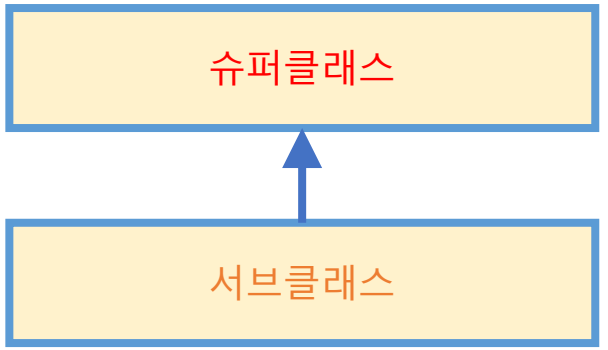
상속

상속(Inheritance)

- 상속(Inheritance)
  - ✓ 어떤 클래스의 필드와 메소드를 다른 클래스가 물려 받아 사용하는 것
  - ✓ 부모클래스가 자식클래스에게 필드와 메소드를 물려 줌
  - ✓ 자바에서는 부모클래스를 "슈퍼클래스(super)", 자식클래스를 "서브클래스(sub)"라고 함
- 상속의 장점
  - ✓ 동일한 메소드를 클래스마다 여러 번 정의할 필요가 없음
  - ✓ 클래스를 계층(부모-자식) 관계로 관리할 수 있음
  - ✓ 클래스의 재사용과 확장이 쉬움
  - ✓ 새로운 클래스의 작성 속도가 빠름



상속 관계를 나타내는 방법

상속 구조

Student is a Person.

상속 도입 전

class Student

먹는다
잔다
걷는다
공부한다

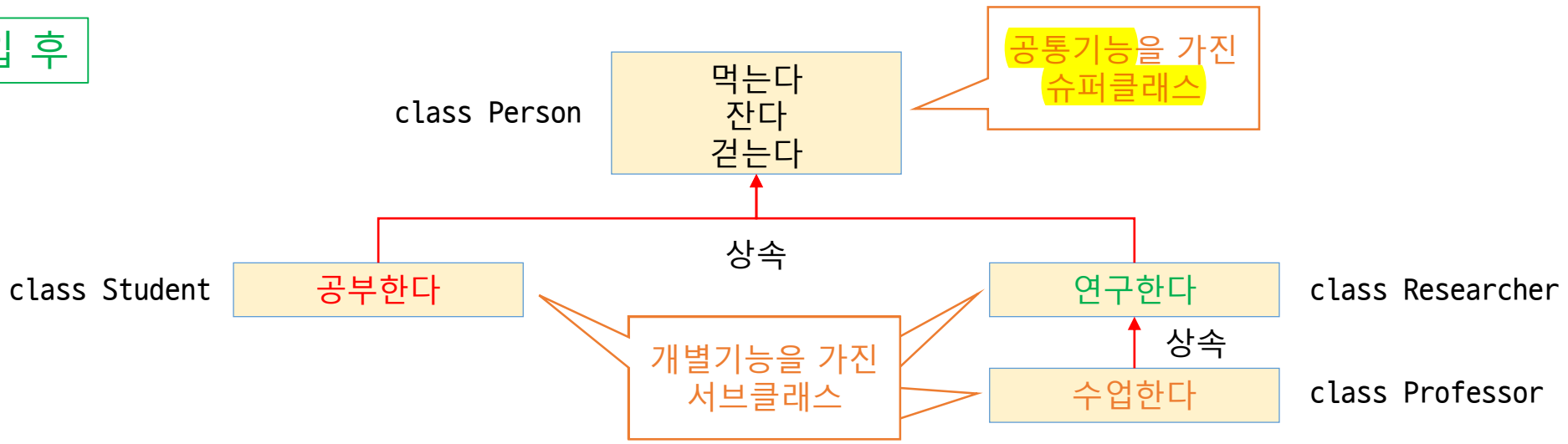
class Professor

먹는다
잔다
걷는다
연구한다
수업한다

class Researcher

먹는다
잔다
걷는다
연구한다

상속 도입 후



상속 관계의 클래스

```
public class Person {
    public void eat()   { }
    public void sleep() { }
    public void walk()  { }
}

public class Student extends Person {
    public void study() { }
}

public class Researcher extends Person {
    public void research() { }
}

public class Professor extends Researcher {
    public void teach() { }
}
```

