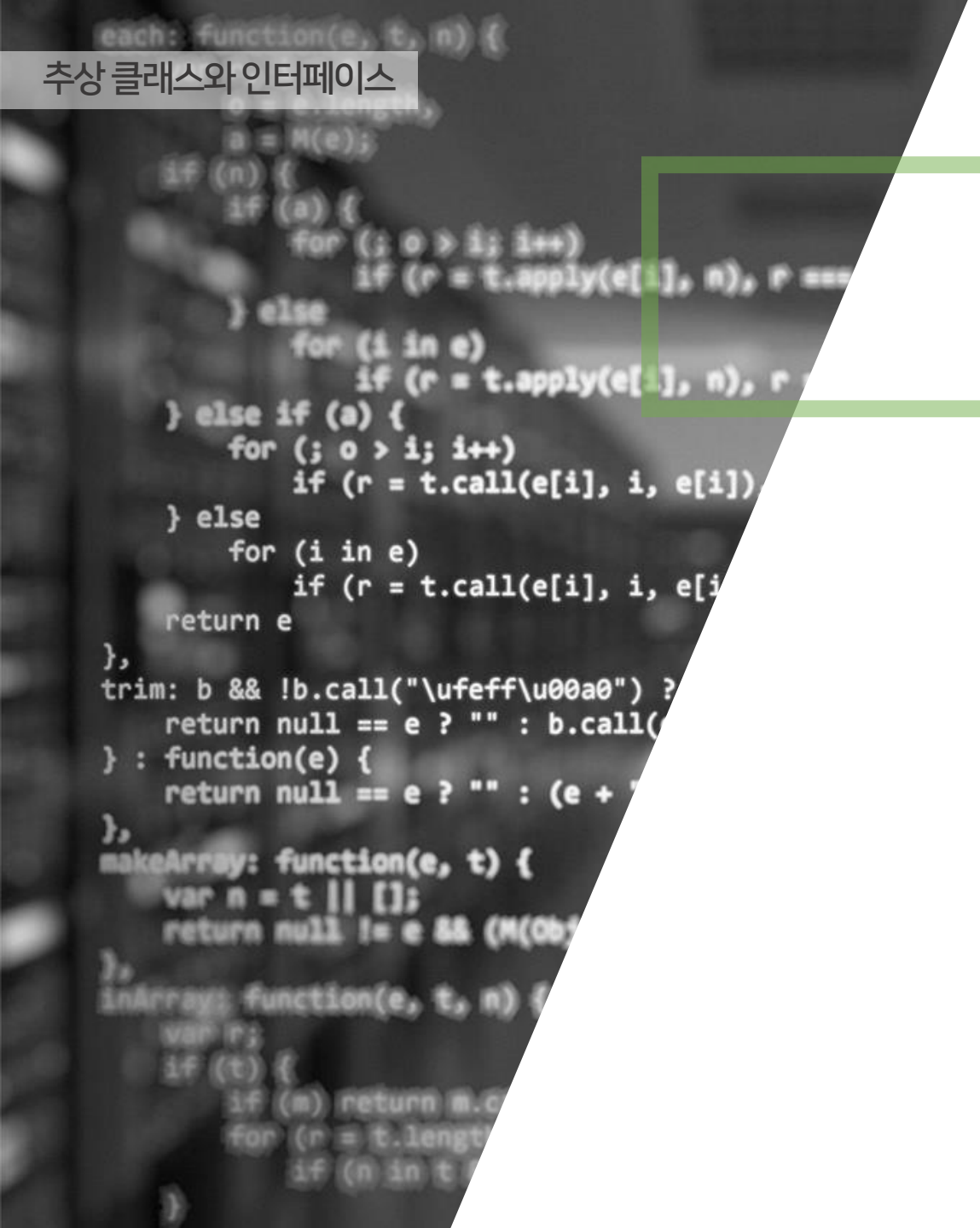
제10장

추상 클래스와 인터페이스

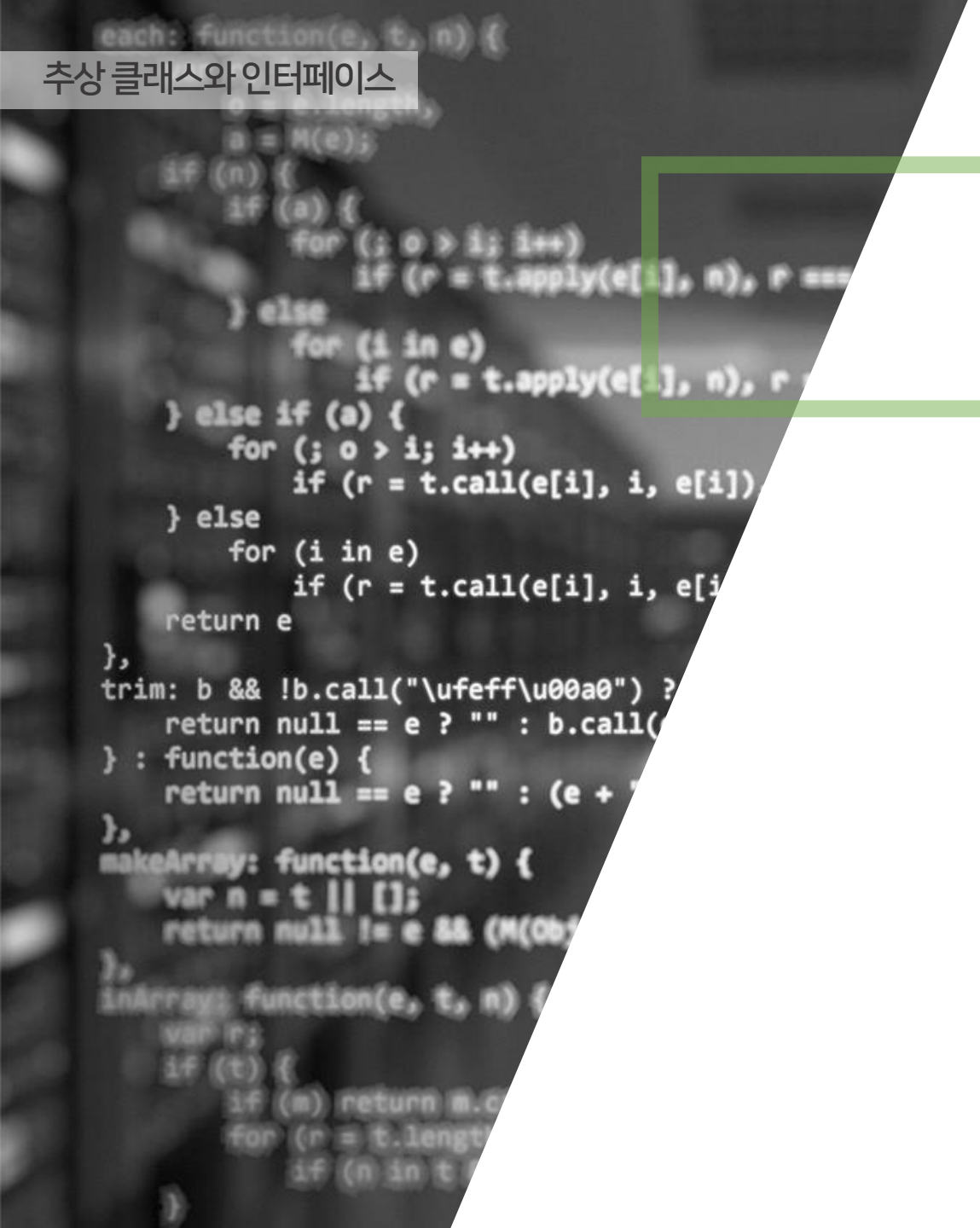
구디아카데미 ▷ 민경태 강사

Java



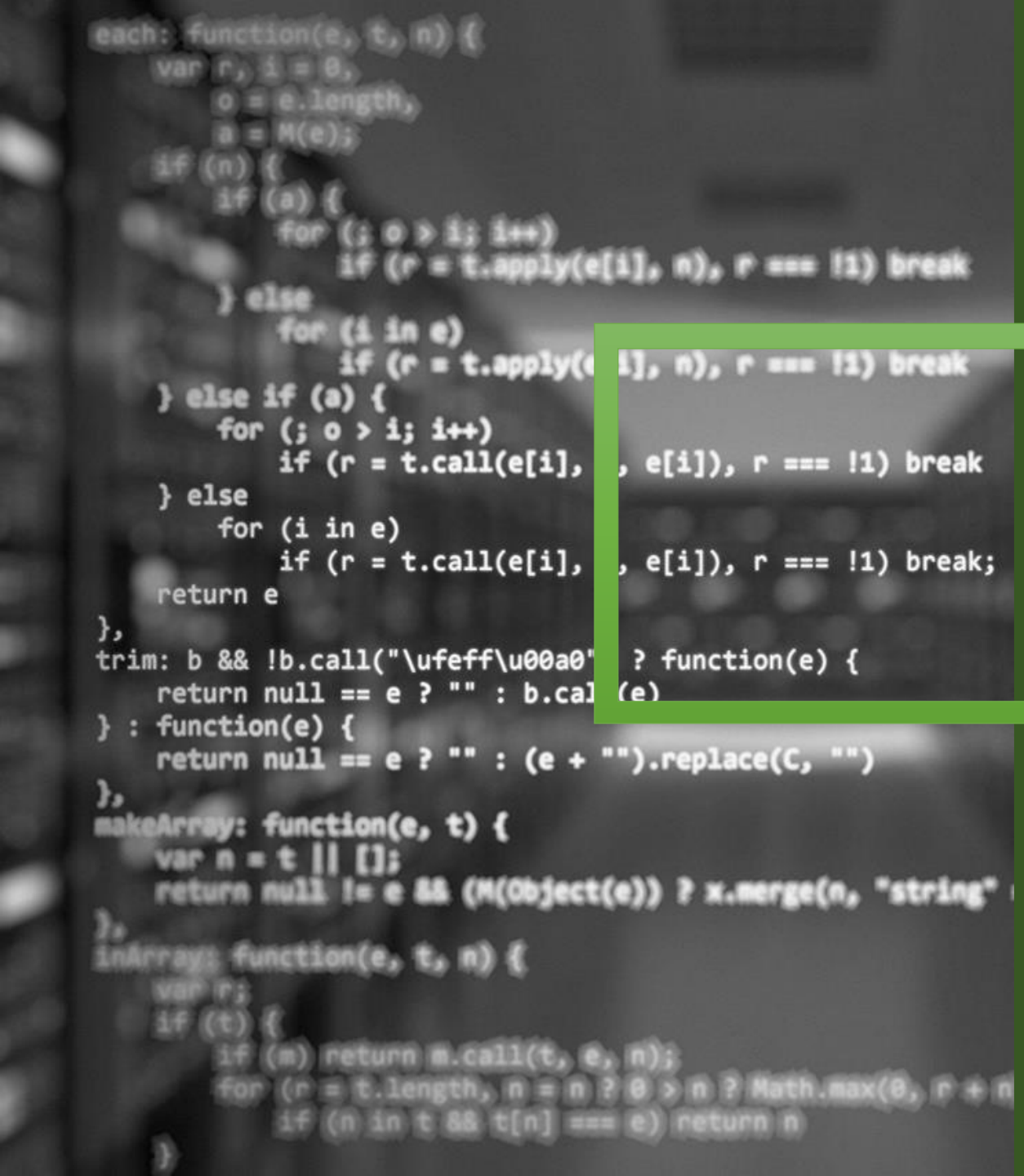
학습목표

1. 추상 메소드와 추상 클래스에 대해서 알 수 있다.
2. 인터페이스에 대해서 알 수 있다.



목차

1. 추상 메소드와 추상 클래스
2. 인터페이스



1. 추상 메소드와 추상 클래스

추상적이란?

- "추상적"의 사전적 의미

어떤 사물이 직접 경험하거나 지각할 수 있는
일정한 형태와 성질을 갖추고 있지 않은.

추상 메소드

- Abstract Method
- 형태를 갖추지 못한 메소드
- 정확히는 본문이 없는 메소드를 의미함
- 본문이 필요 없는 메소드를 추상 메소드로 만듬

추상 메소드 형식

- 메소드 본문을 구성하는 코드 블록({})을 제거하고 abstract 키워드를 추가하면 추상 메소드를 만들 수 있음

```
public abstract void method();
abstract public void method();
```

abstract의 위치는 조정이 가능하다.

중괄호 {} 없이 세미콜론으로 마무리한다.

추상 클래스

- Abstract class
- 형태를 갖추지 못한 클래스
- 추상 메소드를 포함하고 있는 클래스를 의미함
- 추상 클래스는 일반 메소드와 추상 메소드를 모두 가질 수 있음

추상 클래스 형식

