## Assignment 1

## Java Operators

Operators are used to perform operations on variables and values.

Java divides the operators into the following groups:

* Arithmetic operators
* Assignment operators
* Comparison operators
* Logical operators
* Bitwise operators

## Arithmetic Operators

Arithmetic operators are used to perform common mathematical operations.

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Name | Description | Example |
| + | Addition | Adds together two values | x + y |
| - | Subtraction | Subtracts one value from another | x - y |
| \* | Multiplication | Multiplies two values | x \* y |
| / | Division | Divides one value by another | x / y |
| % | Modulus | Returns the division remainder | x % y |
| ++ | Increment | Increases the value of a variable by 1 | ++x |
| -- | Decrement | Decreases the value of a variable by 1 | --x |

## Assignment Operators

Assignment operators are used to assign values to variables.  
We use the assignment operator (=) to assign the value. For ex-

int x = 10;

The **addition assignment** operator (+=) adds a value to a variable:

int x = 10;

x += 5;

A list of all assignment operators:

|  |  |  |
| --- | --- | --- |
| Operator | Example | Same As |
| = | x = 5 | x = 5 |
| += | x += 3 | x = x + 3 |
| -= | x -= 3 | x = x - 3 |
| \*= | x \*= 3 | x = x \* 3 |
| /= | x /= 3 | x = x / 3 |
| %= | x %= 3 | x = x % 3 |
| &= | x &= 3 | x = x & 3 |
| |= | x |= 3 | x = x | 3 |
| ^= | x ^= 3 | x = x ^ 3 |
| >>= | x >>= 3 | x = x >> 3 |
| <<= | x <<= 3 | x = x << 3 |

## Comparison Operators

Comparison operators are used to compare values:

|  |  |  |
| --- | --- | --- |
| Operator | Name | Example |
| == | Equal to | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

## Logical Operators

Logical operators are used to determine the logic between variables or values:

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Name | Description | Example |
| && | Logical and | Returns true if both statements are true | x < 5 && x < 10 |
| || | Logical or | Returns true if one of the statements is true | x < 5 || x < 4 |
| ! | Logical not | Reverse the result, returns false if the result is true | !(x < 5 && x < 10) |

Java Statement

Java statement is just an instruction that explains what should happen.

## Types of Java Statements

Java supports three different types of statements:

* [Expression statements](https://www.thoughtco.com/expression-2034097) change values of variables, call methods, and create objects.
* [Declaration statements](https://www.thoughtco.com/declaration-statement-2034076) declare variables.
* Control-flow statements determine the order that statements are executed. Typically, Java statements parse from the top to the bottom of the program. However, with control-flow statements, that order can be interrupted to implement branching or [looping](https://www.thoughtco.com/loops-2034224) so that the Java program can run particular sections of code based on [certain conditions](https://www.thoughtco.com/conditional-statements-2034048).

## Examples of Java Statements:

//declaration statement

int number;

//expression statement

number = 4;

//control flow statement

if (number < 10 )

{

//expression statement

System.out.println(number + " is less than ten");

}

# Unicode System

|  |
| --- |
| Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages. |

### Why java uses Unicode System?

|  |
| --- |
| Before Unicode, there were many language standards: |
| * **ASCII**(American Standard Code for Information Interchange) for the United States. * **ISO 8859-1**for Western European Language. * **KOI-8**for Russian. * **GB18030 and BIG-5**for chinese, and so on. |

|  |
| --- |
| In unicode, character holds 2 byte, so java also uses 2 byte for characters. |
| lowest value:\u0000 |
| highest value:\uFFFF |

|  |
| --- |
| Java Naming conventions Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method, etc.  But, it is not forced to follow. So, it is known as convention not rule. These conventions are suggested by several Java communities such as Sun Microsystems and Netscape.  All the classes, interfaces, packages, methods and fields of Java programming language are given according to the Java naming convention. If you fail to follow these conventions, it may generate confusion or erroneous code. Advantage of naming conventions in java By using standard Java naming conventions, you make your code easier to read for yourself and other programmers. Readability of Java program is very important. It indicates that less time is spent to figure out what the code does.  The following are the key rules that must be followed by every identifier:   * The name must not contain any white spaces. * The name should not start with special characters like & (ampersand), $ (dollar), \_ (underscore).   Let's see some other rules that should be followed by identifiers. Class  * It should start with the uppercase letter. * It should be a noun such as Color, Button, System, Thread, etc. * Use appropriate words, instead of acronyms. * Example: -  1. **publicclass**Employee 2. { 3. //codesnippet 4. }  Interface  * It should start with the uppercase letter. * It should be an adjective such as Runnable, Remote, ActionListener. * Use appropriate words, instead of acronyms. * Example: -  1. **interface**Printable 2. { 3. //codesnippet 4. }  Method  * It should start with lowercase letter. * It should be a verb such as main(), print(), println(). * If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed(). * Example:-  1. **class**Employee 2. { 3. //method 4. **void**draw() 5. { 6. //codesnippet 7. } 8. }  Variable  * It should start with a lowercase letter such as id, name. * It should not start with the special characters like & (ampersand), $ (dollar), \_ (underscore). * If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName. * Avoid using one-character variables such as x, y, z. * Example :-  1. **class**Employee 2. { 3. //variable 4. **int**id; 5. //codesnippet 6. }  Package  * It should be a lowercase letter such as java, lang. * If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang. * Example :-  1. **package**com.javatpoint;//package 2. **class**Employee 3. { 4. //codesnippet 5. }  Constant  * It should be in uppercase letters such as RED, YELLOW. * If the name contains multiple words, it should be separated by an underscore(\_) such as MAX\_PRIORITY. * It may contain digits but not as the first letter. * Example :-  1. **class**Employee 2. { 3. //constant 4. **staticfinalint**MIN\_AGE=18; 5. //codesnippet 6. }  CamelCase in java naming conventions Java follows camel-case syntax for naming the class, interface, method, and variable.  If the name is combined with two words, the second word will start with uppercase letter always such as actionPerformed(), firstName, ActionEvent, ActionListener, etc. |