Repetition with for loops

- So far, repeating a statement is redundant:

  ```java
  System.out.println("Mike says:");
  System.out.println("Do Practice-It problems!");
  System.out.println("Do Practice-It problems!");
  System.out.println("Do Practice-It problems!");
  System.out.println("Do Practice-It problems!");
  System.out.println("It makes a HUGE difference.");
  ```

- Java's for loop statement performs a task many times.

  ```java
  System.out.println("Mike says:");
  for (int i = 1; i <= 5; i++) { // repeat 5 times
    System.out.println("Do Practice-It problems!");
  }
  System.out.println("It makes a HUGE difference.");
  ```

for loop syntax

```java
for (<initialization>; <test>; <update>) {
  <statement>
  ...
  <statement>
}
```

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

Initialization

```java
for (int i = 1; i <= 5; i++) {
  System.out.println("Do Practice-It!");
}
```

- Tells Java compiler what variable to use in the loop
  - Performed once as the loop begins
  - The variable is called a loop counter or loop control variable
    - can use any name, not just i
    - can start at any value, not just 1
for (int i = 1; i <= 5; i++) {
    System.out.println("Do Practice-It!");
}

- 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
    - == equality  != not equals

for (int i = 1; i <= 5; i++) {
    System.out.println("Do Practice-It!");
}

- If the test is true, the statements in the body of the loop execute in sequential order one time
- The body of the loop is between the curly braces
- If the body is one statement the curly braces are not required, but by convention we still add them
- After the body of the loop completes the update statement is executed.

for (int i = 1; i <= 5; i++) {
    System.out.println("Do Practice-It!");
}

- Perform update step
  - Generally adding one to loop control variable
  - Could be other operations such as subtracting one, multiplying

for (int i = 1; i <= 5; i++) {
    System.out.println("Do Practice-It!");
}

- Shorthand
  - `<variable>`++;  
  - `<variable>`--;  

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

double gpa = 2.5;
gpa--;  // gpa = gpa - 1;
// gpa now stores 1.5
Aside: Modify-and-assign operators
shortcuts to modify a variable’s value

<table>
<thead>
<tr>
<th>Shorthand</th>
<th>Equivalent longer version</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;variable&gt; += &lt;exp&gt;;</td>
<td>&lt;variable&gt; = &lt;variable&gt; + (&lt;exp&gt;);</td>
</tr>
<tr>
<td>&lt;variable&gt; -= &lt;exp&gt;;</td>
<td>&lt;variable&gt; = &lt;variable&gt; - (&lt;exp&gt;);</td>
</tr>
<tr>
<td>&lt;variable&gt; *= &lt;exp&gt;;</td>
<td>&lt;variable&gt; = &lt;variable&gt; * (&lt;exp&gt;);</td>
</tr>
<tr>
<td>&lt;variable&gt; /= &lt;exp&gt;;</td>
<td>&lt;variable&gt; = &lt;variable&gt; / (&lt;exp&gt;);</td>
</tr>
<tr>
<td>&lt;variable&gt; %= &lt;exp&gt;;</td>
<td>&lt;variable&gt; = &lt;variable&gt; % (&lt;exp&gt;);</td>
</tr>
</tbody>
</table>

x += 3;    // x = x + 3;
gpa -= 0.5; // gpa = gpa - 0.5;
number *= 2 + 1; // number = number * (2 + 1);

Clicker 1

What is output by the following code?

```java
int x = 2;
int y = 5;
x *= 3 + y + x;
System.out.println(x + " " + y);
```

A. 20 5
B. 2 5
C. 13 5
D. 20 10
E. Something other than A - D

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.”

Repetition over a range

```java
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!

```java
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**

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

Output:
1 squared = 1
2 squared = 4
3 squared = 9
4 squared = 16
Whoo!

**Simple Loop Example**

- Write a program to calculate and print out the values of N! from 1 to 50 using a for loop
  - 0! = 1
  - 1! = 1 * 0! = 1 * 1 = 1
  - 2! = 2 * 1! = 2 * 1 * 1 = 2
  - 3! = 3 * 2! = 3 * 2 * 1 * 1 = 6
  - 4! = 4 * 3! = 4 * 3 * 2 * 1 * 1 = 24

**Multi-line loop body**

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

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

**Expressions for counter**

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

```java
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 `<` (or logic error)
  ```java
  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.

