CCBP TECH 4.0 INTENSIVE
With Industry-Ready Certification (IRC)

Java Summary Cheat Sheet - 2

https://www.ccbp.in/

Loops

Loops: It allow us to execute a block of code several times.

while loop: It allows us to execute a block of code several times as long as the condition evaluates to true

```
int counter = 0;
while (counter < 2) {
    System.out.println(counter);
    counter = counter + 1;
}
```

```
// Output is:
0
1
```

do...while loop: It is similar to the while loop. The only difference is that in the do...while loop, the check is performed after the do...while block of code has been executed.

```
do {
    System.out.println("Line Executed");
} while (3 > 10);
```

```
// Output is:
Line Executed
```

for loop: It is used to execute a block of code a certain number of times. It is generally used where the number of iterations is known.

```
for (int i = 0; i < 2; i++ ) {
    System.out.println(i);
}
```

```
// Output is:
0
1
```

for-each loop: It is used to iterate over arrays and various collections.

```
String[] players = {"Bryant", "Wade"};
for(String name: players) {
    System.out.println(name);
```

```
// Output is:
Bryant
Wade
```

Arrays

Array: In Java, an array is an object used to store similar data type values. In Java, the number of elements that an array can store is always fixed.

Accessing Array Elements: We can access the elements of an `array` using these index values.

```
int[] arr = {12, 4, 5, 2, 5};
System.out.println(arr[0]); //12
```

Creating an Array using a new Keyword: We can also create the `array` using `new` keyword. We can create an `array` of required length and later assign the values.

```
int[] arr;
arr = new int[3];
```

Printing an Array: The `Arrays` class provides a method `toString()`, which can be used to print the `array` in Java.

```
int[] arr = {12, 4, 5, 2, 5};
System.out.println(Arrays.toString(arr));
```

```
// Output is:
[12, 4, 5, 2, 5]
```

Iterating Over an Array: We can use loops to iterate over an array.

Using for Loop

```
int[] arr = {12, 4, 5};
for (int i = 0; i < arr.length; i++)
    System.out.println(arr[i]);
```

```
// Output is:
12
4
5
```

Using for-each Loop

```
int[] arr = {12, 4, 5};
for(int element: arr)
    System.out.println(element);
```

```
// Output is:
12
4
5
```

Length of an Array: In Java, we can find the array length by using the attribute `length`.

```
int[] arr = {12, 4, 5, 2, 5, 7};
System.out.println(arr.length); // 6
```

Array Concatenation: Joining two arrays is called Array Concatenation. In Java, the System class contains a method named `arraycopy()` to concatenate two arrays.

```
int[] arr1 = {12, 4, 5, 2, 5};
int[] arr2 = {6, 10,11,6};
int arr1Len = arr1.length;
int arr2Len = arr2.length;
int[] concatenatedArr  = new int[arr1Len + arr2Len];
System.arraycopy(arr1, 0, concatenatedArr,  0, arr1Len);
System.arraycopy(arr2, 0, concatenatedArr,  arr1Len, arr2Len);
System.out.println(Arrays.toString(concatenatedArr));
```

```
// Output is:
[12, 4, 5, 2, 5, 6, 10, 11, 6]
```

Array Slicing: It is a method of obtaining a subarray of an array. We can get the subarray from an array using `Arrays.copyOfRange()` method.

```
int[] originalArr = { 1, 2, 3, 4, 5 };
int startIndex = 2;
int endIndex = 4;
int[] slicedArr = Arrays.copyOfRange(originalArr, startIndex, endIndex);
System.out.println(Arrays.toString(slicedArr));
```

```
// Output is:
[3, 4]
```

Multi-dimensional Array: It consists of an array of arrays.
A two-dimensional array is a collection of one-dimensional arrays.

```
int[][] arr = {{12, 4, 5}, {16, 18, 20}};
System.out.println(arr[1][2]); // 20
```

Reversing Arrays: The `Collections.reverse()` method is used for reversing the elements present in the array passed as an argument to this method.

```
Integer[] arr = {3, 30, 8, 24};
Collections.reverse(Arrays.asList(arr));
System.out.println(Arrays.toString(arr)); // [24, 8, 30, 3]
```

Sorting Arrays: The `Arrays.sort()` method can be used to sort an Array.
The sorting can be done in two different ways:

Ascending Order

```
int[] arr = {3, 1, 2};
Arrays.sort(arr);
System.out.println(Arrays.toString(arr)); // [1, 2, 3]
```

Descending Order: The reverse sorting is done by passing `Collections.reverseOrder()` as an argument to the `Arrays.sort()` method.

```
Integer[] arr = {3, 1, 2};
Arrays.sort(arr, Collections.reverseOrder());
System.out.println(Arrays.toString(arr)); // [3, 2, 1]
```

Methods

Methods: Java doesn't have independent functions because every Java function belongs to a class and is called a Method.

