

# 객체

강 지 훈

*jhkang@cnu.ac.kr*

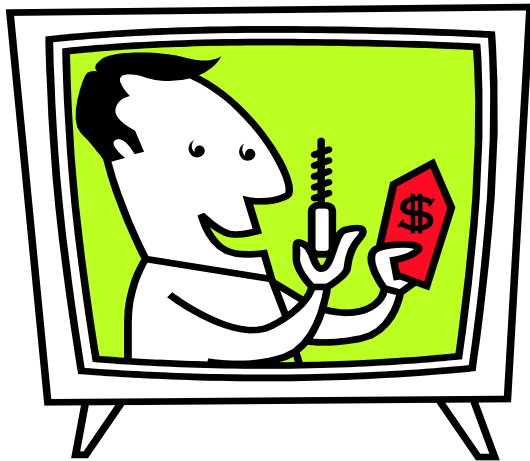


# 객체의 사용자와 구현자



# □ 객체 사용자는 어떻게 ?

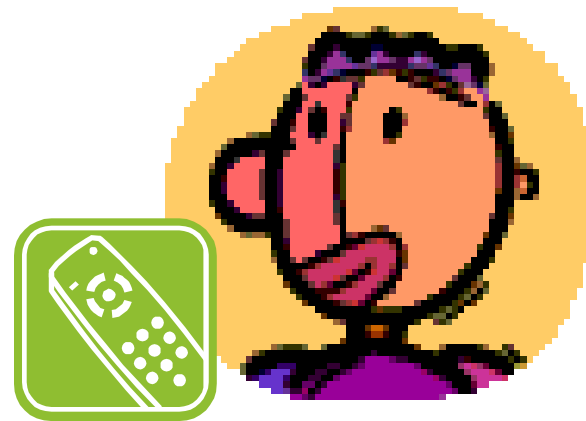
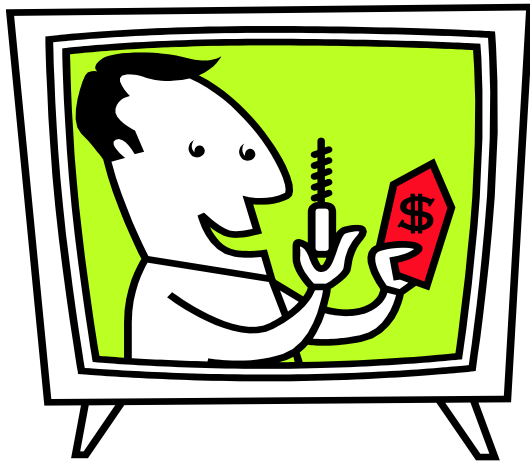
- 객체의 내부 구조나 속성을 알고 싶을까?
  - TV는 그 내부가 매우 복잡한 객체
  - TV 사용자는 TV 객체 내부의 복잡한 전자적/기계적 구조나 속성을 알고 싶을까?



# □ 객체 사용자는 어떻게 ?

■ 사용자는 단지 TV 객체를 쉽게 사용하기를 원할 뿐!

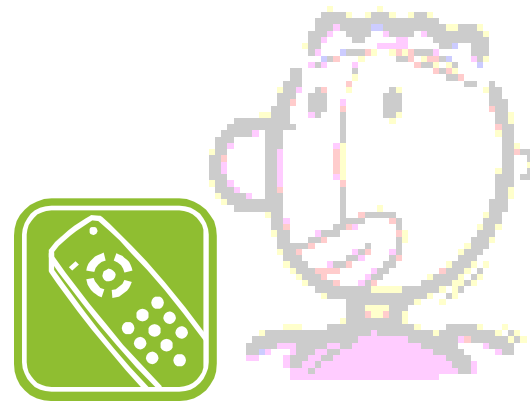
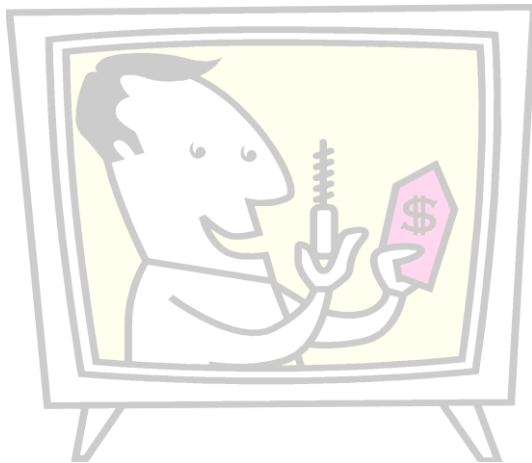
- 전원 스위치로 켜고 끈다
- 볼륨 스위치로 소리 크기를 조절한다
- 채널 스위치로 원하는 채널을 선택한다



# □ 객체 사용자는 어떻게 ?

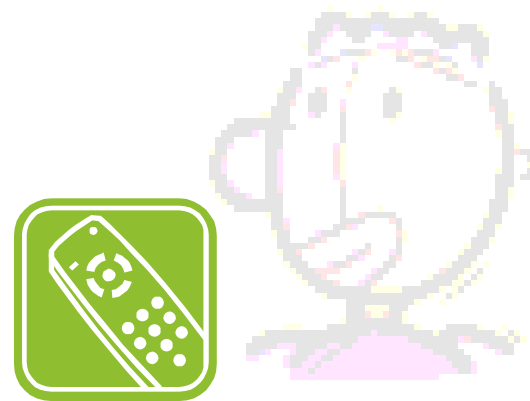
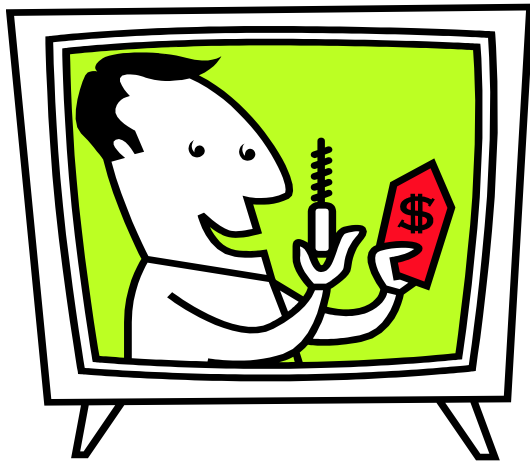
## ■ 리모콘의 역할은?

- 사용자에게 TV 객체의 내부를 알 필요 없이 편리하게 TV 시청을 하게 해 준다.
- 리모콘의 각각의 단추가 TV 를 쉽고 편리하게 사용하도록 만들어 준다.



# □ 객체 구현자는 어떻게 ?

- 구현자는 TV 객체를 사용하기 쉽고 편하고, 성능 좋고, 싸게 만들어야 한다!
- TV 제조회사(구현자)는 사용자의 요구 사항에 맞도록 그 내부의 전자적/기계적 구조를 설계하고 만들어야 한다.
- 사용법이 바뀌지 않으면서, 지속적으로 성능을 혁신시킨다.



# 추상화 (Abstraction) 란?

# □ 추상화(Abstraction) 란?

## ■ 추상화 행위는:

- 구체적인 것을 감추는 것

## ■ 왜 감추는가?

- **사용자의 편의**를 위해서는 필수적!
  - ◆ **사용자에게는**, TV를 어떻게 만들었는지 보다는,
  - ◆ **어떻게 하면 TV를 잘 사용할 수 있을지가 더 중요!**



# □ 우리의 삶 자체가 추상화의 연속!

## ■ 우리의 삶을 잘 살펴 보라!

- 문명의 발전은 추상화의 과정

## ■ 나 혼자 무엇을 얼마나 할 수 있을까?

- 왜 협업이 중요한가?
- 상대방이 내게 제공하는 결과가 무엇으로 구성되어 있고 어떻게 만들어졌는지를, 내가 자세히 알아야만 협업이 가능할까?

