# Java Mastery 2 Cheat Sheet

by Bayan (Bayan.A) via cheatography.com/122738/cs/31553/

## Java Structure

| | |
|---|---|
| **Package** | contains one or more classes. |
| **Class** | contains one or more fields and methods. |
| **Method** | contains declarations and statements. |
| **Statement** | contains declarations, statements, and expressions. |
| *These may contain comments:* | |
| **Single line:** | // |
| **Multi-line** | starts with / *and ends with* / |
| **Documentation notes** | starts with /* *and ends with* /, put before the definition of a variable, method or class |

## Data Types and Variable Declaration

| | |
|---|---|
| `int` | holds integer values (between −231 and 231 − 1). |
| `double` | holds floating-point numbers (i.e., numbers containing a decimal point). |
| `boolean` | holds a true or false value. |
| `char` | holds single characters(can also be used as numbers) |

## Data Types and Variable Declaration (cont)

| | |
|---|---|
| `float` | holds less accurate floating-point numbers. |
| `byte, short and long` | holds integers with fewer or more digits. |
| `String` | composed of zero or more chars, it is an object, not a primitive type. |

Variable declaration examples:

```
int age;      //without initial value
int count = 0;       //with initial value of 0
double distance = 37.95;
boolean isReadOnly = true;
String greeting = " Welcome to SP2";
String output Line;
```

## Enumeration types

An easy way to name a finite list of values that a variable can hold

Like declaring a new type, with a list of possible values

Can have any number of values, but you must include them all in the enum declaration

Can declare variables of the enumeration type:

## Enumeration types (cont)

Can use the comparison operator with them:

Example:

**Claring a new type with a list of possible values:**

```
public enum Filing Status { SINGLE, MARRIED, MARRIE D_F ILI - NG_ SEP ARATELY }
```

**Enumeration type variable declaration:**

```
Filing Status status = Filing St a tus.SI NGLE;
```

**With comparison operator:**

```
if (status == Filing Sta tus.SI - NGLE)
```

## Reading input

**Structure**

| | |
|---|---|
| Import Scanner class: | `import java.u til.Sc anner` |
| Create a scanner, assign it to a variable: | `Scanner scanner = new Scan n);` |
| | `new Scanne r(..)` creates a ne |
| | System.in says scanner is to take i keyboard |
| Request user to input number | `System.ou t.p rin t("P lea ");` |
| Read in the number: | `myNumber = scanne r.n ext` |
| Read in String: | String myString = scanner.nextLine |

## Reading input (cont)

| | |
|---|---|
| Read in double: | String myDouble = scanner.nextDouble(); |
| Read in char: | char myChar = scanner.next().charAt(0); |

## Printing

| | |
|---|---|
| Print and end line | System.out.println(something); |
| Print and doesn't end the line | System.out.print(something); |

## Assignment statements

alues can be assigned to variables by assignment statements.

Syntax: variable = expression

The expression must be of the same type as the variable.

The expression may be a simple value or it may involve computation

**When a variable is assigned a value, the old value is discarded and totally forgotten.**

## Methods

This is a named group of declarations and statements

They are called or invoked by naming it in a statement

Every method definition must specify a return type

## Methods (cont)

| | |
|---|---|
| Return type void: | used if nothing is to be returned |
| | plain `return` can be used |

If not `void`, return statements that specify the value to be returned must be supplied

Method call:

Request to an object to do something, or compute value

When calling a method, parameter types are not specified

Parameters of the type specified in the definition must be supplied

Method calls can be used as a statement

Methods that return a value may be used as part of an expression

## Arithmetic expressions

number literals (e.g., 42) and variables (e.g., x);

| | |
|---|---|
| + | indicate addition; |
| - | subtraction |
| * | multiplication |
| / | division |
| % | modulo(indicates remainder of an integer only division) |
| ( ) | indicate the order in which to do things. |

## Arithmetic expressions (cont)

An operation involving two ints results in an int.

When dividing one int by another, the fractional part of the result is thrown away, e.g., 14 / 5 gives 2 (and 14 % 5 gives 4).

