

# ???? ??

- [Java Coding Test Cheatsheet](#)
- [Java 11 12 13 14 15](#)
- [Java 11 \(Expression\)](#)
- [Java 11 12 13 14 15 \(Escape Sequences\)](#)
- [Java 11 12 13 14 15 \(11 12 13 14 15 , printf, format\)](#)
- [Java 11 \(Regular Expression\)](#)
- [Java Arrays.sort\(\) 11](#)

# Java Coding Test Cheatsheet

---

## 0. Frequently Used Libraries

```
import java.util.*;      // Data structures
import java.io.*;        // Fast I/O
import java.math.*;      // BigInteger, BigDecimal
```

---

## 1. Variable & Array Declaration

```
String[] arr1 = new String[5];
int[] arr2 = {1, 2, 3};
int N = 3;
int[] arr3 = new int[N];
```

---

## 2. Arrays Utility

```
Arrays.sort(arr);                // Ascending
Arrays.sort(arr, Collections.reverseOrder()); // Descending
Arrays.sort(arr, 0, 4);          // Partial sort
Arrays.binarySearch(arr, 2);      // Binary search
List<String> list = Arrays.asList(arr1); // Convert to List
int[] tmp = Arrays.copyOfRange(arr, 0, 3); // Slice
```

---

## 3. length / length() / size()

```
arr.length    // Arrays
str.length()  // String
list.size()    // Collections
```

---

## 4. String Handling

```
str.split(" ");
str.substring(0, 5);
str.charAt(i);
str.toUpperCase();
str.toLowerCase();
String[] letters = str.split("");
String newStr = str.substring(0,4) + "X" + str.substring(5);
```

---

## 5. HashMap

```
Map<String, Integer> map = new HashMap<>();
map.put("key", 1);
map.get("key");
map.containsKey("key");
map.getOrDefault("key", 0);
for (String k : map.keySet()) map.get(k);
```

---

## 6. ArrayList

```
List<String> list = new ArrayList<>();
list.add("a");
list.set(0, "b");
list.remove("b");
list.contains("a");
list.indexOf("a");
```

---

## 7. Queue (LinkedList)

```
Queue<Integer> q = new LinkedList<>();  
q.offer(1);  
q.poll();  
q.peek();  
q.clear();  
q.isEmpty();
```

---

## 8. PriorityQueue

```
PriorityQueue<Integer> pq = new PriorityQueue<>();  
PriorityQueue<Integer> maxPq = new PriorityQueue<>(Collections.reverseOrder());
```

---

## 9. Math

```
Math.max(a, b);  
Math.min(a, b);  
Math.abs(x);  
Math.ceil(x);  
Math.floor(x);  
Math.round(x);  
String.format("%.2f", d); // Round to 2 decimal  
Math.pow(a, b);  
Math.sqrt(x);
```

---

## 10. HashSet

```
Set<String> set = new HashSet<>();  
set.add("a");  
set.remove("a");  
set.contains("a");
```

---

# 11. Stack

```
Stack<Integer> stack = new Stack<>();  
stack.push(1);  
stack.pop();  
stack.peek();
```

---

# 12. Deque (ArrayDeque)

```
Deque<Integer> dq = new ArrayDeque<>();  
dq.addFirst(1);  
dq.addLast(2);  
dq.pollFirst();  
dq.pollLast();
```

---

# 13. TreeSet

```
TreeSet<Integer> ts = new TreeSet<>();  
ts.add(5);  
ts.first();  
ts.last();  
ts.lower(5);  
ts.higher(5);
```

---

# 14. TreeMap

```
TreeMap<String, Integer> tm = new TreeMap<>();  
tm.put("apple", 3);  
tm.firstKey();  
tm.lastKey();
```

---

# 15. Fast I/O

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
StringTokenizer st = new StringTokenizer(br.readLine());
int a = Integer.parseInt(st.nextToken());
String[] parts = br.readLine().split(" ");

BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));
bw.write("Hello\n");
bw.flush();
bw.close();
```

# 16. DFS

```
void dfs(int node) {
    visited[node] = true;
    for (int next : graph.get(node)) {
        if (!visited[next]) dfs(next);
    }
}
```

# 17. BFS