## ■ 지금 내게:

- 스마트폰이 없다면?
- **자바 프레임워크가 없다면?**

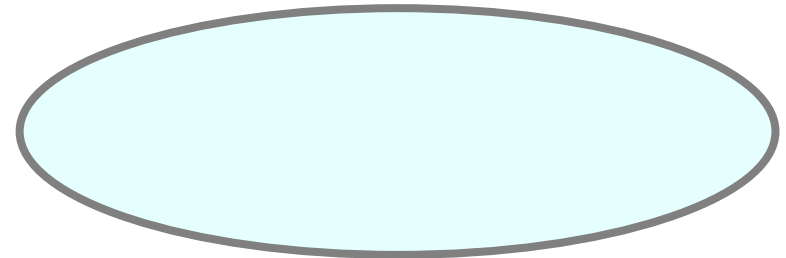
# Java 에서의 객체



# □ Java 에서의 객체의 정의

- 학생 객체를 만들어 사용하고 싶다

Student:



# □ 학생 객체의 사용법은?

학번을 부여한다

학번을 얻어낸다

성적을 부여한다

성적을 얻어낸다

합불 판정을 얻어낸다

Student:



# □ 객체는 언제나 리모콘으로만!

■ Student가 TV라면, 리모콘은?

학번을 부여한다

학번을 얻어낸다

성적을 부여한다

성적을 얻어낸다

합불 판정을 얻어낸다

Student:

# □ 객체 사용법은 공개함수로!

- 공개함수 (public functions / public methods)
  - 리모콘 단추 하나가 한 개의 공개함수

```
void setStudentNo(String aStudentNo)
```

```
String studentNo()
```

```
void setScore(int aScore)
```

```
int score()
```

```
char grade()
```

student:

# □ 객체는 Java에서 어떻게?

## ■ 일반 변수를 다시 생각해보자.

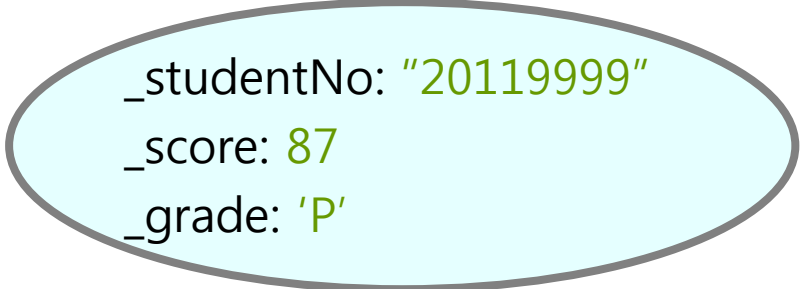
- **double** **d** ;
- 변수 **d** 역시 하나의 객체로 생각할 수 있다.
- 그 자료형이 **double** 이다.
- 직결된 연산:
  - ◆ +, -, \*, / 과 같은 사칙연산
  - ◆ >, <, ==, != 과 같은 비교연산
  - ◆ 이런 연산이 항상 모두 필요한 것은 아니다.
  - ◆ 이러한 연산이 결합된 복잡한 계산일 수도 있다.
    - 판별식의 계산
    - 판별식의 값을 판별

# □ 일반적인 객체의 표현은?

- kim이라는 학생을 위한 학생정보는?
- 우리는 이렇게 하고 표현하고 싶다.

- **Student** kim ;
  - ◆ (cf.) double d ;
- 즉, 일반 변수처럼!

**Student:** kim



\_studentNo: "20119999"  
\_score: 87  
\_grade: 'P'

- **kim**은 하나의 객체 변수이다.
  - ◆ 객체는 Java 에서 변수로 표현할 수 있다.
- 그 자료형이 **Student** 이다.



# □ Class: 객체의 자료형

- 그렇다면, 자료형 **Student**는 Java에서 어떻게 표현?
  - 다양한 객체의 형태가 미리 정의되어 존재할 수는 없다.
    - ◆ int, double, char와 같이 가장 기본적인 것만 미리 존재한다.
  - 우리가 자료형을 만들어 (정의하여) 사용한다.

```
public class Student {  
    private String    _studentNo ;  
    private int       _score ;  
    private char      _grade ;  
    .....  
}
```

- 이러한 “객체를 위한 자료형”을 **class** 라고 한다.

# □ 객체는 Class 로 정의한다

- class는 하나의 자료형 (추상자료형)

- 왜 추상 자료형이라고 할까?

- class 에는 “객체의 속성” 이 포함되게 된다.

- 나중에 좀 더 자세히.....

```
public class Student {  
    private String    _studentNo ;  
    private int       _score ;  
    private char      _grade ;  
    .....  
}
```

# □ 객체의 자료형은 class 이름으로

■ Class 에 이름을 부여하자.

- Student 객체들을 위한 Student class

```
public class Student {  
    private String    _studentNo ;  
    private int       _score ;  
    private char      _grade ;  
    .....  
}
```

## □ Class 의 선언

- 그러므로, 다음과 같이 선언된다.

```
public class Student {  
    private String    _studentNo ;  
    private int       _score ;  
    private char      _grade ;  
    .....  
}
```

## □ 객체의 속성은 **private**으로

### ■ 객체의 속성은 왜 **private**으로 선언할까?

- 속성은 TV 객체의 부품과 같은 것
- TV 내부에 사용된 부품을, 사용자가 알아야 TV를 볼 수 (작동시킬 수) 있다면?

```
public class Student {  
    private String    _studentNo ;  
    private int       _score ;  
    private char      _grade ;  
    .....  
}
```

# □ 완성된 class 인가?

## ■ 아직은.....

```
public class Student {  
    private String    _studentNo ;  
    private int       _score ;  
    private char      _grade ;  
    ..... // 이곳에는?  
}
```

## ■ 객체는 해야 할 일이 있다.

# □ 객체가 할 일은 어디에 어떻게?

- TV 객체의 리모콘 단추가 눌려졌을 때 해야 할 일:

```
public class Student {  
    private String _studentNo ;  
    private int _score ;  
    private char _grade ;  
    ..... // 이곳에 함수를 선언하고 구현한다  
}
```

- 어디에?
  - Class 안에 선언한다
- 어떻게?
  - 함수 형태로 표현 한다
  - 리모콘 단추 하나가 한 개의 함수

## □ 객체 변수 (Object Variable)

- 이제 우리는 일단은, 객체 변수를 만들어 사용할 수 있다.
- 일반 변수의 선언 방법과 마찬가지로.

```
public class Student {  
    private String  _studentNo ;  
    private int     _score ;  
    private char    _grade ;  
    // ..... // 일단은 주석으로  
}
```

```
Student kim ;
```



# □ 객체 변수가 바로 객체?

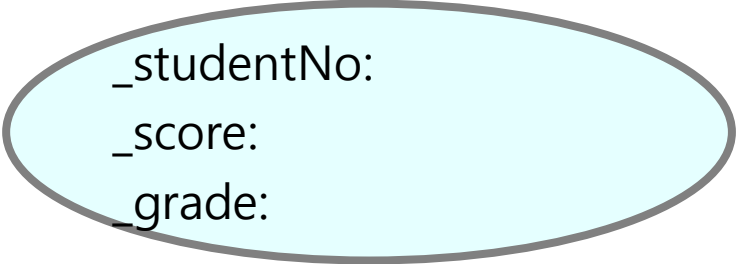
```
public class Student {  
    private String _studentNo ;  
    private int    _score ;  
    private char   _grade ;  
    // .....  
}
```

```
Student kim ;
```

## ■ 객체 변수 kim의 선언으로 그림처럼 될까?

- 마치 일반 변수처럼.....
- 즉, 객체 변수의 선언으로 객체가 바로 존재하게 될까?

