Java 스터디

서버팀 2파트 백승찬

1주차 – Java 소개

Java의 태동

- SUN Microsystems사 에서 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 Microsystems사 에서 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 "

James Gosling (1955.5.19~)

Write Once, Run Anywhere!

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

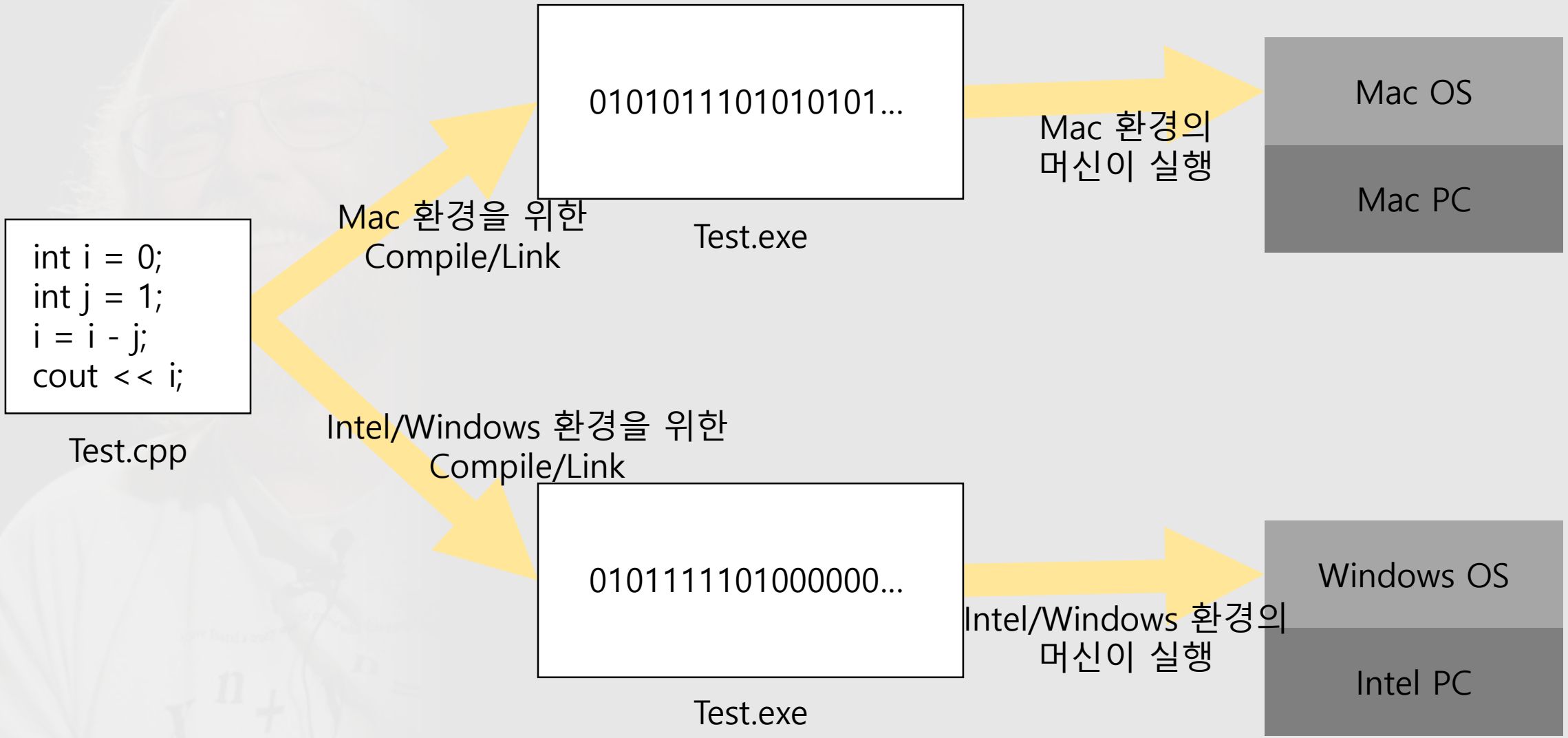
Java의 WORA

- 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.

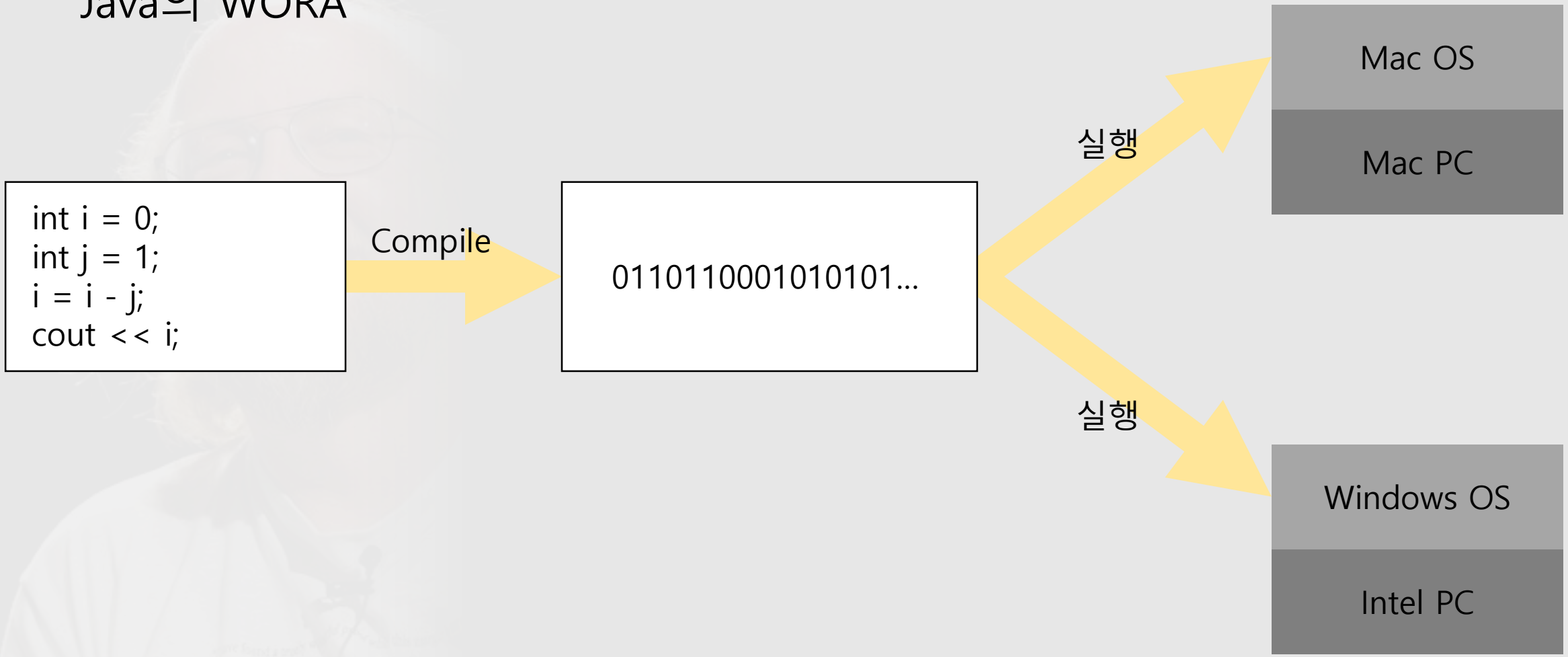
- Byte Code
  - Javac (자바 컴파일러) 가 Java 소스코드를 컴파일한 결과로 추출된 기계어
  - JVM이 실행할 수 있는 바이너리 코드 (타겟 머신의 CPU가 실행하지 않음)
  - JVM이 인터프리터 방식으로 Byte Code를 해석해 타겟 머신에서 실행

- 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


int i = 0;
int j = 1;
i = i - j;
cout << i;
Test.cpp
Mac 환경을 위한
Compile/Link
0101011101010101...
Test.exe
Mac 환경의
머신이 실행
Mac OS
Mac PC
Intel/Windows 환경을 위한
Compile/Link
0101111101000000...
Test.exe
Intel/Windows 환경의
머신이 실행
Windows OS
Intel PC

Java의 WORA


int i = 0;
int j = 1;
i = i - j;
cout << i;
Compile
0110110001010101...
실행
실행
Mac OS
Mac PC
Windows OS
Intel PC

Java의 WORA

```
int i = 0;
int j = 1;
i = i - j;
System.out.println(i);
```

Test.java

JVM을 위한 Compile

011011000101010 1...

Test.class

JVM이 해석해 실행

JVM

Mac OS

Mac PC

JVM이 해석해 실행

JVM

Windows OS

Intel PC

"Java는 Byte Code로 컴파일되며, JVM에 의해 실행된다"

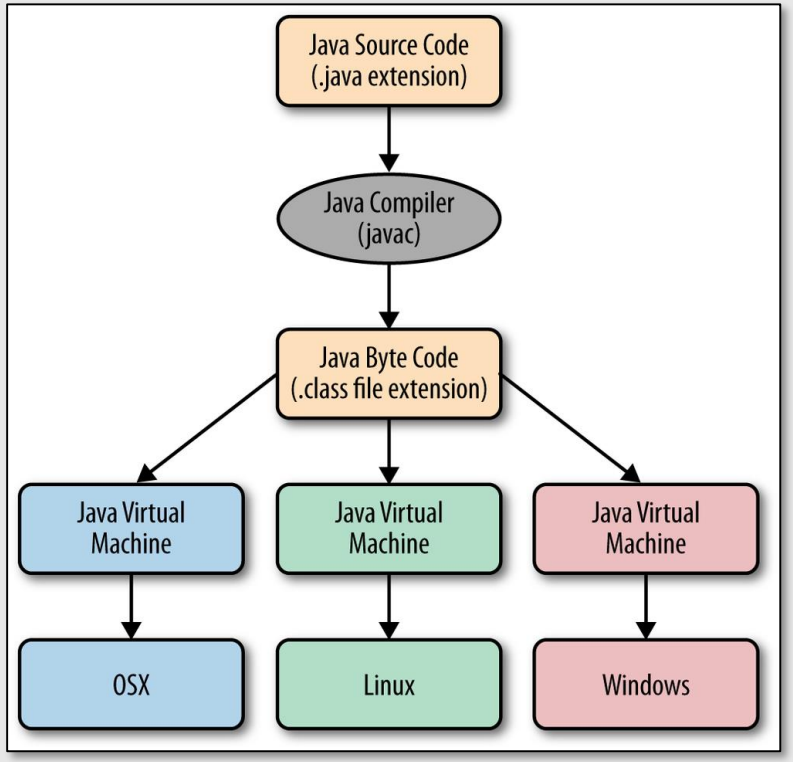
Java의 WORA

- 실행환경의 차이
  - [C/C++]

    - 컴파일/링크 결과 타겟 머신에서 실행가능한 실행 파일을 생성

  - [Java]

    - 컴파일 결과 JVM에서 실행가능한 Byte Code 형태의 클래스 파일을 생성

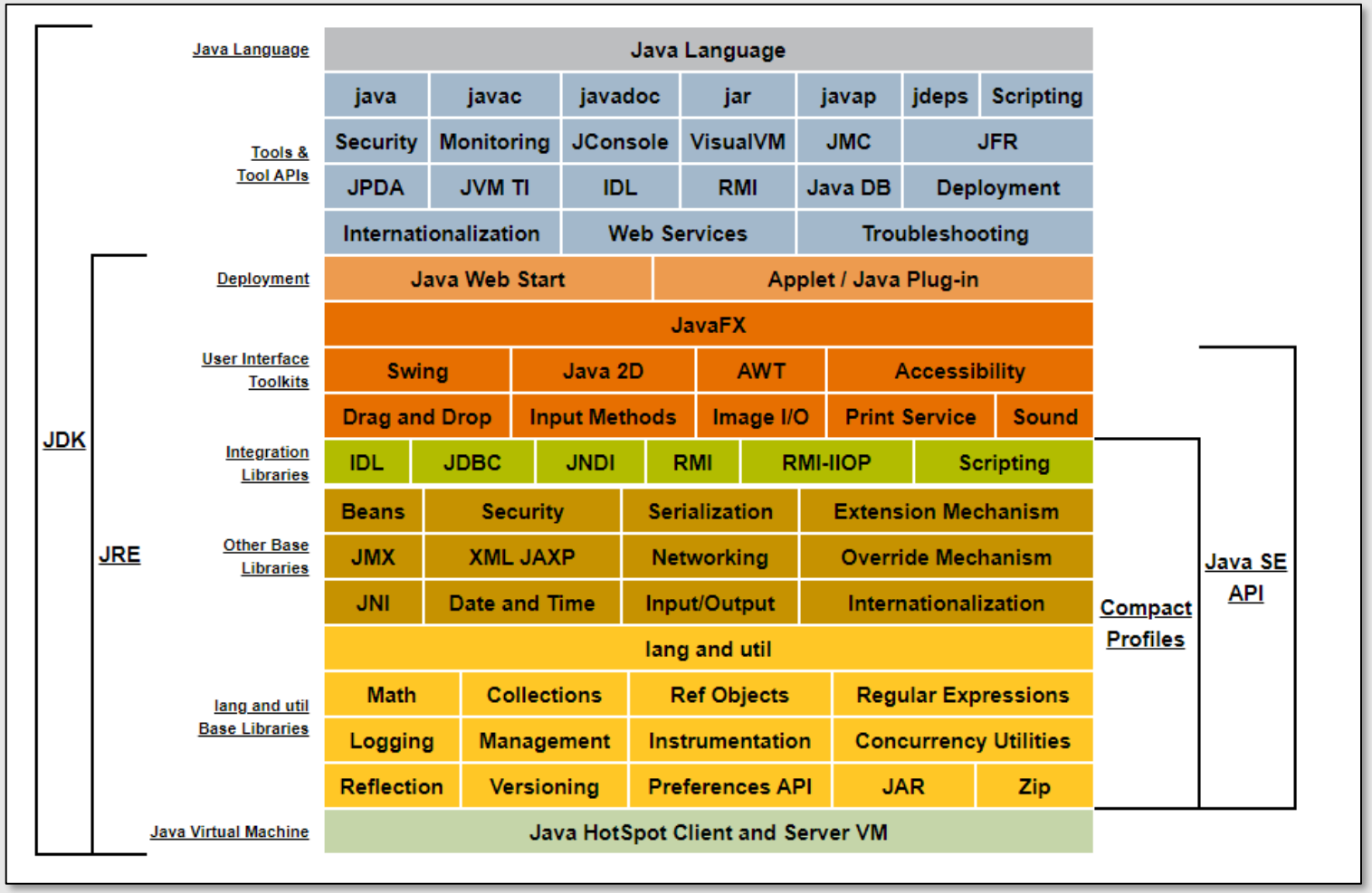


http://www.developingthefuture.net/compilation-process-and-jit-compiler/

Java 개발 환경 구축

- JDK (Java Development Kit)
  : 개발 및 실행 도구

- JRE (Java Runtime Environment)
  : 실행 도구
  (대부분 컴퓨터에 설치되어있음)



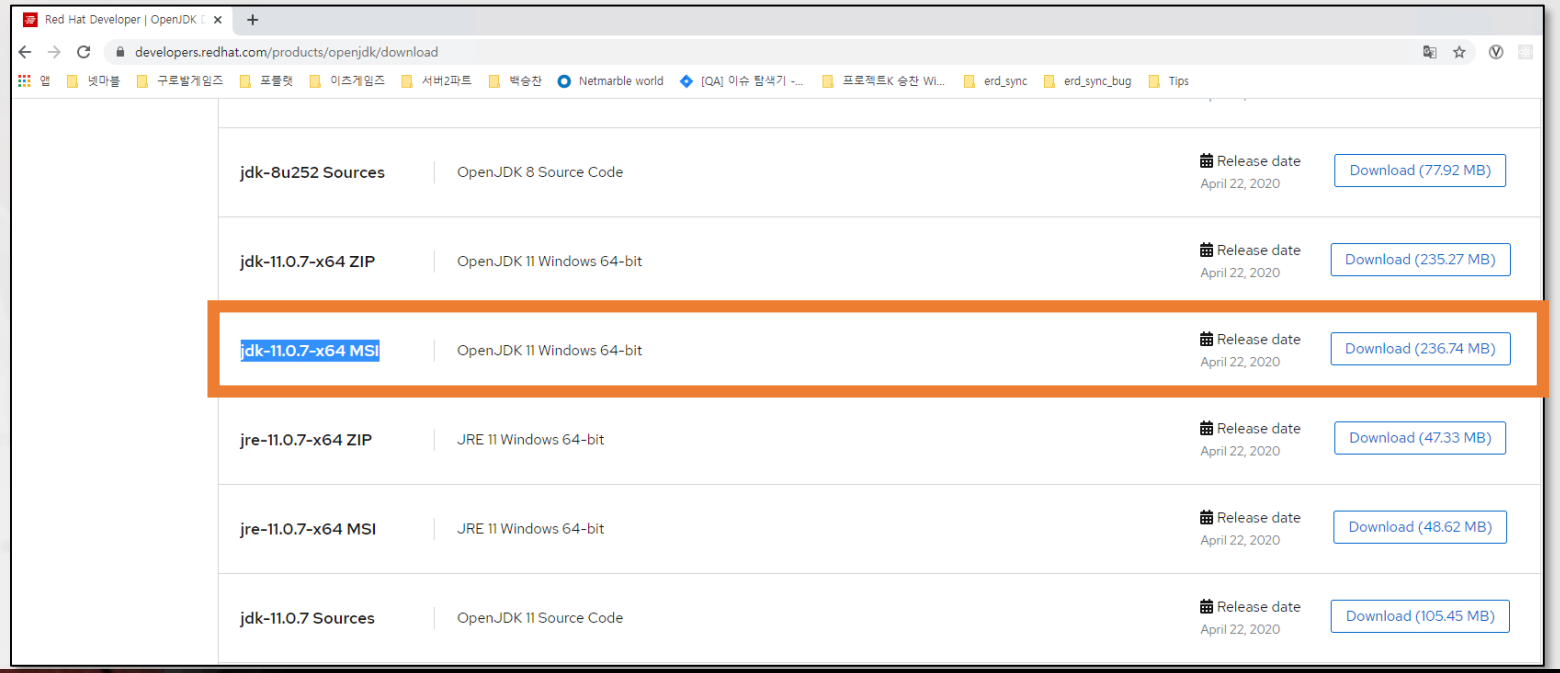
Oracle jdk 8 구조

Java 개발 환경 구축

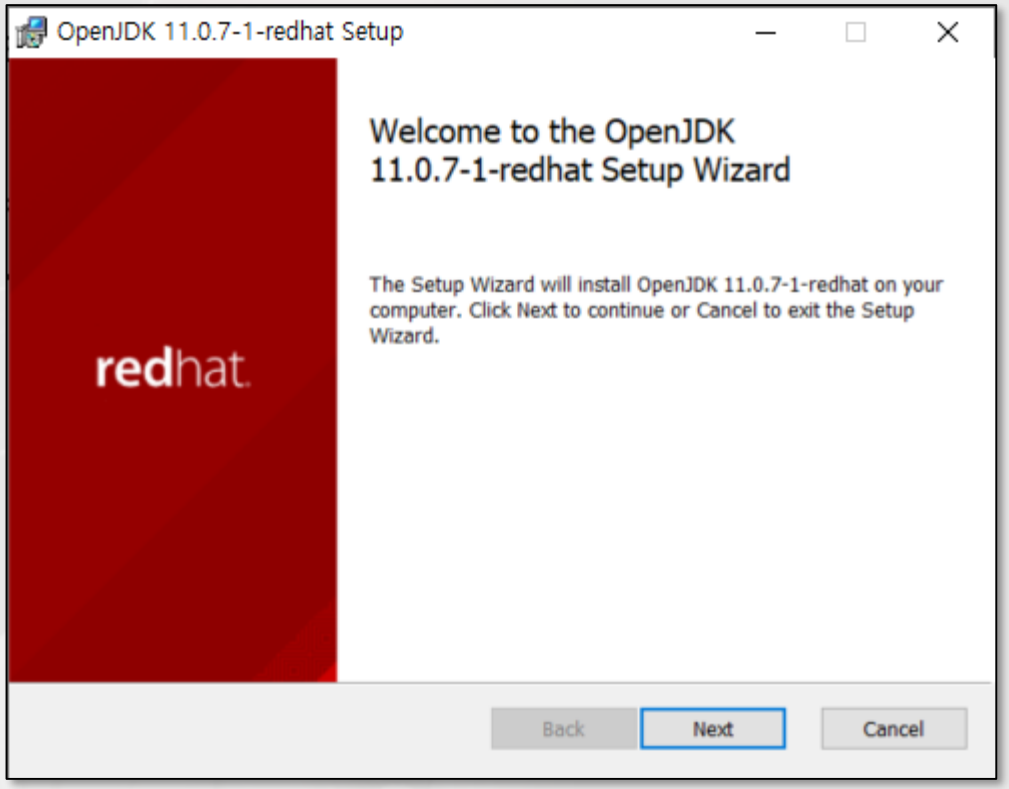
- JDK 설치
  - Red hat Openjdk11 설치

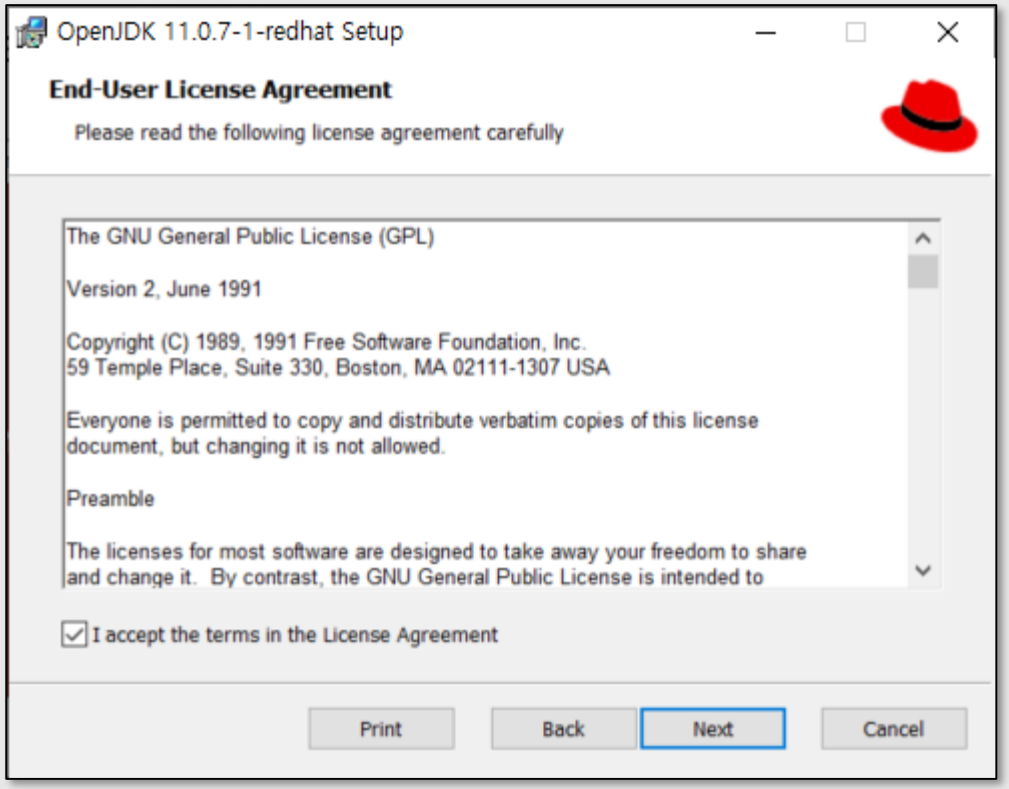
    - 구글 드라이브로 다운받기
    https://drive.google.com/open?id=1LMmIWyGKIG3GuYvxXFfA6ukRcNeEJNIy

    - Red hat 공식 홈페이지에서 다운 받기
    https://developers.redhat.com/products/openjdk/download



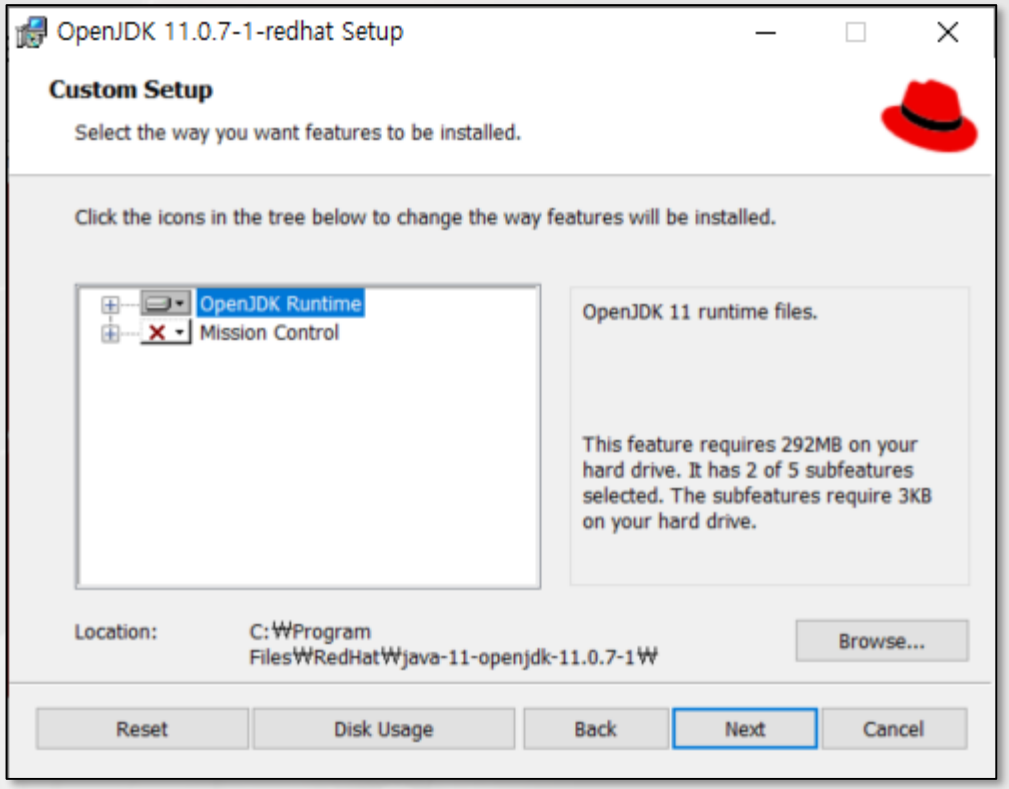
Java 개발 환경 구축 (JDK 설치 & 세팅)