Method Declaration: A Method must be declared before it is used anywhere in the program.

```
accessModifier static returnType methodName() {
    // method body
}
```

Calling a Method: The block of code in the methods is executed only when the method is called.

```
static void greet() {
    System.out.println("Hello, I am in the greet method");
}
```

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

```
// Output is:
Hello, I am in the greet method
```

Returning a Value: We can pass information from a method using the `return` keyword. In Java, `return` is a reserved keyword.

```java
static String greet() {
    return "Hello, I am in the greet method";
}
public static void main(String[] args){
    System.out.print(greet());
}
```

```
// Output is:
Hello, I am in the greet method
```

Method with Parameters: The information can be passed to methods as arguments in Java.

```java
static void greet(String username){
    System.out.println("Hello "+ username);
}
public static void main(String[] args){
    String name = "Rahul";
    greet(name);
}
```

```
// Output is:
Hello Rahul
```

Method Overloading: If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

```java
static int addition(int a, int b) {
    return a + b;
}
static double addition(double a, double b) {
    return a + b;
}
public static void main(String[] args) {
    System.out.println(addition(20, 34)); // 54
    System.out.println(addition(45.6, 32.3)); // 77.9
}
```

Passing Mutable Objects: when mutable objects are passed as method arguments, the changes done to the object inside the method will affect the original object.

```java
static void twice(int[] arr2 ) {
    for(int i = 0; i < arr2.length ; i++)
        arr2[i] = 2 * arr2[i];
}
public static void main(String[] args) {
    int[] arr1 = {36,43,10,112,66,18};
    twice(arr1);
    System.out.println(Arrays.toString(arr1));
}
```

```
// Output is:
[72, 86, 20, 224, 132, 36]
```

Passing Immutable Objects: Immutable objects are passed as method arguments, the changes done to the object inside the method will not affect the original object.

```
static void fullName(String str2) {
    str2 = "William Smith";
    System.out.println("Inside fullName() method: " + str2);
}
public static void main(String[] args) {
    String str1 = "William";
    fullName(str1);
    System.out.println("Inside main() method: " + str1);
}
```

```
// Output is:
Inside fullName() method: William Smith
Inside main() method: William
```

Recursion: Recursion is a process in which a method calls itself in the process of its execution. A recursive method terminates when a condition is met. This condition is also called the Base Case.

```
static int factorial(int num) {
    if (num == 1)
        return 1;
    return num * factorial(num - 1);
}
public static void main(String[] args) {
    System.out.println(factorial(5)); // 120
}
```

Nested Loops

Nested Loops: If a loop exists inside the body of another loop, it is called a nested loop. The inner loop will be executed one time for each iteration of the outer loop.

Example 1:

```
for (int i = 0; i < 2; i = i + 1) {
    System.out.println("Outer: " + i);
    for (int j = 0; j < 2; j = j + 1)
        System.out.println("  Inner: " + j);
}
System.out.println("After nested for loops");
```

```
// Output is:
Outer: 0
  Inner: 0
  Inner: 1
Outer: 1
  Inner: 0
  Inner: 1
After nested for loops
```

Example 2:

```
for (int i = 0; i < 2; i = i + 1) {
    System.out.println("Outer For Loop: " + i);
    int counter = 0;
    while (counter < 2) {
        System.out.println("  Inner While Loop: " + counter);
        counter = counter + 1;
    }
}
```

```
//Output is:
Outer For Loop: 0
  Inner While Loop: 0
  Inner While Loop: 1
Outer For Loop: 1
  Inner While Loop: 0
  Inner While Loop:1
```

Loop Control Statements: The statement which alters the flow of control of a loop is called a Loop Control Statement.

Name	Usage
Break	break keyword is used to stop the execution of the loop.
Continue	continue keyword is used to skip the current execution of the loop.
Break (in nested loops)	break keyword in the inner loop stops the execution of the inner loop.

Big Integers

BigInteger: The `BigInteger` is the class used for mathematical operations which involve very big integer calculations that are outside the limit of all available primitive data types.

```
String str = "51090942171709440000";
BigInteger bigNum = new BigInteger(str);
System.out.println(bigNum);
```

```
// Output is:
51090942171709440000
```

BigInteger Methods

Method	Syntax	Usage
add()	bigNum1.add(bigNum2);	performs the addition for the given two BigIntegers and returns the sum.
subtract()	bigNum1.subtract(bigNum2);	performs the subtraction for the given two BigIntegers and returns the difference.
multiply()	bigNum1.multiply(bigNum2);	performs the multiplication for the given two BigIntegers and returns the product.
divide()	bigNum1.divide(bigNum2);	performs the division for the given two BigIntegers and returns the quotient.
pow()	bigNum.pow(exponent);	performs the exponentiation operation and returns the result.
abs()	bigNum.abs();	returns a value that is equal to the absolute value of the given BigInteger.

Converting Integers to BigInteger: The `valueOf()` method can be used to convert integers values to a `BigInteger`.

```
int num = 2023;
long longNum = 435468567463L;
BigInteger bigNum1 = BigInteger.valueOf(num);
BigInteger bigNum2 = BigInteger.valueOf(longNum);
```

```java
System.out.println(bigNum1); // 2023
System.out.println(bigNum2); // 435468567463
```

Converting BigInteger to Integers or String: We can convert BigInteger to int, long, and String data types.

```java
String str = "45900";
BigInteger bigNum1 = new BigInteger(str);
str = bigNum1.toString();
long longNum = bigNum1.longValue();
float floatNum = bigNum1.floatValue();
double doubleNum = bigNum1.doubleValue();
System.out.println(str); // 45900
System.out.println(longNum); // 45900
System.out.println(floatNum); // 45900.0
System.out.println(doubleNum); // 45900.0
```

BigInteger Constants: The BigInteger class defines some constants for the ease of initialization.

BigInteger.ZERO: The BigInteger constant for 0
BigInteger.ONE: The BigInteger constant for 1
BigInteger.TWO: The BigInteger constant for 2
BigInteger.TEN: The BigInteger constant for 10