

삼성 청년 SW 아카데미

Java

객체지향 프로그래밍

- 접근제한자
- JVM 메모리구조
- 객체배열관리

접근 제한자

✓ 패키지

- PC의 많은 파일을 관리하기 위해서 폴더를 이용한다.
- 프로그램의 많은 클래스를 관리하기 위해서 패키지를 이용한다.
- 패키지는 클래스와 관련 있는 인터페이스들을 모아두기 위한 이름 공간.
- 패키지의 구분은 .(dot) 연산자를 이용한다.
- 패키지의 이름은 시중에 나와 있는 패키지들과 구분되게 지어야 한다.
- 일반적으로 소속이나 회사의 도메인을 사용한다.

`com.ssafy.project_이름.module_이름`

✓ 임포트

- 다른 패키지에 있는 클래스를 사용하기 위해서는 import 과정이 필요하다.

```
com.ssafy.project.dto
> Person.java
com.ssafy.project.service
> PersonService.java
```

```
package com.ssafy.project.service;

import com.ssafy.project.dto.Person;

public class PersonService {
    Person p;
}
```

- PersonService.java에서 Person 클래스를 사용하기 위해서는 import 해야 한다.
- import를 선언 할 때는 import 키워드 뒤에 package 이름과 클래스 이름을 모두 입력하거나, 해당 패키지의 모든 클래스를 포함할 때는 '*' 를 사용하기도 한다.

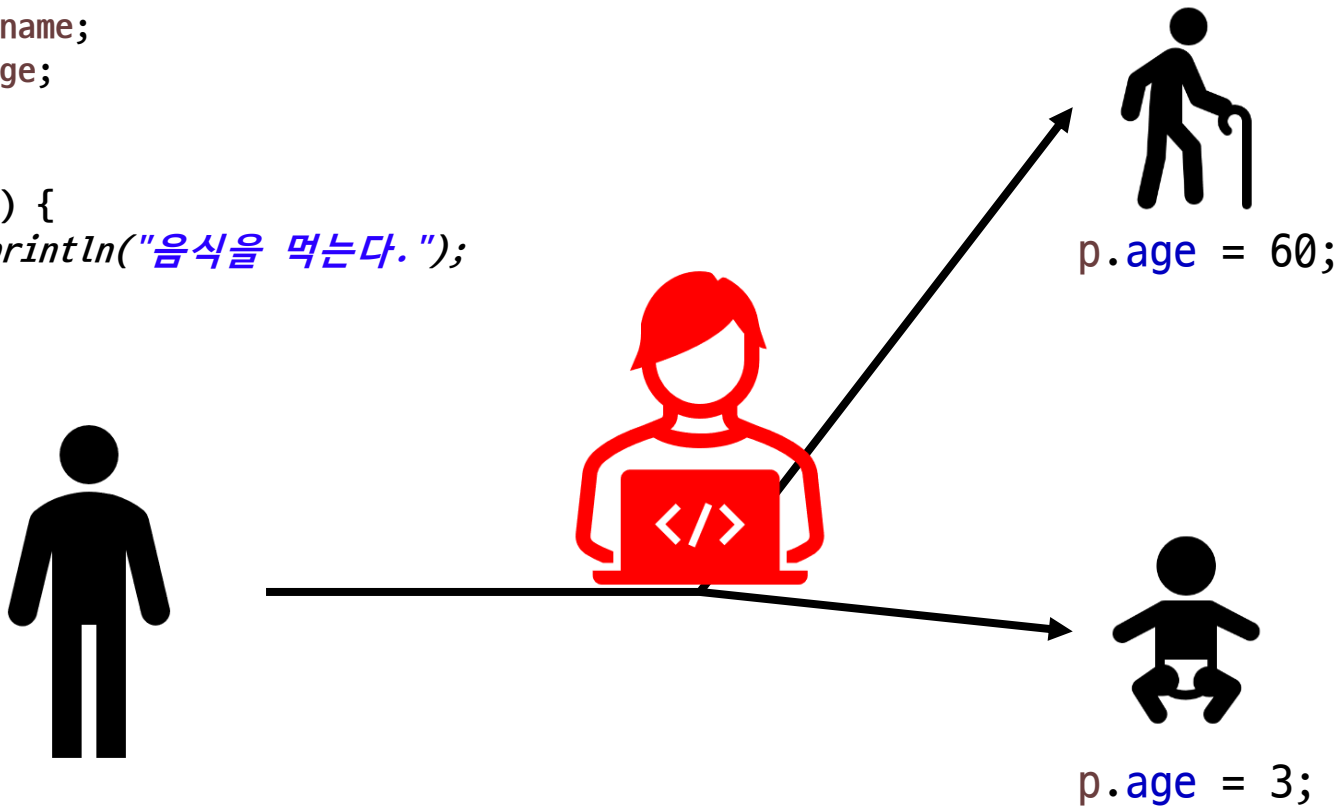
```
import package_name.class_name;
import package_name.*;
```

