JAVA 프로그래밍

Chapter 09 상속과 인터페이스

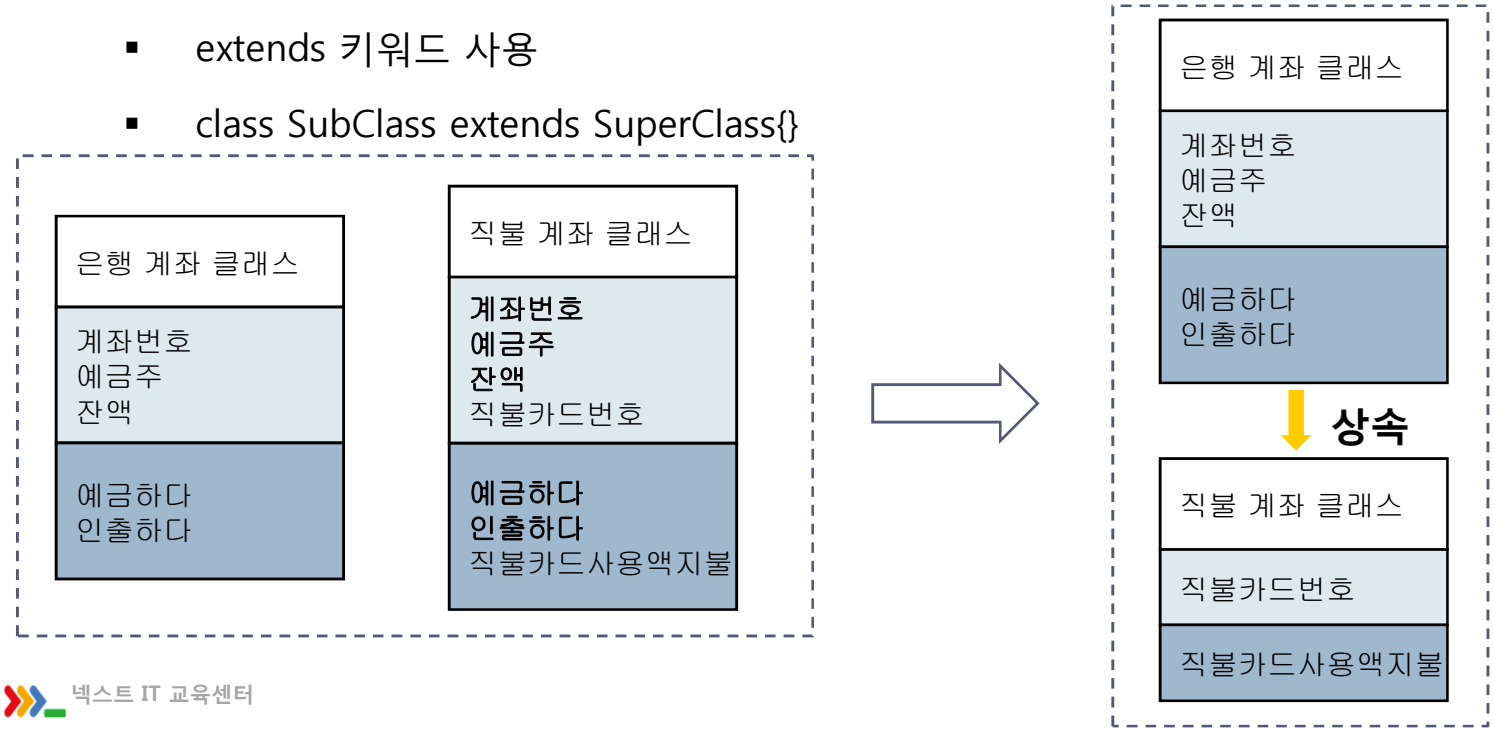
목차

- 클래스의 상속
- 추상 클래스
- 인터페이스

상속

❖ 상속이란?

- 이미 존재하는 클래스를 확장해서 새로운 클래스를 만드는 기술
- 기존 클래스가 가지고 있는 데이터 구조와 기능을 그대로 물려받아서 사용하는 기술.
- 기존에 작성된 클래스를 재활용할 수 있다.
- 클래스 간의 계층적 관계를 구성함으로써 다형성의 문법적 토대를 마련.
- extends 키워드 사용
- class SubClass extends SuperClass{}



상속

- 상속과 생성자
  - 생성자를 통한 슈퍼 클래스 필드 초기화 방법
  - 생성자가 있는 슈퍼 클래스 상속하는 방법
- super
  - 현재 객체의 바로 상위인 super클래스(부모클래스)를 참조할 수 있는 미리 정의된 상위 클래스의 객체변수.
  - 부모 클래스로부터 상속 받은 필드나 메서드를 자식 클래스에서 참조하는데 사용하는 참조 변수.
- super()
  - super클래스의 생성자를 의미한다.
  - 부모 클래스의 멤버를 초기화하기 위해서는 자식 클래스의 생성자에서 부모 클래스의 생성자까지 호출해야만 한다.
  - 자바 컴파일러는 부모 클래스의 생성자를 명시적으로 호출하지 않는 모든 자식 클래스의 생성자 첫 줄에 자동으로 super(); 명령문을 추가하여 부모 클래스의 멤버를 초기화할 수 있도록 한다.