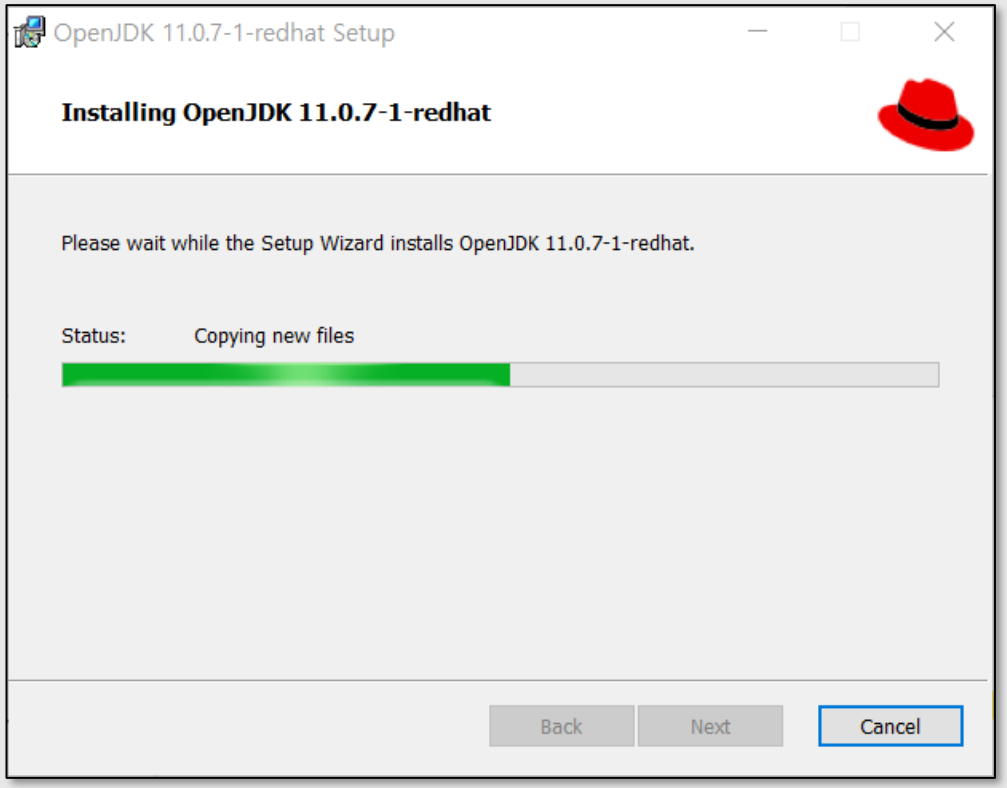




다운 받은 msi 설치파일을 열고 라이선스 동의 및 쭈욱 Next~

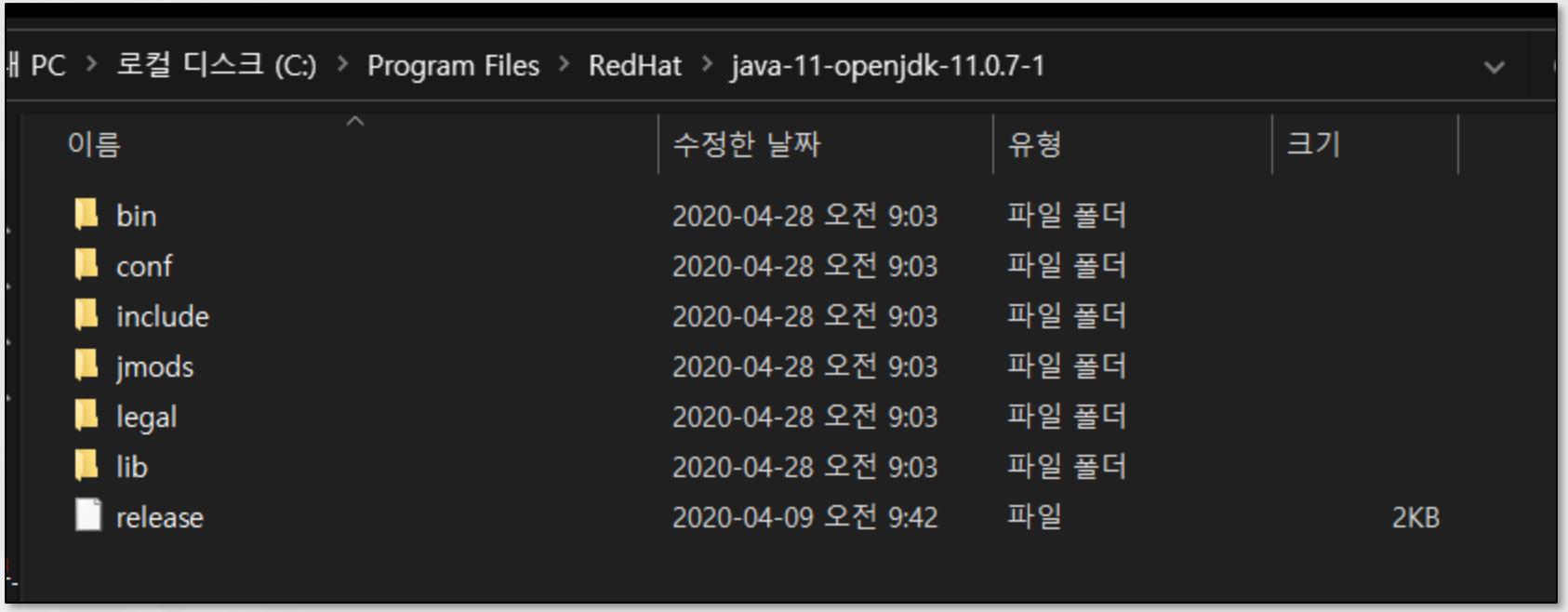
Java 개발 환경 구축 (JDK 설치 & 세팅)





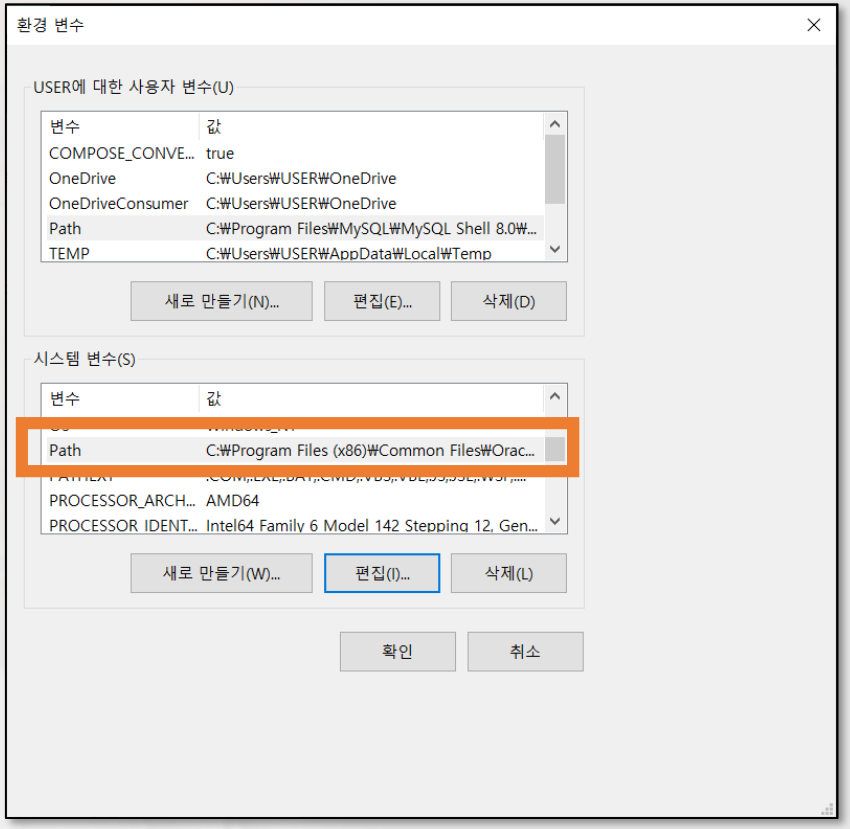
설정 다시 할 수 있으니 쭉쭉 설치

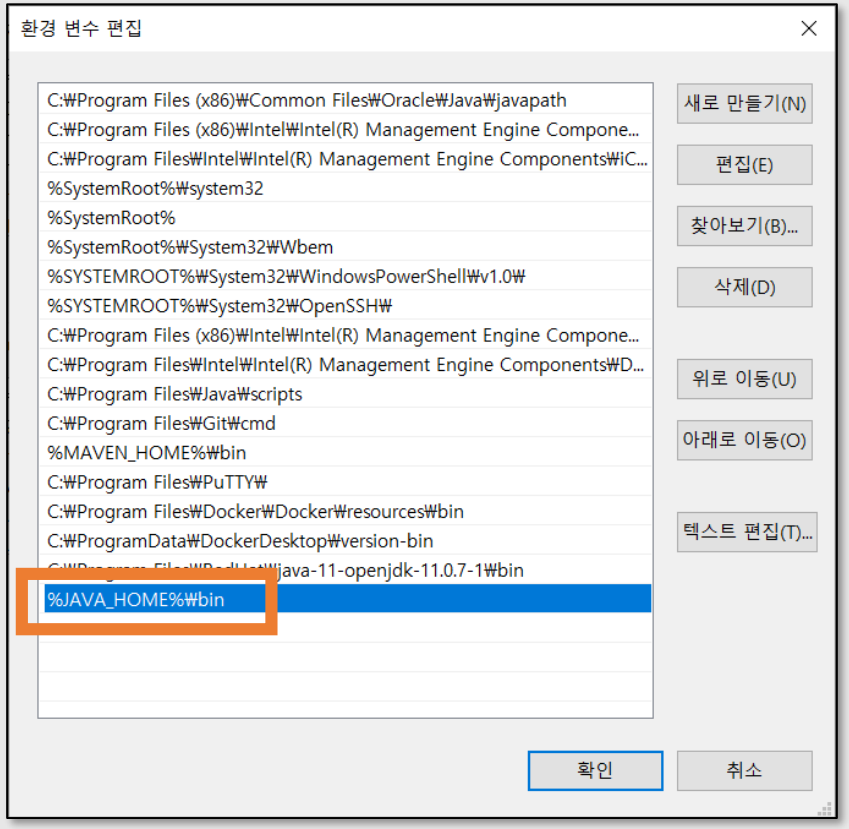
Java 개발 환경 구축 (JDK 설치 & 세팅)



RedHat 디렉터리 하위에 파일 생성됨

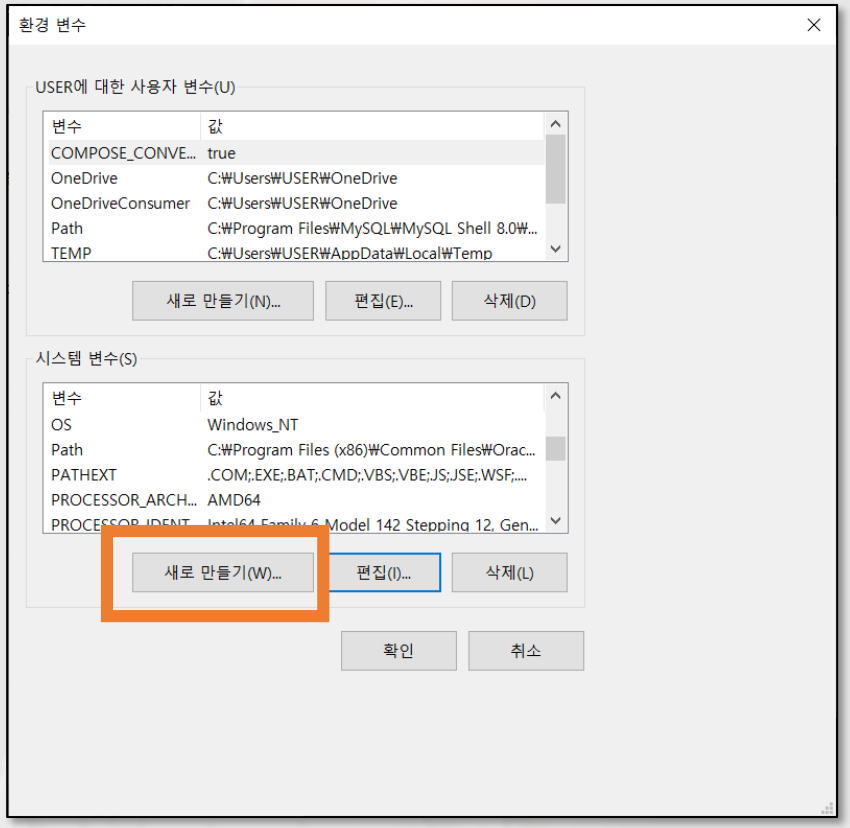
Java 개발 환경 구축 (JDK 설치 & 세팅)

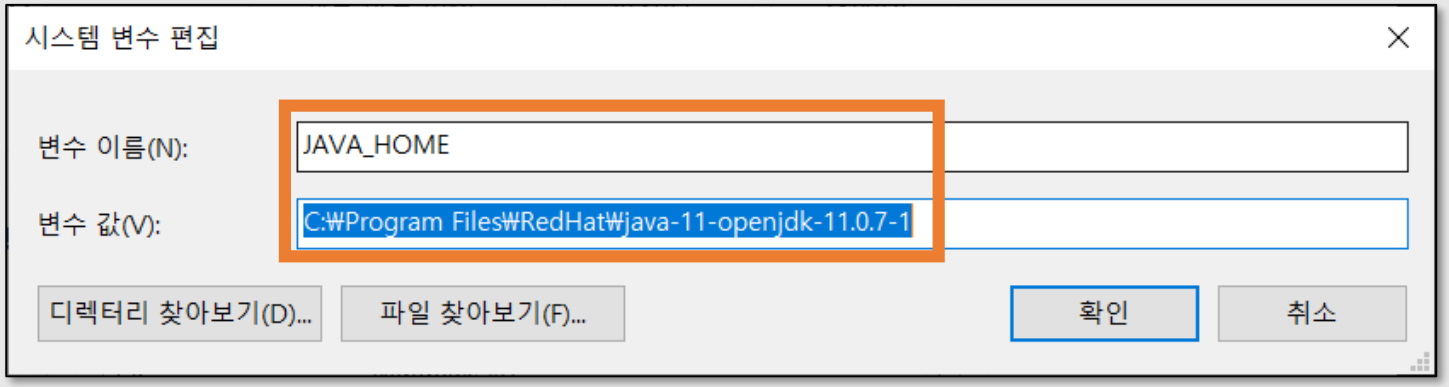




시스템변수 `Path` 에 `JAVA_HOME` 이라는 이름의 항목을 추가해줍니다.
복붙 → %JAVA_HOME%₩bin

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)

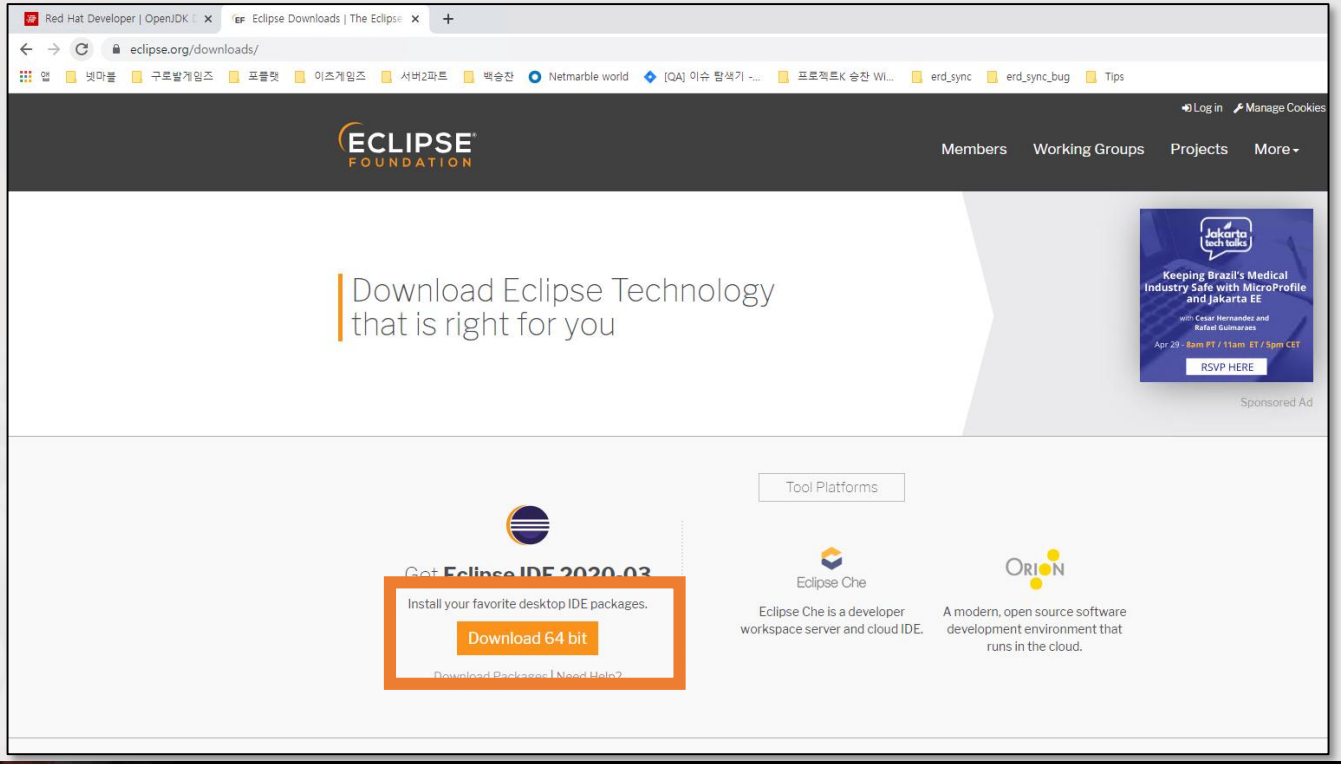
```
C:₩Users₩USER>java -version
openjdk version "11.0.7" 2020-04-14 LTS
OpenJDK Runtime Environment 18.9 (build 11.0.7+10-LTS)
OpenJDK 64-Bit Server VM 18.9 (build 11.0.7+10-LTS, mixed mode)

C:₩Users₩USER>
```

java –version
정상 설치 완료

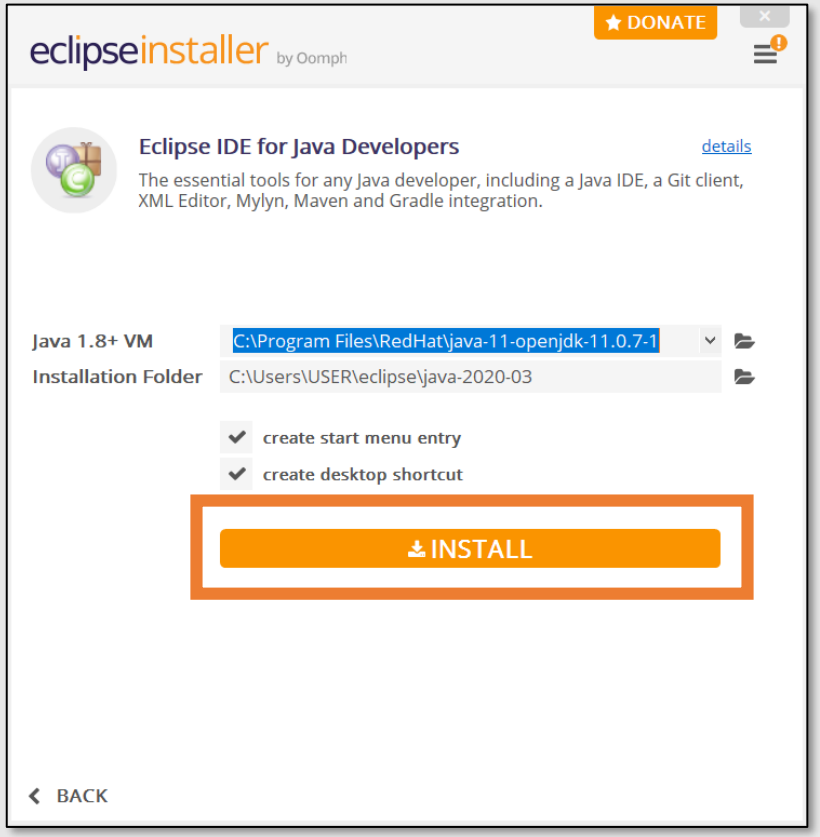
Java 개발 환경 구축

- IDE 설치
  - Eclipse 설치

    - 공식 홈페이지에서 다운 받기
    https://www.eclipse.org/downloads/

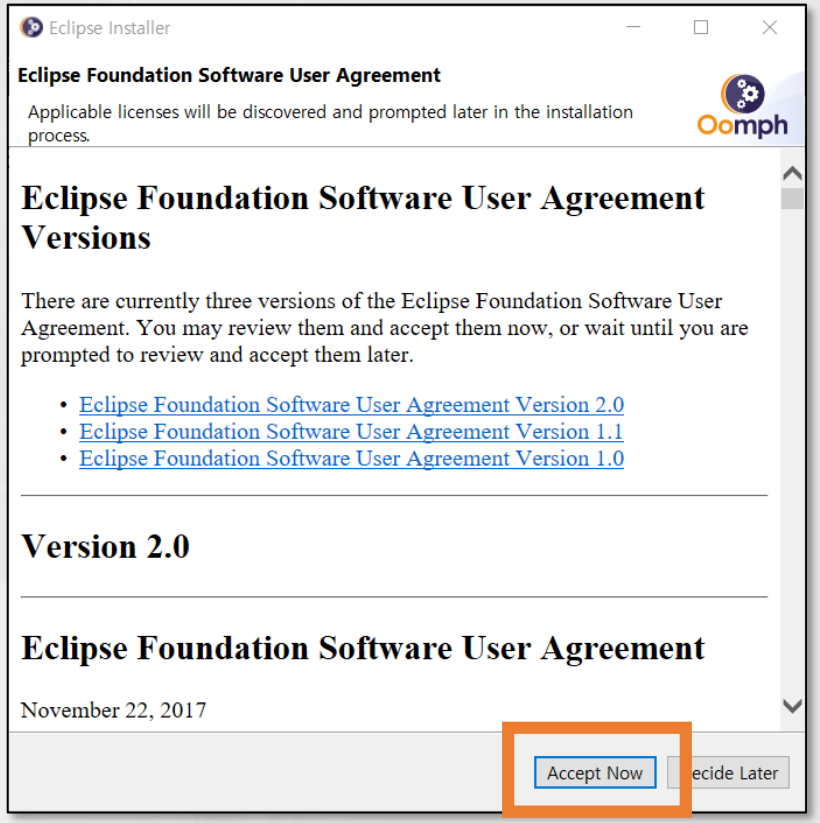


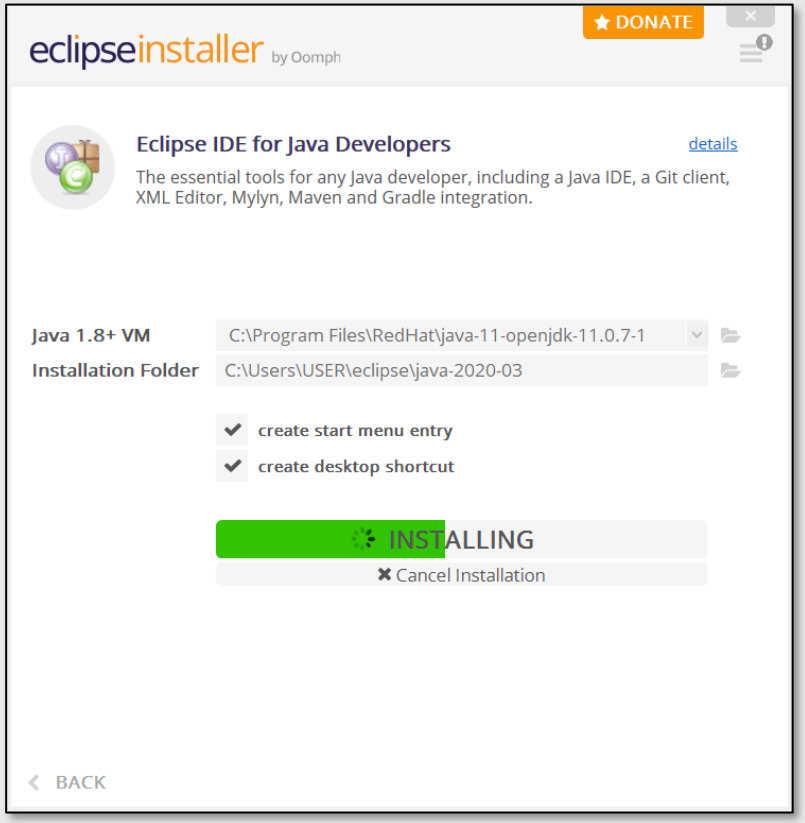
Java 개발 환경 구축 (Eclipse IDE 설치)



