

Section 02


추상 클래스

2. 추상 클래스

■ 추상 클래스의 선언

- 추상 클래스는 `abstract` 키워드를 사용하여 선언
- 클래스 내에 일반 메서드뿐 만 아니라 추상 메서드를 포함할 수 있음
- 본문이 없는 메서드인 추상 메서드는 `abstract` 키워드를 사용하여 선언

```
abstract class 클래스명 {  
    반환유형 메서드명([매개변수목록]);  
    abstract 반환유형 메서드명([매개변수목록]);  
}
```

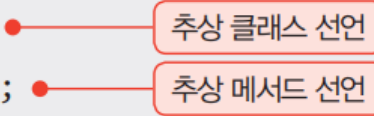


2. 추상 클래스

■ 추상 클래스의 선언

- 추상 메서드는 자식 클래스에서 구현됨
 - 이는 부모 클래스가 메서드명만 가지고 있고 자식 클래스가 해당 메서드명을 사용하여 필요에 따라 본문을 정의한다는 것을 의미
- 클래스에 추상 메서드가 포함되어 있으면 반드시 추상 클래스로 선언해야 함

```
abstract class Animal {  
    abstract printSound();  
}
```



2. 추상 클래스

■ 추상 클래스의 선언

추상 클래스 생성 예시

Animal.java

```
public abstract class Animal {  
    public abstract void printSound();  
  
    public void displayInfo() {  
        System.out.println("나는 동물입니다");  
    }  
}
```

Example01.java

```
public class Example01 {  
    public static void main(String[] args) {  
        Animal myObj = new Animal();  
    }  
}
```

실행 결과

오류 발생

2. 추상 클래스

■ 추상 클래스 사용 시 주의 사항

- 추상 클래스는 항상 `abstract` 키워드를 사용하여 선언해야 함
- 추상 클래스의 모든 메서드를 추상으로 사용할 필요는 없음
 - 추상 클래스에 일반적인 메서드가 포함될 수도 있음
- 클래스의 메서드 중 하나가 추상 메서드이면 해당 클래스는 추상 클래스여야 함
- 추상 메서드를 선언하면 자식 클래스에서 해당 메서드를 재정의해야 함
 - 추상 클래스를 상속하는 경우 추상 메서드 재정의는 필수
 - 상속받은 클래스와 자식 클래스가 추상 클래스라면 메서드를 재정의하지 않아도 됨

2. 추상 클래스

■ 추상 클래스 사용 시 주의 사항

- 추상 클래스는 자신의 인스턴스를 가질 수 없음
 - 추상적인 클래스의 객체를 가질 수 없으나 추상 클래스 변수로 추상 클래스를 상속받은 클래스로 생성한 객체를 참조할 수는 있음
- 프로그램이 자식 클래스의 객체를 만들 때 컴파일러가 추상 클래스의 생성자를 호출함
- 추상 클래스의 내부에 일반 메서드만 존재한다고 해도 이 추상 클래스로 객체를 생성할 수 없음
- 추상 클래스에 `final()` 메서드를 포함할 수도 있음
 - 하지만 `final` 클래스는 추상 메서드를 가질 수 없음
- `final` 키워드가 붙으면, 메서드의 경우 `Overriding`을 할 수 없고 클래스의 경우는 상속을 할 수 없음

2. 추상 클래스

■ 추상 클래스의 상속

- 추상 클래스는 그 자체의 인스턴스를 만들 수 없기 때문에 상속해서 사용하며, 보통의 클래스처럼 extends 키워드를 이용하여 상속함
- 일반적으로 자식 클래스는 이러한 추상 클래스를 상속받고 추상 메서드를 재정의하여 사용함

```
abstract class Animal {
```

```
// 코드
```

```
}
```

```
class Cat extends Animal {
```

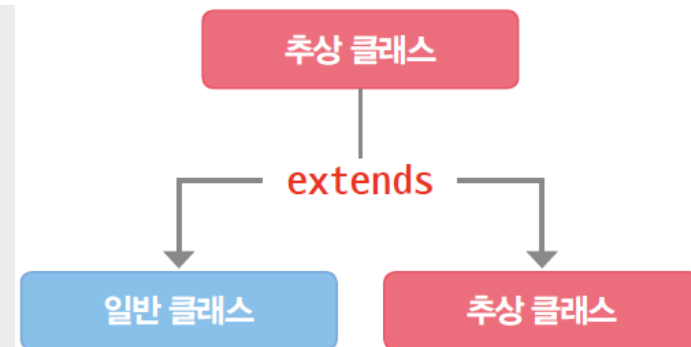
```
// 코드
```

```
}
```

```
abstract class Mammal extends Animal {
```

```
// 코드
```

```
}
```

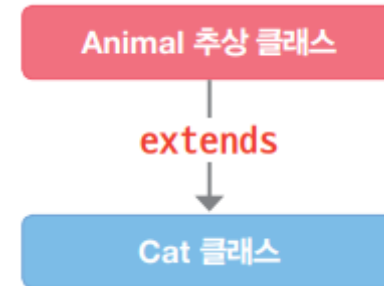


[그림 9-3] 추상 클래스의 상속

2. 추상 클래스

■ 추상 클래스의 상속

- 추상 클래스를 상속받는 일반 클래스



[그림 9-4] 추상 클래스(Animal)를 상속받는 일반 클래스(Cat)

추상 클래스 상속 예시 1

Animal.java

```
public abstract class Animal {  
    public abstract void printSound();  
  
    public void displayInfo() {  
        System.out.println("나는 동물입니다.");  
    }  
}
```

Cat.java

```
public class Cat extends Animal {  
    @Override  
    public void printSound() {  
        System.out.println("고양이는 야옹야옹");  
    }  
}
```


