COP 3330 – Object Oriented Programming in Java
Boolean Expressions and the If Statement

Boolean Expressions

boolean (lowercase) is a type in Java. The two possible boolean literals are true and false, both lowercase. Thus, the following is a valid Java statement:

```
boolean ok = true;
```

This initializes a variable named ok to true.

Relational Operators

We can create Boolean expressions without boolean variables via the six following relational operators that compare primitive quantitative types:

Relation	Operator
Greater Than	>
Greater Than or Equal To	>=
Less Than	<
Less Than or Equal To	<=
Equal To	==
Not Equal To	!=

Note that these are the exact same in C and Python. In C however, the result of an operation using these is an integer (0 or 1) instead of a boolean, since C has not separate boolean type like Python and Java do.

Thus, if the variable age is set to 40, the expression

```
age >= 21
```

evaluates to true, while the expression

```
age < 40
```

evaluates to false.

Boolean Operators

Once we have relational operators, we can create simple boolean expressions. To create more complex boolean expressions, we can put together multiple simple boolean expressions with boolean operators. Here are Java's boolean operators:

Operator	Symbol in Java
and	&&
or	||
not	!

You'll note that these are the same in C and different in Python, which just uses the words.

If a variable age was set to 17, then the expression

```
age >= 16 and age <= 18
```

would evaluate to true.

Note that in Java, the expression

```
16 <= age <= 18
```

does not compile. In both C and Java, this compiles. In C, `16 <= age`, would evaluate to 1 (true), and then 1 would be deemed to be less than or equal to 18 so the expression would be true in C. Python secretly inserts the keyword and when interpreting, so the interpreter actually sees

```
16 <= age and age <= 18
```

without the programmer typing it. (Python is notorious for having syntactic sugar that fixes common programming errors.)

The nice thing about Java having more strict rules about boolean expressions is that the compiler catches more errors that the programmer makes, and if the compiler catches it, it draws the programmer's attention and more often than not, they fix it immediately instead of having to spend a long time debugging.

Note that if age were set to 19 then

```
age >= 16 and age <= 18
```

would evaluate to true.

Finally, note that if we switch the and to an or,

```
age >= 16 or age <= 18
```

is always true regardless of what age is equal to. Do you see why?

<u>If Statement</u>

The if statement in all three languages (C, Python, Java) works similarly, each with slightly different syntax. C's is more similar to Java's than Python, mostly due to how blocks of statements are defined in Python versus both C and Java.

<u>Block of Statements</u>

In Python, indenting is used to signify a block of statements inside of a control structure. In both C and Java, curly braces are used to signify a block of statements inside a control structure. There are two commonly used styles with a block of statements:

```
construct {                      construct
    // Indent optional           {
    // but strongly suggested        // same here!

}                                }
```

After the beginning syntax of the control structure, an open brace is either placed on that line right afterwards, OR an open brace is placed on the next line left-justified with the control structure start. The block of statements ends whenever there is a matching closing curly brace. This should be left-justified to match the start of the control structure.

In all three languages, the official syntax only allows for one statement to be within a control structure. BUT, it's natural to want more than one statement in the body of a control structure, so all three compilers treat a block of statements as a single statement.

In Java (also in C), if you forget the curly braces, then only the very next statement will be inside of the construct. In Python, this is not an issue because in Python, the indenting is very clear as to what belongs in the control structure and what does not.

<u>If statement – Version 1</u>

The most simple version of the if statement has the following syntax:

```
if (boolean expression)
    stmt;
```

If you want more than one statement inside the if, make a block of statements:

```
if (boolean expression) {
    stmt1;
    ...
    stmtk;
}
```

This works as you expect. If the boolean expression evaluates to true, execute all the statements inside of the if. If it does not, skip over them and go to the first statement after the if completes.