```
Queue<Integer> q = new LinkedList<>();
q.offer(start);
visited[start] = true;
while (!q.isEmpty()) {
    int cur = q.poll();
    for (int next : graph.get(cur)) {
        if (!visited[next]) {
            q.offer(next);
            visited[next] = true;
        }
    }
}
```

---

## 18. Sorting

```
list.sort(Comparator.naturalOrder());  
list.sort(Comparator.reverseOrder());  
people.sort(Comparator.comparingInt(p -> p.age));
```

---

## 19. DP (Fibonacci)

```
int[] dp = new int[N+1];  
dp[0] = 0; dp[1] = 1;  
for (int i = 2; i <= N; i++) dp[i] = dp[i-1] + dp[i-2];
```

---

## 20. Union-Find

```
int find(int x) {  
    if (x != parent[x]) parent[x] = find(parent[x]);  
    return parent[x];  
}  
  
void union(int a, int b) {  
    a = find(a); b = find(b);  
    if (a != b) parent[b] = a;  
}
```

---

## 21. Bitmask

```
int bit = 0;  
bit |= (1 << 3);  
bit &= ~(1 << 3);  
boolean on = (bit & (1 << 3)) != 0;  
Integer.bitCount(bit);
```

---

## 22. Greedy

```
int[] coins = {500, 100, 50, 10};
int count = 0;
for (int coin : coins) {
    count += target / coin;
    target %= coin;
}
```

## 23. Prefix Sum

```
int[] prefix = new int[N+1];
for (int i = 0; i < N; i++) prefix[i+1] = prefix[i] + arr[i];
```

## 24. Sliding Window

```
int sum = 0;
for (int i = 0; i < k; i++) sum += arr[i];
int max = sum;
for (int i = k; i < N; i++) {
    sum += arr[i] - arr[i-k];
    max = Math.max(max, sum);
}
```

## 25. Sieve of Eratosthenes

```
boolean[] isPrime = new boolean[N+1];
Arrays.fill(isPrime, true);
for (int i = 2; i*i <= N; i++) {
    if (isPrime[i]) {
        for (int j = i*i; j <= N; j += i) isPrime[j] = false;
    }
}
```



## 26. Dijkstra

```
PriorityQueue<int[]> pq = new PriorityQueue<>(Comparator.comparingInt(a -> a[1]));
int[] dist = new int[N+1];
Arrays.fill(dist, INF);
dist[start] = 0;
pq.offer(new int[]{start, 0});
```

## 27. Tree / LCA

```
void dfs(int node, int par, int d) {
    parent[node] = par;
    depth[node] = d;
    for (int next : tree.get(node)) {
        if (next != par) dfs(next, node, d+1);
    }
}
```

## 28. Coding Test Patterns

- Use BufferedReader & BufferedWriter
- HashMap, HashSet, TreeMap, PriorityQueue
- Always validate inputs
- Use boolean[], Set, or visited[][] for visited states
- Practice common patterns (DFS/BFS, DP, Greedy, Two Pointers)

## 29. Backtracking (Permutation, Combination)

```
void permute(List<Integer> list, boolean[] used) {
    if (list.size() == N) {
        System.out.println(list);
        return;
    }
}
```

```
for (int i = 0; i < N; i++) {  
    if (!used[i]) {  
        used[i] = true;  
        list.add(arr[i]);  
        permute(list, used);  
        list.remove(list.size() - 1);  
        used[i] = false;  
    }  
}  
}
```

## 30. Binary Search / Parametric Search

```
int left = 1, right = maxVal;  
while (left <= right) {  
    int mid = (left + right) / 2;  
    if (condition(mid)) {  
        answer = mid;  
        right = mid - 1;  
    } else {  
        left = mid + 1;  
    }  
}
```

## 31. Recursion Optimization

- Tail recursion is not optimized in Java.
- Prefer loops for heavy stack recursion.
- Consider memoization or iterative conversion.

## 32. String Algorithms (KMP)

```
int[] makeTable(String pattern) {  
    int[] table = new int[pattern.length()];  
    int j = 0;  
    for (int i = 1; i < pattern.length(); i++) {
```