상속 - 예제

부모클래스(상위클래스)

```
class Account {
    String accountNo;
    String ownerName;
    int balance;
    void deposit(int amount) {
        balance += amount;
    }
    int withdraw(int amount){
        if (balance < amount){
            return 0;
        }
        balance -= amount;
        return amount;
    }
}
```

상속

자식클래스(하위클래스)

```
class CheckingAccount  extends Account  {

  String cardNo;      // 카드번호

  int pay(String cardNo, int amount){

    // 부모로 부터 상속받은 필드
    if (!cardNo.equals(this.cardNo) || (balance < amount)){
      System.out.println("지불이 불가능합니다.");
      return 0;
    }
    // 부모로 부터 상속받은 메소드
    return withdraw(amount);
  }
}
```

상속

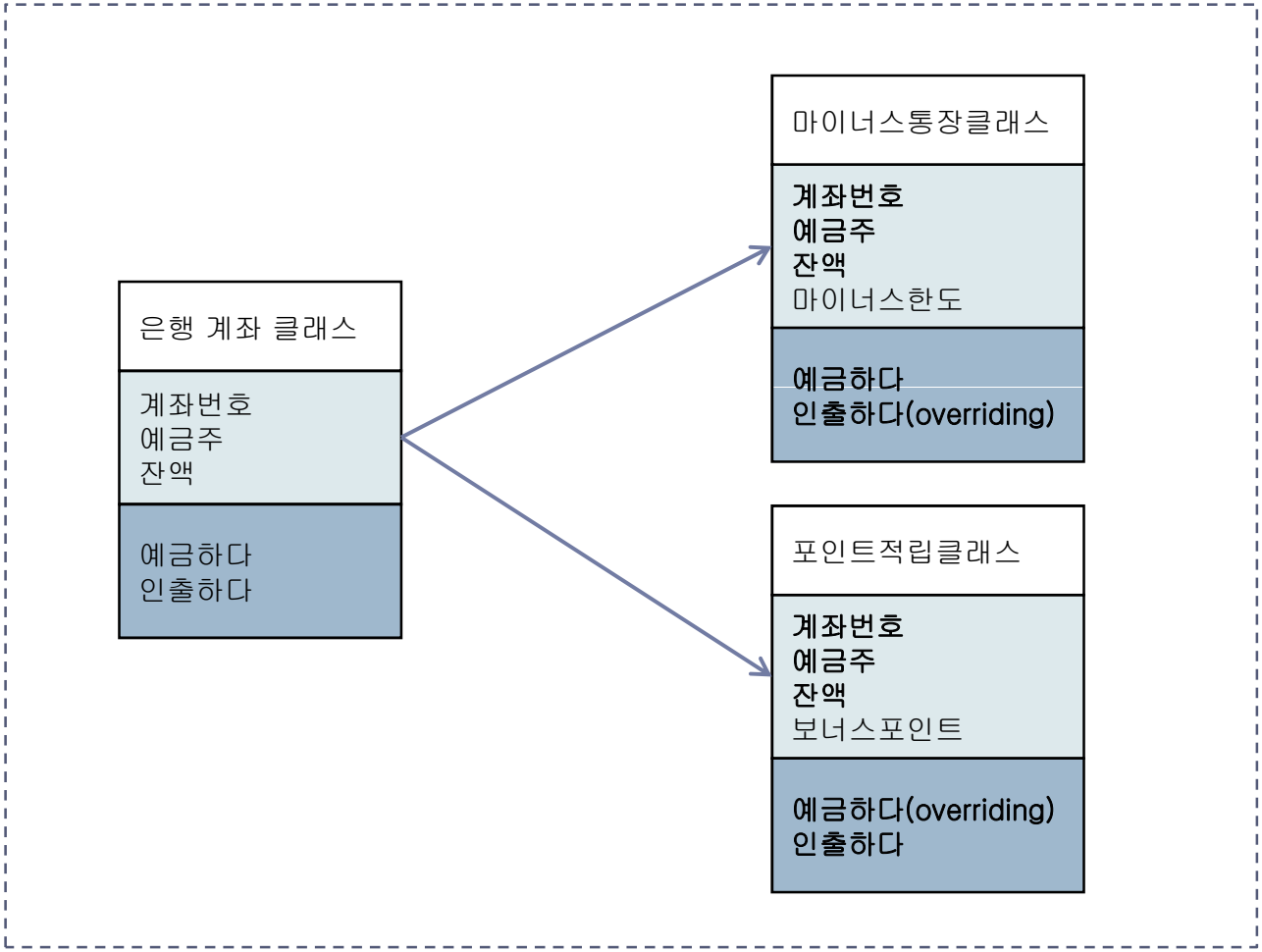
❖ 메소드 오버라이딩(재정의)

- 상위(부모)클래스의 메소드와 똑같은 메소드를 선언해서 상위클래스의 메소드를 무시하도록 만드는 것.
- 하위(자식) 클래스에서 메소드를 재정의하는 것
- 상위(부모) 클래스의 메소드를 무시하고 자식 클래스의 메소드를 실행한다.

