Designing the Translator

- Follow the basic functional requirements
  - Read in Java source
  - Analyze Java
    - Recursively read in and analyze dependencies
  - Transform Java into C++
  - Pretty print C++ code
- How to represent source code internally?
  - Large string is not going to be helpful
  - Lots of strings ("tokens") are not much more helpful
  - Instead: Tree representation that captures relevant syntax
    - But omits comments, spacing, etc.
    - Called Abstract Syntax Tree (AST)
- How to represent AST as objects?
  - One class per language construct (i.e., a statically typed AST)
    - Each class has fields, getters, setters to capture components
      - E.g.: ClassDeclaration has fields for name, superclass, interfaces, fields, methods, and nested classes
      - Provides more safety
      - Class hierarchy encodes knowledge about well-formed programs
  - One generic class (i.e., a dynamically typed AST)
    - Has name, some number of arbitrary children
    - Provides more flexibility
    - Any tree node can plug into any other tree node
    - Can create new nodes simply by using a new name
- How to represent translator phases as objects?
  - One method per phase (does not work with generic class)
    - All functionality for construct included in a single class
      - Easier to add new constructs
    - But adding new phases requires modifying all classes representing constructs
  - One class per phase (i.e., "Visitor")
    - All functionality and state for phase included in visitor
      - Easier to add new phases
    - But adding new constructs requires modifying all visitors
- xtc
  - Supports both statically and dynamically typed ASTs
    - All AST nodes are subclasses of xtc.tree.Node
    - Generic nodes are instances of xtc.tree.GNode
  - Organizes phases as visitors
    - Subclasses of xtc.tree.Visitor
  - Includes other helpful functionality
    - xtc.util.Tool provides skeleton for overall tool
    - xtc.util.SymbolTable helps with the analysis of ASTs by mapping names to other objects
    - xtc.lang.JavaParser converts Java source code into Java ASTs (using generic nodes)
    - xtc.lang.JavaPrinter and xtc.lang.CPrinter pretty print Java and C ASTs (respectively)
    - xtc.type.* contains representations of programming language types, including for Java
    - xtc.lang.JavaAnalyzer implements a type checker for Java
      - Look at xtc.lang.Java for how to use it