캡슐화(Encapsulation)

Confidential

```
public class Person {  
    public String name;  
    public int age;  
  
    public Person(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    public void eat() {  
        System.out.println("음식을 먹는다.");  
    }  
}
```

✓ 이런 일이 가능한 이유는 무엇일까??

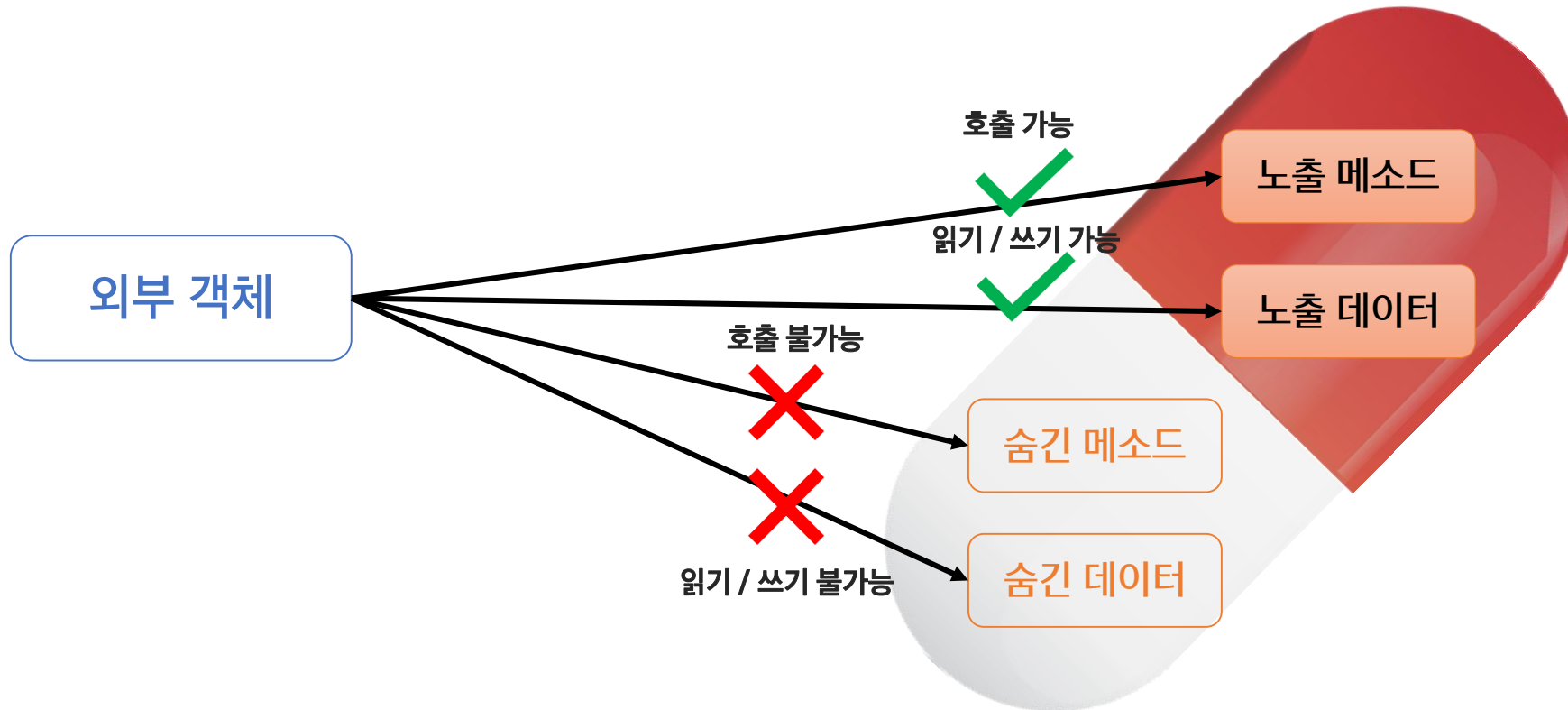


```
Person p = new Person("김싸피", 28);
```

