



# Java 스터디

서버팀 2파트 백승찬

A faded background image of a man with a beard and glasses, wearing a white t-shirt with a mathematical formula. The text "1주차 - Java 소개" is overlaid on the right side of the image.

# 1주차 - Java 소개

## Java의 태동

- SUN Microsystems사에서 1991년 James Gosling 을 중심으로 **가전제품에 사용할 소프트웨어**를 개발하는 중 기존 프로그램들은 플랫폼간 호환성이 없어 소스를 수정하거나 다시 컴파일해야하는 단점에 직면. 임베디드 시스템에 걸맞게 적은 메모리를 사용하면서 다양한 플랫폼을 지원하는 **플랫폼 독립적인 특성 (platform independent)** 을 가진 기술이 필요했음.
- **WORA (Write Once, Run Anywhere)** 를 핵심가치로 둔 oak 언어 개발. 웹 역시 위 특성을 필요로 했기때문에 SUN Microsystems는 oak를 인터넷 환경에 맞게 발전시켜 **1995년 Java 언어를 발표**
- 가장 많이 이용되는 Netscape 브라우저에 Java가 탑재되며 급격히 발전

## Java의 태동

- SUN Microsystems사에서 1991년 James Gosling 을 중심으로 **가전제품에 사용할 소프트웨어**를 개발하는 중 기존 프로그램들은 플랫폼간 호환성이 없어 소스를 수정하거나 다시 컴파일해야하는 단점에 직면. 임베디드 시스템에 걸맞게 적은 메모리를 사용하면서 다양한 플랫폼을 지원하는 **플랫폼 독립적인 특성 (platform independent)** 을 가진 기술이 필요했음.
- **WORA (Write Once, Run Anywhere)** 를 핵심가치로 둔 oak 언어 개발. 웹 역시 위 특성을 필요로 했기때문에 SUN Microsystems는 oak를 인터넷 환경에 맞게 발전시켜 **1995년 Java 언어를 발표**
- 가장 많이 이용되는 Netscape 브라우저에 Java가 탑재되며 급격히 발전

**// Java의 핵심가치는 WORA 이며, 플랫폼 독립적인 언어이다 //**



Write Once, Run Anywhere!

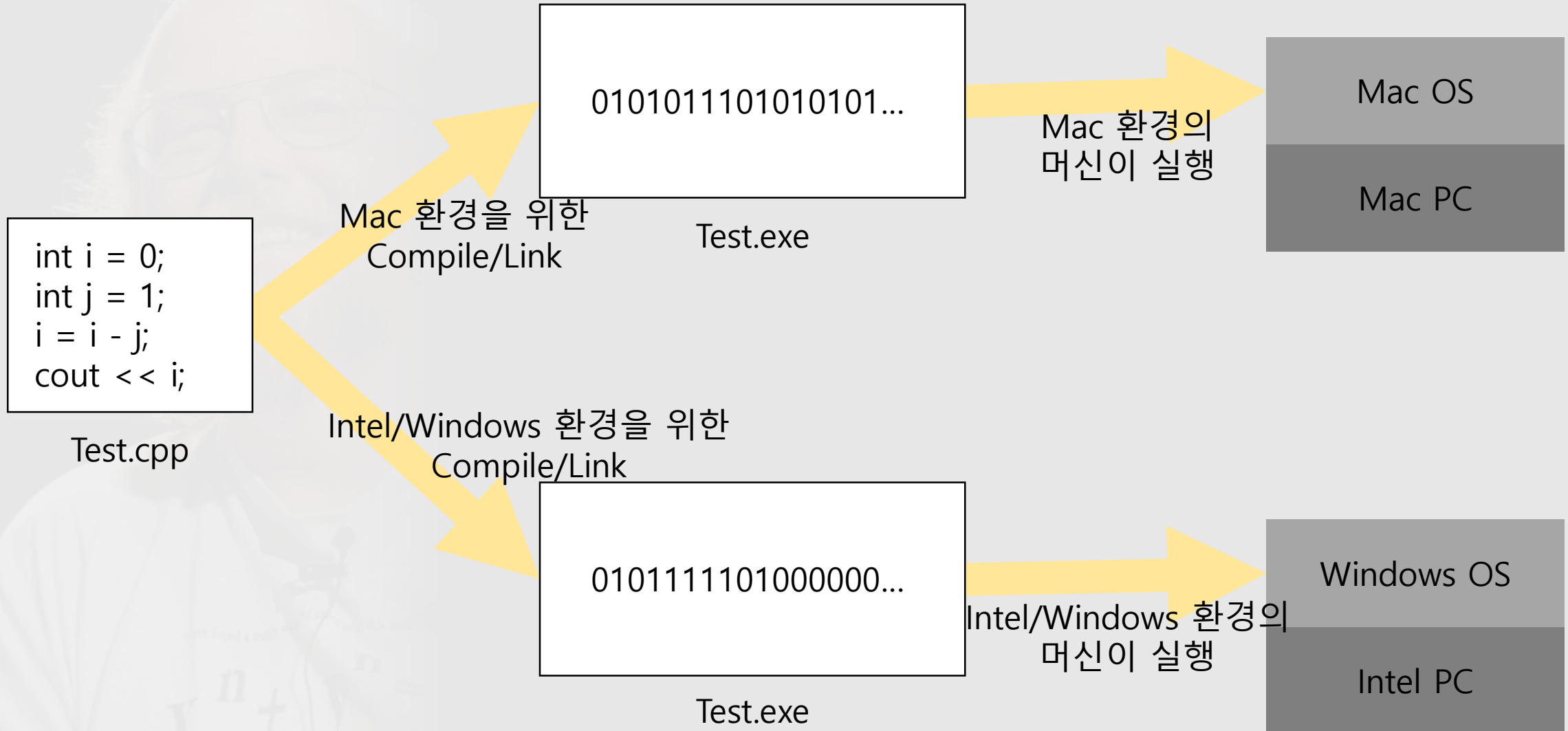
어떤 장비에서도 개발될 수 있고, 표준 바이트코드로 컴파일되고, 자바 가상 머신이 장착된 어떤 장비에서도 실행될 수 있어야 한다.

James Gosling (1955.5.19~)

# Java의 WORA

- WORA 를 가능케 한 이유
  - C/C++ 언어는 컴파일 결과 타겟 머신의 CPU가 실행할 수 있는 기계어로 직접 변환하지만, Java 언어는 컴파일 결과로 **JVM(Java Virtual Machine)이 실행할 수 있는 바이트코드로 변환**해 타겟 머신에 JVM만 있으면 어디서나 동일한 실행 환경을 제공한다.
- Byte Code
  - Javac (자바 컴파일러) 가 Java 소스코드를 컴파일한 결과로 추출된 기계어
  - JVM이 실행할 수 있는 바이너리 코드 (타겟 머신의 CPU가 실행하지 않음)
  - JVM이 인터프리터 방식으로 Byte Code를 해석해 타겟 머신에서 실행
- JVM (Java Virtual Machine)
  - 서로 다른 타겟 머신별로 동일한 자바 실행 환경 제공.
  - JVM 자체는 플랫폼 종속적.
  - Byte Code 를 입력받아 프로세스를 실행.

# Java의 WORA



# Java의 WORA

```
int i = 0;  
int j = 1;  
i = i - j;  
cout << i;
```

Compile

0110110001010101...

실행

Mac OS

Mac PC

실행

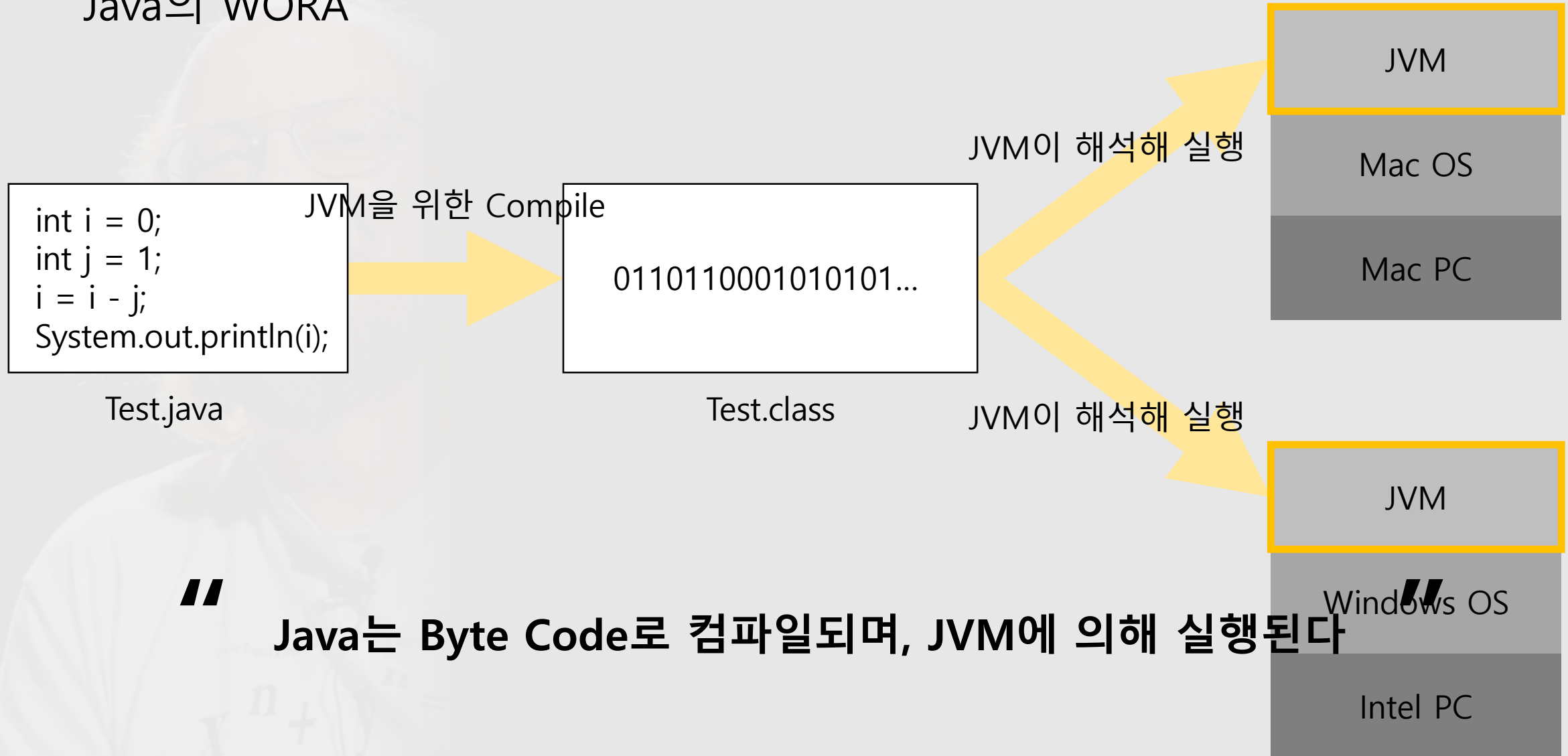
Windows OS

Intel PC

$$x^n + y^n = z^n$$



## Java의 WORA



# Java의 WORA

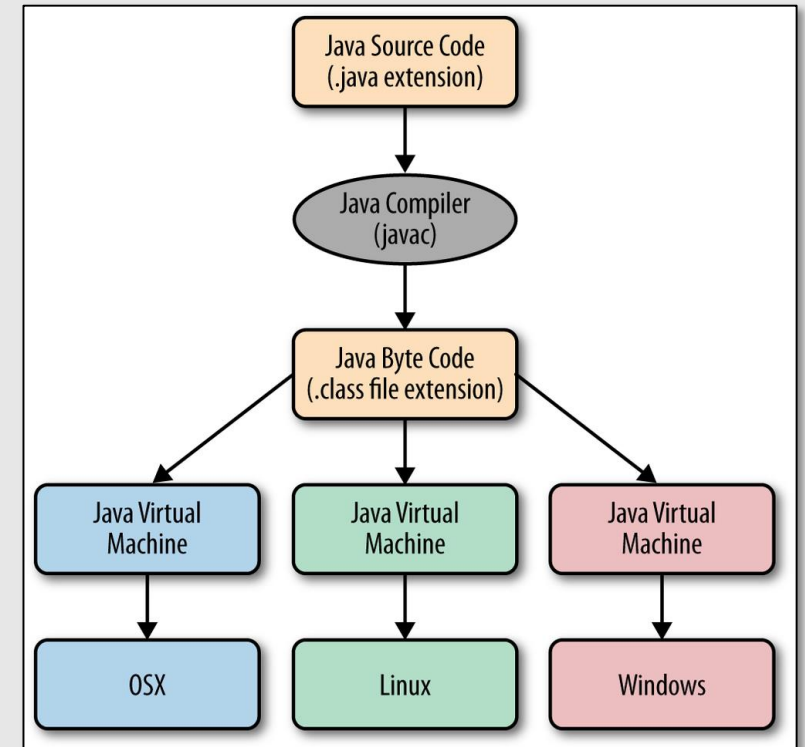
- 실행환경의 차이

- [C/C++]

- 컴파일/링크 결과 타겟 머신에서 실행가능한 실행 파일을 생성

- [Java]

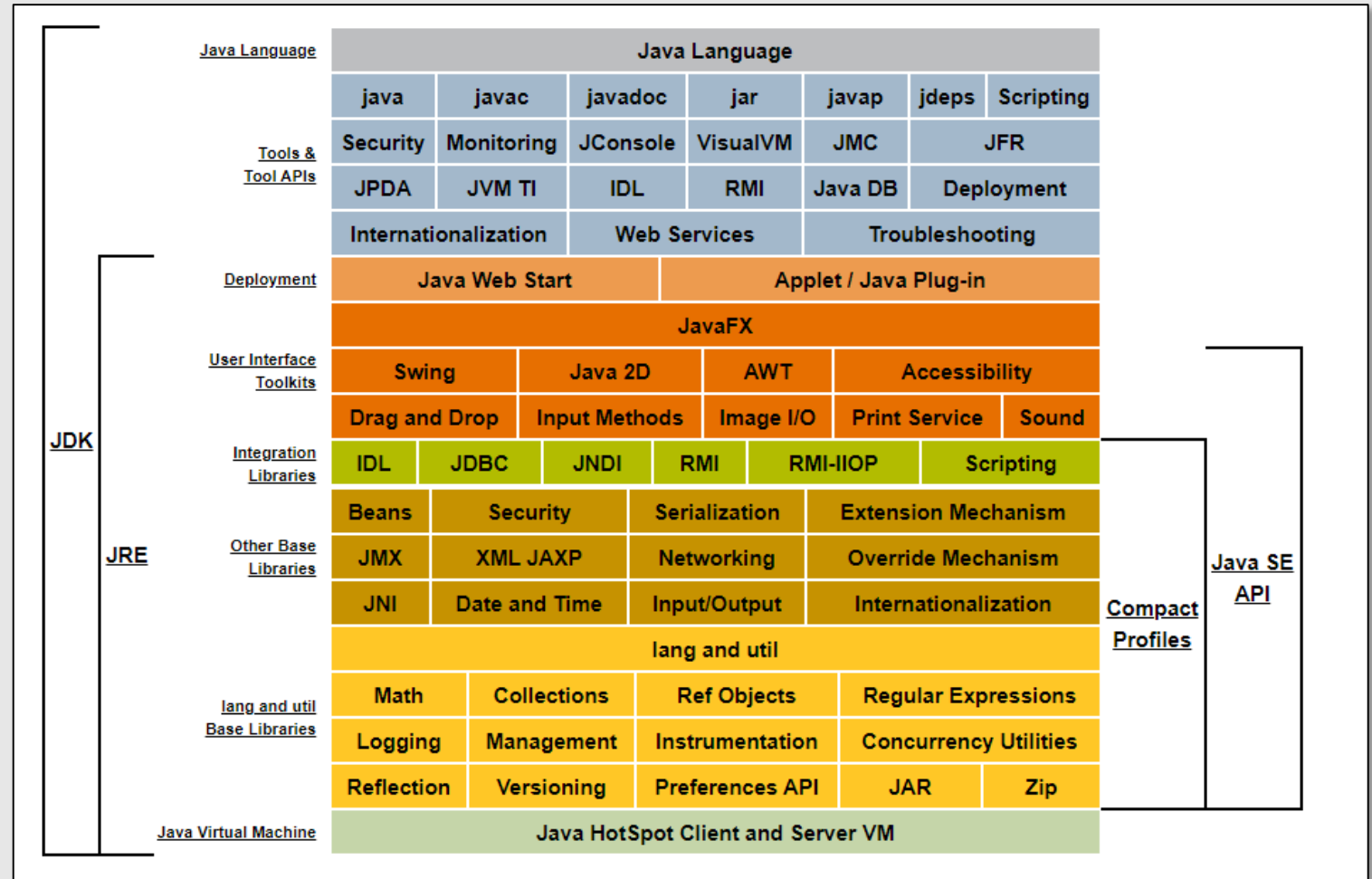
- 컴파일 결과 JVM에서 실행가능한 Byte Code 형태의 클래스 파일을 생성



<http://www.developingthefuture.net/compilation-process-and-jit-compiler/>

# Java 개발 환경 구축

- JDK (Java Development Kit)  
: 개발 및 실행 도구
- JRE (Java Runtime Environment)  
: 실행 도구  
(대부분 컴퓨터에 설치되어있음)



