

JAVA 객체지향I



**Chap01. 객체지향
프로그래밍이란?**

Chap02. 객체, 클래스

Chap03. 필드

Chap04. 메소드

Chap05. 생성자

**Chap06. package와
import**



Chap01. 객체지향 프로그래밍이란?

객체지향언어

현실 세계는 사물이나 개념처럼 독립되고 구분되는 각각의 객체로 이루어져 있으며, 발생하는 모든 사건들은 객체간의 상호 작용이다.

이 개념을 컴퓨터로 옮겨 놓아 만들어낸 것이 객체지향 언어이다.

객체지향 프로그래밍

프로그래밍에서는 현실세계의 객체(사물, 개념)를 클래스(class)와 객체(Object)의 개념으로 컴퓨터에서 구현한다.

Chap02. 객체, 클래스

객체의 사전적의미 : 실재하는 모든 사물. 살아있지 아닐 것.

객체 지향 언어에서 객체의 개념은 new연산자를 통해 클래스의 설계대로 데이터를 메모리에 할당한 결과물(instance)이다.

클래스(Class)

객체(Instance)

학생(Student)

객체화 instantiate

김철수

김영희

학생이 가져야할
속성, 기능을
추상화하여
클래스 정의

현실세계에
존재하는 고유 객체를
메모리 상에 할당.

클래스

객체를 정의해 놓은 것. 객체의 설계도, 틀.

사물이나 개념의 공통 요소(속성, 기능)를 용도에 맞게

추상화(abstraction) 함.

예) 제품의 설계도.

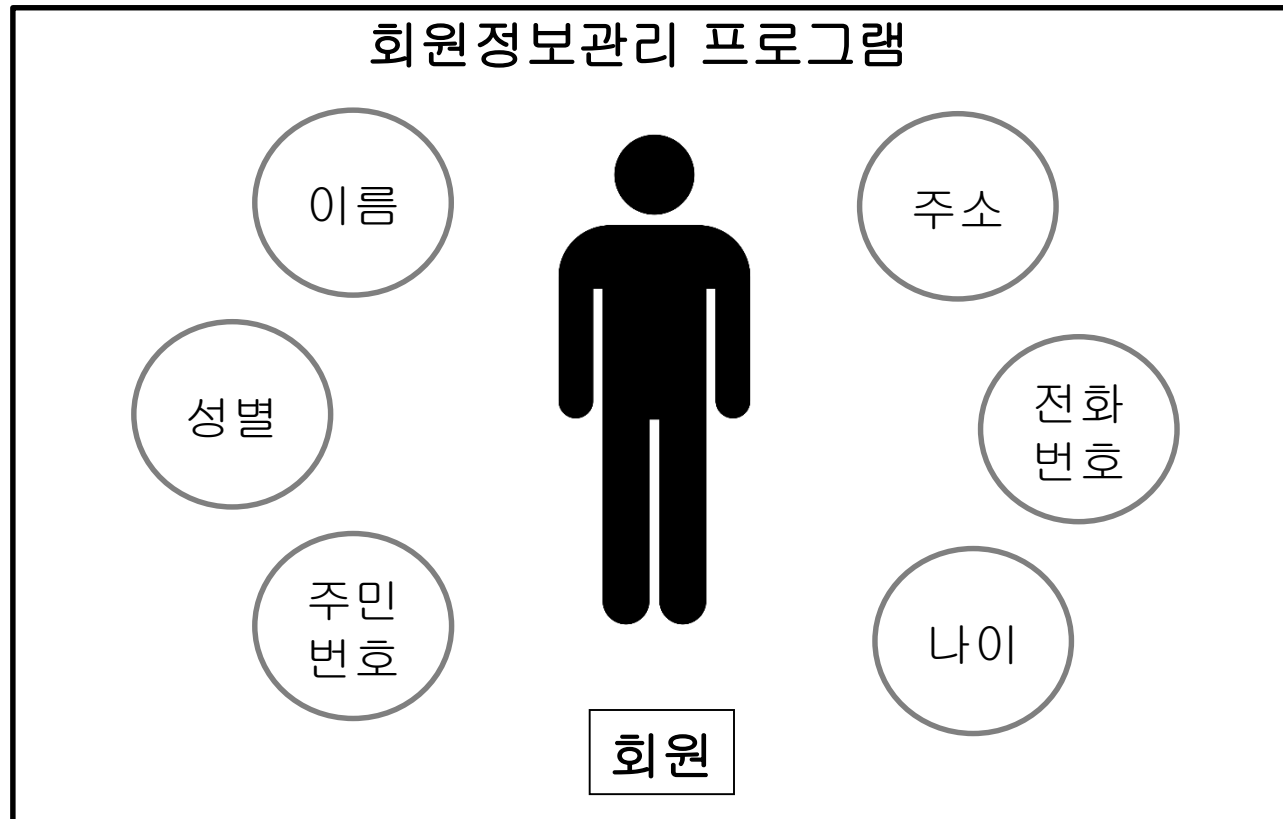
추상화(abstraction)

사전적의미 : 구체적 사물들의 공통된 특징.

프로그램에서 필요한 기능/속성을 추출하고, 불필요한 것을 제거하는 과정

추상화 Abstraction

기업에서 회원정보관리 프로그램을 만들려고 할 때,
프로그램에서 요구되는 “회원 객체”를 만들기 위해 추상화해본다.



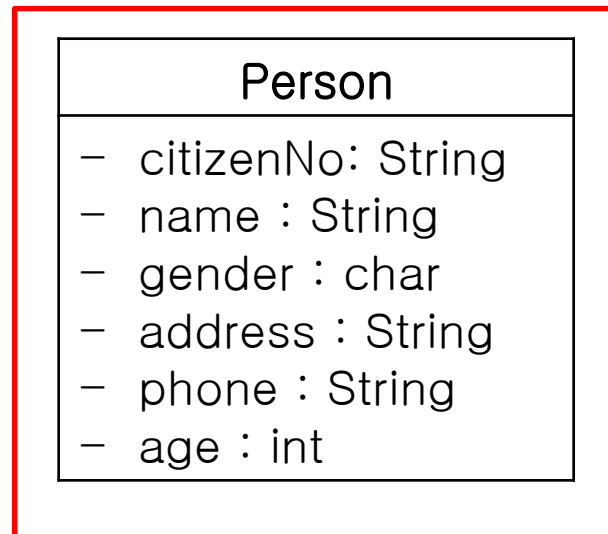
추상화(abstraction) → 클래스 작성

추상화한 결과물을 객체 지향 프로그래밍 언어를 사용해서
변수명(데이터이름)과 자료형(데이터타입)을 작성.

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

UML 다이어그램 표현

앞 페이지에서 정리된 변수명과 자료형을 클래스 다이어그램(UML)으로 표현한다면 아래와 같다.



클래스

다음 상황에서 객체지향적으로 프로그래밍을 하고자한다. 각각 클래스를 설계해보자.

- 1)야구 스포츠게임에서 플레이어객체를 만들고자 한다. 실제 야구게임에서 모티브를 얻어 추상화해보자.
- 2)결혼정보프로그램을 만들고자 한다. 회원객체를 설계해보자.
- 3)가구점에서 판매할 상품들에 대해 판매정보를 객체화하려고 한다. 어떤 클래스를 어떻게 설계하겠는가.

[접근제한자] [예약어] class 클래스명{

[접근제한자] 자료형 변수명;
[접근제한자] 자료형 변수명;

속성

[접근제한자] 생성자명(){}

기능

[접근제한자][예약어]리턴형 메소드명(){
//기능정의
}

}

public class Person{

**private String name;
private int age;**

속성

public Person(){}

기능

```
public String getName(){  
    return name;  
}  
public void introduce(){  
    System.out.println("반갑습니다. "  
        +age+"살, "+name+"입니다.");  
}
```

}

접근제한자

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

예) public class 클래스명 {

.....

}

class 클래스명{//생략하면 default클래스

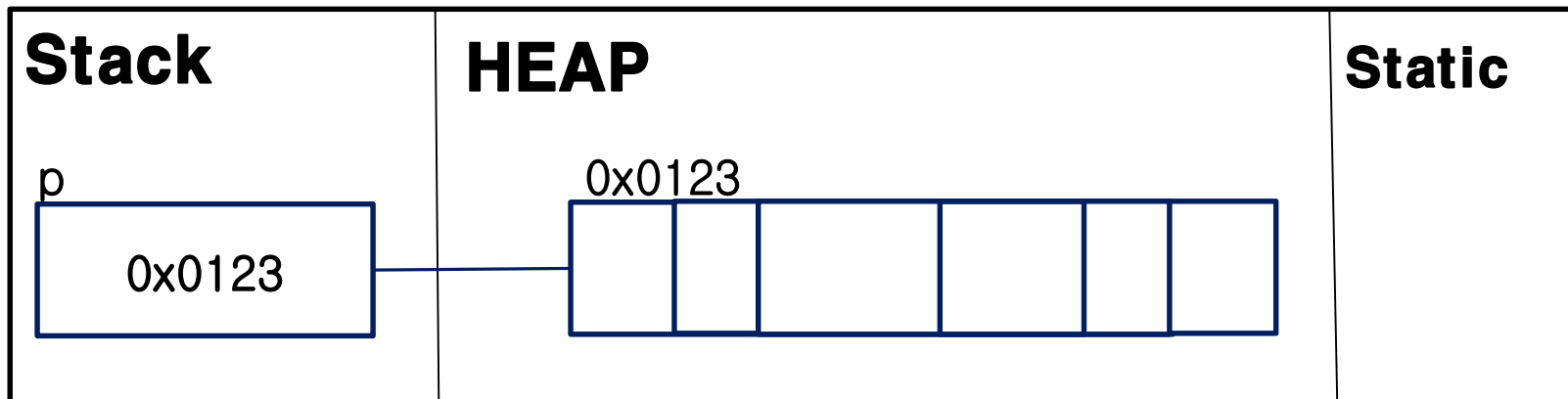
.....

