Today we continue our discussion of Featherweight Java. Specifically, we introduce the syntax of the untyped language and a few ancillary components that will be used in the type system (next lecture).

1 Review

In our previous lecture we introduced and discussed Featherweight Java. Specifically we talked about

- What features should it have?
- What should the syntactic subset look like?
- What are values? We suggested classes and objects - Choose Objects (importantly, they will be irreducible terms much like lambda values in the simple lambda calculus); More specifically values are defined as:
  \[
  e ::= \ldots \text{object-name}(\tau) \\
  v ::= \text{new class-name}(\tau)
  \]

- No references (side effects, assignment). Instead, we have a pure object calculus.

- We also talked about what features of Java would be interesting to model in the Calculus (small subset). The goal being an untyped system + typing system to guarantee some property (that applies to the full language in some way). The property that we are interested in is safety which means that it does not get stuck which means that we do not invoke a method on an object that doesn’t support it.

In addition, we noted that full Java does not have structural subtyping, but nominal subtyping. This means that the subtyping does not compare structure, instead all types are explicitly named and the subtyping relationship explicitly described.
We also identified three types of casting in full/featherweight Java:

1. Up-Cast: Become super type; hide subtype stuff; statically checked
2. Down-Cast: Become subtype; reveal subtype stuff; determined at runtime
3. Stupid-Cast: Casting between “unrelated” objects; disallowed in full java, but part of FJ for technical reasons (see p. 259)

2 FJ - Untyped

Reviewed type rules (refer to Figure 19-1 on p. 255) for meaning and intent.

A few points about constructors.

1. Constructors have to be defined carefully to maintain the semantics. The new object is a value and all of the arguments to the constructor are values. Observe that the invocation of the constructor `new class-name(...)` is a self-contained description of the object. The state of the object is explicit.

2. The call to `super` is because FJ is a perfect subset of full java. This makes it easy to check (compile and run) and easy for people (who know java) to understand the semantics/syntax.

3. Constructors must contain one argument for each field in the class, and each field is set explicitly.

Methods have no statements. Only `return` expressions. The type of the return value is always some class C.

**Side Note:** Figure 19-1 is missing some constraints that apply to the untyped language depicted (FJ). The author pushes these constraints to the type system. Others, Prof. Taha included, feel that there should be a separation of the untyped and typed language and the untyped definition should contain all applicable constraints.
3 FJ’s Type System

In Figure 19-2 are defined *Auxiliary Definitions*. These are functions used in FJ’s Type system. Important to note that each of these functions has a type associated with it.