

EXPRESSIONS

TODAY'S OBJECTIVES

- Things that can make up an Expression.
- Statements in a programming language.
- Blocks: what are they and how they are used.
- Boolean Expressions: what are they and how they are used.
- Comparison Operators: what are they and how they are used.
- Logical Operators: what are they and how they are used.
- Casting/data conversion: when it occurs, and why it's used.
- Grouping things in () in Boolean Expressions and why we'd want to.
- Truth Table: How to use it to figure out AND and OR interactions.

JAVA EXPRESSIONS

- Code is made up of expressions and statements.
- Expressions evaluate to a single value.
- Computers evaluate each expression separately.
- We use balanced parentheses to control order of evaluation or to make the order unambiguous.
 - $x + y / 100$ // ambiguous
 - $(x + y) / 100$ // unambiguous, recommended

BOOLEAN EXPRESSIONS

- A **boolean expression** is an expression that evaluates to a boolean value (true or false).
- Used to conditionally execute blocks of code

JAVA STATEMENTS

- Statements are roughly equivalent to sentences in natural languages.
- A statement forms a complete unit of execution.
- Some expressions can be made into a statement by terminating the expression with a semicolon (;).
 - Assignment expressions
 - `aValue = 8933.234;`
 - Any use of `++` or `--`
 - `aValue++;` // (more on this shortly)
 - Method invocations
 - `System.out.println("Hello World!");`
- Declaration Statements
 - `double aValue;`
- Control flow statements
 - (... more on these later this week)

EXAMPLES

Variable declaration:

```
int numOfDay;
```

EXAMPLES

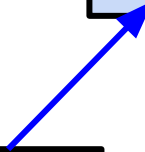
Variable declaration:

```
int numOfDay;
```

Assignment Expressions:

```
int x = 5 + 1;
```

Declaration
Expression



EXAMPLES

Variable declaration:

```
int numOfDay;
```

Assignment Expressions:

```
int x = 5 + 1;
```

Declaration
Expression

Assignment
Expression

EXAMPLES

Variable declaration:

```
int numOfDay;
```

Assignment Expressions:

```
int x = 5 + 1;
```

Declaration
Expression

Assignment
Expression

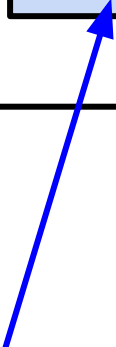
Addition
Expression

EXAMPLES

More complex statements:

