Computer Science 1-2
Lab #2

We are now going to talk about one way Java has to allow a program to take an alternative action based on whether a certain condition is true or false. In an **if statement**, code is executed if the condition is true. If the condition is false, the program will skip ahead to the next section of code. Here is a table of ways we can compare values in Java (which is slightly different than in math subjects).

Standard Algebraic Expression	Java Expression	Java Example	Meaning of Java Condition
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y

Here is an example of what a basic **if statement** looks like:

```
 if (firstNumber == secondNumber) {

    System.out.println("These numbers are the same!");

 }
```

Note that we put the condition we want to test inside parentheses next to the "if" and the code we want to execute lives inside the curly braces (which lets us know that these statements "belong" to this particular "if"). For readability we indent the statements inside the braces. It turns out that indentation is one of the more important conventions we follow (along with comments!) when writing code.

One important thing that students don't always understand is that when we say "statement" we mean like in the example above where we embed that print statement inside the "set braces". But that's not quite right. We could put a whole bunch of statements inside a pair of set braces and the "if condition" would control them. So, we could do this:

```
if (firstNumber == secondNumber) {

   System.out.println("These numbers are the same!");

   a = a+2;
   System.out.println(“The value of a is “ + a);
 }
```

The other thing to note is that a statement can be just a single statement with no set braces.

So, our original example:

```
 if (firstNumber == secondNumber) {
   System.out.println(“These numbers are the same!”);
 }
```

could be written equivalently as:

```
 if (firstNumber == secondNumber)
   System.out.println(“These numbers are the same!”);
```

In this case the **if condition** controls only the very next statement.

To summarize all the above, a **statement** can either be a single statement followed by a semicolon or a **compound statement** which is bounded by set braces.

This simple statement is great for when we only want to carry out some code when something is true…but more often than not, we will want to do one thing if the statement is true and something else if the statement is false. To accomplish this, let’s look at the Java statement which accomplishes this. This statement **if … else statement** behaves exactly as it sounds. If the condition is true, the code associated with the **if** is executed. If the condition is false, the code associated with the **else** is executed.

Here is a quick example:

```
If (grade >= 60) {
   System.out.println(“You pass!”);
} else {
    System.out.println(“You fail”);
    System.out.println(“You must take the class again!”);

  }
```

The lining up of the brackets is not a compiler issue, but is one convention shared by many programs. There are other conventions. Just pick one convention and always follow it as it will make it easier for you to read and understand your code.

When we want to test a series of conditions, we will want to “nest” **if … else statements** together. For example:

```
If (firstNumber == secondNumber) {
  System.out.println(“These numbers are the same!”);
}  else if (firstNumber > secondNumber) {
     System.out.println(“The first number is larger!”);
   } else {
       System.out.println(“The second number is largest”);

     }
```

In this example, notice that we can put the **else if** statement together on the same line right after the curly brace. The final **else** sits by itself because it goes with the “if” statement directly above it. We won’t have any more conditions to test (we checked to see if they were equal, then if the first number was bigger…if neither statement is true, then the second number must be larger, so we don’t need another if statement). Remember that elegant programs try to take advantage of known information.

One last note…last week we talked about the modulus operator (briefly). It is a function that lets us look at the remainder after we divide two numbers. So 5 % 4 gives 1 (that’s the remainder when dividing 5 by 4). In math class we write this as 5 mod 4. This turns out to be very useful for some programming tasks.

THE PROBLEMS!

1. Body Mass Index (BMI) calculator. Write a program that asks the user to input their weight in pounds and height in inches and then calculates and displays the user’s BMI. Here is the BMI formula.

$$BMI = \frac{weight \times 703}{height^2}$$

The program should also indicate if the person is underweight, normal, overweight or obese as defined by this Health and Human Services Chart:

HHS Classification	**BMI Value**
Underweight	Less than 18.5
Normal	Between 18.5 and 25
Overweight	Between 25 and 30
Obese	Greater than 30

2. Write a program that takes in the total points on an exam, the score the student got on the exam, and outputs their letter grade. Use a 90,80,70 etc. grading scale to decide which percentage scores give which letter grades. Note: the total points on the exam can be any integer number. The percentage score should be a **double.**
3. Write a program that reads an integer and determines and prints whether it is odd or even. (Hint: Use the modulus operator (%); any multiple of 2 has a remainder of zero when divided by 2).
4. Write a program that reads in two integers and determines and prints if the first is a multiple of the second (Hint: use the modulus operator).
5. **Challenge:** Write a program that inputs a five-digit integer, separates the integer into its digits and prints them separated by 3 spaces each. For instance, if the user enters 42339, the program should print out: 4   2   3   3   9. **Hint:** Think about some of the behavior we have seen with integers, division, and the modulus operator. One more hint: In this example, suppose the number 42339 is in an integer. If I divide that number by 10, the result is 4233. If I "mod" that number by 10 (using the % operator), the result is 9. This gives us a way of peeling apart the number.

Caution: to get full credit for the program the "challenge" program must be done. It is the hardest part of the assignment and that's why it's called "challenge".