<u>If statement – Version 2</u>
In the first version of the if statement, no action is taken in some cases. But, it's possible that no matter what is true, one action or another is going to take place. For these types of situations, we use the second version of the if statement:

```
if (boolean expression)
    stmt;
else
    stmt;
```

With blocks of statements it would look like this:

```
if (boolean expression) {
    stmta1;
    ...
    stmtak;
}
else {
    stmtb1;
    ...
    stmtbk;
}
```

For this construct, we evaluate the boolean expression. If it's true, we execute all the statements from stmta1 through stmtak and then skip to the end of the if-else. If it's false, we execute all the statements form stmtb1 through stmtbk and then continue to the next line after the end of the if-else.

<u>If statement – Version 3</u>
If we want to check multiple different conditions, in a particular order we can use the following:

```
if (boolean expression 1)
    stmt1;
else if (boolean expression 2)
    stmt2;
else if (boolean expression 3)
    stmt3;
....
else if (boolean expression k)
    stmtk;
```

For this construct, we evaluate each boolean expression in order. When we find the first one that evaluates to true (if there is one), we would execute the block of statements inside of that portion of the construct. If none of the boolean expressions are true, then none of the blocks of statements inside of construct get executed. Thus, either 1 of these blocks of statements runs or none of them do. Note that in Python, the keyword elif is used instead of two separate words "else if".

If statement – Version 4
If we want to check multiple different conditions, in a particular order we can use the following:

```
if (boolean expression 1)
    stmt1;
else if (boolean expression 2)
    stmt2;
else if (boolean expression 3)
    stmt3;
....
else if (boolean expression k)
    stmtk;
else
    stmtn;
```

Same as the previous one, but if all of the boolean expressions evaluate to false, then we will execute stmtn before moving on.

Generally speaking, both C and Java have these four versions of the if statement and they work exactly the same in both of those languages (minus nuaces due to blocks of statements and the boolean type), but the actual rules for the if construct are identical in the three languages and work as described above.

If Statement Design/Caveats
With just these four constructs, a programmer can create billions of different programs using the same 30-40 lines of code. Logically, two separate if statements are not equivalent to a single if-else statement. So rearranging statements, changing where blocks are, etc. can affect the meaning of the code and how it executes. The most common error that I see introductory students making is that they always feel that they have to attach an else to an if. The reality is that many logical tasks do NOT necessitate this.

<u>Example Using Framework #1</u>
Here is an example using the first framework. Some items are taxed at the grocery store while others are not. In this program we'll ask the user the original price of the item and whether or not sales tax is applied. Then we'll print out the total cost of the item. The tax percentage will be a constant in the program. Here's the entire program:

```
import java.util.*;

public class SalesTax {

     final public static double SALES_TAX_RATE = 0.065;

     public static void main(String[] args) {

          // Get cost.
          Scanner stdin = new Scanner(System.in);
          System.out.println("How much did your item cost?");
          double cost = stdin.nextDouble();

          // Get if it's taxed or not.
          System.out.println("Is it taxed?(0=no,1=yes)");
          int tax = stdin.nextInt();

          // Add tax only if we need to.
          if (tax == 1)
               cost += (cost*SALES_TAX_RATE);

          // Print out final cost.
          System.out.printf("Your final cost is $%.2f.\n", cost);
     }
}
```

Since we only need one statement inside of our if, no block of statements is needed. One should indent though. In Python the indenting is required, but in Java, the code would still compile and work even if the line inside of the if wasn't properly indented. But not indenting is bad style and makes it more difficult to debug.

A good exercise is to change the double equal sign in the if to a single equal. What happens? Is it the same as other languages you have used? (This is intentionally wrong, but it's good to experiment to play around so that you start learning what happens when you make typos. That way you can fix the errors faster.)

Example Using Framework #2

A classic program shown in nearly every intro programming class is converting temperature from Celsius to Fahrenheit. Naturally, it makes sense to also be able to convert from Fahrenheit to Celsius (to help the Europeans out =)) In this program, we ask the user which of the two conversions they want to carry out and then ask them for the temperature that they want to convert. This type of program lends itself very nicely to the if-else structure, since we're offering two alternatives and we will always do one or the other.