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

Declaration
Expression



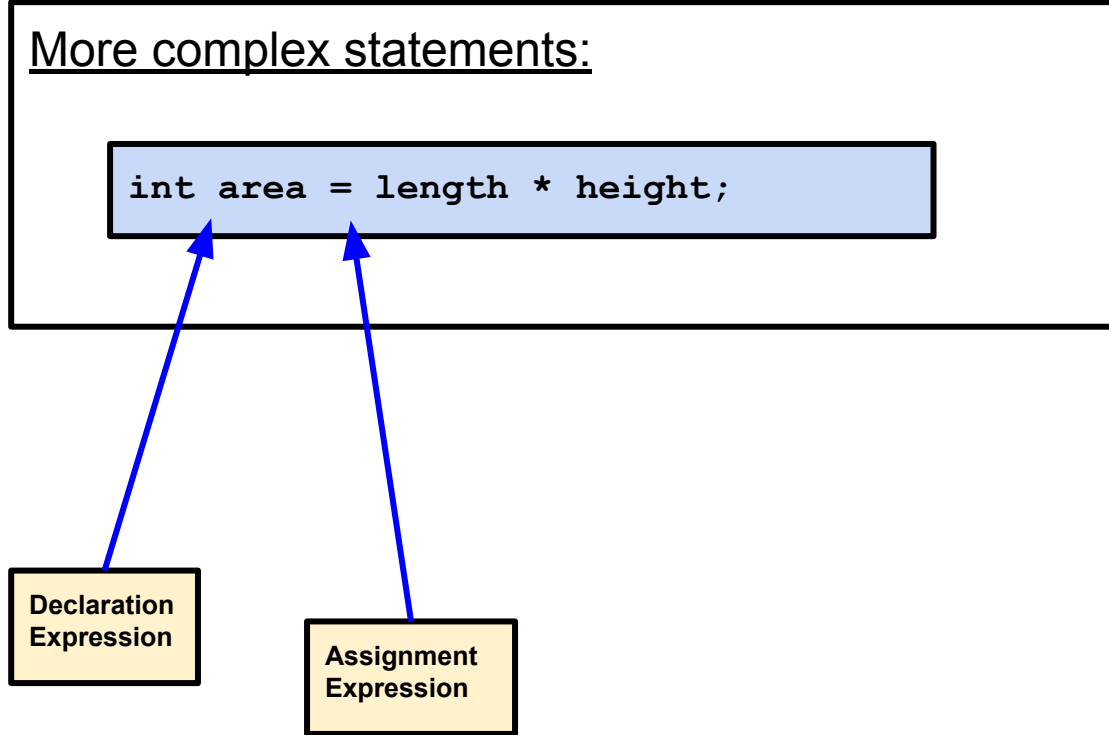
EXAMPLES

More complex statements:

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

Declaration
Expression

Assignment
Expression



EXAMPLES

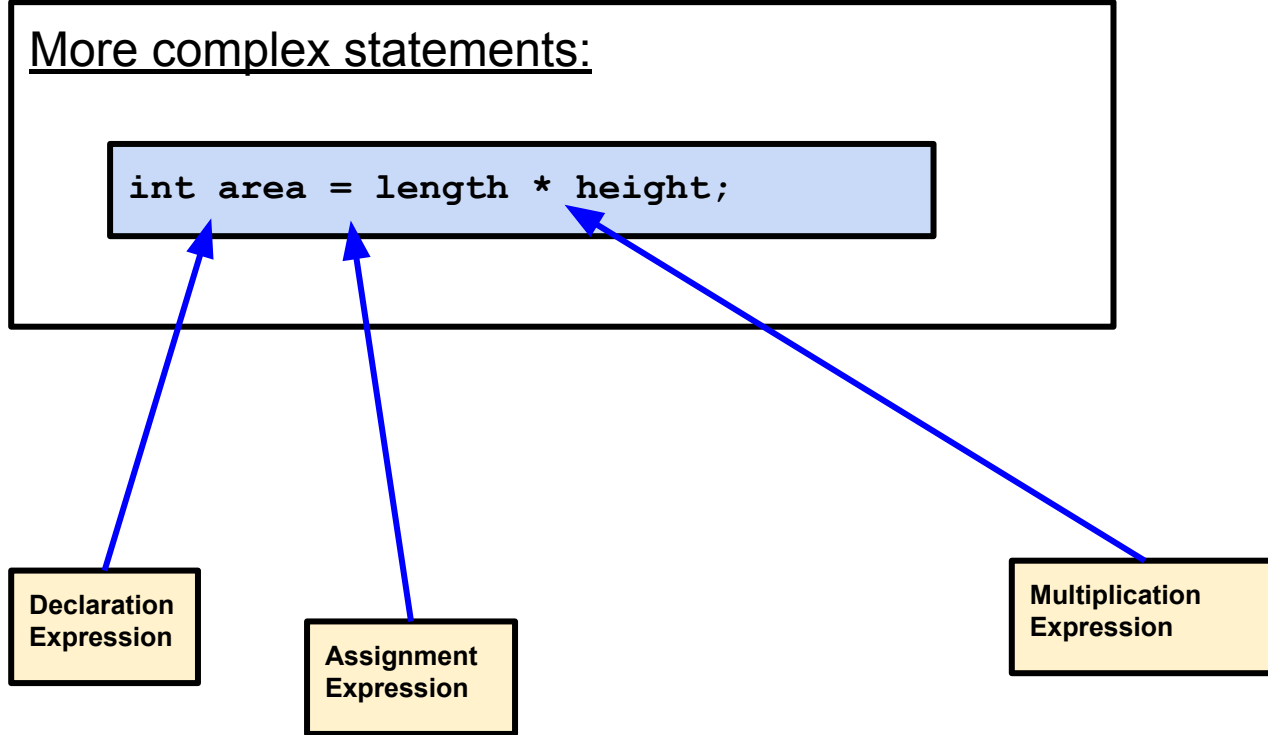
More complex statements:

