Repetition statements

Problem 1: how to display 1..N

- Task
  - Display the integer number from 1 to N (>1) one by one
- Output
  - Print the integer number 1, 2, 3, ….., N
- Algorithm 1

```
public class Counter {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int i = 1;
        if (i<N)
         System.out.println(i);
        if (i+1<N)
         System.out.println(i+1);
        if (i+2<N)
         System.out.println(i+2);
        if
          .
          .
           System.out.println(N);
    }
}
```

Problem 2: square root

- Task
  - Find out square root of a perfect square number
- Output
  - Print the square root of a given perfect square number
- Algorithm 1
  - if 2*2 = the given number, print 2;
  - else if 3*3 = the given number, print 3;
  - else if 4*4 = the given number, print 4;
  - …

Outline

- **`while`** repetition statement
- **`for`** repetition statement
- **`do…while`** repetition statement
- **`break`** statement
- **`continue`** statement

Counter-Controlled Repetition

- Counter-controlled repetition requires
  - a **control variable** (or loop counter)
  - the **initial value** of the control variable
  - the **increment** by which the control variable is modified each time through the loop (also known as **each iteration of the loop**)
  - the **loop-continuation condition** that determines if looping should continue.

`while` Repetition Statement

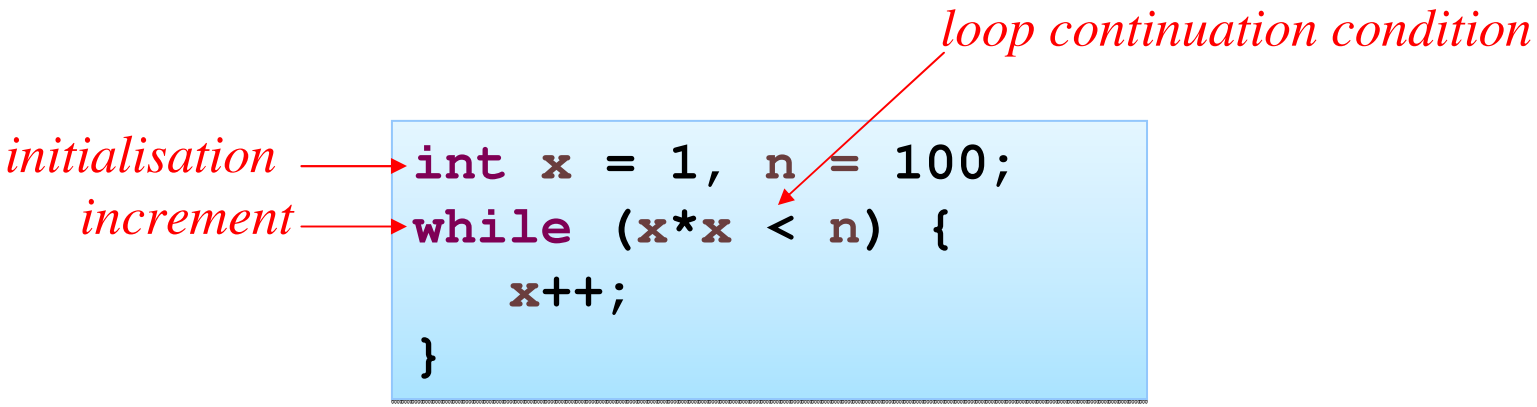
- Repetition statement - repeats an action while a condition remains true.
- Pseudocode

  While there are more items on my shopping list
  Purchase next item and cross it off my list

- The repetition statement's body may be a single statement or a block.
- Eventually, the condition will become false. At this point, the repetition terminates, and the first statement after the repetition statement executes.

`while` Repetition Statement

```
while (<boolean expression>) {
        <statements>
}
```



