OOP WITH JAVA

[KDT]AI&빅데이터 분석활용 풀스택 개발

담당 : 박경미

전체 목차

1. 자바프로그래밍 시작

목차

목차

프로그래밍 언어

❖ 프로그램 작성 언어

❖ 기계어(machine language)

- 0, 1의 이진수로 구성된 언어
- 컴퓨터의 CPU는 기계어만 이해하고 처리가능

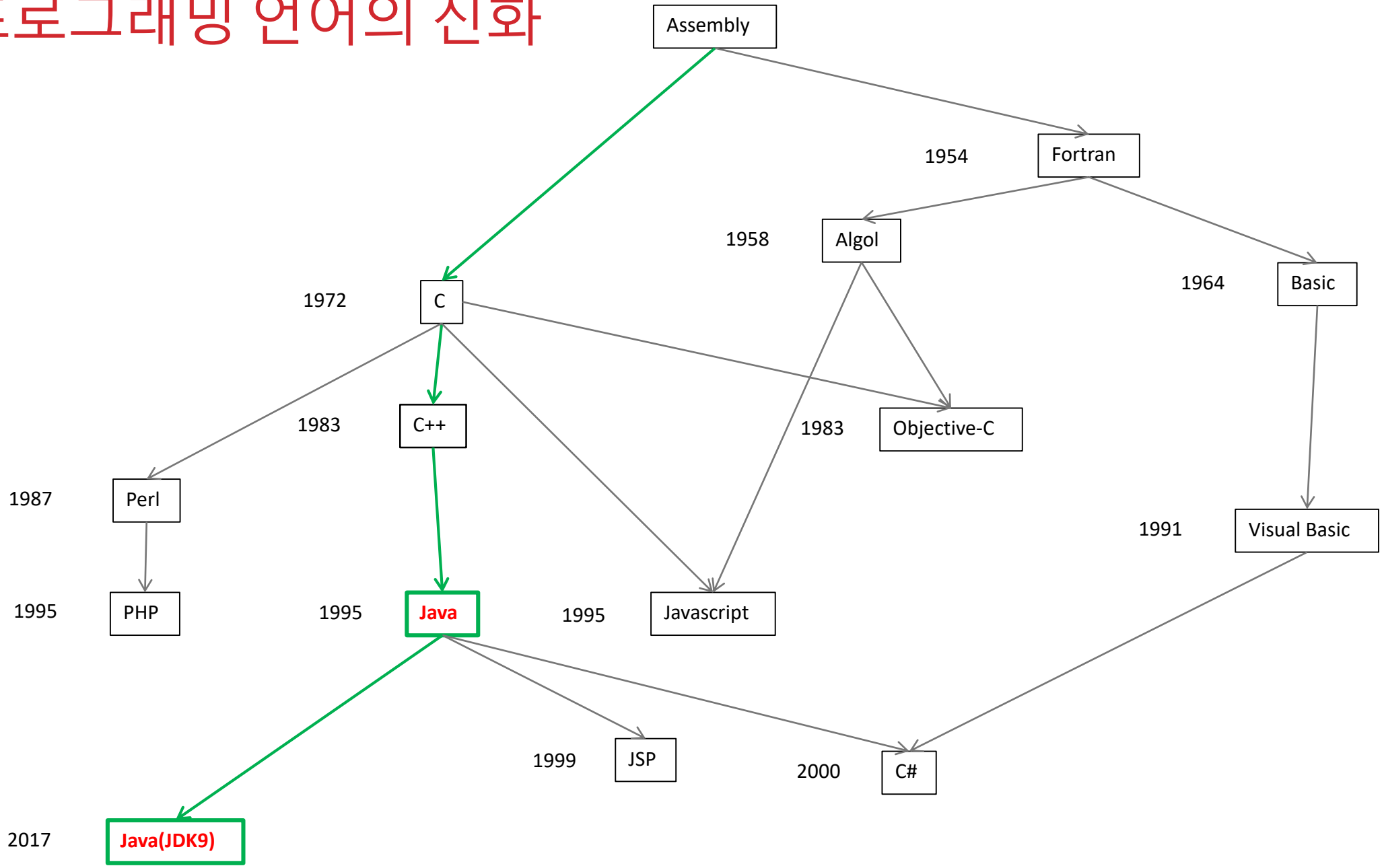
❖ 어셈블리어

- 기계어 명령을 ADD, SUB, MOVE 등과 같은 표현하기 쉬운 상징적인 단어인 니모닉 기호(mnemonic symbol)로 일대일 대응시킨 언어

