COMP 202
Foundations of Programming

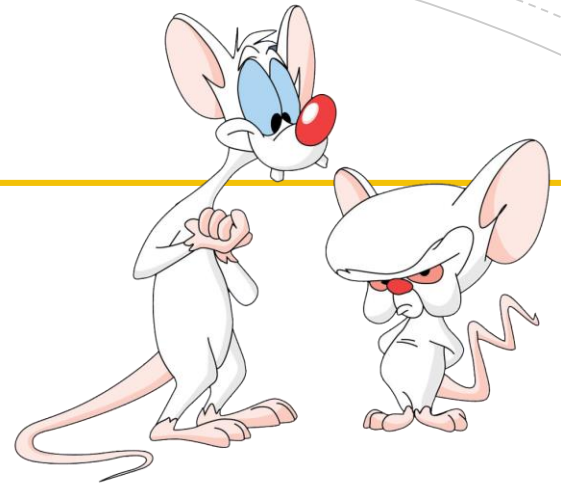
Lecture 5: Relational and Logical Operators, Conditional Statements

Giulia Alberini, Fall 2018

FROM LAST CLASS

- Expressions
- Command Line Arguments
- Random numbers

WHAT ARE WE GOING TO DO TODAY?



- Relational & Logical Operators
- Conditional Statements

RELATIONAL OPERATORS

RELATIONAL OPERATORS

- Used to check conditions and make comparisons.
  - Is chicken cheaper than beef?
- The result is either true or false, that is the result is a **boolean value**.
- Expressions containing relational operators are called **boolean expressions**.
- We can use a **boolean variable** to store the result of a boolean expression.

RELATIONAL OPERATORS

`x == y`	Is x equal to y?
`x != y`	Is x not equal to y?
`x > y`	Is x greater than y?
`x < y`	Is x less than y?
`x >= y`	Is x greater than or equal to y?
`x <= y`	Is x less than or equal to y?

Examples

- `5 > 2` is true.
- `7 < 1` is false

ORDER OF OPERATIONS

Which operator has higher priority?

1. Relational: <, >, <=, >=
2. Equality: ==, !=

You can always use parenthesis to raise priority.

DISPLAY BOOLEAN VALUES

- **As with other types of variable, you can display the value of a boolean variable using a print statement.**

```
double priceChicken = 7.71;
double priceBeef = 16.92;
boolean isChickenCheaper = priceChicken < priceBeef;
System.out.println("Chicken costs " + priceChicken + " dollars per kg.");
System.out.println("Chicken is cheaper than beef: " + isChickenCheaper);
```

```
Chicken costs 7.71 dollars per kg.
Chicken is cheaper than beef: true.
```

BE CAREFUL!

- Common error: use a single = instead of a double ==.
  - The single = is the assignment operator.
  - The double == is the equality operator.

- There no such thing as ≠, ≥, =<, or =>.

- The two sides of the relational operator need to be comparable.
  - `2 == 2.0` is true.
  - `2 ==` “2” does not compile.

TRY IT!

Write a program `isEven` that take an integer as input and displays on your screen whether it is true or false that such integer is even.

An example of what you could see on the Interactions pane is:

```
> run isEven 5
5 is an even number: false
```

LOGICAL OPERATORS

SO FAR…

We have seen

- `int/double` operators (‘+’,‘-’,‘*’,‘/’,‘%’):
  - They operate on `int/double`
  - Expressions containing them **evaluate to an `int/double` value**
- One `String` operator (‘+’):
  - It operates on `String`s
  - Expression containing such operator **evaluate to a `String` value**.
- Relational operators (‘==’,‘!=’,‘<’,‘>’,‘<=’,‘>=’):
  - They operates on compatible values (not on `String`!)
  - Expression containing them **evaluate to a `boolean` value**.

