자바프로그래밍

컴퓨터공학전공
박요한

강의 목차

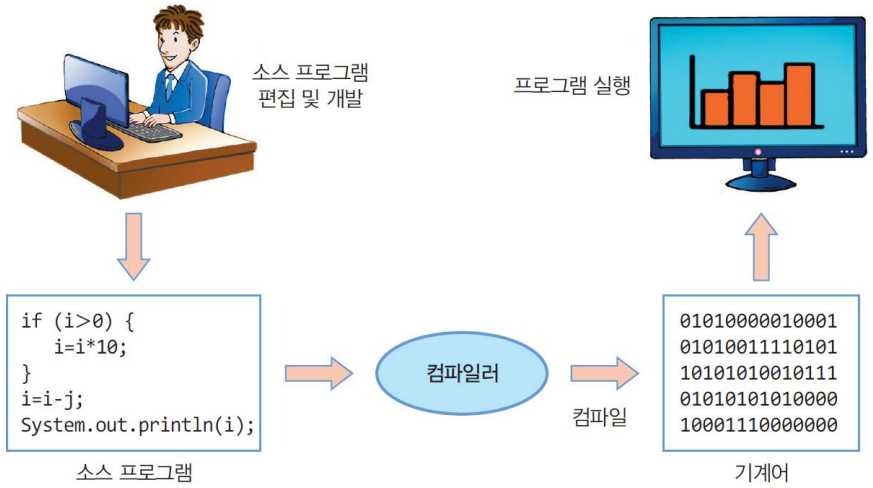
- 자바의 개요
- 자바 개발 환경
- 자바 사용법

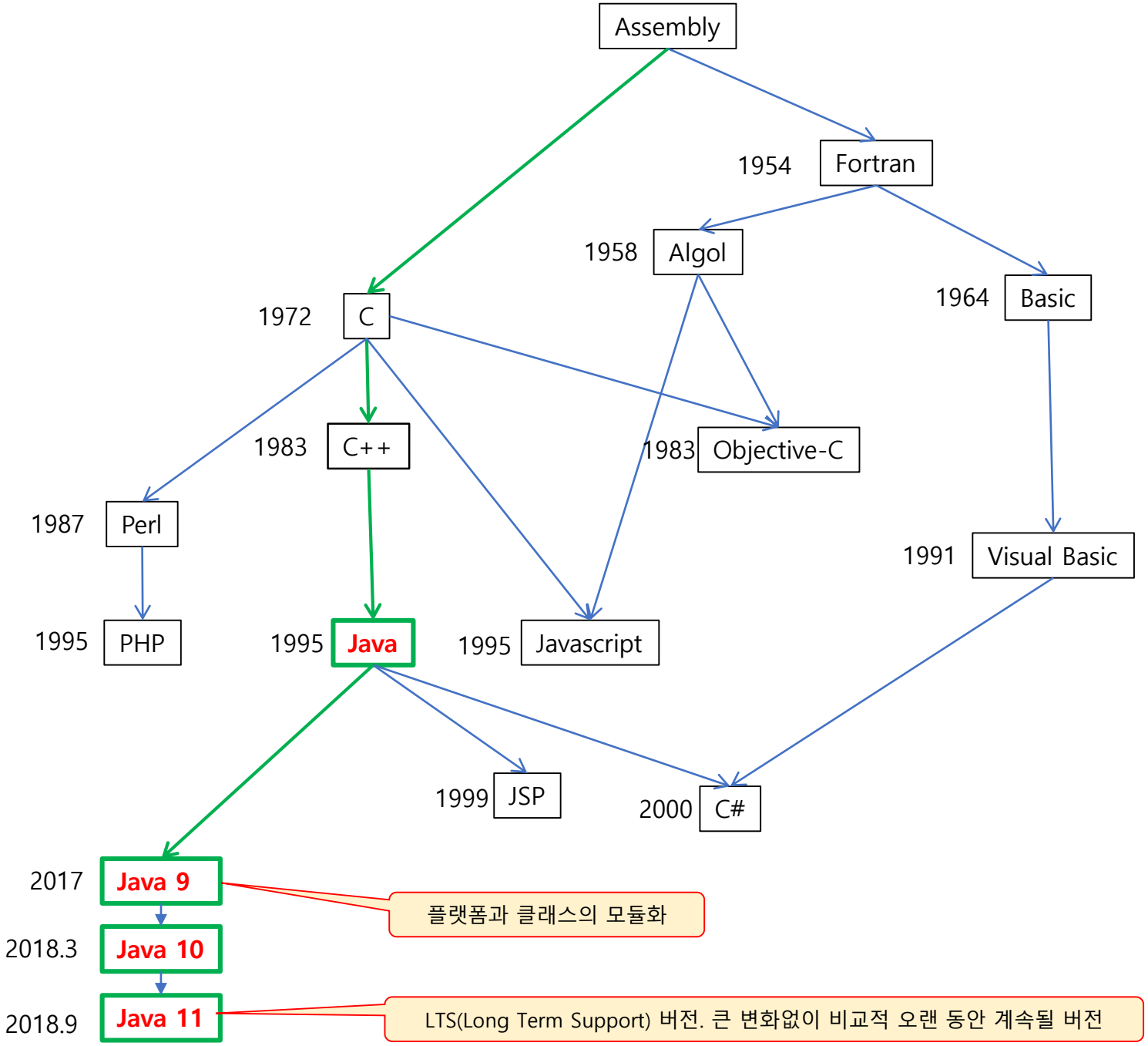
프로그래밍 언어

- 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

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




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 9
플랫폼과 클래스의 모듈화
2018.3 Java 10
2018.9 Java 11
LTS(Long Term Support) 버전. 큰 변화없이 비교적 오랜 동안 계속될 버전

