CS212

Java Practicum

Code Generation
Spring 2006

Announcements

- ACSU Meeting, 5pm today, Upson Lounge
- Improved website!
- P2 posted:
  - P2A: planning, meeting with TAs
  - P2B: implementing your plans
- Reminder:
  - The CS212 Pledge! What is it...?

Programming

- Programming reminders:
  - Brainstorm
  - Research/Planning:
    - encapsulation: _____
    - inheritance: _____
  - Outline: _____
  - Draft: _____
  - Write: _____ _____
  - Polish: _____
- OOP reminders:
  - Nouns:
    - is it composite?
    - is it an attribute/property?
  - Verbs:
    - is it an action that's not built-in?
    - is it a "common" action?

High-level Plan

`Bali` → ? → `Samcode`

- The ? is _____
- Approach:
  - Parse the source
  - Translate to target
- Structure? Two kinds of trees!
  - Convert grammar to code
  - Parse Bali code into Samcode
- Legality of code?
- Recursion and ASTs?

Bali-- Reminder

```
program → mainfunction
mainfunction → main { declblock statblock }
declblock → { decl* }
decl → int name ;
statblock → { statement* }
statement → if ( expr ) statblock ;
statement → while ( expr ) statblock ;
statement → name = expr ;
expr → ( expr + expr )
expr → ( expr < expr )
expr → int | name
```

Some semantics: variables get values before being used, **main** function must return ***int***

Example AST

```
// t1.bali
main {
  { int x; int y; }
  { x = 1; y = x ; }
}
```


Designing a class for your AST

- Process:
  - read a Bali token
  - create appropriate AST node
- Class design of for each AST node:
  - classes correspond to...?
  - nouns?
  - verbs?
- Subtyping and contracts...

AST Interface

- Java truths:
  - strongly typed
  - subclassing and reusibility
- Compiler needs:
  - each AST node must be part of same AST
  - nonterminals may yield different types
- Use subtyping to resolve!
  - all classes for your grammar implement same interface:
    ```
    interface ASTNode { String toSamcode(); }
    ```
  - implications for writing the classes?

Example Class Stubs

```
public interface ASTNode {
    public static final String TABBY="|___";
    abstract public String toSamcode();
    abstract public String toAST(String tab);
    abstract public String toBali();
    abstract public String toString();
}

public class BC { public static void main(String[] args){} }
public class BaliCompiler implements CS212Compiler { }
public class Program implements ASTNode { }
public class Main implements ASTNode { }
public class DeclBlock implements ASTNode { }
public class Decl implements ASTNode { }
public class StatBlock implements ASTNode { }
public class Stat implements ASTNode { }
public class WhileStat implements ASTNode { }
public class IfStat implements ASTNode { }
public class AssignStat implements ASTNode { }
public class Expr implements ASTNode { }
public class IntExpr implements ASTNode { }
public class NameExpr implements ASTNode { }
```

Note: expression class shortened a bit.
See website for utility files (also in Jar format)

9

The "Finger"

- Some questions:
  - How does your compiler know where to parse?
  - How does the compiler go from lines of Bali code to AST?
- Answer...the "finger" (see below – drawn in lecture)

10

Supplied Classes

- How does all this work?
  - **BC** sets up a **Tokenizer** object (the finger)
  - **BC** sets up a **BaliCompiler** object
  - Building your AST:
    - **BaliCompiler** sets up **Program**
    - **Program** sets up **Main**
    - **Main** sets up blocks and so forth
- Responsibilities of each AST class:
  - create nodes for your sub-constructs (from grammar)
  - pass along the lexing "finger" to those sub-constructs
  - generate Samcode when asked

11

Some Examples Ahead...

- Some specifications:
  - Everything posted on-line in Jar file
  - You must not alter given packages
  - You must actually use those packages
  - You must use the given files (website also has them)
- **The CS212 Pledge:**
  - you will solemnly swear that...
    - **you realize that Bali-- is not your Part 2 grammar**
    - **your AST interface only needs one method to generate Samcode, not the tree or the other "stuff"**
  - so, you no longer need to ask us why the grammar in Chapter 2 and the lecture notes is different than Part2's

12

Root Node

```
import edu.cornell.cs.sam.io.*;
import edu.cornell.cs.bali.compiler.*;

public class Program implements ASTNode {

  private ASTNode mainFunction;
  public static int labelCount; // for control structure jump labels

  public Program(Tokenizer f) throws IllegalBaliException {
    mainFunction = new Main(f); }

  public String toSamcode() {
    return "//--Program--\n" + mainFunction.toSamcode() + "STOP\n"; }

  public String toAST(String tab) {
    return "\n" + tab + this + "\n" +
    mainFunction.toAST(tab+TABBY);
  }

  public String toBali() {
    return "" + mainFunction.toBali();
  }

  public String toString() {
    return getClass().getName();
  }

}
```


Node with Many Subnodes

```
public class DeclBlock implements ASTNode {

  private static HashMap<String,Integer> symbolTable;
  private Vector<ASTNode> decls;
  private int varcount;
  private String bali;

  public DeclBlock(Tokenizer f) throws IllegalBaliException {
    beginDecls(f); compileDecls(f); endDecls(f); }

  public void beginDecls(Tokenizer f) throws IllegalBaliException {
    bali = ""; bali += f.getOp()+"\n"; }  // eat "{"

  public void compileDecls(Tokenizer f) throws IllegalBaliException {
    symbolTable = new HashMap<String,Integer>(0);
    decls = new Vector<ASTNode>(0);
    int offset = 0;
    while ( f.peekAtKind() != Tokenizer.TokenType.OPERATOR ) {
      Decl d = new Decl(f);
      decls.add(d);
      symbolTable.put( ((Decl) d).getName(), ++offset );
      bali += d.toBali();
    }
    varcount = symbolTable.size();
  }

  public void endDecls(Tokenizer f) throws IllegalBaliException {
    bali += f.getOp() + "\n"; } // eat "}"

  // class continues
```


Terminal Node

```
import edu.cornell.cs.sam.io.*;
import edu.cornell.cs.bali.compiler.*;

public class IntExpr implements ASTNode {

  private int value;
  private String bali;

  public IntExpr(Tokenizer f) throws IllegalBaliException {
    bali = "";
    value = f.getInt();
    bali += value;
  }

  // Generate code:
  public String toSamcode() {
    return "PUSHIMM "+value+"\n"; }

  public String toAST(String tab) {
    return tab + this + "\n"; }

  public String toBali() {
    return bali; }

  public String toString() {
    return getClass().getName(); }

}
```