

2c

Control Statements: Part 2



OBJECTIVES

In this lecture you will learn:

- The essentials of counter-controlled repetition.
- To use the `for` and `do...while` repetition statements to execute statements in a program repeatedly.
- To understand multiple selection using the `switch` selection statement.
- To use the `break` and `continue` program control statements to alter the flow of control.
- To use the logical operators to form complex conditional expressions in control statements.



Outline

- 2c.1 Introduction**
- 2c.2 Essentials of Counter-Controlled Repetition**
- 2c.3 for Repetition Statement**
- 2c.4 Examples Using the for Statement**
- 2c.5 do...while Repetition Statement**
- 2c.6 switch Multiple-Selection Statement**
- 2c.7 break and continue Statements**
- 2c.8 Logical Operators**
- 2c.9 Structured Programming Summary**
- 2c.10 GUI and Graphics Case Study: Using Dialog Boxes with JavaFX**
- 2c.11 GUI and Graphics Case Study: Drawing Rectangles and Ovals with JavaFX**



2c.1 Introduction

Continue structured-programming discussion

- Introduce Java's remaining control structures
 - for, do...while, switch



2c.2 Essentials of Counter-Controlled Repetition

Counter-controlled repetition requires:

- **Control variable (loop counter)**
- **Initial value of the control variable**
- **Increment/decrement of control variable through each loop**
- **Loop-continuation condition that tests for the final value of the control variable**



Outline

whileCounter.java

```

1 // Fig. 5.1: whileCounter.java
2 // Counter-controlled repetition with the while repetition statement.
3
4 public class whileCounter
5 {
6     public static void main( String args[] )
7     {
8         int counter = 1; // declare and initialize control variable
9
10        while ( counter <= 10 ) // loop-continuation condition
11        {
12            System.out.printf( "%d ", counter );
13            ++counter; // increment control variable
14        } // end while
15
16        System.out.println(); // output a newline
17    } // end main
18 } // end class whileCounter

```

Control-variable name is **counter**

Control-variable initial value is **1**

Condition tests for
counter's final value

Increment for **counter**

1 2 3 4 5 6 7 8 9 10



Error-Prevention Tip

Control counting loops with integers.



Good Programming Practice

Place blank lines above and below repetition and selection control statements, and indent the statement bodies to enhance readability.



2c.3 for Repetition Statement

Handles counter-controlled-repetition details



Outline

ForCounter.java

Line 10
int counter = 1;

Line 10
counter <= 10;

Line 10
counter++;

```

1 // Fig. 5.2: ForCounter.java
2 // Counter-controlled repetition with the for repetition statement.
3
4 public class ForCounter
5 {
6     public static void main( String args[] )
7     {
8         // for statement header includes initialization,
9         // loop-continuation condition and increment
10        for ( int counter = 1; counter <= 10; counter++ )
11            System.out.printf( "%d ", counter );
12
13        System.out.println(); // output a newline
14    } // end main
15 } // end class Fo

```

Control-variable name is **counter**

Control-variable initial value is **1**

Increment for **counter**

Condition tests for
counter's final value



Common Programming Error

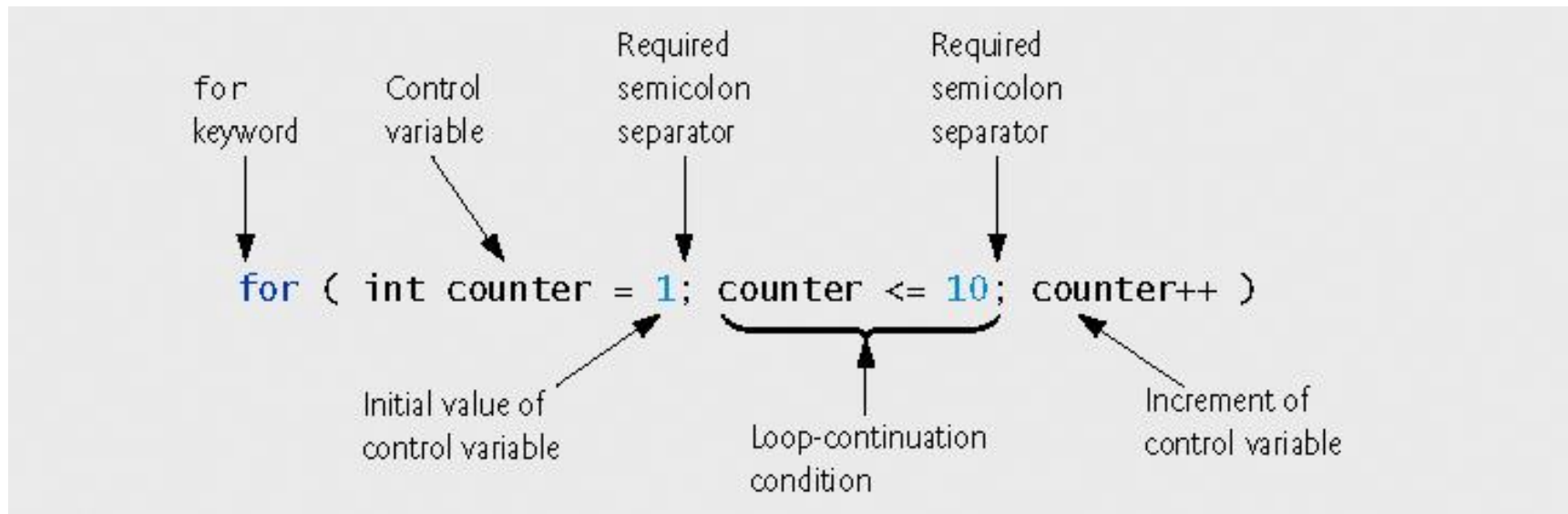
Using an incorrect relational operator or an incorrect final value of a loop counter in the loop-continuation condition of a repetition statement can cause an off-by-one error.



Good Programming Practice

Using the final value in the condition of a `while` or `for` statement and using the `<=` relational operator helps avoid off-by-one errors. For a loop that prints the values 1 to 10, the loop-continuation condition should be `counter <= 10` rather than `counter < 10` (which causes an off-by-one error) or `counter < 11` (which is correct). Many programmers prefer so-called zero-based counting, in which to count 10 times, `counter` would be initialized to zero and the loop-continuation test would be `counter < 10`.





for statement header components.

2c.3 for Repetition Statement (Cont.)

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

can usually be rewritten as:

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

Common Programming Error

Using commas instead of the two required semicolons in a for header is a syntax error.

Common Programming Error

When a for statement's control variable is declared in the initialization section of the for's header, using the control variable after the for's body is a compilation error.



Performance Tip

There is a slight performance advantage to preincrementing, but if you choose to postincrement because it seems more natural (as in a for header), optimizing compilers will generate Java bytecode that uses the more efficient form anyway.



Good Programming Practice

In the most cases, preincrementing and postincrementing are both used to add 1 to a variable in a statement by itself. In these cases, the effect is exactly the same, except that preincrementing has a slight performance advantage. Given that the compiler typically optimizes your code to help you get the best performance, use the idiom with which you feel most comfortable in these situations.



Common Programming Error

Placing a semicolon immediately to the right of the right parenthesis of a for header makes that for's body an empty statement. This is normally a logic error.



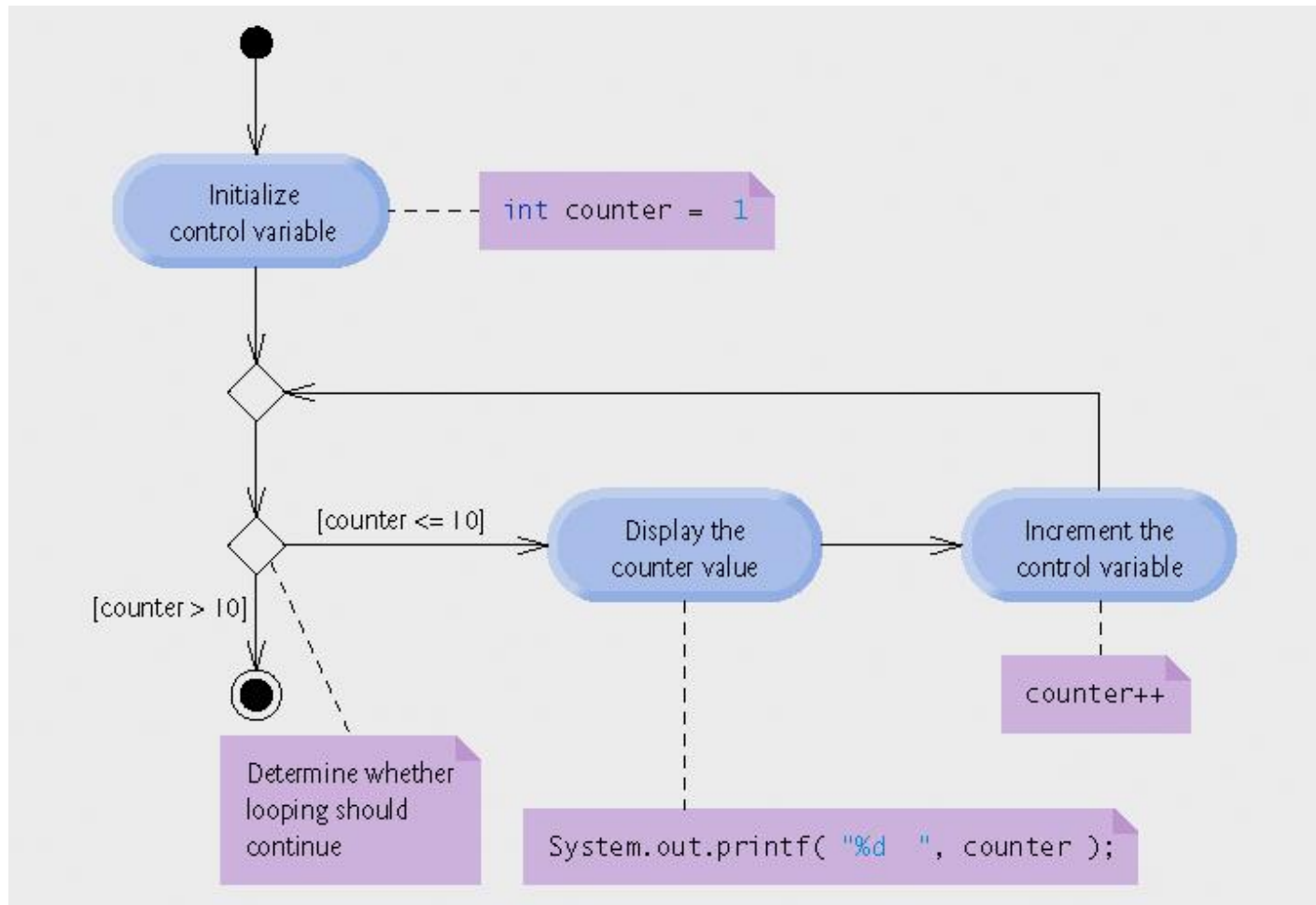
Error-Prevention Tip

Infinite loops occur when the loop-continuation condition in a repetition statement never becomes false. To prevent this situation in a counter-controlled loop, ensure that the control variable is incremented (or decremented) during each iteration of the loop. In a sentinel-controlled loop, ensure that the sentinel value is eventually input.



Error-Prevention Tip

Although the value of the control variable can be changed in the body of a for loop, avoid doing so, because this practice can lead to subtle errors.



UML activity diagram for the for statement in Fig. 2c.2.

