Cheatography

Java Mastery - Fundamentals Cheat Sheet

by Bayan (Bayan.A) via cheatography.com/122738/cs/22947/

Primitive Data Types

int	32-bit
long	64-bit
short	6-bit
byte	8-bit
double	double-precision 64-bit
float	single-precision 32-bit
boolean	Boolean value (`true or false`)
char	16-bit Unicode character

Variables/Identifiers

Start with a letter (or_ or $)

Rest **must** be letters, _, $ or digits

Case sensitive	Start with a lower-case letter

Assignment statement **replaces the previously** stored value

Use **camelCasing**	thisIsCamelCasing

Operator Precedence and function

From high (16) to low (1)

Operator	Description
(16) [ ], . , ()	Access to array element, access to object member, parantheses
(15) ++, --	Unary post-increment, unary post-decrement
(14) ++, --, +, -, !, ~	unary pre-increment, unary pre-decrement, unary plus, unary minus, unary logical NOT, unary bitwise NOT
(13) (), new	cast, object creation

Operator Precedence and function (cont)

(12)*, /, %	multiply, divide, modulus
(11)+-, +	additive, string concatenation
(10<<, >>, >>>	shift
(9) <, <=, >, >=	relational; greater than, less than (or equal to)
(8) ==, !=	equaly, not equal
(7) &	bitwise AND
(6) ^	bitwise XOR
(5) |	bitwise OR
(4) &&	logical AND
(3) ||	logical OR
(2) ?:	Ternary
(1) =, +=, -=, *=, =, /=, %=, &=	Assignment

Syntax

A specific set of rules, using a combination of keywords and other things

Each *keyword* has a spoecific meaning, and sometimes need ot be used in specific orders.

Case-sensitive. public, Public and PUBLIC are all different

Semi-colon defines the end of a statement

;	Must be at the end of every statement

class

Defines a class with the name after the keyword

Curly braces defines the **class body**

Anything in the curly braces is "part" of this class

note, **semi-colon is not inserted** after the class name

```
public class Hello {
}
```

Access Modifiers

These are java keywords

Allows defining the scope, how other parts of the code can access this code

Access Modifiers	Access Levels
public	Same Class, same package, other subclass, other package
protected	Same Class, same package, other subclass
no access modifier	Same Class, same package
private	Same Class

Access to members permitted by each modifier

Method

Collection of statements that perform an operation

main method	Entry point of any Java code
void	Java keyword
	Indicates method returns nothing
( )	**mandatory** method declaration

By **Bayan** (Bayan.A)
cheatography.com/bayan-a/

Published 11th February, 2021.
Last updated 24th May, 2020.

Cheatography

Java Mastery - Fundamentals Cheat Sheet

by Bayan (Bayan.A) via cheatography.com/122738/cs/22947/

Method (cont)

	can include 1 or more parameters
{ }	**Code block**
	Mandatory in a method declaration
	Defines start and end of method
	Place statements to perform tasks
Statement	Complete command to be executed
	Can include more than one expressions

```
public static void main(String[] args) {
}
```

Variables

Way to store information	
Accessed via name given	
Can be changed	
Must define the variables type of data	known as **Data Types**
Must initialise before use	
Declaration Statement	Specify data type, then varaiable name
	optionally, add an expression to intialise a value
Data types do not form part of the expression	
Example: `int myNumber = 50`	**`myNumber = 50`** is the expression, **not** `int`

Literals

Boolean	`true` represents a true boolean value		
	`false` represents a false boolean value		
String data	**"string"**	Sequence of characters (including Unicode)	
Numeric	There are three main types: int, double, char		
	`int`	integer	Whole number- (without decimal points)
	`double`	Floating point	decimal fractions / exponential notation
	`char`	character	Stores the 16-bit Unicode integer value of the character in question.

Operand

Describes any object manipulated by an operator

`int myVar = 15 + 12;`	`+` is the operator `15` and `12` are the operands
	Variables instead of literals are also operands

Expression

Combination of variables, literals, method return values, and operators

Variable assignment without the data type declaration, or the string in " " being printed, and **not** the semi-colon

Examples:

`int myVar = 15 + 12;`	`15 + 12` is the expression
	same if variables replace number literals

```
int myVariable= 50;
```

`myVariable= 50`	Expression

```
System.out.println("Random string");
```

`"Random string"`	Expression

```
if(myVariable > 50)
```

`myVariable > 50`	Expression

Expressions and Statements

A **statement** is the entire code, from data type declaration, ending at the semi-colon,

`int myVariable= 50;`	Statement

Expressions and Statements (cont)

`System.out.println("random string");	Statement
`myVariable++`	Statement

Wrapper Class Limit

Can be experienced by all primitve data types

Overflow	Putting too large a number allocated by the computer for an integer	
e.g.	`Integer.MAX_VALUE + 1 =`	-2147483648
Underflow:	Putting too small a number allocated by the computer for an integer	
e.g	`Integer.MIN_VALUE - 1 =`	21474-83647