```
        while (j > 0 && pattern.charAt(i) != pattern.charAt(j)) j = table[j-1];
        if (pattern.charAt(i) == pattern.charAt(j)) table[i] = ++j;
    }
    return table;
}
```

## 33. Permutations / Combinations / Bitmask

```
// Bitmask Combination
for (int mask = 0; mask < (1 << N); mask++) {
    for (int i = 0; i < N; i++) {
        if ((mask & (1 << i)) != 0) {
            // element i is selected
        }
    }
}
```

## 34. BigInteger / BigDecimal

```
BigInteger a = new BigInteger("12345678901234567890");
BigInteger b = new BigInteger("98765432109876543210");
BigInteger sum = a.add(b);
BigDecimal dec = new BigDecimal("1234.5678");
BigDecimal result = dec.setScale(2, RoundingMode.HALF_UP);
```

# Java ?? ?? ?? ??? ??

## 1. ?? ??: Arrays.equals()

```
Arrays.equals(array1, array2);
```

??	??
?? ??	?? ?? ?? ?? ?? ?? ??
??	true(?? ), false(?? )
??	== ?? .equals() ?? ?? ?? ?? ??

?? ?? :

```
int[] a = {1, 2, 3};
int[] b = {1, 2, 3};
System.out.println(Arrays.equals(a, b)); // ? true
System.out.println(a == b);             // ? false (?? ??)
```

## 2. ?? ??: Arrays.copyOf()

```
Arrays.copyOf(????, ?????);
```

?? ?? ???? ?? ?? ?? ?? ?? ?? .  
????? ???? , ?? ?? ???? .

?? ?? :

```
int[] original = {1, 2, 3, 4};
int[] copied = Arrays.copyOf(original, 2); // [1, 2]
```

## ? ?? ??: ?? ?? ? 0?? ??

```
int[] extended = Arrays.copyOf(original, 6); // [1, 2, 3, 4, 0, 0]
```

---

### 3. ?? ??: Arrays.copyOfRange()

```
Arrays.copyOfRange(原数组, 起始索引, 结束索引);
```

原数组 起始索引 结束索引 新数组。

新数组 起始索引 结束索引 (原数组 索引) 索引。

原数组 起始索引 结束索引 新数组。

代码示例：

```
int[] sub = Arrays.copyOfRange(original, 1, 3); // [2, 3]
```

---

### 4. ?? ??: Arrays.sort()

```
Arrays.sort(数组);
```

数组 数组 数组 数组 (in-place 排序, 返回 void)

void 数组 数组 数组 数组 数组 数组 数组。

数组 数组 (数组 : 数组, 数组 : 数组 数组)

代码示例：

```
int[] arr = {5, 1, 3};  
Arrays.sort(arr); // arr → [1, 3, 5]
```

---

### 5. ?? ??: Arrays.toString()

```
Arrays.toString(数组);
```

数组 数组 数组 数组 (数组)

代码示例：

```
System.out.println(Arrays.toString(arr)); // [1, 3, 5]
```

## 6. ?? ????: Arrays.fill()

```
Arrays.fill(??, ?);
```

?? ?? ?? ?? ?? .

?? ?? :

```
int[] arr = new int[5];  
Arrays.fill(arr, 7); // arr → [7, 7, 7, 7, 7]
```

## 7. ?? ?? ????: Arrays.binarySearch()

```
Arrays.binarySearch(?????, ???);
```

?? ?? ?? ?? ?? .  
?? ?? . (?? ?? )

?? ?? :

```
int[] sorted = {1, 2, 3, 4};  
int idx = Arrays.binarySearch(sorted, 3); // 2
```

## ? ??

??	??
?? ?? ??	<code>Arrays.equals()</code>
?? ??	<code>Arrays.copyOf()</code>
?? ??	<code>Arrays.copyOfRange()</code>
?? ??	<code>Arrays.sort()</code>
?? ??	<code>Arrays.toString()</code>

배열 초기화	<code>Arrays.fill()</code>
배열 검색	<code>Arrays.binarySearch</code>

?
 ?????
 ??
 ??
 ??

배열	배열
문자 배열, 정수 배열, 문자열 배열, 배열	<code>toCharArray()</code> → <code>Arrays.sort()</code> → <code>Arrays.equals()</code>
배열 복사, 정렬, 배열	<code>Arrays.copyOf()</code> + <code>Arrays.sort()</code>
배열 범위	<code>Arrays.copyOfRange()</code>

# Java ???(Expression)

## 1. ???????

“\*\*??? (Expression)\*\*”, ??? ??? .  
??, ??, ?? ??, ?? ?? ? ??? ??? ?? ?

## 2. ??? ?? ???

??	??	??
3 + 4	7	?? ??
"Hi" + " there"	"Hi there"	??? ??
new String("abc")	"abc"	?? ??
arr.length	?? ??	??? ?? ??
x > 5	true/false	?? ??

## 3. ????? ??(statement)? ??

- ??? : ?? ???
- ?? : ??? ??? (?? )

