클래스와 객체1

목차

Chap01.
객체지향 프로그래밍

▶ 객체 지향 프로그래밍

KH정보교육원

✓ OOP(Object Oriented Programing)

데이터 중심이 아닌 **객체 중심**의 프로그래밍

Software 위기의 대안으로 제시됨

연관되는 속성과 기능을 묶어서 부품화 시킴(캡슐화)

현실세계에 대한 모델링

✓ 객체(Object)

컴퓨터, 고객, 학생, 자동차 등 현실 세계에서 흔히 찾아볼 수 있는 대상을 추상화(Abstraction) 하여 프로그램 상에서 만들어낸 결과물

속성과 기능을 가지고 있는 하나하나의 모든 것

▶ 객체 지향 프로그래밍

✓ 절차지향 vs 객체지향

절차지향 : **작업의 흐름**에 따라 코드를 작성

객체지향 : **객체의 관계**에 따라 코드를 작성

※ 실제 프로그램이 동작하는 방식이 다른 게 아니라 코드를 어떻게 작성하느냐 에 대한 개념적인 차이

✓ 객체지향 프로그래밍의 장점

프로그램 **모듈의 재사용** 가능

프로그램의 확장 및 유지 보수가 용이함

쉬운 프로그램의 개발로 인한 생산성 향상

완성도 높은 모듈 사용으로 프로그램의 안정성 확보

Chap02.
클래스 (Class)

클래스, 객체, 추상화

객체 지향 언어 - 클래스

✓ 클래스(Class)

객체를 생성하기 위해 **속성과 기능을 정의**한 일종의 틀(설계도)

속성(상태) = 변수, 기능(행위) = 메소드 로 구성됨

값을 가지는 것 (전원이 꺼져있다,손을 가지고 있다.)

사물이나 개념의 공통 요소를 **추상화(abstraction)**하여 정의

클래스를 설계하는 단계

ex) 제품의 설계도, 빵 틀

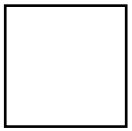
✓ 멤버

멤버 변수 : 클래스 내에 선언 된 변수

멤버 메소드 : 클래스 내에 정의 된 메소드

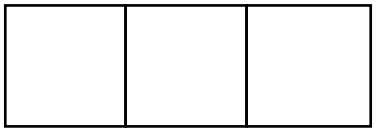
객체 지향 언어 - 클래스

✓ 클래스의 등장 배경



변수

1개의 자료형
1개의 데이터



배열

1개의 자료형
여러 개의 데이터

내가 묶고싶은 것 다 묶어서
새로운 이름 하나 만들어줌

'student' 라는 이름 붙임
사용자 정의

구조체

여러 개의 자료형
여러 개의 데이터

자료형에 상관없이
연관된 데이터들을 사용할 수 있는 것

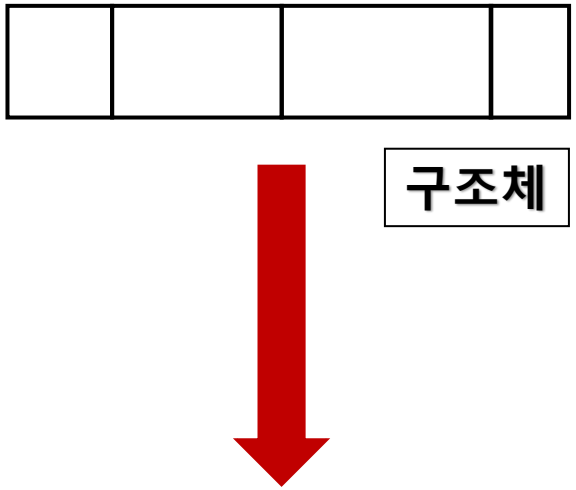
int 4byte
double 8byte
char 2byte
세개를 묶어서 하나의 이름으로 공간을 사용

