Session Plan

- Session 4b: ***MiniJava abstract syntax trees***
  - re-introduce MiniJava
  - MiniJava abstract syntax trees in Java
  - JavaCC for generating MiniJava abstract syntax trees

MiniJava

- a subset of Java – example program:

```
class Factorial {
    public static void main(String[] a) {
        System.out.println(new Fac().ComputeFac(10));
    }
}

class Fac {
    public int ComputeFac(int num) {
        int num_aux ;
        if (num < 1)
            num_aux = 1 ;
        else
            num_aux = num * (this.ComputeFac(num-1)) ;
        return num_aux ;
    }
}
```

Abstract Syntax for MiniJava (I)

```
package syntaxtree;

Program(MainClass m, ClassDecList c1)
MainClass(Identifier i1, Identifier i2, Statement s)


abstract class ClassDecl
ClassDeclSimple(Identifier i, VarDeclList vl,
                methodDeclList m1)
ClassDeclExtends(Identifier i, Identifier j,
                  VarDecList vl, MethodDeclList ml)


VarDecl(Type t, Identifier i)
MethodDecl(Type t, Identifier I, FormalList fl,
           VariableDeclList vl, StatementList sl, Exp e)
Formal(Type t, Identifier i)
```

Abstract Syntax for MiniJava (II)

```
abstract class Type
IntArrayType()
BooleanType()
IntegerType()
IndentifierType(String s)


abstract class Statement
Block(StatementList sl)
If(Exp e, Statement s1, Statement s2)
While(Exp e, Statement s)
Print(Exp e)
Assign(Identifier i, Exp e)
ArrayAssign(Identifier i, Exp e1, Exp e2)
```

Abstract Syntax for MiniJava (III)

```
abstract class Exp
And(Exp e1, Exp e2)              LessThan(Exp e1, Exp e2)
Plus(Exp e1, Exp e2)             Minus(Exp e1, Exp e2)
Times(Exp e1, Exp e2)            Not(Exp e)
ArrayLookup(Exp e1, Exp e2)      ArrayLength(Exp e)
Call(Exp e, Identifier i, ExpList el)
IntergerLiteral(int i)
True()                           False()
IdentifierExp(String s)
This()
NewArray(Exp e)                  NewObject(Identifier i)

Identifier(String s) holds identifiers

--list classes:
ClassDecList() ExpList() FormalList() MethodDeclList()
StatementLIst() VarDeclList()
```

Syntax Tree Nodes - Details

```
package syntaxtree;
import visitor.Visitor;

public class Program {
  public MainClass m;
  public ClassDeclList cl;

  public Program(MainClass am, ClassDeclList acl)
   {
    m=am; cl=acl;
  }

  public void accept(Visitor v) {
    v.visit(this);
  }
}
```

StatementList.java (all lists are like this)

```
package syntaxtree;
import java.util.Vector;

public class StatementList {
   private Vector list;
   public StatementList() {
      list = new Vector();
   }
   public void addElement(Statement n) {
      list.addElement(n);
   }
   public Statement elementAt(int i)  {
      return (Statement)list.elementAt(i);
   }
   public int size() {
      return list.size();
   }
}
```

Building AST lists in JavaCC

```
ExpList ExpressionList() :
{ Exp e1,e2;
  ExpList el = new ExpList();
}
{
  e1=Expression()              { el.addElement(e1); }
  (  e2=ExpressionRest()       { el.addElement(e2); }  )*
  { return el; }
}

Exp ExpressionRest() :
{ Exp e; }
{
  "," e=Expression()
  { return e; }
}
```

init

add

add

x = y.m(1,4+5)

Statement → AssignmentStatement

AssignmentStatement → Identfier$_1$ "=" Expression

Identifier$_1$ → <IDENTIFIER>

Expression → Expression$_1$ "." Identifier$_2$ "(" (ExpList)? ")"

