Definite Loops

Computer Science S-111
Harvard University
David G. Sullivan, Ph.D.

Using a Variable for Counting

• Let's say that we're using a variable \( i \) to count the number of times that something has been done:

\[
\text{int } i = 0;
\]

\[
\begin{array}{c}
i \quad 0 \\
0 + 1 \\
1 \\
\end{array}
\]

• To increase the count, we can do this:

\[
i = i + 1;
\]

\[
\begin{array}{c}
i \quad 0 \\
0 + 1 \\
1 \\
\end{array}
\]

• To increase the count again, we repeat the same assignment:

\[
i = i + 1;
\]

\[
\begin{array}{c}
i \quad 2 \\
2 + 1 \\
3 \\
\end{array}
\]
Increment and Decrement Operators

- Instead of writing
  \[ i = i + 1; \]
  we can use a shortcut and just write
  \[ i++; \]
- \[ ++ \] is known as the \textit{increment operator}.
  - \textit{increment} = increase by 1
- Java also provides a \textit{decrement operator} (\[ -- \]).
  - \textit{decrement} = decrease by 1
  - example:
    \[ i--; \]

Review: Flow of Control

- Flow of control = the order in which instructions are executed
- By default, instructions are executed in sequential order.
  
  \begin{align*}
  \text{instructions} & \quad \text{flowchart} \\
  \text{int sum} = 0; & \quad \text{int sum} = 0; \\
  \text{int num1} = 5; & \quad \text{int num1} = 5; \\
  \text{int num2} = 10; & \quad \text{int num2} = 10; \\
  \text{sum} = \text{num1} + \text{num2}; & \quad \text{sum} = \text{num1} + \text{num2};
  \end{align*}

- When we make a method call, the flow of control "jumps" to the method, and it "jumps" back when the method completes.
Altering the Flow of Control: Repetition

• To solve many types of problems, we need to be able to modify the order in which instructions are executed.

• One reason for doing this is to allow for repetition.

• We saw this in Scratch:

Example of the Need for Repetition

• Here’s a method for writing a large block letter L:

```java
public static void writeL() {
    System.out.println("|");
    System.out.println("|");
    System.out.println("|");
    System.out.println("|");
    System.out.println("|");
    System.out.println("|");
    System.out.println("+----------");
}
```

• Rather than duplicating the statement
  System.out.println("|");
seven times, we’d like to have this statement appear just once and execute it seven times.
for Loops

• To repeat one or more statements multiple times, we can use a construct known as a for loop.

• Here’s a revised version of our writeL method that uses one:

```java
public static void writeL() {
    for (int i = 0; i < 7; i++) {
        System.out.println("|");
    }
    System.out.println("+----------");
}
```

for Loops

• Syntax:

```
for (<initialization>; <continuation test>; <update>) {
    <one or more statements>
}
```

• In our example: initialization continuation test

```
for (int i = 0; i < 7; i++) {
    System.out.println("|");
}
```

• The statements inside the loop are known as the body of the loop.

• In our example, we use the variable i to count the number of times that the body has been executed.
Executing a for Loop

for (<initialization>; <continuation test>; <update>) {
   <body of the loop>
}

- perform the initialization
- is the test true?
- yes: execute body
- no: continue
- perform the update
- execute statement after the loop

Notes:
- the initialization is only performed once
- the body is only executed if the test is true
- we repeatedly do: test body update until the test is false

Executing Our for Loop

for (int i = 0; i < 7; i++) {
   System.out.println("|");
}

<table>
<thead>
<tr>
<th>i</th>
<th>i &lt; 7</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>true</td>
<td>print 1st &quot;</td>
</tr>
<tr>
<td>1</td>
<td>true</td>
<td>print 2nd &quot;</td>
</tr>
<tr>
<td>2</td>
<td>true</td>
<td>print 3rd &quot;</td>
</tr>
<tr>
<td>3</td>
<td>true</td>
<td>print 4th &quot;</td>
</tr>
<tr>
<td>4</td>
<td>true</td>
<td>print 5th &quot;</td>
</tr>
<tr>
<td>5</td>
<td>true</td>
<td>print 6th &quot;</td>
</tr>
<tr>
<td>6</td>
<td>true</td>
<td>print 7th &quot;</td>
</tr>
<tr>
<td>7</td>
<td>false</td>
<td>execute stmt. after the loop</td>
</tr>
</tbody>
</table>
Definite Loops

• For now, we'll limit ourselves to definite loops – which repeat actions a fixed number of times.

