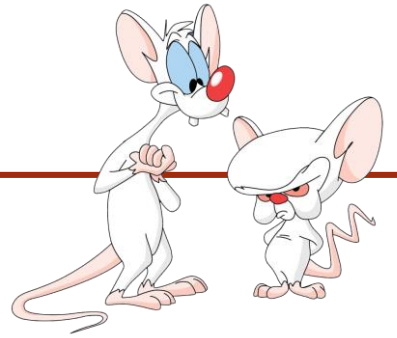
COMP 250
INTRODUCTION TO COMPUTER SCIENCE

Week 1-2: Java syntax Part 2

Giulia Alberini, Fall 2020

WHAT ARE WE GOING TO DO IN THIS VIDEO?



- More on Java syntax
- Scope of variables

MORE OPERATORS

OPERATION-ASSIGNMENT +=, -=, *=, /=

The following two blocks are equivalent


```
int x = 2;

x += 5;
```



```
int x = 2;

x = x + 5;
```


The same notation can be used for subtraction, multiplication, and division.

POST INCREMENT (DECREMENT)

- Post-increment: `x++`
- Post-decrement: `x--`

You can use these notations as statements as well as part of a more complex expression.

POST INCREMENT (DECREMENT)

The following statements are equivalent:


```
x++;                    x = x + 1;

x--;                    x = x - 1;
```

POST INCREMENT (DECREMENT)

When used as part of expressions, the increment happens after the statement is executed.

The following blocks are equivalent:

```
int x = 5;
int y = 2*x++;
```

```
int x = 5;
int y = 2*x;
x = x + 1;
```

RECOMMENDATION

Use ++ or -- by themselves. Don't write code with pre/post increment/decrement inside other expressions. It makes the code harder to read and more error prone.


```
int x = 5;

x = x++ + ++x + x++; // legal, but why??
```

MORE SYNTAX

IF-ELSE IF-ELSE

- 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");
}
```

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.

WHILE LOOP – SYNTAX


```
while (condition) {
   // some code
}
```


The block of code is repeatedly executed as long as the condition evaluates to true.

FOR LOOP – GENERAL STRUCTURE


```
for (statement1; boolean expression; statement2) {
   // loop body
}
```


- Any or all of the above statement/expression can be left out.
- The semicolons always need to be there.

EXAMPLE


```
for (int i = 1; i <= 10; i++) {
    // some code
}
```

EXAMPLE


```
for (int i = 1; i <= 10; i++) {
    // some code
}
```


The ***initializer*** is executed once, before the loop starts.

EXAMPLE


```
for (int i = 1; i <= 10; i++) {
    // some code
}
```


The ***condition*** is checked at the beginning of each iteration. If it evaluates to false, the loop ends. Otherwise, the body is repeated.

EXAMPLE


```
for (int i = 1; i <= 10; i++) {
    // some code
}
```


The ***update*** is executed at the end of each iteration.

TO RECAP

1) The initializer is executed.

2) The condition is checked. If true, the body is executed. Otherwise, the loop ends.

3) The update is executed and we go back to step 2).

VOID METHODS


```
public static void newMethod()
```


When used as part of a method header, the keyword **`void`** tells the computer that the method does not return anything.

VALUE METHODS

Compare to void methods, value methods differ in 2 ways:

- They declare the type of the return value
- They use at least one return statement to provide a return value.


```
public static double circleArea(double radius) {
   double area = Math.PI * Math.pow(radius, 2.0);
   return area;
}
```

- A call to this method could be: `double a = circleArea(2.5);`

DEAD CODE

- Code that appear after a return statement, or somewhere where it can never be executed, is called **dead code.**
- If your program contains dead code you will receive a **compile-time error**: "Unreachable statement".