}

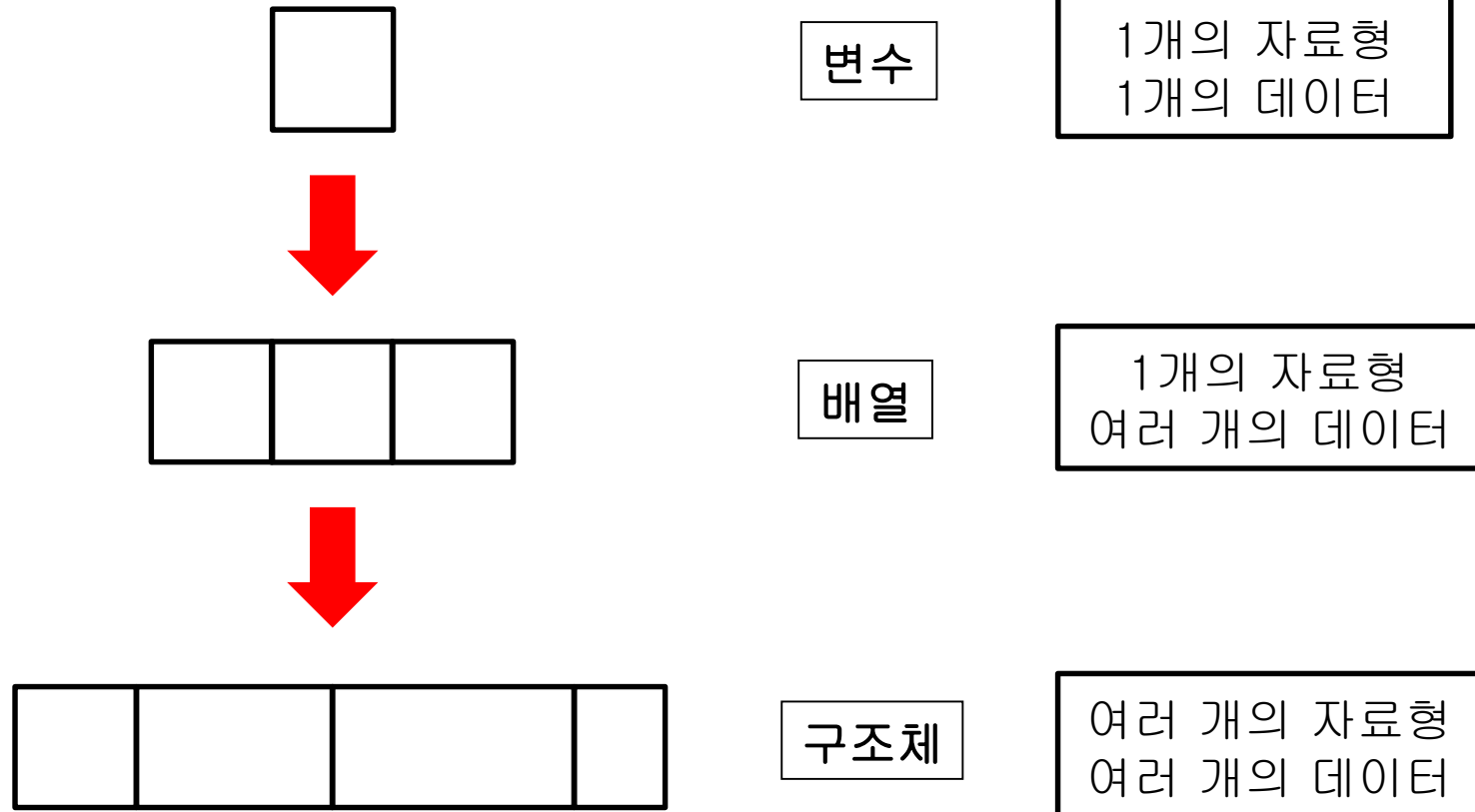
객체(Instance)의 할당

new 연산자(생성자)를 사용하여 객체를 생성하면 heap 메모리 공간에 서로 다른 자료형의 데이터가 연속으로 나열 할당된 객체 공간이 만들어진다.
이것을 인스턴스(instance)라고 한다.

예) Person p = new Person();



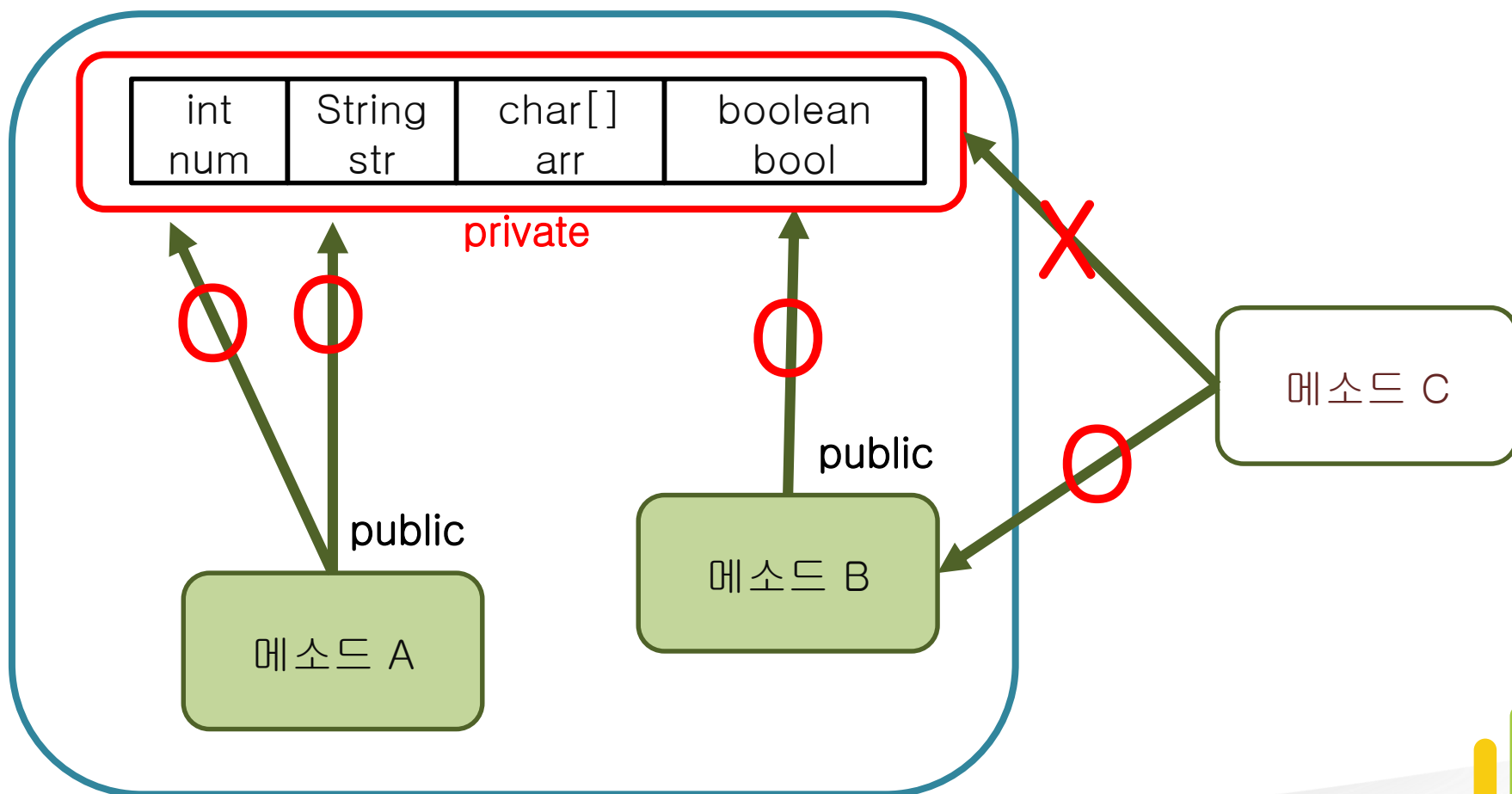
클래스의 등장 배경



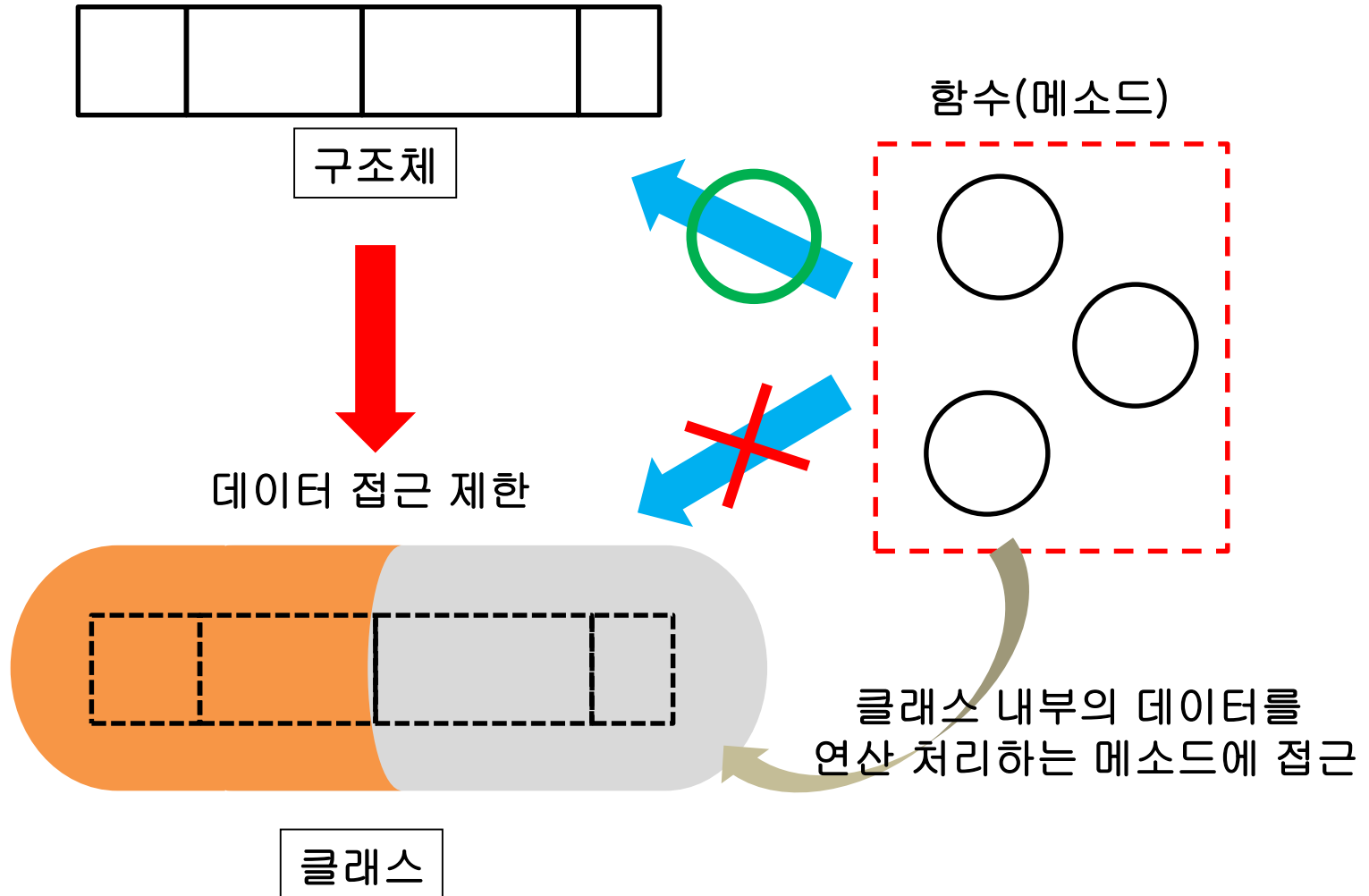
클래스의 등장 : 필드/메소드

여러 데이터타입을 가진 구조체의 보안문제로 인해 직접접근을 금지

→ 이게 데이터냐?