❖ 고급언어

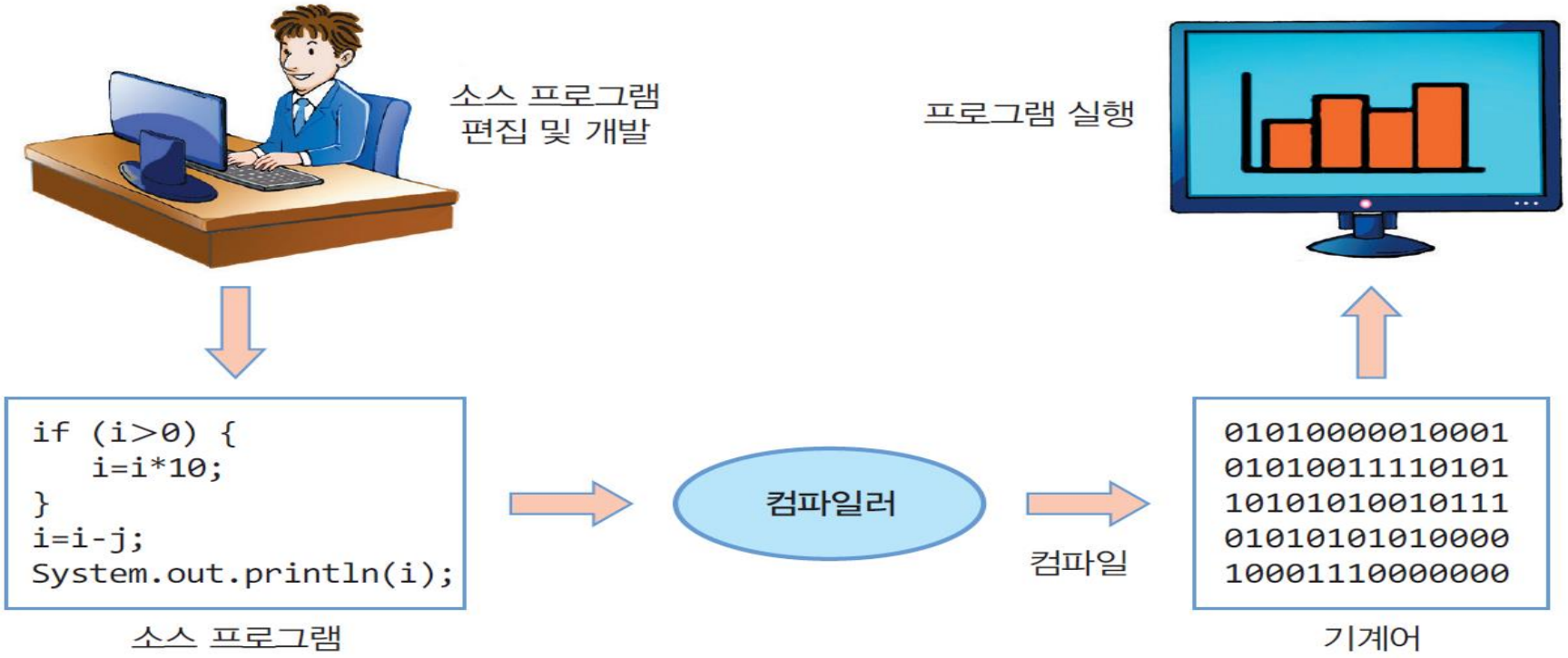
- 사람이 이해하기 쉽고, 복잡한 작업, 자료 구조,알고리즘을 표현하기 위해 고안된 언어
- Pascal, Basic, C/C++, Java, C#
- 절차 지향 언어와 객체 지향 언어

프로그래밍 언어의 진화


Assembly
1954
Fortran
1958
Algol
1964
Basic
1972
C
1983
C++
1983
Objective-C
1987
Perl
1991
Visual Basic
1995
PHP
1995
Java
1995
Javascript
1999
JSP
2000
C#
2017
Java(JDK9)

컴파일

- ❖ **소스 : 프로그래밍 언어로 작성된 텍스트 파일**
- ❖ **컴파일 : 소스 파일을 컴퓨터가 이해할 수 있는 기계어로 만드는 과정**
  - 소스 파일 확장자와 컴파일 된 파일의 확장자
    - **자바 : .java -> .class**
    - **C : .c -> .obj-> .exe**
    - **C++ : .cpp -> .obj -> .exe**



자바의 태동

❖ 1991년 그린 프로젝트(Green Project)