서브클래스 / 슈퍼클래스

서브클래스 / 슈퍼클래스

서브클래스 / 슈퍼클래스

extends 키워드를 이용해서
상속 관계를 나타냄

Java

상속 관계에 있는 객체의 생성

- 서브클래스의 객체 생성
  - ✓ 서브클래스의 객체는 슈퍼클래스의 메소드를 사용할 수 있음
  - ✓ 서브클래스의 객체를 생성할 때 슈퍼클래스의 객체가 먼저 생성됨

```
public class Person {
    public Person() {
        System.out.println("Person 생성");
    }
}
public class Student extends Person {
    public Student(){
        System.out.println("Student 생성");
    }
}
public class Ex {
    public static void main(String[] args) {
        Student student = new Student();
    }
}
```

슈퍼클래스의 생성자가
먼저 호출되고,
서브클래스의 생성자가
나중에 호출된다.

실행결과

```
Person 생성
Student 생성
```

Java

부모클래스의 생성자 선택

- 서브클래스의 생성자는 부모클래스의 생성자를 super()를 이용해서 호출
- 부모클래스의 디폴트 생성자를 호출하는 경우 super() 생략 가능

```
public class Person {

    private String name;

    public Person() {

    }

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

}
```

호출

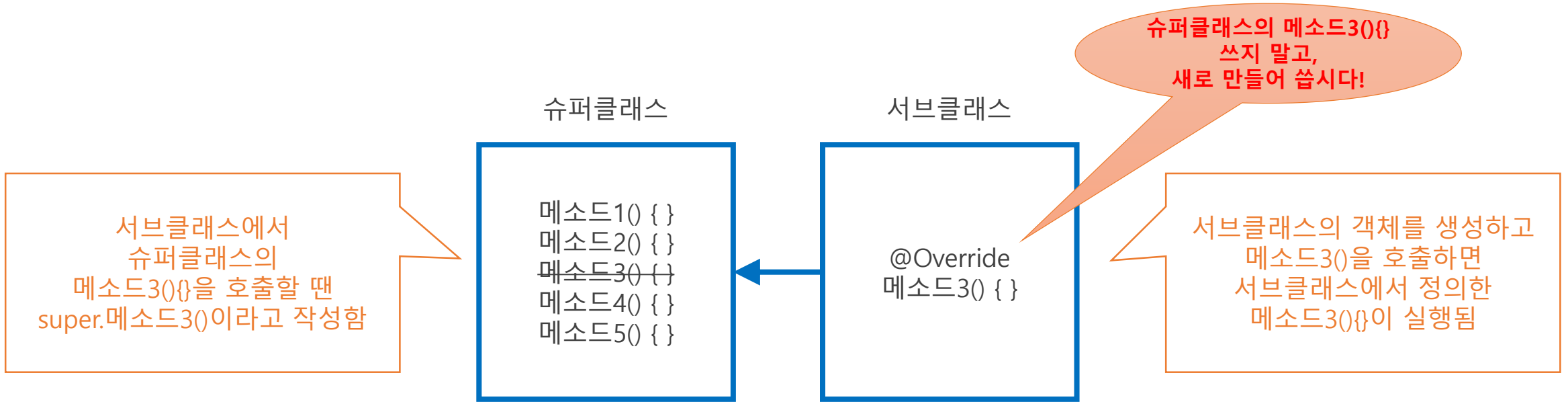
호출

```
public class Student extends Person {

    private String school;

    public Student() {
        super();
    }

    public Student(String name, String school) {
        super(name);
        this.school = school;
    }

}
```

생략 가능

메소드 오버라이딩(Method Overriding)

