Building Java Programs

Chapter 2
Lecture 2-2: The `for` Loop

reading: 2.3

Repetition with `for` loops

- So far, repeating a statement is redundant:

```
System.out.println("Homer says:");
System.out.println("I am so smart");
System.out.println("I am so smart");
System.out.println("I am so smart");
System.out.println("I am so smart");
System.out.println("S-M-R-T... I mean S-M-A-R-T");
```
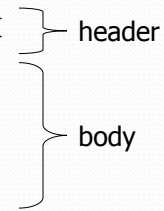
- Java's **`for` loop** statement performs a task many times.

```
System.out.println("Homer says:");
for (int i = 1; i <= 4; i++) {    // repeat 4 times
    System.out.println("I am so smart");
}
System.out.println("S-M-R-T... I mean S-M-A-R-T");
```

for loop syntax

```
for (<initialization>; <test>; <update>) {      } — header
    <statement>;
    <statement>;
    ...
    <statement>;                                  } — body
}
```

- Perform **<initialization>** once.
- Repeat the following:
  - Check if the **<test>** is true. If not, stop.
  - Execute the **<statement>**s.
  - Perform the **<update>**.

3

Initialization

```
for (int i = 1; i <= 4; i++) {
    System.out.println("I am so smart");
}
```

- Tells Java what variable to use in the loop
  - Performed once as the loop begins

  - The variable is called a *loop counter*
    - can use any name, not just i
    - can start at any value, not just 1

4

2

Test

```
for (int i = 1; i <= 4; i++) {
    System.out.println("I am so smart");
}
```

- Tests the loop counter variable against a limit
  - Uses comparison operators:
    - `<`   less than
    - `<=` less than or equal to
    - `>`   greater than
    - `>=` greater than or equal to

Increment and decrement

shortcuts to increase or decrease a variable's value by 1

Shorthand	Equivalent longer version
\<variable>++;	**\<variable>** = **\<variable>** + 1;
\<variable>--;	**\<variable>** = **\<variable>** - 1;

```
int x = 2;
x++;                    // x = x + 1;
                        // x now stores 3

double gpa = 2.5;
gpa--;                  // gpa = gpa - 1;
                        // gpa now stores 1.5
```

Modify-and-assign operators

shortcuts to modify a variable's value

Shorthand	Equivalent longer version
\<variable\> += **\<exp\>**;	**\<variable\>** = **\<variable\>** + (**\<exp\>**);
\<variable\> -= **\<exp\>**;	**\<variable\>** = **\<variable\>** - (**\<exp\>**);
\<variable\> *= **\<exp\>**;	**\<variable\>** = **\<variable\>** * (**\<exp\>**);
\<variable\> /= **\<exp\>**;	**\<variable\>** = **\<variable\>** / (**\<exp\>**);
\<variable\> %= **\<exp\>**;	**\<variable\>** = **\<variable\>** % (**\<exp\>**);

```
x += 3;                // x = x + 3;

gpa -= 0.5;            // gpa = gpa - 0.5;

number *= 2 + 1;       // number = number * (2 + 1);
```

7

`for` loop is **NOT** a method

- The `for` loop is a ***control structure***—a syntactic structure that *controls* the execution of other statements.

- Example:
  - "Shampoo hair. Rinse. **Repeat**."

8

4

Repetition over a range

```
System.out.println("1 squared = " + 1 * 1);
System.out.println("2 squared = " + 2 * 2);
System.out.println("3 squared = " + 3 * 3);
System.out.println("4 squared = " + 4 * 4);
System.out.println("5 squared = " + 5 * 5);
System.out.println("6 squared = " + 6 * 6);
```

- Intuition: "I want to print a line for each number from 1 to 6"

- The `for` loop does exactly that!

```
for (int i = 1; i <= 6; i++) {
      System.out.println(i + " squared = " + (i * i));
}
```

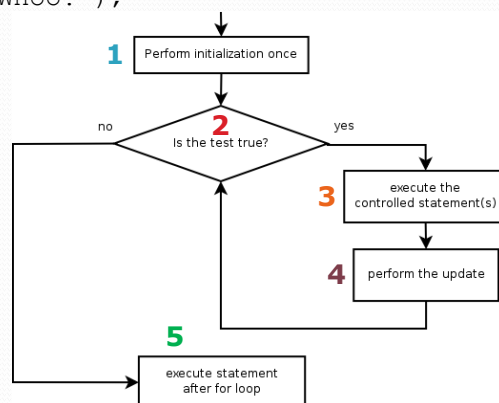- "For each integer **i** from 1 through 6, print ..."

