Control Structures

(Deitel chapter 4,5)

Plan

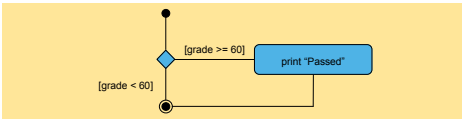
- Control Structures
- `if` Single-Selection Statement
- `if else` Selection Statement
- `while` Repetition Statement
- `for` Repetition Statement
- `do…while` Repetition Statement
- `switch` Multiple-Selection Statement
- `break` and `continue` Statements
- Structured Programming Summary

Control Structures

- Java has a sequence structure "built-in"
- Java provides three selection structures
  - `if`
  - `If…else`
  - `switch`
- Java provides three repetition structures
  - `while`
  - `do…while`
  - `do`
- Each of these words is a Java keyword

`if` Single-Selection Statement

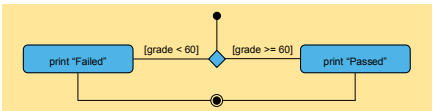
- Single-entry/single-exit control structure
- Perform action only when condition is `true`
- Action/decision programming model



`if` single-selections statement activity diagram.

`if…else` Selection Statement

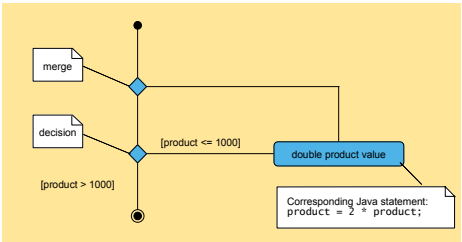
- Perform action only when condition is `true`
- Perform different specified action when condition is `false`
- Conditional operator (`?:`)
- `System.out.println( grade >= 60 ? "Passed" : "Failed" );`



`if…else` double-selections statement activity diagram.

`while` Repetition Statement

- Repeat action while condition remains `true`



`while` repetition statement activity diagram.

Formulating Algorithms

- Counter-controlled repetition
  - Variable that controls number of times set of statements executes
- Sentinel-controlled repetition
  - User enters sentinel value (`-1`) to end repetition

Outline

Average1.java

gradeCounter

Line 21

```
1  // Fig. 4.7: Average1.java
2  // Class-average program with counter-controlled repetition.
3  import javax.swing.JOptionPane;
4
5  public class Average1 {
6
7     public static void main( String args[] )
8     {
9        int total;        // sum of grades input by user
10       int gradeCounter; // number of grade to be entered next
11       int grade;        // grade value
12       int average;      // average of gra
13
14       String gradeString; // grade typed by
15
16       // initialization phase
17       total = 0;        // initialize total
18       gradeCounter = 1; // initialize loop counter
19
20       // processing phase
21       while ( gradeCounter <= 10 ) {  // loop 10 times
22
23          // prompt for input and read grade from user
24          gradeString = JOptionPane.showInputDialog(
25             "Enter integer grade: " );
26
27          // convert gradeString to int
28          grade = Integer.parseInt( gradeString );
29
```

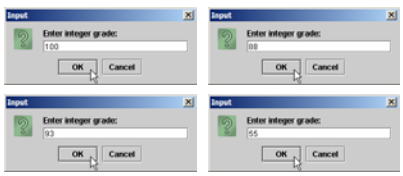
Declare variables; `gradeCounter` is the counter

Continue looping as long as `gradeCounter` is less than or equal to 10

Outline

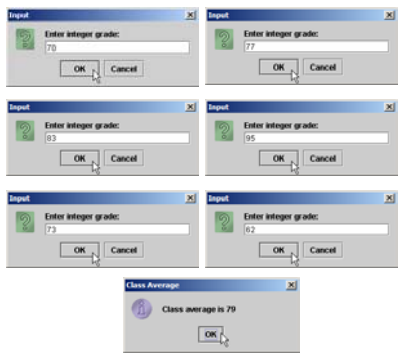
Average1.java

```
30          total = total + grade;           // add grade to total
31          gradeCounter = gradeCounter + 1;  // increment counter
32
33       } // end while
34
35       // termination phase
36       average = total / 10;  // integer division
37
38       // display average of exam grades
39       JOptionPane.showMessageDialog( null, "Class average is " + average,
40          "Class Average", JOptionPane.INFORMATION_MESSAGE );
41
42       System.exit( 0 );  // terminate the program
43
44    } // end main
45
46 } // end class Average1
```



Outline

Average1.java



Compound Assignment Operators

- Assignment Operators
  - Abbreviate assignment expressions
  - Any statement of form
    - *variable* = *variable operator expression*;
  - Can be written as
    - *variable operator*= *expression*;
  - e.g., addition assignment operator +=
    - `c = c + 3`
  - can be written as
    - `c += 3`

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`c = c + 7`	10 to `c`
`-=`	`d -= 4`	`d = d - 4`	1 to `d`
`*=`	`e *= 5`	`e = e * 5`	20 to `e`
`/=`	`f /= 3`	`f = f / 3`	2 to `f`
`%=`	`g %= 9`	`g = g % 9`	3 to `g`

Fig. 4.12 Arithmetic assignment operators.