Any operation involving a double results in a double, e.g., 3.7 + 1 gives 4.7 (int values are automatically converted to double where needed

## Boolean expressions

| | |
|---|---|
| < | less than |
| <= | less than or equals |
| == | equals |
| > | greater than |
| >= | >= |
| != | not equals |
| && | "and" true if and only if both operands are true |
| \|\| | "or" true if and only if at least one operand is true |
| ! | "not" reverses the truth value of its one operand |

Example:

```
(x > 0) && !(x > 99)
```

“x is greater than zero and is not greater than 99”

## Conditional expressions

| | |
|---|---|
| `condition ? expr1 : expr2` | Becomes expr1 if condition is true, otherwise expr2. |

Example:

`x < 0 ? -1 : 1`

“if x is less than zero, then -1, otherwise 1”

## String concatenation

| | |
|---|---|
| You can concatenate (join together) Strings with the + operator | fullName = firstName + " " + lastName; |
| ou can concatenate any value with a String and that value will automatically be turned into a String. | System.ou-t.println-("There are " + count + " apples."); |

## If statements

An if statement lets you choose whether or not to execute one statement, based on a boolean condition.

Condition **must** be boolean.

Syntax: if (boolean_condition) statement;

## If statements (cont)

Example: `if (x < 100) x = x + 1; // add 1 to x, but only if x is less than 100`

An if statement may have an optional else part, to be executed if the boolean condition is false.

Syntax: if (boolean_condition) statement else statement

Example

```
if (x >= 0 && x < limit) y = x / limit; else System.ou t.p rin
);
```

## Compound statements

Group multiple statements into a single statement by surrounding them with braces, `{ }`.

there is no semicolon after a compound statement

Braces can also be used around a single statement, or no statements at all (to form an “empty” statement).

It is good style to always use braces in the if part and else part of an if statement, even if they surround only a single statement.

Example:

```
if (score > 100) {
score = 100;
System.ou t.p rin tln ("score h
as been adjust ed");
}
```

## While Loops

A while loop will execute the enclosed statement as a boolean condition remains true.

Syntax: `while (boole an_ con ditio men`

he condition must be boolean.

If the condition never becomes false, the loop exits, and the program never stops.

Example:

```
n = 1;
while (n < 4) {
System.out.println(n + " squared is " + (n *
n));
n = n + 1;
}
```

Result:

1 squared is 1

2 squared is 4

3 squared is 9

## for loop

The for loop looks complicated, but is very handy.

Syntax: for (initialise ; test ; increment) statement

There is no semicolon after the increment.

The initialise part is done first and only once

Then, the test is performed, and, as long as it is true,

the statement is executed, and

the increment is executed

Initia-lise: define the loop variable with an assignment statement, or with a declaration and initialisation.

## for loop (cont)

Test, or condition: A boolean condition.

Example:

Print the numbers 1 through 10 and their squares:

```
for (int i = 1; i < 11; i++) {
System.out.println(i + " " + (i * i));
}
```

Print the squares of the first 100 integers, ten per line:

```
for (int i = 1; i < 101; i++) {
System.out.print(" " + (i * i));
if (i % 10 == 0)
System.out.println();
}
```

## When do you use each loop

| | |
|---|---|
| for loop | if you know ahead of time how many times you want to go through the loop. |
| while loop | in almost all other cases. |
| do-while loop | if you must go through the loop at least once before it makes sense to do the test. |

## break statement

Inside any loop, the break statement will immediately get you out of the loop.

If you are in nested loops, break gets you out of the innermost loop

It doesn't make any sense to break out of a loop unconditionally; you should do it only as the result of an if test

## break statement (cont)

break should not be the normal way to leave a loop

Use it when necessary, but don't overuse it.

Example:

```
for (int i = 1; i <= 12; i++) {
if (badEgg(i))
break;
}
```

## continue statemen

Inside any loop, the continue statement will jump right before the end of the loop body.

In a while or do-while loop, the continue statement will bring you to the test.

In a for loop, the continue statement will bring you to the increment, then to the test

## Multiway decisions

The if-else statement chooses one of two statements, based on the value of a boolean expression

The switch statement chooses one of several statements, based on the value

## switch statement

works with the byte, short, char, and int primitive data types

works with enumeration types, the String class, and a few special classes that wrap certain primitive types: Byte, Short, Character, and Integer.

Notice that colons ( : ) are used as well as semicolons.

The last statement in every case should be a break;

## switch statement (cont)

The default: case handles every value not otherwise handled.

```
public static void printStatus(FilingStatus status) {
switch (status) {
case SINGLE: // SINGLE rather than Filing-Status.SINGLE
System.out.print("Single filing");
break;
case MARRIED:
System.out.print("Married joint filing");
break;
case MARRIED_FILING_SEPARATELY:
System.out.print("Married separate filing");
break;
default:
System.out.print("Unexpected case");/ / better: throw an exception if code. needs to be updated to handle new case
}
}
```