- 선마이크로시스템즈의 제임스 고슬링(James Gosling)에 의해 시작
- 가전 제품에 들어갈 소프트웨어를 위해 개발
- 1995년에 자바 발표

❖ 목적

- 플랫폼 호환성 문제 해결
  - **기존 언어로 작성된 프로그램은 PC, 유닉스, 메인 프레임 등 플랫폼 간에 호환성 없음**
  - **소스를 다시 컴파일하거나 프로그램을 재 작성해야 하는 단점**
- 플랫폼 독립적인 언어 개발
  - **모든 플랫폼에서 호환성을 갖는 프로그래밍 언어 필요**
  - **네트워크, 특히 웹에 최적화된 프로그래밍 언어의 필요성 대두**
- 메모리 사용량이 적고 다양한 플랫폼을 가지는 가전 제품에 적용
  - **가전 제품 : 작은 량의 메모리를 가지는 제어 장치**
  - **내장형 시스템 요구 충족**

❖ 초기 이름 : 오크(OAK)

- 인터넷과 웹의 엄청난 발전에 힘입어 퍼지게 됨
- 웹 브라우저 Netscape에서 실행

❖ 2009년에 선마이크로시스템즈를 오라클에서 인수

WORA

❖ WORA(Write Once Run Anywhere)

- 한번 작성된 코드는 모든 플랫폼에서 바로 실행
- C/C++ 등 기존 언어가 가진 플랫폼 종속성 극복
  - **OS, H/W에 상관없이 자바 프로그램이 동일하게 실행**
- 네트워크에 연결된 어느 클라이언트에서나 실행
  - **웹 브라우저, 분산 환경 지원**

❖ WORA를 가능하게 하는 자바의 특징

- 바이트 코드(byte code)
  - **자바 소스를 컴파일한 목적 코드**
  - **CPU에 종속적이지 않은 중립적인 코드**
  - **JVM에 의해 해석되고 실행됨**
- JVM(Java Virtual Machine)
  - **자바 바이트 코드를 실행하는 자바 가상 기계(소프트웨어)**

플랫폼 종속성(PLATFORM DEPENDENCY)

플랫폼 = 하드웨어 플랫폼 + 운영체제 플랫폼

프로그램의 플랫폼 호환성 없는 이유
- 기계어가 CPU마다 다름
- 운영체제마다 API 다름
- 운영체제마다 실행파일 형식 다름

인텔 CPU를 가진 리눅스 환경에서 개발

C/C++ 프로그램 → (컴파일) → 기계어

실행

인텔 CPU + 리눅스

실행되지 않음

Apple 사의 MAC PC

실행되지 않음

인텔 CPU + 윈도우 노트북

실행되지 않음

임베디드 컴퓨터 + 리눅스

자바의 플랫폼 독립성, WORA

Write Once !!

자바 응용프로그램

바이트코드

자바 가상 기계가 설치된 모든 컴퓨터에서 실행됨

Run Anywhere!!

실행

실행

실행

실행

자바 가상 기계

인텔 CPU + 리눅스

자바 가상 기계

Apple 사의 MAC PC

자바 가상 기계

인텔 CPU +
윈도우 노트북

자바 가상 기계

임베디드 컴퓨터 + 리눅스

바이트 코드와 자바 가상 기계

❖ 바이트 코드

- 자바 가상 기계에서 실행 가능한 바이너리 코드
  - **바이트 코드는 컴퓨터 CPU에 의해 직접 실행되지 않음**
  - **자바 가상 기계가 작동 중인 플랫폼에서 실행**
  - **자바 가상 기계가 인터프리터 방식으로 바이트 코드 해석**
- 클래스 파일(.class)에 저장

❖ 자바 가상 기계(JVM : Java Virtual Machine)