```
// ???
3 + 4           // ?: 7
"Hello" + "World" // ?: "HelloWorld"
x > 5           // ?: true ?? false

// ??
int a = 3 + 4;           // ?? ?? (??)
System.out.println("Hi"); // ??? ??
if (x > 5) { ... }       // ???
```



4. ???? ??? ??

Java	C#
return	return x + y;
int	int z = x * 2;
if	if (a > b)
System.out.println	System.out.println("Hi");

5. ??

-   .
-       .
-   ,  ,    .

# Java ?????? ??? (Escape Sequences)

1. ????? ?

“     (□ : □□ , □ □)□

      □ □□□ □ □ □ □ .

2. ?? ???? ? ?

エスケープシーケンス	説明	対応する文字列
<code>\n</code>	改行 (newline)	改行
<code>\t</code>	タブ	4~8文字 (タブ文字)
<code>\"</code>	ダブル引用符	ダブル引用符
<code>\'</code>	シングルクォーテーション	シングルクォーテーション
<code>\\</code>	バックslash	バックslash
<code>\r</code>	リターン (CR)	リターン
<code>\b</code>	バックスペース	バックスペース
<code>\f</code>	フォーマー (Form feed)	フォーマー

3. ?? ?? ??

```
public class EscapeExample {
    public static void main(String[] args) {
        System.out.println("\t\t\t\t\t");
        System.out.println("\t\t\t\t\t20\t");
        System.out.println("\t\t\t\t\t: \t\t\t\t\t");
    }
}
```

```
        System.out.println("C:\\Program Files\\Java");  
    }  
}
```

?? ??:

```
□□□□□□□□  
□□20□  
□□ □□: "□□□□□"  
C:\\Program Files\\Java
```

## 4. ?????

- □□ □□ \*\*□□□□ (")□ □□□□ (\\)\*\*□ □□□□ □ □
- □□ □□ □□□□ □□ □ □
- □□□□ □□ □□ □□□□ □□□□ □□ □□
- □□□□ □□□□ □□□( "..." )□□ □□( '...' ) □□□□ □□ □□

```
System.out.println("Hello\\nWorld"); // □□□  
char quote = '\\'; // □□□□□ □□
```

# Java 转义字符 (Escape Sequence, printf, format)

## 1. 转义字符

在 Java 中，转义字符 (Escape Sequence) 用于在字符串中表示特殊字符。转义字符通常以反斜杠 (\) 开头，后跟一个或多个字符。转义字符的格式为：\ 转义字符。转义字符的列表如下：

### 1.1 转义字符

转义字符	转义后的字符	转义后的字符串
\n	换行符	在字符串中表示换行
\t	制表符	在字符串中表示制表符 (通常 4~8 个空格)
\"	双引号	" 转义字符: \" \"
\'	单引号	'I\'m fine'
\\	反斜杠	"C:\\Users\\Dain"
\r	回车符	在字符串中表示回车 (通常 1 个空格)
\b	退格符	在字符串中表示退格 (通常 1 个空格)
\f	换页符	在字符串中表示换页 (通常 1 个空格)

### 1.2 转义字符

```
System.out.println("转义字符");
System.out.println("转义字符25");
System.out.println("转义字符: \"转义字符\"");
System.out.println("C:\\Program Files\\Java");
```

转义字符 :

```
int a = 10;
int b = 25;
String path: "C:\\Program Files\\Java"
```

## 2. ??? ?? ?? ??

Java에서 printf와 format 메서드를 사용하여 문자열을 포맷팅하는 방법.

### 2.1 ?? ??

