웹 개발자 부트캠프 과정

CodingOn

객체지향 프로그래밍 (OOP)

- 필요한 데이터와 코드를 묶어 하나의 객체로 만들고 이 **객체들 간에 상호작용**을 하도록 프로그램을 만드는 방식
- OOP, Object Oriented Programming

- 현실 세계
  - 제품 생산 시 부품을 만들고, 부품을 조립해서 하나의 완성품을 제작함
- 소프트웨어 개발
  - 부품에 해당하는 객체를 만들고, 객체를 조립해서 완성된 프로그램을 만듦

객체란? (Object)

- 물리적으로 존재하거나 개념적인 것 중에서 다른 것과 식별 가능한 것
- ex. 학생, 자동차, 학과 등

- 속성(필드, field)과 동작(메소드, method)으로 구성
- ex. 학생 객체
  - 속성: 이름, 나이
  - 동작: 공부하다, 시험보다

객체지향 프로그래밍의 상호작용

- 객체는 다른 객체와 상호작용함
- 객체의 동작인 메소드(method)를 이용해 서로 상호작용을 한다!!

- 상호작용은 데이터를 주고 받음을 의미. 어떻게?
- `메소드명(매개변수1, 매개변수2, …)`
  - 다음과 같이 메소드를 호출함으로써 다른 객체에게 정보를 넘겨줌

객체지향 프로그래밍 특징

- 캡슐화 (Encapsulation)
  - 객체의 필드와 메소드를 묶고, 내용을 외부에 감추는 것
- 상속 (Inheritance)
  - 부모 객체에 필드/메소드를 자식 객체에게 물려주는 것
- 다형성 (Polymorphism)
  - 사용 방법은 동일하지만, 결과가 다양하게 나오는 것

객체지향 프로그래밍

- 장점
  - 코드 재사용에 용이
  - 유지보수 용이
  - 대형 프로젝트에 적합

- 단점
  - 처리속도가 느림
  - 설계가 복잡
  - 많은 시간 투자 필요

절차지향 프로그래밍

- 장점
  - 실행 속도 빠름 (코드가 기술 순서대로 실행되어 컴퓨터 처리 구조와 비슷하기 때문)

- 단점
  - 프로그램이 커진다면 유지보수 어려움
  - 순서가 바뀌면 결과 도출의 어려움
  - 대형 프로젝트에 적합하지 않음

클래스

클래스란?

- 사용자가 정의하는 자료형이라고 생각하면 쉽다!

- 실수형 변수를 선언한다면, float 자료형을 사용하고,
- 정수형 변수를 선언한다면, int 자료형을 사용했다

- 반면, 자동차 정보를 저장하는 변수를 선언한다면..?
  - 제조사, 모델명, 가격, 연비 등의 정보를 저장해야 하는데, Primitive 자료형으로는 불가능
  - 사용자가 클래스(아마도 Car 클래스가 되겠죠?)를 만들어서 새로운 자료형을 정의해야 한다!

객체와 클래스

- 현실 세계에서 자동차 객체를 만든다고 가정
  - 설계도를 바탕으로 실제 자동차 하나하나를 생산한다!

- 소프트웨어 개발에서 자동차 객체를 만든다고 가정
  - 클래스(Class)를 바탕으로 자동차 객체 하나하나를 생성한다!

  - 인스턴스화 : 클래스로부터 객체를 만듦
  - 인스턴스(Instance) : 클래스로부터 만들어진 객체

  - 즉, 클래스로 부터 여러 개의 인스턴스를 만들 수 있음

클래스 선언과 객체 생성

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

// 객체 생성
// new 연산자: 객체의 주소를 리턴
클래스명 변수 = new 클래스명();
```

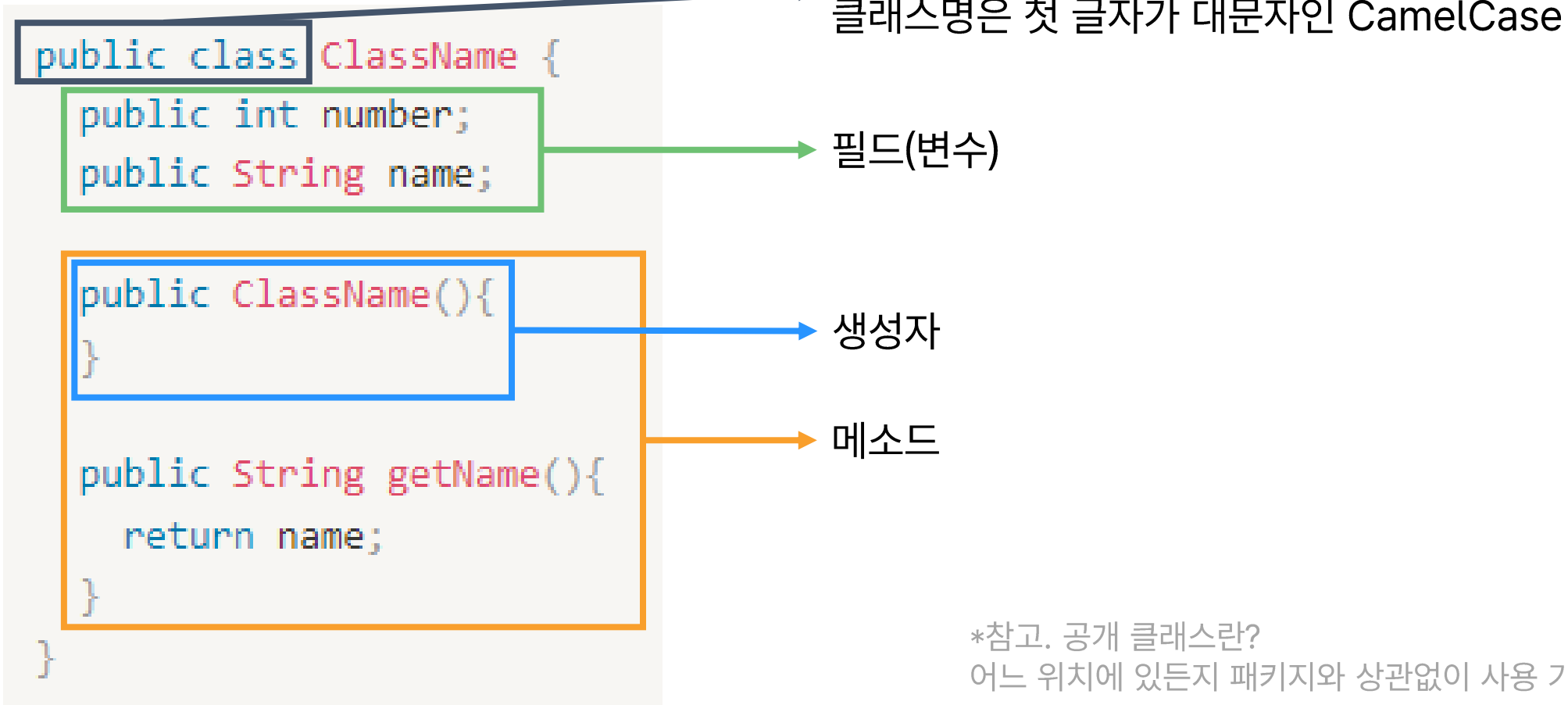
클래스 선언과 객체 생성

- new 키워드 이용

```
public class Main {

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

- Main 클래스에서 test 라는 객체를 생성함
- test는 ClassName 클래스의 인스턴스!