EXAMPLES

Consider the following variables:

```
int x = 1;
String s = "Today is the ";
```

- `4.0*(x/2)`
  - `4.0 * (1/2)`
  - `4.0 * 0`
  - `0.0`
- `s + x + 8`
  - `"Today is the " + 1 + 8`
  - `"Today is the 1" + 8`
  - `"Today is the 18"`
- `x > 5`
  - `1 > 5`
  - `false`
- `s > x`
  - Compile-time error!
- `true == x < 5`
  - `true == 1 < 5`
  - `true == true`
  - `true`

LOGICAL OPERATORS

- Logical operators take boolean expressions (i.e. expressions that evaluate to a boolean value) as inputs and produce a result of type boolean
- Java has 3 logical operators:
  - NOT `'!'`
  - AND `'&&'`
  - OR `'||'`
- NOT is a **unary** operator: it takes 1 input
- AND and OR are **binary** operators: they take 2 inputs

! OPERATOR

Let `b` be a variable of type `boolean`:

b	!b
true	false
false	true

`!b` evaluates to the opposite value of `b`.

! OPERATOR – EXAMPLES

- `!(2<3)`
  - `!true`
  - `false`

- `!(1.0 == 2.0)`
  - `!false`
  - `true`

&& OPERATOR

Let `a` and `b` be two variables of type `boolean`,

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

`a && b` evaluates to `true` if and only if both `a` and `b` are `true`.

&& OPERATOR – EXAMPLES

- `(1<2) && true`
  - `true && true`
  - `true`

- `(2 == 2) && !(3<5)`
  - `true && ! true`
  - `true && false`
  - `false`

|| OPERATOR

Let `a` and `b` be two variables of type `boolean`,

a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

`a || b` evaluates to `false` if and only if both `a` and `b` are `false`.

|| OPERATOR – EXAMPLES

- `(1>2) || true`
  - `false || true`
  - `true`

- `(2 == 1) || ! (1<2)`
  - `false || ! true`
  - `false || false`
  - `false`

ORDER OF OPERATIONS

From left to right:

1. ‘!’
2. ‘&&’
3. ‘||’

As usual, you can use parenthesis in order to change the priority.

EXAMPLES OF BOOLEAN EXPRESSIONS

What does `b && !a || b` **evaluate to if** `a = false` **and** `b = true`**?**

- `true && !false || true`
- `true && true || true`
- `true || true`
- `true`

EXAMPLES OF BOOLEAN EXPRESSIONS

What does `a && !(a || b)` **evaluate to if** `a = true` **and** `b = false`**?**

- `true && !(true || false)`
- `true && !true`
- `true && false`
- `false`

SHORT CIRCUIT EVALUATION

The evaluation of && and || stops as soon as the end result can be inferred.

- **If the left operand of && is `false`, then the whole expression is `false`. Therefore there is no need to evaluate the operand on the right!**
  Example:

```
2<1 && !(x >= 1 || y == 3)
```

  is `false` no matter what `!(x >= 1 || y == 3)` evaluates to!

- **If the left operand of || is `true`, then the whole expression is `true`. Therefore, Java will not evaluate the operand on the right.**
  Example:

```
1==1 || (x < 5)
```

  is `true` no matter what `x < 5` evaluates to.

SHORT CIRCUIT EVALUATION

It might seem like a small detail, but it is actually important.

Why is it useful?

- It can save time!
- It can avoid unnecessary errors.

Example:

```
(x != 0) && ( 5/x < 1)
```

If we try to evaluate the right operand when the left is false we will get a runtime error (you cannot divide by 0). Therefore, exploiting short circuit evaluation we can write conditions that will ensure us to not get a runtime error.

ORDER OF OPERATIONS

From left to right:

1. Parenthesis
2. !
3. Typecasting
4. Arithmetic
    i. *, /, %
    ii. +, -
5. Comparison
    i. Relational: <, >, <=, >=
    ii. Equality: ==, !=
6. Boolean: &&, ||

NOTE: Assignment happens after the evaluation of the expression!