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

Declaration
Expression

Assignment
Expression

Multiplication
Expression



EXAMPLES

More complex statements:

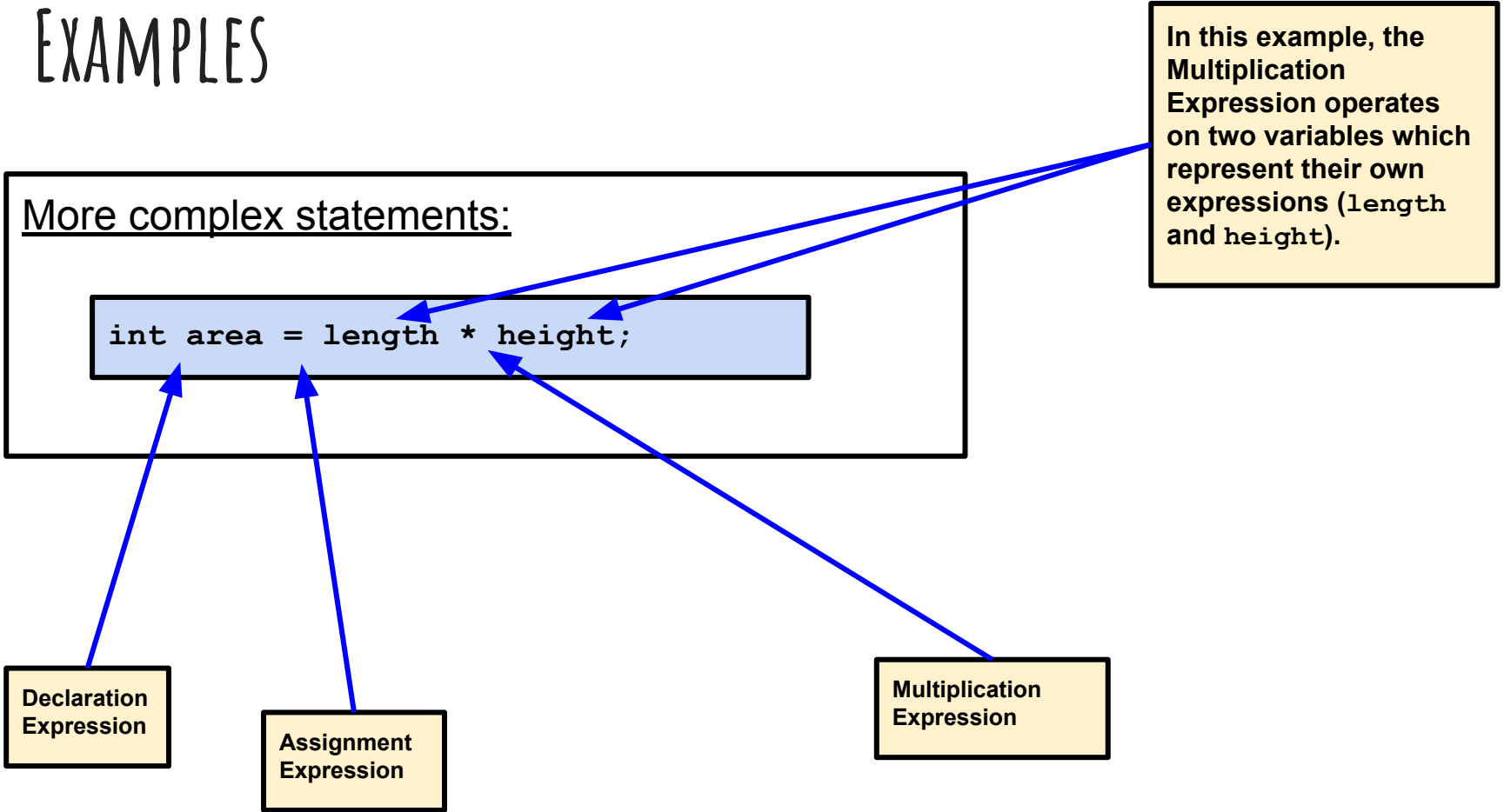
```
int area = length * height;
```

Declaration
Expression

Assignment
Expression

Multiplication
Expression

In this example, the
Multiplication
Expression operates
on two variables which
represent their own
expressions (length
and height).