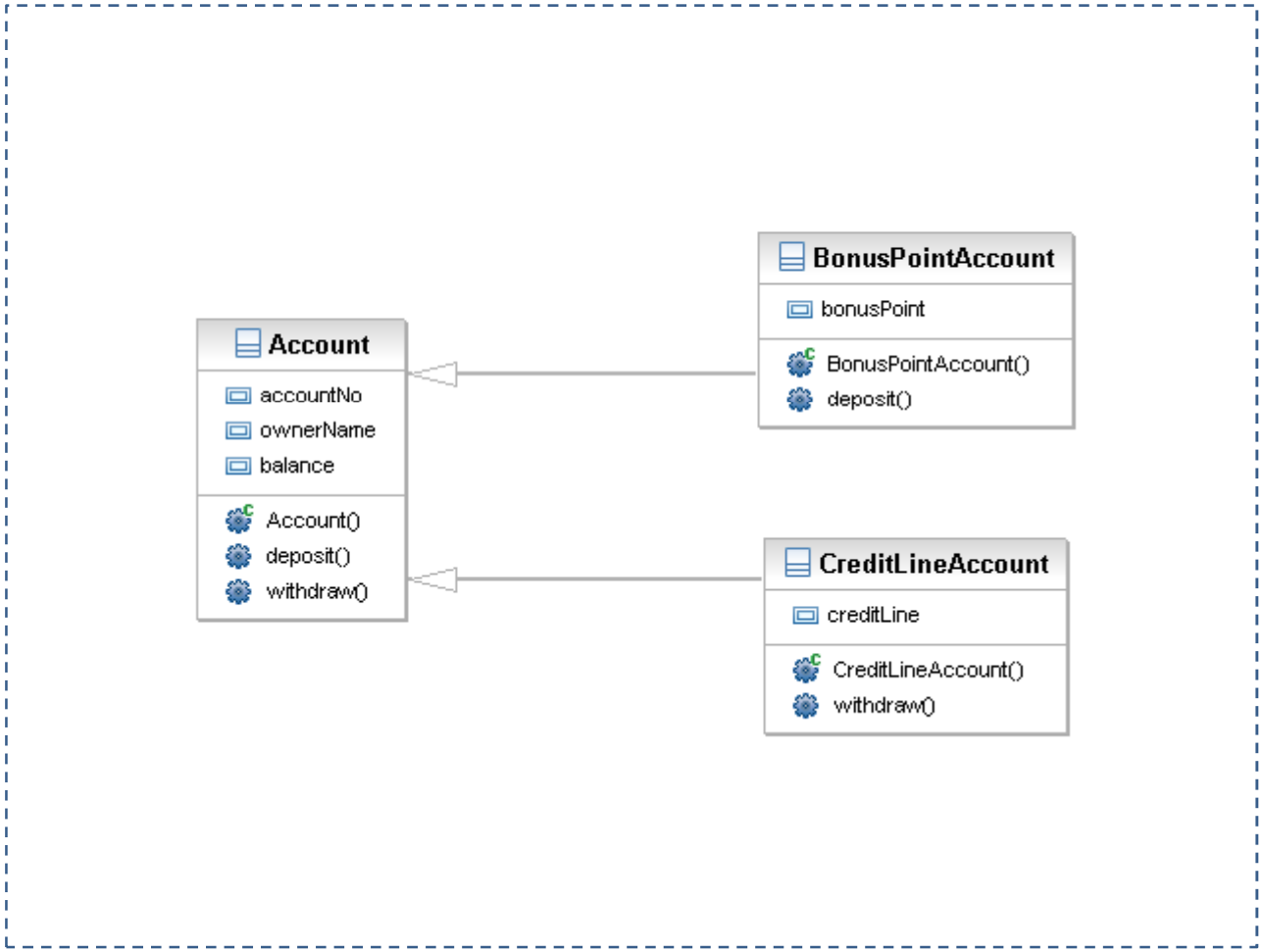
❖ 상속 금지 final 키워드

- final 키워드를 이용한 클래스의 상속금지
- final 키워드를 이용한 메소드 오버라이딩 금지

상속 - Overriding



상속 - Overriding



상속 – Overriding 예제

```
class Account {
    String accountNo;
    String ownerName;
    int balance;
    void deposit(int amount) {
        balance += amount;
    }
    int withdraw(int amount){
        if (balance < amount){
            return 0;
        }
        balance -= amount;
        return amount;
    }
}
```

```
class CreditLineAccount extends Account {
    int creditLine;   // 마이너스 한도 필드

    // 상위 클래스 메소드 오버라이딩
    int withdraw(int amount){
        if ((balance + creditLine) < amount){
            System.out.println("인출이 불가능합니
다.");
                return 0;
        }
        balance -= amount;
        return amount;
    }
}
```

```
class BonusPointAccount extends Account {
    int bonusPoint;

    // 상위 클래스 메소드 오버라이딩
    void deposit(int amount) {
        balance += amount;
        bonusPoint += (int) (amount * 0.001);
    }
}
```

다형성 (Polymorphism)

❖ 변수의 다형성(Polymorphism)

- 하나의 변수에 여러 종류의 데이터를 대입할 수 있는 성질을 변수의 다형성이라 한다.
- 클래스 변수의 다형성은 여러 종류의 객체들을 똑같은 로직으로 처리하는 프로그램을 작성할 수 있다.
- Upcasting
- Downcasting

❖ 메소드의 다형성

- 메소드 오버라이딩(overriding : 재정의)

객체간 형 변환(Casting)

- ❖ 객체들의 형 변환
  - 서로 상속관계에 있는 객체들 사이에만 타입 변환을 할 수 있다. (일반적인 상속, 추상클래스, 인터페이스)
  - 자식 클래스 타입에서 부모 클래스 타입으로의 타입 변환은 생략할 수 있다.
  - 부모 클래스 타입에서 자식 클래스 타입으로의 타입 변환은 반드시 명시해야 한다.
  - Upcasting과 Downcasting으로 구분
- ❖ 업캐스팅(Upcasting)
  - 특정 객체가 하위 클래스의 형에서 상위 클래스의 형으로 캐스팅되는 것.
  - 상위클래스 변수 = 하위클래스객체
  - 형만 정확하다면 묵시적으로 가능(자동 캐스팅)
  - 상위클래스에 선언된 메서드만 사용 가능
  - 하위클래스에서 재정의된 메서드가 호출
  - 다형성(polymorphism)을 극대화한다.