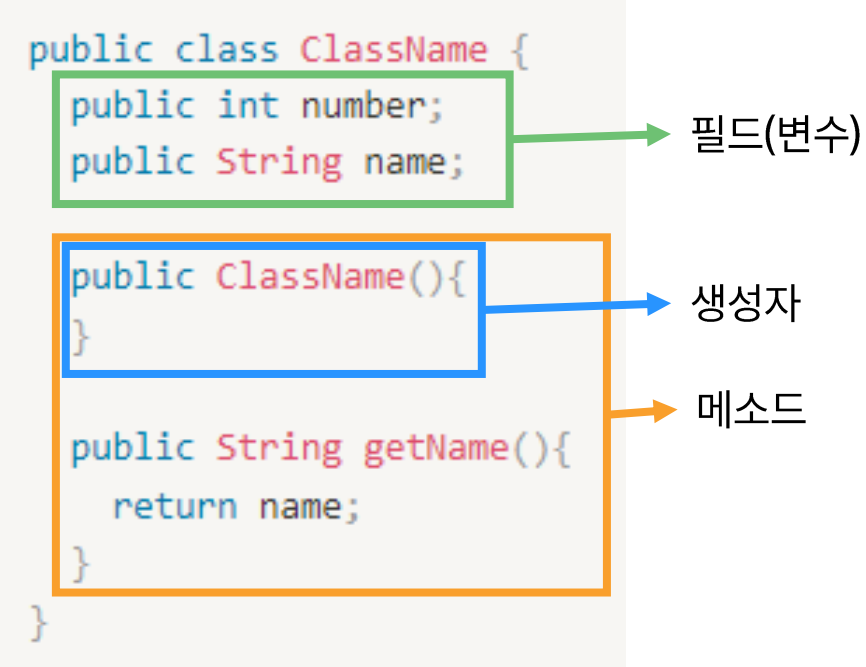
클래스의 구조



*참고. 공개 클래스란?
어느 위치에 있든지 패키지와 상관없이 사용 가능한 클래스

(뒤에서 할 거예요!! 지금은 그렇구나~ 정도로 이해해도 충분!)

클래스의 구조



- 필드(변수)
  - 클래스 내에서 값을 저장하는 변수
- 메소드
  - 객체 동작으로 호출 시 실행하는 블록
  - 객체 내부의 함수
- 생성자
  - 클래스 이름과 동일한 이름을 갖는 특별한 종류의 메소드
  - 객체가 생성될 때 자동으로 호출되는 메소드
  - 객체의 초기화 담당

필드 선언

- 클래스 블록에서 선언되어야 필드 선언이 됨
- 필드의 초기값을 할당하지 않으면 자동으로 기본 값으로 초기화
  - https://paulyun.tistory.com/30

```
// 필드 선언
자료형 필드명 [= 초기값];
```

필드 사용

- 필드는 객체 내부 데이터이므로 객체가 존재해야 필드를 사용 가능
- 클래스로부터 객체를 생성한 후 필드에 접근할 수 있음

- 필드가 속한 클래스 내부에서는 필드명을 그대로 사용
- 외부 객체에서는 참조 변수와 dot(.) 연산자를 이용해 필드 사용

```
// 객체 내부
int name; // 필드 선언
name = "아이오닉6"; // 필드 사용
```

```
// 외부 객체
Car c1 = new Car(); // 객체 생성 후
c1.name = "아이오닉6"; // dot 연산자를 이용해 필드 접근
```

메소드 기본 구조

- 메소드 형태


```
public void test(){
  System.out.print("test 메소드");
}
```

- 생성자
  - 생성자의 이름은 클래스 이름과 같아야 함
  - 생성자는 리턴 타입을 적지 않음
  - 접근제한자가 public 인 경우만 호출됨

```
public class ClassName {
  public ClassName(){} // 생성자
}
```

실습. 클래스 실습

- Rectangle 클래스 만들기 (사각형의 넓이를 구하는 클래스)
- 필드(변수): width, height
- 생성자: width와 height 설정할 2개의 숫자를 매개변수로 받기.
- 메소드: width와 height를 이용하여 사각형의 넓이를 반환하는 area 메소드 만들기

- 객체 생성 시에 width와 height을 사용자에게 입력 받아 생성자로 넘겨주기

```
사각형의 가로와 세로 길이를 띄어쓰기를 기준으로 입력해주세요.
5 8
40
```

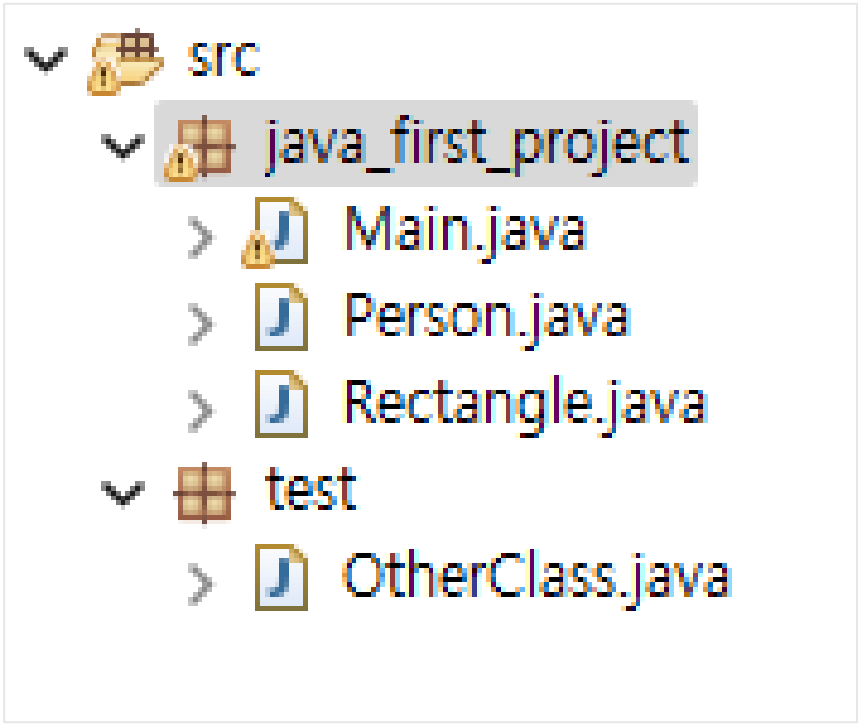
접근 제어자

패키지 (package)