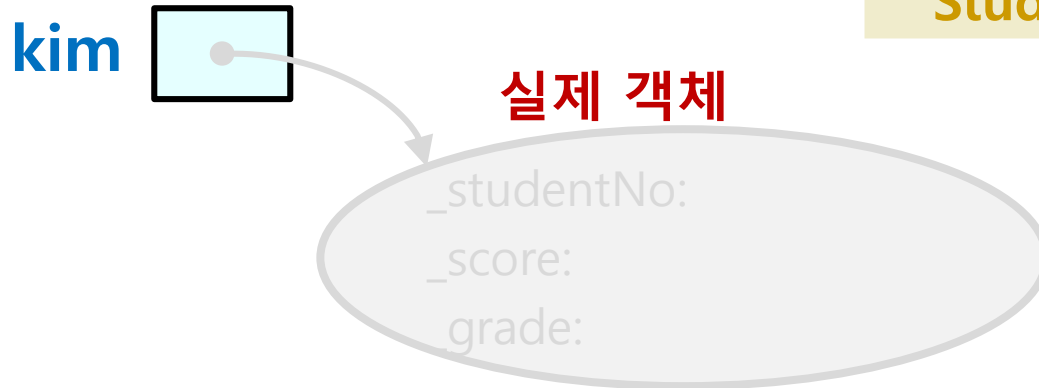
Student: kim



\_studentNo:  
\_score:  
\_grade:

# □ 변수 선언으로 객체는 만들어지지 않는다!

- 객체 변수 kim의 선언의 결과는...



```
public class Student {  
    private String _studentNo ;  
    private int    _score ;  
    private char   _grade ;  
    // .....  
}
```

```
Student kim ;
```

- 객체 변수 kim 은 단지 객체를 소유할 수 있을 뿐이다.
  - 객체는 아직 존재하지 않는다.

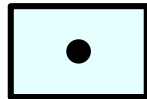
# □ 객체를 생성시켜 보자 [1]

## ■ 객체 변수는 선언과 동시에 null로.

- 즉, 현재 kim 은 아무 객체도 소유하고 있지 않다.

```
Student kim = null ;
```

kim



```
public class Student {  
    private String    _studentNo ;  
    private int       _score ;  
    private char      _grade ;  
  
    // 생성자  
    public Student() {  
        this._studentNo = null ;  
        this._score = 0 ;  
        this._grade = ' ' ;  
    }  
  
    // ...  
}
```

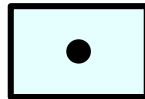
# □ 객체를 생성시켜 보자 [2-1]

- 모든 객체는 **객체 생성자**와 함께 **"new"**를 사용하여 생성시킨다.

```
Student kim = null ;  
kim = new Student() ;
```

```
public class Student {  
    private String    _studentNo ;  
    private int       _score ;  
    private char      _grade ;  
  
    // 생성자  
    public Student() {  
        this._studentNo = null ;  
        this._score = 0 ;  
        this._grade = ' ' ;  
    }  
    // ...  
}
```

kim



실제 객체

\_studentNo: null  
\_score: 0  
\_grade: ' '

# □ 객체를 생성시켜 보자 [2-2]

## ■ 객체 생성자

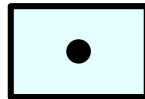
- 모든 객체에게 필수적인 리모콘 단추
- 이름은 Class와 동일
- 객체를 생성하고, 초기 상태를 설정

```
Student kim = null ;
kim = new Student() ;
```

```
public class Student {
    private String    _studentNo ;
    private int       _score ;
    private char      _grade ;

    // 생성자
    public Student() {
        this._studentNo = null ;
        this._score = 0 ;
        this._grade = ' ' ;
    }
    // ...
}
```

kim



실제 객체

\_studentNo: null  
\_score: 0  
\_grade: ' '

# □ 객체를 생성시켜 보자 [3]

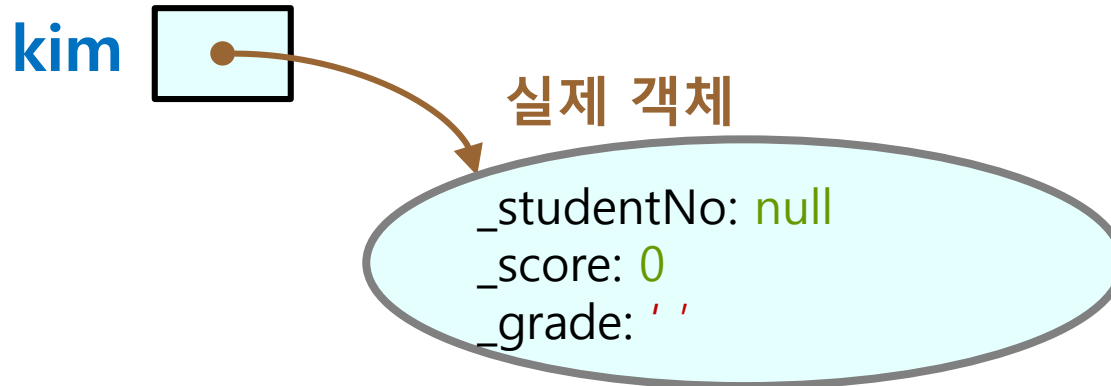
## ■ **new** 의 결과값은 객체의 소유권

- 메모리에서의 객체의 주소값
- 생성자 함수의 return 값

```
Student kim = null ;
kim = new Student() ;
```

```
public class Student {
    private String    _studentNo ;
    private int       _score ;
    private char      _grade ;

    // 생성자
    public Student() {
        this._studentNo = null ;
        this._score = 0 ;
        this._grade = ' ' ;
    }
    // ...
}
```



# □ 객체의 생성은?

- Class는 스스로 객체 생성 방법을 가지고 있어야 한다.
- 생성자 (Constructor): Class 안에 함수로 정의

```
public class Student {  
    private String    _studentNo ;  
    private int       _score ;  
    private char      _grade ;  
  
    // 생성자  
    public Student () {  
        this._studentNo = null ;  
        this._score = 0 ;  
        this._grade = ' ' ;  
  
    }  
    // ...  
}
```

반드시  
class 이름과 동일

# □ 객체의 생성자는 반드시 public

■ 생성자는 사용자가 사용!

반드시 public:  
객체를 외부에서  
사용(생성) 해야  
하므로

```
public class Student {  
    private String    _studentNo ;  
    private int       _score ;  
    private char      _grade ;  
  
    // 생성자  
    public Student () {  
        this._studentNo = null ;  
        this._score = 0 ;  
        this._grade = ' ' ;  
    }  
    // ...  
}
```