- 클래스 선언 시 abstract 키워드를 추가함

```
public abstract class MyClass {
    public abstract void method();
}
```

추상 클래스가 필요한 이유

- 추상 클래스는 실행할 수 없는 메소드(본문이 없는 추상 메소드)를 포함하고 있으므로 객체를 생성할 수 없음*

- 추상 클래스는 객체를 만들기 위해 존재하는 클래스가 아님

- 추상 클래스는 추상 클래스를 상속받는 서브 클래스들의 공통 타입을 제공하기 위해서 존재함

* 필요하다면 Anonymous Inner Type(익명 내부 타입)으로 객체를 생성할 수 있다.

```
추상클래스 참조변수 = new 추상클래스() {
   추상 메소드 오버라이드
};
```

추상 클래스와 상속

- 추상 클래스를 상속 받는 클래스들은 추상 클래스에 포함된 모든 추상 메소드를 반드시 오버라이드 해야 함

- 일반 메소드의 경우 오버라이드는 필수가 아니므로 오버라이드 여부를 선택할 수 있음

추상 클래스와 상속 예시

- 추상 클래스 Animal과 이를 상속 받는 Dog 클래스

```
public abstract class Animal {
    public abstract void sound();
}
```

추상 메소드

```
public class Dog extends Animal {
    @Override
    public void sound() {
        System.out.println("멍멍");
    }
}
```

추상 메소드는 반드시 오버라이드 해야 한다.

