J**AVA CHEATSHEET(https://github.com/manni2000)**

# Basics

Basic syntax and functions from the Java programming language.

## Boilerplate

class HelloWorld{

public static void main(String args[]){ System.out.println("Hello World");

}

}

## Showing Output

It will print something to the output console.

System.out.println([text])

## Taking Input

It will take string input from the user

import java.util.Scanner; //import scanner class

// create an object of Scanner class

Scanner input = new Scanner(System.in);

// take input from the user

String varName = input.nextLine();

# Primitive Type Variables

The eight primitives defined in Java are int, byte, short, long, float, double, boolean, and char those aren't considered objects and represent raw values.

## byte

byte is a primitive data type it only takes up 8 bits of memory.

age = 18;

## long

long is another primitive data type related to integers. long takes up 64 bits of memory.

viewsCount = 3\_123\_456L;

## float

We represent basic fractional numbers in Java using the float type. This is a single-precision decimal number. Which means if we get past six decimal points, this number becomes less precise and more of an estimate.

price = 100INR;

## char

Char is a 16-bit integer representing a Unicode-encoded character.

letter = 'A';

## boolean

The simplest primitive data type is boolean. It can contain only two values: true or false. It stores its value in a single bit.

isEligible = true;

## int

int holds a wide range of non-fractional number values.

var1 = 256;

## short

If we want to save memory and byte is too small, we can use short.

short var2 = 786;

# Comments

A comment is the code that is not executed by the compiler, and the programmer uses it to keep track of the code.

## Single line comment

// It's a single line comment

## Multi-line comment

/\* It's a multi-line comment

\*/

# Constants

Constants are like a variable, except that their value never changes during program execution.

final float INTEREST\_RATE = 0.04;

# Arithmetic Expressions

These are the collection of literals and arithmetic operators.

## Addition

It can be used to add two numbers

int x = 10 + 3;

## Subtraction

It can be used to subtract two numbers

int x = 10 - 3;

## Multiplication

It can be used to multiply add two numbers

int x = 10 \* 3;

## Division

It can be used to divide two numbers

int x = 10 / 3;

float x = (float)10 / (float)3;

## Modulo Remainder

It returns the remainder of the two numbers after division

int x = 10 % 3;

# Augmented Operators

## Addition assignment

var += 10 // var = var + 10

## Subtraction assignment

var -= 10 // var = var - 10

## Multiplication assignment

var \*= 10 // var = var \* 10

## Division assignment

var /= 10 // var = var / 10

## Modulus assignment

var %= 10 // var = var % 10

# Escape Sequences

It is a sequence of characters starting with a backslash, and it doesn't represent itself when used inside string literal.

## Tab

It gives a tab space

\t

## Backslash

It adds a backslash

\\

## Single quote

It adds a single quotation mark

\'

## Question mark

It adds a question mark

\?

## Carriage return

Inserts a carriage return in the text at this point.

\r

## Double quote

It adds a double quotation mark

\"

# Type Casting

Type Casting is a process of converting one data type into another

## Widening Type Casting

It means converting a lower data type into a higher

// int x = 45;

double var\_name = x;

## Narrowing Type Casting

It means converting a higher data type into a lower

double x = 165.48

int var\_name = (int)x;

# Decision Control Statements

Conditional statements are used to perform operations based on some condition.

## if Statement

if (condition) {

// block of code to be executed if the condition is true

}

## if-else Statement

if (condition) {

// If condition is True then this block will get executed

} else {

// If condition is False then this block will get executed

}

## if else-if Statement

if (condition1) {

// Codes

}

else if(condition2) {

// Codes

}

else if (condition3) {

// Codes

}

else {

// Codes

}

## Ternary Operator

It is shorthand of an if-else statement.

variable = (condition) ? expressionTrue : expressionFalse;

## Switch Statements

It allows a variable to be tested for equality against a list of values (cases).

switch(expression) { case a:

// code block break;

case b:

// code block break;

default:

// code block

}

# Iterative Statements

Iterative statements facilitate programmers to execute any block of code lines repeatedly and can be controlled as per conditions added by the coder.

## while Loop

It iterates the block of code as long as a specified condition is True

while (condition) {

// code block

}

## for Loop

for loop is used to run a block of code several times

for (initialization; termination; increment) { statement(s)

}

## for-each Loop

for(dataType item : array) {

...

}

## do-while Loop

It is an exit controlled loop. It is very similar to the while loop with one difference, i.e., the body of the do-while loop is executed at least once even if the condition is False

do {

// body of loop

} while(textExpression)

## Break statement

break keyword inside the loop is used to terminate the loop

break;

## Continue statement

continue keyword skips the rest of the current iteration of the loop and returns to the starting point of the loop

continue;

# Arrays

Arrays are used to store multiple values in a single variable

## Declaring an array

Declaration of an array

String[] var\_name;

## Defining an array

Defining an array

String[] var\_name = {''Harry", "Rohan", "Aakash"};

## Accessing an array

Accessing the elements of an array

String[] var\_name = {''Harry", "Rohan", "Aakash"}; System.out.println(var\_name[index]);

## Changing an element

Changing any element in an array

String[] var\_name = {''Harry", "Rohan", "Aakash"}; var\_name[2] = "Shubham";

## Array length

It gives the length of the array

System.out.println(var\_name.length);

## Loop through an array

It allows us to iterate through each array element

String[] var\_name = {''Harry", "Rohan", "Aakash"}; for (int i = 0; i < var\_name.length; i++) {

System.out.println(var\_name[i]);

}

## Multi-dimensional Arrays

Arrays can be 1-D, 2-D or multi-dimensional.

