13. 디폴트 메소드

13. 디폴트 메소드

개요

인터페이스에 새로운 메소드를 추가하는 등 인터페이스를 변경하고 싶을 때 문제가 발생한다. 인터페이스를 변경하면 이전에 해당 인터페이스를 구현했던 모든 클래스의 구현도 고쳐야 한다.

Java 8에서는 기본 구현을 포함하는 인터페이스를 정의하는 2가지 방법을 제공한다. 인터페이스 내부에 **정적 메소드**를 사용하거나 **디폴트 메소드**를 사용하는 것이다.

결과적으로 기존 인터페이스를 구현하는 클래스는 자동으로 인터페이스에 추가된 새로운 디폴트 메소드를 상속받게 된다.

```
public interface TestInterface<E> {
    default void sort(Comparator<? super E> c) {
        // 구현 내용
    }
}
```

> `default` 키워드는 해당 메소드가 디폴트 메소드임을 나타낸다.

추상 클래스와 인터페이스(default 포함)

그렇다면 결국 인터페이스가 아니라 추상 클래스가 아닐까?

1) 기본 개념

인터페이스는 행동의 규약을 정의한다. 다중 구현이 가능하며 인스턴스 생성이 불가능하다.

추상 클래스는 공통 상태(State)와 행동을 제공한다. 단일 상속만 가능하다(extends는 1개만). 인스턴스 생성이 불가능하지만 생성자는 가질 수 있다.

2) 구현 방식의 차이

인터페이스

- `default` 키워드로 메소드 구현 가능
- `public static final` 상수 선언만 가능
- 생성자가 없다
- 항상 public
- 상태 유지가 불가능하다

추상 클래스

- `abstract` 혹은 일반 메소드 모두 가능하다
- 일반 필드 포함 가능하다
- 생성자를 가질 수 있다
- 모든 접근 제어자 가능하다
- 상태 유지가 가능하다

3) 사용 관점

인터페이스

- 다중 구현이 가능하다
- 공통 필드/상태, 생성자가 없다
- 행동만 공유한다
- 규약 + 선택적 기본 구현이다

추상 클래스

- 다중 구현이 불가능하다
- 공통 필드/상태, 생성자를 가질 수 있다
- 상태 + 행동 패턴 공유가 가능하다
- 공통 로직과 기본 상태를 제공한다

4) 결론

> **인터페이스**는 규약 중심이다.
> **추상 클래스**는 공통 상태와 구현 중심이다.

디폴트 메소드를 이용하면 인터페이스의 **기본 구현을 그대로 상속받는다.** 인터페이스에 자유롭게 새로운 메소드를 추가할 수 있다.

디폴트 메소드는 다중 상속 동작이라는 유연성을 제공한다. 클래스는 여러 디폴트 메소드를 상속받을 수 있다.

13.1 변화하는 API

우리가 도형을 그리는 라이브러리 설계자라고 가정해보자.

1) API 버전 1

```
public interface Resizable extends Drawable {
    int getWidth();
    int getHeight();
    void setWidth(int width);
    void setHeight(int height);
    void setAbsoluteSize(int width, int height);
}
```

이때 라이브러리 사용자가 `Resizable`을 구현하는 `Ellipse`를 만들었다.

```
public class Ellipse implements Resizable {
    // 구현 내용
}
```

그리고 이를 통해 프로그램을 만들었다.

2) API 버전 2

몇 개월이 지나 개선 요청에 따라 `Resizable`을 수정하게 되었다(메소드 추가). 하지만 몇 가지 문제가 발생하게 된다.

- `Resizable`을 구현하는 모든 클래스를 수정해야 한다
- `Resizable`을 구현한 사용자가 직접 수정하지 않는 경우 바이너리 호환성은 유지된다
- 사용자가 구현한 클래스를 포함하는 전체 애플리케이션을 재빌드할 때 컴파일 에러가 발생한다

> **바이너리 호환성**: 새로 추가된 메소드를 호출하지 않으면 새로운 메소드 구현이 없어도 기존 클래스 파일 구현이 잘 동작하는 것을 말한다.

