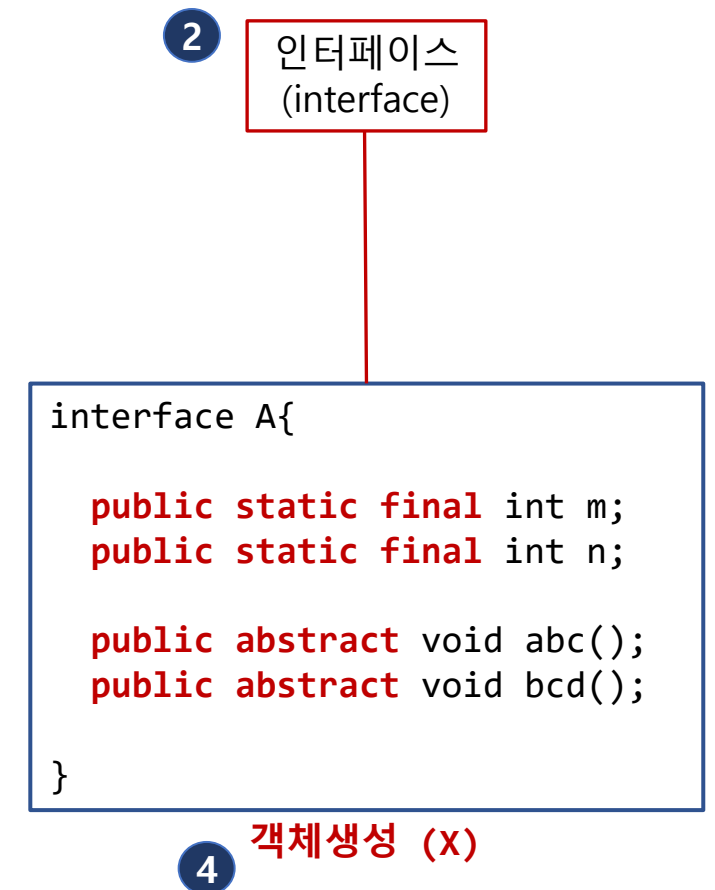
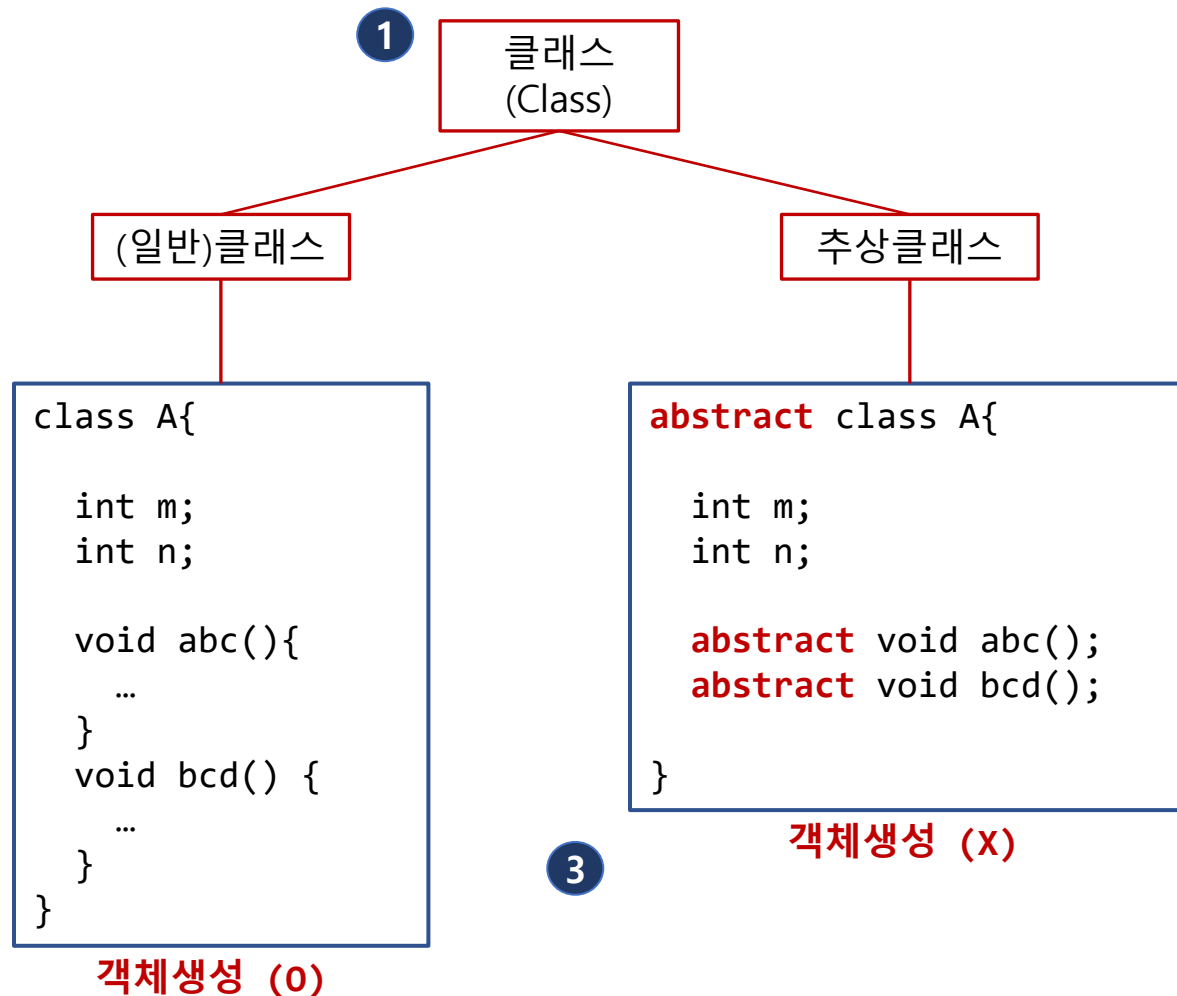


추상클래스(abstract class)와 인터페이스(interface)

추상클래스 (abstract class)

추상클래스(abstract class)

☞ 추상클래스의 위치



추상클래스(abstract class)

☞ 추상클래스란

- ① - 추상메서드(abstract method)를 포함한 클래스

메서드의 본체가 정의되지 않은 **미완성** 메서드

②

추상메서드

abstract void abc();

메서드의 본체({ })가 없고
세미콜론(;)으로 끝남

③

일반메서드

```
void abc(){  
    System.out.println("일반메서드");  
}
```

☞ 추상클래스의 정의

④

```
abstract class A {  
  
    abstract void abc();  
    void bcd(){  
        ...  
    }  
}
```

⑥

TIP

- 추상메서드를 하나도 포함하지 않아도 **추상클래스로 정의는 가능함**
- 다만 추상메서드가 없는 경우 추상메서드로 정의할 이유가 없음

⑤

```
abstract class A {  
    void bcd(){  
        ...  
    }  
}
```

추상클래스(abstract class)