// Creating a 2x3 array (two rows, three columns) int[2][3] matrix = new int[2][3];

matrix[0][0] = 10;

// Shortcut

int[2][3] matrix = {

{ 1, 2, 3 },

{ 4, 5, 6 }

};

# Methods

Methods are used to divide an extensive program into smaller pieces. It can be called multiple times to provide reusability to the program.

## Declaration

Declaration of a method

returnType methodName(parameters) {

//statements

}

## Calling a method

Calling a method

methodName(arguments);

## Method Overloading

Method overloading means having multiple methods with the same name, but different parameters.

class Calculate

{

void sum (int x, int y)

{

System.out.println("Sum is: "+(a+b)) ;

}

void sum (float x, float y)

{

System.out.println("Sum is: "+(a+b));

}

Public static void main (String[] args)

{

Calculate calc = new Calculate();

calc.sum (5,4); //sum(int x, int y) is method is called. calc.sum (1.2f, 5.6f); //sum(float x, float y) is called.

}

}

## Recursion

Recursion is when a function calls a copy of itself to work on a minor problem. And the function that calls itself is known as the Recursive function.

void recurse()

{

... .. ...

recurse();

... .. ...

}

# Strings

It is a collection of characters surrounded by double quotes.

## Creating String Variable

String var\_name = "Hello World";

## String Length

Returns the length of the string

String var\_name = "Harry";

System.out.println("The length of the string is: " + var\_name.length());

## String Methods toUpperCase()

Convert the string into uppercase

String var\_name = "Harry";

System.out.println(var\_name.toUpperCase());

## toLowerCase()

Convert the string into lowercase

String var\_name = ""Harry"";

System.out.println(var\_name.toLowerCase());

## indexOf()

Returns the index of specified character from the string

String var\_name = "Harry";

System.out.println(var\_name.indexOf("a"));

## concat()

Used to concatenate two strings

String var1 = "Harry"; String var2 = "Bhai";

System.out.println(var1.concat(var2));

# Math Class

Math class allows you to perform mathematical operations.

## Methods max() method

It is used to find the greater number among the two

Math.max(25, 45);

## min() method

It is used to find the smaller number among the two

Math.min(8, 7);

## sqrt() method

It returns the square root of the supplied value

Math.sqrt(144);

## random() method

It is used to generate random numbers

Math.random(); //It will produce random number b/w 0.0 and 1.0

int random\_num = (int)(Math.random() \* 101); //Random num b/w 0 and 100

# Object-Oriented Programming

It is a programming approach that primarily focuses on using objects and classes. The objects can be any real-world entities.

## object

An object is an instance of a Class.

className object = new className();

## class

A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.

class ClassName {

// Fields

// Methods

// Constructors

// Blocks

}

## Encapsulation

Encapsulation is a mechanism of wrapping the data and code acting on the data together as a

single unit. In encapsulation, the variables of a class will be hidden from other classes and can be accessed only through the methods of their current class.

public class Person {

private String name; // using private access modifier

// Getter

public String getName() { return name;

}

// Setter

public void setName(String newName) { this.name = newName;

}

}

## Inheritance

Inheritance can be defined as the process where one class acquires the properties of another. With the use of inheritance the information is made manageable in a hierarchical order.

class Subclass-name extends Superclass-name

{

//methods and fields

}

## Polymorphism

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

// A class with multiple methods with the same name public class Adder {

// method 1

public void add(int a, int b) { System.out.println(a + b);

}

// method 2

public void add(int a, int b, int c) { System.out.println(a + b + c);

}

// method 3

public void add(String a, String b) { System.out.println(a + " + " + b);

}

}

// My main class

class MyMainClass {

public static void main(String[] args) {

Adder adder = new Adder(); // create a Adder object adder.add(5, 4); // invoke method 1

adder.add(5, 4, 3); // invoke method 2

adder.add("5", "4"); // invoke method 3

}

}

# File Operations

File handling refers to reading or writing data from files. Java provides some functions that allow us to manipulate data in the files.

## canRead method

Checks whether the file is readable or not

file.canRead()

## createNewFile method

It creates an empty file

file.createNewFile()

## canWrite method

Checks whether the file is writable or not

file.canWrite()

## exists method

Checks whether the file exists

file.exists()

## delete method

It deletes a file

file.delete()

## getName method

It returns the name of the file

file.getName()

## getAbsolutePath method

It returns the absolute pathname of the file

file.getAbsolutePath()

## length Method

It returns the size of the file in bytes

file.length()

## list Method

It returns an array of the files in the directory

file.list()

## mkdir method

It is used to create a new directory

file.mkdir()

## close method

It is used to close the file

file.close()

## To write something in the file

import import

java.io.FileWriter; // Import the FileWriter class

java.io.IOException; // Import the IOException class to handle errors

public class WriteToFile {

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

FileWriter myWriter = new FileWriter("filename.txt");

myWriter.write("Laal Phool Neela Phool, Harry Bhaiya Beautiful"); myWriter.close();

System.out.println("Successfully wrote to the file.");

} catch (IOException e) {

System.out.println("An error occurred."); e.printStackTrace();

}

}

}

# Exception Handling

An exception is an unusual condition that results in an interruption in the flow of the program.

## try-catch block

try statement allow you to define a block of code to be tested for errors. catch block is used to handle the exception.

try {

// Statements

}

catch(Exception e) {

// Statements

}

## finally block

finally code is executed whether an exception is handled or not.

try {

//Statements

}

catch (ExceptionType1 e1) {

// catch block

}

finally {

// finally block always executes

}