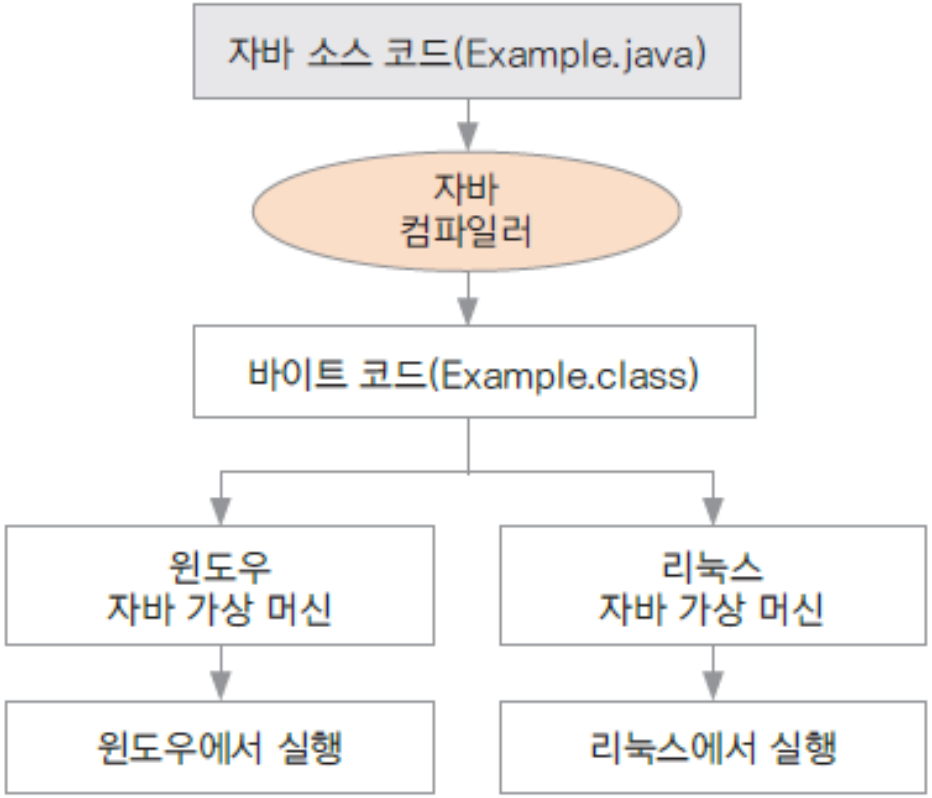
- 동일한 자바 실행 환경 제공
  - **각기 다른 플랫폼에 설치**
- 자바 가상 기계 자체는 플랫폼에 종속적
  - **자바 가상 기계는 플랫폼마다 각각 작성됨**
  - **예) 리눅스에서 작동하는 자바 가상 기계는 윈도우에서 작동하지 않음**
- 자바 가상 기계 개발 및 공급
  - **자바 개발사인 오라클 외 IBM, MS 등 다양한 회사에서 제작 공급**

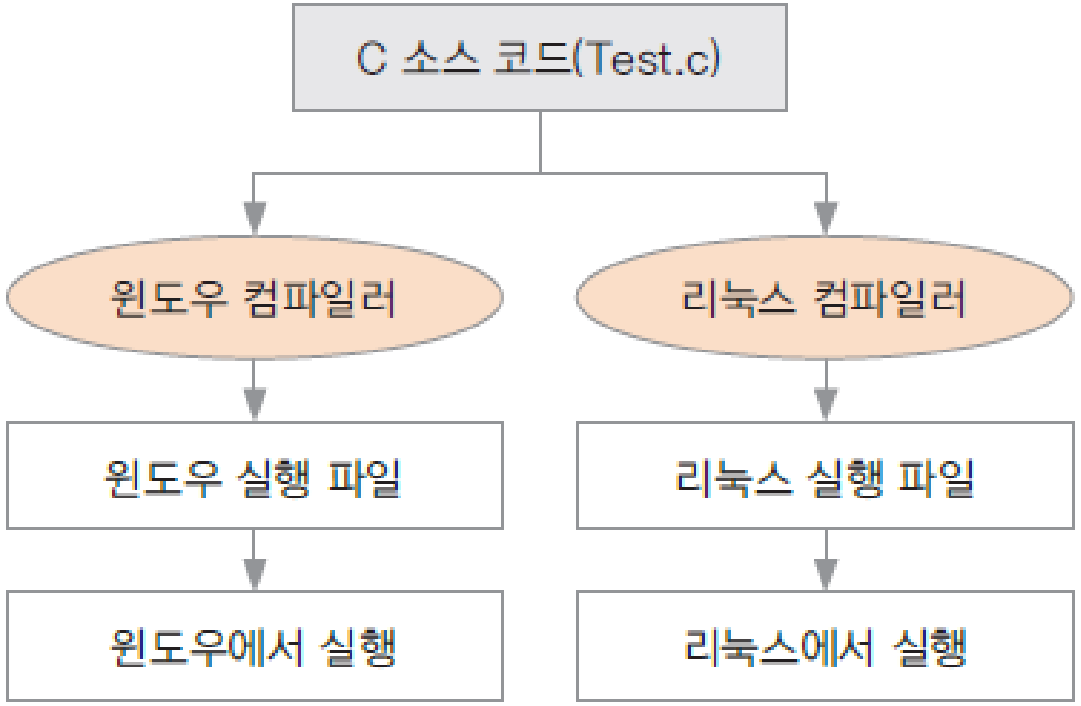
❖ 자바의 실행

- 자바 가상 기계가 클래스 파일(.class)의 바이트 코드 실행

자바를 쓰면 왜 좋을까요?