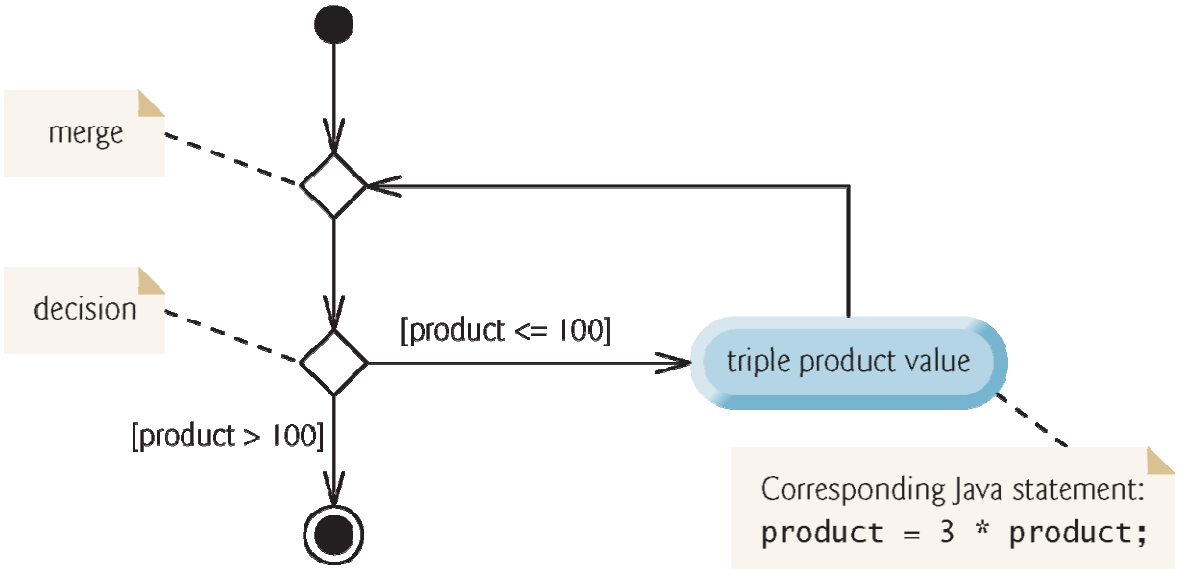
`while` Repetition Statement

- Example of Java's **`while`** repetition statement:

  - find the first power of 3 larger than 100.

```
int product = 1;
while (product <= 100)
   product = 3 * product;
System.out.println(product);
```

- Each iteration multiplies product by 3, so product takes on the values 9, 27, 81 and 243 successively.
- When product becomes 243, **`product <= 100`** becomes false.
- Repetition terminates. The final value of product is 243.
- Program execution continues with the next statement after the **`while`** statement.



`while` repetition statement UML activity diagram.

`while` Repetition Statement

- The decision and merge symbols can be distinguished by the number of “incoming” and “outgoing” transition arrows.
  - A decision symbol has one transition arrow pointing to the diamond and two or more pointing out from it to indicate possible transitions from that point. Each transition arrow pointing out of a decision symbol has a guard condition next to it.
  - A merge symbol has two or more transition arrows pointing to the diamond and only one pointing from the diamond, to indicate multiple activity flows merging to continue the activity. None of the transition arrows associated with a merge symbol has a guard condition.

```
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       }
15
16       System.out.println();
17    }
18 } // end class WhileCounter
```