2. 추상 클래스

■ 추상 클래스의 상속

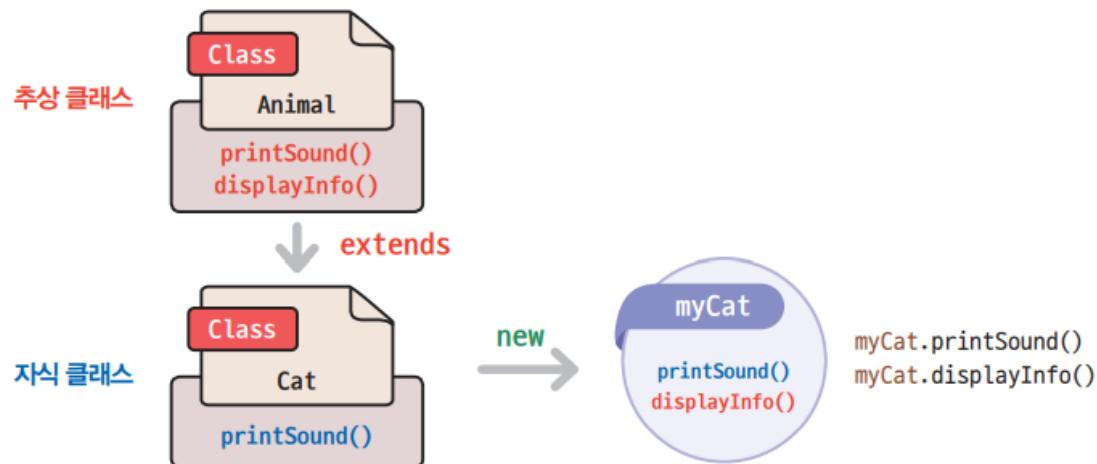
- 추상 클래스를 상속받는 일반 클래스

Example02.java

```
public class Example02 {  
    public static void main(String[] args) {  
        Cat myCat = new Cat();  
        myCat.printSound();  
        myCat.displayInfo();  
    }  
}
```

실행 결과

고양이는 야옹야옹
나는 동물입니다.

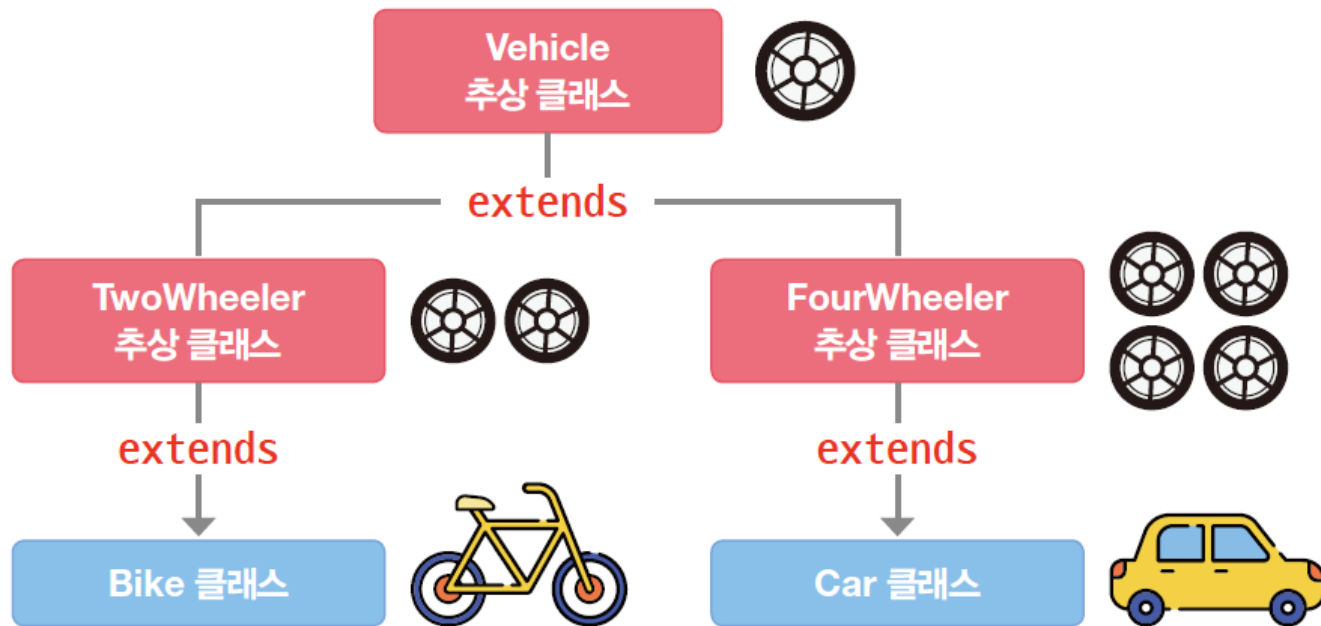


[그림 9-5] Example02.java의 프로그램 구조

2. 추상 클래스

■ 추상 클래스의 상속

- 추상 클래스를 상속받는 추상 클래스



[그림 9-6] 추상 클래스(Vehicle)를 상속받는 추상 클래스(TwoWheeler, FourWheeler)

2. 추상 클래스

■ 추상 클래스의 상속

- 추상 클래스를 상속받는 추상 클래스

추상 클래스 상속 예시 2

Vehicle.java

```
public abstract class Vehicle {  
    public abstract void printPrice();  
}
```

TwoWheeler.java

```
public abstract class TwoWheeler extends Vehicle {  
    public abstract void printType();  
}
```

FourWheeler.java

```
public abstract class FourWheeler extends Vehicle {}
```

2. 추상 클래스

■ 추상 클래스의 상속

- 추상 클래스를 상속받는 추상 클래스

Bike.java

```
public class Bike extends TwoWheeler {  
    public void printPrice() {  
        System.out.println("가격 : 150,000");  
    }  
    public void printType() {  
        System.out.println("이것은 자전거입니다.");  
    }  
    public void printBrand() {  
        System.out.println("브랜드 : 삼천리");  
    }  
}
```

Car.java

```
public class Car extends FourWheeler {  
    public void printPrice() {  
        System.out.println("가격 : 50,000,000");  
    }  
    public void printType() {  
        System.out.println("이것은 자동차입니다.");  
    }  
    public void printBrand() {  
        System.out.println("브랜드 : BMW");  
    }  
}
```

2. 추상 클래스

■ 추상 클래스의 상속

- 추상 클래스를 상속받는 추상 클래스

Example03.java

```
public class Example03 {  
    public static void main(String[] args) {  
        Bike myBike = new Bike();  
        Car myCar = new Car();  
        myBike.printType();  
        myBike.printBrand();  
        myBike.printPrice();  
        System.out.println("-----");  
        myCar.printType();  
        myCar.printBrand();  
        myCar.printPrice();  
    }  
}
```

실행 결과

이것은 자전거입니다.