Clicker 2

• How many asterisks are output by the following code?

```java
for(int i = -2; i <= 13; i++) {
    System.out.print("*");
    System.out.print("**");
}
```

A. 0  B. 15  C. 45  D. 48  E. 68

Practice Problem

• Newton's method for approximating square roots adapted from the Dr. Math website
  The goal is to find the square root of a number. Let's call it num
  1. Choose a rough approximation of the square root of num, call it approx.
  2. Divide num by approx and then average the quotient with approx,
     in other words we want to evaluate the expression ((num/approx) + approx) / 2
  3. How close are we? In programming we would store the result of the expression back into the variable approx.
  4. How do you know if you have the right answer?
Sample of Newton's Method

<table>
<thead>
<tr>
<th>num</th>
<th>approx</th>
<th>(num/approx)+approx)/2</th>
<th>approx*approx</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>6</td>
<td>(12 / 6 + 6) / 2 = 4</td>
<td>16</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>(12 / 4 + 4) / 2 = 3.5</td>
<td>12.25</td>
</tr>
<tr>
<td>12</td>
<td>3.5</td>
<td>(12 / 3.5 + 3.5) / 2 = 3.4642857...</td>
<td>12.0012..</td>
</tr>
<tr>
<td>12</td>
<td>3.4642857</td>
<td>= 3.46410162...</td>
<td>12.00000003</td>
</tr>
<tr>
<td>12</td>
<td>3.46410162</td>
<td>= 3.46410161...</td>
<td>11.99999999</td>
</tr>
</tbody>
</table>

3.4641016151377544 after 5 steps

3.4641016151377545870548926830117 (from calculator)

Nested loops

- **nested loop**: A loop placed inside another loop.

```java
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?

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

Output:

```
*
**
***
****
*****
```
Nested for loop exercise

What is the output of the following nested for loops?

```java
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

Clicker 3

What is output by the following code?

```java
int total = 0;
for(int i = 1; i <= 4; i++) {
    for(int j = 1; j <= i; j++) {
        total += i;
    }
}
System.out.println(total);
```

A. 4  B. 10  C. 16  D. 24  E. 30

Common errors

Both of the following sets of code produce infinite loops:

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

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

Complex output

Write a nested for loop to produce the following output.

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

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.

```java
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

```java
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

```java
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.

```
<table>
<thead>
<tr>
<th>count</th>
<th>number to print</th>
<th>5 * count</th>
<th>5 * count - 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>20</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>22</td>
<td>25</td>
<td>22</td>
</tr>
</tbody>
</table>
```

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 ...

```
<table>
<thead>
<tr>
<th>count</th>
<th>number to print</th>
<th>-4 * count</th>
<th>-4 * count + 21</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17</td>
<td>-4</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>-8</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>-12</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>-16</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>-20</td>
<td>1</td>
</tr>
</tbody>
</table>
```
Another view: Slope-intercept

- The next three slides present the mathematical basis for the loop tables.

<table>
<thead>
<tr>
<th>count (x)</th>
<th>number to print (y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>22</td>
</tr>
</tbody>
</table>

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, \(y = 2\).

\[
\begin{align*}
  y &= m \cdot x + b \\
  2 &= 5 \cdot 1 + b \\
  \text{Then } b &= -3
\end{align*}
\]
- So the equation is

\[
\begin{align*}
  y &= m \cdot x + b \\
  y &= 5 \cdot x - 3 \\
  y &= 5 \cdot \text{count} - 3
\end{align*}
\]

Another view: Slope-intercept

- Algebraically, if we always take the value of \(y\) at \(x = 1\), then we can solve for \(b\) as follows:
  \[
  \begin{align*}
    y &= m \cdot x + b \\
    y_1 &= m \cdot 1 + b \\
    y_1 &= m + b \\
    b &= y_1 - m
  \end{align*}
  \]
- 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
    \[
    \begin{align*}
      y &= m \cdot x + b \\
      y &= 5 \cdot x - 3 \\
      y &= 5 \cdot \text{count} - 3
    \end{align*}
    \]
    (which is exactly the equation from the previous slides)

Nested for loop exercise

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

<table>
<thead>
<tr>
<th>line</th>
<th># of dots</th>
<th>-1 * line</th>
<th>-1 * line + 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>-1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>-2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>-3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>-4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>-5</td>
<td>0</td>
</tr>
</tbody>
</table>

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

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

- **Answer:**
  ```java
class NestedForLoopSolution {
    public static void main(String[] args) {
      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?**
  ```java
class NestedForLoopExercise {
  public static void main(String[] args) {
    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();
    }
  }
}
```

- **Answer:**
  
  ```
  ....1
  ...22
  ..333
  .4444
  55555
  ```

- **Modify the previous code to produce this output:**
  ```java
class ModifiedNestedForLoopExercise {
  public static void main(String[] args) {
    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();
    }
  }
}
```

- **Output:**
  
  ```
  ....1
  ...2
  ..3..
  .4...
  5....
  ```