SELECTION IDIOMS

The programming idioms for selection statements depend on the concept of *mutual exclusion*. Two truth values are *mutually exclusive* if no more than one of them can be true. Two actions are mutually exclusive if no more than one of them can be done.

Here is a summary of appropriate selection idioms:

<table>
<thead>
<tr>
<th>Action</th>
<th>Condition</th>
<th>Construct to Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutually Exclusive</td>
<td>Mutually Exclusive</td>
<td>Sequential <em>if</em> statements†</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cascading <em>if-else</em> statements†</td>
</tr>
<tr>
<td></td>
<td></td>
<td>switch statement*</td>
</tr>
<tr>
<td>Mutually Exclusive</td>
<td>NOT Mutually Exclusive</td>
<td>Cascading <em>if-else</em> statements</td>
</tr>
<tr>
<td>NOT Mutually Exclusive</td>
<td>doesn’t matter</td>
<td>Sequential <em>if</em> statements</td>
</tr>
</tbody>
</table>

† Either of these constructs may be used.
* The switch statement can only be used for integer, *char* or *String* data.
If both the actions and the conditions under which they are to be selected are mutually exclusive, you can use either *if* statements, cascading *if-else* statements or sometimes even a *switch*.

**Example**

We want to simulate the tossing of a coin by choosing 0 or 1 randomly and interpreting 0 as “tails” and 1 as “heads.”

The actions are mutually exclusive – we want either TAILS or HEADS but not both.

The conditions are mutually exclusive – the randomly chosen number is either 0 or 1 but not both.

Therefore, any of the three following code fragments will work correctly.

```java
int toss = (int)(Math.random() * 2);
if ( toss == 0 )
    coinToss = "TAILS";
if ( toss == 1 )
    coinToss = "HEADS";

int toss = (int)(Math.random() * 2);
if ( toss == 0 )
    coinToss = "TAILS";
else // toss == 1
    coinToss = "HEADS";

switch ( (int)(Math.random() * 2) )
{
    case 0:
        coinToss = "TAILS";
        break;
    case 1:
        coinToss = "HEADS";
        break;
}
```
If the actions are mutually exclusive but the conditions under which they are to be selected are not, then you must use a cascading if-else.

**Example**
An university confers awards on its graduates as follows:

<table>
<thead>
<tr>
<th>Grade Point Average</th>
<th>Award</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.9 – 4.0</td>
<td>Summa cum laude</td>
</tr>
<tr>
<td>3.55 – 3.9</td>
<td>Magna cum laude</td>
</tr>
<tr>
<td>3.25 – 3.55</td>
<td>Cum laude</td>
</tr>
</tbody>
</table>

The actions are mutually exclusive – a student receives at most 1 award.

The conditions are not mutually exclusive – a grade point average that exceeds 3.9 also exceeds 3.55.

Thus, a cascading if-else is the best choice.

```java
if ( gpa >= 3.90 )
    award = "Summa cum laude";
else if ( gpa >= 3.55 )
    award = "Magna cum laude";
else if ( gpa >= 3.25 )
    award = "Cum laude";
else
    award = "NONE";
```
When the actions are not mutually exclusive, then use serial `if` statements.

*Example*

In a certain state, a highway driver who travels over the speed limit gets a $100 fine, which is doubled if he or she is also in a construction zone.

The actions – speeding and traveling in a construction zone – are not mutually exclusive since both can happen.

Thus, the clearest code uses serial `if` statements.

```c
fine = 0;
if ( speed > speedLimit )
    fine = 100;
if ( constructionZone )
    fine *= 2;
```
### Programming Exercises

For each of the following exercises, invent an algorithm that solves the problem, code it into a Java application and deploy it. Follow all programming conventions. Incorporate user-friendly input techniques using scanners and/or dialog boxes and create readable output through the use of formatter objects or string formatting methods.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| **1.** | Input a temperature scale (the character **F** or **C**) and value (floating point) as shown in the picture to the right.  

“F” scale means “Fahrenheit”; “C” means “Celsius.”  

