

# 프로그래밍기초

## Chapter 09. 추상클래스와 인터페이스

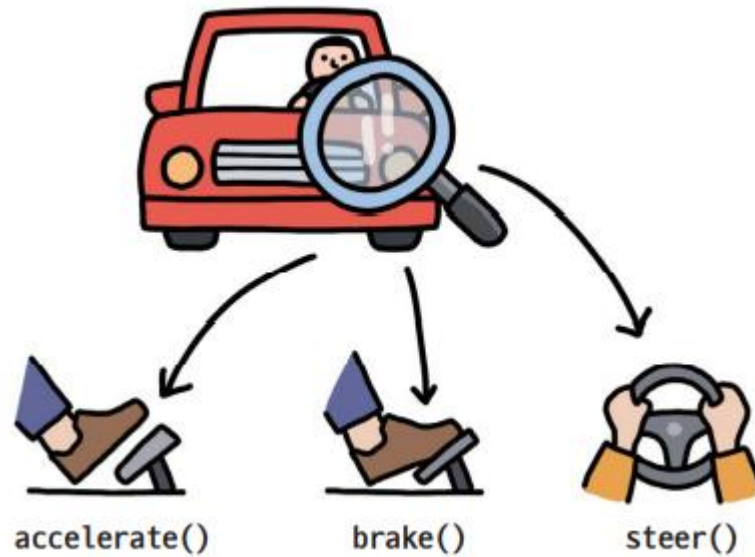
# Section 01

## 추상화

# 추상화

## ■ 추상화의 개념

- 추상화란 불필요한 정보를 숨기고 중요한 정보만을 나타내는 것을 의미
- 추상화를 이용하면 어떤 영역에서 필요한 공통의 속성이나 행동을 추출함으로써 효율적인 코드를 작성할 수 있음



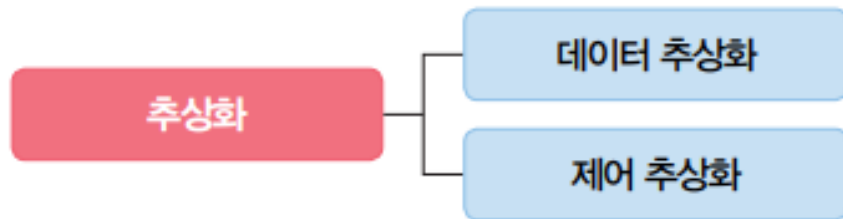
# 추상화

## ■ 추상화 사용의 장점

- 객체 간의 복잡성이 줄어듦
- 코드의 중복을 막고 재사용성을 높일 수 있음
- 사용자에게 중요한 세부 정보만 제공하므로 응용 프로그램이나 프로그램의 보안에 도움이 됨

## ■ 추상화의 유형

- 데이터 추상화  
→ 주로 복잡한 자료형을 만들고 구현을 숨기는 것으로, 구현의 세부 사항으로 이동 하지 않고 데이터 유형을 조작하는 작업만 노출
- 제어 추상화  
→ 작업의 단위 정의를 만들고 필요할 때마다 재사용하는 것으로, 반복되는 모든 코드 를 수집하고 이를 하나의 단위로 노출



# 추상화

## ■ 추상화 구현 방법

- 추상화는 추상 클래스 또는 인터페이스를 통해 구현할 수 있음

추상 클래스	인터페이스
한 번에 하나의 클래스 또는 추상 클래스만 상속할 수 있다.	한 번에 원하는 만큼의 인터페이스를 상속할 수 있다.
다른 구체적인(일반) 클래스 또는 추상 클래스를 상속할 수 있다.	다른 인터페이스만 상속할 수 있다.
추상 메서드와 구체적인 메서드를 모두 가질 수 있다.	추상 메서드만 가질 수 있다.
메서드를 추상으로 선언하는 데 abstract 키워드가 필요하다.	메서드를 추상으로 선언하는 데 abstract 키워드가 선택 사항이다.
접근제한자 protected, public이 선언된 추상 메서드를 가질 수 있다.	접근제한자 public이 선언된 추상 메서드를 가질 수 있다.
모든 접근제한자와 함께 static, final, static final 변수를 가질 수 있다.	static final(상수) 변수만 가질 수 있다.