캡슐화(Encapsulation)

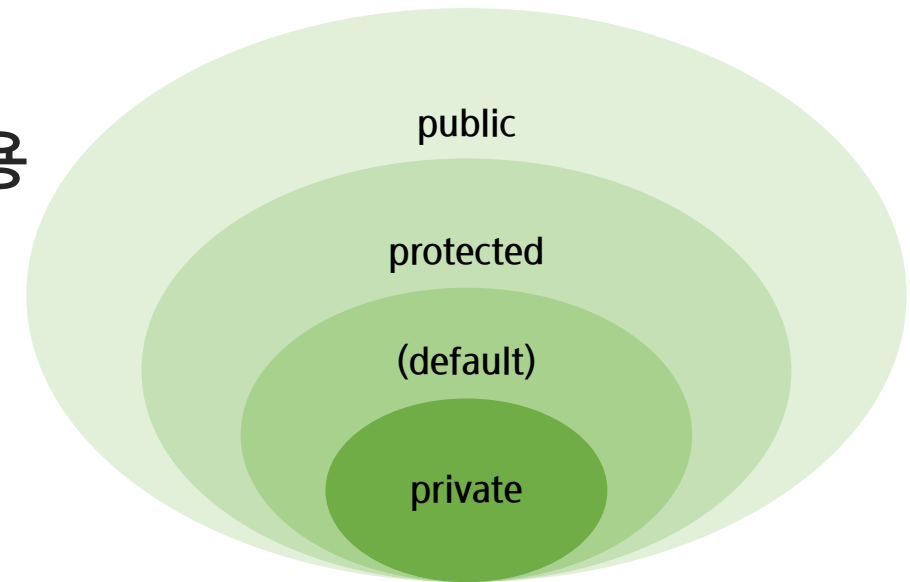
Confidential

- ✓ 객체의 속성(data fields)과 행위(메서드, methods)를 하나로 묶고
- ✓ 실제 구현 내용 일부를 외부에 감추어 은닉한다



- ✓ 클래스, 멤버 변수, 멤버 메서드 등의 선언부에서 접근 허용 범위를 지정하는 역할의 키워드 이다.
- ✓ 접근 제한자의 종류
 - public
 - protected
 - (default)
 - private
- ✓ 그 외 제한자
 - static : 클래스 레벨의 요소 설정
 - final : 요소를 더 이상 수정할 수 없게 함
 - abstract : 추상 메서드 및 추상 클래스 작성
 - ...