INCREMENT/DECREMENT EXPRESSIONS

Increment/Decrement Expressions

```
int numOfDay = 5;
```

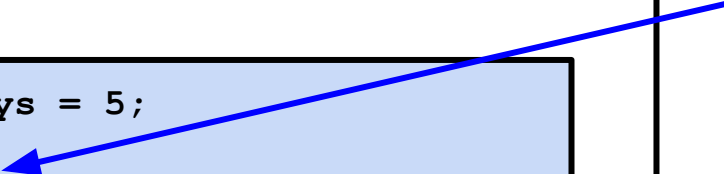
```
numOfDay++;
```

```
numOfDay--;
```

INCREMENT/DECREMENT EXPRESSIONS

Increment/Decrement Expressions

```
int numOfDay = 5;  
numOfDay++; // numOfDay will be 6  
numOfDay--;
```



This adds 1 to `numOfDay`.

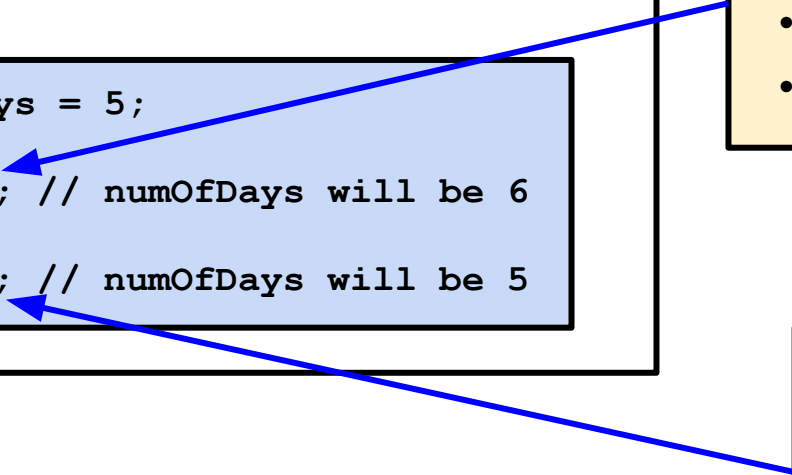
This is the same as:

- `numOfDay = numOfDay + 1;`
- `numOfDay += 1;`

INCREMENT/DECREMENT EXPRESSIONS

Increment/Decrement Expressions

```
int numOfDay = 5;  
numOfDay++; // numOfDay will be 6  
numOfDay--; // numOfDay will be 5
```



This adds 1 to `numOfDay`.

This is the same as:

- `numOfDay = numOfDay + 1;`
- `numOfDay += 1;`

This subtracts 1 to `numOfDay`.

This is the same as:

- `numOfDay = numOfDay - 1;`
- `numOfDay -= 1;`

BLOCKS

- Code that is related (either to conform to the Java language standard or by choice) is enclosed in a set of curly braces ({ ... }). The contents inside the curly braces is known as a “**block**.”
- A block is a group of zero or more statements between balanced braces and can be used anywhere a single statement is allowed.
- Blocks are used in:
 - Conditional Statements (we will talk about this today)
 - Methods (ditto)
 - Loops
- Blocks allow multiple statements where normally only one is allowed.

METHODS

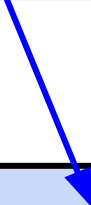
Methods can accept input parameters but are not required to.

```
public void printHello() {  
    System.out.println("Hello, World!")  
}
```

METHODS

Methods can accept input parameters but are not required to.

No input parameters



```
public void printHello() {  
    System.out.println("Hello, World!")  
}
```

METHODS

Methods can accept input parameters but are not required to.

No input parameters

```
public void printHello() {  
    System.out.println("Hello, World!")  
}
```

```
public int addNums(int num1, int num2) {  
    return num1 + num2;  
}
```

Two input parameters

METHODS

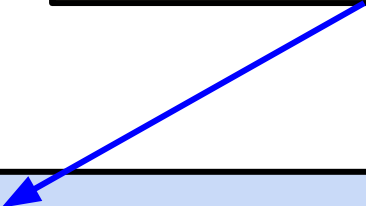
Methods can return a value but are not required to.

```
public void printHello() {  
    System.out.println("Hello, World!")  
}
```

METHODS

Methods can return a value but are not required to.