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

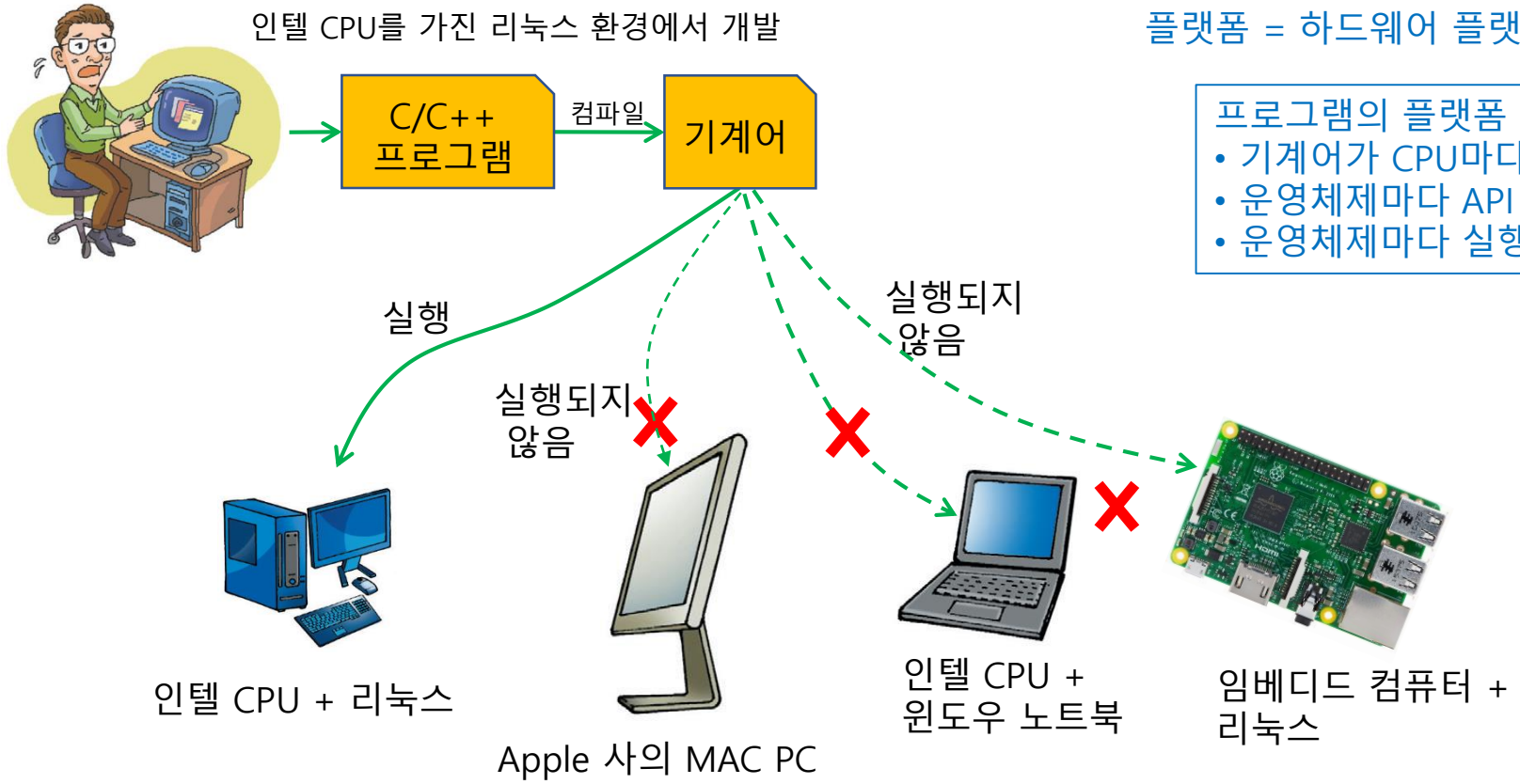
자바의 특징

- JVM 기방네서 작동하는 OOP 언어
- Native languages(C, C++)의 메모리 관리와 책임이슈를 제거