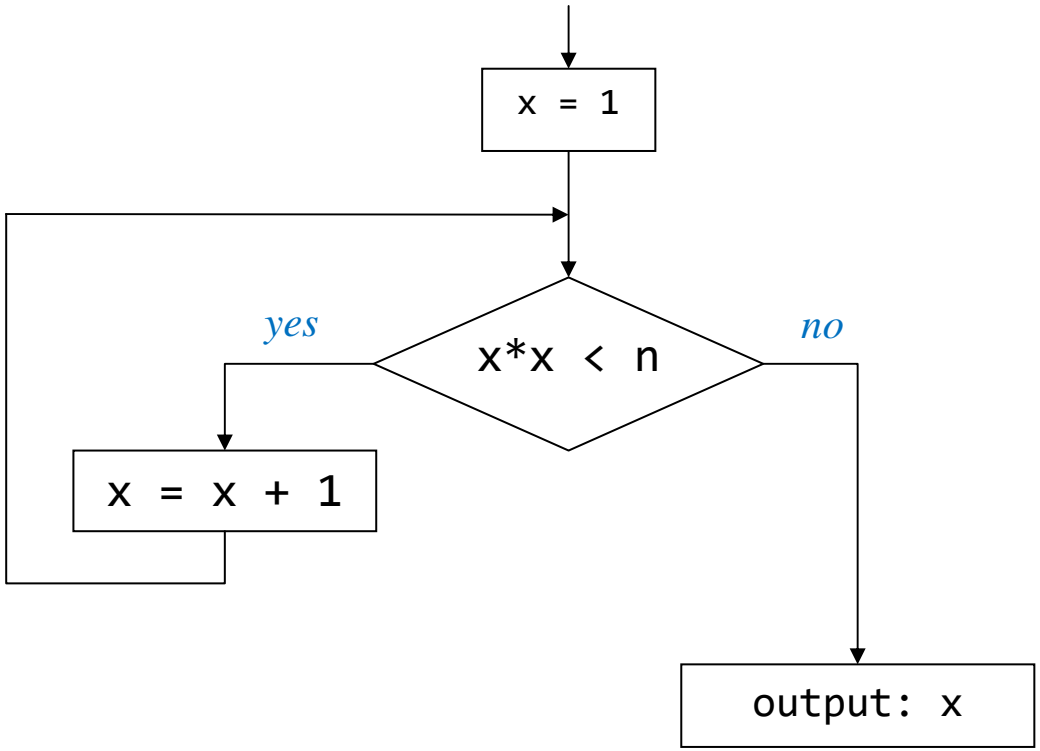
```
1  2  3  4  5  6  7  8  9  10
```

Counter-controlled repetition with the `while` repetition statement.

Problem 2: square root

- Task
  - Find out square root of a perfect square number
- Target
  - Print the square root of a given perfect square number
- Algorithm 2
  1. let x = 1;
  2. while x*x < (the given perfect square number), x=x+1 and repeat step 2;
  3. else, print x;

Flowchart: square root



Example: square root

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

        int n = 16;            // a perfect square

        int x = 1;
        while (x*x < n) {
               x++;
        }
        System.out.println(x);
    }
}
```

iteration	n	x	x*x	x*x < n
1	16	1	1	true
2		2	4	true
3		3	9	true
4		4	16	false

output: x

Infinite loops

- Loop continuation condition in a **`while`** loop is always satisfied

always true

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

    int x = 1;
    while (x*x > 0) {
      x++;
    }
    System.out.println(x);
  }
}
```

This statement will never execute

Don't run this program; it will never stop.
If you happen to run it, use Ctrl-C to quit it.

for Repetition Statement

- The general format of the for statement is

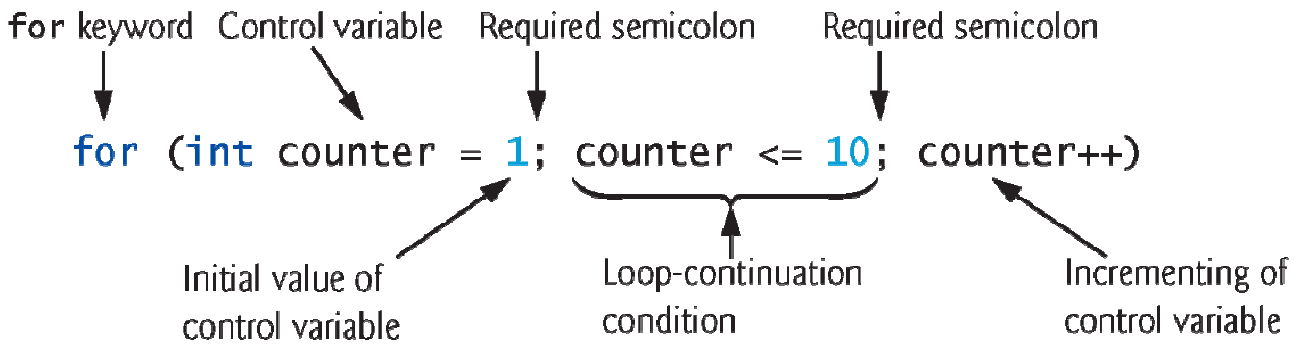
  `for` (*initialization; loopContinuationCondition; increment)*
  { statement/s }

  – the *initialization* expression names the loop's control variable and provides its initial value
  – *loopContinuationCondition* determines whether the loop should continue executing
  – *increment* modifies the control variable's value, so that the loop-continuation condition eventually becomes false.

- The two semicolons in the for header are required.

`for` Repetition Statement

- **`for` repetition statement**
  - Specifies the counter-controlled-repetition details in a single line of code.



`for` statement header components.

`for` Repetition Statement

1. When the **`for`** statement begins executing, the control variable is *declared* and *initialized*.
2. Next, the program checks the loop-continuation condition, which is between the two required semicolons.
3. If the condition initially is true, the body statement executes.
4. After executing the loop's body, the program increments the control variable in the increment expression, which appears to the right of the second semicolon.
5. Then the loop-continuation test is performed again to determine whether the program should continue with the next iteration of the loop.