- ✓ public : 모든 위치에서 접근이 가능
- ✓ protected : 같은 패키지에서 접근이 가능, 다른 패키지 접근 불가능
단, 다른 패키지의 클래스와 상속관계가 있을 경우 접근 가능
- ✓ default : 같은 패키지에서만 접근이 허용
접근제한자가 선언이 안 되었을 경우 기본 적용
- ✓ private : 자신 클래스에서만 접근이 허용
- ✓ 클래스(외부) 사용가능 : public, default
- ✓ 내부클래스, 멤버변수, 메소드 사용가능 : 4가지 모두 가능



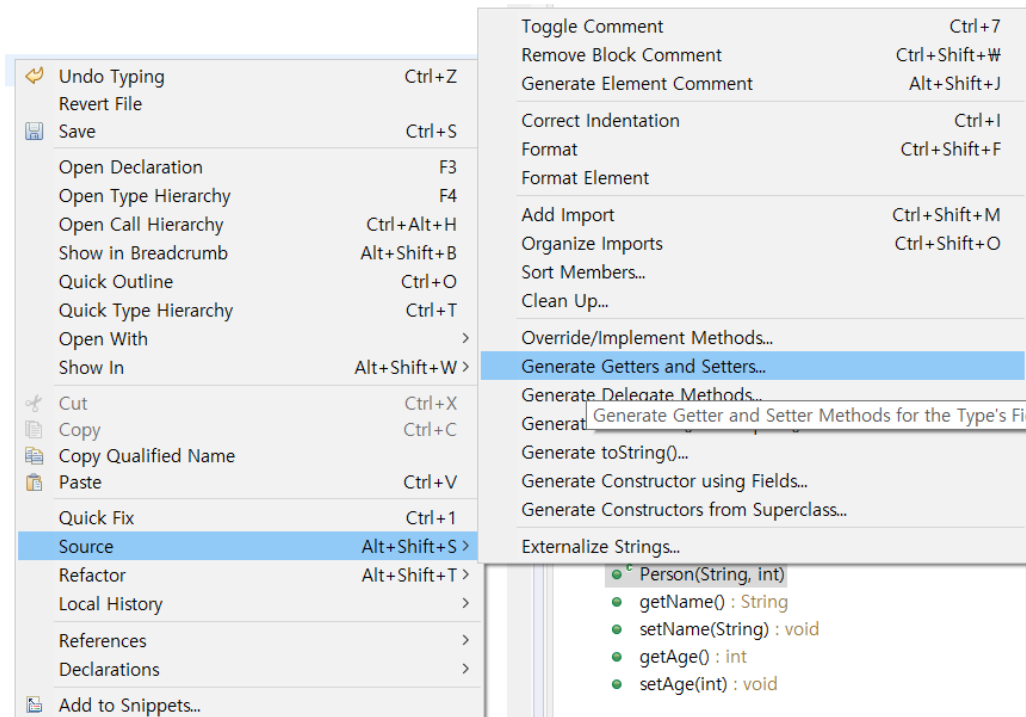
수식어	클래스 내부	동일 패키지	(다른 패키지내의) 하위 클래스	다른 패키지
private	O			
(default)	O	O		
protected	O	O	O	
public	O	O	O	O

접근자(getter) / 설정자(setter)

Confidential

- 클래스에서 선언된 변수 중 접근제한에 의해 접근할 수 없는 변수의 경우 다른 클래스에서 접근할 수 없기 때문에, 접근하기 위한 메서드(설정자와 접근자)를 public으로 선언하여 사용

```
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;  
    }  
}
```



자동 완성 기능 제공

- ✓ 소프트웨어 디자인 패턴에서 싱글턴 패턴(Singleton pattern)을 따르는 클래스는, 생성자가 여러 차례 호출되더라도 실제로 생성되는 객체는 하나이고 최초 생성 이후에 호출된 생성자는 최초의 생성자가 생성한 객체를 리턴

```
public class Manager {  
  
    private static Manager manager = new Manager();  
  
    private Manager() {}  
  
    public static Manager getManager() {  
        return manager;  
    }  
}
```

