Logical Operators

This unit I am trying something with the slides. Please send me your feedback if this helps with lecture content or not. You can still download the old slide set / powerpoint, if you find that helpful - and the videos I record are using the old slide sets.

Review: Conditional Operators

- They take two **primative** values
  - provide a true or false answer

Standard Operations	Conditional Operators
a `+` b - Adds a and b	a `>` b - **true** when a is greater than b
a `-` b - subtracts b from a	a `<` b - **true** when a is less than b
a `*` b - multiply a and b	a `>=` b - **true** when a is greater than *or* equal to b
a `/` b - divide a by b	a `<=` b - **true** when a is less than *or* equal to b
a `%` b - remainder of a divided by b	a `==` b - **true** only when a equals b
a `=` b - assign the value of b to a	a `!=` b - **true** only when a does not equal b

When we want to compare two objects (such as Strings), we use `obj1.equals(obj2)`

- Why?
  - `==` only compares the local stack, which is just the memory addres of the object!
  - `.equals` allow the object to decide what that means
    - For `String` that means each character is compared from left to right.
- The .equals() method returns `true` or `false`

> **Pro Tip**
> We can use the not operator `!` to flip a sign. Which means `!(a === b)` and `a != b` are quivalent.
> Also the following code is valid
>
> ```
> boolean val1 = 10 < 5;  // sets false to val1
> System.out.println(!val1); // prints true
> ```
>
> It is especially useful when we want an object not equal to another object
>
> ```
> boolean isEquel = "hello".equals("hello");
> boolean notEqual = !("hello".equals("hello"));
> ```

Inclass Activity: BoundsChecker.java

The inclass activity is available on github under the following location (link is also on the syllabus under resources)

Download the activity either directly, or using git. You can also clone the entire handouts repo, by saying new project from version control.

Task 1: Basic Conditionals and Operations

For this first task, you will be writing a method called `boundsCheckConditionalsOnly` that takes in three paremeters

- lower
- upper
- value

You will then return a String based on the following:

- if the value is lower than the lower, return `null`
- if the value is equal to the lower, and lower than the upper
  - return the String "Near upper" if the value is closer to upper than lower
  - return the String "Near lower" if the value is closer to lower than upper
- if the value is equal to upper or higher, then return `null`
- Cavaet: You should *NOT* use logical operators yet (we will get to that in the next activity)

You will also see this method is tested with a few values. The method signature is already in the file: `BoundsChecker.java`

Reminder:

Try to draw out your logic **before** coding your logic!

```java
In [8]: public static String boundsCheckConditionalsOnly(int upper, int lower, int value) {
    if(value < upper) {
        if(value >= lower) {
            int difference = upper - value;
            if (value - lower < difference) {
                return "Near lower";
            }else {
                return "Near upper";
            }
        }
    }

    return null;
}

public static void testBounds(String expected, String actual) {
    System.out.printf("Results should be %s => Result is %s%n", expected, actual);
```

```
}

testBounds("Near upper", boundsCheckConditionalsOnly(10, 3, 8));
testBounds(null, boundsCheckConditionalsOnly(100, 0, 101));
testBounds(null, boundsCheckConditionalsOnly(100, 90, 80));
testBounds("Near lower", boundsCheckConditionalsOnly(100, 0, 49));
```

```
Results should be Near upper => Result is Near upper
Results should be null => Result is null
Results should be null => Result is null
Results should be Near lower => Result is Near lower
```

Reading Check-in

Given the following code, what value do we need to make ??

A - `&&`
B - `||`
C - `!=`

In [12]:
```
public static boolean readingCheckin(boolean value1, boolean value2) {
    return value1 ?? value2;
}

System.out.println(readingCheckin(false, false)); // prints false
System.out.println(readingCheckin(false, true));  // prints false
System.out.println(readingCheckin(true, false));  // prints false
System.out.println(readingCheckin(true, true));   // print true
```

```
false
false
false
true
```

Logical Operators

This works, but there is another way to do it!

- Conditional Operators work on primative values
- Logical Operators work *exclusively* on boolean values

Logical Operator	Properties
a `&&` b	both a *and* b are `true`
a `||` b	either a *or* b are `true`

This allows us to increase representational power of our logic.