- 관련 있는 클래스 파일을 저장하는 공간
- 디렉터리의 역할 뿐만 아니라 클래스의 일부분으로 클래스를 식별하는 공간

animal 패키지

person 패키지



패키지 (package)

- 패키지 선언
  - 항상 소스파일의 최상단에 위치해야 함
  - 소문자로만 작성하는 것이 관례

```
package 상위패키지.하위패키지;

public class 클래스명 { ... }
```

- 같은 패키지에 있는 클래스
  - 아무 조건 없이 사용 가능
- 다른 패키지에 있는 클래스
  - import 문으로 어떤 패키지의 클래스인지 명시
  - 특정 패키지에서 다수의 클래스를 사용한다면?
    - import 패키지.*;

```
pacakge _05_class; // Test 클래스의 패키지

import _04_other_package.Hello ; // 다른 패키지의 Hello 클래스

public class Test {
  Hello h1 = new Hello();
}
```

접근 제어자 (Access Modifier)

- 접근 제어자의 목적
  - 클래스나 일부 멤버(필드, 메소드)를 외부에서의 접근을 제한하는 것
  - 중요한 필드와 메서드가 외부로 노출되지 않도록 하기 위함

- 접근 제어자의 종류
  - public
  - protected
  - default
  - private

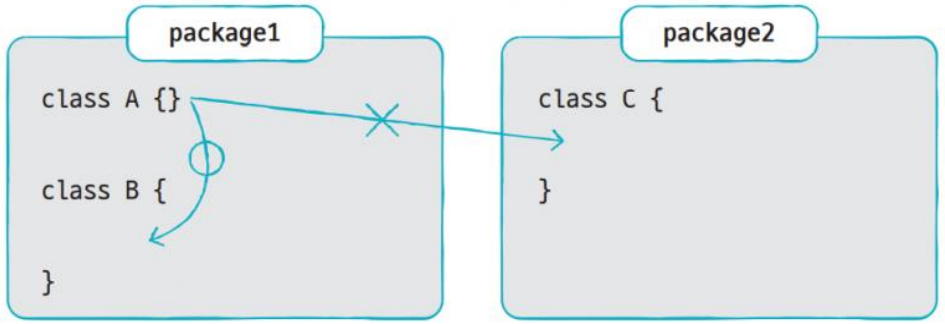
접근 제어자 (Access Modifier)

- public
  - 패키지에 관계 없이 **모든 클래스**에서 접근 가능
- protected
  - **같은 패키지 내의 모든 클래스**에서 접근 가능 (다른 패키지에서 접근 불가)
  - 단, 다른 패키지에 있더라도 **상속 받은 서브클래스에서는 접근 가능**
- default **default* 의 경우, 접근 제한자가 붙지 않은 상태
  - **같은 패키지 내의 모든 클래스**에서 접근 가능 (다른 패키지에서 접근 불가)
- private
  - **같은 클래스에서만** 접근 가능

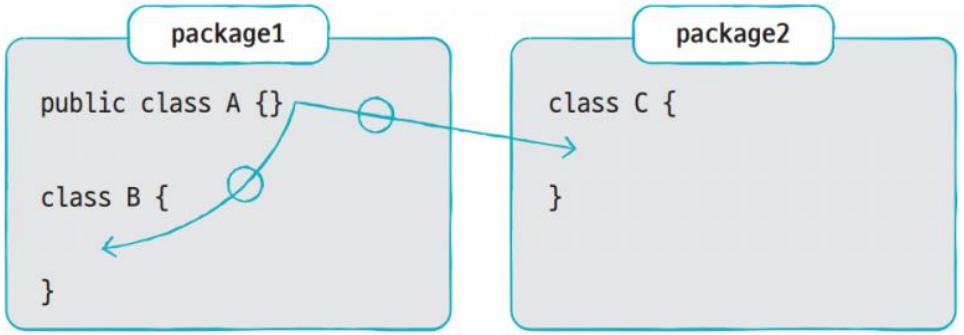
접근 제어자 (Access Modifier)

제어자	같은 클래스	같은 패키지	자손클래스	전체
public	O	O	O	O
protected	O	O	O	
default	O	O		
private	O			

클래스의 접근 제한



- public 생략하면, default 접근 제한
  - 같은 패키지에서는 제한 없음
  - 다른 패키지에서는 사용 불가



- public 접근 제한
  - 같은 패키지에서는 제한 없음
  - 다른 패키지에서도 제한 없음

이미지 출처: https://hongong.hanbit.co.kr/%EC%9E%90%EB%B0%94-%EC%A0%91%EA%B7%BC-%EC%A0%9C%ED%95%9C%EC%9E%90%EC%9D%98-%EC%A2%85%EB%A5%98%EC%99%80-%EC%B0%A8%EC%9D%B4public-protected-private/

필드와 메소드의 접근 제한

- 필드와 메소드는 어떤 접근 제한자를 갖느냐에 따라 호출 가능 여부가 결정됨

getter와 setter

- 클래스에서 필드(변수)는 private로 지정하 는 것이 일반적이다. 외부에서 마음대로 데이터를 변경한다면 무결성이 깨짐
  - ex. 학생의 시험 점수는 음수가 될 수 없는데, 외부에서 음수로 변경한다면?
- private 로 선언하면 다른 클래스에서 접근하지 못하는 것이 아닌가?
  - 따라서 간접적으로 필드에 접근할 수 있도록 public 메소드를 제공한다.
  - 이것을 getter와 setter 라고 한다.
  - 따라서, 객체지향 프로그래밍에서는 데이터의 직접적인 변경을 막고, 메소드를 통해 필드에 접근하는 것을 선호