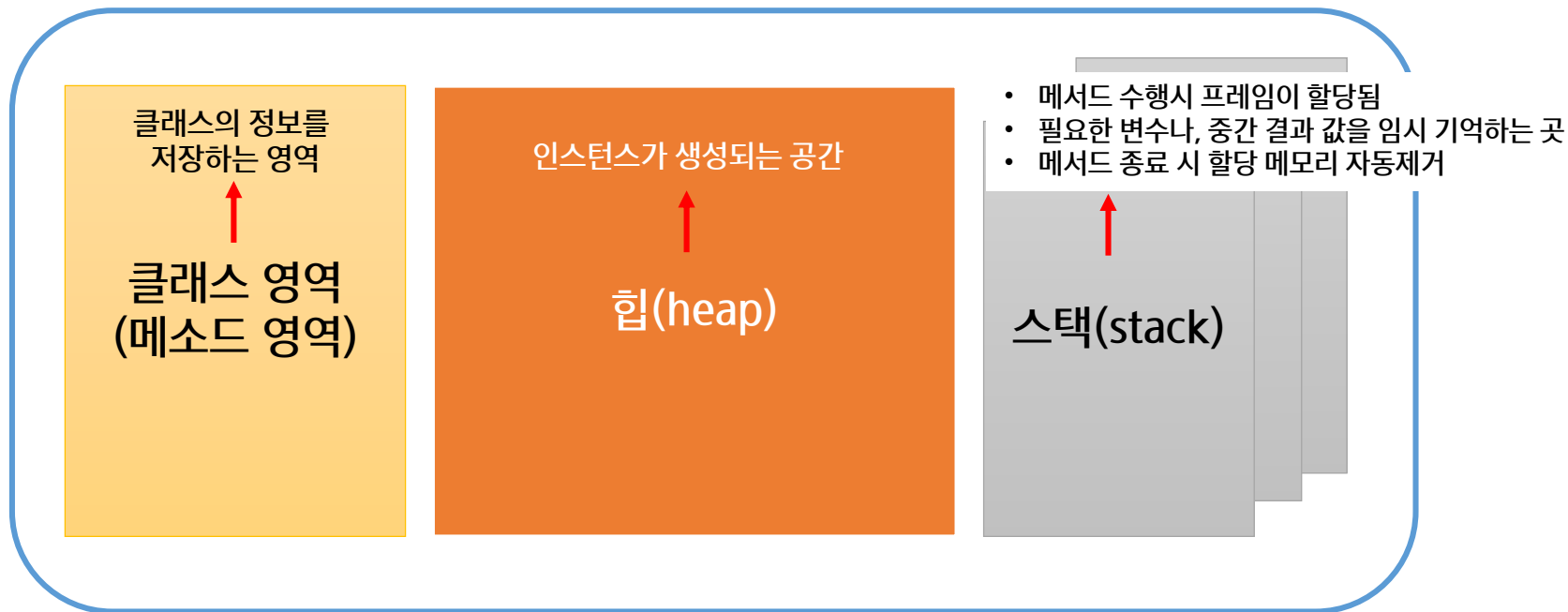
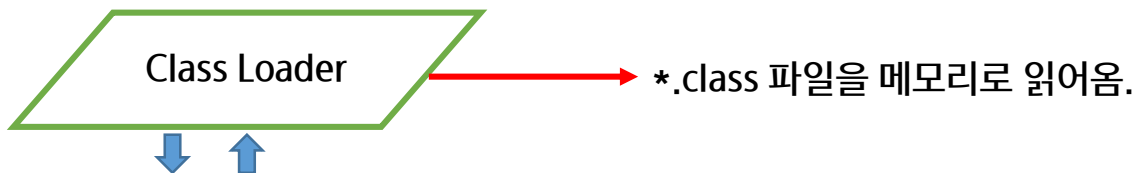
JVM 메모리 구조

✓ JVM 메모리 구조

- Java 언어는 메모리 관리를 개발자가 하지 않음.
- GC(Garbage Collection)가 메모리 관리

Garbage Collection

- Heap 영역(class 영역 포함)에 생성된 메모리 관리 담당
- 더 이상 사용되지 않는 객체들을 점검하여 제거
- 자동적 실행 / CPU가 한가 or 메모리 부족
- JVM에 의해서 실행
- System.gc()를 통해 호출 (시스템 영향을 줘서 하지 않기...)