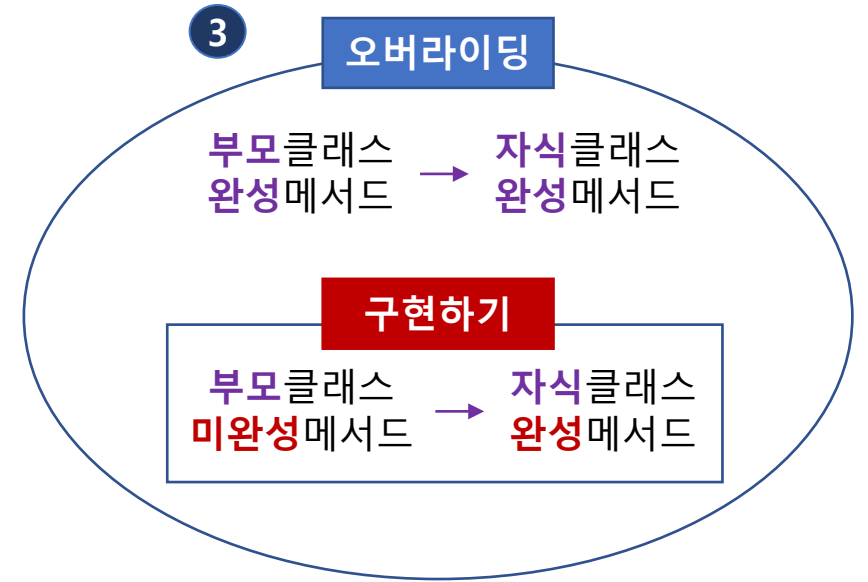
👉 오버라이딩(overriding) vs. 구현하기(implements)

1 - **오버라이딩** (overriding)

: 부모클래스의 메서드(완성/미완성)를 자식클래스에서 재정의(완성)

2 - **구현하기** (implements)

: 부모클래스의 미완성메서드(추상메서드)를 자식클래스에서 재정의(완성)



4 **TIP**

- 메서드의 완성과 미완성의 구분 기준
→ **중괄호의 존재** 여부

중괄호안에 아무런 코드가 없어도 **완성된 메서드**
→ 아무일도 하지 말라고 **명확히 기능이 표현된** 메서드

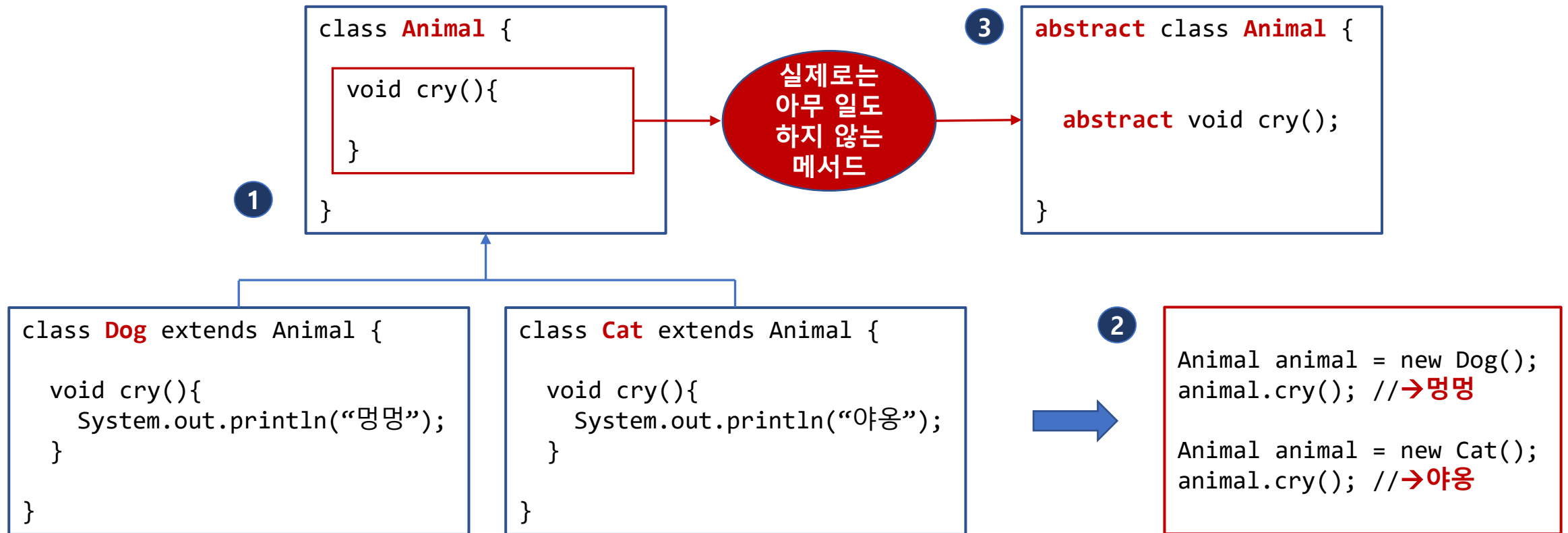
5

```
abstract class A {  
    abstract void abc();  
}
```

```
class B extends A {  
    void abc(){  
    }  
}
```

추상클래스(abstract class)

👉 추상클래스의 필요성



추상클래스(abstract class)

☞ 추상클래스의 특징

- 추상클래스는 그 자체로는 **객체 생성 불가**함 (추상메서드(abstract method/미완성메서드)를 포함하기 때문)

1

```
abstract class A {  
    abstract void abc();  
    void bcd(){  
        ...  
    }  
}
```

객체생성불가

A a = new A(); (X)

2

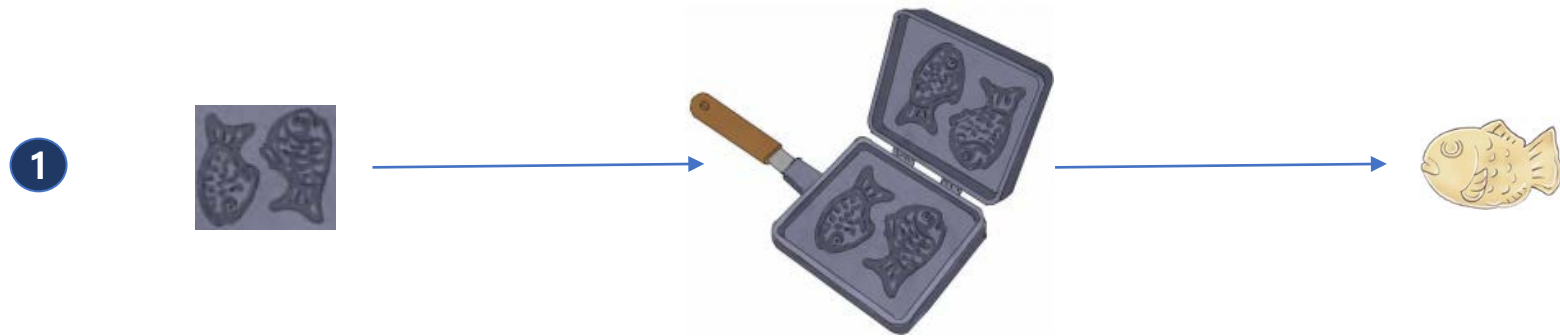
TIP

- 힙(Heap) 메모리에 객체 생성을 위해서는 모두 완성된 메서드를 가질 때만 가능

추상클래스(abstract class)

☞ 추상클래스의 특징

- 추상클래스는 그 자체로는 **객체 생성 불가**함 (추상메서드(abstract method/미완성메서드)를 포함하기 때문)



2 바로 붕어빵 생성 불가 ← 붕어빵기계부품

추상클래스

붕어빵기계

(일반)클래스

붕어빵

객체 생성

3

```
abstract class A {  
    abstract void abc();  
}
```

```
class B extends A {  
    void abc(){ ... }  
}
```

```
A a = new B(); (O)
```

```
B b = new B(); (O)
```