클래스의 등장 배경 - 캡슐화

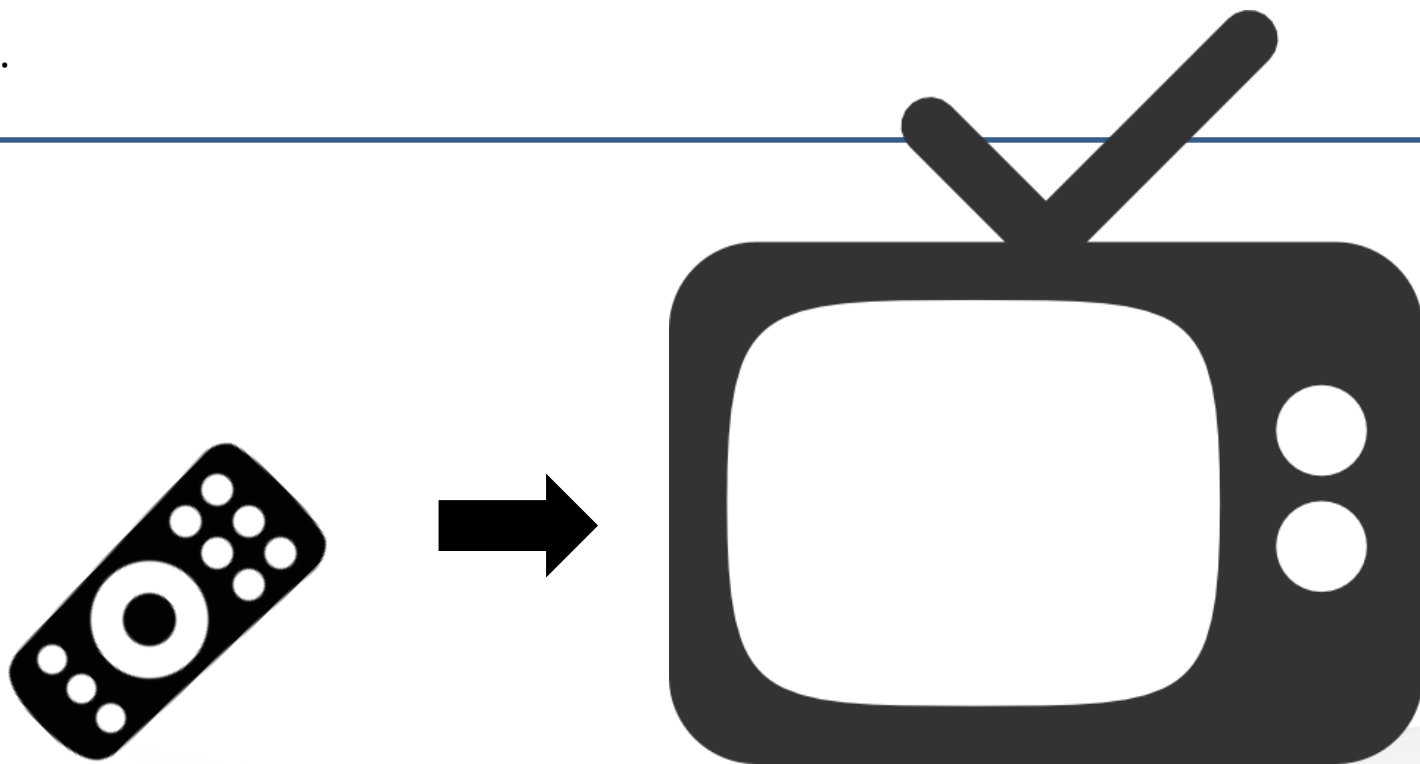


캡슐화(encapsulation)

상속(inheritance)

다형성(polymorphism)

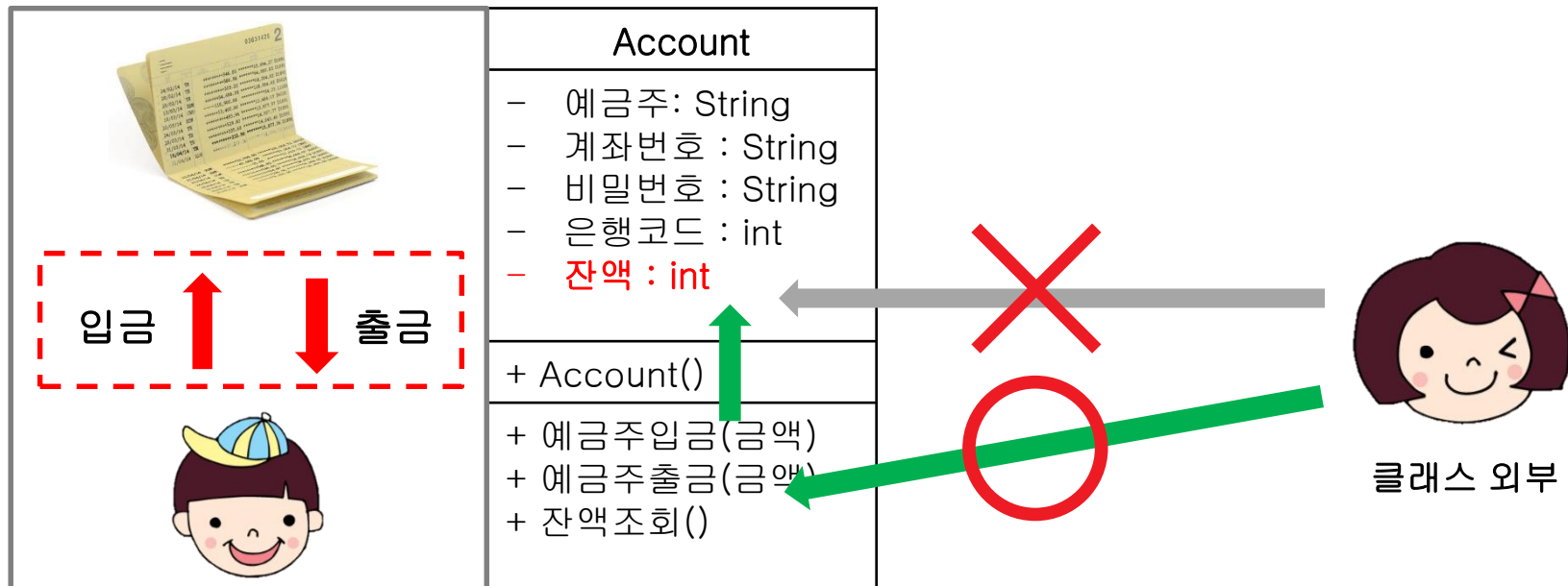
추상화를 통해 정의된 데이터들과 기능을 하나로 묶어 관리하는 기법이다. 클래스의 가장 중요한 목적인 데이터의 접근제한을 원칙으로 하여 클래스 외부에서 데이터의 직접 접근을 막고, 대신 데이터를 처리하는 함수(메소드)들을 클래스 내부에 작성하여 데이터에 접근하는 방식을 캡슐화라고 한다.



캡슐화 원칙

필드 : 클래스의 멤버 변수에 대한 접근권한은 **private**을 원칙으로 한다.

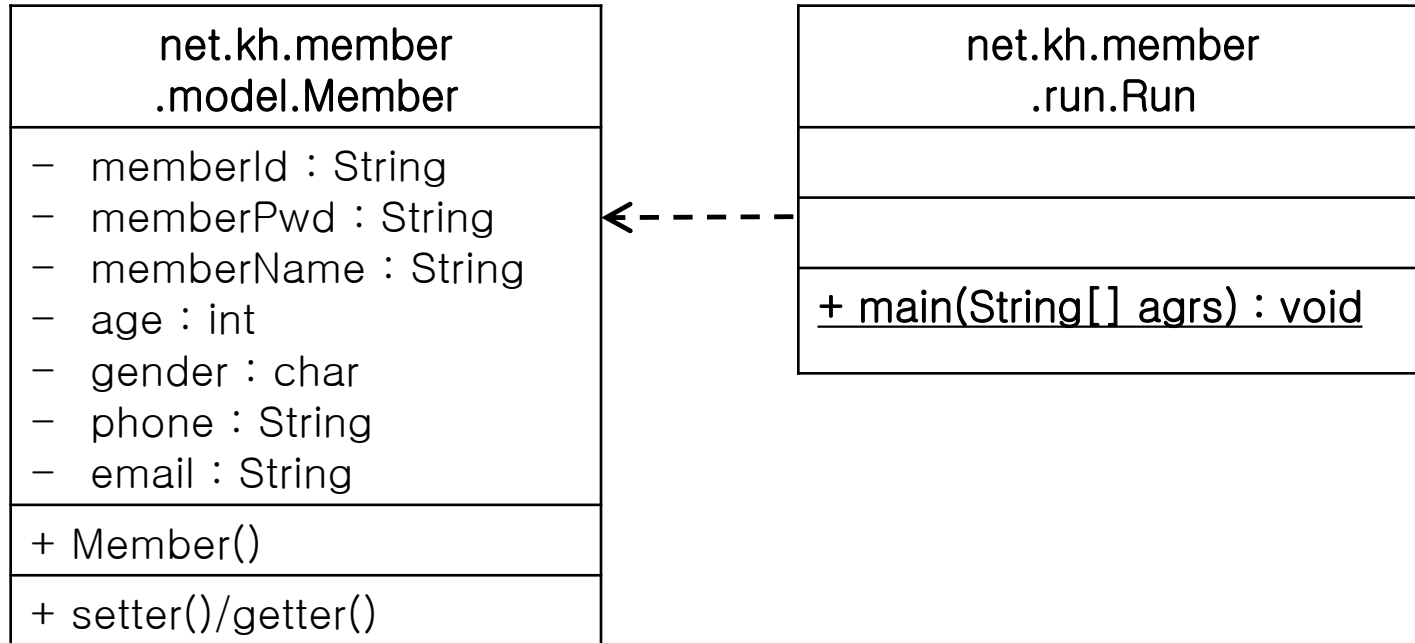
메소드 : 클래스의 멤버 변수에 대한 연산처리를 목적으로 하는 메소드를
클래스 내부에 작성하고, 클래스 밖에서 접근할 수 있도록
public으로 설정한다.



Account 클래스로 생성된 김철수학생 명의의 계좌 객체

실습문제1

아래 클래스 다이어그램을 보고 클래스를 작성하세요.

