Ryan Sheffler

Source Code Modeling based on User Interaction: A Java Parsing Approach

Ryan Sheffler

Class of 2021, School of Computer Science

❝ I'm Ryan Sheffler, a senior working on my Bachelor's in Computer Science all here at Marist. I grew up equipped with a GameBoy. I remember the fun I had as a little kid fooling about with the few games I had and how much I learned from them. Those experiences are where my fascination with computers began. From playing with computers, I developed an interest in learning how they work and how to make them operate. As time passed, my interests only grew and I began learning computer programming and eventually decided to pursue Computer Science through high school and as my degree in college.

❞

Abstract

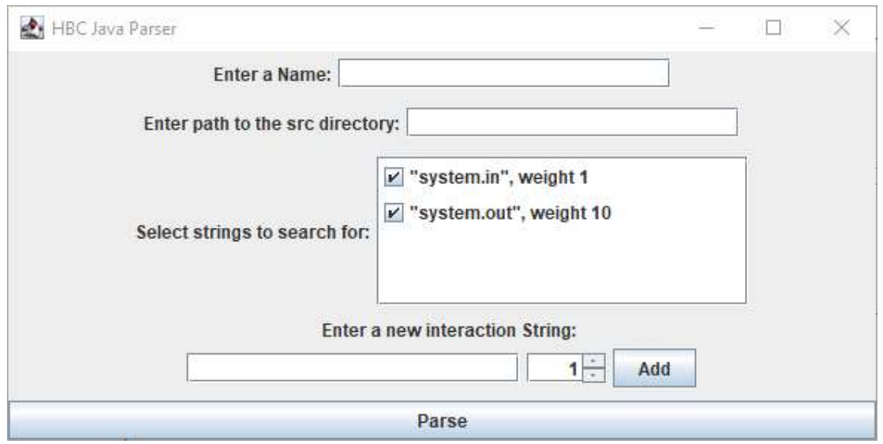
There is an old adage, "A picture is worth a thousand words." Modeling and visualizing the intricacies and complex relationships in systems such software follows this principle. In a previous Honors By Contract project, I created a program to parse Java programs and produce an analysis of how the individual parts interact with each other and the user. The goal of this project is to extend that work to generate a 3D model of the digested data the parser produces. Other similar code parsers typically focus on simply improving code. What can be exploited? How can my code be more efficient? What may cause an error as libraries and other programs are updated in the future? These are all very important questions to be asked, and the answers can be quite valuable for developers. However, they are not the only questions to ask. The solution I propose takes that analysis of how the program interacts with itself and the user and presents it visually to reduce the difficulty for new or returning developers to understand how exactly large projects work and comprehend the dependencies among the parts. The model can show how changing one small part can affect other segments of the program.

Code Parsing?

For those unfamiliar, *code parsing* is the process of taking in code and processing it. Specifically, it refers to a *program* doing that. A *code parser* is a program that exists to read other programs, digest them, and provide certain insights to them. A common example is PMD. Parsers are great for developers, allowing a new set of eyes to quickly be run over their code, reviewing it for any potential errors. Most common parsers will also offer code suggestions and insights, such as pointing out what may soon no longer be supported and suggesting a different way of doing that task that isn't at risk of deprecation. Simpler parsers are also built into most IDEs, highlighting and notifying developers of issues as they type. Standalone parsers tend to try to go deeper, providing the developers with all kinds of feedback on their code, from simple formatting to providing alternatives to inefficient code.

