Java Fundamentals 2

- More B/F - Java program shape
- More on fundamentals
  - Variable declarations, constants, expressions, assignment statements
- NIM as an example Java program

What form is a Java program?

```
<simple-program> →
   class <classname> { <main-method> }
<main-method> →
  public static void main(String [ ] <argname>) {
   <declarations>  <statements >  }
<statements> → <statement>
<declarations> → <declaration>
```

A Very Simple Java Program

```
class DoNothing {
  public static void main (String [ ] args)
  {   }
}
```

Bishop, p 25

Is this a <simple-program> ?

NO!

How can we tell?

Try to match to (or produce from) the rule for <simple-program>,

class <classname> { <main-method> }

A Very Simple Java Program

```
class DoNothing {
```

class <classname> {

```
  public static void main (String [ ]
  args)
```

public static void main (String [ ] <argname>)

but

```
{    }
```

{<declarations> <statements > } **doesn't match**

There is part of <main-method> missing!

Another Rule for Program

<simpler-program> →

class <classname> { <simple-main-method> }
| <simple-program>

<simple-main-method> →

public static void main(String [ ] <argname>)
{ }

DoNothing is a <simpler-program> but not a <simple-program>.

Recursive BNF Rules, Revisited

<statement> → <var> = <expr>;

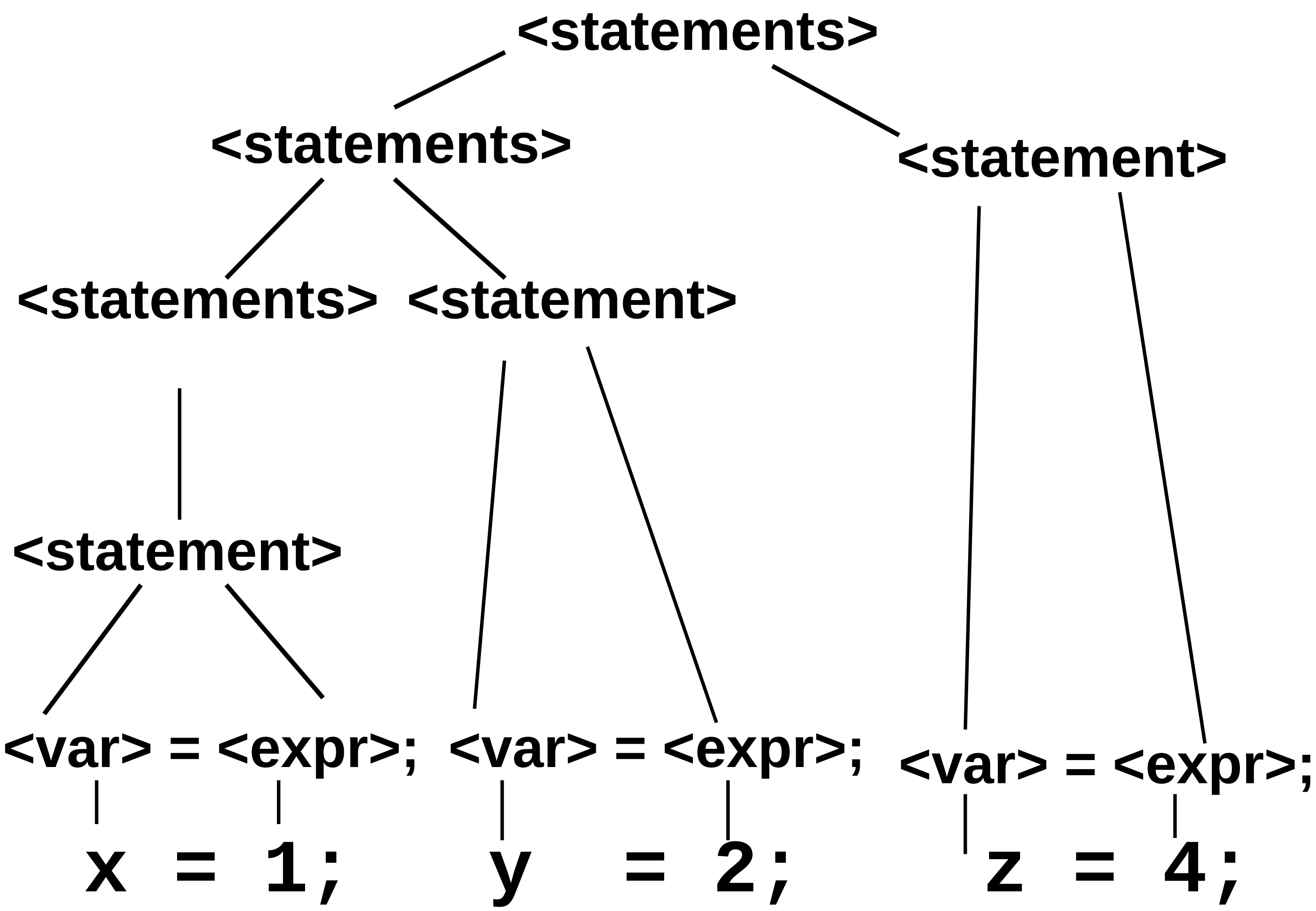
How can we get three assignment statements in a sequence?

```
x = 1; y = 2; z = 4;
```