Examples Using the `for` Statement

- a)Vary the control variable from 1 to 100 in increments of 1.
  - `for (int i = 1; i <= 100; i++)`
- b)Vary the control variable from 100 to 1 in decrements of 1.
  - `for (int i = 100; i >= 1; i--)`
- c)Vary the control variable from 7 to 77 in increments of 7.
  - `for (int i = 7; i <= 77; i += 7)`

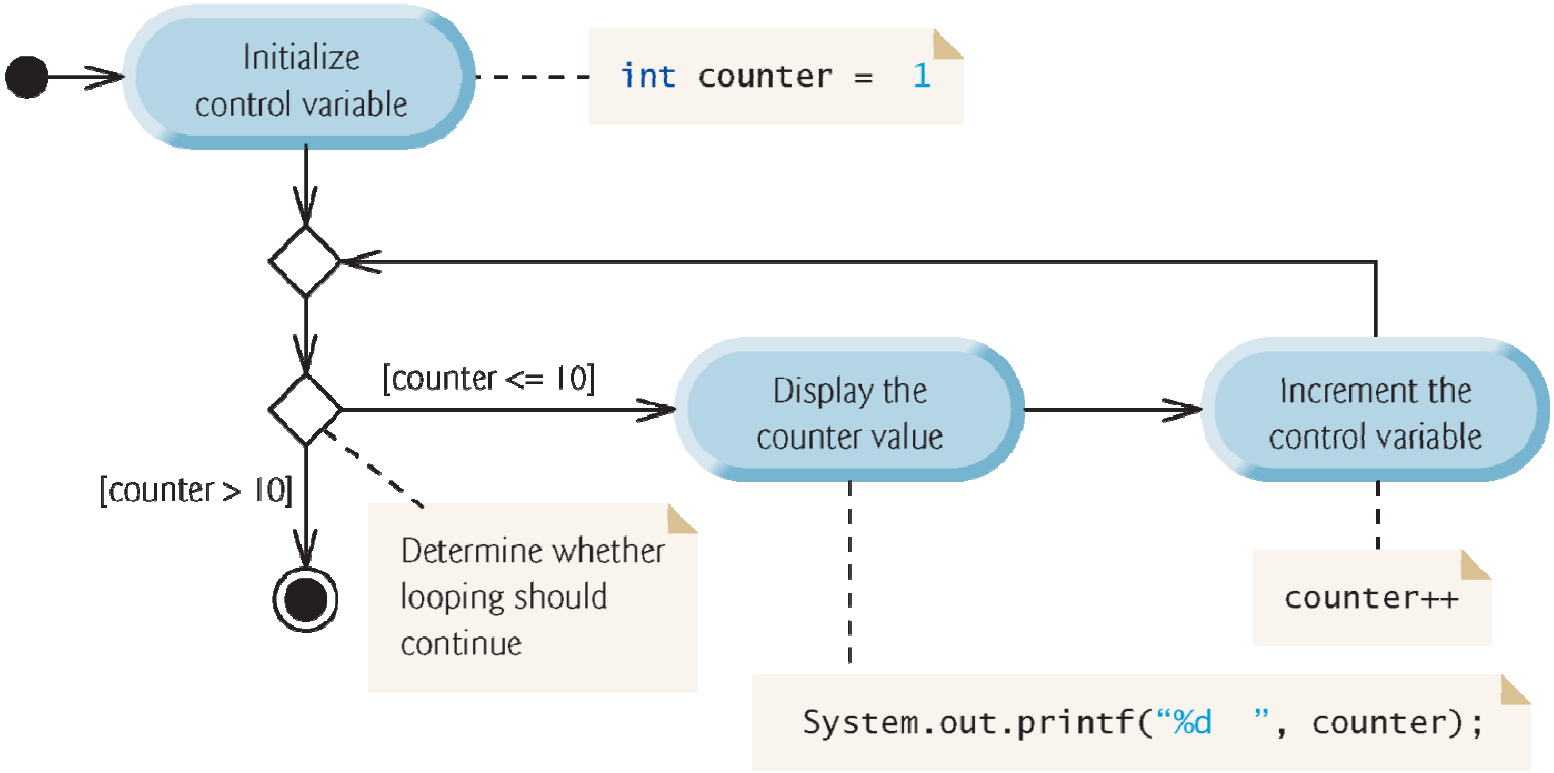
Examples Using the for Statement

- d)Vary the control variable from 20 to 2 in decrements of 2.
  - `for (int i = 20; i >= 2; i -= 2)`
- e)Vary the control variable over the values 2, 5, 8, 11, 14, 17, 20.
  - `for (int i = 2; i <= 20; i += 3)`
- f)Vary the control variable over the values 99, 88, 77, 66, 55, 44, 33, 22, 11, 0.
  - `for (int i = 99; i >= 0; i -= 11)`

```
1  //            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();
14    }
15 } // end class ForCounter
```

```
1  2  3  4  5  6  7  8  9  10
```

Counter-controlled repetition with the `for` repetition statement.



