# Referência Rápida de Java

POO, coleções, streams, tratamento de exceções

## Básico

### Hello World

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

### Compilar e Executar

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

### Convenções de Nomenclatura

| | |
|---|---|
| **`ClassName`** | PascalCase para classes e interfaces |
| **`methodName`** | camelCase para métodos e variáveis |
| **`CONSTANT_NAME`** | UPPER_SNAKE para constantes |
| **`com.example.pkg`** | Domínio reverso em minúsculas para pacotes |

## Tipos de Dados

### Primitivos

| | |
|---|---|
| **`byte`** | 8 bits com sinal (-128 a 127) |
| **`short`** | 16 bits com sinal |
| **`int`** | 32 bits com sinal (inteiro padrão) |
| **`long`** | 64 bits com sinal (sufixo **`L`**) |
| **`float`** | 32 bits IEEE-754 (sufixo **`f`**) |
| **`double`** | 64 bits IEEE-754 (decimal padrão) |
| **`boolean`** | **`true`** / **`false`** |
| **`char`** | Caractere Unicode de 16 bits |

### Strings

```
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
```

### Conversão de Tipos

```
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
```

### Arrays

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

## Fluxo de Controle

### 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";
};
```

### Laços

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

## Métodos

### Definição

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

### Varargs e Sobrecarga

```
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
```

### Modificadores de Acesso

| | |
|---|---|
| **`public`** | Acessível de qualquer lugar |
| **`protected`** | Mesmo pacote + subclasses |
| **`(default)`** | Somente mesmo pacote (sem palavra-chave) |
| **`private`** | Somente mesma classe |

## Classes e Objetos

### Definição de Classe

```
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; }
}
```

### Records (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 e Final

| | |
|---|---|
| **`static`** | Pertence à classe, não à instância |
| **`final field`** | Não pode ser reatribuído após inicialização |
| **`final method`** | Não pode ser sobrescrito |
| **`final class`** | Não pode ser herdado |

## Herança

### Extends

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

### Classes Abstratas

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

### Conceitos-chave

| | |
|---|---|
| **`super`** | Chama construtor ou método do pai |
| **`@Override`** | Verificação de sobrescrita em tempo de compilação |
| **`instanceof`** | Verificação de tipo em tempo de execução |
| **`sealed (17+)`** | Restringe quais classes podem herdar |

## Interfaces

### Definição

```
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);
    }
}
```

### Implementação

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

### Interfaces Funcionais

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

## Coleções

### List

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

# Referência Rápida de Java

### 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");
```

### Implementações Comuns

| | |
|---|---|
| **`ArrayList`** | Array redimensionável, acesso aleatório rápido |
| **`LinkedList`** | Duplamente encadeada, inserção/remoção rápidas |
| **`HashMap`** | Tabela hash, get/put O(1) |
| **`TreeMap`** | Ordenado por chave, O(log n) |
| **`HashSet`** | Elementos únicos, busca O(1) |
| **`LinkedHashMap`** | HashMap com ordem de inserção |

## Tratamento de Exceções

### 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
```

### Hierarquia de Exceções

| | |
|---|---|
| **`Throwable`** | Raiz de todos os erros e exceções |
| **`Error`** | Problemas graves (OutOfMemoryError) |
| **`Exception`** | Exceções verificadas (devem ser tratadas) |
| **`RuntimeException`** | Não verificadas (NullPointer, IndexOutOfBounds) |

### Exceção Personalizada

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

## Streams e Lambdas

### Sintaxe de Lambda

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

### Pipeline de Stream

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

### Operações Comuns de Stream

| | |
|---|---|
| **`.filter(pred)`** | Mantém elementos que satisfazem o predicado |
| **`.map(func)`** | Transforma cada elemento |
| **`.flatMap(func)`** | Mapeia e achata streams aninhados |
| **`.sorted()`** | Ordena (natural ou com Comparator) |
| **`.distinct()`** | Remove duplicatas |
| **`.limit(n)`** | Obtém os primeiros n elementos |
| **`.collect()`** | Terminal: agrupa em coleção |
| **`.forEach()`** | Terminal: executa ação em cada elemento |
| **`.reduce()`** | Terminal: combina em valor único |
| **`.count()`** | Terminal: conta elementos |

## Generics

### Classe e Método Genérico

```
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);
}
```

### Tipos Limitados e Wildcards

| | |
|---|---|
| **`<T extends Number>`** | T deve ser Number ou subclasse |
| **`<T extends A & B>`** | Múltiplos limites (classe + interfaces) |
| **`<?>`** | Tipo desconhecido (somente leitura) |
| **`<? extends T>`** | Wildcard com limite superior (produtor) |
| **`<? super T>`** | Wildcard com limite inferior (consumidor) |

## Optional e Java Moderno

### 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("");
```

### Blocos de Texto (Java 15+)

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

### Utilitários Úteis

| | |
|---|---|
| **`var (10+)`** | Inferência de tipo de variável local |
| **`record (16+)`** | Classe portadora de dados imutáveis |
| **`sealed (17+)`** | Hierarquias de classes restritas |
| **`pattern matching (21+)`** | **`instanceof`** com conversão automática |
| **`virtual threads (21+)`** | Threads leves via **`Thread.ofVirtual()`** |