Featherweight Java Implementation

Introduction

Featherweight Java (FJ), as introduced by Igarashi, Pierce, and Wadler, is a minimal core calculus for Java, designed to model essential features of the language while omitting many complexities. Inspired by the lambda calculus and the object calculus, FJ focuses on object creation, method invocation, field access, typecasts, and variable usage, all formalized within a small-step operational semantics.

This project presents an implementation of an interactive interpreter for Featherweight Java. The interpreter enables parsing, typechecking, and evaluation of Featherweight Java programs, closely following the rules of the original paper

A) Features

The project includes several components:

- A Parser that reads .fj files containing Featherweight Java class definitions;
- A Typechecker that verifies the well-formedness of classes and the typing of expressions according to FJ's typing rules;
- An Evaluator implementing small-step operational semantics to perform computation as described in the FJ paper;
- A REPL (Read-Eval-Print-Loop) for interactively loading class tables, typechecking, declaring variables, and evaluating expressions; Variable Context Management allowing users to manually declare variables and types via commands like :var NAME TYPE;

B) Commands

- :parse FILENAME.fj — Parse a Featherweight Java source file.

- :typecheck — Typecheck the currently loaded class table.

- :var NAME TYPE — Declare a new variable manually in the context.
- :ct — Show the current class table.
- :help — Display available commands.
- :q — Quit the interpreter.

C) Design Highlights

The interpreter mirrors the main characteristics of Featherweight Java:

- **Simplicity**: Only five expression forms (object creation, method invocation, field access, cast, and variable).
- **Static Typing**: Type safety is enforced at runtime via a sound type system.
- **Functional Style**: No assignment, no mutable state; all objects are created fully initialized and never mutated.
- **Error Handling**: Type errors, unbound variables, unknown classes, and invalid field or method accesses are gracefully reported.

```
FJ> :parse Example.fj
Parsed! (but not typechecked yet)
FJ> :typecheck
Typecheck succeeded!
FJ> :var x Object
Added variable x :: Object
FJ> :var y Object
Added variable y :: Object
FJ> (new Pair(x, y)).fst
Starting expression:
FieldAccess (New "Pair" [Var "x",Var "y"]) "fst"
Type: Object
 =>
Var "x"
Result:
Var "x"
FJ>
```

Conclusion

By building this project, I was able to understand the main ideas behind Featherweight Java. Even though the system is much smaller than real Java, it still captures many important concepts like classes, fields, methods, inheritance, and type safety. Working through the typechecker and evaluator helped me see how typing and execution rules are designed carefully to avoid runtime errors.

How to run this project:

1- install GHC(https://www.haskell.org/downloads/)

2- Open a terminal in the project folder (where the .hs files are).

3-Launch ghci

run **ghci**

4-Enable the containers package

run **:set -package containers**

5-Load the source files:

run **:l Syntax.hs Parser.hs Typechecker.hs Evaluator.hs Main.hs**

6-Set active module to `Main`

run **:module +Main**

7-Finally run the REPL:

main