Lesson 4: Control Flow

4.1 Introduction:

In a futuristic world, every person has a personal robot butler (robutle). These robots can complete simple tasks but need to be given explicit instructions for every task. You finish cooking a meal and want to have your robutle wash the dishes. Unfortunately, robutles can't just "do the dishes" they have to be given explicit, simple tasks. Washing a single plate could require a number of instructions:

1. Take the plate out of the sink.
2. Put soap on the brush.
3. Brush the plate.
4. Rinse the plate.
5. Put it in the plate drawer.

As you can imagine, robutles aren't very valuable when it comes to saving time. What if there was a way to make the robutle complete the same task over and over? If they are coded in Java, there is a simply construct called loops that can do that for us. Before we look at loops however, there is an important construct we must learn called "conditional clauses", but everyone calls them if-statements. And if-statement allows a program, or a robtle to perform different tasks depending on whether a condition is true. In the dish washing example, we could use an if statement to either put a dish in the plate drawer (if it is a plate) or in the bowl drawer (if not).

4.2 Conditional Clauses:

Conditional clauses are regularly used in the real world, but they often go unnoticed. Think about the subconscious decision you make sitting at a traffic light: If it is green, then you go. Else, you stay. Or when the Monday morning alarm clock shatters your peaceful somber: if I'm still tired, then I press snooze. Else, I get out of bed.

Java makes conditional clauses extremely simple to use by nearly copying the English language. Check out this example:

```
int four = 4;
if (four > 3) {
    System.out.println("Four is greater than three");
}
else {
    System.out.println("Four is not greater than three");
}
```

This block of code is creating a variable to hold the value 4, and then checking to see if that variable is greater than 3 (Spoiler alert: It is, 4 is a bigger number than 3). If

the condition (four > 3) is true, the code in the "if" block is executed. If it is false, the code in the "else" block is executed.

The condition (part in parenthesis) of an if-statement can be nearly anything, a comparison of two numbers, a boolean value, essentially if something generates a true or false result, it can be in the condition. Below you will find a chart of some of the most common operators in Java. Note that the operators at the top of the list you should be familiar with, and the operators at the bottom should be new and important to conditionals.

	Operator	Name	Description
Arithmetic	+	Addition	Adds two numbers, 8+4 gives 12.
	-	Subtraction	Subtracts the right from the left, 8-4 gives 4.
	*	Multiplication	Multiplies two numbers, 8*4 gives 32.
	/	Division	Divides two numbers, 8/4 gives 2.
	%	Modulus	Divides two numbers, returns the remainder. 9/4 gives 1.
Assignment	=	Assignment	Assigns the value on the right to the variable on the left.
Logical	==	Equality	Returns true if the left and right side are equal, otherwise false.
	!=	Not Equal	Returns true if the left and right side are NOT equal, otherwise true.
	>	Greater Than	Returns true if the left is greater than the right side.
	>=	Greater or Equal	Returns true if the left is greater than or equal to the right side.
	<	Less than	Returns true if the left is less than the right side.
	<=	Less or Equal	Returns true if the left is less than or equal to the right side.
	&&	AND	Returns true if both left AND right side are true
	||	OR	Returns true if either left OR right side is true
	!	NOT	Returns true if operand is false.

You've probably seen all of the arithmetic operators except for Modulus. Although strange and unfamiliar, modulus is a simple operator. When two numbers are divided, and are not perfectly divisible, there is a remainder. It's like if you have boxes that fit 4 oranges, and you need to put 18 oranges in the boxes. You can successfully fill 4 boxes, but then you have 2 oranges left over. Modulus gives 2, the number of oranges left over.

The logical operators should be new in computer terms, but if you flashback to middle school math, the meanings of these should come back quickly. The bottom three, AND, OR, and NOT are the completely unfamiliar operators and can best be demonstrated with short examples.

```
if (5 > 3 && 2 < 7)
if (5 < 3 || 2 < 7)
if (!(5 < 3))
```

These three conditional statements all return true. Let's see how.

The first statement contains two conditional clauses joined by a logical AND. Each conditional clause is first evaluated independently. Is 5 > 3? True. Is 2 < 7? True. Since both statements are true, the compound clause is true.

The second statement also contains two conditional clauses, this time joined by a logical OR. Like above, the clauses are evaluated independently. Notice the other difference between this and the first statement, the first equality operator. Is 5 < 3? False. Is 2 < 7? True. Since at least one of the conditions is true, this entire compound clause is true.

The third statement only contains one conditional clause, but it is modified by a logical NOT operator. First we evaluate the conditional clause, and then reverse it due to the not. Is 5 < 3? False. We then flip this false result, and the entire compound clause returns the result true.

A final point on the if-statement is that it multiple if statements can be combined using the "if else" syntax.

4.3 Loops

Let's step back to the dishwashing robutle for a second. What we really want out of our robutle is to repeat the same task over and over again, washing a dish. This is done in Java through the use of Loops. There are 3 fundamental types of loops, but they all do essentially the same thing. Repeat an action until a predetermined condition is met. First we will look at the "for loop".

```
for (int i = 0; i < 10; i++)
{
    System.out.println(i);
}
```

This looks new, complicated and confusing, but bare with me for a second. Let's look at it one piece at a time. The keyword for indicates that we are using a for loop. Next we have "int i = 0". You probably recognize this from Lesson 3. Next is a conditional clause, "i < 10" and finally an incrementation operator "i++". The operator ++ is often used in for loops and simply indicates that we are increasing the value of i by 1. i++ is the exact same as i = i +1, and either can be used in a for loop. Two lines down from that we have our system output. This simply prints the value of i during each iteration of the loop. Execution of the for loop looks something like this:

1. Declare and initialize i to be the value 0.
2. Check that i < 10 (0 < 10)
3. Print i (0)
4. Increment i
5. Check that i < 10 (1 < 10)
6. …

This pattern continues until the "Check that i < 10" condition fails.

The next construct we will look at is the while loop. The while loop accomplishes the same goal as the for loop, repeating a section of code over and over, but does so with more flexibility. Typically in a for loop, the program knows how many iterations they want the program to perform. The counter and conditional structure works perfectly for this. What if, the programmer has no idea how many operations to perform? Say, for example, the dish washing system. A while loop is perfect here because the task can be repeated while any type of conditional clause is met. Check this out:

```
boolean dirtyDishes = true;
while (dirtyDishes) {
    System.out.println("Cleaning Dishes");

}
```

What's happening here? First we create a boolean variable, dirtyDishes (either true or false) and initialize it as true. This Boolean serves as the condition in our while loop. When dirtyDishes is true, our while loop repeats itself over and over. This example is called an infinite loop. There is nothing to change the dirtyDishes condition, so it will always prove to be true. Somehow, we have to update the condition after we clean a dish. This could be as simple as a check to see if the sink is empty, and if it is, changing the value of dirtyDishes to be false. Something like this:

```
boolean dirtyDishes = true;
boolean sinkEmpty;
while (dirtyDishes)
{
    System.out.println("Cleaning Dishes");

    if (sinkEmpty)
    {
        dirtyDishes = false;
    }
    else
    {

    }
}
```

The final type of loop is called a do-while loop. It is exactly identical to a while loop, but a first iteration is always guaranteed to occur, even if the condition is false.

```
boolean dirtySink = false;
do
{
    System.out.println("Cleaning the sink");
} while (dirtySink);
```

Even though our sink is clean as a whistle, with the do-while loop we can still make our robutle clean the sink once.

Loops are a critical component of computer programming and it is strongly recommended to study this lesson and the homework until you are confident in your understanding of the structures.