student st;
라고 하면 14바이트짜리 공간이 만들어짐

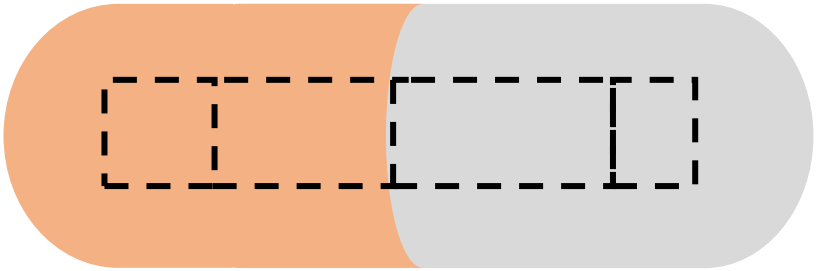
객체 지향 언어 - 클래스

속성과 기능을 묶은 것
변수 메소드

✓ 클래스의 등장 배경

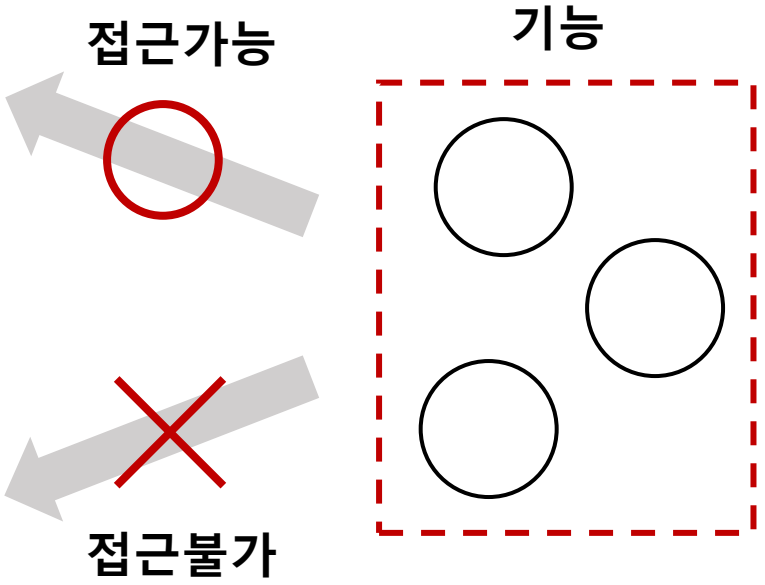


데이터 접근 제한



학생관리 구조체를 만들었음

이 안에 학생 정보 넣을 것임
이 학생의 정보를 빼서 쓰고 기록하는 등의 기능은 또 따로 만들어야함
이 기능들까지 같이 묶자 => 클래스



클래스 내부의 데이터를
연산 처리하는 기능을 포함하게 됨

설계도일뿐임
이런이런 식으로 만들어 질거야 -> 사용자 정의 자료형
설계도를 가져다 쓸거면 메모리에 공간을 만들어야함

▶ 객체 지향 언어 - 객체

클래스로 공간을 만든 것
메모리에 저장해서 실체화 시킨 것

KH정보교육원

✓ 객체(Object or Instance)

현실에 존재하는 독립적이면서 하나로 취급되는 사물이나 개념으로 객체 지향 언어에서 객체의 개념은 클래스에 정의된 내용대로 메모리에 할당된 결과물(Object)

클래스 (Class) **객체(Instance)**

학생 (Student)
기능과 속성을 이용해서 클래스 만듬

인스턴스화 instantiation

클래스를 가지고 객체를 만드는 과정
인스턴스화 시킨다

김철수

김영희

학생이 가지는 공통적인
요소를 추상화 하여
클래스를 정의

현실세계에 존재하는
고유 객체를
메모리에 할당

객체 지향 언어 - 추상화

추상화 -> 클래스 -> 객체

KH정보교육원

✓ 추상화(abstraction)

클래스를 만들기 위해서는
추상화를 통해 만들어야 함

프로그램이 필요로 하는 실제 데이터들을 모델링하는 기술

컴퓨터를 만들고싶어
컴퓨터에는 무슨 기능이 있지?
꼭 필요한 기능 뭐지?

유연성을 확보하기 위해 구체적인 것은 제거한다는 의미

프로그램에서 필요한 공통점을 추출하고, 불필요한 공통점을 제거하는 과정

너무 구체적이면 이 클래스를 가지고 내 마음대로 사용하기 힘들다

✓ 클래스 vs 객체 vs 추상화

추상화 : 객체에서 필요로 하는 데이터와 동작들을 **정리하는 과정**

클래스 : 추상화한 내용을 정리한 **설계도**

객체 : 클래스를 실제 사용한 프로그램 상에서의 **결과물**

객체 지향 언어 - 추상화

KH정보교육원

✓ 추상화(abstraction) 예시

국가에서 국민 정보 관리용 프로그램을 만들려고 할 때,
프로그램에서 요구되는 “국민 한 사람”의 정보를 추상화 한다면?

