Java 핵심 치트시트

기본 구조 및 컴파일

자바 프로그램의 기본 구조는 클래스 내부에 정의된 `main` 메서드로부터 시작됩니다.

```
// HelloWorld.java
public class HelloWorld {
    // 프로그램 실행 시 최초로 호출되는 진입점(Entry Point)
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

- **컴파일 및 실행**:

```
javac HelloWorld.java  # 소스 파일(.java)을 바이트코드(.class)로 컴파일
java HelloWorld        # JVM에서 컴파일된 클래스 실행
```

변수와 데이터 타입

- **기본 타입 (Primitive Types)**: 실제 데이터 값을 저장합니다.
  - **정수형**: `byte` (1), `short` (2), `int` (4, 기본), `long` (8)
  - **실수형**: `float` (4), `double` (8, 기본)
  - **문자형**: `char` (2, 유니코드)
  - **논리형**: `boolean` (1, true/false)
- **참조 타입 (Reference Types)**: 객체의 주소값을 저장합니다. (`String`, 배열, 클래스, 인터페이스 등)
- **변수 선언**: `타입 변수명 = 값;` (예: `int age = 30;`)
- **상수(Constant)**: `final` 키워드를 사용하며, 초기화 후 변경이 불가능합니다. (예: `final double PI = 3.14159;`)
- **형변환 (Casting)**:
  - 묵시적 형변환: 작은 타입에서 큰 타입으로 자동 변환됩니다.
  - 명시적 형변환: `(타입)값` 형식을 사용하여 강제로 변환합니다. (예: `(int) 3.14`)

제어문 (Control Flow)

- **조건문 (if-else)**:

```
if (score ≥ 90) {
    System.out.println("A 학점");
} else if (score ≥ 80) {
    System.out.println("B 학점");
} else {
    System.out.println("기타");
}
```

- **선택문 (switch)**: (Java 12+ 에서는 화살표 구문 지원)

```
switch (day) {
    case "MONDAY" -> System.out.println("월요일");
    case "FRIDAY" -> System.out.println("금요일");
    default -> System.out.println("평일");
}
```

- **반복문**:
  - `for`: 정해진 횟수만큼 반복합니다. `for (int i=0; i<5; i++) { ... }`
  - **향상된 for 루프**: 배열이나 컬렉션 순회 시 편리합니다.

```
for (String s : list)
{ System.out.println(s); }
```

  - `while` / `do-while`: 조건에 따라 반복하며, `do-while`은 최소 1회 실행을 보장합니다.

객체 지향 프로그래밍 (OOP)

- **클래스와 객체**: 클래스는 객체를 만들기 위한 설계도이며, 객체는 그 설계도를 바탕으로 생성된 실체(인스턴스)입니다.

```
public class Car {
    private String model; // 필드 (멤버 변수)

    public Car(String model) { // 생성자
        this.model = model;
    }

    public void drive() { // 메서드
        System.out.println(model + "이(가) 달립니다.");
    }
}
```

- **핵심 4대 원칙**:
  1. **캡슐화**: 접근 제어자(`private`, `public` 등)를 통해 정보를 은닉합니다.
  2. **상속 (extends)**: 부모 클래스의 기능을 물려받아 재사용 및 확장합니다.
  3. **다형성**: 부모 타입의 참조 변수로 자식 객체를 다루는 기술입니다.
  4. **추상화**: 공통적인 특징을 추출하여 상위 개념으로 정의합니다.

인터페이스와 추상 클래스

- **추상 클래스** (`abstract`): 미완성 설계도로, 하나 이상의 추상 메서드를 가질 수 있으며 단일 상속만 가능합니다.
- **인터페이스** (`interface`): 규범(스펙)을 정의하며, 다중 구현이 가능합니다. (Java 8+ 부터 `default`, `static` 메서드 허용)