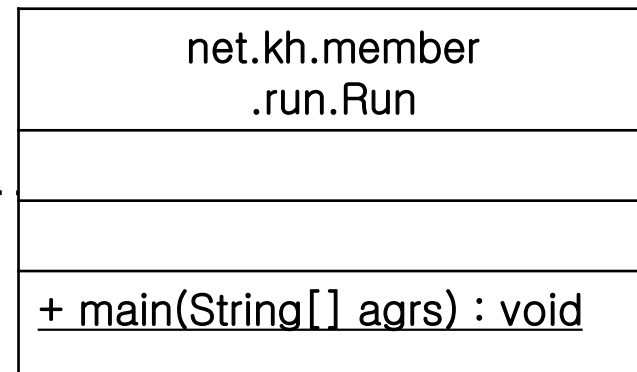
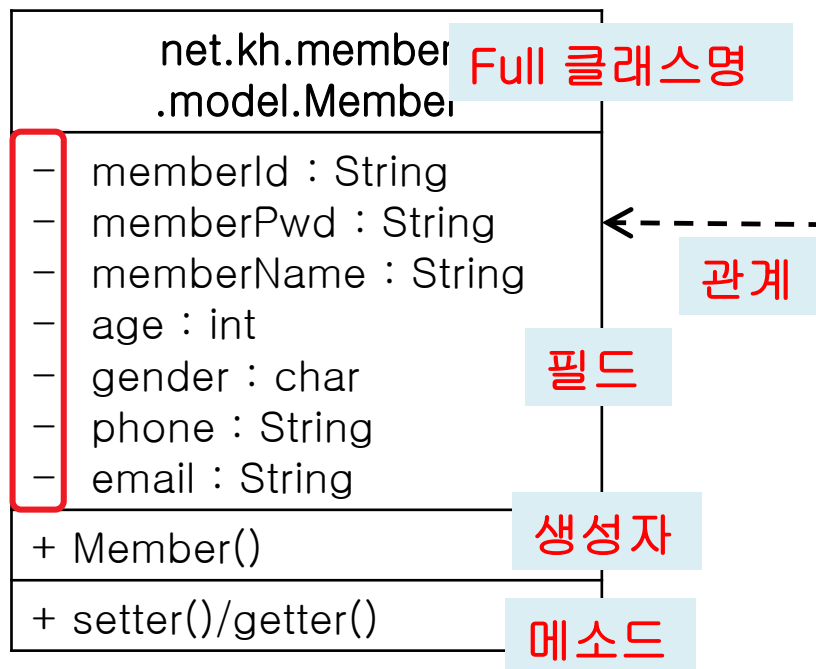


package	class	method	설명
net.kh.member.run	Run	<u>Main(args:String[])</u> : void	실행용 메소드 기본 생성자로 객체를 만들고 Setter를 이용해 값 변경 후 Getter를 이용해 출력







UML 다이어그램 이해하기

접근제한자

- private
+ public
~ default
protected

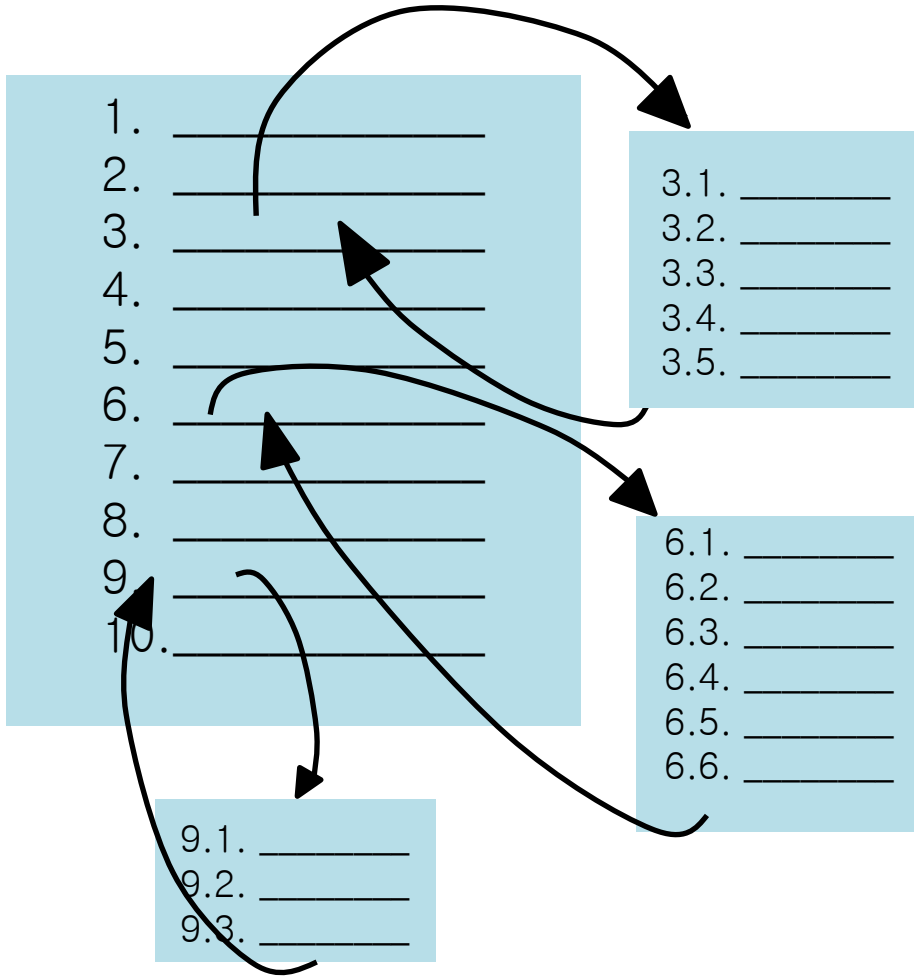


UML 다이어그램 이해하기

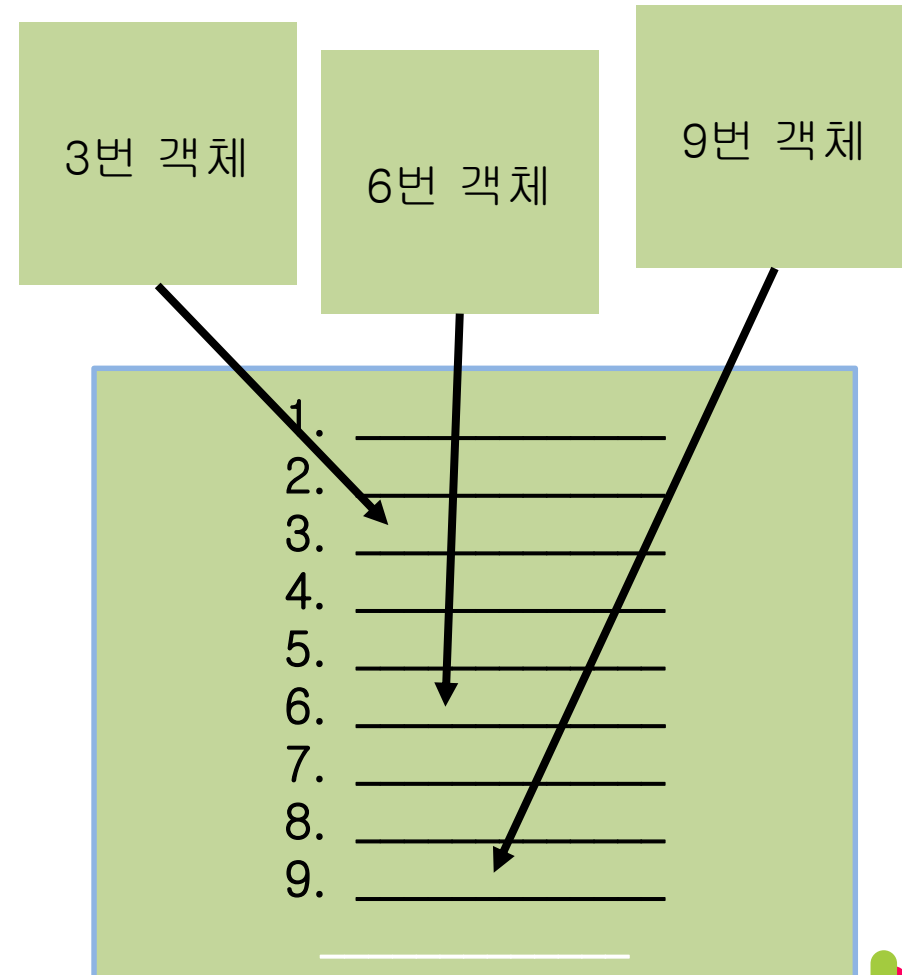
관계	UML 표기	설명
Genelization 일반화		상속관계 자식클래스→부모클래스
Realization 실체화		구현관계 구현클래스→인터페이스
Dependency 의존		객체생성, 객체사용 사용클래스→피사용클래스
Association 연관		필드로 다른 객체를 참조. 사용클래스→피사용클래스
Aggregation 약집합		필드로 다른 객체를 참조. 1:다 사용클래스→피사용클래스
Composition 강집합 (완전포함)		필드로 다른 객체를 참조. 1:1 사용클래스→피사용클래스

절차지향 vs 객체지향

절차지향식



객체지향식



Chap03. 필드(field)



변수의 선언위치에 따라 구분한다.

1. 클래스변수 : 클래스영역에 static키워드를 가짐.
2. 멤버변수(인스턴스변수) : 클래스영역에 선언.
3. 지역변수 : 클래스영역이 아닌, 메서드, 생성자, 초기화블럭내부에서 선언

```
class Var {  
    public static int size;  
    public int num;  
  
    public void test(){  
        int num = 100;  
    }  
}
```

클래스영역

메소드영역

변수에 따른 소멸시기

변수구분	소멸시기
클래스변수	프로그램 종료시
멤버변수(인스턴스변수)	객체소멸시(GC 소관)
지역변수	메소드 종료시

클래스변수 예약어 - static

static : 같은 타입의 여러 객체가 공유할 필드에
사용하며, 프로그램 **start**시에 정적 메모리영역에
자동 할당되는 멤버에 적용한다.

