

CS 460

Programming Languages

Fall 2008

Dr. Watts

(Week 13)



Java

- File must have same name as public class
- With .java extension
- `javac file.java` creates `file.class`
- `java file` interprets byte code in `file.class`
- Missing line in file `Tutorial3/RList.java` (page 46)

```
for (int i = 0; i < size; i++)
{
    int left = ulX + (2*i+1) * deltaX;
    int right = left + deltaX;
    int top = lrY - list[i] * deltaY;
    int bottom = lrY;
    Color c = new Color (list[i] * deltaG, list[i] * deltaR,
        list[i] * deltaB);
    g2.setPaint (c);
    g2.fillRect (left, top, right-left, bottom-top);
}
```



Java Tutorial 3



- Two Buttons
 - Sort
 - Call complete (or rest) of sorting routine
 - Multiple calls to “SortPass”
 - Inserting a `sleep()` call requires creating a thread
 - `Thread.sleep(4000);` causes thread to pause for 4 seconds
 - Pass
 - Call one pass of your sort
 - Single call to “SortPass”

Stack-based evaluation



- Example 8 input:

```
A = 5;  
B = 7;  
C = 3 * (A + 2) * B;  
D = 7;  
D += 4 * (B + 2.3) * (C + A);
```



Stack-based evaluation

- Example 8 output:

```
[1] A = 5;
[2] B = 7;
[3] C = 3 * (A + 2) * B;
[4] D = 7;
[5] D += 4 * (B + 2.3) * (C + A);
0 errors found.
A      5
B      7
C     147
D    5661.39990
```



Stack-based evaluation

- Example 8 C version:

```
#include <stdio.h>

int main ()
{
    int A, B, C;
    float D;
    A = 5;
    B = 7;
    C = 3 * (A + 2) * B;
    D = 7;
    D += 4 * (B + 2.3) * (C + A);
    printf ("A\t%d\n", A);
    printf ("B\t%d\n", B);
    printf ("C\t%d\n", C);
    printf ("D\t%.5f\n", D);
}
```



Annotated Grammar

- Symbol Creation
 17. `<var>` -> IDENT [create-symbol]
 18. `<var>` -> NUMLIT [create-symbol]
- Symbol Output
 1. `<program>` -> `<stmt>` SEMI `<more_stmts>` EOFT [print-symbols]
- Statement Queue Manipulation
 1. `<program>` -> `<stmt>` [evaluate-queue] SEMI `<more_stmts>` EOFT
 2. `<more_stmts>` -> `<stmt>` [evaluate-queue] SEMI `<more_stmts>`
 6. `<term>` -> LPAREN `<stmt>` [evaluate-queue] RPAREN
 9. `<uoppre>` -> PLUS [add-symbol-to-queue]
 10. `<uoppre>` -> MINUS [add-symbol-to-queue]
 17. `<var>` -> IDENT [create-symbol] [add-symbol-to-queue]
 18. `<var>` -> NUMLIT [create-symbol] [add-symbol-to-queue]
 19. `<post>` -> PLUSPLUS [add-symbol-to-queue]
 20. `<post>` -> MINUSMINUS [add-symbol-to-queue]
 25. `<binop>` -> PLUS [add-symbol-to-queue]
 26. `<binop>` -> MINUS [add-symbol-to-queue]



Statement Level Control Structures

- Allow the program to select among alternate control flow paths
- Control statements
 - Selective
 - Iterative
- Control structure
 - Control statement
 - The collection of statements whose execution is controlled by the control statement



Selection Statements

- One-way selection statements
 - if <conditional_expression> <then_clause>
 - IF A < B THEN 20
 - if a < b then
 - ...
 - endif
 - if (a < b)
 - {
 - ...
 - }



Selection Statements

- Two-way selection statements
 - if <conditional_expression> <then_clause> <else_clause>
 - if a < b then
 - ...
 - else
 - ...
 - endif
 - if (a < b)
 - {
 - ...
 - }
 - else
 - {
 - ...
 - }
 - Parsing and the “else” problem



Selection Statements

- Multi-way selection statements
 - FORTRAN Arithmetic IF
 - IF VALUE THEN 10, 20, 30
 - BASIC
 - ON VALUE GOTO 10, 20, , 30, 40
 - BASIC
 - ON VALUE GOSUB 10, 20, , 30, 40