국민을 관리하기 위해서
필요한게 뭐가 있지?

클래스는 1개가 되고
우리반으로 치면
객체는 30개가 된다

객체 지향 언어 - 추상화

KH정보교육원

객체들마다 같은 항목에 대한 변수명을 다 다르게 쓰면 나중에 쓰기 어렵기 때문에 미리 변수명과 자료형을 정리한다.

✓ 추상화(abstraction) 예시

앞 페이지에서 추상화한 결과물을 객체 지향 프로그래밍 언어를 사용해서 변수명(데이터 이름)과 자료형(데이터 타입) 정리

항목	변수명	자료형(type)
주민등록번호	pNo	String
이름	name	String
성별	gender	char
주소	address	String
전화번호	phone	String
나이	age	int

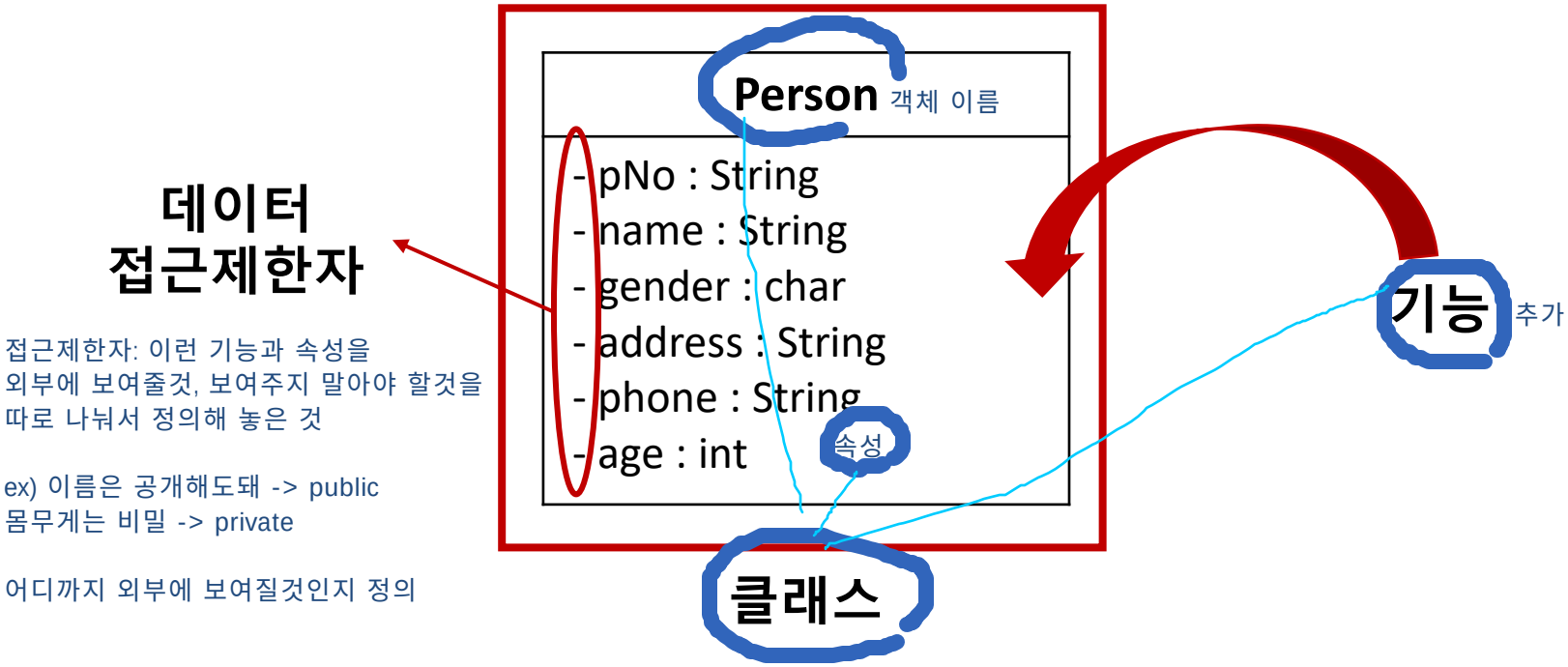
객체 지향 언어 - 추상화

✓ 추상화(abstraction) 예시

UML
클래스 다이어그램

추상화한 결과물들간의 관계를
쉽게 알 수 있도록 변수명과 데이터타입을
정리한다

앞 페이지에서 정리된 변수명과 자료형을 클래스 다이어그램으로
표현 시 아래와 같음



▶ 클래스 정의(선언)

