CHAPTER 4

LOOPS

Slides by Donald W. Smith
TechNeTrain.com

Final Draft
Oct 30, 2011

Chapter Goals

- To implement `while`, `for`, and `do` loops
- To hand-trace the execution of a program
- To become familiar with common loop algorithms
- To understand nested loops
- To implement programs that read and process data sets
- To use a computer for simulations

In this chapter, you will learn about loop statements in Java, as well as techniques for writing programs that simulate activities in the real world.

Contents

4.1 The `while` Loop

- Examples of loop applications
  - Calculating compound interest
  - Simulations, event driven programs…
- Compound interest algorithm (Chapter 1)

Start with a year value of 0, a column for the interest, and a balance of $10,000.

year	interest	balance
0		$10,000

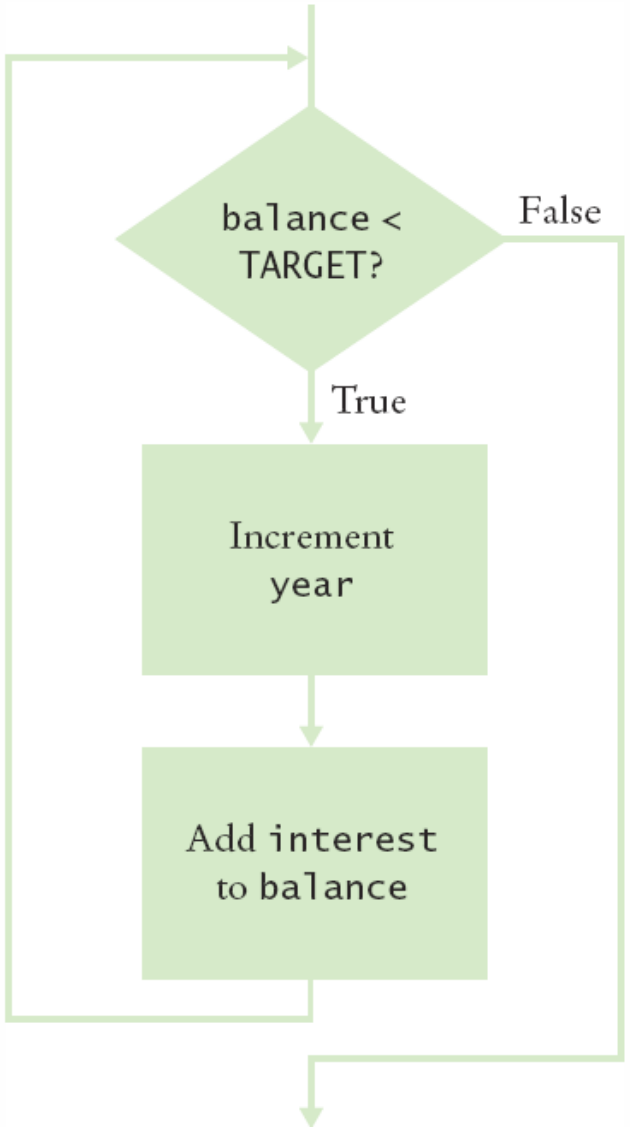
Repeat the following steps while the balance is less than $20,000.

Steps

Add 1 to the year value.
Compute the interest as balance x 0.05 (i.e, 5 percent interest).
Add the interest to the balance.

Report the final year value as the answer.

Planning the `while` Loop



A loop executes instructions repeatedly while a condition is true.

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE/100;
   balance = balance + interest;
}
```

Syntax 4.1: `while` Statement

This variable is declared outside the loop and updated in the loop.

If the condition never becomes false, an infinite loop occurs. See page 145.

Beware of "off-by-one" errors in the loop condition. See page 145.

Don't put a semicolon here! See page 86.

```
double balance = 0;
.
.
.
while (balance < TARGET)
{
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

This variable is created in each loop iteration.

These statements are executed while the condition is true.

Lining up braces is a good idea. See page 86.

Braces are not required if the body contains a single statement, but it's good to always use them. See page 86.

Execution of the Loop

1 Check the loop condition

The condition is true

4 After 15 iterations

balance = 20789.28

year = 15

The condition is no longer true

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

5 Execute the statement following the loop

balance = 20789.28

year = 15

```
while (balance < TARGET)
{
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
System.out.println(year);
```

balance = 10500

year = 1

```
   year++;
   double interest = balance * RATE / 100;
   balance = balance + interest;
}
```

DoubleInvestment.java

```
1  /**
2     This program computes the time required to double an investment.
3  */
4  public class DoubleInvestment
5  {
6     public static void main(String[] args)
7     {
8        final double RATE = 5;
9        final double INITIAL_BALANCE = 10000;
10       final double TARGET = 2 * INITIAL_BALANCE;
11
12       double balance = INITIAL_BALANCE;
13       int year = 0;
14
15       // Count the years required for the investment to double
16
17       while (balance < TARGET)
18       {
19          year++;
20          double interest = balance * RATE / 100;
21          balance = balance + interest;
22       }
23
24       System.out.println("The investment doubled after "
25          + year + " years.");
26    }
27 }
```

Declare and initialize a variable outside of the loop to count `years`

Increment the `years` variable each time through

Program Run

```
The investment doubled after 15 years.
```

while Loop Examples (1)

Loop	Output	Explanation

`i = 0; sum = 0;`
`while (sum < 10)`
`{`
`  i++; sum = sum + i;`
`  Print i and sum;`
`}`
	1 1
	2 3
	3 6
	4 10
		When sum is 10, the loop condition is false, and the loop ends.

`i = 0; sum = 0;`
`while (sum < 10)`
`{`
`  i++; sum = sum - i;`
`  Print i and sum;`
`}`
	1 -1
	2 -3
	3 -6
	4 -10
	. . .
		Because sum never reaches 10, this is an "infinite loop" (see Common Error 4.2 on page 132).