브랜드 : 삼천리

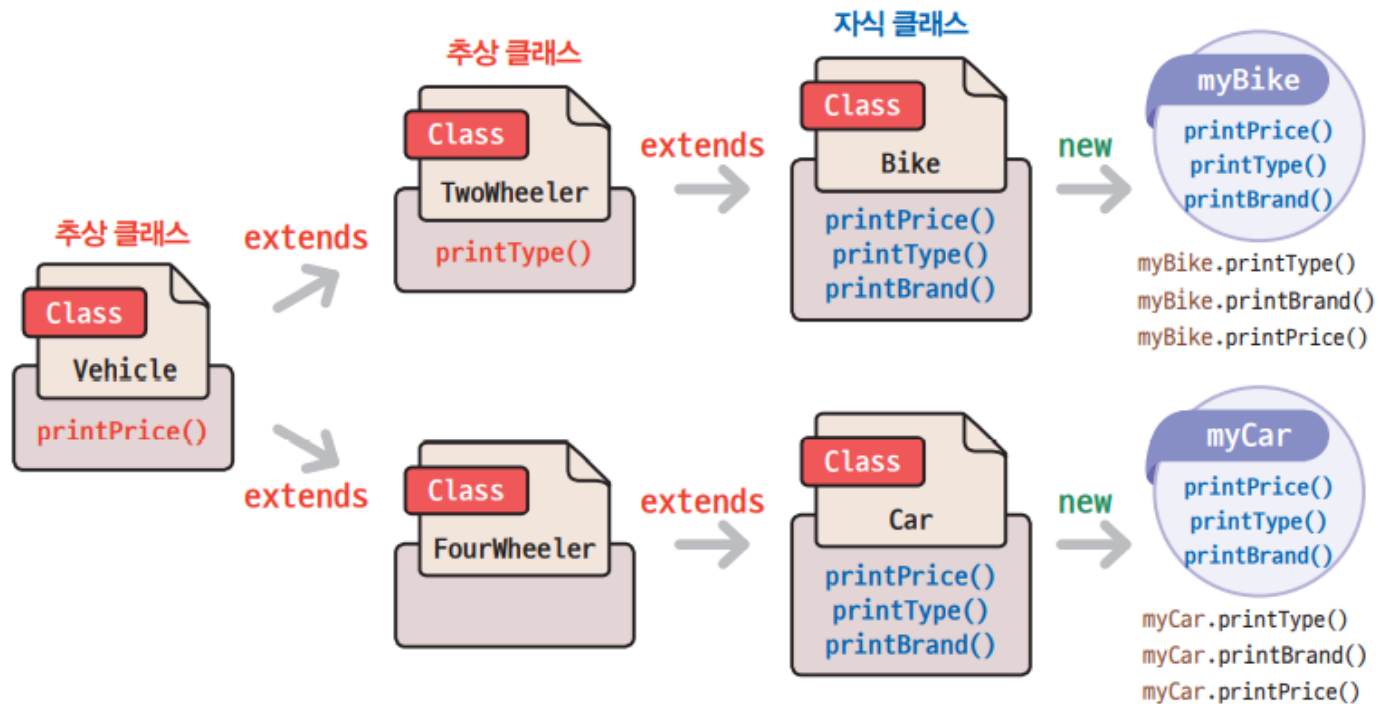
가격 : 150,000

이것은 자동차입니다.

브랜드 : BMW

가격 : 50,000,000

2. 추상 클래스



[그림 9-7] Example03.java의 프로그램 구조

2. 추상 클래스

예제 9-1

추상 클래스를 상속하여 인스턴스 생성하기

Shape.java

```
01 public abstract class Shape {  
02     String color;  
03  
04     abstract double area();  
05     public abstract String toString();  
06  
07     public Shape(String color) {  
08         System.out.println("Shape 클래스 생성자 호출");  
09         this.color = color;  
10     }  
11  
12     public String getColor() { return color; }  
13 }
```

2. 추상 클래스

Circle.java

```
01 public class Circle extends Shape {
02     double radius;
03
04     public Circle(String color, double radius) {
05         super(color);
06         System.out.println("Circle 클래스 생성자 호출");
07         this.radius = radius;
08     }
09
10     double area() {
11         return radius * radius * 3.14;
12     }
13
14     public String toString() {
15         return "원 색상은 " + super.getColor() + " 그리고 면적은 : " + area();
16     }
17 }
```


2. 추상 클래스

Rectangle.java

```
01 public class Rectangle extends Shape {
02
03     double length;
04     double width;
05
06     public Rectangle(String color, double length, double width) {
07         super(color);
08         System.out.println("Rectangle 클래스 생성자 호출");
09         this.length = length;
10         this.width = width;
11     }
12
13     double area() { return length * width; }
14
15     public String toString() {
16         return "사각형 색상은 " + super.getColor() + " 그리고 면적은 : " + area();
17     }
18 }
```

2. 추상 클래스

AbstractClass01.java

```
01 public class AbstractClass01 {  
02     public static void main(String[] args) {  
03         Shape s1 = new Circle("빨간색", 2.2);  
04         Shape s2 = new Rectangle("노란색", 2, 4);  
05  
06         System.out.println(s1.toString());  
07         System.out.println(s2.toString());  
08     }  
09 }
```