✓ 클래스 소스 파일 생성

클래스명.java 로 생성

반드시 클래스명과 대/소문자까지 일치해야 함

✓ 클래스명명 규칙

하나 이상의 문자로 이루어져야 함

첫 글자는 숫자가 올 수 없음

$, _ 외의 특수문자 사용 불가

예약어 사용 불가

관례적으로 첫 글자는 대문자, 나머지 소문자로 사용

관례적으로 두 단어 이상이면 각 단어의 첫 글자 대문자로 사용

한글도 가능하지만 가능하면 영어로 작성

▶ 클래스 정의(선언)

✓ 클래스 용도

1. 라이브러리용 클래스
   - API 용도
   - 다른 클래스에서 이용할 목적으로 설계되는 클래스

2. 실행용 클래스
   - 프로그램의 진입점인 main() 메소드를 제공하는 역할
   - 프로그램 전체에서 사용되는 클래스 중 하나만 실행용 클래스임

▶ 클래스 정의(선언)

✓ 클래스 정의 형식

클래스만 있는 경우에는
이 변수 공간 만들어 지지 않음 (설계도일뿐)
=> 나중에 이 클래스로 공간 만들면
이러이러한 공간 만들거야
라고 정의만 하는 것

[접근제한자] [예약어] class 클래스명 {

생략가능

[접근제한자] [예약어] 자료형 변수명;
[접근제한자] [예약어] 자료형 변수명; → 속성 (Field)

[접근제한자] 생성자명() { } → 생성자 (Constructor)

[접근제한자] 반환형 메소드명(매개변수) {
// 기능 정의
} → 기능 (Method)

}

▶ 클래스 정의(선언)

✓ 클래스 정의 예시(Member.java 파일)

```
public class Member {
        public String name;
        public int age;

        public Member() {}

        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;
        }
}
```

속성 (Field)

생성자 (Constructor)

기능 (Method)

▶ 클래스 정의(선언)

✓ 클래스 파일

일반적으로 소스파일 당 하나의 클래스를 선언

두 개 이상의 클래스 선언도 가능

컴파일 시 각 클래스마다 따로 .class 파일이 생성 됨

파일 이름과 동일한 클래스명에만 public 사용 가능

가급적이면 소스 파일 하나엔 하나의 클래스만 선언하는 것이 좋음

✓ 예시

```
public class 클래스1 {
}
class 클래스2 {
}
```

객체 생성(할당)

✓ 객체 생성

클래스에 미리 정의되어 있는 형식으로 객체 생성

new 연산자를 사용하여 객체를 생성

heap영역에 공간 만들고 주소를 우리에게 준다.

✓ 객체 사용 순서

1. 클래스 설계
2. 클래스를 이용해 객체 생성
3. 생성된 객체 사용

객체 생성(할당)

✓ 객체 생성 형식

```
클래스명 참조변수명 = new 클래스명();
```

객체의 주소를 저장 4byte
우리가 만든 클래스 형태 공간의 주소를 저장할 수 있는 참조변수가 만들어진다.

✓ 객체 생성 예시(Run.java 파일)

```
public class Run {
    public static void main(String[] args) {

        Member mb = new Member();

    }
}
```

객체 생성(할당)