2c.4 Examples Using the for Statement

Varying control variable in for statement

- Vary control variable from 1 to 100 in increments of 1
 - `for (int i = 1; i <= 100; i++)`
- Vary control variable from 100 to 1 in increments of -1
 - `for (int i = 100; i >= 1; i--)`
- Vary control variable from 7 to 77 in increments of 7
 - `for (int i = 7; i <= 77; i += 7)`
- Vary control variable from 20 to 2 in decrements of 2
 - `for (int i = 20; i >= 2; i -= 2)`
- Vary control variable over the sequence: 2, 5, 8, 11, 14, 17, 20
 - `for (int i = 2; i <= 20; i += 3)`
- Vary control variable over the sequence: 99, 88, 77, 66, 55, 44, 33, 22, 11, 0
 - `for (int i = 99; i >= 0; i -= 11)`



Common Programming Error

Not using the proper relational operator in the loop-continuation condition of a loop that counts downward (e.g., using $i \leq 1$ instead of $i \geq 1$ in a loop counting down to 1) is usually a logic error.



Outline

Sum.java

Line 11

```
1 // Fig. 5.5: Sum.java
2 // Summing integers with the for statement.
3
4 public class Sum
5 {
6     public static void main( String args[] )
7     {
8         int total = 0; // initialize total
9
10        // total even integers from 2 through 20
11        for ( int number = 2; number <= 20; number += 2 )
12            total += number;
13
14        System.out.printf( "Sum is %d\n", total ); // display results
15    } // end main
16 } // end class Sum
```

increment number by 2 each iteration

Sum is 110



2c.4 Examples Using the for Statement (Cont.)

Initialization and increment expression can be comma-separated lists of expressions

- E.g., lines 11-12 of Fig. 2c.5 can be rewritten as

```
for ( int number = 2; number <= 20; total += number, number += 2 )  
    ; // empty statement
```



Good Programming Practice

Limit the size of control statement headers to a single line if possible.

Good Programming Practice

Place only expressions involving the control variables in the initialization and increment sections of a for statement. Manipulations of other variables should appear either before the loop (if they execute only once, like initialization statements) or in the body of the loop (if they execute once per iteration of the loop, like increment or decrement statements).

Outline

Interest.java (1 of 2)

Line 8

Line 13

```
1 // Fig. 5.6: Interest.java
2 // Compound-interest calculations with for.
3
4 public class Interest
5 {
6     public static void main( String args[]
7     {
8         double amount; // amount on deposit at end of each year
9         double principal = 1000.0; // initial amount before interest
10        double rate = 0.05; // interest rate
11
12        // display headers
13        System.out.printf( "%s%20s\n", "Year", "Amount on deposit" );
14
```

Java treats literal values with decimal points as type `double`

Second string is right justified and displayed with a field width of 20



Outline

Calculate amount with for statement

Interest.java

(2 of 2)

```

15 // calculate amount on deposit for each of ten years
16 for ( int year = 1; year <= 10; year++ )
17 {
18     // calculate new amount for specified year
19     amount = principal * Math.pow( 1.0 + rate, year );
20
21     // display the year and the amount
22     System.out.printf( "%4d%,20.2f\n", year, amount );
23 } // end for
24 } // end main
25 } // end class Interest

```

Use the comma (,) formatting flag to display the amount with a thousands separator

Year	Amount on deposit
1	1,050.00
2	1,102.50
3	1,157.63
4	1,215.51
5	1,276.28
6	1,340.10
7	1,407.10
8	1,477.46
9	1,551.33
10	1,628.89

Line 22

Program output



2c.4 Examples Using the for Statement (Cont.)

- **Formatting output**
 - **Field width**
 - **Minus sign (-) formatting flag for left justification**
 - **Comma (,) formatting flag to output numbers with grouping separators**
- **static method**
 - *ClassName.methodName(arguments)*



Good Programming Practice

Do not use variables of type `double` (or `float`) to perform precise monetary calculations. The imprecision of floating-point numbers can cause errors that will result in incorrect monetary values. In the exercises, we explore the use of integers to perform monetary calculations. [*Note:* Some third-party vendors provide for-sale class libraries that perform precise monetary calculations. In addition, the Java API provides class `java.math.BigDecimal` for performing calculations with arbitrary precision floating-point values.]

Performance Tip

In loops, avoid calculations for which the result never changes—such calculations should typically be placed before the loop. [Note: Many of today's sophisticated optimizing compilers will place such calculations outside loops in the compiled code.]

2c.5 do...while Repetition Statement

do...while statement

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



Outline

DowhileTest.java

```

1 // Fig. 5.7: DowhileTest.java
2 // do...while repetition statement.
3
4 public class DowhileTest
5 {
6     public static void main( String args[]
7     {
8         int counter = 1; // initialize counter
9
10        do
11        {
12            System.out.printf( "%d ", counter );
13            ++counter;
14        } while ( counter <= 10 ); // end do...while
15
16        System.out.println(); // outputs a newline
17    } // end main
18 } // end class DowhileTest

```

Declares and initializes
control variable **counter**

Variable **counter**'s value is displayed
before testing **counter**'s final value

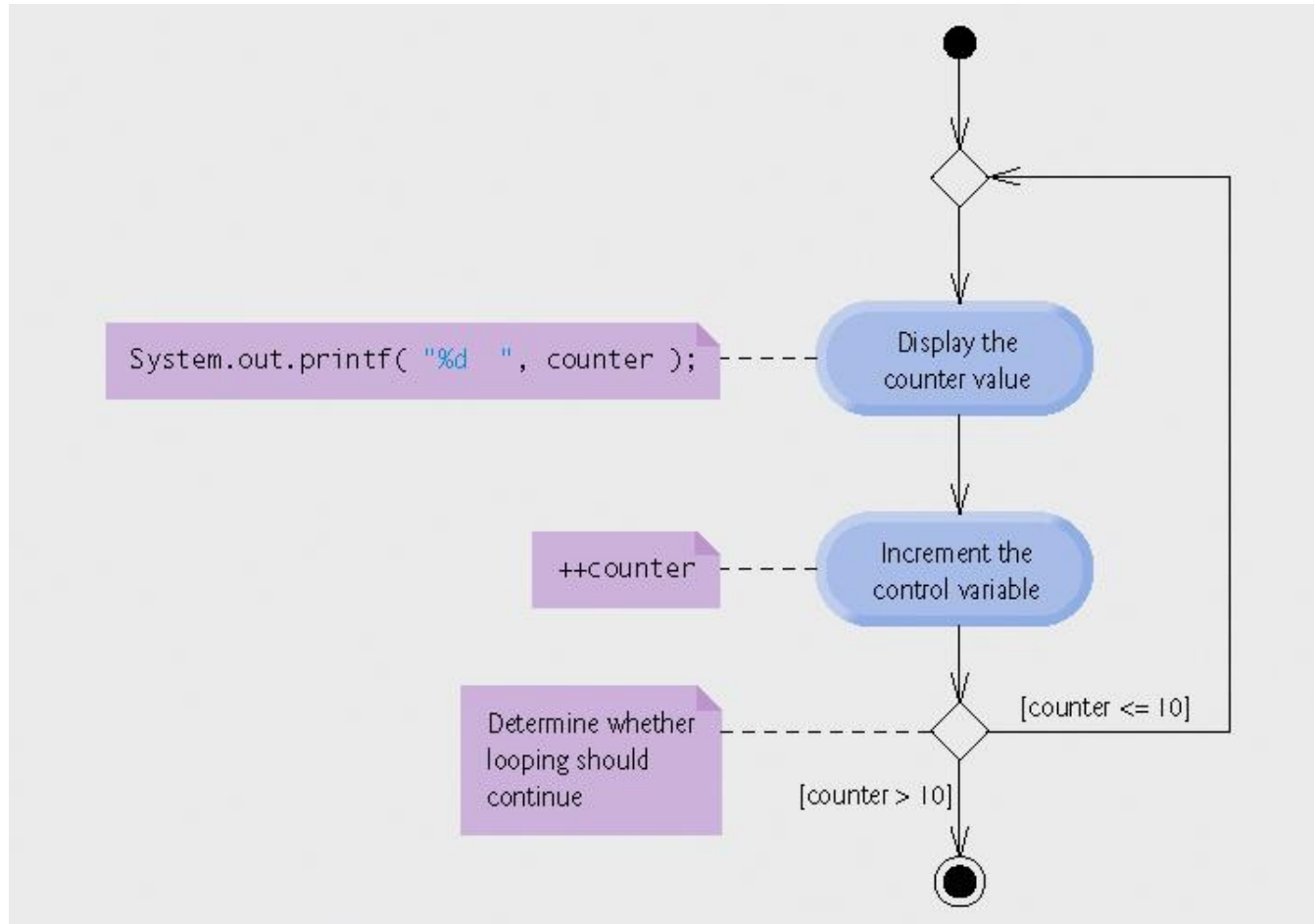
Line 8

Lines 10-14

Program output

1 2 3 4 5 6 7 8 9 10





do...while repetition statement UML activity diagram.

Good Programming Practice

Always include braces in a `do...while` statement, even if they are not necessary. This helps eliminate ambiguity between the `while` statement and a `do...while` statement containing only one statement.



2c.6 switch Multiple-Selection Statement

switch statement

- **Used for multiple selections**



Outline

GradeBook.java

(1 of 5)

Lines 8-14

```
1 // Fig. 5.9: GradeBook.java
2 // GradeBook class uses switch statement to count A, B, C, D and F grades.
3 import java.util.Scanner; // program uses class Scanner
4
5 public class GradeBook
6 {
7     private String courseName; // name of course this GradeBook represents
8     private int total; // sum of grades
9     private int gradeCounter; // number of grades entered
10    private int aCount; // count of A grades
11    private int bCount; // count of B grades
12    private int cCount; // count of C grades
13    private int dCount; // count of D grades
14    private int fCount; // count of F grades
15
16    // constructor initializes courseName;
17    // int instance variables are initialized to 0 by default
18    public GradeBook( String name )
19    {
20        courseName = name; // initializes courseName
21    } // end constructor
22
23    // method to set the course name
24    public void setCourseName( String name )
25    {
26        courseName = name; // store the course name
27    } // end method setCourseName
28
```



Outline

GradeBook.java

(2 of 5)

Lines 50-54

```
29 // method to retrieve the course name
30 public String getCourseName()
31 {
32     return courseName;
33 } // end method getCourseName
34
35 // display a welcome message to the GradeBook user
36 public void displayMessage()
37 {
38     // getCourseName gets the name of the course
39     System.out.printf( "welcome to the grade book for\n%s!\n\n",
40         getCourseName() );
41 } // end method displayMessage
42
43 // input arbitrary number of grades from user
44 public void inputGrades()
45 {
46     Scanner input = new Scanner( System.in );
47
48     int grade; // grade entered by user
49
50     System.out.printf( "%s\n%s\n  %s\n  %s\n",
51         "Enter the integer grades in the range 0-100.",
52         "Type the end-of-file indicator to terminate input:",
53         "On UNIX/Linux/Mac OS X type <ctrl> d then press Enter",
54         "On windows type <ctrl> z then press Enter" );
55
```

Display prompt



Outline

GradeBook.java

(3 of 5)

Line 57

Line 72 controlling
expression

Lines 72-94

// loop until user enters the end-of-file indicator

```
56 while ( input.hasNext() )
57 {
58     grade = input.nextInt(); // read
59     total += grade; // add grade to t
60     ++gradeCounter; // increment numb
```

Loop condition uses method `hasNext` to determine whether there is more data to input

```
61 // call method to increment appropriate counter
62 incrementLetterGradeCounter( grade );
63 } // end while
64 } // end method inputGrades
```

```
65 // add 1 to appropriate counter for specified grade
66 public void incrementLetterGradeCounter( int numericGrade )
67 {
```

```
68 // determine which grade was entered
```

```
69 switch ( grade / 10 )
```

```
70 {
```