Here's the program:

```
import java.util.*;

public class Temp {

	public static void main(String[] args) {

		Scanner stdin = new Scanner(System.in);

		// Get the desired conversion.
		System.out.println("Would you like to...");
		System.out.println("1. Convert from Celsius to Fahrenhiet.");
		System.out.println("2. Convert from Fahrenhiet to Celsius.");
		int ans = stdin.nextInt();

		double fahr, cel;

		// cel -> fahr.
		if (ans == 1) {

			// Read in the input temperature.
			System.out.println("What is the temperature in Celsius?");
			cel = stdin.nextDouble();

			// Calculate the corresponding temp in Fahrenheit and output.
			fahr = 1.8*cel + 32;
			System.out.print("The converted temperature is "+fahr);
			System.out.println(" degrees Fahrenheit.");
		}

		// fahr -> cel
		else {

			// Read in the input temperature.
			System.out.println("What is the temperature in Fahrenheit?");
			fahr = stdin.nextDouble();

			// Calculate the corresponding temp in Celsius and output.
			cel = 5*(fahr - 32)/9;
			System.out.print("The converted temperature is "+cel);
			System.out.println(" degrees Celsius.");
		}
	}
}
```

Here we provide the user a menu of 2 choices and then put nearly all of the code for each choice in the corresponding block. Thus, we use two blocks of statements. Then, inside each block, we do the work of that particular task. Since the print statement we must present the user is different in each case, two separate prints reside in the two branches of the if-else. From there, we do the appropriate calculation and then print out the final result.

Example Using Framework #3
The example chosen for the notes to show the third framework is calculating the user's age. In most instances, we can get someone's age by subtracting the year they were born from the current year. But this doesn't always work. Specifically, if a person's birthday has not yet occurred in the current year this does not work. So a good strategy for a program that calculates someone's age is:

Subtract years as previously described.

Then, check if the birthday hasn't happened yet in the current year. If so, subtract one from the original estimate. For our purposes, we'll break down the if check into two separate cases:

1) It's not yet the user's birthday month.
2) It's the user's birthday month, but the day hasn't yet occurred.

Here's the code:

```
import java.util.*;

public class Age {

    final public static int CURYEAR = 2026;

    public static void main(String argv[]) {
            Scanner stdin = new Scanner(System.in);
            System.out.println("Enter the month of your birthday:");
            int month = stdin.nextInt();
            System.out.println("Enter the day of your birthday:");
            int day = stdin.nextInt();
            System.out.println("Enter the year of your birthday:");
            int year = stdin.nextInt();
            System.out.println("Enter today's month:");
            curmonth = stdin.nextInt();
            System.out.println("Enter today's day:");
            curday = stdin.nextInt();

            int age = CURYEAR - year;

            if (curmonth < month)
                  age--;
            else if (curmonth == month && curday < day)
                  age--;

            System.out.println("You are " + age + " years old.");
    }
}
```

Essentially, after reading in all of the input, we assign age, but then in two cases, handled by and if and else if, we update the age as necessary. Note that there are many other ways to write this program.

Example Using Framework #4

The classic example utilizing framework #4 is the grade program where the user enters a numeric grade and the program outputs the equivalent letter grade, A, B, C, D or F. Here, there are several options and exactly one of them always occurs.

```
import java.util.*;

public class ClassGrade {

      public static void main(String[] args) {

            // Get input.
            Scanner stdin = new Scanner(System.in);
            System.out.println("What percent(integer) did you get?");
            int grade = stdin.nextInt();
            char letter;

            // Criterion for an A.
            if (grade >= 90)
                  letter = 'A';

            // Only works since we alread screened out As.
            else if (grade >= 80)
                  letter = 'B';

            // If we get here and this is true it's a C.
            else if (grade >= 70)
                  letter = 'C';

            // A D here.
            else if (grade >= 60)
                  letter = 'D';

            // Sorry if none of those conditions are true, you get an F!
            else
                  letter = 'F';

            // Show letter grade.
            System.out.println("Your letter grade is "+letter+".");
      }
}
```

Here we only had one line of code for each option, but it's typical to have multiple lines of code for each option in which case a block of code would be necessary for each. It's important to note that the order of these branches matter, since the boolean expressions produced aren't mutually exclusive. If I checked >= 60 first and assigned a D, then everyone would get a D or F =)