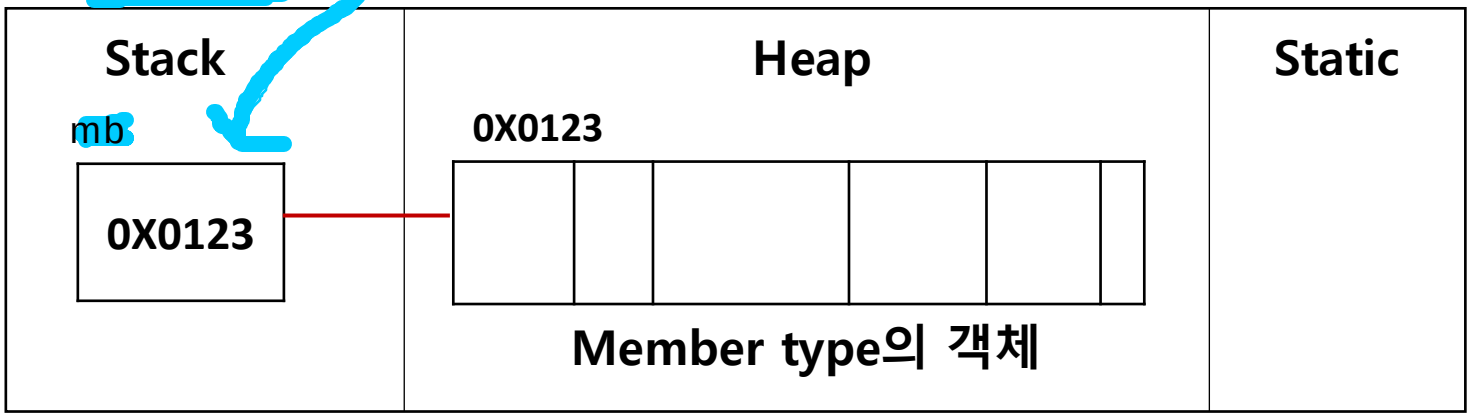
✓ 객체 할당

heap 영역에 객체를 생성한 후 주소를 리턴

참조변수를 만들어 주소를 담아서 사용

참조변수는 객체와 동일한 클래스 타입으로 생성해야 함

예) Member mb = new Member();



클래스 → 인스턴스(객체)

인스턴스화

클래스를 가지고 객체를 만드는 것을 인스턴스화라고 한다.

객체 사용

✓ 객체 사용 객체 안에 만들어져있는 변수, 메소드를 사용하겠다.

생성된 객체의 속성(Field)과 기능(Method)을 사용하는 것

객체 접근 연산자를 이용해서 객체에 접근

객체 접근 연산자 = 도트(.) 연산자

scan.next();
scan라는 객체 안에 있는 next라는 기능

✓ 객체 사용 형식

객체명.멤버변수명

객체명.멤버메소드명( )

객체 사용

✓ 객체 사용 예시

```
Member mb = new Member();      // 객체 생성
mb.name = "홍길동";            // 멤버변수에 값 대입
mb.setAge(20);                 // 멤버메소드 실행

// 출력
System.out.println("이름 : " + mb.name);
System.out.println("나이 : " + mb.getAge());
```

mb라는 객체 안에 있는 name이라는 변수 쓰겠다.

✓ 실행결과

```
이름 : 홍길동
나이 : 20
```

Chap03.
접근제한자

▶ 접근 제한자(Access Modifier)

✓ 접근제한자

외부에서 접근 가능한 멤버와 불가능한 멤버를 구분하는 용도로 사용

클래스의 **캡슐화**를 지원

✓ 캡슐화

추상화를 통해 정리된 데이터들과 기능을 하나로 묶어 관리하는 기법

데이터의 접근 제한을 원칙으로 함

외부에서는 노출된 필드와 메소드만 사용 가능

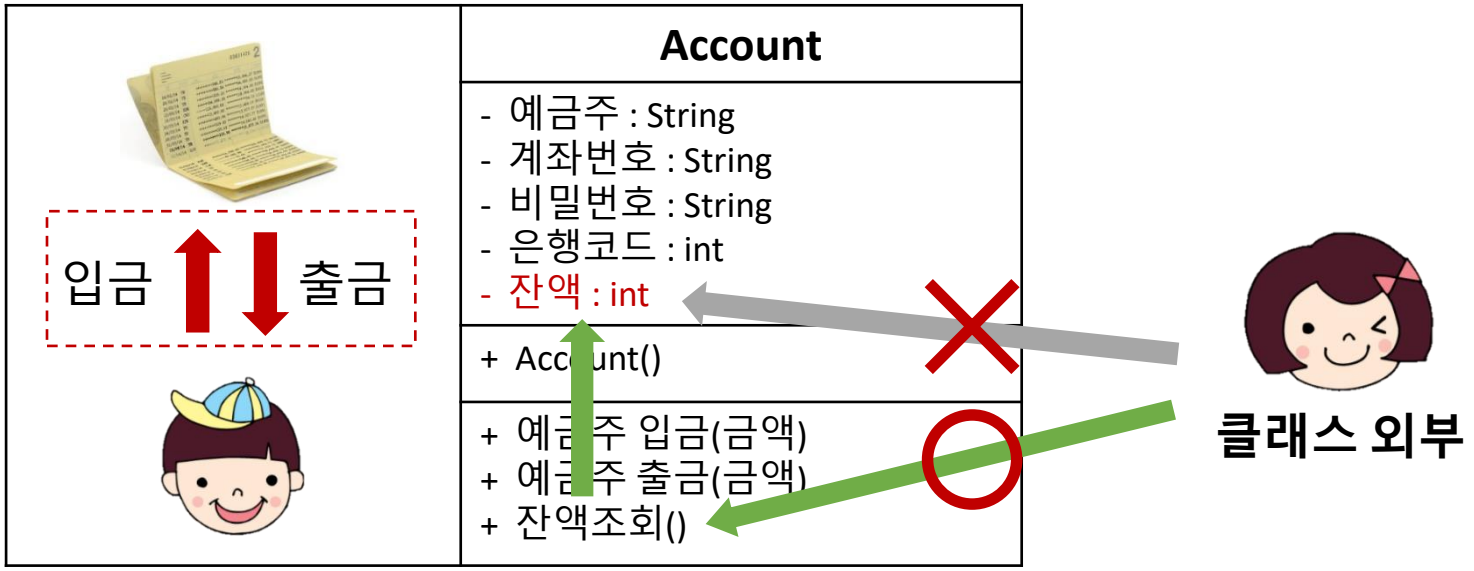
외부에 노출할 필요가 없는 부분은 감춰서 외부에서 잘못 사용하는 일이 없도록 하기 위함