추상 클래스가 만들어지는 과정

1. 매운라면과 자장라면 클래스가 있음

class 매운라면

public void 조리법() {
물 500ml를 끓인다.
라면과 스프를 넣고 4분간 끓인다.
}

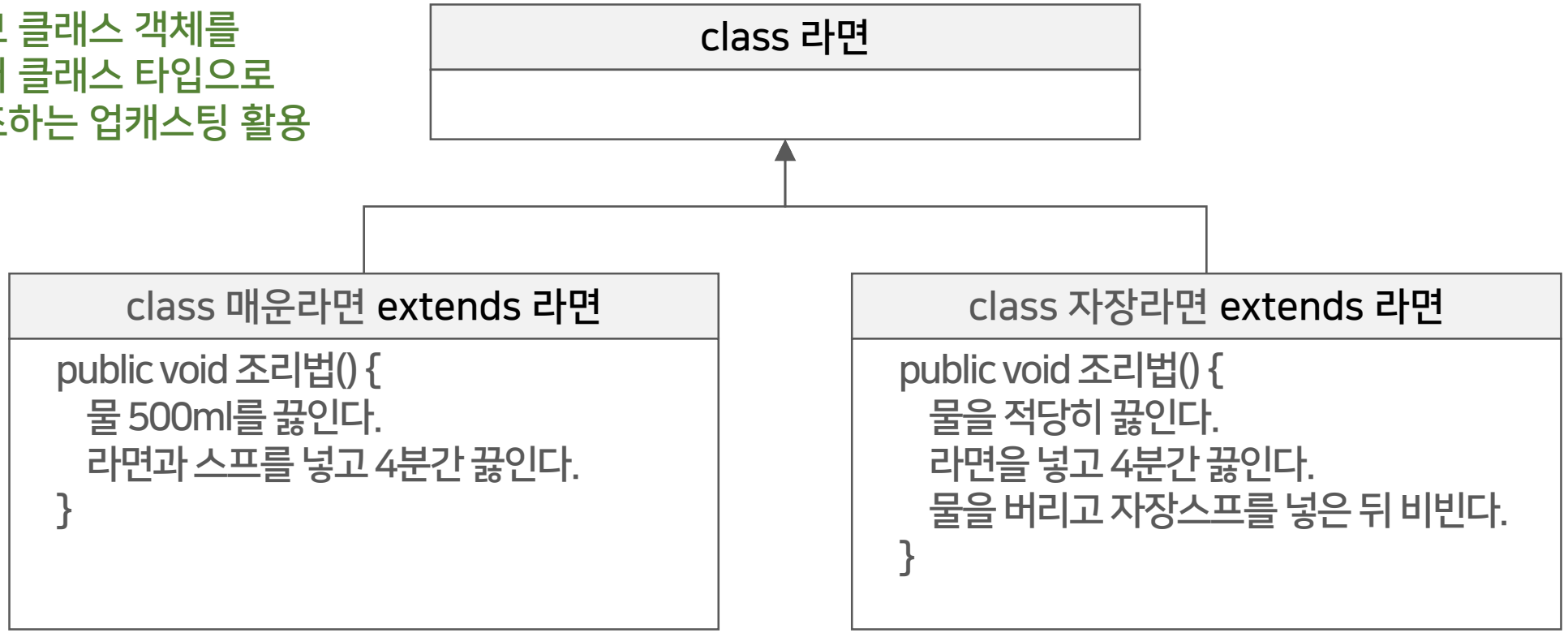
class 자장라면

public void 조리법() {
물을 적당히 끓인다.
라면을 넣고 4분간 끓인다.
물을 버리고 자장스프를 넣은 뒤 비빈다.
}

추상 클래스가 만들어지는 과정

2. 매운라면과 자장라면을 하나의 타입으로 관리하기 위해서 라면 클래스를 만들고 상속 관계로 구성함

서브 클래스 객체를
슈퍼 클래스 타입으로
참조하는 업캐스팅 활용



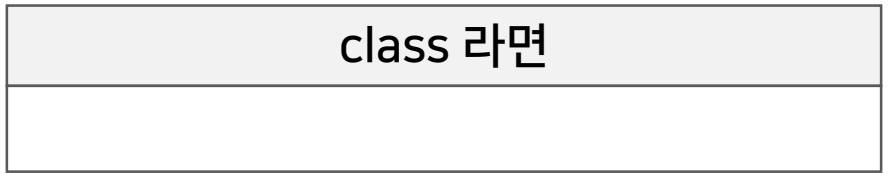
라면 ramen = new 매운라면();

라면 ramen = new 자장라면();

추상 클래스가 만들어지는 과정

3. 업캐스팅시 슈퍼 클래스의 메소드만 호출할 수 있으므로
   조리법() 메소드를 호출할 수 없음

라면 클래스에
조리법() 메소드가
없는 상태



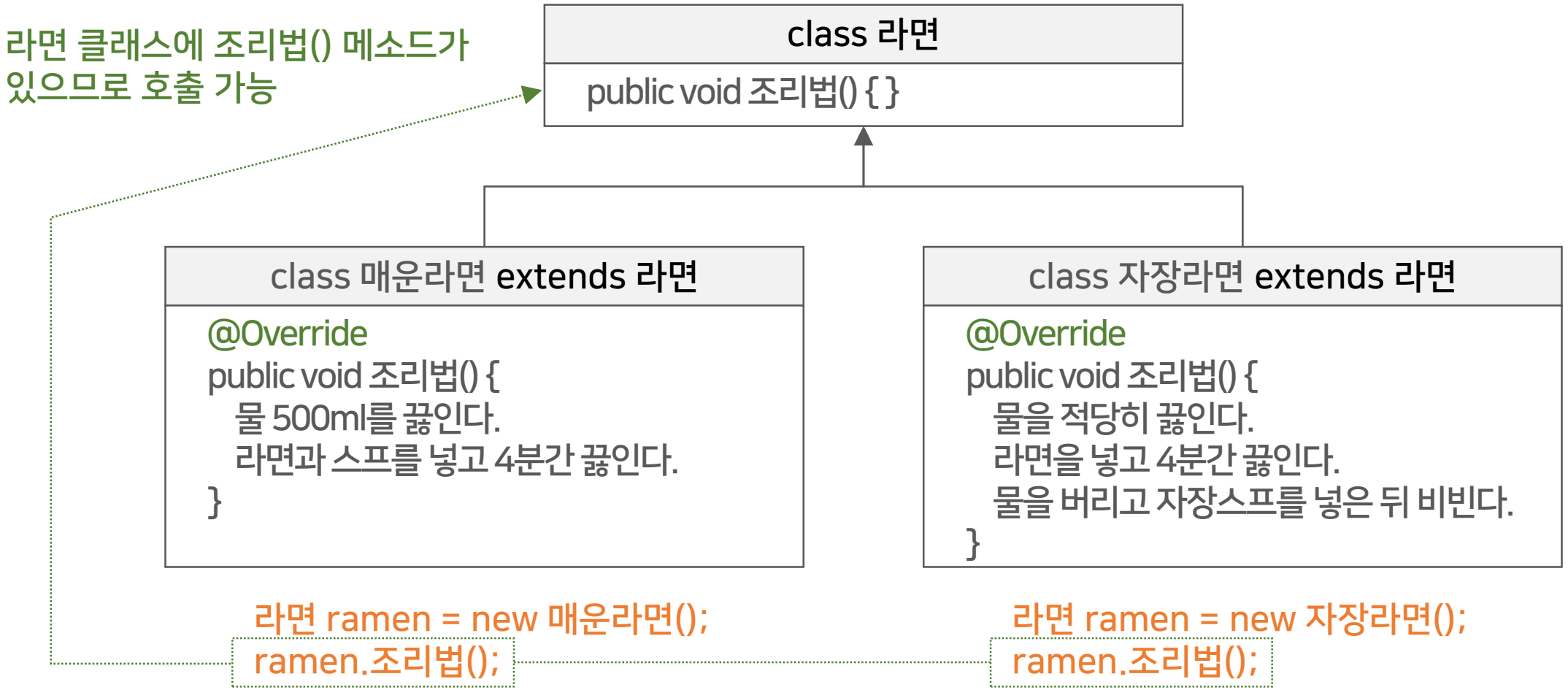
라면 ramen = new 매운라면();
ramen.**조리법()**;

라면 ramen = new 자장라면();
ramen.**조리법()**;