표현식

```
접근제한자 class 클래스명 {  
    접근제한자 static 자료형 변수명;  
}
```

```
예) public class Student{  
    private static String academy = "kh정보교육원";  
    private String name;  
}
```

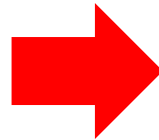
예) class Student{
 public int var1;
 protected int var2;
 int var3; //접근제한자 생략하면 default임
 private int var4; //캡슐화의 원칙임
}

필드 접근제한자

구분		해당클래스 내부	같은 패키지 내	후손 클래스 내	전체
+	public	○	○	○	○
#	protected	○	○	○	
~	(default)	○	○		
-	private	○			

필드 접근제한자

kh.java.edu.Hello
- a : String = "a" + b : int = 10 ~ c : String = "c" - <u>d : String = "d"</u>
+Hello()
+getter(), setter()



```
kh.java.World

public static void main(String args[]){

    Hello h = new Hello();

    System.out.println(h.a);
    System.out.println(h.b);
    System.out.println(h.c);
    System.out.println(h.d);

}
```


final : 하나의 값만 계속 저장해야 하는 변수에 사용
표현식

class 클래스명 {

접근제한자 **final** 자료형 변수명 = 초기값;

}

예) class Product {

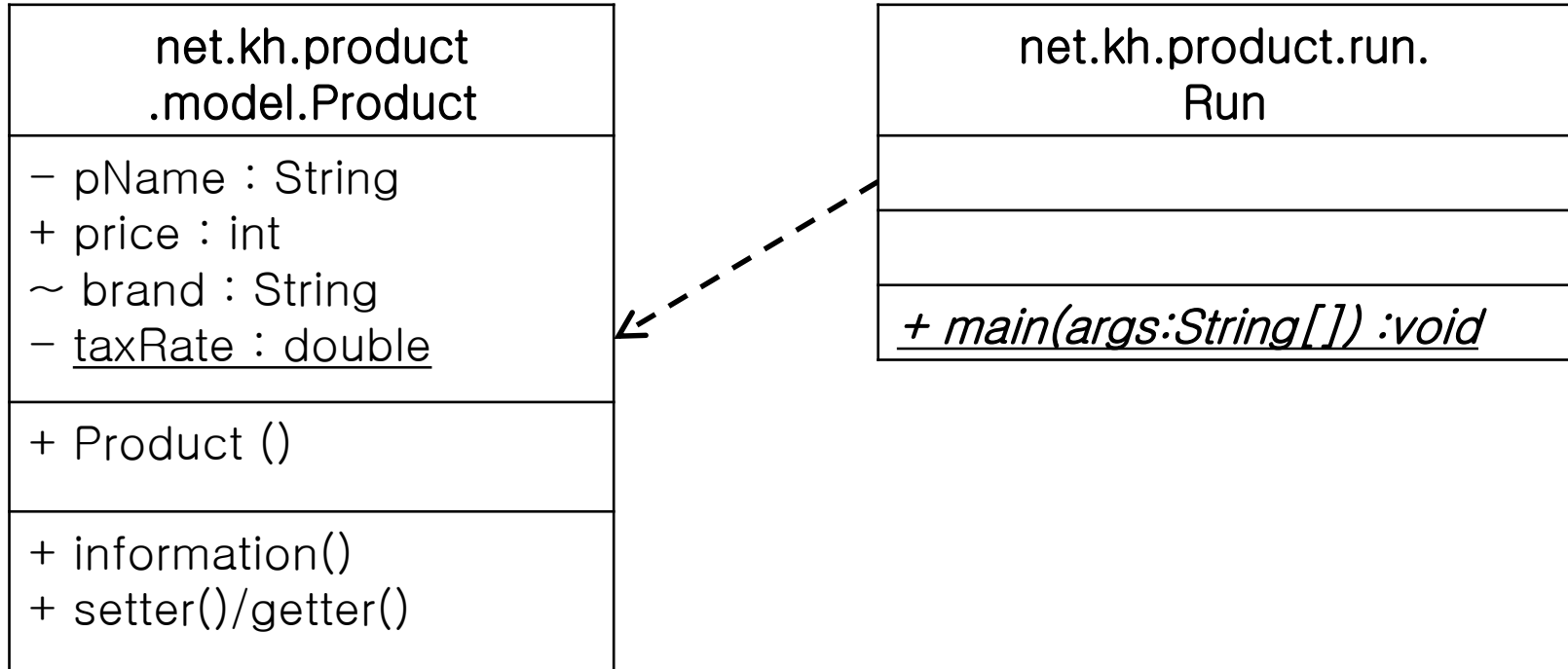
public static final double TAX = 3.3;

private int price;

}

실습문제2

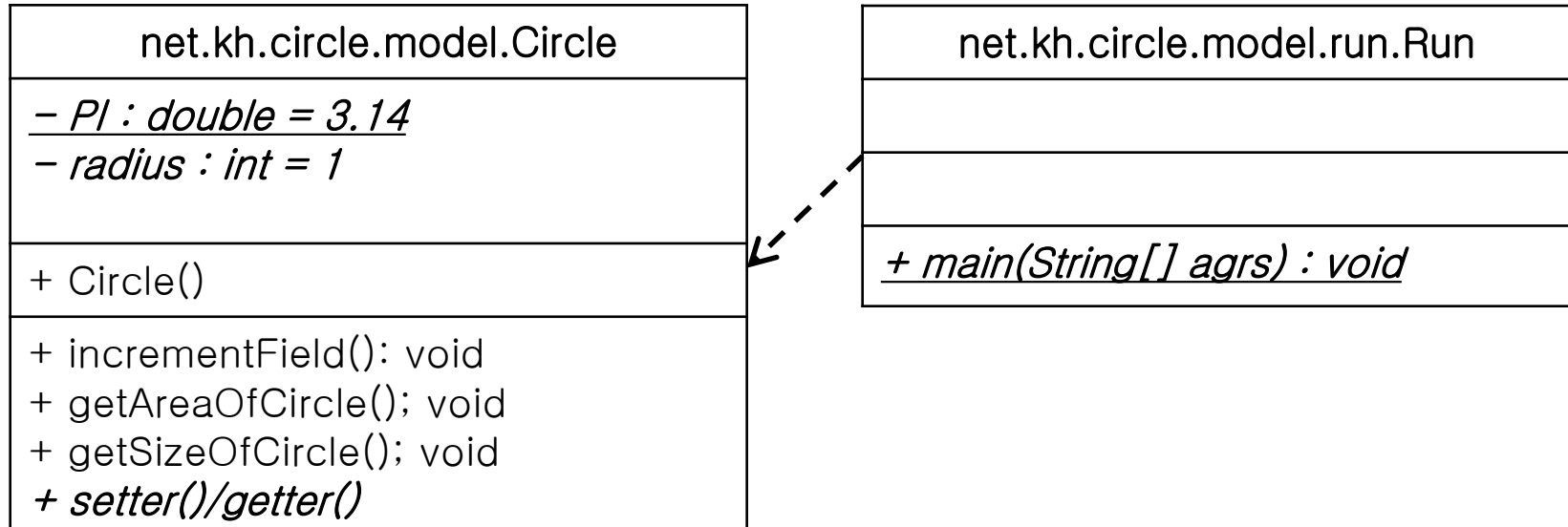
아래 클래스 다이어그램을 보고 클래스를 작성하세요.



package	class	method	설명
net.kh.product.run	Run	<u>Main(args:String []): void</u>	실행용 메소드 Product객체생성후 setter를 통 해 값을 지정 information()으로 출력 각필드에 직접접근해서 값을 출 력. 접근이 불가능한 필드가 있다 면, 그 이유에 대해 주석달기.

실습문제3

아래 클래스 다이어그램을 보고 클래스를 작성하세요.



package	class	method	설명
net.kh.circle.run	Run	<u>Main(args:String[]): void</u>	실행용 메소드 필드에 값 셋팅 후 원주, 원너비 구하기. incrementField를 통해 반지름 1증가후, 다시 원주/원너비 구하기

{ } : 클래스의 멤버 변수를 초기화 시키는 블록

static 블록 : static 필드 초기화, 프로그램 시작시 초기화

인스턴스 블록 : 인스턴스변수 초기화, 객체 생성시 마다
초기화

표현식

접근제한자 **class** 클래스명 {

접근제한자 **static** 자료형 필드명;

접근제한자 자료형 필드명;

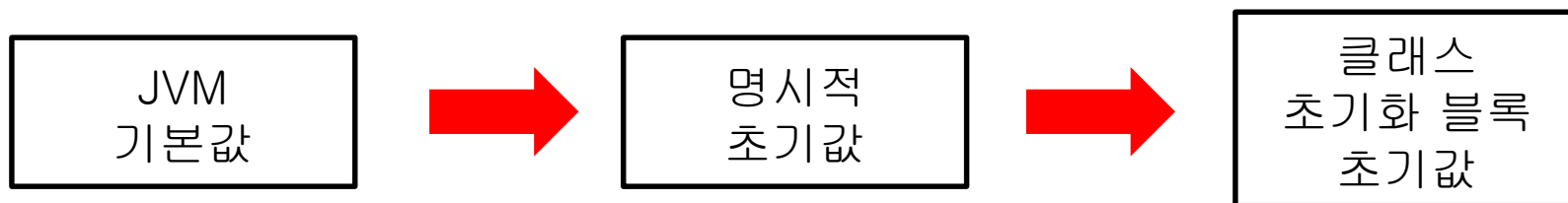
static{ 필드 = 초기값 };

{ 필드 = 초기값 };

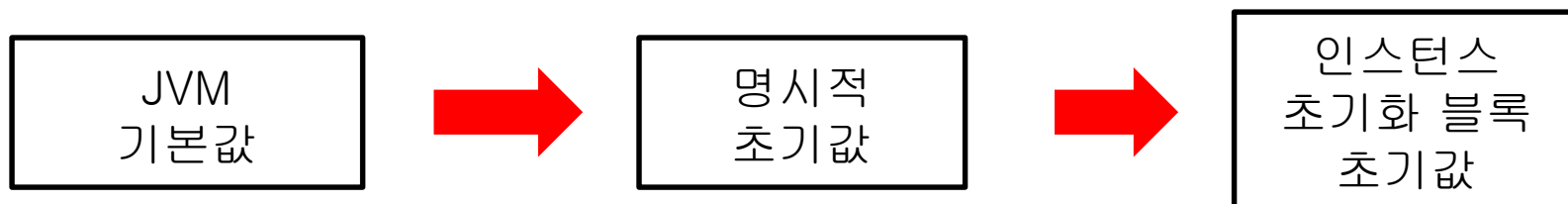
}

초기화 순서

- 클래스 변수



- 인스턴스 변수



실습문제

다음 코드의 콘솔결과를 예상해보고, 실제 출력값을 확인하세요.

