Fundamentals 3

Recap Session

USE RANDOMISER TO ANSWER THESE QUESTIONS:

> - Can you name me 3 primitive data types?
> - Can you name me a non primitive data type?
> - What are the different types of variables you can create and how are they different?

Statements

A statement in Java is similar to how you would write a sentence in English, with a full stop.

In Java a statement will usually end in a semi-colon.

A statement forms a complete unit of execution, meaning that that one piece of code can be executed to give an outcome.

Let's go through some examples of statements:

```
// assignment statement
numberOfTeams = 4;

// method invocation statement
System.out.println("Hello There");

// Object creation statement:
Car ferrari = new Car();
```

There is also another form of statement, a declarative statement:

```
double interestRate = 0.5; // here we are declaring something
```

In general in Java, statements can be broken down into three areas:

- Declaration statements
- Expression statements
- Control flow statements

Declaration Statements

We have seen an example of this just a moment ago:

```
double interestRate = 0.5;
int num;
String name;
```

The others we will cover throughout and I'll mention where they are seen.

Expressions

An expression is a construct made up of variables, operators and method invocations/calls which evaluate to a single value.

You've seen several examples of these already. Let's run through some more:

```
int numberOfGears = 4;

int years = 5;
double interestRate = 0.4;
double amountOfInterestAccrued = years * interestRate;

System.out.println("Interest accrued over " + years + " years " + " is " +
amountOfInterestAccrued);
```

Some more examples:

```
3/2
5 % 3
pi + (10 * 2)

int secondsInDay = 0;
int daysInWeek = 7;
int hoursInDay = 24;
int minutesInHour = 60;
int secondsInMinute = 60;
boolean calculateWeek = true;

secondsInDay = secondsInMinute * minutesInHour * hoursInDay; //7
System.out.println("The number of seconds in a day is: " + secondsInDay);

if (calculateWeek == true)
{
  System.out.println("The number of seconds in a week is: " + secondsInDay
* daysInWeek);
}
```

```
double totalSalary = 45000 * 1.2; // <-- expression is everything except
the data type
```

```
// variables, values and operators make up an expression

// the whole line becomes a statement (including the data type)
int classSize = 10; // with a literal, this is a statement

if (classSize > 20) // <-- expression is within brackets
  System.out.println("Class size is too large")
```

Open question, show on board:

```
// In this code, where are the expressions
int score = 2000;

if (score > 1000){
  System.out.println("High score");
  score = 0;
}
```

Best practice for expressions:

```
x + y / 100;  // <== this is ambiguous as the left hand side as it's
unclear what should be calculated first
```

```
(x + y) / 100; // here the parenthesis is calculated first. this is the
recommended approach.

// These are examples of compound expressions as we are using more than
one item to determine an outcome.
```

Operators

Create new project and call it operators

There will be times where you'll come across **Operators**.

By Operators I don't mean mobile operators.

When using operators, you use one or many **operands** in so to speak and for demonstration purposes I'll add some logic to show you all of the ones you'll use:

Addition

```
// Explain this logic:
int result = 1 + 2;

System.out.println("1 + 2 = " + result);
```

Subtraction

```
int previousResult = result;

result = result - 1;
System.out.println(previousResult + " - 1 = " + result);
```

Multiplication

```
previousResult = result;

result = result * 10;

System.out.println(previousResult + " * 10 = " + result);
```

Division

```
previousResult = result;

result = result / 2;

System.out.println(previousResult + " / 5 = " + result);
```

Remainder or Modulus

```
// So this is saying return the remainder when we divide the number by 3:

previousResult = result;

result = result % 3;

System.out.println(previousResult + " % 3 = " + result);
```

Increment Numbers

```
result = result + 1; // this adds one to our result
System.out.println("Result is now ==> " + result);

result++;
System.out.println("Result is now ==> " + result);

result++;
System.out.println("Result is now ==> " + result);
```