(`grade / 10`) is
controlling expression

`switch` statement determines
which `case` label to execute,
depending on controlling expression

```
71 case 9: // grade was between 90
```

```
72 case 10: // and 100
```

```
73 ++aCount; // increment aCount
```

```
74 break; // necessary to exit swi
```

```
75 case 8: // grade was between 80 and 89
```

```
76 ++bCount; // increment bCount
```

```
77 break; // exit switch
```



Outline

GradeBook.java

(4 of 5)

Line 91 default case

```

83      case 7: // grade was between 70 and 79
84          ++cCount; // increment cCount
85          break; // exit switch
86
87      case 6: // grade was between 60 and 69
88          ++dCount; // increment dCount
89          break; // exit switch
90
91      default: // grade was less than 60
92          ++fCount; // increment fCount
93          break; // optional: will exit switch anyway
94  } // end switch
95  } // end method incrementLetterGradeCounter
96
97  // display a report based on the grades entered by user
98  public void displayGradeReport()
99  {
100      System.out.println( "\nGrade Report:" );
101
102      // if user entered at least one grade...
103      if ( gradeCounter != 0 )
104      {
105          // calculate average of all grades entered
106          double average = (double) total / gradeCounter;
107

```

default case for grade less than 60



Outline

GradeBook.java

(5 of 5)

```
108 // output summary of results
109 System.out.printf( "Total of the %d grades entered is %d\n",
110     gradeCounter, total );
111 System.out.printf( "Class average is %.2f\n", average );
112 System.out.printf( "%s\n%s%d\n%s%d\n%s%d\n%s%d\n%s%d\n",
113     "Number of students who received each grade:",
114     "A: ", aCount, // display number of A grades
115     "B: ", bCount, // display number of B grades
116     "C: ", cCount, // display number of C grades
117     "D: ", dCount, // display number of D grades
118     "F: ", fCount ); // display number of F grades
119 } // end if
120 else // no grades were entered, so output appropriate message
121     System.out.println( "No grades were entered" );
122 } // end method displayGradeReport
123} // end class GradeBook
```



Portability Tip

The keystroke combinations for entering end-of-file are system dependent.

Common Programming Error

Forgetting a break statement when one is needed in a switch is a logic error.



Outline

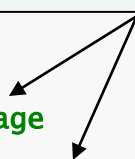
GradeBookTest.java

(1 of 2)

Lines 13-15

```
1 // Fig. 5.10: GradeBookTest.java
2 // Create GradeBook object, input grades and display grade report.
3
4 public class GradeBookTest
5 {
6     public static void main( String args[] )
7     {
8         // create GradeBook object myGradeBook and
9         // pass course name to constructor
10        GradeBook myGradeBook = new GradeBook(
11            "CS101 Introduction to Java Programming" );
12
13        myGradeBook.displayMessage(); // display welcome message
14        myGradeBook.inputGrades(); // read grades from user
15        myGradeBook.displayGradeReport(); // display report based on grades
16    } // end main
17 } // end class GradeBookTest
```

Call GradeBook public
methods to count grades



Outline

GradeBookTest.java

(2 of 2)

Program output

```
Welcome to the grade book for  
CS101 Introduction to Java Programming!
```

```
Enter the integer grades in the range 0-100.  
Type the end-of-file indicator to terminate input:  
    On UNIX/Linux/Mac OS X type <ctrl> d then press Enter  
    On windows type <ctrl> z then press Enter
```

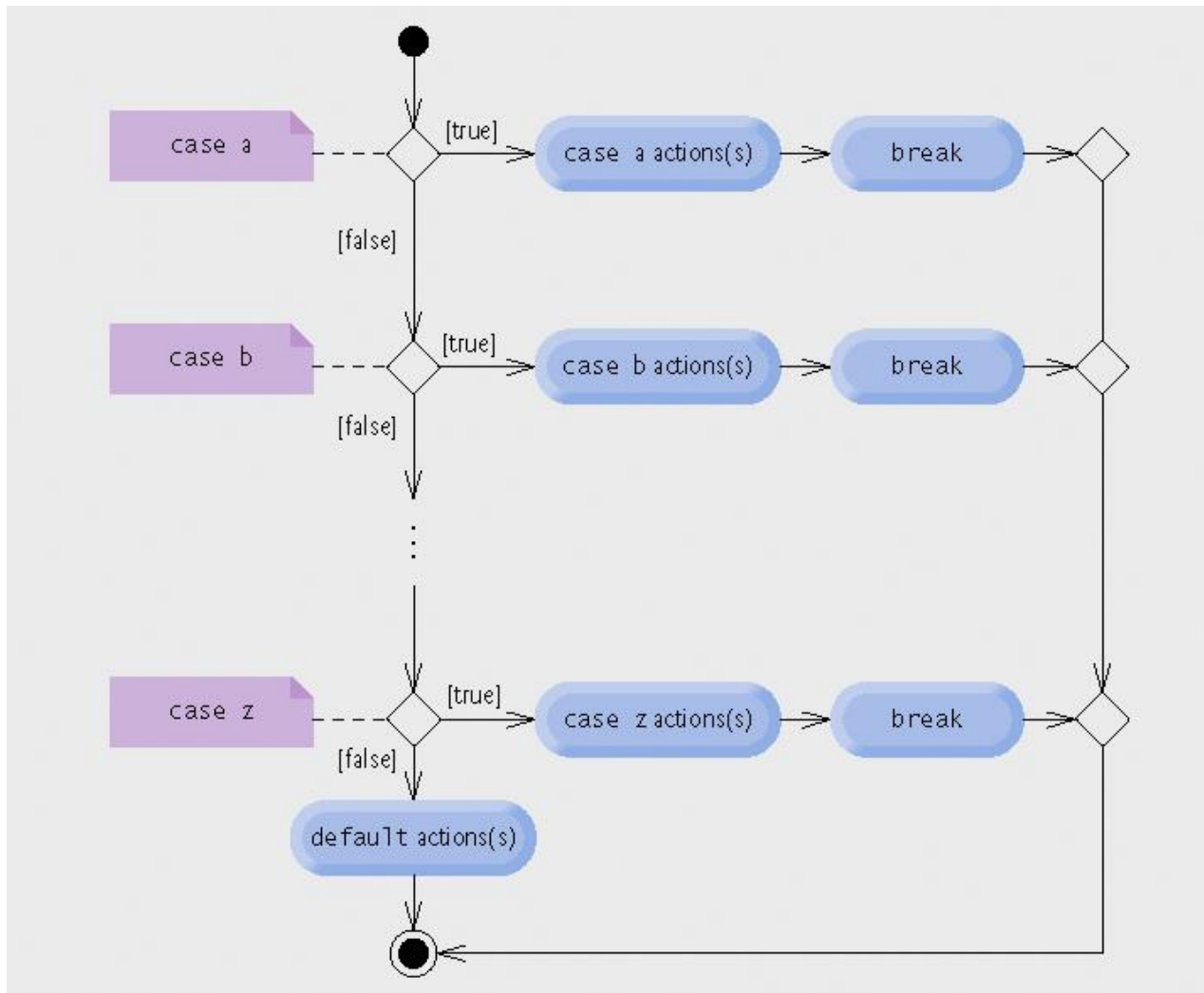
```
99  
92  
45  
57  
63  
71  
76  
85  
90  
100  
^Z
```

```
Grade Report:  
Total of the 10 grades entered is 778  
Class average is 77.80  
Number of students who received each grade:  
A: 4  
B: 1  
C: 2  
D: 1  
F: 2
```



Software Engineering Observation

**Provide a default case in switch statements.
Including a default case focuses you on the
need to process exceptional conditions.**



switch multiple-selection statement UML activity diagram with break statements.

Good Programming Practice

Although each case and the default case in a switch can occur in any order, place the default case last. When the default case is listed last, the break for that case is not required. Some programmers include this break for clarity and symmetry with other cases.

2c.6 switch Multiple-Selection Statement (Cont.)

Expression in each case

- Constant integral expression
 - Combination of integer constants that evaluates to a constant integer value
- Character constant
 - E.g., 'A', '7' or '\$'
- Constant variable
 - Declared with keyword `final`



2c.7 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



switch using Strings

```
int numberOfLetters;  
switch ("MONDAY") {  
    case "MONDAY":  
    case "FRIDAY":  
    case "SUNDAY":  
        numberOfLetters = 6; break;  
    case "TUESDAY":  
        numberOfLetters = 7; break;  
    case "THURSDAY":  
    case "SATURDAY":  
        numberOfLetters = 8; break;  
    case "WEDNESDAY":  
        numberOfLetters = 9; break;  
    default:  
        numberOfLetters = -1; }  
System.out.println(numberOfLetters);
```

Output: 6



Strings in a *switch* command

```
public static void main(String[] args) {  
    String color = "red";  
    String colorRGB = null;  
    // Convert to switch over Strings  
    if (color.equals("black")) {  
        colorRGB = "008090";  
    }  
    else if (color.equals("red"))  
        colorRGB = "00FF00";  
    else {  
        colorRGB = "00FFFF";  
    }  
}
```

Strings in a *switch* command

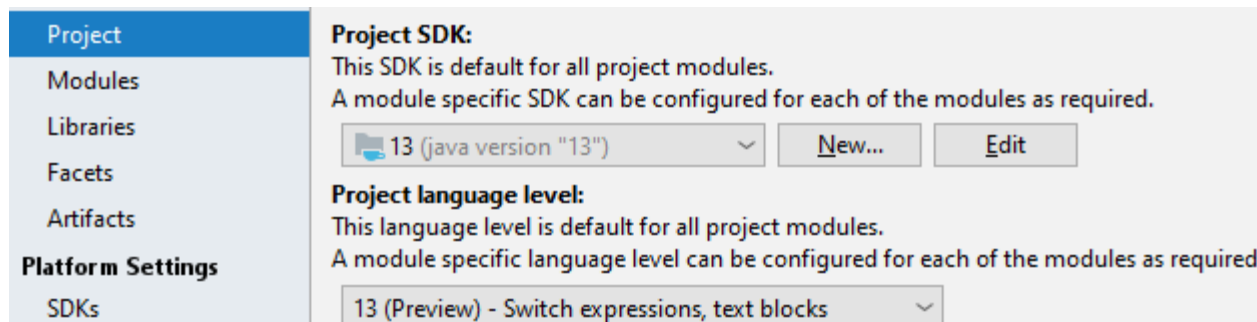
```
public static void main(String[] args) {  
    String color = "red";  
    String colorRGB = null;  
    switch (color) {  
        case "black":  
            colorRGB = "008090";  
            break;  
        case "red":  
            colorRGB = "00FF00";  
            break;  
        default:  
            colorRGB = "00FFFF";  
            break;  
    }  
}
```



switch statement in JDK 13 (preview)

```
String dayOfWeek = "MONDAY";  
int numberOfLetters = switch (dayOfWeek) {  
    case "MONDAY", "FRIDAY", "SUNDAY" -> 6;  
    case "TUESDAY" -> 7; // return value  
    case "THURSDAY", "SATURDAY" -> 8;  
    case "WEDNESDAY" -> 9 ;  
    default -> -1;  
};  
System.out.println( numberOfLetters);
```

Output: 6



Outline

BreakTest.java

Line 9

Lines 11-12

Program output

```
1 // Fig. 5.12: BreakTest.java
2 // break statement exiting a for statement.
3 public class BreakTest
4 {
5     public static void main( String args[] )
6     {
7         int count; // control variable also used
8
9         for ( count = 1; count <= 10; count++ ) // Loop 10 times
10        {
11            if ( count == 5 ) // if count is 5,
12                break; // terminate loop
13
14            System.out.printf( "%d ", count );
15        } // end for
16
17        System.out.printf( "\nBroke out of loop at count = %d\n", count );
18    } // end main
19 } // end class BreakTest
```

