Building Java Programs

Chapter 2
Lecture 3: Variables, For Loops, Nested Loops

reading: 2.2 – 2.3

(Slides adapted from Stuart Reges, Hélène Martin, and Marty Stepp)
Software is sometimes written by people that understand software.

As a programmer all the bad code I've seen would state otherwise....

Not saying I'm perfect, but there is lot of terrible terrible code out there.

[DanielMallory] E.G.

Minecraft Symantec

It's hard to explain "bad code" to someone who doesn't understand code.

but have you LOOKED at Minecraft's source code? Christ, my cat on catnip could write a better program in Java.

[BusinessCasualty] He'd get too distracted by all the strings
Variables

reading: 2.2
Receipt example

What's bad about the following code?

```java
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        System.out.print("Subtotal: ");
        System.out.println(38 + 40 + 30);
        System.out.print("Tax: ");
        System.out.println((38 + 40 + 30) * .08);
        System.out.print("Tip: ");
        System.out.println((38 + 40 + 30) * .15);
        System.out.print("Total: ");
        System.out.println(38 + 40 + 30 +
                         (38 + 40 + 30) * .08 +
                         (38 + 40 + 30) * .15);
    }
}
```

- The subtotal expression `(38 + 40 + 30)` is repeated
- So many `println` statements
Variables

• **variable**: A piece of the computer's memory that is given a name and type, and can store a value.
  • Like preset stations on a car stereo, or cell phone speed dial:

• Steps for using a variable:
  • *Declare it* - state its name and type
  • *Initialize it* - store a value into it
  • *Use it* - print it or use it as part of an expression
Declaration

- **variable declaration**: Sets aside memory for storing a value.
  - Variables must be declared before they can be used.

**Syntax:**

```plaintext
type name;
```

- The name is an *identifier*.

- `int zipcode;`
- `double myGPA;`
Assignment

- **assignment**: Stores a value into a variable.
  - The value can be an expression; the variable stores its result.

- Syntax:
  
  \[
  \text{name} = \text{expression};
  \]

- `int zipcode;`  
  `zipcode = 90210;`  

- `double myGPA;`  
  `myGPA = 1.0 + 2.25;`
Using variables

• Once given a value, a variable can be used in expressions:

```java
int x;
x = 3;
System.out.println("x is " + x);  // x is 3
System.out.println(5 * x - 1);   // 5 * 3 - 1
```

• You can assign a value more than once:

```java
int x;
x = 3;
System.out.println(x + " here");  // 3 here

x = 4 + 7;
System.out.println("now x is " + x); // now x is 11
```
Declaration_INITIALIZATION

• A variable can be declared/initialized in one statement.

• Syntax:
  
  \textbf{type name} = \textbf{value};

• double myGPA = 3.95;

• int x = (11 \% 3) + 12;

\begin{tabular}{|c|c|}
  \hline
  myGPA & 3.95 \\
  \hline
\end{tabular}

\begin{tabular}{|c|c|}
  \hline
  x & 14 \\
  \hline
\end{tabular}
Assignment and algebra

• Assignment uses $=$, but it is not an algebraic equation.
  • $=$ means, "store the value at right in variable at left"
  • The right side expression is evaluated first, and then its result is stored in the variable at left.

• What happens here?

```java
int x = 3;
x = x + 2; // ???
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>5</td>
</tr>
</tbody>
</table>
Increment and decrement

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

**Shorthand**

- `variable++;`
- `variable--;`

**Equivalent longer version**

- `variable = variable + 1;`
- `variable = variable - 1;`

```cpp
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

<table>
<thead>
<tr>
<th>Shorthand</th>
<th>Equivalent longer version</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable += value;</td>
<td>variable = variable + value;</td>
</tr>
<tr>
<td>variable -= value;</td>
<td>variable = variable - value;</td>
</tr>
<tr>
<td>variable *= value;</td>
<td>variable = variable * value;</td>
</tr>
<tr>
<td>variable /= value;</td>
<td>variable = variable / value;</td>
</tr>
<tr>
<td>variable %= value;</td>
<td>variable = variable % value;</td>
</tr>
</tbody>
</table>

x += 3;  // x = x + 3;
gpa -= 0.5;  // gpa = gpa - 0.5;
number *= 2;  // number = number * 2;
Assignment and types

- A variable can only store a value of its own type.
  - `int x = 2.5; // ERROR: incompatible types`

- An `int` value can be stored in a `double` variable.
  - The value is converted into the equivalent real number.
  - `double myGPA = 4;`
  - `double avg = 11 / 2;`
    - Why does `avg` store `5.0` and not `5.5`?
Compiler errors

