To start formalizing the Java type system, we need to determine what we want to include in the formalization (what we need).

- Syntax(1). In Java, do we have:
  - Variables (something with an introduction instance and a usage instance)? Yes. We have method arguments, fields, local variables.
  - Lambdas (binding structures)? Yes. We have method arguments, local variables, class declarations, method and field declarations.
  - Applications? Yes. We have method invocation, field/method lookup.

- Type system. What kind of guarantees do we want?
  - Goal: Objects understand the methods we send.

    Nominal subtyping: \( A \) is a subtype of \( B \) if \( A \) inherits from \( B \). In other words, the definition of \( A \) declares that \( A \) inherits from \( B \). This is different from structural subtyping where we needed subtyping rules.

Now we need to pick a subset of Java to study what we need. (...) in the syntax below indicates extra information to provide type information.

- Syntax(2)
  - class decls (name)
    - superclass (name)
    - methods (name, arg names, body) \( \langle \text{ret class name, arg, class names} \rangle \)
      - constructor (var names, call superclass, field assignment) \( \langle \text{var (class names)} \rangle \)
  - body
    - return (exp) There is no assignment here. See new below for reason.
  - exp
    - variables (var name)
- method invocation (var names, method name, arg exp(s))
- field lookup (var name, field name)
- new (class name, arg exp(s)) In this way we can create different objects; we don’t need imperative features.

Remember Lambda Calculus. What was irreducible there? What would be irreducible in our Java subset?

- Syntax(3)
  - values?
    - class description?
    - new (name) (values) which interacts with the constructor?

We will pickup on this topic next time.