실행 결과

Shape 클래스 생성자 호출

Circle 클래스 생성자 호출

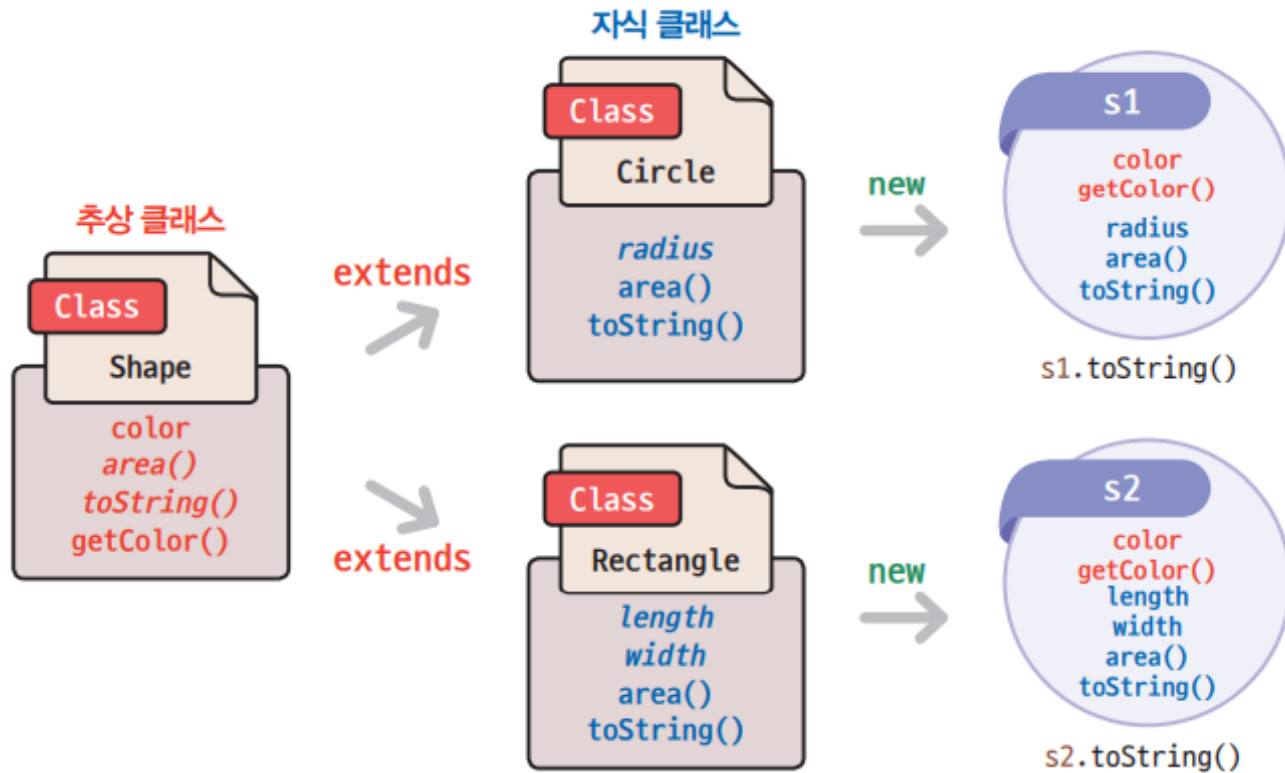
Shape 클래스 생성자 호출

Rectangle 클래스 생성자 호출

원 색상은 빨간색 그리고 면적은 : 15.197600000000003

사각형 색상은 노란색 그리고 면적은 : 8.0

2. 추상 클래스



[그림 9-8] AbstractClass01.java의 프로그램 구조

Section 03

인터페이스

3. 인터페이스

■ 인터페이스의 개념

- 추상 클래스와 마찬가지로 인터페이스는 그 자체의 객체를 만들 수 없음
- 추상 클래스는 추상 메서드와 일반 메서드를 포함할 수 있지만 인터페이스는 추상 메서드만 포함할 수 있음

■ 인터페이스 사용 이유

- 완전한 추상화를 구현할 수 있음
- 다중 상속을 구현할 수 있음

3. 인터페이스

■ 인터페이스의 선언

- 인터페이스는 interface 키워드를 사용하여 선언
- 인터페이스 내부에 있는 모든 메서드는 추상 메서드, 즉 본문이 없는 메서드임

```
interface 인터페이스명 {  
    반환유형 변수 [= 값];  
    반환유형 메서드명([매개변수목록]);  
}  
  
interface Parent {  
    int age = 50;  
    void printInfo();  
}
```

3. 인터페이스

■ 인터페이스의 선언

- 인터페이스 메서드

- 추상 메서드처럼 본문이 없이 접근제한자, 이름, 반환 유형, 메서드에 대한 매개변수만을 포함하고, 인터페이스를 구현하는(상속받는) 클래스에서 모든 메서드를 구현해야 함

- 인터페이스 메서드는 기본적으로 `public abstract`이므로 생략 가능

- 인터페이스 변수

- 인터페이스에 선언하는 모든 변수는 기본적으로 `public static final`이므로 생략 가능

3. 인터페이스

■ 인터페이스의 선언

인터페이스 정의 예시

Phone.java

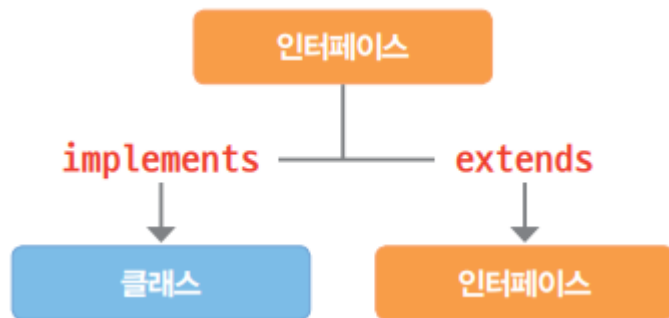
```
public interface Phone {  
  
    int PHONE_NUM_1 = 1; // public static final 생략 가능  
    int PHONE_NUM_2 = 2;  
    int PHONE_NUM_3 = 3;  
    int PHONE_NUM_4 = 4;  
    int PHONE_NUM_5 = 5;  
    int PHONE_NUM_6 = 6;  
    int PHONE_NUM_7 = 7;  
    int PHONE_NUM_8 = 8;  
    int PHONE_NUM_9 = 9;  
    int PHONE_NUM_0 = 0;  
  
    void call(); // public abstract 생략 가능  
  
    void turnOn();  
}
```