Expression$_1$ → IdentifierExp

IdentifierExp → <IDENTIFIER>

Identifier$_2$ → <IDENTIFIER>

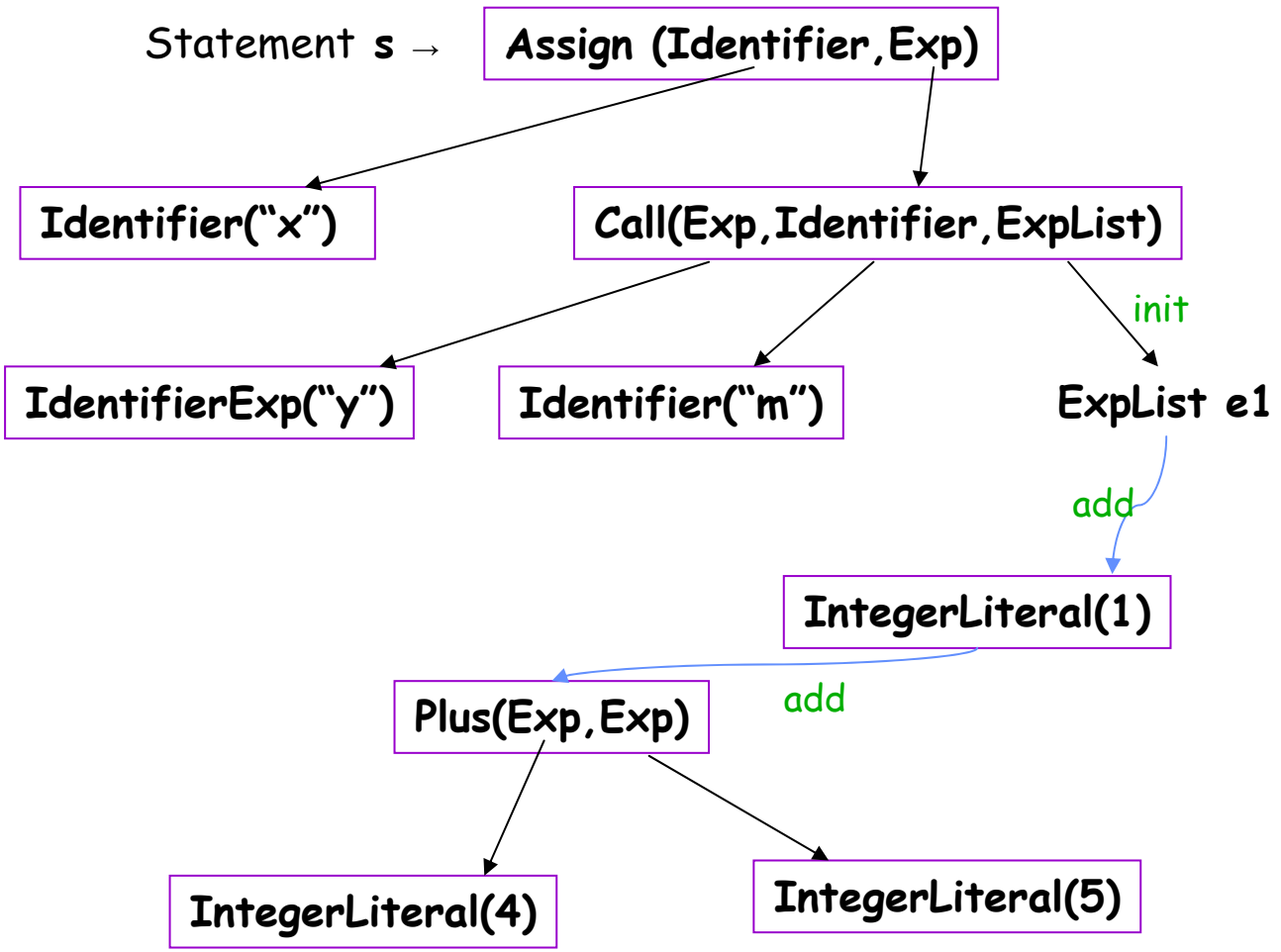
ExpList → Expression$_2$ ("," Expression$_3$)*

Expression$_2$ → <INTEGER_LITERAL>

Expression$_3$ → PlusExp → Expression "+" Expression

→ <INTEGER_LITERAL> "+" <INTEGER_LITERAL>

AST


Statement s →
Assign (Identifier,Exp)
Identifier("x")
Call(Exp,Identifier,ExpList)
IdentifierExp("y")
Identifier("m")
init
ExpList e1
add
IntegerLiteral(1)
add
Plus(Exp,Exp)
IntegerLiteral(4)
IntegerLiteral(5)