설치 파일을 실행해 Eclipse IDE for Java Developers 설치

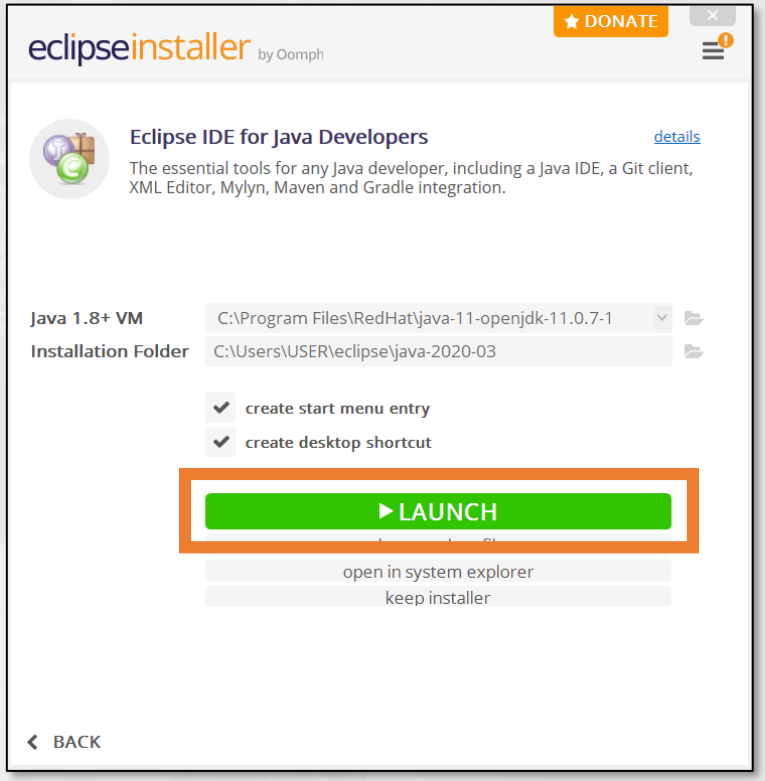
Java 개발 환경 구축 (Eclipse IDE 설치)

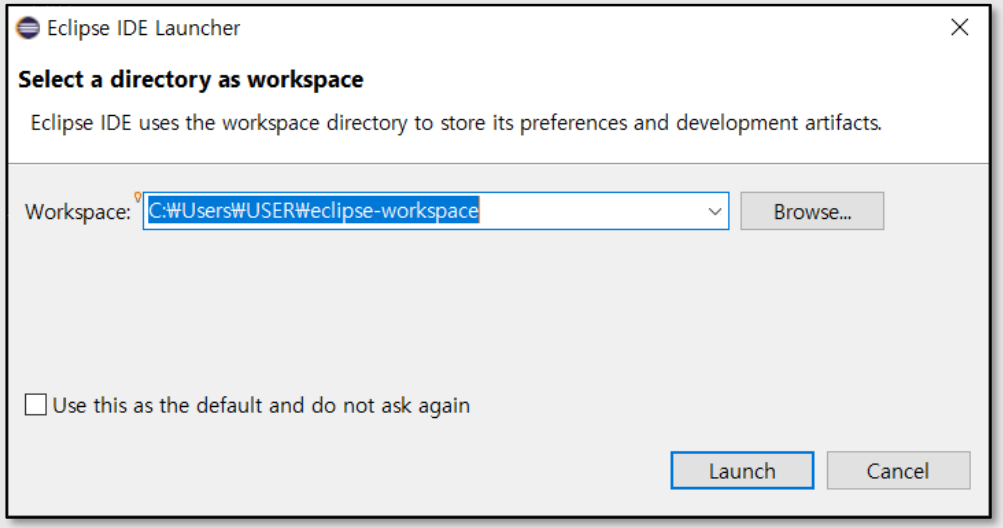




라이선스 동의 후 설치 진행

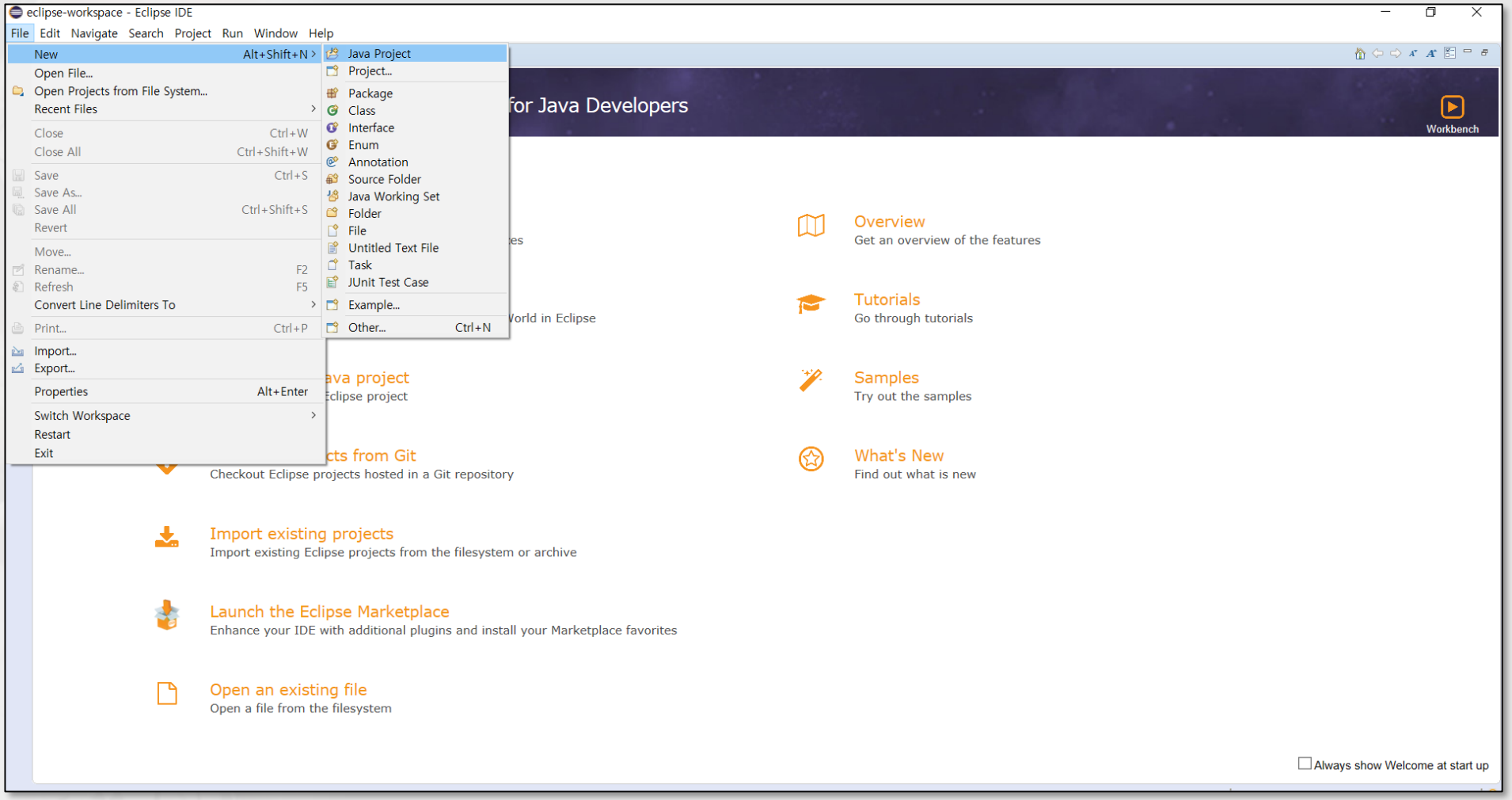
Java 개발 환경 구축 (Eclipse IDE 설치)





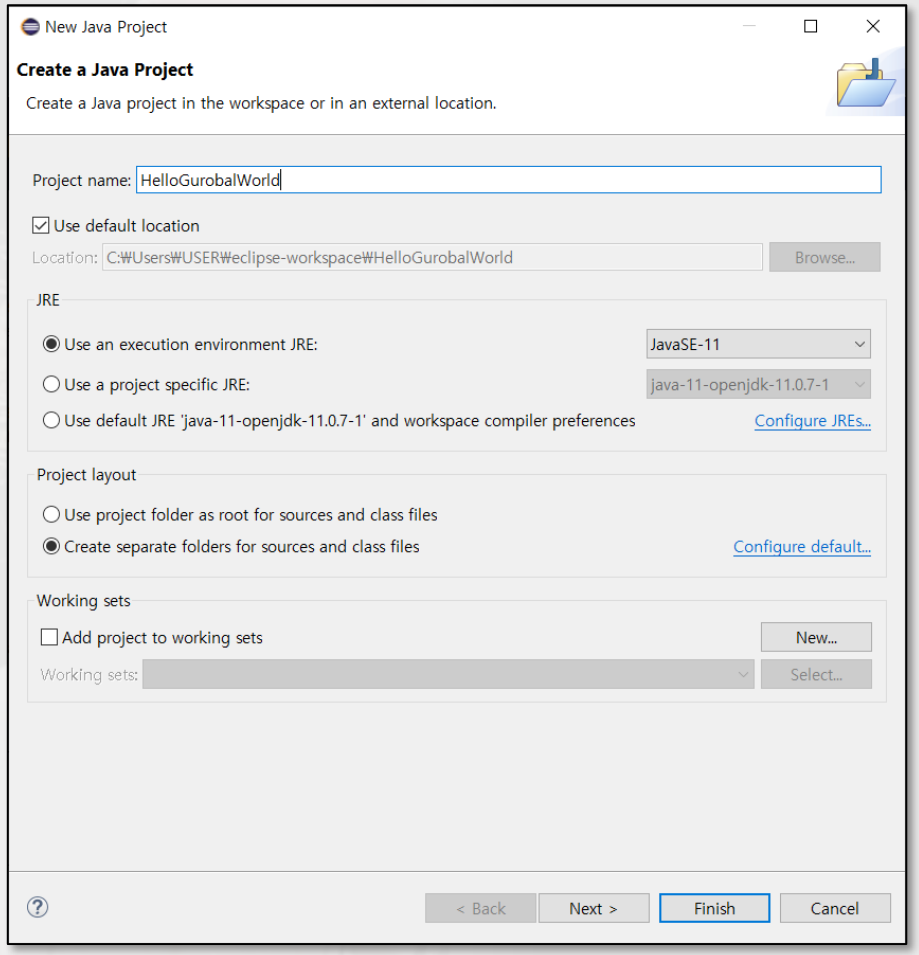
설치 완료후 workspace 경로 지정

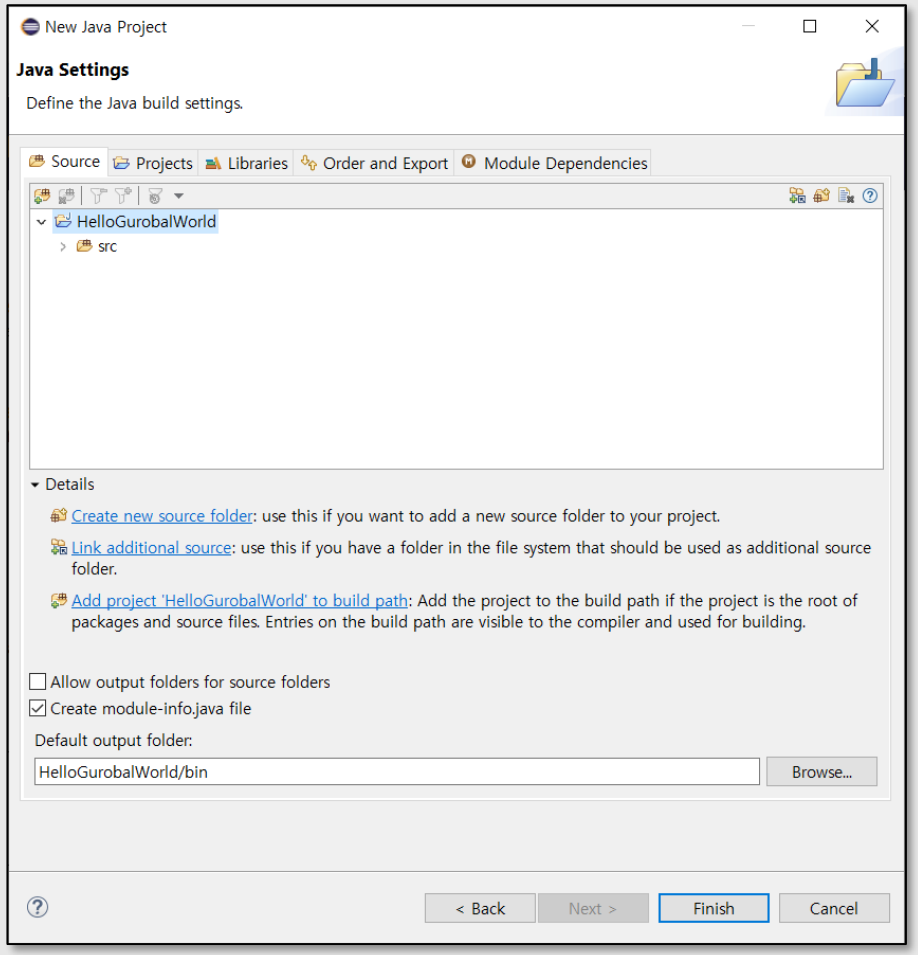
Java 개발 환경 구축 (Eclipse IDE 설치)



File – New – Java Project 선택해 hello world 작성해보기

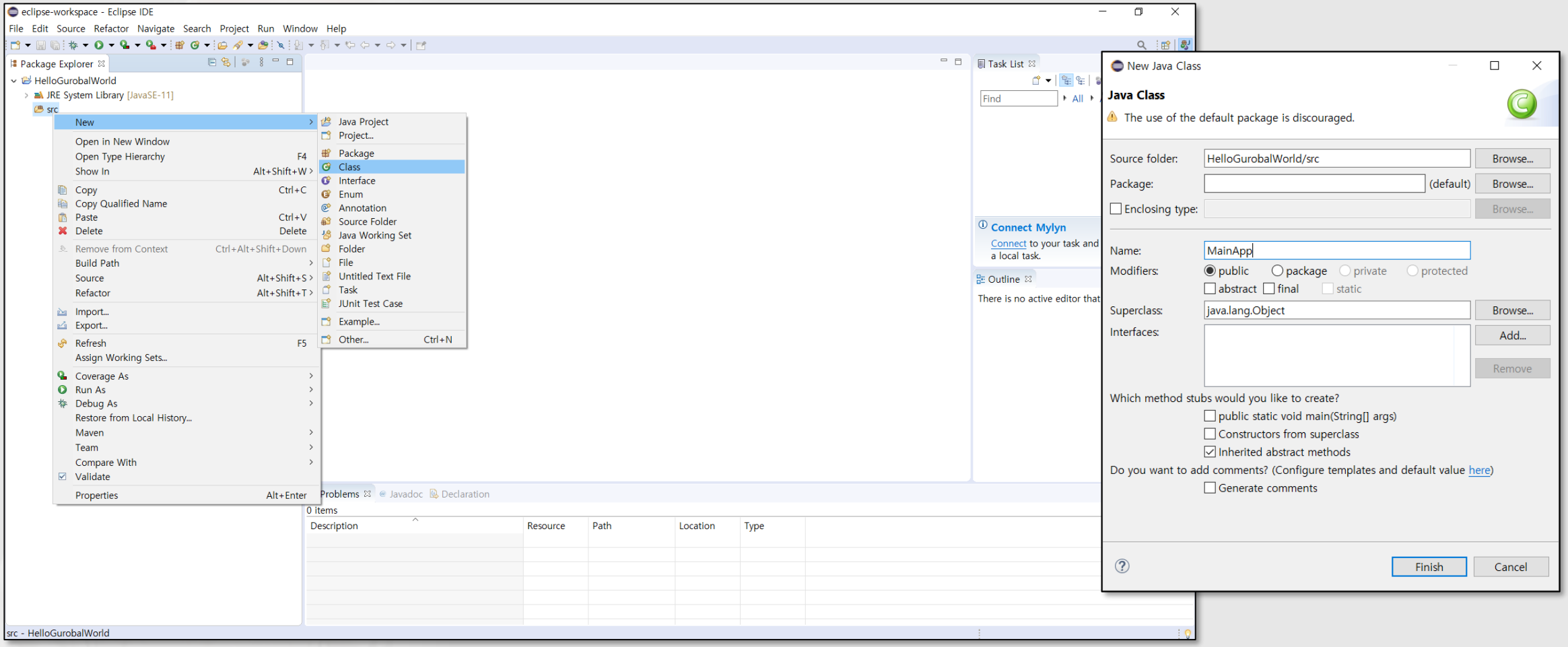
Java 개발 환경 구축 (Eclipse IDE 설치)





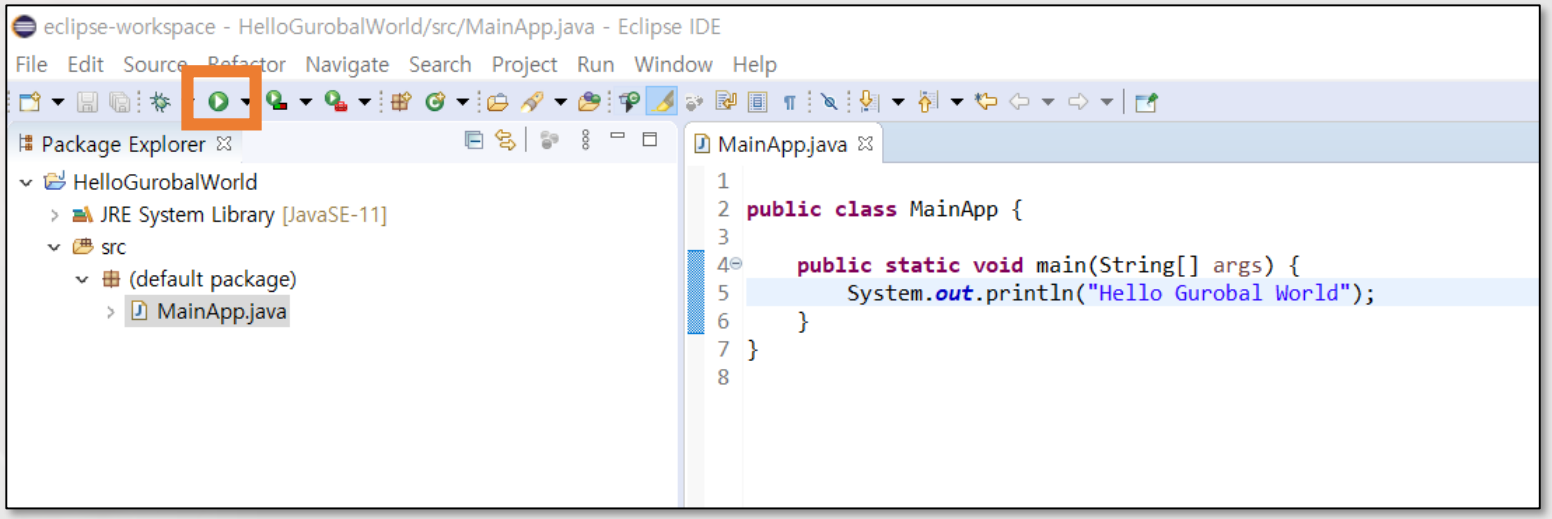
HelloGurobalWorld 프로젝트 생성

Java 개발 환경 구축 (Eclipse IDE 설치)



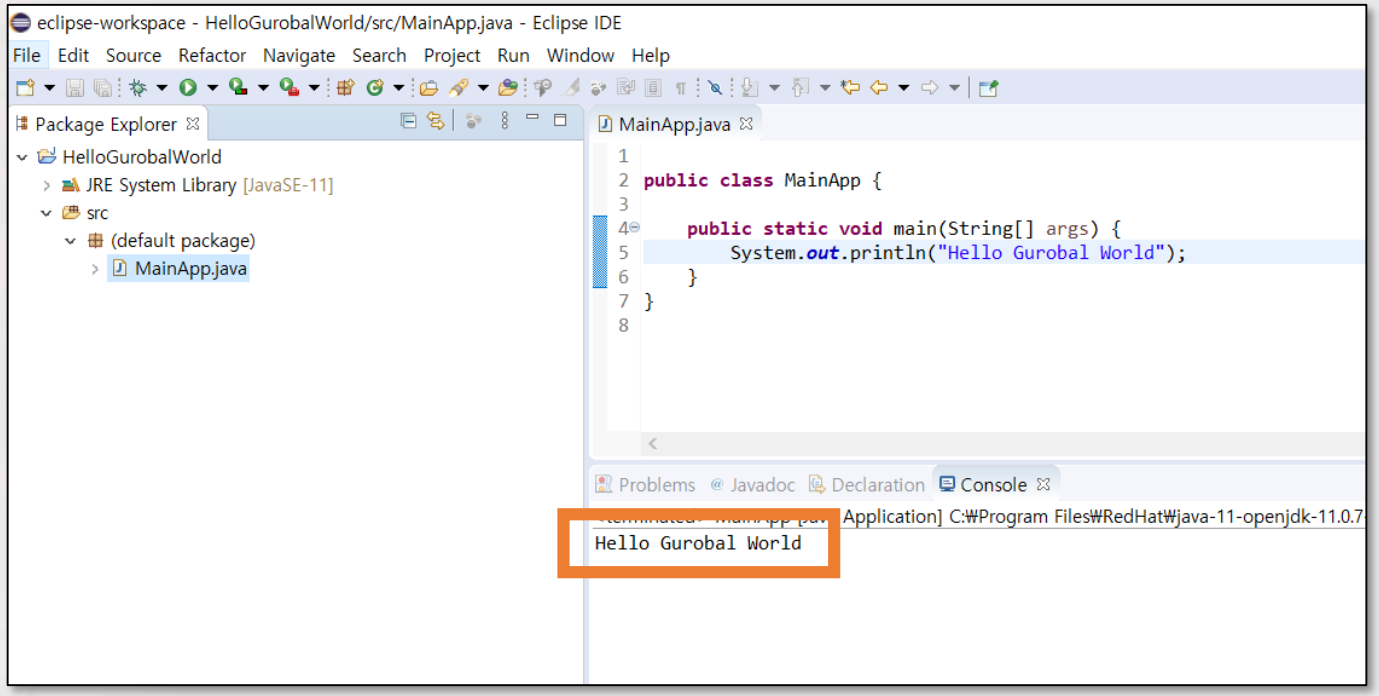
HelloGurobalWorld/src 디렉터리 우클릭 – New – Class
클래스 생성

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 Annotaion

- 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, Annotaion 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

1. 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)

2. 멀티스레드

   - OS의 도움 없이 java 프로그램에서 멀티스레드를 동시에 실행할 수 있음

3. 플랫폼 독립적

   - 머신에 종속되지 않는 독립적인 Byte Code로 컴파일해 머신의 JVM이 실행

Java의 특징

4. 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.

```
public class A {
    ...
}
class B {
    ...
}
class C {
    ...
    class D{...}
}
```

A.java

Compile

101100... ByteCode

A.class

101110... ByteCode

C.class

100110... ByteCode

B.class

101111... ByteCode

C$D.class

Java의 특징

5. 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)

6. 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

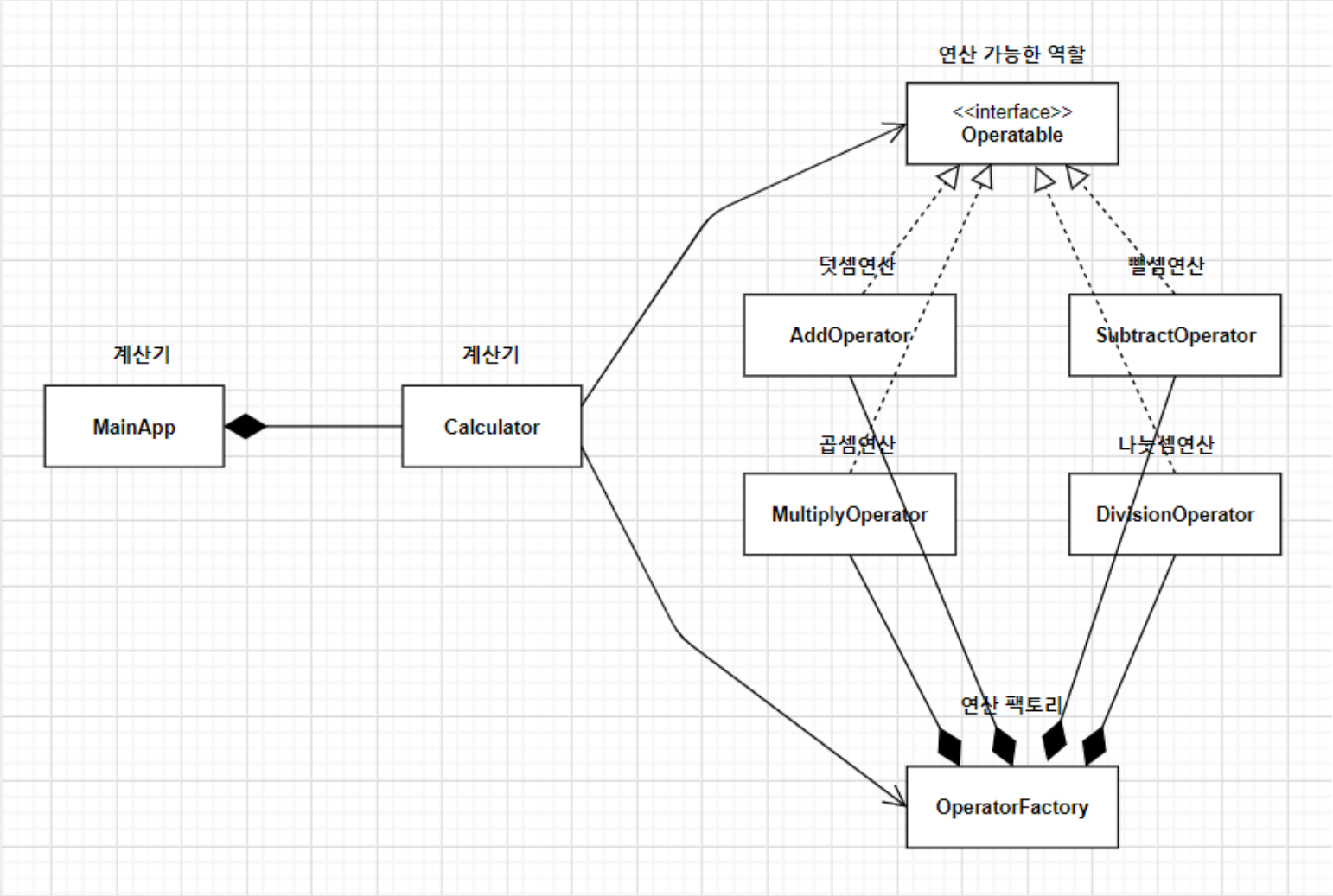
IDE 없이 간단히 테스트

https://www.w3schools.com/java/java_data_types.asp

Java 초단단 예시

```java
/**
 * 계산기 클래스
 */
public class Calculator { // 클래스 네이밍 켄벤션은 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



https://github.com/mokdonjr/studyJava

2주차 – Java 문법

클래스 선언

접근지정자 class 클래스이름 {...}