```
System.out.printf(" ", 1, 2...);
String result = String.format(" ", 1, 2...);
```

### 2.2 ?? ?? ??

Format Specifier	Description	Example
%s	String	" : %s" → " 10"
%d	Decimal (10 base)	" : %d"
%f	Float (double)	" : %.2f"
%c	Character	" : %c"
%n	Newline (OS dependent)	" %n"
%%	Percent sign (%)	"100% "

### 2.3 ???/?? ??

Format Specifier	Description
%5d	Decimal, right-aligned, width 5
%-5d	Decimal, left-aligned, width 5
%05d	Decimal, zero-padded, width 5
%.2f	Float, rounded to 2 decimal places

%6.2f

6 2

## 2.4 ?? ??

```
String name = " ";
int age = 25;
double score = 93.756;

System.out.printf(" : %s, : %d%n", name, age);
System.out.printf(" : %.2f%n", score);
System.out.printf(" : %d%%n", 100);
```

: :

```
 : , : 25
 : 93.76
 : 100%
```

## 3. ?? ???

	\n, \t, \\	
(printf)	%d, %.2f, %s	

## 4. ?? ?? ?? (ANSI ??)

Java , ANSI .

```
public class ColorExample {
    public static final String RED = "\u001B[31m";
    public static final String RESET = "\u001B[0m";

    public static void main(String[] args) {
        System.out.println(RED + "  ." + RESET);
    }
}
```

```
    }  
}
```

格式	ANSI 格式
yyyy-MM-dd HH:mm:ss.SSS	\u001B[31m
yyyy-MM-dd HH:mm:ss.SSS	\u001B[32m
yyyy-MM-dd HH:mm:ss.SSS	\u001B[33m
yyyy-MM-dd HH:mm:ss.SSS	\u001B[34m
yyyy-MM-dd HH:mm:ss.SSS	\u001B[0m

## 5. 日期时间格式

Java 提供了 `java.time.LocalDateTime` 和 `DateTimeFormatter` 类，用于处理日期时间。本文介绍了如何使用它们来格式化日期时间。

### 5.1 DateTimeFormatter 使用

```
import java.time.LocalDateTime;  
import java.time.format.DateTimeFormatter;  
  
LocalDateTime now = LocalDateTime.now();  
DateTimeFormatter fmt = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");  
System.out.println("当前时间: " + now.format(fmt));
```

### 5.2 日期时间格式

格式	说明	示例
yyyy	年份	2025
MM	月份	05
dd	日期	22
HH	小时 (24小时制)	14
hh	小时 (12小时制)	02
mm	分钟	07

ss		59
a	MM /MM	AM / PM
E	MM	Wed

## 5.3 ??? ?? ?? ??

```
DateTimeFormatter customFmt = DateTimeFormatter.ofPattern("yyyy-MM-dd (E) a hh:mm");
String formatted = now.format(customFmt);
System.out.println("현재 시간: " + formatted);
```

## 6. ??? ?? ??

Java에서 로그를 기록하기 위해 `java.util.logging.Logger` 클래스를 사용합니다.

### 6.1 Logger ?? ??

```
import java.util.logging.*;

Logger logger = Logger.getLogger("MyLogger");

logger.info("로그 정보");
logger.warning("로그 경고");
logger.severe("로그 심각");
```

### 6.2 ?? ????? ?? ??