• To repeat the body of a loop \(<N>\) times, we typically take one of the following approaches:

  ```java
  for (int i = 0; i < <N>; i++) {
    <body of the loop>
  }
  OR
  for (int i = 1; i <= <N>; i++) {
    <body of the loop>
  }
  ```

• Each time that the body of a loop is executed is known as an iteration of the loop.
  • the loops shown above perform <N> iterations

Other Examples of Definite Loops

• What does this loop do?

  ```java
  for (int i = 0; i < 3; i++) {
    System.out.println("Hip! Hip!");
    System.out.println("Hooray!");
  }
  ```

• What does this loop do?

  ```java
  for (int i = 0; i < 10; i++) {
    System.out.println(i);
  }
  ```
Using Different Initializations, Tests, and Updates

• The second loop from the previous page would be clearer if we expressed it like this:

   ```java
   for (int i = 0; i <= 9; i++) {
       System.out.println(i);
   }
   ```

• Different problems may require different initializations, continuation tests, and updates.

• What does this code fragment do?

   ```java
   for (int i = 2; i <= 10; i = i + 2) {
       System.out.println(i * 10);
   }
   ```

Tracing a for Loop

• Let's trace through the final code fragment from the last slide:

   ```java
   for (int i = 2; i <= 10; i = i + 2) {
       System.out.println(i * 10);
   }
   ```

<table>
<thead>
<tr>
<th>i</th>
<th>i &lt;= 10</th>
<th>value printed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Common Mistake

- You should not put a semi-colon after the for-loop header:
  ```java
  for (int i = 0; i < 7; i++) {
      System.out.println("|");
  }
  ```

- The semi-colon ends the for statement.
  - thus, it doesn't repeat anything!

- The println is independent of the for statement, and only executes once.

Practice

- Fill in the blanks below to print the integers from 1 to 10:
  ```java
  for (_________; ___________; ___________) {
      System.out.println(i);
  }
  ```

- Fill in the blanks below to print the integers from 10 to 20:
  ```java
  for (_________; ___________; ___________) {
      System.out.println(i);
  }
  ```

- Fill in the blanks below to print the integers from 10 down to 1:
  ```java
  for (_________; ___________; ___________) {
      System.out.println(i);
  }
  ```
Other Java Shortcuts

- Recall this code fragment:
  ```java
  for (int i = 2; i <= 10; i = i + 2) {
    System.out.println(i * 10);
  }
  ```

- Instead of writing
  ```java
  i = i + 2;
  ```
we can use a shortcut and just write
  ```java
  i += 2;
  ```

- In general
  ```java
  <variable> += <expression>;
  ```
is equivalent to
  ```java
  <variable> = <variable> + (<expression>);
  ```

Java Shortcuts

- Java offers other shortcut operators as well.

- Here's a summary of all of them:

<table>
<thead>
<tr>
<th>shortcut</th>
<th>equivalent to</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;var&gt;++;</td>
<td>&lt;var&gt; = &lt;var&gt; + 1;</td>
</tr>
<tr>
<td>&lt;var&gt;--;</td>
<td>&lt;var&gt; = &lt;var&gt; - 1;</td>
</tr>
<tr>
<td>&lt;var&gt; += &lt;expr&gt;;</td>
<td>&lt;var&gt; = &lt;var&gt; + (&lt;expr&gt;);</td>
</tr>
<tr>
<td>&lt;var&gt; -= &lt;expr&gt;;</td>
<td>&lt;var&gt; = &lt;var&gt; - (&lt;expr&gt;);</td>
</tr>
<tr>
<td>&lt;var&gt; *= &lt;expr&gt;;</td>
<td>&lt;var&gt; = &lt;var&gt; * (&lt;expr&gt;);</td>
</tr>
<tr>
<td>&lt;var&gt; /= &lt;expr&gt;;</td>
<td>&lt;var&gt; = &lt;var&gt; / (&lt;expr&gt;);</td>
</tr>
<tr>
<td>&lt;var&gt; %= &lt;expr&gt;;</td>
<td>&lt;var&gt; = &lt;var&gt; % (&lt;expr&gt;);</td>
</tr>
</tbody>
</table>