- OS(Platform)에 대한 의존성 없음
- 인터프리터 특징을 가짐

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)

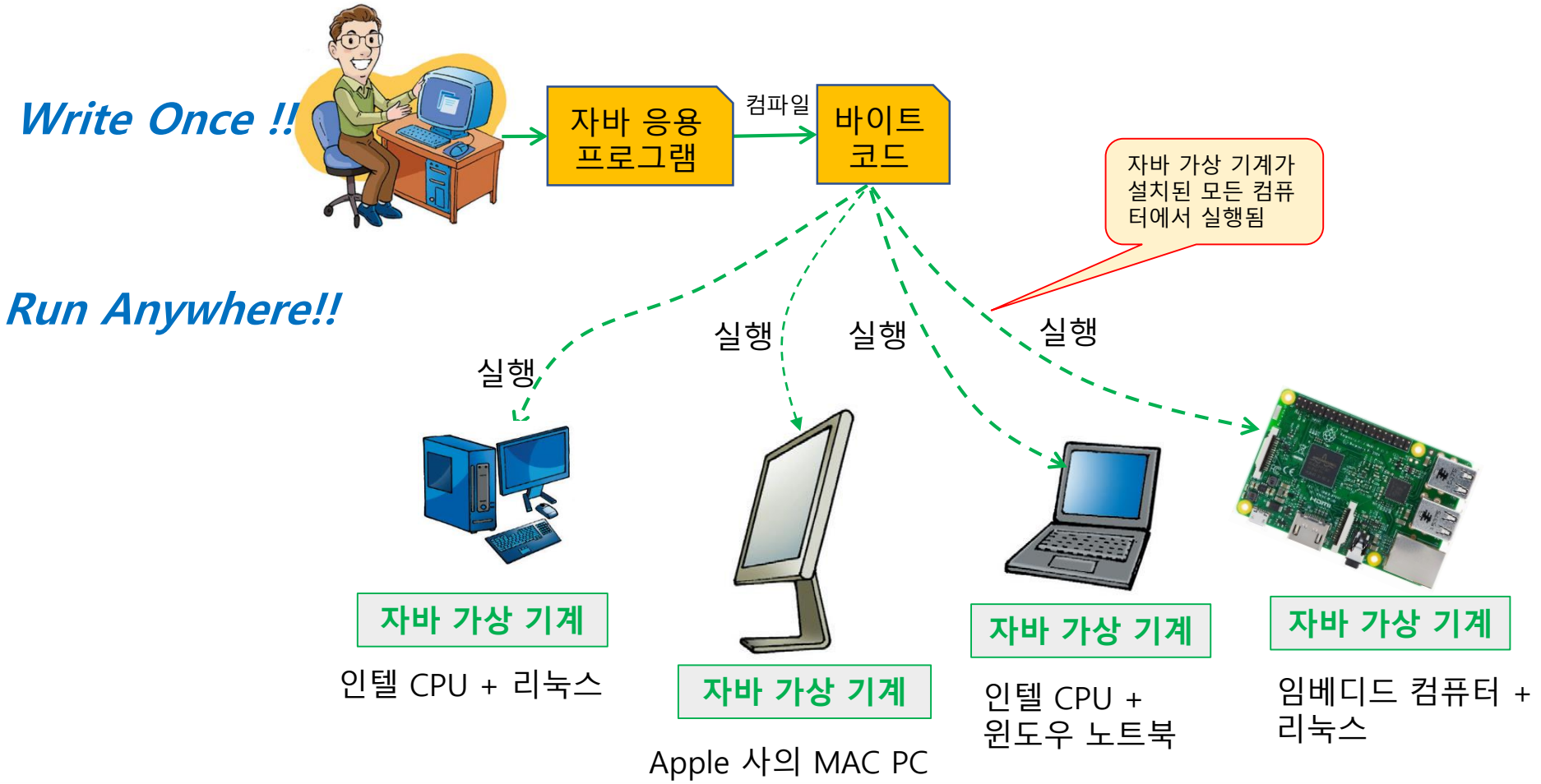
플랫폼 종속성(platform dependency)