- 메소드 오버라이딩(Method Overriding)
  - ✓ 메소드 덮어쓰기
  - ✓ 슈퍼클래스의 메소드를 서브클래스에서 재정의하는 것
  - ✓ 슈퍼클래스의 메소드를 서브클래스가 사용하지 못하는 경우 메소드 오버라이딩이 필요함
  - ✓ 반드시 슈퍼클래스의 메소드와 동일한 원형(반환타입, 메소드명, 매개변수)으로 만들어야 함
  - ✓ 오버라이드 된 메소드 앞에는 @Override 애너테이션을 작성해서 오버라이드 된 메소드임을 알림



메소드 오버라이딩이 필요한 경우

규칙 : 모양은 같게.

```
public class 꼬깔콘 {
    public void 맛() {
        System.out.println("고소한맛");
    }
}
```

상속

```
public class 매콤달콤꼬깔콘 extends 꼬깔콘 {
    @Override
    public void 맛() {
        System.out.println("매콤달콤맛");
    }
}
```

부모클래스 꼬깔콘과 맛이 다르기 때문에 맛() 메소드를 새로 다시 만들어야 함

```
public class 군옥수수꼬깔콘 extends 꼬깔콘 {
    @Override
    public void 맛() {
        System.out.println("군옥수수맛");
    }
}
```

부모클래스 꼬깔콘과 맛이 다르기 때문에 맛() 메소드를 새로 다시 만들어야 함

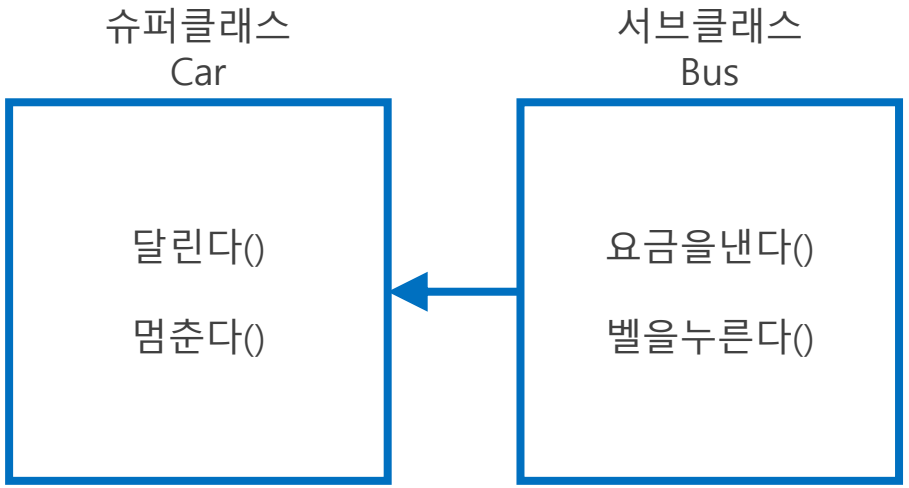
```
public class 찰옥수수꼬깔콘 extends 꼬깔콘 {
    @Override
    public void 맛() {
        System.out.println("찰옥수수맛");
    }
}
```

부모클래스 꼬깔콘과 맛이 다르기 때문에 맛() 메소드를 새로 다시 만들어야 함

Java

업캐스팅(Up-Casting)

- 업캐스팅(Up-Casting)
  - ✓ 서브클래스 객체를 슈퍼클래스 타입으로 변환하는 것
  - ✓ 자동으로 타입이 변환되는 promotion 방식으로 처리됨
  - ✓ 업캐스팅된 서브클래스 객체는 슈퍼클래스의 메소드만 호출 가능



서브클래스 객체 new Bus()를 슈퍼클래스 Car 타입으로 저장하는 업캐스팅이 자동으로 진행

Car car = new Bus();

Car 타입의 car 객체는 Car 클래스 내부의 메소드만 호출 가능

car.달린다();
car.멈춘다();