Loop 10 times

Exit **for** statement (break)
when count equals 5

```
1 2 3 4
Broke out of loop at count = 5
```



Outline

ContinueTest.java

Line 7

Lines 9-10

Program output

```
1 // Fig. 5.13: ContinueTest.java
2 // continue statement terminating an iteration of a for statement.
3 public class ContinueTest
4 {
5     public static void main( String args[] )
6     {
7         for ( int count = 1; count <= 10; count++ )
8         {
9             if ( count == 5 ) // if count is 5,
10                continue;    // skip remaining code in loop
11
12             System.out.printf( "%d ", count );
13         } // end for
14
15         System.out.println( "\nUsed continue to skip printing 5" );
16     } // end main
17 } // end class ContinueTest
```

Loop 10 times

Skip line 12 and proceed to
line 7 when count equals 5

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing 5
```



Software Engineering Observation

Some programmers feel that break and continue violate structured programming. Since the same effects are achievable with structured programming techniques, these programmers do not use break or continue.

Software Engineering Observation

There is a tension between achieving quality software engineering and achieving the best-performing software. Often, one of these goals is achieved at the expense of the other. For all but the most performance-intensive situations, apply the following rule of thumb: First, make your code simple and correct; then make it fast and small, but only if necessary.

2c.8 Logical Operators

Logical operators

- Allows for forming more complex conditions
- Combines simple conditions

Java logical operators

- **&&** (conditional AND)
- **||** (conditional OR)
- **&** (boolean logical AND)
- **|** (boolean logical inclusive OR)
- **^** (boolean logical exclusive OR)
- **!** (logical NOT)



2c.8 Logical Operators (Cont.)

Conditional AND (&&) Operator

- Consider the following `if` statement

```
if ( gender == FEMALE && age >= 65 )  
    ++seniorFemales;
```
- Combined condition is `true`
 - if and only if both simple conditions are `true`
- Combined condition is `false`
 - if either or both of the simple conditions are `false`



expression1	expression2	expression1 && expression2
false	false	False
false	true	False
true	false	False
true	true	True

&& (conditional AND) operator truth table.

2c.8 Logical Operators (Cont.)

Conditional OR (||) Operator

- Consider the following **if** statement

```
if ( ( semesterAverage >= 90 ) || ( finalExam >= 90 )  
    System.out.println( "Student grade is A" );
```

- Combined condition is **true**
 - if either or both of the simple condition are **true**
- Combined condition is **false**
 - if both of the simple conditions are **false**



expression1	expression2	expression1 expression2
false	false	false
false	true	true
true	false	true
true	true	true

|| (conditional OR) operator truth table.



2c.8 Logical Operators (Cont.)

Short-Circuit Evaluation of Complex Conditions

- Parts of an expression containing **&&** or **||** operators are evaluated only until it is known whether the condition is true or false
- E.g., (**gender == FEMALE**) **&&** (**age >= 65**)
 - Stops immediately if gender is not equal to FEMALE



Common Programming Error

In expressions using operator `&&`, a condition—we will call this the dependent condition—may require another condition to be true for the evaluation of the dependent condition to be meaningful. In this case, the dependent condition should be placed after the other condition, or an error might occur. For example, in the expression `(i != 0) && (10 / i == 2)`, the second condition must appear after the first condition, or a divide-by-zero error might occur.

2c.8 Logical Operators (Cont.)

Boolean Logical AND (&) Operator

- Works identically to **&&**
- Except **&** always evaluate both operands

Boolean Logical OR (|) Operator

- Works identically to **||**
- Except **|** always evaluate both operands



Error-Prevention Tip

For clarity, avoid expressions with side effects in conditions. The side effects may look clever, but they can make it harder to understand code and can lead to subtle logic errors.

2c.8 Logical Operators (Cont.)

Boolean Logical Exclusive OR (\wedge)

- One of its operands is **true** and the other is **false**
 - Evaluates to **true**
- Both operands are **true** or both are **false**
 - Evaluates to **false**

Logical Negation (!) Operator

- Unary operator



expression1	expression2	expression1 \wedge expression2
false	false	false
false	true	true
true	false	true
true	true	false

\wedge (boolean logical exclusive OR) operator truth table.



expression	!expression
false	true
true	false

! (logical negation, or logical NOT) operator truth table.



Outline

LogicalOperators.
java

(1 of 3)

Lines 9-13

Conditional AND truth table

Lines 16-20

Conditional OR truth table

Boolean logical AND
truth table

```

1 // Fig. 5.18: LogicalOperators.java
2 // Logical operators.
3
4 public class LogicalOperators
5 {
6     public static void main( String args[] )
7     {
8         // create truth table for && (conditional AND) operator
9         System.out.printf( "%s\n%s: %b\n%s: %b\n%s: %b\n%s: %b\n\n",
10             "Conditional AND (&&)", "false && false", ( false && false ),
11             "false && true", ( false && true ),
12             "true && false", ( true && false ),
13             "true && true", ( true && true ) );
14
15         // create truth table for || (conditional OR) operator
16         System.out.printf( "%s\n%s: %b\n%s: %b\n%s: %b\n%s: %b\n\n",
17             "Conditional OR (||)", "false || false", ( false || false ),
18             "false || true", ( false || true ),
19             "true || false", ( true || false ),
20             "true || true", ( true || true ) );
21
22         // create truth table for & (boolean logical AND) operator
23         System.out.printf( "%s\n%s: %b\n%s: %b\n%s: %b\n%s: %b\n\n",
24             "Boolean logical AND (&)", "false & false", ( false & false ),
25             "false & true", ( false & true ),
26             "true & false", ( true & false ),
27             "true & true", ( true & true ) );
28

```



Outline

Boolean logical inclusive
OR truth table

(2 of 3)

Lines 30-35

Boolean logical exclusive
OR truth table

Lines 46-47

Logical negation truth table

```

29 // create truth table for | (boolean logical inclusive OR) operator
30 system.out.printf( "%s\n%s: %b\n%s: %b\n%s: %b\n%s: %b\n\n",
31     "Boolean logical inclusive OR (|)",
32     "false | false", ( false | false ),
33     "false | true",  ( false | true  ),
34     "true | false",  ( true  | false ),
35     "true | true",   ( true  | true  ) );
36
37 // create truth table for ^ (boolean logical exclusive OR) operator
38 system.out.printf( "%s\n%s: %b\n%s: %b\n%s: %b\n%s: %b\n\n",
39     "Boolean logical exclusive OR (^)",
40     "false ^ false", ( false ^ false ),
41     "false ^ true",  ( false ^ true  ),
42     "true ^ false",  ( true  ^ false ),
43     "true ^ true",   ( true  ^ true  ) );
44
45 // create truth table for ! (logical negation) operator
46 system.out.printf( "%s\n%s: %b\n%s: %b\n", "Logical NOT (!)",
47     "!false", ( !false ), "!true", ( !true ) );
48 } // end main
49 } // end class LogicalOperators
  
```



Outline

LogicalOperators.
java

(3 of 3)

Program output

```
Conditional AND (&&)
false && false: false
false && true: false
true && false: false
true && true: true
```

```
Conditional OR (||)
false || false: false
false || true: true
true || false: true
true || true: true
```

```
Boolean logical AND (&)
false & false: false
false & true: false
true & false: false
true & true: true
```

```
Boolean logical inclusive OR (|)
false | false: false
false | true: true
true | false: true
true | true: true
```

```
Boolean logical exclusive OR (^)
false ^ false: false
false ^ true: true
true ^ false: true
true ^ true: false
```