Need a recursive rule to get repetition in a construc t.

<statements> → <statement> |
<statements> <statement>

Parse Tree for < statements >



<statements>

<statements> <statement>

<statements> <statement>

<statement>

<var> = <expr>; <var> = <expr>; <var> = <expr>;

```
x = 1;    y  = 2;    z = 4;
```

Parse Tree

- Rules for formation
  - Parent node is nonterminal in a rule
  - Children of parent from left to right are right-hand-side of the rule
- Each subtree in tree is a rule
- Leaves of the tree from left to right are the sequence of terminal symbols being recognized

Corresponding Leftmost (Canonical) Derivation

<statements> →<statements> <statement>

→<statements> <statement> <statement>

→<statement> <statement><statement>

→ <var> = <expr>; <statement><statement>

→ **x** = <expr>; <statement><statement>

→ **x** = **1** ; <var> = <expr>; <statement>

→ **x** = **1** ; **y** = <expr>; <statement>

→ **x** = **1** ; **y** = **2** ; <statement>

→ **x** = **1** ; **y** = **2** ; <var> = <expr>;

→ **x** = **1** ; **y** = **2** ; **z** = <expr>;

→ **x** = **1** ; **y** = **2** ; **z** = **4** ;

Method Invocation

- **Parameters are incoming values for use by the method, in addition to the values associated with the object on which the method is called**
- **Without parameters**

  <object> . <method-name>( );

  Y101 . landing(); // Y101 refers to a plane object
- **With parameters**

  <object>   . <method-name> (<params>);

  pilot . assignToFlight(101); // pilot refers to a crew
  // member object

Variable Declarations

<variable-dcl> →<type> <var>;

<variable-dcl> →<type> <varlist>;

<variable-dcl> →<type> <var> = <value>;

<varlist> →<var>

<varlist> →<varlist>, <var>

```
int   x, y; // pos or neg number or 0
double  sum; // real values
boolean win = false; // true or false
```

Constants

<constant> → static final <type> <name> = <value>;

```
static final int year = 1997;
static final boolean T = true;
```

Constants are variables whose values cannot change, once set. Used for mnemonic names for significant quantities.

Expressions

- Familiar arithmetic operators available
  - + - * /
  - % (modulus: remainder after integer divide)

  `20 % 2` is `0;  21 % 2` is `1`

- Need to use parentheses to override precedence

  `2+3*4` yields `14`

  `(2+3)*4` yields `20`

  `* /` higher precedence than `+ -`

  when in doubt, use parentheses!

Assignment Statement

- Variables can be associated with values

<assign-stmt> → <var> = <expr>;

where <expr> is an expression.

