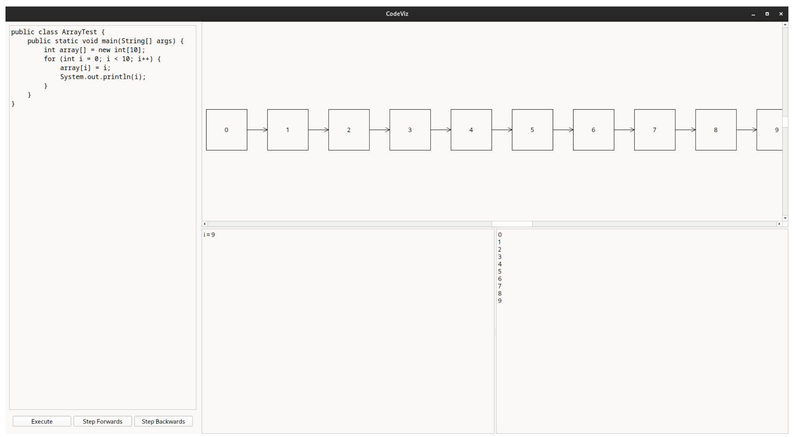
Code Visualization

Curtice Gough, Joshua Hartzfeld, Catherine DiResta

Faculty Advisor(s): Dr. Ryan Stansifer, Dept. of Computer Science, Florida Institute of Technology

GUI



Custom Classes

In order to visualize custom data structures, any Java source file may be placed in the “cp/codeviz/” directory. The source code for David Pritchard’s “traceprinter” was modified such that before execution, the backend initialization will iterate over every file in “cp/codeviz/” ending in “.java”, add it to the “cpFiles” structure, and compile everything alongside the user-provided code.

```
139         String[] cpFiles;
140         try { // List all files in cp/codeviz
141             cpFiles = Stream.of(new File("codeviz").listFiles())
142                 .filter(file -> !file.isDirectory())
143                 .map(File::getName)
144                 .collect(Collectors.toSet())
145                 .toArray(new String[0]);
146         }
147         catch(NullPointerException e) {
148             System.err.println(System.getProperty("user.dir") + "codeviz directory contains no valid .java files");
149             e.printStackTrace();
150             cpFiles = new String[0];
151         }
152
153         String[][] fileinfo = new String[cpFiles.length + 1][2];
154
155         for (int i = 0; i < cpFiles.length; i++) { // Stage all java files for compilation
156             String className = cpFiles[i].substring(0, cpFiles[i].indexOf('.')); // Remove ".java"
157             fileinfo[i][0] = className;
158             fileinfo[i][1] = getFileContents("codeviz/" + cpFiles[i]);
159         }
160         fileinfo[fileinfo.length - 1][0] = mainClass;
161         fileinfo[fileinfo.length - 1][1] = usercode;
162
163         bytecode = c2b.compileFiles(fileinfo); // Compile everything
```

Goal

The primary objective of this project is to achieve code visualization at a medium/high level (as opposed to algorithm visualization). Code tracing is often a complex task. Visualizing structures and the movement of data will aid users with debugging tasks while not spending too much time tracing execution by hand.

Python 3.11 Modules

The framework for this project is built upon the following imported Python modules:

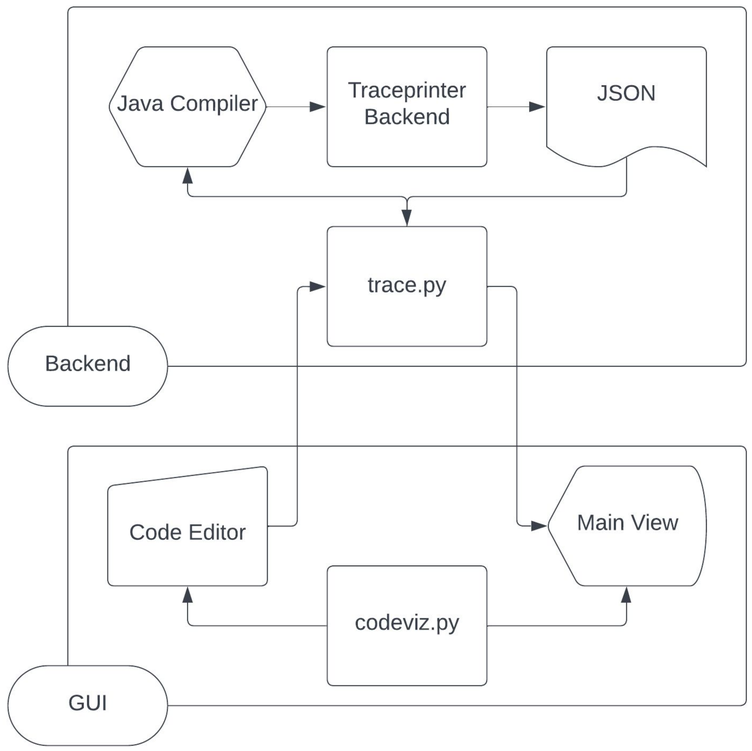
- **PyQt6**
  - Used for GUI development
  - Custom widgets allow modularization of data structure diagrams
- **sys**
  - Provides access to environment variables, command-line arguments, and passing of exit codes
- **json**
  - Provides conversion between Python dictionaries and JSON data readable by “traceprinter”
- **subprocess**
  - Allows execution of child processes
  - Used to pass user-provided code to “traceprinter” and capture JSON output

Code Sources

This project contains code from the following sources:

- https://github.com/curtico/code-visualization
- https://github.com/daveagp/java_jail/tree/master/cp
- https://download.oracle.com/otn/java/jdk/8u20-b26

System Architecture



Structure Diagrams

The visual aspects of this project take inspiration from the University of Waterloo’s “Java Visualizer”. An example of an “array” diagram is shown below.

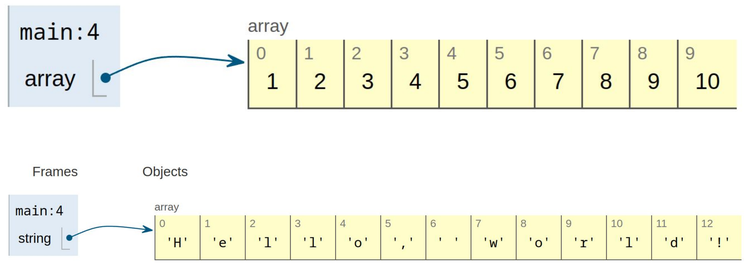


Image Source:
https://cscircles.cemc.uwaterloo.ca/java_visualize/

https://curtico.github.io/code-visualization/