이때 **디폴트 메소드**를 사용하면 새롭게 바뀐 인터페이스에서 자동으로 기본 구현을 제공하므로 기존 코드를 고치지 않아도 된다.

13.1.1 호환성

1) 바이너리 호환성

뭔가 바뀐 이후에도 에러 없이 실행될 수 있는 상황이다.

예를 들어, 인터페이스에 메소드를 추가한 후 추가된 메소드를 호출하지 않으면 문제가 없다.

2) 소스 호환성

코드를 고쳐도 기존 프로그램을 성공적으로 재컴파일할 수 있다.

3) 동작 호환성

코드가 바뀐 다음에도 같은 입력 값이 주어지면 프로그램이 같은 동작을 실행한다.

13.2 디폴트 메소드란?

`default` 키워드로 시작하여 메소드 바디를 포함한다.

```
public interface Sized {
    int size();

    default boolean isEmpty() {
        return size() == 0;
```

```
    }
}
```

모든 클래스는 `isEmpty`의 구현도 상속받으며 소스 호환성도 유지된다.

함수형 인터페이스는 **오직 하나의 추상 메소드**를 포함한다. 디폴트 메소드는 추상 메소드에 해당하지 않는다.

13.2.1 추상 클래스와 Java 8의 인터페이스

1. 클래스는 하나의 추상 클래스만 상속할 수 있지만, 인터페이스는 여러 개 구현할 수 있다.
2. 추상 클래스는 인스턴스 변수(필드)로 공통 상태를 가질 수 있지만, 인터페이스는 인스턴스 변수를 가질 수 없다.

13.3 디폴트 메소드 활용 패턴

디폴트 메소드를 이용하면 라이브러리를 바꿔도 호환성 유지가 가능하다. 우리가 만드는 인터페이스에도 디폴트 메소드를 추가할 수 있다.

1) 선택형 메소드

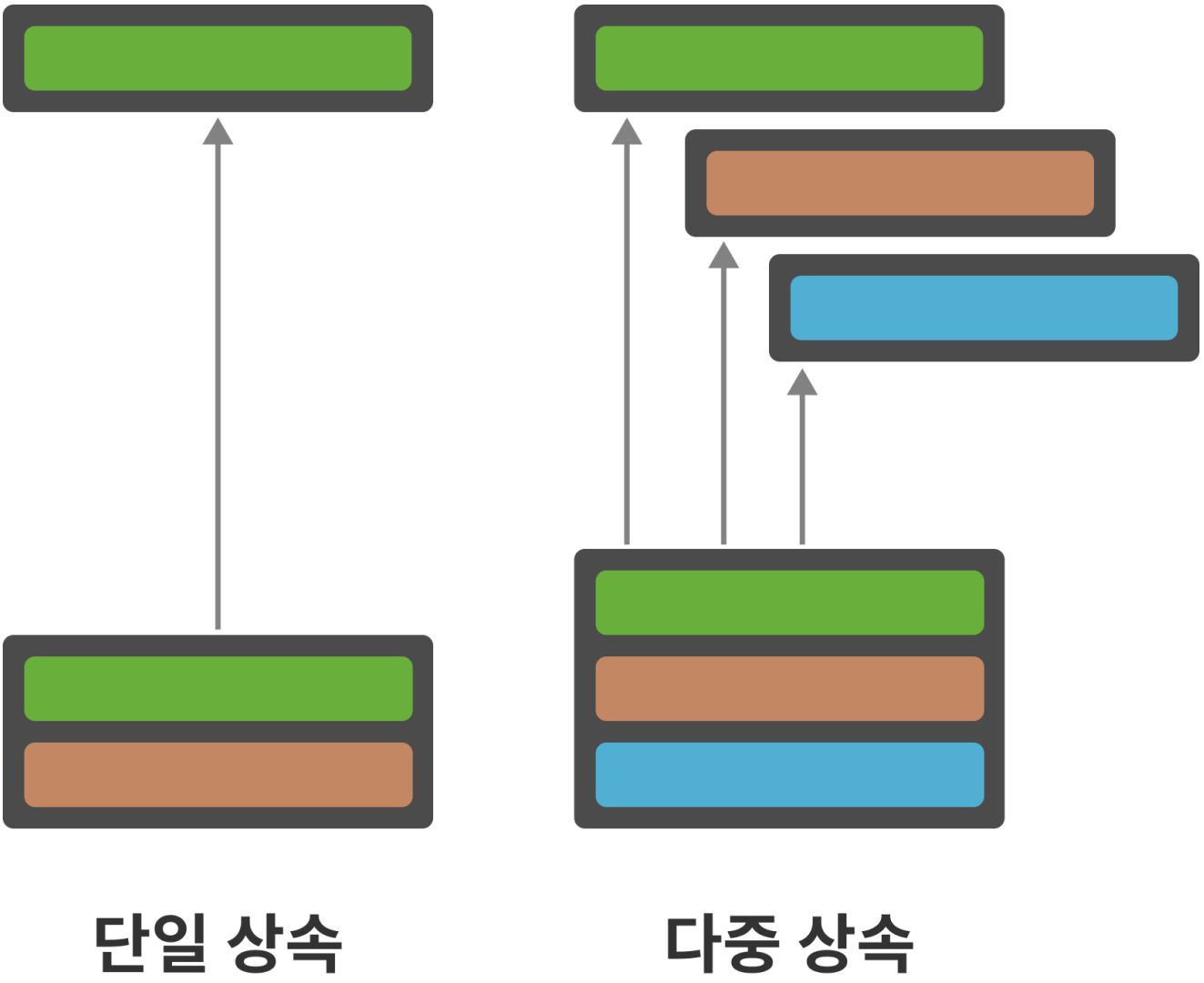
인터페이스를 구현하는 클래스에서 메소드 내용이 비어있는 상황이 있다.

```
interface Iterator<T> {
    boolean hasNext();
    T next();

    default void remove() {
        throw new UnsupportedOperationException();
    }
}
```