`i = 0; sum = 0;`
`while (sum < 0)`
`{`
`  i++; sum = sum - i;`
`  Print i and sum;`
`}`
	(No output)	The statement sum < 0 is false when the condition is first checked, and the loop is never executed.

while Loop Examples (2)

Loop	Output	Explanation

`i = 0; sum = 0;`
`while (sum >= 10)`
`{`
`  i++; sum = sum + i;`
`  Print i and sum;`
`}`
	(No output)	The programmer probably thought, "Stop when the sum is at least 10." However, the loop condition controls when the loop is executed, not when it ends (see Common Error 4.1 on page 132).

`i = 0; sum = 0;`
`while (sum < 10) ;`
`{`
`  i++; sum = sum + i;`
`  Print i and sum;`
`}`
	(No output, program does not terminate)	Note the semicolon before the `{`. This loop has an empty body. It runs forever, checking whether `sum < 0` and doing nothing in the body.

Common Error 4.1

- Don't think "Are we there yet?"
  - The loop body will only execute if the test condition is **True**.
  - "Are we there yet?" should continue if **False**
  - If `bal` should grow until it reaches `TARGET`
    - Which version will execute the loop body?

```
while (bal < TARGET)
{
   year++;
   interest = bal * RATE;
   bal = bal + interest;
}
```

```
while (bal >= TARGET)
{
   year++;
   interest = bal * RATE;
   bal = bal + interest;
}
```

Common Error 4.2

- Infinite Loops
  - The loop body will execute until the test condition becomes **False**.
  - What if you forget to update the test variable?
    - `bal` is the test variable (`TARGET` doesn't change)
    - You will loop forever! (or until you stop the program)

```
while (bal < TARGET)
{
   year++;
   interest = bal * RATE;
   bal = bal + interest;
}
```

Common Error 4.3

- Off-by-One Errors
  - A 'counter' variable is often used in the test condition
  - Your counter can start at 0 or 1, but programmers often start a counter at 0
  - If I want to paint all 5 fingers, when I am done?
    - **Start at 0, use <**

```
int finger = 0;
final int FINGERS = 5;
while (finger < FINGERS)
{
   // paint finger
   finger++;
}
```

0, 1, 2, 3, 4

Start at 1, use <=

```
int finger = 1;
final int FINGERS = 5;
while (finger <= FINGERS)
{
   // paint finger
   finger++;
}
```

1, 2, 3, 4, 5

4.2: Hand-Tracing

n	sum	digit
1729	0	

```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
System.out.println(sum);
```

- Example: Calculate the sum of digits (1+7+2+9)
  - Make columns for key variables (n, sum, digit)
  - Examine the code and number the steps
  - Set variables to state before loop begins

Tracing Sum of Digits

n	sum	digit
1729	~~0~~	
	9	9

```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
System.out.println(sum);
```

- Start executing loop body statements changing variable values on a new line
  - Cross out values in previous line

Tracing Sum of Digits

n	sum	digit
~~1729~~	~~0~~	
172	9	9

```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
System.out.println(sum);
```

- Continue executing loop statements changing variables
  - 1729 / 10 leaves 172 (no remainder)

Tracing Sum of Digits

n	sum	digit
~~1729~~	~~0~~	
~~172~~	~~9~~	~~9~~
17	11	2

```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
System.out.println(sum);
```

- Test condition. If true, execute loop again
  - Variable n is 172, Is 172 > 0?, True!
- Make a new line for the second time through and update variables

Tracing Sum of Digits

n	sum	digit
1729	0	
172	9	9
17	11	2
1	18	7

```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
System.out.println(sum);
```

- Third time through
  - Variable n is 17 which is still greater than 0
- Execute loop statements and update variables

Tracing Sum of Digits

n	sum	digit
1729	0	
172	9	9
17	11	2
1	18	7
0	19	1

```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
System.out.println(sum);
```

- Fourth loop iteration:
  - Variable n is 1 at start of loop. 1 > 0? True
  - Executes loop and changes variable n to 0 (1/10 = 0)

Tracing Sum of Digits

n	sum	digit
1729	0	
172	9	9
17	11	2
1	18	7
0	19	1

```
int n = 1729;
int sum = 0;
while (n > 0)
{
   int digit = n % 10;
   sum = sum + digit;
   n = n / 10;
}
System.out.println(sum);
```

- Because n is 0, the expression `(n > 0)` is False
- Loop body is not executed
  - Jumps to next statement after the loop body
- Finally prints the sum!

Summary of the `while` Loop

- `while` loops are very commonly used
  - Initialize variables before you test
  - The condition is tested BEFORE the loop body
    - This is called *pre-test*
    - The condition often uses a counter variable
  - Something inside the loop should change one of the variables used in the test
- Watch out for infinite loops!

4.3 The for Loop

- Use a for loop when you:
  - Can use an integer counter variable
  - Have a constant increment (or decrement)
  - Have a fixed starting and ending value for the counter

```
int i = 5;  // initialize
while (i <= 10)  // test
{
   sum = sum + 1;
   i++; // update
}
```

while version

Use a for loop when a value runs from a starting point to an ending point with a constant increment or decrement.

```
for (int i = 5; i <= 10; i++)
{
   sum = sum + i;
}
```

for version

Execution of a for Loop

1 Initialize counter

counter = 1

```
for (counter = 1; counter <= 10; counter++)
{
   System.out.println(counter);
}
```

2 Check condition

counter = 3

```
for (counter = 1; counter <= 10; counter++)
{
   System.out.println(counter);
}
```

3 Execute loop body

counter = 3

```
for (counter = 1; counter <= 10; counter++)
{
   System.out.println(counter);
}
```