↓ 바로 객체 생성 불가

4

```
A a = new A(); (X)
```


추상클래스(abstract class)

☞ 추상클래스의 객체 생성 (자체로는 객체 생성 불가)

1 - **방법 #1.** 추상클래스를 일반클래스로 상속하여 객체 생성

```
abstract class A {  
    abstract void abc();  
}
```

A a = new A(); (X)

```
class B extends A {  
    void abc(){  
        ...  
    }  
}
```

A a = new B(); (O)
B b = new B(); (O)

3

TIP

- 익명 이너클래스를 사용하면 컴파일러가 내부적으로 클래스를 생성한 후 메서드 오버라이딩 수행 (클래스 이름을 알 수 없음)

2

- **방법 #2.** 익명이너클래스 사용

```
abstract class A {  
    abstract void abc();  
}
```

```
A a = new A() {  
    void abc(){  
        ...  
    }  
};
```

미완성 메서드를
이 메서드로 완성하여
A의 객체를 생성

4

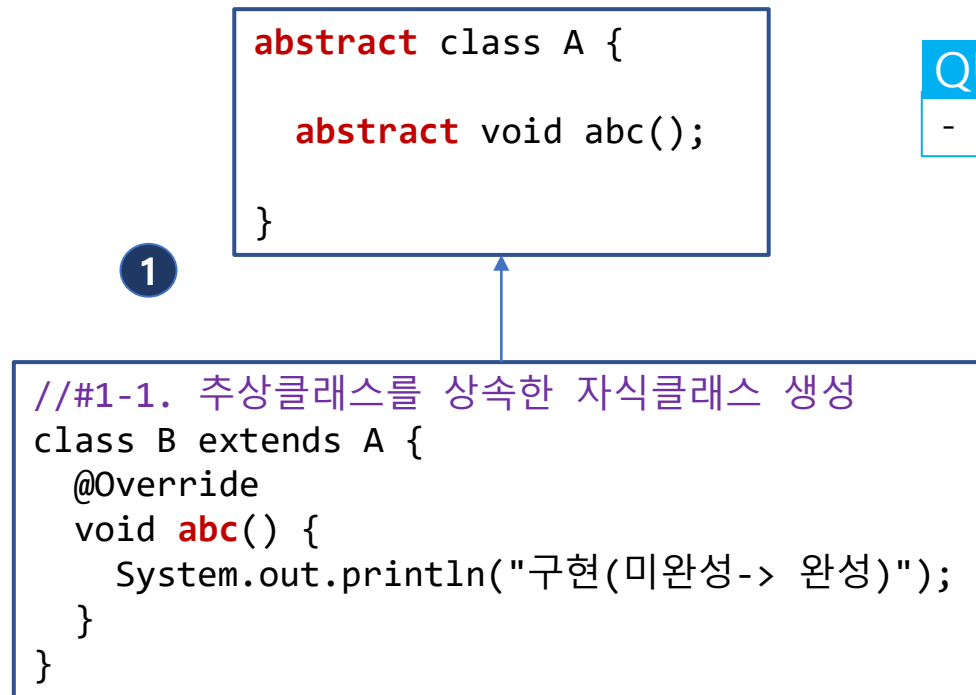
QUIZ

- 추상클래스의 객체 생성시 방법#1과 방법#2의 각각의 장단점은?

추상클래스(abstract class)

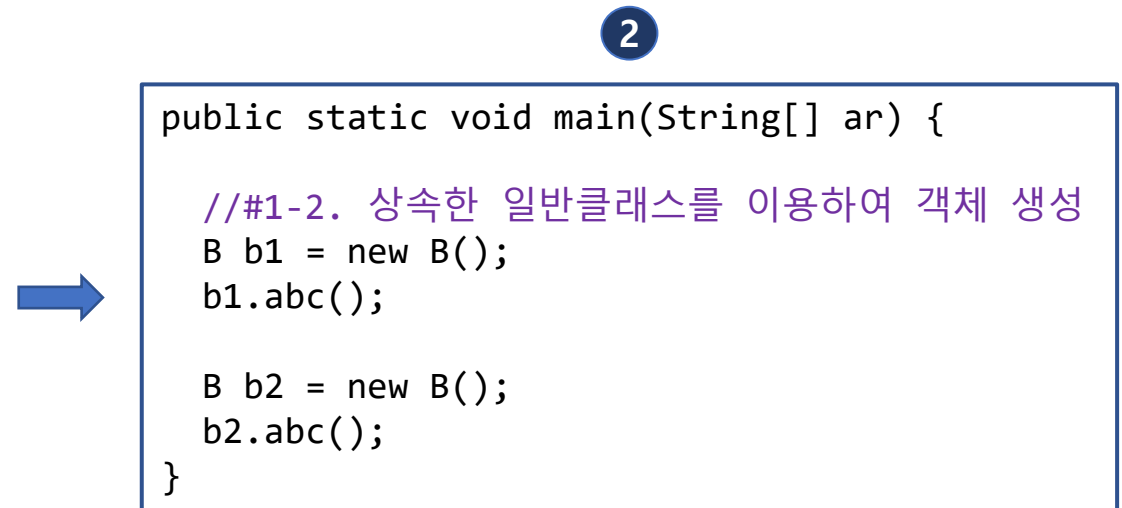
☞ 추상클래스의 객체 생성시 **방법#1과 방법#2의 각각의 장단점**은?

- **방법 #1.** 추상클래스를 일반클래스로 상속하여 객체 생성



QUIZ

- 추상클래스의 객체 생성시 방법#1과 방법#2의 각각의 장단점은?



추상클래스(abstract class)

☞ 추상클래스의 객체 생성시 **방법#1과 방법#2의 각각의 장단점**은?

- **방법 #2.** 익명이너클래스 사용

3 QUIZ

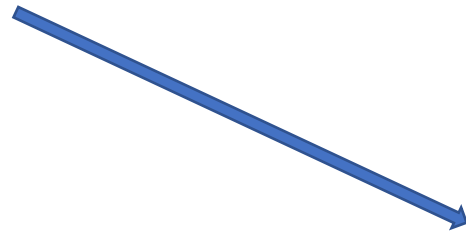
- 추상클래스의 객체 생성시 방법#1과 방법#2의 각각의 장단점은?

1

```
abstract class A {  
    abstract void abc();  
}
```

2

```
public static void main(String[] ar) {  
  
    // #2. 익명 이너클래스  
    A a1 = new A() {  
        @Override  
        void abc() {  
            System.out.println("구현 (미완성-> 완성)");  
        }  
    };  
    A a2 = new A() {  
        @Override  
        void abc() {  
            System.out.println("구현 (미완성-> 완성)");  
        }  
    };  
}
```



The End

인터페이스(interface)

인터페이스(interface)

☞ 인터페이스란

- 1 - 모든 필드가 **public static final**로 정의
- 모든 메서드가 **public abstract**로 정의 (디폴트메서드 제외)
- 디폴트 메서드는 **public**로 정의
- 자체적으로 객체생성 불가

- 2 이 콘센트에 꼽히는 가전제품이 TV인지 냉장고인지 중요하지 않음



여기에 플러그가 꼽히는 가전제품이면
종류에 상관없이 사용할 수 있음
=
이 인터페이스를 충족하면 사용할 수 있음

☞ 인터페이스의 정의

- 3

```
interface A {  
  
    public static final int a = 3;  
    public abstract void abc();  
  
}
```

