```
{
       "code" : "int[] values = {4, 1, 2, 3};",
       "pc" : 1,
       "step" : 1,
       "state" : {
              "values" : {4, 1, 2, 3},
       }
},
{
       "code" : "int i = values.length",
       "pc" : 2,
       "step" : 2,
       "state" : {
              "values" : {4, 1, 2, 3},
              "i" : 4,
       }
},
{
       "code" : "i > 0 -> true",
       "pc" : 2,
       "step" : 3,
       "state" : {
              "values" : {4, 1, 2, 3},
              "i" : 4,
       }
},
{
       "code" : "int j = 0",
       "pc" : 4,
       "step" : 4,
       "state" : {
              "values" : {4, 1, 2, 3},
              "i" : 4,
              "j" : 0,
```

```
        }
},
{
        "code" : "j < i - 1 -> true",
        "pc" : 4,
        "step" : 5,
        "state" : {
                "values" : {4, 1, 2, 3},
                "i" : 4,
                "j" : 0,
        }
},
{
        "code" : "values[j] > values[j+1] -> true",
        "pc" : 6,
        "step" : 6,
        "state" : {
                "values" : {4, 1, 2, 3},
                "i" : 4,
                "j" : 0,
        }
},
{
        "code" : "int temp = values[j+1];",
        "pc" : 8,
        "step" : 7,
        "state" : {
                "values" : {4, 1, 2, 3},
                "i" : 4,
                "j" : 0,
                "temp" : 1,
        }
},
{
        "code" : "values[j+1] = values[j];",
        "pc" : 9,
        "step" : 8,
        "state" : {
                "values" : {4, 4, 2, 3},
                "i" : 4,
                "j" : 0,
                "temp" : 1,
        }
},
{
        "code" : "values[j] = temp;",
        "pc" : 10,
        "step" : 9,
        "state" : {
                "values" : {1, 4, 2, 3},
                "i" : 4,
                "j" : 0,
                "temp" : 1,
        }
},
{
        "code" : "j++",
        "pc" : 4,
        "step" : 10,
        "state" : {
```

```
                "values" : {1, 4, 2, 3},
                "i" : 4,
                "j" : 1,
                "temp" : 1,
        }
},
{
        "code" : "j < i - 1 -> true",
        "pc" : 4,
        "step" : 11,
        "state" : {
                "values" : {1, 4, 2, 3},
                "i" : 4,
                "j" : 1,
                "temp" : 1,
        }
},
{
        "code" : "values[j] > values[j+1] -> true",
        "pc" : 6,
        "step" : 12,
        "state" : {
                "values" : {1, 4, 2, 3},
                "i" : 4,
                "j" : 1,
                "temp" : 1,
        }
},
{
        "code" : "int temp = values[j+1];",
        "pc" : 8,
        "step" : 13,
        "state" : {
                "values" : {1, 4, 2, 3},
                "i" : 4,
                "j" : 1,
                "temp" : 2,
                "temp" : 2,
        }
},
{
        "code" : "values[j+1] = values[j];",
        "pc" : 9,
        "step" : 14,
        "state" : {
                "values" : {1, 4, 4, 3},
                "i" : 4,
                "j" : 1,
                "temp" : 2,
                "temp" : 2,
        }
},
{
        "code" : "values[j] = temp;",
        "pc" : 10,
        "step" : 15,
        "state" : {
                "values" : {1, 2, 4, 3},
                "i" : 4,
                "j" : 1,
```

```
                "temp" : 2,
                "temp" : 2,
        }
},
{
        "code" : "j++",
        "pc" : 4,
        "step" : 16,
        "state" : {
                "values" : {1, 2, 4, 3},
                "i" : 4,
                "j" : 2,
                "temp" : 2,
                "temp" : 2,
        }
},
{
        "code" : "j < i - 1 -> true",
        "pc" : 4,
        "step" : 17,
        "state" : {
                "values" : {1, 2, 4, 3},
                "i" : 4,
                "j" : 2,
                "temp" : 2,
                "temp" : 2,
        }
},
{
        "code" : "values[j] > values[j+1] -> true",
        "pc" : 6,
        "step" : 18,
        "state" : {
                "values" : {1, 2, 4, 3},
                "i" : 4,
                "j" : 2,
                "temp" : 2,
                "temp" : 2,
        }
},
{
        "code" : "int temp = values[j+1];",
        "pc" : 8,
        "step" : 19,
        "state" : {
                "values" : {1, 2, 4, 3},
                "i" : 4,
                "j" : 2,
                "temp" : 3,
                "temp" : 3,
                "temp" : 3,
        }
},
{
        "code" : "values[j+1] = values[j];",
        "pc" : 9,
        "step" : 20,
        "state" : {
                "values" : {1, 2, 4, 4},
                "i" : 4,
```

```
                "j" : 2,
                "temp" : 3,