```
package classtest.model;

public class Person {

}
```

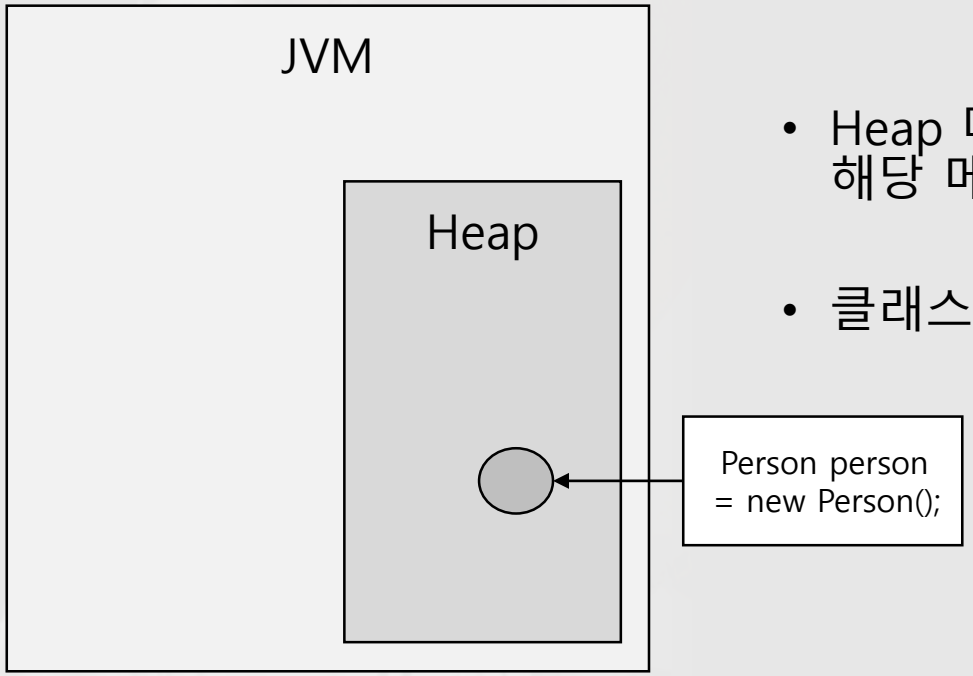
클래스 인스턴스 생성

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

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.)

생성자

public 클래스이름(인자타입 인자, ...) {...}

```
package classtest.model;

public class Person {

    public Person() {

    }

}
```

클래스 접근지정자

다른 클래스에서 이 클래스를 import 할 수 있는지 허용 여부

공개

폐쇄

- **public**

  - 외부 클래스에서 import 가능

- **없음** (디폴트 접근지정자)

  - 동일한 패키지내 모든 클래스가 import 가능

클래스 접근지정자

Package Ex... | Project Expl...

- Calculator [studyJava master]
- Example [studyJava master]
  - JRE System Library [JavaSE-11]
  - src
    - calltest
    - classtest
      - MainApp.java
    - classtest.model
      - Person.java
    - test

MainApp.java

```java
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
```

Person.java

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

Package Ex... | Project Expl...

- Calculator [studyJava master]
- Example [studyJava master]
  - JRE System Library [JavaSE-11]
  - src
    - calltest
    - classtest
      - MainApp.java
    - classtest.model
      - Person.java
    - test

MainApp.java

```java
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
```

Person.java

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

클래스 접근지정자

Package Ex... | Project Expl...

- Calculator [studyJava master]
- Example [studyJava master]
  - JRE System Library [JavaSE-11]
  - src
    - calltest
    - classtest
      - MainApp.java
    - classtest.model
      - Person.java
    - test

MainApp.java

```java
package classtest;

import classtest.model.Person;

public class MainApp {

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

Person.java

```java
package classtest.model;

public class Person {

}
```

Package Ex... | Project Expl...

- Calculator [studyJava master]
- Example [studyJava master]
  - JRE System Library [JavaSE-11]
  - src
    - calltest
    - classtest
      - MainApp.java
    - classtest.model
      - Person.java
    - test

MainApp.java

```java
package classtest;

import classtest.model.Person;

public class MainApp {

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

Person.java

```java
package classtest.model;

class Person {

}
```

" public 없는 class는 테스트나 지역적으로만 쓰는 용도니 참조하려고 하지 말아야지 "

멤버변수

접근지정자 타입 변수이름[= 초기값];

Package Ex... | Project Expl...

- Calculator [studyJava master]
- Example [studyJava master]
  - JRE System Library [JavaSE-11]
  - src
    - calltest
    - classtest
      - MainApp.java
    - classtest.model
      - Person.java
    - test

MainApp.java

```java
package classtest;

import classtest.model.Person;

public class MainApp {

    public static void main(String[] args) {
        Person person = new Person();
        person.name = "승찬";
        System.out.println(person.name);
    }
}
```

Person.java

```java
package classtest.model;

public class Person {
    public String name;
    public int age;
}
```

멤버함수

접근지정자 리턴타입 메서드이름(인자타입 인자) {...}

MainApp.java

```java
package classtest;

import classtest.model.Person;

public class MainApp {

    public static void main(String[] args) {
        Person vampire = new Person();
        vampire.setName("뱀파이어");
        vampire.setAge(120);

        Person victim = new Person();
        victim.setName("희생자");
        victim.setAge(20);

        // 흡혈
        vampire.swapAge(victim);
        System.out.println(vampire.getAge());
        System.out.println(victim.getAge());
    }
}
```

Person.java

```java
package classtest.model;

public class Person {
    private String name;
    private int age;

    public void swapAge(Person target) {
        int tempAge = this.age;
        this.age = target.getAge();
        target.setAge(tempAge);
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
```

지역변수

멤버 메서드의 클래스 인스턴스 레퍼런스

Console

```
<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

공개

폐쇄

- **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 내무에서만 호출 가능

멤버 접근지정자

다른 클래스에서 이 클래스의 멤버에 접근 할 수 있는지 허용 여부

	동일한 클래스 내부	동일한 패키지 내부 모든 클래스	패키지 무관 상속 받은 모든 자식 클래스	패키지 무관 모든 클래스 외부
public	O	O	O	O
Protected	O	O	O	X
없음 (디폴트)	O	O	X	X
Private	O	X	X	X

멤버 접근지정자

Package Ex... | Project Expl...

- Calculator [studyJava master]
- Example [studyJava master]
  - JRE System Library [JavaSE-11]
  - src
    - calltest
    - classtest
      - MainApp.java
    - classtest.model
      - Person.java
    - test

MainApp.java

```java
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 }
```

Person.java

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

Package Ex... | Project Expl...

- Calculator [studyJava master]
- Example [studyJava master]
  - JRE System Library [JavaSE-11]
  - src
    - calltest
    - classtest
      - MainApp.java
    - classtest.model
      - Person.java
    - test

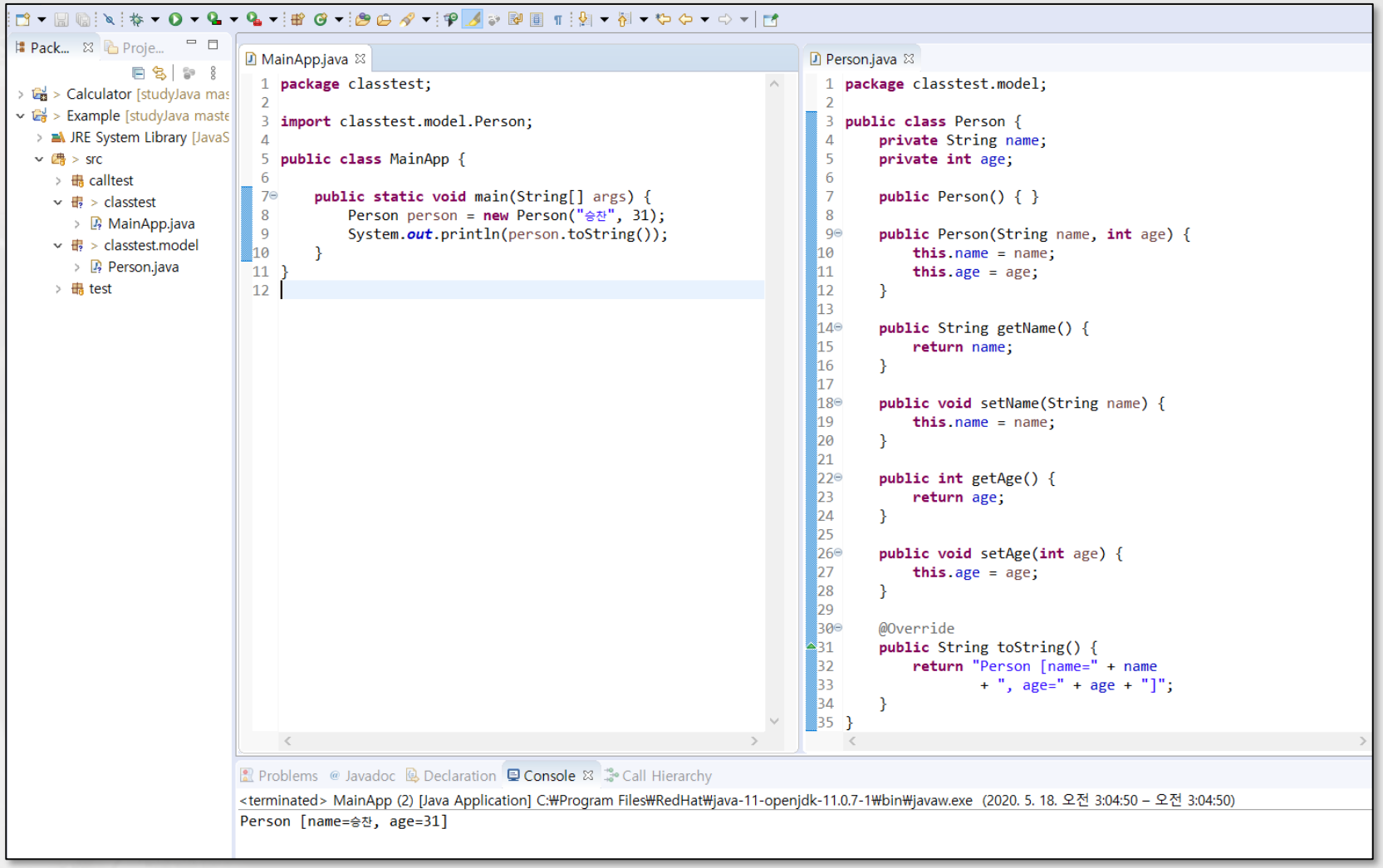
MainApp.java

```java
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 }
```

Person.java

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

결론



" 캡슐화 원칙을 지켜 클래스를 정의 하고 공개된 범위의 멤버를 호출하자 "
멤버변수는 왠만하면 private 으로 감추고 공개된 getter/setter 작성

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

Pack... Proje...

- Calculator [studyJava mas
- Example [studyJava maste
  - JRE System Library [JavaS
  - src
    - calltest
    - classtest
      - MainApp.java
    - classtest.model
      - Person.java
    - test

MainApp.java

```java
package classtest;

import classtest.model.Person;

public class MainApp {

    public static void main(String[] args) {
        Person son = new Person();
        son.setName("손흥민");
        son.setAge(28);
        System.out.println(Math.max(0, son.getAge()));

        Person trump = new Person();
        trump.setName("트럼프");
        trump.setAge(70);
        System.out.println(Math.max(0, trump.getAge()));
    }
}
```

Person.java

```java
package classtest.model;

public class Person {
    private String name;
    private int age;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

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 4:24:42 – 오전 4:24:42)

```
28
70
```

static 멤버

Pack... Proje...

- Calculator [studyJava mas
- Example [studyJava maste
  - JRE System Library [JavaS
  - src
    - calltest
    - classtest
      - MainApp.java
    - classtest.model
      - Person.java
    - classtest.util
      - FileUtil.java
    - test
    - test.txt

MainApp.java | StringTokenizer.class

```java
import java.util.StringTokenizer;

import classtest.model.Person;
import classtest.util.FileUtil;

public class MainApp {

    static List<String> footballPlayerList = new ArrayList<>();

    static {
        String content = FileUtil.readFile("C:\\dev-repository\\studyJava\\Exa
        StringTokenizer tokenizer = new StringTokenizer(content, ", ");
        while (tokenizer.hasMoreTokens()) {
            footballPlayerList.add(tokenizer.nextToken());
        }
    }

    public static void main(String[] args) {

        Person person1 = new Person();
        person1.setName("손흥민");
        person1.setAge(28);
        System.out.println(footballPlayerList.contains(person1.getName()));

        Person person2 = new Person();
        person2.setName("이상화");
        person2.setAge(30);
        System.out.println(footballPlayerList.contains(person2.getName()));
    }
}
```

Person.java | FileUtil.java | test.txt

```java
package classtest.util;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;

public class FileUtil {

    public static String readFile(String path) {
        StringBuilder sb = new StringBuilder();
        try{
            File file = new File(path);
            FileReader filereader = new FileReader(file);
            BufferedReader bufReader = new BufferedReader(filereader
            String line = "";
            while((line = bufReader.readLine()) != null){
                sb.append(line);
            }
            bufReader.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return sb.toString();
    }
}
```

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
```

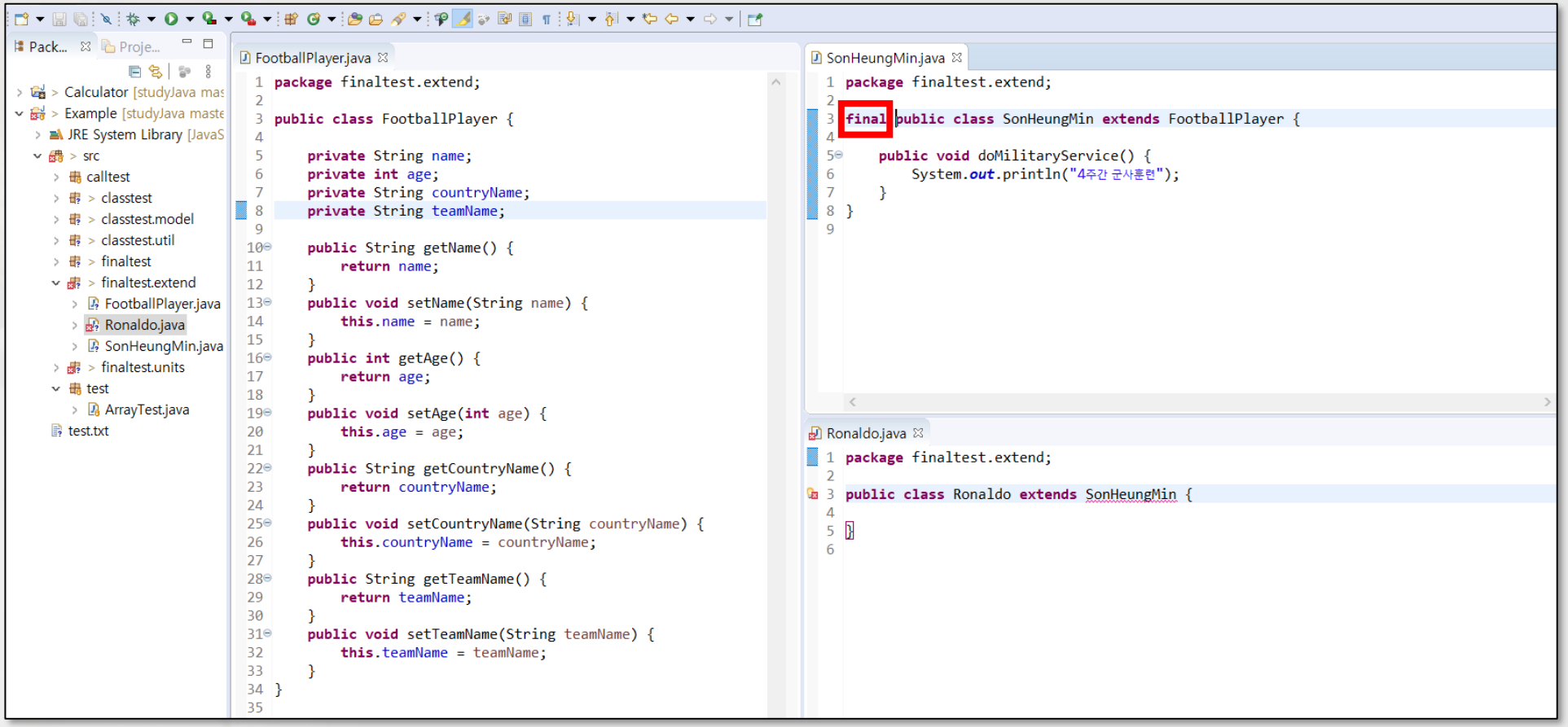
" 주로, 전역적으로 쓰일 기능 (Utility) 필요시 "

또는 프로그램 시작시 데이터 로드 필요시 static 키워드를 붙인다.

final 클래스

final 접근지정자 class 클래스이름 {...}

- final 클래스를 상속받을 수 없다.



final 클래스

final 접근지정자 class 클래스이름 {...}

- final 클래스를 상속받을 수 없다.

```
FootballPlayer.java
package finaltest.extend;

public class FootballPlayer {

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

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String getCountryName() {
        return countryName;
    }
    public void setCountryName(String countryName) {
        this.countryName = countryName;
    }
    public String getTeamName() {
        return teamName;
    }
    public void setTeamName(String teamName) {
        this.teamName = teamName;
    }
}
```

```
SonHeungMin.java
package finaltest.extend;

final public class SonHeungMin extends FootballPlayer {

    public void doMilitaryService() {
        System.out.println("4주간 군사훈련");
    }
}
```

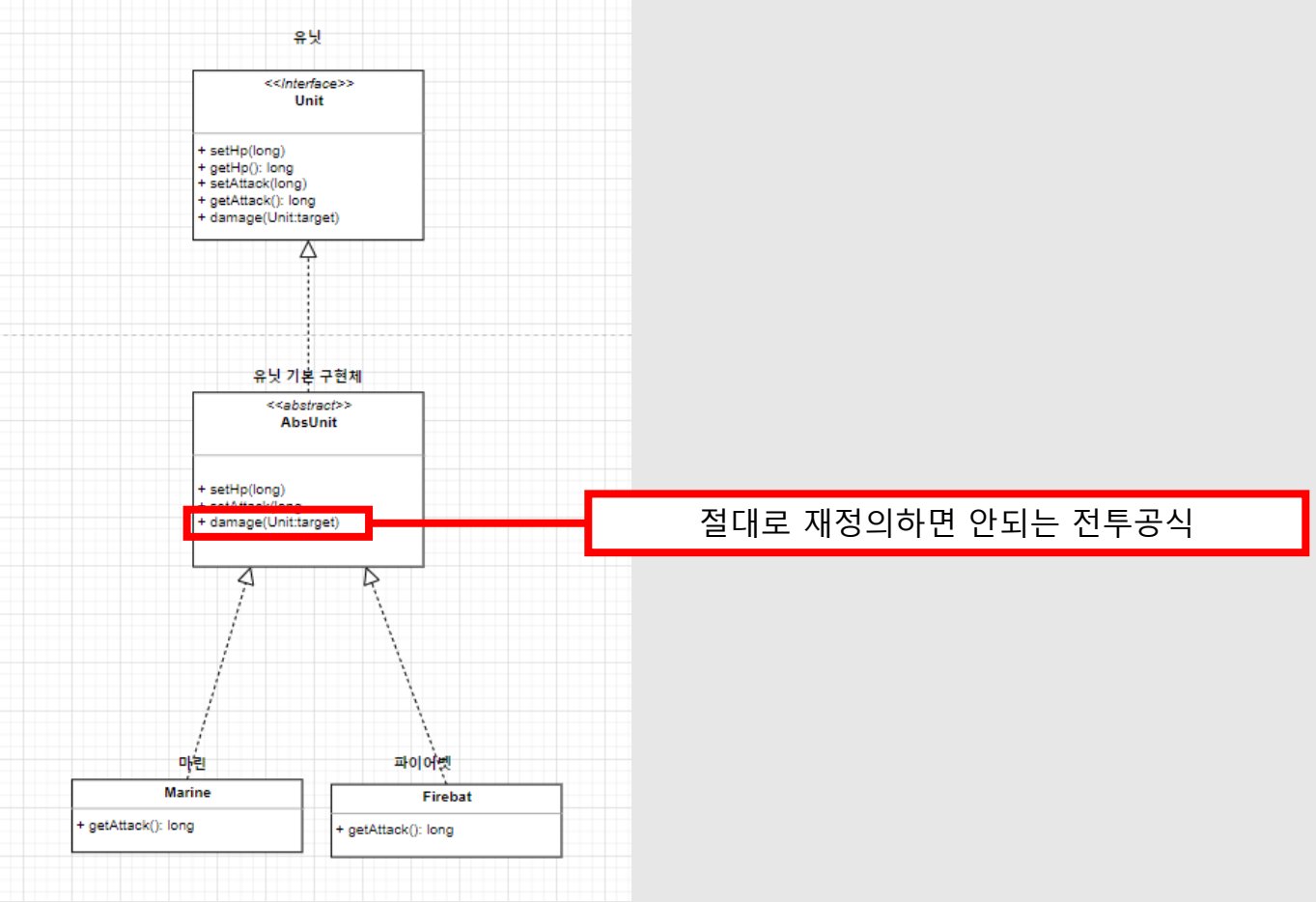
```
Ronaldo.java
package finaltest.extend;

public class Ronaldo extends SonHeungMin {

}
```

" 다른 개발자가 상속받지 않도록 최하위 타입임을 final 로 명시가능하다 "

final 메서드

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

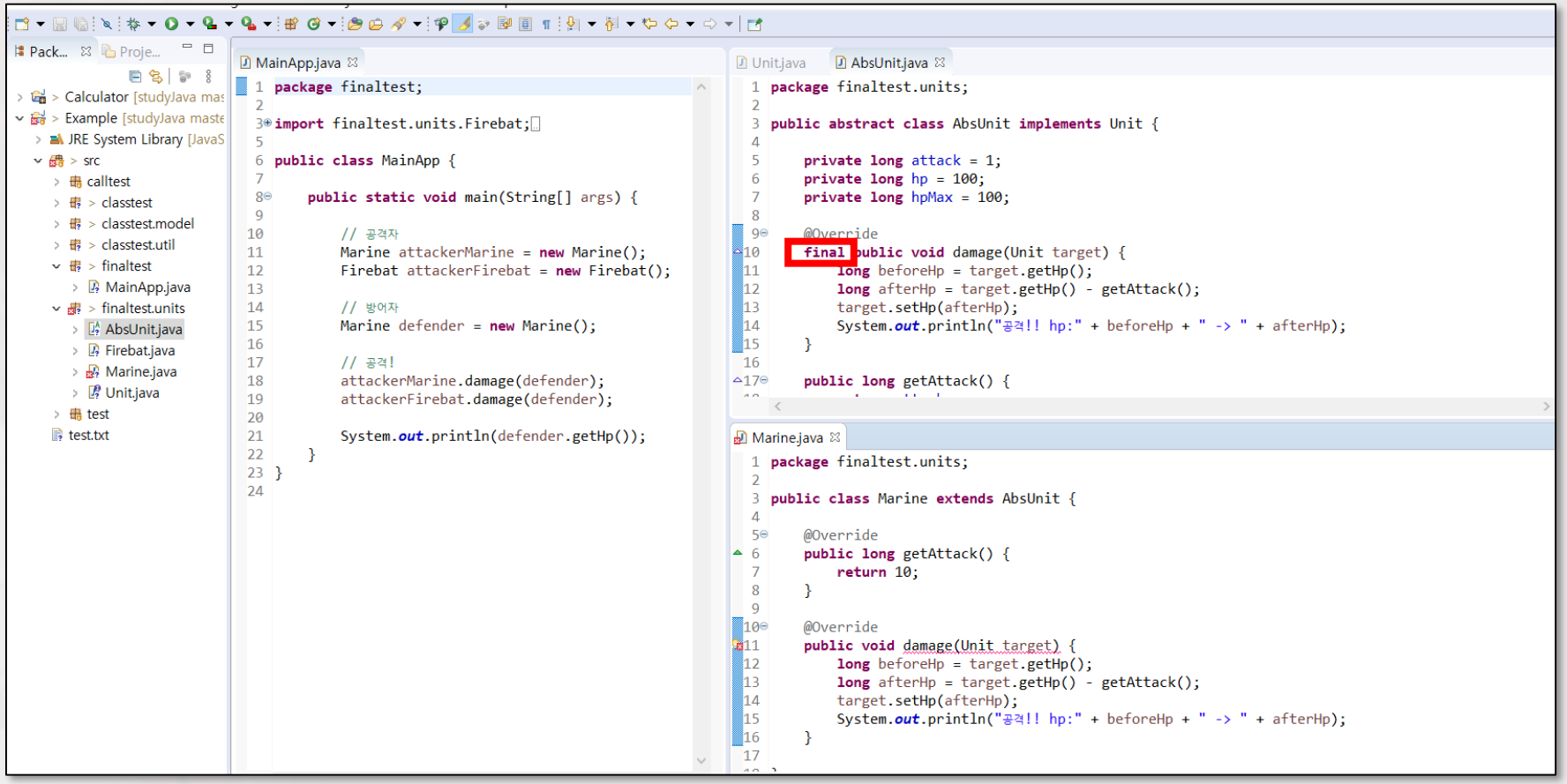
- final 메서드를 재정의 할 수 없다.