✓ 객체 생성과 메모리 할당

```
Person p1 = new Person();  
p1.name = "Yang";  
p1.age = 45;  
p1.hobby = "유튜브";
```

클래스 영역
(메소드 영역)

힙(heap)

스택(stack)

✓ 1. 로딩 시점

- static : 클래스 로딩 시
- non-static : 객체 생성시

✓ 2. 메모리상의 차이

- static : 클래스당 하나의 메모리 공간만 할당
- non-static : 인스턴스 당 메모리가 별도로 할당

✓ 3. 문법적 특징

- static : 클래스 이름으로 접근
- non-static : 객체 생성 후 접근

```
public class Person {  
    static int pCount;  
  
    String name;  
    int age;  
    String hobby;  
}
```

```
public class PersonTest {  
    public static void main(String[] args) {  
        Person p = new Person();  
        p.name = "Kim";  
  
        Person.pCount++;  
  
        p.pCount++; //오류는 나지 않지만 경고  
    }  
}
```

✓ 4. static 영역에서는 non-static 영역을 직접 접근이 불가능

```
public class Main {  
  
    String str = "문장";  
  
    public static void main(String[] args) {  
        System.out.println(str);  
    }  
}
```

✓ 5. non-static 영역에서는 static 영역에 대한 접근이 가능

```
public class Main {  
  
    static String str = "문장";  
  
    public void print() {  
        System.out.println(str);  
    }  
}
```

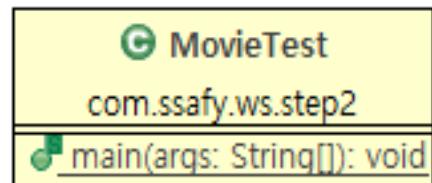
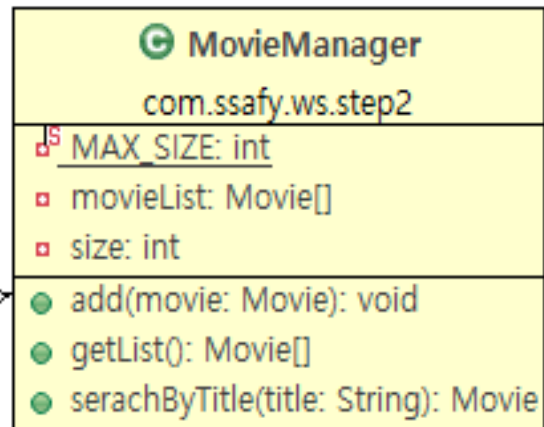
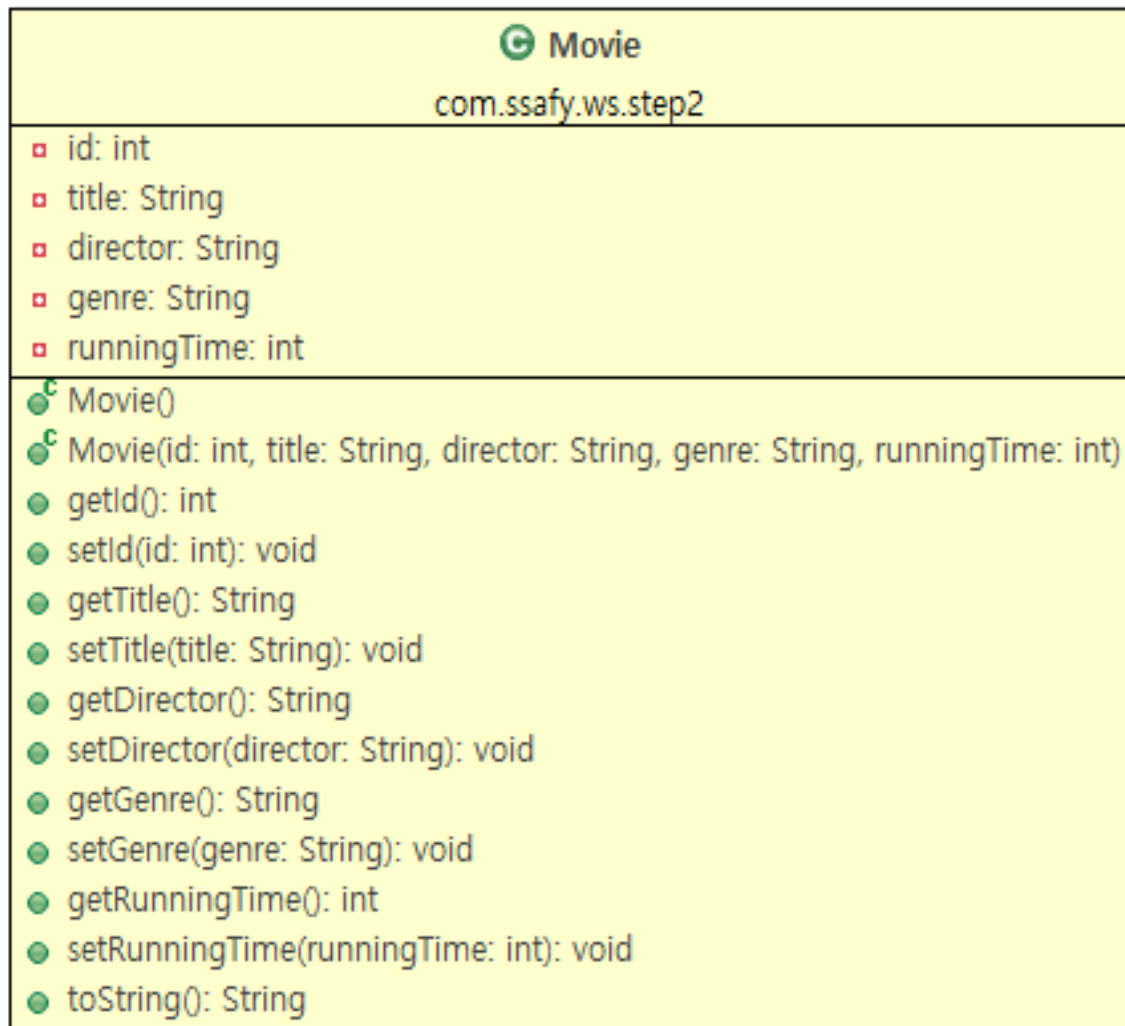
객체 배열 관리