객체간 형 변환(Casting)

- Downcasting
  - 하위클래스 변수 = (하위클래스)상위클래스객체
  - 묵시적으로 불가능 하며, Casting 연산자인 ()를 사용해서 형을 명시해야 함
  - 반드시 Upcasting이 선행해야 함
  - 원래의 형으로 복귀

추상 클래스

❖ 추상 메소드(abstract method)

- 프로토타입(prototype)만을 가진 메소드, 즉 몸체가 없는 메소드.
- 반드시 abstract 키워드를 명시해야 한다.

❖ 추상 클래스(abstract class)

- 완전하지 못한 미완성 클래스.
- 추상메소드를 단 하나라도 포함하면 추상클래스가 됨.
- 임의로 abstract를 명시하여 추상클래스로 만들 수도 있다.
- 추상클래스는 반드시 abstract 를 명시해야 함.
- 자체적으로 객체를 생성하지 못함.
- 반드시 상속을 통해 하위 클래스에서 추상 메소드를 재정의(overriding)해야 객체 생성이 가능함.

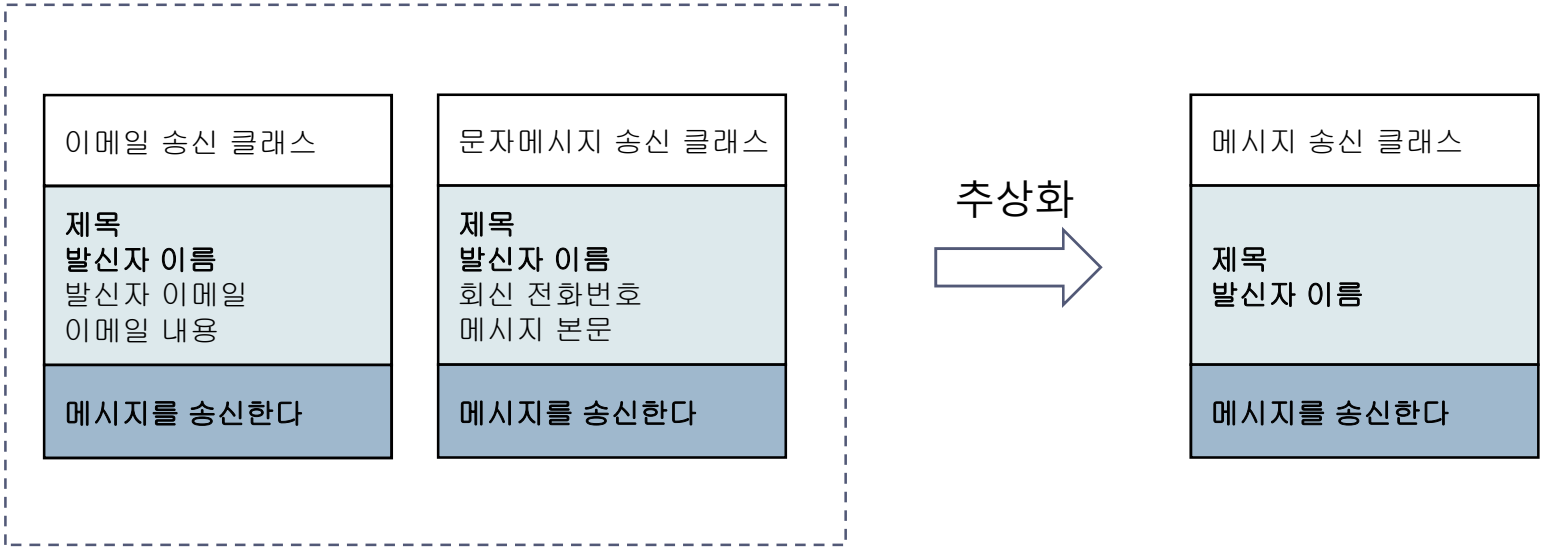
```
public abstract class EmptyCan {
    public abstract void printContent();  //추상 메서드
    public abstract void printName();     //추상 메서드

    public void printMessage(){           // 일반 메소드
        System.out.println("안녕하세요....");
    }
}
```