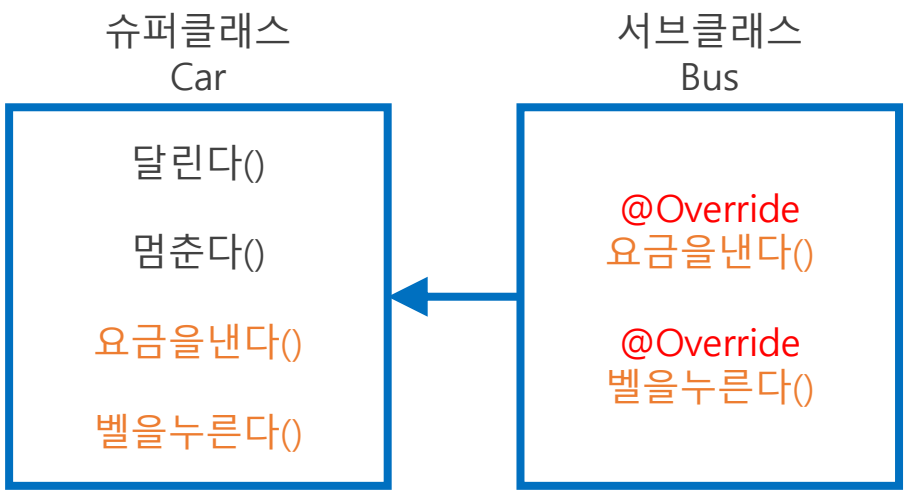
실제로는 new Bus()로 생성한 객체이지만 Bus 클래스의 메소드는 호출할 수 없음

~~**car.요금을낸다();**~~
~~**car.벨을누른다();**~~

문제발생

업캐스팅과 메소드 오버라이드

- 업캐스팅된 서브클래스 객체는 슈퍼클래스의 메소드만 호출할 수 있음
- 따라서 슈퍼클래스에 서브클래스의 메소드를 정의해 주고 서브클래스가 오버라이드 하는 방식으로 서브클래스의 메소드를 호출함



서브클래스 객체 new Bus()를 슈퍼클래스 Car 타입으로 저장하는 업캐스팅이 자동으로 진행

```
Car car = new Bus();
```

Car 타입의 car 객체는 Car 클래스 내부의 메소드만 호출 가능

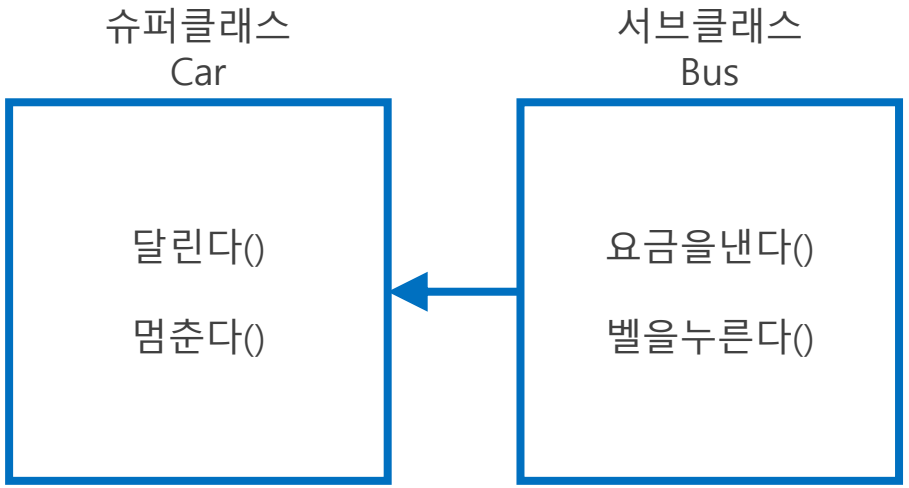
```
car.달린다();
car.멈춘다();
car.요금을낸다();
car.벨을누른다();
```

슈퍼클래스의 요금을낸다() 메소드를 호출하면 오버라이드에 의해서 서브클래스의 요금을낸다() 메소드가 실행됨

Java

다운캐스팅(Down-Casting)

