Multiple Alternative if Statements

```c
if (score >= 90.0)
    grade = 'A';
else
    if (score >= 80.0)
        grade = 'B';
    else
        if (score >= 70.0)
            grade = 'C';
        else
            if (score >= 60.0)
                grade = 'D';
            else
                grade = 'F';
```

Equivalent

```c
if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';
```
Multi-Way if-else Statements

- If score >= 90, grade = 'A'
- If score >= 80, grade = 'B'
- If score >= 70, grade = 'C'
- If score >= 60, grade = 'D'
- Otherwise, grade = 'F'

Borrowed from Dr. Anca Doloc-Mihu (lecture 7) http://www.mathcs.emory.edu/~cs170003/cal.html
Trace if-else statement

Suppose score is 70.0

if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';

The condition is false
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
Suppose score is 70.0

if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';

The condition is true
Trace if-else statement

Suppose score is 70.0

```java
if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';
```

grade is C
Trace if-else statement

Suppose score is 70.0

if (score >= 90.0)
    grade = 'A';
else if (score >= 80.0)
    grade = 'B';
else if (score >= 70.0)
    grade = 'C';
else if (score >= 60.0)
    grade = 'D';
else
    grade = 'F';

Exit the if statement
Note

The **else** clause matches the most recent **if** clause in the same block.

```java
int i = 1;
int j = 2;
int k = 3;

if (i > j)
   if (i > k)
      System.out.println("A");
else
   System.out.println("B");
```

Equivalent

```java
int i = 1;
int j = 2;
int k = 3;

if (i > j)
   if (i > k)
      System.out.println("A");
else
   System.out.println("B");
```
Note, cont.

Nothing is printed from the preceding statement. To force the `else` clause to match the first `if` clause, you must add a pair of braces:

```java
int i = 1;
int j = 2;
int k = 3;
if (i > j) {
    if (i > k)
        System.out.println("A");
}
else
    System.out.println("B");
```

This statement prints B.
Common Errors

Adding a semicolon at the end of an if clause is a common mistake.

```java
if (radius >= 0);
{
    area = radius*radius*PI;
    System.out.println("The area for the circle of radius " + radius + " is " + area);
}
```

This mistake is hard to find, because it is not a compilation error or a runtime error, it is a logic error.

This error often occurs when you use the next-line block style.
if (number % 2 == 0)
    even = true;
else
    even = false;

Equivalent

boolean even = number % 2 == 0;

(a) (b)
CAUTION

(a) if (even == true)  
   System.out.println("It is even.");

(b) Equivalent

   if (even)  
   System.out.println("It is even.");
Problem: Computing Taxes

The US federal personal income tax is calculated based on the filing status and taxable income. There are four filing statuses: single filers, married filing jointly, married filing separately, and head of household. The tax rates for 2009 are shown below.

<table>
<thead>
<tr>
<th>Marginal Tax Rate</th>
<th>Single</th>
<th>Married Filing Jointly or Qualifying Widow(er)</th>
<th>Married Filing Separately</th>
<th>Head of Household</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>$0 – $8,350</td>
<td>$0 – $16,700</td>
<td>$0 – $8,350</td>
<td>$0 – $11,950</td>
</tr>
<tr>
<td>15%</td>
<td>$8,351 – $33,950</td>
<td>$16,701 – $67,900</td>
<td>$8,351 – $33,950</td>
<td>$11,951 – $45,500</td>
</tr>
<tr>
<td>35%</td>
<td>$372,951+</td>
<td>$372,951+</td>
<td>$186,476+</td>
<td>$372,951+</td>
</tr>
</tbody>
</table>

Borrowed from Dr. Anca Doloc-Mihu (lecture 8) http://www.mathcs.emory.edu/~cs170003/cal.html
Problem: Computing Taxes, cont.

if (status == 0) {
    // Compute tax for single filers
}
else if (status == 1) {
    // Compute tax for married file jointly
    // or qualifying widow(er)
}
else if (status == 2) {
    // Compute tax for married file separately
}
else if (status == 3) {
    // Compute tax for head of household
}
else {
    // Display wrong status
}
# Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
</tr>
<tr>
<td>&amp; &amp;</td>
<td>and</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>exclusive or</td>
</tr>
</tbody>
</table>
### Truth Table for Operator `!`