플랫폼 = 하드웨어 플랫폼 + 운영체제 플랫폼

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

자바의 플랫폼 독립성, WORA



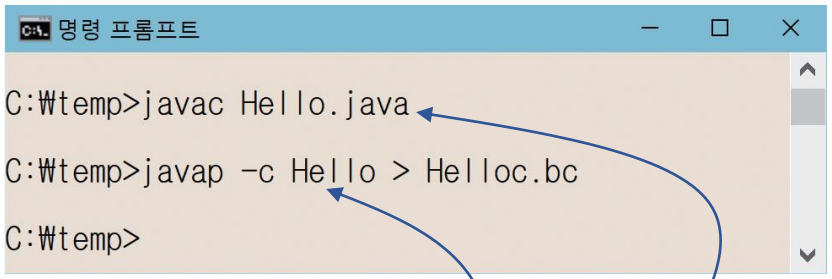
바이트 코드와 자바 가상 기계

- 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, IBM 등
- 자바 응용프로그램 실행
  - ✓자바 가상 기계가 응용프로그램을 구성하는 클래스 파일(.class)의 바이트 코드 실행

바이트 코드의 디어셈블(disassemble)

- 디어셈블
  - ✓클래스 파일에 들어 있는 바이트 코드를 텍스트로 볼 수 있게 변환하는 작업
  - ✓JI

```
public class Hello {
    public static int sum(int i, int j) {
        return i + j;// i와 j의 합을 리턴
    }
    public static void main(String[] args) {
        int i;
        int j;
        char a;
        String b;
        final int TEN = 10;
        i = 1;
        j = sum(i, TEN);
        a = '?';
        b = "Hello";
        java.lang.System.out.println(a);
        System.out.println(b);
        System.out.println(TEN);
        System.out.println(j);
    }
}
```



- Hello.java를 컴파일하는 명령
- 컴파일되면 Hello.class 생성

- Hello.class 파일을 디어셈블하는 명령
- 디어셈블된 결과 Hello.bc 파일 생성

디어셈블하여 바이트 코드 보기

Helloc.bc - 메모장

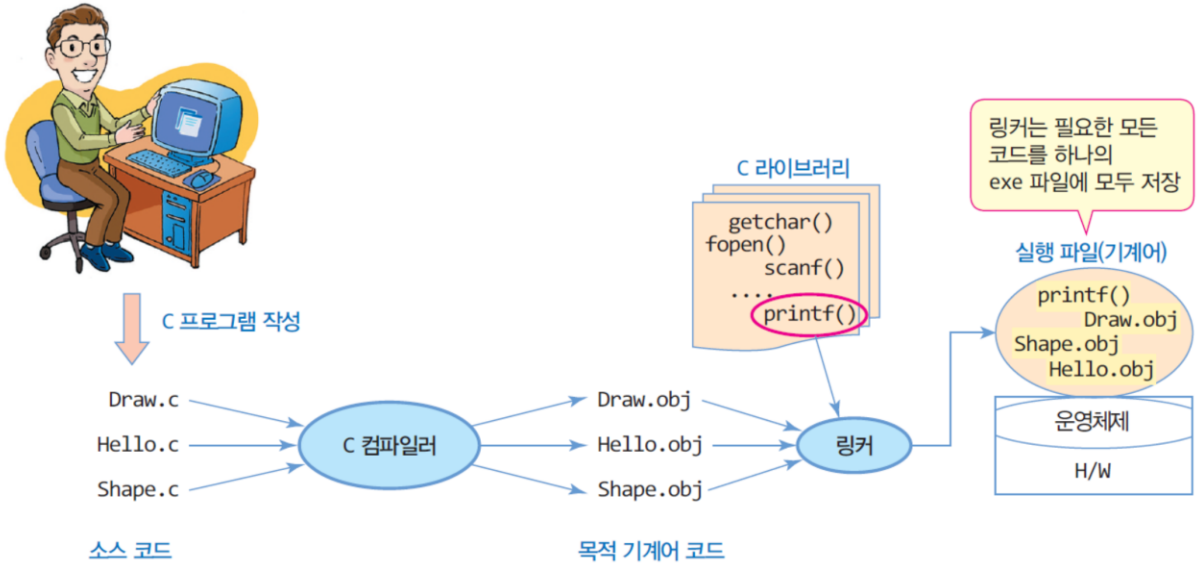
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

```
Compiled from "Hello.java"
public class Hello {
  public Hello();
    Code:
       0: aload_0
       1: invokespecial #1                  // Method java/lang/Object."<init>":()V
       4: return

  public static int sum(int, int);
    Code:
       0: iload_0
       1: iload_1
       2: iadd
       3: ireturn

  public static void main(java.lang.String[]);
    Code:
       0: iconst_1
       1: istore_1
       2: iload_1
       3: bipush        10
       5: invokestatic  #2                  // Method sum:(II)I
       8: istore_2
       9: bipush        63
      11: istore_3
      12: ldc           #3                  // String Hello
      14: astore        4
      16: getstatic     #4                  // Field java/lang/System.out:Ljava/io/PrintStream;
      19: iload_3
      20: invokevirtual #5                  // Method java/io/PrintStream.println:(C)V
      23: getstatic     #4                  // Field java/lang/System.out:Ljava/io/PrintStream;
      26: aload         4
      28: invokevirtual #6                  // Method java/io/PrintStream.println:(Ljava/lang/String;)V
      31: getstatic     #4                  // Field java/lang/System.out:Ljava/io/PrintStream;
      34: bipush        10
      36: invokevirtual #7                  // Method java/io/PrintStream.println:(I)V
      39: getstatic     #4                  // Field java/lang/System.out:Ljava/io/PrintStream;
      42: iload_2
      43: invokevirtual #7                  // Method java/io/PrintStream.println:(I)V
      46: return
}
```