final 메서드

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

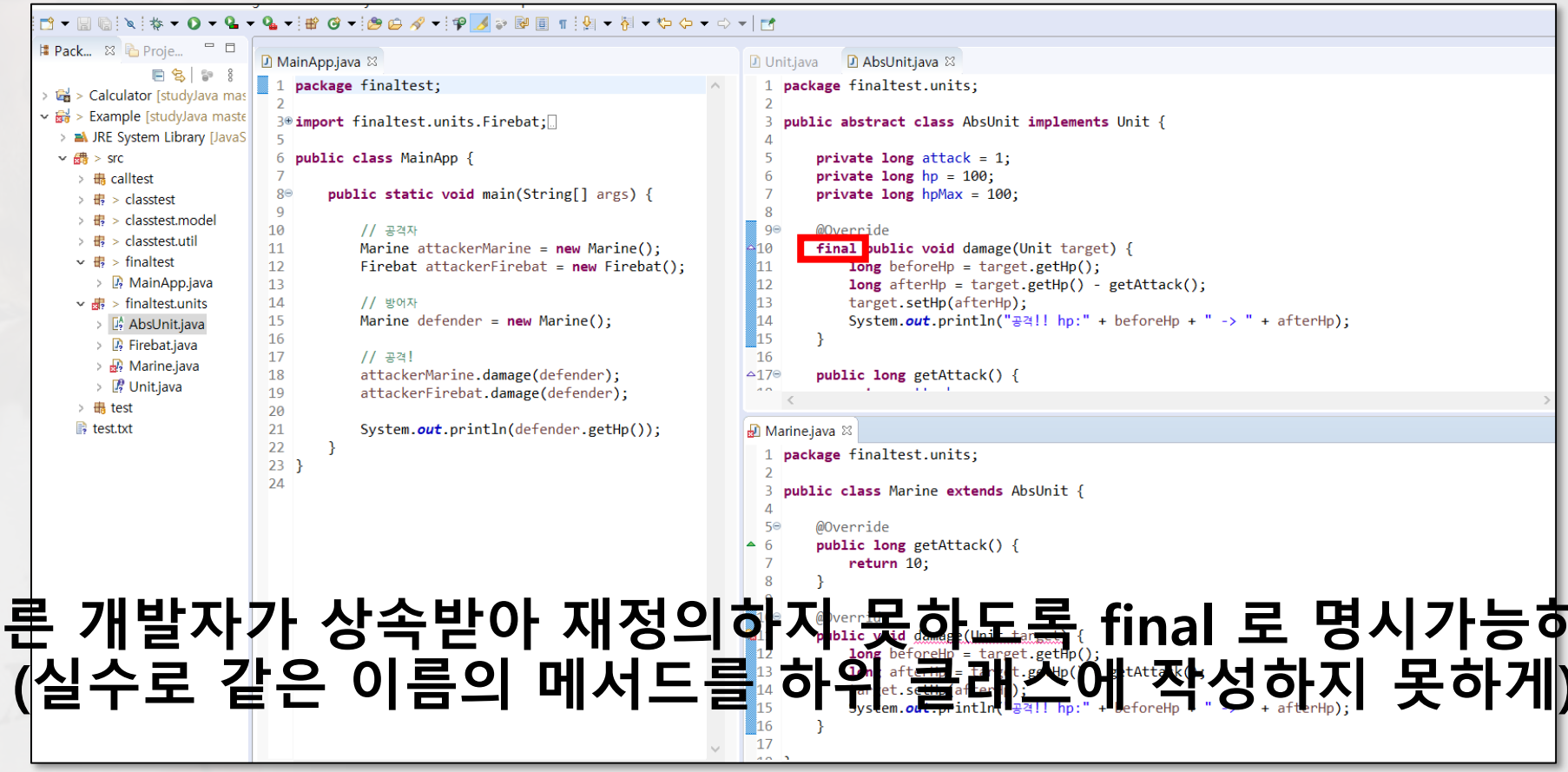
- final 메서드를 재정의 할 수 없다.



final 메서드

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

- final 메서드를 재정의 할 수 없다.

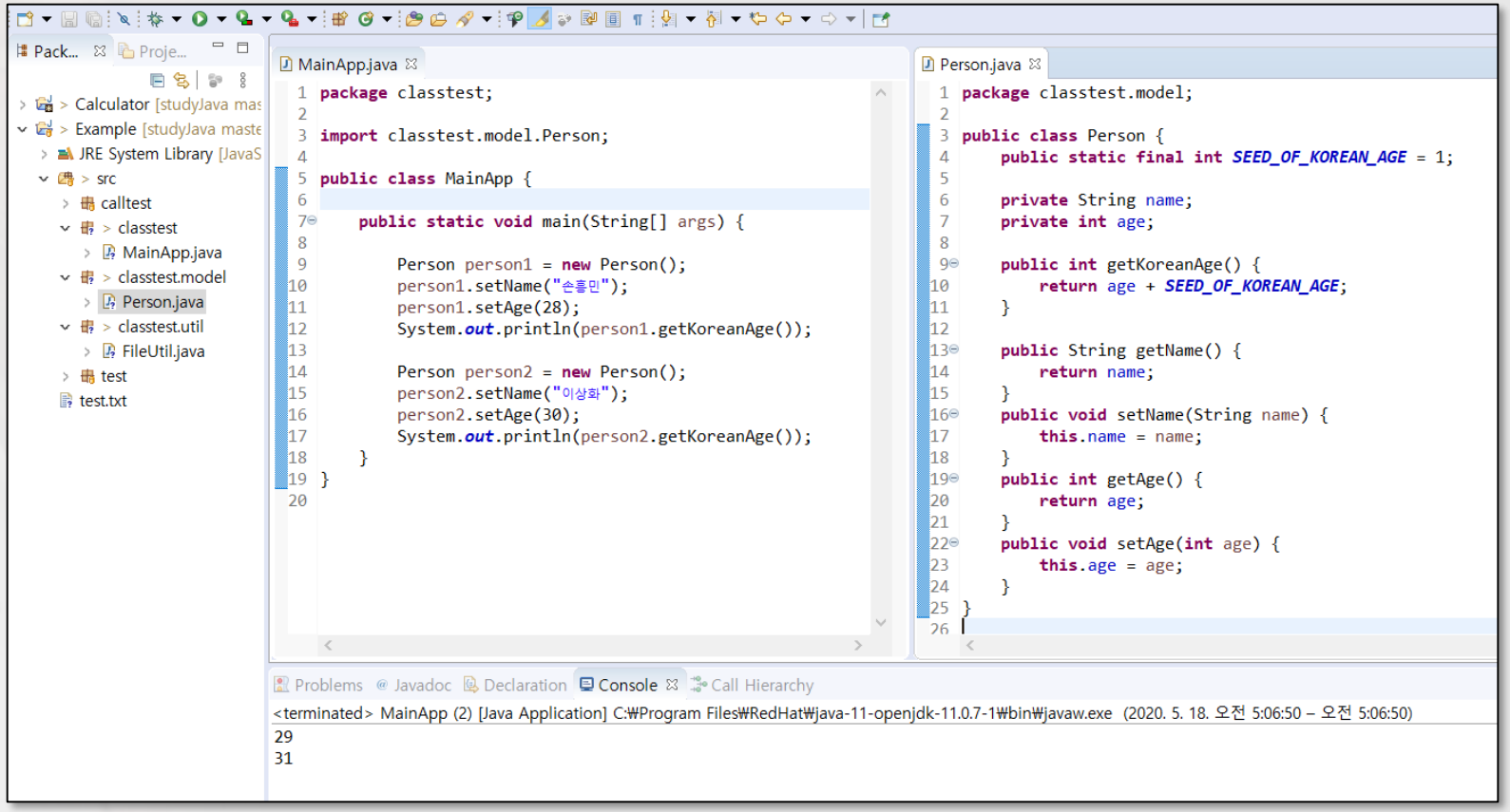


" 다른 개발자가 상속받아 재정의하지 못하도록 final 로 명시가능하다
(실수로 같은 이름의 메서드를 하위 클래스에 작성하지 못하게) "

final 상수

final [static] 타입 상수이름 = **초기값**;

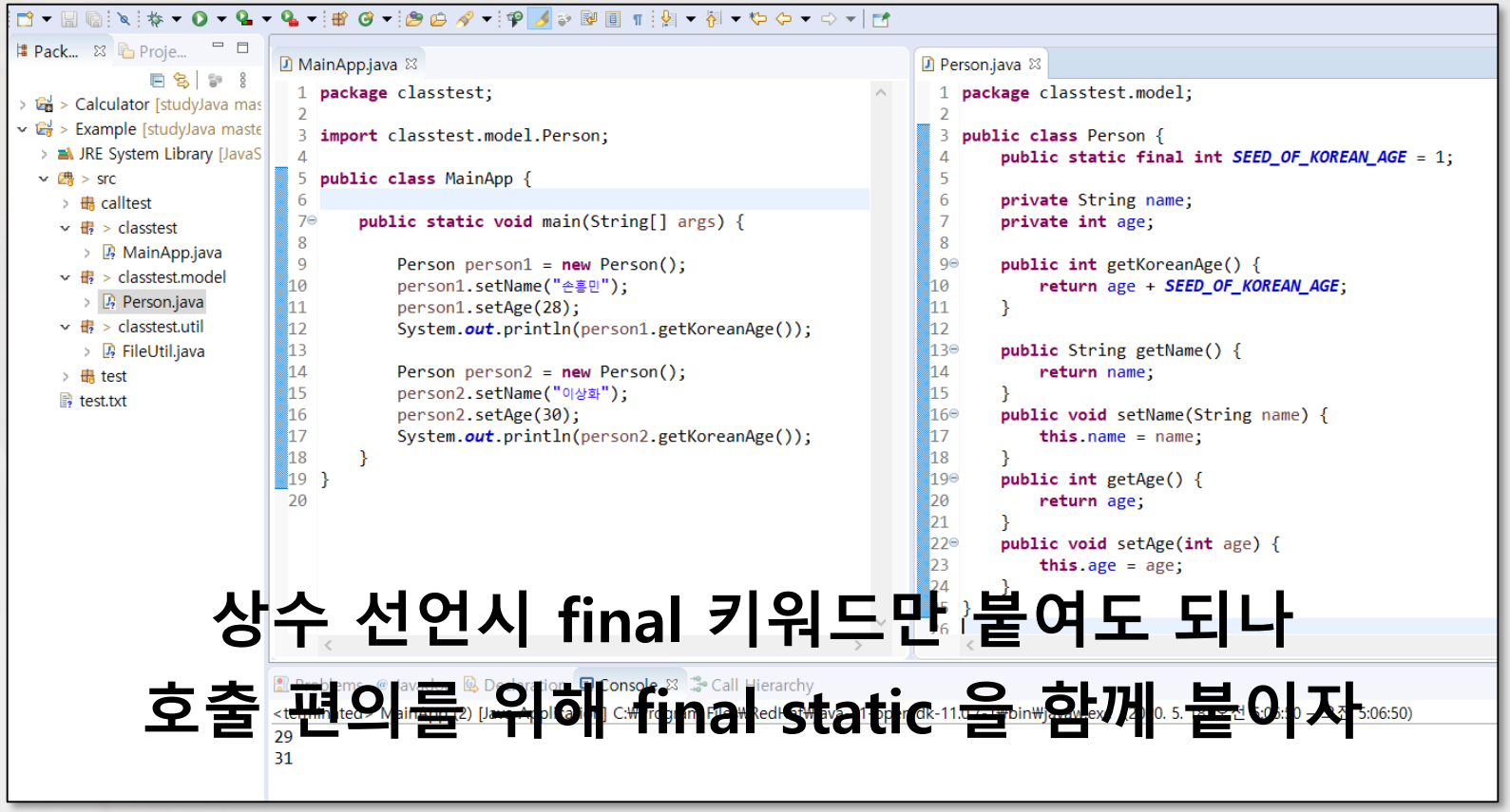
- final 멤버변수는 선언시 초기값을 지정해야 하며, 한번 정의되면 값을 변경할 수 없어 상수로 쓴다.



final 상수

final [static] 타입 상수이름 = **초기값**;

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

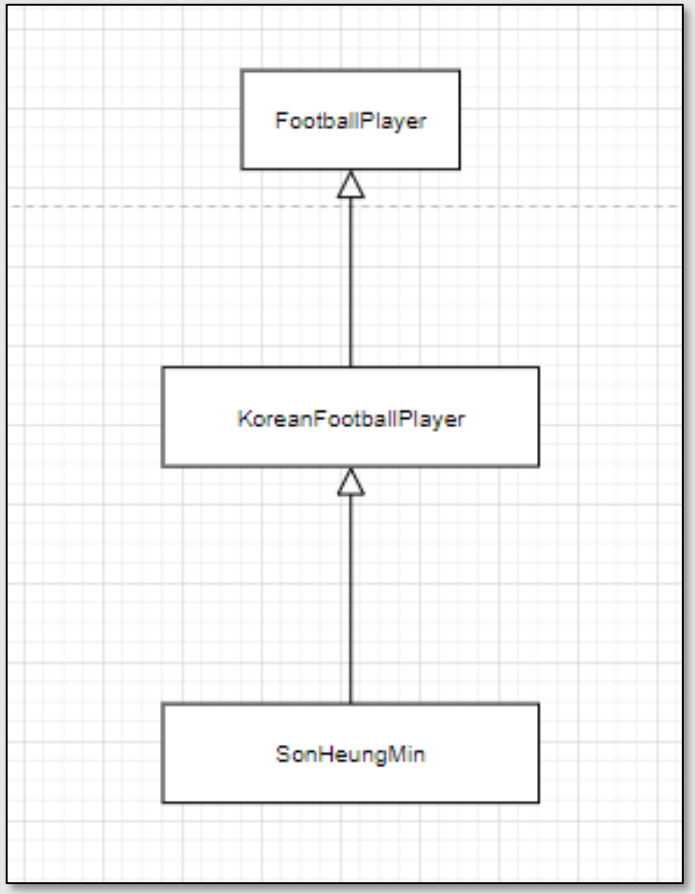


상수 선언시 final 키워드만 붙여도 되나
호출 편의를 위해 final static 을 함께 붙이자

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

Pack... Proje...

- Calculator [studyJava mas
- Example [studyJava maste
  - JRE System Library [JavaS
  - src
    - calltest
    - classtest
    - classtest.model
    - classtest.util
    - finaltest
    - finaltest.extend
      - FootballPlayer.java
      - KoreanFootballPlay
      - Ronaldo.java
      - SonHeungMin.java
    - finaltest.units
    - test
      - ArrayTest.java
  - test.txt

MainApp.java

```java
package finaltest;

import finaltest.extend.SonHeungMin;

public class MainApp {

    public static void main(String[] args) {
        new SonHeungMin();
    }
}
```

FootballPlayer.java

```java
package finaltest.extend;

public class FootballPlayer {

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

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

```

KoreanFootballPlayer.java

```java
package finaltest.extend;

public class KoreanFootballPlayer extends FootballPlayer {

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

Ronaldo.java SonHeungMin.java Marine.java

```java
package finaltest.extend;

final public class SonHeungMin extends KoreanFootballPlayer {

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

```

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 6:26:04 – 오전 6:26:04)
init FootballPlayer
init KoreanFootballPlayer
init SonHeungMin
```

상속 선언

Package ... | Project Ex...

- Calculator [studyJava master]
- Example [studyJava master]
  - JRE System Library [JavaSE-11]
  - src
    - calltest
    - classtest
    - classtest.model
    - classtest.util
    - finaltest
    - finaltest.extend
      - FootballPlayer.java
      - KoreanFootballPlayer.java
      - Ronaldo.java
      - SonHeungMin.java
    - finaltest.units
    - test
      - ArrayTest.java
  - test.txt

MainApp.java

```java
package finaltest;

import finaltest.extend.SonHeungMin;

public class MainApp {

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

FootballPlayer.java

```java
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");
    }
```

KoreanFootballPlayer.java

```java
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;
```

Ronaldo.java | SonHeungMin.java | Marine.java

```java
package finaltest.extend;

final public class SonHeungMin extends KoreanFootballPlayer {

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

    @Override
```

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 6:29:43 – 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.

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

```java
public class MainApp {

    public static void main(String[] args) {
        SonHeungMin sonHeungMin = new SonHeungMin();
        KoreanFootballPlayer koreanFootballPlayer = new KoreanFootballPlayer();
        FootballPlayer footballPlayer = new FootballPlayer();

        boolean sonHeungMinIsSonHeungMin = sonHeungMin instanceof SonHeungMin;
        boolean sonHeungMinIsKoreanFootballPlayer = sonHeungMin instanceof KoreanFootballPlayer;
        boolean sonHeungMinIsFootballPlayer = sonHeungMin instanceof FootballPlayer;
        boolean footballPlayerIsSonHeungMin = footballPlayer instanceof SonHeungMin;

        System.out.println(sonHeungMinIsFootballPlayer);
        System.out.println(sonHeungMinIsKoreanFootballPlayer);
        System.out.println(sonHeungMinIsFootballPlayer);
        System.out.println(footballPlayerIsSonHeungMin);
    }
}
```

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 6:4
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

```java
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

```java
public class MainApp {

    public static void main(String[] args) {
        FootballPlayer footballPlayer = new KoreanFootballPlayer("임꺽정");
        System.out.println(footballPlayer.getName());

        // 다운캐스팅 성공
        KoreanFootballPlayer koreanFootballPlayer = (KoreanFootballPlayer) footballPlayer;
        System.out.println(koreanFootballPlayer.getMilitaryServiceMonths());

        // 다운캐스팅 실패!!
        // ClassCastException 발생!! cannot be cast to class
        SonHeungMin sonHeungMin = (SonHeungMin) footballPlayer;
        System.out.println(sonHeungMin.getMilitaryServiceMonths());
    }
}
```

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)
임꺽정
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

MainApp.java | SonHeungMin.java | KoreanFootballPlayer.java | FootballPlayer.java | Unit.java

```java
package finaltest;

import finaltest.units.Firebat;
import finaltest.units.Marine;

public class MainApp {

    public static void main(String[] args) {
        // 공격자
        Marine marineAttacker = new Marine();
        Firebat firebatAttacker = new Firebat();

        // 방어자
        Marine defender = new Marine();

        // 전투 수행!!
        marineAttacker.damage(defender);
        firebatAttacker.damage(defender);

        // 전투 결과
        System.out.println(defender.getHp());
    }
}
```

AbsUnit.java

```java

public abstract class AbsUnit implements Unit {

    private long attack = 1;
    private long hp = 100;
    private long hpMax = 100;

    @Override
    final public void damage(Unit target) {
        long beforeHp = target.getHp();
        long afterHp = target.getHp() - getAttack();
        target.setHp(afterHp);
        System.out.println("공격!! hp:" + beforeHp + " -> " + afterHp);
    }

    public long getAttack() {
        return attack;
    }
```

Marine.java

```java
package finaltest.units;

public class Marine extends AbsUnit {

    @Override
    public long getAttack() {
        return 10;
    }
}
```

Firebat.java

```java
package finaltest.units;

public class Firebat extends AbsUnit {

    @Override
    public long getAttack() {
        return 15;
    }
}
```

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

MainApp.java | SonHeungMin.java | KoreanFootballPlayer.java | FootballPlayer.java | Unit.java

```java
package finaltest;

import finaltest.units.Firebat;
import finaltest.units.Marine;

public class MainApp {

    public static void main(String[] args) {
        // 공격자
        Marine marineAttacker = new Marine();
        Firebat firebatAttacker = new Firebat();

        // 방어자
        Marine defender = new Marine();

        // 전투 수행!!
        marineAttacker.damage(defender);
        firebatAttacker.damage(defender);

        // 전투 결과
        System.out.println(defender.getHp());
    }
}
```

AbsUnit.java

```java

public abstract class AbsUnit implements Unit {

    private long attack = 1;
    private long hp = 100;
    private long hpMax = 100;

    @Override
    final public void damage(Unit target) {
        long beforeHp = target.getHp();
        long afterHp = target.getHp() - getAttack();
        target.setHp(afterHp);
        System.out.println("공격!! hp:" + beforeHp + " -> " + afterHp);
    }

    public long getAttack() {
        return attack;
    }
```

Marine.java

```java
package finaltest.units;

public class Marine extends AbsUnit {

    @Override
    public long getAttack() {
        return super.getAttack() + 1;
    }
}
```

Firebat.java

```java
package finaltest.units;

public class Firebat extends AbsUnit {

    @Override
    public long getAttack() {
        return super.getAttack() + 2;
    }
}
```

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
```

패키지 시스템

관련있는 클래스나 인터페이스를 한 디렉터리에 묶어 놓는 것

- 클래스 파일들을 각각의 디렉터리에 별도로 보관하며 **package 패키지명** 을 먼저 선언 해야한다. **pakcage 패키지명** 을 선언하지 않으면 디폴트 패키지에 속한다고 한다.
- 하나의 프로그램을 여러 개발자가 각각의 패키지를 나누어 개발한다.
- 서로 다른 디렉터리에 같은 이름의 클래스가 존재해도 경로명으로 구분이 가능
- 다른 패키지의 클래스를 사용하기 위해서는 **import 패키지명.클래스명** 을 먼저 선언 해야한다.
- 모듈의 재사용을 위해 패키지를 설계하고, 여러 프로젝트 개발시 같은 기능에 대해 패키지를 그대로 포함시킨다.

패키지 시스템 – java.lang 패키지

Java 프로그래밍에 필요한 기본적인 클래스를 포함해 제공한다.

- 특별히, **컴파일러가 모든 클래스에 java.lang 패키지를 import 한다.**

- 포함된 클래스 예시
  Object, String, StringBuilder, StringBuffer, Math, Integer, Boolean, Character, …

패키지 시스템 – java.util 패키지

Java 프로그래밍에 필요한 다양한 유틸 클래스를 포함해 제공한다.

- **import java.util.클래스명;** 을 먼저 선언후 호출해야한다.
- 포함된 클래스 예시
  Date, Calendar, Random, Scanner, StringTokenizer, …

패키지 시스템 – java.io 패키지

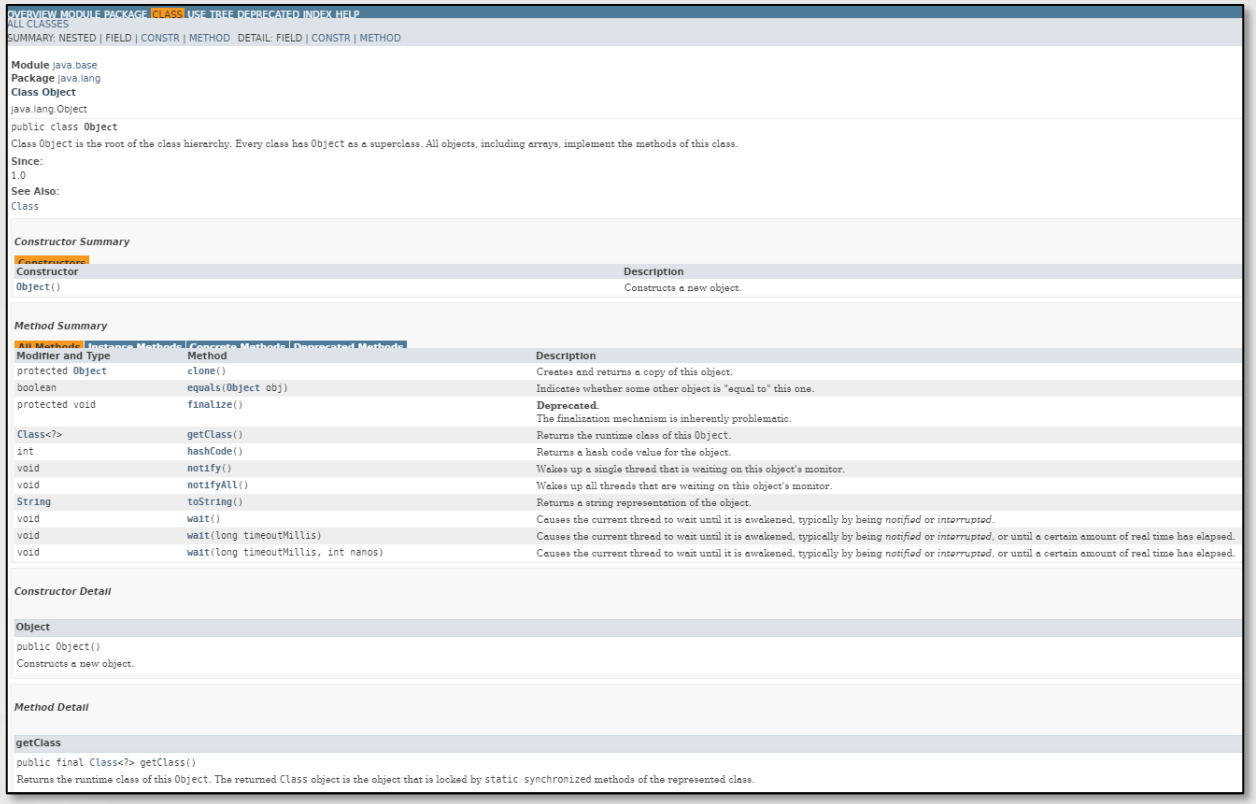
키보드, 모니터, 프린터, 디스크 등 입출력에 사용되는 클래스를 포함해 제공한다.

- **import java.io.클래스명;** 을 먼저 선언후 호출해야한다.
- 포함된 클래스 예시
  File, InputStream, OutputStream, Writer, Reader, …

Object

Java 클래스 최상위 클래스이다.

- java.lang 패키지에 속한 클래스이며, 컴파일러에 의해 Java의 모든 클래스는 Object 클래스를 상속 받게 된다.



OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
ALL CLASSES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Module java.base
Package java.lang
Class Object
java.lang.Object

public class **Object**

Class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.

Since:
1.0
See Also:
Class

Constructor Summary

Constructors

Constructor	Description
Object()	Constructs a new object.

Method Summary

All Methods | Instance Methods | Concrete Methods | Deprecated Methods

Modifier and Type	Method	Description
protected Object	clone()	Creates and returns a copy of this object.
boolean	equals(Object obj)	Indicates whether some other object is "equal to" this one.
protected void	finalize()	**Deprecated.** The finalization mechanism is inherently problematic.
Class<?>	getClass()	Returns the runtime class of this Object.
int	hashCode()	Returns a hash code value for the object.
void	notify()	Wakes up a single thread that is waiting on this object's monitor.
void	notifyAll()	Wakes up all threads that are waiting on this object's monitor.
String	toString()	Returns a string representation of the object.
void	wait()	Causes the current thread to wait until it is awakened, typically by being *notified* or *interrupted*.
void	wait(long timeoutMillis)	Causes the current thread to wait until it is awakened, typically by being *notified* or *interrupted*, or until a certain amount of real time has elapsed.
void	wait(long timeoutMillis, int nanos)	Causes the current thread to wait until it is awakened, typically by being *notified* or *interrupted*, or until a certain amount of real time has elapsed.

Constructor Detail

Object

public Object()
Constructs a new object.

Method Detail

getClass

public final Class<?> getClass()
Returns the runtime class of this Object. The returned Class object is the object that is locked by static synchronized methods of the represented class.

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Object.html