컴파일 오류 발생

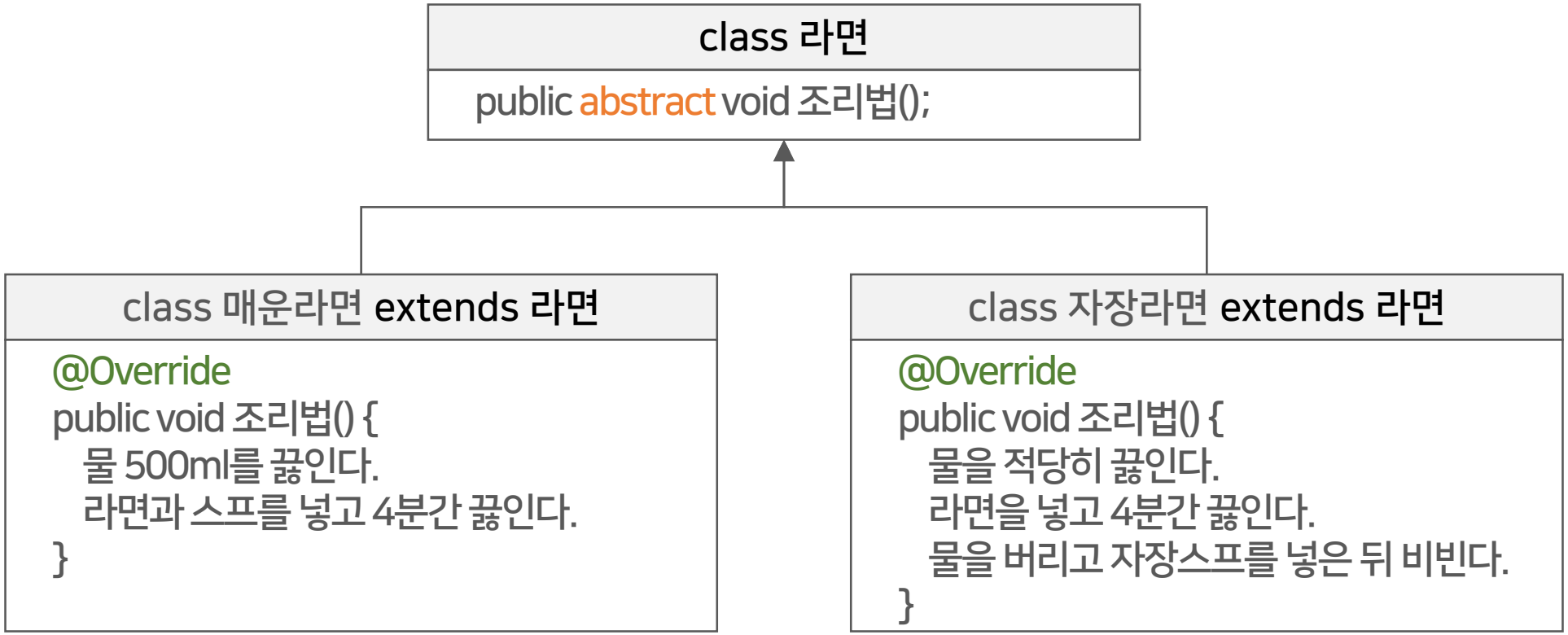
추상 클래스가 만들어지는 과정

4. 매운라면과 자장라면이 조리법() 메소드를 호출할 수 있도록
   라면 클래스에 조리법() 메소드를 추가하고 오버라이드 하도록 함



추상 클래스가 만들어지는 과정

5. 라면 클래스의 조리법() 메소드는 실행하지 않는 메소드이므로 본문이 없는 추상 메소드로 바꿈



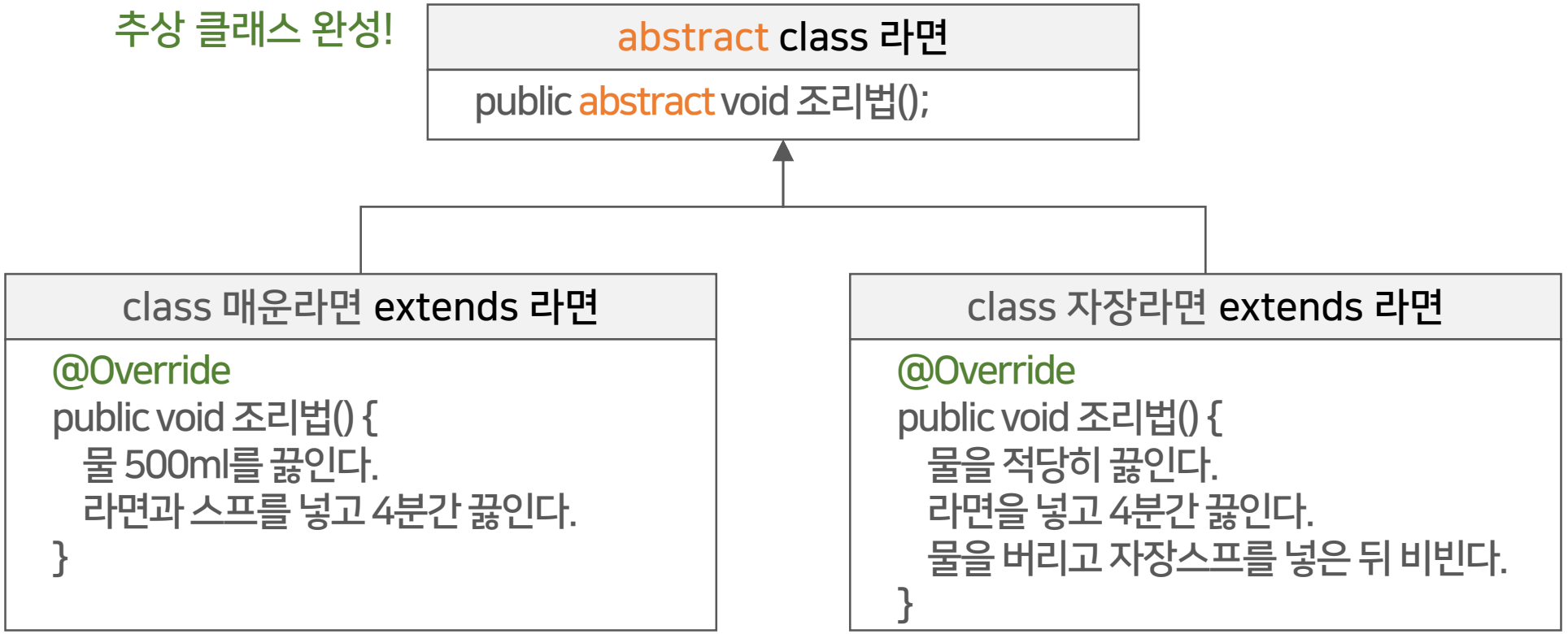
라면 ramen = new 매운라면();
ramen.조리법();

라면 ramen = new 자장라면();
ramen.조리법();