No return value (void return type specified).



```
public void printHello() {  
    System.out.println("Hello, World!")  
}
```

METHODS

Methods can return a value but are not required to.

No return value (void return type specified).

```
public void printHello() {  
    System.out.println("Hello, World!")  
}
```

```
public int addNums(int num1, int num2) {  
    return num1 + num2;  
}
```

Returns an int value.

METHOD SIGNATURE

Methods have a signature, which is made up of:

- Name (should be descriptive)

Method name

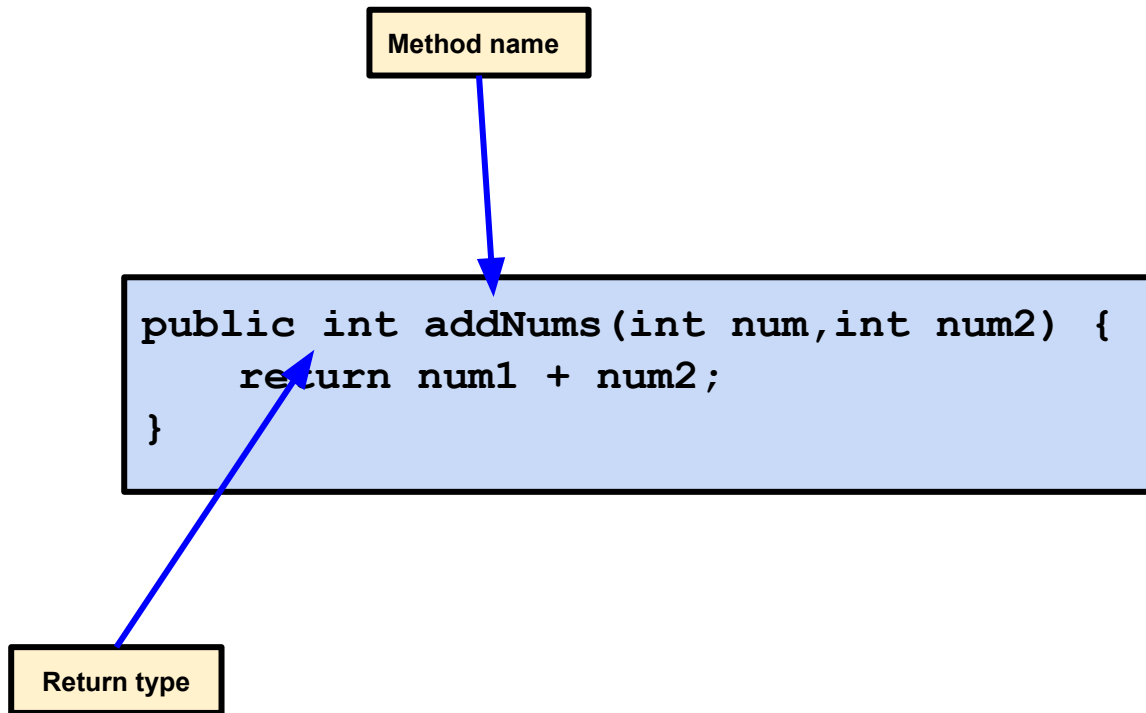


```
public int addNums(int num,int num2) {  
    return num1 + num2;  
}
```


METHOD SIGNATURE

Methods have a signature, which is made up of:

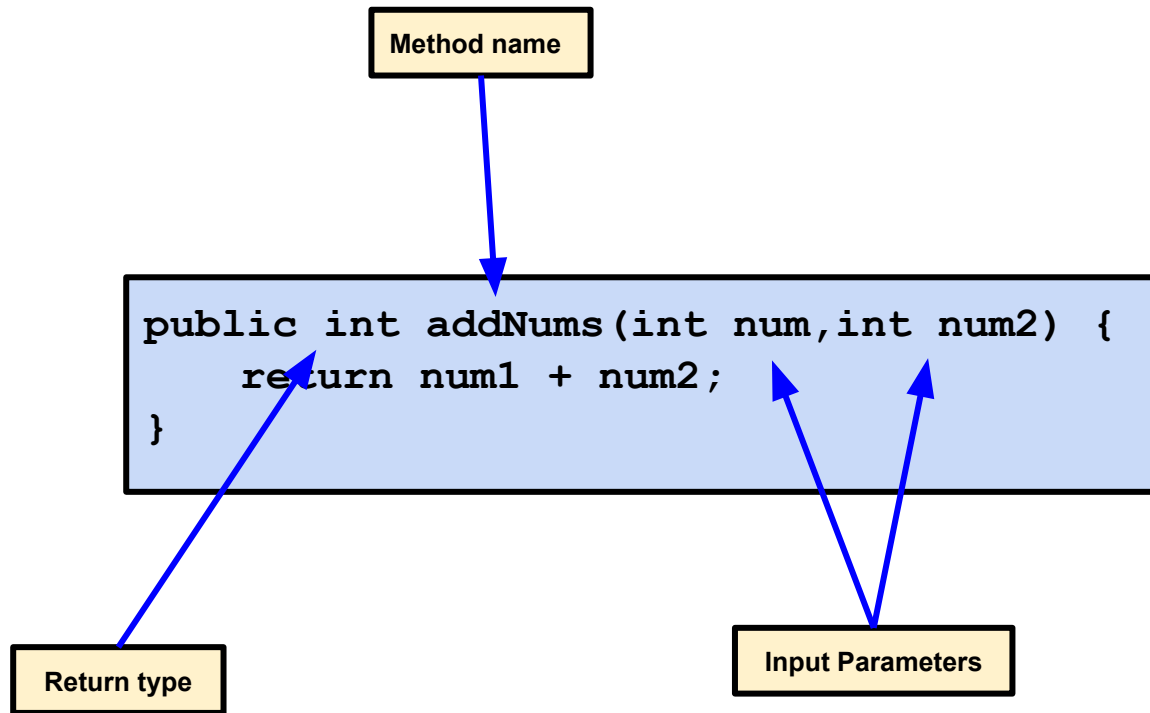
- Name (should be descriptive)
- Return Type (e.g. int, long, double, float, boolean, ...)



METHOD SIGNATURE

Methods have a signature, which is made up of:

- Name (should be descriptive)
- Return Type (e.g. int, long, double, float, boolean, ...)
- Input Parameters



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.  
}
```

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:

The parentheses around the condition are required.

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

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 parentheses around the condition are required.

The `else` block is optional, but you cannot have an `else` block by itself without an `if` block.

CONDITIONAL STATEMENTS: EXAMPLE 1

```
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 expression between the parentheses will be evaluated. If it evaluates to true, the code within the block will be executed.

CONDITIONAL STATEMENTS: EXAMPLE 1

```
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 expression between the parentheses will be evaluated. If it evaluates to true, the code within the block will be executed.

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

CONDITIONAL STATEMENTS: EXAMPLE 1

```
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 expression between the parentheses will be evaluated. If it evaluates to true, the code within the block will be executed.

If expression in the `if` block evaluates to false, the code within the `else` block will be executed.

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

CONDITIONAL STATEMENTS: EXAMPLE 1

```
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 expression between the parentheses will be evaluated. If it evaluates to true, the code within the block will be executed.

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!"**

If expression in the **if** block evaluates to false, the code within the **else** block will be executed.

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

CONDITIONAL STATEMENTS: EXAMPLE 1 REDUX

```
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.

CONDITIONAL STATEMENTS: EXAMPLE 1 REDUX

```
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: EXAMPLE 1 REDUX

```
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`.