- 4 생략시 자동으로 추가

```
public static final  
public abstract
```

```
interface A {  
  
    int a = 3;  
    void abc();  
  
}
```

인터페이스(interface)

☞ 인터페이스의 특징

1

```
interface A {  
    public static final int a=3;  
    public abstract void abc();  
}  
  
interface B {  
    int b=3;      //public static final 자동 추가  
    void bcd();  //public abstract 자동 추가  
}
```



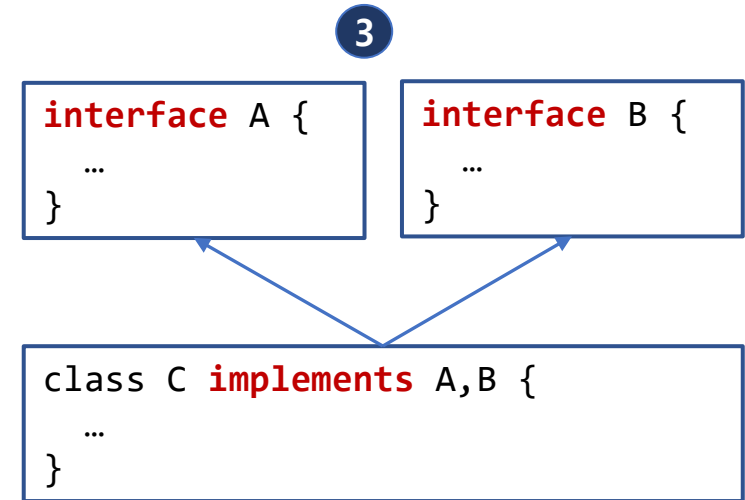
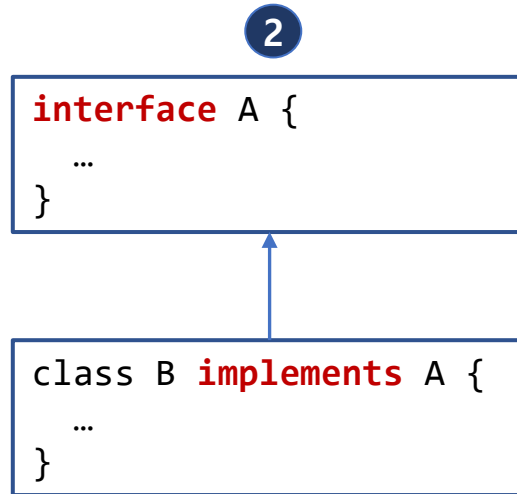
2

```
public static void main(String[] ar) {  
  
    // #1. 필드의 static 특징 확인 (클래스/인터페이스 이름으로 바로 접근 가능)  
    System.out.println(A.a);  
    System.out.println(B.b);  
  
    // #2. 필드의 final 특징 확인 (값 변경 불가)  
    A.a=4;  //(불가능)  
    B.b=4;  //(불가능)  
}
```

인터페이스(interface)

☞ 인터페이스의 상속

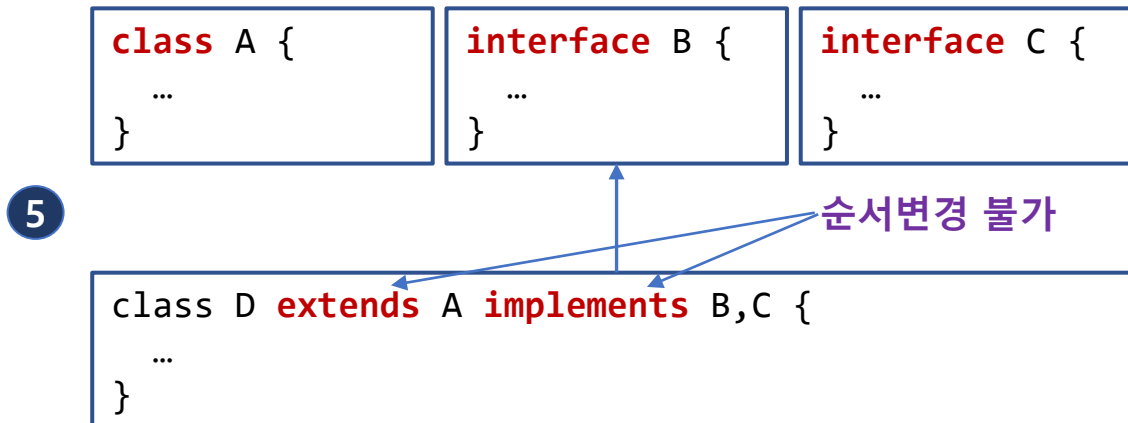
- 1
- 상속시 implements 사용
 - 다중상속 가능



4

TIP

- 클래스(일반클래스/추상클래스)는 다중상속 불가



인터페이스(interface)

☞ 인터페이스의 상속

1 참고. 상속 키워드

```
클래스 extends 클래스 {  
    ...  
}
```

```
클래스 implements 인터페이스 {  
    ...  
}
```

2

TIP

- 동일한 타입(클래스/인터페이스) 상속시 **extends**
- 다른 타입을 상속하는 경우 **implements**

```
인터페이스 extends 인터페이스 {  
    ...  
}
```

3

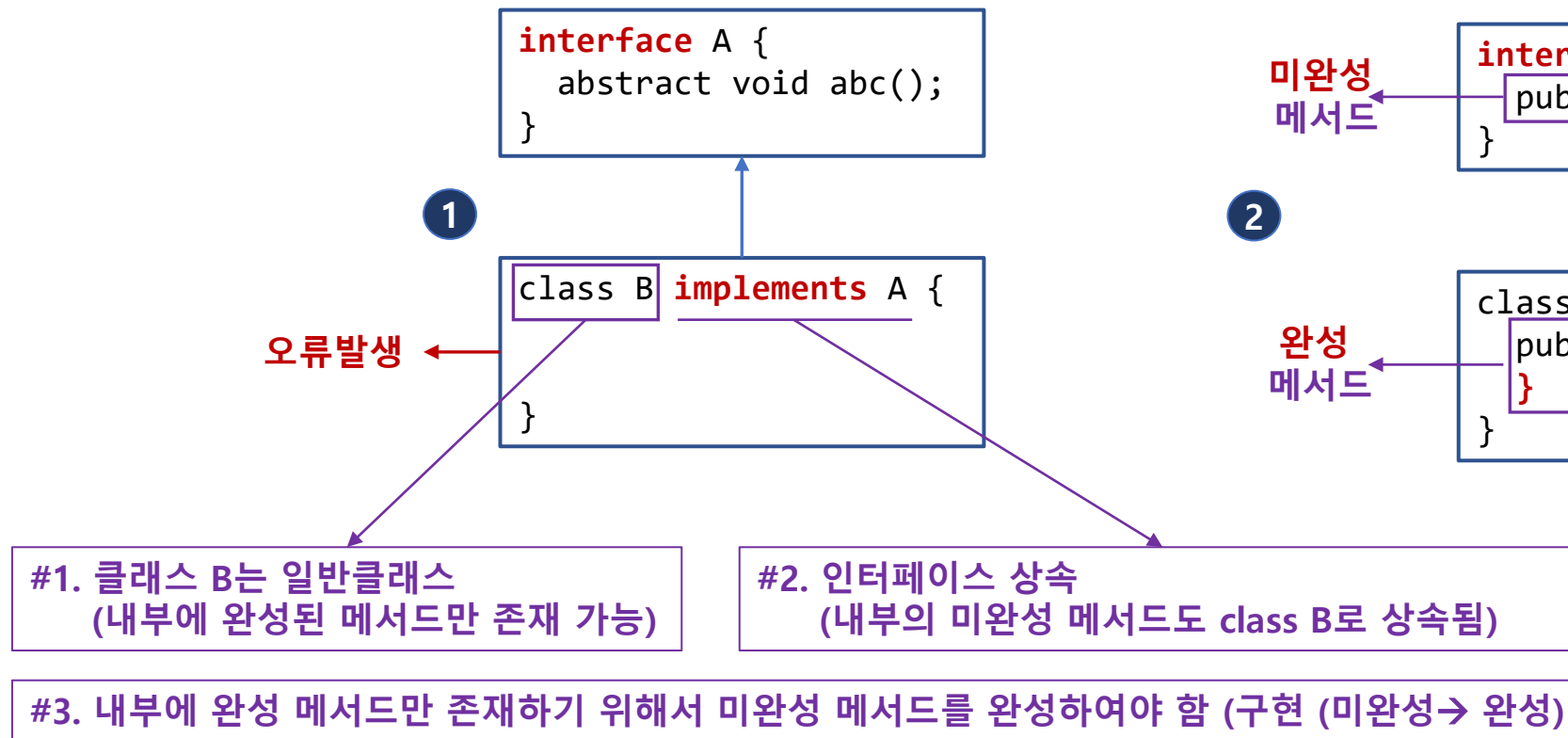
```
인터페이스 implements 클래스 {  
    ...  
}
```

