1. Java 언어 개요 및 개발 환경

Java란 무엇인가? (역사, 특징, 플랫폼 독립성)

1. 개요

Java는 **객체지향 프로그래밍 언어**이며, 플랫폼 독립성과 보안성, 확장성, 풍부한 표준 API를 특징으로 하는 범용 프로그래밍 언어이다. 1995년 **Sun Microsystems**에 의해 처음 발표되었으며, 현재는 **Oracle**에서 관리하고 있다.

Java는 컴파일된 바이트코드가 JVM (Java Virtual Machine) 위에서 실행되는 구조를 갖고 있어, 한 번 작성한 프로그램을 여러 운영체제에서 실행할 수 있다는 특징을 가진다.

2. 역사 (Java의 탄생과 발전)

연도	사건
1991	James Gosling이 주도한 **Green 프로젝트** 시작 (가전제품용 언어)
1995	**Java 1.0** 공개 (Sun Microsystems 발표)
1997	JavaOne 컨퍼런스 최초 개최
2006	Java가 **오픈소스로 공개됨 (OpenJDK)**
2009	Sun Microsystems가 **Oracle에 인수**됨
2017	Java 9: **모듈 시스템 (JPMS)** 도입
2018~	**6개월 릴리즈 주기** 도입 (Java 10부터)
2021	Java 17 출시 (LTS)
2023	Java 21 출시 (LTS, 최신 기능 포함)

3. Java의 주요 특징

🧱 3.1 객체지향 언어 (OOP)

- 클래스와 객체 중심으로 구성됨
- **캡슐화, 상속, 다형성, 추상화** 등 OOP 원칙 지원

🌐 3.2 플랫폼 독립성 (Write Once, Run Anywhere)

- Java 코드는 `.java`로 작성 → `.class` 바이트코드로 컴파일
- JVM이 설치된 어느 플랫폼에서든 실행 가능

```
Java Source → (javac) → Bytecode → (JVM) → 실행
```

🔐 3.3 보안성

- 메모리 접근 제한, 런타임 체크
- 샌드박스 환경에서의 코드 실행
- ClassLoader, SecurityManager 등 기반 보안 프레임워크 존재

🧮 3.4 풍부한 라이브러리

- `java.lang`, `java.util`, `java.io`, `java.net`, `javax.swing` 등 기본 패키지 제공
- 대규모 애플리케이션 구축을 위한 다양한 오픈소스 프레임워크와의 호환

⚡ 3.5 멀티스레드 지원

- `Thread` 클래스 및 `Runnable` 인터페이스를 통한 다중 실행 흐름 구현
- `ExecutorService`, `ForkJoinPool` 등의 고급 API 존재

🧠 3.6 자동 메모리 관리

- **Garbage Collector**를 통해 불필요한 객체 자동 제거
- 메모리 누수 방지, 프로그래머의 부담 감소

☕ 3.7 단순성과 안정성

- 포인터 제거, 명시적 메모리 해제 없음
- 일관된 문법으로 초보자도 학습이 쉬움

4. Java의 실행 구조와 플랫폼 독립성

📌 Java 실행 단계

1. `.java` 파일: 사람이 읽을 수 있는 소스코드
2. `javac` 컴파일러 → `.class` 바이트코드 생성
3. `java` 명령어로 JVM이 바이트코드를 실행

📌 JVM의 역할

- 운영체제별로 설계된 JVM이 바이트코드를 동일한 방식으로 실행함
- 바이트코드 ↔ 머신코드 변환을 통해 OS 독립성 유지

📌 예시

OS	JVM 종류
Windows	HotSpot Windows x64
macOS	HotSpot macOS AArch64
Linux	OpenJ9 Linux x64

바로 이 JVM 덕분에 Java는 **"한 번 작성하면, 어디서나 실행 가능"**한 언어로 평가받는다.