추상 클래스가 만들어지는 과정

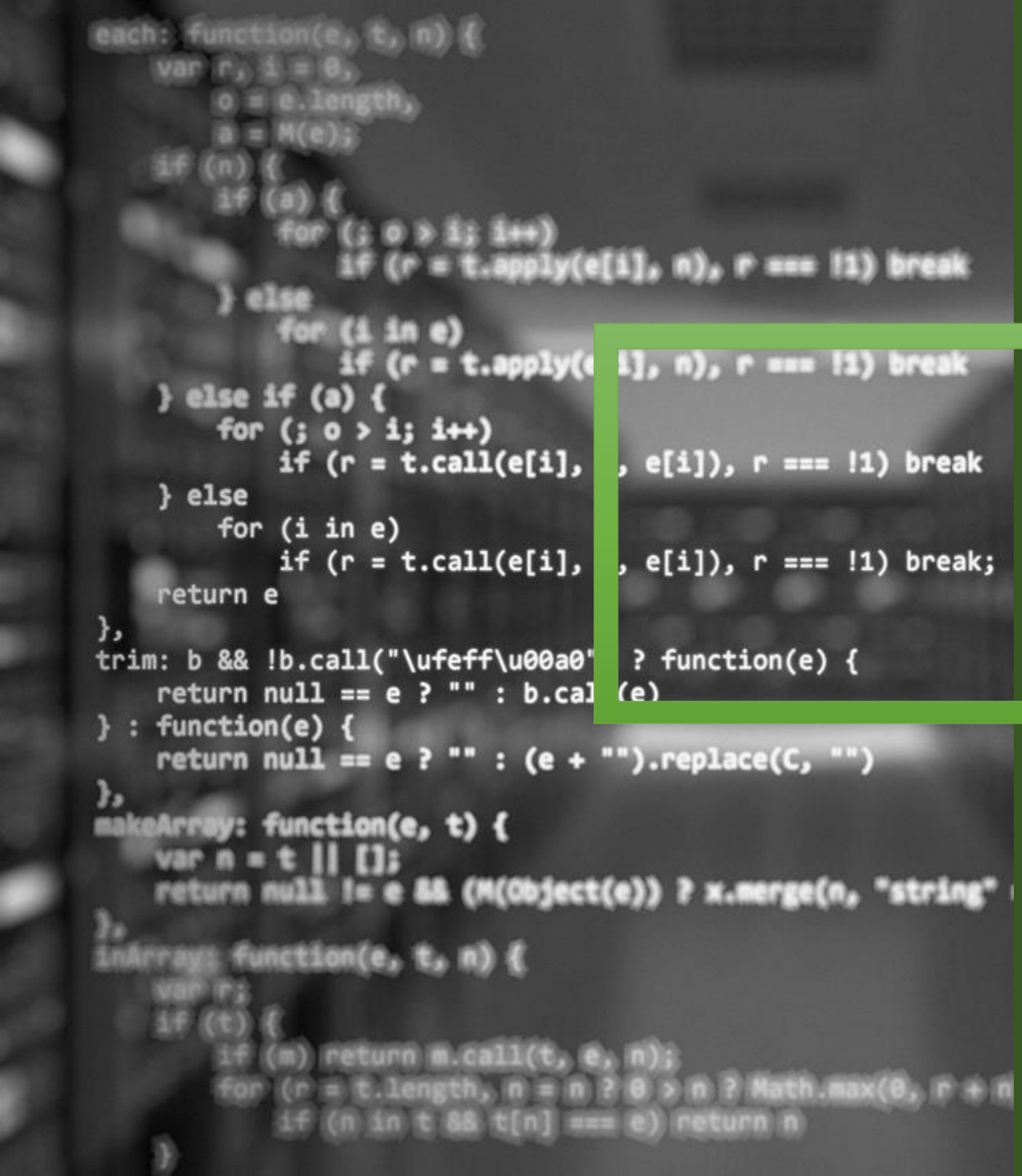
6. 라면 클래스는 추상 메소드를 가지고 있으므로 추상 클래스로 바꿈

추상 클래스 완성!



라면 ramen = new 매운라면();
ramen.조리법();

라면 ramen = new 자장라면();
ramen.조리법();



2. 인터페이스

현실 세계의 인터페이스

- 어떤 것을 사용하기 위한 접점
  USB 제품들은 모두 동일한 인터페이스를 가진다.

같은 인터페이스로 USB메모리, 외장하드, USB 키보드 모두 연결 가능하다.

자바의 인터페이스

- 어떤 클래스들의 공통 타입
  인터페이스를 구현한 클래스들은 모두 인터페이스 타입으로 관리한다.

```
public interface Usb {

}
```

USB메모리, 외장하드, 키보드 모두 Usb 인터페이스 타입으로 관리한다.

```
Usb usb1 = new UsbMemory();
Usb usb2 = new Hdd();
Usb usb3 = new Keyboard();
```

인터페이스

- 모든 필드가 public static final로 정의된(상수) 자바 구성요소
- 대부분의 메소드가 public abstract로 정의된(추상 메소드) 자바 구성요소
- JDK 1.8 이후 default 메소드와 static 메소드가 추가되었음
- 인터페이스를 정의할 때는 class 대신 interface 키워드를 사용함

인터페이스 형식

- 상수, 추상 메소드, 디폴트 메소드, 정적 메소드로 구성됨

```
interface 인터페이스명 {
    [public static final] 자료형 필드명 = 값;
    [public abstract] 반환타입 메소드명(매개변수);
    [public] default 반환타입 메소드명(매개변수) { ... }
    [public] static 반환타입 메소드명(매개변수) { ... }
}
```

대괄호 [] 부분은 생략해도 된다.

인터페이스와 상속 비교하기

- 인터페이스를 상속 받는 클래스를 만들 수 있으나 방법이 약간 다름

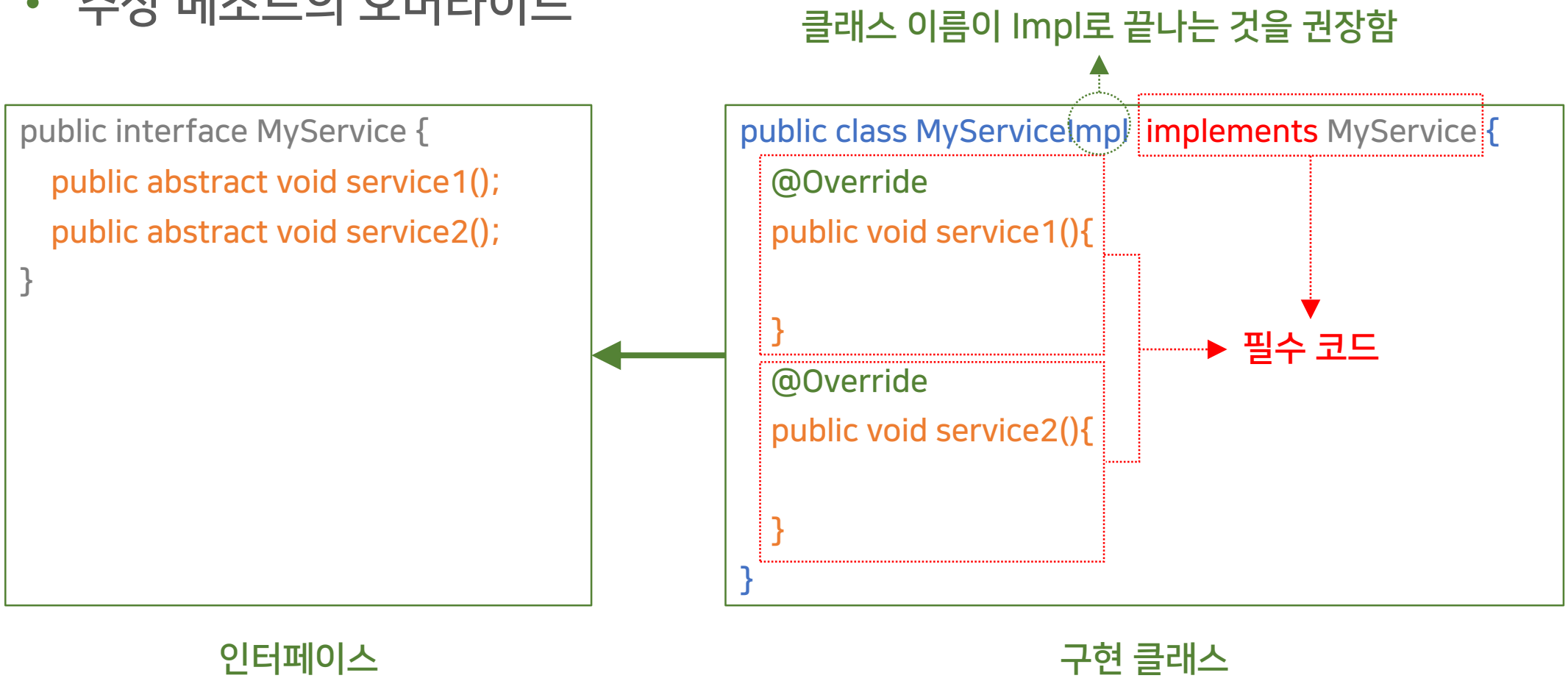
- 인터페이스 구현과 클래스 상속 비교

구분	인터페이스 구현	클래스 상속
키워드	implements	extends
용어	인터페이스를 구현한다.	클래스를 상속한다.
문법	class 클래스 implements 인터페이스 { }	class 클래스 extends 슈퍼클래스 { }
특징	다중 상속 가능	단일 상속만 가능

- 인터페이스를 구현할 땐 반드시 인터페이스의 모든 추상 메소드를 오버라이드 해야 함(추상 클래스와 동일한 규칙)