Decrementing Numbers

```
result = result - 1; // this adds one to our result
System.out.println("Result is now ==> " + result);

result--;
System.out.println("Result is now ==> " + result);

result--;
System.out.println("Result is now ==> " + result);
```

Abbreviation Form

We can also abbreviate our operator usages:

```
int anotherResult = 1;
System.out.println("Another Result is now ==> " + anotherResult);

anotherResult += 3;
System.out.println("Another Result is now ==> " + anotherResult);
```

```
anotherResult -= 2;
System.out.println("Another Result is now ==> " + anotherResult);
```

```
anotherResult *= 2;
System.out.println("Another Result is now ==> " + anotherResult);
```

```
anotherResult /= 2;
System.out.println("Another Result is now ==> " + anotherResult);
```

Now we've covered common operators but we'll look at other operators but we'll need to use an if-then statement to demonstrate these operators.

```
boolean isRainingToday = false;

if (isRainingToday == false)
  System.out.println("You won't need a jacket today");


// so here you have used two operators, can you spot them? Explain equals
and and double equals (testing if two values are the same in value. If
they are variables then we are comparing the values the variables are
assigned)

int number1 = 10;
int number2= 20;

if (number1 == number2)
  System.out.println("Numbers are equal");
```

We can also use the logic where we do not expect variables/values to be the same in order to run some logic.

Not Operator

```
int topScore = 20;
```

```
if (topScore != 20)
  System.out.println("You didn't get the high score!");
```

Greater Than Operator

```
if (topScore > 20)
  System.out.println("You got the high score!");
```

Greater Than and Equal To

```
if (topScore >= 20)
  System.out.println("You got the high score!");
```

Less Than

```
topScore = 10;

if (topScore < 20)
  System.out.println("You got the high score!");
```

Less Than or Equal To

```
if (topScore <= 20)
  System.out.println("You got the high score!");
```

Lets add some extra logic:

AND Operator

Both of conditions need to be true on the left hand side and right hand side.

```
int degreeScore = 60;
int overallYearScore = 100;

if ( (overallYearScore > degreeScore) && (overallYearScore < 100) )
  System.out.println("Greater than overall score and less than 100");
```

OR Operator

Only one of the conditions need to be true

```
if ( (overallYearScore > 90) || (degreeScore <= 90) )
  System.out.println("Degree score outcome is true");

degreeScore = 95;
overallYearScore = 50;

  // change degree score to 100

if ( (overallYearScore > 90) || (degreeScore <= 90) )
  System.out.println("Degree score outcome is true"); // <<--- should not
appear
```

Some Gothcas

```
int someValue = 1;

if (someValue = 1)
  System.out.println("I've errored!!!");
```

```
boolean isAnimal = false;

if (isAnimal = true){
  System.out.println("This is not meant to happen");
}
```

When you assign a value to a variable then it get's given the new value, so here isAnimal has been assigned to **true**.

However as an expression, someValue wouldn't work as the compiler pre-empts that we can't assign a boolean to an int.

To improve the last example, you could do:

```
if (isAnimal){
  System.out.println("This is not meant to happen");
}
```

Ternary Operator

We've looked at all the operators but there is one more, called the Ternary operator.

This is a way to set a value based on two conditions.

For example:

```
boolean wasAnimal = isAnimal ? true : false; // think of this as short cut
for an if and else

if (wasAnimal)
  System.out.println("wasAnimal is true");

isAnimal = true;
wasAnimal = isAnimal ? true : false; // think of this as short cut for an
if and else

if (wasAnimal)
  System.out.println("wasAnimal is true");
```

NOTE: You cannot use the ternary operator for all if-else statements. You can use the ternary operator in place of the if-else statement only when the if and else parts in the if-else statement contain only one statement and both statements return the same type of values.

For reference, there is a list of all operators on Oracle's website:

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/opsummary.html

---

Brackets

You may have seen from time to time the use of brackets or no brackets! Let's demonstrate this through some examples:

```
boolean isMorning = true;

if (isMorning)
  System.out.println("Time for breakfast");
```

This is fine given we only have one statement to return or as our outcome.

