Download

Day 4: Conditionals

Download

Agenda

- Booleans
- Conditional Chaining
- Conditional Statements
- While Loops
- Primitive Type Casting
- Random Numbers
- Lab 2: Number Guessing Game

Download

Booleans

Review

Explicit/Literals	Implicit/Evaluated
`true`	`3 > 2`
`false`	`2 < 10`
	`3 == 3`
	`4 != 8`
	`5 >= 1`
	`9 <= 9`

Download

Conditional Chaining

Download

Remember that we can always check for multiple conditions at once using the **and/or** boolean keywords and can check for negation with the **not** keywords.

not

Unlike Python, **not** is not the "keyword" to use here. In Java, we use ! instead to denote **not**

```
!true -> false
```

not truth table

x	!x
`true`	`false`
`false`	`true`

and

Unlike Python, **and** is not the "keyword" to use here. In Java, we use && instead to denote **and**

```
true && false -> false
```

Download

and truth table

x	y	x && y
true	true	true
true	false	false
false	true	false
false	false	false

or

Unlike Python, **or** is not the "keyword" to use here. In Java, we use `||` instead to denote **or**

```
true || false -> true
```

Download

or truth table

x	y	x && y
true	true	true
true	false	true
false	true	true
false	false	false

Download

Boolean Algebra

Order of operations

- Left to Right
- Parentheses get evaluated first
- Not evaluated next
- And evaluated after
- Or evaluated last
- Parentheses do change this order, just like they do with regular arithmetic operations

```
true || (true && false) = true |&124; false = true
```

Download

Conditional Statements

if

Same as in Python, we have the **if** keyword.

However, unlike Python, we must wrap the condition in parentheses and have curly brackets after.

```
if (boolean condition){
  // code if the condition is true
}
```

else

If the condition in the **if** clause is incorrect, or false, and we want some other code to execute, we can use an **else** statement.

This is the same as in Python, where we don't give the else a condition. We add curly braces after and place our code between the braces.

```
if (boolean condition) {
  // code if the condition is true
}
else {
  // code if the condition is false
}
```

else if

If we have multiple conditions to check for, we can use an else if.

If the first condition is `false`, but we want to check for a secondary condition, we can use an `else if`.

This is better than having multiple `if` statements, as we can include an `else` at the end to catch all the other cases, if all cases are false.

```
if (condition 1){

  // code if condition 1 is true

}

else if (condition 2){

  // code if condition 2 is true

}

else {

  // code if the condition 1 and 2
  are false

}
```

else if

You can have as many else if statements as you want/need.

```
if (condition 1){
  // code if condition 1 is true
} else if (condition 2){
  // code if condition 2 is true
} else if (condition 3){
  // code if condition 3 is true
} else {
  // code if all conditions are
  false
}
```

While Loops

If we want to repeat code, we can use a `while` loop.

We can use the `while` keyword, with parentheses after, with a condition that must be true for the loop to run. If the condition becomes false, the loop will naturally stop.

```
while(true condition to run the loop) {

  // looping code goes here

}
```

Infinite Loops

We can create an infinite loop using `true` as our condition to run the loop.

This will never end unless it runs into a `break`

```
while (true) {

  // code

}
```

Stopping a loop

If we want to stop a loop in its tracks, we can use the `break` keyword.

The example to the right will keep going until x is greater than 5.

```
int x = 1;

while (true) {
  if (x > 5) {
    break;
  }
  x += 1;
}
```

Download

Conditioned Loops

Normally, we will end the loop based on some condition, instead of breaking when the condition of an if-statement is met.

The example to the right is the same while loop code as the code in the previous slide, but without an if-statement and using break.

```
int x = 1;

while(x > 5){

  x += 1;

}
```

When to use either

Infinite loops are great when multiple possible conditions may stop the loop

Conditioned loops are the "normal" way to write while loops. They will end when some condition is no longer true.

Primitive Type Casting

Download

What is the term we use to describe converting between types?

We did this in Python when we converted a String into an integer

Who remembers this term?

Download

Cast

Download

As a review, casting is the process of converting data from one type into another type.

Download

Between primitive data types, we can cast a value into a different type if we need.

We just need to put the type we want to cast the data into in parentheses before the value.

The example to the right casts the double into an integer.

```
double d = 13.024;

int i = (int) d;

System.out.println(i + 1); // gives
us 14
```

Download

This doesn't work with strings in the same way. Instead, we have to call another function to parse the data.

```
String s = "123";

int i = Integer.parseInt(s);

System.out.println(i + 3); // Gives
126
```

Random Numbers

There are multiple ways of generating a random number in Java. We'll use the `Math.random()` approach.

Download

- Using this method, we use `Math.random()` to generate a random number between 0 and 1.
- We then multiply the number by the different of a maximum and minimum value. This will shift our range from [0 to 1] to [0 to (max-min)]
- We then add a minimum to shift our minimum value. This shifts our range to [min to max].
- Lastly, we cast it into an integer to remove the decimal place.

```
int num = (int) (Math.random() *
(max-min) + min);
```

- In the example to the right, we would get a random number between 5 and 25, excluding 25.
- Math.random() gives us a double between 0 and 1 (exclusive of 1.)
- Multiplying that by (25-5) or 20, shifts our range of values to 0 to 20 (exclusive of 20).
- We then add 5 to shift the values to 5 to 25 (exclusive of 25)
- Casting it to an integers happens at the end, removing the decimals.

```
int num = (int) (Math.random() *
(25-5)) + 5;
```

Removing the parentheses would maintain the order of operations, as multiplication happens first, however, separating the operations leads to more readable code.

Ensure the order or operations is correct, or you risk having the incorrect range of [0, 20)

```
int num = (int) (Math.random() *
(25-5 + 5));
```

Download

Lab 2: Number Guessing Game