디폴트 메소드를 사용하면 구현 클래스에서 빈 구현을 제공할 필요가 없다.

2) 동작 다중 상속



기존에 불가능했던 다중 상속 기능도 구현이 가능하다.

3) 다중 상속 형식

`ArrayList` 는 한 개의 클래스를 상속받고, 여러 개의 인터페이스를 구현한다. 결과적으로 `ArrayList` 는 이들의 서브 형식이 된다.

디폴트 메소드를 사용하지 않아도 다중 상속을 활용할 수 있다.

13.4 해석 규칙

Java 8에는 디폴트 메소드가 추가되었으므로 같은 시그니처를 갖는 디폴트 메소드를 상속받는 상황이 생길 수 있다.

13.4.1 다중 상속의 모호성: 디폴트 메소드가 겹치는 경우

A, B, C라는 인터페이스들이 `hello()` 라는 디폴트 메소드를 구현한다. `MyClass` 가 A, B, C를 모두 구현한 클래스라면, `hello()` 라는 디폴트 메소드들이 겹치는 상황이 발생한다.

첫 번째: 컴파일 에러가 발생한다

A, B, C에서 `hello()` 라는 default 메소드를 물려받아 무엇을 사용할지 모른다. 이는 자바가 명시적 선택을 강제하는 설계 철학이기 때문이다. 명시하지 않으면 모호성 때문에 위험하다.

두 번째: 명시적으로 오버라이딩한다

```
@Override
public void hello() {
    A.super.hello(); // 명시적 선택
}
```

특정 인터페이스의 default 구현을 명시한다.

세 번째: 구현 클래스에서 직접 오버라이딩한다

```
@Override
public void hello() {
    System.out.println("My Class");
}
```

`MyClass` 가 자신의 `hello()` 를 제공했기 때문에 더 이상 A, B, C의 default 메소드를 상속받지 않는다.