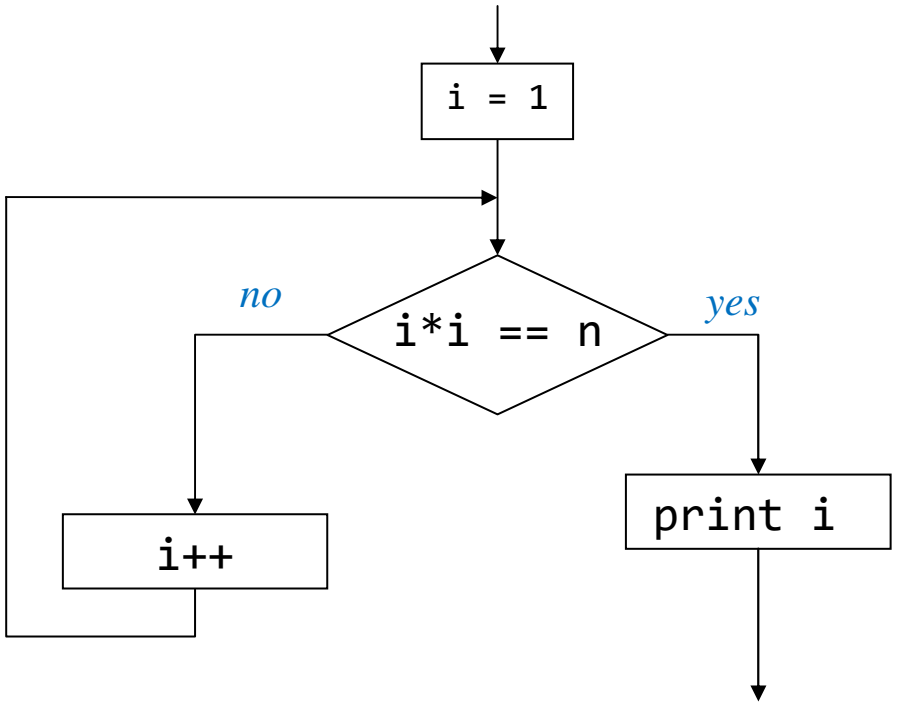
UML activity diagram for the `for` statement

Problem 2: squares root

- Task
  - Find out square root of a perfect square number
- Output
  - Print the square root of a given perfect square number
- Algorithm 3

1. let x = 1;
2. if x*x == the given number, stop the loop;
   else x=x+1 and repeat step 2.
3. print x;