```
Logical NOT (!)
!false: true
!true: false
```



Operators					Associativity	Type
++	--				right to left	unary postfix
++	-	+	-	! (type)	right to left	unary prefix
*	/	%			left to right	multiplicative
+	-				left to right	additive
<	<=	>	>=		left to right	relational
==	!=				left to right	equality
&					left to right	boolean logical AND
^					left to right	boolean logical exclusive OR
					left to right	boolean logical inclusive OR
&&					left to right	conditional AND
					left to right	conditional OR
?:					right to left	conditional
=	+=	-=	*=	/=	%=	right to left assignment

Precedence/associativity of the operators discussed so far.



2c.9 Structured Programming Summary

Sequence structure

- “built-in” to Java

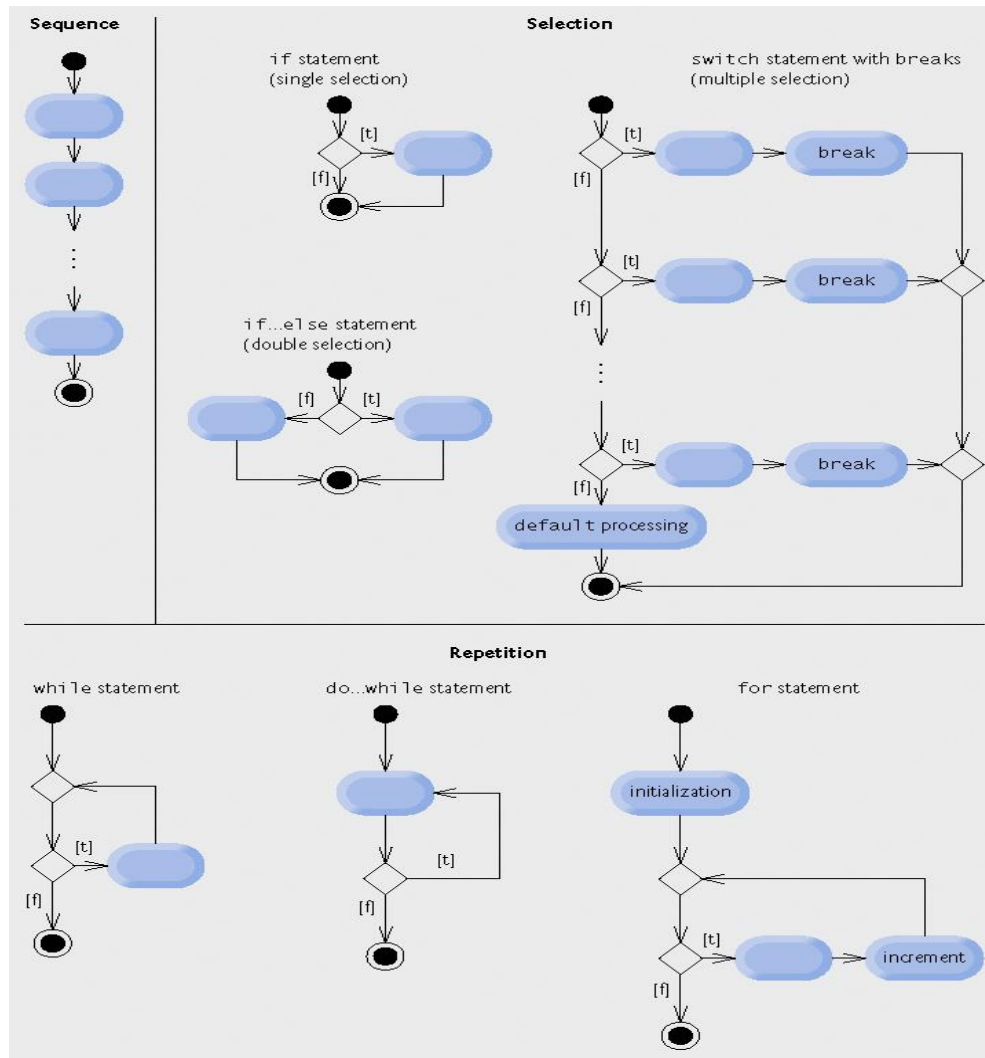
Selection structure

- `if`, `if...else` and `switch`

Repetition structure

- `while`, `do...while` and `for`



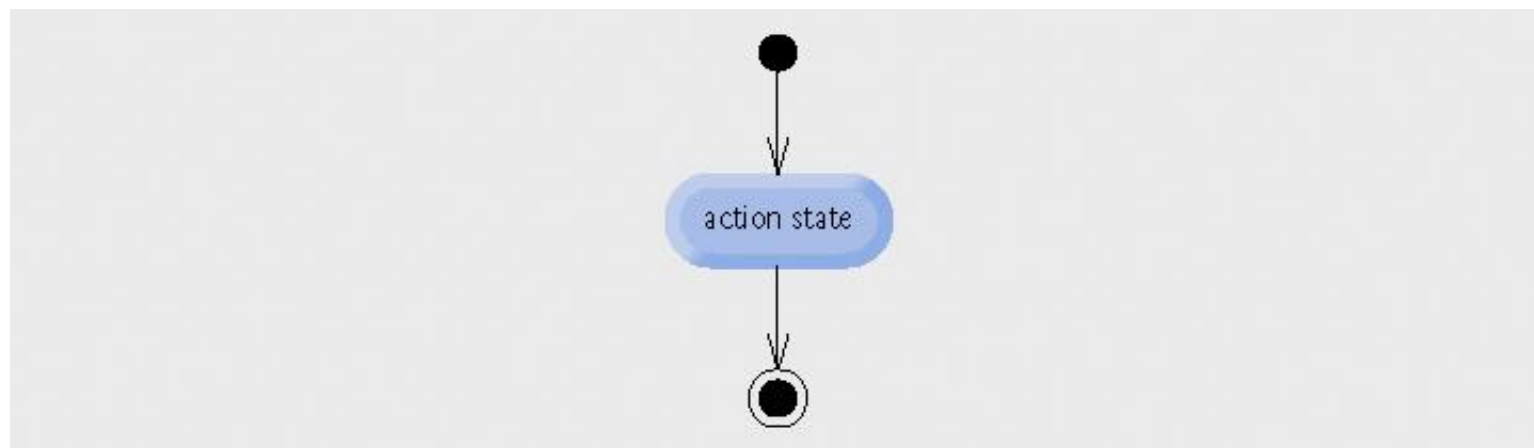


Java's single-entry/single-exit sequence, selection and repetition statements.

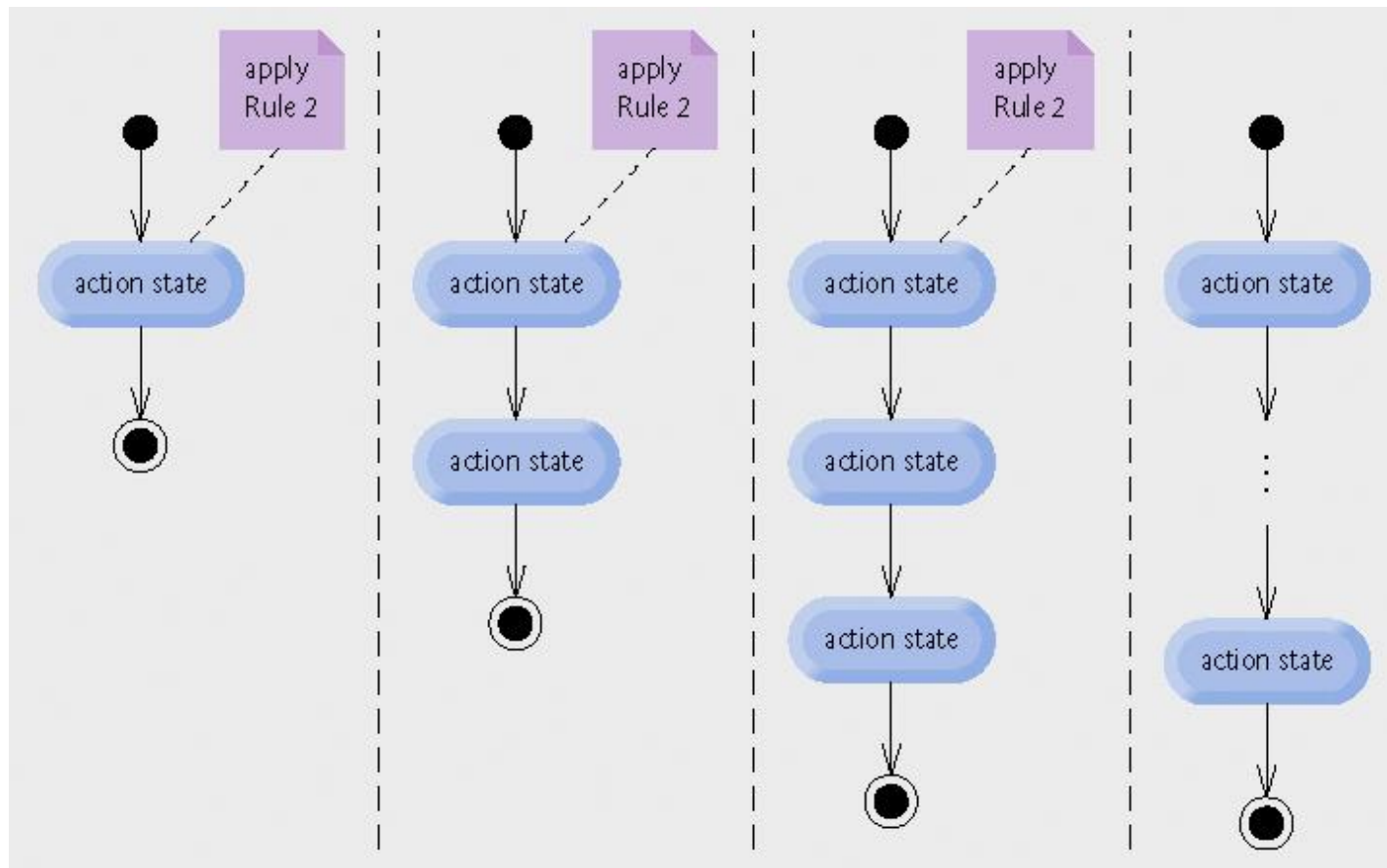
Rules for Forming Structured Programs

- 1 Begin with the simplest activity diagram (Fig. 5.22).**
- 2 Any action state can be replaced by two action states in sequence.**
- 3 Any action state can be replaced by any control statement (sequence of action states, if, if...else, switch, while, do...while or for).**
- 4 Rules 2 and 3 can be applied as often as you like and in any order.**

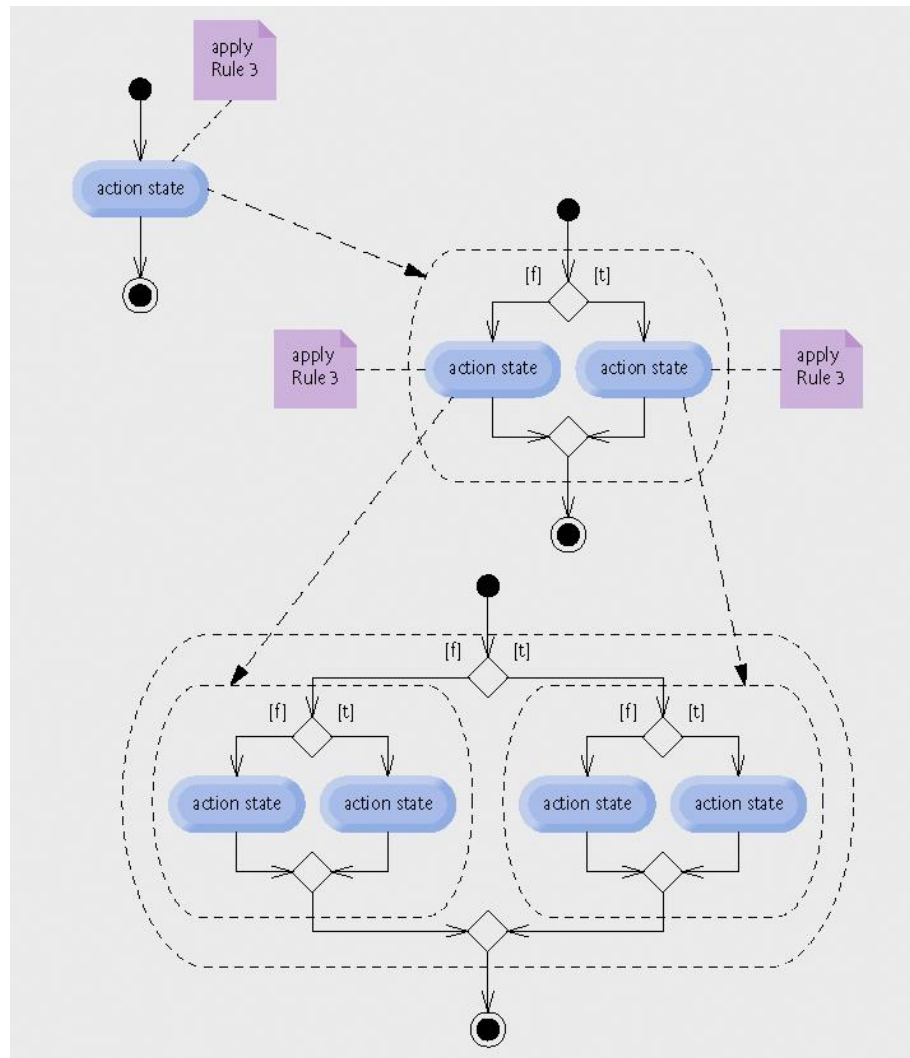
Rules for forming structured programs.



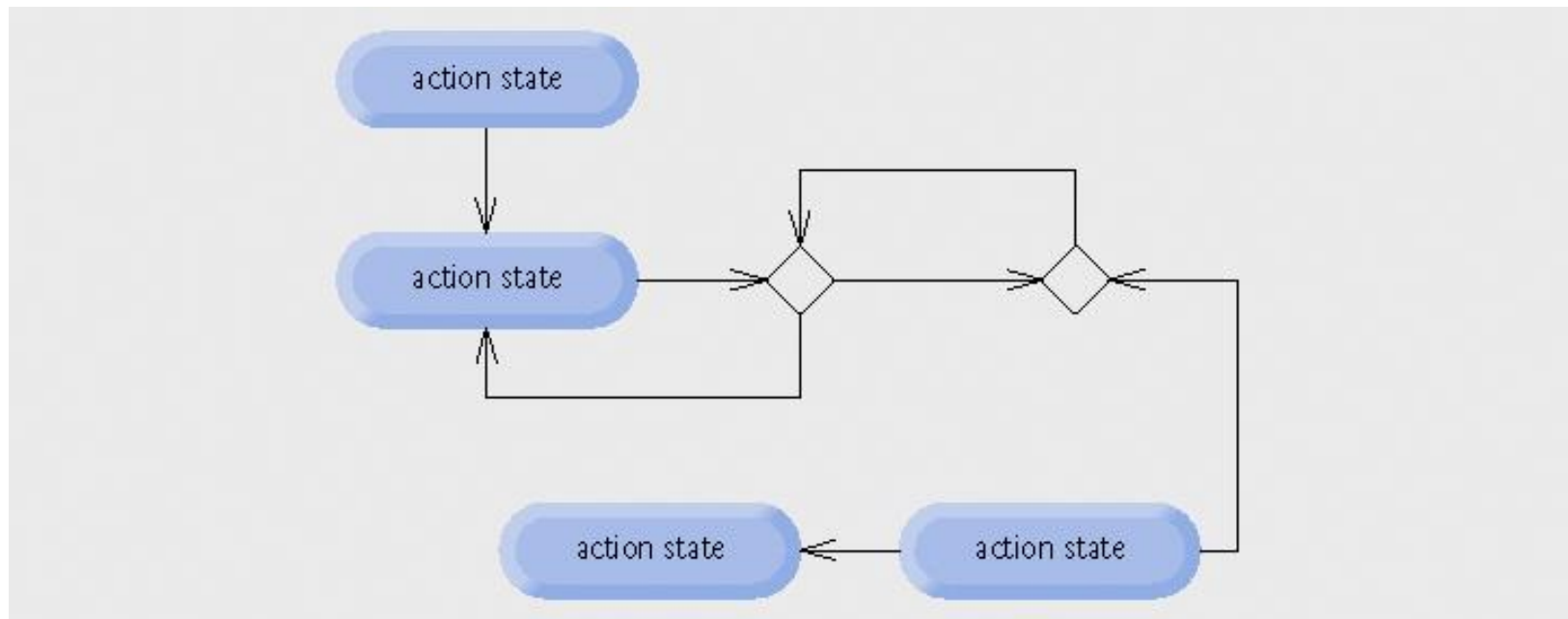
Simplest activity diagram.



Repeatedly applying the stacking rule (rule 2) of Fig. 2c.21 to the simplest activity diagram.



Repeatedly applying the nesting rule (rule 3) of Fig. 2c.21 to the simplest activity diagram.



“Unstructured” activity diagram.

2c.10 GUI and Graphics Case Study: Using Dialog Boxes with JavaFX

Windows and dialog boxes

- Many Java applications use these to display output
- JavaFX provides prepackaged dialog boxes

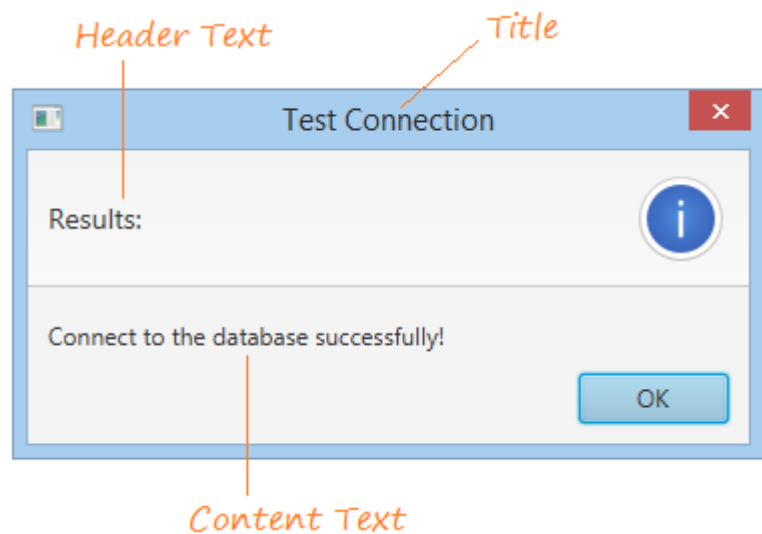
We study two of the JavaFX provides prepackaged dialog boxes

- Alert dialog box of type `AlertType.Information`
- `TextInputDialog` box



2c.10 GUI and Graphics Case Study: Using Dialog Boxes with JavaFX

Information Alert means a **dialog** window, displaying information. Below is an image of a standard **Information Alert**



2c.10 GUI and Graphics Case Study: Using Dialog Boxes with JavaFX

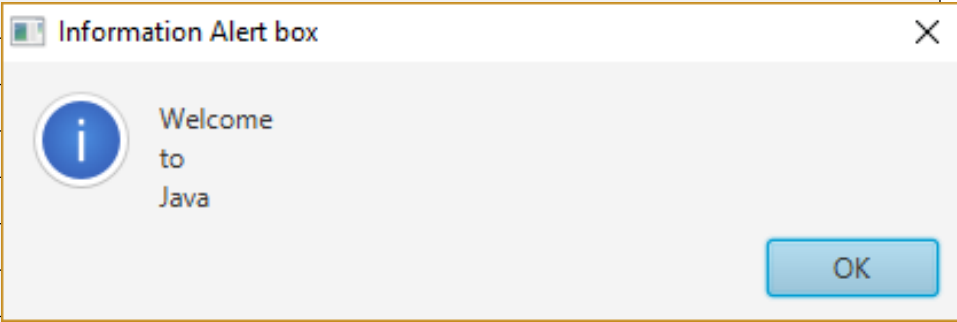
It is a good practice the Alert dialog box and its properties to be defined in a static method that is called always when needed.