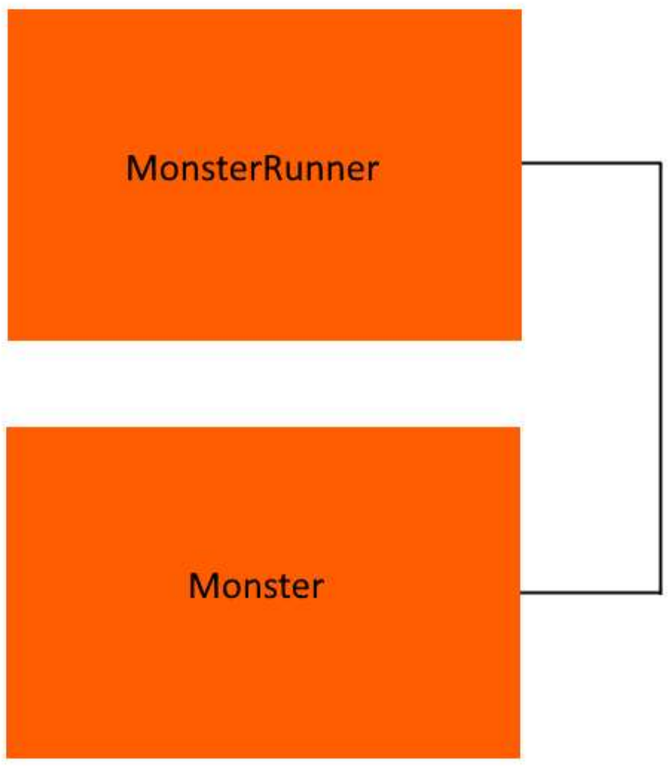
My Implementation

As discussed in the Abstract, my parser keeps its eyes open for a different set of things from most parsers. As it reads through each file in its target directory, it takes a note of the other resources it imports, the methods of the class, and where it mentions other files in the directory, among a few other things. Most importantly, though, it keeps track of where the file interacts with the user, and generates a score based on how much it interacts with the user. What counts as an interaction with the user can be defined by the user of the parser, but defaults to the standard "System.out" and "System.in" calls. It then stores these in a file, to be used by the modeler component of my project.



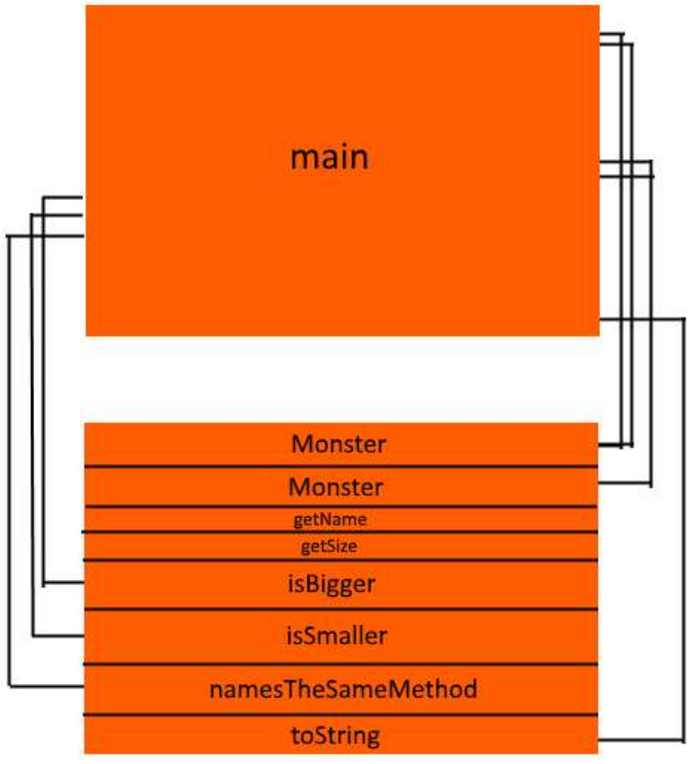
A view of the parser component's GUI.

The modeler unfortunately isn't yet complete. However, I do have a simple mockup to show. In this case, I took a simple 2 file/class program I wrote while learning Java. MonsterRunner.java is the main method, being what the user would run to begin the program, which calls on Monster.java to create and store data. A simple view may look something like the following:



A low-detail model of the program.

This simple view just shows the files and their direct connections. With this trivial program, this view isn't particularly interesting, but with a larger program, this alone could be valuable information. But where does the 3D aspect of this model apply? I couldn't demonstrate it here, but that is determined by the user interaction score the parser gave the file. In this case, MonsterRunner is the only one printing to or receiving information from the console, so it would be more "in front" or closer to the viewer than Monster. What the end user sees is a very important thing to developers, so it's important to know what is shaping their view. What if we wanted to see a bit more information, though?



A sample model showing more detail about the program.

My parser keeps track of what methods are in a file and where it calls other files in the directory. A higher detail view could reveal these methods, and show these more advanced connections, making it easier to see just what depends on what else. This unique relational view of a program is great for developers who are unfamiliar with a project. In the technology industry, developers are constantly being shifted around projects and a major difficulty with this is the speed at which they're able to adjust to their new project. Seeing a simplified model of a program represented visually like my modeler produces makes it much easier to wrap one's head around a large program quicker, meaning it takes much less time for a new developer to truly pick up and become a member of their new team.

My current code (and paper when it's completed) can be found at my GitHub repository here.