sum() 메소드를 컴파일한 바이트 코드를 디어셈블한 결과(자바의 어셈블리 코드로 출력)

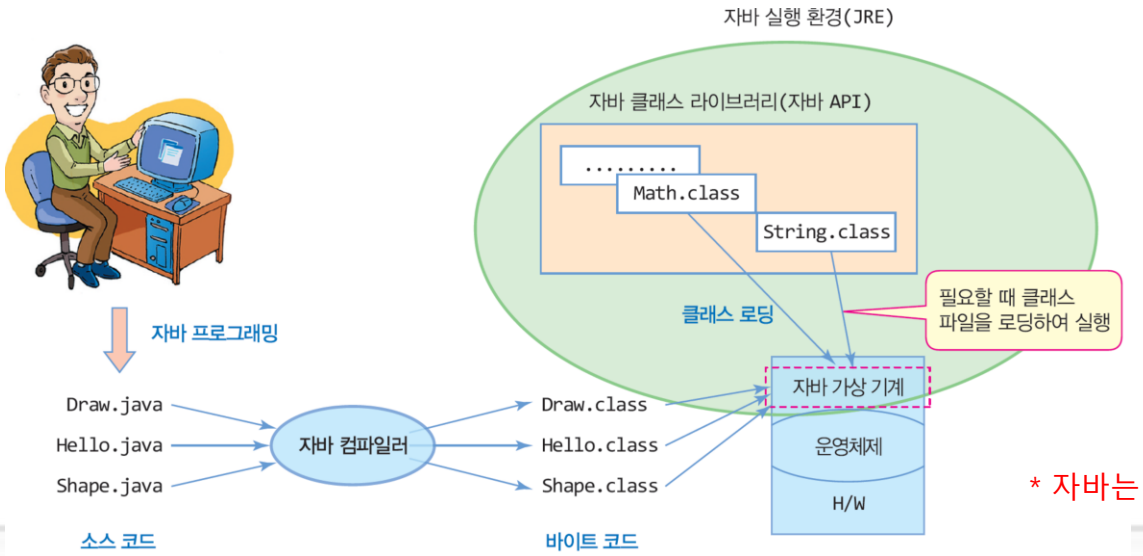
C/C++ 프로그램의 개발 및 실행 환경

- C/C++ 프로그램의 개발
  - ✓ 여러 소스(.c) 파일로 나누어 개발
  - ✓ 링크를 통해 실행에 필요한 모든 코드를 하나의 실행 파일(.exe)에 저장
- 실행
  - ✓ 실행 파일(exe)은 모두 메모리에 올려져야 실행, 메모리가 적은 경우 낭패