# Java 개발 환경 구축

- JDK 설치

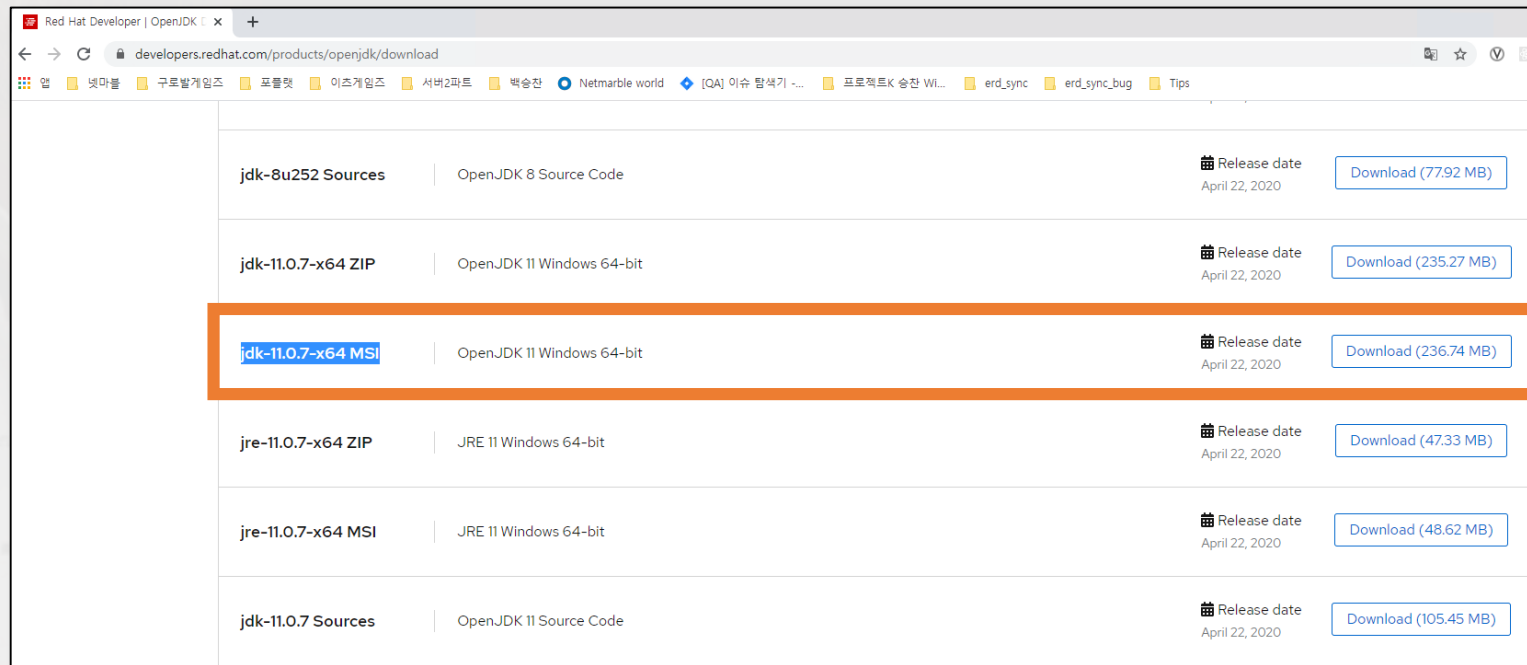
- Red hat Openjdk11 설치

- 구글 드라이브로 다운받기

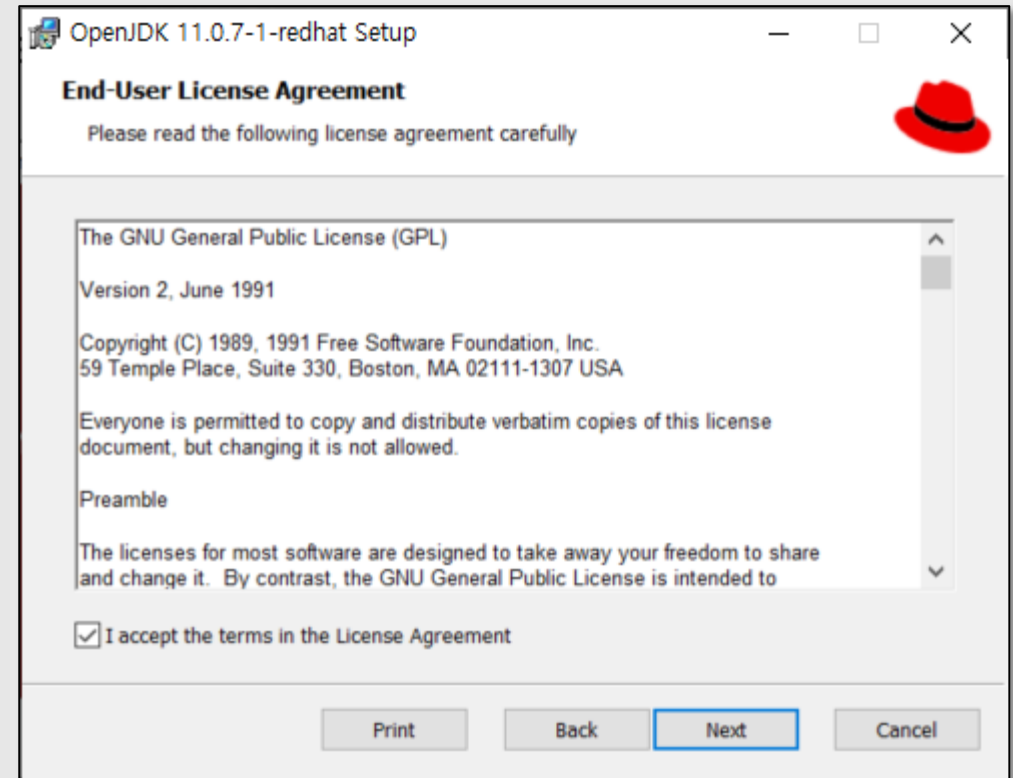
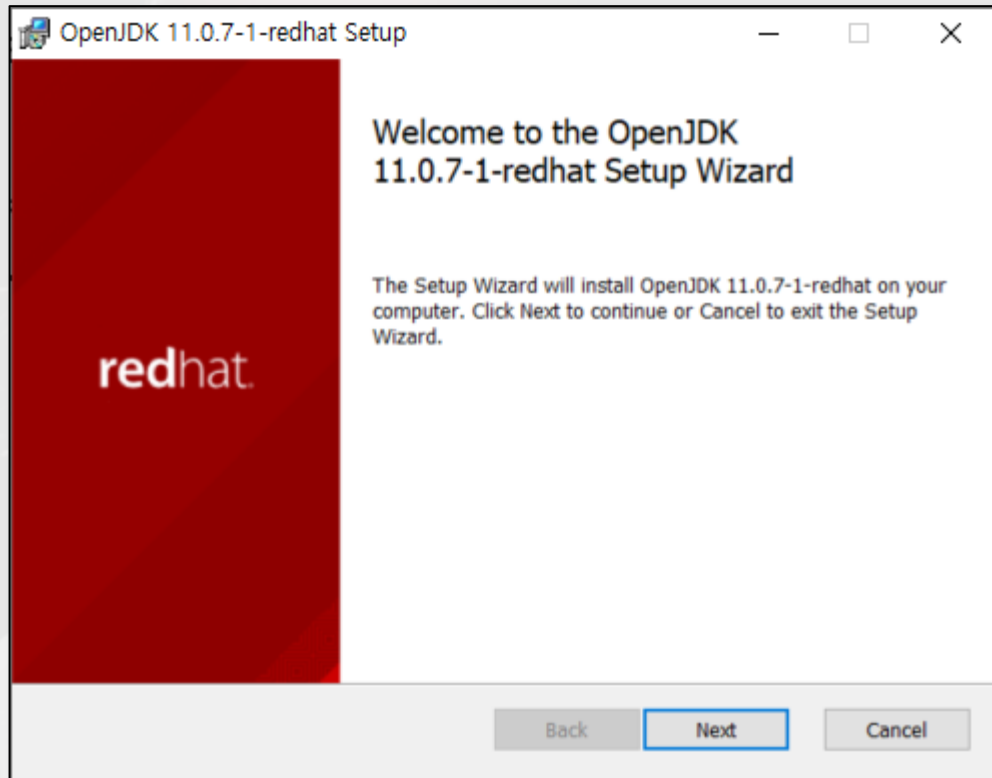
- <https://drive.google.com/open?id=1LMmIWYGKIG3GuYvxXFfA6ukRcNeEJNly>

- Red hat 공식 홈페이지에서 다운 받기

- <https://developers.redhat.com/products/openjdk/download>

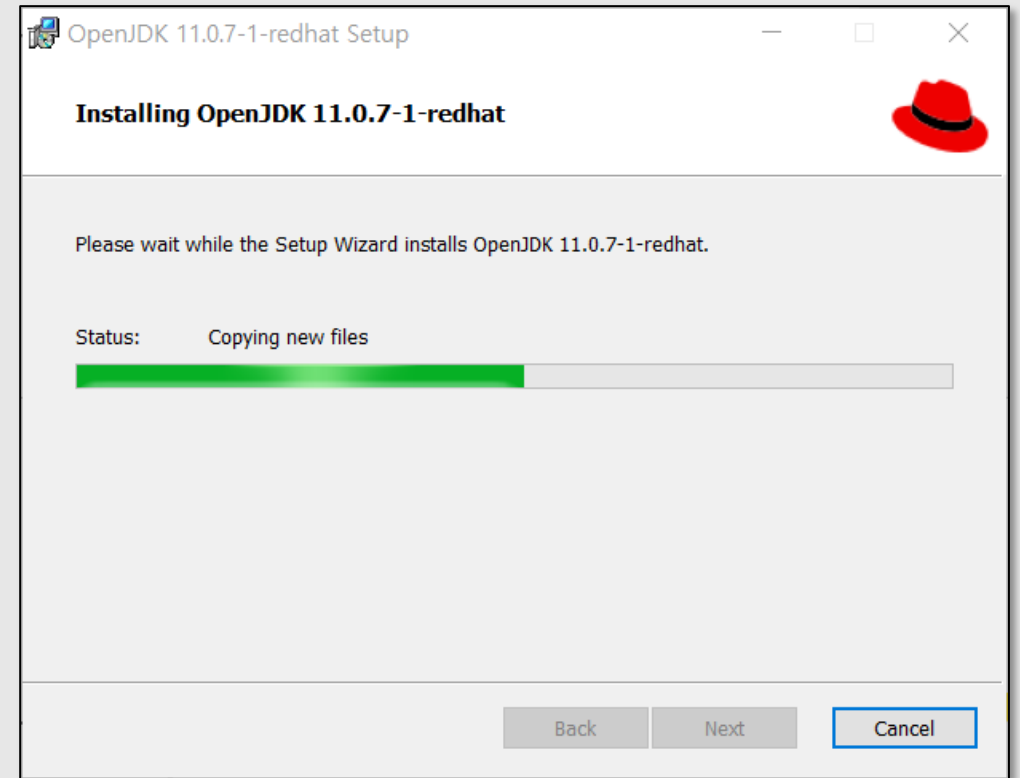
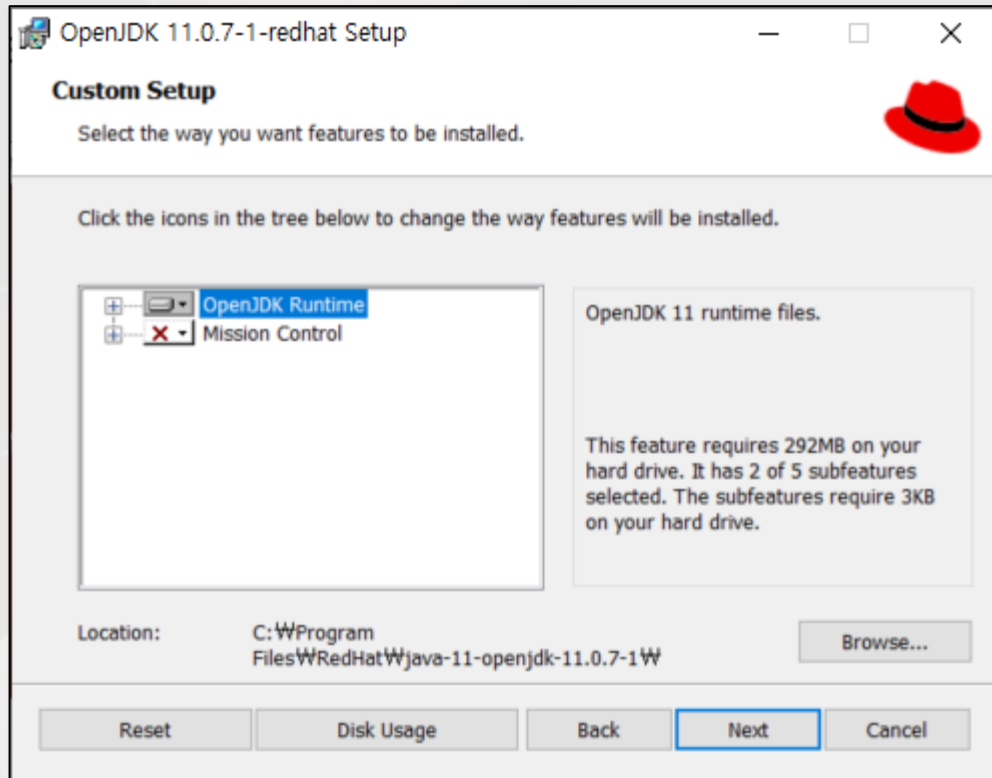


## Java 개발 환경 구축 (JDK 설치 & 세팅)



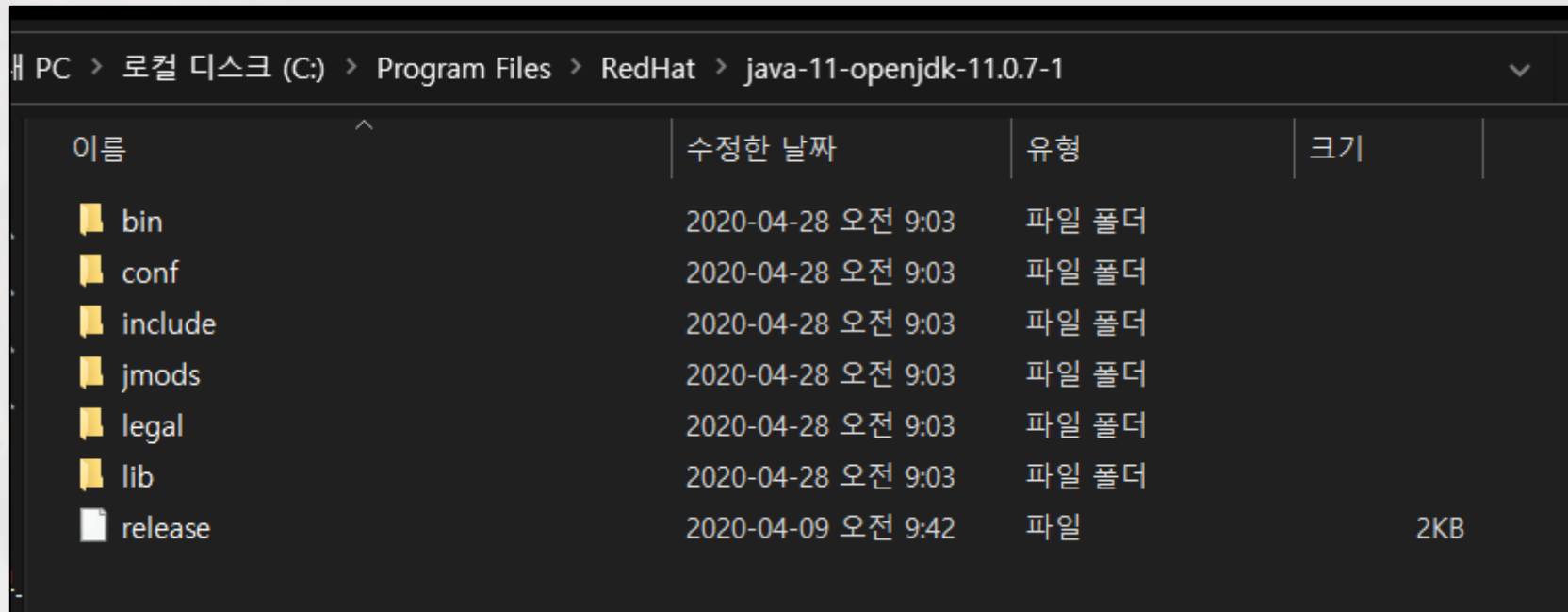
다운 받은 msi 설치파일을 열고 라이선스 동의 및 째욱 Next~

## Java 개발 환경 구축 (JDK 설치 & 세팅)



설정 다시 할 수 있으니 쪽쪽 설치

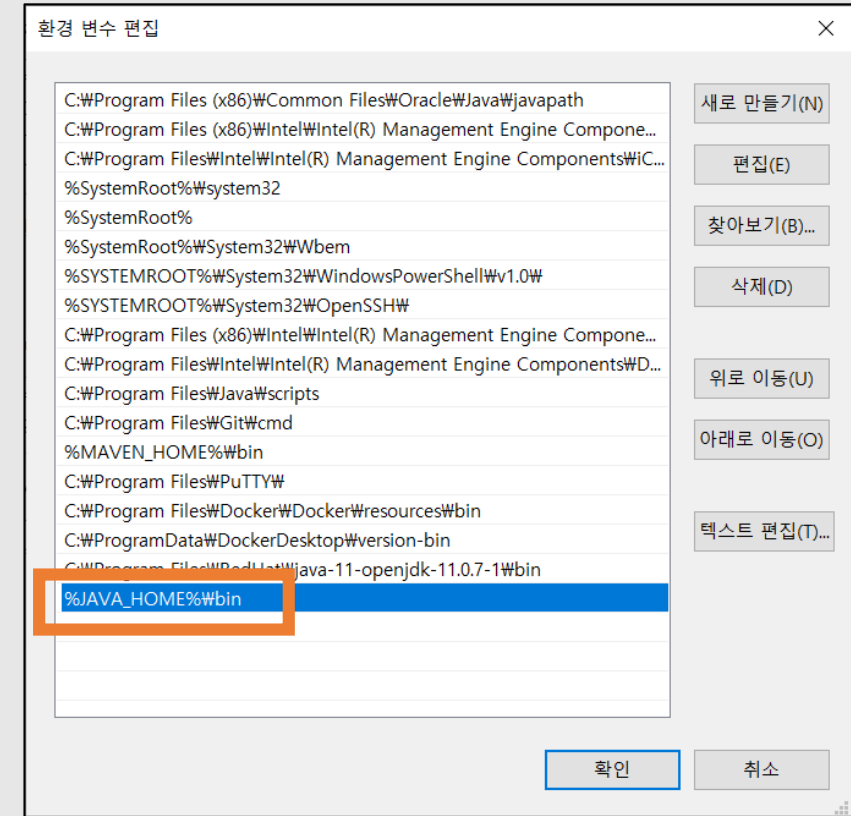
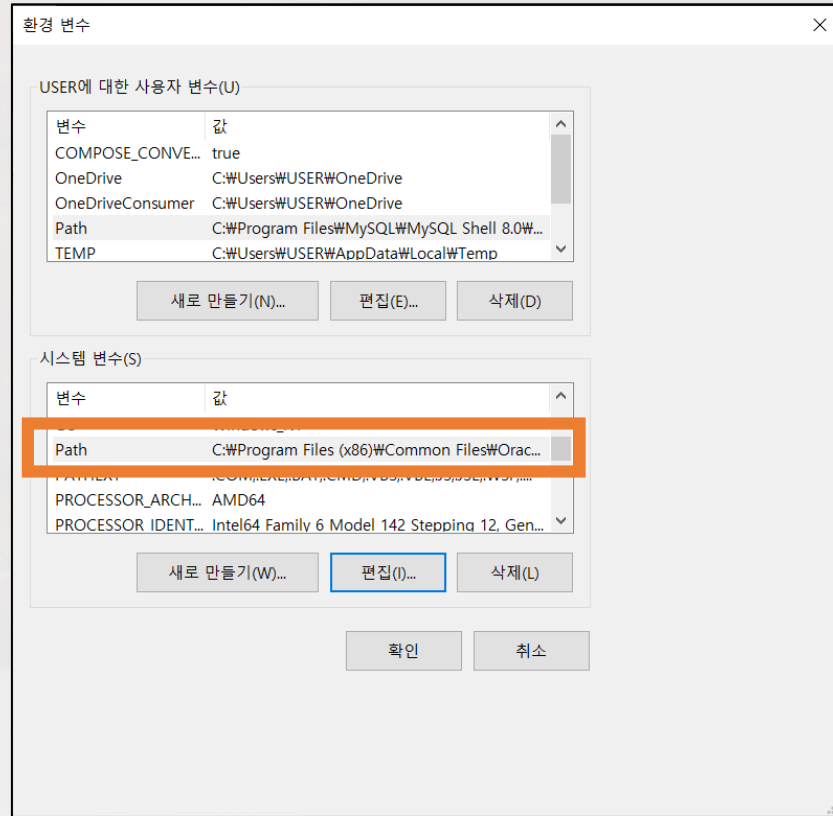
## Java 개발 환경 구축 (JDK 설치 & 세팅)



| PC > 로컬 디스크 (C:) > Program Files > RedHat > java-11-openjdk-11.0.7-1 |                    |       |     |  |
|--|--------------------|-------|-----|--|
| 이름   | 수정한 날짜             | 유형    | 크기  |  |
| bin  | 2020-04-28 오전 9:03 | 파일 폴더 |     |  |
| conf   | 2020-04-28 오전 9:03 | 파일 폴더 |     |  |
| include  | 2020-04-28 오전 9:03 | 파일 폴더 |     |  |
| jmods  | 2020-04-28 오전 9:03 | 파일 폴더 |     |  |
| legal  | 2020-04-28 오전 9:03 | 파일 폴더 |     |  |
| lib  | 2020-04-28 오전 9:03 | 파일 폴더 |     |  |
| release  | 2020-04-09 오전 9:42 | 파일    | 2KB |  |