# □ 객체의 생성자는 무엇을 돌려주는가?

- 객체를 생성하고 그 소유권을 돌려준다.
  - 소유권: (또는 사용권)
    - ◆ 실제로는 메모리에 생성된 객체의 주소값

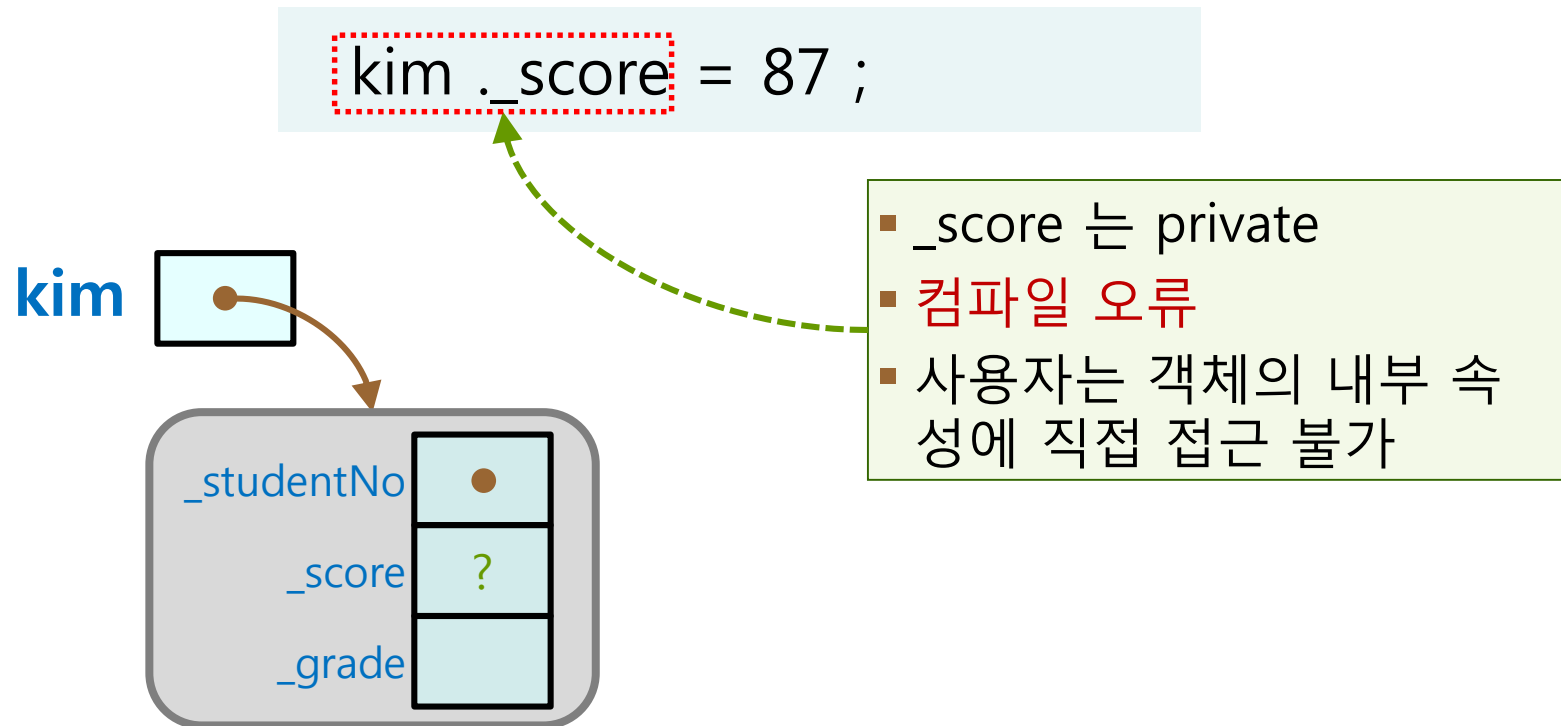
```
public class Student {  
    private String    _studentNo ;  
    private int       _score ;  
    private char      _grade ;  
  
    // 생성자  
    public Student () {  
        this._studentNo = null ;  
        this._score = 0 ;  
        this._grade = ' ' ;  
    }  
    // ...  
}
```

돌려주는 값은 있지만,  
그 type은 명시하지 않는다

return 문도 없다

# □ 객체의 사용자는 어떻게 객체를 사용?

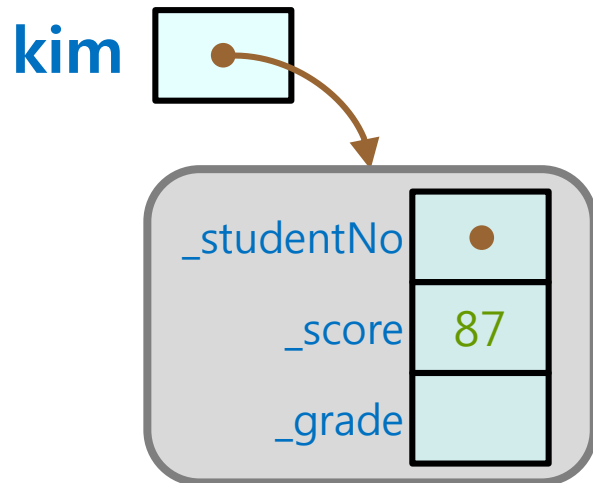
- 객체의 사용자는 객체의 private 속성을 알지 못한다.



## □ 객체에게 일을 시킨다

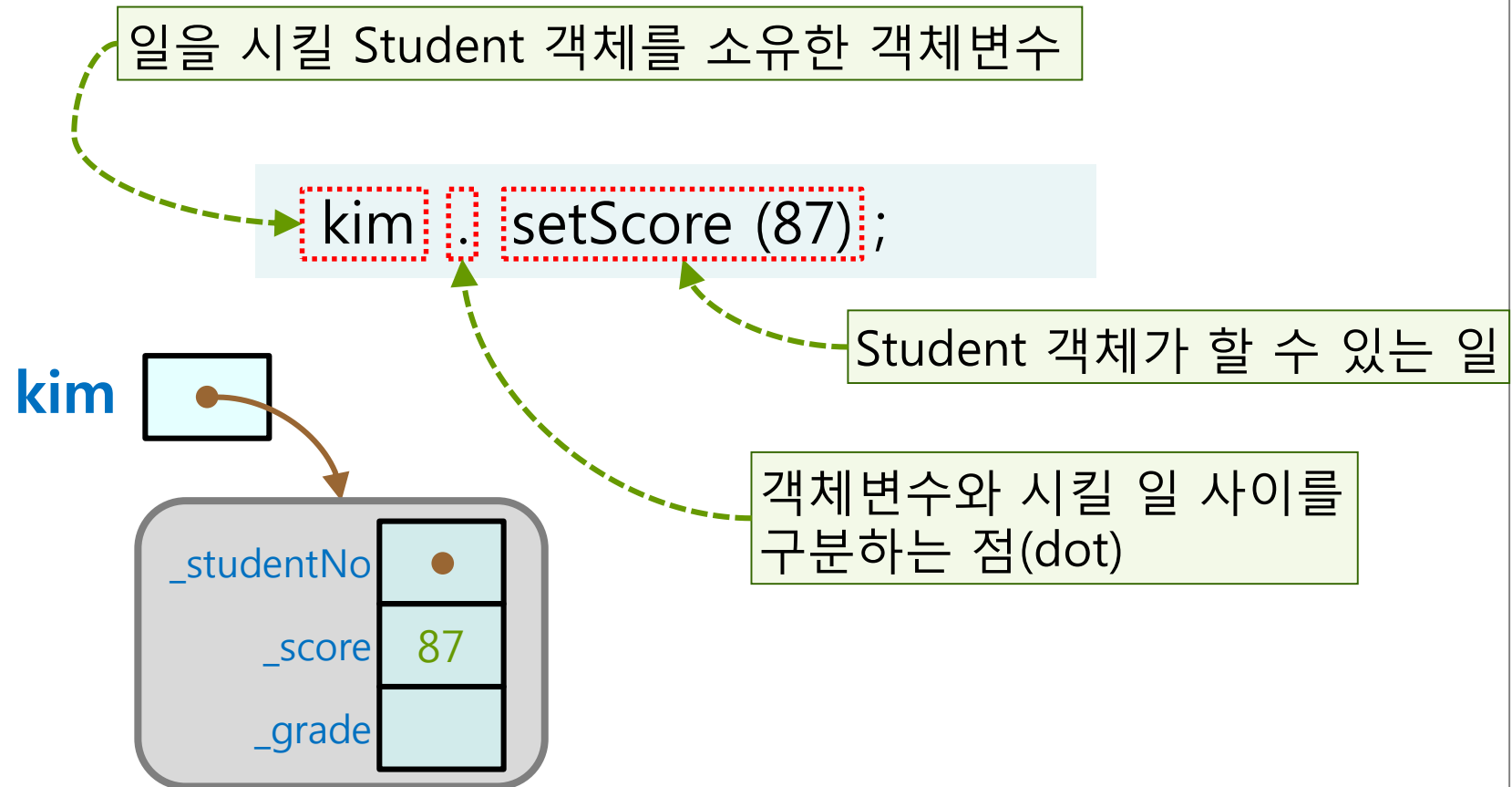
- 리모콘 단추 `setScore()`를 누른다.
- 객체의 사용자는 객체에 대해 직접 일을 하지 않고, 반드시 객체에게 시켜서 목적을 이룬다.
- TV 리모콘을 생각해 보라

```
kim . setScore (87) ;
```