Going past a limit on either side(max/min) often results in cycling to opposite side. i.e. less than the min cycles to the max, and more than max cycles to the min

Integer (Wrapper Class)

Occupies 32 bits	has a width of 32	
`Integer`	Gives ways to perform operations on an `int`	
int numbers can be written with _ for readability	e.g. 2_147_483_647	(version 7 or higher)

Integer (Wrapper Class) (cont)

	`Integer.MAX_VALUE`	-2147483648
	`Integer.MAX_VALUE`	2147483647
A whole number	Doesn't handle the remainders	e.g. `int myInt = 5 / 2;` myInt = 2

Byte (Wrapper Class)

Occupies 8 bits	"byte has a width of 8"	
`byte`	Mostly used as documentation to show it is small	
	Smaller data type takes less space and provides quicker access	
	e.g. `byte myMinByteValue =`	-128
	`byte myMaxByteValue =`	127

Not used as often, due to computers today having more space.

Short - Wrapper Class

Occupies 16 bits	"has a width of 16"	
`short`		
	e.g. `Short.MIN_VALUE`	-32768
	e.g. `Short.MAX_VALUE`	32767

Long (Wrapper Class)

Used for an integer larger than the amount an `int` can store

Has a width of 64	can store 2 to the power of 63

Long variables require an uppercase "L" at the end of a number

	e.g. `myLongValue = 100L;`
	Otherwise, it is treated as an `int`

Single and Double Precision

Refers to format and space occupied by type.

Single Precision	*Has a width of 32*
	(Occupies 32 bits)
Double Precision	*Has a width of 64*
	(Occupies 64 bits)

Floating Point Numbers

`float`	`float myFloatValue = 5.25f;`
	By default, Java assumes it's a `double`, requiring the `f` after the number
Unlike whole numbers	Has fractional parts expressed with a decimal point
	e.g. 3.14159

Also known as "real numbers"

Used for more precise calculations

Aren't recommended to use much these days

A **single precision number**

By **Bayan** (Bayan.A)
cheatography.com/bayan-a/

Published 11th February, 2021.
Last updated 24th May, 2020.

Floating Point Numbers (cont)

Smaller and less precise than Double	Range: 1.4E to 3.4028235E+38
Requires less memory	32 bits / 4 bytes

Double

`double`	`double myDoubleValue = 5.25d;`

A **Double Precision Number**

Requires more memory	64 bits / 8 bytes
Larger range and more precise than Single	Range: 4.9E-324 to 1.797693134862315-7E+308

char

`char`	`char myChar = 'D';`
Stores only 1 character	>1 character prompts an error

Single ' used, not like that used for "strings"

Occupies 16 bits	"width of 15"
	Not a single byte, as it allows to store Unicode characters

Used to store data in arrays

Using Unicode, `\u` must be before the specific code is used	`char myUnicodeChar = '\0044';` Displays "D"

Unicode

International encoding standard

Use with different languages & scripts

Each letter, digits, or symbol is assigned a **unique numeric value**

Unicode (cont)

This value applies accross different platforms and programs

Allows representation of different languages

Can represent any one of 65535 different types of characters	via combination of two bytes in memory
Full list of unicode characters:	www.unicode-table.com/en/#control-character

Boolean

Allows only two choices	`true` or `false`
Variable names commonly written as a question	`boolean isJavaEasy = true;`

String

A datatype that is NOT a primitive type	Actually a **Class**
A sequence of characters	Can contain a single character
	`String myString = "This is a string";`
Can use Unicode characters	`String myString + "-\u00A9 2019";`
Treats texts or digits typed as text only	No numerical calculations are done.
String variables added with another variable append them only	`String myNumber = "250"; String yourNumber = "654"; myNumber + yourNumber = 250654`

String (cont)

Strings are immutable	Can't be changed after created

Code Blocks

Variables that exist outside the code block can be accessed inside the code block

But variables created within an if statement are deleted once the program leaves the code block

e.g.:

```
int score = 10
if(gameOver) {
  int finalScore = score + bonus;
}
int saveScore = finalScore;
```

The final line of code would produce an error, because finalScore only exists within the if code block

The concept of variables inside a code block is called **Scope**

Arithmetic Operators

Name	Example	
Addition	`int result = 1 + 2;`	`result = 3`
Subtraction	`result = result - 1; // 3 - 1`	`result = 2`
Multiplication	`result = result 10; //2 10`	`result = 20`
Division	`result = result / 5; //20 / 5`	`result = 4`
Modulus %	`result = result % 3; //remainder of (4 % 3)`	`result = 1`

Modulus(aka remainder) retains the remainder of two operands

Cheatography

Java Mastery - Fundamentals Cheat Sheet

by Bayan (Bayan.A) via cheatography.com/122738/cs/22947/

Operator Abbreviation

Original	Abbreviated
`result = result + 1;`	`result++;`
`result = result - 1;`	`result--;`
`result = result + 2;`	`result += 2;`
`result = result * 10;`	`result *= 10;`
`result = result / 3;`	`result /= 3;`
`result = result - 2;`	`result -= 2;`

if-then

Conditional Logic Checks a condition, executing code based on whether the condition(or expression) is true or false