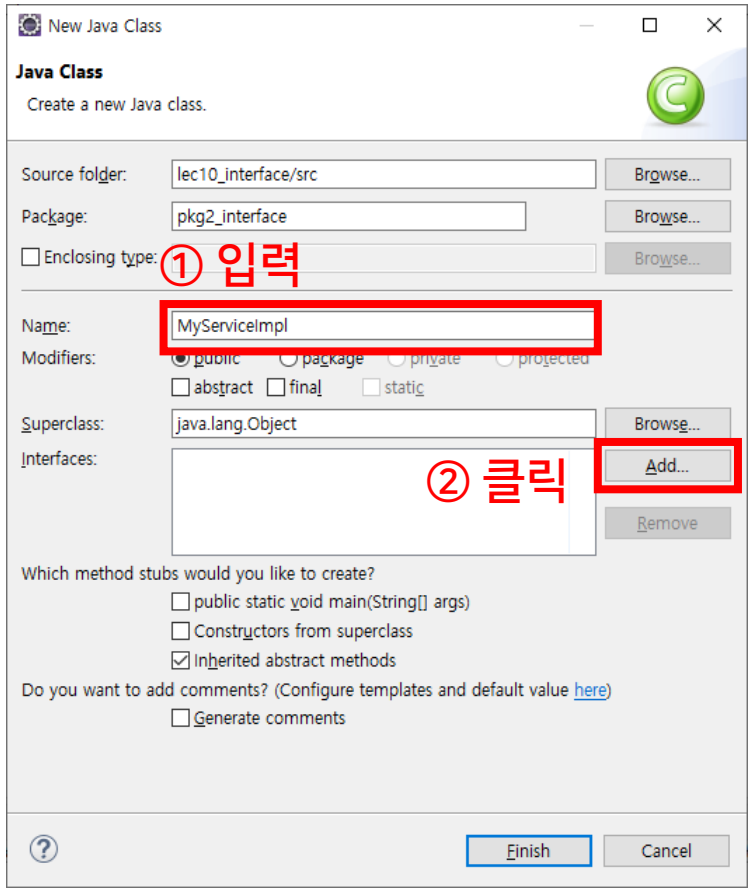
인터페이스와 구현 클래스

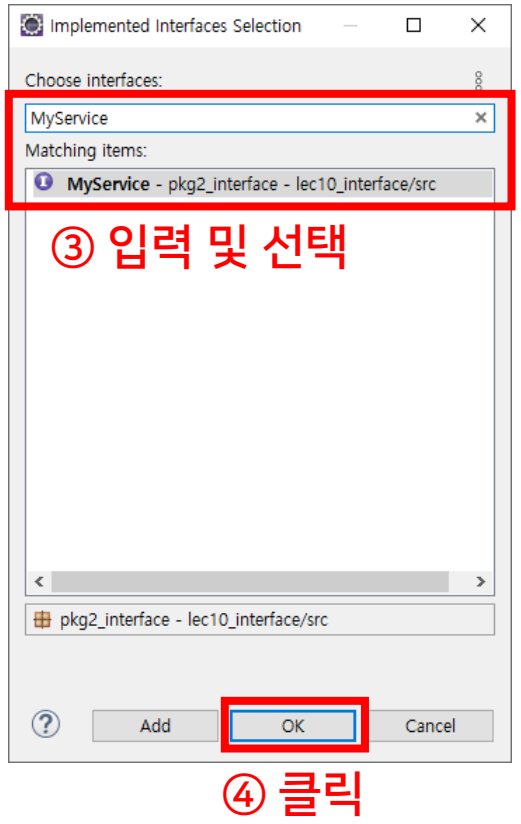
- 인터페이스를 구현하는 클래스는 반드시 다음 작업을 수행해야 함
  - implements 인터페이스
  - 추상 메소드의 오버라이드

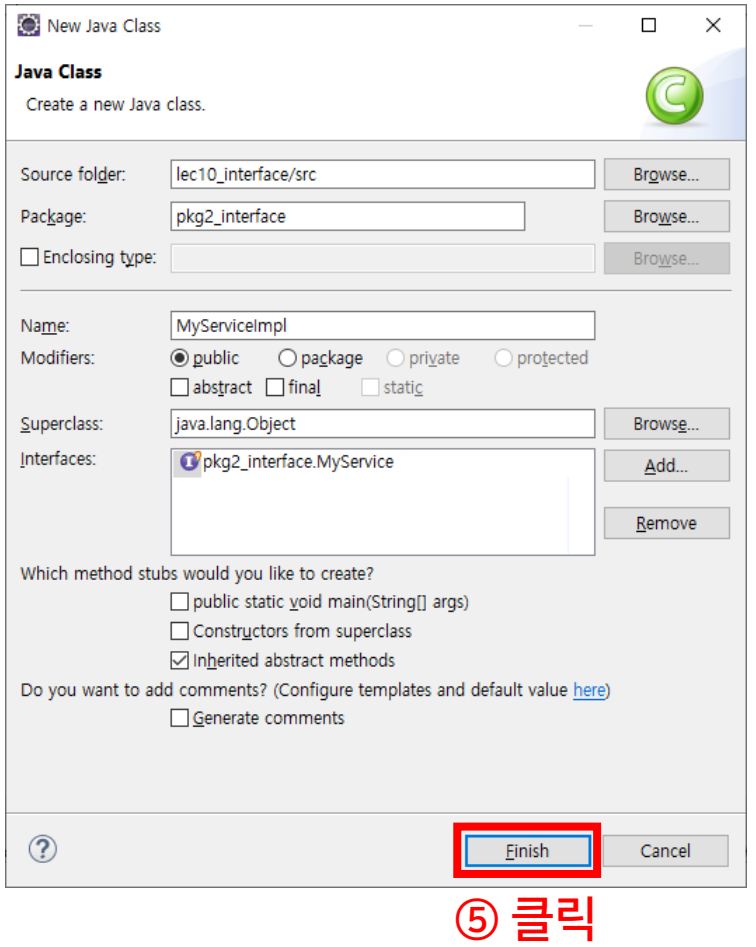


이클립스의 인터페이스 구현 클래스 생성

- 클래스 생성시 구현할 인터페이스를 선택하면 자동으로 필요한 코드 작업을 수행함



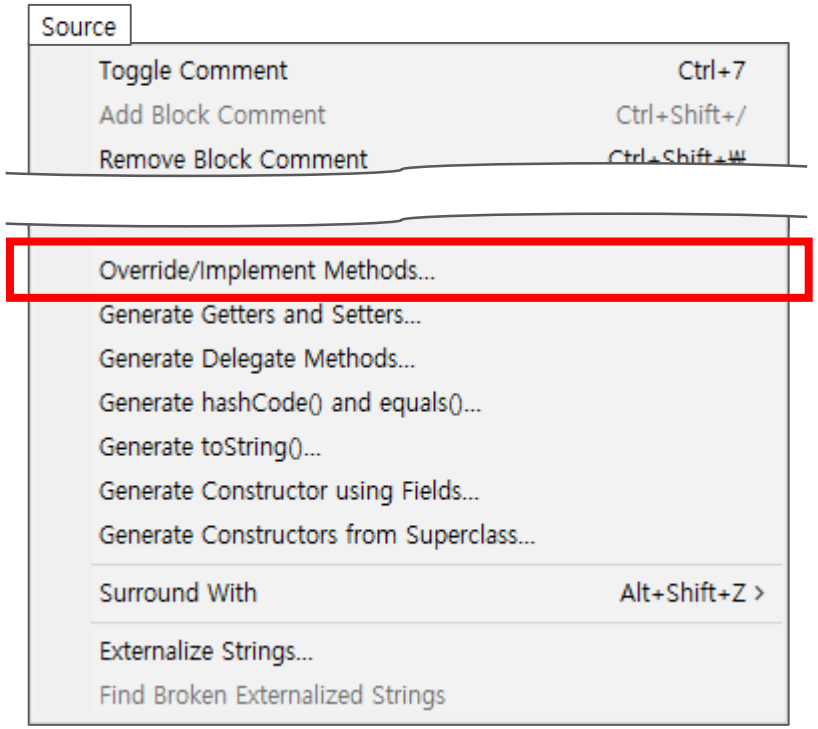




Override/Implement Methods

- 이클립스에는 메소드 오버라이드를 위한 메뉴가 있음

- [Source] - [Override/Implement Methods...]



인터페이스 목적

- 인터페이스에 작성된 메소드는 대부분 본문이 없는 추상 메소드임

- 인터페이스가 제공되면 이를 제공 받은 사람(개발자)은
  인터페이스의 추상 메소드를 모두 그대로
  구현해야 하므로 인터페이스가 작업지시서 역할을 수행할 수 있음

작업지시서

- 예약 서비스 구축을 위한 예약 인터페이스 예시

```
public interface ReserveService {
    public abstract void reserve();
    public abstract void cancel();
    public abstract void confirm();
}
```

```
public class ReserveServiceImpl implements ReserveService {
    @Override
    public void reserve(){ ... }
    @Override
    public void cancel(){ ... }
    @Override
    public void confirm(){ ... }
}
```

예약 서비스 인터페이스

예약 메소드는 public void reserve() { } 형식으로 만드시오.

