

OLIGARCHY


Jiamin Zhu, Eduardo Preciado, Imon
Daneshmand, Stephanie Contreras, Daniel
Cardenas



An Object Oriented Programming inspired language that makes use of class based inheritance. A lower level, target language is being used with the intent of providing a deeper understanding of compilers and programming languages. Non-trivial features include High Order Functions and Expressions. Because we are going to a low level, target language, both features introduce greater complexity.

We based our language on other object oriented languages such as Java. This inspired our languages syntax and formatting.

**HIGH-LEVEL
DESCRIPTION OF
YOUR LANGUAGE
DESIGN. ARE
THERE ANY
LANGUAGES
WHICH YOU
BASED THIS ON?**

Several white lines of varying lengths and angles are drawn on the right side of the slide, extending from the text area towards the bottom right corner.



Our target language is JVM Bytecode. This was chosen as an opportunity to learn more about code generation.




We implemented our compiler in Scala. This was to make use of features such as pattern matching. Though unfamiliar with Scala, because it stemmed from Java we believed it to be functional.

WHAT IS YOUR TARGET LANGUAGE, AND WHAT LANGUAGE DID YOU IMPLEMENT YOUR COMPILER IN?

- ▶ Key features include:
 - ▶ High Order Functions : chosen since JVM already has class and inheritance bytecode
 - ▶ Inheritance : similar to Java with “extend” keyword used
 - ▶ Expressions : adds complexity since we're doing a low level target
 - ▶ Class definitions : includes statements and methods which can be called
 - ▶ Method definitions : can take a list of parameters and different types of returns such as void or an int
 - ▶ Arithmetic Operations : elementary binary operations such as add, multiply, lesser than

**WHAT SORT OF KEY
FEATURES DOES
YOUR LANGUAGE
HAVE, INCLUDING
YOUR NON-TRIVIAL
FEATURES?**



WHAT SORT OF KEY LIMITATIONS DOES YOUR LANGUAGE HAVE?



Does not include memory de-allocation



Lacks setters and getters



Limitations in specific features such as High Order Functions and types of Expressions have been brought to light during development of code generation.



Does not feature sub-typing or type-casting



No optimizations



No “this” feature for classes

outside variables must be named differently from local variables

Learning Scala

- learned from prior code examples

Issues introduced by the Covid pandemic, while also juggling class changes and personal life and work

- removed sub-typing as a feature


Using a low level, target language (JVM bytecode)

- reworked much of our prior work such as in our parser and type-checker
- made use of code generation examples extensively

The writing process for type-checking and code generation revealed limitations in our syntax

- we changed how the ast was structured when needed
- updated syntax for more complex features

WHAT WERE THE BIGGEST CHALLENGES THAT AROSE DURING DEVELOPMENT? HOW DID YOU RESPOND TO THESE? (E.G., DID YOU CHANGE YOUR SYNTAX, DID YOU CHANGE HOW SOMETHING WORKED, ETC.)



**WHAT SORT OF
LESSONS WERE
LEARNED? CLOSELY
RELATED: IF YOU
COULD DO IT ALL
OVER AGAIN
KNOWING WHAT
YOU KNOW NOW,
WHAT WOULD YOU
DO DIFFERENTLY?**

Being able to make use of Scala Test and SBT better would have helped tremendously.

A more robust concrete grammar. As changes or updates were needed, things became more confusing.

Given the complexities of some features and our limited time, putting some of the work on the user

Making the user write out more information in their code, the token generator and type-checker would have more details regarding their context.

May have been easier to start with a simple program in our designed language and then slowly introducing additional features into the program