3. 인터페이스

■ 인터페이스의 상속

- 단일 상속

- 추상 클래스와 마찬가지로 인터페이스도 인스턴스를 만들 수 없기 때문에 상속을 사용
- 인터페이스의 상속에는 **implements** 키워드를 사용하고 상속받은 클래스를 인터페이스를 구현하는(상속받는) 클래스 또는 인터페이스 구현체라고 부름
- 인터페이스를 구현하는 클래스는 반드시 인터페이스 내에 선언된 모든 메서드를 구현해야 함

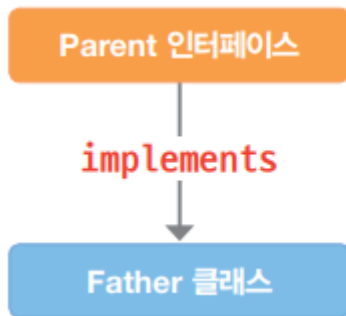


[그림 9-9] 인터페이스의 단일 상속

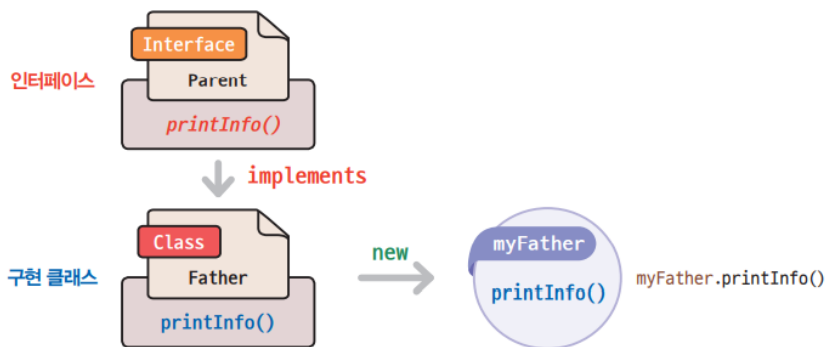
```
interface Parent {  
    // 추상 메서드 포함  
}  
  
class Father implements Parent {  
    // 추상 메서드 구현  
}
```

3. 인터페이스

■ 인터페이스의 상속



[그림 9-10] 인터페이스(Parent)를 상속받는 클래스(Father)



[그림 9-11] Example05.java의 프로그램 구조

인터페이스 정의 예시

Parent.java

```
public interface Parent {  
    public void printInfo();  
}
```

Father.java

```
public class Father implements Parent {  
  
    @Override  
    public void printInfo() {  
        System.out.println("아버지입니다");  
    }  
}
```

Example05.java

```
public class Example05 {  
    public static void main(String[] args) {  
        Father myFather = new Father();  
        myFather.printInfo();  
    }  
}
```

실행 결과

아버지입니다.

3. 인터페이스

예제 9-2

인터페이스 구현 클래스 만들기

Animal02.java

```
01 public interface Animal02 {  
02     public void animalSound();  
03     public void animalWalk();  
04 }
```

Pig.java

```
01 public class Pig implements Animal02 {  
02     public void animalSound() {  
03         System.out.println("꿀꿀꿀하고 소리 내다");  
04     }  
05     public void animalWalk() {  
06         System.out.println("네발로 걷다");  
07     }  
08 }
```

3. 인터페이스

Interface01.java

```
01 public interface Interface01 {  
02     public static void main(String[] args) {  
03         Pig myPig = new Pig();  
04         myPig.animalSound();  
05         myPig.animalWalk();  
06     }  
07 }
```

실행 결과

꿀꿀꿀하고 소리 내다
네발로 걷다

3. 인터페이스

■ 인터페이스의 상속

- 다중 상속

→ 단일 클래스가 둘 이상의 클래스를 상속할 수 없어 다중 상속이 불가능하기 때문에 인터페이스를 사용하여 다중 상속을 구현함



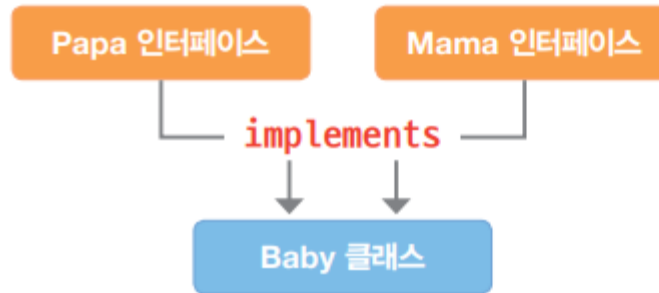
[그림 9-12] 인터페이스의 다중 상속

```
interface Papa {
    // 추상 메서드 포함
}
interface Mama {
    // 추상 메서드 포함
}
class Baby implements Papa, Mama {
    // 추상 메서드 구현
}
```

3. 인터페이스

■ 인터페이스의 상속

- 다중 상속



[그림 9-13] 2개의 인터페이스(Papa, Mama)를 상속받는 단일 클래스(Baby)