Let's take the same example and do this:

```
boolean isMorning = true;

if (isMorning)
  System.out.println("Time for breakfast");
putTheKettleOn();
```

Now what do you think would happen here?

putTheKettleOn would not be called as part of the *isMorning* block.

To achieve what we want to, we need to enclose the lines into brackets. Otherwise known as a code block:

```
boolean isMorning = true;

if (isMorning) {
  System.out.println("Time for breakfast");
  putTheKettleOn();
}
```

Notice the difference?

There is nothing wrong with using no brackets like we saw in the first example, but we do have to abide by good coding standards.

We need to achieve KISS across our code base and to help readability we should include brackets wherever we can, even for single lined outcomes.

```
/*
1. Create a double variable with the value of 20
2. Create a second variable of type double with the value of 80
3. Add the numbers and then multiply by 25
4. Using the remainder operator, figure out the remainder and divide it by
40
5. Write and IF STATEMENT that compares the total again 20. If 20 or less
```

```
than or equal to then display a message, "Total was over the limit"

Operator precedence as we should have added the values and then multiplied
them. Here the multiply takes higher precedence.

https://introcs.cs.princeton.edu/java/11precedence/

*/
int limit = 20;
double first = 20d;
double second = 80d;

double total = (first + second) * 25; // <--- don't put brackers first
double remainder = total % 40;

System.out.println("Remainder is: " + remainder);

if (remainder <= limit)
  System.out.println("Total was over the limit")
```


```
Given two non-negative int values, return true if they have the same last
digit, such as with 27 and 57. Note that the % "mod" operator computes
remainders, so 17 % 10 is 7.
```

```
lastDigit(7, 17) → true
lastDigit(6, 17) → false
lastDigit(3, 113) → true
```

Solution:

```
public static boolean lastDigit(int a, int b) {
  // True if the last digits are the same
  return(a % 10 == b % 10);
}
```

Block statements

A block statement is a sequence of zero or more statements enclosed in braces.

A block statement is generally used to group together several statements, so they can be used in a situation that requires you to use a single statement.

There may be times where only one line would be sufficient in your block of code, whereas in others you may end up with several.

In Java Block statements start with a curly brace and end with the closing curly brace.

Anything between those braces is known as a block statement.

Let's see some examples (this is not the whole code but just to give you an idea):

```
// Declare a variable num1
int num1;

{ // Start of a block statement
        // Declares a variable num2, which is a local variable for this
block
        int num2;
        // num2 is local to this block, so it can be used here
        num2 = 200;
        // We can use num1 here because it is declared outside and before
this block
        num1 = 100;
} // End of the block statement
```

You can also nest block statements:

```
// Declare a variable num1
int num1;

{ // Start of a block statement
```

```
        // Declares a variable num2, which is a local variable for this
block
        int num2;
        // num2 is local to this block, so it can be used here
        num2 = 200;
        // We can use num1 here because it is declared outside and before
this block
        num1 = 100;

        {
          int num4 = 100;
          int num5 = 500;

          System.out.println(num4);
          System.out.println(num5);
        }
} // End of the block statement
```

Scope

Scope is an important part of writing code. We'll cover scope in this session.

```
{
  int numberOfStudents = 20;

  {
    String studentName = "John";
    System.out.println("In a class size of " + numberOfStudents + " there
is a student called " + studentName);

    // we can access numberOfStudents inside the INNER as it is declared
in the OUTER block.
  }
}
```

However ...

```
{
  int numberOfStudents = 20;

  {
    String studentName = "John";
    System.out.println("In a class size of " + numberOfStudents + " there
is a student called " + studentName);

    // we can access numberOfStudents inside the INNER as it is declared
in the OUTER block.
  }
```

```
    // this will produce a compilation error:
    System.out.println("Student is called " + studentName);
  }
```

This also would result in a compilation error:

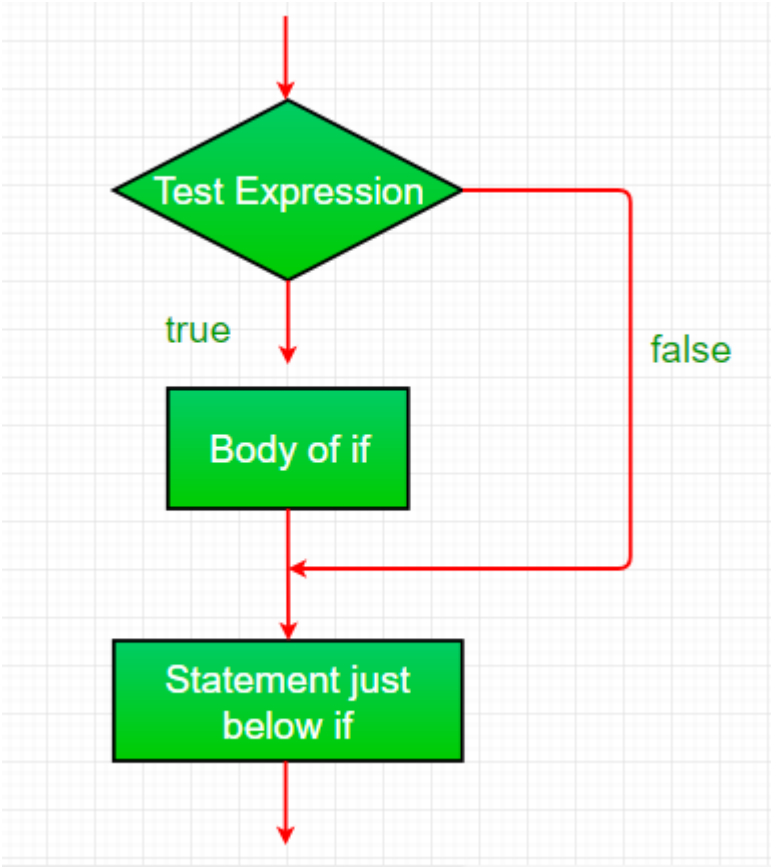
```
{
  int numberOfStudents = 20;

  {
    String studentName = "John";
    System.out.println("In a class size of " + numberOfStudents + " there
is a student called " + studentName);


    // this will produce a compilation error as it is already defined in
the outer scope:
    int numberOfStudents = 50;
  }
}
```

Conditionals

If Statements



The structure for a conditional is:

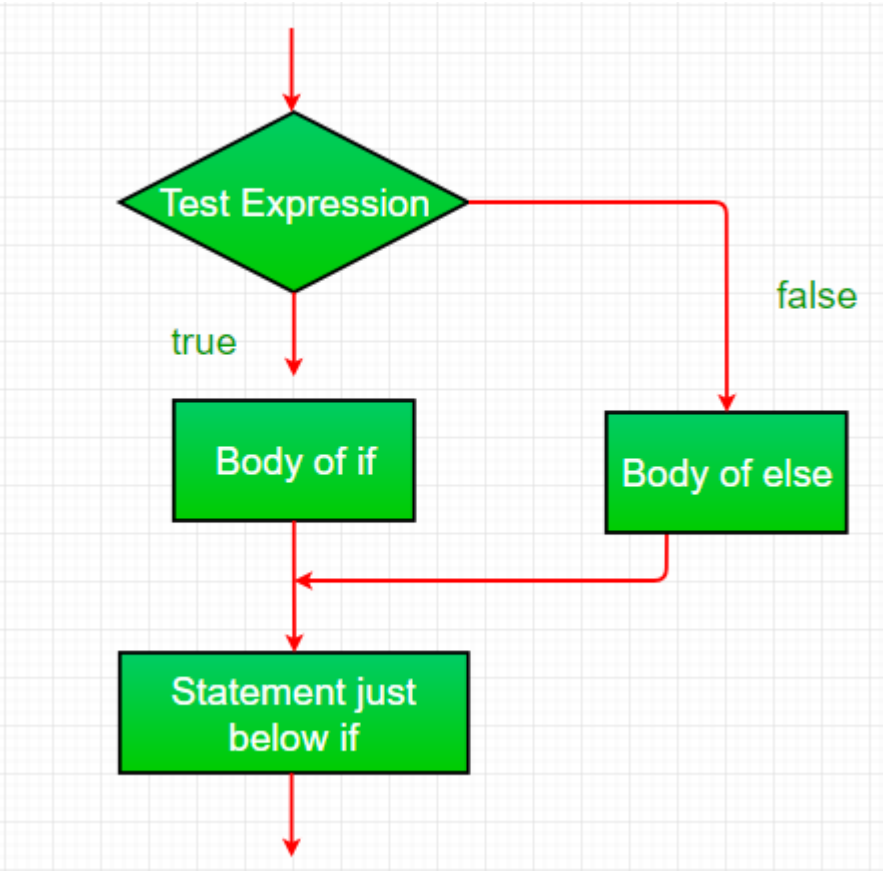
if (condition) { // if statement block }