RedHat 디렉터리 하위에 파일 생성됨

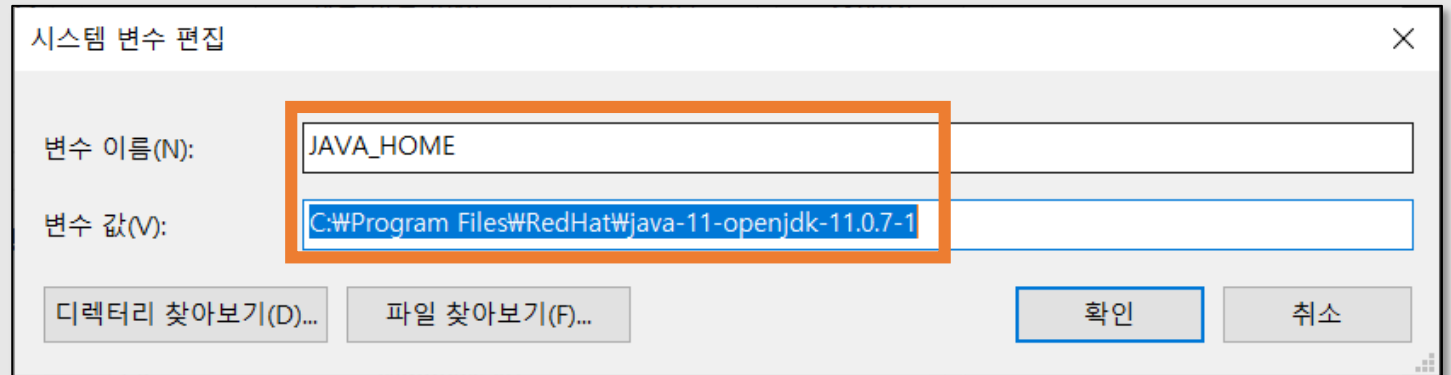
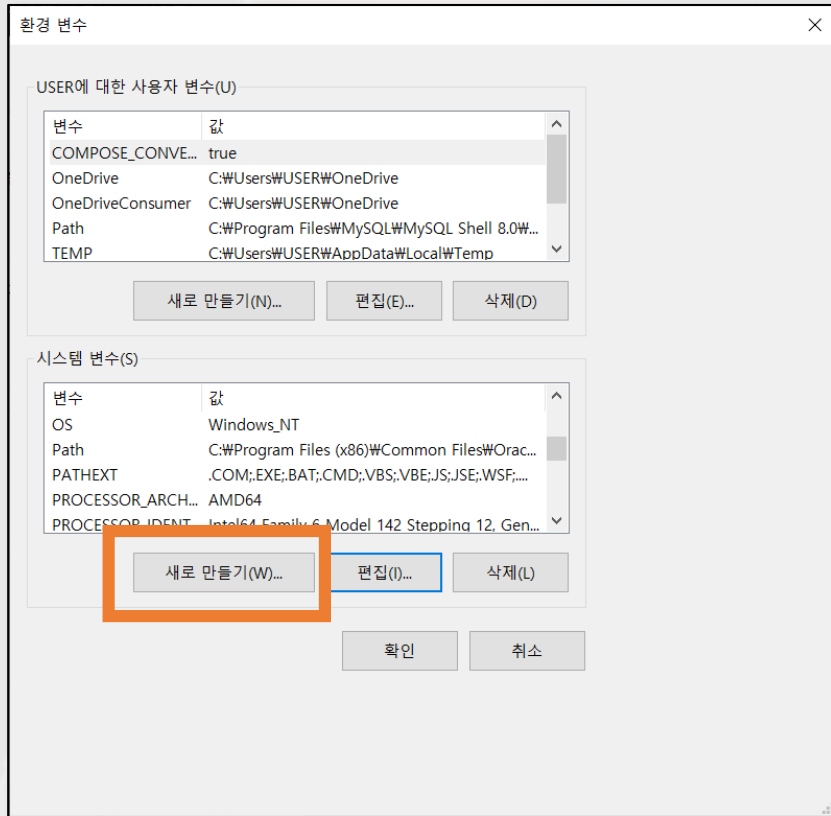
# Java 개발 환경 구축 (JDK 설치 & 세팅)



시스템변수 'Path' 에 'JAVA\_HOME' 이라는 이름의 항목을 추가해줍니다.  
복붙 → %JAVA\_HOME%\bin



# Java 개발 환경 구축 (JDK 설치 & 세팅)



시스템 변수 'JAVA\_HOME' 이라는 이름의 환경변수를 추가하고  
jdk11 디렉터리를 지정해줍니다.

## Java 개발 환경 구축 (JDK 설치 & 세팅)

```
C:\Users\USER>java -version
openjdk version "11.0.7" 2020-04-14 LTS
OpenJDK Runtime Environment 18.9 (build 11.0.7+10-LTS)
OpenJDK 64-Bit Server VM 18.9 (build 11.0.7+10-LTS, mixed mode)

C:\Users\USER>
```

java -version  
정상 설치 완료

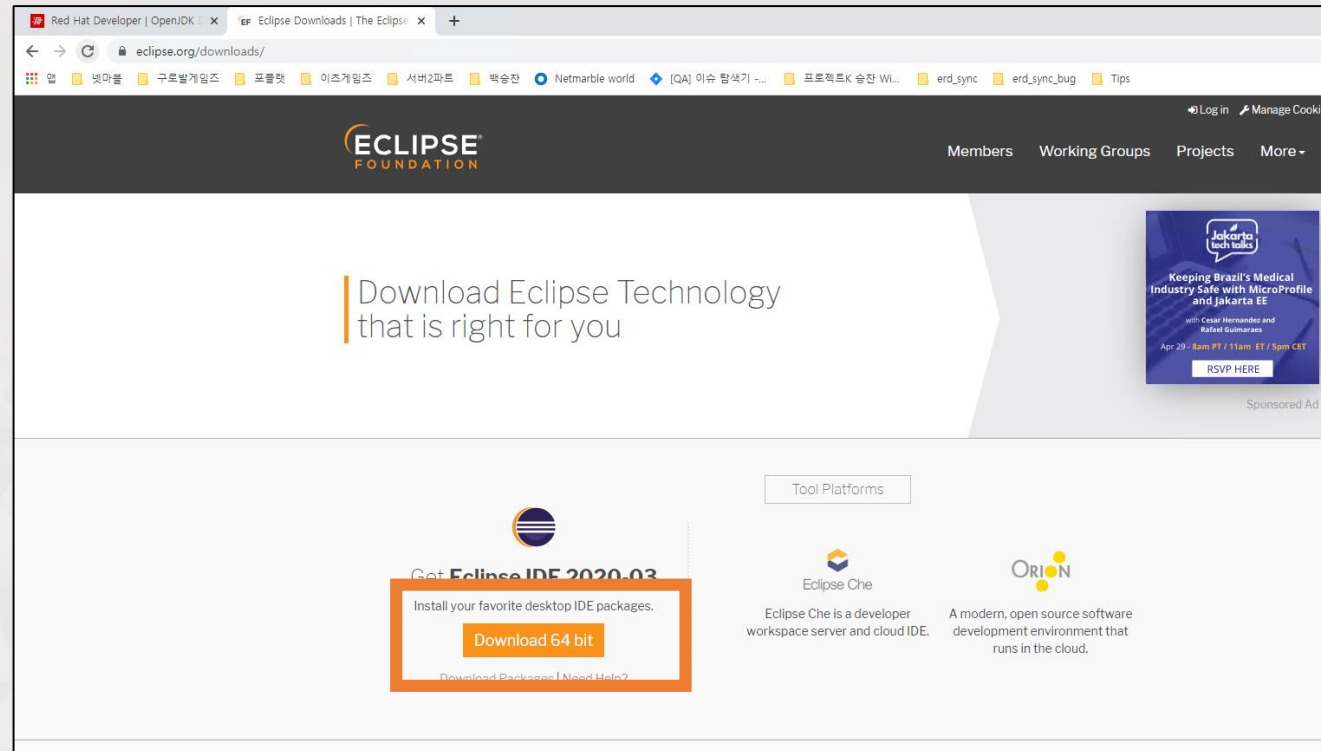
# Java 개발 환경 구축

- IDE 설치

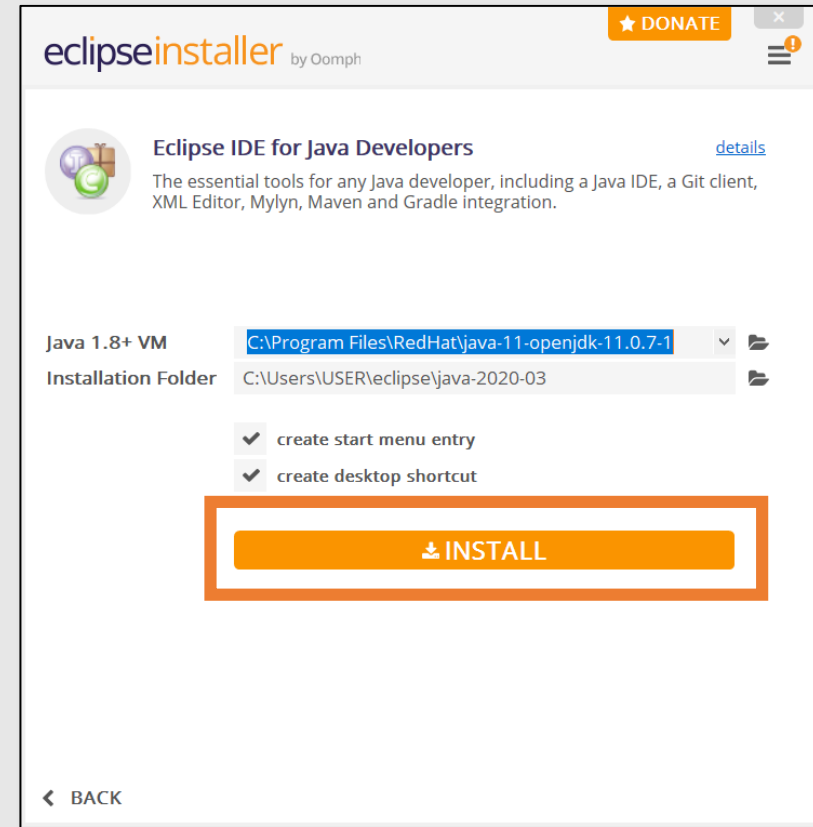
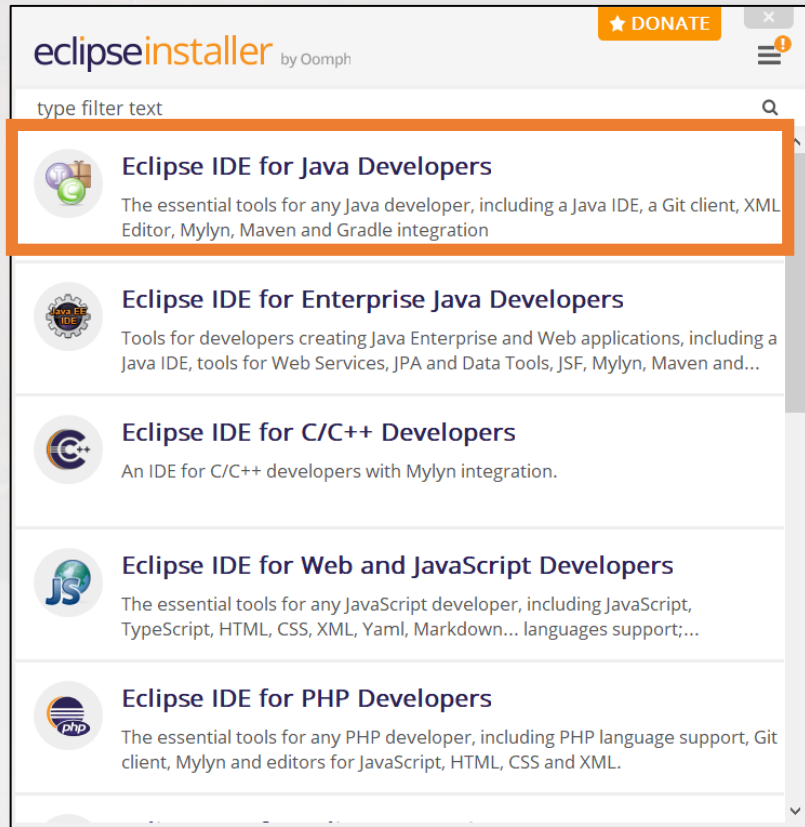
- Eclipse 설치

- 공식 홈페이지에서 다운 받기

<https://www.eclipse.org/downloads/>

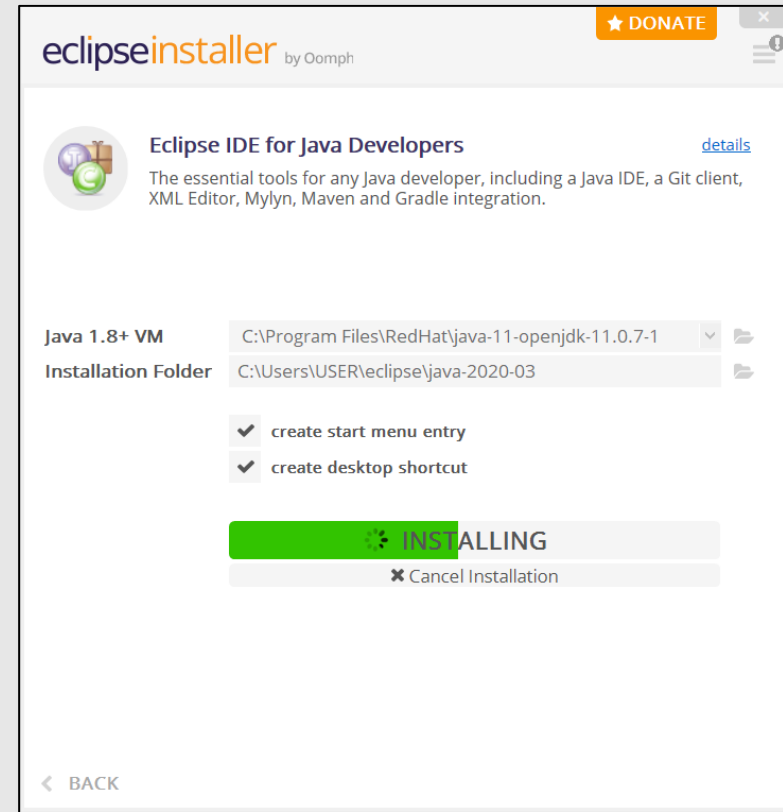
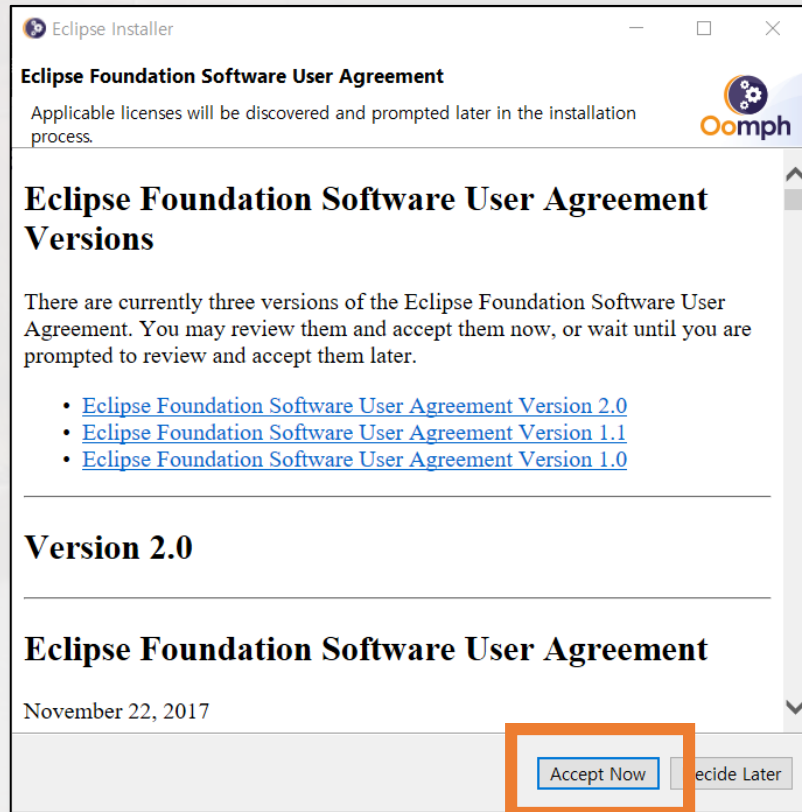


# Java 개발 환경 구축 (Eclipse IDE 설치)



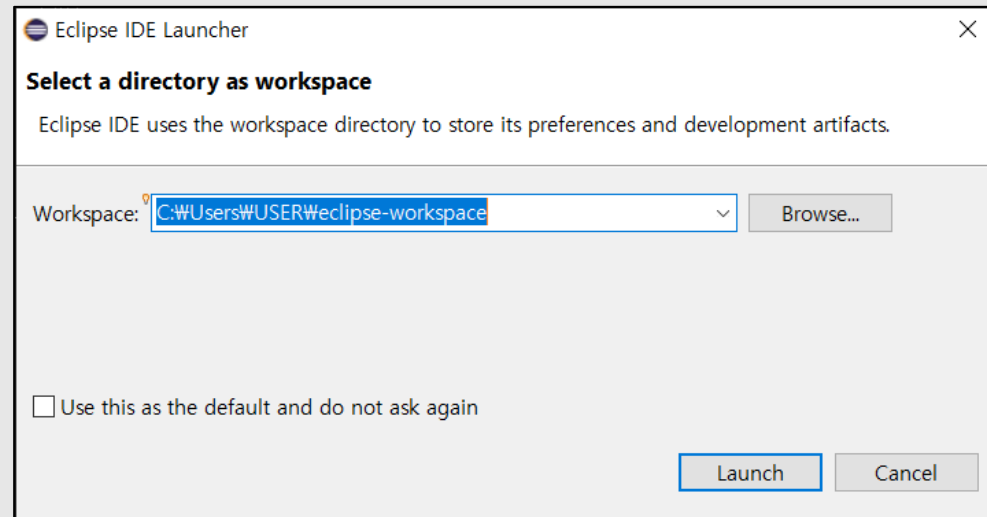
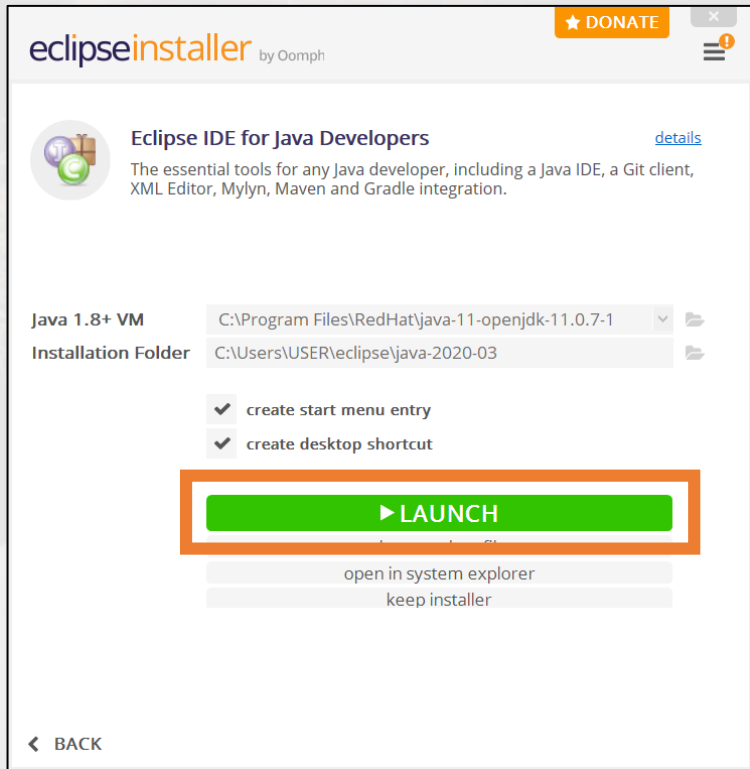
설치 파일을 실행해 Eclipse IDE for Java Developers 설치