Selection Statements

- Multi-way selection statements
 - C/C++ switch statement
 - switch (expression)
 - {
 - case lit1: statement;
 - statement;
 - break;
 - case lit2: statement;
 - statement;
 - case lit3: case lit4:
 - statement;
 - statement;
 - break;
 - default: statement;
 - statement;



Selection Statements

- Multi-way selection statements
 - Pascal case statement
 - case expression of

```
{
    lit1: statement;
    lit2: begin
        statement;
        statement;
    end;
    case lit3, lit4:
        statement;
    case lit5 .. lit6:
        statement;
    else statement;
}
```



Iterative Data Structures

- FORTRAN do loop
 - Example 1

```
DO 30 I = 1,10,2
Other FORTRAN statements
30 CONTINUE
```
 - Example 2

```
DO 40 J = 10,1,-2
Other FORTRAN statements
40 CONTINUE
```
 - Issues
 - Pre-calculation of number of iterations
 - Will execute same number of iterations even if loop-control-variable (I,J) is modified
 - Good for parallelization

Iterative Data Structures



- BASIC for-next loop

- Example 1

```
10 FOR I = 1 TO 10 STEP 2
20 Other BASIC statements
30 NEXT I
```

- Example 2

```
40 FOR J = 10 TO 1 STEP -2
50 Other BASIC statements
60 NEXT J
```

- Issues

- Loop-control-variable in FOR and NEXT must match
 - Can modify loop-control-variable to change loop behavior

Iterative Data Structures



- COBOL PERFORM loop

- Example 1

```
PERFORM paragraph-name
      VARYING I FROM 1 BY 2 UNTIL I > 10.
```

- Example 2

```
PERFORM paragraph-name
      VARYING I FROM 10 BY -2 UNTIL I < 1.
```

- Issues

- Loop-control-variable is global
 - Can modify loop-control-variable to change loop behavior
 - Good for structured programming



Iterative Data Structures

- Pascal for loop

- Example 1

```
for i := 1 to 10 do  
    Pascal statement
```

- Example 2

```
for i := 10 downto 1 do  
begin  
    Pascal statement;  
    Pascal statement  
end;
```

- Issues

- Step is always 1 or -1
 - Body is a single Pascal statement
 - Must use begin-end compound statement to expand body to more than one statement



Iterative Data Structures

- C/C++ for loop

- Format

```
for (initialization; continuation condition;  
    transition)  
    C/C++ statement;
```

- Example 1

```
for (int i = 1; i <= 10; i += 2)  
    C/C++ statement;
```

- Example 2

```
for (int i = 10; i >= 1; i -= 2)  
{  
    C/C++ statement;  
    C/C++ statement;  
}
```



Iterative Data Structures

- C/C++ for loop

- Example 3

```
for (cin >> var; var != terminal; cin >> var)
    C/C++ statement;
```

- Issues

- Flexibility
 - Body is a single C/C++ statement
 - Must use {} block statement to expand body to more than one statement



Nested Loops

- All appropriate iterations of inner loop executed for each iteration of outer loop

- BASIC FOR loop

```
10 FOR I = 1 TO 10 STEP 2
15 FOR J = 1 TO I
20 Other BASIC statements
25 NEXT J
30 NEXT I
```

- COBOL PERFORM loop

```
PERFORM paragraph-name
    VARYING I FROM 10 BY -2 UNTIL I < 1
    AFTER J FROM 1 BY 1 UNTIL J = I.
```



Nested Loops

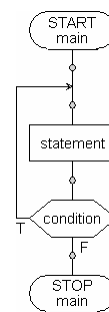
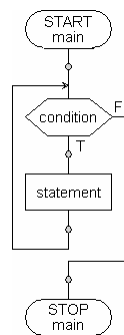
- Multiple nesting
- C/C++ for loop

```
for (cin >> var; var != terminal; cin >> var)
  for (int i = 1; i <= var; i++)
    for (int j = i, k = 1; j >= 1; j--, k++)
    {
        C/C++ statement;
        C/C++ statement;
    }
```