The **condition** must be a boolean expression, i.e. true/false. Ideally it is a true unless you state otherwise.

Let's go through some examples:

```
public static void main(String [] args){
  int age = 30;

  if (age > 20) {
    System.out.println("No more free travel for you!");
  }
}
```



You can also use an **if then else** logic. Using our previous example:

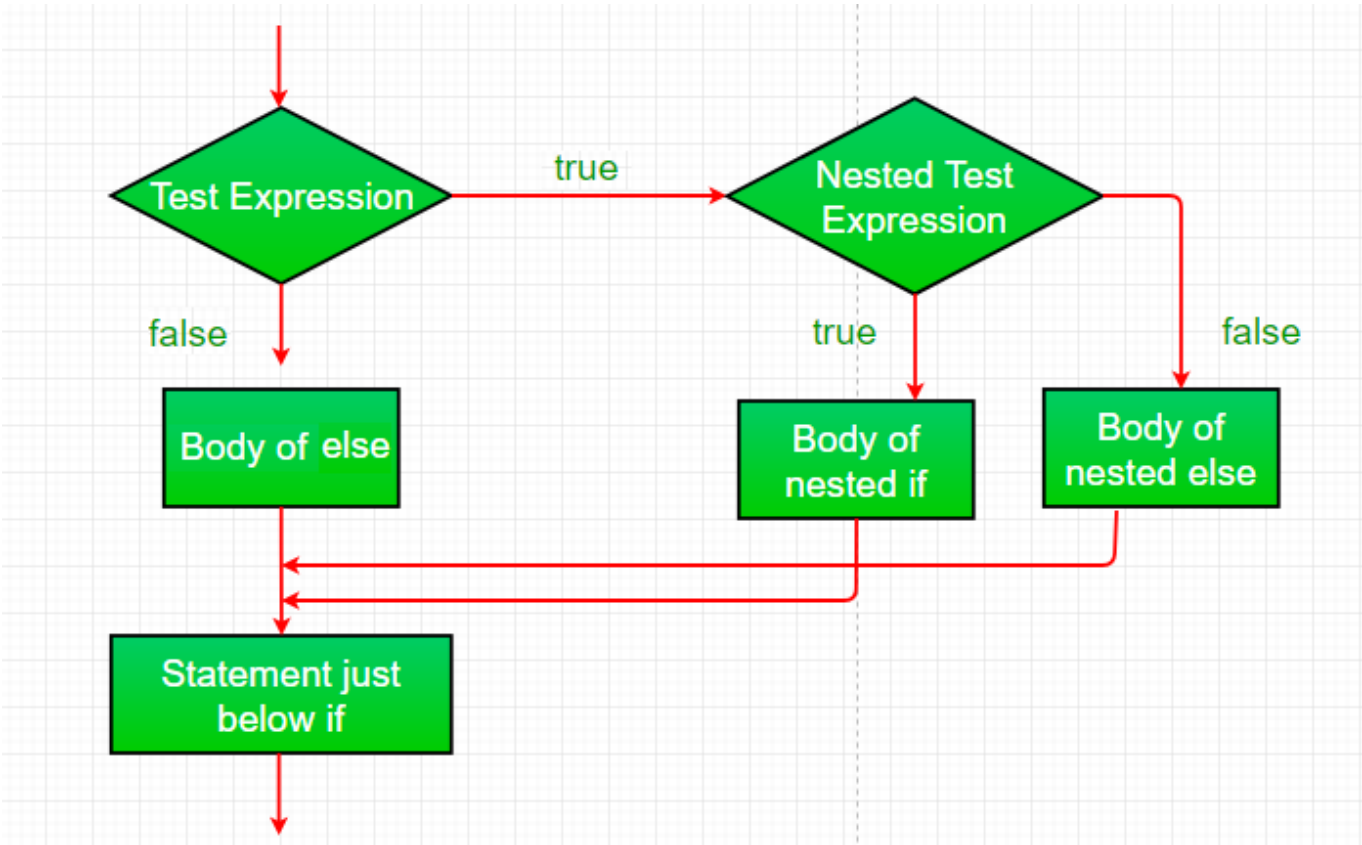
```
public static void main(String [] args){
  int age = 30;

  if (age > 20) {
    System.out.println("No more free travel for you!");
  }
  else { // <<< this is new
    System.out.println("Woohoo!");
  }
}
```