- 다운캐스팅(Down-Casting)
  - ✓ 업캐스팅 된 서브클래스 객체를 다시 서브클래스 타입으로 변환하는 것
  - ✓ 강제로 타입을 변환하는 casting 방식으로 처리해야 함
  - ✓ 업캐스팅의 문제를 해결하기 위한 또 다른 방법임



서브클래스 객체 new Bus()를 슈퍼클래스 Car 타입으로 저장하는 업캐스팅이 자동으로 진행

```
Car car = new Bus();
```

Car 타입의 car 객체는 Car 클래스 내부의 메소드만 호출 가능

```
car.달린다();
car.멈춘다();
```

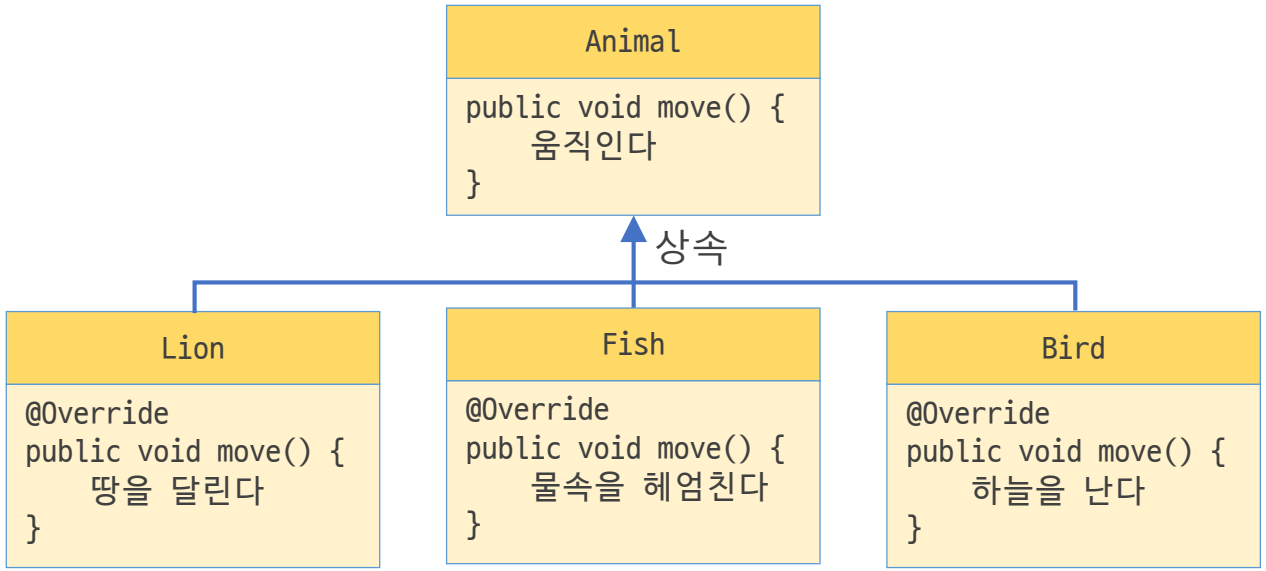
업캐스팅된 car 객체를 다시 Bus 클래스 타입으로 강제로 캐스팅하면 Bus 클래스의 메소드를 호출할 수 있음

```
((Bus)car).요금을낸다();
((Bus)car).벨을누른다();
```

Java

다형성(Polymorphism)

- 다형성(Polymorphism)
  - ✓ 객체지향의 중요한 특징
  - ✓ 하나의 메소드가 여러 가지 기능을 수행할 수 있음을 의미함
  - ✓ 하나의 메소드를 호출하는 코드가 전달되는 객체에 따라서 다르게 동작함



animal.move() 는 아래와 같이 동작한다.

print(new Lion()) 호출 - 땅을 달린다
print(new Fish()) 호출 - 물속을 헤엄친다
print(new Bird()) 호출 - 하늘을 난다

```
public void print(Animal animal) {
    animal.move();
}
```

추상클래스(Abstract Class)

