Module 1-3

Expressions

Java Statements

- Java statements are like sentences in a natural language.
- In Java, statements end in a semicolon (;) You have seen some of these already:

```
int i = 0;

boolean forReal = true;

double j = 1.84 * 2;
```

A quick note about methods

We will talk about methods in greater detail, but the homework assignment requires a basic understanding of how methods work. For now,

- A method is a subunit that performs a specific task, these are very similar to the functions you learned about in prework.

A quick note about methods

- In Java methods are part of a class.
- Methods can take parameters
- Methods can return values to whoever calls the method.

```
public class Panda {

        public String eatBamboo (int numOfStrips) {

                String result = "Nom nom, I am eating " + numOfStrips + " pieces of bamboo.";
                return result;
        }

}
```

Here we have a method called *eatBamboo* which takes an integer as a parameter, and returns a String

Blocks

- Code that is related (either to conform to the Java language or by choice) is enclosed in a set of curly braces { … } . The contents inside the curly braces is known as a “block.”
- Blocks are commonly used in:
    - Conditional Statements
    - Methods
    - Loops

Blocks: Variable Scope

The following code contains an error on the last line:

```
int k =2;
if (k >= 2) {
	String value = "Potato";
}

System.out.println(value);
```

- The string value is defined within the if-block, hence it is not accessible to the System.out.println() statement.

Let’s work with some methods

Conditional Statements

- A conditional statement allows for the execution of code only if a certain condition is met. The condition **must be, or must evaluate to a boolean value (true or false)**.
- The if statement follows this pattern:

```
if (condition) {
        // do something if condition is true.
}
else {
        // do something if condition is false.
}
```

- The else is optional… but you cannot have an else by itself without an if.

Conditional Statements

Here is an example:

```
boolean isItFall = true;

if (isItFall == true) {
        System.out.println("ok Hibernation time zzzz.");
}
else {
        System.out.println("let's see what the humans are up to!");
}
```

The output of this code is “ok Hibernation time zzzz. Changing isItFall to false would cause the output to be *let’s see what the humans are up to!*

Conditional Statements

Here is an example:

```
boolean isItFall = true;

if (isItFall) {
        System.out.println("ok Hibernation time zzzz.");
}
else {
        System.out.println("let's see what the humans are up to!");
}
```

Since isItFall is a boolean already, typing isItFall == true is redundant.

Likewise, to negate the boolean isItFall, the preferred style is to write !isItFall as opposed to isItFall == false.

Conditional Statements

Here is another example:

```
int season = 4;

if (season == 4) {
        System.out.println("ok Hibernation time zzzz.");
}
```

The output of this code is *ok Hibernation time zzzz*.

Conditional Statements

Here is a tricky example. What do you think the output is?

```
boolean isWinter = false;

if (isWinter = true) {
        System.out.println("ok Hibernation time zzzz.");
}
else {
        System.out.println("I'm starving! Time for breakfast.");
}
```

Here we have the assignment operator used instead of using a comparison operator. This code will always print *ok Hibernation time zzzz*.

Conditional Statements: Numerical Comparisons

The following operators allow you to compare numbers:

- **==** : Are two numbers equal to each other?
- **>** : Is a number greater than another number.
- **<** : Is a number less than another number.
- **>=** : Is a number greater or equal to another number.
- **<=** : Is a number less than or equal to another number.

Conditional Statements: Numerical Comparisons

Here is an example:

```
double myTEGradeAverage = 2.3;

if (myTEGradeAverage >= 2.0) {
        System.out.println("I am in good standing!");
}
else {
        System.out.println("I must work harder!");
}
```

The value of myTEGradeAverage is certainly greater than 2, *I am in good standing* will print.

Conditional Statements : Ternary Operator

The ternary operator can sometimes be used to simplify conditional statements.

- The following format is used:

  ```
  (condition to evaluate) ? //do this if condition is true : //do this if condition is false;
  ```

- You can assign the result of the above statement to a variable if needed.

Conditional Statements : Ternary Operator Example

These 2 blocks of code accomplish the same thing.

Using a ternary:

```
double myNumber = 5;
String divisbleBy2 = (myNumber%2 == 0) ? "Even" : "Odd";
System.out.println(divisbleBy2);
```

Using an if/else:

```
int myNumber = 5;
String divisibleBy2 = "";

if (myNumber%2 == 0 ) {
        divisibleBy2 = "Even";
}
else {
        divisibleBy2 = "False";
}
System.out.println(divisibleBy2);
```

Let’s write some conditionals

A quick note on comparing Strings

When comparing Strings we want to use the .equals() method. Here is an example:

```
String bear = “Panda”;

if ( bear.equals(“Panda”) ) {
        System.out.println("Is it really a bear?");
}
```

The if-block will execute, printing *Is it really a bear?*

AND / OR

- Recall that the condition in an if block needs to somehow be resolved into a true or false value.
- We can use AND / OR statements to craft more complex conditions.
- The AND operator in Java is: **&&**
- The OR operator in Java is **||** (these are pipe symbols, it is typically located under the backspace and requires a shift).

AND / OR: Truth Table

We evaluate AND / OR using truth tables:

- For AND statements:
  - True AND True is True
  - True AND False is False
  - False AND True is False
  - False AND False is False
- For OR statements:
  - True AND True is True
  - True AND False is True
  - False AND True is True
  - False AND False is False

AND / OR: Exclusive OR

———

There is a third case called an “Exclusive Or” or XOR for short. The operator is the carrot symbol (^).

- For XOR statements:
  - True XOR True is False
  - True XOR False is True
  - False XOR True is True
  - False XOR False is False

In most day to day programming, XOR is not used very often.

AND / OR: Exclusive OR

There is a third case called an “Exclusive Or” or XOR for short. The operator is the carrot symbol (^).

- For XOR statements:
    - True XOR True is False
    - True XOR False is True
    - False XOR True is True
    - False XOR False is False

AND / OR: Example1

```
String bearSpecies = “Panda”;

if ( bearSpecies.equals("Grizzly") || bearSpecies.equals("Brown") ) {
        System.out.println("ok hibernation time zzzz.");
}
else {
        System.out.println("Nope, I'm ok.");
}
```

The output of this code is *Nope, I’m ok.*

AND / OR: Example 2

```
- int grade = 70;

if (grade >= 90) {
        System.out.println("A");
}

if (grade >= 80 && grade< 90) {
        System.out.println("B");
}

if (grade >= 70 && grade< 80) {
        System.out.println("C");
}

if (grade >= 60 && grade < 70) {
        System.out.println("D");
}
```

70 is not greater or equal to 90.
The check is false.
Statement won't execute.

70 is not greater or equal to 80 and less than 90.
The check is false.
Statement won't execute.

70 is greater or equal to 70, and less than 80.
The check is true.
Statement will execute.

70 is not greater or equal to 60 and less than 70.
The check is false.
Statement won't execute.

AND / OR: Example 3

```
int myInteger = 2;

if(myInteger==2 && myInteger==3 || myInteger==4 || myInteger%2==0 || myInteger==6) {
        System.out.println("the combined statement is true.");
}
else {
        System.out.println("the combined statement is false.");
}
```

We evaluate what's inside the parentheses from left to right. The output of this is: ***the combined statement is true.***

Let’s practice AND / OR / XORs