- Important: the = must come after the mathematical operator.
  ```java
  += is correct
  += is not!
  ```
More Practice

• Fill in the blanks below to print the even integers in reverse order from 20 down to 6:

   for (___________; ___________; ___________) {
       System.out.println(i);
   }

Find the Error

• Let's say that we want to print the numbers from 1 to n.

• Where is the error in the following code?

   for (int i = 1; i < n; i++) {
       System.out.println(i);
   }

• This is an example of an off-by-one error. Beware of these when writing your loop conditions!
Example Problem: Printing a Pattern, version 1

- Ask the user for a positive integer (call it n), and print a pattern containing n asterisks.
  - example:
    Enter a positive integer: 3
    ***

- Let's use a for loop to do this:
  // code to read n goes here...
  for ( ) {
    System.out.print("*");
  }
  System.out.println();

Example Problem: Printing a Pattern, version 2

- Print a pattern containing n lines of n asterisks.
  - example:
    Enter a positive integer: 3
    ***
    ***
    ***

- One way to do this is to use a nested loop – one loop inside another:
  // code to read in n goes here...
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      System.out.print("*");
    }
    System.out.println();
  }

- This makes it easier to create a similar box of a different size.
Nested Loops

• When you have a nested loop, the inner loop is executed to completion for every iteration of the outer loop.

• Recall our Scratch drawing program:

```
pen down
repeat 5
  repeat 6
    move 80 steps
turn 60 degrees
```

• How many times is the move statement executed?

Nested Loops (cont.)

• How many times is the println statement executed below?

```java
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 7; j++) {
        System.out.println(i + " " + j);
    }
}
```

• How many times is the println statement executed below?

```java
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < i; j++) {
        System.out.println(i + " " + j);
    }
}
```
Tracing a Nested for Loop

```java
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < i; j++) {
        System.out.println(i + " " + j);
    }
}
```

- **i** : i < 5
- **j** : j < i

value printed

Recall: Variable Scope

- The scope of a variable is the portion of a program in which the variable can be used.
- By default, the scope of a variable in Java:
  - begins at the point at which it is declared
  - ends at the end of the innermost block that encloses the declaration

```java
public class MyProgram2 {
    public static void main(String[] args) {
        System.out.println("Welcome!");
        System.out.println("Let's do some math!");
        int j = 10;
        System.out.println(j / 5);  // scope of j
    }
}
```
Special Case: for Loops and Variable Scope

- When a variable is declared in the initialization clause of a for loop, its scope is limited to the loop.

- Example:

```java
public static void myMethod() {
    for (int i = 0; i < 5; i++) {
        int j = i * 3;
        System.out.println(j);
    }
    // the following line won't compile
    System.out.print(i);
    System.out.println(" values were printed.");
}
```

Special Case: for Loops and Variable Scope (cont.)

- To allow `i` to be used outside the loop, we need to declare it outside the loop:

- Example:

```java
public static void myMethod() {
    int i;
    for (i = 0; i < 5; i++) {
        int j = i * 3;
        System.out.println(j);
    }
    // now this will compile
    System.out.print(i);
    System.out.println(" values were printed.");
}
```
Special Case: for Loops and Variable Scope (cont.)

- Limiting the scope of a loop variable allows us to use the standard loop templates multiple times in the same method.