자바의 개발 및 실행 환경

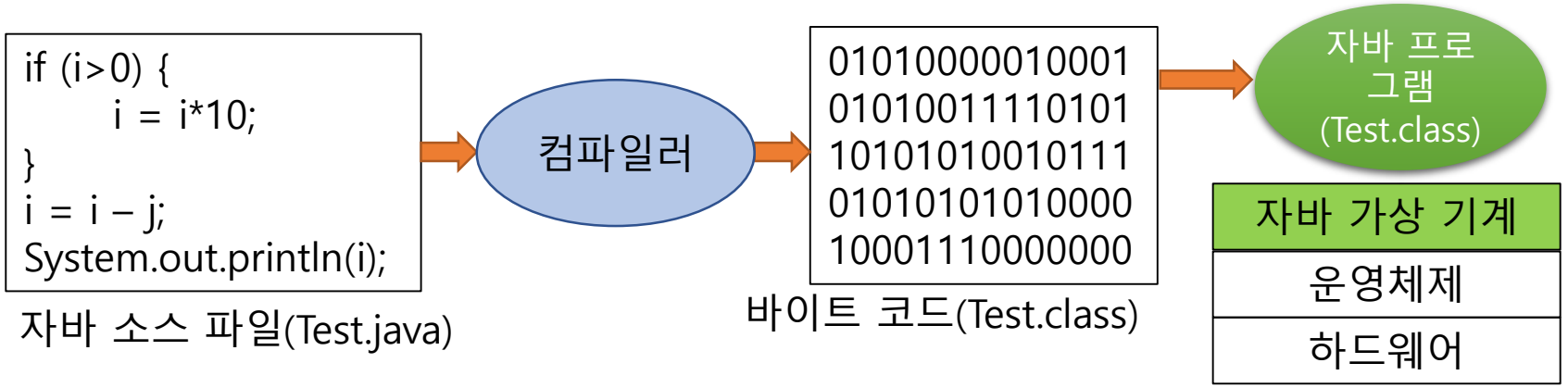
- 자바 프로그램의 개발
  - ✓ 여러 소스(.java)로 나누어 개발
  - ✓ 바이트 코드(.class)를 하나의 실행 파일(exe)로 만드는 링크 과정 없음
- 실행
  - ✓ main() 메소드를 가진 클래스에서 부터 실행 시작
  - ✓ 자바 가상 기계는 필요할 때, 클래스 파일 로딩, 적은 메모리로 실행 가능