net.kh.init.Sample

```
private int a = 10;
public static int b;

{
    System.out.println(a);
    a = 100;
    System.out.println(a);
}

static {
    System.out.println(b);
    b = 99;
    System.out.println(b);
}
```

net.kh.init.Run

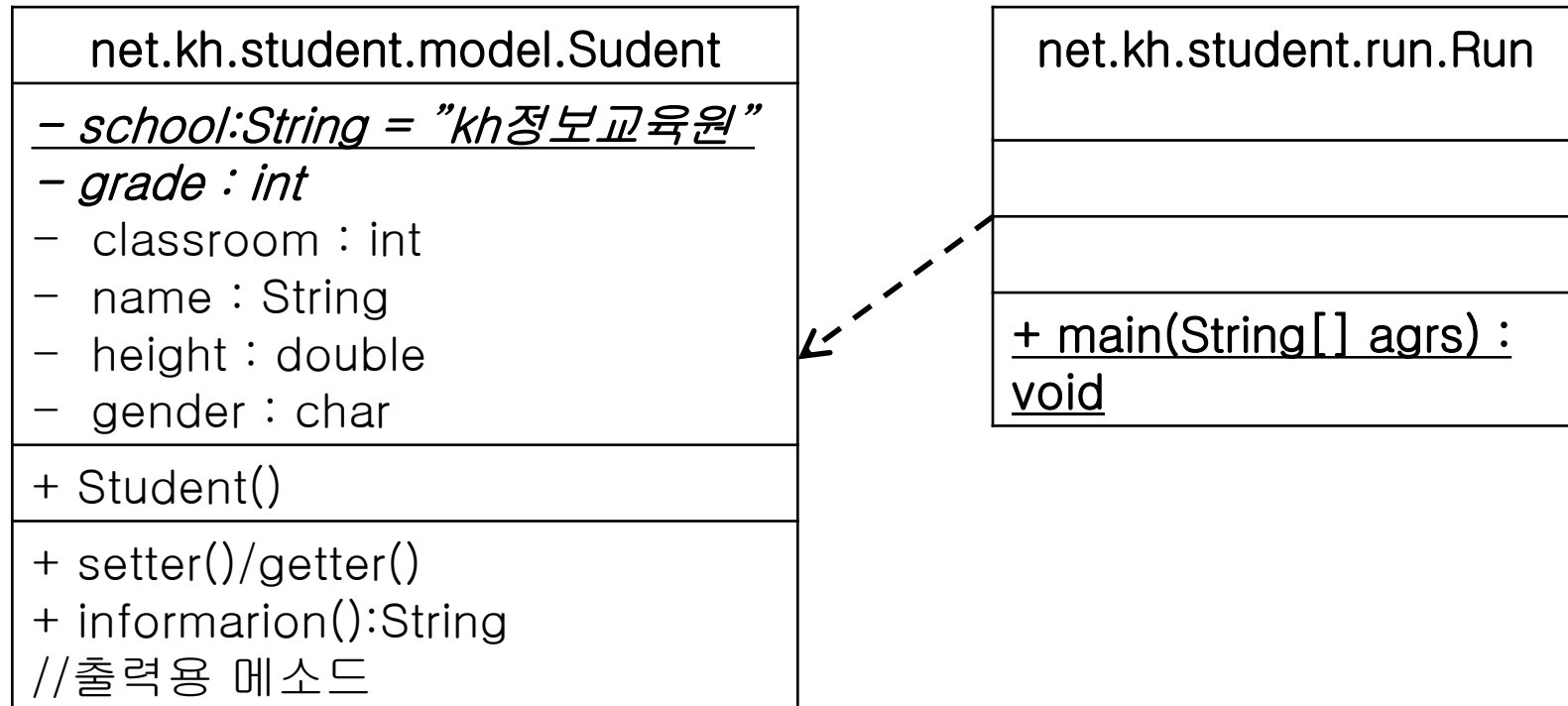
```
public static void main(String args[]){

    Sample s1 = new Sample();
    Sample s2 = new Sample();

}
```

실습문제4

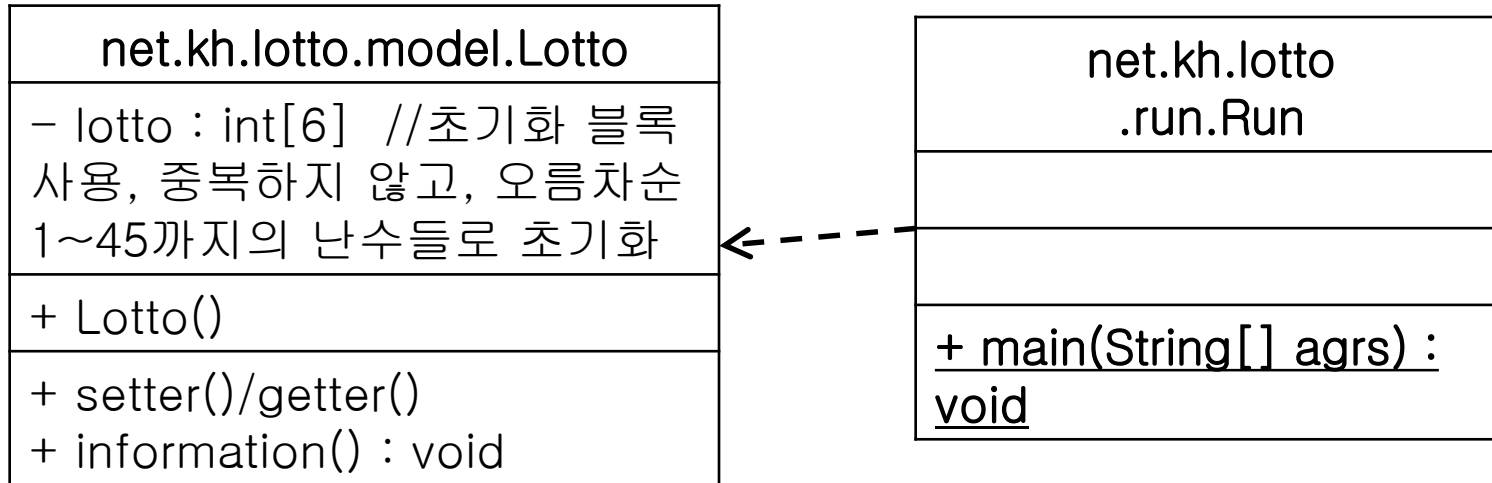
아래 클래스 다이어그램을 보고 클래스를 작성하세요.



package	class	method	설명
net.kh.student.controller	Run	<u>Main(args:String []): void</u>	실행용 메소드 Student 객체 생성 후 setter이 용 값 대입. Information()으로 출력

실습문제5

아래 클래스 다이어그램을 보고 클래스를 작성하세요.



package	class	method	설명
net.kh.lotto.model.vo	Lotto	informarion():String	반복문을 이용하여 배열을 출력하는 메소드
net.kh.lotto.run	Run	<u>Main(args:String[]) : void</u>	실행용 메소드 Lotto객체 생성 후 Information()으로 출력

Chap04. 메소드(method)

메소드란

수학의 함수와 비슷하며 호출을 통해 사용한다. 호출시 전달값은 있거나 없을수 있다. 함수가 호출되면, 내부에 작성된 연산을 수행하게되며, 연산 후 결과값/반환값은 있거나 없을 수 있다.



표현식

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

```
예) public void showInformation(){  
    System.out.println(userName);  
}
```

메소드의 접근제한자

구분		해당클래스 내부	같은 패키지 내	후손 클래스 내	전체
+	public	○	○	○	○
#	protected	○	○	○	
~	(default)	○	○		
-	private	○			

메소드의 예약어

구 분	설 명
static	static 영역에 할당하여 객체 생성 없이 사용함
final	종단의 의미, 상속시 오버라이딩 불가능
abstract	미완성된, 상속하여 오버라이딩으로 완성시켜 사용해야 함
synchronized	동기화처리, 공유 자원에 한 개의 스레드만 접근 가능함
static final (final static)	static과 final의 의미를 둘 다 가짐

구 분	설 명
void	반환값이 없을 경우
기본자료형	반환값이 8가지 기본 자료형일 경우
배열	기본형, 클래스의 배열 모두 가능.
클래스	반환값이 해당 클래스 타입의 객체일 경우 사용함 (클래스 == 타입)

* 반환형은 단 한 개만 가질 수 있다.

메소드의 매개변수

구 분	설 명
()	매개변수가 없는것을 의미함
기본자료형	기본형 매개변수 사용시 값을 복사하여 전달함. 매개변수 값을 변경하여도 원래 값은 변경되지 않는다.
배열	배열, 클래스등 참조형을 매개변수로 전달시에는 데이터의 주소값을 전달하므로 매개변수를 수정하면 본래 데이터가 수정된다.
클래스	
가변인자	매개변수의 개수를 유동적으로 설정하는 방법, 가변매개변수와 다른 매개변수가 있으면 가변매개변수를 마지막에 설정해야 함. 방법 : (자료형 ... 변수명)

* 매개변수의 수에 제한이 없다.

오버로딩 Overloading

한 클래스 내에서 파라미터 선언부가 다르고, 이름이 같은 메서드를 여러 개 정의하는 것.



오버로딩 성립조건 2가지

1. 메소드 이름이 같아야한다.
2. 매개변수 선언부가 달라야한다.
 - 매개변수 타입, 개수, 순서

오버로딩 주의점

1. 매개변수명은 상관치 않는다.
2. 리턴타입은 오버로딩시 상관치 않는다.

java.io.PrintStream의 println() 메소드의 오버로딩을 살펴보자.


```
public int test(){}  
public int test(int a){}  
public int test(int a, int b){}  
public int test(int a, String s){}  
  
public int test(int b, int a){}  
public int test(String s, int a){}  
  
public String test(int a, int b, String str){}  
private int test(String str, int a, int b){}  
public int test(int a, long b, String str){}
```

메소드의 종류

매개변수가 없고 리턴값이 있거나 없거나

표현식

- 리턴값 있는것

[접근제한자] 리턴형 메소드명() {

.....
return 반환값;

}

- 리턴값 없는것

[접근제한자] void 메소드명() {

.....

}

메소드의 종류

매개변수가 있고 리턴값이 있거나 없거나

표현식

- 리턴값 있는것

접근제한자 리턴형 메소드명(자료형 변수명){

.....
return 반환값;

}

- 리턴값 없는것

접근제한자 void 메소드명(자료형 변수명){

.....

}

메소드 호출

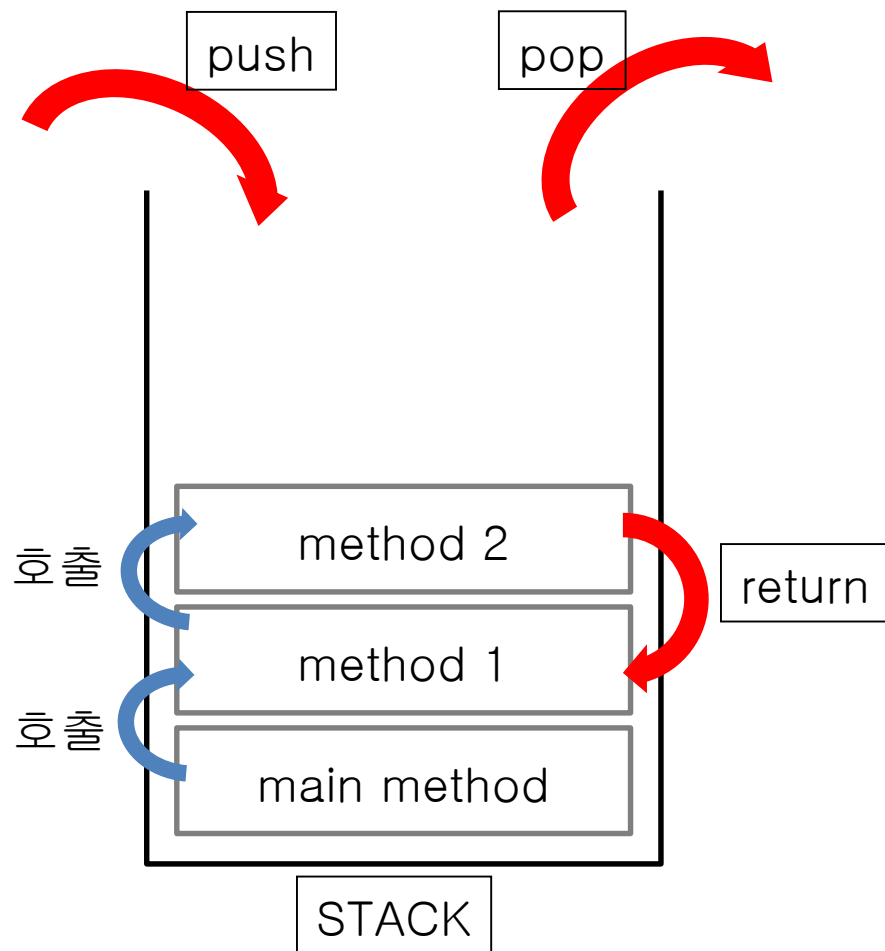
호출할 클래스명, 메소드명을 적고 설정된 매개변수와 동일한 자료형의 값을 전달.