EXAMPLES MIXED EXPRESSIONS

What does `false || 1 / (int) 2.0 < 3.5` **evaluate to?**

- `false || 1 / 2 < 3.5`
- `false || 0 < 3.5`
- `false || true`
- `true`

You don't need to memorize all this, use parenthesis when in doubt!

TRY IT!

Write a program that takes 3 integers x, y, z as inputs and prints out `true` if y is a number between x and z, `false` otherwise.

CONDITIONAL STATEMENTS

HOW CAN WE USE BOOLEANS?

To write useful programs, we almost always need to check conditions.

- We might want to execute certain statements only in specific situations.
- **Conditional statement** give us this ability

IF STATEMENT

The simplest conditional statement is the **if statement**.

```
if (x > 0) {
   System.out.println("x is positive");
}
```

IF STATEMENT

The simplest conditional statement is the **if statement**.

```
if (x > 0) {
   System.out.println("x is positive");
}
```

The expression in parentheses is called the **condition**.

It must be a boolean expression.

IF STATEMENT

The simplest conditional statement is the **if statement**.

```
if (x > 0) {
   System.out.println("x is positive");
}
```

The block of code gets executed *only if* the condition evaluates to `true`.

Otherwise, the block of code is skipped.

IF-ELSE STATEMENTS

If-else statements have two blocks of code:

- one gets executed *if* the condition evaluates to `true`
- the other gets executed *if* the condition evaluates to `false`
- The blocks are called **branches**

```
if (x > 0) {
   System.out.println("x is positive.");
}
else {
   System.out.println("x is not positive.");
}
```

IF-ELSE STATEMENTS – EXAMPLE

If `x > 0` evaluates to `true`, then the first block of code is executed and the second one is skipped.

```
if (x > 0) {
   System.out.println("x is positive.");
}
else {
   System.out.println("x is not positive.");
}
```

IF-ELSE STATEMENTS – EXAMPLE

If `x > 0` evaluates to `false`, then the first block of code is skipped and the second one is executed.

```
if (x > 0) {
   System.out.println("x is positive.");
}
else {
   System.out.println("x is not positive.");
}
```

TRY IT!

Let's go back to the program `isEven`. Let `X` be the integer the program takes as input. Then the program should either print: `The number X is even` OR `The number X is odd`.

TWO IFs VS IF-ELSE

What is the difference?

```
if (condition) {
   // some instructions
}
if (not condition) {
   // more instructions
}
```

```
if (condition) {
   // some instructions
}
else {
   // more instructions
}
```

- **Both blocks on the left could execute if somehow at the end of the first block, (not condition) becomes true.**

EXAMPLE

Both branches execute:

```
int x = 3;
if (x > 0) {
   System.out.println("Positive. Resetting value.");
   x = 0;
}
if (x <= 0) {
   System.out.println("Not positive.");
}
```

EXAMPLE

Only the first branch executes:

```
int x = 3;
if (x > 0) {
   System.out.println("Positive. Resetting value.");
   x = 0;
} else {
   System.out.println("Not positive.");
}
```

IF-ELSE IF-ELSE CHAINING

You might want to check related condition and choose one of several actions. You can do so by chaining a series of if and else statements.

- Only one of these blocks will get executed. **Order matters!**
- As soon as one block is executed, the remaining will be skipped
- You can have as many `else if`s as you want
- The final `else` is not required.

```
if (x > 0) {
   System.out.println("Positive");
} else if (x < 0) {
   System.out.println("Negative");
} else {
   System.out.println("Zero");
}
```

EXAMPLE

Is there anything wrong?

```
if (money > 0.0) {
   System.out.println("Positive balance");
} else if (money > 1000.0) {
   System.out.println("You're rich! Go celebrate!");
} else {
   System.out.println("Uh-oh.");
}
```

IF-ELSE IF-ELSE NESTING

You can also nest one conditional statement inside another.

