Jasper Loverude, Dylan Tymkiw, Cassidy Correll
CS361: Object-Oriented Programming
Professor Skrien, Colby College
May 6, 2022

Project 10: Transpiler

Design Overview

The ultimate goal of this project was to use the AST generated by the parser to create a transpiler that could generate legal Java programs from Bantam Java. We created a TranspilerVisitor which extended the Visitor class, as following the Visitor pattern was necessary to traverse the AST. The TranspilerVisitor stores a StringBuilder, which receives java compliant code appended by the various visit methods. The Transpiler class then writes the String from the TranspilerVisitor class to a java file. We also changed the IDE buttons from "Compile" and "Compile & Run" to "Transpile" & "Transpile & Run". The Transpile button transpiles a bantam java file into a new java file, compiles it with the builtin java compiler, and opens the .java file in a new tab.

We added this opening in a new tab addition initially for the sake of presentation– however, we kept it in as we felt it demonstrated a number of our project features well. The Transpile & Run button does everything Transpile does, but then executes the compiled .class files.

Within our TranspileVisitor, we added indentation support when writing the .java file to make the transpiled file legible. We stop short of calling this a PrettyPrinter, as our program does not retain comments or spacing- while it does format it in an easy-to-read style, it is not intended to fully prettyprint the file.

Elegant Design Features

In designing our project layout, we decided to split up the building of the program string, and the actual writing of the .java file. We figured that the IO work of writing a file did not belong in the TranspilerVisitor, whose only job was to traverse the tree and build a string from it. The class response for IO was the Transpiler itself, within the same transpiler package, which was the director of all previous parts of the Bantam compiler project (lexer, parser, and semantic analyzer.) This made debugging and implementation much easier, as each class had only a single task to do.

Shortcomings

There are two primary shortcomings with our project, one related to elegancy and the other related to implementation. The first shortcoming, related to functionality, is that our semantic analyzer does not work exactly as it should. It does not allow for dispatch statements with method calls, such as (Class).method(). In implementing the print statement translation, we had to take a fairly inelegant approach because of this. With dispatch method calls not working io.Stdout was not being analyzed properly so we added a “print()” method to the object class, in order to successfully pass analysis as every class is an object. We then added a special translation for the print() method to the proper java version. We also used a similar special translation logic for the main method.

Other semantic bugs were also not fixed, however these did not lead to us making drastic workarounds. One such problem is that our program does not properly catch cast statements that try to cast to classes that are not super or subclasses of the original data type.

From project 6, some inelegancies remain still. The find, of find and replace, highlights and replaces entire words instead of just the characters searched for in find. Additionally, our Controller class remains very bloated. It would be cleaner to create helper classes for things like running, checking, or transpiling selected tabs and Input/Output handling. Controller does way too many things, and should instead talk to separate, specialized helper classes.

A new shortcoming, related to our Transpiler buttons, was with the feature we added of opening the .java file in a new tab. We tried implementing booleanProperties to disable the Transpile and Check buttons, which should not be clickable when a non .btm file is open. This quickly grew inelegant as we sought to find every case where a file would certainly be a .btm file again, or certainly not be one. The more elegant solution would have been to implement a listener for when the user changed tabs, and update it then. However, we realized this far too late in the process and scrapped it in favor of not submitting our project later than it needed to be.

Division of Labor

For this project we all worked together to design the new classes we would need, then we evenly split up the new visit methods we needed to override. Dylan also fixed bugs from project 6. Cassidy added the Check button to the IDE. Jasper made the Transpile and Transpile & Run buttons on the IDE.