- Example:

```java
public static void myMethod() {
    for (int i = 0; i < 5; i++) {
        int j = i * 3;
        System.out.println(j);
    }
    for (int i = 0; i < 7; i++) {
        System.out.println("Go Crimson!");
    }
}
```

Review: Simple Repetition Loops

- Recall our two templates for performing `<N>` repetitions:

  ```java
  for (int i = 0; i < `<N>`; i++) {
      // code to be repeated
  }
  for (int i = 1; i <= `<N>`; i++) {
      // code to be repeated
  }
  ```

- How may repetitions will each of the following perform?

  ```java
  for (int i = 1; i <= 15; i++) {
      System.out.println("Hello");
      System.out.println("How are you?");
  }
  for (int i = 0; i < 2*j; i++) {
      ...
  }
  ```
More Practice: Tracing a Nested for Loop

```java
for (int i = 1; i <= 3; i++) {
    for (int j = 0; j < 2*i + 1; j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

output

Case Study: Drawing a Complex Figure

- Here's the figure:

```
()  
((()))  
(((()))  

======
|::::::|
|::::|
|::|
|::|
|::|
|::|
+==+
```

- To begin with, we'll focus on creating this exact figure.

- Then we'll modify our code so that the size of the figure can easily be changed.
  - we'll use for loops to allow for this
Problem Decomposition

• We begin by breaking the problem into subproblems, looking for groups of lines that follow the same pattern:

\[
\begin{align*}
() & \quad \text{flame} \\
(()) & \quad \text{rim of torch} \\
(()()) & \quad \text{top of torch} \\
((())) & \quad \text{handle of torch} \\
(((()))) & \quad \text{bottom of torch}
\end{align*}
\]

This gives us the following initial pseudocode:

\[
\begin{align*}
&\text{draw the flame} \\
&\text{draw the rim of the torch} \\
&\text{draw the top of the torch} \\
&\text{draw the handle of the torch} \\
&\text{draw the bottom of the torch}
\end{align*}
\]

• This is a high-level description of what needs to be done.

• We'll gradually expand the pseudocode into more and more detailed instructions – until we're able to implement them in Java.
### Drawing the Flame

- Let's begin by refining our specification for drawing the flame.
- Here's our initial pseudocode for this task:
  ```
  for (each of 4 lines) {
    print some spaces (possibly 0)
    print some left parentheses
    print some right parentheses
    go to a new line
  }
  ```
- We need formulas for how many spaces and parens should be printed on a given line.

### Finding the Formulas

- To begin with, we:
  - number the lines in the flame
  - form a table of the number of spaces and parentheses on each line:

<table>
<thead>
<tr>
<th>line</th>
<th>spaces</th>
<th>parens (each type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

- Then we find the formulas.
  - assume the formulas are linear functions of the line number:
    \[ c_1 \times \text{line} + c_2 \]
    where \( c_1 \) and \( c_2 \) are constants
- parens = ?
- spaces = ?
Refining the Pseudocode

- Given these formulas, we can refine our pseudocode:

```plaintext
for (each of 4 lines) {
    print some spaces (possibly 0)
    print some left parentheses
    print some right parentheses
    go to a new line
}
```

```plaintext
for (line going from 1 to 4) {
    print 4 - line spaces
    print line left parentheses
    print line right parentheses
    go to a new line
}
```

Implementing the Pseudocode in Java

- We use nested for loops:

```java
for (int line = 1; line <= 4; line++) {
    for (int i = 0; i < 4 - line; i++) {
        System.out.print(" ");
    }
    for (int i = 0; i < line; i++) {
        System.out.print("(");
    }
    for (int i = 0; i < line; i++) {
        System.out.print(")");
    }
    System.out.println();
}
```
A Method for Drawing the Flame

• We put the code in its own static method, and add some explanatory comments:

```java
public static void drawFlame() {
    for (int line = 1; line <= 4; line++) {
        // spaces to the left of the current line
        for (int i = 0; i < 4 - line; i++) {
            System.out.print(" ");
        }
        // left and right parens on the current line
        for (int i = 0; i < line; i++) {
            System.out.print("(");
        }
        for (int i = 0; i < line; i++) {
            System.out.print(")");
        }
        System.out.println();
    }
}
```

Drawing the Top of the Torch

• What's the initial pseudocode for this task?

