Object-Oriented Programming

CSCI-UA 0470-001
Class 6
Instructor: Randy Shepherd

Scope

What is Scope?

- Scope is the area of visibility of identifiers such as variables, fields, methods, etc.
- For example, all the fields and methods of a class are visible within the class.
- Java & C++ have ‘lexical’ or ‘static’ scoping rules.
- (There is another type called ‘dynamic scoping’, which is uncommon in modern languages.)

```java
public class Scoping {  // class scope begins

    // Because 'a' is declared just inside the *class curly braces* it is in 'class scope'
    // therefore its visible everywhere inside the class. This is the 'widest' scope in this file.
    private int a = 0;

    public void methodOne() {  // methodOne scope begins
        // Because 'b' is declared inside the *methodOne curly braces*
        // its only visible in methodOne
        int b = 0;
        if (a == b) { // conditional scope begins
            // Because 'c' is declared inside conditional curly braces its only
            // visible inside the conditional
            int c = 1;
            // Because 'b' is declared in a scope which surrounds the scope of
            // the conditional, 'b' is visible.
            b += c;
            // Because 'a' is declared in a scope that surrounds this scope of
            // the conditional, 'a' is visible.
            a += c;
        }
        // 'c' is no longer visible
    }
    // 'b' no longer visible

    public void methodTwo() { // methodTwo scope begins
        int d = a;
        // 'a' is is visible since its in a scope that surrounds this scope (class scope)
        // 'b' & 'c' not visible, they are in a scope that does not surround the scope of this method
        // 'd' only visible inside methodTwo
    }
    // 'd' no longer visible

    // Note: Constructors obey the same rules as methods w.r.t. variable scoping
}
```

Xtc

What is Xtc?

- Stands for eXTensible Compiler
- “..a project … exploring novel programming languages and tools to improve the expressiveness, safety, and efficiency of complex systems.”
- Translation: a library for source-to-source translators
- Development lead by Robert Grimm as a research tool here at NYU
- Last major release was 8/17/14. Grimm has left NYU
- You can find its “documentation” here http://cs.nyu.edu/rgrimm/xtc/

Xtc Tradeoff

- Xtc provides lots of functionality that can facilitate our projects.
    - ex. Xtc has a method that creates a complete Java Ast given an input source file in .java
- Trade-off: xtc is an extensive library but is complex.
- Ex. Xtc has the construction of a C Ast builtin which we could utilize in our project; however, it does NOT support C++ Ast
    - Extend the C Ast? Build out own? Might actually take longer to do the former.
- Something for each group to think about

Random Xtc Facts!

- Xtc will not tell you if you have malfunctioning Java code
- Xtc does not have pretty printing in C++
- Xtc does not support multithreading

Xtc's Visitor

Visitor and Nodes

- The Xtc mechanism for traversing Ast's.

- Ast's take the form of a Node in Xtc. We are primarily interested in GNode, which is short for GenericNode.

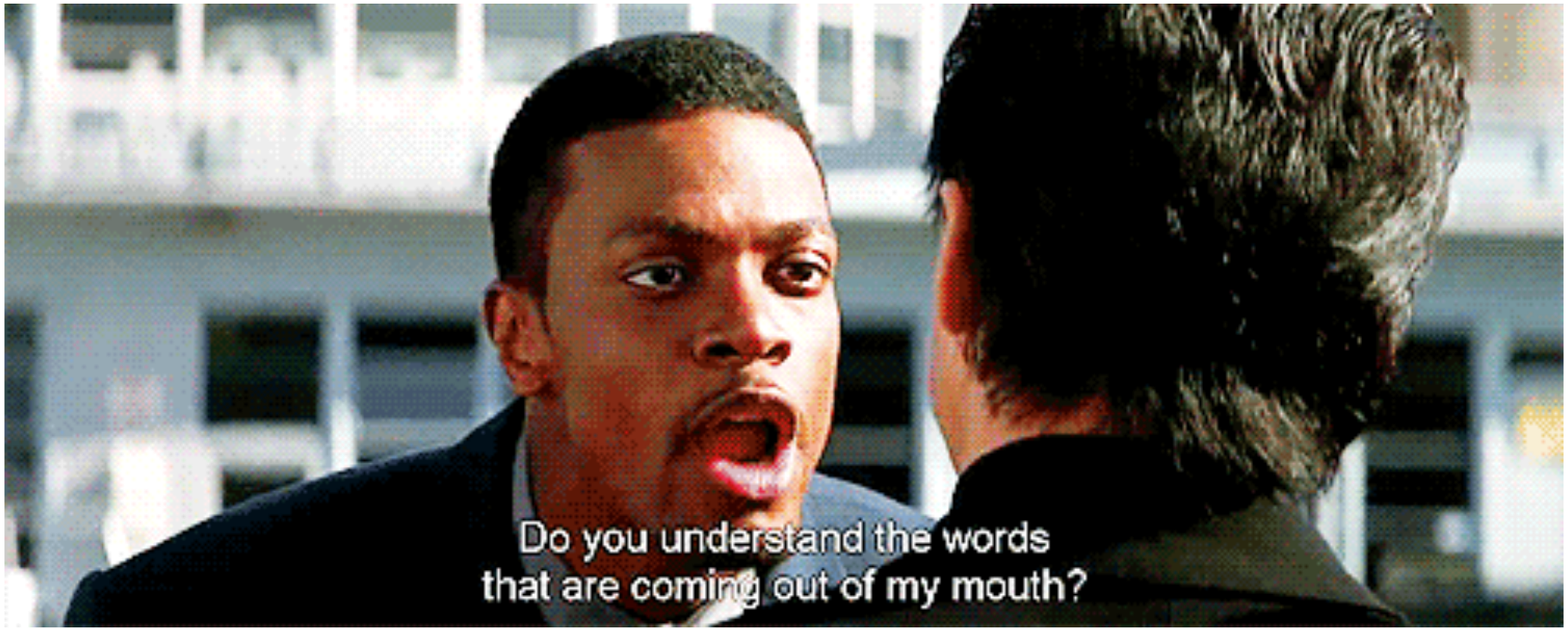
- GNode might be better called DNode for Dynamically-typed Node.

Dispatch Method

- The key method of the visitor class is “dispatch"
- The contract is as follows:
  - if the dispatched node is a dynamically typed GNode with name NAME then find the method "visitNAME" and call it with the dispatched node.
  - if the dispatched node has not method to handle it and is some type that extends Node, then find the "visit" method that takes a C node and call it

```
new Visitor() {
  // Notice that the Visitor class inside the process method is an example of an anonymous class.
  // http://docs.oracle.com/javase/tutorial/java/javaOO/anonymousclasses.html

  private int count = 0;

  // The names of these methods are in fact very important...
  // any method named visitXXX will be called on any Ast node of type XXX.
  // There are some subtle things going on related to xtc even in this small class.
  public void visitCompilationUnit(GNode n) {
    visit(n);
    runtime.console().p("Number of methods: ").p(count).pln().flush();
  }

  public void visitMethodDeclaration(GNode n) {
    runtime.console().p("Name of node: ").p(n.getName()).pln();
    runtime.console().p("Name of method: ").p(n.getString(3)).pln();
    visit(n);
    count++;
  }

  // The method call visit(n) will recursively visit the children of node n.
  // Each time visit is called, we use for loop to iterate over all the children of the node and
  // for each child which is a node call dispatch which dispatches this visitor on that node.
  public void visit(Node n) {
    for (Object o : n) if (o instanceof Node) dispatch((Node) o);
  }

}.dispatch(node);
```

Lets look at some code…



https://github.com/nyu-oop-fall16/xtc-demo