# Java 개발 환경 구축 (Eclipse IDE 설치)



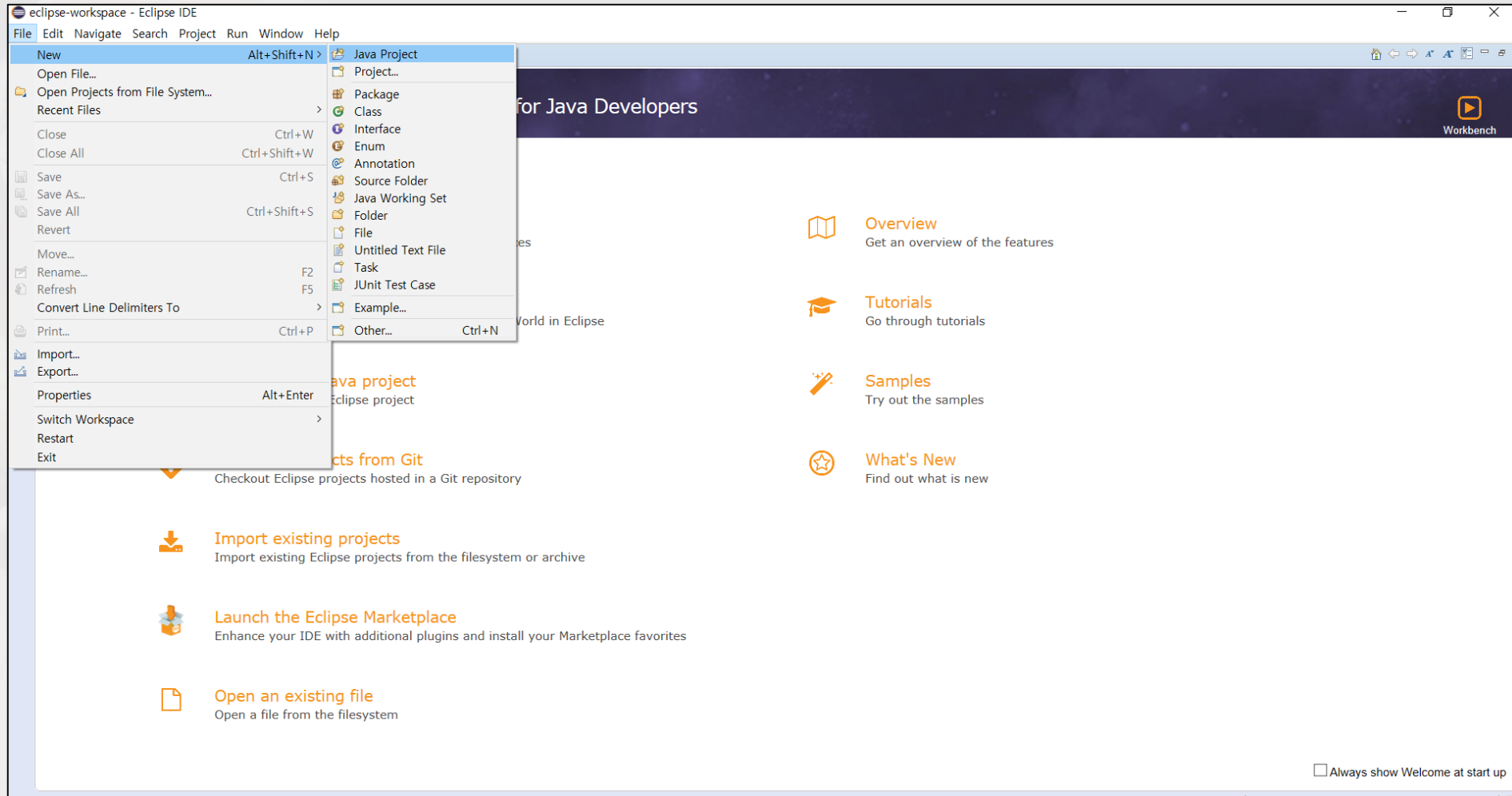
라이선스 동의 후 설치 진행

# Java 개발 환경 구축 (Eclipse IDE 설치)



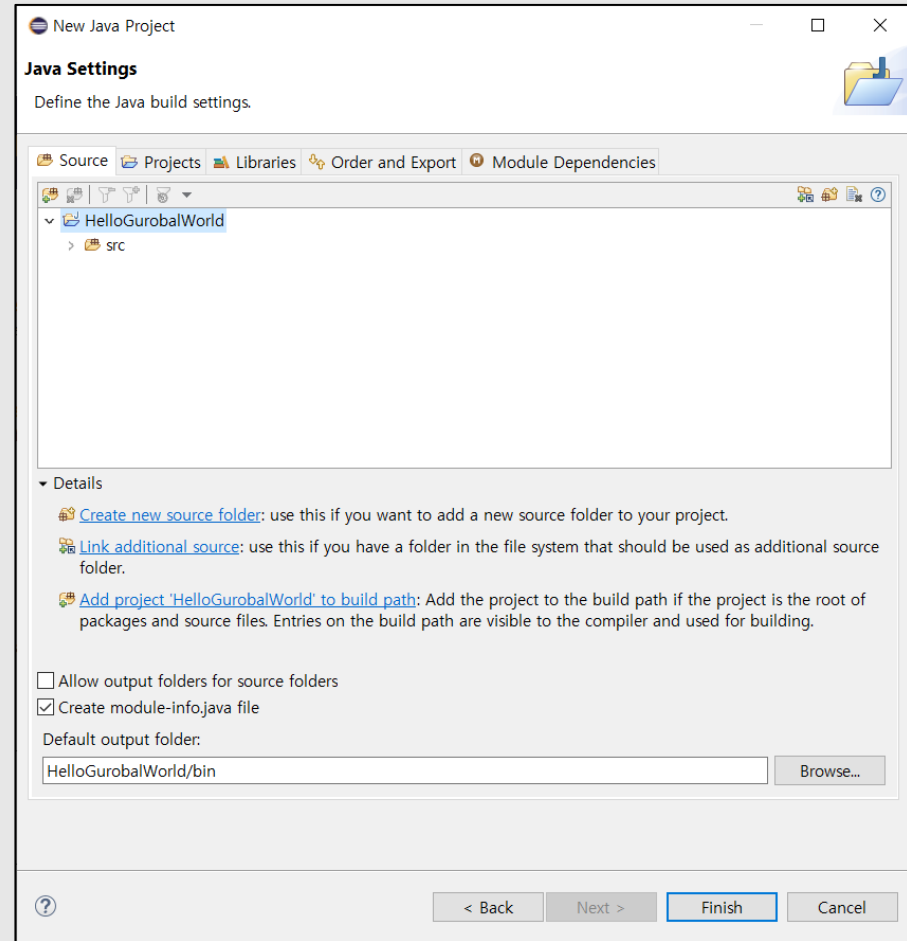
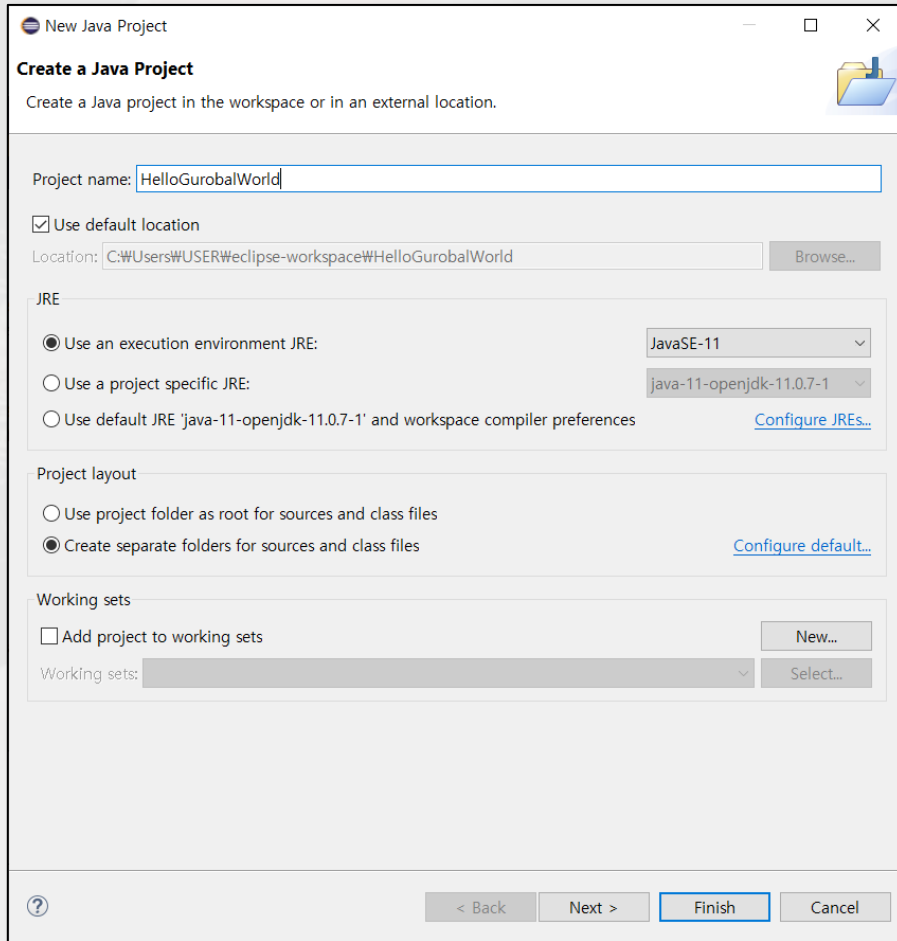
설치 완료후 workspace 경로 지정

# Java 개발 환경 구축 (Eclipse IDE 설치)



File – New – Java Project 선택해 hello world 작성해보기

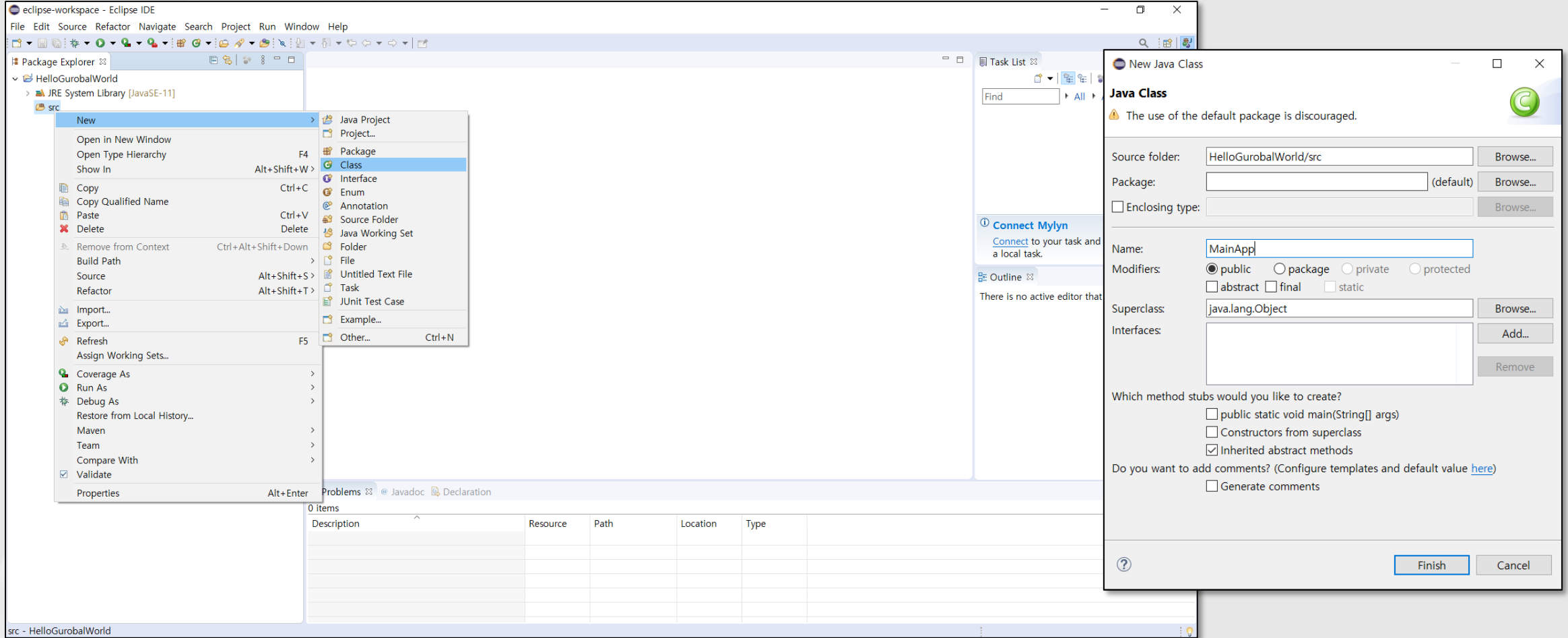
# Java 개발 환경 구축 (Eclipse IDE 설치)



HelloGurobalWorld 프로젝트 생성

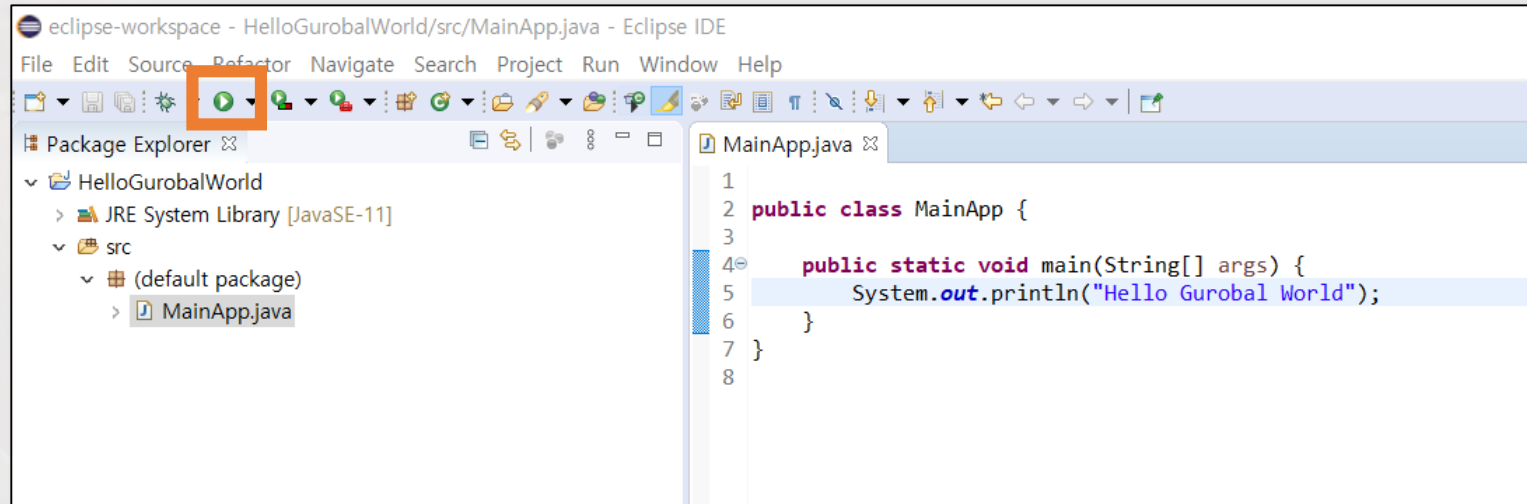


# Java 개발 환경 구축 (Eclipse IDE 설치)



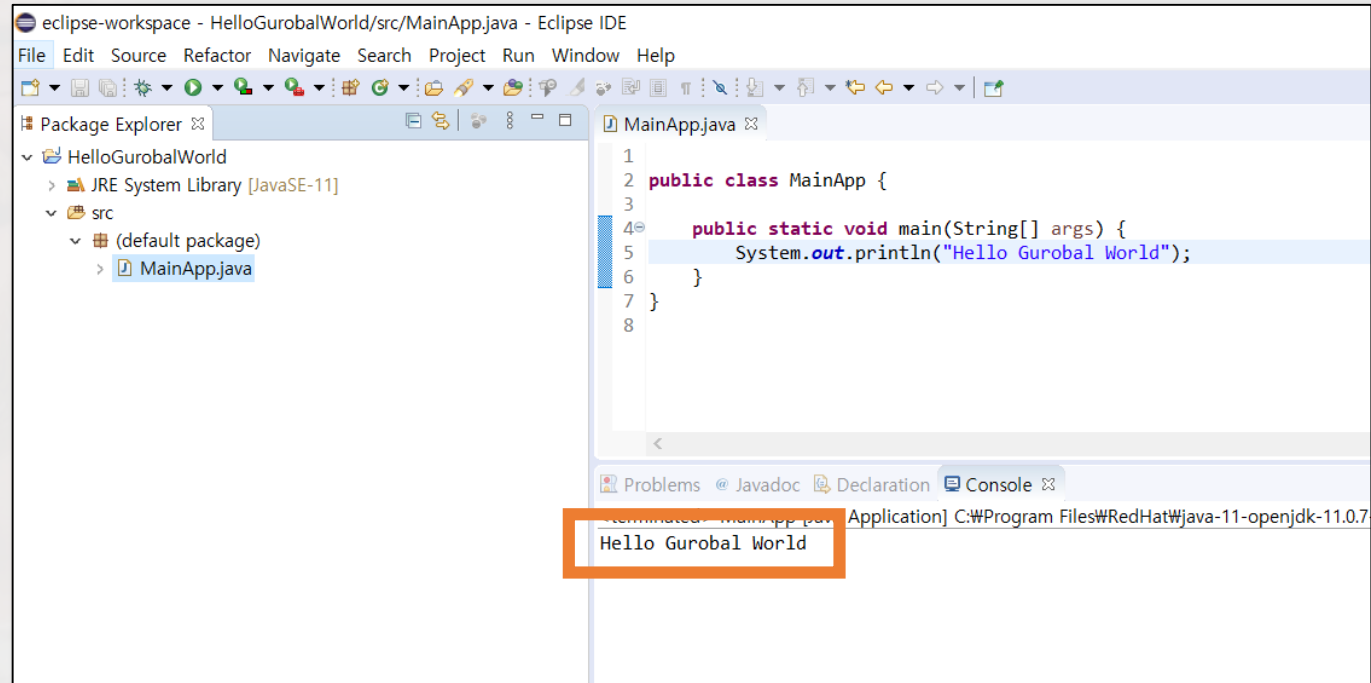
HelloGurobalWorld/src 디렉터리 우클릭 – New – Class  
클래스 생성