- Execution of an assignment statement causes expression evaluation and binding of the resulting value to the variable.
- The type of the variable must match the type of the expression. Some conversions done automatically (e.g., `int` to `double`)

Barbara G. Ryder © Spring 1998

Output Statements

<output-stmt> →System.out.println(<items>);

<output-stmt> →System.out.println( );

<output-stmt> →System.out.print (<items>);

```
System.out.println (“Hello world!”);
```

causes the string “`Hello world!`” to be printed on your output window

```
System.out.println();
```

causes a blank line to be printed on your output window

Output Statements

<items> → <string-value>

<string-value> →” <characters> “

<string-value> →

    <string-value> + <string-value>

where <characters> can be any sequence of individual symbols and + is concatenation.

`“abc”   “1” “1+2” “a3”`

`“abc”+”def”` yields `“abcdef”`

`“1” + “2”` yields `“12”` not `3` nor `“3”`

Using Strings for Output

- All types in Java must be converted to String type in order to be output
- *toString()* method defined by default for the primitive types
- Concatenation operator +
- Examples:

```
System.out.println(“abc” + “def”
      + “1”);
System.out.println(“abcd”+”ef1”);
```

both print as `abcdef1`

Example 1

```
int a, b, c;
int x = 5;
a = 1;
System.out.println(b);//ints initially 0
b = 2; c = 3;
System.out.println(“a+b/c  is “+ (a+b)/c);
System.out.println(“a equals b is “+ a==b);
System.out.println(x);
```

```
 0
 a+b/c is 1
 a equals b is false
 5
```

Example 2

```
System.out.println(1+2*3);
System.out.print(8%2 +“ “);
System.out.println(2*5 + “ == 10”);
System.out.println(5 + ”is my age”);
System.out.println(1+2+ ”==“ +1+2);
System.out.println(“\n”);
```

```
7
0 10 == 10      Because print stays on same line
5is my age      Note need for blank before “is”
3==12           Unexpected result!
2  blank lines printed
```

Operator Overloading

Why the unexpected result from

`System.out.println(1+2+ ”==“ +1+2)` ?

Evaluate the expression.

`1+2` yields `3`

`3 + “==“` yields `“3==“`

`“3==“ + 1` yields `“3==1”`

`“3==1” + 2` yields `“3==12”`

Problem: + has different meanings for different typed operands: overloading

Sometimes it’s addition, sometimes concatenation.

How Output Works?

- *System* is a built-in class in Java
- *out* is a special variable associated with this built-in class, a class variable which is referred to by <class-name> . <var>
- There is only 1 instance of a class variable shared by all objects created in that class; different from instance variables
- *println* and *print* are methods which can be invoked on *out* to cause their string parameter to be printed on the screen

NIM Program Specification

```
NimState class
```

- **Attributes**

```
int count \\ for initialization
```

- **Methods**

```
\\creates new NimState object with 1 less
\\stone on its pile
NimState removeOne()\\ private
NimState removeTwo()\\ private
boolean win()\\ returns true if initial
\\state is a winner for moving player,
\\else false
int move()\\ chooses the move to make
```

Algorithm

- Initialize Nim game with pile of stones
- If first player to move can win by making a move, then make that move
  - Play game forward, exploring all possibilities for moves, to see which move is a possible win, if any
- Else, first player must concede loss, but play the game out

How to Accomplish Algorithm

- Need to keep number of stones in instance variable associated with Nim game object
- Need method that can explore all possible outcomes of the current game - recursion
- Need method that makes a move to advance the game towards conclusion
- Need main method to do printing and to call other methods

Things to Notice

- `import`- inclusion of Java i/o package
- Class definition, instance variables and methods
  - Methods with and without parameters
  - Method calls
  - Object creation with `new`
  - Functions which return objects
  - Constructor method
  - Main method

Things to Notice

- Use of i/o statements
- Statements governing conditional control flow to allow choice of next step in algorithms
  - switch, if
- Use of modifiers on attributes and methods
  - private, public

Object Creation

<create-obj> → *[* <modifier>*]* <classname>

<objectname> = new <classname>

[**(<params>)** ***]*** **;**

use of *[ ]* implies enclosed construct is optional

```
public NimState board =
        new NimState(12);
```