- 추상메소드(Abstract Method)
  - ✓ 선언되어 있으나 본문이 없는 메소드
  - ✓ 본문이 없기 때문에 중괄호 {}가 없는 형태를 가짐
  - ✓ 메소드 선언 앞에 abstract 키워드를 붙임

- 추상클래스(Abstract Class)
  - ✓ 추상메소드가 하나 이상 포함된 클래스
  - ✓ 클래스 앞에 abstract 키워드를 붙임
  - ✓ 추상클래스를 상속 받는 서브클래스는 반드시 추상메소드를 모두 구현해야 함 (메소드 오버라이드 이용)
  - ✓ 추상클래스는 객체 생성이 불가능한 클래스
  - ✓ 구체적이지 않고 추상적인 단어들은 실제로 추상클래스로 구현될 수 있음

추상적인 단어 (추상클래스)	구체적인 단어 (추상클래스를 상속 받는 서브클래스)
동물	사자, 곰, 호랑이, …
도형	원, 사각형, 삼각형, …
음식	피자, 돈까스, 라면, …

Java

추상클래스(Abstract Class)

추상클래스는 일반메소드를 가질 수 있음

```
public abstract class GameUnit {

    private int energy;

    public void unitState() {
        System.out.println(energy + "남음");
    }

    public abstract void attack();
    public abstract void defense();

}
```

GameUnit들의 공격력(attack)과 방어력(defense)은 서로 다르기 때문에 공통 코드를 넣을 수 없음

이런 경우 본문이 없는 추상메소드로 정의해서 각 GameUnit별로 구현할 수 있도록 처리함

추상클래스 상속

```
public class Tank extends GameUnit {
    @Override
    public void attack() {
        System.out.println("공격력 10");
    }
    @Override
    public void defense() {
        System.out.println("방어력 100");
    }
}
```

```
public class Airplane extends GameUnit {
    @Override
    public void attack() {
        System.out.println("공격력 50");
    }
    @Override
    public void defense() {
        System.out.println("방어력 500");
    }
}
```

추상클래스를 상속 받으면 반드시 모든 추상메소드를 오버라이드 해야 함

Java

인터페이스(Interface)

- 인터페이스(Interface)
  - ✓ 클래스가 구현해야 할 메소드를 선언해 둔 자바 파일
  - ✓ 작업지시서 역할을 수행
  - ✓ 인터페이스 구현은 implements 키워드를 이용
  - ✓ 인터페이스를 구현하는 클래스는 반드시 인터페이스의 모든 추상메소드를 오버라이드 해야 함
  - ✓ 인터페이스에 작성하는 추상메소드는 abstract 키워드를 생략할 수 있음

- 형식 및 구성

```
public interface 인터페이스명 {
    상수
    추상메소드
    default 메소드
    private 메소드
    static 메소드
}
```

회원관리 인터페이스

```
로그인하기();
로그아웃하기();
회원가입하기();
회원탈퇴하기();
아이디찾기();
비밀번호찾기();
자동로그인();
...
```

여기 있는 메소드
모양 그대로
내용만 구현하세요!

인터페이스(Interface)

```java
public interface Shape {

    public default void info() {
        System.out.println("도형");
    }

    public double getArea();
    public void shapeState();

}
```

추상메소드의
abstract 생략 가능

인터페이스는 일반메소드
대신 default 메소드를
가질 수 있음

인터페이스 구현

```java
public class Circle implementss Shape {
    private double radius;
    @Override
    public double getArea() {
        return Math.PI * radius * radius
    }
    @Override
    public void shapeState() {
        System.out.println("반지름:" + width);
   }
}
```

```java
public class Rect implements Shape {
    private int width;
    private int height;
    @Override
    public double getArea() {
        return width * height;
    }
    @Override
    public void shapeState() {
        System.out.println("너비:" + width);
        System.out.println("높이:" + height);
    }
}
```

Shape 인터페이스를
구현할 때는 반드시
getArea() 메소드와
shapeState() 메소드를
오버라이드 해야 함

Java