## Java 개발 환경 구축 (Eclipse IDE 설치)



main() 메서드에 Hello Gurobal World 출력하도록 작성  
및 실행 버튼 클릭

# Java 개발 환경 구축 (Eclipse IDE 설치)



콘솔 출력 테스트 완료

# Java의 역사

- **JDK 1.0** (1996.1)
- **JDK 1.1** (1997.2)
  - inner class, JavaBeans, RMI, 리플렉션, Unicode 지원, 국제화
- **J2SE 1.2** (1998.12)
  - strictfp, Swing GUI, JIT, Java Applet 웹프라우저 플러그인, COBRA, Collections  
엔터프라이즈 서버 J2EE, 모바일 응용 프로그램 J2ME, 데스크탑 J2SE
- **J2SE 1.3** (2000.3)
  - HotSpot JVM, JNDI, JPDA, JavaSound
- **J2SE 1.4** (2002.2)
  - assert, 정규표현식, IPv6, Non-Blocking IO, XML api, JCE, JSSE, JAAS, Java Web Start
- **J2SE 5** (2004.9)
  - Generics, Annotation, Auto Boxing/Unboxing, Enumeration, 가변길이 파라미터, Static Import, Concurrency Api, java.util.Scanner

# Java의 역사

- JVM 핵심코드를 오픈소스화 (2006.11)
- **Java SE 6** (2006.12)
  - Scripting Language Support (Rhino JavaScript 엔진), JDBC 4.0, Java Compiler Api, Pluggable Annotation
- Oracle이 SUN Microsystems 인수 (2009~2010)
- **Java SE 7** (2011.7)
  - Dynamic Language 지원, switch문에 문자열 사용, try 문 자원관리, Diamond Operator <>, FILE NIO
- **Java SE 8** (2014.3)
  - **LTS 버전**, Lambda Expression, Nashor JavaScript 엔진, Annotation on Java Types, Unsigned Integer 계산, Repeating Annotation, JodaTime, Static Link JNI Library, Interface Default Method, String Api

# Java의 역사

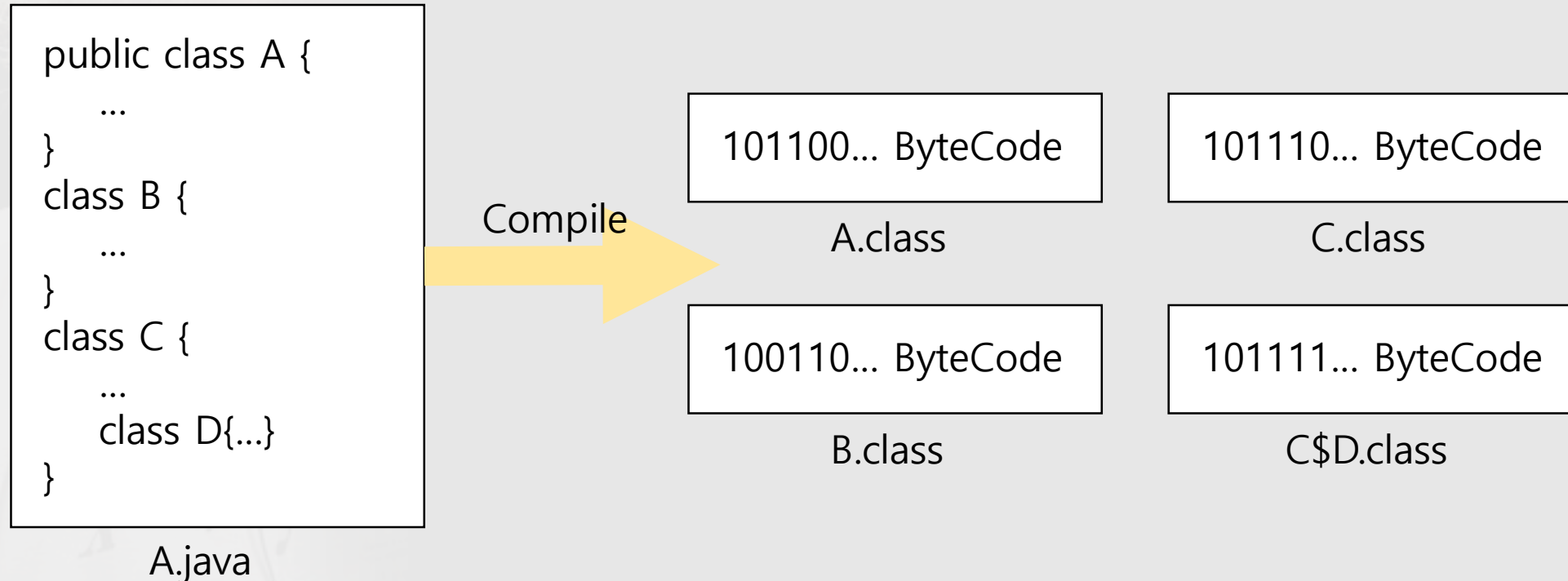
- **Java SE 9** (2017.9)
  - Jigsaw 기반 Runtime 모듈화 (하위 호환성에 문제 발생), Jshell, HTTP/2, private Interface Method, Java Applet 지원 종료
- **Java SE 10** (2018.3)
  - var 예약어를 이용한 지역변수 타입 추론, 병렬 처리 gc, 개별 스레드로 분리된 stop-the-world, JVM 힙영역을 시스템 메모리 외의 공간에 할당 가능, deprecated 처리된 api 삭제
- **Java SE 11** (2018.9)
  - LTS 버전, Java EE 삭제, JavaFX 별도 모듈로 분리, HTTP 클라이언트 표준화, OracleJDK 구독형 유료 모델 전환. OracleJDK 독점 기능이 OpenJDK로 이식 (완전 동일)
- **Java SE 12** (2019.3)
  - switch 문 확장, gc 개선, 마이크로 벤치마크 툴
- **Java SE 13** (2019.9)
  - switch 문 개선 (yield 예약어)
- **Java SE 14** (2020.3)

## Java의 특징

- 객체 지향 언어
  - 클래스 계층구조, 상속성 (다중상속 지원안함), 다형성, 캡슐화 지원
  - 지역 변수나 메서드를 반드시 클래스내에 구현한다.  
(클래스에 속하지 않은 변수나 메서드는 있을 수 없음)
- 멀티스레드
  - OS의 도움 없이 java 프로그램에서 멀티스레드를 동시에 실행할 수 있음
- 플랫폼 독립적
  - 머신에 종속되지 않는 독립적인 Byte Code로 컴파일해 머신의 JVM이 실행

# Java의 특징

- 소스 파일(.java), 클래스 파일(.class)
  - 하나의 소스 파일(.java) 에는 여러 개의 클래스를 작성할 수 있지만, 하나의 클래스 파일(.class) 에는 반드시 하나의 컴파일된 자바 클래스만이 포함된다. (내부 클래스 \$으로 구분)
  - 하나의 소스 파일(.java) 에는 오직 한 클래스만 public 으로 선언할 수 있다.
  - 소스파일 (.java) 이름과 작성된 public 클래스 이름은 동일해야한다.





# Java의 특징

- 실행 모듈
  - java 프로그램은 1개 또는 N개의 클래스파일(.class) 로 구성된다.
  - N개의 클래스파일(.class) 을 jar 형태로 압축해 배포 및 실행이 가능하다.
  - java 프로그램의 실행은 main() 메서드에서 시작된다.  
(하나의 클래스 파일에 여러 개의 main() 메서드가 있을 수 없다. 다만, 다수의 클래스 파일이 각각 main() 메서드를 가질 수 있다)
- 패키지 시스템
  - 서로 관련 있는 클래스를 패키지로 묶어 관리한다.

# Java 데이터 타입

- Primitive Type

- boolean
- char
- byte
- short
- int
- long
- float
- double

- Reference Type

- 배열
- 클래스 (String 포함)
- 인터페이스

IDE 없이 간단히 테스트

[https://www.w3schools.com/java/java\\_data\\_types.asp](https://www.w3schools.com/java/java_data_types.asp)

# Java 예시

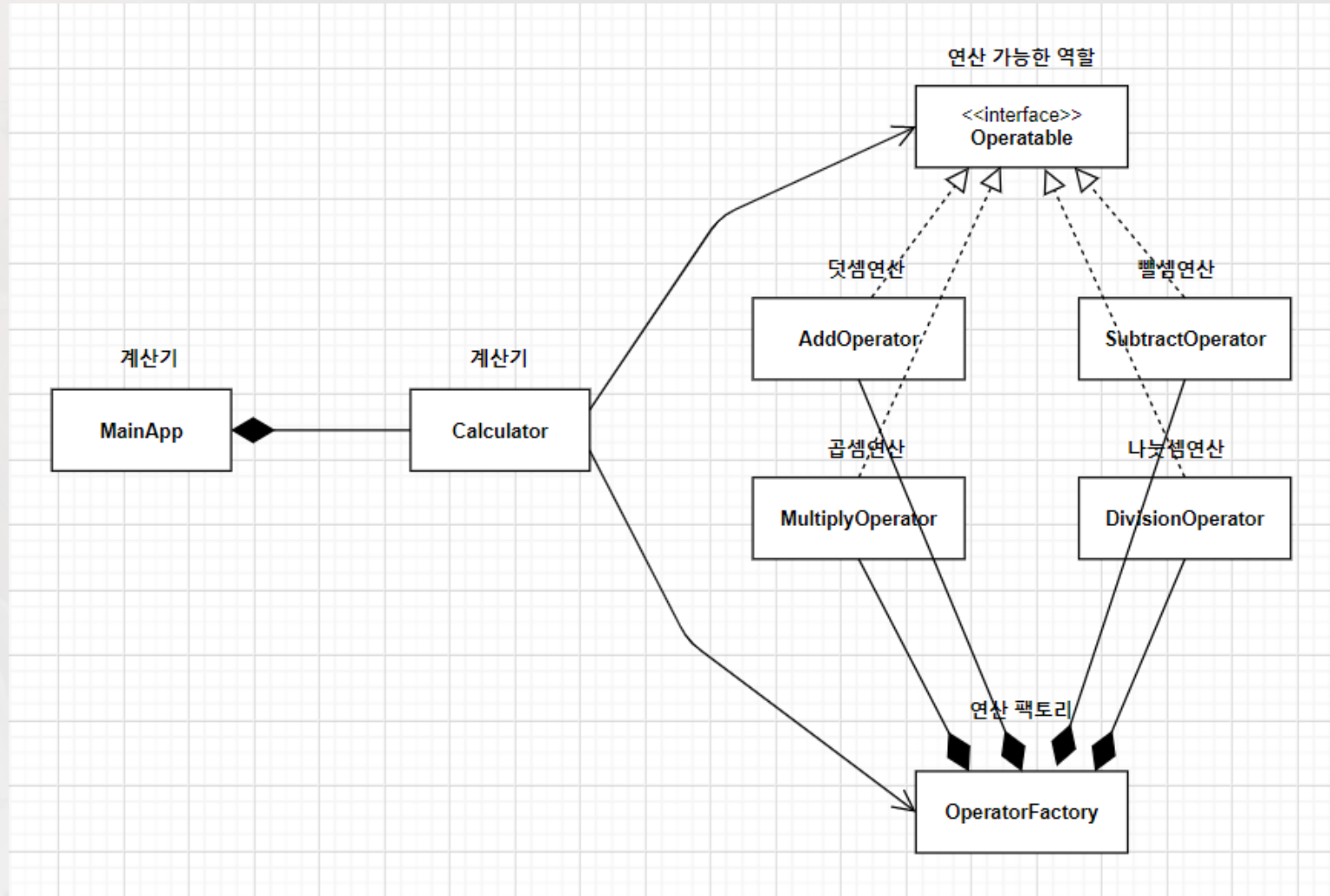
```
/**
 * 계산기 클래스
 */
public class Calculator { // 클래스 네이밍 컨벤션은 PascalCase 이다.

    // 상수 네이밍 컨벤션은 전체 대문자이다.
    // 전역변수 역시 클래스 내부에 있어야한다.
    final static int MY_VALUE = 20;

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

    public static void main(String[] args) {
        int initialValue = MY_VALUE;
        // 변수 선언 방법은, `데이터타입` `변수이름` 을 나란히 작성한다.
        // 변수, 메서드 네이밍 컨벤션은 camelCase 이다.
        int resultValue = sum(a, 10);
        System.out.println(a); // 20
        System.out.println(s); // 30
    }
}
```

## Java 계산기 예제



A faded background image of a man with a beard and glasses, wearing a white t-shirt with a mathematical formula. The text "2주차 - Java 문법" is overlaid on the right side of the image.

## 2주차 - Java 문법

## 클래스 선언

접근지정자 class 클래스이름 {...}

```
package classtest.model;  
  
public class Person {  
  
}
```

## 클래스 인스턴스 생성

클래스이름 인스턴스이름 = new 클래스이름(인자);

```
package classtest;

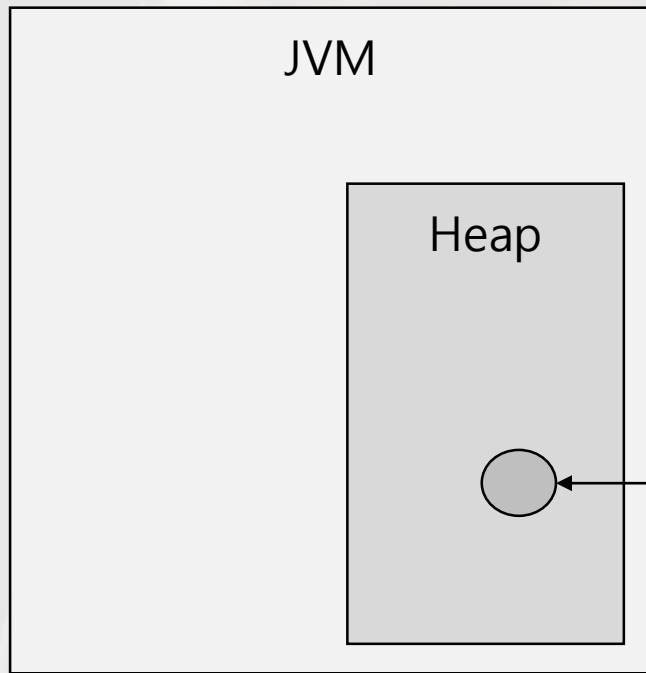
import classtest.model.Person;

public class MainApp {

    public static void main(String[] args) {
        Person person = new Person();
    }
}
```

new 연산자

클래스 타입의 인스턴스를 동적으로 생성



- Heap 메모리 영역을 동적으로 할당 받아 해당 메모리의 주소를 반환
- 클래스의 생성자를 호출

```
Person person  
= new Person();
```



## 생성자

```
public 클래스이름(인자타입 인자, ... ) {...}
```

- new 연산자에 의해 인스턴스가 생성되는 시점에 호출되며 인스턴스 초기화 로직을 수행하는 메서드
- 메서드 이름은 클래스 이름과 같아야 하며 리턴 타입이 없음
- 생성자가 작성되지 않은 클래스의 경우, 컴파일러에 의해 디폴트 생성자가 자동으로 작성된다. (생성자가 하나라도 작성된 클래스의 경우, 컴파일러가 디폴트 생성자를 작성하지 않는다.)

## 클래스 접근지정자

다른 클래스에서 이 클래스를 import 할 수 있는지 허용 여부

공개

- **public**

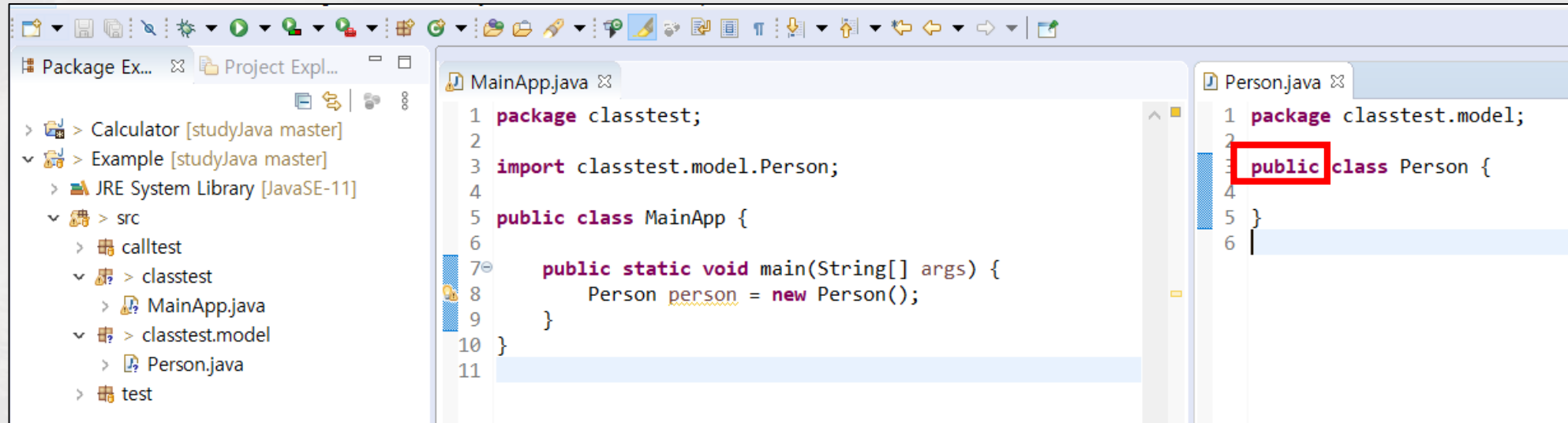
- 외부 클래스에서 import 가능

- **없음** (디폴트 접근지정자)

- 동일한 패키지내 모든 클래스가 import 가능

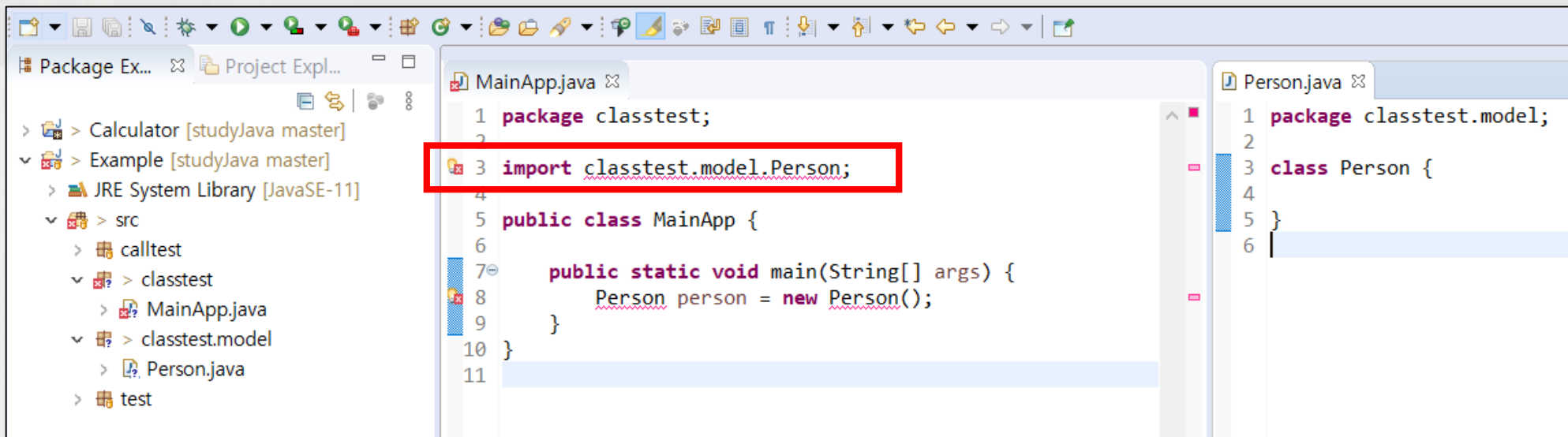
폐쇄

## 클래스 접근지정자



```
1 package classtest;
2
3 import classtest.model.Person;
4
5 public class MainApp {
6
7     public static void main(String[] args) {
8         Person person = new Person();
9     }
10 }
11
```

```
1 package classtest.model;
2
3 public class Person {
4
5 }
6
```