Test at Top vs. Test at Bottom

- Test at top
 - Condition tested *before* executing loop body
- Test at bottom
 - Condition tested *after* executing loop body



Test at Bottom loop formats



- C/C++ do-while
 - Loop body executed *while* the condition is true
- Pascal repeat-until
 - Loop body executed *until* the condition is true

```
do
{
  C/C++ statement;
  C/C++ statement;
} while (x > y);
```

```
repeat
  Pascal statement;
  Pascal statement
until x > y;
```

Test in middle loop formats



- C/C++ user controlled exit
 - Break
 - Continue
 - Exit
 - Return

```
while (true)
{
  C/C++ statement;
  C/C++ statement;
  if (expression)
    break;
  C/C++ statement;
  C/C++ statement;
}
```

Loop Syntax Notation



- COBOL

```

PERFORM [ 1stProc [ {THRU} EndProc ] ] [ WITH TEST {BEFORE} ]
        [ {THROUGH} ]
        VARYING {Counter1#i} FROM {StartValue#il}
              {IndexName1} {IndexName2}
        BY StepValue#il UNTIL Condition1
        [ AFTER {Counter2#i} FROM {StartValue2#il} ] ...
          [ {IndexName3} {IndexName4} ]
          BY StepValue2#il UNTIL Condition2
        [ StatementBlock END - PERFORM ]
    
```

Control Structure Implementation



- IF

- Source code


```

if (A > B)
    C := A + B;
            
```
- Intermediate code


```

>    $$2    A    B
CBRF  $$1    $$2  []
+    $$3    A    B
:=    C      $$3  []
LABEL $$1    []  []
            
```

- IF ELSE

- Source code


```

if (A > B)
    C := A - B;
else
    C := B - A;
            
```
- Intermediate code


```

>    $$2    A    B
CBRF  $$1    $$2  []
-    $$3    A    B
:=    C      $$3  []
JMP   $$4    []  []
LABEL $$1    []  []
-    $$5    B    A
:=    C      $$5  []
LABEL $$4    []  []
            
```

Control Structure Implementation



- WHILE

- Test at top
 - Source code
- ```
while (i < 3)
 i := i + 1;
```

- Intermediate code

```
LABEL $$1 [] []
< $$3 i 3
CBRF $$2 $$3 [] []
+ $$4 i 1
:= i $$4 [] []
JMP $$1 [] []
LABEL $$2 [] []
```

- REPEAT

- Test at bottom
  - Source code
- ```
repeat
    i := i - 1;
until (i < 0);
```

- Intermediate code

```
LABEL $$1 [] []
$$2 i 1
:= i $$2 [] []
< $$3 i 0
CBRF $$1 $$3 [] []
```

Control Structure Implementation



- FOR loops

- Test at top
 - Source code
- ```
for (i := 0; i < 3; i := i + 1)
 s := s + i;
for (i := 0; i < 3; i := i + 1)
 for (j := 1; j < 4; j := j + 1)
 s := s + i * j;
```

- Intermediate code

```
:= i 0 [] []
LABEL $$1 [] []
< $$3 i 3
CBRF $$2 $$3 [] []
+ $$5 s i
:= s $$5 [] []
+ $$4 i 1
:= i $$4 [] []
JMP $$1 [] []
LABEL $$2 [] []

:= i 0 [] []
LABEL $$6 [] []
< $$8 i 3
CBRF $$7 $$8 [] []
:= j 1 [] []
LABEL $$10 [] []
< $$12 j 4
CBRF $$11 $$12 [] []
* $$14 i j
+ $$15 s $$14
:= s $$15 [] []
+ $$13 j 1
:= j $$13 [] []
JMP $$10 [] []
LABEL $$11 [] []
+ $$9 i 1
:= i $$9 [] []
JMP $$6 [] []
LABEL $$7 [] []
```

# Control Structure Implementation



- SWITCH

- Source code

```
switch (a)
{
 case 1: x = y + 1;
 break;
 case 2: x = 2 * y;
 break;
 case 3: x = y - 4;
 break;
 case 4: x = 16 / y;
 break;
 default:
 x = 0;
}
```

- Implementation

- Large if-else structure
  - Jump table
    - If selection values in small range
    - Table containing label addresses