

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.

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.

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
 - 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:

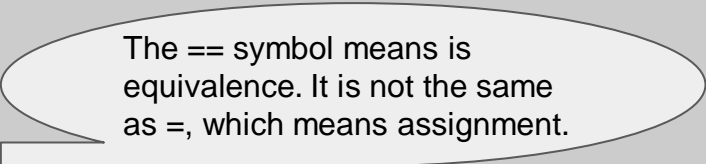
```
if (condition) {  
    // do something if condition is true.  
}  
else {  
    // do something if condition is false.  
}
```

- 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!");  
        }  
    }  
}
```



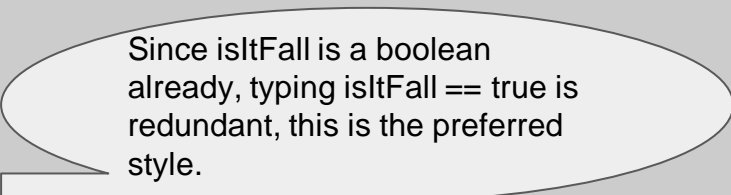
The == symbol means is equivalence. It is not the same as =, which means assignment.

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!");  
        }  
    }  
}
```



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.");  
        }  
    }  
}
```



IntelliJ will give a compiler error!

Conditional Statements: Numerical Comparisons

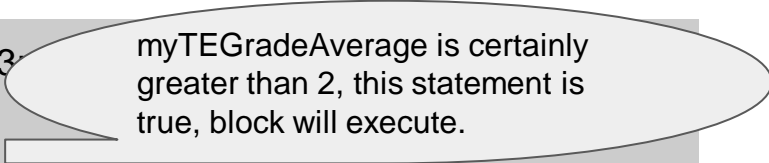
The following operators allow you to compare numbers:

- **==** : 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;
```



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

```
if(myTEGradeAverage >= 2) {  
    System.out.println("I am in good standing!");  
}  
else {  
    System.out.println("I must work harder!");  
}
```

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(divisibleBy2);
```

```
// 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 (\wedge).

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

Truth Tables

A	B	A && B	A B
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
FALSE	TRUE	FALSE	TRUE
FALSE	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!");  
        }  
    }  
}
```

We will branch into this if it is raining or the temperature is less than 70

The output of this code is "No coat needed!"

AND / OR: Examples

```
int gradePercentage = 70;
```

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

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

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

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.

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

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

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

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.

Second Deck: Logical Branching

Module 1: 03

Today's Objectives

1. Expression, Statements, and Blocks
2. Introduction to Methods
3. Boolean Expressions
 - a. Comparison Operators
 - b. Logical Operators
4. Conditional Statements

Expressions

In programming, an expression can be made up of variables, operators, or method invocations constructed according to the syntax of the language, that **evaluates to a single value**.

Examples: $x - y$

$a + 9$

$a + b - c * 8$

Statements

Statements in programming form a unit of execution.

Statements in Java are terminated by a semi-colon (;), and may contain expressions. Many expressions can be made into statements by terminating them with a semicolon.

Examples: `int x;`

`x = 5 - 2;`

`double area = length * height;`

Blocks

A **block** is a group of statements that needs to be executed as a single unit.

In Java, **blocks** are identified by curly-braces { }

Example: {

```
    int currentFloor = 1;  
    currentFloor = currentFloor + 1;  
    System.out.println(currentFloor);  
}
```

Introduction to Methods

A **method** creates a *reusable block* of code.

A method takes input and returns output that is often directly correlated to the input, similar to mathematical function: $f(n) = n^2$

Methods are defined by a **Method Signature** that identifies

1. Who can use it (accessor)
2. What to call it (name)
3. What it will return (return type)
4. What input it takes (arguments)

Method Signature

accessor return_type name (parameters)

accessor	A keyword, like <i>public</i> , that identifies who can use the method
return type	A <i>Data Type</i> , (int, double, String), that identifies what type of data the method will return - the output
name	A descriptive name that can be used to call the method, causing the code in it's code block to execute
parameters	A list of 0...n variables, contained in (), that must be populated when calling the method. - the input

Example: public int addNumbers(int x, int y)



The method signature is one of the most important fundamentals of the Java Language, and you will be expected to know the parts in many interviews.
Memorize these parts!

Parts of a Method

```
accessor return name( parameters ) {  
    code...  
    return statement;  
}
```

Method Signature	defines the method
Code Block	Code to execute when the method is used. Defined by { } following the method signature.
Return statement	Statement of code that tells the method what value to return. <i>This value must be the same data type as a methods return type.</i>