5. Java와 다른 언어 비교 (요약)

항목	Java	C++	Python
컴파일 방식	바이트코드 (JVM)	바이너리 (기계어)	인터프리터/바이트코드
메모리 관리	GC 자동 관리	수동 (`new/delete`)	GC 자동 관리
플랫폼 독립성	JVM으로 가능	불가능 (OS 의존적)	가상환경으로 가능
실행 속도	중간	빠름	느림
주요 용도	서버, 모바일, IoT 등	시스템, 게임, 하이엔드	AI, 웹, 데이터 분석

JDK, JRE, JVM 구조

1. 개요: 세 용어의 관계

구성요소	역할 요약	포함 관계
JDK	개발 도구 포함된 Java 전체 키트	**JDK ⊃ JRE ⊃ JVM**
JRE	실행 전용 런타임 환경	JVM + Java 표준 라이브러리 포함
JVM	바이트코드 실행 가상 머신	JRE의 핵심 엔진

---

2. JVM (Java Virtual Machine)

✅ 정의

JVM은 Java 바이트코드를 실제 머신에서 실행되도록 해주는 **가상화된 실행 환경**이다. OS에 따라 구현이 다르며, 플랫폼 독립성을 제공하는 핵심 역할을 한다.

✅ JVM 주요 구성

구성요소	설명
클래스 로더(Class Loader)	`.class` 파일을 JVM 메모리로 로딩
실행 엔진(Execution Engine)	바이트코드를 해석 or JIT 컴파일
메모리 영역 (Runtime Data Areas)	스택, 힙, 메서드 영역 등
네이티브 인터페이스 (JNI)	Java → C/C++ 등의 네이티브 코드 호출
가비지 컬렉터 (GC)	사용되지 않는 객체 자동 메모리 해제

✅ JVM 메모리 구조

```
+--------------------+
| Method Area        | ← 클래스 메타데이터 저장
+--------------------+
| Heap               | ← 객체 인스턴스 저장 영역 (GC 대상)
+--------------------+
| Java Stack         | ← 쓰레드별 스택 프레임 저장
+--------------------+
| PC Register        | ← 현재 실행 중인 명령 주소
+--------------------+
| Native Method Stack| ← C 코드 등 네이티브 메서드 호출용
+--------------------+
```

---

3. JRE (Java Runtime Environment)

✅ 정의

JRE는 **Java 애플리케이션을 실행하기 위한 환경**으로, JVM + Java 표준 클래스 라이브러리로 구성된다.

JRE만으로는 **코드를 개발하거나 컴파일할 수 없고**, 오직 실행만 가능하다.

✅ 구성 요소

구성요소	설명
JVM	바이트코드 실행
Java 클래스 라이브러리	`rt.jar`, `java.base`, `java.util`, `java.io` 등
지원 파일들	보안 설정, 로깅 설정, 로컬라이징 리소스 등

---

4. JDK (Java Development Kit)

✅ 정의

JDK는 Java 개발자가 애플리케이션을 **작성, 컴파일, 디버깅, 실행**할 수 있도록 하는 **완전한 개발 도구 세트**이다. JRE를 포함하고, 개발에 필요한 컴파일러와 툴이 추가되어 있다.

✅ 구성 요소

구성요소	설명
JRE	실행 환경 포함
javac	Java 컴파일러
java	JVM 런처
javadoc	API 문서 자동 생성기

구성요소	설명
jar	JAR 패키징 도구
jdb	디버깅 도구
jshell	Java REPL (Java 9+)
jlink / jpackage	모듈 기반 실행환경 생성 (Java 9~)

---

5. JDK, JRE, JVM의 관계 (도식)

```
+-----------------------------+
|          JDK                |
| +--------------------------+|
| |         JRE              ||
| | +----------------------+ ||
| | |        JVM           | ||
| | +----------------------+ ||
| +--------------------------+|
+-----------------------------+
```

---

6. 개발 vs 실행 비교

항목	JDK	JRE
개발 가능 여부	✅ 가능 (`javac` 있음)	❌ 불가능
실행 가능 여부	✅ 가능 (`java`) 있음	✅ 가능
대상 사용자	개발자 (IDE 포함 가능)	최종 사용자
포함된 구성요소	JRE + 컴파일러, 도구	JVM + 라이브러리

---

7. 실습 예시

📄 Hello.java

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

📦 컴파일 & 실행

```
javac Hello.java   # JDK의 javac로 컴파일
java Hello         # JVM으로 실행 (JRE 필요)
```

---

8. 요약 정리

용어	정의	구성 요소	용도
JVM	바이트코드 실행 가상 머신	실행 엔진, 메모리 구조 등	Java 코드 실행
JRE	실행 전용 환경	JVM + 라이브러리	사용자 실행용
JDK	개발 도구 모음	JRE + 컴파일러 등	개발자용