QUIZ

- 인터페이스가 클래스를 상속할 수 없는 이유는?

인터페이스(interface)

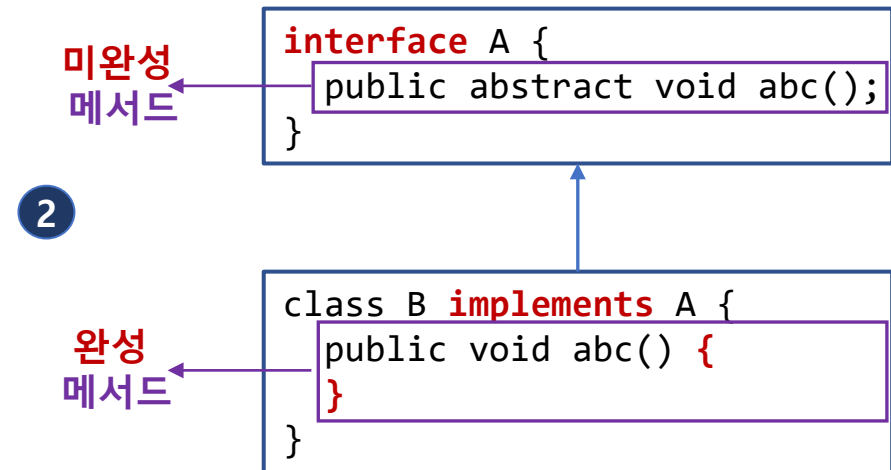
☞ 인터페이스의 상속



3

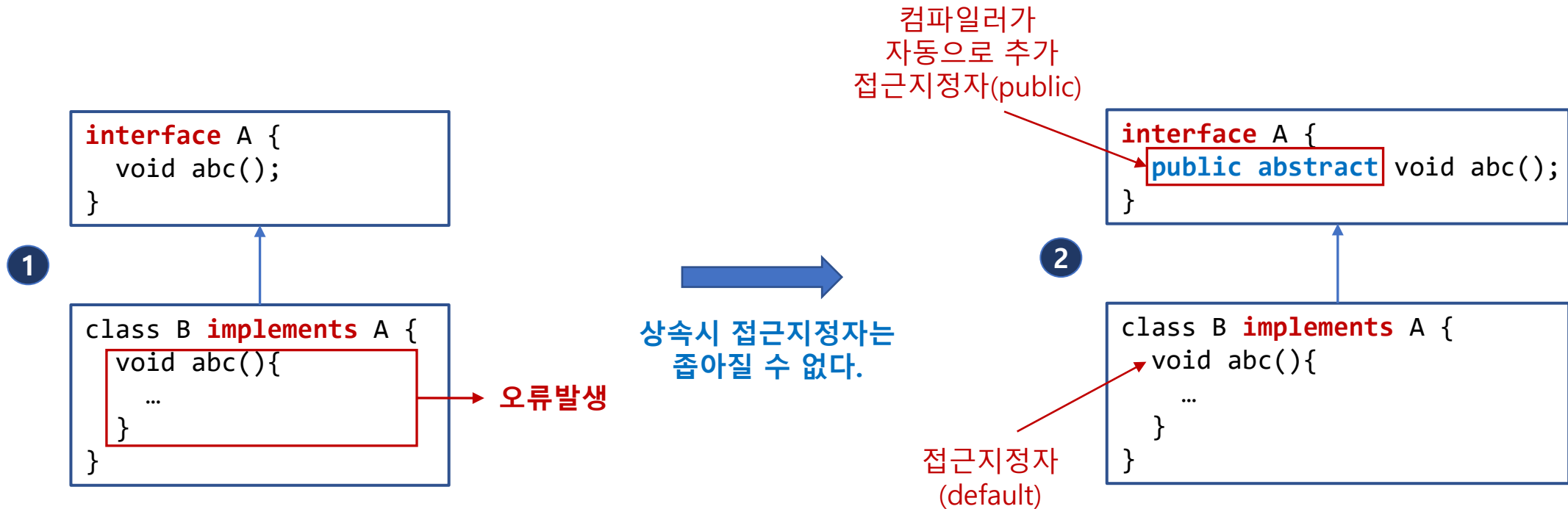
TIP

- 미완성 메서드와 완성메서드를 구분하는 것은 메서드의 내용이 아니라 중괄호의 존재 여부 {}



인터페이스(interface)

☞ 인터페이스의 상속



인터페이스(interface)

☞ 인터페이스의 객체생성 (자체로는 객체 생성 불가)

① - **방법 #1.** 인터페이스를 일반클래스로 상속하여 객체 생성

```
interface A {  
    int a=3;  
    void abc();  
}
```

A a = new A(); (X)

```
class B implements A {  
    public void abc(){  
        ...  
    }  
}
```

A a = new B(); (O)
B b = new B(); (O)

③

TIP

- 인터페이스의 객체생성방법은 추상클래스(abstract class) 객체생성방법과 동일

②

- **방법 #2.** 익명이너클래스 사용

```
interface A {  
    int a=3;  
    void abc();  
}
```

