
WHAT I THINK I LOOK LIKE WHILE I'M ASLEEP
WHAT I REALLY LOOK LIKE

Module 1-3

Expressions

Objectives

- Explain what types of things can comprise an expression
- Define what is meant by a statement in a programming language
- Describe the purpose and use of a block in reference to a programming language
- Know what is meant by a boolean expression and how it is used in a program
- Understand what a comparison operator is and how to use it
- Understand what a logical operator is and how to use it
- Understand how () work with boolean expressions and why using them makes code more clear
- Understand the Truth Table and how to figure out AND and OR interactions

Working with Numbers: Type Conversion

ints, doubles and floats can be used together in the same statement, but Java will apply certain rules:

- Mixed mode expressions are automatically promoted to the higher data type (in this case, a double):

```
public class MyClass {

   public static void main(String[] args) {

        int myInt = 4;
        double myDouble = 2.14;

        int firstAttempt = myInt - myDouble; // Won't work, Java will complain!
   }
}
```

The result of myInt and myDouble is promoted to a double, it will no longer fit in firstAttempt, which is a int.

Working with Numbers: Type Conversion

(continued from previous page)...

- We can overcome this problem by doing a cast:

  int secondAttempt = (int) (myInt - myDouble);

- We can also overcome this problem by making the variable secondAttempt a double:

  double secondAttempt = myInt - myDouble;

Working with Numbers: Type Conversion

Remember this problem?

```
public class MyClass {

        public static void main(String[] args) {

                double tempInF = 98.6;
                double tempInC = (tempInF -32.0) * (5/9);
                System.out.println(tempInC);
        }
}
```

Note that instead of 5.0/9.0, it is now 5/9, and when run, the result is 0.0. Using the rules we just discussed, can you figure out why?

Combining Strings

The plus sign can also be used with Strings:

```
public class MyClass {

        public static void main(String[] args) {

                String firstName = “Carl”;
                String lastName = “Jung”;

                String combinedName = lastName + “, ” + firstName;
                System.out.println(combinedName);
        }
}
```

The following code will print ***Jung, Carl***. This process is known as **concatenation**.

Formatting output

Money should have 2 decimal places to the right of the decimal point

- System.out.printf method allows us to use a specifier
- System.out.printf(“%.2f\n”, myDouble); - will print 2 decimal places to the right of the decimal point.
- \n – escape sequence that says hit the enter key
- \t – escape sequence that says tab over 5 spaces
- \a – escape sequence that sounds an alert

Java Statements and Expressions

Java statements are like sentences in a natural language and are made up of expressions.

- In Java, statements end in a semicolon (;)

  You have statements already in:

```
System.out.println(“Hello World”);
int x = 5 + 1;
```

Java Expressions are constructs that evaluate to a single value. Expressions are made up of ONLY identifiers, literals, and operators.

Java Expressions

```
int i = 0
boolean forReal = true
double j = 1.84 * 2
System.out.println(“The value of j is “ + j)
```

Java Expressions are constructs made up of variables, operators, and method invocations, which follow the rules of the language and evaluate to a single value.

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.”

```
if (notDone) {
    // do something
}
```

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

Methods

- A named block of code.
  - Can take multiple values (parameters)
  - Returns a single value
- Similar to mathematical function.
  - f(n) = n ^ 2
  - Output is often directly related to input

Methods

- Method Signature
  - Descriptive Names
  - Return type (such as int, double, long, String, void, etc)
  - Input parameters
    - Parameters are variables that only live in the method.

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 is 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) {

                int season = 1;

                if (season == 1) {
                        System.out.println("It's winter!\nok Hibernation time zzzz.");
                }
        }
}
```

season is not a boolean, but it is used as part of an evaluation: Is the season equal to 1?

The output of this code is "It's winter!

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.

```
color = (date == 28) ? “blue” : “red”;

if (date == 28) {
    color = “blue”;
}
else {
    color = “red”;
}
```

Conditional Statements : Ternary Operator Example

These 2 blocks of code accomplish the same thing.

```
// Using Ternary Operator:
double myNumber = 5;
String divisibleBy2 = (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 = “Odd";
}
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: Exclusive OR

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

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

Truth Tables

A	B	!A	A && B	A || B	A ^ B
TRUE	TRUE	FALSE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	FALSE	TRUE	TRUE
FALSE	TRUE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	TRUE	FALSE	FALSE	FALSE

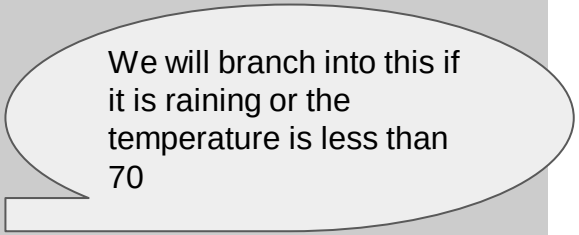
AND / OR: Examples

```
public class Weather {

        public static void main(String[] args) {

           boolean isRaining = false;
           int tempInF = 70;

           if (isRaining == true || tempInF < 70) {
              System.out.println("Wear a coat!");
           }
           else {
              System.out.println("No coat needed!");
           }
        }
}
```



The output of this code is “No coat needed!”