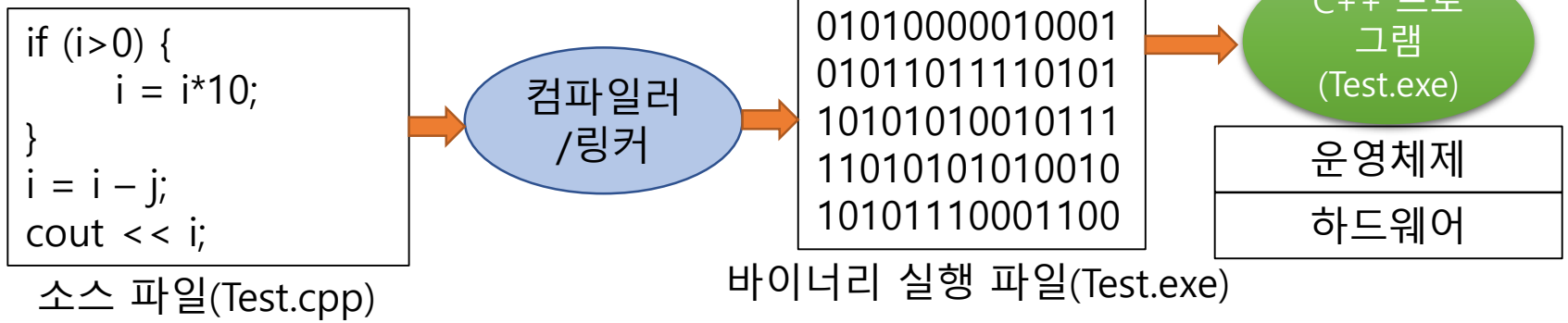
* 자바는 하나의 실행 파일(exe)로 만드는 링크 과정 없음

자바와 C/C++의 실행 환경 차이

- 자바



- C/C++

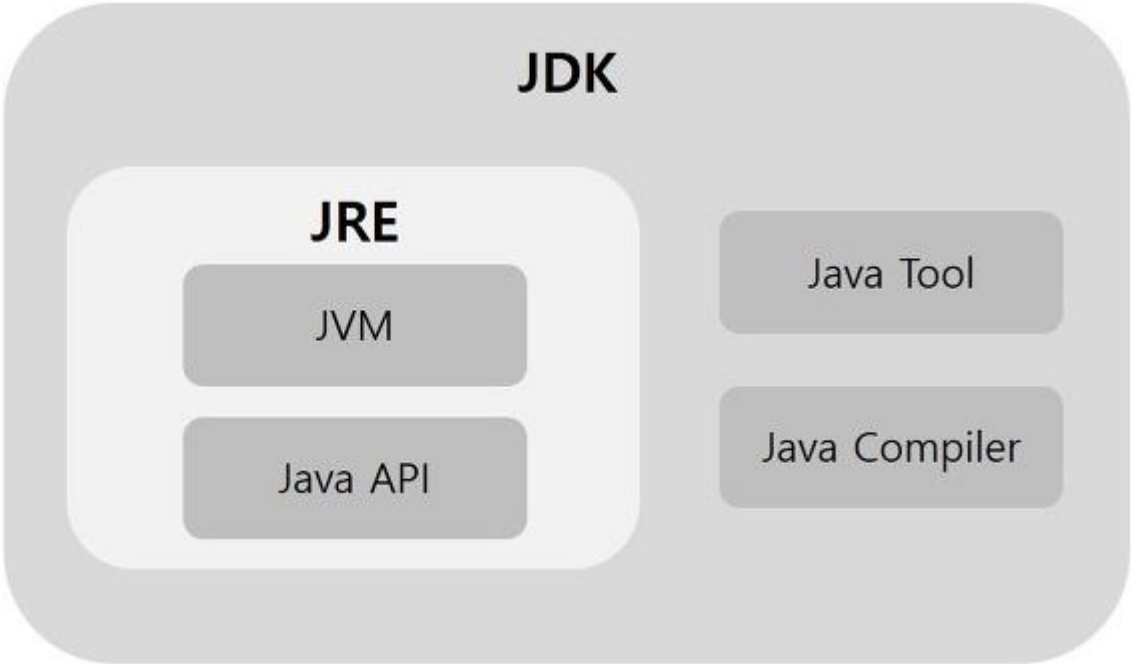


JAVA 개발 환경

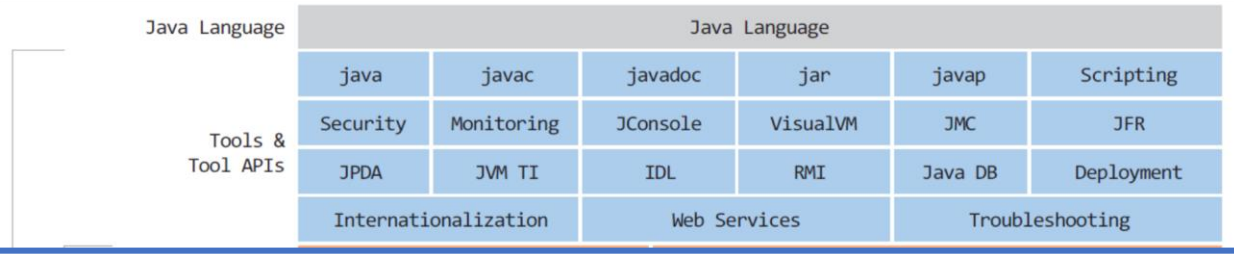
JDK와 JRE

- JRE(Java Runtime Environment)
  - ✓자바 실행 환경. JVM 포함
  - ✓컴파일된 자바 API 들이 들어 있는 모듈 파일
  - ✓개발자가 아닌 경우 JRE만 따로 다운

- JDK(Java Development Kit)
  - ✓자바 응용 개발 환경. 개발에 필요한
    - 컴파일러, 컴파일된 자바 API 클래스들