```
public static double circleArea(double radius) {
   double area = Math.PI * Math.pow(radius, 2.0);
   return area;
   System.out.println("It will never print");
}
```

OVERLOADING

- Having more than one method with the same name is called **overloading**.
- It is legal as long as the methods have different parameters.
- Java will know, based on the inputs, which method has been called.

RECAP ON METHODS – GENERAL FORM

- **Keywords (e.g.,** `public static`**)**
- **Return type (e.g.** `void, int, boolean`**)**
- **Name of the method**
- **Parentheses with parameters declarations**
- **Body of the method (the instructions)**

  Written between curly brackets and including at least one `return` **statement if not** `void`**.**

HOW TO READ THE DOCUMENTATION

DOCUMENTATION

- The Java Application Programming Interface (API) is a list of all classes that are part of the JDK. You can find the complete list here: https://docs.oracle.com/javase/8/docs/api/

- You can visit the above link if you want to know more about library or methods you'd like to use.
  E.g. All about the Math library:
  https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html

Method Summary

Methods

Modifier and Type	Method and Description
static double	**abs**(double a) Returns the absolute value of a double value.
static float	**abs**(float a) Returns the absolute value of a float value.
static int	**abs**(int a) Returns the absolute value of an int value.
static long	**abs**(long a) Returns the absolute value of a long value.
static double	**acos**(double a) Returns the arc cosine of a value; the returned angle is in the range 0.0 through *pi*.
static double	**asin**(double a) Returns the arc sine of a value; the returned angle is in the range -*pi*/2 through *pi*/2.
static double	**atan**(double a) Returns the arc tangent of a value; the returned angle is in the range -*pi*/2 through *pi*/2.
static double	**atan2**(double y, double x) Returns the angle *theta* from the conversion of rectangular coordinates (x, y) to polar coordinates (r, *theta*).
static double	**cbrt**(double a) Returns the cube root of a double value.
static double	**ceil**(double a) Returns the smallest (closest to negative infinity) double value that is greater than or equal to the argument and is e
static double	**copySign**(double magnitude, double sign) Returns the first floating-point argument with the sign of the second floating-point argument.
static float	**copySign**(float magnitude, float sign) Returns the first floating-point argument with the sign of the second floating-point argument.
static double	**cos**(double a) Returns the trigonometric cosine of an angle.
static double	**cosh**(double x) Returns the hyperbolic cosine of a double value.
static double	**exp**(double a)

SAMPLE ENTRY

```
static double        abs(double a)
```

Returns the absolute value of a double value.

SAMPLE ENTRY

```
static double        abs(double a)
```

Returns the absolute value of a double value.

It specifies the **name** of the method

SAMPLE ENTRY

```
static double        abs(double a)
```

Returns the absolute value of a double value.

The **number and type of input parameters**

SAMPLE ENTRY

`static double` **`abs(double a)`**

Returns the absolute value of a double value.

Which together they constitute the **method's signature**

SAMPLE ENTRY

`static` **`double`** `abs(double a)`

Returns the absolute value of a double value.

The **return type** of the method

SAMPLE ENTRY

`static` `double` `abs(double a)`

Returns the absolute value of a double value.

Keyword that tell you how the method can be called.

We'll find out more about this in the following weeks,

so don't worry about it for now.

SAMPLE ENTRY

`static double abs(double a)`

Returns the absolute value of a double value.

And a description of what the method does.

abs

```
public static double abs(double a)
```

Returns the absolute value of a double value. If the argument is not negative, the argument is returned. If the ar

- If the argument is positive zero or negative zero, the result is positive zero.
- If the argument is infinite, the result is positive infinity.
- If the argument is NaN, the result is NaN.

In other words, the result is the same as the value of the expression:

```
Double.longBitsToDouble((Double.doubleToLongBits(a)<<1)>>>1)
```


Parameters:

a - the argument whose absolute value is to be determined

Returns:

the absolute value of the argument.

EXAMPLE OF OVERLOADING

Note that `Math.abs()` is overloaded

`abs``(double a)`
Returns the absolute value of a `double` value.

`abs``(float a)`
Returns the absolute value of a `float` value.

`abs``(int a)`
Returns the absolute value of an `int` value.

`abs``(long a)`
Returns the absolute value of a `long` value.

THE SCOPE OF A VARIABLE

SCOPE OF A VARIABLE

- A variable **only exists** inside the block in which it is declared.
- It does not exist anywhere else outside that block.
- When inside a block,
  - a variable ***starts*** to exists when it is declared, and
  - it ***ends*** to exists at the blocks in which it was declared ends.

EXAMPLE 1