✓ 객체 배열 관리란?

- 정보 관리 시스템 ex) 학사 관리 시스템
- 캡슐화를 이용하여 클래스 작성
- DB 대신 배열을 사용해 객체의 정보를 저장
- 객체의 조회, 추가, 수정, 삭제(CRUD)를 구현
- 싱글턴 패턴을 사용하여 정보 관리 일원화

✓ 영화 관리 (실습 5-2)

- 1. Movie 클래스에 외부에서 필드값을 바꾸지 못하게 접근 지정자를 설정한다.
- 2. Movie 클래스에 영화 정보를 문자열로 반환하는 toString 메소드를 생성한다.
- 3. 영화리스트를 배열로 관리할 MovieManager 클래스를 생성한다. (영화 정보는 최대 100개를 넘지 않는다.)
- 4. MovieManager 클래스에 영화정보를 등록하는 add 메소드를 생성한다.
- 5. MovieManager 클래스에 영화리스트를 반환하는 getList 메소드를 생성한다.
- 6. MovieManager 클래스에 영화제목으로 영화정보를 반환하는 searchByTitle 메소드를 생성한다.
- 7. MovieTest.java 를 생성한다.
 - A. 2개의 영화를 생성하고 출력이 잘 되는지 확인한다.
 - B. MovieManger 객체를 생성하여 영화를 앞서 생성한 영화를 등록하고 출력해본다.
 - C. MovieManger 객체를 이용하여 영화제목 검색을 해본다.



다음 방송에서 만나요!

삼성 청년 SW 아카데미