The output of this code will still be **"ok Hibernation time zzzz."**
Changing the condition to `!isItFall` would cause the output to be **"let's see what the humans are up to!"**

CONDITIONAL STATEMENTS: EXAMPLE 2

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.");  
        }  
  
    }  
  
}
```

COMPARISON OPERATORS

The following operators allow you to compare numbers:

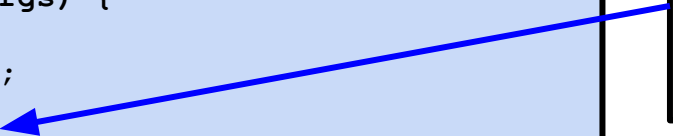
- `==` : Are 2 numbers equal to each other?
- `>` : Is a number greater than another number?
- `<` : Is a number less than another number?
- `>=` : Is a number greater or equal to another number?
- `<=` : Is a number less than or equal to another number?

COMPARISON OPERATORS: EXAMPLE

Here is an example of using a comparison operator in an **if** statement:

```
public class Bear {  
    public static void main(String[] args) {  
        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.



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.

TERNARY OPERATOR: EXAMPLE

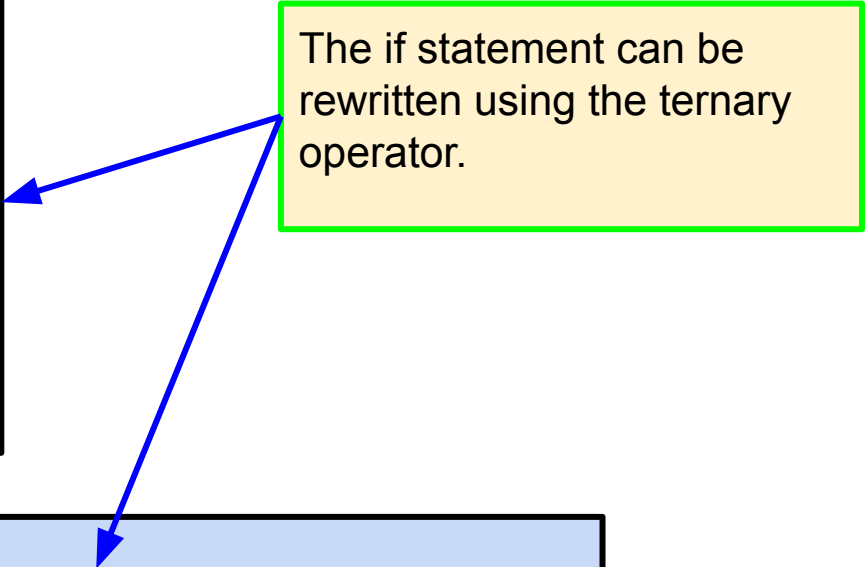
```
int myNumber = 5;

String divisibleBy2 = "";

if (myNumber % 2 == 0 ) {
    divisibleBy2 = "Even";
}
else {
    divisibleBy2 = "False";
}

System.out.println(divisibleBy2);
```

The if statement can be rewritten using the ternary operator.



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

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

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: EXCLUSIVE OR

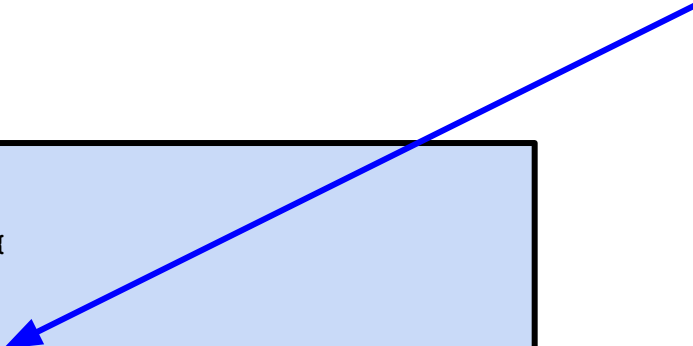
There is a third case called an “Exclusive Or” or XOR for short. The operator is the carrot symbol (\wedge).

- 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: EXAMPLE 1

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



```
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.");  
        }  
    }  
}
```

The output of this code is "Nope, I'm ok."

AND / OR: EXAMPLE 2

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

```
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");
}
```

AND / OR: EXAMPLE 2

```
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.

AND / OR: EXAMPLE 2

```
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.

AND / OR: EXAMPLE 2

```
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 90.
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.");
}
```

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.

Operator reference:

[Summary of Operators \(The Java™ Tutorials > Learning the Java Language > Language Basics\) \(oracle.com\)](#)

Operator precedence reference:

[Operators \(The Java™ Tutorials > Learning the Java Language > Language Basics\) \(oracle.com\)](#)