A a = new A() {

```
    public void abc(){  
        ...  
    }
```

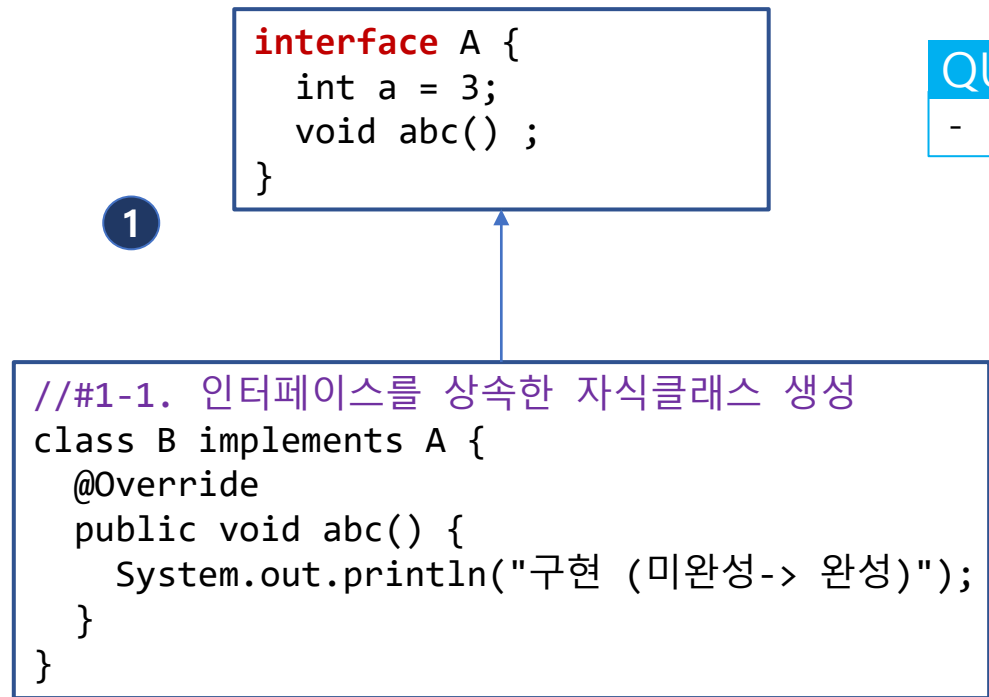
};

미완성 메서드를
이 메서드로 완성하여
A의 객체를 생성

인터페이스(interface)

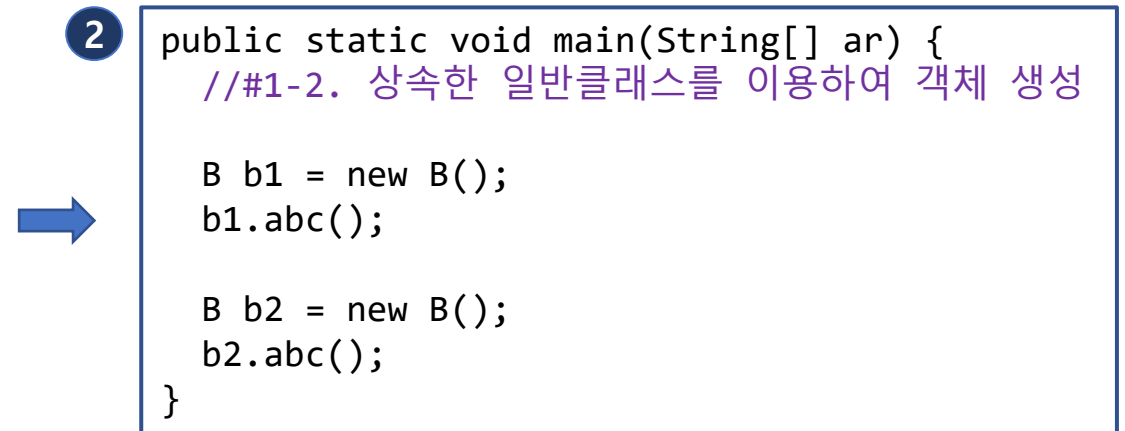
☞ 인터페이스의 객체 생성시 **방법#1과 방법#2의 각각의 장단점**은?

- **방법 #1.** 인터페이스를 일반클래스로 상속하여 객체 생성



QUIZ

- 인터페이스의 객체 생성시 방법#1과 방법#2의 각각의 장단점은?



인터페이스(interface)

☞ 인터페이스의 객체 생성시 **방법#1과 방법#2의 각각의 장단점**은?

- **방법 #2.** 익명이너클래스 사용

1

```
interface A {  
    int a = 3;  
    void abc() ;  
}
```

QUIZ

- 인터페이스의 객체 생성시 방법#1과 방법#2의 각각의 장단점은?

2

```
public static void main(String[] ar) {  
  
    // #2. 익명 이너클래스  
    A a1 = new A() {  
        @Override  
        public void abc() {  
            System.out.println("구현 (미완성-> 완성)");  
        }  
    };  
    A a2 = new A() {  
        @Override  
        public void abc() {  
            System.out.println("구현 (미완성-> 완성)");  
        }  
    };  
}
```

인터페이스(interface)

☞ 인터페이스의 필요성

- 일상생활에서의 인터페이스 vs. 프로그래밍 속에서의 인터페이스

1 인터페이스를 사용하지 않는 경우

어플리케이션

2

```
Graphic_A g_A = new Graphic_A();  
g_A.brightness_A(80);  
g_A.contrast_A(90.3);  
g_A.display_A();
```

A사 A사 그래픽드라이버 설치

```
class Graphic_A {  
    public void brightness_A(int value) {...}  
    public void contrast_A(double value) {...}  
    public void display_A() {...}  
}
```

3

그래픽 카드를 변경한 경우



어플리케이션

수정필요

4

```
Graphic_B g_B = new Graphic_B();  
g_B.brightness_B(80);  
g_B.contrast_B(90.3);  
g_B.display_B();
```

B사 B사 그래픽드라이버 설치

```
class Graphic_B {  
    public void brightness_B(int value) {...}  
    public void contrast_B(double value) {...}  
    public void display_B() {...}  
}
```

인터페이스(interface)

☞ 인터페이스의 필요성

- 일상생활에서의 인터페이스 vs. 프로그래밍 속에서의 인터페이스

어플리케이션

인터페이스를 사용한 경우

4

```
Graphic g = new Graphic_Impl();  
g.brightness(80);  
g.contrast(90.3);  
g.display();
```

1

```
interface Graphic {  
    void brightness(int value);  
    void contrast(double value);  
    void display();  
}
```



2

A사 A사 그래픽드라이버 설치



```
class Graphic_Impl implements Graphic {  
    public void brightness(int value) {...}  
    public void contrast(double value) {...}  
    public void display() {...}  
}
```