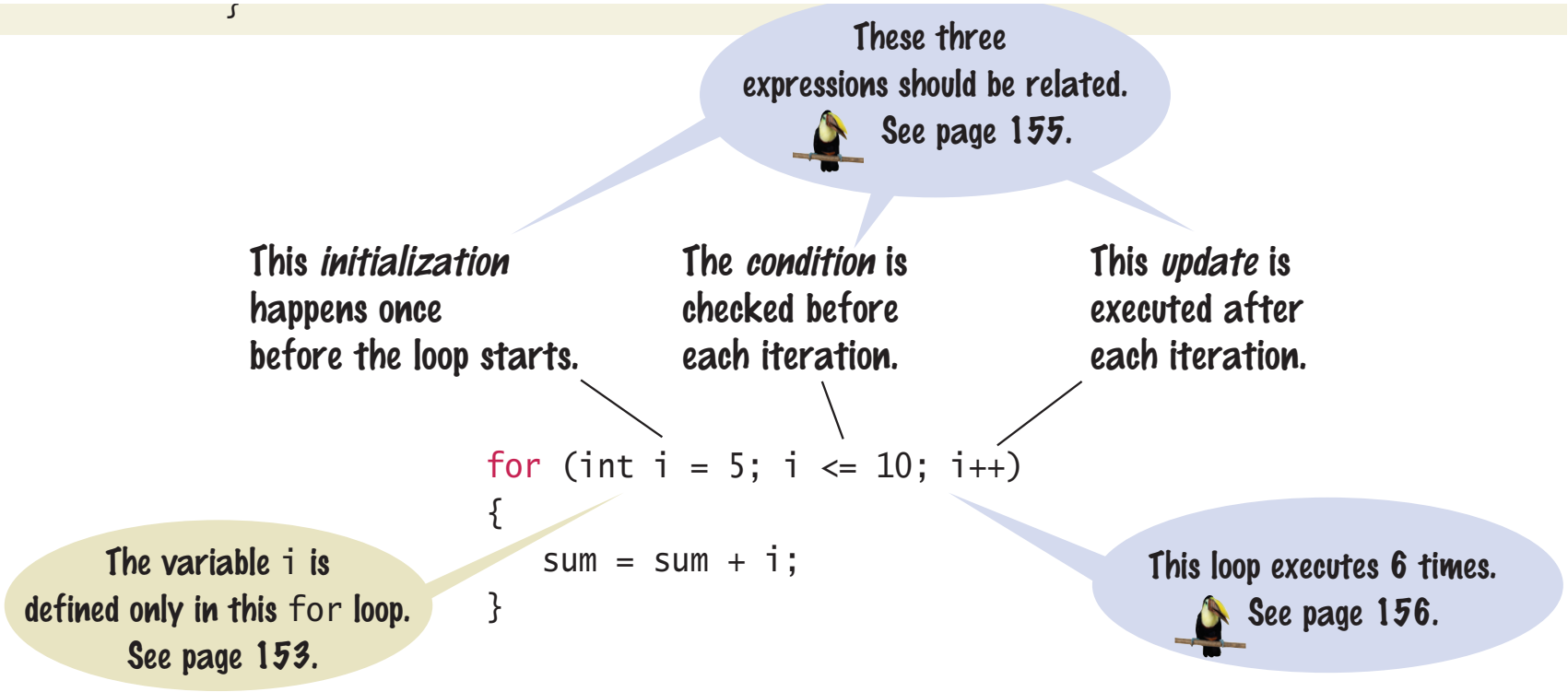
4 Update counter

counter = 3

```
for (counter = 1; counter <= 10; counter++)
{
   System.out.println(counter);
}
```

Syntax 4.2: for Statement

- Two semicolons separate the three parts
  - Initialization ; Condition ; Update



When to use a `for` Loop?

- Yes, a `while` loop can do everything a `for` loop can do
- Programmers like it because it is concise
  - Initialization
  - Condition
  - Update
  - All on one line!

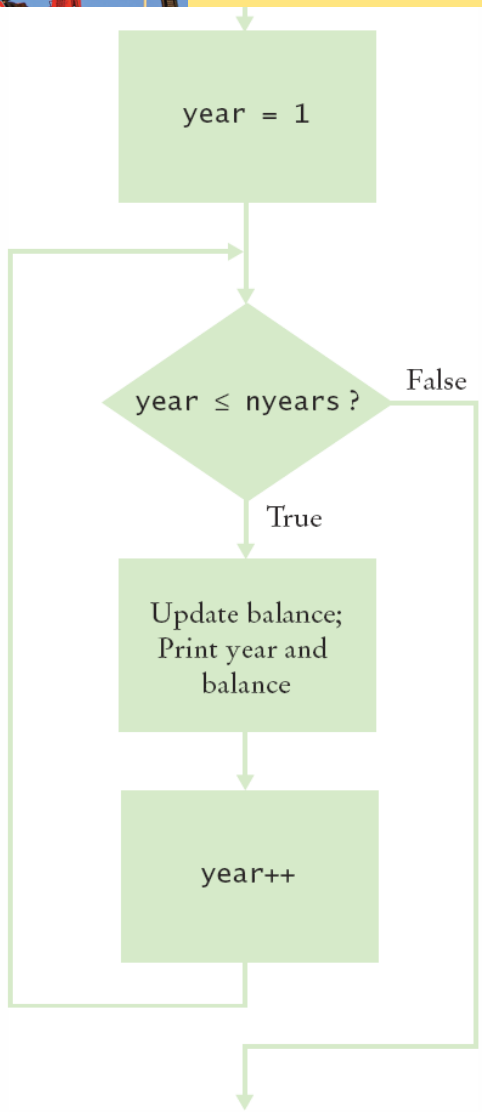
In general, the `for` loop:

```
for (initialization; condition; update)
{
   statements
}
```

has exactly the same effect as the `while` loop:

```
initialization;
while (condition)
{
   statements
   update
}
```

Planning a `for` Loop



- Print the balance at the end of each year for a number of years

Year	Balance
1	10500.00
2	11025.00
3	11576.25
4	12155.06
5	12762.82

```
for (int year = 1; year <= nyears; year++)
{
   Update balance.
   Print year and balance.
}
```

InvestmentTable.java

```
1  import java.util.Scanner;
2
3  /**
4     This program prints a table showing the growth of an investment.
5  */
6  public class InvestmentTable
7  {
8     public static void main(String[] args)
9     {
10       final double RATE = 5;
11       final double INITIAL_BALANCE = 10000;
12       double balance = INITIAL_BALANCE;
13
14       System.out.print("Enter number of years: ");
15       Scanner in = new Scanner(System.in);
16       int nyears = in.nextInt();
17
18       // Print the table of balances for each year
19
20       for (int year = 1; year <= nyears; year++)
21       {
22          double interest = balance * RATE / 100;
23          balance = balance + interest;
24          System.out.printf("%4d %10.2f\n", year, balance);
25       }
26    }
27 }
```