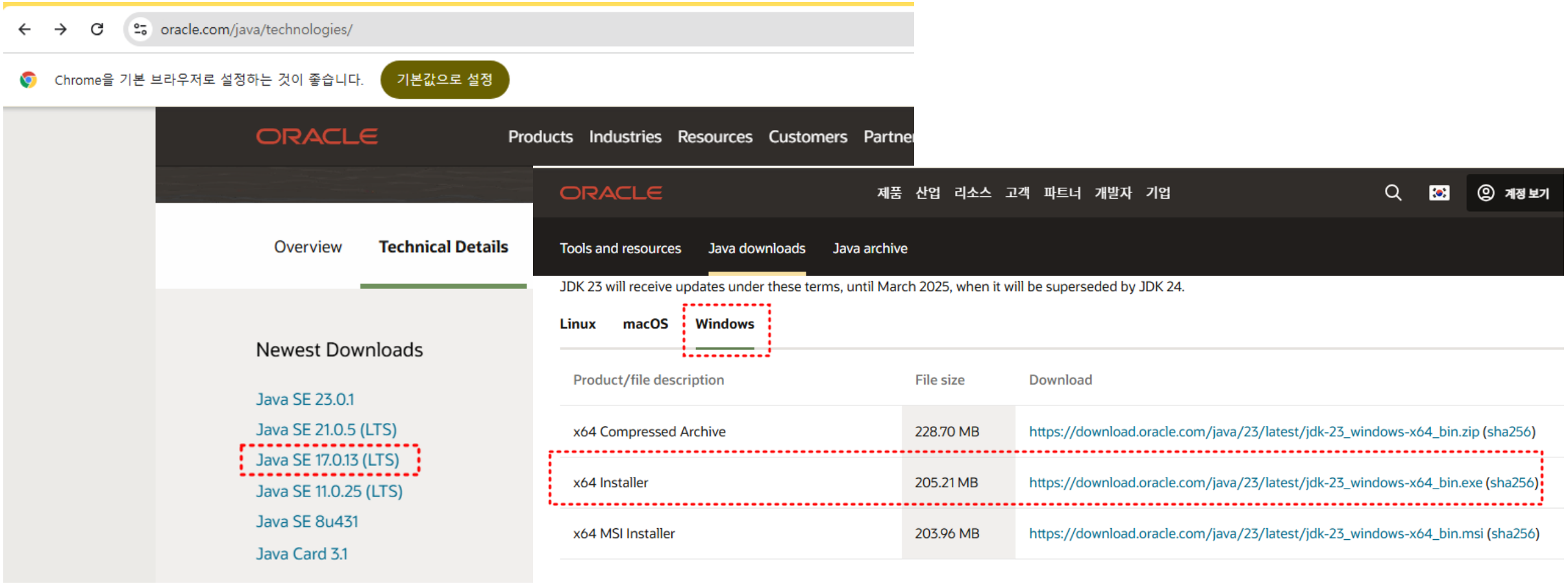
❖ 플랫폼에 영향을 받지 않으므로 다양한 환경에서 사용





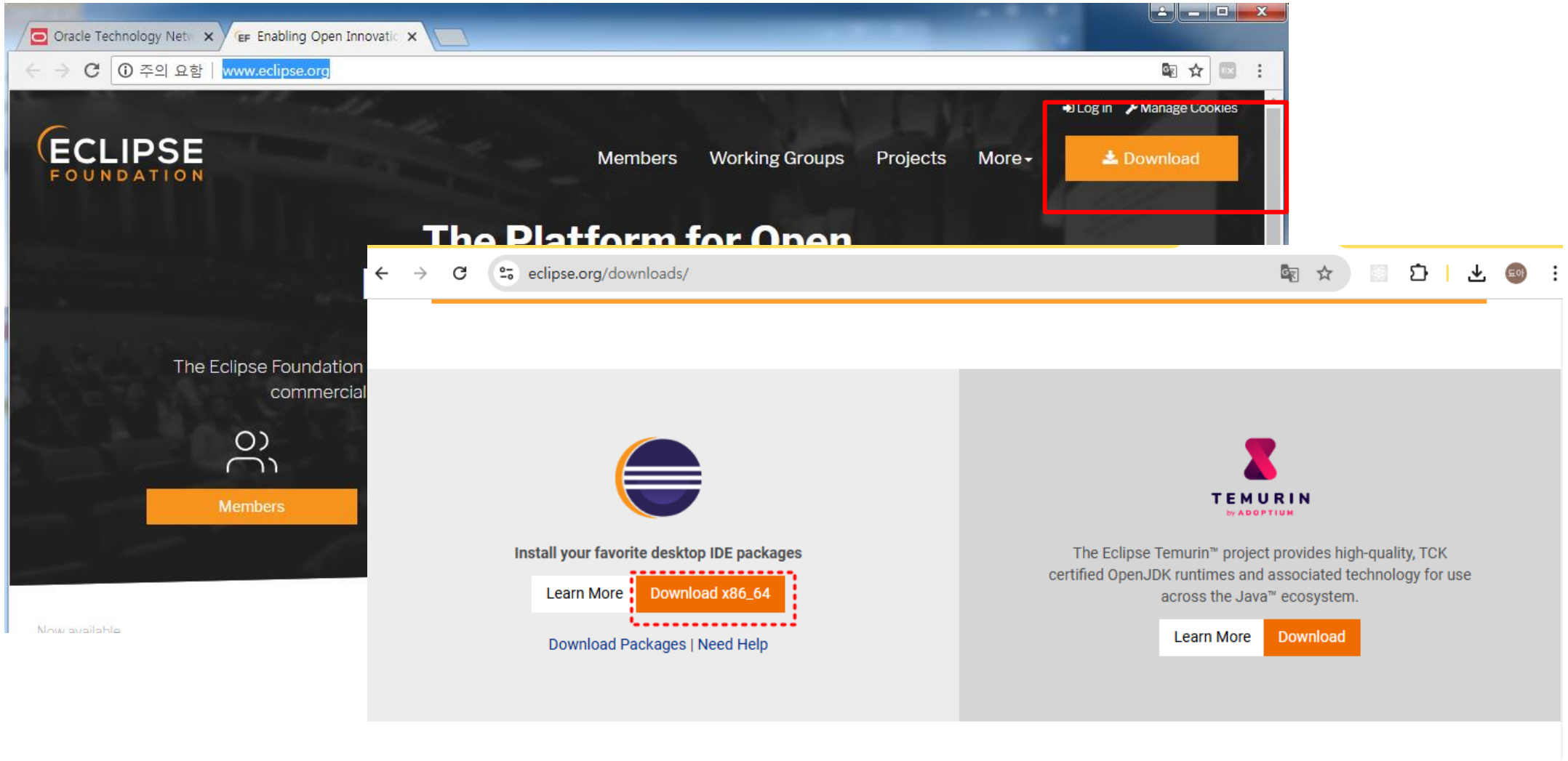
자바 설치 하기

❖ http://www.oracle.com/technologies/

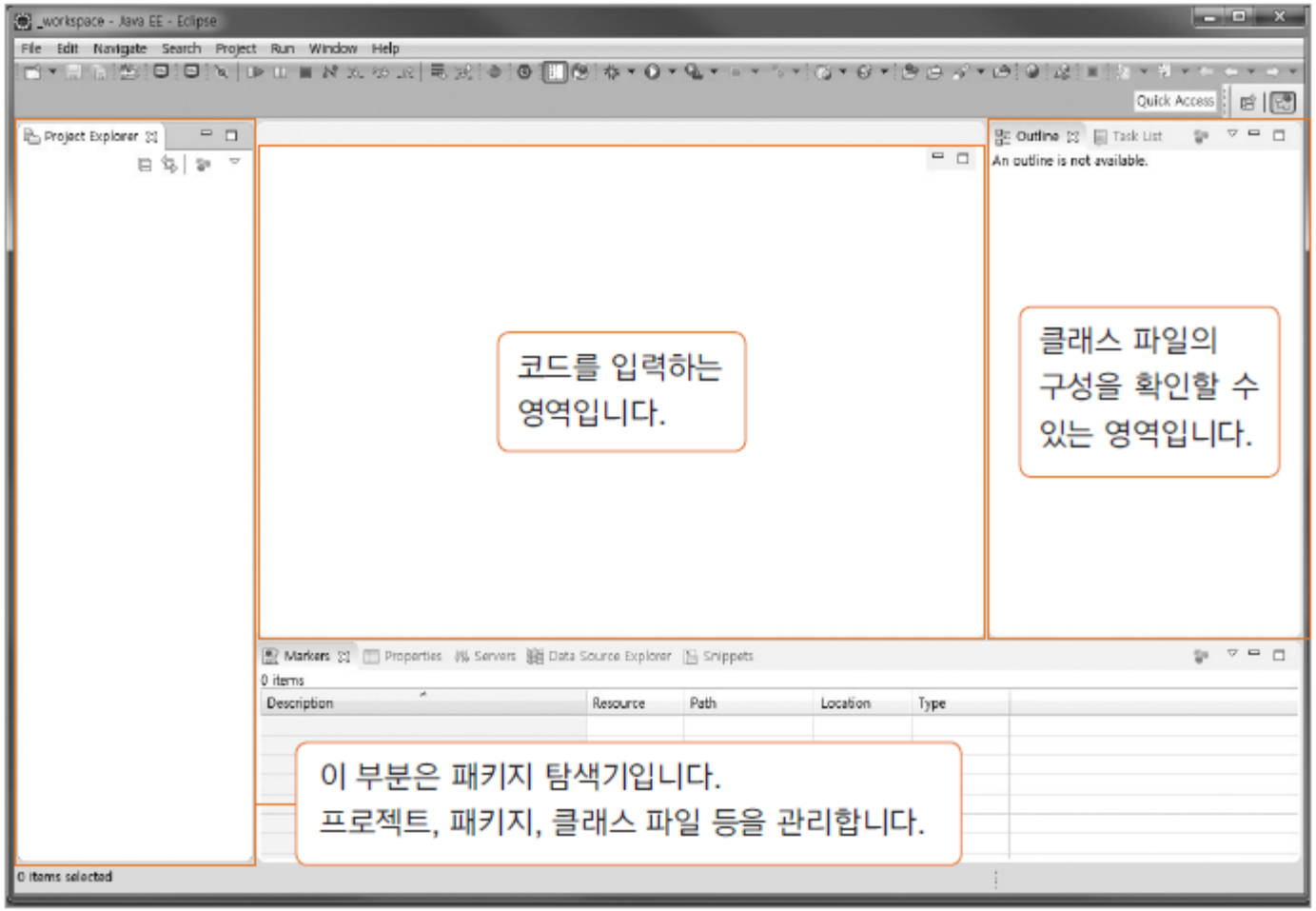


이클립스 설치 하기

- http://www.eclipse.org/



첫 번째 프로그램 – Hello, Java



자바 프로그램 구조

```
/*
*  소스 파일 : Hello.java
*/
public class Hello {

    public static int sum(int n, int m) {
        return n + m;
    }

    // main() 메소드에서 실행 시작
    public static void main(String[] args) {
        int i = 20;
        int s;
        char a;

        s = sum(i, 10); // sum() 메소드 호출
        a = '?';
        System.out.println(a); // 문자 '?' 화면 출력
        System.out.println("Hello"); // "Hello" 문자열 화면 출력
        System.out.println(s); // 정수 s 값 화면 출력
    }
}
```

메소드

메소드

클래스

```
?
Hello
30
```

자바의 특성(1)