You can also use an **if then else if then** logic. Using our previous example:

```
public static void main(String [] args){
  int age = 31;

  if (age > 20) {
    System.out.println("No more free travel for you!");
  }
  else if (age > 30) { // <<< this is new
    System.out.println("I feel old!");
  }
  else {
    System.out.println("Still young!");
  }
}
```



Print my schedule:

Given that a day is represented by a number and Monday is represented as the number 1.

1. Print out when the day is a weekday.
2. On a Tuesday I go to the gym. Display this.
3. On a Thursday I go to the gym. Display this.
4. On a Sunday I do gardening. Display this.

You can use an **if statement** with any data type as long as it's **condition** evaluates to a boolean outcome.

For example:

```
String firstName = "John";

if (firstName.startsWith("J")) {
  System.out.println("A name starting with J!");
}
else {
  System.out.println("Found " + firstName + " instead?!");
}

double waterFall = 1.2;

if (waterFall >= 1.2) {
  System.out.println("It rained a lot!");
}

int temperatureDegrees = 100;
```

```
  if (temperatureDegrees > 30) {
    System.out.println("It's hot!!!!");
  }
```

Run this example and see what we get.

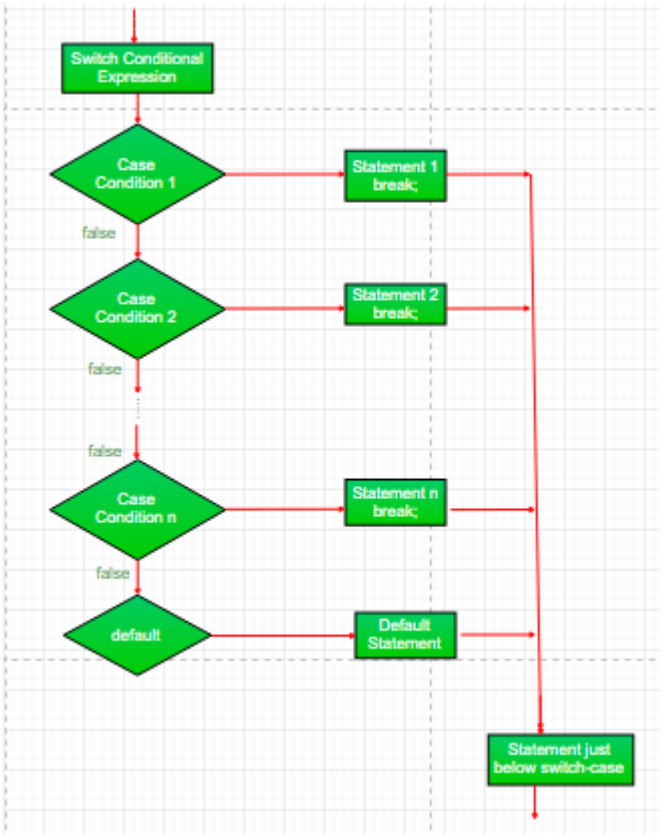
Scenario: You go to a restaurant however the waiter might say something rude to you instead of seating you. The variable "style" represents how well dressed you are and the variable "bribe" represents the bribe presented to the waiter. The waiter is satisfied if style is 8 or more and bribe is 5 or more. The waiter says "Not allowed in", if they are not satisfied.