- Setup variables
- Get input
- Loop
  - Calc
  - Output

Good Examples of for Loops

Table 2 for Loop Examples

Loop	Values of i	Comment
`for (i = 0; i <= 5; i++)`	0 1 2 3 4 5	Note that the loop is executed 6 times. (See Programming Tip 4.4 on page 153.)
`for (i = 5; i >= 0; i--)`	5 4 3 2 1 0	Use `i--` for decreasing values.
`for (i = 0; i < 9; i = i + 2)`	0 2 4 6 8	Use `i = i + 2` for a step size of 2.
`for (i = 0; i != 9; i = i + 2)`	0 2 4 6 8 10 12 14 ... (infinite loop)	You can use `<` or `<=` instead of `!=` to avoid this problem.
`for (i = 1; i <= 20; i = i * 2)`	1 2 4 8 16	You can specify any rule for modifying `i`, such as doubling it in every step.
`for (i = 0; i < str.length(); i++)`	0 1 2 . . . until the last valid index of the string `str`	In the loop body, use the expression `str.charAt(i)` to get the `i`th character.

- Keep it simple!

for Loop variable Scope

```
for( int x = 1; x < 10; x = x + 1) {
   // steps to do inside the loop
   // You can use 'x' anywhere in this box
}
if (x > 100)   // Error! x is out of scope!
```

- Scope is the 'lifetime' of a variable.
- When 'x' is declared in the for statement:
  - 'x' exists only inside the 'block' of the for loop { }
- Solution: Declare 'x' outside the for loop

```
int x;
for(x = 1; x < 10; x = x + 1)
```

Programming Tip 4.1

- Use `for` loops for their intended purposes only
  - Increment (or decrement) by a constant value
  - Do not update the counter inside the body
    - Update in the third section of the header

```
for (int counter = 1; counter <= 100; counter++)
{
   if (counter % 10 == 0) // Skip values divisible by 10
   {
      counter++; // Bad style: Do NOT update the counter inside loop
   }
   System.out.println(counter);
}
```

- Most counters start at one ‘end’ (0 or 1)
  - Many programmers use an integer named `i` for ‘index’ or ‘counter’ variable in `for` loops

Programming Tip 4.3

- Count Iterations
  - Many bugs are ‘off by one’ issues
  - One too many or one too few
- How many posts are there?
- How many pairs of rails are there?

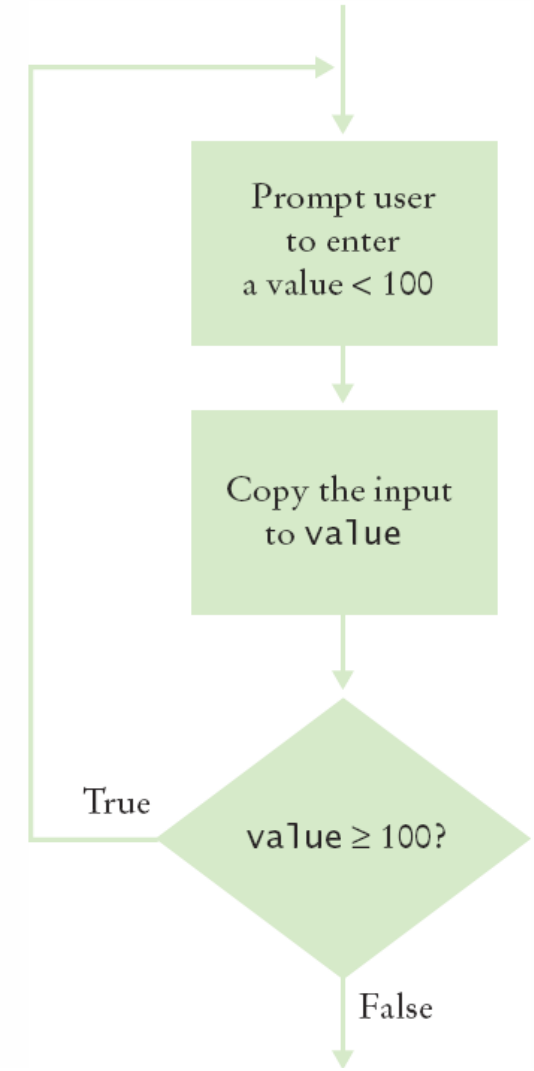
```
final int RAILS = 5;
for (int i = 1; i < RAILS; i++ )
{
   System.out.println("Painting rail " + i);
}
```

```
Painting rail 1
Painting rail 2
Painting rail 3
Painting rail 4
```

Summary of the `for` Loop

- `for` loops are very commonly used
- They have a very concise notation
  - Initialization ; Condition ; Update
  - Initialization happens once at the start
  - Condition is tested every time BEFORE executing the body (*pre-test*)
  - Increment is done at the end of the body
- Great for integer counting, String (array) processing and more

4.4 The do Loop



Use a do loop when you want to:

- Execute the body at least once
- Test the condition AFTER your first loop

```
int i = 1;  // initialize
final int FINGERS = 5;
do
{
   // paint finger
   i++; // update
}
while (i <= FINGERS);  // test
```

Note the semicolon at the end!

do Loop Example

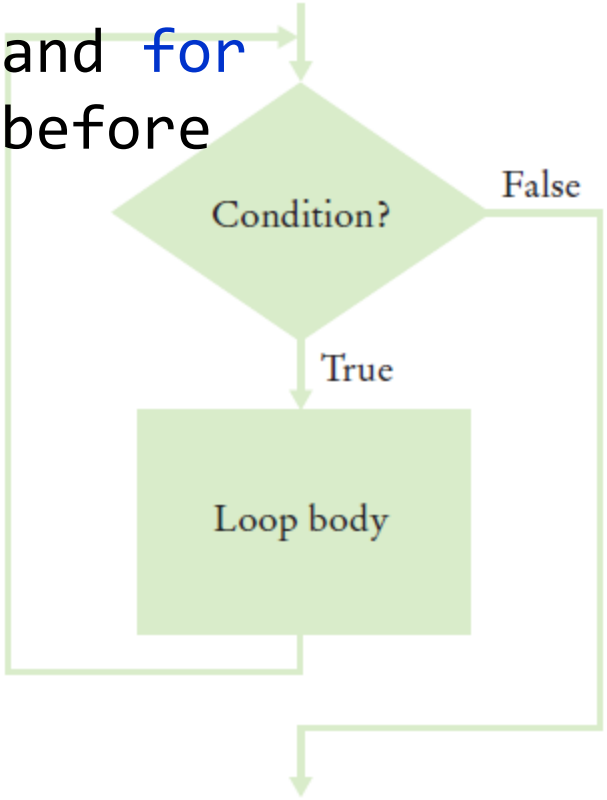
- User Input Validation:
  - Range check a value entered
  - User must enter something to validate first!