표현식

매개변수 있는 경우 : 참조형 변수명.메소드명 (매개변수);

매개변수 없는 경우 : 참조형 변수명.메소드명();

```
public void showInformation(){  
    Person ps=new Person();  
    ps.setAge(19);           //매개변수 있는 경우  
    int age = ps.getAge() ;  //매개변수 없는 경우  
}
```



return이란?

- 해당 메소드를 종료하고, 자신을 호출한 메소드로 돌아가는 예약어
- 반환값(리턴값)을 가지고 자신을 호출한 메소드로 돌아갈 수 있다.

STACK의 자료구조

-LIFO(Last-Input-First-Out)

Setter 메소드

- 필드에 변경할 값을 전달 받아서 필드값을 변경하는 메소드
- 매개변수가 필드와 동일한 이름일 경우 **this**연산자를 붙여서 구분

```
접근제한자 void set필드명(자료형 변수) {  
    (this.)필드명 = 자료형 변수;  
}
```

Getter 메소드

- 필드에 기록된 값을 읽어서 요구하는 쪽으로 읽은 값을 넘기는 메소드

```
접근제한자 반환형 get필드명() {  
    return 필드명;  
}
```

Setter 메소드

```
public void setBalance(int balance){  
    this.balance = balance;  
}
```

Getter 메소드

```
public int getBalance(){  
    return balance;  
}
```

this란

모든 인스턴스의 메소드에 숨겨진 채 존재하는 지역변수로,
해당 객체의 주소값을 담고 있다.

```
예) class Academy{  
    String name;  
  
    public Academy(){  
    public Academy(String name){  
        this.name = name;  
    }  
}
```

매개변수를 가지는 생성자 또는 메소드에서 매개변수명이 필드명과 같을 경우
매개변수의 변수명이 우선하기 때문에 this를 활용해서 매개변수와 필드를
구분한다.

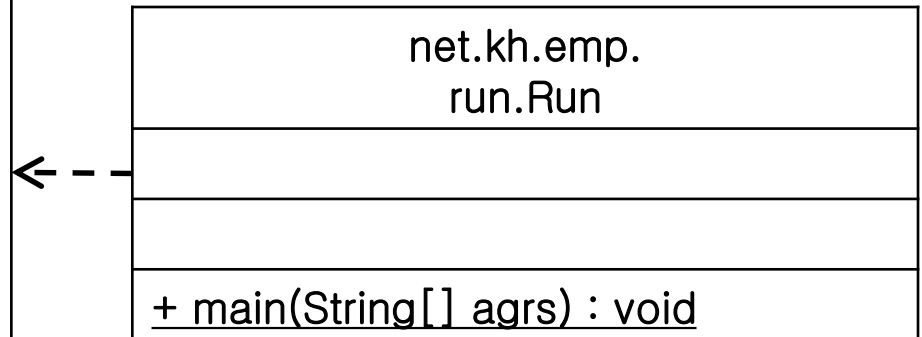
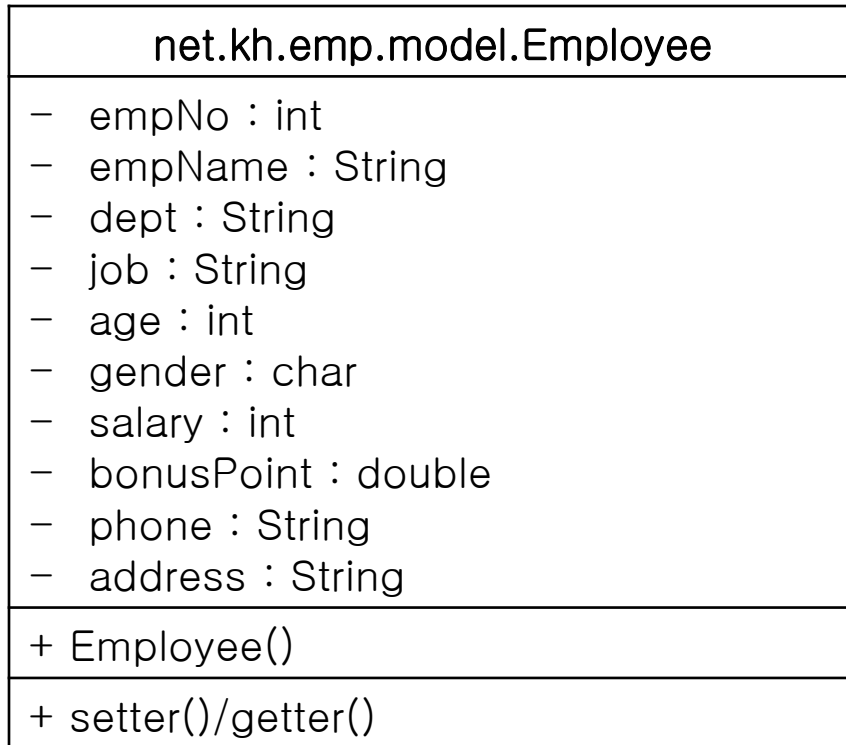
재귀호출(recursive call)

```
public class RecursionTest {  
    public static void main(String[] args){  
        RecursionTest r = new RecursionTest();  
        int result = r.factorial(5);  
        System.out.println(result);  
    }  
  
    public int factorial(int num) {  
        if(num==1)  
            return 1;  
        else  
            return num*factorial(num-1);  
    }  
}
```

실습문제6

아래 클래스 다이어그램을 보고 클래스를 작성하세요.

empNo	empName	dept	job	age	gender	salary	bonusPoint	phone	address
100	홍길동	영업부	과장	25	남	2500000	0.05	010-1234-5678	서울시 강남구



package	class	method	설명
net.kh.emp. run	Run	<u>Main(args:String[]): void</u>	실행용 메소드 기본 생성자로 객체를 만들고 Setter를 이용해 값 변경 후 Getter를 이용해 출력

Chap05. 생성자(constructor)

생성자란

new 연산자를 통해 실행되는 메소드로써, 객체가 heap에 할당될 때 객체 안에서 만들어지는 필드의 초기화를 담당함. 생성자를 통해 전달된 초기값이 있다면, 이를 필드에 기록한다.

생성자 규칙

생성자는 선언은 메소드 선언과 유사하나 리턴값이 없다.
클래스 이름이랑 생성자의 이름은 같다.

생성자 종류

기본 생성자 (매개변수가 없는 생성자)

- 작성하지 않은 경우, 클래스 사용시 JVM이 기본 생성자 자동 생성

매개변수 있는 생성자

- 객체 생성시 전달 받은 값으로 객체를 초기화 하기 위해 사용함.
- 매개변수 있는 생성자 작성시 JVM이 기본 생성자 자동 생성 하지 않음
- 상속 사용시 반드시 기본 생성자를 작성해야 함
- 오버로딩을 이용하여 작성함

클래스 매개변수 있는 생성자

표현식

```
class 클래스명 {
```

```
    [접근제한자] 클래스명(){} //기본 생성자
```

```
    [접근제한자] 클래스명(매개변수){
```

```
        (this.)필드명 = 매개변수
```

```
    };
```

```
}
```

클래스 매개변수 있는 생성자

클래스에 생성자를 만들면 디폴트 생성자가 자동으로 생성되지 않아 디폴트 생성자를 활용 하려면 코드를 추가 해줘야 함.

```
예) class Person{  
    private int age;  
    private String name;
```

```
    public Person(){} //직접추가해야함.
```

```
    public Person(int age, String name){  
        this.age = age;  
        this.name = name;  
    }
```

```
}
```

this란

모든 인스턴스의 메소드에 숨겨진 채 존재하는 지역변수로,
해당 객체의 주소값을 담고 있다.

```
예) class Student{  
    String name;  
  
    public Student(){  
    public Student(String name){  
        this.name = name;  
    }  
}
```

매개변수를 가지는 생성자 또는 메소드에서 매개변수명이 필드명과 같을 경우
매개변수의 변수명이 우선하기 때문에 this를 활용해서 매개변수와 필드를
구분한다.

오버로딩 Overloading – 생성자

한 클래스 내에서 **파라미터선언부가 다른 생성자**를 여러 개 정의하는 것.



오버로딩 성립조건 2가지

1. 메소드 이름이 같아야한다.
2. 매개변수 선언부가 달라야한다.
 - 매개변수 타입, 개수, 순서

java.util.Scanner의 생성자 오버로딩을 살펴보자.

오버로딩 Overloading - 생성자

```
class Truck {  
    public Truck(){}  
  
    public Truck(String str){}  
  
    public Truck(String k){}  
  
    public Truck(String str, int x, int j){}  
  
    public Truck(int x, int j, String str){}  
}
```



this()

this()란


생성자, 같은 클래스의 다른 생성자를 호출할 때 사용

**** 반드시 첫번째 줄에 선언해야 한다.**

예)

```
class Student{
    int age;
    String name;

    public Student(){
        this(20, "김철수");
    }
    public Student(int age, String name){
        this.age = age;
        this.name = name;
    }
}
```



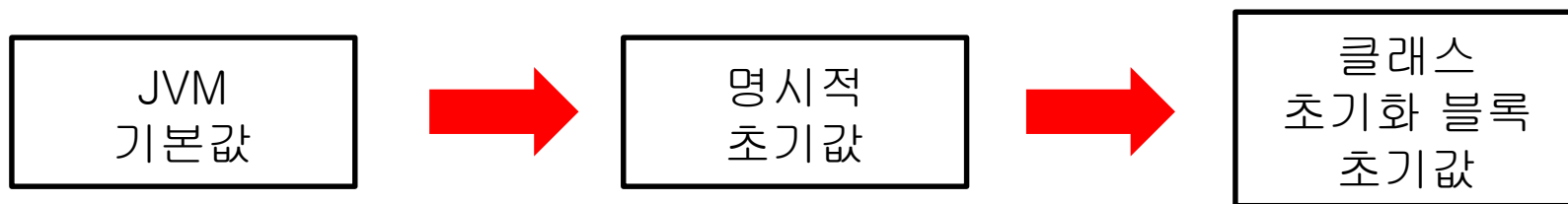
this()를 통한 실행순서

Suit s = new Suit(100); 를 실행한다면??

