

# Java Cheatsheet

## Variables

### Annotated Examples in Java

```
double pi = 3.14159;
double copyPi = pi;

char someCharacter = 'A'; // Requires character to be in ''.
boolean alive = true;
boolean[] homeTeam = {alive, true, false, false};
boolean[] enemyTeam = new boolean [4];
```

Annotations:

- type: double, char, boolean, boolean[]
- name: pi, copyPi, someCharacter, alive, homeTeam, enemyTeam
- assignment operator: =
- literal: 3.14159, 'A', true, false
- size: 4
- initializer list: {alive, true, false, false}

Note: With Python bools, "true" and "false" must be capitalized into "True" and "False"!

## Scanner

```
1 import java.util.Scanner; // Required to use the scanner.
2
3 // ...
4
5 Scanner scan = new Scanner(System.in); // Initialize a scanner.
6
7 int num = scan.nextInt(); // `nextInt` gets an integer.
8 double num2 = scan.nextDouble(); // `nextDouble` gets a floating-point.
9 String s = scan.nextLine(); // `nextLine` gets a string.
10
11 scan.close() // Remember to close the scanner.
```

## Operators

Operators	Precedence
postfix	<i>expr++ expr--</i>
unary	<i>++expr --expr +expr -expr ~ !</i>
multiplicative	<i>* / %</i>
additive	<i>+ -</i>
shift	<i>&lt;&lt; &gt;&gt; &gt;&gt;&gt;</i>
relational	<i>&lt; &gt; &lt;= &gt;= instanceof</i>
equality	<i>== !=</i>
bitwise AND	<i>&amp;</i>
bitwise exclusive OR	<i>^</i>
bitwise inclusive OR	<i> </i>
logical AND	<i>&amp;&amp;</i>
logical OR	<i>  </i>
ternary	<i>? :</i>
assignment	<i>= += -= *= /= %= &amp;= ^=  =</i> <i>&lt;&lt;= &gt;&gt;= &gt;&gt;&gt;=</i>

## Casting

```
1  double num = 3.14159;
2  int num2 = (int)num; // `num` casted to an integer.
3
4  // Casting to a String is a bit different:
5  String numString = Double.toString(num); // `Double` is uppercase!
6  String numString2 = Integer.toString(num2);
7
8  // Casting from a String to other data types:
9  String numString3 = "123";
10 int num3 = Integer.valueOf(numString3);
```

## Arrays

```
1  int[] arr = {5, 6, 7};
2  int i = arr[0]; // First element: 5
3  int j = arr[1]; // Second element: 6
4  // ...
5
6  int size = arr.length; // Number of elements in array.
7
8  int[][] matrix = { // Multidimensional
9      {2, 4, 6},
10     {8, 10, 12},
11     {14, 16, 18}
12 };
```

## Strings

```
1  String s = "Hello, world!";
2  char c = s.charAt(7); // Get character at index 7.
3
4  // Get a substring from string `s`, starting at index 0 and stopping right before index 5.
5  String sub = s.substring(0, 5);
6
7  String s1 = "Hello, ";
8  String s2 = "world!";
9  String s3 = s1 + s2; // → s3 is "Hello, world!"
```

## Basic Escape Sequences

`\"`, `\'`, `\\`, `\n`, `\t`, ...

# Functions

'final' in this context indicates a mathematical constant

Global Variable

```
// Global variable used by 'volumeOfCylinder'.
private static final double pi = 3.14159;
```

Function with return value and parameters

```
// Function with return value.
private static double volumeOfCylinder(double radius, double height) {
    double baseArea = pi * radius * radius;
    double volume = baseArea * height;
    return volume;
}
```

Return Value

Function Body

Return type 'void' indicates no return value

Function Name

Function with no return value or parameters

```
// Function without return value.
private static void cow() {
    // Note: we are using escape sequences here so that we can use backslash (\) characters inside strings.
    System.out.println("^__^");
    System.out.println("(oo)\\_____");
    System.out.println("(__)\\    )\\//\\");
    System.out.println("  ||---w |");
    System.out.println("  ||    ||");
}
```

Arguments --> Parameters

Function Call

Calling Functions

```
double r = 27;
double volume = volumeOfCylinder(r, 10);
cow();
```