Loop walkthrough

```
         1         2        4
  for (int i = 1; i <= 4; i++) {
    3 System.out.println(i + " squared = " + (i * i));
  }
5 System.out.println("Whoo!");
```

Output:

```
1 squared = 1
2 squared = 4
3 squared = 9
4 squared = 16
Whoo!
```


1 Perform initialization once

2 Is the test true?   no   yes

3 execute the controlled statement(s)

4 perform the update

5 execute statement after for loop

Multi-line loop body

```
System.out.println("+----+");
for (int i = 1; i <= 3; i++) {
    System.out.println("\\     /");
    System.out.println("/     \\");
}
System.out.println("+----+");
```

- Output:
```
+----+
\     /
/     \
\     /
/     \
\     /
/     \
+----+
```

Expressions for counter

```
int highTemp = 5;
for (int i = -3; i <= highTemp / 2; i++) {
    System.out.println(i * 1.8 + 32);
}
```

- This computes the Fahrenheit equivalents for -3 degrees Celsius to 2 degrees Celsius.

- Output:
```
26.6
28.4
30.2
32.0
33.8
35.6
```

System.out.print

- Prints without moving to a new line
  - allows you to print partial messages on the same line

  ```
  int highestTemp = 5;
  for (int i = -3; i <= highestTemp / 2; i++) {
      System.out.print((i * 1.8 + 32) + "  ");
  }
  ```

  - Output:
  ```
  26.6  28.4  30.2  32.0  33.8  35.6
  ```

    - Concatenate " " to separate the numbers

Counting down

- The **<update>** can use -- to make the loop count down.
  - The **<test>** must say > instead of <

  ```
  System.out.print("T-minus ");
  for (int i = 10; i >= 1; i--) {
      System.out.print(i + ", ");
  }
  System.out.println("blastoff!");
  System.out.println("The end.");
  ```

  - Output:
  ```
  T-minus 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, blastoff!
  The end.
  ```

Nested loops

reading: 2.3

Nested loops

- **nested loop**: 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.
  - "sets and reps" exercise analogy

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();
}
```

- Output:
```
1
22
333
4444
55555
```

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();
}
```

19

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

20

10

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.

Mapping loops to numbers

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

- What statement in the body would cause the loop to print:

```
4 7 10 13 16
```

```
for (int count = 1; count <= 5; count++) {
    System.out.print(3 * count + 1 + " ");
}
```

Loop tables

- What statement in the body would cause the loop to print:
  `2 7 12 17 22`

- To see patterns, make a table of `count` and the numbers.
  - Each time count goes up by 1, the number should go up by 5.
  - But `count * 5` is too great by 3, so we subtract 3.

count	number to print	5 * count	5 * count - 3
1	2	5	2
2	7	10	7
3	12	15	12
4	17	20	17
5	22	25	22

23

Loop tables question

- What statement in the body would cause the loop to print:
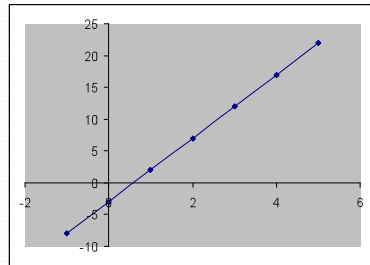 `17 13 9 5 1`

- Let's create the loop table together.
  - Each time `count` goes up 1, the number printed should ...
  - But this multiple is off by a margin of ...

count	number to print	-4 * count	-4 * count + 21
1	17	-4	17
2	13	-8	13
3	9	-12	9
4	5	-16	5
5	1	-20	1

24

Another view: Slope-intercept

- The next three slides present the mathematical basis for the loop tables. Feel free to skip it.

count (x)	number to print (y)
1	2
2	7
3	12
4	17
5	22

Another view: Slope-intercept

- *Caution*: This is algebra, not assignment!
- Recall: slope-intercept form (`y = mx + b`)
- Slope is defined as "rise over run" (i.e. rise / run). Since the "run" is always `1` (we increment along `x` by 1), we just need to look at the "rise". The rise is the difference between the `y` values. Thus, the slope (`m`) is the difference between `y` values; in this case, it is `+5`.
- To compute the y-intercept (`b`), plug in the value of `y` at `x = 1` and solve for `b`. In this case, `y = 2`.

```
y = m * x + b
2 = 5 * 1 + b
```
Then `b = -3`
- So the equation is

```
y = m * x + b
y = 5 * x - 3
y = 5 * count - 3
```

count (x)	number to print (y)
1	2
2	7
3	12
4	17
5	22

Another view: Slope-intercept

- Algebraically, if we always take the value of y at $x = 1$, then we can solve for b as follows:

```
y = m * x + b
y₁ = m * 1 + b
y₁ = m + b
b = y₁ - m
```

- In other words, to get the y-intercept, just subtract the slope from the first y value ($b = 2 - 5 = -3$)
  - This gets us the equation

```
y = m * x + b
y = 5 * x - 3
y = 5 * count - 3
```

   (which is exactly the equation from the previous slides)

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
}
```

Nested `for` loop solution

- Answer:
```
for (int line = 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:
  ```java
  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();
  }
  ```

31