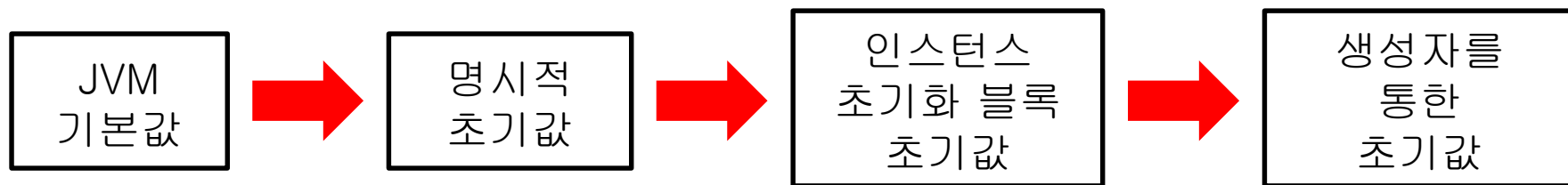
```
public class Suit {  
    private int size;  
    private String brand;  
    private int price;  
  
    public Suit(int size) {  
        this(size, "Hazzys");  
        System.out.println("Suit(int) 호출!!");    //(1)  
    }  
    public Suit(int size, String brand){  
        this(size, brand, 100000);  
        System.out.println("Suit(int, String)호출");    //(2)  
    }  
    public Suit(int size, String brand, int price){  
        this.size = size;  
        this.brand = brand;  
        this.price = price;  
        System.out.println("Suit(int, String,int)호출");    //(3)  
    }  
}
```

초기화 순서

- 클래스 변수

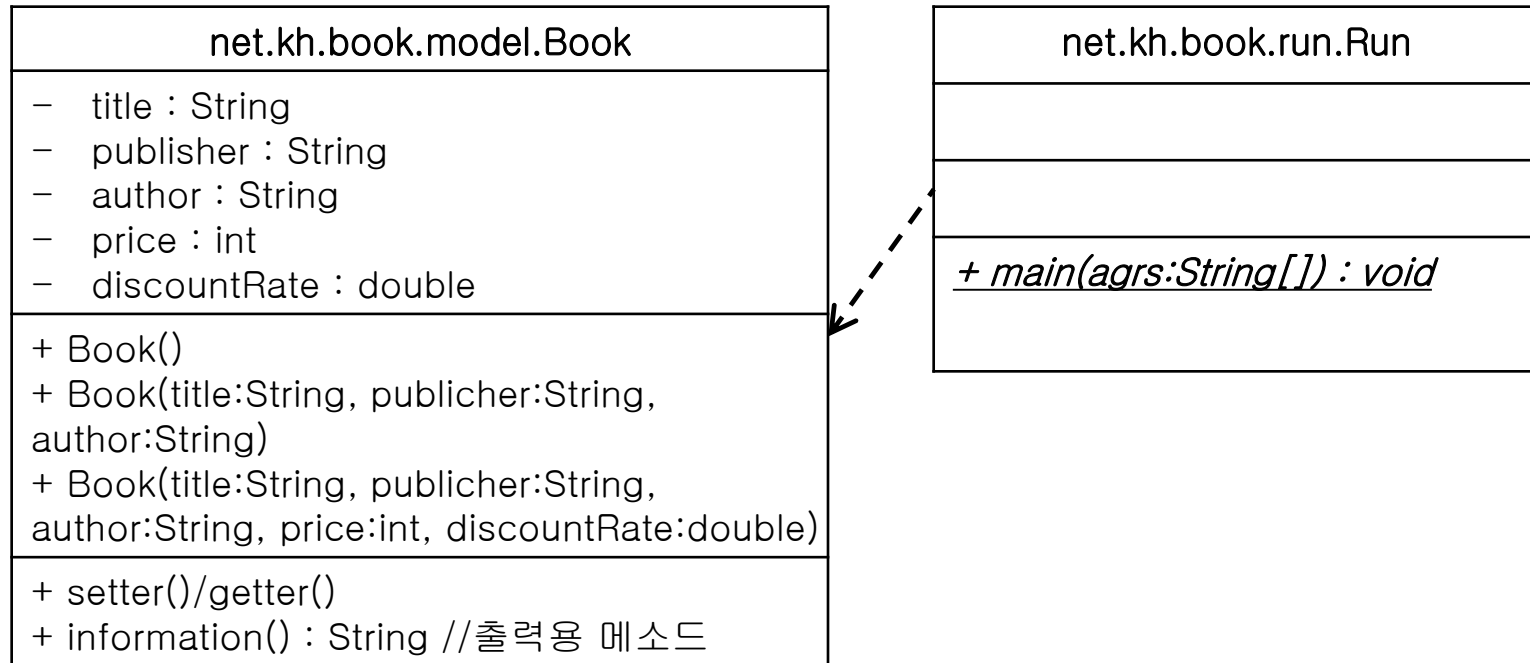


- 인스턴스 변수



실습문제7

아래 클래스 다이어그램을 보고 클래스를 작성하세요.



package	class	method	설명
net.kh.book.model.vo	Run	<u>Main(ags:Strin g[]) : void</u>	실행용 메소드 매개변수 3개인 생성자와 매개변수 5개인 생성자로 각각 객체를 생성하여 Information()이용하여 출력. 사용자 입력값 없을시 기본값 출력(0)

실습문제8

아래 클래스 다이어그램을 보고 클래스를 작성하세요.

net.kh.book2.model.vo.Book
<ul style="list-style-type: none"> - label: String - year: int - bookName: String - author: String - publisher: String - genre: char - stock: int - location: String
+ Book() + Book(year:int, bookName:String, author:String, publisher:String, genre:char, stock:int, location: String)
+ setter()/getter() + bookInfo() : String //출력용 메소드

net.kh.book2.run.Run
<u>+ main(String[] agrs) : void</u>

※ label 변수는
year+"_"+genre+"_"+location의
조합으로 이루어진 문자열이다.
생성자매개변수로 직접 입력되지
않지만, bookInfo()메소드에서는
출력되어야 한다.

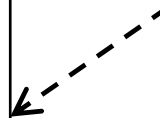
package	class	method	설명
net.kh.book2.run	Run	<u>Main(args:String[]) : void</u>	실행용 메소드 매개변수 7개인 생성자로 객체 생성후, bookInfo()을 사용하여 출력

실습문제9

아래 클래스 다이어그램을 보고 클래스를 작성하세요.

net.kh.product2.model.vo.Product
<ul style="list-style-type: none"> - sort: String - productName: String - price: int - stock: int - isDiscount: boolean - discountRate: double
+ Product() + Product(sort:String, productName:String, price:String, stock:int, isDiscount: boolean) + Product(sort:String, productName:String, price:String, stock:int, isDiscount: boolean, discountRate: double)
+ setter()/getter() + productInfo() : String //출력용 메소드

net.kh.product2.run.Run
<u>+ main(String[] agrs) : void</u>



※ 할인여부에 따라서 생성자를 각각 만들어보자.
 할인여부가 true이면,
 discount금액을 매개변수로 전달한다.
 productInfo()출력시에도
 isDiscount가 true일때만 할인율이 적용된 price를 출력한다.

package	class	method	설명
net.kh.product2. run	Run	<u>Main(args:String[]): void</u>	실행용 메소드 매개변수 5,6개인 생성자를 이용해서 객체 3개 생성후 productInfo()을 사용하여 출력.

Chap06. package와 import

패키지란?

서로 관련된 클래스 혹은 인터페이스의 묶음이다.

클래스가 클래스파일(*.class)인 것 처럼, 패키지는 폴더와 비슷하다.

패키지는 서브 패키지를 가질 수 있으며 ‘.’으로 구분한다.

예시) Scanner클래스의 full name은 패키지명이 포함된
java.util.Scanner이다.



패키지의 선언

패키지는 소스 파일 첫 번째 문장에 단 한번 선언한다.

하나의 소스 파일에 둘 이상의 클래스가 포함된 경우, 모두 같은 패키지에 속한다.

모든 클래스는 하나의 패키지에 속하며, 패키지가 선언되지 않은 클래스는 지동적으로 이름없는 패키지(default)에 속하게 된다.

예시) `package java.util;`



import란?

사용할 클래스가 속한 패키지를 지정하는데 사용

import문을 사용하면 클래스를 사용할 때 패키지명을 생략.

java.lang 패키지의 클래스는 JVM이 자동으로 import함. 사용자가 import할 필요없음.(String, Object, System...)



import문의 선언

import문은 패키지명과 클래스 선언의 사이에 선언한다.

import문은 컴파일시에 처리되므로 프로그램 성능에 아무런 영향을 주지 않는다.

와일드카드 * 를 이용해서 지정된 패키지에 포함된 클래스는 모두 import 가능하지만, **서브패키지에 속하는 모든 클래스까지**

import는 불가능하다.

예) `import java.util.Calendar;`
`import java.util.Date;`
`import java.util.*;`
`import java.text.*;`
`import java.*; //java.util.* , java.text.*를 대체불가`

import문 주의사항

이름이 같은 클래스가 속한 두 패키지를 import 할 때는 클래스 앞에 패키지명을 붙여 구분해 주어야 한다.

예시) `import java.sql.Date;`
`import java.util.Date;`

```
public class ImportTest{  
    public static void main(String[] args){  
        java.util.Date today = new java.util.Date();  
    }  
}
```



클래스의 static import문

클래스명.static필드명/메소드명


→ static 필드명/메소드명으로 접근가능.

```
import static java.lang.Integer.*; //Integer 클래스의 모든 static자원 접근
```

```
import static java.lang.Math.random; //Math.random 메소드만 import
```

```
import static java.lang.System.out; //System클래스의 필드 out
```

```
public class StaticImportTest {  
    public static void main(String[] args) {  
        //System.out.println(Math.random());  
        out.println(random());  
  
        //out.println(Integer.MIN_VALUE+"~"+Integer.MAX_VALUE);  
        out.println(MIN_VALUE+"~"+ MAX_VALUE);  
    }  
}
```



클래스 선언시 사용 가능한 예약어

- **final** : 변경될 수 없는 클래스, 상속 불가

예) public final class 클래스명 {

.....

}

- **abstract** : 클래스내 추상 메서드가 선언되어있는 추상클래스, 반드시 후손이 완성해야 함

예) public abstract class 클래스명 {

abstract 리턴값 메소드명(매개변수);

abstract 리턴값 메소드명(매개변수);

}

두 개 예약어를 동시에 사용할 수 없음