# □ 객체에게 일을 시킨다

- 사용자는 공개함수를 사용하여 객체에게 일을 시킨다.

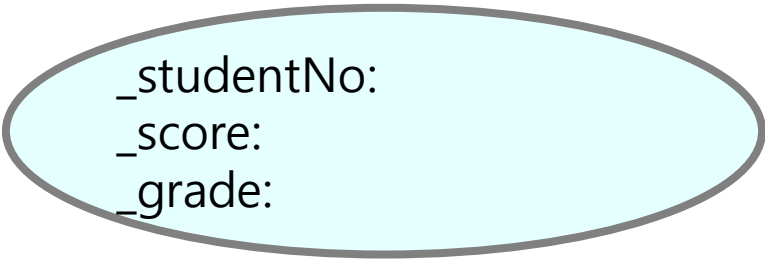


# Java 에서의 객체의 구현

## □ 객체의 내부 구조는?

- 학번 (\_studentNo): 9개 문자를 저장할 수 있는 문자열
- 성적 (\_score): 정수형
- 학점 (\_grade): 문자
  - 성적이 60점 이상이면 학점은 'P'의 값을, 60점 미만이면 'F'의 값을 갖는다.

**Student:**



\_studentNo:  
\_score:  
\_grade:

# □ 객체가 해야 할 일은?

- 학번의 set / get
- 성적의 set / get
- 성적 합불 평가
  - ◆ 성적을 판단하여, 통과/실패의 결과를 얻는다.
- 합불 평가의 get

Student: kim

\_studentNo: "20119999"  
\_score: 87  
\_grade: 'P'

```
if (kim의 _score >= 60)
    kim의 _grade = 'P';
else
    kim의 _grade = 'F' ;
```

# □ Class 안에서의 사용법의 정의

```
public class Student {  
    private String    _studentNo ;  
    private int       _score ;  
    private char      _grade ;  
  
    // 생성자  
    public    Student() {  
        .....  
    }  
  
    public void setScore (int aScore) {  
        this._score = aScore ;  
    }  
  
    public int score () {  
        return this._score ;  
    }  
}
```

성적의 setter

성적의 getter



# □ Class 안에서의 사용법의 정의

사용자의  
사용법이므로  
반드시 public

```
public class Student {  
    private String    _studentNo ;  
    private int       _score ;  
    private char      _grade ;  
  
    // 생성자  
    public    Student() {  
        .....  
    }  
  
    public void setScore (int aScore)  
    {  
        ..... // 여기에 실제 할 일을  
    }  
  
    public int score ()  
    {  
        ..... // 여기에 실제 할 일을  
    }  
}
```

# □ 객체가 할 일의 구현

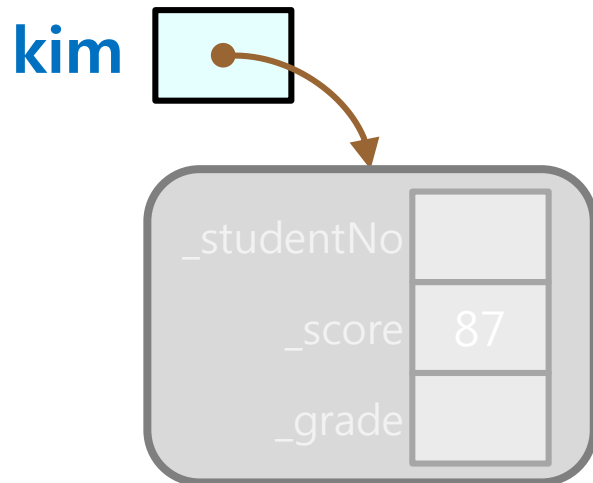
```
public class Student {  
    private String    _studentNo ;  
    private int       _score ;  
    private char      _grade ;  
  
    // 생성자  
    public    Student() {  
        .....  
    }  
  
    public void setScore (int aScore)  
    {  
        this._score = aScore ;  
    }  
  
    public int score ()  
    {  
        return this._score ;  
    }  
}
```

각 사용법에 대해,  
객체가 실제로 할  
일

## □ Class 안에서 속성의 사용법

- 공개함수가, 자신에게 일을 시킨 (사용자가 지정한) 객체를 인지하는 방법은?

```
public void setScore (int aScore)
{
    this ._score = aScore ;
}
```



```
kim . setScore (87) ;
```

사용자가 객체 kim에게  
"성적을 부여하는 일"을 시켰다

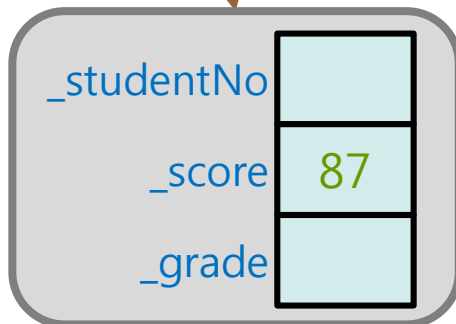
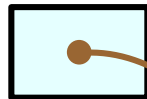
# □ Class 안에서 속성의 사용법

## ■ "this"

일을 하도록 지시 받은 객체 자신을 가리킨다  
"this" 표현은 객체 외부에서는 사용 불가

```
public void setScore (int aScore)
{
    this._score = aScore ;
}
```

kim



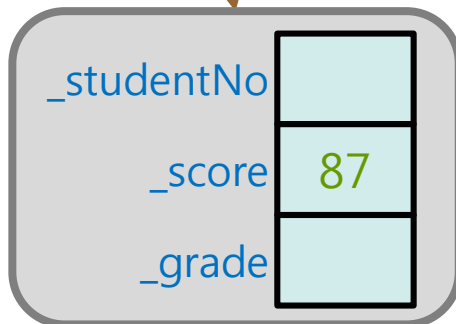
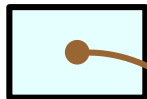
```
kim.setScore (87) ;
```

# □ Class 안에서 속성의 사용법

## ■ this.\_score

```
public void setScore (int aScore )
{
    this._score = aScore ;
}
```

kim



일을 하도록 명령을 받은 객체가 가지고 있는 속성의 이름

```
kim . setScore ( 87 ) ;
```

# Java 프로그램 코딩에 대해

## □ 공통 사항

### ■ 변수 이름과 함수 이름

- 소문자로 시작한다
- 여러 단어로 구성될 경우에는 두 번째 이후의 단어는 대문자로 시작한다
  - ◆ `int numberOfStudents ;`

### ■ 상수

- 전체를 대문자로 한다
- 여러 단어로 구성될 경우에는 단어 사이에 '\_'를 넣는다.