```
//ClassName.messageDialog("YOUR INFORMATION HERE", "TITLE BAR MESSAGE", "HEADER MESSAGE");  
// Info message dialog with a header  
public static void messageDialog(String infoMessage,  
                                String titleBar, String headerMessage) {  
    Alert alert = new Alert(AlertType.INFORMATION);  
    alert.setTitle(titleBar);  
    alert.setHeaderText(headerMessage);  
    alert.setContentText(infoMessage);  
    alert.showAndWait();  
}
```

```
//ClassName.messageDialog("YOUR INFORMATION HERE", "TITLE BAR MESSAGE");  
// Info message dialog without a header  
public static void plainMessageDialog(String infoMessage, String titleBar) {  
    /* By specifying a null headerMessage String, we cause the dialog to  
       not have a header */  
    messageDialog(infoMessage, titleBar, null);  
}
```



```
1package alertbox;
2
3import javafx.application.Application;
4import javafx.application.Platform;
5import javafx.scene.Scene;
6import javafx.scene.control.Alert;
7import javafx.scene.control.Alert.AlertType;
8import javafx.scene.layout.Pane;
9import javafx.stage.Stage;
10
11public class DialogBoxesWithJavaFX extends Application {
12
13    public static void messageDialog(String infoMessage,
14        String titleBar, String headerMessage) {
15        Alert alert = new Alert(AlertType.INFORMATION);
16        alert.setTitle(titleBar);
17        alert.setHeaderText(headerMessage);
18        alert.setContentText(infoMessage);
19        alert.showAndWait();
20    }
21
22    @Override
23    public void start(Stage primaryStage) {
24        messageDialog("Welcome\nto\nJava", "Information Alert box", null);
25        Platform.exit();
26    }
27
28    public static void main(String[] args) {
29        launch(args);
30    }
31}
```

An information alert dialog box titled "Information Alert box" with a close button (X) in the top right corner. It features a blue circular icon with a white lowercase 'i' on the left. The text "Welcome to Java" is displayed in the center. An "OK" button is located at the bottom right.

Define the Alert dialogBox and setup its properties

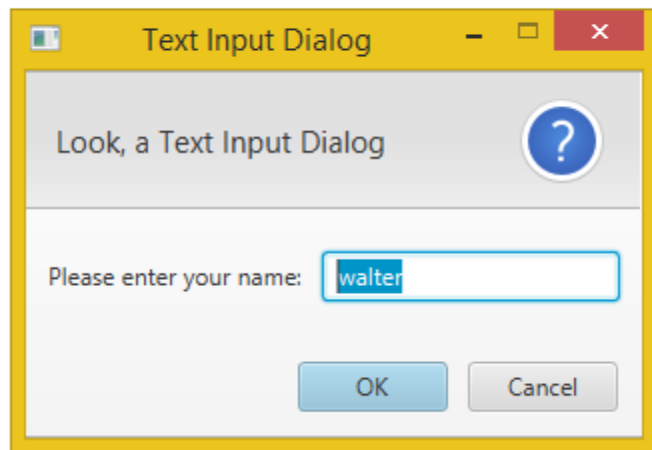
Display the Alert dialog box

Command to quit a JavaFX application

2c.10 GUI and Graphics Case Study: Using Dialog Boxes with JavaFX

The Text Input dialog has the same properties as the Alert dialog box.. It is used to input String data.

Text Input Dialog



2c.10 GUI and Graphics Case Study: Using Dialog Boxes with JavaFX

The `ContentText` property defines the prompt text. It returns a `String` or `null` depending on whether data has been input in the textbox of the dialog box.

```
public static String inputDialog(String infoMessage, String titleBar,
                                String headerMessage) {
    TextInputDialog dialog = new TextInputDialog();
    dialog.setTitle(titleBar);
    dialog.setHeaderText(headerMessage);
    dialog.setContentText(infoMessage);
    // Traditional way to get the response value.
    Optional<String> result = dialog.showAndWait();
    if (result.isPresent()) {
        return result.get();
    }
    else
        return null;
}

String infoMessage = "Въведи начална стойност (цяло число) : ";
String titleMessage = "Задай нова начална стойност";
String input = Add3JavaFX.inputDialog(infoMessage, titleMessage, null);
```

1	<code>package inputbox;</code>	0
2		
3	<code>import java.util.Optional;</code>	
4	<code>import javafx.application.Application;</code>	
5	<code>import javafx.application.Platform;</code>	
6	<code>import javafx.scene.control.Alert;</code>	
7	<code>import javafx.scene.control.Alert.AlertType;</code>	
8	<code>import javafx.scene.control.TextInputDialog;</code>	
9	<code>import javafx.stage.Stage;</code>	
10		
11	<code>public class InputBoxesWithJavaFX extends Application {</code>	
12		
13	<code> public static void messageDialog(String infoMessage,</code>	
14	<code> String titleBar, String headerMessage) {</code>	
15	<code> Alert alert = new Alert(AlertType.INFORMATION);</code>	
16	<code> alert.setTitle(titleBar);</code>	
17	<code> alert.setHeaderText(headerMessage);</code>	
18	<code> alert.setContentText(infoMessage);</code>	
19	<code> alert.showAndWait();</code>	
20	<code> }</code>	
21		
22	<code> public static String inputDialog(String infoMessage, String titleBar,</code>	
23	<code> String headerMessage) {</code>	
24	<code> TextInputDialog dialog = new TextInputDialog();</code>	
25	<code> dialog.setTitle(titleBar);</code>	
26	<code> dialog.setHeaderText(headerMessage);</code>	
27	<code> dialog.setContentText(infoMessage);</code>	
28	<code>// Traditional way to get the response value.</code>	
34	<code> return dialog.showAndWait().get();</code>	
35	<code> }</code>	