If the input is a Fahrenheit temperature, convert it to the Celsius scale; and vice-versa. Output the result so that it looks like shown to the right. |
| ![Input](image1.png) | ![Message](image2.png) |
| **2.** | There are several types of triangles – scalene, isosceles, equilateral – based on the relationship between the lengths of the sides. Write a Java application that inputs three values representing the lengths of the sides. First determine if the three values can make a triangle and print a message if they cannot. If you are unfamiliar with this idea, Google *triangle inequality* to learn about it.  

If the three values can make up a triangle then print a message indicating its type (scalene, isosceles or equilateral). |
| **3.** | Modify your solution to the previous problem so that it also prints a message indicating if the triangle is a right triangle. Note that a right triangle can also be scalene or isosceles. |
4. Many supermarkets now have self-checkout kiosks where the shopper tallies his or her own purchases. The machine tallies the total sales amount owed by the shopper and asks for payment. If the shopper enters cash or asks for cash back, the machine must dispense his or her change in the form of 20-dollar, 10-dollar, 5-dollar and 1-dollar bills; and quarters, dimes, nickels and pennies.

Write a Java application that computes the amount to be dispensed from two input values – the sales amount owed and the amount of money tendered by the customer. Output the numbers of bills and coins to be dispensed.

Suppress any output where the count is zero. Also, make noun plurality agree. For example:

<table>
<thead>
<tr>
<th>OK</th>
<th>Not OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 quarters</td>
<td>2 quarters</td>
</tr>
<tr>
<td>1 dime</td>
<td>1 dimes</td>
</tr>
<tr>
<td>1 penny</td>
<td>0 nickels</td>
</tr>
<tr>
<td></td>
<td>1 pennies</td>
</tr>
</tbody>
</table>

5. A wholesale glass company sells flower vases to retail flower shops, which must order a minimum of 10 and in multiples of 10. The glass company wants a program that inputs the number of vases ordered and outputs a packing slip.

The packing slip tells the shipping clerk how to pack the order. The clerk has small, medium, and large boxes that hold ten, twenty, and fifty vases, respectively. He or she always tries to use the least number of boxes by filling larger boxes before smaller ones. For example, for an order of 180 vases the clerk would use three large boxes, one medium box, and one small box.

Suppress any output where the count is zero. Also, make noun plurality agree. For example:

<table>
<thead>
<tr>
<th>OK</th>
<th>Not OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packing slip</td>
<td>Packing slip</td>
</tr>
<tr>
<td>Order: 120 vases</td>
<td>Order: 120 vases</td>
</tr>
<tr>
<td>2 large boxes</td>
<td>2 large boxes</td>
</tr>
<tr>
<td>1 medium box</td>
<td>1 medium boxes</td>
</tr>
<tr>
<td></td>
<td>0 small boxes</td>
</tr>
</tbody>
</table>
6. Write a Java application that reads a 3-letter word and determines whether or not it is a palindrome, which reads the same forwards as backwards. For example, *tot, dad, mom* and *sis* are palindromes.

7. Write a Java application that inputs a person’s name in the format:

   \[ \text{first name } \text{space} \text{middle name} \text{space} \text{last name} \]

Then displays the name in the format:

   \[ \text{last name comma } \text{space} \text{first name} \text{space} \text{middle initial dot} \]

Subject to the following truncation rules, applied in the order given:

1. If the name’s length exceeds a predefined length limit, remove the dot after the middle initial.
2. If it still exceeds the limit, truncate the first name to a single initial without a dot.
3. If it still exceeds the limit, eliminate the middle initial and its preceding space.
4. If it still exceeds the limit, truncate the last name to a prefix such that the result (i.e. *last name comma space first initial*) fits exactly within the limit.

The length limit must be defined as a constant identifier.

For example, for a length limit of 20, the name *JOHN QUINCY ADAMS* would be output as *ADAMS, JOHN Q.* since it is 14 characters long and so none of the truncation rules apply.

The name *KALAMBADI MUHAMMAD MUSLIYAR* would be output as *MUSLIYAR, K M* after applying truncation rules 1 and 2.