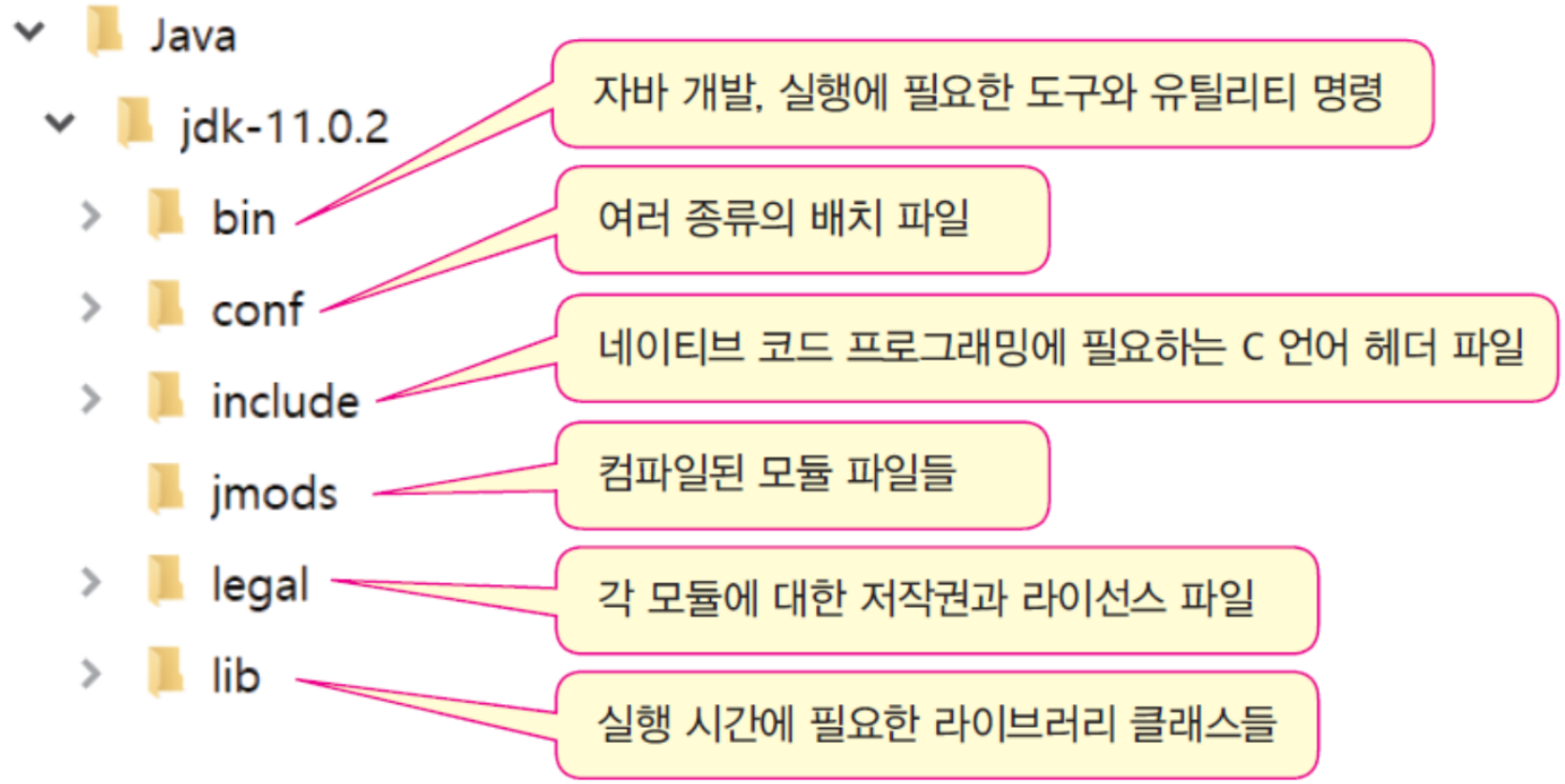
Java SE 구성



JDK의 bin 디렉터리에 포함된 주요 개발 도구

- ✓ javac - 자바 소스를 바이트 코드로 변환하는 컴파일러
- ✓ java - 자바 응용프로그램 실행기. 자바 가상 기계를 작동시켜 자바프로그램 실행
- ✓ javadoc – 자바 소스로부터 HTML 형식의 API 도큐먼트 생성
- ✓ jar – 자바 클래스들(패키지포함)을 압축한 자바 아카이브 파일(.jar) 생성 관리
- ✓ jmod: 자바의 모듈 파일(.jmod)을 만들거나 모듈 파일의 내용 출력
- ✓ jlink: 응용프로그램에 맞춘 맞춤형(custom) JRE 제공
- ✓ jdb - 자바 응용프로그램의 실행 중 오류를 찾는 데 사용하는 디버거
- ✓ javap - 클래스 파일의 바이트 코드를 소스와 함께 보여주는 디어셈블러

JDK 설치 후 디렉터리 구조



자바의 배포판 종류

- 오라클은 개발 환경에 따라 다양한 자바 배포판 제공
- Java SE
  - ✓자바 표준 배포판(Standard Edition)
  - ✓데스크탑과 서버 응용 개발 플랫폼
- Java ME
  - ✓자바 마이크로 배포판
    - 휴대 전화나 PDA, 셋톱박스 등 제한된 리소스를 갖는 하드웨어에서 응용 개발을 위한 플랫폼
    - 가장 작은 메모리 풋프린트
  - ✓Java SE의 서브셋 + 임베디드 및 가전 제품을 위한 API 정의
- Java EE
  - ✓자바 기업용 배포판
    - 자바를 이용한 다중 사용자, 기업용 응용 개발을 위한 플랫폼
  - ✓Java SE + 인터넷 기반의 서버사이드 컴퓨팅 관련 API 추가

JAVA 설치 및 사용

자바 설치

1. JDK 설치
2. 시스템 환경변수 설정
3. Eclipse 설치
4. Hello World 프로그램 작성 및 실행