JAVA 빠른 참조

OOP, 컬렉션, 스트림, 예외 처리 핵심

기본

Hello World

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

컴파일 및 실행

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

네이밍 컨벤션

`ClassName`	클래스와 인터페이스는 PascalCase
`methodName`	메서드와 변수는 camelCase
`CONSTANT_NAME`	상수는 UPPER_SNAKE
`com.example.pkg`	패키지는 소문자 역방향 도메인

데이터 타입

기본 타입

`byte`	8비트 부호 있는 정수 (-128 ~ 127)
`short`	16비트 부호 있는 정수
`int`	32비트 부호 있는 정수 (기본 정수형)
`long`	64비트 부호 있는 정수 (접미사 `L`)
`float`	32비트 IEEE-754 부동소수점 (접미사 `f`)
`double`	64비트 IEEE-754 부동소수점 (기본 소수형)
`boolean`	`true` / `false`
`char`	16비트 유니코드 문자

문자열

```
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
```

타입 변환

```
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
```

배열

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

제어 흐름

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";
};
```

반복문

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

메서드

메서드 정의

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

가변 인수 & 오버로딩

```
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
```

접근 제한자

`public`	어디서든 접근 가능
`protected`	같은 패키지 + 서브클래스
`(default)`	같은 패키지만 (키워드 없음)
`private`	같은 클래스만

클래스 & 객체

클래스 정의

```
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; }
}
```

레코드 (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 & final

`static`	인스턴스가 아닌 클래스에 속함
`final field`	초기화 후 재할당 불가
`final method`	오버라이드 불가
`final class`	서브클래스 생성 불가

상속

extends

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

추상 클래스

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

핵심 개념

`super`	부모 생성자 또는 메서드 호출
`@Override`	컴파일 타임 오버라이드 검사
`instanceof`	런타임 타입 검사
`sealed (17+)`	확장 가능한 클래스를 제한

인터페이스

인터페이스 정의

```
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);
    }
}
```

구현

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

함수형 인터페이스

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

컬렉션

List

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

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");
```

주요 구현체

`ArrayList`	크기 조절 가능한 배열, 빠른 랜덤 접근
`LinkedList`	이중 연결 리스트, 빠른 삽입/삭제
`HashMap`	해시 테이블, O(1) get/put
`TreeMap`	키 기준 정렬, O(log n)
`HashSet`	고유 요소, O(1) 조회
`LinkedHashMap`	삽입 순서를 유지하는 HashMap

예외 처리

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
```

예외 계층 구조

`Throwable`	모든 오류 및 예외의 루트
`Error`	심각한 문제 (OutOfMemoryError)
`Exception`	체크 예외 (반드시 처리해야 함)
`RuntimeException`	언체크 예외 (NullPointer, IndexOutOfBounds)

사용자 정의 예외

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

스트림 & 람다

람다 문법

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

스트림 파이프라인

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

주요 스트림 연산

`.filter(pred)`	조건을 만족하는 요소만 유지
`.map(func)`	각 요소를 변환
`.flatMap(func)`	매핑 후 중첩 스트림 펼침
`.sorted()`	정렬 (자연 순서 또는 Comparator)
`.distinct()`	중복 제거
`.limit(n)`	앞의 n개 요소만 가져오기
`.collect()`	터미널: 컬렉션으로 수집
`.forEach()`	터미널: 각 요소에 동작 수행
`.reduce()`	터미널: 단일 값으로 결합
`.count()`	터미널: 요소 개수 세기

제네릭

제네릭 클래스 & 메서드

```
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);
}
```

경계 타입 & 와일드카드

`<T extends Number>`	T는 Number 또는 그 서브클래스여야 함
`<T extends A & B>`	다중 경계 (클래스 + 인터페이스)
`<?>`	알 수 없는 타입 (읽기 전용)
`<? extends T>`	상한 와일드카드 (생산자)
`<? super T>`	하한 와일드카드 (소비자)

Optional & 모던 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("");
```

텍스트 블록 (Java 15+)

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

유용한 기능

`var (10+)`	지역 변수 타입 추론
`record (16+)`	불변 데이터 캐리어 클래스
`sealed (17+)`	제한된 클래스 계층 구조
`pattern matching (21+)`	자동 캐스팅이 포함된 `instanceof`
`virtual threads (21+)`	`Thread.ofVirtual()`을 통한 경량 스레드