

Java クイックリファレンス

OOP、コレクション、ストリーム、例外処理の基礎

基本

Hello World

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

コンパイルと実行

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

命名規則

ClassName	クラスとインターフェースにはPascalCase
methodName	メソッドと変数にはcamelCase
CONSTANT_NAME	定数にはUPPER_SNAKE
com.example.pkg	パッケージにはリバースドメインの小文字

データ型

プリミティブ型

byte	8ビット符号付き (-128 ~ 127)
short	16ビット符号付き
int	32ビット符号付き (デフォルト整数)
long	64ビット符号付き (サフィックス L)
float	32ビット IEEE-754 (サフィックス f)
double	64ビット IEEE-754 (デフォルト小数)
boolean	true/false
char	16ビット Unicode 文字

文字列

```
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
```

型キャスト

```
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
```

配列

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

制御フロー

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";
};
```

ループ

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

メソッド

定義

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

可変長引数とオーバーロード

```
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
```

アクセス修飾子

public	どこからでもアクセス可能
protected	同一パッケージ+サブクラス
(default)	同一パッケージのみ (キーワードなし)
private	同一クラスのみ

クラスとオブジェクト

クラス定義

```
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; }
}
```

レコード (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 と final

static	インスタンスではなくクラスに属する
final field	初期化後に再代入できない
final method	オーバーライドできない
final class	サブクラス化できない

継承

extends

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

抽象クラス

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

主要な概念

super	親のコンストラクターまたはメソッドを呼び出す
@Override	コンパイル時にオーバーライドを検証
instanceof	実行時の型チェック
sealed (17+)	継承できるクラスを制限する

インターフェース

定義

```
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);
    }
}
```

実装

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

関数型インターフェース

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

コレクション

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 クイックリファレンス

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");
```

主な実装クラス

ArrayList	可変長配列、ランダムアクセスが高速
LinkedList	双方向リスト、挿入/削除が高速
HashMap	ハッシュテーブル、get/putがO(1)
TreeMap	キー順にソート済み、O(log n)
HashSet	一意の要素、ルックアップがO(1)
LinkedHashMap	挿入順を保持するHashMap

例外処理

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
```

例外の階層

Throwable	すべてのエラーと例外のルート
Error	深刻な問題 (OutOfMemoryError)
Exception	チェック例外 (処理必須)
RuntimeException	非チェック例外 (NullPointerException, IndexOutOfBoundsException)

カスタム例外

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

ストリームとラムダ

ラムダ構文

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

ストリームパイプライン

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

よく使うストリーム操作

.filter(pred)	述語にマッチする要素を保持
.map(func)	各要素を変換
.flatMap(func)	マップしてネストしたストリームをフラット化
.sorted()	ソート (自然順または Comparator 指定)
.distinct()	重複を除去
.limit(n)	最初の n 要素を取得
.collect()	終端: コレクションに集約
.forEach()	終端: 各要素にアクションを実行
.reduce()	終端: 単一の値に集約
.count()	終端: 要素数をカウント

ジェネリクス

ジェネリッククラスとメソッド

```
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);
}
```

境界型とワイルドカード

<T extends Number>	T は Number またはそのサブクラス
<T extends A & B>	複数の境界 (クラス + インターフェース)
<?>	不明な型 (読み取り専用)
<? extends T>	上限境界ワイルドカード (プロデューサー)
<? super T>	下限境界ワイルドカード (コンシューマー)

Optional とモダン 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("");
```

テキストブロック (Java 15+)

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

便利なユーティリティ

var (10+)	ローカル変数の型推論
record (16+)	イミュータブルなデータキャリアクラス
sealed (17+)	制限されたクラス階層
pattern matching (21+)	自動キャスト付き instanceof
virtual threads (21+)	Thread.ofVirtual() による軽量スレッド