❖ **플랫폼 독립성**

- 자바 프로그램은 플랫폼에 상관없이 어디서든지 실행

❖ **객체지향**

- 상속성, 다형성, 캡슐화

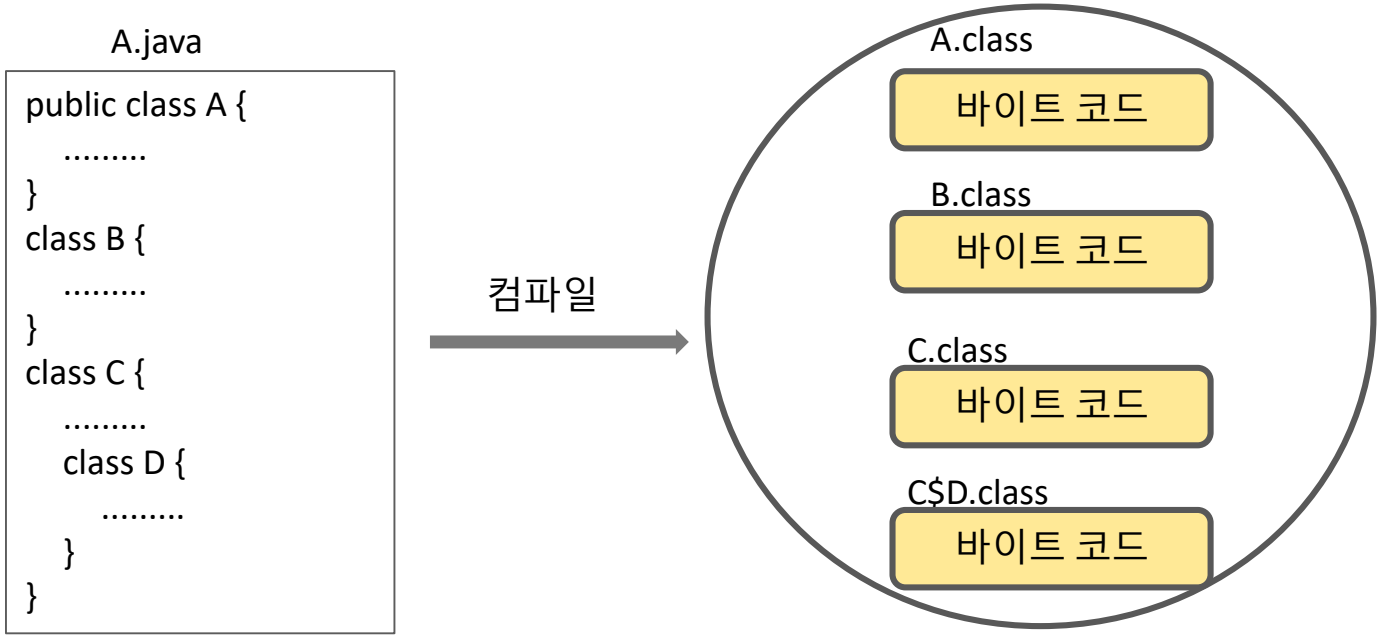
❖ **클래스로 캡슐화**

- 클래스 내에 모든 변수(필드), 함수(메소드) 구현해야 함
- 클래스 안에서 새로운 클래스(내부 클래스) 작성 가능

자바의 특성(2)

❖ **소스(.java)와 클래스(.class) 파일**

- 하나의 소스 파일에 여러 클래스 작성 가능
  - **public 클래스는 하나만 가능**
  - **소스 파일의 이름과 public으로 선언된 클래스 이름은 같아야 함**
- 컴파일된 클래스 파일(.class)에는 클래스는 하나만 존재
  - **다수의 클래스를 가진 자바 소스(.java)를 컴파일하면 클래스마다 별도 클래스 파일(.class) 생성**



4 개의 클래스 파일이 생성됨

소스 파일과 클래스, 클래스 파일의 관계

자바의 특징(3)

❖ 실행 모듈

- 한 개의 class 파일 또는 다수의 class 파일로 구성
- 여러 폴더에 걸쳐 다수의 클래스 파일로 구성된 경우 : jar 파일 형태로 배포 가능
- main() 메소드 : 자바 응용프로그램의 실행은 main() 메소드에서 시작
- 하나의 클래스 파일에 하나 이상의 main() 메소드가 있을 수 없음
  - **각 클래스 파일이 main() 메소드를 포함하는 것은 상관없음**

❖ 패키지

- 관련된 여러 클래스를 패키지로 묶어 관리
- 패키지는 폴더 개념 : 예) java.lang.System은 java₩lang 디렉터리의 System.class 파일

❖ 멀티스레드

- 자바는 운영체제의 도움 없이 자체적으로 멀티스레드 지원
  - **C/C++ 등에서는 멀티스레드 운영체제 API를 호출**

자바의 특징(4)

- **가비지 컬렉션**
  - 자바는 응용 프로그램에서 메모리 반환 기능 없음, 메모리 할당 기능(new)만 있음- 개발자의 부담 대폭 감소
  - 가비지 : 할당 후 사용되지 않는 메모리
  - 자바 가상 기계가 자동으로 가비지 회수
- **실시간 응용 시스템에 부적합**
  - 자바 응용프로그램은 실행 도중 예측할 수 없는 시점에 가비지 컬렉션 실행
  - 일정 시간(deadline) 내에 반드시 실행 결과를 내야만 하는 실시간 시스템에는 부적합
- **자바 프로그램은 안전**
  - 타입 체크가 매우 엄격
  - 포인터의 개념 없음
- **프로그램 작성이 쉬움**
  - 포인터 개념이 없어 부담 적음
  - 다양하고 강력한 라이브러리가 많음
- **실행 속도를 개선하기 위해 JIT 컴파일러 사용**
  - 자바의 느린 실행 요인 : 인터프리터 방식으로 바이트 코드 실행
  - JIT(Just in Time) 컴파일링 기법으로 개선 :
    - **실행 도중 바이트 코드를 해당 CPU의 기계어 코드로 컴파일, 해당 CPU가 기계어를 실행**