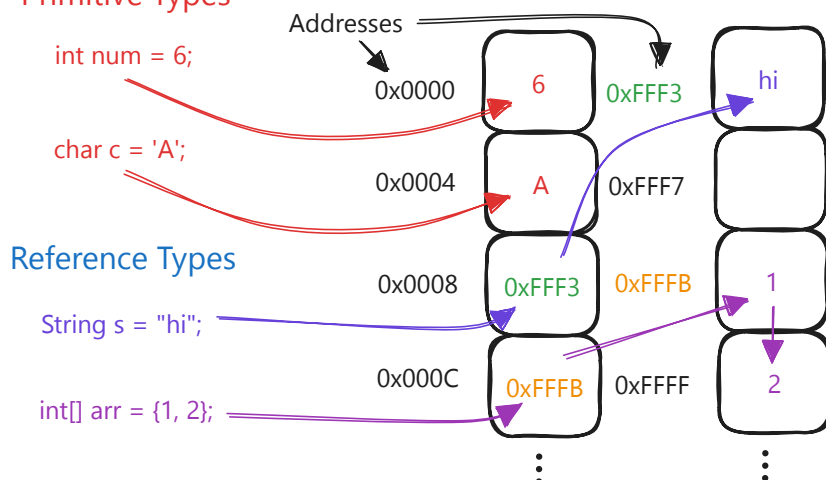
Calling a void function

Return value is assigned to new variable

## By Reference

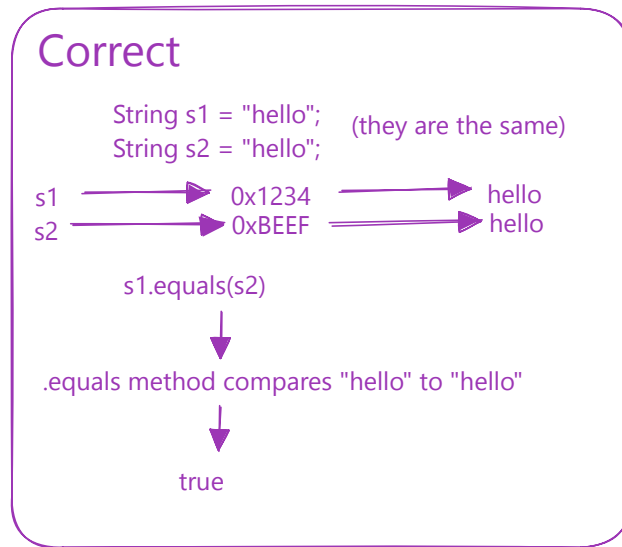
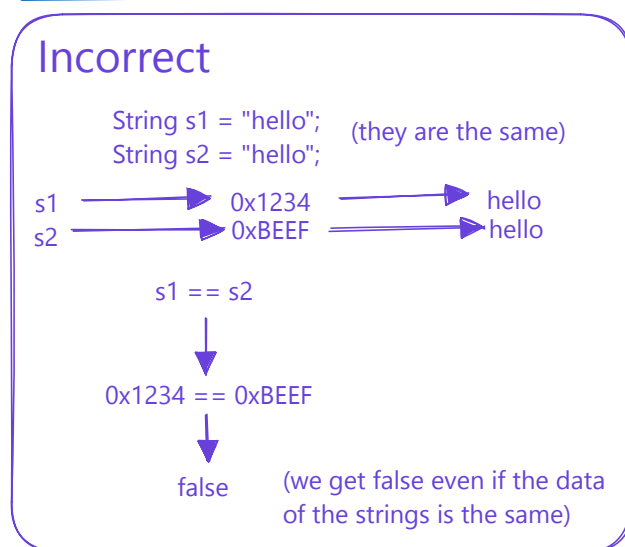
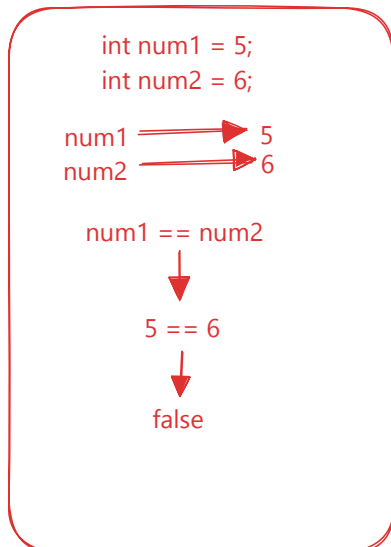
Primitive Types

Computer Memory (32-bit)