디폴트 메소드 우선순위 규칙

1. **클래스가 항상 이긴다.** 클래스나 슈퍼 클래스에서 정의한 메소드가 디폴트 메소드보다 우선권을 갖는다.
2. 1번 규칙 이외의 상황에서는 **서브 인터페이스가 이긴다.** 상속 관계를 갖는 인터페이스에서 같은 시그니처를 갖는 메소드를 정의할 때는 서브 인터페이스가 이긴다.
3. 디폴트 메소드의 우선순위가 결정되지 않았다면 여러 인터페이스를 상속받는 클래스가 **명시적으로 디폴트 메소드를 오버라이드하고 호출**해야 한다.

13.4.2 예방 및 대처법

1) default 메소드는 기본 동작만 제공

- 도우미 성격의 코드만 작성한다
- 비즈니스 로직을 포함하지 않으며 공통 유틸 수준으로만 작성한다

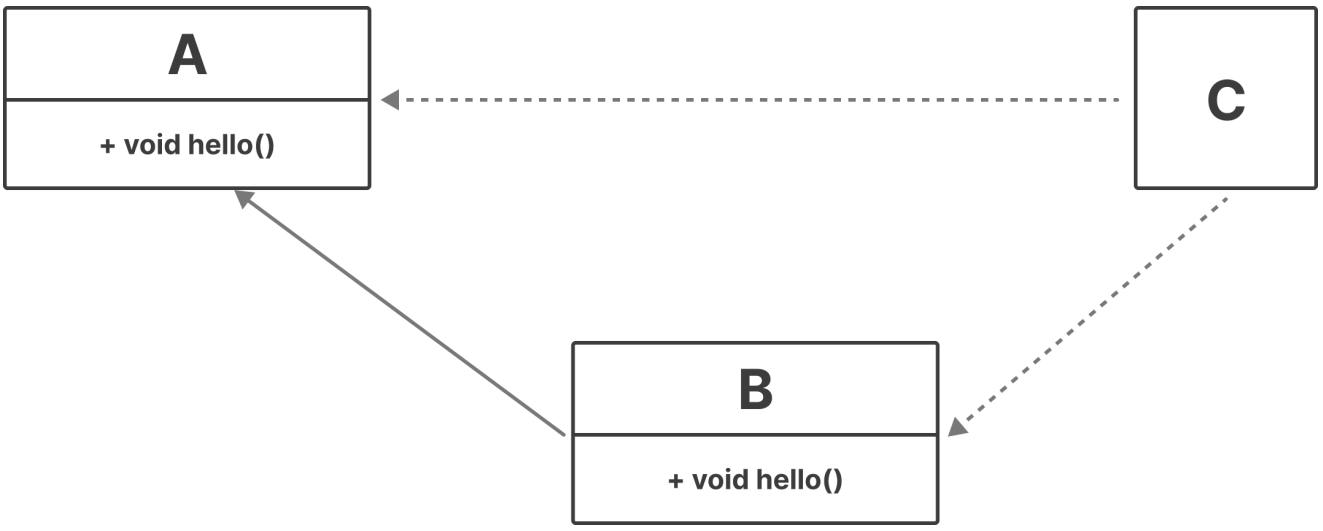
2) 다중 default 메소드를 지양한다

- 여러 인터페이스에 같은 기능 및 이름을 작성하지 않으며 역할로 분리한다
- 의도적으로 다른 시그니처를 사용한다

3) 중간 추상 클래스를 도입한다