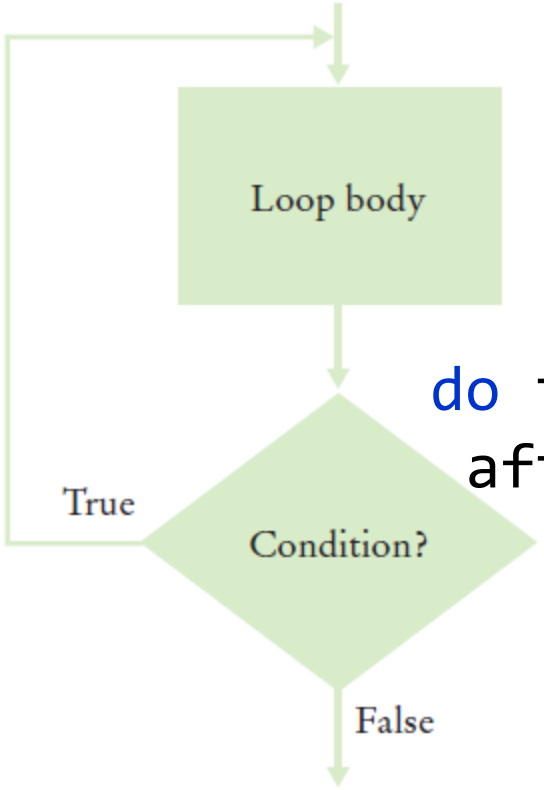
```
int value;
do
{
   System.out.println("Enter an integer < 100: ");
   value = in.nextInt();
}
while (value >= 100);  // test
```

Programming Tip 4.4

- Flowcharts for loops
  - To avoid 'spaghetti code', never have an arrow that points inside the loop body

`while` and `for` test before





`do` tests after

4.5 Processing Sentinel Values

Sentinel values are often used:

- When you don't know how many items are in a list, use a 'special' character or value to signal no more items.
- For numeric input of positive numbers, it is common to use the value -1:

```
salary = in.nextDouble();
while (salary != -1)
{
   sum = sum + salary;
   count++;
   salary = in.nextDouble();
}
```

A sentinel value denotes the end of a data set, but it is not part of the data.

Averaging a set of values

- Declare and initialize a ‘sum’ variable to 0
- Declare and initialize a ‘count’ variable to 0
- Declare and initialize an ‘input’ variable to 0
- Prompt user with instructions
- Loop until sentinel value is entered
  - Save entered value to input variable
  - If input is not -1 (sentinel value)
    - Add input to sum variable
    - Add 1 to count variable
- Make sure you have at least one entry before you divide!
  - Divide sum by count and output. Done!

SentinelDemo.java (1)

```
 8    public static void main(String[] args)
 9    {
10       double sum = 0;
11       int count = 0;
12       double salary = 0;
13       System.out.print("Enter salaries, -1 to finish: ");
14       Scanner in = new Scanner(System.in);
15
16       // Process data until the sentinel is entered
17
18       while (salary != -1)
19       {
20          salary = in.nextDouble();
21          if (salary != -1)
22          {
23             sum = sum + salary;
24             count++;
25          }
26       }
27
```

Outside the while loop, declare and initialize variables to use

Since `salary` is initialized to 0, the while loop statements will be executed

Input new `salary` and compare to sentinel

Update running `sum` and `count` to average later

SentinelDemo.java (2)

```
28          // Compute and print the average
29
30          if (count > 0)
31          {
32             double average = sum / count;
33             System.out.println("Average salary: " + average);
34          }
35          else
36          {
37             System.out.println("No data");
38          }
39       }
40    }
```

Prevent divide by 0

Calculate and output the average salary using `sum` and `count` variables

Program Run

```
Enter salaries, -1 to finish: 10 10 40 -1
Average salary: 20
```

Boolean variables and sentinels

- A boolean variable can be used to control a loop
  - Sometimes called a ‘flag’ variable

```
System.out.print("Enter salaries, -1 to finish: ");
boolean done = false;
while (!done)
{
   value = in.nextDouble();
   if (value == -1)
   {
      done = true;
   }
   else
   {
      // Process value
   }
}
```

Initialize done so that loop will execute

Set done ‘flag’ to true if sentinel value is found

To input any numeric value…

- When valid values can be positive or negative
  - You cannot use -1 (or any other number) as a sentinel
- One solution is to use a non-numeric sentinel
  - But Scanner's `in.nextDouble` will fail!
  - Use Scanner's `in.hasNextDouble` first
    - Returns a `boolean`: `true` (all's well) or `false` (not a number)
    - Then use `in.nextDouble` if `true`

```
System.out.print("Enter values, Q to quit: ");
while (in.hasNextDouble())
{
   value = in.nextDouble();
   // Process value
}
```

4.6 Storyboards

- One useful problem solving technique is the use of storyboards to model user interaction. It can help answer:
  - What information does the user provide, and in which order?
  - What information will your program display, and in which format?
  - What should happen when there is an error?
  - When does the program quit?

A storyboard consists of annotated sketches for each step in an action sequence.

Storyboard Example

- Goal: Converting a sequence of values
  - Will require a loop and some variables
  - Handle one conversion each time through the loop

Converting a Sequence of Values

What unit do you want to convert from? cm
What unit do you want to convert to? in
Enter values, terminated by zero — Allows conversion of multiple values
30
30 cm = 11.81 in — Format makes clear what got converted
100
100 cm = 39.37 in
0
What unit do you want to convert from?