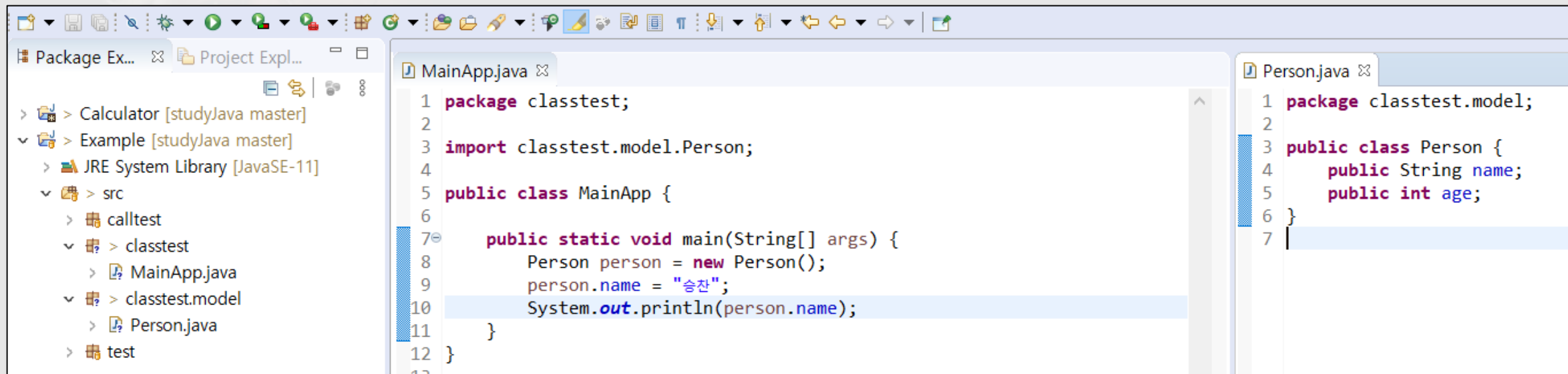


```
1 package classtest;
2
3 import classtest.model.Person;
4
5 public class MainApp {
6
7     public static void main(String[] args) {
8         Person person = new Person();
9     }
10 }
11
```

```
1 package classtest.model;
2
3 class Person {
4
5 }
6
```

## 멤버 변수

접근지정자 타입 변수이름[ = 초기값];



## 멤버함수

접근지정자 리턴타입 메서드이름(인자타입 인자) {...}

The screenshot shows an IDE with two Java files open: `MainApp.java` and `Person.java`.

**MainApp.java** (lines 1-22):

```
1 package classtest;
2
3 import classtest.model.Person;
4
5 public class MainApp {
6
7     public static void main(String[] args) {
8         Person vampire = new Person();
9         vampire.setName("뱀파이어");
10        vampire.setAge(120);
11
12        Person victim = new Person();
13        victim.setName("희생자");
14        victim.setAge(20);
15
16        // 호출
17        vampire.swapAge(victim);
18        System.out.println(vampire.getAge());
19        System.out.println(victim.getAge());
20    }
21 }
22
```

**Person.java** (lines 1-24):

```
1 package classtest.model;
2
3 public class Person {
4     private String name;
5     private int age;
6
7     public void swapAge(Person target) {
8         int tempAge = this.age;
9         this.age = target.getAge();
10        target.setAge(tempAge);
11    }
12
13    public String getName() {
14        return name;
15    }
16    public void setName(String name) {
17        this.name = name;
18    }
19    public int getAge() {
20        return age;
21    }
22    public void setAge(int age) {
23        this.age = age;
24    }
25 }
```

Annotations in the image:

- A red box highlights the line `int tempAge = this.age;` in `Person.java` with the label **지역변수** (Local Variable).
- A red box highlights the line `this.name = name;` in `Person.java` with the label **멤버 메서드의 클래스 인스턴스 레퍼런스** (Member Method's Class Instance Reference).

At the bottom, the console output shows:

```
<terminated> MainApp (2) [Java Application] C:\Program Files\RedHat\java-11-openjdk-11.0.7-1\bin\javaw.exe (2020. 5. 18. 오전 3:42:10 - 오전 3:42:11)
20
120
```

## 멤버 접근지정자

다른 클래스에서 이 클래스의 멤버에 접근 할 수 있는지 허용 여부

- **public**

- 클래스 내부/외부에서 모두 호출 가능

- **protected**

- 클래스 내부에서 호출가능
  - 상속 받은 자식 클래스에서 (클래스 외부) 호출 가능

- **없음** (디폴트 접근지정자)

- 동일한 패키지내 모든 클래스가 호출 가능

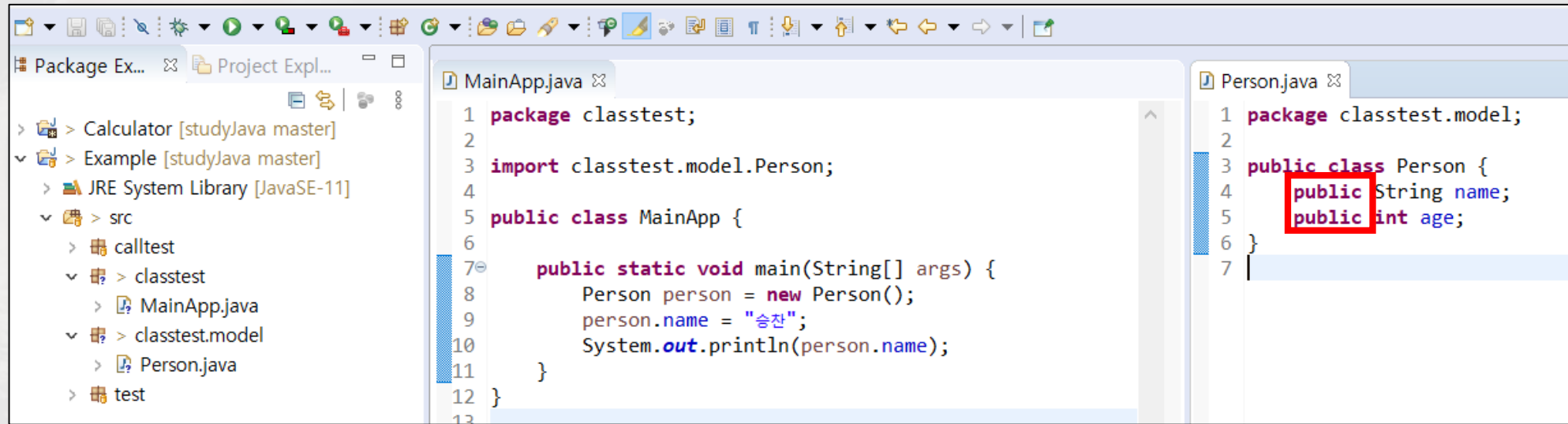
- **private**

- 클래스 내부에서만 호출 가능

공개

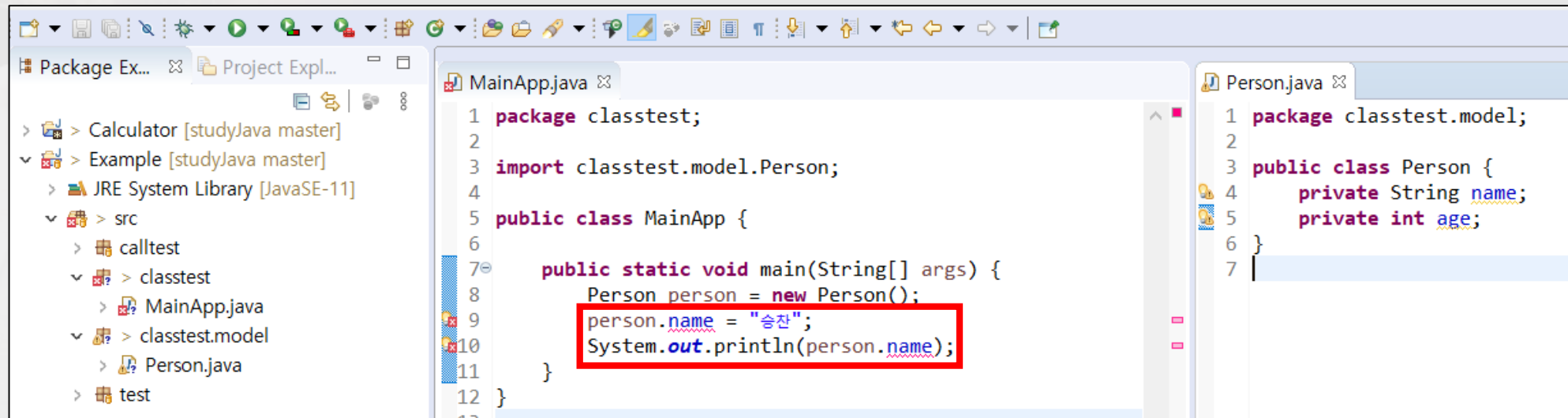
폐쇄

## 멤버 접근지정자



```
1 package classtest;
2
3 import classtest.model.Person;
4
5 public class MainApp {
6
7     public static void main(String[] args) {
8         Person person = new Person();
9         person.name = "승찬";
10        System.out.println(person.name);
11    }
12 }
13
```

```
1 package classtest.model;
2
3 public class Person {
4     public String name;
5     public int age;
6 }
7
```



```
1 package classtest;
2
3 import classtest.model.Person;
4
5 public class MainApp {
6
7     public static void main(String[] args) {
8         Person person = new Person();
9         person.name = "승찬";
10        System.out.println(person.name);
11    }
12 }
13
```

```
1 package classtest.model;
2
3 public class Person {
4     private String name;
5     private int age;
6 }
7
```

## 결론

```
1 package classtest;
2
3 import classtest.model.Person;
4
5 public class MainApp {
6
7     public static void main(String[] args) {
8         Person person = new Person("송찬", 31);
9         System.out.println(person.toString());
10    }
11 }
12
```

```
1 package classtest.model;
2
3 public class Person {
4     private String name;
5     private int age;
6
7     public Person() { }
8
9     public Person(String name, int age) {
10         this.name = name;
11         this.age = age;
12    }
13
14    public String getName() {
15        return name;
16    }
17
18    public void setName(String name) {
19        this.name = name;
20    }
21
22    public int getAge() {
23        return age;
24    }
25
26    public void setAge(int age) {
27        this.age = age;
28    }
29
30    @Override
31    public String toString() {
32        return "Person [name=" + name
33            + ", age=" + age + "]";
34    }
35 }
```

<terminated> MainApp (2) [Java Application] C:\Program Files\RedHat\java-11-openjdk-11.0.7-1\bin\javaw.exe (2020. 5. 18. 오전 3:04:50 - 오전 3:04:50)  
Person [name=송찬, age=31]

“

캡슐화 원칙을 지켜 클래스를 정의 및 멤버를 호출하는 코드를 작성하자

”



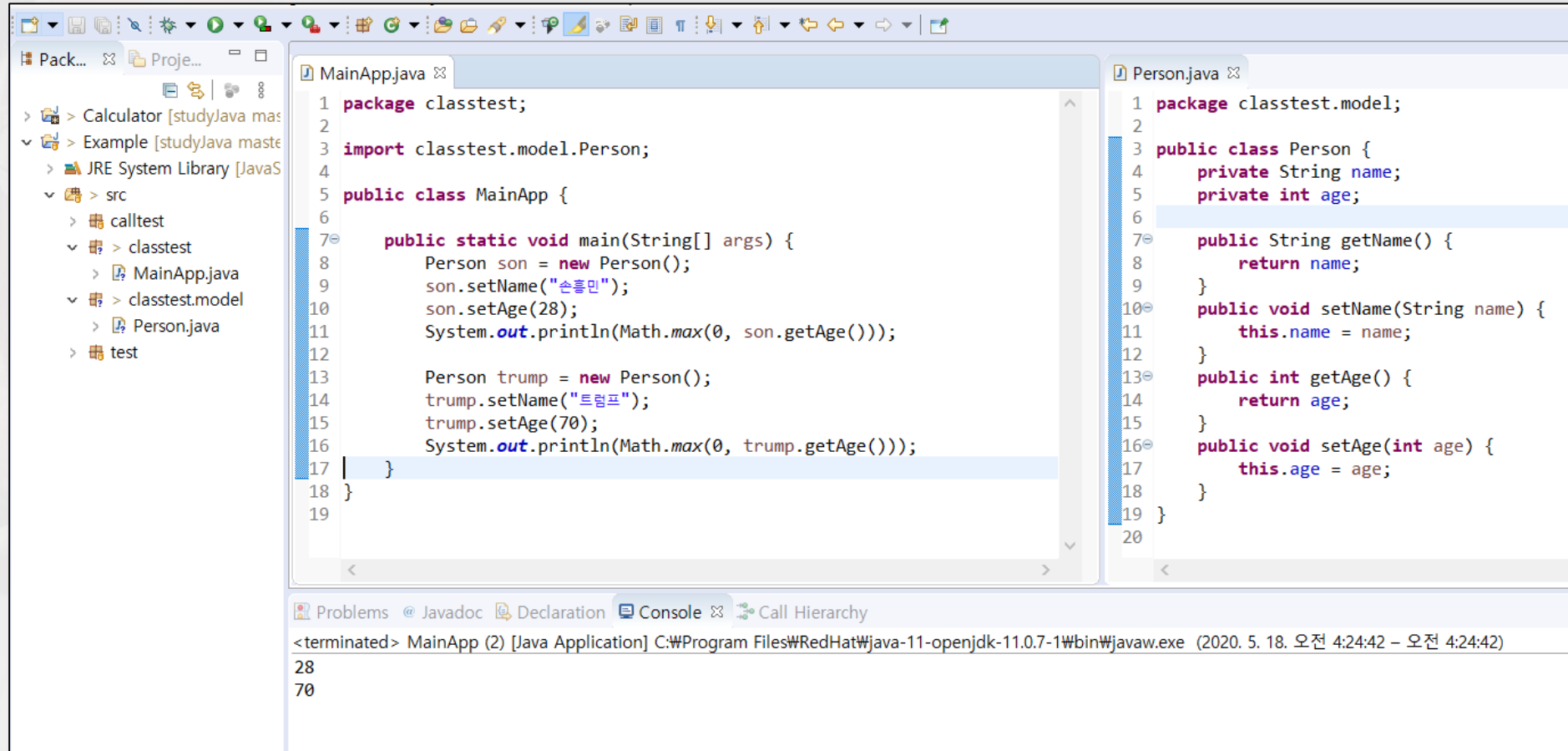
## static 멤버

접근지정자 **static** 리턴타입 메서드이름(인자타입 인자) {...}

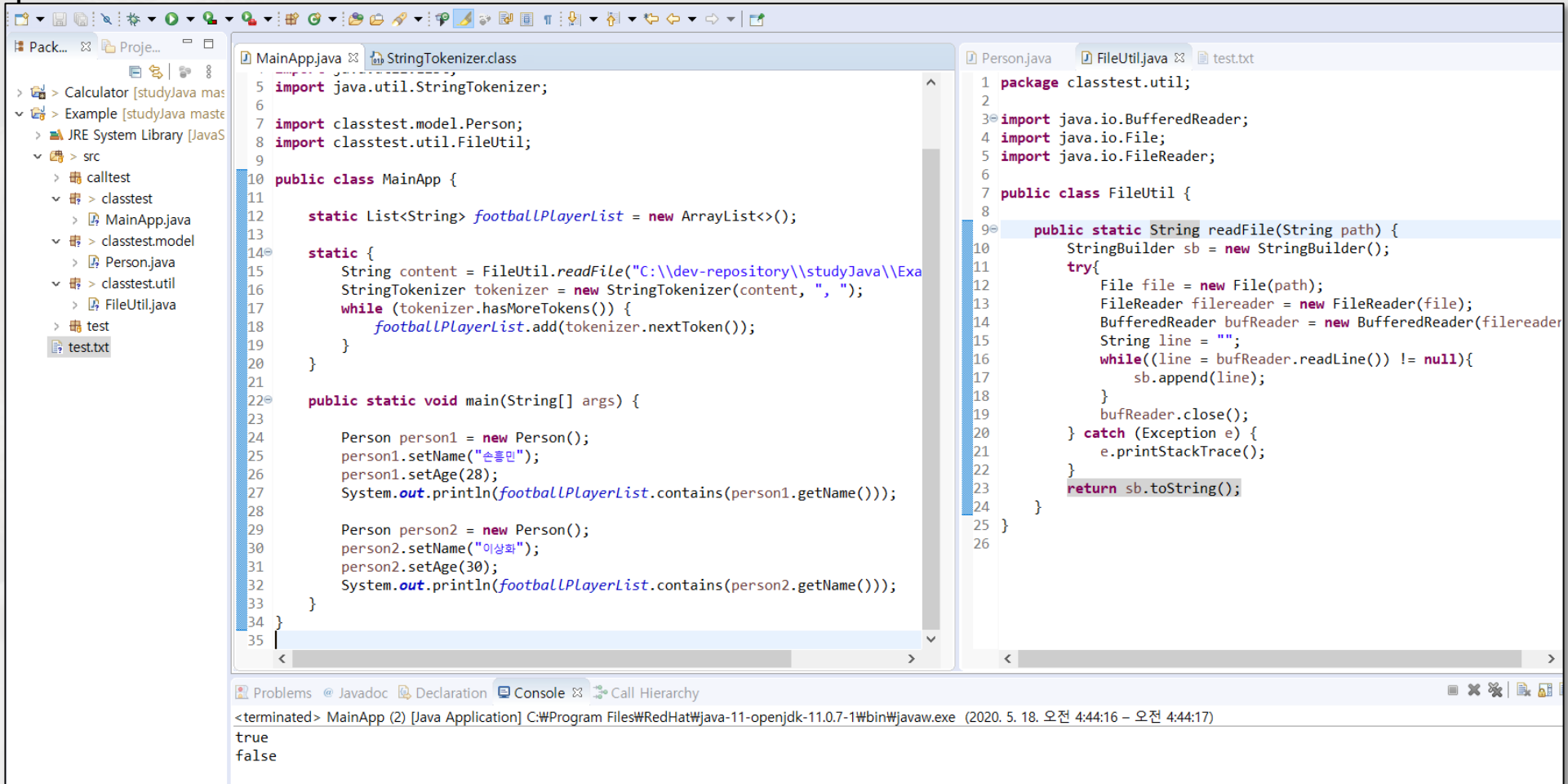
접근지정자 **static** 타입 변수이름[ = 초기값];

- JVM에 의해 java 프로그램 실행시 클래스를 로드하는데 클래스에 static 멤버가 있다면 메모리를 할당하며, 인스턴스의 라이프 사이클과는 관련 없이 프로그램 종료시 메모리 반환.
- 프로그램 실행 시점에 메모리 할당이 필요한 경우 사용한다.
- 모든 클래스에서 접근가능한 전역 적인 멤버가 필요할 때 사용한다.
- 동일한 클래스의 모든 인스턴스들이 공유하기 때문에 공유 멤버가 필요할 때 사용한다.
- static 메서드는 오직 static 멤버만 접근 가능하다.
- static 메서드 내에 this 키워드 사용 불가.