Comparing Primitive Types

Comparing Reference Types



## Comparison Operators

- `==` → equality operator, tests if two values are equal
- The single equals sign `=` is already used as the assignment operator!
- `!=` → inequality operator, tests if two values are not equal
- `<` → less than
- `>` → greater than
- `<=` → less than or equal to
- `>=` → greater than or equal to

## Logical Operators

- `&&` → AND operator
- `||` → OR operator
- `!` → NOT operator

## Conditionals

```
1  if (/* boolean expression */) {
2      // ...
3  } else if (/* boolean expression */) {
4      // ...
5  } else {
6      // ...
7  }
```

## Loops

```
1  int num = 0;
2  while (num < 10) {
3      System.out.print(num + " ");
4      ++num; // This is the prefix increment operator.
5      // It does the same thing as num = num + 1.
6  }
```

```
1  //   init.      cond.    update
2  for (int i = 0; i ≤ 20; i += 2) {
3      System.out.print(i + " ");
4  }
```

```
1  String[] arr = {"Alice", "Bob", "Charlie"};
2  for (String s : arr) {
3      System.out.print(s + " ");
4  }
```

```
1  String input;
2  do {
3      System.out.print("Please enter \"confirm\": ");
4      input = scan.nextLine();
5  } while (!input.equals("confirm"));
```

Loop statements: `break` , `continue`

## Other Data Structures

```
1  import java.util.Arrays;
2  import java.util.ArrayList;
3
4  // ...
5
6  ArrayList<Integer> nums = new ArrayList<>(Arrays.asList(2, 4, 6, 8));
7  int element = nums.get(2); // Gets the integer at index 2 → 6
8  // Add an element to the end of the array-list.
9  nums.add(5);
10
11 // Insert an element at index 3.
12 nums.add(3, 6);
13
14 // Remove an element at index 1.
15 nums.remove(1);
16
17 // Get the number of elements in the linked-list.
18 int count = nums.size();
```

```
1  import java.util.Arrays;
2  import java.util.LinkedList;
3
4  // ...
5
6  LinkedList<Integer> nums = new LinkedList<>(Arrays.asList(2, 4, 6, 8));
7  int element = nums.get(2); // Gets the integer at index 2 → 6
8  System.out.println(element);
9  // Other methods.
10 element = nums.getFirst();
11 System.out.println(element);
12 element = nums.getLast();
13 System.out.println(element);
14
15 // Add an element to the beginning and end of the linked-list.
16 nums.addFirst(5);
17 nums.addLast(5);
18
19 // Insert an element at index 3.
20 nums.add(3, 6);
21
22 // Remove an element at index 1.
23 nums.remove(1);
24
25 // Remove the first and last elements.
26 nums.removeFirst();
27 nums.removeLast();
28
29 // Get the number of elements in the linked-list.
30 int count = nums.size();
```

```

1  import java.util.HashMap;
2
3  // ...
4
5  // Key type: `String`
6  // Value type: `Integer` / `int`
7  HashMap<String, Integer> map = new HashMap<>();
8
9  // Add / change a key-value pair.
10 map.put("Alice", 2000);
11 map.put("Bob", 1900);
12 map.put("Charlie", 1950);
13
14 map.put("Alice", 3000); // Changes the Alice's existing balance.
15
16 // Check if the map has a specific key.
17 System.out.println("Contains Alice: " + map.containsKey("Alice"));
18 System.out.println("Contains Dave: " + map.containsKey("Dave"));
19
20 // Get a value associated with a key.
21 System.out.println("Alice's Balance: " + map.get("Alice"));
22 System.out.println("Dave's Balance: " + map.get("Dave")); // `null` since Dave is not in the map.
23
24 // Get the number of key-value pairs in the hashmap.
25 int count = map.size();

```

## Exception Handling

```

1  // Start of the `try` block!
2  try {
3
4      // ArrayIndexOutOfBoundsException
5      int[] nums = {1, 2, 3};
6      System.out.println(nums[10]); // Index 10 is out-of-bounds.
7
8      // ArithmeticException
9      int n = 10 / 0; // Dividing by 0 is illegal.
10
11     // NullPointerException
12     String s = null;
13     s.length(); // Don't use an object which is `null`.
14
15     // Your own exception!
16     throw new Exception("MY EXCEPTION");
17
18 // End of the `try` block. Start of the `catch` block!
19 } catch (Exception e) {
20     System.out.println("A problem occurred!");
21
22     // You can also print the details of the exception.
23     System.out.println(e.toString());
24     System.out.println(e.getMessage());
25 }

```