Flowchart: square root



Example: squares

```
public class Squares {

    public static void main(String[] args)
{
        int n = 81;
        int i;
        for (i=1; i*i!=n; i++)
           {}

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

`do…while` Repetition Statement

- The **`do…while` repetition statement** is similar to the **`while`** statement.
- In the **`while`**, the program tests the loop-continuation condition at the *beginning* of the loop, *before* executing the loop's body; if the condition is *false*, the body *never* executes.
- The **`do…while`** statement tests the loop-continuation condition *after* executing the loop's body; therefore, *the body always executes at least once*.
- When a **`do…while`** statement terminates, execution continues with the next statement in sequence.

`do…while` Repetition Statement

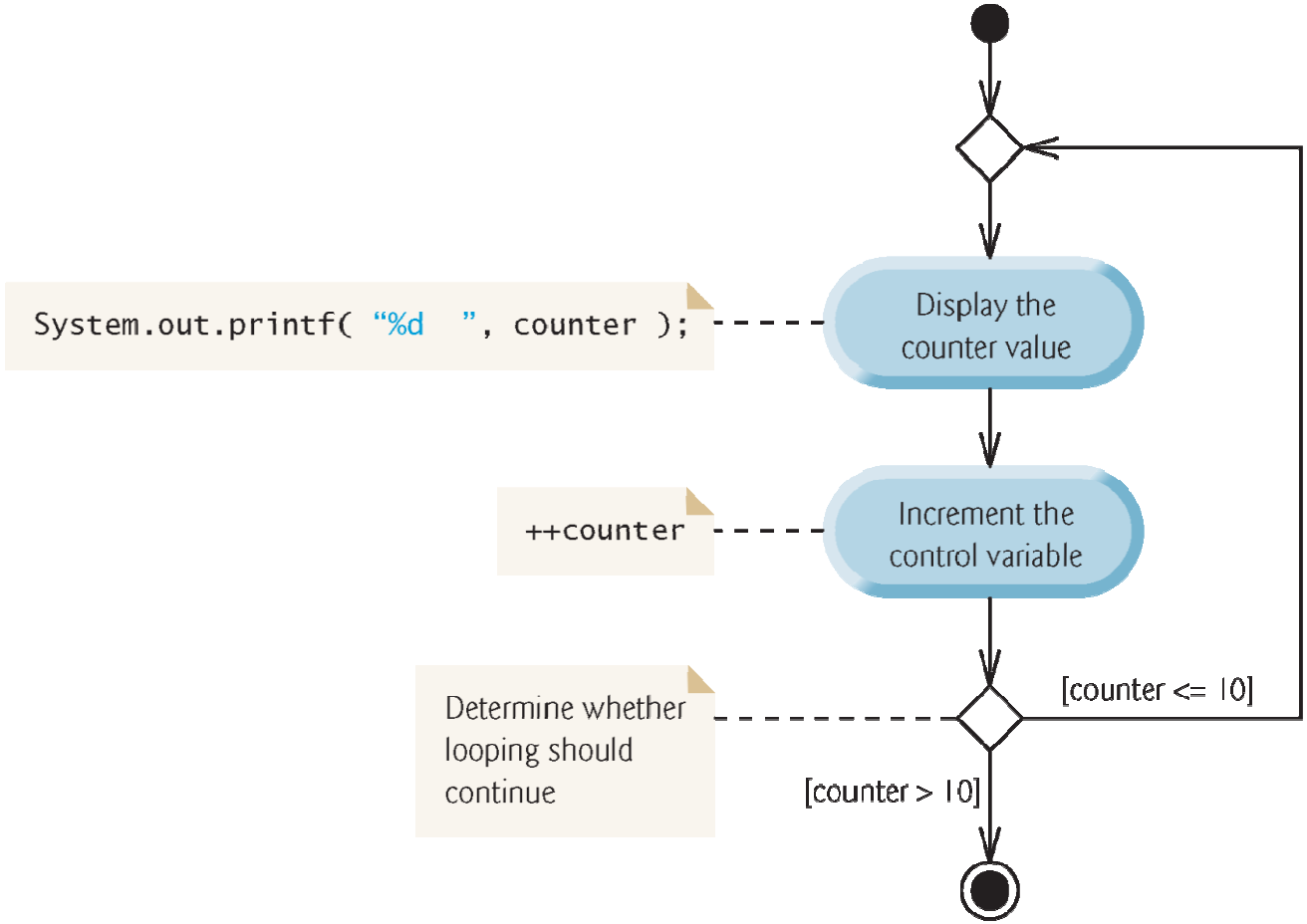
- Braces are not required in the **`do…while`** repetition statement if there's only one statement in the body.
- Most programmers include the braces, to avoid confusion between the while and **`do…while`** statements.
- Thus, the **`do…while`** statement with one body statement is usually written as follows:

```
do
{
   statement/s
} while (condition);
```

```
1  //            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;
9
10       do
11       {
12          System.out.printf("%d  ", counter);
13          ++counter;
14       } while (counter <= 10); // end do...while
15
16       System.out.println();
17    }
18 } // end class DoWhileTest
```