Java 버전별 주요 변화 (8, 11, 17, 21 등)

✅ Java 8 (2014년 3월, LTS)

> 🟤 Java의 근본을 바꾼 대격변! 함수형 프로그래밍 시대의 시작

🔷 주요 기능

기능	설명
Lambda Expressions	익명 함수 문법 도입 (`(x) -> x + 1`)
Stream API	컬렉션의 함수형 처리 (필터링, 매핑, 리듀스)
Functional Interfaces	`@FunctionalInterface` 도입 (`Function` , `Predicate` , `Consumer`)
Method References	`Class::method` 구문 도입
Optional	Null 안전 처리를 위한 타입
Date and Time API (java.time)	`LocalDate` , `ZonedDateTime` , `Duration` 등 도입
Default & Static methods in interfaces	인터페이스에 메서드 정의 가능
Nashorn JavaScript Engine	내장 JS 엔진 (11에서 제거 예정)
Parallel Arrays, CompletableFuture	비동기 프로그래밍 지원

✅ Java 11 (2018년 9월, LTS)

> 🟤 모듈화 이후의 새로운 안정판. 대거 제거 및 자바의 군살 다이어트!

🔷 주요 기능

기능	설명
`var` in lambda parameters	람다 매개변수에서도 `var` 사용 가능
String API 개선	`isBlank()`, `lines()`, `strip()`, `repeat()` 등
HTTP Client (java.net.http)	기존 `HttpURLConnection` 대체, HTTP/2 지원
Files.readString/writeString	파일 간단 입출력 메서드
ZGC (Experimental GC)	초저 지연 GC (Z Garbage Collector)
Flight Recorder & Mission Control	성능 분석 툴 내장
Launch Single-File Program	`java Hello.java` 한 줄 실행 가능

🔻 주요 제거

제거 항목	설명
Java EE / CORBA API	`javax.xml`, `javax.activation` 등 제거
Nashorn JavaScript Engine (deprecated)	JS 실행 기능 제거 예정

✅ Java 17 (2021년 9월, LTS)

> 🟤 진짜 현대 Java. 패턴 매칭, 레코드, 새 GC, 보안 모델 정비 등 최신 문법의 집대성

🔷 주요 기능

기능	설명
Sealed Classes	상속 가능한 클래스를 제한할 수 있음 (`permits`)
Pattern Matching for instanceof	`if (obj instanceof String s)` 와 같이 캐스팅 생략 가능
Record (Java 16 → 정식)	데이터 전달용 불변 클래스 간결하게 작성 (`record Person(...)`)
Text Blocks	`"""` 다중 행 문자열 지원
Switch Expressions (preview in 12~14)	`yield` 사용 가능하고 더 간결해짐
New macOS Rendering Pipeline	macOS 환경에서 Metal 기반 렌더링
Strong Encapsulation of JDK Internals	`sun.misc.Unsafe` 같은 내부 API 접근 불가

✅ Java 21 (2023년 9월, LTS)

> 🧠 실전 함수형 + 더 똑똑한 switch + 구조적 동시성 = 지금부터 Java는 모던 언어

🔷 주요 기능

기능	설명
String Templates (preview)	문자열 내 변수 직접 삽입 지원 (`STR."Hello, \{name}!"`)
Unnamed Classes and Instance Main Methods (preview)	`public class` 없이도 실행 가능
Record Patterns (finalized)	레코드 분해 구조 지원 (`case Person(String name, int age)`)
Pattern Matching for switch (finalized)	switch문에서 타입/조건 분기 가능
Virtual Threads (finalized)	수백만 개 경량 스레드 → 고성능 동시성 지원 (`Thread.ofVirtual().start(...)`)
Structured Concurrency (preview)	쓰레드 라이프사이클 통합 관리
Scoped Values (preview)	쓰레드 간 데이터 전달 → 기존 `ThreadLocal` 대체 가능

📊 요약 비교표

항목	Java 8	Java 11	Java 17	Java 21
LTS 여부	✅	✅	✅	✅
Lambda, Stream	✅	✅	✅	✅
var 지원	❌	람다에서만 ✅	✅	✅
Record	❌	❌	✅	✅
Pattern Matching	❌	❌	`instanceof`, `switch` (일부)	✅ (완전 지원)
Virtual Threads	❌	❌	❌	✅
Text Block	❌	❌	✅	✅
HTTP Client	❌	✅	✅	✅

🚀 개발자는 어떤 버전을 써야 할까?