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\neg p$</th>
<th>Example (assume $age = 24$, gender = 'M')</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
<td>$(age &gt; 18)$ is false, because $(age &gt; 18)$ is true.</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>$(gender \neq 'M')$ is true, because $(grade \neq 'M')$ is false.</td>
</tr>
</tbody>
</table>
Truth Table for Operator &&

<table>
<thead>
<tr>
<th>p1</th>
<th>p2</th>
<th>p1 &amp;&amp; p2</th>
<th>Example (assume age = 24, gender = 'F')</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>(age &gt; 18) &amp;&amp; (gender == 'F') is true, because (age &gt; 18) and (gender == 'F') are both true.</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>(age &gt; 18) &amp;&amp; (gender != 'F') is false, because (gender != 'F') is false.</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td></td>
</tr>
</tbody>
</table>

Borrowed from Dr. Anca Doloc-Mihu (lecture 8) http://www.mathcs.emory.edu/~cs170003/cal.html
### Truth Table for Operator $\lor$

<table>
<thead>
<tr>
<th>p1</th>
<th>p2</th>
<th>p1 $\lor$ p2</th>
<th>Example (assume age = 24, gender = 'F')</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>(age &gt; 34) $\lor$ (gender == 'F') is true, because (gender == 'F') is true.</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>(age &gt; 34) $\lor$ (gender == 'M') is false, because (age &gt; 34) and (gender == 'M') are both false.</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td></td>
</tr>
</tbody>
</table>
Truth Table for Operator ^

<table>
<thead>
<tr>
<th>p1</th>
<th>p2</th>
<th>p1 ^ p2</th>
<th>Example (assume age = 24, gender = 'F')</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>(age &gt; 34) ^ (gender == 'F') is true, because (age &gt; 34) is false but (gender == 'F') is true.</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>(age &gt; 34)</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td></td>
</tr>
</tbody>
</table>
Examples

Program that checks whether a number is divisible by 2 and 3:

```java
System.out.println("Is "+ number + " divisible by 2 and 3? " +
    ((number % 2 == 0) && (number % 3 == 0)));
```

Program that checks whether a number is divisible by 2 or 3:

```java
System.out.println("Is "+ number + " divisible by 2 or 3? " +
    ((number % 2 == 0) || (number % 3 == 0)));
```
Examples

Program that checks whether a number is divisible by 2 or 3 but not both:

System.out.println("Is " + number + " divisible by 2 or 3, but not both? " + 
((number % 2 == 0) ^ (number % 3 == 0)));
Problem: Determining Leap Year?

This program first prompts the user to enter a year as an `int` value and checks if it is a leap year.

A year is a leap year if it is divisible by 4 but not by 100, or it is divisible by 400.

\[(\text{year} \mod 4 == 0 \land \text{year} \mod 100 != 0) \lor (\text{year} \mod 400 == 0)\]

Borrowed from Dr. Anca Doloc-Mihu (lecture 8) http://www.mathcs.emory.edu/~cs170003/cal.html
Problem: Lottery

Write a program that randomly generates a lottery of a two-digit number, prompts the user to enter a two-digit number, and determines whether the user wins according to the following rule:

- If the user input matches the lottery in exact order, the award is $10,000.
- If the user input matches the lottery, the award is $3,000.
- If one digit in the user input matches a digit in the lottery, the award is $1,000.
**switch Statements**

Too many nested if statements make program difficult to read. Solution is to use the switch statement instead.

```java
switch (status) {
    case 0: compute taxes for single filers;
           break;
    case 1: compute taxes for married file jointly;
           break;
    case 2: compute taxes for married file separately;
           break;
    case 3: compute taxes for head of household;
           break;
    default: System.out.println("Errors: invalid status");
             System.exit(1);
}
```
switch Statement Flow Chart

- status is 0
  - Compute tax for single filers
  - break

- status is 1
  - Compute tax for married jointly or qualifying widow(er)
  - break

- status is 2
  - Compute tax for married filing separately
  - break

- status is 3
  - Compute tax for head of household
  - break

- default
  - Default actions
  - break
**switch Statement Rules**

The `switch`-expression must yield a value of `char`, `byte`, `short`, or `int` type and must always be enclosed in parentheses.

```java
switch (switch-expression) {
    case value1: statement(s)1;
        break;
    case value2: statement(s)2;
        break;
    «
    case valueN: statement(s)N;
        break;
    default: statement(s)-for-default;
}
```

The `value1`, `value2`, ..., and `valueN` must have the same data type as the value of the `switch-expression`. The resulting statements in the `case` statement are executed when the value in the `case` statement matches the value of the `switch-expression`. Note that `value1`, `value2`, ..., and `valueN` are constant expressions, meaning that they cannot contain variables in the expression, such as `1 + x`.