- getter: 필드 값을 가져올 때
- setter: 필드 값을 설정할 때

getter와 setter

- 형태

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

    public Person(String name){
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

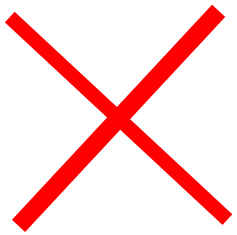
getter

setter

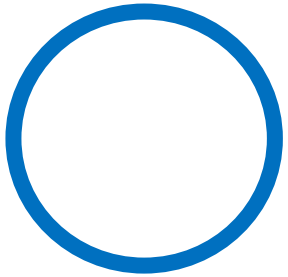
getter와 setter

- 사용

```
Person codee = new Person("코디");
codee.age = 1;
System.out.println(codee.name);
```



```
Person codee = new Person("코디");
codee.setAge(1);
System.out.println(codee.getName());
```



실습. 클래스 실습 upgrade

- 이전 실습에서 만든 Rectangle 클래스를 활용
- 필드(변수) 접근제어자를 private으로 변경
- width와 height 필드에 대한 getter, setter 생성
- 생성자 매개변수에서 height 매개변수 지우고, height는 setter로 추후 설정

- Rectangle 클래스를 담는 ArrayList 생성하고, 사용자로부터 가로와 세로 입력받아 ArrayList에 Rectangle 인스턴스를 차례로 추가.
- 사용자가 가로 세로 모두 0 값을 입력하면, 배열 안에 있는 값을 사진과 같이 출력

실습. 클래스 실습 upgrade

getter 이용하여 출력하기

```
사각형의 가로와 세로 길이를 띄어쓰기를 기준으로 입력해주세요.
3 5
사각형의 가로와 세로 길이를 띄어쓰기를 기준으로 입력해주세요.
4 6
사각형의 가로와 세로 길이를 띄어쓰기를 기준으로 입력해주세요.
8 10
사각형의 가로와 세로 길이를 띄어쓰기를 기준으로 입력해주세요.
0 0
가로 길이는: 3
세로 길이는: 5
넓이는: 15
--------------------------
가로 길이는: 4
세로 길이는: 6
넓이는: 24
--------------------------
가로 길이는: 8
세로 길이는: 10
넓이는: 80
--------------------------
```

- 힌트!

```
ArrayList<Rectangle> rect = new ArrayList<>();
```

static

static 멤버 (변수, 메소드)

- 객체마다 생성되는 것이 아니고, 클래스당 하나만 생성됨
- 클래스가 생성되는 순간에 메모리를 할당 받음
- 즉, 객체를 생성하지 않아도 static 멤버에 접근 가능
- 동일한 클래스의 모든 인스턴스에 공유됨 (같은 메모리 공간 공유)
- non-static의 경우 객체가 생성될 때마다 멤버 공간을 새로 만듦
  → 메모리 공간이 공유되지 않음

```
public class SomeClass {
  // 정적 필드
  static 타입 필드명 [= 초기값];

  // 정적 메소드
  static 리턴타입 메소드명 (매개변수1, 매개변수2, ...) { ... }
}
```

static의 활용

- 보통 static 변수는 공유의 목적으로 많이 사용됨

```
public static int COUNT = 0;
```

- static 메소드는 유틸리티성 메소드를 작성할 때 많이 사용됨
  ex) 오늘의 날짜 구하기, 숫자에 콤마 추가하기 등등..
  ex) java.lang 패키지의 Math 클래스

```
public final class Math {
  public static int abs(int a);
  public static double cos(double a);
  public static int max(int a, int b);
  public static double random();
  ...
}

// 사용시엔
int n = Math.abs(-5);
```

static의 제약

- static 메소드에서 non-static 멤버에 접근할 수 없음.
  - 객체가 생성되지 않은 상태에서 non-static멤버는 만들어져 있지 않기 때문에
- 반대로, non-static 메소드에서 static 멤버에 접근 할 수 있음.
- static 메소드에서는 this 키워드 사용 불가

실습. 클래스 실습 upupgrade

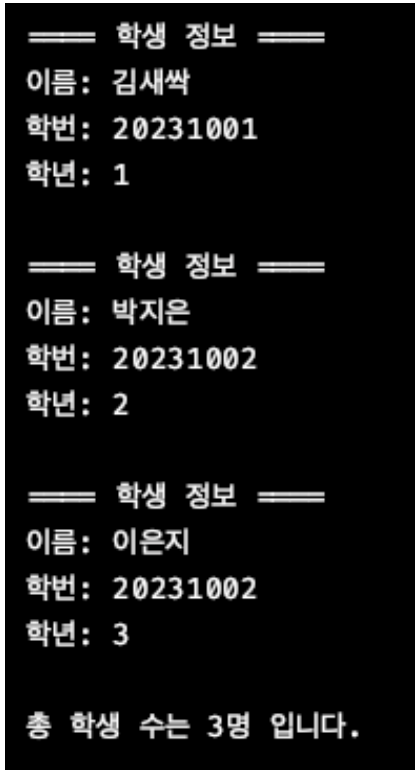
- 이전 실습에서 사용자가 Rectangle의 인스턴스를 몇 개 만들었는지를 알려주는 필드 추가하기!
- 사진과 같은 형태의 결과 나오게 작성

```
사각형의 가로와 세로 길이를 띄어쓰기를 기준으로 입력해주세요.
10 5
사각형의 가로와 세로 길이를 띄어쓰기를 기준으로 입력해주세요.
3 4
사각형의 가로와 세로 길이를 띄어쓰기를 기준으로 입력해주세요.
4 6
사각형의 가로와 세로 길이를 띄어쓰기를 기준으로 입력해주세요.
8 9
사각형의 가로와 세로 길이를 띄어쓰기를 기준으로 입력해주세요.
0 0
```

```
가로 길이는: 10
세로 길이는: 5
넓이는: 50
--------------------------
가로 길이는: 3
세로 길이는: 4
넓이는: 12
--------------------------
가로 길이는: 4
세로 길이는: 6
넓이는: 24
--------------------------
가로 길이는: 8
세로 길이는: 9
넓이는: 72
--------------------------
Rectangle 인스턴스의 개수는: 4
```

실습. static 실습

- Student 클래스는 이름(name), 학번(student_ID), 학년(grade)을 저장함
- 각 필드에 대한 getter, setter 추가
- 클래스 변수 (static 변수) totalStudents 생성해 전체 학생 수 저장
- 생성자를 통해 이름, 학번, 학년을 초기화하고 totalStudnets 값 증가
- displayInfo 메소드를 작성해 학생 정보 출력



final

final 필드 선언

- 인스턴스 필드, 정적 필드는 언제든지 값 변경이 가능했다
- 그러나 값이 변경되면 안되는 경우가 있어야 한다면?
  즉, 값 변경을 막고 읽기만 허용하고 싶을 때에는 final 키워드 사용하자!