```java
package java.lang;

import jdk.internal.HotSpotIntrinsicCandidate;

/**
 * Class {@code Object} is the root of the class hierarchy.
 * Every class has {@code Object} as a superclass. All objects,
 * including arrays, implement the methods of this class.
 *
 * @author  unascribed
 * @see     java.lang.Class
 * @since   1.0
 */
public class Object {

    private static native void registerNatives();
    static {
        registerNatives();
    }

    @HotSpotIntrinsicCandidate
    public Object() {}

    @HotSpotIntrinsicCandidate
    public final native Class<?> getClass();

    @HotSpotIntrinsicCandidate
    public native int hashCode();

    public boolean equals(Object obj) {
        return (this == obj);
    }

    @HotSpotIntrinsicCandidate
    protected native Object clone() throws CloneNotSupportedException;

    public String toString() {
        return getClass().getName() + "@" + Integer.toHexString(hashCode());
    }

    @HotSpotIntrinsicCandidate
    public final native void notify();

    @HotSpotIntrinsicCandidate
    public final native void notifyAll();

    public final void wait() throws InterruptedException {
        wait(0L);
    }

    public final native void wait(long timeoutMillis) throws InterruptedException;

    public final void wait(long timeoutMillis, int nanos) throws InterruptedException {
        if (timeoutMillis < 0) {
            throw new IllegalArgumentException("timeoutMillis value is negative");
        }

        if (nanos < 0 || nanos > 999999) {
            throw new IllegalArgumentException(
                                "nanosecond timeout value out of range");
        }

        if (nanos > 0) {
            timeoutMillis++;
        }

        wait(timeoutMillis);
    }

    @Deprecated(since="9")
    protected void finalize() throws Throwable { }
}
```

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Object.html

Object - hashcode

인스턴스의 고유한 주소 값

- Object 클래스에 **public int hashCode();** 메서드가 선언되어있음.
  (Java의 모든 클래스 인스턴스들은 hashCode() 메서드를 호출해 고유한 주소 값을 반환할 수 있다)

- 인스턴스의 주소가 int 형 값으로 반환된다.

- JVM 프로세스가 실행되는 동안 한 인스턴스에서 여러 번 hashCode(); 메서드를 호출해도 지속적으로 동일한 정수를 반환한다.

- 동일한 프로그램이 서로 다른 JVM 프로세스에서 실행되었다면 hashCode 는 동일한 정수를 반환하지 않을 수 있다.

- 고유하기 때문에 Java의 HashMap 에서 key로 사용된다

Object - hashcode

인스턴스의 고유한 주소 값

Package Explorer

- Calculator [studyJava master]
- Example [studyJava master]
  - JRE System Library [JavaSE-1
  - src
    - abstracttest
    - calltest
    - classtest
    - classtest.model
    - classtest.util
    - equalstest
      - MainApp.java
      - Person.java
    - finaltest
    - finaltest.extend
    - finaltest.units
    - generictest
    - generictest.building
    - generictest.unit
    - interfaces

MainApp.java

```java
package equalstest;

public class MainApp {

    public static void main(String[] args) throws Exception {

        // heap 메모리에 Person 타입의 인스턴스를 위한 메모리를 할당받아
        // person 이라는 레퍼런스로 주소를 할당받는다.
        Person person = new Person();

        // person 인스턴스의 고유 값 hashCode
        int hashCode = person.hashCode();
        System.out.println(hashCode); // 1705929636

        for (int i = 0; i < 100; i++) {

            if (hashCode != person.hashCode()) {
                System.out.println(person.hashCode());
                throw new Exception();
            }
        }
    }
}
```

Person.java

```java
package equalstest;

public class Person /* extends Object */ {
    private String name;
    private int age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

"레퍼런스의 hashCode 정수값은 프로세스 내에서 변하지 않는다."

Object - hashcode

인스턴스의 고유한 주소 값

MainApp.java

```java
package equalstest;

public class MainApp {

    public static void main(String[] args) throws Exception {

        // heap 메모리에 Person 타입의 인스턴스를 위한 메모리를 할당받아
        // person 이라는 레퍼런스로 주소를 할당받는다.
        Person person = new Person();

        // heap 메모리에 Person 타입의 인스턴스를 위한 메모리를 할당받아
        // person2 이라는 레퍼런스로 주소를 할당받는다.
        Person person2 = new Person();

        System.out.println(person == person2);
        System.out.println(person != person2);

        System.out.println(person.hashCode());
        System.out.println(person2.hashCode());
    }
}
```

Problems @ Javadoc Declaration Search Console Call Hierarchy

<terminated> MainApp (4) [Java Application] C:₩Program Files₩RedHat₩java-11-openjdk-11.

```
false
true
1252585652
2036368507
```

MainApp.java

```java
package equalstest;

public class MainApp {

    public static void main(String[] args) throws Exception {

        // heap 메모리에 Person 타입의 인스턴스를 위한 메모리를 할당받아
        // person 이라는 레퍼런스로 주소를 할당받는다.
        Person person = new Person();

        // person2 이라는 레퍼런스에 person 레퍼런스를 할당한다.
        Person person2 = person;

        System.out.println(person == person2);
        System.out.println(person != person2);

        System.out.println(person.hashCode());
        System.out.println(person2.hashCode());
    }
}
```

Problems @ Javadoc Declaration Search Console Call Hierarchy

<terminated> MainApp (4) [Java Application] C:₩Program Files₩RedHat₩java-11-openjdk-11.

```
true
false
1252585652
1252585652
```

" 레퍼런스의 hashCode 값은 인스턴스마다 고유하다. "

인스턴스가 동일한지 비교할 때 쓰인다.

Object - hashcode

인스턴스의 고유한 주소 값

- 하지만.... 고유하지 않을 수 있다...
  HashMap 의 Key 로 사용하는 경우 반드시 hashCode 를 재정의하자.

  https://eclipsesource.com/blogs/2012/09/04/the-3-things-you-should-know-about-hashcode/

Object – equals()

두 인스턴스가 같은지 비교

- Object 클래스에 **public boolean equals(Object obj);** 메서드가 선언 및 구현되어있음.

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

- 호출한 인스턴스와 인자로 입력된 인스턴스가 같은지 여부를 판단해 같으면 true, 다르면 false 를 반환한다.

- 인스턴스간 같은지 값 비교가 필요하므로 Java는 모든 클래스가 equals() 메서드를 재정의하는 것을 권장한다.
  (단, equals() 메서드를 재정의할때 hashCode() 메서드도 함께 재정의해야만 한다!)

Object – equals()

두 인스턴스가 같은지 비교

MainApp.java

```java
package equalstest;

public class MainApp {

    public static void main(String[] args) throws Exception {

        // heap 메모리에 Person 타입의 인스턴스를 위한 메모리를 할당받아
        // person 이라는 레퍼런스로 주소를 할당받는다.
        Person person = new Person();

        // heap 메모리에 Person 타입의 인스턴스를 위한 메모리를 할당받아
        // person2 이라는 레퍼런스로 주소를 할당받는다.
        Person person2 = new Person();

        System.out.println(person.equals(person2));
        System.out.println(!person.equals(person2));

        System.out.println(person.hashCode());
        System.out.println(person2.hashCode());
    }
}
```

Problems @ Javadoc Declaration Search Console Call Hierarchy

<terminated> MainApp (4) [Java Application] C:₩Program Files₩RedHat₩java-11-openjdk-11.0

```
false
true
1252585652
2036368507
```

MainApp.java

```java
package equalstest;

public class MainApp {

    public static void main(String[] args) throws Exception {

        // heap 메모리에 Person 타입의 인스턴스를 위한 메모리를 할당받아
        // person 이라는 레퍼런스로 주소를 할당받는다.
        Person person = new Person();

        // person2 이라는 레퍼런스에 person 레퍼런스를 할당한다.
        Person person2 = person;

        System.out.println(person.equals(person2));
        System.out.println(!person.equals(person2));

        System.out.println(person.hashCode());
        System.out.println(person2.hashCode());
    }
}
```

Problems @ Javadoc Declaration Search Console Call Hierarchy

<terminated> MainApp (4) [Java Application] C:₩Program Files₩RedHat₩java-11-openjdk-11.0

```
true
false
1252585652
1252585652
```

Object – equals()

두 인스턴스가 같은지 비교

```java
package equalstest;

public class Person /* extends Object */ {
    private String name;
    private int age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

equals 메서드를 재정의

```java
package equalstest;

public class Person /* extends Object */ {
    private String name;
    private int age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + age;
        result = prime * result
            + ((name == null) ? 0 : name.hashCode());
        return result;
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null) return false;
        if (getClass() != obj.getClass()) return false;
        Person other = (Person) obj;
        if (age != other.age) return false;
        if (name == null) {
            if (other.name != null) return false;
        } else if (!name.equals(other.name))
            return false;
        return true;
    }
}
```

Object – 재정의된 equals()

두 인스턴스가 같은지 비교

MainApp.java

```java
package equalstest;

public class MainApp {

    public static void main(String[] args) throws Exception {

        // heap 메모리에 Person 타입의 인스턴스를 위한 메모리를 할당받아
        // person 이라는 레퍼런스로 주소를 할당받는다.
        Person person = new Person();

        // heap 메모리에 Person 타입의 인스턴스를 위한 메모리를 할당받아
        // person2 이라는 레퍼런스로 주소를 할당받는다.
        Person person2 = new Person();

        System.out.println(person.equals(person2));
        System.out.println(!person.equals(person2));

        System.out.println(person.hashCode());
        System.out.println(person2.hashCode());
    }
}
```

Problems | Javadoc | Declaration | Search | Console | Call Hierarchy

```
<terminated> MainApp (4) [Java Application] C:₩Program Files₩RedHat₩java-11-openjdk-11.
true
false
961
961
```

" 두 인스턴스간 내용이 같은지를 비교할 수 있도록 equals 메서드를 재정의하자 "

Object – 재정의된 equals()

두 인스턴스가 같은지 비교

```java
package equalstest;

import java.util.Date;

public class MainApp {

    public static void main(String[] args) throws Exception {

        // 이벤트 예약 시간
        Date reservedDate = new Date(1590686600000L);

        // 현재 시간
        Date currentDate = new Date();

        System.out.println(reservedDate == currentDate);
    }
}
```

Object – 재정의된 equals()

두 인스턴스가 같은지 비교

이벤트는 발생하지 않았다.... ㅠㅠㅠㅠ

```java
package equalstest;

import java.util.Date;

public class MainApp {

    public static void main(String[] args) throws Exception {

        // 이벤트 예약 시간
        Date reservedDate = new Date(1590686600000L);

        // 현재 시간
        Date currentDate = new Date();

        System.out.println(reservedDate == currentDate);
    }
}
```

Object – 재정의된 equals()

두 인스턴스가 같은지 비교

```java
package equalstest;

import java.util.Date;

public class MainApp {

    public static void main(String[] args) throws Exception {

        // 이벤트 예약 시간
        Date reservedDate = new Date(1590686600000L);

        // 현재 시간
        Date currentDate = new Date();

        System.out.println(reservedDate.getTime() == currentDate.getTime());
    }
}
```

```java
package equalstest;

import java.util.Date;

public class MainApp {

    public static void main(String[] args) throws Exception {

        // 이벤트 예약 시간
        Date reservedDate = new Date(1590686600000L);

        // 현재 시간
        Date currentDate = new Date();

        System.out.println(reservedDate.equals(currentDate));
    }
}
```

" **Primitive Type 간 비교는 ==, != 으로 하고**
Reference Type 간 비교는 equals 로 하자! "

(Reference Type 간 ==, != 으로 비교하면 주소를 비교하는것...)

Object – toString()

인스턴스의 문자열 표현을 반환

- Object 클래스에 **public String toString();** 메서드가 선언 및 구현되어있음.

```
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

- 사람이 읽기 쉽고 간결해야하며 Java는 모든 클래스가 toString() 메서드를 재정의하는 것을 권장한다.

Object – toString()

인스턴스의 문자열 표현을 반환

MainApp.java

```java
package tostringtest;

import equalstest.Person;

public class MainApp {

    public static void main(String[] args) throws Exception {

        Person person = new Person();
        person.setName("백승찬");
        person.setAge(31);
        System.out.println(person.toString());
    }
}

```

Person.java

```java
package equalstest;

public class Person /* extends Object */ {
    private String name;
    private int age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}

```

Problems @ Javadoc Declaration Search Console Call Hierarchy

```
<terminated> MainApp (5) [Java Application] C:₩Program Files₩RedHat₩java-11-openjdk-11.0.4-1₩bin₩javaw.exe (2020. 5. 29. 오전 2:09:52)
equalstest.Person@65ae6ba4
```

Object – toString()

인스턴스의 문자열 표현을 반환

MainApp.java

```java
package tostringtest;

import equalstest.Person;

public class MainApp {

    public static void main(String[] args) throws Exception {

        Person person = new Person();
        person.setName("백승찬");
        person.setAge(31);
        System.out.println(person.toString());
    }
}
```

Person.java

```java
package equalstest;

public class Person /* extends Object */ {
    private String name;
    private int age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    @Override
    public String toString() {
        return "Person [name=" + name + ", age=" + age + "]";
    }
}
```

Problems | Javadoc | Declaration | Search | Console | Call Hierarchy

```
<terminated> MainApp (5) [Java Application] C:₩Program Files₩RedHat₩java-11-openjdk-11.0.4-1₩bin₩javaw.exe (2020. 5. 29. 오전 2:10:41)
Person [name=백승찬, age=31]
```

Wrapper

Java의 8개 Primitive Type 을 Reference Type 으로 다루기 위해 제공되는 클래스들을 부르는 이름

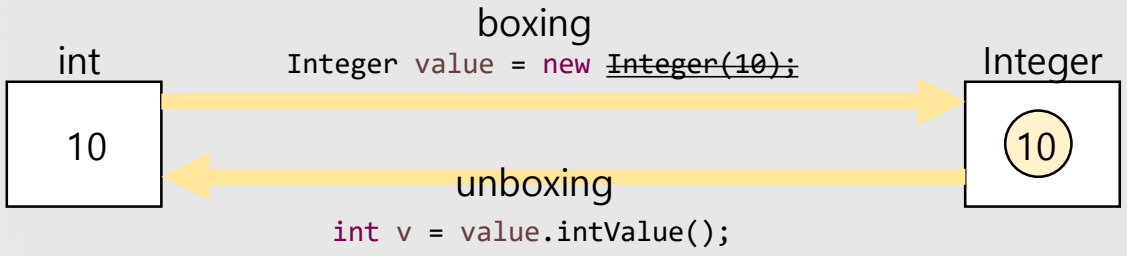
- Primitive Type → Reference Type 으로 Boxing 과
  Reference Type → Primitive Type 으로 Unboxing 이 가능하다.

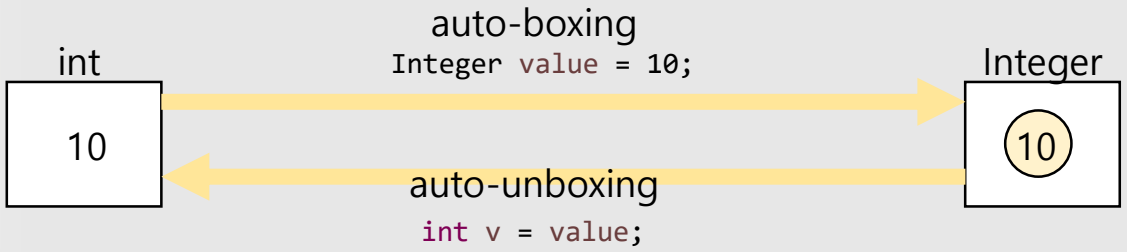
- Wrapper 는 Reference Type 이므로 null 이 할당될 수 있다.

Primitive type	byte	short	int	long	char	float	double	boolean
Wrapper	Byte	Short	Integer	Long	Character	Float	Double	Boolean

Wrapper – Boxing/Unboxing

Java의 8개 Primitive Type 을 Reference Type 으로 다루기 위해 제공되는 클래스들을 부르는 이름





Wrapper – equals(), ==

Java의 8개 Primitive Type 을 Reference Type 으로 다루기 위해 제공되는 클래스들을 부르는 이름

- 조심..

https://stackoverflow.com/questions/1700081/why-is-128-128-false-but-127-127-is-true-when-comparing-integer-wrappers-in-ja

"

Primitive Type 간 비교는 ==, != 으로 하고
Reference Type 간 비교는 equals 로 하자!
(Reference Type 간 ==, != 으로 비교하면 주소를 비교하는것...)

"

String

수정 불가능한 문자열 클래스

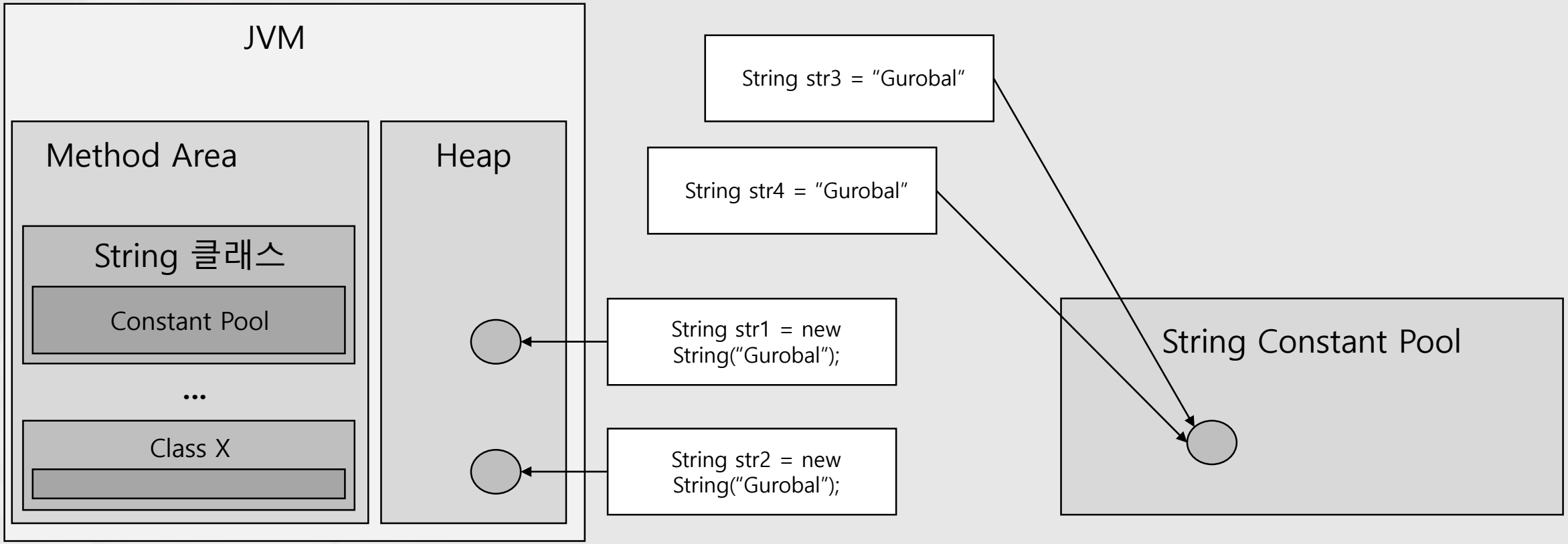
- Java의 모든 스트링 리터럴은 (“” 로 표기되는 모든것) String 객체로 처리된다.

- 스트링 리터럴은 리터럴 테이블로 특별히 관리하고 동일한 스트링 리터럴은 공유하여 사용한다.
  그러나, new String() 에 의해 생성된 문자열은 heap 메모리에 독립적으로 생성된다.

- 스트링 객체는 수정이 불가능하기 때문에 공유되어도 문제가 발생하지 않는다.
  ← 쓰지말자 StringBuilder, StringBuffer

String

수정 불가능한 문자열 클래스



String

수정 불가능한 문자열 클래스

```java
package stringconstantpooltest;

public class MainApp {

    public static void main(String[] args) {

        String str1 = new String("Gurobal");
        String str2 = new String("Gurobal");
        String str3 = "Gurobal";
        String str4 = "Gurobal";


        System.out.println(str1 == str2);
        System.out.println(str1 == str3);

        System.out.println(str3 == str4);
    }

}
```

Primitive Type 간 비교는 ==, != 으로 하고
Reference Type 간 비교는 equals 로 하자!

(Reference Type 간 ==, != 으로 비교하면 주소를 비교하는것...)

StringBuilder, StringBuffer

수정 가능한 문자열 클래스

- 문자열을 저장하는 가변 크기의 버프를 가지고 있기 때문에 스트링 수정이 가능하다.
  StringBuilder 는 공유되면 문제가 발생한다.
  StringBuffer 는 공유해도 문제가 발생하지 않는다. (append 에 synchronized 키워드가 붙어있음)

  수정이 가능하기 때문에 append 속도가 빠르고 gc 를 줄일 수 있다.

Collection

가변 크기의 컨테이너

- 여러 타입의 요소를 다룰 수 있는 자료 구조
- 컬렉션은 제네릭 기법 (타입 매개 변수) 으로 구현되어 있다.
- 요소는 반드시 Reference Type 이어야 한다.
  (Primitive Type 안됨)

Generic

모든 종류의 타입을 다룰 수 있도록 일반화된 타입 매개 변수

- C++ 의 템플릿과 기능이 동일하다.

- 다양한 데이터 타입을 적용할 수 있도록 클래스와 함수를 일반화 시켜 제네릭 메서드, 제네릭 클래스를 만들 수 있다.

- 다른 변수와 혼동을 피하기 위해 일반적으로 하나의 대문자를 사용.
  E : Element 를 의미. 컬렉션 요소를 표시할때 사용
  T : Type 을 의미.
  V : Value 를 의미.
  K : Key 를 위미.

Generic

모든 종류의 타입을 다룰 수 있도록 일반화된 타입 매개 변수

- 타입 매개 변수는 컴파일 할 때 까지만 존재, 컴파일 후에는 존재하지 않음.

MainApp.java

```java
package generictest2;

public class MainApp {

    public static void main(String[] args) {

        MyClass<String> v = new MyClass<>();
        MyClass<Integer> v2 = new MyClass<>();
        MyClass<Double> v3 = new MyClass<>();
    }
}
```

외부에서 타입을 주입시켜 사용
컴파일러에 의해 이루어진다.