```
int x = 5;
if (x > 0) {
      int y = 0;
} else {
      int y = x;
}
System.out.println(y);
```


```
int x = 5;
int y;
if (x > 0) {
      y = 0;
} else {
      y = x;
}
System.out.println(y);
```

EXAMPLE 2

```
int x = 2;
int y = 3;
if (x < y) {
    x = x + y;
    int z = 5;
    y = z*x;
}
System.out.println(x + " " + y + " " + z);
```


```
int x = 2;
int y = 3;
int z = 0;
if (x < y) {
    x = x + y;
    z = 5;
    y = z*x;
}
System.out.println(x + " " + y + " " + z);
```

EXAMPLE 3

```
for (int i = 0; i < 5; i++) {
      System.out.println(i);
}
System.out.println(i);
```


```
int i;
for (i = 0; i < 5; i++) {
      System.out.println(i);
}
System.out.println(i);
```

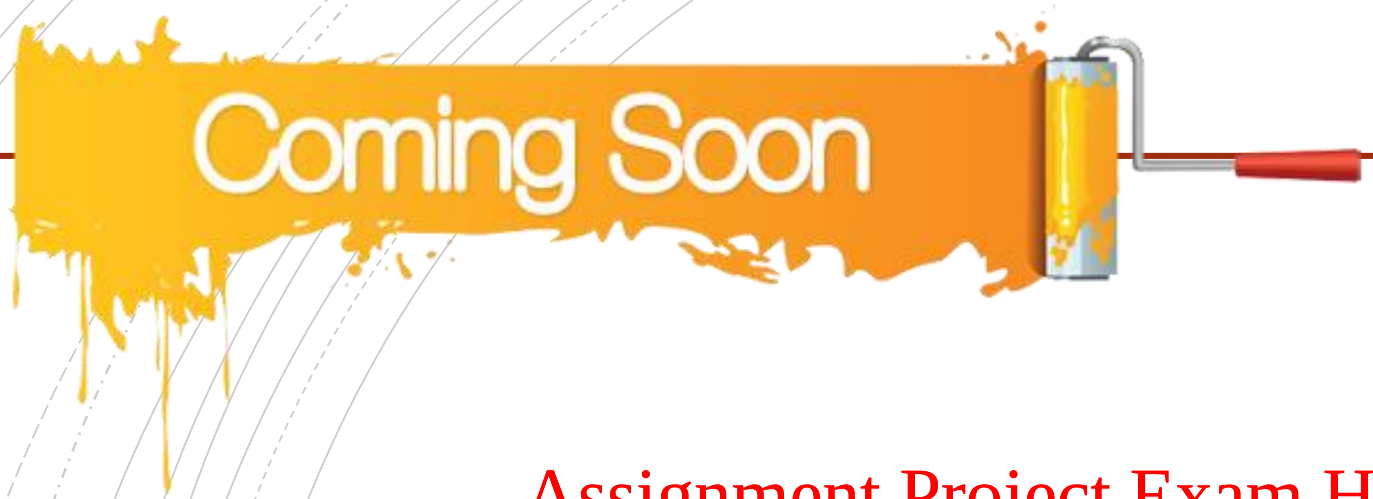
EXAMPLE 4

```
int x = 5;
if (x > 0 || isSnowing) {
      System.out.println(x);
}
```


```
int x = 5;
boolean isSnowing = false;
if (x > 0 || isSnowing) {
      System.out.println(x);
}
```

RECOMMENDED EXERCISES

- Look at code you have previously written in Python (or in any other programming language) which uses numbers (like integers or doubles), variables, conditional statements, loops, and functions. Translate this code into Java.



In the next video we will be talking about primitive data types and strings.