- **Java 8**: 여전히 일부 레거시 시스템에서 사용 중이나 신규 개발에 부적합
- **Java 11**: 안정적인 엔터프라이즈 플랫폼
- **Java 17**: 대부분의 기업과 프레임워크의 기준 (Spring Boot 3.0 이상 필수)

- **Java 21**: 모던한 Java 개발의 정석, **지금 가장 추천되는 LTS**

🔍 참고: Java 릴리즈 주기

- **2017 이후**: 6개월마다 정기 릴리즈
- **LTS 버전은 3년마다 제공**
  - LTS: 8, 11, 17, 21, (예정: 25)

개발 도구 설치 (Eclipse, IntelliJ, VSCode)

✅ 1. Eclipse IDE 설치

📌 개요

- Java 개발 전통의 강자
- Eclipse Foundation이 오픈소스로 운영
- 플러그인 기반 확장성 우수
- Spring, Android, JEE 개발 도구도 존재

💡 설치 절차

🔹 1. 설치 파일 다운로드

- 사이트: https://www.eclipse.org/downloads/
- 추천: **Eclipse IDE for Java Developers** 또는 **Eclipse IDE for Enterprise Java and Web Developers**

🔹 2. 설치

- 설치 프로그램 실행 → 설치 디렉토리 선택
- Java 설치 여부 자동 감지 (없으면 JDK 설치 유도)

🔹 3. 워크스페이스 설정

- 첫 실행 시 `workspace` 디렉토리 지정 필요
- 하나의 프로젝트 폴더 집합을 의미

🔹 4. 새 Java 프로젝트 생성

- File → New → Java Project → 이름 입력 → Finish

🔹 5. 소스코드 실행

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

- 오른쪽 클릭 → Run As → Java Application

✅ 장점

- 다양한 플러그인 (Spring, Maven, Git 등)
- 리소스 관리 효율적
- 무료, 오픈소스

❌ 단점

- UI와 UX가 다소 복잡하고 무거움
- Gradle 등 최신 도구와 통합이 IntelliJ보다 불편함

✅ 2. IntelliJ IDEA 설치

📌 개요

- JetBrains에서 개발
- 현재 **가장 인기 있는 Java IDE**
- 강력한 자동완성, 리팩토링, 디버깅, Gradle/Maven 완벽 통합

💡 설치 절차

🔹 1. 다운로드

- 사이트: https://www.jetbrains.com/idea/
- **무료 버전**: Community Edition (Java SE, Android 개발 가능)
- **유료 버전**: Ultimate Edition (Spring, JPA, Java EE, REST 등 포함)

🔹 2. 설치 및 실행

- 설치 파일 실행 후 기본 설정 유지
- 테마 선택 (Darcula or Light)

🔹 3. 프로젝트 생성

- New Project → Java 선택 → JDK 설정
- Maven/Gradle 프로젝트도 여기서 바로 가능

🔹 4. 코드 작성 및 실행

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

- Run 버튼 클릭 또는 `Shift + F10`

✅ 장점

- 스마트한 코드 자동완성 (`Ctrl + Space`)
- 리팩토링, 디버깅, 빌드 자동화까지 지원
- Git, Docker, Database 등 통합 도구 완벽 지원

❌ 단점

- 유료 기능 제한 (Ultimate 필요)
- Eclipse에 비해 상대적으로 무거움

✅ 3. Visual Studio Code + Java

📌 개요

- Microsoft의 경량 코드 편집기
- VSCode 자체는 IDE가 아니지만, Java 개발을 위한 확장 지원
- JDK, 빌드툴, 확장 설치 필요

💡 설치 절차

🔹 1. VSCode 설치

- 사이트: https://code.visualstudio.com/

🔹 2. Java 환경 설치

- **JDK 설치**: OpenJDK 17+ 설치
- **빌드툴 설치**: Maven or Gradle 추천
- **PATH 설정**: `java` 명령이 동작해야 함

