Java Primitive Types

- Can be remember as Java native types
- These numeric types or character types are built in to help calculation
- Java primitive types are really sensitive to the value it get assigns to
    - A floating point number cannot be assigned to a int
    - A non-floating number cannot be assigned to a boolean
- Gramma matters

All Java Primitive Types

- short: positive and negative integer - 16 bit - 2 bytes
- int: positive and negative integer - 32 bit - 4 bytes
- long: positive and negative integer - 64 bit - 8 bytes
- float: positive and negative floating point number - 32 bit - 4 bytes
- double: positive and negative floating point number - 64 bit - 8 bytes
- boolean: ideally 1 bit, but different in every JVM (true and false)
- char: 16 bit - 2 bytes (represent letters and symbols)

Java Statement

- A complete Java statement includes identifier, variable name, variable value, and a semicolon.
- A identifier defines the java type, Java primitive type can be a identifier.
- A variable name have to follow the java gramma standard
    - All alphabet (upper case and lower case) are allowed
    - Can connect alphabet with underscore _ or dollar sign $. All other symbols are illegal gramma
    - Number 0-9 are allowed, bet a initial letter is required. Just number alone is invalid
- To finish one Java Statement, the semicolon is a must.

boolean

- The boolean identifier is a Java primitive type

- A boolean variable can be assigned with true of false

Java Operators

- Java Operators are built in Java special grammas to do mathematical or logical calculations.

- There are
  - Arithmetic Operators: for mathematical calculations
  - Relational Operators: compare java variables
  - Logical Operators: for logical calculations
  - Assignment Operators: for advanced calculations with assignment ability
  - Increment and Decrement Operators: give variable easy +1 and -1 java gramma options

Arithmetic Operators

- +: used for addition
- -: used for subtraction
- *: used for multiply
- /: used for division
- % (mod): used to get the remains of a division

Relational Operators

- Execute the statement from left to right, relational operators give either true or false
- == : determine whether the left side is equals to the right side value
- !=: determine whether the left side is not equals to the right side value
- >: determine whether the left side is larger than the right side value
- <: determine whether the left side is smaller than right side value
- >=: determine whether the left side is larger or equals to right side value
- <=: determine whether the left side is smaller or equals to right side value

Logical Operators ||

Statement 1.	Statement 2.	Operator.	Value
true	true	||	true
true	false	||	true
false	true	||	true
false	false	||	false

One of the conditions need to be satisfied

Logical Operators &&

Statement 1.	Statement 2.	Operator.	Value
true	true	&&	true
true	false	&&	false
false	true	&&	false
false	false	&&	false

Both condition need to be satisfied

Logical Operators !

Statement	Operator.	Value
true	!	false
false	!	true

Opposite

Logical Operation Append Rules

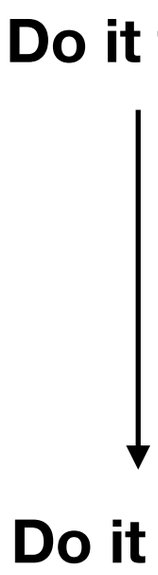
Condition1 && Condition2 && Condition3 && ….

The more && statements get appended, the more strict the condition is

Condition1 || Condition2 || Condition3 || ….

The more || statements get appended, the more flexible the condition is

Operators Computing Order



1. ! ++ —
2. * / %
3. + -
4. < > <= >=
5. == !=
6. &&
7. ||
8. = += -= *= /= %=

note: The horizontal order does not matter

Java if else statement

- Control the where the next code execution goes to
- By given one or more logical statement to establish statement
  - if
  - if + else
  - if + else if + … + else
- Nest if else statement
  - The inner statement can be executed only if outer statement is passed
  - Nested statement can be understand as logical condition dependency