```
String name = "홍길동";
int age = 25;
logger.info(String.format("로그 정보: %s, 나이: %d", name, age));
```

### 6.3 ?? ?? ??



级别	描述
SEVERE	严重错误，系统可能无法正常运行
WARNING	警告，系统可能无法正常运行
INFO	信息，系统正常运行
CONFIG	配置信息，系统正常运行
FINE	详细日志，系统正常运行

## 6.4 配置日志输出

- 配置日志输出 (日志 / 文件 )
- 配置日志输出 (日志 , 文件 , 文件 目录 )
- logging.properties 配置 日志 输出 目录

# Java ???(Regular Expression)

## 1. ??

正则表达式 (Regular Expression) 是一种用于匹配字符串中模式的强大工具。在 Java 中，正则表达式通过 `java.util.regex` 包中的类来实现。

### 1.1. 正则表达式的基本语法

正则表达式的基本语法包括元字符、转义字符、量词、分组和断言等。以下是一些基本的元字符和转义字符：

- `[]` 字符类：用于匹配一个字符集中的任意一个字符。
  - `[a-z]` 匹配任意小写字母。
- `{n,m}` 量词：用于匹配指定次数。
  - `a{2,4}` 匹配 'a' 2 到 4 次。
- `^`, `$`, `.`, `|`, `\` 特殊字符。
  - `^abc` 匹配以 'abc' 开头的字符串。

### 1.2. 正则表达式预定义模式

- `\d`: 匹配任意数字 (0-9)
- `\w`: 匹配任意单词字符 (a-zA-Z0-9\_)
- `\s`: 匹配任意空白字符 (空格, 制表符, 换行符)
- `\b`: 匹配任意边界

## 2. 正则表达式的应用

### 2.1. Pattern 类

`Pattern` 类用于编译正则表达式，并提供匹配的方法。它位于 `java.util.regex` 包中。

#### 2.1.1. Pattern 编译

```
Pattern pattern = Pattern.compile("a*b");
```

编译正则表达式 "a\*b" 时，'a' 后面的 0 个或多个 'b' 都会被匹配。

## 2.1.2. Matcher ??? ??

Matcher 是 Pattern 的子类，用于匹配字符串。

```
Pattern pattern = Pattern.compile("a*b");
Matcher matcher = pattern.matcher("aaab");
boolean matches = matcher.matches(); // true
```

## 2.2. Pattern? Matcher? ?? ???

- matches(): 是否匹配整个字符串
- find(): 是否匹配下一个子字符串
- replaceAll(): 替换所有匹配的字符串

## 3. ?????? ?? ??

### 3.1. ??? ?? ??

正则表达式用于验证电子邮件地址。

```
Pattern pattern = Pattern.compile("^([a-zA-Z0-9_+&*~]+(?:\\.[a-zA-Z0-9_+&*~]+)*)@(?:[a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,7}$");
Matcher matcher = pattern.matcher("example@domain.com");
boolean isValid = matcher.matches(); // true
```

### 3.2. ????? ?? ??

正则表达式用于验证电话号码。

```
Pattern pattern = Pattern.compile("^\\d{3}-\\d{3,4}-\\d{4}$");
Matcher matcher = pattern.matcher("010-1234-5678");
boolean isValid = matcher.matches(); // true
```



# Java Arrays.sort() ??

## 1. ??

- `Arrays.sort()` Java 6 开始 支持 对 数组 进行 排序 .
- 对 数组 进行 排序 是 **in-place** 的 , 即 排序 后 数组 的 元素 位置 不变 . `void sort()`

## 2. ?? ??

<code>Arrays.sort(arr);</code>	
arr	arr (T : <code>int[]</code> , <code>char[]</code> , <code>String[]</code> )

## 3. void? ??

arr	arr
arr 是 数组	<code>Arrays.sort()</code> 对 数组 进行 排序
arr 是 数组 的 元素	对 数组 进行 排序 后 数组 的 元素 位置 不变
arr 是 数组	对 数组 进行 排序 后 数组 的 元素 位置 不变
arr X	<code>int[] sorted = Arrays.sort(arr);</code> 对 数组 进行 排序

## 4. ??

```
int[] numbers = {3, 1, 4};
Arrays.sort(numbers); // numbers 数组 排序

System.out.println(Arrays.toString(numbers)); // [1, 3, 4]
```

# 5. ?? ?? ?? ?

```
int[] sorted = Arrays.sort(numbers); // 返回排序后的数组: void 返回 int[] 数组
```

# 6. ??: ??? ??? ?? ??? ?????

```
int[] original = {3, 1, 2};
int[] copy = Arrays.copyOf(original, original.length);
Arrays.sort(copy);
```

# 7. ?? ???

方法	说明
Arrays.sort(arr)	对 arr 数组进行排序
Arrays.sort(arr, Comparator)	对 arr 数组进行排序，使用 Comparator 比较器 (Comparator 接口)
Collections.sort(list)	对 list 列表进行排序 (Comparable 接口)