What can go wrong?

- Unknown unit types
  - How do you spell centimeters and inches?
  - What other conversions are available?
  - Solution:
    - Show a list of the acceptable unit types

```
From unit (in, ft, mi, mm, cm, m, km, oz, lb, g, kg, tsp, tbsp, pint, gal): cm
To unit: in
```

No need to list the units again

What else can go wrong?

- How does the user quit the program?

Exiting the Program

```
From unit (in, ft, mi, mm, cm, m, km, oz, lb, g, kg, tsp, tbsp, pint, gal): cm
To unit: in
Enter values, terminated by zero
30
30 cm = 11.81 in
0
More conversions (y, n)? n
(Program exits)
```

Sentinel triggers the prompt to exit

- Storyboards help you plan a program
  - Knowing the flow helps you structure your code

4.7 Common Loop Algorithms

1: Sum and Average Value

2: Counting Matches

3: Finding the First Match

4: Prompting until a match is found

5: Maximum and Minimum

6: Comparing Adjacent Values

Sum and Average Examples

```
double total = 0;
while (in.hasNextDouble())
{
   double input = in.nextDouble();
   total = total + input;
}
```

- Sum of Values
  - Initialize total to 0
  - Use `while` loop with sentinel

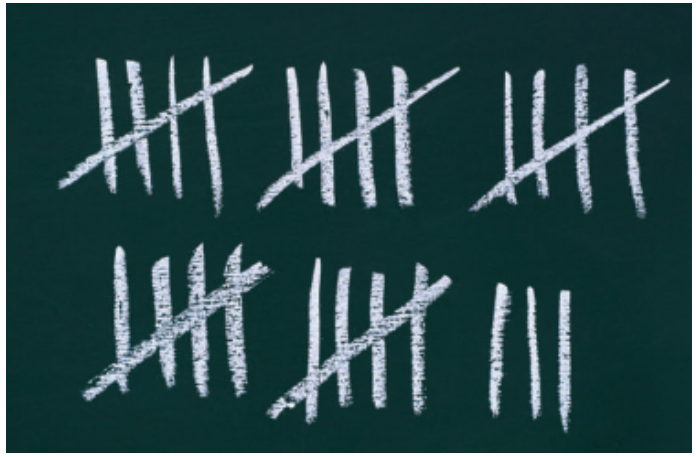
- Average of Values
  - Use Sum of Values
  - Initialize count to 0
    - Increment per input
  - Check for count 0
    - Before divide!

```
double total = 0;
int count = 0;
while (in.hasNextDouble())
{
   double input = in.nextDouble();
   total = total + input;
   count++;
}
double average = 0;
if (count > 0)
  { average = total / count; }
```

Counting Matches

- Counting Matches
  - Initialize count to 0
  - Use a `for` loop
  - Add to count per match

```
int upperCaseLetters = 0;
for (int i = 0; i < str.length(); i++)
{
   char ch = str.charAt(i);
   if (Character.isUpperCase(ch))
   {
      upperCaseLetters++;
   }
}
```



Finding the First Match

```
boolean found = false;
char ch;
int position = 0;
while (!found &&
       position < str.length())
{
   ch = str.charAt(position);
   if (Character.isLowerCase(ch))
   {
      found = true;
   }
   else { position++; }
}
```

A pre-test loop (while or for) will handle the case where the string is empty!

- Initialize `boolean` sentinel to `false`
- Initialize position counter to 0
  - First char in String
- Use a compound conditional in loop

Prompt Until a Match is Found

```
boolean valid = false;
double input;
while (!valid)
{
   System.out.print("Please enter a positive value < 100: ");
   input = in.nextDouble();
   if (0 < input && input < 100) { valid = true; }
   else { System.out.println("Invalid input."); }
}
```

- Initialize `boolean` flag to `false`
- Test sentinel in `while` loop
  - Get input, and compare to range
    - If input is in range, change flag to true
    - Loop will stop executing

Maximum and Minimum

```
double largest = in.nextDouble();
while (in.hasNextDouble())
{
   double input = in.next
   if (input > largest)
   {
      largest = input;
   }
}
```

```
double smallest = in.nextDouble();
while (in.hasNextDouble())
{
   double input = in.nextDouble();
   if (input > smallest)
   {
      smallest = input;
   }
}
```

- Get first input value
  - This is the largest (or smallest) that you have seen so far!
- Loop `while` you have a valid number (non-sentinel)
  - Get another input value
  - Compare new input to largest (or smallest)
  - Update largest (or smallest) if necessary

Comparing Adjacent Values

```
double input = in.nextDouble();
while (in.hasNextDouble())
{
   double previous = input;
   input = nextDouble();
   if (input == previous)
   {
      System.out.println("Duplicate input");
   }
}
```

- Get first input value
- Use `while` to determine if there are more to check
  - Copy input to previous variable
  - Get next value into input variable
  - Compare input to previous, and output if same

Steps to Writing a Loop

Planning:

1. Decide what work to do inside the loop
2. Specify the loop condition
3. Determine loop type
4. Setup variables before the first loop
5. Process results when the loop is finished
6. Trace the loop with typical examples

Coding:

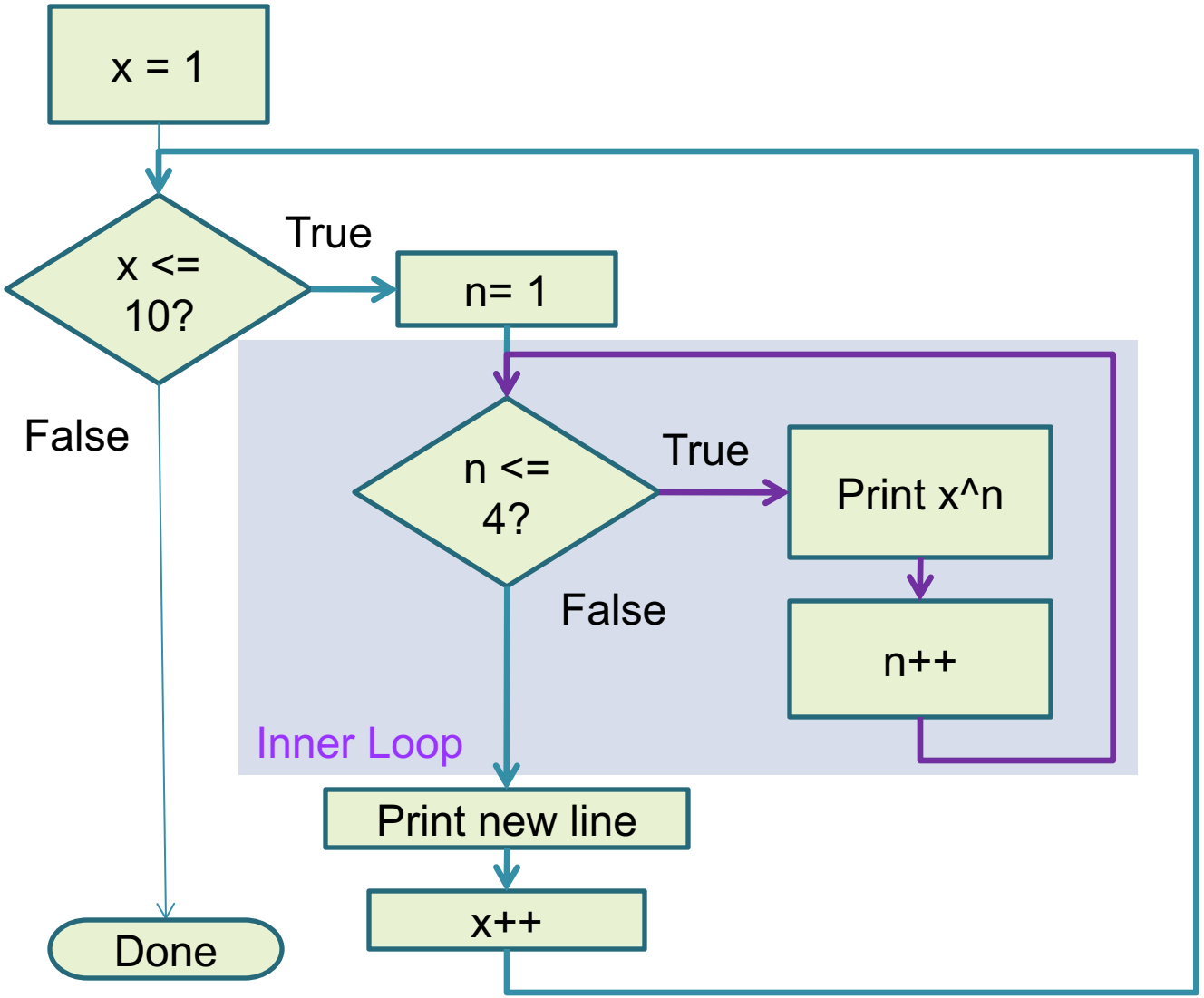
7. Implement the loop in Java

4.8 Nested Loops

- How would you print a table with rows and columns?
  - Print top line (header)
    - Use a `for` loop
  - Print table body…
    - How many rows?
    - How many columns?
  - Loop per row
    - Loop per column

x^1	x^2	x^3	x^4
1	1	1	1
2	4	8	16
3	9	27	81
…	…	…	…
10	100	1000	10000

Flowchart of a Nested Loop



PowerTable.java

```
1  /**
2     This program prints a table of powers of x.
3  */
4  public class PowerTable
5  {
6     public static void main(String[] args)
7     {
8        final int NMAX = 4;
9        final double XMAX = 10;
10
24       // Print table body
25
26       for (double x = 1; x <= XMAX; x++)
27       {
28          // Print table row
29
30          for (int n = 1; n <= NMAX; n++)
31          {
32             System.out.printf("%10.0f", Math.pow(x, n));
33          }
34          System.out.println();
35       }
36    }
37 }
```

```
         1         2         3         4
         x         x         x         x

         1         1         1         1
         2         4         8        16
         3         9        27        81
         4        16        64       256
         5        25       125       625
         6        36       216      1296
         7        49       343      2401
         8        64       512      4096
         9        81       729      6561
        10       100      1000     10000
```

Body of outer loop

Body of inner loop

Nested Loop Examples (1)

```
for (i = 1; i <= 3; i++)
{
   for (j = 1; j <= 4; j++)  { print "*" }
   print new line
}
```

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

Prints 3 rows of 4 asterisks each.

```
for (i = 1; i <= 4; i++)
{
   for (j = 1; j <= 3; j++) { print "*" }
   print new line
}
```

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

Prints 4 rows of 3 asterisks each.

```
for (i = 1; i <= 4; i++)
{
   for (j = 1; j <= i; j++) { print "*" }
   print new line
}
```

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

Prints 4 rows of lengths 1, 2, 3, and 4.

Nested Loop Examples (2)

`for (i = 1; i <= 3; i++)`
`{`
`  for (j = 1; j <= 5; j++)`
`  {`
`  if (j % 2 == 0) { Print "*" }`
`  else { Print "-" }`
`  }`
`  System.out.println();`
`}`
	-*-*-
	-*-*-
	-*-*-
		Prints asterisks in even columns, dashes in odd columns.

`for (i = 1; i <= 3; i++)`
`{`
`  for (j = 1; j <= 5; j++)`
`  {`
`  if (i % 2 == j % 2) { Print "*" }`
`  else { Print " " }`
`  }`
`  System.out.println();`
`}`
	* * *
	* *
	* * *
		Prints a checkerboard pattern.

