EECS 293
Software Craftsmanship
2019 Spring Semester

Programming Assignment 11

Due at the beginning of your discussion session on

April 9-12, 2019

Reading

- Section 6.2 (skip "Present a consistent level of abstraction in the class interface" and "Provide services in pairs with their opposites") in Code Complete
- Section 6.4 in Code Complete
- Items 15, 16, 17 in Effective Java
- Section 19.6 in Code Complete
- Module "Routine Names" on canvas

Programming

The final project builds upon assignments 2 through 5 on expression parsing. You are now expected to design classes and their methods to support additional features revolving around type checking. As you have learned in Section 5.1, you fully expect the design process to be sloppy, non-deterministic, based on heuristics, and iterative (and possibly very frustrating). However, you also shoot for a product that is tidy, clearly addresses the relevant trade-offs, priorities, and restrictions: in other words, it will be a beautiful architecture.

Type Synthesis: Examples

The EXPRESSION parser can generate parse trees for arithmetic expressions. However, the type of the root EXPRESSION is not generally clear. Type synthesis refers to the process of determining the type of an expression from the type of its subexpressions. Some

applications, such as arithmetic type synthesis, are expected, but there are other scenarios where the expression parser can also be applied and where the expression type is unrelated to arithmetic. Here are a few examples.

Java Arithmetic

Java follows standard rules to determine the type of an expression. For example, if *a* and *b* are both integers, then the expression *a*+*b* is an integer. If both are double, then the result should be a double. If *a* is a double and *b* an integer, then the result should be a double.

Extended Arithmetic

The project should be able to support novel numeric types that may be introduced in a programming language. For example, suppose that Java were extended with a complex data type. In this case, the expression *a*+*b* would be a complex if either *a* or *b* are complex.

Function Compositions

Expressions can also be applied in a functional context. In the case of functional composition, the expression type is a departure from the conventional arithmetic interpretation. Suppose that *f* and *g* are `Function<T>` and *a* is of type `T`. Then, *f* * *g* is defined as `g.andThen(f)`, which is a `Function<T>` and *f*+*a* as `f.apply(a)`, which is a `T`. The types of other expressions can be defined analogously and assign a special non-arithmetic meaning to the operators.

Type Synthesis

From these examples, the main takeaway is that type synthesis should be flexible so that it could work among these different examples and more. In other words, type conversions should be an input to the type synthesis rather than a hardwired component of the algorithm.

Your job is to design the architecture of a system that given a (simplified) parse tree, variable types, and type conversion rules, finds out the type of the parse tree root. You are allowed to modify the architecture of the parser if you need to do so to support type inference.

Design

Create a design document that describes your architectural decisions. Your document can optionally contain sketches and diagrams of your architecture. Your objective is that your design document should be used as a blueprint for the implementation. You can define classes or interfaces as you see fit. Commonalities among classes or interfaces should be expressed through inheritance or containment. For each class or interface, you should describe at least the abstraction it captures, its position in the inheritance hierarchy, the signature of its constructors and public methods, its private data structures (if any), and the pseudo-code of any complicated method. An integral part of software architecture is an approach for error handling. You should also outline your approach to testing.

You are not required to submit an implementation, which is the main topic of the next assignments.

Discussion Guidelines

The discussion will focus on class design.

Submission

Submit design documents that describes your architectural decisions. No implementation is required: you will implement your design in the next programming assignments. This assignment can be submitted on Canvas.