인터페이스 정의 예시

Papa.java

```
interface Papa {  
    public void genderFather();  
}
```

Mama.java

```
interface Mama {  
    public void genderMother();  
}
```

3. 인터페이스

■ 인터페이스의 상속

- 다중 상속

Baby.java

```
class Baby implements Papa, Mama {  
    public void genderFather() {  
        System.out.println("나는 아버지입니다.");  
    }  
    public void genderMother() {  
        System.out.println("나는 어머니입니다.");  
    }  
    public void printInfo() {  
        System.out.println("나는 아기입니다.");  
    }  
}
```

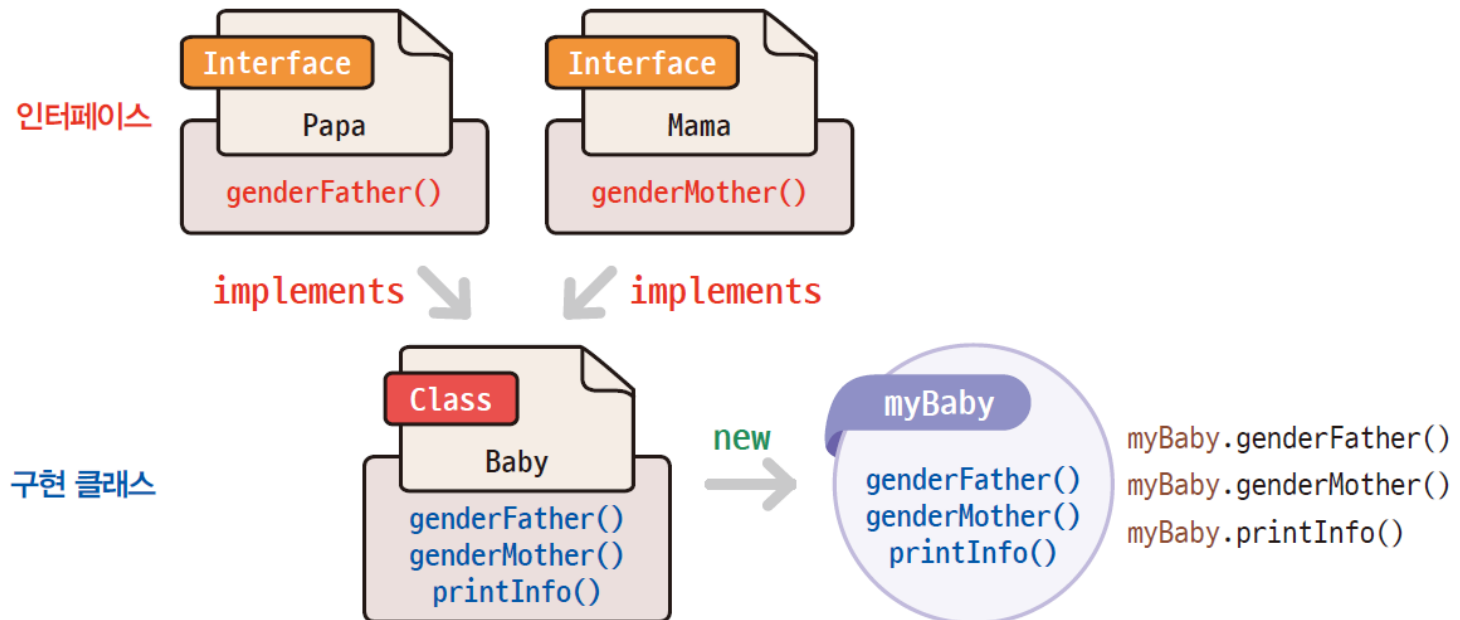
Example06.java

```
public class Example06 {  
    public static void main(String[] args) {  
        Baby myBaby = new Baby();  
        myBaby.genderFather();  
        myBaby.genderMother();  
        myBaby.printInfo();  
    }  
}
```

실행 결과

나는 아버지입니다.
나는 어머니입니다.
나는 아기입니다.

3. 인터페이스



[그림 9-14] Example06.java의 프로그램 구조

3. 인터페이스

예제 9-3

인터페이스 다중 상속의 구현 클래스 만들기

Fly.java

```
01 public interface Fly {  
02     public void fly();  
03 }
```

Walk.java

```
01 public interface Walk {  
02     public void walk();  
03 }
```

3. 인터페이스

Chicken.java

```
public class Chicken implements Fly, Walk {  
    public void fly() {  
        System.out.println("닭은 40cm를 날수 있다..");  
    }  
    public void walk() {  
        System.out.println("닭은 땅에 먹이를 쪼아 먹으며 걷는다.");  
    }  
}
```

Bird.java

```
public class Bird implements Fly , Walk{  
  
    public void fly() {  
        System.out.println("새는 하늘 높이 날아 둥지를 찾는다.");  
    }  
  
    public void walk() {  
        System.out.println("새는 주변을 살피며 걷는다.");  
    }  
}
```

3. 인터페이스

Interface02.java

```
import java.util.Scanner;
public class Interface02 {
    public static void main(String[] args) {
        System.out.println("동물선택:닭(1), 새(1)");
        Scanner input=new Scanner(System.in);
        int select=input.nextInt();
        switch(select) {
            case 1:
                Interface02.animalFly(new Chicken());
                Interface02.animalWalk(new Chicken());
                break;
            case 2:
                Interface02.animalFly(new Bird());
                Interface02.animalWalk(new Bird());
                break;
        }
    }
    public static void animalFly(Fly animal) {
        animal.fly();
    }
    public static void animalWalk(Walk animal) {
        animal.walk();
    }
}
```

3. 인터페이스

예제 9-4

인터페이스와 클래스를 다중 상속하는 클래스 만들기

Computer.java

```
public class Computer {  
    void calculator() {  
        System.out.println("컴퓨팅이 가능합니다.");  
    }  
}
```