🔹 3. Java 확장팩 설치

- 확장 탭(좌측) → "Java Extension Pack" 검색 후 설치
  포함 구성:
  - Language Support for Java™
  - Debugger for Java
  - Java Test Runner
  - Maven for Java
  - IntelliCode

🔹 4. 프로젝트 실행

- `main.java` 작성 후 `Ctrl + F5` 또는 우클릭 → Run Java

✅ 장점

- 가볍고 빠름 (경량 에디터)
- 플러그인 기반 유연성
- Java + TypeScript + Python 등 멀티 언어 개발에 적합

❌ 단점

- 완전한 IDE 기능은 부족 (리팩토링, 코드 분석 등은 IntelliJ 미만)
- 설정 복잡도 약간 존재 (JDK, 빌드툴 직접 설정 필요)

🔁 요약 비교

항목	Eclipse	IntelliJ IDEA	VSCode (with Java)
개발사	Eclipse Foundation	JetBrains	Microsoft
라이선스	완전 무료	Community(무료), Ultimate(유료)	무료
성능	중간	무거움 (기능 많음)	가벼움
자동완성	보통	매우 우수	좋음
설정 난이도	쉬움	매우 쉬움	중간
빌드툴 통합	Maven/Gradle 지원	Maven/Gradle 최상 지원	외부 설정 필요
UI/UX	복잡	직관적	현대적
추천 대상	전통 Java 개발자	전문 Java 개발자	Java 입문자, 멀티 개발자

🧭 무엇을 선택해야 할까?

상황	추천 IDE
Java만 집중적으로 개발	**IntelliJ IDEA (Community/Ultimate)**
오픈소스 기반 + 가벼운 사용	**Eclipse**
Python/Node.js/웹 개발도 같이 한다	**VSCode**

첫 번째 프로그램 (`Hello, World`)

✅ 1. 소스 코드 구조

```
// Hello.java

public class Hello {
    public static void main(String[] args) {
```

```
        System.out.println("Hello, World!");
    }
}
```

✅ 2. 코드 설명

부분	설명
`public class Hello`	Java는 모든 코드가 **클래스** 안에 있어야 함. `Hello` 라는 이름의 클래스를 정의.
`public static void main(String[] args)`	Java 프로그램의 **시작점(main method)**. JVM은 이 메서드부터 실행함.
`System.out.println(...)`	표준 출력(콘솔)에 문자열을 출력함.

> 🔸 `System.out` → 표준 출력 스트림
> 🔸 `println(...)` → 문자열 출력 후 줄바꿈

✅ 3. 컴파일과 실행 (터미널/명령 프롬프트 사용)

🔹 전제: JDK 설치되어 있고 `java`, `javac` 명령이 동작해야 함

📁 예제 디렉토리

```
/workspace/
  └── Hello.java
```

🔸 컴파일 단계

```
javac Hello.java
```

- `Hello.class` 라는 **바이트코드 파일** 생성됨
- `javac` 는 Java Compiler로 `.java` → `.class` 변환

🔸 실행 단계

```
java Hello
```

- JVM이 `Hello.class` 의 `main()` 메서드를 실행
- 결과:

```
Hello, World!
```

> 💡 `java Hello.java` 처럼 소스 파일 직접 실행은 Java 11+에서만 지원됨

✅ 4. 오류 발생 예시

🔸 파일명과 클래스명이 다를 경우

```
// 파일명: Hello.java
public class MyClass { ... }
```

- 컴파일 시 오류 발생:

```
Error: class MyClass is public, should be declared in a file named MyClass.java
```

🔸 main 메서드가 없을 경우

```
public class Hello { }
```

- 실행 시 오류:

```
Error: Main method not found in class Hello, please define the main method as:
   public static void main(String[] args)
```

✅ 5. IDE에서 실행 (예: IntelliJ / Eclipse / VSCode)

- 새 Java 프로젝트 생성 후 `Hello.java` 작성
- Run 버튼 클릭 또는 `Ctrl + Shift + F10` (IntelliJ 기준)
- 콘솔에 “Hello, World!” 출력됨

🔄 정리

항목	내용
Java의 최소 실행 단위	클래스 + `main()` 메서드
컴파일 명령	`javac Hello.java`
실행 명령	`java Hello`
출력 함수	`System.out.println()`