추상 클래스

❖ 추상클래스 만들기 예제

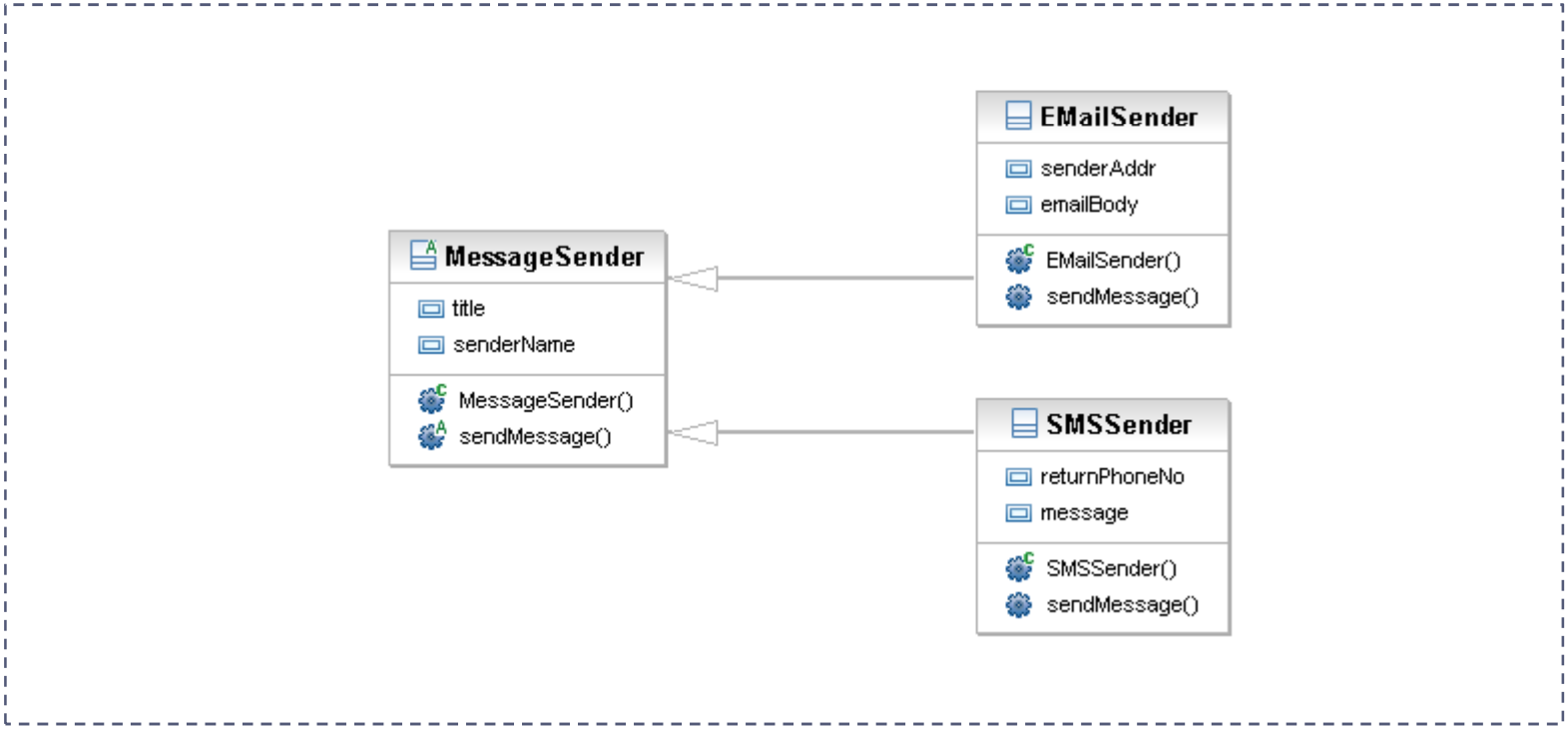
- 데이터의 구조와 기능에 공통점이 많은 클래스들이 존재할 때 이들 공통점을 뽑아서 슈퍼 클래스를 만들어 두는 것이 좋다.



추상 클래스 - 다이어그램

❖ 추상클래스 만들기 예제

- 데이터의 구조와 기능에 공통점이 많은 클래스들이 존재할 때 이들 공통점을 뽑아서 슈퍼 클래스를 만들어 두는 것이 좋다.



추상 클래스 예제 - 소스

```
// 추상클래스
abstract  class MessageSender {
    String title;                  // 제목
    String senderName;      // 발신자 이름

    // 생성자
    MessageSender(String title, String
senderName) {...}

    // 추상메소드
    abstract void sendMessage(String recipient);
}
```

```
class EMailSender extends MessageSender {
    String senderAddr;     // 발신자 이메일
    String emailBody;       // 이메일내용

    EMailSender(String title, String senderName,
                String senderAddr, String emailBody)
{...}
    // 오버라이드
    void sendMessage(String recipient) {...}
}
```

```
class SMSSender extends MessageSender {
    String returnPhoneNo;     // 회신 전화번호
    String message;                 // 메시지 본문

    SMSSender(String title, String senderName,
              String returnPhoneNo, String message)
{...}
    // 오버라이드
    void sendMessage(String recipient) {...}
}
```

추상 클래스

❖ 추상 클래스를 사용하는 이유

- 클래스의 구조를 디자인하기 위함.
- 작업을 수평적으로 분할.
- 작업을 단계별 레벨 단위의 구조로 만든다.
- 추상 메서드를 사용하는 목적은 추상 메서드가 포함된 클래스를 상속받는 자식 클래스가 반드시 추상 메서드를 구현하도록 하기 위함이다.