Executing a section only if a particular test evaluates to true

No `;` after if parentheses

```
boolean isAlien = false;
if (isAlien == false) {
System.out.println("It is not an alien!");{
```

Use curly brackets if executing a code block

`==` tests if operands are identical	"Does `isAlien` equal or have the value `false` The expression `is isAlien false` is true
	it would return `false` if they are NOT equal

`if` keyword determines if the expression in the parenthesis evaluates to true, only then executing the next line of code.

Logical AND

Symbol: &&

Returns the boolean value **`true`** if both operands are `true` and returns `false` otherwise.

Example:

```
topScore = 80
secondTopScore = 60
if ((topScore > secondTopScore)
&& (topScore < 100))
```

Breakdown:

if ((topScore is greater than secondTop-Score) AND (topScore is less than 100))

if ((true) AND (true))

both operands are true, therefore the expression is true and will execute the next line

Truth Table:

p	q	p && q
T	T	T
T	F	F
F	T	F
F	F	F

Logical OR

Symbol: | | (two pipe characters)

Either or **both** conditions must be true for the boolean value to return `true`

Example:

```
topScore = 80
secondTopScore = 60
if ((topScore > 90) || (secon-
dTopScore <= 90))
```

Breakdown:

if ((topScore is greater than 90) OR (secondTopScore <= 90))

if ((`false`) OR (`true`))	One operand is `true`

Logical OR (cont)

boolean value returns `true` and will execute the next line.

True Table

p	q	p || q
T	T	T
T	F	F
F	T	T
F	F	T

Assignment and Equal to Operators

Assignment Operator =

Assigns value to variable

e.g. `int newValue = 50`

In an if expression, it will produce an error as the type required in the if condition is boolean

`if (newValue = 50);`	`Incompatible types. Required boolean Found: int`

However, if a boolean is in the if condition, the boolean value can be reassigned. No error will be produced

Equal to operator ==

Compares operand values are equal to eachother

e.g. `(50 == 50)`	e.g. (newValue == oldValue)
`boolean isCar = false; if (isCar = true)`	This turns isCar from `false` to `true`

By **Bayan** (Bayan.A)
cheatography.com/bayan-a/

Published 11th February, 2021.
Last updated 24th May, 2020.

Assignment and Equal to Operators (cont)

Normal:	Equivalent with NOT operator	Abbreviations:
`if (isCar == true)`	`if(isCar != false)`	`if (isCar)`
`if (isCar == false)`	`if(isCar != true)`	`if(!isCar)`
		Prevents mistakes and is more concise

Ternary Operator

A shortcut to assigning one of two values to a variable depending on a given condition

Like an **if-then-else** statement

Question mark comes after the condition

After the question mark, two values that can return are separated by a colon (:)

Takes 3 operands:	condition **?**	operand1 **:**	operand2
	Cond-ition we're testing against	**First value** to assign if first condition was true	**Second value** to assign if first condition was false

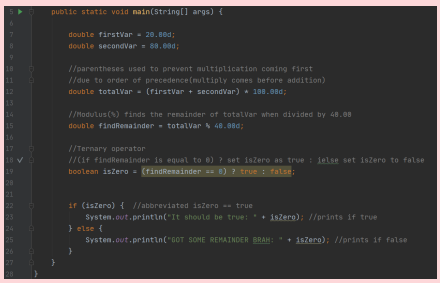
Example:

```
int age = 20
```

Ternary Operator (cont)

`boolean isOver18 =`	`(age == 20) ?`	`true :`	`false`
	is age equal to 20?	if it is, `isOver18= true`	if false, `isOver18 = false`

EXAMPLE CODE



```
public static void main(String[] args) {

    double firstVar = 20.00d;
    double secondVar = 80.00d;

    //parentheses used to prevent multiplication coming first
    //due to order of precedence(multiply comes before addition)
    double totalVar = (firstVar + secondVar) * 100.00d;

    //Modulus(%) finds the remainder of totalVar when divided by 40.00
    double findRemainder = totalVar % 40.00d;

    //Ternary operator
    //(if findRemainder is equal to 0) ? set isZero as true : ielse set isZero to false
    boolean isZero = (findRemainder == 0) ? true : false;


    if (isZero) {  //abbreviated isZero == true
        System.out.println("It should be true: " + isZero); //prints if true
    } else {
        System.out.println("GOT SOME REMAINDER BRAH: " + isZero); //prints if false
    }
}
```

Example code using most concepts outlined in this cheatsheet

See comments for explanation

By **Bayan** (Bayan.A)
cheatography.com/bayan-a/

Published 11th February, 2021.
Last updated 24th May, 2020.