취소 메소드는 public void cancel() { } 형식으로 만드시오.

확인 메소드는 public void confirm() { } 형식으로 만드시오.

인터페이스 단일 구현

- 하나의 인터페이스만 구현하는 경우

```
public interface Service {
    public abstract void detail();
}
```



```
public class ServiceImpl implements Service {
    @Override
    public void detail() {

    }
}
```

추상 메소드
오버라이드는 필수

인터페이스 다중 구현

- 2개 이상의 인터페이스를 구현하는 경우

```
public interface Phone {
    public abstract void call();
}
```

```
public interface Computer {
    public abstract void game();
}
```

```
public class SmartPhone implements Phone, Computer {
    @Override
    public void call() {

    }
    @Override
    public void game() {

    }
}
```

추상 메소드
오버라이드는 필수

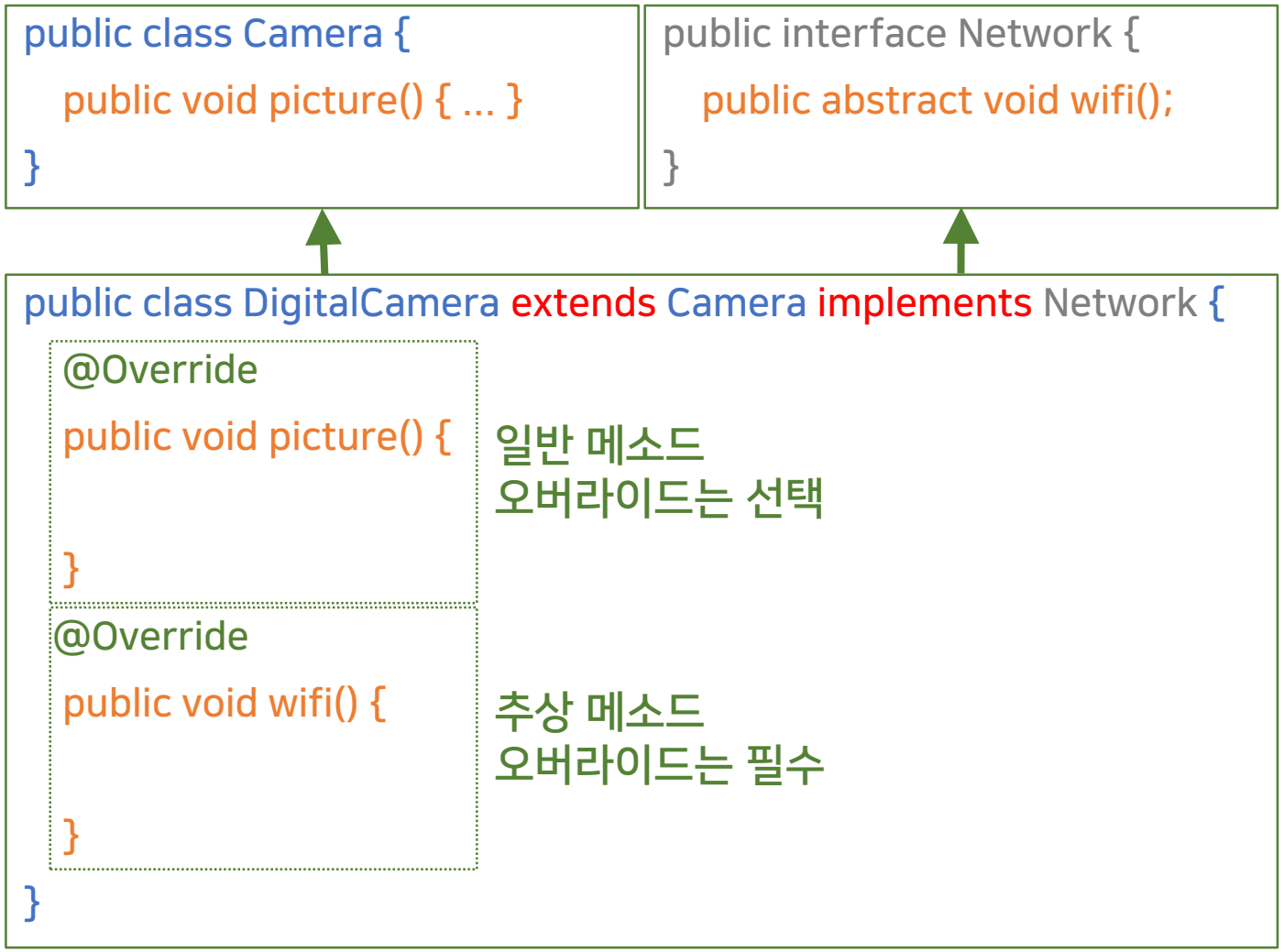
추상 메소드
오버라이드는 필수

클래스 상속과 인터페이스 구현

- 클래스 상속과 인터페이스 구현을 동시에 진행할 수 있음
- 클래스는 오직 하나만 상속할 수 있고 인터페이스는 여러 개를 구현할 수 있음
- 클래스 상속 extends를 먼저하고 인터페이스 구현 implements를 나중에 함(순서 조정 불가능)

클래스 상속과 인터페이스 구현

- 클래스 상속과 인터페이스 구현을 동시에 처리하기



디폴트 메소드

- JDK 1.8부터 지원 시작
- 인터페이스 내부에 본문이 있는 메소드를 추가할 수 있는데 이를 디폴트 메소드라고 함
- 메소드 앞에 default 키워드를 추가하고 생성함
- 사실상 일반 메소드와 같음
- 인터페이스를 구현한 클래스를 만들고 객체를 생성한 뒤 사용할 수 있음

디폴트 메소드의 오버라이드

- 인터페이스 구현시 디폴트 메소드의 오버라이드는 필수가 아님

```
public interface Animal {

    // 추상 메소드
    public abstract void sound();

    // 디폴트 메소드
    public default void alive() {

    }

}
```

```
public class  Dog implements Animal {

    @Override
    public void sound() {

    }

    @Override
    public void alive() {

    }

}
```

추상 메소드
오버라이드는 필수

디폴트 메소드의
오버라이드는 선택

정적 메소드

- JDK 1.8부터 지원 시작
- 메소드에 static 키워드를 추가하면 정적 메소드가 됨
- 클래스의 static 메소드와 개념이 동일함
- 클래스의 static 메소드와는 다르게 객체를 이용한 호출은 불가능하고 인터페이스를 이용한 호출만 가능함
- 호출 형식

  인터페이스.메소드()

정적 메소드 예시

- 인터페이스의 정적 메소드는 인터페이스를 이용해 곧바로 호출 가능함

```
public interface Animal {

   // 정적 메소드
   public static void born() {

   }

}
```

Animal.born(); 형식으로 호출한다.

인터페이스간 상속

- 어떤 인터페이스를 다른 인터페이스가 상속 받을 수 있음
- 인터페이스간 상속에서는 implements 대신 extends 키워드를 사용함
- 클래스의 상속과 달리 다중 상속이 가능함
- 형식

```
interface 인터페이스 extends 인터페이스1, 인터페이스2 {

}
```

인터페이스간 상속 예시

- Interface Phone을 상속 받은 Interface SmartPhone

```
public interface Phone {
    public abstract void call();
}
```

```
public interface SmartPhone extends Phone {
    public abstract void game();
}
```

```
public class SmartPhoneImpl implements SmartPhone {
    @Override
    public void call() { ... }
    @Override
    public void game() { ... }
}
```

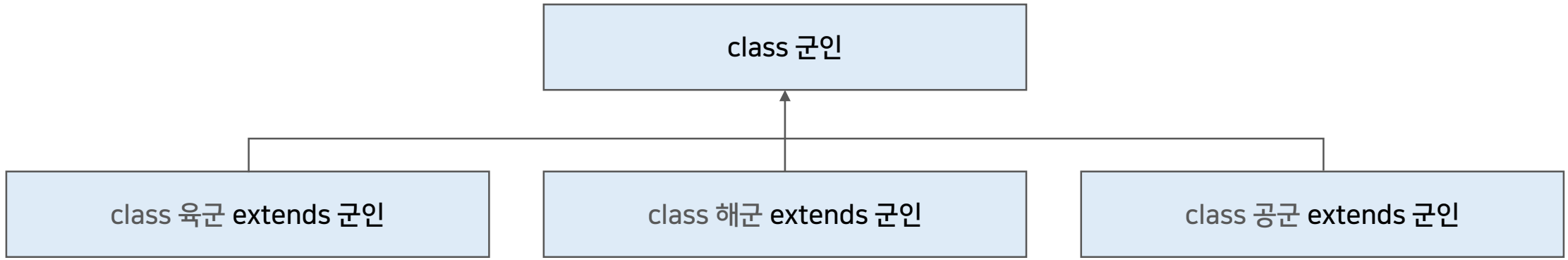
SmartPhone 인터페이스의 추상 메소드와
SmartPhone이 상속 받은 Phone 인터페이스의
추상 메소드를 모두 오버라이드 해야 한다.