```
1  2  3  4  5  6  7  8  9  10
```

do...while repetition statement.



`do...while` repetition statement UML activity diagram.

Problem 2: square root

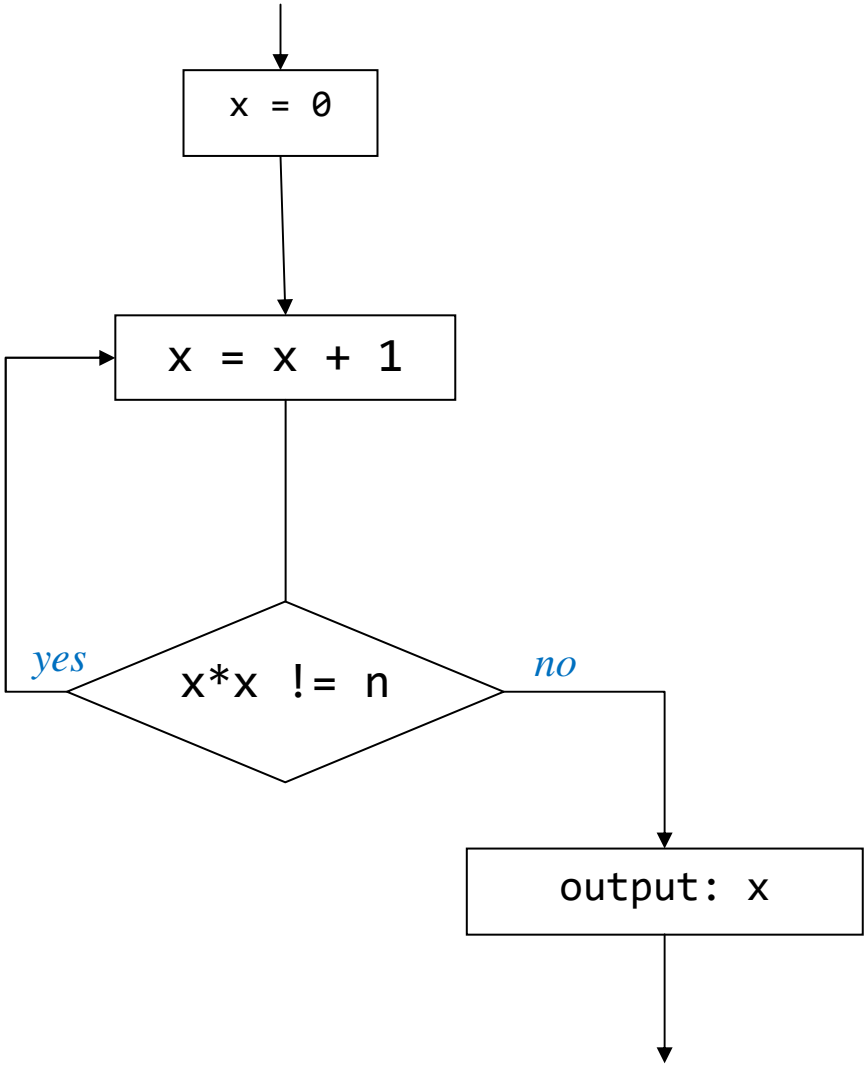
- Task
  - Find out square root of a perfect square number
- Target
  - Print the square root of a given perfect square number
- Algorithm 3

1. let x = 0;
2. do

   x=x+1;

   if x*x != the given perfect square number, repeat step 2;
3. else, print x;

Flowchart: square root



Example: square root

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

      int n = 16;          // a perfect square
      int x = 0;

      do {
          x++;
      } while (x*x != n)

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

`break` and `continue` Statements

- The **`break`** statement, when executed in a **`while`**, **`for`**, **`do…while`** or **`switch`**, causes immediate exit from that statement.
- Execution continues with the first statement after the control statement.
- Common uses of the **`break`** statement are to escape early from a loop or to skip the remainder of a **`switch`**.

```
1  // [illegible]  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 after loop terminates
8
9        for (count = 1; count <= 10; count++) // loop 10 times
10       {
11          if (count == 5)
12             break; // terminates loop if count is 5
13
14          System.out.printf("%d ", count);
15       }
16
17       System.out.printf("%nBroke out of loop at count = %d%n", count);
18    }
19 } // end class BreakTest
```

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

break statement exiting a for statement.

`break` and `continue` statements

- The **`continue`** statement, when executed in a **`while`**, **`for`** or **`do…while`**, skips the remaining statements in the loop body and proceeds with the *next iteration* of the loop.
- In **`while`** and **`do…while`** statements, the program evaluates the loop-continuation test immediately after the continue statement executes.
- In a **`for`** statement, the increment expression executes, then the program evaluates the loop-continuation test.

```
1  // 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++) // loop 10 times
8        {
9           if (count == 5)
10             continue; // skip remaining code in loop body if count is 5
11
12          System.out.printf("%d ", count);
13       }
14
15       System.out.printf("%nUsed continue to skip printing 5%n");
16    }
17 } // end class ContinueTest
```

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

continue statement terminating an iteration of a for statement.

Nested loops

- Just as a selection structure can be nested within another selection structure (or within a loop), a loop can also be nested
- When one loop is nested within another, each iteration of the “outer” loop contains several iterations of the “inner” loop

Multiplication table program output

```
    |   2   3   4   5   6   7   8   9
----------------------------------------
  2|   4   6   8  10  12  14  16  18
  3|   6   9  12  15  18  21  24  27
  4|   8  12  16  20  24  28  32  36
  5|  10  15  20  25  30  35  40  45
  6|  12  18  24  30  36  42  48  54
  7|  14  21  28  35  42  49  56  63
  8|  16  24  32  40  48  56  64  72
  9|  18  27  36  45  54  63  72  81
```

Nested loops

- A loop placed inside another loop.

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 10; j++) {
        System.out.print("*");
    }
    System.out.println();   // to end the line
}
```

- Output:

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

- The outer loop repeats 5 times; the inner one 10 times.