- final 필드
  - 한 번만 초기화할 수 있으며 초기화 된 값이 최종 값 (즉, 변경 불가능)

```
final 타입 필드 [=초기값];
```

- final 필드 초기값 대입 방법
  1. 필드 선언 시 초기화 : 간단
  2. 생성자에서 초기화 : 객체 생성시 외부에서 전달하는 값으로 초기화 해야 할 때 사용

상수 선언

- 상수(constant) : 변하지 않는 값
- 상수는 객체마다 저장할 필요가 없고, 단 한 번만 값이 선언되면 되기 때문에 static 이면서 final 인 특성을 가지므로 다음과 같이 선언

```
static final 타입 상수 [=초기값];
```

- 대문자로 작성하는 것이 관례
- 여러 단어를 사용해야 한다면, 언더스코어(_)를 이용해 연결

```
static final double PI = 3.141592;
static final String HELLO_WORLD = "Hello, World!";
```

실습. final 실습

- Circle 클래스는 반지름(radius) 필드를 가짐
- 반지름은 생성자를 통해 초기화 되어야 하며, 이후에 변경될 수 없도록 final 필드로 선언되어야 함
- 원의 넓이를 계산하는 calculagteArea 메소드 작성
- 원주율(PI) 값은 상수로 선언
- 사진과 같이 출력되도록 작성

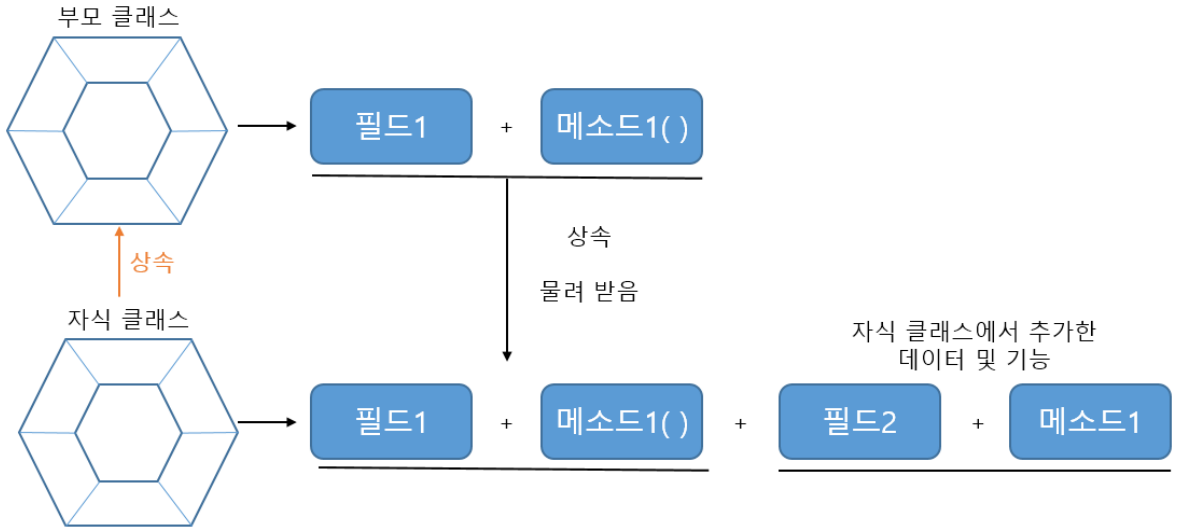
```
원의 반지름을 입력하세요: 10
원의 반지름: 10.0
원의 넓이: 314.1592653589793
```

클래스의 상속

클래스의 상속 (Inheritance)

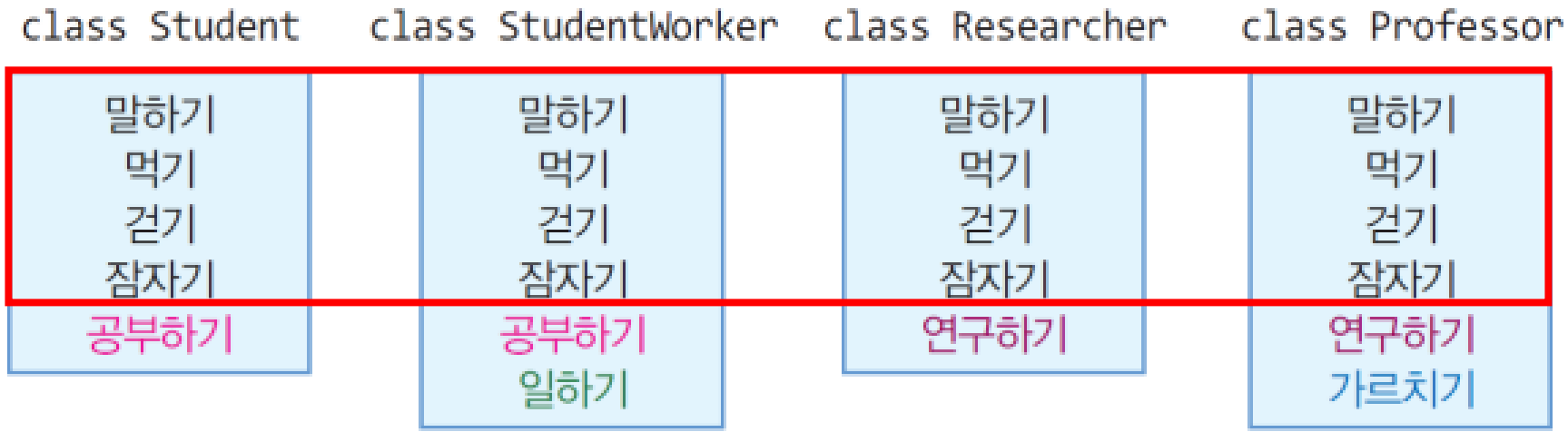
- 부모 클래스에서 정의된 필드와 메소드를 자식 클래스가 물려받는 것
- 프로그래밍에서는 자식 클래스가 어떤 부모로부터 상속받을 것인지 선택
- Java는 다중 상속을 허용하지 않음. 상속받을 수 있는 부모는 단 하나!