그래픽 카드를 변경한 경우

3

B사 B사 그래픽드라이버 설치

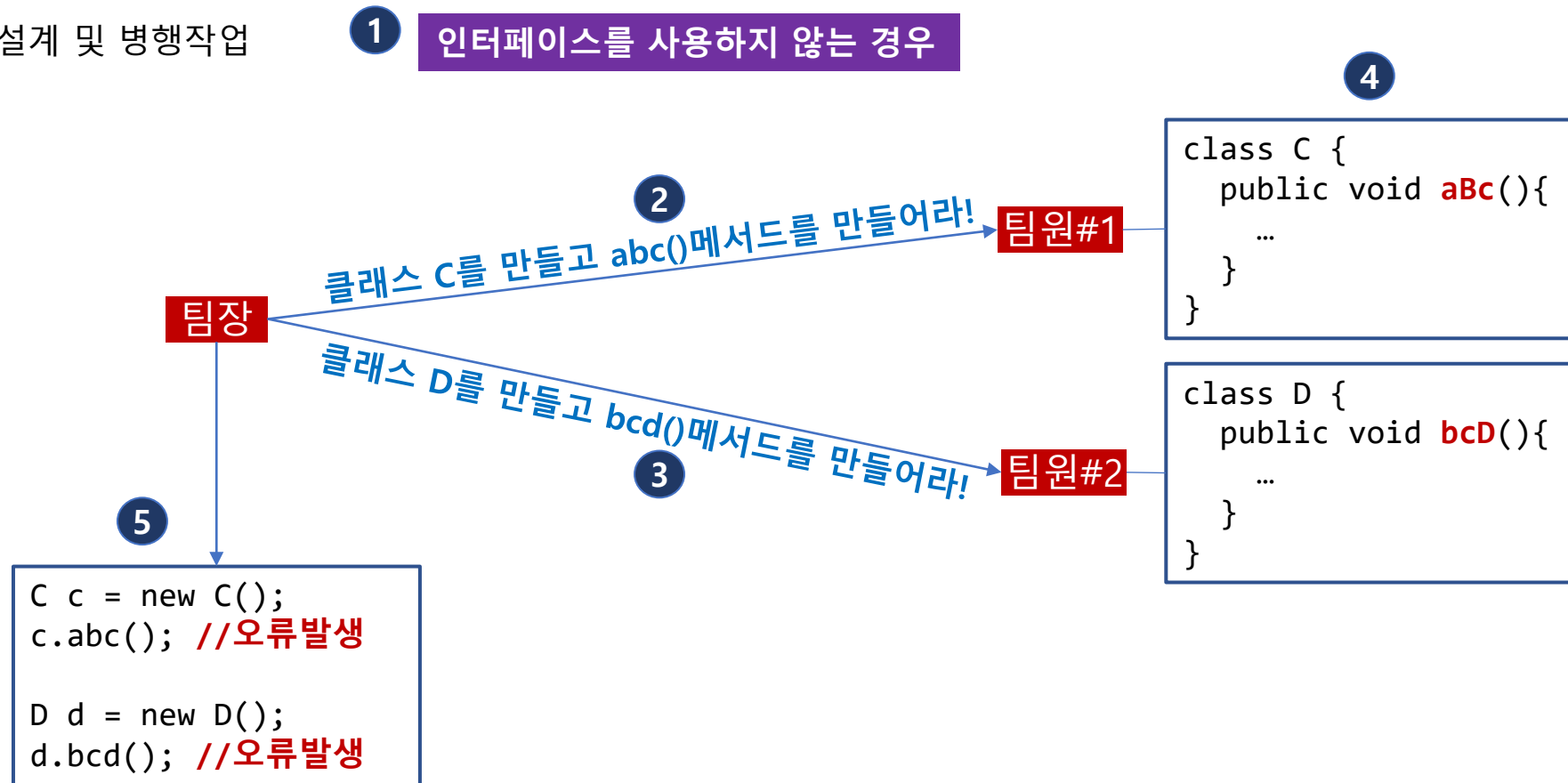


```
class Graphic_Impl implements Graphic{  
    public void brightness(int value) {...}  
    public void contrast(double value) {...}  
    public void display() {...}  
}
```


인터페이스(interface)

☞ 인터페이스의 필요성

- 프로그램 설계 및 병행작업

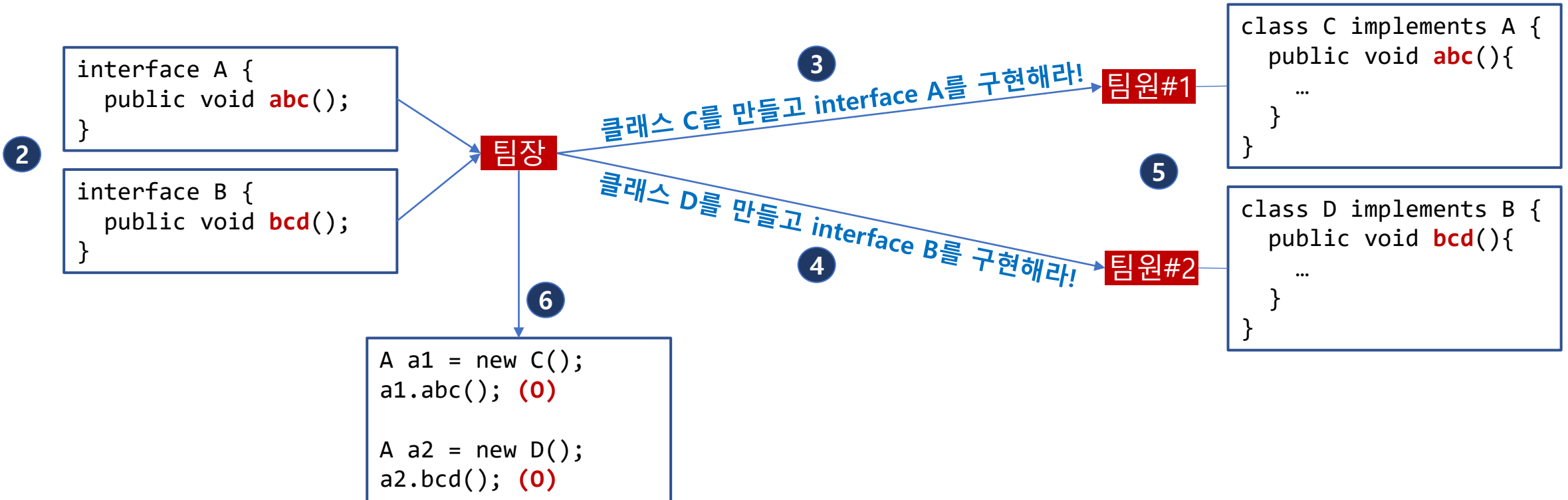


인터페이스(interface)