Nested `for` loop exercise

- What is the output of the following nested **`for`** loops?

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

- Output:

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

Nested for loop exercise

- What is the output of the following nested **`for`** loops?

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print(i);
    }
    System.out.println();
}
```

System.out.print(j);

- Output:

```
1
22
333
4444
55555
```

Output:

```
1
12
123
1234
12345
```

Common errors

- Both of the following sets of code produce *infinite loops*:

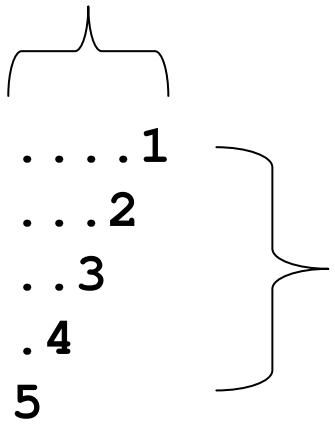
```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; i <= 10; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= 10; i++) {
        System.out.print("*");
    }
    System.out.println();
}
```

Complex lines

- What nested **`for`** loops produce the following output?

inner loop (repeated characters on each line)



```
....1
...2
..3
.4
5
```

outer loop (loops 5 times because there are 5 lines)

- We must build multiple complex lines of output using:
  - an *outer "vertical" loop* for each of the lines
  - *inner "horizontal" loop(s)* for the patterns within each line

Outer and inner loop

- First write the outer loop, from 1 to the number of lines.

```
for (int line = 1; line <= 5; line++) {
    ...
}
```

- Now look at the line contents. Each line has a pattern:
  - some dots (0 dots on the last line), then a number

```
....1
...2
..3
.4
5
```

  - Observation: the number of dots is related to the line number.

Nested `for` loop exercise

- Make a table to represent any patterns on each line.

```
....1
...2
..3
.4
5
```

line	# of dots	-1 * line	-1 * line + 5
1	4	-1	4
2	3	-2	3
3	2	-3	2
4	1	-4	1
5	0	-5	0

- To print a character multiple times, use a **`for`** loop.

```
for (int j = 1; j <= 4; j++) {
    System.out.print(".");          // 4 dots
}
```

Somehow modify this number

Nested for loop solution

- Answer:

```
for (int i = 1; line <= 5; line++) {
    for (int j = 1; j <= (-1 * line + 5); j++){
        System.out.print(".");
    }
    System.out.println(line);
}
```

- Output:

```
....1
...2
..3
.4
5
```

Nested for loop exercise

- What is the output of the following nested **`for`** loops?

```
for (int line = 1; line <= 5; line++) {
    for (int j = 1; j <= (-1 * line + 5); j++) {
        System.out.print(".");
    }
    for (int k = 1; k <= line; k++) {
        System.out.print(line);
    }
    System.out.println();
}
```

- Answer:

```
....1
...22
..333
.4444
55555
```

Nested for loop exercise

- Modify the previous code to produce this output:

```
....1
...2.
..3..
.4...
5....
```

- Answer:

```
for (int line = 1; line <= 5; line++) {
    for (int j = 1; j <= (-1 * line + 5); j++) {
        System.out.print(".");
    }
    System.out.print(line);
    for (int j = 1; j <= (line - 1); j++) {
        System.out.print(".");
    }
    System.out.println();
}
```

Writing an Indefinite Loop – Sentinel-Controlled Repetition

- Sentinel-controlled repetition is often called indefinite repetition because the number of repetitions is not known before the loop begins executing.
- A special value called a sentinel value (also called a signal value, a dummy value or a flag value) can be used to indicate “end of data entry.”
- A sentinel value must be chosen that cannot be confused with an acceptable input value.

Writing an Indefinite Loop – Sentinel-Controlled Repetition

```
public class SumInputs {
   public static void main(String[] args)
   {
        int num, sum=0;
        Scanner scan=new Scanner(System.in);

        System.out.println("insert int(0 for end):");
        num=scan.nextInt();

        while(num!=0){
              sum+=num;
              num=scan.nextInt();
        }
        System.out.println ("sum="+sum);
   }
}
```

Suggested reading

Java: How to Program (Early Objects), 10th Edition

- Chapter 4: Control statements: Part 1
  - 4.8 - 4.14
- Chapter 5: Control statements: Part 2
  - 5.1 – 5.10