Revised Project Proposal

Tao Wang
CS 6460: Educational Technology
Spring 2016

Changes

- Rewrote Project Description.
- Rewrote Risk Assessment.
- Rewrote Milestones and Tasks.
- Minor changes to Citations.

Introduction

"Programming languages are the least usable, but most powerful human-computer interfaces ever invented" [1]. The primary goal of this project is to make the Java programming language more usable for students in first-year computer science courses. The focus will be on creating a tool that can provide visual aids to make the tracing of Java programs more accessible. The tracing of code is a difficult skill to acquire and demonstrate since it requires both precision (it's easy to make mistakes) and abstract thinking (having the intuition for a model of the computer). The complexity of tracing a program seems to grow exponentially with the number of lines of code. Automating the process can have benefits for both students and teachers.

Background

We can think of computing culture as split into two broad camps: users and programmers [2]. First-year computer science students largely come from user culture, where software is a tool that can be used to accomplish a specific task. These tools have well-defined interfaces and affordances[1] that clearly define how subtasks are completed while also limiting the user from being able to do too much. To be successful in computer science, students must be acclimated to programmer culture, where software is open-ended, and where programmers accomplish tasks by communicating with software using an artificial language.

To be able to program well, students then must develop mental models of how the computer will interpret the instructions written in that language as well as how different sets and orderings of instructions will combine and interact with each other. Unfortunately, current methods in programming education do not emphasize the development of such a model [3]. This may be due to the legacy of a traditionalist practices in teaching that place emphasis on *how* to accomplish certain tasks as well as the amount of time it takes to prepare accurate illustrations of program execution. There is also little evidence of the development of debugging tools that

[1] An affordance is a relationship between a tool and the user that leads the user to perform an action, such as a knob on a door or the ▶ symbol on a media player.

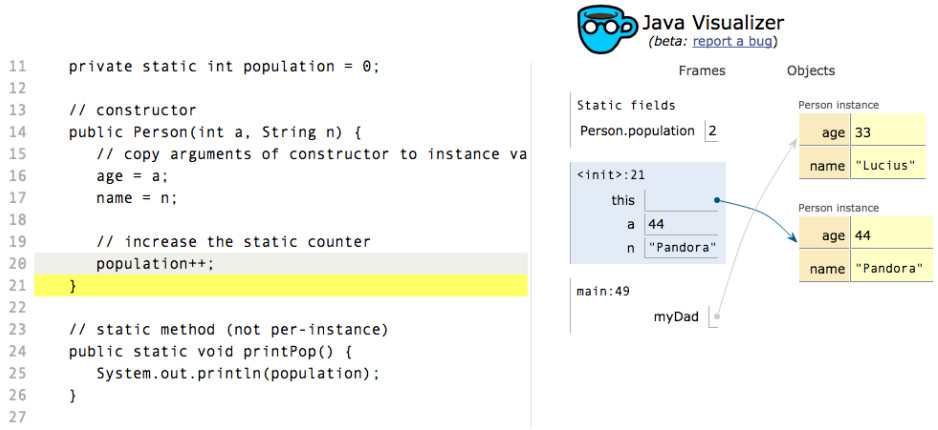
are friendly to beginners. This may be because such a tool would be of little use to veteran programmers.

A beginner-friendly debugging tool could provide a lot of benefit to students and teachers. Students would be able to analyze execution traces of their programs even if their conception of how to do these traces by themselves with pen and paper may be limited at the beginning of their studies. Teachers would also have access to a simple way to demonstrate program execution in class, which would allow for accurate on-the-fly modifications to respond to student questions.

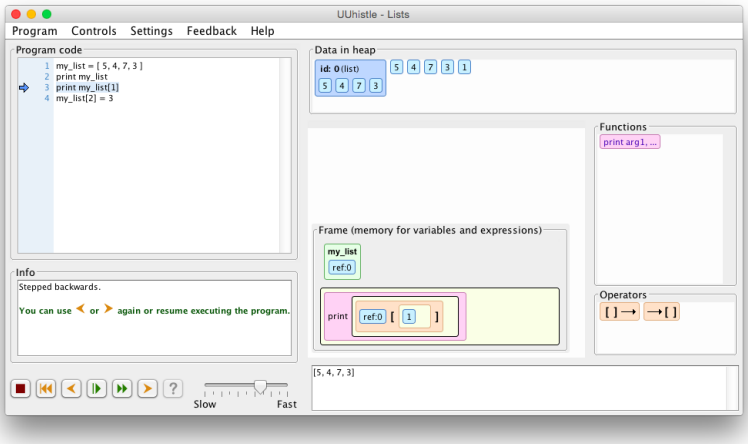
Existing Solutions

This project will build upon Java Visualizer created by David Pritchard and Will Gwozdz [4], which in turn is based off of Online Python Tutor by Philip Guo [5].