### ■ Class 이름

- 대문자로 시작한다

# □ Class 내부에서

Class 이름은  
대문자로 시작

함수 이름은  
소문자로 시작

class 내부에  
선언된 객체의  
속성이나 함수  
에는 앞에  
"this."를 붙인  
다

```
public class Student {
    private String _studentNo ;
    private int _score ;
    private char

    // 생성자
    public Student () {
        .....
    }

    public void setScore (int aScore)
    {
        this._score = aScore ;
    }

    public int score ()
    {
        return this._score ;
    }
}
```

객체의 속성 이름  
은 '\_', 그리고  
이어서 소문자로  
시작

Getter 의 이름은  
얻고자 하는 대상  
의 이름으로 한다.  
Setter의 이름은  
getter 이름에  
"set"을 붙인다 .



# 기본 자료형 (Basic Data Types)

# □ 이론적인 관점에서의 기본 자료형

## ■ 이론적인 관점에서, 자료형은 원소 (값) 들의 집합

### ■ 예: "int"

- $\text{int} = \{ -2^{31}, \dots, -1, 0, 1, 2, 3, \dots, +(2^{31}-1) \}$
- 변수 선언의 의미:
  - ◆ `int count;`
    - 변수 `count` 는 집합 `int` 중에서 한 개의 원소 (값)을 가질 수 있다.

### ■ 예: "boolean"

- `boolean = { false, true }`
- `boolean found;`
  - ◆ 변수 `found` 는 집합 `boolean`의 원소 중 하나의 값을 갖는다.  
즉, `found`는 `false` 이거나 `true` 의 값을 갖는다.

## □ 자료형을 사용하는 이유는?

- 자료의 형을 명시적으로 선언하면 비명시적인 경우에 비해 사용자가 잘못 사용할 가능성을 줄여줄 수 있다.
  - 컴파일러는 프로그래머가 잘못 사용하여 자료형이 일치하지 않은 것을 찾아낸다.
- 변수의 자료형을 명시적으로 선언하면, 다른 사용자에게 그 변수의 용도를 이해하는데 도움을 줄 수 있다.

# □ Java 에서의 기본 자료형

- Java 에서 사용할 수 있는 기본 자료형
  - boolean, byte, char
  - short, int, long, float, double
- Java를 포함한 대부분의 프로그램 언어는 최소한의 기본자료형을 제공한다.
- 더 복잡한 자료형이 필요하다면?
  - 배열
  - Class (추상자료형 )

# □ 기본 자료형과 관련 연산

- 자료형은 **관련된 연산**을 가지고 있다.
  - 그 자료형의 값에 적용할 수 있는 행위가 있다.
  - 자료형마다 관련된 연산들은 동일하지는 않다.
- 예:
  - int: +, -, \*, /, %, ++, --, ...
  - double: +, -, \*, /, ...
    - ◆ 실수(double) 자료형은 %, ++, -- 과 같은 연산은 가지고 있지 않다.
- 기본자료형은 관련된 연산을 **묵시적으로** 가지고 있다.
- 모든 자료형은 관련된 연산을 가지고 있다.
  - Class (추상자료형)의 관련 연산은 **명시적으로** 선언

# 추상 자료형 (Abstract Data Types)

# □ 추상 자료형 (ADT)

## ■ 추상자료형도 자료형

■ 객체 인스턴스에 적용되는 관련 연산들이 있다.

- 객체 인스턴스: 객체의 실체

## □ 추상화란?

### ■ 가루로 된 매우 쓴 마이신 약을 캡슐로 감싸자!

- 마이신은 염증 치료에 사용되는 매우 쓴 가루약
- 염증 환자: **사용자 (User)**
  - ◆ 치료를 위해 마이신 가루약을 먹는다.
  - ◆ 그렇지만 먹어서 낫기만 하면 되는 것이 목적이지만, 그 약의 쓴 맛을 보는 것은 목적이 아니다.
- 제약회사: **구현자 (Implementer)**
  - ◆ 마이신 약의 목적은 살리되, 환자가 원하지 않는 약의 쓴 맛은 모르게 하고 싶다.
  - ◆ **캡슐화 (encapsulation)**: 약의 효능에는 영향을 주지 않으면서도 맛은 쓰지 않은 **캡슐(capsule)**을 만들어 그것으로 마이신 약을 감싸서 알약으로 만든다.
  - ◆ 즉, 캡슐화된 알약을 먹으면 염증치료가 된다



## □ 캡슐화가 추상화!

- 추상화 (abstraction): 어떻게 만들어졌는지와는 관계없이, 염증치료가 되도록 가루약을 알약으로 캡슐화 하는 행위
- 사용자는 구현 방법에 독립적 (implementation-independent):
  - 사용자는 치료를 위해 단지 캡슐로 된 염증치료 알약을 먹으면 된다. 이미 효능이 보장된 캡슐 알약 속에 무엇이 들어 있는지 또는 어떻게 만들었는지는 관심을 가질 필요가 없다.
  - 구현자는 기술이 발달하여 최신기법의 염증치료 약제를 개발하게 되면 단지 캡슐 안의 성분만 바꾸면 된다.

# □ TV 에서의 추상화

## ■ TV의 사용자와 생산자

- TV를 시청하려는 사용자:
  - ◆ TV를 만드는 방법에는 별 관심이 없다.
  - ◆ 단지 리모콘의 전원 단추를 누르고, 채널을 선택하고 볼륨을 조절할 수 있으면 된다.
- TV 생산자:
  - ◆ 효율적이면서도 편리하게 사용할 수 있는 구체적인 제조 방법을 알아야만 한다.

## ■ 리모콘과 함께 Capsule화 된 TV

- 사용자는 TV 내부는 몰라도 리모콘 만으로 시청이 가능
- 캡슐화가 됨으로써, TV는 누구나 편리하게 사용할 수 있도록 추상화 되었다!

## □ 객체를 사용하려면...

- 사용자에게 좋은 리모콘이 제공되어야 한다
  - 리모콘은 바로 객체의 공개함수
  - 객체에게 필요하면서도 사용하기 편리한 공개함수는?
- 구현자는 리모콘의 각 단추가 눌러졌을 때 객체가 무슨 일을 해야 할지를 정확하고 효율적으로 구현해야 한다.