```

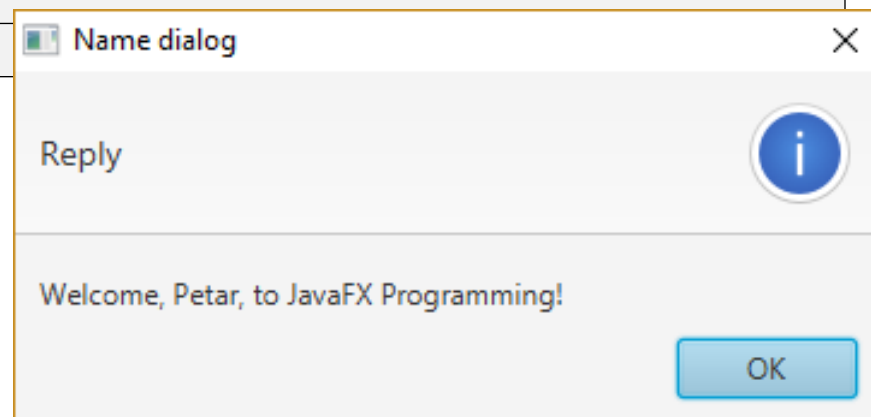
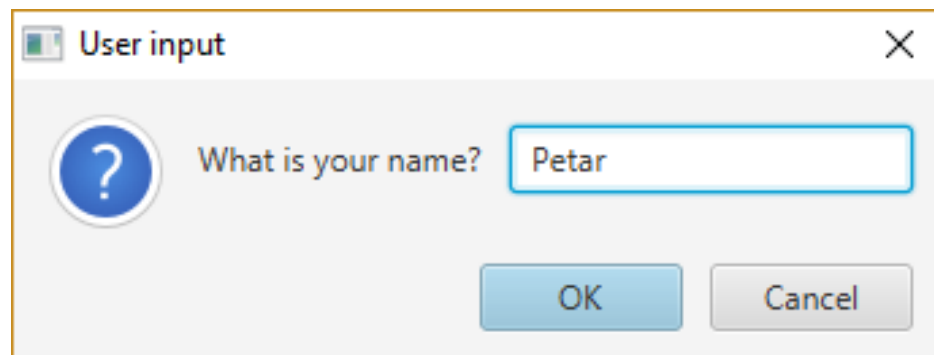
36
37  @Override
38  public void start(Stage primaryStage) {
39
40      String name = inputDialog("What is your name?", "User input", null);
41      // create the message
42      String message = String.format("Welcome, %s, to JavaFX Programming!", name);
43      // display the message to welcome the user by name
44      messageDialog(message, "Name dialog", "Reply");
45      Platform.exit(); // quit the JavaFX application
46  }
47
48  public static void main(String[] args) {
49      launch(args);
50  }
51}

```

Format a String to output to user

Show input dialog

Quit the JavaFX Application



2c.11 GUI and Graphics Case Study: Drawing Rectangles and Circles

Draw rectangles

- Class `Rectangle` of JavaFX

```
Rectangle(x, y, width, height)
```

```
Rectangle r = new Rectangle(30, 50, 200, 70);
```

Draw circles

- Class `Circle` of JavaFX

```
Circle(centerX, centerY, radius)
```

```
Circle c = new Circle(100, 150, 40);
```

Draw circles

- Class `Ellipse` of JavaFX

```
Ellipse(centerX, centerY, radiusX, radiusY)
```

```
Ellipse e = new Ellipse(100, 50, 80, 30);
```



2c.11 GUI and Graphics Case Study: Drawing Arcs

```
public Arc(double centerX, double centerY,  
           double radiusX, double radiusY,  
           double startAngle, double endAngle)
```

Creates a new instance of Arc.

Parameters:

- **centerX** - the X coordinate of the center point of the arc
- **centerY** - the Y coordinate of the center point of the arc
- **radiusX** - the overall width (horizontal radius) of the full ellipse of which this arc is a partial section
- **radiusY** - the overall height (vertical radius) of the full ellipse of which this arc is a partial section
- **startAngle** - the starting angle of the arc in degrees
- **endAngle** - the angular extent of the arc in degrees



2c.11 GUI and Graphics Case Study: Drawing Arcs

```
public Arc(double centerX, double centerY,  
           double radiusX, double radiusY,  
           double startAngle, double endAngle)
```

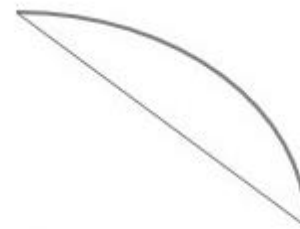
You can also set the type of the arc (round, chord or open) by using the **setType()** method.

//Setting the properties of the arc

```
arc.setCenterX(300.0);  
arc.setCenterY(150.0);  
arc.setRadiusX(90.0);  
arc.setRadiusY(90.0);  
arc.setStartAngle(40.0);  
arc.setLength(239.0);  
arc.setType(ArcType.ROUND);
```



Open arc



Closed arc



Round arc

```
1package drawshapeswithdialog;
2
3import javafx.application.Application;
4import javafx.scene.Group;
5import javafx.scene.Scene;
6import javafx.scene.control.TextInputDialog;
7import javafx.scene.paint.Color;
8import javafx.scene.shape.Circle;
9import javafx.scene.shape.Rectangle;
10import javafx.stage.Stage;
11
12public class DrawShapesWithDialog extends Application {
13
14    public static String inputDialog(String infoMessage, String titleBar,
15                                     String headerMessage) {
16        TextInputDialog dialog = new TextInputDialog();
17        dialog.setTitle(titleBar);
18        dialog.setHeaderText(headerMessage);
19        dialog.setContentText(infoMessage);
20        return dialog.showAndWait().get();
21    }
22
23    @Override
24    public void start(Stage primaryStage) {
25        // obtain user's choice
26        String input = inputDialog("Enter 1 to draw rectangles\n" +
27                                   "Enter 2 to draw ovals",
28                                   "User choice", "Select a shape");
29
30        int choice = Integer.parseInt(input); // convert input to int
```

Get user choice