Solution:

```
int style = 8;
int bribe = 5;

if ( !((style >= 8) && (bribe >= 5)) ) {
  System.out.println("Not allowed in");
}
```

Switch Statements




```
int coinInserted = 50;

switch (coinInserted) {
  case 50:
    System.out.println("50p");
    break;
  ...
  default:
    // ... must be provided
}
```

Operator Precedence

Having demonstrated all operators, it is important to cover the precedence operators have over one another.

```
int temperature = 100;
int excitementLevel = 60;

if (temperature > 30 && excitementLevel > 50) {
  System.out.println("Super excited");
}
```

but what if we had a request to change this so that when the ***temperature*** was below 30 and the ***excitementLevel*** was set to 20 then to print "Super Excited"?

changing just the first part wouldnt work:

```
int temperature = 30;
int excitementLevel = 20;

if (!temperature > 30 && excitementLevel > 50) {
  System.out.println("Super excited");
}
```

As this would only effect the temperature.

What do you think we need to do?

```
int temperature = 30;
int excitementLevel = 30;

if (!(temperature > 30 && excitementLevel > 50)) {
  System.out.println("Super excited");
}
```

The NOT operator has higher precedence.

Another challenge!

Best Practices

Braces

When an if statement contains only one line as a block of code, we can simply do the following:

```
boolean sayHello = true;

if (sayHello)
  System.out.println("Hello");
```

Notice that are no brackets. The compiler doesn't complain. Why? because it's valid.

However...

Readability

We use braces generally to make it easier to read through code as we can identify "blocks" naturally.

So our previous example would be:

```
boolean sayHello = true;

if (sayHello) {
```

```
  System.out.println("Hello");
}
```

Take the following example:

```
if (score >= 100) { ... }

if (100 <= score) { ... }

if (score > 99) { ... }
```

Which one is best? The first one. Why? Because it reads better "naturally". "Score is 100 or more".

You may wonder, what if I ever wanted to produce three or more outcomes? Well, you can use an **if then else if**.

This type of control flow statement allows you to produce multiple outcomes based on conditions being met on each "branch", for example, say we had a program or logic that alerted people of the time of the day:

```
int hourOfTheDay = 15;

if (hourOfTheDay < 15) {
  System.out.println("It's afternoon!");
}
else if (hourOfTheDay < 11) {
  System.out.println("Good morning!");
}
else {
  System.out.println("Evening!!");
}
```

TIP: The key is when you have more than 3 outcomes, you should re-think your solution to prevent working against the principles of KISS (readability, maintenance etc).

Whenever I mentioned readability and consistency I am referring to a key principle in development and in general: KISS.

> Given 3 int values, a b c, return their sum. However, if one of the values is 13 then it does not count towards the sum and values to its right do not count. So for example, if b is 13, then both b and c do not count.

luckySum(1, 2, 3) → 6 luckySum(1, 2, 13) → 3 luckySum(1, 13, 3) → 1

More Best Practies for If statements

Parenthesis

```
int a = 2;
int b = 3;
int c = 5;
int d = 6;

if ( a == b && c == d ) { // AVOID
}

if ( (a == b) && (c == d) ) { // AVOID
}
```