```
Example: public int addNumbers(int x, int y) {  
    int sum = x + y;  
    return sum;  
}
```

Boolean Expressions

A **Boolean Expression** is an *expression* that *evaluates* to a single boolean value (*true* or *false*).

Boolean Expressions are commonly used to conditionally decide what code to execute.

The most simple boolean expressions are the true/false keywords or a boolean variable. Examples:

true

```
boolean isRed = false;
```

isRed

Why are boolean expressions important?

All programs are built on **State** and **Behavior**.

Boolean expressions allows the programmer to ask questions about the current condition (**state**) of the program and make decisions on what code should be executed based on that state.

State: The particular condition or value that variables and other constructs in our code have at a specific time.

Same as the usage in natural language. For example:

What is the **state** of the company's finances?

Meaning: What is the *current condition* of the company's finances?

Comparison Operators

Used to compare two values. Similar to familiar mathematical comparison operators.

Operators	Meaning
<code>==</code>	Equals to
<code>!=</code>	Not Equal to
<code>></code>	Greater Than
<code><</code>	Less Than
<code>>=</code>	Greater Than or Equal To
<code><=</code>	Less Than or Equal To

Comparison operators can only be used with `int`, `long`, `short`, `byte`, `boolean`, `double`, and `float` data types.

Comparison operators cannot be used with `String`.

Logical Operators

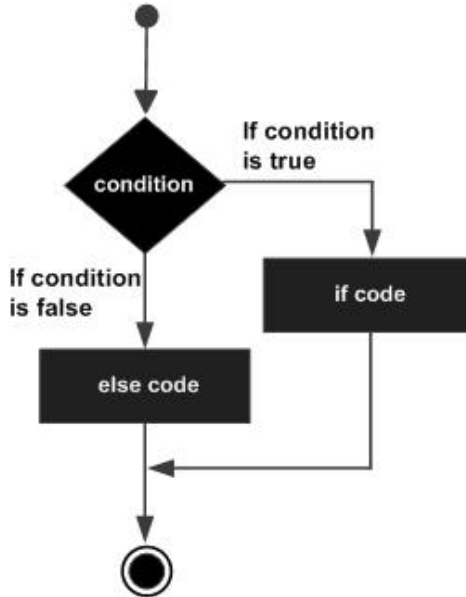
A Logical Operator combines two or more boolean expressions, such that they evaluate to a single value. In natural language these are represented by AND, OR and NOT

Operators	Meaning
&&	AND (Both conditions must be true)
	OR (At least one condition must be true)
^	XOR (Both conditions cannot be the same)
!	NOT (switches the result)

Similar to Arithmetic operators, Logical Operators can have the precedence set by using parentheses.

Conditional Statements and Blocks

An *conditional* statement uses a *Boolean Expression* to determine whether or not a block of code should run.



if statement

Natural language

*On the way home, if we out of bread,
then go to the store.*

Code

```
if (isOutOfBread == true) {  
    goToStore();  
}  
  
goHome();
```



if Syntax

```
if ( boolean condition ) {  
    code to run when true  
}
```

The IF Statement and block of code to run when true are required.

There must be 1 and only 1 if statement and block.

```
if (x % 2 == 0) {  
    System.out.println("even");  
}
```

IF... ELSE Syntax

The block following the if condition is run when the if condition is true. An else statement and block can be added to set code that will be run when the condition is false.

```
if ( boolean condition ) {  
    code to run when true  
} else {  
    code to run when false  
}
```

An **else** is optional, but if it is added, there can only be 1.

```
if (x % 2 == 0) {  
    System.out.println("even");  
} else {  
    System.out.println("not even");  
}
```

[Visual Explanation](#)

Ternary Operator

If an if... else condition is determining which of 2 values to set, then it can *optionally* be shortened using the ternary operator. (ternary – use three as a base)

`variable = boolean condition ? true result : false result;`

Original if statement:

```
String message = "";

if (studentsInClass > 10) {
    message = "Enough Students";
} else {
    message = "Not Enough
students";
}
```

Equivalent statement with Ternary Operator:

```
String message = studentsInClass > 10 ? "Enough Students" : "Not Enough Students";
```

IF...ELSE IF

Multiple boolean conditions can be chained together using ELSE IF. The IF and ELSE IF statements are mutually exclusive, so only the code block for the *first* true condition will be executed, so the ORDER OF THE ELSE IF statements matter.

```
if ( x < 10 ) {  
    x = x + 1;  
} else if ( x < 20 ) {  
    x = x + 2;  
} else if ( x < 30 ) {  
    x = x + 3;  
} else {  
    x = x - 1;  
}
```

- 1 if () is required
- else if () is optional. There can be as many as needed. So there can be 0...n else if () statements.
- else is optional. But if used there can be only 1. So there can be 0...1 else blocks.

[Visual Explanation](#)