Marker Interface

- 선언된 상수나 메소드가 없는 비어 있는 인터페이스를 의미함
- 자바 프로그램을 실행할 때 객체의 타입을 확인하기 위한 용도로 사용함
- 일반적으로 "~할 수 있는(able)"으로 끝나는 이름을 가짐
- 자바 표준 API에 있는 대표적인 Marker Interface
  - java.io.Serializable
  - java.lang.Cloneable

Marker Interface 활용 상황1

- 먼저, 상속 구조를 확인합니다.



Marker Interface 활용 상황2

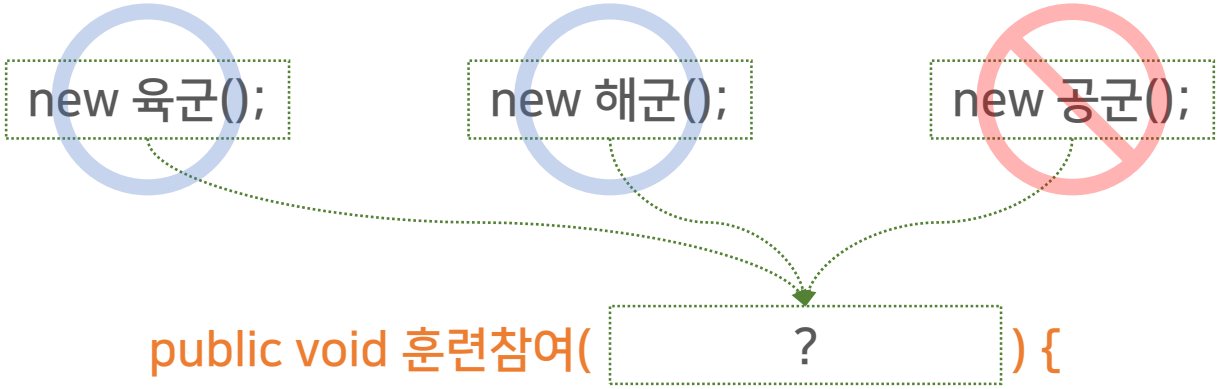
- 육군, 해군은 되고 공군은 안 되는 매개변수는 무엇입니까?

① public void 훈련참여(군인 soldier) { … }

② public void 훈련참여(육군 soldier) { … }

③ public void 훈련참여(해군 soldier) { … }

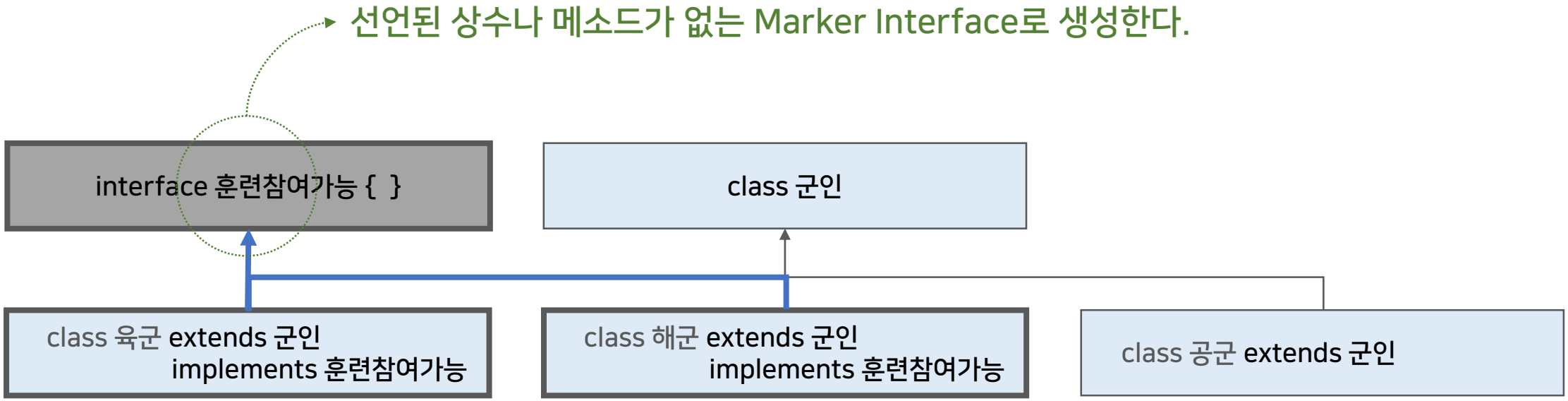
④ public void 훈련참여(공군 soldier) { … }



정답 : 없음

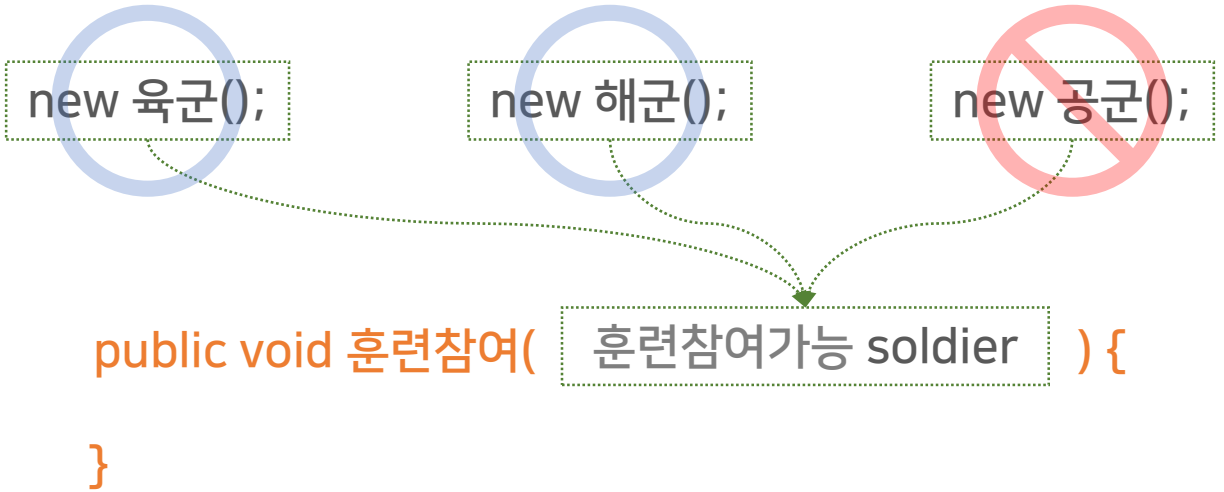
Marker Interface 활용 상황3

- 문제 해결을 위해서 "훈련참여가능" 인터페이스를 만들고 육군, 해군이 구현하도록 한다.



Marker Interface 활용 상황4

- 매개변수를 "훈련참여가능" 인터페이스 타입으로 설정한다.



훈련참여가능 soldier = new 육군(); 가능하다.
훈련참여가능 soldier = new 해군(); 가능하다.
훈련참여가능 soldier = new 공군(); 컴파일 오류가 발생한다.