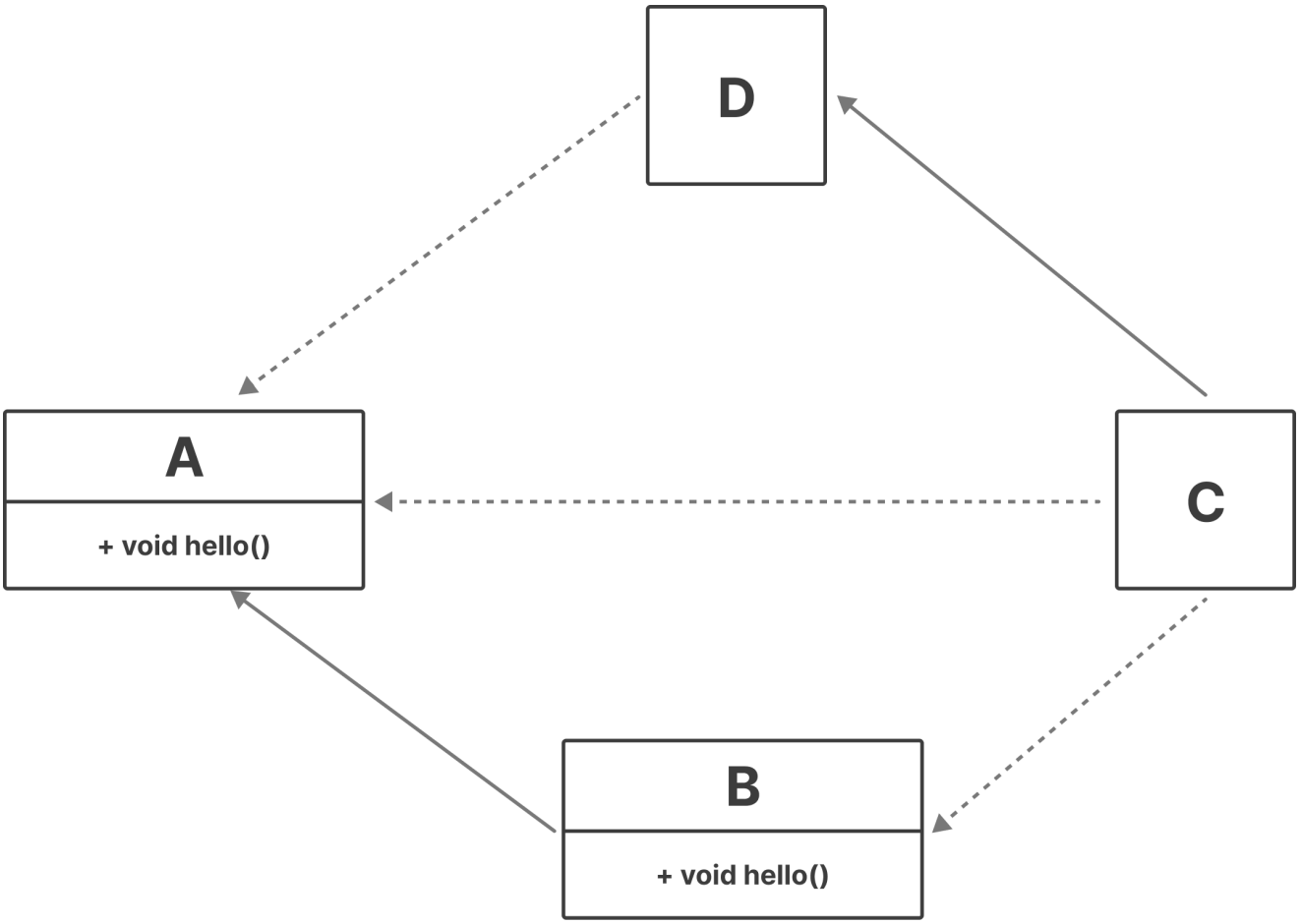
중간 계층에서 명시적 선택을 처리한다.

13.4.3 충돌 해결 예제



- B와 A는 `hello()` 라는 default method를 정의한다.
- B는 A를 상속받는다.

> 서브 인터페이스가 이기므로 B의 `hello()` 가 선택된다.



> `new C().hello()` 는 무엇이 출력될까?

D는 A의 디폴트 메소드 구현을 상속한다(오버라이드하지 않고 구현만 한다). 클래스나 슈퍼 클래스에 메소드 정의가 없을 때 디폴트 메소드를 정의하는 서브 인터페이스가 선택되므로 B의 `hello()` 가 선택된다.

정리

- 디폴트 메소드는 인터페이스에 기본 구현을 제공하여 API의 호환성을 유지한다.
- 함수형 인터페이스는 하나의 추상 메소드만 포함하며, 디폴트 메소드는 개수에 포함되지 않는다.
- 디폴트 메소드가 충돌할 때는 명시적으로 선택하거나 오버라이드해야 한다.
- 클래스 메소드 > 서브 인터페이스 메소드 > 명시적 선택 순으로 우선순위가 결정된다.