## static 멤버



## static 멤버



```
5 import java.util.StringTokenizer;
6
7 import classtest.model.Person;
8 import classtest.util.FileUtil;
9
10 public class MainApp {
11
12     static List<String> footballPlayerList = new ArrayList<>();
13
14     static {
15         String content = FileUtil.readFile("C:\\dev-repository\\studyJava\\Exa
16         StringTokenizer tokenizer = new StringTokenizer(content, ", ");
17         while (tokenizer.hasMoreTokens()) {
18             footballPlayerList.add(tokenizer.nextToken());
19         }
20     }
21
22     public static void main(String[] args) {
23
24         Person person1 = new Person();
25         person1.setName("손흥민");
26         person1.setAge(28);
27         System.out.println(footballPlayerList.contains(person1.getName()));
28
29         Person person2 = new Person();
30         person2.setName("이성화");
31         person2.setAge(30);
32         System.out.println(footballPlayerList.contains(person2.getName()));
33     }
34 }
35
```

```
1 package classtest.util;
2
3 import java.io.BufferedReader;
4 import java.io.File;
5 import java.io.FileReader;
6
7 public class FileUtil {
8
9     public static String readFile(String path) {
10         StringBuilder sb = new StringBuilder();
11         try{
12             File file = new File(path);
13             FileReader filereader = new FileReader(file);
14             BufferedReader bufReader = new BufferedReader(filereader);
15             String line = "";
16             while((line = bufReader.readLine()) != null){
17                 sb.append(line);
18             }
19             bufReader.close();
20         } catch (Exception e) {
21             e.printStackTrace();
22         }
23         return sb.toString();
24     }
25 }
26
```

Problems Javadoc Declaration Console Call Hierarchy

<terminated> MainApp (2) [Java Application] C:\Program Files\RedHat\java-11-openjdk-11.0.7-1\bin\javaw.exe (2020. 5. 18. 오전 4:44:16 – 오전 4:44:17)

true  
false

//

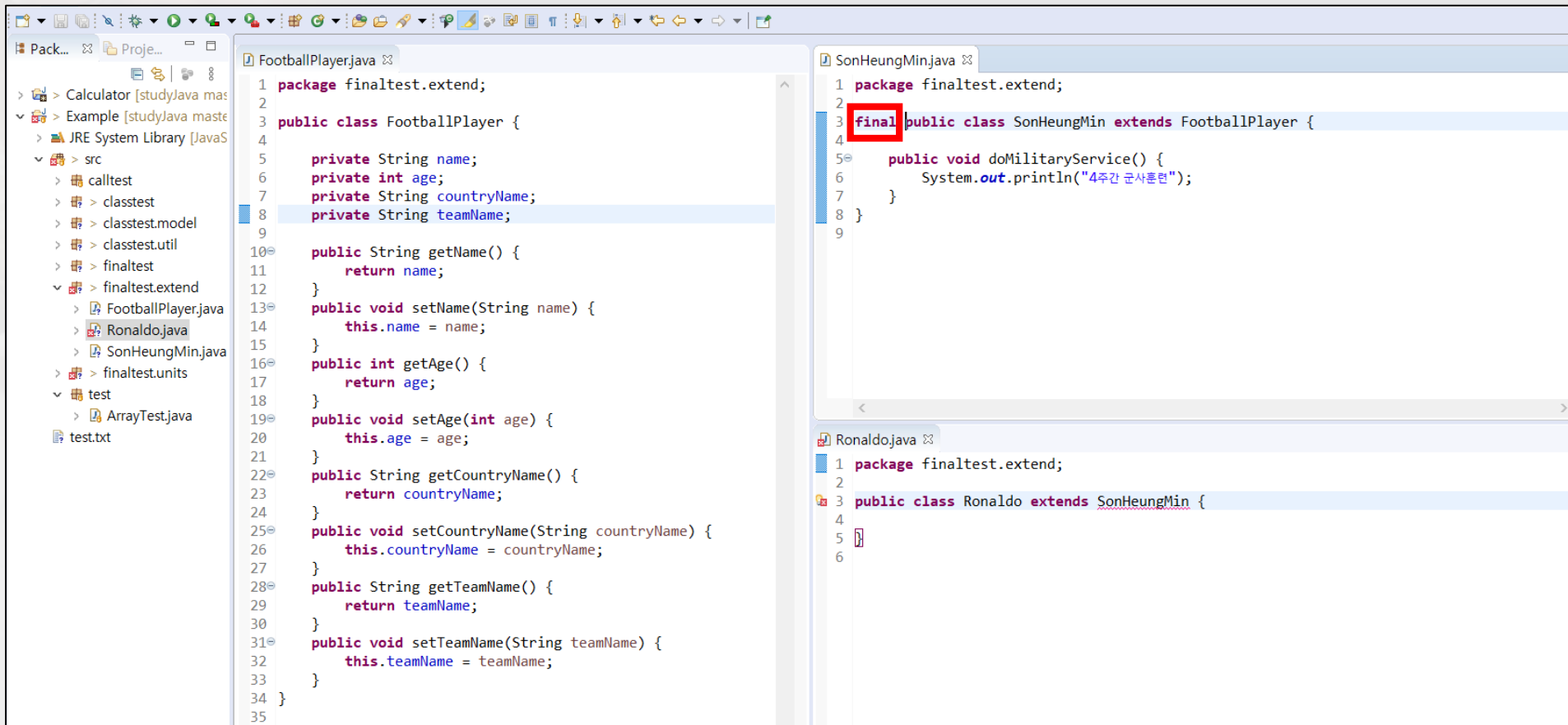
주로, 전역적으로 쓰일 기능 (Utility) 필요시  
또는 프로그램 시작시 데이터 로드 필요시 static 키워드를 붙인다.

//

## final 클래스

**final** 접근지정자 class 클래스이름 {...}

- final 클래스를 상속받을 수 없다.



The screenshot shows an IDE with three Java files open. The left pane shows a project structure with a package named 'finaltest.extend'. The main editor displays the following code:

```
FootballPlayer.java
1 package finaltest.extend;
2
3 public class FootballPlayer {
4
5     private String name;
6     private int age;
7     private String countryName;
8     private String teamName;
9
10    public String getName() {
11        return name;
12    }
13    public void setName(String name) {
14        this.name = name;
15    }
16    public int getAge() {
17        return age;
18    }
19    public void setAge(int age) {
20        this.age = age;
21    }
22    public String getCountryName() {
23        return countryName;
24    }
25    public void setCountryName(String countryName) {
26        this.countryName = countryName;
27    }
28    public String getTeamName() {
29        return teamName;
30    }
31    public void setTeamName(String teamName) {
32        this.teamName = teamName;
33    }
34 }
35
```

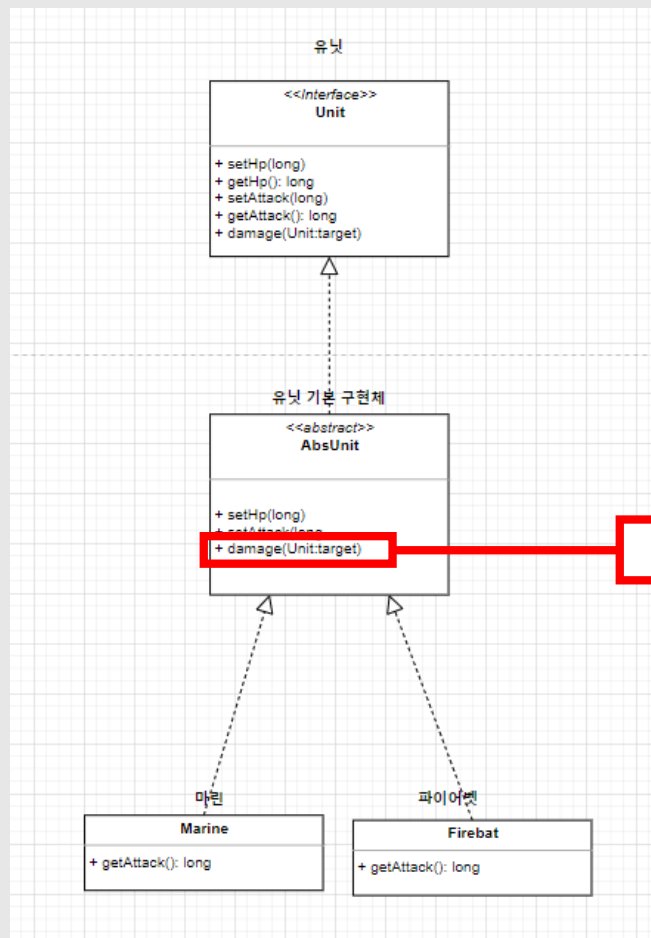
```
SonHeungMin.java
1 package finaltest.extend;
2
3 final public class SonHeungMin extends FootballPlayer {
4
5     public void doMilitaryService() {
6         System.out.println("4주간 군사훈련");
7     }
8 }
9
```

```
Ronaldo.java
1 package finaltest.extend;
2
3 public class Ronaldo extends SonHeungMin {
4
5 }
6
```

## final 메서드

**final** 접근지정자 리턴타입 메서드이름 (인자타입 인자, ...) {...}

- final 메서드를 재정의 할 수 없다.

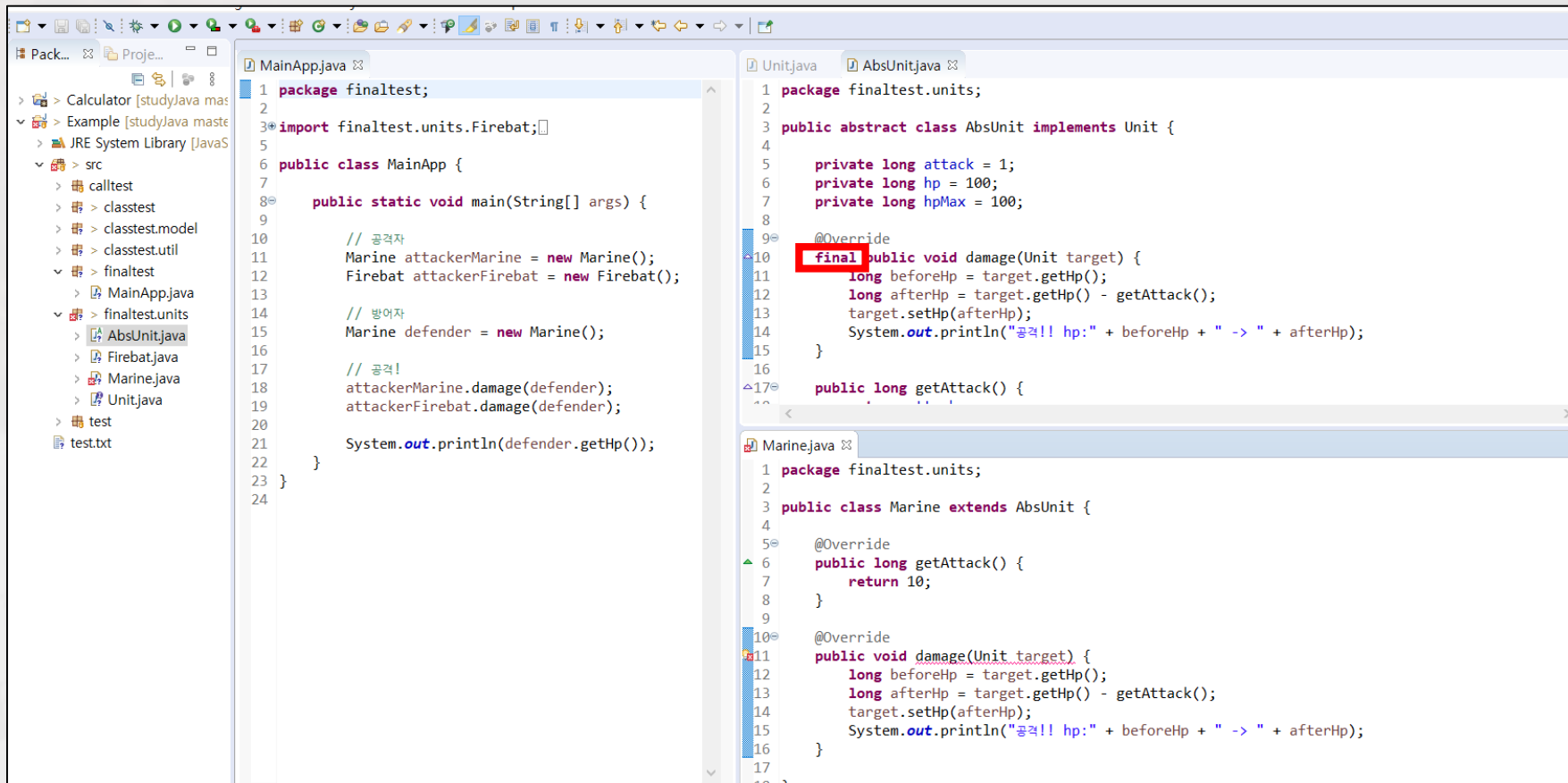


절대로 재정의하면 안되는 전투공식

## final 메서드

**final** 접근지정자 리턴타입 메서드이름 (인자타입 인자, ...) {...}

- final 메서드를 재정의 할 수 없다.



The screenshot shows an IDE with three Java files open: MainApp.java, AbsUnit.java, and Marine.java. The left sidebar shows a project structure with a 'finaltest' package containing 'MainApp.java', 'AbsUnit.java', 'Firebat.java', 'Marine.java', 'Unit.java', and 'test.txt'.

```
1 package finaltest;
2
3 import finaltest.units.Firebat;
4
5 public class MainApp {
6
7     public static void main(String[] args) {
8
9         // 공격자
10        Marine attackerMarine = new Marine();
11        Firebat attackerFirebat = new Firebat();
12
13        // 방어자
14        Marine defender = new Marine();
15
16        // 공격!
17        attackerMarine.damage(defender);
18        attackerFirebat.damage(defender);
19
20        System.out.println(defender.getHp());
21    }
22 }
23
24
```

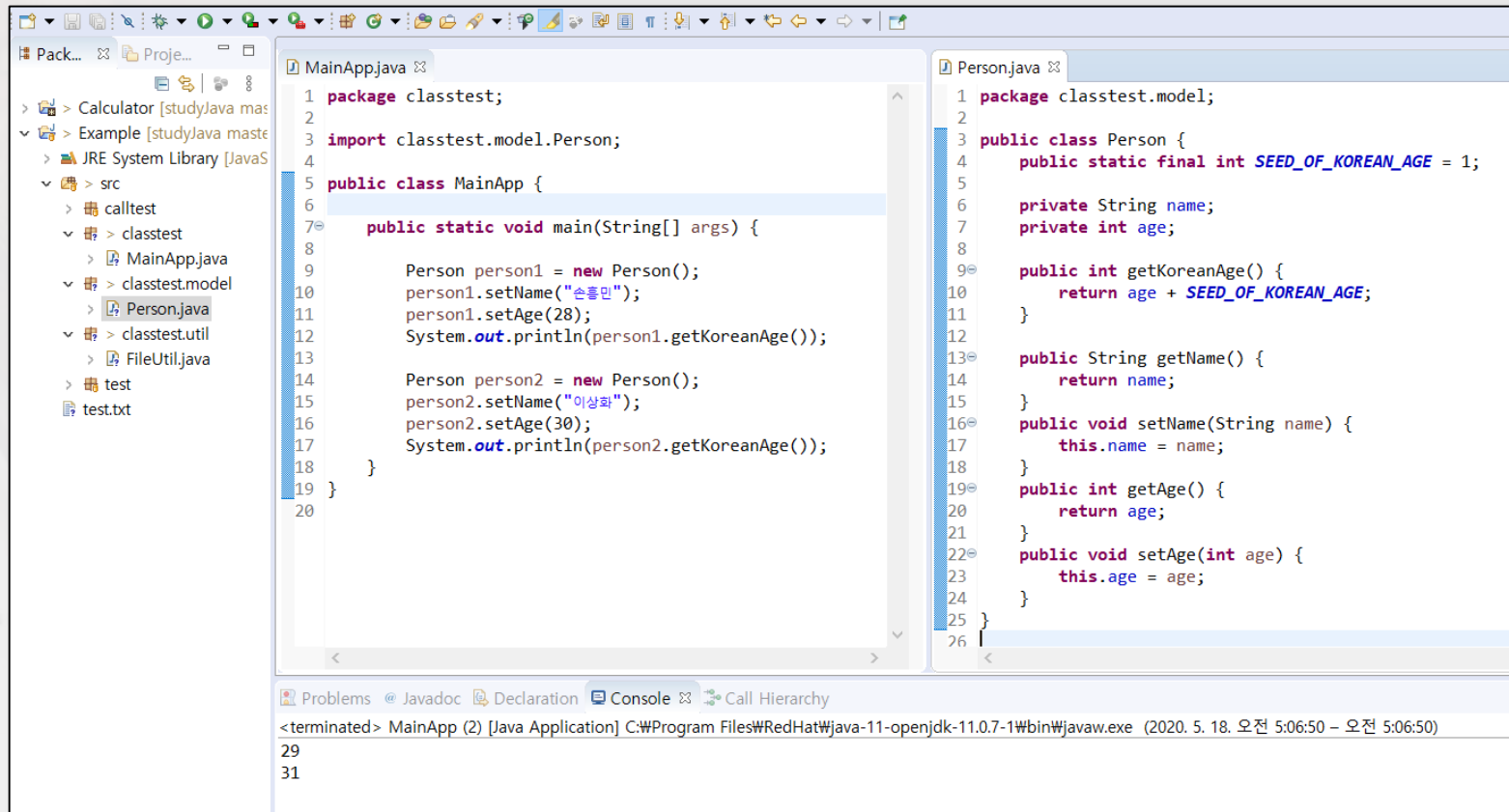
```
1 package finaltest.units;
2
3 public abstract class AbsUnit implements Unit {
4
5     private long attack = 1;
6     private long hp = 100;
7     private long hpMax = 100;
8
9     @Override
10    final public void damage(Unit target) {
11        long beforeHp = target.getHp();
12        long afterHp = target.getHp() - getAttack();
13        target.setHp(afterHp);
14        System.out.println("공격!! hp:" + beforeHp + " -> " + afterHp);
15    }
16
17    public long getAttack() {
18        return attack;
19    }
20 }
21
```

```
1 package finaltest.units;
2
3 public class Marine extends AbsUnit {
4
5     @Override
6     public long getAttack() {
7         return 10;
8     }
9
10    @Override
11    public void damage(Unit target) {
12        long beforeHp = target.getHp();
13        long afterHp = target.getHp() - getAttack();
14        target.setHp(afterHp);
15        System.out.println("공격!! hp:" + beforeHp + " -> " + afterHp);
16    }
17 }
18
```

## final 상수

**final** [static] 타입 상수이름 = 초기값;

- final 멤버변수는 선언시 초기값을 지정해야 하며, 한번 정의되면 값을 변경할 수 없어 상수로 쓴다.



```
1 package classtest;
2
3 import classtest.model.Person;
4
5 public class MainApp {
6
7     public static void main(String[] args) {
8
9         Person person1 = new Person();
10        person1.setName("손흥민");
11        person1.setAge(28);
12        System.out.println(person1.getKoreanAge());
13
14        Person person2 = new Person();
15        person2.setName("이상화");
16        person2.setAge(30);
17        System.out.println(person2.getKoreanAge());
18    }
19 }
20
```

```
1 package classtest.model;
2
3 public class Person {
4     public static final int SEED_OF_KOREAN_AGE = 1;
5
6     private String name;
7     private int age;
8
9     public int getKoreanAge() {
10         return age + SEED_OF_KOREAN_AGE;
11     }
12
13     public String getName() {
14         return name;
15     }
16     public void setName(String name) {
17         this.name = name;
18     }
19     public int getAge() {
20         return age;
21     }
22     public void setAge(int age) {
23         this.age = age;
24     }
25 }
26
```

Problems @ Javadoc Declaration Console Call Hierarchy

<terminated> MainApp (2) [Java Application] C:\Program Files\RedHat\java-11-openjdk-11.0.7-1\bin\javaw.exe (2020. 5. 18. 오전 5:06:50 – 오전 5:06:50)

29

31



## final 상수

**final** [static] 타입 상수이름 = 초기값;

- final 멤버변수는 선언시 초기값을 지정해야 하며, 한번 정의되면 값을 변경할 수 없어 상수로 쓴다.

```
1 package classtest;
2
3 import classtest.model.Person;
4
5 public class MainApp {
6
7     public static void main(String[] args) {
8
9         Person person1 = new Person();
10        person1.setName("손흥민");
11        person1.setAge(28);
12        System.out.println(person1.getKoreanAge());
13
14        Person person2 = new Person();
15        person2.setName("이상화");
16        person2.setAge(30);
17        System.out.println(person2.getKoreanAge());
18    }
19 }
20
```

```
1 package classtest.model;
2
3 public class Person {
4     public static final int SEED_OF_KOREAN_AGE = 1;
5
6     private String name;
7     private int age;
8
9     public int getKoreanAge() {
10         return age + SEED_OF_KOREAN_AGE;
11     }
12
13     public String getName() {
14         return name;
15     }
16     public void setName(String name) {
17         this.name = name;
18     }
19     public int getAge() {
20         return age;
21     }
22     public void setAge(int age) {
23         this.age = age;
24     }
25 }
26
```

상수 선언시 final static 키워드를 붙여 사용하고  
또는 클래스 상속이나 메서드 오버라이딩을 막기위해 쓰인다.