```

31 Rectangle rectangle; //shape
32 Circle circle; //shape
33 Group group = new Group(); // Create a layout Panel
34 Scene scene = new Scene(group, 400, 400); // Create the Scene
35 // create the shape nodes
36 for (int i = 0; i < 10; i++) {
37     // pick the shape based on the user's choice
38     switch (choice) {
39         case 1: // draw rectangles
40             rectangle = new Rectangle(10 + i * 20, 10 + i * 20,
41                                     40 + i * 10, 40 + i * 10);
42             rectangle.setStroke(Color.BLACK);
43             rectangle.setFill(null);
44             group.getChildren().add(rectangle);
45             break;
46         case 2: // draw ovals
47             circle = new Circle(50 + i * 20, 50 + i * 20,
48                                40 + i * 10);
49             circle.setStroke(Color.BLACK);
50             circle.setFill(null);
51             group.getChildren().add(circle);
52             break;
53     } // end switch
54 } // end for
55
56
57 primaryStage.setTitle("Drawing shapes in a loop");
58 primaryStage.setScene(scene);
59 primaryStage.show();
60 }
61
62 public static void main(String[] args) {
63     launch(args);
64 }
65 }

```

Draw rectangles

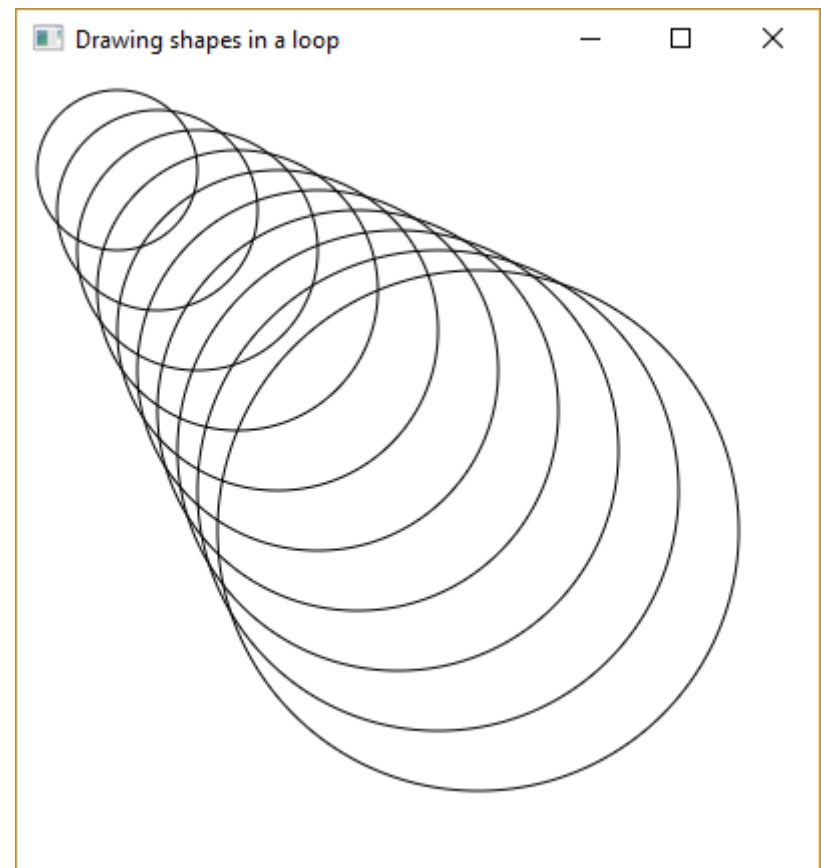
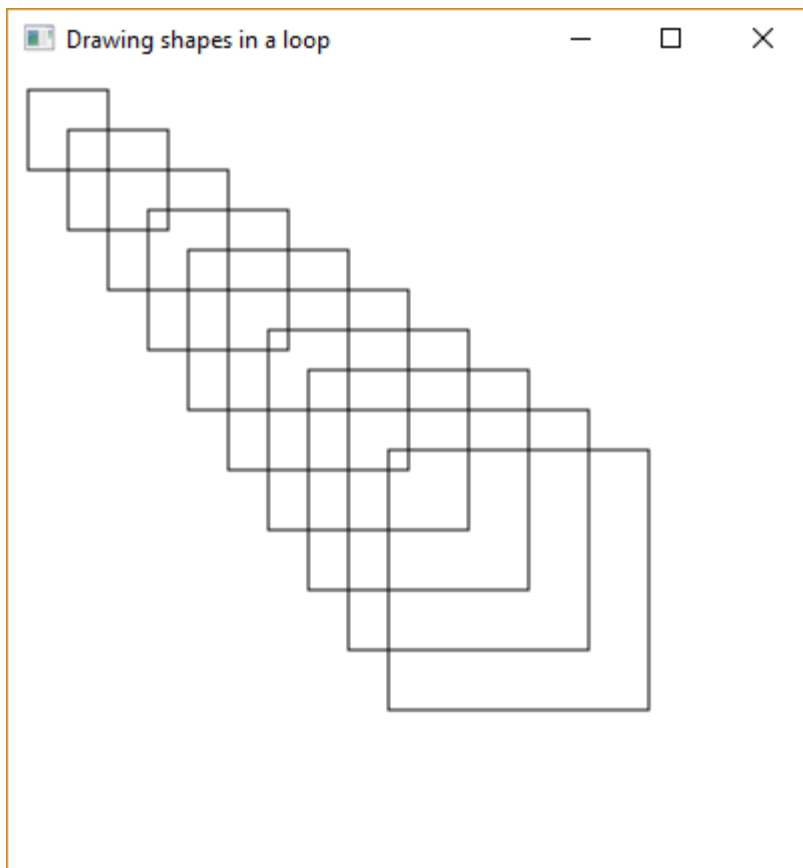
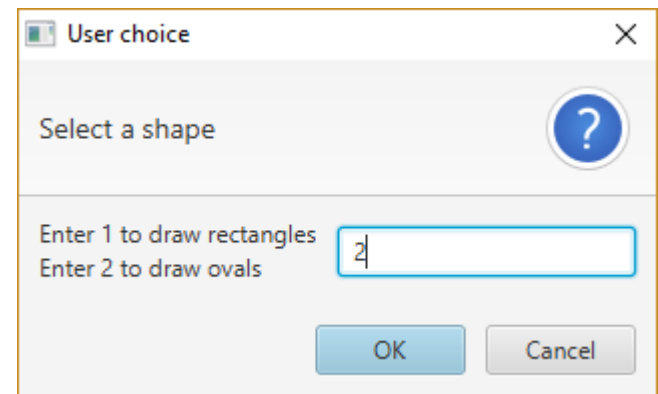
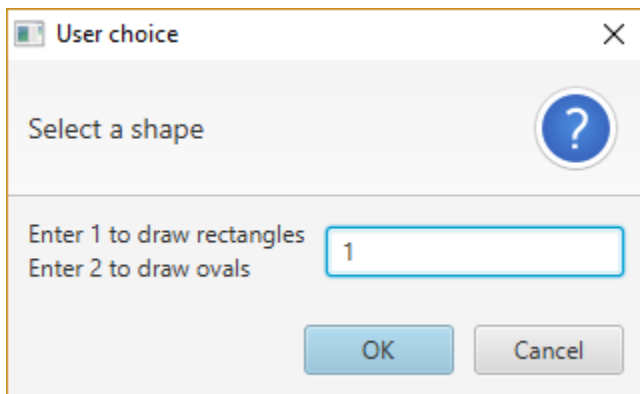
Draw circles

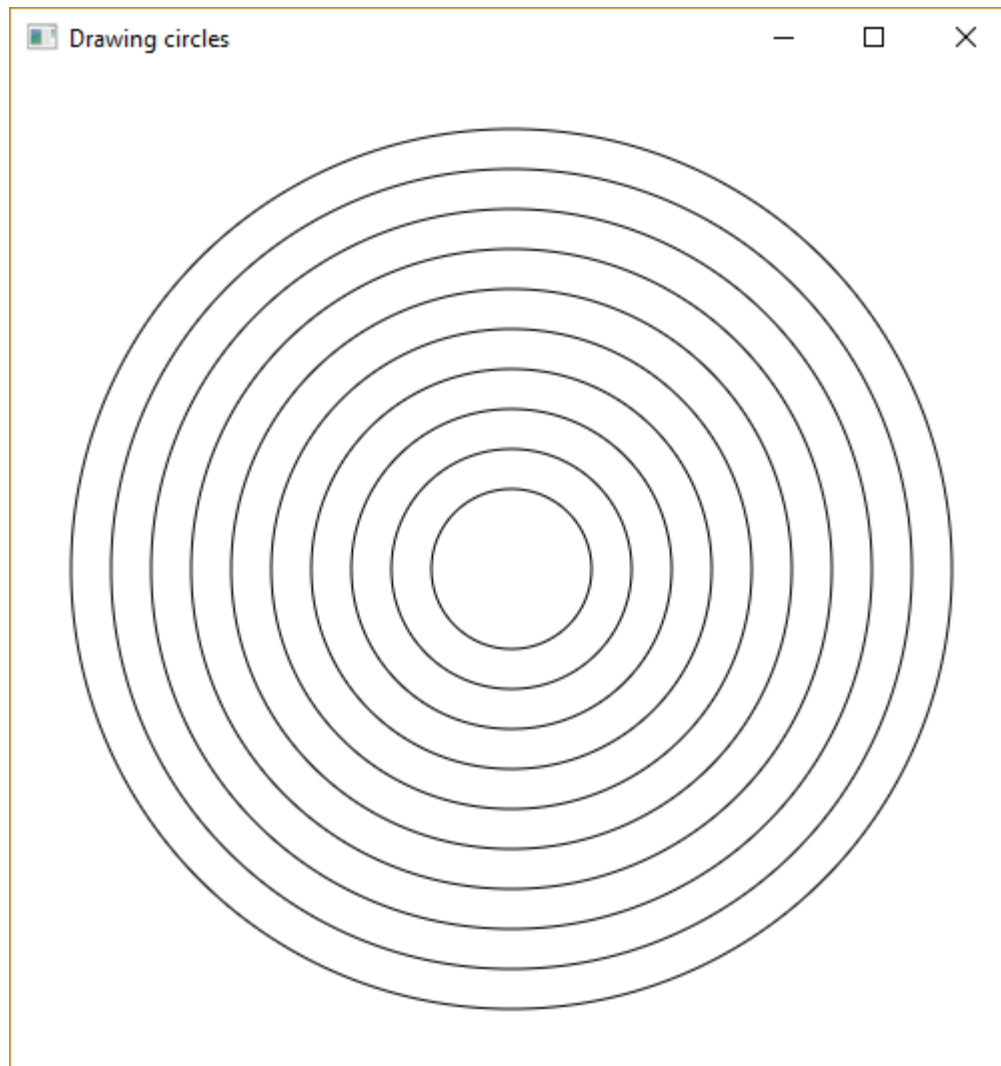
Problems to solve

Problem 1.

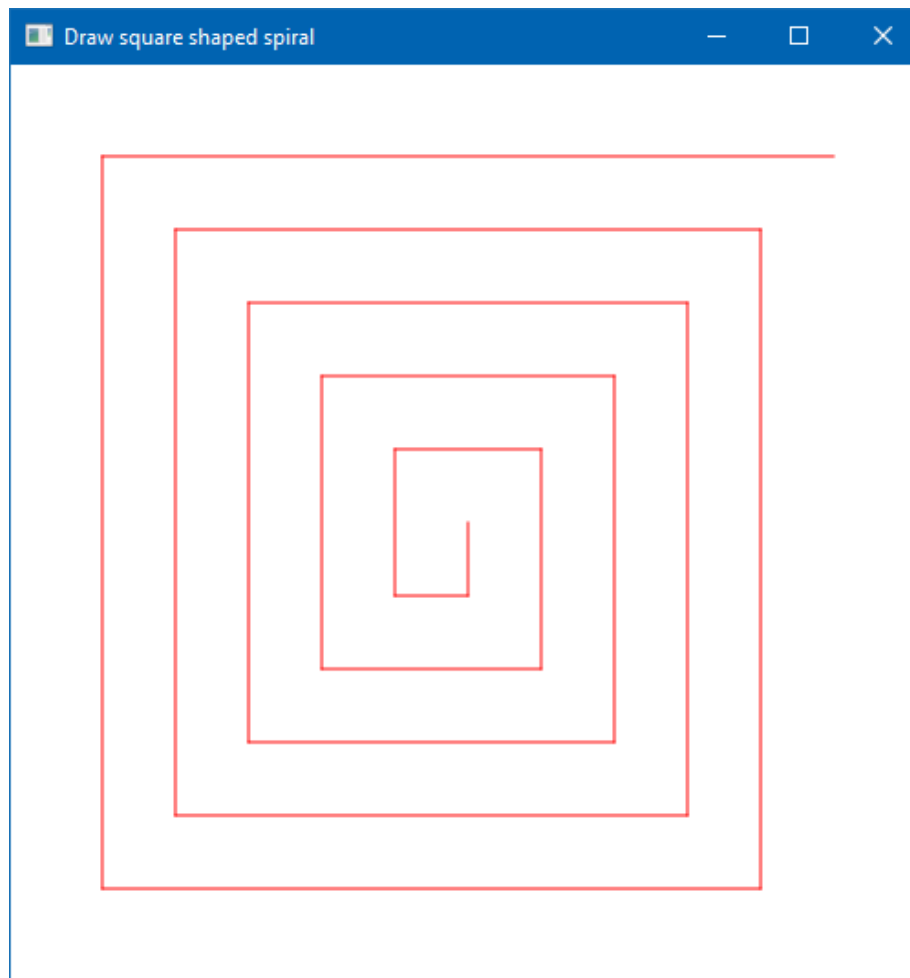
Draw the following shapes in JavaFX making use of appropriately defined loops.



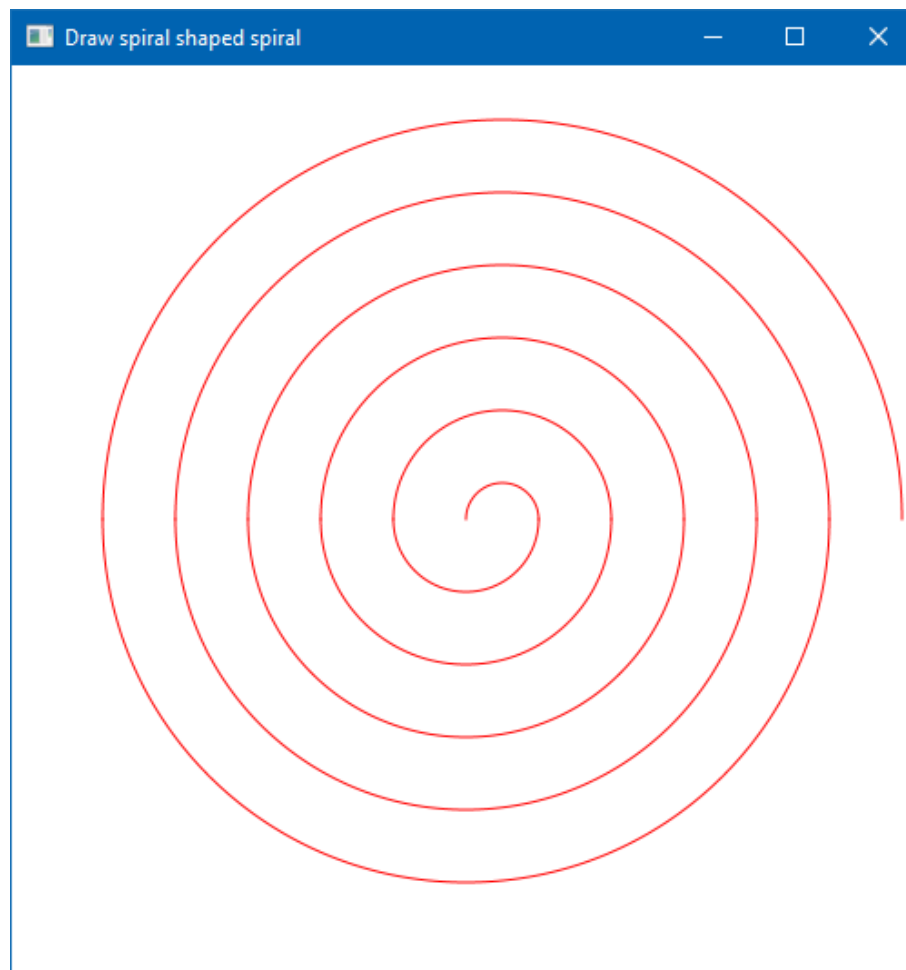




Drawing concentric circles.



Drawing spiral of line segments



Drawing spiral of circles.

Problems to solve

Problem 2.

Write Java program to create pyramid of numbers like in Patterns 1- 6 on the following image?



Problems to solve

```

      1
     2 2
    3 3 3
   4 4 4 4
  5 5 5 5 5
 6 6 6 6 6 6
 7 7 7 7 7 7 7
 8 8 8 8 8 8 8 8
 9 9 9 9 9 9 9 9 9
  
```

Pattern1

```

      1
     1 2
    1 2 3
   1 2 3 4
  1 2 3 4 5
 1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
  
```

Pattern2

```

      *
     * *
    * * *
   * * * *
  * * * * *
 * * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * *
* * * * * * * *
  
```

Pattern3

```

              1
            1 2 1
          1 2 3 2 1
        1 2 3 4 3 2 1
      1 2 3 4 5 4 3 2 1
    1 2 3 4 5 6 5 4 3 2 1
  1 2 3 4 5 6 7 6 5 4 3 2 1
1 2 3 4 5 6 7 8 7 6 5 4 3 2 1
  
```

Pattern4

```

1 2 3 4 5 6 7 8 9 8 7 6 5 4 3 2 1
 1 2 3 4 5 6 7 8 7 6 5 4 3 2 1
   1 2 3 4 5 6 7 6 5 4 3 2 1
    1 2 3 4 5 6 5 4 3 2 1
     1 2 3 4 5 4 3 2 1
      1 2 3 4 3 2 1
       1 2 3 2 1
        1 2 1
         1
  
```

Pattern5

```

              9
            8 9 8
          7 8 9 8 7
        6 7 8 9 8 7 6
      5 6 7 8 9 8 7 6 5
    4 5 6 7 8 9 8 7 6 5 4
  3 4 5 6 7 8 9 8 7 6 5 4 3
2 3 4 5 6 7 8 9 8 7 6 5 4 3 2
1 2 3 4 5 6 7 8 9 8 7 6 5 4 3 2 1
  
```

Pattern6