- 장점
  - 멤버의 중복 작성 제거
  - 클래스 수정 최소화
  - 클래스의 계층적 분류 가능
  - 클래스의 재사용과 확장성 용이



클래스 상속 예시

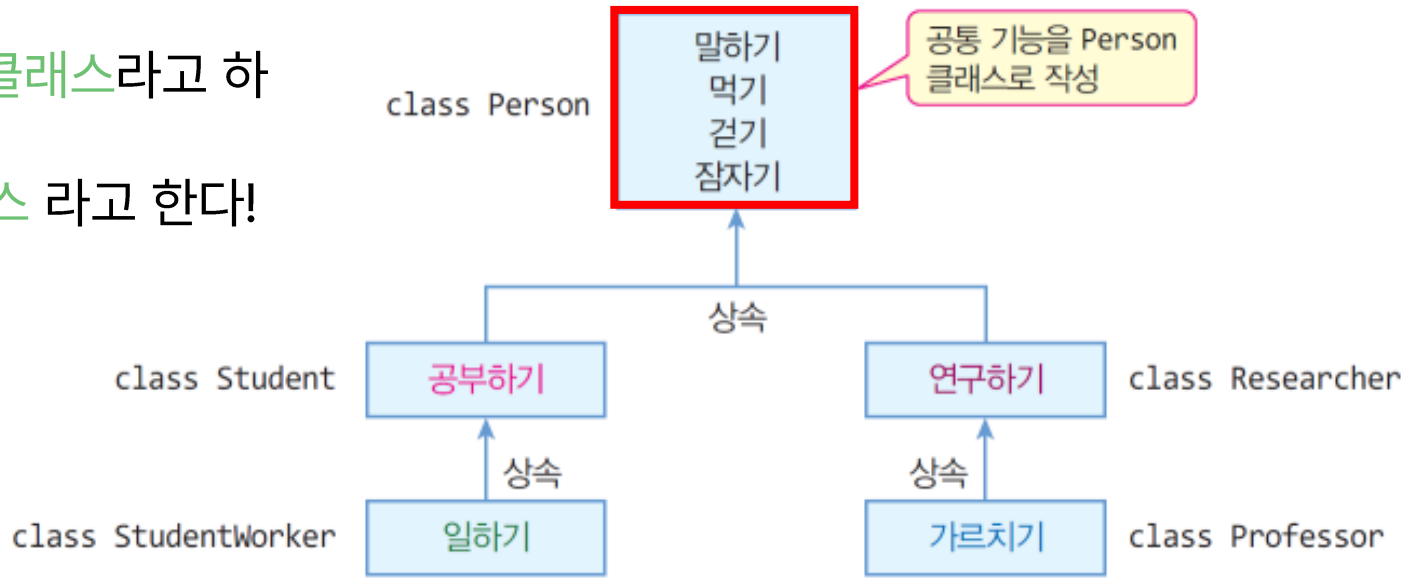
- 아래와 같이 신분 별로 클래스를 정의했다고 했을 때,
  "말하기, 먹기, 걷기, 잠자기" 의 행동은 모든 신분에서 동일하게 정의됨.



클래스 상속 예시

- 이를 계층화 하여, 아래와 같이 표현할 수 있음.
- Person이라는 부모 클래스를 만들고
  Student, Researcher, … 클래스에서 상속받도록 하는 것!

이때 Person을 부모클래스 혹은 슈퍼클래스라고 하고,
Student를 자식클래스 혹은 서브클래스 라고 한다!



상속 문법

- **extends** 키워드 사용
- 다중 상속을 허용하지 않으므로 extends 뒤에는 부모 클래스 하나만!

```
public class Person {
  ...
}

public class Researcher extends Person { // Person을 상속받는 Researcher 선언
  ...
}

public class Professor extends Researcher { // Researcher를 상속받는 Professor 선언
  ...
}

public class Student extends Person { // Person을 상속받는 클래스 Student 선언
  ...
}
```

상속 간단 예제

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

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

```java
public class Student extends Person{
  private String school;

  public Student(int age, String name) {
    setAge(age);
    setName(name);
  }

  public String getSchool() {
    return school;
  }

  public void setSchool(String school) {
    this.school = school;
  }
}
```

부모 클래스의 멤버에 접근 가능!

```java
Student student = new Student(20, "코딩온");
student.setSchool("코딩온대학교");
```

부모 생성자 호출

- 자식 객체를 생성하면 부모 객체가 먼저 생성된다!
  - 부모 객체가 먼저 생성된 후, 자식 객체인 Student 가 생성 된 것!
  - 부모인 Person 객체를 생성하지 않았는데, 어떻게 가능하지?

```
Student student = new Student(20, "코딩온");
```

부모 생성자 호출, super()

- 모든 객체는 생성자를 호출해야만 생성된다. 즉, 비밀스럽게 부모 생성자가 실행된 것!
  - 자식 생성자의 맨 첫 줄에 super()가 숨겨져 있다!
  - super() 가 컴파일 과정에서 자동으로 추가되면서 부모의 기본 생성자를 호출함!
  - 만약, 부모 클래스에 기본 생성자가 아닌 매개변수를 갖는 생성자만 있다면, super()에 매개변수를 넣어주어야 함!

```
// 자식 클래스의 생성자
public Student(...) {
  super();
  // ...
}
```

```
// 자식 클래스의 생성자
public Student(...) {
  super(매개변수1, ...);
  // ...
}
```

상속 간단 예제

- 부모클래스(슈퍼클래스)의 생성자가 매개변수를 필요로 할 경우.
- 즉, 부모 클래스의 생성자를 호출해해야 한다면?

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

  public Person(String name) {
    setName(name);
  }

  ...