인터페이스

❖ 인터페이스 정의

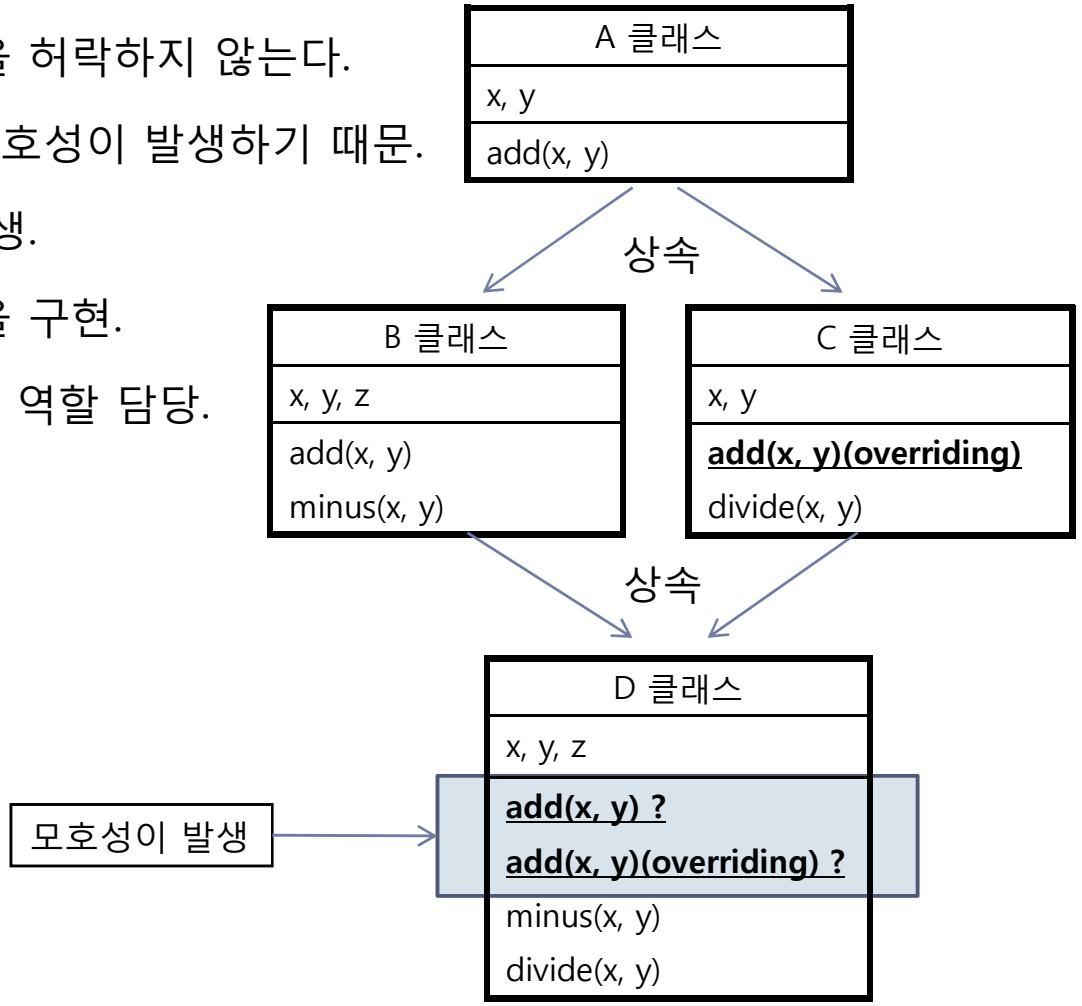
- 클래스들의 공통점을 추출해서 만든 클래스의 한 형태.
- 인터페이스는 클래스의 공통 기능만 표현 할 수 있고, 공통 데이터는 표현 할 수 없다는 제약을 갖는다.
- 인터페이스에 속하는 메소드는 추상 메소드로 선언(추상클래스의 일종).
- 메소드는 묵시적으로 public abstract가 된다.
- interface 키워드를 사용해서 정의한다.
- implements 키워드를 사용해서 구현한다.
- public class Sample implements Cloneable, Serializable{ }
- 클래스의 다중 상속을 가능케한다.
- 공동 작업을 위한 상호 인터페이스로 사용된다.

인터페이스

- 인터페이스 변수의 다형성
- 인터페이스 정적 상수 필드
  - 구현하는 클래스에서 자주 사용하는 상수를 선언.
  - 정적 상수 필드는 객체의 생성과 무관하기 때문에 사용가능.
  - 필드에는 묵시적으로 public final static 이 붙는다.
  - public static final int MAXIMUM = 100;
  - public static final은 생략 가능.
- 인터페이스의 상속 및 다중 상속
  - 인터페이스간 상속 및 다중 상속이 가능하다.
  - 이때 extends 라는 키워드를 사용한다.

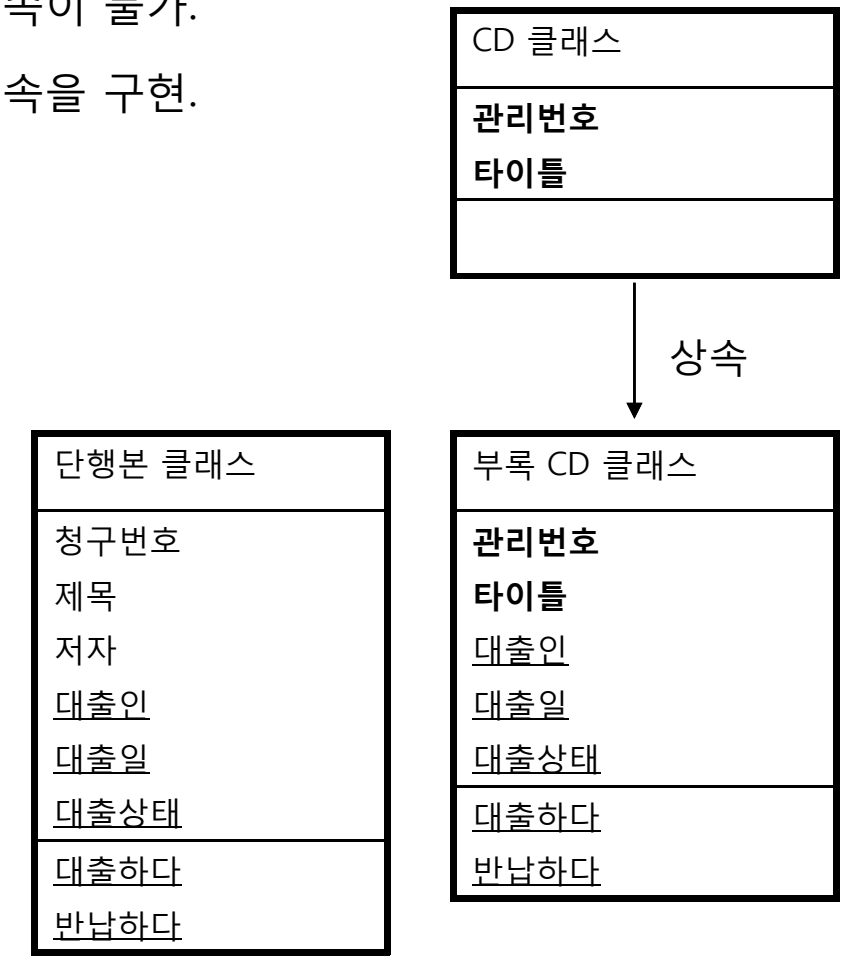
인터페이스 (다중상속 문제점)

- ❖ 자바는 기본적으로 다중상속을 허락하지 않는다.
- ❖ 다중 상속시 메서드 출처의 모호성이 발생하기 때문.
- ❖ 또한 캐스팅시에 문제점이 발생.
- ❖ 인터페이스를 통해 다중상속을 구현.
- ❖ 다른 클래스 사이의 중간 매개 역할 담당.