Increment and Decrement Operators

- Unary increment operator (++)
  - Increment variable's value by 1
- Unary decrement operator (--)
  - Decrement variable's value by 1
- Preincrement / predecrement operator
- Post-increment / post-decrement operator

Operator	Called	Sample expression	Explanation
++	preincrement	++a	Increment a by 1, then use the new value of a in the expression in which a resides.
++	postincrement	a++	Use the current value of a in the expression in which a resides, then increment a by 1.
--	predecrement	--b	Decrement b by 1, then use the new value of b in the expression in which b resides.
--	postdecrement	b--	Use the current value of b in the expression in which b resides, then decrement b by 1.

Fig. 4.13 The increment and decrement operators.

Outline

Increment.java

Line 13 postincrement

Line 21 preincrement

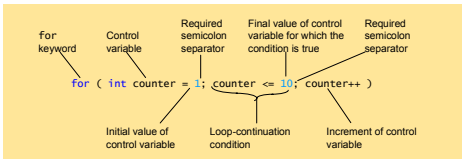
```
1  // Fig. 4.14: Increment.java
2  // Preincrementing and postincrementing operators.
3
4  public class Increment {
5
6     public static void main( String args[] )
7     {
8        int c;
9
10       // demonstrate postincrement
11       c = 5;                      // assign 5 to c
12       System.out.println( c );    // print
13       System.out.println( c++ ); // print
14       System.out.println( c );    // print 6
15
16       System.out.println();       // skip a line
17
18       // demonstrate preincrement
19       c = 5;                      // assign 5 to c
20       System.out.println( c );    // print 5
21       System.out.println( ++c ); // preincrement then print 6
22       System.out.println( c );    // print 6
23
24    } // end main
25
26 } // end class Increment
5
5
6

5
6
6
```

Line 13 postincrements c

Line 21 preincrements c

for Repetition Statement

- Handles counter-controlled-repetition details



for Repetition Structure (cont.)

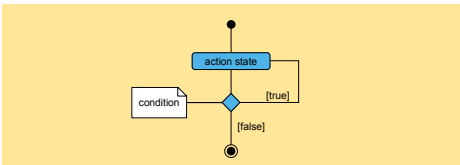
```
for ( initialization; loopContinuationCondition; increment )
   statement;
```

can usually be rewritten as:

```
initialization;
while ( loopContinuationCondition ) {
   statement;
   increment;
}
```


do…while Repetition Statement

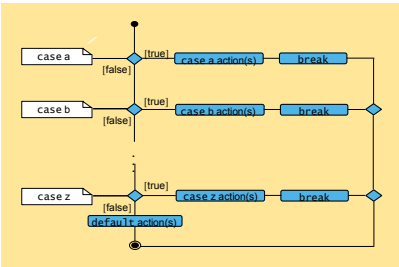
- do…while structure
  - Similar to while structure
  - Tests loop-continuation after performing body of loop
    - i.e., loop body always executes at least once



do…while **repetition statement activity diagram.**

switch Multiple-Selection Statement

- switch statement
  - Used for multiple selections



switch multiple-selection statement activity diagram with break statements.

Outline

```
32          switch ( choice ) {  // determine shape to draw
33
34             case 1:  // draw a line
35                g.drawLine( 10, 10, 250, 10 + i
36                break;  // done processing case
37
38             case 2:  // draw a rectangle
39                g.drawRect( 10 + i * 10, 10 + i * 10,
40                   50 + i * 10, 50 + i * 10 );
41                break;  // done processing case
42
43             case 3:  // draw an oval
44                g.drawOval( 10 + i * 10, 10 + i * 10,
45                   50 + i * 10, 50 + i * 10 );
46                break;  // done processing case
47
48             default: // draw string indicating invalid value entered
49                g.drawString( "Invalid value entered",
50                   10, 20 + i * 15 );
51
52          } // end switch
53
54       } // end for
55
56    } // end method paint
57
58 } // end class SwitchTest
```

user input (choice) is *controlling expression* which case label to execute, depending on controlling expression

default case for invalid entries

chTest.java

Line 32: controlling expression

Line 32: switch statement

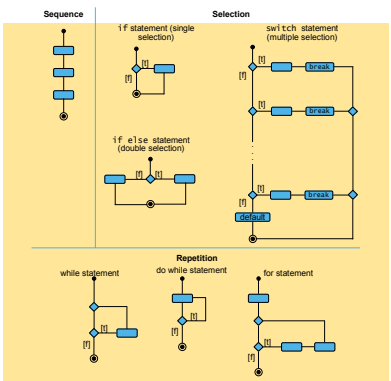
Line 48

break and continue Statements

- break/continue
  - Alter flow of control
- break statement
  - Causes immediate exit from control structure
    - Used in while, for, do…while or switch statements
- continue statement
  - Skips remaining statements in loop body
  - Proceeds to next iteration
    - Used in while, for or do…while statements

Structured Programming Summary

- Sequence structure
  - "built-in" to Java
- Selection structure
  - if, if…else and switch
- Repetition structure
  - while, do…while and for



Java's single-entry/single-exit sequence, selection and repetition statements.