  public void setName(String name) {
    this.name = name;
  }
}
```

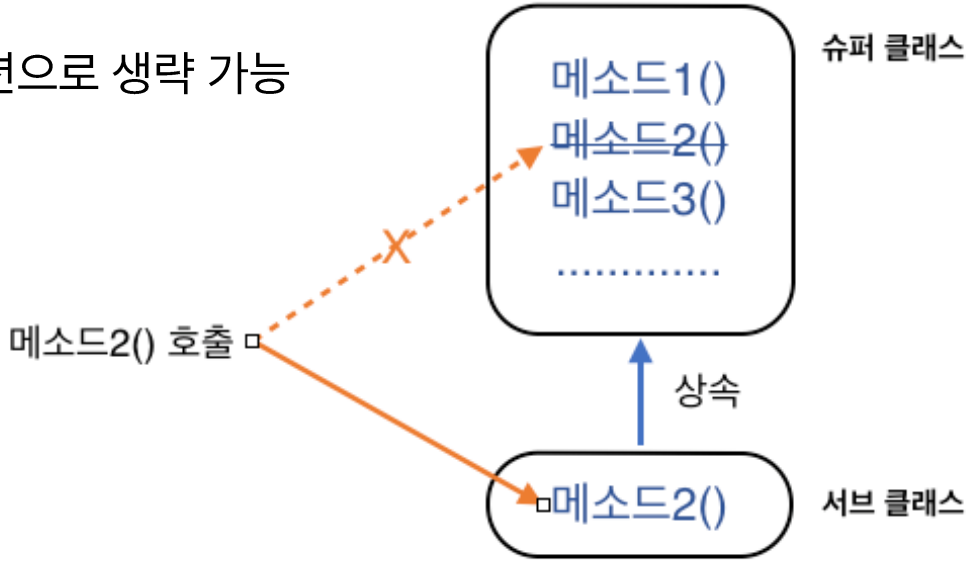
```java
public class Student extends Person{
  private String school;

  public Student(int age, String name) {
    super(name);
    setAge(age);
  }

  ...
}
```

서브 클래스 첫 줄에서 super() 사용!!

메소드 오버라이딩 (Overriding)

- 오버라이딩이란?
  - 슈퍼 클래스의 메소드를 서브 클래스에서 재정의 하는 것!
  - 왜? 상황에 따라 부모 클래스의 메소드가 자식 클래스가 사용하기에 적합하지 않을 수도 있기 때문
  - 이러한 경우, 자식 클래스에서 재정의해서 사용함
  - 메소드의 이름, 매개변수 타입 및 개수, 리턴 타입 등 모든 것을 동일하게 작성.
- @Override
  - 컴파일시 정확히 오버라이딩 되었는지 확인하는 어노테이션으로 생략 가능

```java
// Person 클래스
public void printName() {
  System.out.println("익명");
}

// Student 클래스
@Override
public void printName() {
  System.out.println(getName());
}
```



실습. 상속 실습

- Animal 이라는 클래스 제작
  - 조건1) "종“, "이름", "나이"을 의미하는 변수를 가지고 있어야 한다.
  - 조건2) makeSound라는 메소드를 가지고 있어야 한다.
  - 조건3) makeSound라는 메소드는 "동물은 소리를 낸다" 라는 문자열 출력.
- Animal 클래스를 상속받는 Cat과 Dog 클래스 제작
  - 조건1) Animal 클래스의 "종“, "이름", "나이" 을 의미하는 변수를 생성자에서 초기화할 것. (종은 "고양이" 혹은 "강아지"로, 나머지는 객체를 생성할 때 원하는 값을 받아서 초기화.)
  - 조건2) makeSound 메소드를 오버라이드하여 각 동물이 내는 소리를 출력할 것.
  - 강아지와 고양이 고유의 속성과 동작을 생각해보고 필드, 메소드 만들어보기.

실습. 상속 실습2

차량(Vehicle)에 대하여 클래스화 해보기.

오른쪽 사진을 참고하여, 스스로 각각의 차량에 대한 공통점을 찾고, 부모 자식 관계를 만들어서 java 코드로 작성!

```
=== Bus 정보 ===
Bus {brand='Hyundai', model='City Bus', year=2022, passengerCapacity=30}
차량 시동을 걸었습니다.
승객을 운송합니다.
차량을 정지했습니다.

=== Car 정보 ===
Car {brand='Toyota', model='Camry', year=2023, convertible=true}
차량 시동을 걸었습니다.
주차했습니다.
차량을 정지했습니다.

=== Motorcycle 정보 ===
Motorcycle {brand='Harley-Davidson', model='Sportster', year=2021, licenseType='A'}
차량 시동을 걸었습니다.
울림통을 합니다.
차량을 정지했습니다.
```

예시일 뿐! 다르게 하셔도 됩니다!

추상 클래스와 추상 메소드

추상 (abstract)

- 실체 간의 공통되는 특성

- ex. 직각 삼각형, 원, 정사각형, 정오각형, 평행사변형, …
  - 이들의 공통점은 도형!
  - 도형은 실체들의 공통되는 특성을 갖는 추상적인 것임

추상 클래스

- 하나 이상의 추상 메소드(선언만 있고, 구현되지 않은 메소드)를 포함하는 클래스
- 상속 관계에서 부모 클래스 역할을 하며, 추상 메소드와 일반 메소드를 가질 수 있음
- 서브클래스에서 반드시 구현되어야 함
- 객체의 공통적인 특징을 추상화해서 정의하고, 이를 상속받는 서브클래스에서 구현
  - 실체 클래스는 추상 클래스를 상속해서 공통적인 필드나 메소드를 물려받음

- 추상 클래스는 new 연산자를 통해 인스턴스 객체를 만들 수 없음
- 추상 클래스를 상속받는 서브 클래스에서 추상 메소드를 오버라이딩 해줘야 함!

추상 클래스의 용도

- 설계와 구현 분리
- 슈퍼 클래스에서는 개념 정의
  - 서브클래스마다 다른 구현이 필요한 메소드는 추상 메소드로 선언
- 서브 클래스에서 구체적 행위 구현
  - 서브클래스마다 목적에 맞게 추상 메소드를 다르게 구현

- 추상 클래스는 new 연산자 사용해 객체 직접 생성 불가능!
- 새로운 클래스를 만들기 위한 부모 클래스로만 사용되므로 extends 뒤에만 올 수 있으

```
Shape shape = new Shape(); // 불가능!
```

추상 클래스의 구현

- **abstract** 키워드 사용

```
public abstract class Person {
  private int age;
  private String name;
  ...

  abstract public void printName();
}
```

```
// Student Class
public void printName() {
  System.out.println("학생" + getName());
}
```

실습. 추상클래스 실습

- Student 라는 추상 클래스 클래스 제작
  - 조건1) 이름, 학교, 나이, 학번을 의미하는 변수를 가져야한다.
  - 조건2) 모든 변수의 값은 하위클래스에서 할당된다.
  - 조건3) todo 라는 추상 메소드 만들기
- Student 클래스를 상속받는 Kim 클래스와 Baek 클래스 만들기
  - Kim 클래스와 Baek 클래스 내의 변수 값 다양하게 넣기
  - todo 오버라이딩.
    -> Kim 클래스에서는 “점심은 김가네 김밥” 출력
    -> Baek 클래스에서는 “점심은 백종원 피자” 출력