```java
for (each of 2 lines) {
    
}
```

• Here's a table for the number of spaces and number of colons:

<table>
<thead>
<tr>
<th>line</th>
<th>spaces</th>
<th>colons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

• spaces = ?
• colons decreases by 2 as line increases by 1
  ➔ colons = -2*line + c2 for some number c2
• try different values, and eventually get: colons = ?
Refining the Pseudocode

- Once again, we use the formulas to refine our pseudocode:

```plaintext
for (each of 2 lines) {
    print some spaces (possibly 0)
    print a single vertical bar
    print some colons
    print a single vertical bar
    go to a new line
}
```

```plaintext
for (line going from 1 to 2) {
    print line - 1 spaces
    print a single vertical bar
    print –2*line + 8 colons
    print a single vertical bar
    go to a new line
}
```

A Method for Drawing the Top of the Torch

```java
public static void drawTop() {
    for (int line = 1; line <= 2; line++) {
        // spaces to the left of the current line
        for (int i = 0; i < line - 1; i++) {
            System.out.print(" ");
        }
        // bars and colons on the current line
        System.out.print("|");
        for (int i = 0; i < -2*line + 8; i++) {
            System.out.print(":");
        }
        System.out.print("|");
        System.out.println();
    }
}
```
Drawing the Rim

- This always has only one line, so we don't need nested loops.

- However, we still need a single loop, because we want to be able to scale the size of the figure.

- What should the code look like?

  ```java
  for ( ; ; ) {
  }
  ```

- This code also goes in its own method, called `drawRim()`

Incremental Development

- We take similar steps to implement methods for the remaining subtasks.

- After completing a given method, we test and debug it.

- The `main` method just calls the methods for the subtasks:

  ```java
  public static void main(String[] args) {
    drawFlame();
    drawRim();
    drawTop();
    drawHandle();
    drawBottom();
  }
  ```

- See the example program `DrawTorch.java`
Using Class Constants

• To make the torch larger or smaller, we'd need to make many changes.
  • the size of the figure is hard-coded into most methods

• To make the program more flexible, we can store info. about the figure's dimensions in one or more class constants.
  • like variables, but their values are fixed
  • can be used throughout the program

Using Class Constants (cont.)

• We only need one constant for the torch.
  • for the default size, it equals 2
  • its connection to some of the dimensions is shown at right

• We declare it at the very start of the class:
  ```java
  public class DrawTorch2 {
      public static final int SCALE_FACTOR = 2;
      ...
  }
  ```

• General syntax:
  ```java
  public static final <type> <name> = <expression>;
  ```

  • conventions:
    • capitalize all letters in the name
    • put an underscore ('_') between multiple words
Scaling the Figure

• Here are some other versions of the figure:

```
()  ()
()()  ()()
()()()  ====
()()()()  |::|
()()()()()  ||
()()()()()()  ++
```

SCALE_FACTOR = 3

```
()  ()
()()  ()()
()()()  ====
()()()()  |::|
()()()()()  ||
()()()()()()  ++
```

SCALE_FACTOR = 1

```
()  ()
()()  ()()
()()()  ====
()()()()  |::|
()()()()()  ||
()()()()()()  ++
```

SCALE_FACTOR = 3

Revised Method for Drawing the Flame

• We replace the two 4s with 2*SCALE_FACTOR:

```java
public static void drawFlame() {
    for (int line = 1; line <= 2*SCALE_FACTOR; line++) {
        // spaces to the left of the flame
        for (int i = 0; i < 2*SCALE_FACTOR – line; i++) {
            System.out.print(" ");
        }
        // the flame itself, both left and right halves
        for (int i = 0; i < line; i++) {
            System.out.print("(");
        }
        for (int i = 0; i < line; i++) {
            System.out.print(")");
        }
        System.out.println();
    }
}
```
Making the Rim Scaleable

• How does the width of the rim depend on SCALE_FACTOR?