접근제한자를 이용하여 노출시킬 부분과 감출 부분을 정함

▶ 접근 제한자(Access Modifier)

✓ 캡슐화 원칙

1. 클래스의 **멤버 변수에 대한 접근 권한은 private을 원칙**으로 한다.
2. 클래스의 멤버 변수에 대한 연산처리를 목적으로 하는 함수들을 클래스 내부에 작성한다.
3. **멤버 함수는 클래스 밖에서 접근할 수 있도록 public으로 설정**한다.



Account 클래스로 생성된 김철수 학생 명의의 계좌 객체

▶ 접근 제한자 종류

✓ (+) public

외부 클래스가 자유롭게 사용할 수 있는 공개 멤버

✓ (#) protected

같은 패키지 또는 자식 클래스에서 사용할 수 있는 멤버

✓ (~) private

외부에서 사용할 수 없는 멤버(class 내부에서만 사용할 수 있는 멤버)

✓ (-) default

접근제한자가 적용되지 않은 멤버로 같은 패키지에 소속된 클래스만 사용 가능한 멤버

▶ 클래스 접근 제한자

✓ 클래스 접근제한자

클래스는 **public , default 만** 사용 가능

클래스 생성 시 가장 앞에 public 을 생략하면 default 로 만들어 짐

생성자를 따로 만들지 않으면 default 생성자가 만들어지는데 이때 클래스와 동일한 접근제한자를 가짐

구분		같은 패키지 내	전체
+	public	O	O
~	(default)	O	

▶ 클래스 접근 제한자

KH정보교육원

✓ 클래스 접근제한자

[접근제한자] [예약어] class 클래스명 {

}

✓ 클래스 예시

```
public class 클래스명 {
        // ..............
}

class 클래스명 {          // 접근제한자 생략 시 default
        // ..............
}
```

▶ 필드 접근 제한자

KH정보교육원

✓ 필드 접근제한자

구분		해당 클래스 내부	같은 패키지 내	후손 클래스 내	전체
+	public	O	O	O	O
#	protected	O	O	O	
~	(default)	O	O		
-	**private**	O			

▶ 필드 접근 제한자

KH정보교육원

✓ 필드 접근제한자

[접근제한자] [예약어] class 클래스명 {

[접근제한자] [예약어] 자료형 변수명 [= 초기값];

}

✓ 필드 예시

```
public class Member {
        public int temp1;
        protected int temp2;
        int temp3;          // 접근제한자 생략 시 default
        private int temp4; // 캡슐화 원칙으로 private 사용
}
```

메소드 접근 제한자

✓ 메소드 접근제한자

구분		클래스	패키지	자손 클래스	전체
+	public	O	O	O	O
#	protected	O	O	O	
~	(default)	O	O		
-	private	O			

▶ 메소드 접근 제한자

KH정보교육원

✓ 메소드 접근제한자

```
[접근제한자][예약어] 반환형 메소드명(매개변수){
        //실행내용 작성
  }
```

▶ 메소드 접근 제한자

KH정보교육원

✓ 메소드 접근제한자 예시

```
public class Member {
    public String name;
    public int age;

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