```
═══ 김철수 학생의 정보 ═══
학교: ABC 고등학교
나이: 17
학번: 20220001
점심은 김가네 김밥

═══ 백영희 학생의 정보 ═══
학교: XYZ 고등학교
나이: 18
학번: 20220002
점심은 백종원 피자
```

실습. 추상 클래스 실습

- Shape 추상 클래스
  - color 필드: 도형 색상
  - type 필드: 도형 종류 (원, 사각형 등)
  - calculateArea 추상 메소드: 도형 넓이 계산
  - getColor 메소드: 도형 색상 반환
- Circle 클래스
  - radius 필드: 원의 반지름
- Rectangle 클래스
  - width, height 필드: 사각형의 가로, 세로 길이
- ShapeEx 클래스
  - main 메소드: ArrayList 를 활용해 Circle 과 Rectangle 객체 동적 저장
  - 도형 정보 출력

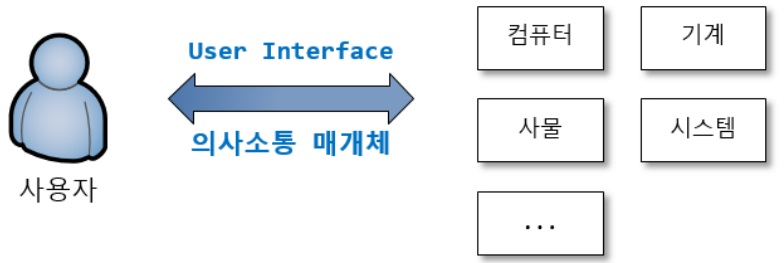
```
=== Circle 도형의 정보 ===
도형의 색상: Red
도형의 넓이: 78.53981633974483

=== Rectangle 도형의 정보 ===
도형의 색상: Blue
도형의 넓이: 24.0
```

인터페이스

인터페이스 (interface) 사전적 의미

- 사전적 의미로 두 장치를 연결하는 접속기
- 인터페이스는 두 객체를 연결하는 역할을 한다!

- ex. UI (User Interface)
  : 사람은 UI를 통해 컴퓨터와 소통을 할 수 있다



이미지 출처: https://codedragon.tistory.com/8996

Java에서 인터페이스 (interface)

- 추상 메소드의 모음
- 여러 구현체에서 공통적인 부분을 추상화 하는 역할
- 서브 클래스가 같은 동작을 한다는 것을 보장하기 위해 사용

- 추상클래스와 다른 점?
  - 추상 클래스는 추상 메소드가 없어도 상관없지만 인터페이스에는 추상 메소드가 무조건 있어야 함!
  - 메소드를 구현하지 못 함. 추상 메소드만 존재.
- 상속과 달리 하나의 클래스가 두개 이상의 인터페이스를 동시에 구현할 수 있음

인터페이스 문법

- 인터페이스에서 정의하는 모든 필드는 public static final 이 자동 적용
- 인터페이스를 구현하는 클래스는 인터페이스의 모든 추상 메소드를 반드시 구현해야 함.
- 인터페이스 선언 시에 class 키워드 대신 interface 키워드 사용
- 인터페이스 구현 시 extends 키워드 대신 implements 키워드 사용

```
interface 인터페이스명 { ... } // default 접근 제한
public interface 인터페이스명 { ... } // public 접근 제한
```

인터페이스 문법

```
public interface Phone {
  public final String CATEGORY = "phone"; // public final 생략 가능
  public abstract void sendCall(); // public abstract 생략 가능
  public abstract void receiveCall();
  public abstract void sendMessage();
  public abstract void receiveMessage();
}
```

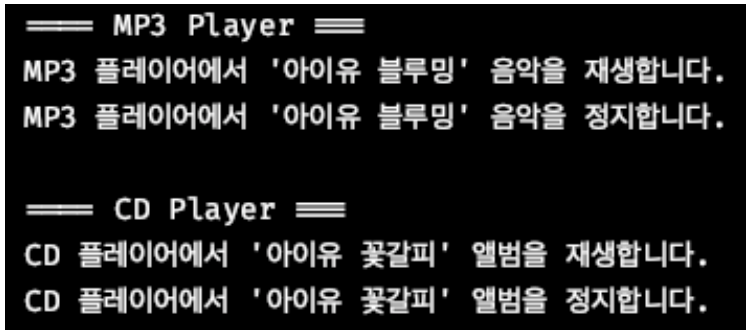
인터페이스 문법

인터페이스를 구현하는 키워드

```
public class Samsung implements Phone {
  public boolean silence = false;

  @Override // 부모 클래스에 있는 메서드를 Override 했다는 것을 명시적으로 선언 (주석과 비슷)
  public void sendCall() {
    if(!silence) System.out.println("뚜루루 뚜루루");
  }
  @Override
  public void receiveCall() {
    if(!silence) System.out.println("전화 받아");
  }
  @Override
  public void sendMessage() {
    if(!silence) System.out.println("슉");
  }
  @Override
  public void receiveMessage() {
    if(!silence) System.out.println("문자 왔숑");
  }
}
```

실습. 인터페이스 실습

- 각 음악 플레이어는 Music 인터페이스를 구현하기
- Music 인터페이스
  - play 메소드: 음악 재생 시작
  - stop 메소드: 음악 재생 중지
- Mp3Player 와 CdPlayer 클래스는 Music 인터페이스를 구현한다.
- MusicPlayer 클래스는 Mp3Player 와 CdPlayer 객체를 생성하고 각각의 음악을 재생하고 중지한다.



실습. 클래스 종합 실습

정의

- Vehicle 추상 클래스를 생성: name, maxSpeed라는 멤버변수와 getter, setter 메서드를 생성하고move() 추상 메서드 생성.
- Flyable 인터페이스를 정의. 이 인터페이스에 fly() 메서드 생성.
- Car와 Airplane 두 개의 서브 클래스를 생성.
- Car 클래스는 Vehicle을 상속받아 move() 메서드를 "도로를 따라 이동 중"이라고 출력하도록 오버라이드
- Airplane 클래스는 Vehicle을 상속받고, Flyable 인터페이스도 구현. move() 메서드는 "하늘을 날아가는 중"이라고 출력하고, fly() 메서드는 "고도 10,000피트에서 비행 중"이라고 출력.

출력

- Vehicle 배열을 만들어 Car와 Airplane 객체를 저장하고, 반복문을 사용하여 각 객체의 move() 메서드를 출력.
- 단, Flyable 인터페이스를 구현한 객체의 경우, fly() 메서드도 호출하세요. (힌트 instanceof 연산자: 변수가 참조하는 객체 타입 확인 가능)