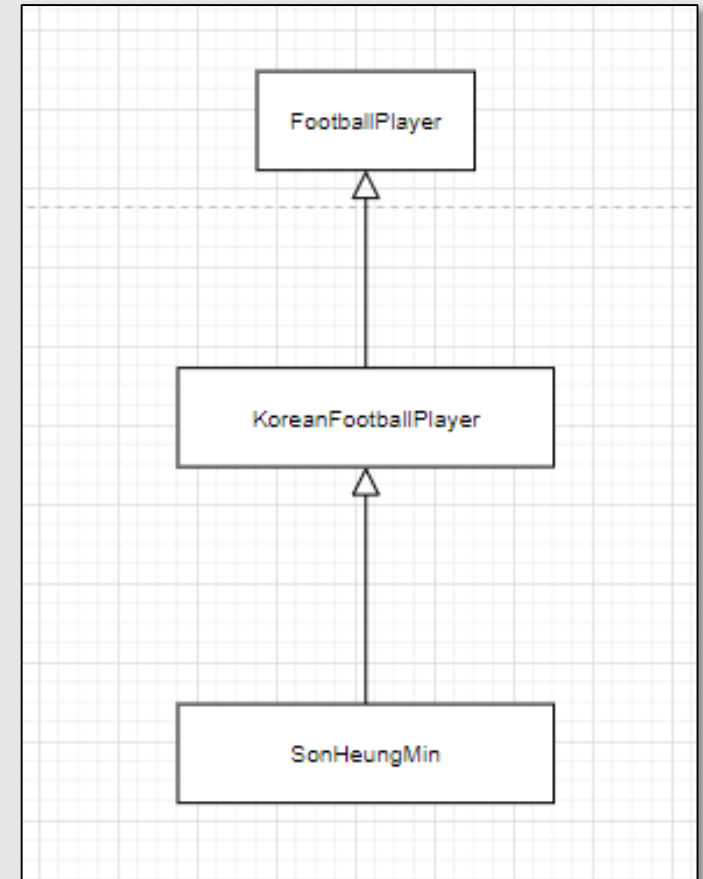
Problems @ Javadoc Declaration Console Call Hierarchy  
<terminated> MainApp (2) [Java Application] C:\Program Files\RedHat\java-11-openjdk-11.0.7-1\bin\javaw.exe (2020. 5. 18. 오전 5:06:50 – 오전 5:06:50)  
29  
31



## 상속 선언

접근지정자 자식클래스이름 **extends** 부모클래스 {...}

- 다중 상속을 지원하지 않는다.
- 상속 계층 최상위에 있는 클래스는 java.lang.Object 클래스 이다.  
(모든 클래스가 Object를 상속받도록 컴파일된다)
- 자식클래스 생성시 자식 → 부모 순서로 생성자를 호출하며,  
부모 → 자식 순서로 생성자가 실행된다.
- 부모 생성자 호출시 super(인자타입 인자); 를 호출할 수 있다.



## 상속 선언

The screenshot displays an IDE with the following components:

- Project Explorer (Left):** Shows a project structure with packages like `finaltest`, `finaltest.extend`, and `finaltest.units`. The `finaltest.extend` package contains `FootballPlayer.java`, `KoreanFootballPlayer.java`, and `Ronaldo.java`.
- MainApp.java:**

```
1 package finaltest;
2
3 import finaltest.extend.SonHeungMin;
4
5 public class MainApp {
6
7     public static void main(String[] args) {
8         new SonHeungMin();
9     }
10 }
11
```
- FootballPlayer.java:**

```
1 package finaltest.extend;
2
3 public class FootballPlayer {
4
5     private String name;
6     private int age;
7     private String countryName;
8     private String teamName;
9
10    public FootballPlayer() {
11        System.out.println("init FootballPlayer");
12    }
13
```
- KoreanFootballPlayer.java:**

```
1 package finaltest.extend;
2
3 public class KoreanFootballPlayer extends FootballPlayer {
4
5     public KoreanFootballPlayer() {
6         System.out.println("init KoreanFootballPlayer");
7     }
8
```
- Ronaldo.java:**

```
1 package finaltest.extend;
2
3 final public class SonHeungMin extends KoreanFootballPlayer {
4
5     public SonHeungMin() {
6         System.out.println("init SonHeungMin");
7     }
8
```
- Console (Bottom):**

```
<terminated> MainApp (3) [Java Application] C:\Program Files\RedHat\java-11-openjdk-11.0.7-1\bin\javaw.exe (2020. 5. 18. 오전 6:26:04 - 오전 6:26:04)
init FootballPlayer
init KoreanFootballPlayer
init SonHeungMin
```

## 상속 선언

```
package finaltest;
import finaltest.extend.SonHeungMin;

public class MainApp {

    public static void main(String[] args) {
        System.out.println(new SonHeungMin().getName());
    }
}
```

```
public class FootballPlayer {

    private String name;
    private int age;
    private String countryName;
    private String teamName;

    public FootballPlayer() {
        System.out.println("init FootballPlayer");
    }

    public FootballPlayer(String name) {
        this.name = name;
        System.out.println("init FootballPlayer");
    }
}
```

```
public class KoreanFootballPlayer extends FootballPlayer {

    public KoreanFootballPlayer() {
        System.out.println("init KoreanFootballPlayer");
    }

    public KoreanFootballPlayer(String name) {
        super(name);
        System.out.println("init KoreanFootballPlayer");
    }

    public int getMilitaryServiceMonths() {
        return 21;
    }
}
```

```
package finaltest.extend;

final public class SonHeungMin extends KoreanFootballPlayer {

    public SonHeungMin() {
        super("손흥민");
        System.out.println("init SonHeungMin");
    }

    @Override
}
```

<terminated> MainApp (3) [Java Application] C:\Program Files\RedHat\java-11-openjdk-11.0.7-1\bin\javaw.exe (2020. 5. 18. 오전 6:29:43 - 오전 6:29:46)  
init FootballPlayer  
init KoreanFootballPlayer  
init SonHeungMin  
손흥민

## 추상클래스 선언

접근지정자 **abstract** class 클래스이름 {...}

- 인스턴스를 생성할 수 없는 클래스 (구현되지 않은 추상메서드가 존재)
- 설계와 구현을 분리하여 개발하기 위함
- 계층적 상속 관계 클래스 구조를 만들때 적합

## 추상메서드 선언

접근지정자 **abstract** 리턴타입 메서드이름(인자타입 인자이름 ...);

- 구현하지 않고 선언만 되어있는 메서드

## 인터페이스 선언

접근지정자 **interface** 타입이름 {...}

- 인스턴스를 생성할 수 없는 클래스 (구현되지 않은 추상메서드가 존재)
- 설계와 구현을 분리하여 개발하기 위함
- 계층적 상속 관계 클래스 구조를 만들때 적합
- 모든 메서드의 접근지정자가 public 이어야함 (생략시에도 public 이다)
- 레퍼런스 타입으로 사용된다.
- 다른 인터페이스를 extends 키워드로 상속 받을 수 있다.
- 다중 상속이 가능하다.

## 인터페이스 구현

접근지정자 [abstract] class 클래스이름 **implements** 인터페이스이름 {...}

- 모든 추상메서드를 구현해야한다.
- 다중 구현이 가능하다.
- 상속과 구현이 가능하다.

## instanceof 연산자

### 인스턴스 instanceof 클래스

- 인스턴스가 클래스 타입인가 판별해 리턴

```
7 public class MainApp {  
8  
9     public static void main(String[] args) {  
10         SonHeungMin sonHeungMin = new SonHeungMin();  
11         KoreanFootballPlayer koreanFootballPlayer = new KoreanFootballPlayer();  
12         FootballPlayer footballPlayer = new FootballPlayer();  
13  
14         boolean sonHeungMinIsSonHeungMin = sonHeungMin instanceof SonHeungMin;  
15         boolean sonHeungMinIsKoreanFootballPlayer = sonHeungMin instanceof KoreanFootballPlayer;  
16         boolean sonHeungMinIsFootballPlayer = sonHeungMin instanceof FootballPlayer;  
17         boolean footballPlayerIsSonHeungMin = footballPlayer instanceof SonHeungMin;  
18  
19         System.out.println(sonHeungMinIsFootballPlayer);  
20         System.out.println(sonHeungMinIsKoreanFootballPlayer);  
21         System.out.println(sonHeungMinIsFootballPlayer);  
22         System.out.println(footballPlayerIsSonHeungMin);  
23     }  
24 }  
25
```

<terminated> MainApp (3) [Java Application] C:\Program Files\RedHat\java-11-openjdk-11.0.7-1\bin\javaw.exe (2020. 5. 18. 오전 6:4

true  
true  
true  
false

## 업캐스팅

부모클래스 인스턴스이름 = 자식인스턴스;

- 자식 인스턴스를 부모 클래스로 강제 타입 변환
- 자식 인스턴스만 호출 가능한 멤버변수가 접근 불가능 할 수 있음

```
public class MainApp {  
    public static void main(String[] args) {  
        SonHeungMin sonHeungMin = new SonHeungMin();  
        System.out.println(sonHeungMin.getMilitaryServiceMonths());  
  
        FootballPlayer footballPlayer = new SonHeungMin();  
        System.out.println(footballPlayer.getMilitaryServiceMonths());  
    }  
}
```



## 다운캐스팅

자식클래스 인스턴스이름 = (자식클래스) 부모인스턴스;

- 부모 인스턴스를 자식 클래스로 강제 타입 변환
- 명시적으로 개발자가 타입을 지정해야하며 자식클래스가 아닐 경우 런타임 에러 발생

```
7 public class MainApp {
8
9     public static void main(String[] args) {
10         FootballPlayer footballPlayer = new KoreanFootballPlayer("임격정");
11         System.out.println(footballPlayer.getName());
12
13         // 다운캐스팅 성공
14         KoreanFootballPlayer koreanFootballPlayer = (KoreanFootballPlayer) footballPlayer;
15         System.out.println(koreanFootballPlayer.getMilitaryServiceMonths());
16
17         // 다운캐스팅 실패!!
18         // ClassCastException 발생!! cannot be cast to class
19         SonHeungMin sonHeungMin = (SonHeungMin) footballPlayer;
20         System.out.println(sonHeungMin.getMilitaryServiceMonths());
21     }
22 }
23
```

Problems Javadoc Declaration Console Call Hierarchy

<terminated> MainApp (3) [Java Application] C:\Program Files\RedHat\java-11-openjdk-11.0.7-1\bin\javaw.exe (2020. 5. 18. 오전 6:54:41 - 오전 6:54:42)

임격정  
21  
Exception in thread "main" java.lang.ClassCastException: class finaltest.extend.KoreanFootballPlayer cannot be cast to class finaltest.extend.SonHeungMin at finaltest.MainApp.main(MainApp.java:19)

## 메서드 오버라이딩

[@Override] 접근지정자 같은리턴타입 같은메서드이름 (같은인자 ...) {...}

- 부모 클래스의 메서드와 동일한 이름의 메서드를 자식 메서드에 중복으로 선언되었고 자식 클래스의 해당 메서드가 호출되면 자식 메서드가 실행된다.  
(동적 바인딩 : 실행할 메서드를 컴파일 타임이 아닌 런타임에 결정)
- 자식클래스가 오버라이딩한 메서드는 부모의 메서드보다 접근지정자가 같거나 넓을 수 있다,  
(접근지정자 범위가 좁아질 수 없다)
- 부모클래스에 static, private, final 로 선언된 메서드는 오버라이딩할 수 없다.
- 자식클래스가 오버라이딩 했으나 super.메서드이름 을 호출시 부모클래스의 메서드가 실행된다.  
(super : 부모클래스 레퍼런스)

## 메서드 오버라이딩

The screenshot displays an IDE with three Java files open: `MainApp.java`, `AbsUnit.java`, and `Marine.java`. The `MainApp.java` file contains a `main` method that creates a `Marine` attacker and defender, and a `Firebat` attacker. It calls `damage` on the attacker and prints the defender's HP. The `AbsUnit.java` file defines an abstract class `AbsUnit` that implements the `Unit` interface, with attributes `attack`, `hp`, and `hpMax`, and methods `damage`, `getAttack`, and `getHp`. The `Marine.java` file defines a class `Marine` that extends `AbsUnit` and overrides the `getAttack` method to return 10. The `Firebat.java` file defines a class `Firebat` that extends `AbsUnit` and overrides the `getAttack` method to return 15. The console output shows the results of the simulation: `공격!! hp:100 -> 90` and `공격!! hp:90 -> 75`.

```
1 package finaltest;
2
3 import finaltest.units.Firebat;
4 import finaltest.units.Marine;
5
6 public class MainApp {
7
8     public static void main(String[] args) {
9         // 공격자
10        Marine marineAttacker = new Marine();
11        Firebat firebatAttacker = new Firebat();
12
13        // 방어자
14        Marine defender = new Marine();
15
16        // 전투 수행!!
17        marineAttacker.damage(defender);
18        firebatAttacker.damage(defender);
19
20        // 전투 결과
21        System.out.println(defender.getHp());
22    }
23 }
24
```

```
1 public abstract class AbsUnit implements Unit {
2
3     private long attack = 1;
4     private long hp = 100;
5     private long hpMax = 100;
6
7     @Override
8     final public void damage(Unit target) {
9         long beforeHp = target.getHp();
10        long afterHp = target.getHp() - getAttack();
11        target.setHp(afterHp);
12        System.out.println("공격!! hp:" + beforeHp + " -> " + afterHp);
13    }
14
15     public long getAttack() {
16         return attack;
17     }
18 }
19
```

```
1 package finaltest.units;
2
3 public class Marine extends AbsUnit {
4
5     @Override
6     public long getAttack() {
7         return 10;
8     }
9 }
10
```

```
1 package finaltest.units;
2
3 public class Firebat extends AbsUnit {
4
5     @Override
6     public long getAttack() {
7         return 15;
8     }
9 }
10
```

Problems Javadoc Declaration Console Call Hierarchy

<terminated> MainApp (3) [Java Application] C:\Program Files\RedHat\java-11-openjdk-11.0.7-1\bin\javaw.exe (2020. 5. 18. 오전 7:09:00 - 오전 7:09:00)

공격!! hp:100 -> 90  
공격!! hp:90 -> 75  
75

## 메서드 오버라이딩

```
1 package finaltest;
2
3 import finaltest.units.Firebat;
4 import finaltest.units.Marine;
5
6 public class MainApp {
7
8     public static void main(String[] args) {
9         // 공격자
10        Marine marineAttacker = new Marine();
11        Firebat firebatAttacker = new Firebat();
12
13        // 방어자
14        Marine defender = new Marine();
15
16        // 전투 수행!!
17        marineAttacker.damage(defender);
18        firebatAttacker.damage(defender);
19
20        // 전투 결과
21        System.out.println(defender.getHp());
22    }
23 }
24
```

```
1 public abstract class AbsUnit implements Unit {
2
3     private long attack = 1;
4     private long hp = 100;
5     private long hpMax = 100;
6
7     @Override
8     final public void damage(Unit target) {
9         long beforeHp = target.getHp();
10        long afterHp = target.getHp() - getAttack();
11        target.setHp(afterHp);
12        System.out.println("공격!! hp:" + beforeHp + " -> " + afterHp);
13    }
14
15     public long getAttack() {
16         return attack;
17     }
18 }
19
```

```
1 package finaltest.units;
2
3 public class Marine extends AbsUnit {
4
5     @Override
6     public long getAttack() {
7         return super.getAttack() + 1;
8     }
9 }
10
```

```
1 package finaltest.units;
2
3 public class Firebat extends AbsUnit {
4
5     @Override
6     public long getAttack() {
7         return super.getAttack() + 2;
8     }
9 }
10
```

Problems Javadoc Declaration Console Call Hierarchy

<terminated> MainApp (3) [Java Application] C:\Program Files\RedHat\java-11-openjdk-11.0.7-1\bin\javaw.exe (2020. 5. 18. 오전 7:08:09 - 오전 7:08:10)

공격!! hp:100 -> 98  
공격!! hp:98 -> 95  
95

A faded background image of a man with glasses and a beard, wearing a white t-shirt. The t-shirt has a mathematical formula  $x^n + y^n = z^n$  printed on it. The text "3주차 - Java 빌드 & Spring 프레임워크" is overlaid on the image.

# 3주차 - Java 빌드 & Spring 프레임워크

# Appendix

- Java 예약어  
abstract, assert, boolean, break, byte, case, catch, char, class, const, continue, default, do, double, else, enum, extends, final, finally, float, for, if, goto, implements, import, instanceof, int, interface, long, native, new, package, private, protected, public, return, short, static, strictfp, super, switch, synchronized, this, throw, throws, transient, try, void, volatile, while
- Android 에서 jvm 사용?  
안함. Dalvik 이라는 전용 가상머신