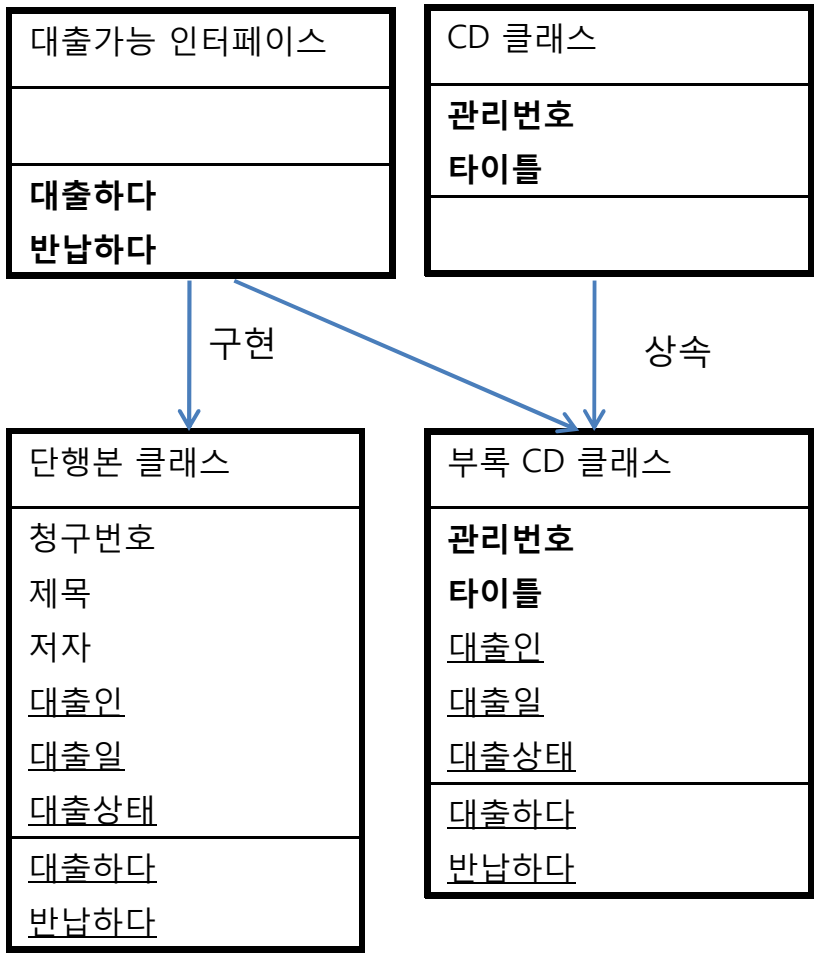
인터페이스 사례1

- ❖ 자바는 기본적으로 다중상속이 불가.
- ❖ 인터페이스를 통해 다중상속을 구현.



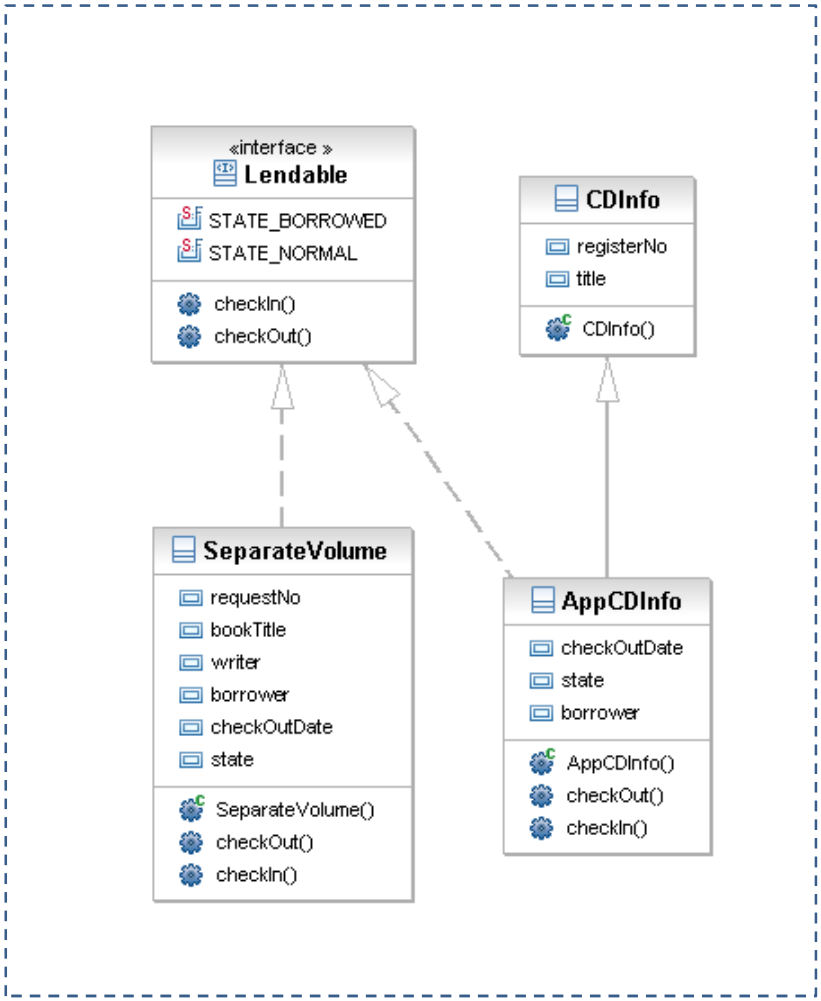
인터페이스 사례2

- ❖ 자바는 기본적으로 다중상속이 불가.
- ❖ 인터페이스를 통해 다중상속을 구현.
- ❖ 공통기능을 추출해서 인터페이스를 정의
- ❖ 메소드의 집합이므로 기능적 측면만을 담당