RandomDemo.java

```
1  /**
2     This program prints ten random numbers between 0 and 1.
3  */
4  public class RandomDemo
5  {
6     public static void main(String[] args)
7     {
8        for (int i = 1; i <= 10; i++)
9        {
10          double r = Math.random();
11          System.out.println(r);
12       }
13    }
14 }
```

Program Run

```
0.2992436267816825
0.4386017604531353 7
0.7365753471168408
0.6880250194282326
0.1608272403783395
0.5362876579988844
0.3098705906424375
0.6602909916554179
0.1927951611482942
0.8632330736331089
```

4.9 Random Numbers/Simulations

- Games often use random numbers to make things interesting
  - Rolling Dice
  - Spinning a wheel
  - Pick a card
- A simulation usually involves looping through a sequence of events
  - Days
  - Events

Simulating Die Tosses

- Goal:
  - Get a random integer between 1 and 6

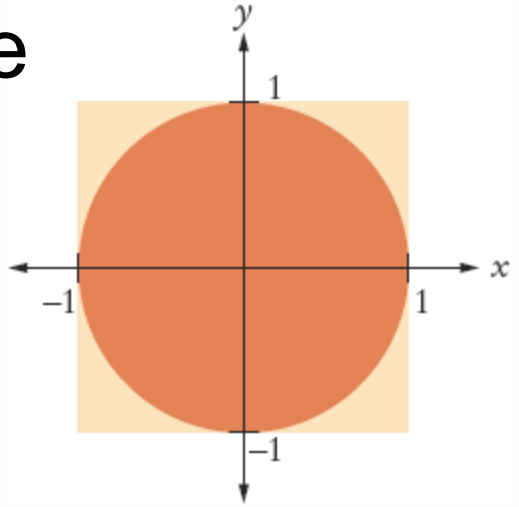
```
1  /**
2     This program simulates tosses of a pair of dice.
3  */
4  public class Dice
5  {
6     public static void main(String[] args)
7     {
8        for (int i = 1; i <= 10; i++)
9        {
10          // Generate two random numbers between 1 and 6
11
12          int d1 = (int) (Math.random() * 6) + 1;
13          int d2 = (int) (Math.random() * 6) + 1;
14          System.out.println(d1 + " " + d2);
15       }
16       System.out.println();
17    }
18 }
```

Program Run

```
5 1
2 1
1 2
5 1
1 2
6 4
4 4
6 1
6 3
5 2
```

The Monte Carlo Method

- Used to find approximate solutions to problems that cannot be precisely solved
- Example: Approximate PI using the relative areas of a circle inside a square
  - Uses simple arithmetic
  - Hits are inside circle
  - Tries are total number of tries
  - Ratio is 4 x Hits / Tries



MonteCarlo.java

```
1  /**
2     This program computes an estimate of pi by simulating dart throws onto a square.
3  */
4  public class MonteCarlo
5  {
6     public static void main(String[] args)
7     {
8        final int TRIES = 10000;
9
10       int hits = 0;
11       for (int i = 1; i <= TRIES; i++)
12       {
13          // Generate two random numbers between -1 and 1
14
15          double r =
16          double x =
17          r = Math.
18          double y =
19
20          // Check
21
22          if (x * x
23       }
24
25          /*
26             The ratio hits / tries is approximately the same as the ratio
27             circle area / square area = pi / 4
28          */
29
30          double piEstimate = 4.0 * hits / TRIES;
31          System.out.println("Estimate for pi: " + piEstimate);
32       }
33 }
```

Program Run

```
Estimate for pi: 3.1504
```

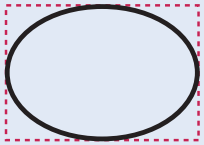
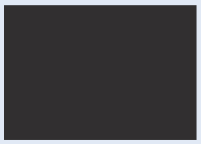
The 'empty' body

- You probably have developed the habit of typing a semicolon at the end of each line
- Don't do this with loop statements!
  - The loop body becomes very short!
    - Between the closing) and ;
    - What type of loop do I have now?

```
while (bal < TARGET);
{
   year++;
   interest = bal * RATE;
   bal = bal + interest;
}
```

Drawing Graphical Shapes

Table 4 Graphics Methods

Method	Result	Notes
`g.drawRect(x, y, width, height)`		`(x, y)` is the top left corner.
`g.drawOval(x, y, width, height)`		`(x, y)` is the top left corner of the box that bounds the ellipse. To draw a circle, use the same value for width and height.
`g.fillRect(x, y, width, height)`		The rectangle is filled in.
`g.fillOval(x, y, width, height)`		The oval is filled in.

Drawing Graphical Shapes

`g.drawLine(x1, y1, x2, y2)`		`(x1, y1)` and `(x2, y2)` are the endpoints.
`g.drawString("Message", x, y)`		`(x, y)` is the basepoint.
`g.setColor(color)`	From now on, draw or fill methods will use this color.	Use `Color.RED`, `Color.GREEN`, `Color.BLUE`, and so on. (See Table 10.1 for a complete list of predefined colors.)

TwoRowsOfSquares.java

```
11    public static void draw(Graphics g)
12    {
13       final int width = 20;
14       g.setColor(Color.BLUE);
15
16       // Top row. Note that the top left corner of the drawing has coordinates (0, 0)
17       int x = 0;
18       int y = 0;
19       for (int i = 0; i < 10; i++)
20       {
21          g.fillRect(x, y, width, width);
22          x = x + 2 * width;
23       }
24       // Second row, offset from the first one
25       x = width;
26       y = width;
27       for (int i = 0; i < 10; i++)
28       {
29          g.fillRect(x, y, width, width);
30          x = x + 2 * width;
31       }
32    }
```

Summary: Types of Loops

- There are three types of loops:
  - `while` Loops
  - `for` Loops
  - `do` Loops
- Each loop requires the following steps:
  - Initialization (setup variables to start looping)
  - Condition (test if we should execute loop body)
  - Update (change something each time through)

Summary

- A loop executes instructions repeatedly while a condition is true.
- An off-by-one error is a common error when programming loops.
  - Think through simple test cases to avoid this type of error.
- The `for` loop is used when a value runs from a starting point to an ending point with a constant increment or decrement.
- The `do` loop is appropriate when the loop body must be executed at least once.

Summary

- A sentinel value denotes the end of a data set, but it is not part of the data.
- You can use a `boolean` variable to control a loop.
  - Set the variable to true before entering the loop, then set it to false to leave the loop.
- When the body of a loop contains another loop, the loops are nested.
  - A typical use of nested loops is printing a table with rows and columns.
- In a simulation, you use the computer to simulate an activity.
  - You can introduce randomness by calling the random number generator.