☞ 인터페이스의 필요성

- 프로그램 설계 및 병행작업

1 인터페이스를 사용하는 경우

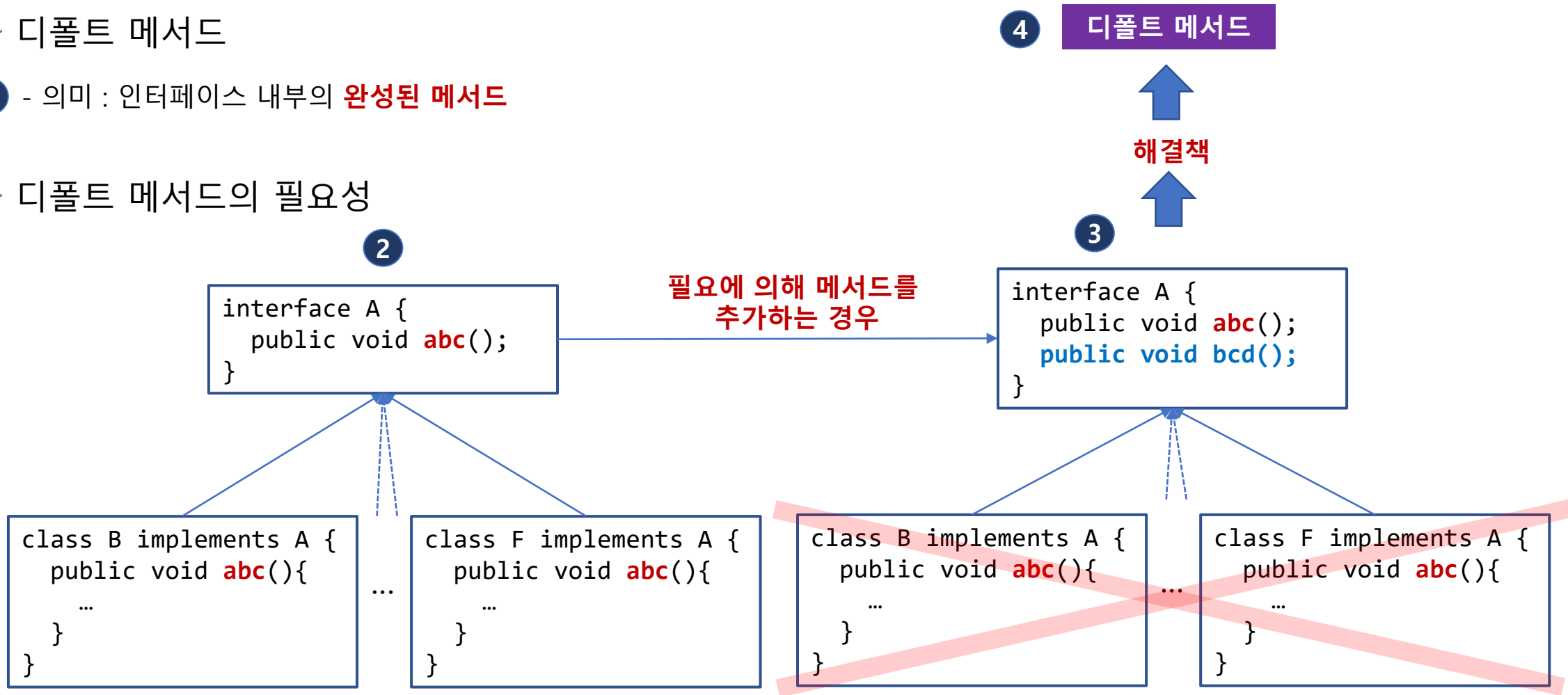


인터페이스(interface)

☞ 디폴트 메서드

① - 의미 : 인터페이스 내부의 **완성된 메서드**

☞ 디폴트 메서드의 필요성



기존에 interface A를 구현한 모든 클래스 오류 발생!!!

인터페이스(interface)

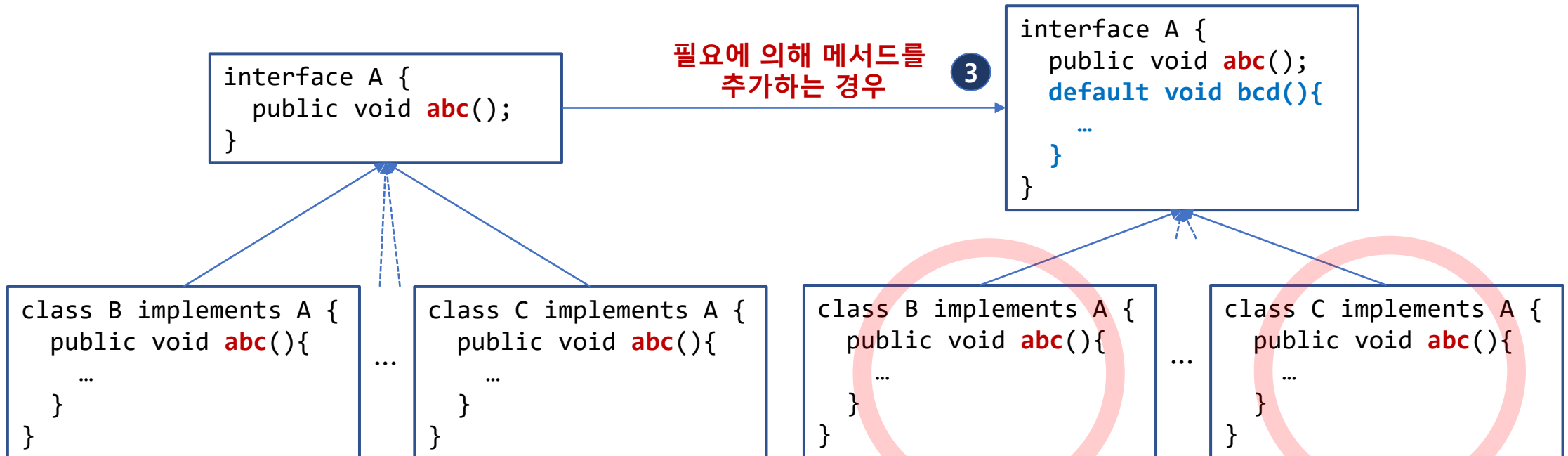
☞ 디폴트 메서드의 작성 방법

- 1 - 메서드 앞에 **default** 키워드 추가
- 오버라이딩도 가능

접근지정자는 자동으로
public 추가됨

```
interface A {  
    public void abc();  
    default void bcd(){  
        ...  
    }  
}
```

완성된 메서드이기 때문에
구현 클래스가 반드시
오버라이딩 할 필요는 없음



인터페이스(interface)

☞ 디폴트 메서드의 작성 방법

1

```
interface A {  
    void abc();  
    default void bcd() {  
        System.out.println("A 인터페이스의 bcd()");  
    }  
}
```

2

```
//#1. 추상메서드만 구현  
class B implements A{  
    @Override  
    public void abc() {  
        System.out.println("B 클래스의 abc()");  
    }  
}
```

3

```
//#2. 추상메서드 구현 + 디폴트 메서드 오버라이딩  
class C implements A {  
    @Override  
    public void abc() {  
        System.out.println("B 클래스의 abc()");  
    }  
    public void bcd() {  
        System.out.println("C 클래스의 bcd()");  
    }  
};
```

4

```
public static void main(String[] ar) {  
    // #1. B 객체 생성 및 메서드 호출  
    B b = new B();  
    b.abc(); // B 클래스의 abc()  
    b.bcd(); // A 인터페이스의 bcd()  
  
    // #2. C 객체 생성 및 메서드 호출  
    C c = new C();  
    c.abc(); // B 클래스의 abc()  
    c.bcd(); // C 클래스의 bcd()  
}
```

5

인터페이스(interface)

☞ 디폴트 메서드의 작성 방법

- 자식클래스에서 부모인터페이스의 디폴트 메서드 호출법

1

부모인터페이스이름.super.디폴트메서드이름

2

```
interface A {  
    default void abc(){  
        System.out.println("A 인터페이스의 abc()");  
    }  
}
```

호출

```
//#1. 자식클래스에서 부모 인터페이스 디폴트 메서드 호출  
class B implements A{  
    @Override  
    public void abc() {  
        A.super.abc();  
        System.out.println("B 클래스의 abc()");  
    }  
}
```

5

```
class C implements A, B{  
    ...  
}
```

=

```
class C extends Object implements A, B {  
    ...  
}
```

super.메서드이름

A.super.메서드이름

B.super.메서드이름

4

그냥 super.abc()를 사용하는 경우
상위클래스인 Object 클래스 내부에서
abc() 메서드를 찾음

3

```
public static void main(String[] ar) {  
    // #1. B 객체 생성 및 메서드 호출  
    B b = new B();  
    b.abc(); // A 인터페이스의 abc() -> B 클래스의 abc()  
}
```

인터페이스(interface)

☞ 정적 메서드의 작성 방법

- 1 - 인터페이스 내에 정적(static) 메서드
: 클래스 내부의 정적메서드와 사용방법 동일 (객체 생성 없이 클래스 이름으로 바로 접근 가능)

TIP

4

- 인터페이스내에 완성된 메서드로는 디폴트 메서드 이외에 정적(static) 메서드도 가능
- 인터페이스의 정적 메서드 특징은 클래스와 동일

2

```
interface A {  
    static void abc(){  
        System.out.println("A 인터페이스의 정적메서드");  
    }  
}
```



객체 생성없이 바로 사용 가능

3

```
public static void main(String[] ar) {  
    //#. 인터페이스 A의 정적메서드 호출  
    A.abc(); //A 인터페이스의 정적메서드  
}
```

The End