Online Python Tutor was motivated by Guo's experience teaching Python and "drawing messy stack and heap diagrams on the whiteboard" [6]. The resulting web application allows users to input Python code through the HTML front-end. An HTTP GET request is made, and the back-end produces an *execution trace* in JSON format which is returned to the front-end. The front-end then interprets the JSON trace, allowing a user to (a) step through the execution of the code one line at a time and (b) see a visual diagram of existing constants, variables, and object data.



UUhistle (pronounced "whistle") is a desktop application that provides animated program traces for Python programs [7]. UUhistle has similar functionality to Online Python Tutor and Java Visualizer but also provides details of expression evaluation. A user can step through a statement such as `print my_list[1]` and see how the reference `my_list[1]` is reduced to the value contained at index 1 of `my_list` before being passed to the `print` function.



Project Description

For this project, I propose to design and develop a prototype for a lightweight student development environment. The prototype should offer the following functionality:

1. Must have: Display visualizations of program state: values of primitive variables, fields of objects, contents of arrays.
2. Nice to have: Create, load, save, and edit Java source files locally.
3. Nice to have: Compile and run Java programs.
4. Stretch goal: display visualizations of expression execution such as the order of operations of a statement like `x = 3 + 5*2`.

The plan is to use BeanShell [8] to evaluate Java programs line by line. The original goal was to modify traceprinter [9], which forms the back-end of Java Visualizer. However, traceprinter is designed with the execution of code submitted through a web front-end in mind. Therefore, it has an extra layer of safety measures that ensure the code it receives is executed safely. This is a little too much for the scale of this project and also presented difficulties in setting up my system.

Risk Assessment

1. I will be implementing an ad-hoc Java interpreter using BeanShell rather than using a pre-existing one such as traceprinter or using the Java Platform Debug Architecture. This is due to technical difficulties that I experienced in the first week of this project. This means that the resulting interpreter will not be as robust as I would have liked and that the stretch goal is a more difficult to achieve. It should be enough for a prototype, but I will monitor this closely in the first week of implementation.
2. As in the original, the GUI aspects of the project are "nice to have". Meaning that they are secondary to the first requirement, which is the visualization of program state. GUI

implementation is left to the later weeks. If time doesn't allow it, dropping the GUI will not be a huge detriment to the project as a whole.

Milestones and Tasks

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.
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.
11	3/27/16	Design visual representations of the trace output. Focus on intuitive representations of objects and arrays.
12	4/3/16	*Intermediate Milestone 3* - Deliverables: (1) GUI program that translates text output into visuals. (2) Documentation containing examples.
13	4/10/16	Implement loading, editing, and compiling source files. Bug fixes and testing.
14	4/17/16	Stretch goal time permitting or continue work from week #13. Bug fixes and testing.
15	4/24/16	*Intermediate Milestone 4* Deliverables (1) Prototype satisfying #1-3 from project description. (#4 time permitting) (2) Documentation. (3) Outline of project paper.
16	5/1/16	Code freeze. Write project paper. Create project presentation.

Citations

[1] Ko, Andy. "Programming languages are the least usable, but most powerful human-computer interfaces ever invented". 25 March 2014. http://blogs.uw.edu/ajko/2014/03/25/programming-languages-are-the-least-usable-but-most-powerful-human-computer-interfaces-ever-invented/

[2] Guo, Philip. "The Two Cultures of Computing". December 2013. http://www.pgbovine.net/two-cultures-of-computing.htm

[3] Ben-Ari, Mordechai. "Constructivism in Computer Science Education". 1998.

[4] Gwozdz, Will & David Pritchard. Java Visualizer. Accessed 21 February 2016. http://cscircles.cemc.uwaterloo.ca/java_visualize/#

[5] Guo, Philip. Online Python Tutor. Accessed 21 February 2016. http://www.pythontutor.com/

[6] Guo, Philip. "Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education". *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM, 2013.

[7] J. Sorva and T. Sirkiä. UUhistle: a software tool for visual program simulation. In Proceedings of the 10th Koli Calling International Conference on Computing Education Research, Koli Calling '10, pages 49–54, New York, NY, USA, 2010. ACM.

[8] BeanShell. http://www.beanshell.org/

[9] Pritchard, David. traceprinter. May 2013. https://github.com/daveagp/java_jail/tree/master/cp/traceprinter