Java 快速参考

OOP、集合、Stream、异常处理精要

基础

Hello World

```
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

编译与运行

```
javac Main.java      # compile
java Main             # run
java Main.java        # single-file (Java 11+)
```

命名约定

`ClassName`	类和接口用 PascalCase
`methodName`	方法和变量用 camelCase
`CONSTANT_NAME`	常量用 UPPER_SNAKE
`com.example.pkg`	包名用小写反向域名

数据类型

基本类型

`byte`	8 位有符号（-128 到 127）
`short`	16 位有符号
`int`	32 位有符号（默认整数）
`long`	64 位有符号（后缀 **`L`**）
`float`	32 位 IEEE-754（后缀 **`f`**）
`double`	64 位 IEEE-754（默认小数）
`boolean`	**`true`** / **`false`**
`char`	16 位 Unicode 字符

字符串

```
String s = "hello";
String joined = s + " world";      // concatenation
int len = s.length();
String sub = s.substring(0, 3);    // "hel"
boolean eq = s.equals("hello");    // content equality
```

类型转换

```
int i = (int) 3.14;               // narrowing cast
double d = i;                      // widening (auto)
int n = Integer.parseInt("42");    // string to int
String s = String.valueOf(42);     // int to string
```

数组

```
int[] nums = {1, 2, 3};
String[] names = new String[5];
int[][] matrix = new int[3][4];
Arrays.sort(nums);
```

控制流

If / Else

```
if (x > 0) {
    System.out.println("positive");
} else if (x == 0) {
    System.out.println("zero");
} else {
    System.out.println("negative");
}
```

Switch

```
// Traditional
switch (day) {
    case "Mon": doWork(); break;
    case "Sat": case "Sun": rest(); break;
    default: routine();
}
// Switch expression (Java 14+)
String type = switch (day) {
    case "Sat", "Sun" -> "weekend";
    default -> "weekday";
};
```

循环

```
for (int i = 0; i < 10; i++) { }
for (String s : list) { }         // enhanced for
while (condition) { }
do { } while (condition);
```

方法

定义

```
public static int add(int a, int b) {
    return a + b;
}
```

可变参数与重载

```
static int sum(int... nums) {
    int total = 0;
    for (int n : nums) total += n;
    return total;
}
// sum(1, 2)  sum(1, 2, 3)  both work
```

访问修饰符

`public`	任何地方可访问
`protected`	同包 + 子类
`(default)`	仅同包（无关键字）
`private`	仅同类

类与对象

类定义

```
public class User {
    private String name;
    private int age;
    public User(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public String getName() { return name; }
}
```

Record（Java 16+）

```
public record Point(double x, double y) {
    // auto: constructor, getters, equals, hashCode, toString
    public double distance() {
        return Math.sqrt(x * x + y * y);
    }
}
```

static 与 final

`static`	属于类而非实例
`final field`	初始化后不可再赋值
`final method`	不可被重写
`final class`	不可被继承

继承

extends

```
public class Animal {
    public void speak() { System.out.println("..."); }
}
public class Dog extends Animal {
    @Override
    public void speak() { System.out.println("Woof!"); }
}
```

抽象类

```
public abstract class Shape {
    abstract double area();
    public void describe() {
        System.out.println("Area: " + area());
    }
}
```

关键概念

`super`	调用父类构造器或方法
`@Override`	编译时重写检查
`instanceof`	运行时类型检查
`sealed (17+)`	限制可继承的类

接口

定义

```
public interface Printable {
    void print();                    // abstract
    default String format() {        // default method
        return toString();
    }
    static Printable of(String s) {  // static method
        return () -> System.out.println(s);
    }
}
```

实现

```
public class Report implements Printable, Serializable {
    @Override
    public void print() {
        System.out.println("Report");
    }
}
```

函数式接口

`Runnable`	**`() -> void`**
`Supplier<T>`	**`() -> T`**
`Consumer<T>`	**`T -> void`**
`Function<T,R>`	**`T -> R`**
`Predicate<T>`	**`T -> boolean`**
`Comparator<T>`	**`(T, T) -> int`**

集合

List

```
List<String> list = new ArrayList<>();
list.add("a");
list.get(0);                  // "a"
list.size();                  // 1
List<String> immutable = List.of("a", "b", "c");
```

Java 快速参考

Map

```
Map<String, Integer> map = new HashMap<>();
map.put("key", 42);
map.getOrDefault("key", 0);   // 42
map.containsKey("key");       // true
map.forEach((k, v) -> { });
```

Set

```
Set<String> set = new HashSet<>();
set.add("a");
set.contains("a");            // true
Set<String> immutable = Set.of("a", "b", "c");
```

常用实现

ArrayList	可变数组，随机访问快
LinkedList	双向链表，插入/删除快
HashMap	哈希表，O(1) get/put
TreeMap	按键排序，O(log n)
HashSet	唯一元素，O(1) 查找
LinkedHashMap	保持插入顺序的 HashMap

异常处理

Try / Catch / Finally

```
try {
    int result = 10 / 0;
} catch (ArithmeticException e) {
    System.err.println(e.getMessage());
} finally {
    // always executes
}
```

Try-with-Resources

```
try (var reader = new BufferedReader(new FileReader(path))) {
    String line = reader.readLine();
}   // auto-closes reader
```

异常层级

Throwable	所有错误和异常的根
Error	严重问题（OutOfMemoryError）
Exception	受检异常（必须处理）
RuntimeException	非受检异常（NullPointer、IndexOutOfBounds）

自定义异常

```
public class AppException extends Exception {
    public AppException(String msg) { super(msg); }
    public AppException(String msg, Throwable cause) {
        super(msg, cause);
    }
}
```

Stream 与 Lambda

Lambda 语法

```
Comparator<String> byLen = (a, b) -> a.length() - b.length();
Runnable task = () -> System.out.println("run");
Function<String, Integer> len = String::length;  // method ref
```

Stream 管道

```
List<String> result = names.stream()
    .filter(n -> n.length() > 3)
    .map(String::toUpperCase)
    .sorted()
    .collect(Collectors.toList());
```

常用 Stream 操作

.filter(pred)	保留满足断言的元素
.map(func)	转换每个元素
.flatMap(func)	映射并展平嵌套 Stream
.sorted()	排序（自然序或 Comparator）
.distinct()	去重
.limit(n)	取前 n 个元素
.collect()	终止：收集到集合
.forEach()	终止：对每个元素执行操作
.reduce()	终止：合并为单个值
.count()	终止：统计元素数量

泛型

泛型类与方法

```
public class Box<T> {
    private T value;
    public Box(T value) { this.value = value; }
    public T get() { return value; }
}
public static <T> List<T> listOf(T... items) {
    return List.of(items);
}
```

有界类型与通配符

<T extends Number>	T 必须是 Number 或其子类
<T extends A & B>	多界（类 + 接口）
<?>	未知类型（只读）
<? extends T>	上界通配符（生产者）
<? super T>	下界通配符（消费者）

Optional 与现代 Java

Optional

```
Optional<String> opt = Optional.ofNullable(getValue());
String result = opt.orElse("default");
opt.ifPresent(v -> System.out.println(v));
String upper = opt.map(String::toUpperCase).orElse("");
```

文本块（Java 15+）

```
String json = """
        { "name": "Alice", "age": 30 }
        """;
```

实用新特性

var (10+)	局部变量类型推断
record (16+)	不可变数据载体类
sealed (17+)	受限类层级
pattern matching (21+)	**instanceof** 带自动转型
virtual threads (21+)	轻量线程（**Thread.ofVirtual()**）