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 but it is less than 90.
The check is false because 1st part 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 greater or equal to 60 and but not less than 70.
The check is false because 2nd part 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(||).
- Use parentheses to make your expression clear

Order of Java Operations…. Given what we know

Precedence	Operator	Type	Associativity
12	() [] .	Parentheses Array subscript Member selection	Left to Right
10	++ -- + - ! (*type*)	Unary increment Unary decrement Unary plus Unary minus Unary logical negation Unary type cast	Right to left
9	* / %	Multiplication Division Modulus	Left to right
8	+ -	Addition Subtraction	Left to right
7	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
6	== !=	Relational is equal to Relational is not equal to	Left to right
5	^	Exclusive OR	Left to right
4	&&	Logical AND	Left to right
3	||	Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= %=	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

Larger number means higher precedence.

Objectives

- Should be able to explain what types of things can comprise an expression

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

```
int gradePercentage = 70;
```

```
If (gradePercentage == 70)
```

Objectives

- Should be able to explain what types of things can comprise an expression
- Should be able to define what is meant by a statement in a programming language

```
int area = length * height;
```

Objectives

- Should be able to explain what types of things can comprise an expression
- Should be able to define what is meant by a statement in a programming language
- Should be able to describe the purpose and use of a block in reference to a programming language

```
if (isCloudy == true) {
   System.out.println(“Bring your umbrella!”);
}
```

```
if (isCloudy) {
   System.out.println(“Bring your umbrella!”);
}
```

Objectives

- Should be able to explain what types of things can comprise an expression
- Should be able to define what is meant by a statement in a programming language
- Should be able to describe the purpose and use of a block in reference to a programming language
- Should know what is meant by a boolean expression and how it is used in a program

```
if (isCloudy == true) {
   System.out.println(“Bring your umbrella!”);
}
```

```
if (isCloudy) {
   System.out.println(“Bring your umbrella!”);
}
```

Objectives

- Should be able to explain what types of things can comprise an expression
- Should be able to define what is meant by a statement in a programming language
- Should be able to describe the purpose and use of a block in reference to a programming language
- Should know what is meant by a boolean expression and how it is used in a program
- Should understand what a comparison operator is and how to use it

```
if (grade = 90) {
   System.out.println(“Wrong operator!!”);
}
```

```
if (grade == 90) {
   System.out.println(“Right operator!!”);
}
```

Objectives

- Should be able to explain what types of things can comprise an expression
- Should be able to define what is meant by a statement in a programming language
- Should be able to describe the purpose and use of a block in reference to a programming language
- Should know what is meant by a boolean expression and how it is used in a program
- Should understand what a comparison operator is and how to use it
- Should understand what a logical operator is and how to use it

```
if (grade >= 80 &&  grade < 90) {
   System.out.println(“This is a B”);
}
```

Objectives

- Should be able to explain what types of things can comprise an expression
- Should be able to define what is meant by a statement in a programming language
- Should be able to describe the purpose and use of a block in reference to a programming language
- Should know what is meant by a boolean expression and how it is used in a program
- Should understand what a comparison operator is and how to use it
- Should understand what a logical operator is and how to use it
- Should understand how () work with boolean expressions and why using them makes code more clear

```
if (num >= 80  && isPassFail == false || isPassFail == true) {
   System.out.println(“You passed the class”);
}
```

```
if ((num >= 80  && isPassFail == false) || isPassFail == true) {
   System.out.println(“You passed the class”);
}
```

Objectives

- Should be able to explain what types of things can comprise an expression
- Should be able to define what is meant by a statement in a programming language
- Should be able to describe the purpose and use of a block in reference to a programming language
- Should know what is meant by a boolean expression and how it is used in a program
- Should understand what a comparison operator is and how to use it
- Should understand what a logical operator is and how to use it
- Should understand how () work with boolean expressions and why using them makes code more clear
- Should understand the Truth Table and how to figure out AND and OR interactions

```
if (num < 0 && num > 10){
   System.out.println("This will never happen!");
}
```

```
if (num < 0 || num > 10){
   System.out.println("Number is out of range");
}
```