MyClass.java

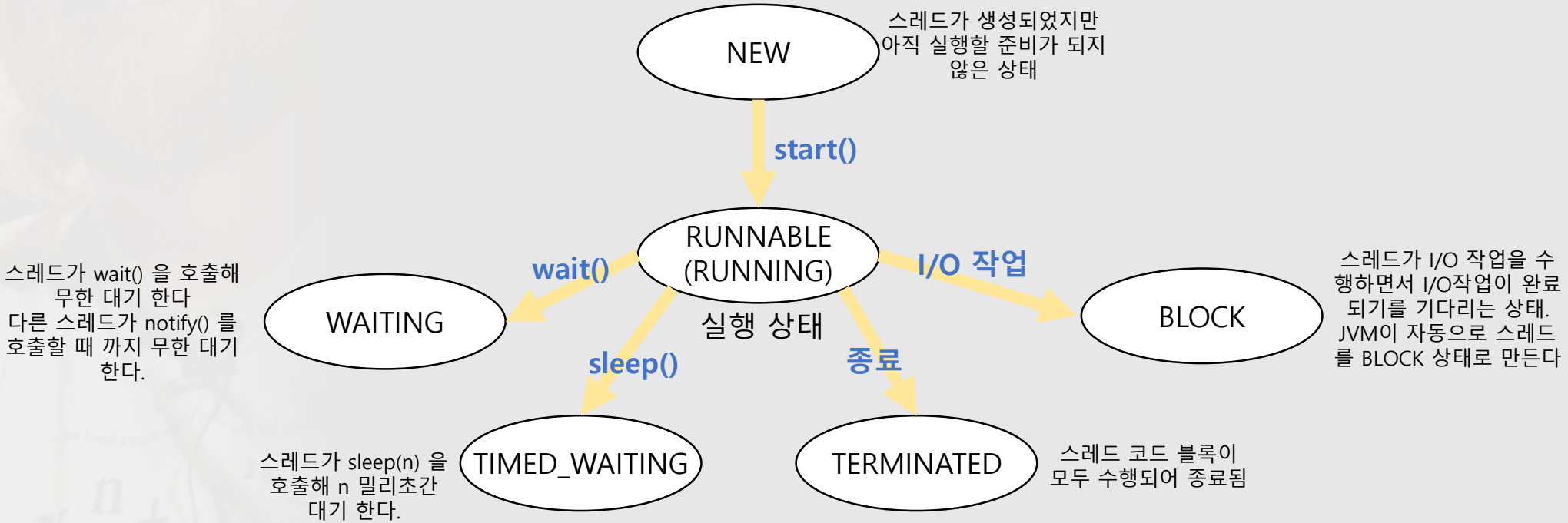
```java
package generictest2;

public class MyClass<T> {

    T value;

    public T getValue() {
        return value;
    }

    public void setValue(T value) {
        this.value = value;
    }

}
```

<> 사이에 일반화된
타입 매개변수를 선언

스레드

JVM에 의해 스케쥴되는 실행 단위 코드 블록

- 여러 개의 스레드는 **JVM 내 상호 독립적인 메모리**를 할당 받는다.
- 스레드 스케쥴링은 전적으로 JVM에 의해 이루어진다.



- public static void main(String[] args); 메서드는 **'메인 스레드'** 에 의해 실행된다.

스레드 – Thread 클래스로 구현

JVM에 의해 스케쥴되는 실행 단위 코드 블록

MainApp.java

```java
package timer;

public class MainApp {

    public static void main(String[] args) {
        System.out.println("main 스레드가 시작됩니다.");

        Timer timer = new Timer();
        timer.start();

        System.out.println("main 스레드가 종료되었습니다.");
    }

}
```

Timer.java

```java
package timer;

public class Timer extends Thread {

    int sec = 0;

    @Override
    public void run() {
        while (true) {
            System.out.println(sec);
            sec++;
            try {
                sleep(1000);
            } catch (Exception e) {
                e.printStackTrace();
                return;
            }
        }
    }

}
```

Problems | Javadoc | Declaration | Console | Call Hierarchy

<terminated> MainApp (5) [Java Application] C:₩Program Files₩RedHat₩java-11-openjdk-11.0.7-1₩bin₩javaw.exe (2020. 5. 29. 오후 12:19:27 – 오후 12:20:46)

```
main 스레드가 시작됩니다.
main 스레드가 종료되었습니다.
0
1
2
3
4
5
```

스레드 – Runnable 인터페이스로 구현

JVM에 의해 스케쥴되는 실행 단위 코드 블록

MainApp.java

```java
package timer;

public class MainApp {

    public static void main(String[] args) {
        System.out.println("main 스레드가 시작됩니다.");

        Timer timer = new Timer();
        timer.start();

        System.out.println("main 스레드가 종료되었습니다.");
    }

}
```

Timer.java

```java
package timer2;

public class Timer implements Runnable {

    int sec = 0;

    @Override
    public void run() {
        while (true) {
            System.out.println(sec);
            sec++;
            try {
                Thread.sleep(1000);
            } catch (Exception e) {
                e.printStackTrace();
                return;
            }
        }
    }
}
```

Problems | Javadoc | Declaration | Console | Call Hierarchy

```
<terminated> MainApp (5) [Java Application] C:₩Program Files₩RedHat₩java-11-openjdk-11.0.7-1₩bin₩javaw.exe (2020. 5. 29. 오후 12:23:28 – 오후 12:24:13)
main 스레드가 시작됩니다.
main 스레드가 종료되었습니다.
0
1
2
3
4
5
```

static 으로 Singleton 인스턴스 구현하기

private 생성자, static getInstance() 작성

- 직접 생성자를 호출하지 못하게 private 접근지정자를 이용
- Heap이 아닌 static 메모리로 관리하기 위해 static getInstance() 메서드를 작성

static 으로 Singleton 인스턴스 구현하기

private 생성자, static getInstance() 작성

```java
package singletontest;

public class Counter {

    private long value = 0L;

    public Counter() {}

    public long getValue() {
        return value;
    }
    public void setValue(long value) {
        this.value = value;
    }
    public long addValue(long value) {
        this.value += value;
        return this.value;
    }
}
```

```java
package singletontest;

public class Counter {

    private static Counter instance;
    private long value = 0L;

    private Counter() { }

    public static Counter getInstance() {
        if (instance == null)
            instance = new Counter();
        return instance;
    }

    public long getValue() {
        return value;
    }
    public void setValue(long value) {
        this.value = value;
    }
    public long addValue(long value) {
        this.value += value;
        return this.value;
    }
}
```

static 으로 Singleton 인스턴스 구현하기

private 생성자, static getInstance() 작성

Package Explorer

- Calculator [studyJava master]
- Example [studyJava master]
  - JRE System Library [JavaSE-1
  - src
    - abstracttest
    - calltest
    - classtest
    - classtest.model
    - classtest.util
    - finaltest
    - finaltest.extend
    - finaltest.units
    - generictest
    - generictest.building
    - generictest.unit
    - interfacetest
    - multiinterfacetest
    - singletontest
      - Counter.java
      - MainApp.java
    - test
    - test.sub
    - test2
  - test.txt
  - flatbuffer.java.builder.generat

MainApp.java

```java
package singletontest;

public class MainApp {

    public static void main(String[] args) {

        // 불가능!
        Counter counter = new Counter();

        // OK~
        Counter counterStatic = Counter.getInstance();

    }
}
```

Counter.java

```java
package singletontest;

public class Counter {

    private static Counter instance;
    private long value = 0L;

    private Counter() {

    }

    public static Counter getInstance() {
        if (instance == null)
            instance = new Counter();
        return instance;
    }

    public long getValue() {
        return value;
    }

    public void setValue(long value) {
        this.value = value;
    }

    public long addValue(long value) {
        this.value += value;
        return this.value;
    }
}
```

synchronized

synchronized 키워드가 붙은 코드 블록을 임계 영역으로 지정.

- 여러 스레드가 공유 데이터에 상호 배타적으로 접근 할 수 있게 동기화 할 때 사용.

- 스레드가 synchronized 블록 진입 할 때 lock 을 걸고, 블록에서 빠져나올 때 lock을 푼다.
  (여러 스레드가 synchronized 블록에 진입 하려고 하면, 먼저 synchronized 블록에 진입하는 스레드가 lock 을 소유하고, lock 을 풀 때 까지 다른 스레드는 대기한다.)

synchronized

synchronized 키워드가 붙은 코드 블록을 임계 영역으로 지정.

MainApp.java

```java
package bank;

public class MainApp {

    public static void main(String[] args) {

        Person lim = new Person("임꺽정", 100);
        Person hwang = new Person("황진이", 100);

        lim.sendMoney(hwang, 10);

        System.out.println(lim.toString());
        System.out.println(hwang.toString());
    }
}
```

Person.java

```java
package bank;

public class Person {

    private String name;
    private long money;

    public Person(String name, long money) {
        super();
        this.name = name;
        this.money = money;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public long getMoney() {
        return money;
    }
    public void setMoney(long money) {
        this.money = money;
    }

    public void sendMoney(Person target, long money) {
        target.setMoney(target.getMoney() + money);
        this.money -= money;
    }

    @Override
    public String toString() {
        return "Person [name=" + name + ", money=" + money + "]";
    }
```

Problems @ Javadoc Declaration Console Call Hierarchy

```
<terminated> MainApp (6) [Java Application] C:₩Program Files₩RedHat₩java-11-openjdk-11.0.7-1₩bin₩javaw.exe (2020. 5. 29. 오후 12:38:42 – 오후 12:38:43)
Person [name=임꺽정, money=90]
Person [name=황진이, money=110]
```

synchronized

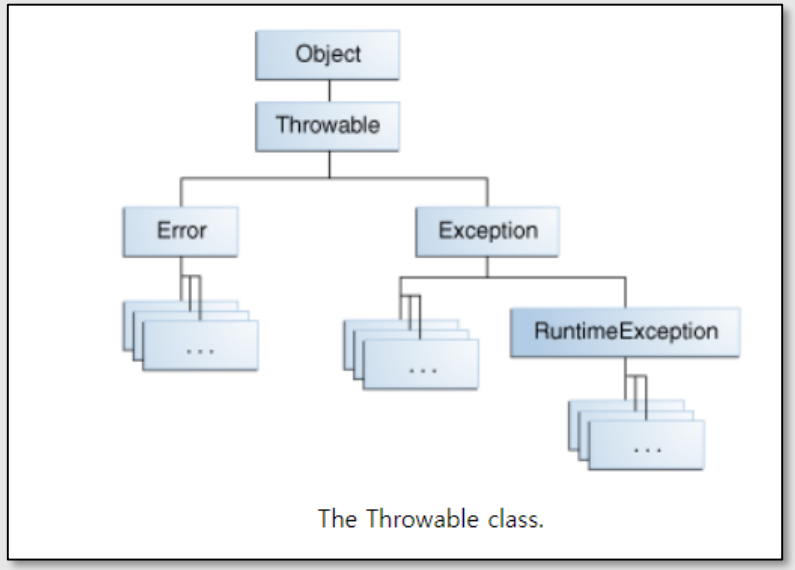
synchronized 키워드가 붙은 코드 블록을 임계 영역으로 지정.

https://github.com/mokdonjr/studyJava/tree/master/Example/src/bank

Exception

Java 프로그램 실행중 정상 실행을 방해하는 이벤트 객체

- Java 는 프로그램 실행중에 발생하는 런타임 에러 발생시 JVM이 Exception을 던진다.
- JVM 에 의해 던져지거나, throw 문에 의해 던져질 수 있다.
- Exception 은 Throwble 를 상속받는다.
  (Throwable 은 Object 를 상속받는다.)



The Throwable class.

https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html

Exception

Java 프로그램 실행중 정상 실행을 방해하는 이벤트 객체

- Object – Throwable – Exception – **RuntimeException**
  RuntimeException 을 상속받는 대표적인 예외 클래스

예외 종류	발생 케이스
ArithmeticException	정수를 0으로 나눌 때 발생
NullPointException	Null 레퍼런스를 참조할 때 발생
ClassCastException	변환할 수 없는 타입으로 캐스팅 할 때 발생
OutOfMemoryError	메모리가 부족한 경우 발생
ArrayIndexOutOfBoundsException	배열 인덱스 범위를 벗어난 접근 시 발생
IllegalArgumentException	잘못된 인자 전달 시 발생
IOException	입출력 동작 실패 또는 인터럽트 시 발생

- 어플리케이션 비즈니스 로직에서 발생하는 예외를 구분하기 위해 Exception 을 상속받아 구현한다.
  (ex. hero_java 프로젝트의 경우 KException 을 구현했음)

Exception – 예외 처리 방법

Java 프로그램 실행중 정상 실행을 방해하는 이벤트 객체

- try ~ catch 블록을 작성해 예외처리가 가능하다.

```
try {
    // 실행 로직
} catch (Exception e) {
    // 예외 발생시 catch 블록이 실행됨
    // 예외가 발생하지 않으면 catch 블록이 실행되지 않음
} finally {
    // 예외가 발생해 catch 블록이 실행된 후 실행됨
    // 예외가 발생하지 않아 catch 블록이 실행되지 않아도 실행됨
}
```

Exception – throw 문 에 의해 던져지는 케이스

throw 문이 실행되면 Exception을 던진다.

MainApp.java

```java
package exceptiontest;

public class MainApp {

    public static void main(String[] args) {
        System.out.println("프로그램이 시작되었습니다.");
        try {
            play();
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("프로그램이 모두 수행되었습니다.");
    }

    private static void play() throws Exception {

        // 마린과 파이어벳 전투 준비!
        Marine marine = new Marine(20);
        Firebat firebat = new Firebat(0);

        // 예외 체크
        if (marine.getHp() <= 0 || firebat.getHp() <= 0)
            throw new Exception("HP 값이 전투에 적합하지 않습니다");

        // 전투 수행!
        marine.damage(firebat);
    }
}
```

```
<terminated> MainApp (1) [Java Application] C:\Program Files\Microsoft\jdk-11.0.7-hotspot\bin\javaw.exe (2020. 5. 29. 오전 9:
java.lang.Exception: HP 값이 전투에 적합하지 않습니다
        at exceptiontest.MainApp.play(MainApp.java:20)
        at exceptiontest.MainApp.main(MainApp.java:8)
프로그램이 시작되었습니다.
프로그램이 모두 수행되었습니다.
```

Exception 이 던져지는 메서드임을 선언하여 호출하는 곳에서 예외 처리를 하지 않으면 컴파일 에러를 발생시키게 한다.

throw 문이 실행되며 Exception 인스턴스를 생성해 던진다.

throw 문이 실행되면 해당 Exception 인스턴스가 던져진다.

" throw 문이 작성된 메서드는 반드시 throws Exception 을 명시해야 하며 throws Exception 이 명시된 메서드를 호출하는 메서드는 반드시 try ~ catch 예외처리를 해야한다. "

Exception – JVM 에 의해 던져지는 케이스

런타임 에러 발생시 JVM이 Exception 을 던진다

MainApp.java

```java
package exceptiontest;

public class MainApp {

    public static void main(String[] args) {
        System.out.println("프로그램이 시작되었습니다.");
        play();
        System.out.println("프로그램이 모두 수행되었습니다.");
    }

    private static void play() {
        // 열거하여 선언된 고정길이 배열
        int[] arr = {1, 2, 3, 4, 5};

        System.out.println("0번째 원소는 : " + arr[0]);
        System.out.println("1번째 원소는 : " + arr[1]);
        System.out.println("2번째 원소는 : " + arr[2]);
        System.out.println("3번째 원소는 : " + arr[3]);
        System.out.println("4번째 원소는 : " + arr[4]);
        System.out.println("5번째 원소는 : " + arr[5]);
    }
}
```

Problems Javadoc Declaration Console Call Hierarchy

```
<terminated> MainApp (4) [Java Application] C:₩Program Files₩RedHat₩java-11-openjdk-11.0.7-1₩bin₩javaw.exe  (2020. 5. 29. 오전 9:16
프로그램이 시작되었습니다.
0번째 원소는 : 1
1번째 원소는 : 2
2번째 원소는 : 3
3번째 원소는 : 4
4번째 원소는 : 5
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5
        at exceptiontest.MainApp.play(MainApp.java:20)
        at exceptiontest.MainApp.main(MainApp.java:7)
```

런타임 에러가 발생해 JVM이 ArrayIndexOutOfBoundException 을 던지며 main 스레드 실행이 종료된다.

Exception – JVM 에 의해 던져지는 케이스

런타임 에러 발생시 JVM이 Exception 을 던진다

MainApp.java

```java
package exceptiontest;

public class MainApp {

    public static void main(String[] args) {
        System.out.println("프로그램이 시작되었습니다.");
        play();
        System.out.println("프로그램이 모두 수행되었습니다.");
    }

    private static void play() {
        // 열거하여 선언된 고정길이 배열
        int[] arr = {1, 2, 3, 4, 5};

        try {
            System.out.println("0번째 원소는 : " + arr[0]);
            System.out.println("1번째 원소는 : " + arr[1]);
            System.out.println("2번째 원소는 : " + arr[2]);
            System.out.println("3번째 원소는 : " + arr[3]);
            System.out.println("4번째 원소는 : " + arr[4]);
            System.out.println("5번째 원소는 : " + arr[5]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("배열에서 선언되지 않은 인덱스를 참조하는 오류가 발생!!");
        }
    }
}
```

try 블록 내에서 예외가 발생하면 catch 블록이 실행된다.

Problems Javadoc Declaration Console Call Hierarchy

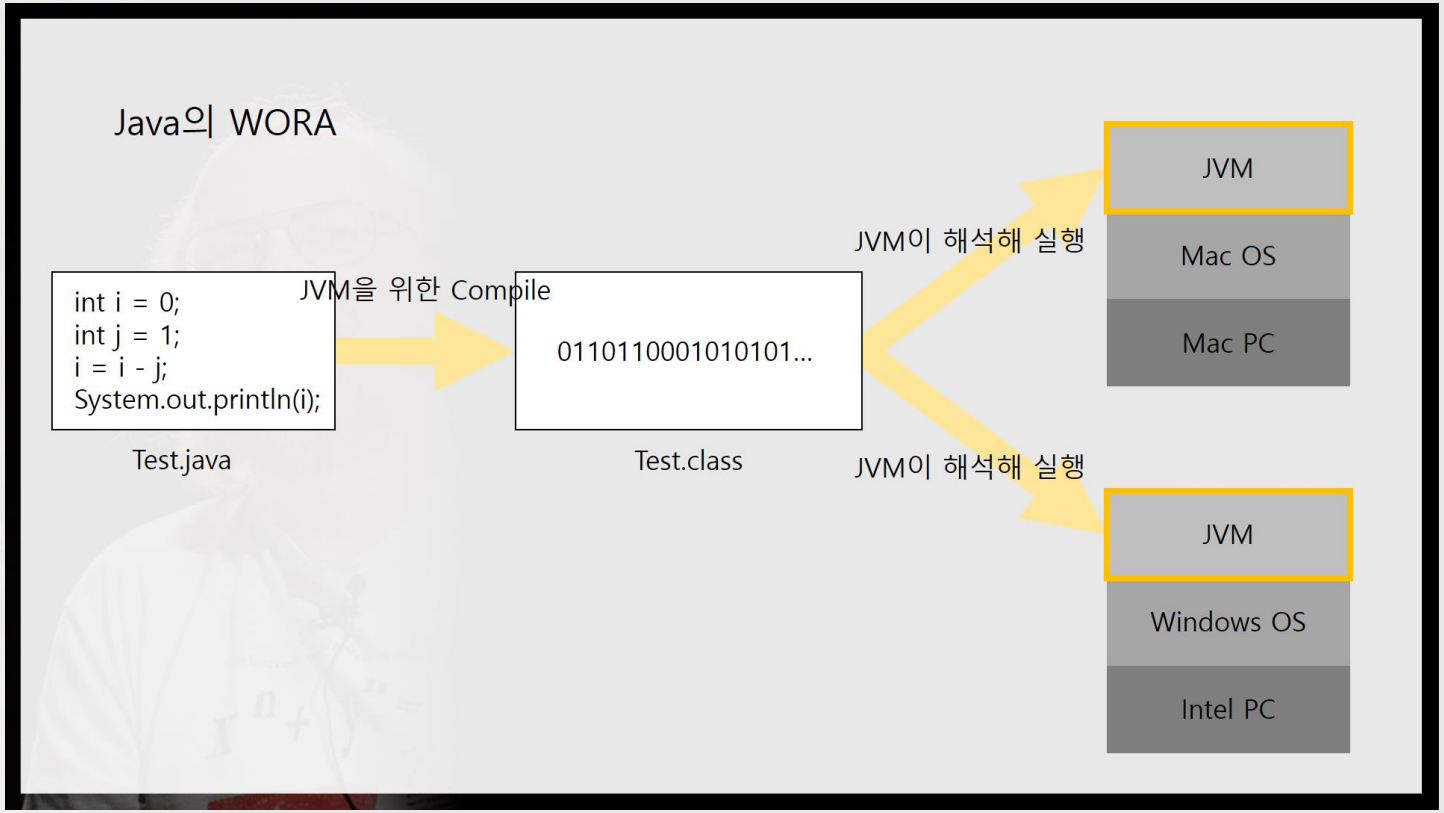
```
<terminated> MainApp (4) [Java Application] C:₩Program Files₩RedHat₩java-11-openjdk-11.0.7-1₩bin₩javaw.exe  (2020. 5. 29. 오전 9:1
프로그램이 시작되었습니다.
0번째 원소는 : 1
1번째 원소는 : 2
2번째 원소는 : 3
3번째 원소는 : 4
4번째 원소는 : 5
배열에서 선언되지 않은 인덱스를 참조하는 오류가 발생!!
프로그램이 모두 수행되었습니다.
```

JVM 메모리 관리

자바 가상 머신 (JVM, Java Virtual Machine)

자바 바이트코드를 실행

- 머신과 어플리케이션 사이에 위치해 바이트코드로 컴파일된 자바 프로그램을 실행 (플랫폼 독립)



JVM 메모리 관리

자바 프로세스 메모리 관리 책임을 기계에 위임

- 제로-오버헤드 vs 비즈니스 생산성을 위한 저수준 제어 간 선택의 배경
- 프로그래머가 malloc, free 함수 직접 호출하지 않고 JVM이 메모리를 관리
  사람이 관리하면 더 확정적인 성능을 낼 수 있으나, 사람이 계산하는 과정을 자동화
- 자바 Heap 메모리 해제는 오직 GC를 통해서만 수행된다.

가비지 컬렉션 (GC, Garbage Collection)

자바 Heap 영역 재사용을 위한 메커니즘

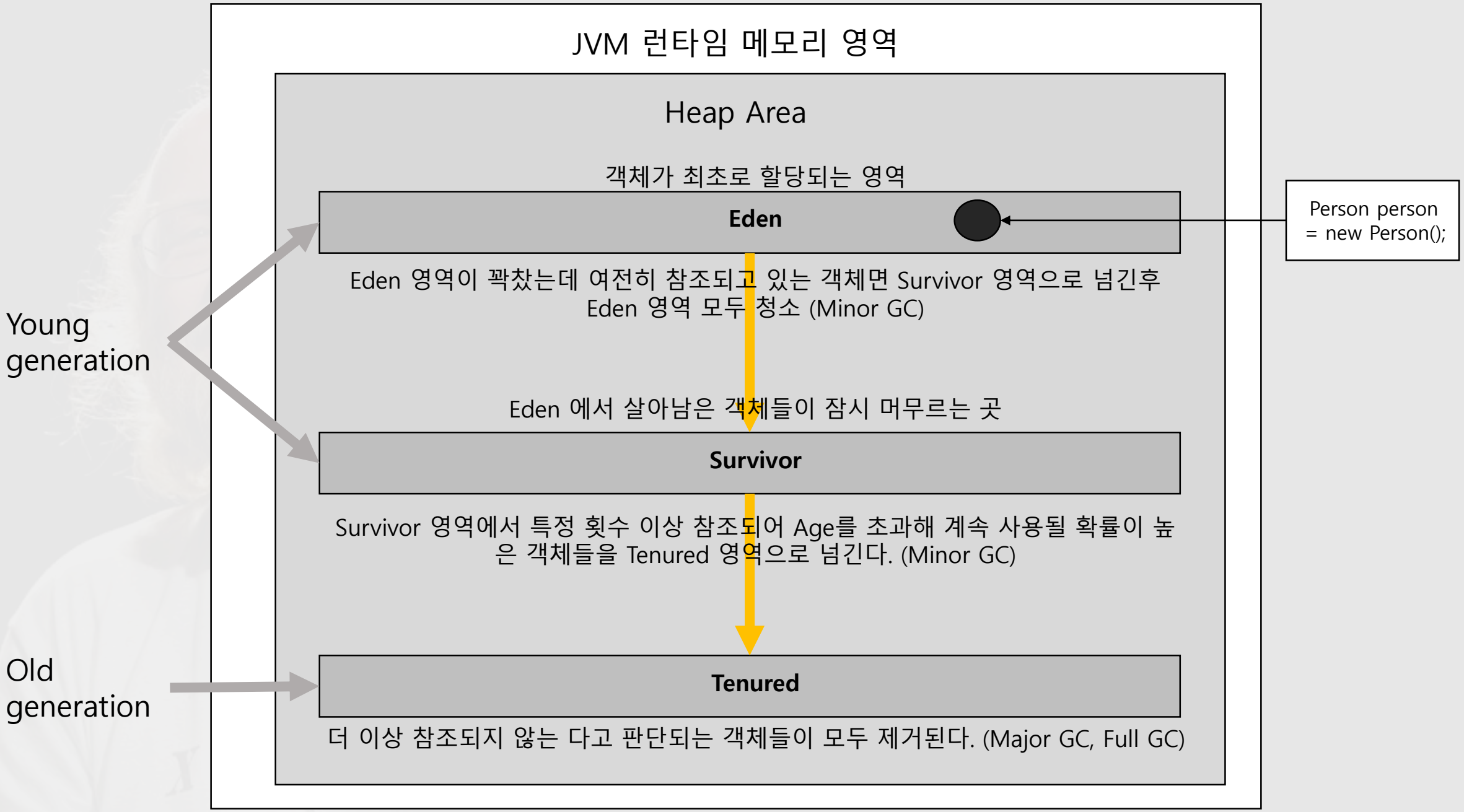
- new 키워드를 통해 JVM이 적절한 양의 자바 Heap 메모리를 할당하며, 메모리가 충분치 않으면 JVM이 GC를 실행
  (그래도 메모리가 충분치 못하면 OutOfMemoryError 가 발생하며 JVM이 종료된다)

- 더 이상 참조되지 않는 메모리를 찾아 반환하는 것이 목적

- 참조되는 객체에 live 표기를 하고 Heap 영역에 있는 모든 객체를 찾아다니며 live 표기가 없는 메모리를 재사용할 수 있도록 한다 mark-and-sweep
  (이 과정에서 JVM 의 모든 스레드가 정지되며 stop-the-world, 이 시간을 최소화하기 위해 변화를 거듭중)