인터페이스 사례2

- 자바는 기본적으로 다중상속이 불가.
- 인터페이스를 통해 다중상속을 구현.
- 공통기능을 추출해서 인터페이스를 정의
- 메소드의 집합이므로 기능적 측면만을 담당



인터페이스 사례2 - 소스

```
// 시디 클래스
class CDInfo {
    String registerNo;   // 등록번호
    String title;        // 타이틀
    CDInfo(String registerNo, String title) {
        this.registerNo = registerNo;
        this.title = title;
    }
}
```

상속

```
// 부록시디 클래스
class AppCDInfo  extends CDInfo  implements
Lendable  {
    String borrower;          // 대출인
    String checkOutDate;      // 대출일
    byte state;               // 대출상태
    AppCDInfo(String registerNo, String title) {…}
    public void checkOut(String borrower, String
date) {…}
    public void checkIn() {…}
}
```

```
// 대출가능 인터페이스
interface Lendable {
    // 인터페이스에서 상수 필드 사용예
    final static byte STATE_BORROWED = 1;  // 대출
중
    final static byte STATE_NORMAL = 0;       // 대
출가능

    void checkOut(String borrower, String date);
    void checkIn();
}
```

구현

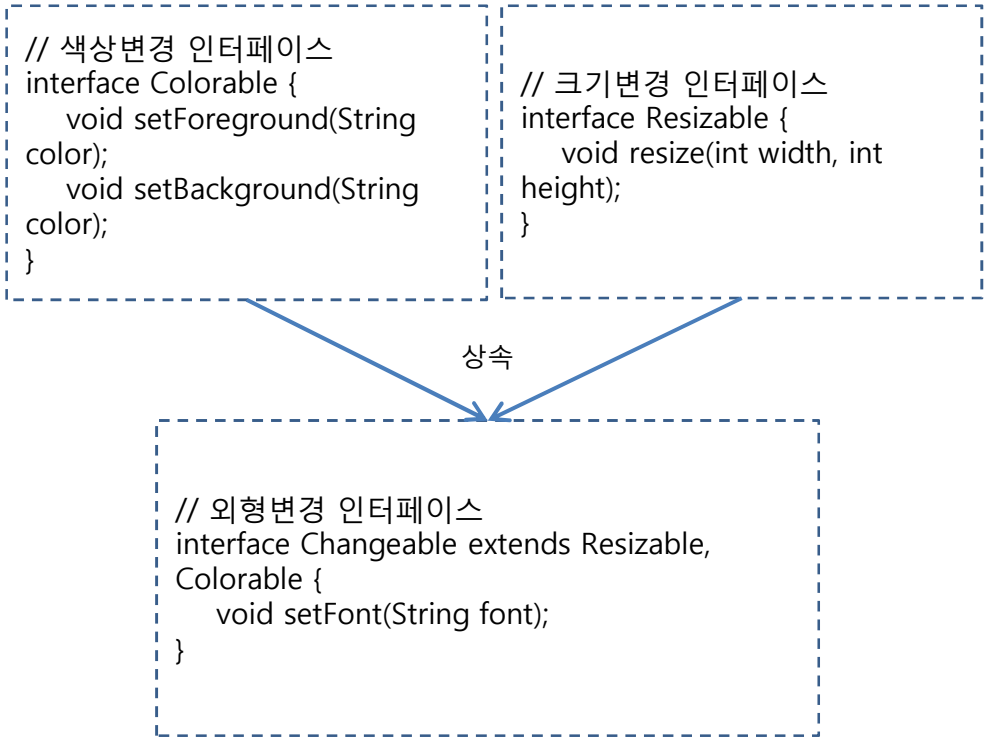
```
// 단행본 클래스
class SeparateVolume  implements Lendable  {
    String requestNo;       // 청구번호
    String bookTitle;       // 제목
    String writer;          // 저자
    String borrower;        // 대출인
    String checkOutDate;    // 대출일
    byte state;             // 대출상태

    public void checkOut(String borrower, String
date) {…}
    public void checkIn() {…}
}
```

인터페이스 다중상속

❖ 인터페이스의 상속 및 다중 상속

- 인터페이스간 상속 및 다중 상속이 가능하다.
- 이때 extends 라는 키워드를 사용한다.



추상클래스 VS 인터페이스

	추상클래스	인터페이스
공통점	모두 클래스임. 하위 클래스에서 모든 추상 메서드를 구현 해야함.	
차이점	• 추상 메서드 외 일반 멤버 변수와 메서드를 가질 수 있다. • extends를 사용. • 단일 상속만 가능. • 작업의 레벨 분할을 위해서 사용.	• 추상 메서드와 static final 변수만 사용. • Implements 키워드를 사용하여 구현. • 중복 구현 가능. • 공동 작업을 위한 상호간의 인터페이스를 위해 사용.