# □ 추상자료형 (Abstract Data Types)

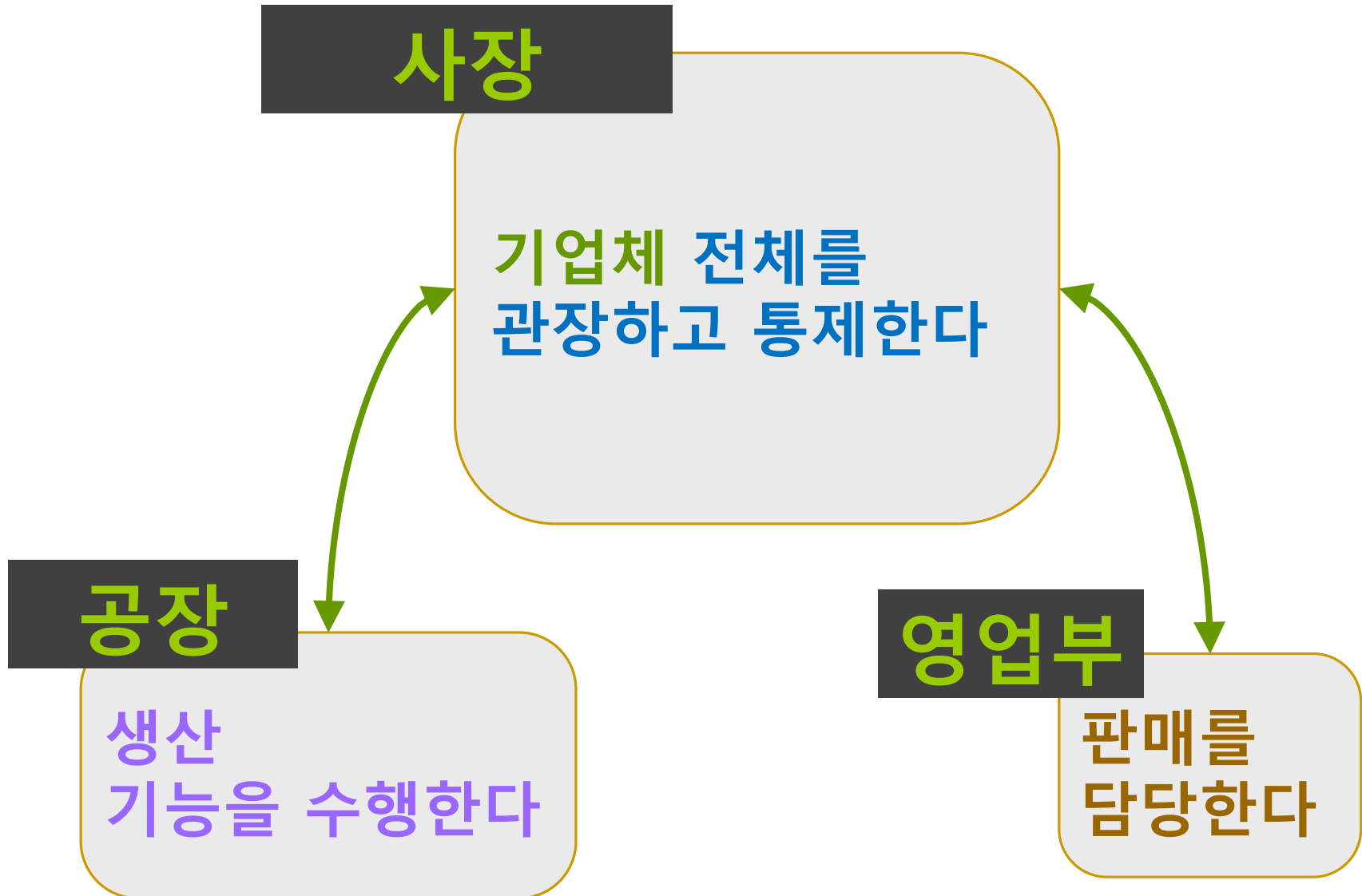
- 자료형
- 객체의 규격
- 객체에 대한 연산의 규격
- 객체의 표현과 구현의 분리
  - 구현에 독립적 (implementation-independent)

# □ 추상자료형 (Abstract Data Types)

- 객체는 저장 공간이며, 또한 값
- 추상자료형의 연산
  - 생성 (Construction)
  - 소멸 (Destruction)
  - 상태 알아보기 (Observation / Reporting)
  - 상태 바꾸기 (Modification / Transformation)