Phone.java

```
public interface Phone {  
    int PHONE_NUM_1 = 1;  
    int PHONE_NUM_2 = 2;  
    int PHONE_NUM_3 = 3;  
    int PHONE_NUM_4 = 4;  
    int PHONE_NUM_5 = 5;  
    int PHONE_NUM_6 = 6;  
    int PHONE_NUM_7 = 7;  
    int PHONE_NUM_8 = 8;  
    int PHONE_NUM_9 = 9;  
    int PHONE_NUM_0 = 0;  
    void call(); < void turnOff();  
    void turnOn(); }
```

3. 인터페이스

예제 9-4

인터페이스와 클래스를 다중 상속하는 클래스 만들기

SmartPhoneImpl.java

```
public class SmartPhoneImpl extends Computer implements Phone {
    boolean power;
    @Override
    public void call() {
        String number1 = String.valueOf(PHONE_NUM_1);
        String number2 = String.valueOf(PHONE_NUM_1);
        String number3 = String.valueOf(PHONE_NUM_9);
        System.out.println(number1+number2+number3+"에 전화합니다. ");
    }

    @Override
    public void turnOn() {
        if(!power) {
            power = true;
        }
    }

    @Override
    public void turnOff() {
        if(power) {
            power = false;
        }
    }
}
```

3. 인터페이스

예제 9-4

인터페이스와 클래스를 다중 상속하는 클래스 만들기

SmartPhoneMain.java

```
public class SmartPhoneMain {  
    public static void main(String[] args) {  
  
        SmartPhoneImpl phone = new SmartPhoneImpl();  
        phone.call();  
        phone.calculator();  
    }  
}
```

3. 인터페이스

예제 9-5

인터페이스간의 상속 만들기

Application.java

```
public interface Application {  
    void appRun();  
    void appStop();  
}
```

SmartDevice.java

```
public interface SmartDevice extends Phone, Application {  
  
    // 인터페이스 Phone과 인터페이스 Application를 상속해서 새로운 인터페이스 정의  
  
}
```

3. 인터페이스

예제 9-5

인터페이스간의 상속 만들기

SmartPhoneImpl2.java

```
public class SmartPhoneImpl2 implements SmartDevice{
    boolean power;
    @Override
    public void call() {
        System.out.println("전화 통화를 합니다.");
    }
    @Override
    public void turnOn() {
        System.out.println("전원을 켭니다.");
    }
    @Override
    public void turnOff() {
        System.out.println("전원을 끕니다.");
    }
    @Override
    public void appRun() {
        System.out.println("앱을 실행합니다.");
    }
    @Override
    public void appStop() {
        System.out.println("앱을 종료합니다.");
    }
}
```


3. 인터페이스

예제 9-5

인터페이스간의 상속 만들기

SmartPhoneMain2.java

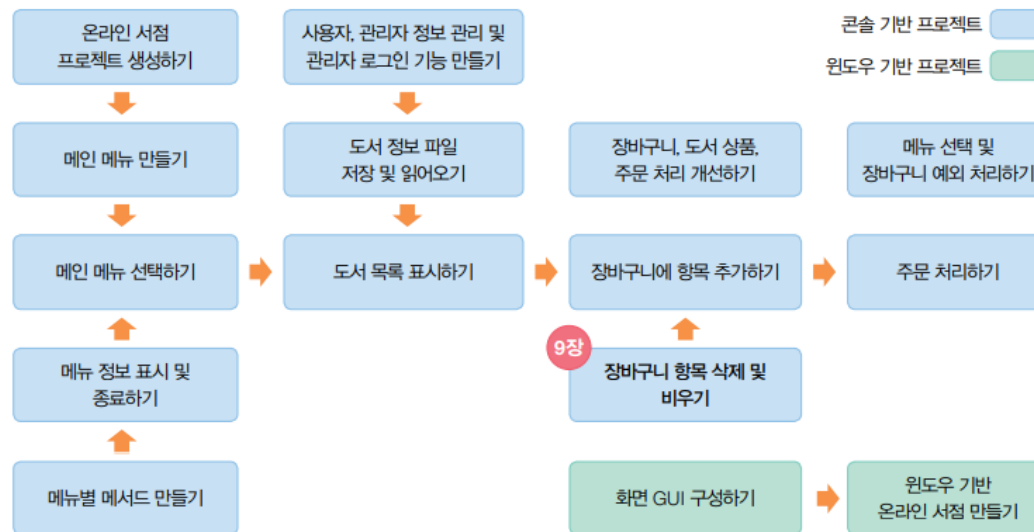
```
public class SmartPhoneMain2 {  
  
    public static void main(String[] args) {  
  
        SmartPhoneImpl2 phone = new SmartPhoneImpl2();  
  
        SmartDevice s = phone; // 상위 타입의 참조변수로 정의  
        s.turnOn();  
        Phone p = phone; // 상위 타입의 참조변수로 정의  
        p.call();  
        Application a = phone; // 상위 타입의 참조변수로 정의  
        a.appRun();  
  
    }  
  
}
```

[프로젝트]

