Nested Loops

A loop can be nested inside another loop.

- Nested loops consist of an outer loop and one or more inner loops.
- Each time the outer loop is repeated, the inner loops are reentered, and started anew.
Example

**Multiplication table**

Write a program which displays the multiplication table.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
<td>21</td>
<td>24</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>24</td>
<td>28</td>
<td>32</td>
<td>36</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>30</td>
<td>35</td>
<td>40</td>
<td>45</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>12</td>
<td>18</td>
<td>24</td>
<td>30</td>
<td>36</td>
<td>42</td>
<td>48</td>
<td>54</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>42</td>
<td>49</td>
<td>56</td>
<td>63</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
<td>56</td>
<td>64</td>
<td>72</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>18</td>
<td>27</td>
<td>36</td>
<td>45</td>
<td>54</td>
<td>63</td>
<td>72</td>
<td>81</td>
</tr>
</tbody>
</table>
You can use `System.out.printf()` to display formatted output on the console.

```java
... double amount = 1234.601; double interestRate = 0.00528; double interest = amount * interestRate; System.out.printf("Interest = %4.2f", interest);
...```

**Format Specifier**

- `%4.2f`  
  - **Field Width**: 4  
  - **Precision**: 2  
  - **Conversion Code**: `f` for double

---

Zheng-Liang Lu

AP Computer Science A: Java Programming
<table>
<thead>
<tr>
<th>Format Specifier</th>
<th>Output</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>%b</td>
<td>a Boolean value</td>
<td>true or false</td>
</tr>
<tr>
<td>%c</td>
<td>a character</td>
<td>‘a’</td>
</tr>
<tr>
<td>%d</td>
<td>a decimal integer</td>
<td>200</td>
</tr>
<tr>
<td>%f</td>
<td>a floating-point number</td>
<td>45.460000</td>
</tr>
<tr>
<td>%e</td>
<td>a number in standard scientific notation</td>
<td>4.556000e + 01</td>
</tr>
<tr>
<td>%s</td>
<td>a string</td>
<td>“Java is cool”</td>
</tr>
</tbody>
</table>

- By default, a floating-point value is displayed with 6 digits after the decimal point.
Multiple Items to Print

```java
int count = 5;
double amount = 45.56;
System.out.printf("count is \%d and amount is \%f", count, amount);
```

display: count is 5 and amount is 45.560000

- Items must match the format specifiers in order, in number, and in exact type.
- If an item requires more spaces than the specified width, the width is automatically increased.
- By default, the output is right justified.
- You may try the plus sign (+), the minus sign (-), and 0 in the middle of format specifiers.
  - Say % + 8.2f, % - 8.2f, and %08.2f.
public static void main(String[] args) {
    for (int i = 1; i <= 9; ++i) {
        for (int j = 1; j <= 9; ++j) {
            System.out.printf("%3d", i * j);
        }
        System.out.println();
    }
}
Exercise: Coupled Loops

*       *****       *       *****
**      ****        **       ****
***     ***         ***      ***
****    **          ****     **
*****   *           *****    *

(a)     (b)         (c)     (d)
public class PrintStarsDemo {
    public static void main(String[] args) {
        // case (a)
        for (int i = 1; i <= 5; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.printf("*");  // printf method to print stars
            }
            System.out.println();  // println method to print a newline
        }
    }
}
Analysis of Algorithms

- First, there may exist some algorithms for the same problem, which are supposed to be correct.
- Then we compare these algorithms.
- The first question is, Which one is more efficient? (Why?)
- We focus on the growth rate of the running time or space requirement as a function of the input size $n$, denoted by $f(n)$. 
In math, $O$-notation describes the limiting behavior of a function when the argument tends towards a particular value or infinity, usually in terms of simpler functions.

- $f(n) \in O(g(n))$ as $n \to \infty$ if and only if there is a constant $c > 0$ and a real number $n_0$ such that

$$|f(n)| \leq c|g(n)| \quad \forall n \geq n_0. \quad (1)$$

- Note that $O(g(n))$ is a set featured by some simple function $g(n)$.

- Hence $f(n) \in O(g(n))$ is equivalent to say that $f(n)$ is one instance of $O(g(n))$.

\[1\] See any textbook for data structures and algorithms.
For example, $8n^2 - 3n + 4 \in O(n^2)$ (tight bound).

We say $8n^2 - 3n + 4 \in O(n^3)$ and $8n^2 - 3n + 4 \not\in O(n)$. 
Growth Rates for Fundamental Functions

\[ \text{constant} \quad \text{logarithm} \quad \text{linear} \quad \text{n-log-n} \quad \text{quadratic} \quad \text{cubic} \quad \text{exponential} \]

| 1   | \( \log n \) | \( n \) | \( n \log n \) | \( n^2 \) | \( n^3 \) | \( a^n \) |

\(^2\text{See Table 4.1 and Figure 4.2 in Goodrich and etc, p. 161.}\)
• All in all, $O$-notation describes the asymptotic upper bound of complexity of the algorithm.

• So $O$-notation is widely used to classify algorithms by how they respond to changes in input size.\(^4\)
  
  • Time complexity
  • Space complexity

---

\(^3\)The asymptotic sense is that the input size $n$ grows toward infinity.

\(^4\)Actually, there are $\Theta$, $\theta$, $o$, $\Omega$, and $\omega$ which are used to classify algorithms.
References

- https://en.wikipedia.org/wiki/Game_complexity