Jump Statements

The keyword **break** and **continue** are often used in repetition structures to provide additional controls.

- **break**: the loop is **terminated** right after a **break** statement is executed.
- **continue**: the loop **skips** this iteration right after a **continue** statement is executed.
- In practice, jump statements in loops should be conditioned.
Example: Primality

Write a program which determines if the input integer is a prime number.

- Let $x > 1$ be any natural number.
- Then $x$ is said to be a **prime number** if $x$ has no positive divisors other than 1 and itself.
- It is then straightforward to check if it is prime by dividing $x$ by all natural numbers smaller than $x$.
- For speedup, you can divide $x$ by only numbers smaller than $\sqrt{x}$. (Why?)
Scanner input = new Scanner(System.in);
System.out.println("Enter x > 2?");
int x = input.nextInt();
boolean isPrime = true;
input.close();

double upperBd = Math.sqrt(x);
for (int y = 2; y < upperBd; y++) {
    if (x % y == 0) {
        isPrime = false;
        break;
    }
}

if (isPrime) {
    System.out.println("Prime");
} else {
    System.out.println("Composite");
}
Exercise (Revisited)

- Redo the cashier problem by using an infinite loop with a break statement.

```java
while (true) {
    System.out.println("Enter price (-1 to exit): ");
    price = input.nextInt();
    if (price == -1) break;
    sum += price;
}
System.out.println("Total = " + sum);
```
Another Example: Compounding

Write a program which determines the holding years for an investment doubling its value.

- Let $balance$ be the current amount, $goal$ be the goal of this investment, and $r$ be the annual interest rate.
- Then this investment should take at least $n$ years so that the balance of the investment can double its value.
- Recall that the compounding formula is given by

$$balance = balance \times (1 + r/100).$$
... 

```java
int r = 18; // 18%
int balance = 100;
int goal = 200;

int years = 0;
while (balance <= goal) {
    balance *= (1 + r / 100.0);
    years++;
}

System.out.println("Balance = " + balance);
System.out.println("Years = " + years);

...
• A for loop can be an infinite loop by setting true or simply leaving empty in the condition statement.

• An infinite for loop with an if-break statement is equivalent to a normal while loop.
In general, a `for` loop may be used if the number of repetitions is known in advance. If not, a `while` loop is preferred.
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` indicates the field width.
- `.2` specifies the precision.
- `f` denotes the conversion code for floating-point numbers.
<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)
import java.util.Scanner;

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("* ");
            }
            System.out.println();
        }
        // case (b), (c), (d)
        // your work here
    }
}
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

$$|f(n)| \leq c|g(n)| \quad \forall n \geq n_0.$$  \hspace{1cm} (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 or https://en.wikipedia.org/wiki/Big_O_notation.
• For example, $8n^2 - 3n + 4 \in O(n^2)$.
• We could say that $8n^2 - 3n + 4 \in O(n^3)$ and $8n^2 - 3n + 4 \notin O(n)$. 
Common Fundamental Functions

See Table 4.1 and Figure 4.2 in Goodrich and etc, p. 161.

---

<table>
<thead>
<tr>
<th></th>
<th>constant</th>
<th>logarithm</th>
<th>linear</th>
<th>n-log-n</th>
<th>quadratic</th>
<th>cubic</th>
<th>exponential</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>log n</td>
<td>n</td>
<td>n log n</td>
<td>n²</td>
<td>n³</td>
<td>aⁿ</td>
<td></td>
</tr>
</tbody>
</table>

---

See Table 4.1 and Figure 4.2 in Goodrich and etc, p. 161.