- Eden -> Survivor -> Tenured 순서의 제너레이션 영역으로 메모리를 재배치

- 가능한 많은 여유 공간을 남겨두기 위해 자주 접근되는 객체들을 묶음. Compaction


JVM 런타임 메모리 영역
Heap Area
객체가 최초로 할당되는 영역
Eden
Person person
= new Person();
Eden 영역이 꽉찼는데 여전히 참조되고 있는 객체면 Survivor 영역으로 넘긴후
Eden 영역 모두 청소 (Minor GC)
Eden 에서 살아남은 객체들이 잠시 머무르는 곳
Survivor
Survivor 영역에서 특정 횟수 이상 참조되어 Age를 초과해 계속 사용될 확률이 높
은 객체들을 Tenured 영역으로 넘긴다. (Minor GC)
Tenured
더 이상 참조되지 않는 다고 판단되는 객체들이 모두 제거된다. (Major GC, Full GC)
Young
generation
Old
generation

메모리 튜닝

JVM의 각 메모리 영역의 크기를 지정할 수 있게 매개변수를 제공

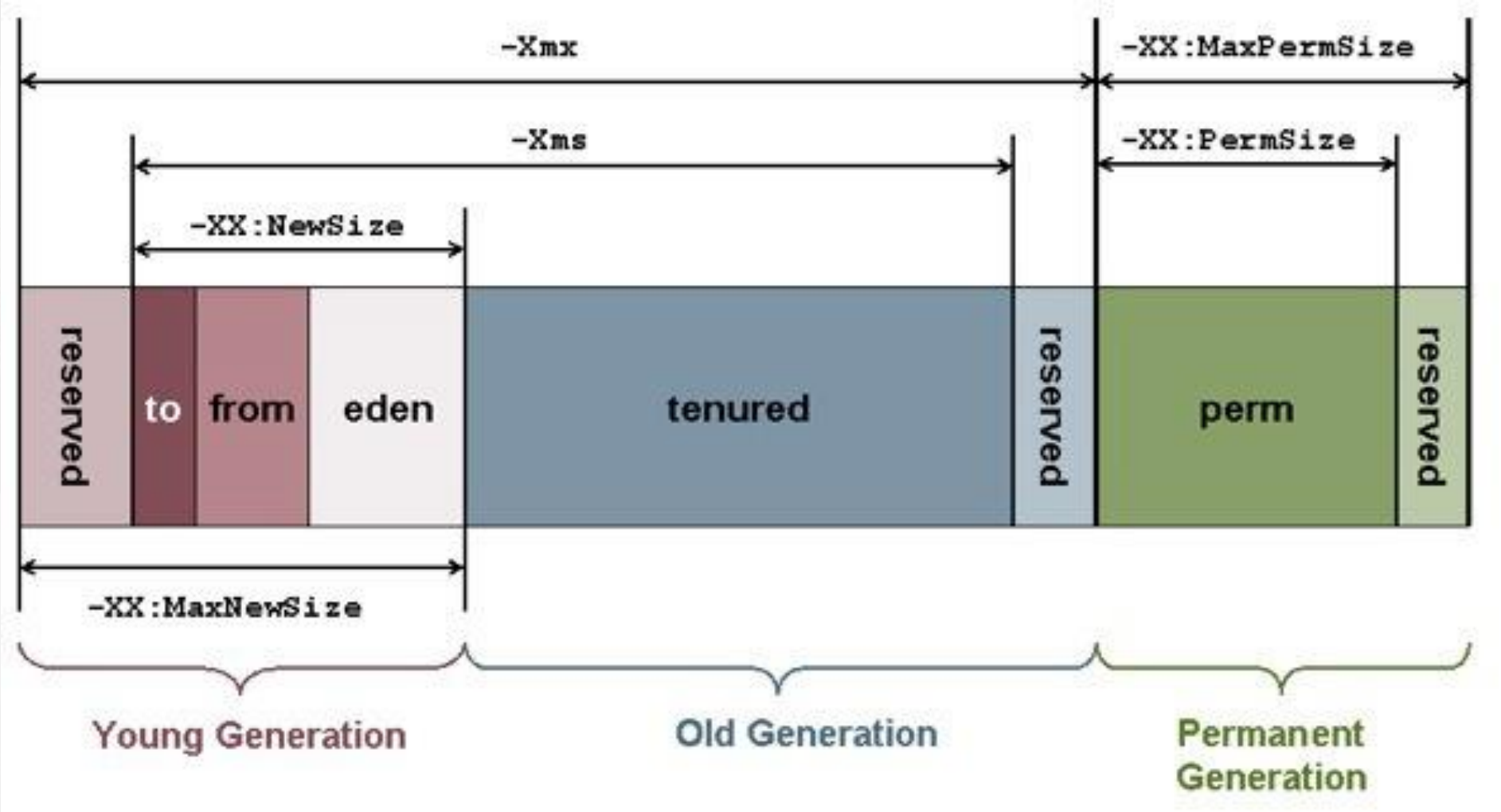
- 메모리 관리 자동화의 장점이 있지만, GC 수행하는 순간 모든 스레드가 동작을 멈춘다.
  상호 참조를 줄이는 등 코드 최적화 후 GC 수행 시간을 줄이기 위한 메모리 튜닝이 가능하다.

- -Xmx 최대 Heap 크기
  (기본값은 1GB 이하, 머신 전체 메모리의 4분의1)
- -Xms 초기 Heap 크기
  (-Xmx, -Xms 값이 같다면 JVM이 운영체제에게 모든 메모리를 할당하도록 요청. 단, 더 이상 메모리 용량이 커지지는 않는다.)

- 어플리케이션에 맞는 GC 를 선택 (hero_java java11 g1gc 선택)

```
java -XX:+UnlockDiagnosticVMOptions -XX:+PrintFlagsFinal -version
```

메모리 튜닝

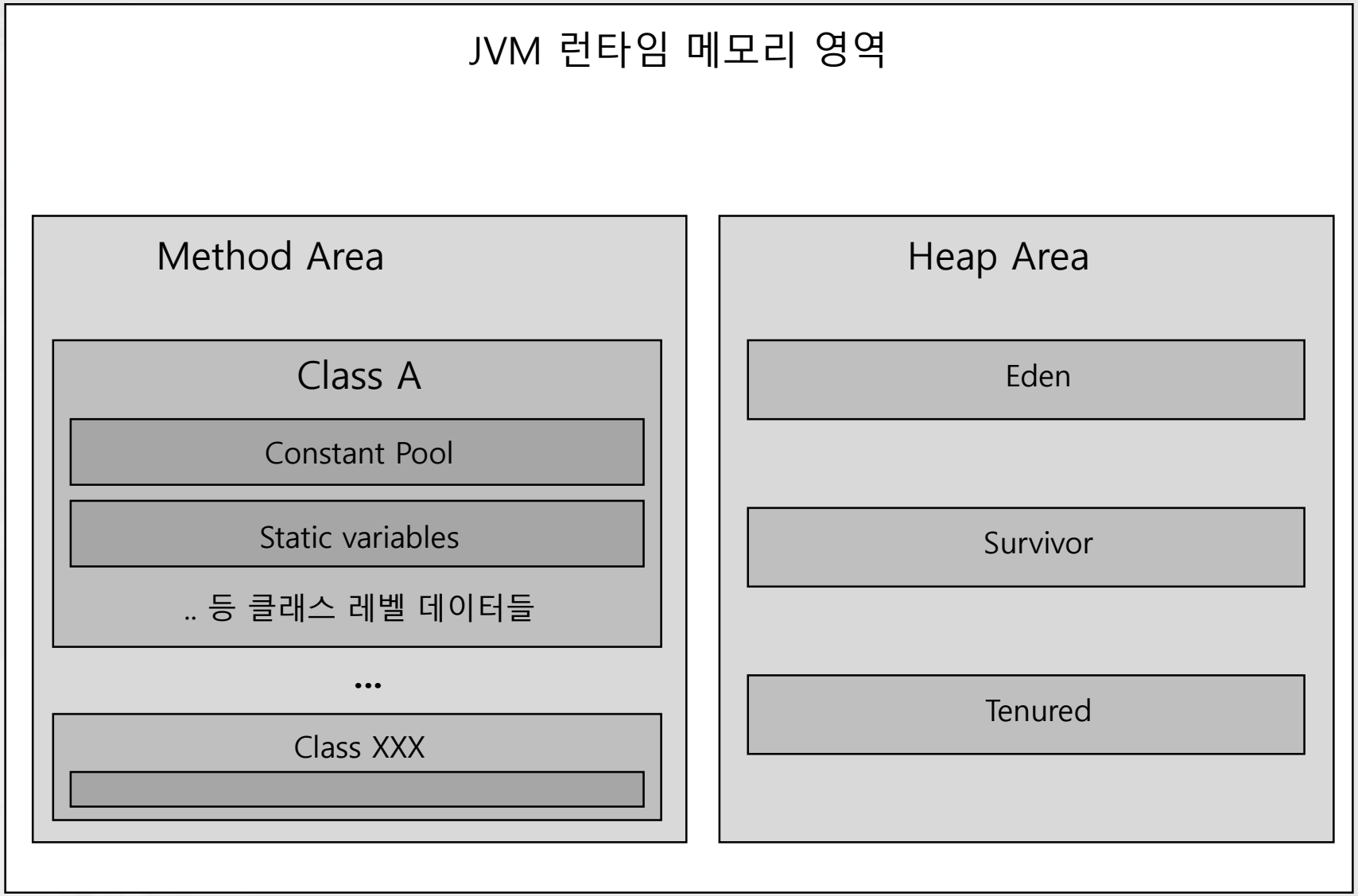

-Xmx
-XX:MaxPermSize
-Xms
-XX:PermSize
-XX:NewSize
reserved
to
from
eden
tenured
reserved
perm
reserved
-XX:MaxNewSize
Young Generation
Old Generation
Permanent Generation

클래스 로더 (Class Loader)

런타임에 Java 클래스를 JVM에 동적으로 로드하는 객체

- 컴파일러(javac)가 소스파일(.java)을 플랫폼 독립적인 기계어인 바이트코드(.class) 로 변환 후, JVM 이 실행되며 자바 프로세스의 클래스로더가 바이트코드를 읽는다.

- 최초에 모든 클래스 로더들의 부모인 bootstrap 클래스가 최소한의 필수 클래스들 특히 자바 런타임 코어 클래스 java.lang.Object, Class, ClassLoader 등을 로드 후, 자바 프로세스의 진입점인 main 메서드에 작업을 넘긴다.

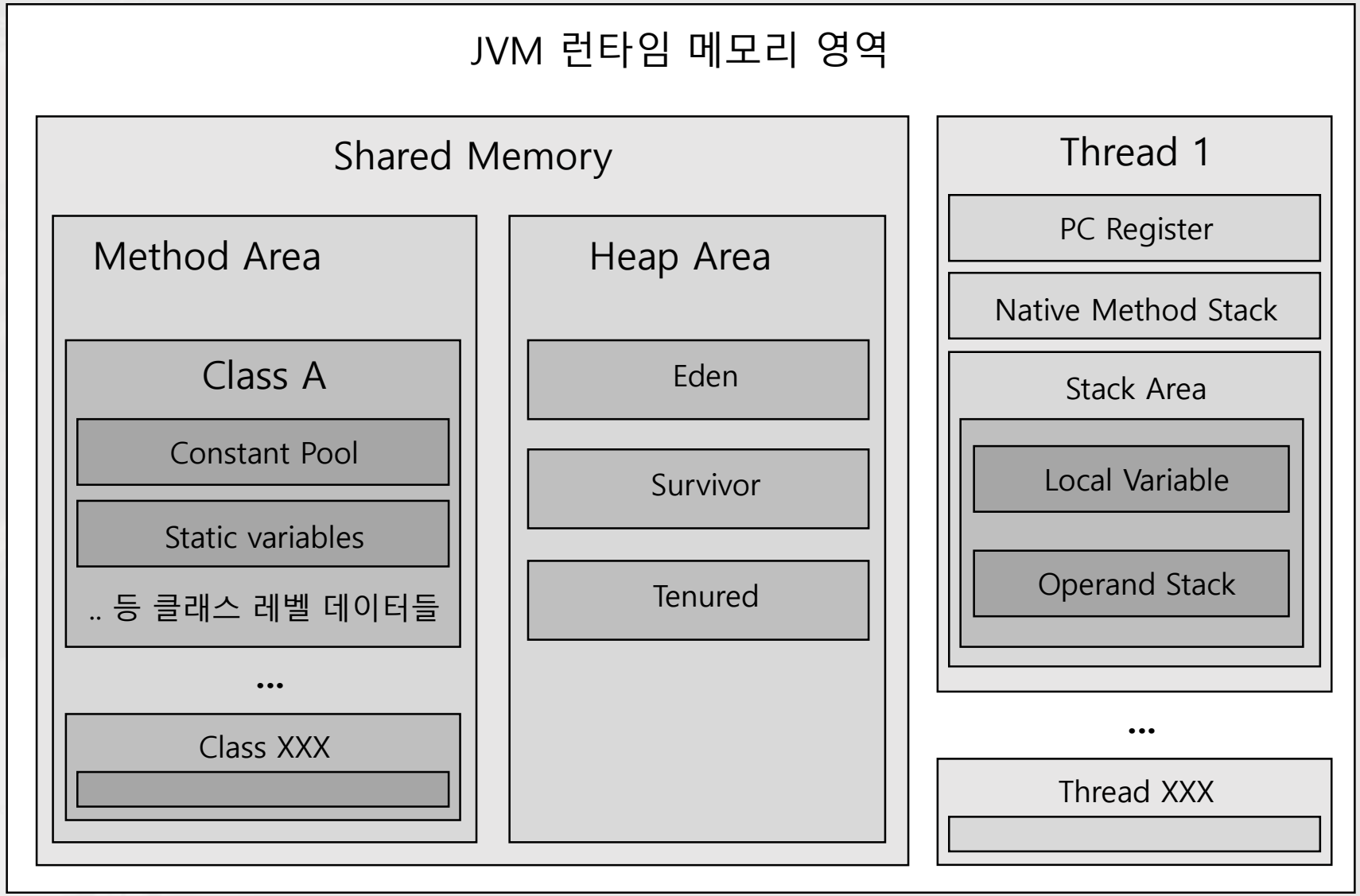

JVM 런타임 메모리 영역
Method Area
Class A
Constant Pool
Static variables
.. 등 클래스 레벨 데이터들
...
Class XXX
Heap Area
Eden
Survivor
Tenured

Thread

런타임 에러 발생시 JVM이 Exception 을 던진다

- 자바 프로세스의 모든 스레드는 가비지가 수집되는 공용 Heap을 가지며, 한 스레드가 생성한 객체를 다른 스레드가 접근할 수 있다.
  (Shared Memory)

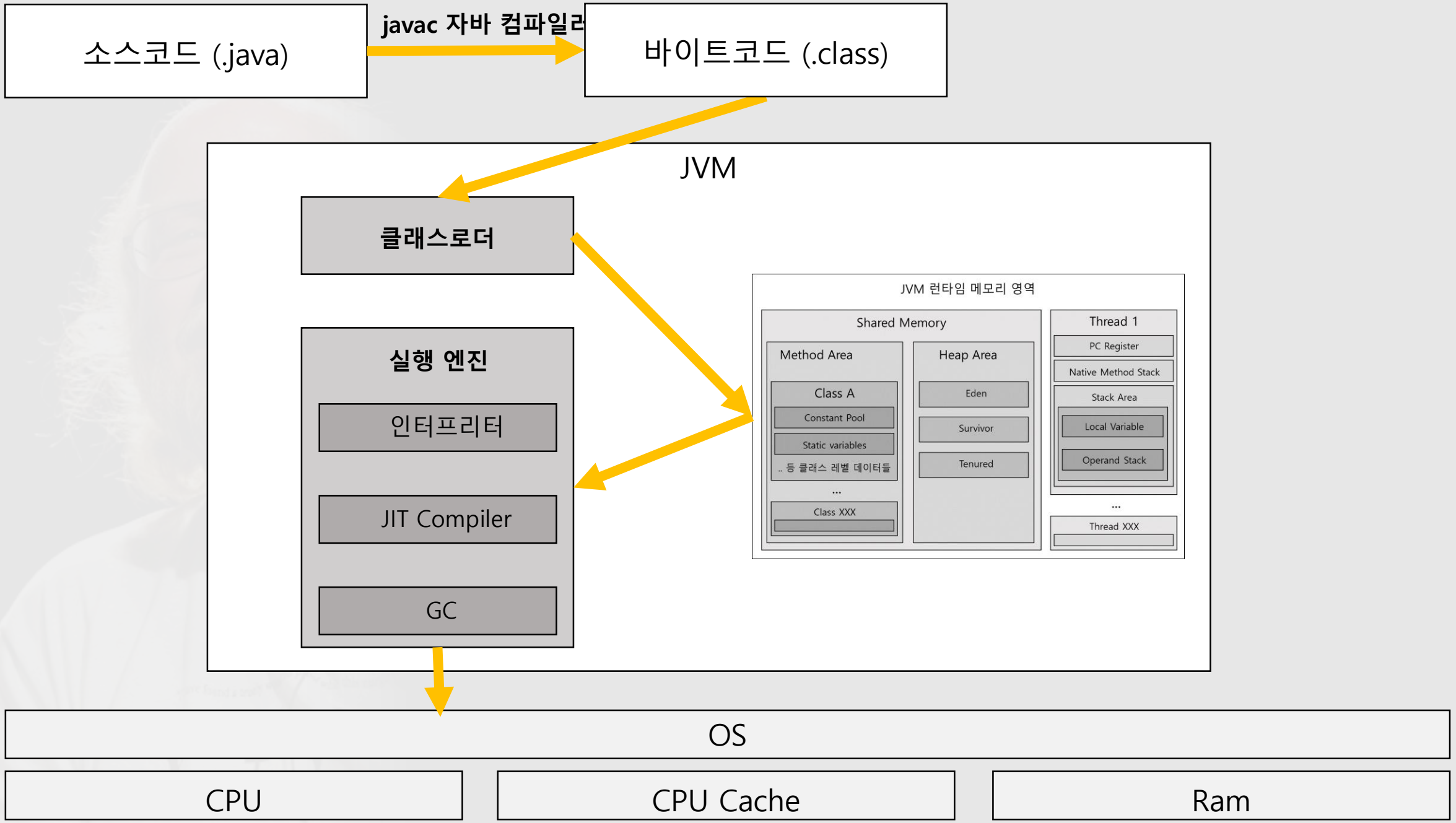
- 자바 프로세스의 모든 스레드는 정확히 하나의 전용 OS 스레드에 대응한다.


JVM 런타임 메모리 영역
Shared Memory
Method Area
Class A
Constant Pool
Static variables
.. 등 클래스 레벨 데이터들
...
Class XXX
Heap Area
Eden
Survivor
Tenured
Thread 1
PC Register
Native Method Stack
Stack Area
Local Variable
Operand Stack
...
Thread XXX

JIT Compiler (just-in-time compiler)

런타임에 기계어로 컴파일 (동적 번역)

- 바이트코드를 네이티브 코드로 컴파일한다.
- JVM이 프로그램을 모니터링하면서 가장 자주 실행되는 코드를 분석해 JIT Compiler 를 수행 및 최적화
- 인터프리트 방식과 정적 컴파일 방식의 혼용
  (javac 자바 컴파일러가 소스코드를 바이트코드로 정적 컴파일 후 JVM 내 인터프리터가 바이트코드를 해석하고 jit compiler 가 동적으로 최적화하며 컴파일)


소스코드 (.java)
javac 자바 컴파일러
바이트코드 (.class)
JVM
클래스로더
실행 엔진
인터프리터
JIT Compiler
GC
JVM 런타임 메모리 영역
Shared Memory
Method Area
Class A
Constant Pool
Static variables
.. 등 클래스 레벨 데이터들
...
Class XXX
Heap Area
Eden
Survivor
Tenured
Thread 1
PC Register
Native Method Stack
Stack Area
Local Variable
Operand Stack
...
Thread XXX
OS
CPU
CPU Cache
Ram

```
int i = 0;
int j = 1;
i = i - j;
System.out.println(i);
```

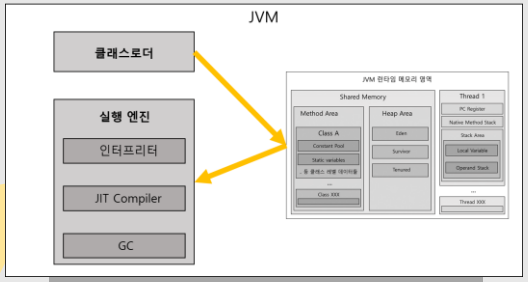
Test.java

javac 컴파일러가 바이트코드로
JVM을 위한 Compile

0110110001010101...

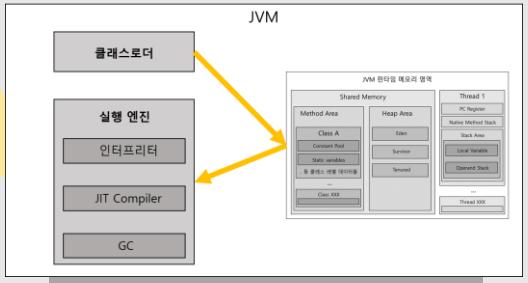
Test.class

바이트코드를 클래스로더로 읽고
실행엔진을 통해 동적 컴파일하며
JVM이 실행



Mac OS

Mac PC



Windows OS

Intel PC

Java 빌드

5주차 – Spring 프레임워크

Appendix

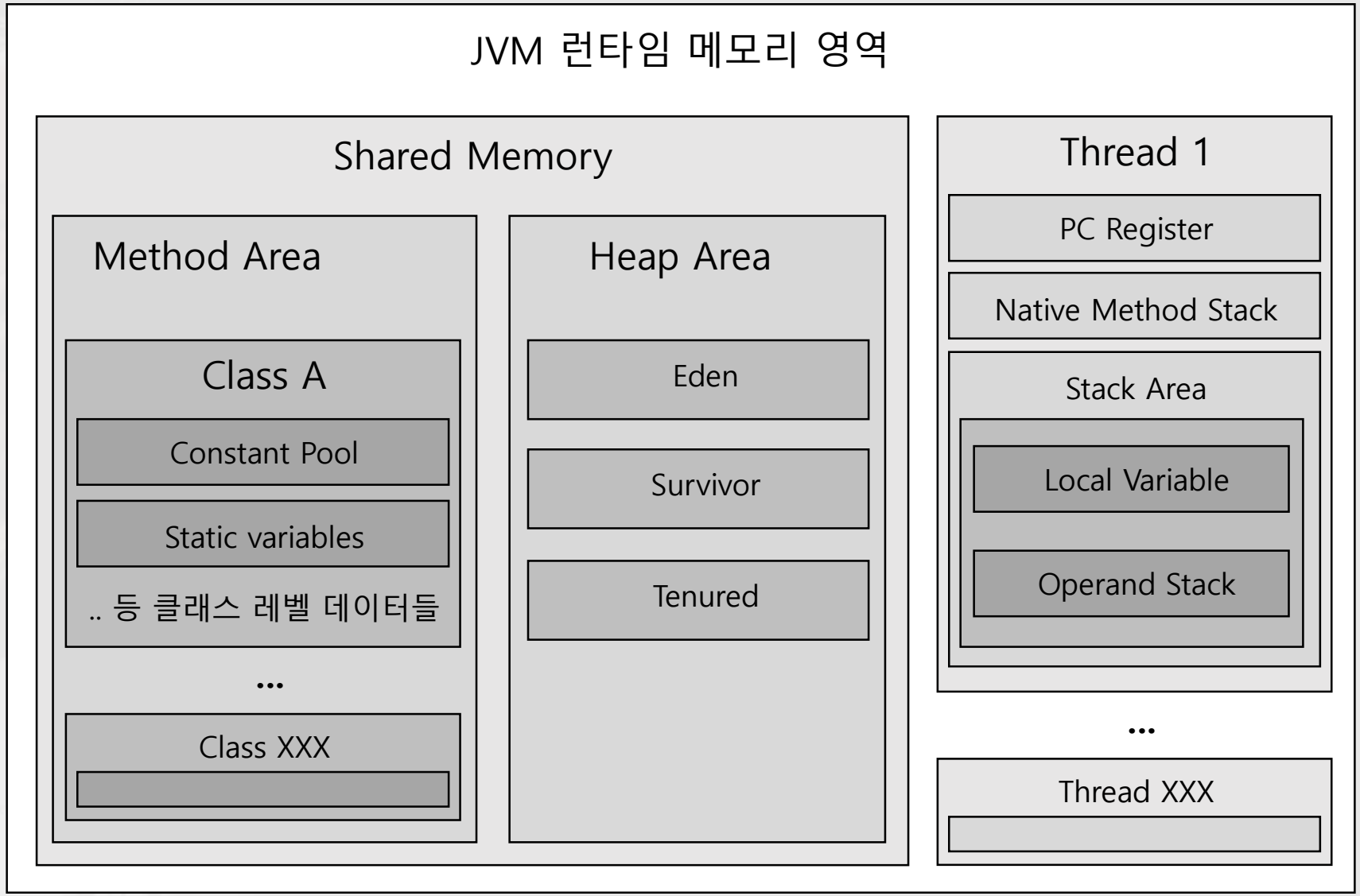
- Java 예약어
  abstract, assert, boolean, break, byte, case, catch, char, class, const, continue, default, do, double, else, enum, extends, final, finally, float, for, if, goto, implements, import, instanceof, int, interface, long, native, new, package, private, protected, public, return, short, static,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

- Android 에서 jvm 사용?
  안함. Dalvik 이라는 전용 가상머신

- HotSpot Runtime Overview
  http://openjdk.java.net/groups/hotspot/docs/RuntimeOverview.html

- Jvm Spec
  https://docs.oracle.com/javase/specs/

- Java api
  https://docs.oracle.com/en/java/javase/11/docs/api/index.html


JVM 런타임 메모리 영역
Shared Memory
Method Area
Class A
Constant Pool
Static variables
.. 등 클래스 레벨 데이터들
...
Class XXX
Heap Area
Eden
Survivor
Tenured
Thread 1
PC Register
Native Method Stack
Stack Area
Local Variable
Operand Stack
...
Thread XXX