장바구니 항목 삭제 및 비우기

장바구니 항목 삭제 및 비우기

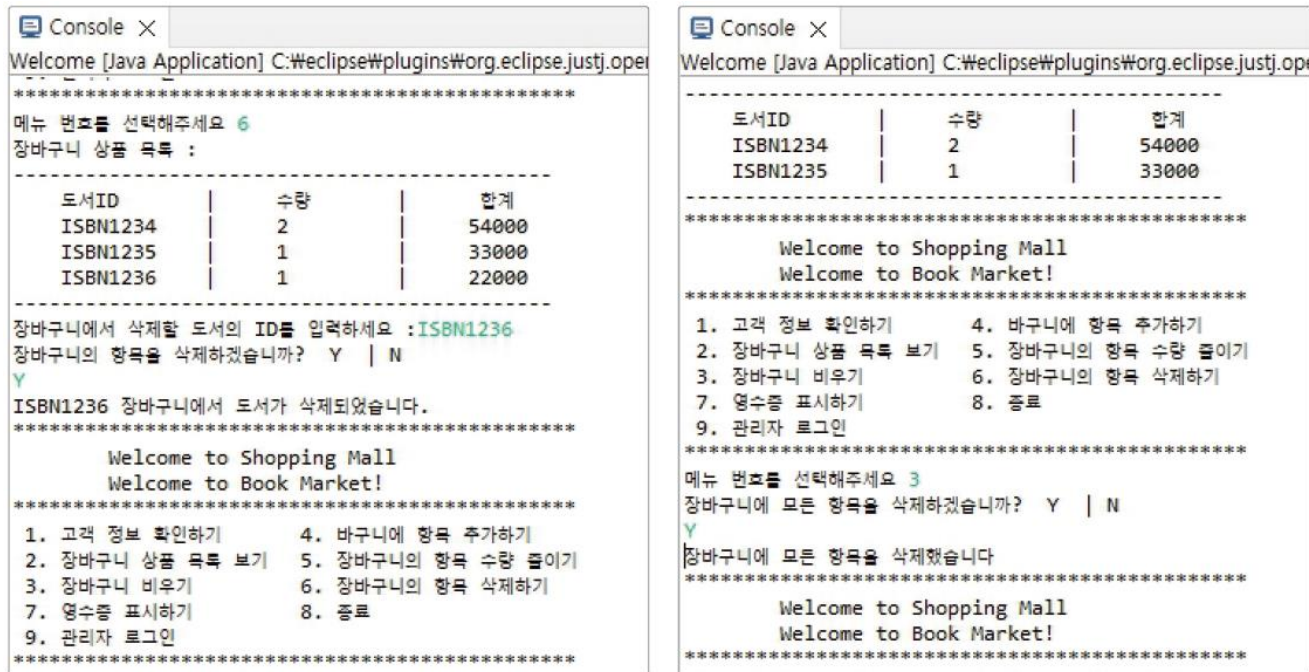
- 장바구니의 항목을 관리하는 추상 클래스를 생성한 뒤 장바구니의 항목을 처리하기 위한 인터페이스를 만들어 장바구니의 항목을 삭제하거나 장바구니 전체를 비울 수 있게 합니다



[그림 9-17] 장바구니 항목 삭제 및 비우기

장바구니 항목 삭제 및 비우기

- 장바구니의 항목을 관리하는 추상 클래스를 생성한 뒤 장바구니의 항목을 처리하기 위한 인터페이스를 만들어 장바구니의 항목을 삭제하거나 장바구니 전체를 비울 수 있게 합니다



[그림 9-18] 장바구니 항목 삭제 및 비우기 실행 결과

장바구니 항목 삭제 및 비우기

■ 장바구니 항목 관리 클래스 만들기

- 01 주요 도서 정보 관리 추상 클래스 생성하기

- Item.java 파일을 생성하고 주요 도서 정보를 관리 하는 Item 추상 클래스를 만듦

- 02 도서 정보 관리 클래스 생성하기

- Book.java 파일을 생성하고 Item 추상 클래스의 자식 클래스로 Book 클래스를 만듦

- 03 장바구니 항목 관리 클래스 수정하기

- Book 클래스를 적용하기 위해 CartItem.java 파일을 수정

장바구니 항목 삭제 및 비우기

■ 장바구니 항목 관리 클래스 만들기

• 04 장바구니 처리 인터페이스 생성하기

→ CartInterface.java 파일을 생성하고, 장바구니 처리를 위한 메서드를 정의하기 위한 CartInterface 인터페이스를 만들

- ✓ void printBookList(Book[] p): 전체 도서 정보 목록을 출력하는 메서드
- ✓ boolean isCartInBook(String id): 장바구니에 담긴 도서의 ID와 장바구니에 담은 도서의 ID를 비교하는 메서드
 - ID가 일치하면 장바구니에 담긴 도서의 개수를 1만큼 증가시키고 true를 반환하며, 그렇지 않으면 false를 반환
- ✓ void insertBook(Book p): 장바구니 항목 관리 클래스 CartItem에 도서 정보를 등록하는 메서드
- ✓ void removeCart(int numId): 장바구니 순번 numId의 항목을 삭제하는 메서드
- ✓ void deleteBook(): 장바구니의 모든 항목을 삭제하는 메서드

장바구니 항목 삭제 및 비우기

■ 장바구니 항목 관리 클래스 만들기

- 04 장바구니 처리 인터페이스 생성하기

- CartInterface.java 파일을 생성하고, 장바구니 처리를 위한 메서드를 정의하기 위한 CartInterface 인터페이스를 만들

- 05 장바구니 처리 클래스 생성하기

- Cart.java 파일을 생성하고 CartInterface 인터페이스의 자식 클래스를 만들

- 06 도서 정보 저장하기

- Book 클래스를 이용하여 도서 정보를 저장하기 위해 Welcome.java 파일의 BookList() 메서드 수정

장바구니 항목 삭제 및 비우기

■ 장바구니의 항목 삭제하기

- 01 장바구니의 항목 삭제하기 메뉴 수정하기

- Welcome.java 파일에서 장바구니의 항목을 삭제하는 menuCartRemoveItem() 메서드 수정

장바구니 항목 삭제 및 비우기

■ 장바구니 비우기

- 01 장바구니 비우기 메뉴 수정하기

- Welcome.java 파일에서 장바구니의 항목을 모두 삭제하는 menuCartClear() 메서드 수정

장바구니 항목 삭제 및 비우기

■ 장바구니 관련 메뉴별 수정하기

- 01 장바구니에 항목 추가하기 메뉴 수정하기

- Welcome.java 파일에서 장바구니에 항목을 추가하는 menuCartAddItem() 메서드 수정

- 02 장바구니의 도서 ID 존재 여부 확인하기

- Welcome.java 파일에서 장바구니에 동일한 항목이 있는지 검사하는 isCartInBook() 메서드를 수정

- 03 장바구니 상품 목록 보기 메뉴 수정하기

- Welcome.java 파일에서 장바구니에 담긴 항목을 출력하는 menuCartItemList() 메서드 수정