- The first conditional statement has two branches
- The first branch contains a print statement
- The second branch contains another conditional statement with two branches of its own.

```
if (x > 0) {
   System.out.println("Positive");
} else {
   if (x < 0) {

   System.out.println("Negative");
   } else {
   System.out.println("Zero");
   }
}
```

BE CAREFUL!

- Nested or chained conditional statements are common, but can become very confusing!
- Indentation is essential for a program to be readable
- An accurate use of the curly brackets is essential for the program to work correctly.

TRY IT!

Modify the program `isEven` so that it either prints:

- `X is even`
- `X is an odd number multiple of 3`
- `X is an odd number not multiple of 3`

A DANGEROUS GAME!

- When a branch has only one statement, curly brackets are optional.

```
if (x > 0)
   System.out.println("Positive");
else
   System.out.println("Non positive");
```

- BUT, it is always better to use them in order to avoid mistakes like the following where the second print statement gets executed no matter how the condition evaluates.

```
if (x > 0)
   System.out.println("Positive");
   System.out.println("and not zero");
```

COMMON MISTAKE

```
if (x > 0) ;{
   System.out.println("Positive");
} else {
   System.out.println("Non positive");
}
```

- **Compile-time error!**

COMMON MISTAKE

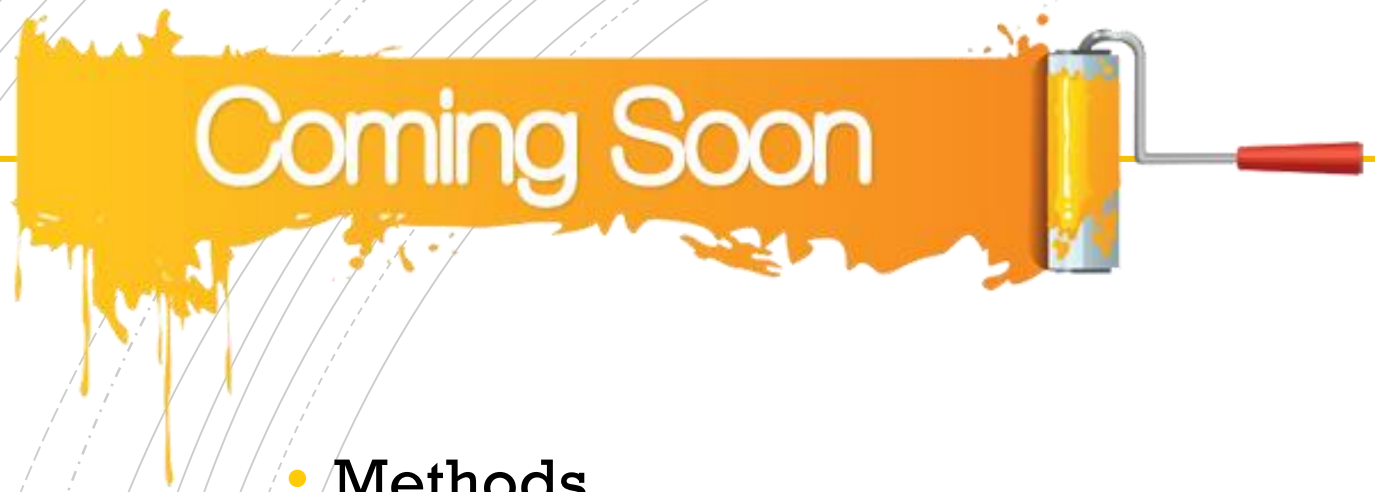
```
if (x > 0) ;{
    System.out.println("Positive");
}
```

- The statements inside the block will get executed no matter how the condition evaluates.

RECOMMENDED EXERCISES

1. Write a program that takes 2 integers x and y as input and prints whether is it true or false that x is a multiple of y.

2. Write a program that takes 2 integers x and y as input and prints which ever value that is the nearest to 10. It prints 0 in case of a tie.



- Methods