```
public interface Flyable {
    void fly(); // 추상 메서드
}
```

컬렉션 프레임워크 (Collections)

데이터 그룹을 다루기 위한 표준화된 라이브러리입니다.

- **List**: 순서가 있고 중복을 허용합니다. (`ArrayList`, `LinkedList`)
- **Set**: 순서가 없고 중복을 허용하지 않습니다. (`HashSet`, `TreeSet`)
- **Map**: 키(Key)와 값(Value)의 쌍으로 저장하며, 키는 중복될 수 없습니다. (`HashMap`, `TreeMap`)

예외 처리 (Exception Handling)

프로그램 실행 중 발생할 수 있는 오류에 대비합니다.

- **try-catch-finally**:

```
try {
    int result = 10 / 0;
} catch (ArithmeticException e) {
    System.err.println("0으로 나눌 수 없습니다.");
} finally {
    // 자원 해제 등 항상 실행될 코드
}
```

- **throws**: 예외 발생 시 직접 처리하지 않고 호출한 메서드에게 처리를 위임합니다.

스트림 API 및 람다 (Java 8+)

- **람다식 (Lambda)**: 함수형 프로그래밍 스타일을 지원하는 간결한 표현식입니다. `(a, b) → a + b`
- **스트림 API**: 데이터를 선언적이고 함수형으로 처리할 수 있는 파이프라인 구조를 제공합니다.

```
List<Integer> list = List.of(1, 2, 3, 4, 5);
int sum = list.stream()
              .filter(n → n % 2 == 0) // 짝수만 선택
              .mapToInt(n → n * 2)    // 2배로 가공
              .sum();                 // 최종 합계
```

제네릭 (Generics)

컴파일 단계에서 타입을 체크하여 타입 안전성을 높이고 형변환 번거로움을 줄입니다.

```
List<String> list = new ArrayList<>();
list.add("Java");
// list.add(10); // 컴파일 에러 발생
```

JVM과 메모리 구조

- **JVM (Java Virtual Machine)**: 자바 바이트코드를 실행하는 가상 머신으로, 플랫폼 독립성을 보장합니다.
- **메모리 영역**:
  - **Heap**: 객체가 생성되는 영역이며 GC(Garbage Collector)의 관리 대상입니다.
  - **Stack**: 메서드 호출 시 생성되는 지역 변수와 매개변수가 저장됩니다.
  - **Method Area**: 클래스 정보, 상수가 저장되는 공유 영역입니다.

Google Style Guide

명명 규칙 (Naming)

- 패키지: `com.example.project` (소문자 연속)
- 클래스/인터페이스: `UpperCamelCase` (명사형)
- 메서드: `lowerCamelCase` (동사형)
- 상수: `UPPER_SNAKE_CASE` (static final)
- 변수 (로컬/필드): `lowerCamelCase`
- 타입 변수: `T`, `E` (단일 대문자) 또는 `RequestT` (T 접미사)

포매팅 (Formatting)

- 들여쓰기: 공백 2개 (탭 금지)
- 줄 길이: 최대 100자
- 중괄호: K&R 스타일 (줄 바꿈 없이 시작 brace { 사용)

- 빈 블록: `{}` 처럼 간결하게 표현 가능 (단, multi-block 문장은 제외)
- 문장: 한 줄에 하나의 문장만 작성

프로그래밍 관례

- Wildcard Imports: 사용 금지 (`import java.util.*;` 금지).
- Override: `@Override` 어노테이션 반드시 사용.
- Exceptions: 예외를 무시하지 말 것 (무시해야 한다면 주석으로 이유 명시).
- Static Members: 클래스 이름으로 접근 (`MyClass.staticMethod()`).
- Modifiers: JLS 권장 순서 준수 (`public protected private abstract ...`).

주요 관례 및 Javadoc

- Javadoc: 모든 public 클래스와 멤버에 필수. 첫 문장은 완전한 문장이 아닌 요약 구문이어야 함.
- Block Tags: `@param`, `@return`, `@throws`, `@deprecated` 순서로 작성.
- Annotations: 하나의 어노테이션은 별도의 줄에 작성 (단, 단일 인자 없는 어노테이션은 예외).
- Local Variables: 처음 사용되는 지점 근처에서 선언하여 범위를 최소화.
- Long Literals: `L` 접미사 사용 (소문자 `l` 금지).