                "temp" : 3,
                "temp" : 3,
        }
},
{
        "code" : "values[j] = temp;",
        "pc" : 10,
        "step" : 21,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 4,
                "j" : 2,
                "temp" : 3,
                "temp" : 3,
                "temp" : 3,
        }
},
{
        "code" : "j++",
        "pc" : 4,
        "step" : 22,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 4,
                "j" : 3,
                "temp" : 3,
                "temp" : 3,
                "temp" : 3,
        }
},
{
        "code" : "j < i - 1 -> false",
        "pc" : 4,
        "step" : 23,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 4,
                "j" : 3,
                "temp" : 3,
                "temp" : 3,
                "temp" : 3,
        }
},
{
        "code" : "i--",
        "pc" : 2,
        "step" : 24,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 3,
        }
},
{
        "code" : "i > 0 -> true",
        "pc" : 2,
        "step" : 25,
        "state" : {
                "values" : {1, 2, 3, 4},
```

```
                "i" : 3,
        }
},
{
        "code" : "int j = 0",
        "pc" : 4,
        "step" : 26,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 3,
                "j" : 0,
        }
},
{
        "code" : "j < i - 1 -> true",
        "pc" : 4,
        "step" : 27,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 3,
                "j" : 0,
        }
},
{
        "code" : "values[j] > values[j+1] -> false",
        "pc" : 6,
        "step" : 28,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 3,
                "j" : 0,
        }
},
{
        "code" : "j++",
        "pc" : 4,
        "step" : 29,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 3,
                "j" : 1,
        }
},
{
        "code" : "j < i - 1 -> true",
        "pc" : 4,
        "step" : 30,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 3,
                "j" : 1,
        }
},
{
        "code" : "values[j] > values[j+1] -> false",
        "pc" : 6,
        "step" : 31,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 3,
```

```
                "j" : 1,
        }
},
{
        "code" : "j++",
        "pc" : 4,
        "step" : 32,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 3,
                "j" : 2,
        }
},
{
        "code" : "j < i - 1 -> false",
        "pc" : 4,
        "step" : 33,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 3,
                "j" : 2,
        }
},
{
        "code" : "i--",
        "pc" : 2,
        "step" : 34,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 2,
        }
},
{
        "code" : "i > 0 -> true",
        "pc" : 2,
        "step" : 35,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 2,
        }
},
{
        "code" : "int j = 0",
        "pc" : 4,
        "step" : 36,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 2,
                "j" : 0,
        }
},
{
        "code" : "j < i - 1 -> true",
        "pc" : 4,
        "step" : 37,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 2,
                "j" : 0,
        }
```

```
},
{
        "code" : "values[j] > values[j+1] -> false",
        "pc" : 6,
        "step" : 38,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 2,
                "j" : 0,
        }
},
{
        "code" : "j++",
        "pc" : 4,
        "step" : 39,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 2,
                "j" : 1,
        }
},
{
        "code" : "j < i - 1 -> false",
        "pc" : 4,
        "step" : 40,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 2,
                "j" : 1,
        }
},
{
        "code" : "i--",
        "pc" : 2,
        "step" : 41,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 1,
        }
},
{
        "code" : "i > 0 -> true",
        "pc" : 2,
        "step" : 42,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 1,
        }
},
{
        "code" : "int j = 0",
        "pc" : 4,
        "step" : 43,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 1,
                "j" : 0,
        }
},
{
```

```
        "code" : "j < i - 1 -> false",
        "pc" : 4,
        "step" : 44,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 1,
                "j" : 0,
        }
},
{
        "code" : "i--",
        "pc" : 2,
        "step" : 45,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 0,
        }
},
{
        "code" : "i > 0 -> false",
        "pc" : 2,
        "step" : 46,
        "state" : {
                "values" : {1, 2, 3, 4},
                "i" : 0,
        }
},
```