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 in:

  ```
  int i = 0;

  boolean forReal = true;

  double j = 1.84 * 2;
  ```

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 used in:
    - Conditional Statements (we will talk about this today)
    - Methods (ditto)
    - Loops

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.
- The parenthesis around the condition if also required.

Conditional Statements

Here is an example:

```
public class Bear {

        public static void main(String[] args) {

                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 == symbol means is equivalence. It is not the same as =, which means assignment.

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:

```
public class Bear {

    public static void main(String[] args) {

        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, this is the preferred style.

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

Conditional Statements

Here is another example:

```
public class Bear {

        public static void main(String[] args) {

                String season = "Winter";

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

season is not a boolean, but it is used as part of an evaluation: Is the season winter?

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

Conditional Statements

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

```
public class Bear {

    public static void main(String[] args) {

        boolean isWinter = false;

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

Conditional Statements: Numerical Comparisons

The following operators allow you to compare numbers:

- **==** : Are 2 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) {
        System.out.println("I am in good standing!");
}
else {
        System.out.println("I must work harder!");
}
```

myTEGradeAverage is certainly greater than 2, this statement is true, block will execute.

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. The data type of this variable would be what the statements on both sides of the colon resolve to.

Conditional Statements : Ternary Operator Example

These 2 blocks of code accomplish the same thing.

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

```
// Using if/else blocks
int myNumber = 5;
String divisibleBy2 = "";

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

AND / OR

- Recall that the condition needs to somehow be resolved into a true or false value, and we can achieve this by using the == operator.
- We can use AND / OR statements to state that code should only be executed if multiple conditions are true.
- 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 statement:
    - True AND True is True
    - True AND False is False
    - False AND True is False
    - False AND False is False
- For OR statement:
    - 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

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

AND / OR: Examples

```
public class Bear {

	public static void main(String[] args) {

		String bearSpecies = "Panda";

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

We will branch into this if the bear species is a Grizzly or Black Bear

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

AND / OR: Examples

```
int gradePercentage = 70;

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

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

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

if (gradePercentage >= 60 && gradePercentage < 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: Examples

```
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.");
}
```

The output of this is “the combined statement is true.”

- We evaluate what’s inside the parentheses from left to right.
- Equality operators (== and !=) take precedence over AND (&&) / OR(||).

Order of Java Operations…. Given what we know

PEMDAS (Arithmetic Rules)
Equality Operators (**==** and **!=**)
AND / OR (&&, ||)

Items at the top of the list take higher priority.