Borrowed from Dr. Anca Doloc-Mihu (lecture 8) http://www.mathcs.emory.edu/~cs170003/cal.html
The keyword `break` is optional, but it should be used at the end of each case in order to terminate the remainder of the `switch` statement. If the `break` statement is not present, the next `case` statement will be executed.

The `default` case, which is optional, can be used to perform actions when none of the specified cases matches the `switch-expression`.

```java
switch (switch-expression) {
    case value1:  statement(s)1;
    break;
    case value2: statement(s)2;
    break;
    ...
    case valueN: statement(s)N;
    break;
    default: statement(s)-for-default;
}
```

The `case` statements are executed in sequential order, but the order of the cases (including the default case) does not matter. However, it is good programming style to follow the logical sequence of the cases and place the default case at the end.
Suppose ch is 'a':

```java
switch (ch) {
    case 'a': System.out.println(ch);
    case 'b': System.out.println(ch);
    case 'c': System.out.println(ch);
}
```
Trace switch statement

```java
switch (ch) {
    case 'a': System.out.println(ch);
    case 'b': System.out.println(ch);
    case 'c': System.out.println(ch);
}
```

ch is 'a':
Trace switch statement

```java
switch (ch) {
    case 'a': System.out.println(ch);
    case 'b': System.out.println(ch);
    case 'c': System.out.println(ch);
}
```
Trace switch statement

```
switch (ch) {
    case 'a': System.out.println(ch);
    case 'b': System.out.println(ch);
    case 'c': System.out.println(ch);
}
```
Trace switch statement

```java
switch (ch) {
    case 'a': System.out.println(ch);
    case 'b': System.out.println(ch);
    case 'c': System.out.println(ch);
}
```

Execute this line.
Trace switch statement

```java
switch (ch) {
    case 'a': System.out.println(ch);
    case 'b': System.out.println(ch);
    case 'c': System.out.println(ch);
}
```

Next statement;
Trace switch statement

Suppose ch is 'a':

```
switch (ch) {
    case 'a': System.out.println(ch);
             break;
    case 'b': System.out.println(ch);
             break;
    case 'c': System.out.println(ch);
}
```
trace switch statement

```java
switch (ch) {
    case 'a': System.out.println(ch);
               break;
    case 'b': System.out.println(ch);
               break;
    case 'c': System.out.println(ch);
}
```

ch is 'a':
Trace switch statement

```java
switch (ch) {
    case 'a': System.out.println(ch);
              break;
    case 'b': System.out.println(ch);
              break;
    case 'c': System.out.println(ch);
}
```

Execute this line
Trace switch statement

```java
switch (ch) {
    case 'a': System.out.println(ch);
        break;
    case 'b': System.out.println(ch);
        break;
    case 'c': System.out.println(ch);
}
```

Execute this line

Borrowed from Dr. Anca Doloc-Mihu (lecture 8) http://www.mathcs.emory.edu/~cs170003/cal.html
Trace switch statement

```java
switch (ch) {
    case 'a': System.out.println(ch);
        break;
    case 'b': System.out.println(ch);
        break;
    case 'c': System.out.println(ch);
}
```

Next statement;
Problem: Chinese Zodiac

Write a program that prompts the user to enter a year and displays the animal for the year.

\[
\text{year} \mod 12 = \begin{cases} 
0: \text{monkey} \\
1: \text{rooster} \\
2: \text{dog} \\
3: \text{pig} \\
4: \text{rat} \\
5: \text{ox} \\
6: \text{tiger} \\
7: \text{rabbit} \\
8: \text{dragon} \\
9: \text{snake} \\
10: \text{horse} \\
11: \text{sheep} 
\end{cases}
\]