```
()
(()
(())
((()))
(((())))
((((()))))

(()
(())
(()
(()
(())
(()
(())
((()))
(((())))
((((()))))

=====

```

• Use a table!

<table>
<thead>
<tr>
<th>SCALE_FACTOR</th>
<th>width of rim</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
</tr>
</tbody>
</table>

width of rim = ?

Revised Method for Drawing the Rim

• Original version (for the default size):

```java
public static void drawRim() {
    for (int i = 0; i < 8; i++) {
        System.out.print("=");
    }
    System.out.println();
}
```

• Scaleable version:

```java
public static void drawRim() {
    for (int i = 0; i < 4*SCALE_FACTOR; i++) {
        System.out.print("=");
    }
    System.out.println();
}
```
Making the Top of the Torch Scaleable

• For SCALE_FACTOR = 2, we got:
  number of lines = 2
  spaces = line – 1
  colons = -2 * line + 8

• What about SCALE_FACTOR = 3?

<table>
<thead>
<tr>
<th>line</th>
<th>spaces</th>
<th>colons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

number of lines = 3
spaces = ?
colons = ?

• in general, number of lines = ?

Making the Top of the Torch Scaleable (cont.)

• Compare the two sets of formulas:
  \[
  \begin{align*}
  \text{SCALE FACTOR} = 2 & \quad \text{SCALE FACTOR} = 3 \\
  \text{spaces} = \text{line} – 1 & \quad \text{spaces} = \text{line} – 1 \\
  \text{colons} = -2 \times \text{line} + 8 & \quad \text{colons} = -2 \times \text{line} + 12 \\
  \end{align*}
  \]

• There's no change in:
  • the formula for spaces
  • the first constant in the formula for colons

• Use a table for the second constant:
  \[
  \begin{array}{c|c}
  \text{SCALE FACTOR} & \text{constant} \\
  \hline
  2 & 8 \\
  3 & 12 \\
  \end{array}
  \]
  constant = ?

• Scaleable formulas: spaces = line – 1
colons = ?
Revised Method for Drawing the Top of the Torch

```java
public static void drawTop() {
    for (int line = 1; line <= SCALE_FACTOR; line++) {
        // spaces to the left of the current line
        for (int i = 0; i < line - 1; i++) {
            System.out.print(" ");
        }
        // bars and colons on the current line
        System.out.print("|");
        for (int i = 0; i < -2*line + 4*SCALE_FACTOR; i++) {
            System.out.print(":");
        }
        System.out.print("|");
        System.out.println();
    }
}
```

Practice: The Torch Handle

- Pseudocode for default size:
  ```
  ()
  ((()))
  (((()))
  =========
  |::::::|
  |::::::|
  1 |:::|
  2 |:::|
  3 |:::|
  4 |:::|
  ```

- Java code for default size:
  ```java
  public static void drawHandle() {
  ```
  ```
  ```
## Practice: Making the Handle Scaleable

- We again compare two different sizes.

<table>
<thead>
<tr>
<th>SCALE_FACTOR</th>
<th># lines</th>
<th>spaces</th>
<th>colons</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

- number of lines = ?
- spaces = ?
- colons = ?

```java
public static void drawHandle() {
    for (int line = 1; line <= 4; line++) {
        for (int i = 0; i < 2; i++) {
            System.out.print(" ");
        }
        System.out.print("|");
        for (int i = 0; i < 2; i++) {
            System.out.print(":");
        }
        System.out.println("|");
    }
    System.out.println("|");
    for (int i = 0; i < 2; i++) {
        System.out.print(":");
    }
    System.out.println("|");
}
```
Extra Practice: Printing a Pattern, version 3

• Print a triangular pattern with lines containing n, n – 1, ..., 1 asterisks.
  • example:
    
    Enter a positive integer: 3
    ***
    **
    *

• How would we use a nested loop to do this?
  
  for ( ) {
    for ( ) {
      System.out.print("*");
    }
    System.out.println();
  }