> Deeper Understanding / Style Comment
> You can also use `==` and `!=` on boolean values.
> For the most part, you don't need to, and is considered poor style

```
"hello".equals("hello") != true // you want it to be false
!("hello".equals("hello")) // same as saying
"hello".equals("hello") != true
```

Using the not operator to "flip" the sign, is the preferred way to doing it.

Let's try it.

Time to play rock, paper, scissors

In [4]:
```java
public static void playGame(String player1, String player2) {
    System.out.printf("Player 1 throws: %s, and Player 2 throws: %s%n", player1, playe
    if(checkAnswer(player1, player2)) {
        System.out.println("Player 1 wins");
    } else if(checkAnswer(player2, player1)) {
        System.out.println("Player 2 wins");
    } else {
        System.out.println("No one wins!");
    }
}

// will be completed in class
public static boolean checkAnswer(String throw1, String throw2) {
    if(throw1.equals("rock") && throw2.equals("scissors")) {
        return true;
    }

    return false;
}

playGame("rock", "scissors");
playGame("rock", "paper");
playGame("paper", "scissors");
```

```
Player 1 throws: rock, and Player 2 throws: scissors
Player 1 wins
Player 1 throws: rock, and Player 2 throws: paper
No one wins!
Player 1 throws: paper, and Player 2 throws: scissors
No one wins!
```

Inclass Activity Task 2

In the method `boundsCheck` , rewrite your if statements from
`boundsCheckerConditionalsOnly` to use Logical Operators.

Yes, every bounds can be rewritten as a logical statement.

In [9]:
```java
public static String boundsCheck(int upper, int lower, int value) {
    if(value < upper && value >= lower) {
        int difference = upper - value;
        if (value - lower < difference) {
            return "Near lower";
        }else {
            return "Near upper";
        }
    }
    return null;
```

```
    }

    testBounds("Near upper", boundsCheck(10, 3, 8));
    testBounds(null, boundsCheck(100, 0, 101));
    testBounds(null, boundsCheck(100, 90, 80));
    testBounds("Near lower", boundsCheck(100, 0, 49));
```

```
Results should be Near upper => Result is Near upper
Results should be null => Result is null
Results should be null => Result is null
Results should be Near lower => Result is Near lower
```

Short Circuiting Operations

The `&&` operator has a special condition. It "short circuits".

- Short circuiting means once a `false` condition is found, it stops processing checks!
  - This is possible because the whole thing has to be true!
  - We can't do this with `||` because anything can be true.

Example:

In [11]:
```java
public static boolean testIt() {
    System.out.println("In test it!");
    return true;
}
boolean val1 = false;
boolean val2 = true;

boolean combined = val1 && val2 && testIt(); // will test it be executed?
```

Null Checks

If we have the following code:

In [13]:
```java
String value = null;
int len = value.length();
```

```
---------------------------------------------------------------------------
java.lang.NullPointerException: Cannot invoke "String.length()" because "REPL.$JShell
$43.value" is null
        at .(#44:1)
```

It will throw a `NullPointerException`

This means we are trying to access the 'empty' memory / null space!

The computer doesn't know what to do with that.

Null checking to the rescue

> Random Trivia
> The Null is called the "Billion Dollar Mistake", so use it carefully!

```
In [15]:  public static void lengthCheck(String value) {
              if(value != null && value.length() > 5) {
                  System.out.println("String is greater than 5");
              }
          }

          lengthCheck(null); // it didn't blow up AND value.length() was never executed!
          lengthCheck("Hello World");
```

String is greater than 5

Inclass Activity Task 3

A useful method for Strings is `.contains(String)` .

It returns `true` is a String contains the other String (case matters!).

For this task

- uncomment the line in main, and run the program.
- fix the program! (using the null check)

```
In [21]:  public static void reprintBounds(int upper, int lower, int value) {
              String check = boundsCheck(upper, lower, value);
              if(check != null && check.contains("upper")) {
                  System.out.printf("%d is closer to upper bound between %d and %d.%n", value, 
              }else if(check != null &&  check.contains("lower")) {
                  System.out.printf("%d is closer to lower bound between %d and %d.%n", value, 
              }else {
                  System.out.println("Out of bounds");
              }
          }

          reprintBounds(10, 3, 8);
          reprintBounds(100, 0, 101);
          reprintBounds(100, 90, 80);
          reprintBounds(100, 0, 49);
```

8 is closer to upper bound between 3 and 10.
Out of bounds
Out of bounds
49 is closer to lower bound between 0 and 100.