• A variable can't be used until it is assigned a value.
  • int x;
    System.out.println(x); // ERROR: x has no value

• You may not declare the same variable twice.
  • int x;
    int x; // ERROR: x already exists
  • int x = 3;
    int x = 5;  // ERROR: x already exists

• How can this code be fixed?
Printing a variable's value

- Use + to print a string and a variable's value on one line.

```java
double grade = (95.1 + 71.9 + 82.6) / 3.0;
System.out.println("Your grade was " + grade);

int students = 11 + 17 + 4 + 19 + 14;
System.out.println("There are " + students + " students in the course.");
```

- Output:

  Your grade was 83.2
  There are 65 students in the course.
Receipt question

Improve the receipt program using variables.

```java
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        System.out.print("Subtotal: ");
        System.out.println(38 + 40 + 30);
        System.out.print("Tax: ");
        System.out.println((38 + 40 + 30) * .08);
        System.out.print("Tip: ");
        System.out.println((38 + 40 + 30) * .15);
        System.out.print("Total: ");
        System.out.println(38 + 40 + 30 + (38 + 40 + 30) * .15 + (38 + 40 + 30) * .08);
    }
}
```
public class Receipt {
    public static void main(String[] args) {
        // Calculate total owed, assuming 8% tax / 15% tip
        int subtotal = 38 + 40 + 30;
        double tax = subtotal * .08;
        double tip = subtotal * .15;
        double total = subtotal + tax + tip;

        System.out.println("Subtotal: " + subtotal);
        System.out.println("Tax: " + tax);
        System.out.println("Tip: " + tip);
        System.out.println("Total: " + total);
    }
}
Building Java Programs

Chapter 2
Lecture 2-2: The for Loop

reading: 2.3
Repetition with  \textit{for} loops

- So far, repeating an action results in redundant code:

  ```java
  makeBatter();
bakeCookies();
bakeCookies();
bakeCookies();
bakeCookies();
bakeCookies();
frostCookies();
  ```

- Java's \textit{for loop} statement performs a task many times.

  ```java
  mixBatter();
  for (int i = 1; i <= 5; i++) {
    // repeat 5 times
    bakeCookies();
  }
frostCookies();
  ```
for loop syntax

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

- Perform **initialization** once.
- Repeat the following:
  - Check if the **test** is true. If not, stop.
  - Execute the **statements**.
  - Perform the **update**.
Control structures

- **Control structure**: a programming construct that affects the flow of a program's execution

- Controlled code may include one or more statements

- The for loop is an example of a looping control structure
Initialization

```java
for (int i = 1; i <= 6; i++) {
    System.out.println("I am so smart");
}
```

- Tells Java what variable to use in the loop
  - The variable is called a *loop counter*
    - can use any name, not just `i`
    - can start at any value, not just `1`
    - only valid in the loop
- Performed once as the loop begins
Test

```java
for (int i = 1; i <= 6; 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
  - `==` exactly equal to
  - `!=` not equal to
Update

```
for (int i = 1; i <= 6; i++) {
    System.out.println("I am so smart");
}
```

- Updates the loop counter to a new value
- If the updates do not eventually make the loop test fail, the loop will never end
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

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!

1 Perform initialization once
2 Is the test true?
3 perform the update
4 execute the controlled statement(s)
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("/ \\
}
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);
}
```

- **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

```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 <

```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.
Nested loops

reading: 2.3
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("*");
    }
    System.out.println();
}
```

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
```
Common errors

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

```java
for (int i = 1; i <= 5; i++) {
    for (int j = 1; i <= 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 lines

• What nested for loops produce the following output?

inner loop (repeated characters on each line)

\[\ldots1\]
\[\ldots2\]
\[\ldots3\]
\[.4\]
\[.5\]

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.

```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

```plaintext
....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.

<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 \( \text{count} \) 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 \times \text{count})</th>
<th>(-4 \times \text{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. Feel free to skip it.

<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, \(y = 2\).

\[
y = m \times x + b
2 = 5 \times 1 + b
\]

Then \(b = -3\)

- So the equation is

\[
y = m \times x + b
y = 5 \times x - 3
y = 5 \times \text{count} - 3
\]

<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

- Algebraically, if we always take the value of $y$ at $x = 1$, then we can solve for $b$ as follows:
  
  
  $y = m \times x + b$
  
  $y_1 = m \times 1 + b$
  
  $y_1 = m + b$
  
  $b = y_1 - m$

- 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
  
  $y = m \times x + b$
  
  $y = 5 \times x - 3$
  
  $y = 5 \times \text{count} - 3$

  (which is exactly the equation from the previous slides)
Nested for loop exercise

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

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