MiniJava : Grammar(I) & JavaCC

Program → *MainClass ClassDecl **

Program(MainClass, ClassDeclList)

```
Program Goal() :
{ MainClass m;
  ClassDeclList cl = new ClassDeclList();
  ClassDecl c;
}
{ m = MainClass() (c = ClassDecl() {cl.addElement(c);})*
  <EOF> {return new Program(m,cl); }
}
```

MiniJava : Grammar(II)

MainClass → **class** *id* { **public static void main** (**String** [] *id*)
{ *Statement* } }

MainClass(Identifier, Identifier, Statement)

ClassDecl → **class** *id* { *VarDecl* * *MethodDecl* * }
→ **class** *id* **extends** *id* { *VarDecl** *MethodDecl* * }

ClassDeclSimple(…), ClassDecExtends(…)

VarDecl → *Type id ;*

VarDecl(Type, Identifier)

MethodDecl → **public** *Type id* (*FormalList*)
{ *VarDecl* * *Statement** **return** *Exp* ; }

MethodDecl(Type,Identifier,FormalList,VarDeclList,StatementList,Exp)

MiniJava : Grammar(III)

FormalList → *Type id FormalRest* *

→

FormalList() :- Formal(type,id), …

FormalRest → , *Type id*

Formal()

Type → **int** []

→ **boolean**

→ **int**

→ *id*

Type(), ArrayType(), BooleanType(), IntegerType(), IdentifierType()

MiniJava : Grammar(IV)

Statement → { *Statement* * }
→ **if** (*Exp*) *Statement* **else** *Statement*
→ **while** (*Exp*) *Statement*
→ **System.out.println** (*Exp*) ;
→ *id* **=** *Exp* **;**
→ *id* [*Exp*] **=** *Exp* **;**

Statement(), Block(), If(), While(), Print(), Assign (), ArrayAssign()

ExpList → *Exp* *ExpRest* *
→

ExpRest → **,** *Exp*

MiniJava : Grammar(V)

Exp	→	*Exp op Exp*	**&& < + - ***
	→	*Exp* **[** *Exp* **]**	
	→	*Exp* **.** **length**	
	→	*Exp* **.** *Id* **(** *ExpList* **)**	
	→	*INTEGER_LITERAL*	
	→	**true**	
	→	**false**	
	→	*id*	
	→	**this**	
	→	**new int [** *Exp* **]**	
	→	**new** *id* **( )**	
	→	**!** *Exp*	
	→	**(** *Exp* **)**	

MainClass, ClassDecl : in JavaCC

```
MainClass MainClass() :
{ Identifier i1,i2;
  Statement s; }
{
  "class" i1=Identifier() "{"
    "public" "static" "void" "main" "(" "String" "[" "]"
       i2=Identifier() ")" "{" s=Statement() "}" "}"
  { return new MainClass(i1,i2,s); }
}

ClassDecl ClassDeclaration() :
{ ClassDecl c; }
{
  (  LOOKAHEAD(3)
     c=ClassDeclarationSimple()
   | c=ClassDeclarationExtends()
   )
  { return c; }
}
```

FormalList, FormalRest : in JavaCC

```
FormalList FormalParameterList() :
{ FormalList fl = new FormalList();   Formal f;
}
{  f=FormalParameter() { fl.addElement(f); }
  ( f=FormalParameterRest() { fl.addElement(f); } )*
  { return fl; }
}
Formal FormalParameter() :
{ Type t; Identifier i;
}
{  t=Type() i=Identifier()
  { return new Formal(t,i); }
}
Formal FormalParameterRest() :
{ Formal f; }
{  "," f=FormalParameter()
  { return f; }
}
```

FormalList → *Type id FormalRest**

→

FormalRest → , *Type id*

What you should do now…

- read and digest chapter 4
- read and understand about MiniJava and its abstract syntax trees and visitors
- if you haven't already, embark on the second part of the coursework