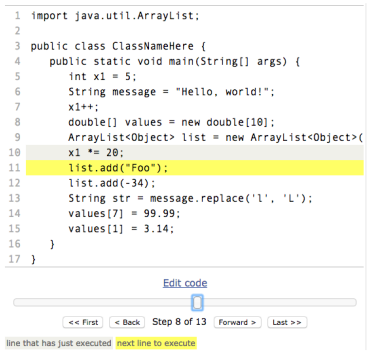
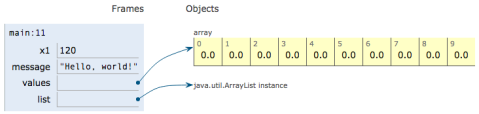
Intermediate Milestone 1

Tao Wang
CS 6460: Educational Technology
Spring 2016

Week #	***Due Date***	***Milestone or Task***
7	2/28/16	Revised Project Proposal.
8	3/6/16	*Intermediate Milestone 1* Deliverables: (1) command-line program that traces execution of the main method of a Java class. Trace output will be formatted as text to standard out. No support for conditional or loop structures at this point -- just simple assignment statements and void method calls. *Format output using JSON. This would allow the use of a pre-existing API for parsing later on.* (2) Documentation containing examples.

Overview





This is the first milestone of a project to develop a Java interpreter that can step through full programs or 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

This week I implemented a simple command line program that produces a trace for a linear Java code segment. This current program parses and executes Java statements that:

- declare and assign variables
- call methods

For each Java statement that is executed, the program produces output about the current program state: a JSON-formatted list of the names of all variables and references declared so

far and their current values. Two examples are provided at the end of this document. The program uses BeanShell (http://www.beanshell.org/) to evaluate Java statements.

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

Example 1

```
int x = 5;
int y = 24;
x = 6;
y++;
int z = y / x;
```

```
"trace" : {
	"code" : "int x = 5;",
	"step" : 1,
	"state" : {
		"x" : 5,
	}
},
{
	"code" : "int y = 24;",
	"step" : 2,
	"state" : {
		"x" : 5,
		"y" : 24,
	}
},
{
	"code" : "x = 6;",
```

```
        "step" : 3,
        "state" : {
                "x" : 6,
                "y" : 24,
        }
},
{
        "code" : "y++;",
        "step" : 4,
        "state" : {
                "x" : 6,
                "y" : 25,
        }
},
{
        "code" : "int z = y / x;",
        "step" : 5,
        "state" : {
                "x" : 6,
                "y" : 25,
                "z" : 4,
        }
}
```


Example 2

```
int x1 = 5;
String message = "Hello, world!";
x1++;
double[] values = new double[10];
LinkedList list = new LinkedList();
x1 *= 20;
list.add("Foo");
list.add(-34);
String str = message.replace('l', 'L');
values[7] = 99.99
values[1] = 3.14;
```

```
"trace" : {
        "code" : "int x1 = 5;",
        "step" : 1,
        "state" : {
                "x1" : 5,
        }
},
{
        "code" : "String message = "Hello, world!";",
        "step" : 2,
        "state" : {
                "x1" : 5,
                "message" : "Hello, world!",
```

```
        }
    },
    {
        "code" : "x1++;",
        "step" : 3,
        "state" : {
            "x1" : 6,
            "message" : "Hello, world!",
        }
    },
    {
        "code" : "double[] values = new double[10];",
        "step" : 4,
        "state" : {
            "x1" : 6,
            "message" : "Hello, world!",
            "values" : {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
        }
    },
    {
        "code" : "LinkedList list = new LinkedList();",
        "step" : 5,
        "state" : {
            "x1" : 6,
            "message" : "Hello, world!",
            "values" : {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
            "list" : [],
        }
    },
    {
        "code" : "x1 *= 20;",
        "step" : 6,
        "state" : {
            "x1" : 120,
            "message" : "Hello, world!",
            "values" : {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
            "list" : [],
        }
    },
    {
        "code" : "list.add("Foo");",
        "step" : 7,
        "state" : {
            "x1" : 120,
            "message" : "Hello, world!",
            "values" : {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
            "list" : [Foo],
        }
    },
    {
        "code" : "list.add(-34);",
        "step" : 8,
        "state" : {
```

```
                "x1" : 120,
                "message" : "Hello, world!",
                "values" : {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
                "list" : [Foo, -34],
        }
},
{
        "code" : "String str = message.replace('l', 'L');",
        "step" : 9,
        "state" : {
                "x1" : 120,
                "message" : "Hello, world!",
                "values" : {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0},
                "list" : [Foo, -34],
                "str" : "HeLLo, worLd!",
        }
},
{
        "code" : "values[7] = 99.99",
        "step" : 10,
        "state" : {
                "x1" : 120,
                "message" : "Hello, world!",
                "values" : {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 99.99, 0.0, 0.0},
                "list" : [Foo, -34],
                "str" : "HeLLo, worLd!",
        }
},
{
        "code" : "values[1] = 3.14;",
        "step" : 11,
        "state" : {
                "x1" : 120,
                "message" : "Hello, world!",
                "values" : {0.0, 3.14, 0.0, 0.0, 0.0, 0.0, 0.0, 99.99, 0.0, 0.0},
                "list" : [Foo, -34],
                "str" : "HeLLo, worLd!",
        }
}
```