# Model-View-Controller

# □ App 에서의 역할 구분



# □ App 에서의 역할 구분

## Controller

App 전체를  
관장하고 통제한다

## Model

핵심 두뇌 (생산)  
기능을 수행한다

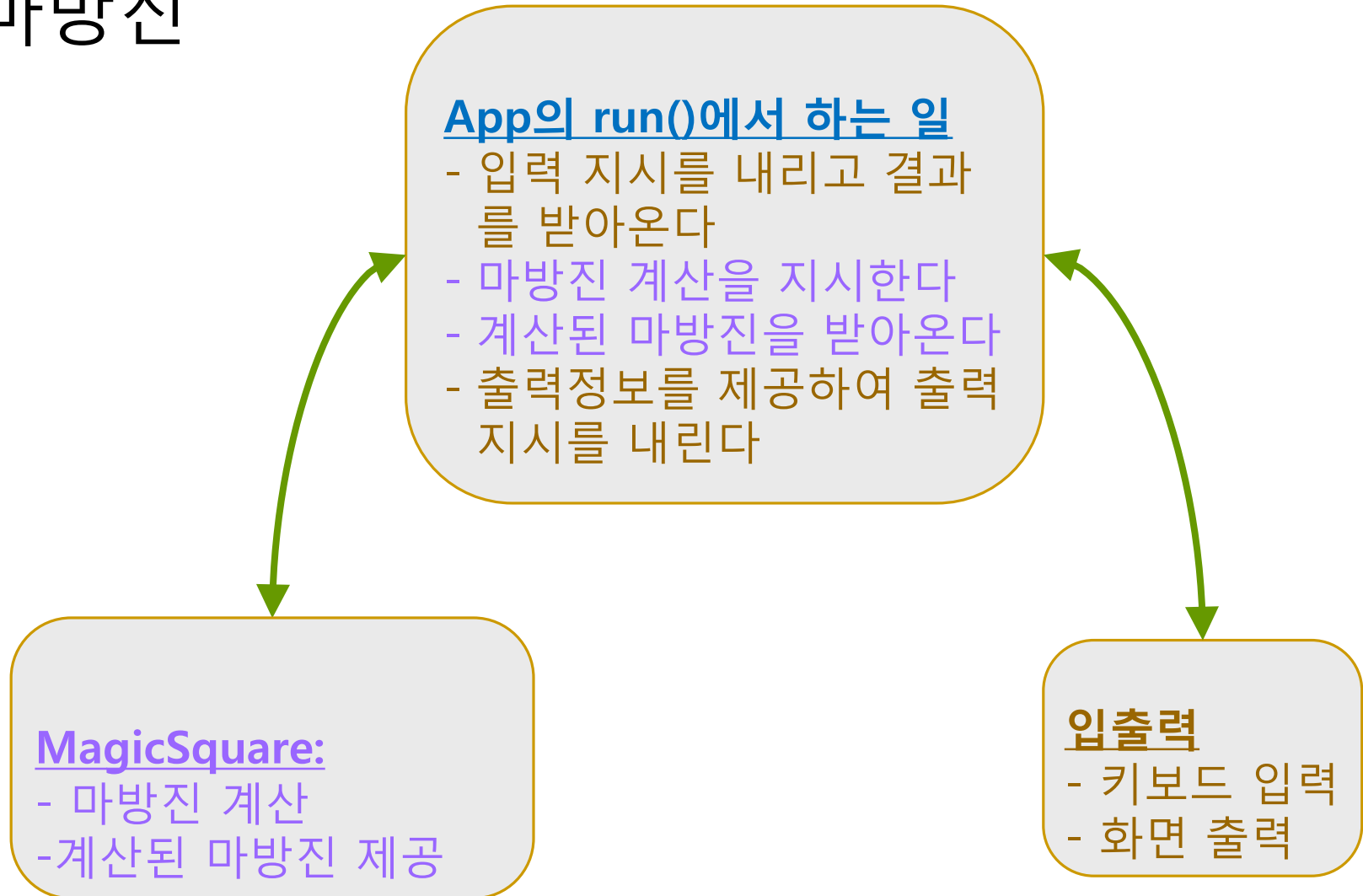
## View

입출력을  
담당한다

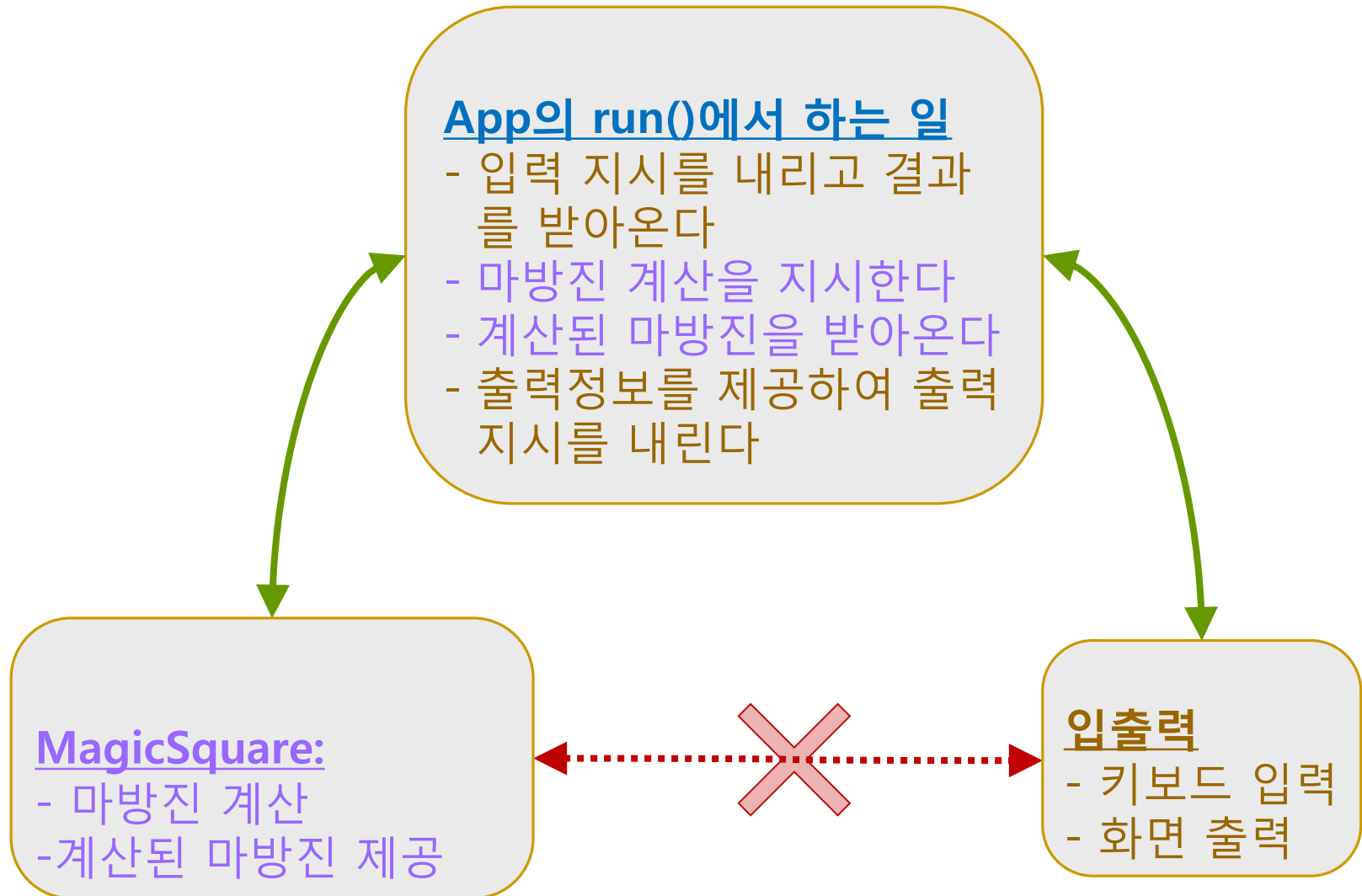


# □ 모든 프로그램은 MVC 형태로

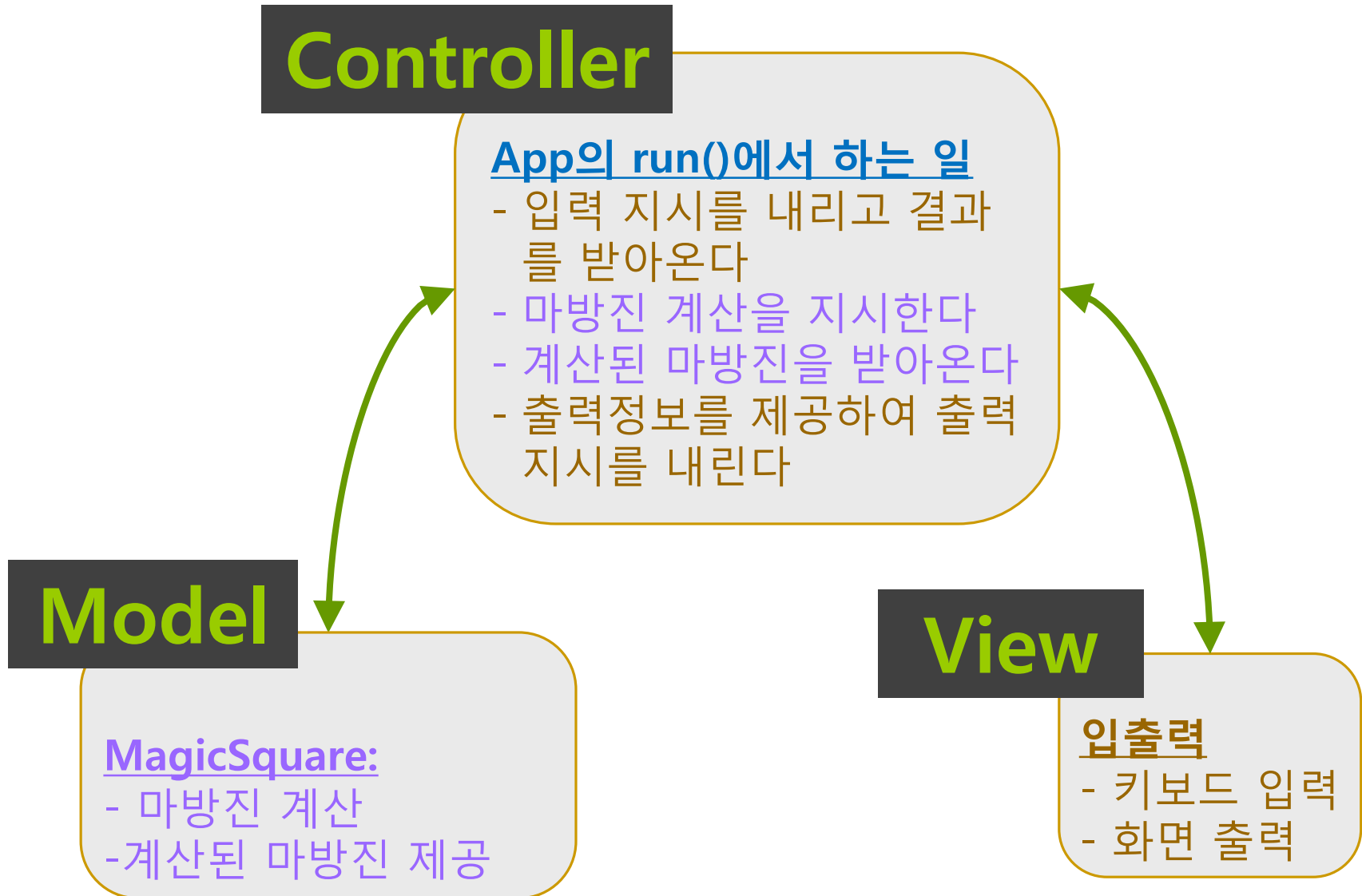
## ■ 마방진



# □ 부하 직원이 상사 몰래?



# □ MVC



# □ 각각을 class 로!

## ■ 마방진 프로그램

- Controller: Class "AppController"
- Model: Class "MagicSquare"
- View: Class "AppView"

# End of “Objects”