# Section 02


## 추상 클래스

# 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]);  
}
```

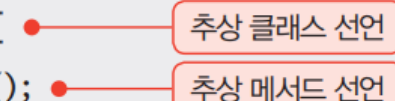


# 추상 클래스

## ■ 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
- 추상 클래스는 추상 메서드를 포함할 수도 있고 포함하지 않을 수도 있음
- 클래스에 추상 메서드가 포함되어 있으면 반드시 추상 클래스로 선언해야 함

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





# 추상 클래스

## ■ 추상 클래스 생성 예시

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

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

실행 결과

오류 발생

# 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 메서드 재정의는 필수
  - 상속받은 클래스와 자식 클래스가 추상 클래스라면 메서드를 재정의하지 않아도 됨
  - 현재 클래스가 상위 클래스처럼 추상적인 경우에는 상위 클래스의 메서드를 재정의할 필요 없음
- 추상 클래스는 자신의 인스턴스를 가질 수 없음
  - 추상적인 클래스의 객체를 가질 수 없으나 추상 클래스를 참조하는 객체는 가질 수 있음
- 프로그램이 자식 클래스의 객체를 만들 때 컴파일러가 추상 클래스의 생성자를 호출함
- 상속만 가능하고 객체를 생성할 수 없는 클래스를 생성하려면 추상 클래스 내부에 일반 메서드만 가질 수 있음
- 단일 클래스가 여러 추상 클래스를 상속받을 수 없음. 다중 상속을 구현하려면 인터페이스를 사용해야 함
- 추상 클래스에 final() 메서드를 포함할 수도 있음
  - 하지만 final 클래스는 추상 메서드를 가질 수 없음

# 추상 클래스

## ■ 추상 클래스의 상속

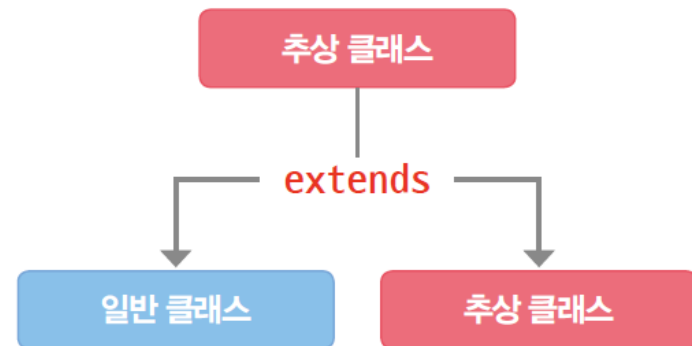
- 추상 클래스는 그 자체의 인스턴스를 만들 수 없기 때문에 상속해서 사용하며, 보통의 클래스처럼 extends 키워드를 이용하여 상속함
- 일반적으로 자식 클래스는 이러한 추상 클래스를 상속받고 추상 메서드를 재정의하여 사용함

```
abstract class Animal {  
    // 코드  
}  
  
class Cat extends Animal {  
    // 코드  
}  
  
abstract class Mammal extends Animal {  
    // 코드  
}
```

추상 클래스

추상 클래스를 상속받는 일반 클래스

추상 클래스를 상속받는 추상 클래스

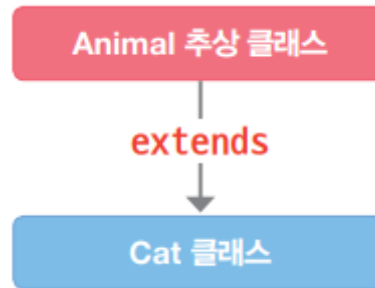


# 추상 클래스

## ■ 추상 클래스의 상속

- 추상 클래스를 상속받는 일반 클래스

## ■ 추상 클래스 상속 예시 1



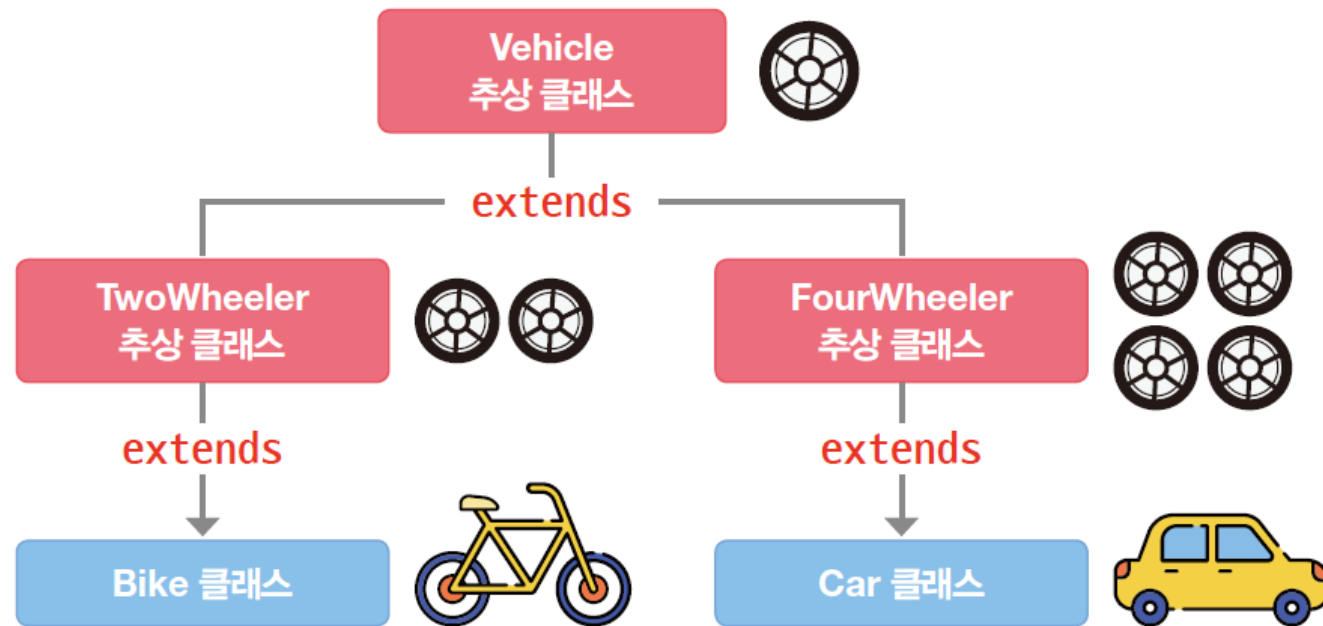
```
public abstract class Animal {  
    public abstract void printSound();  
  
    public void displayInfo() {  
        System.out.println("나는 동물입니다.");  
    }  
}
```

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

# 추상 클래스

## ■ 추상 클래스의 상속

- 추상 클래스를 상속받는 추상 클래스



# 추상 클래스

## ■ 추상 클래스의 상속

→ 추상 클래스를 상속받는 추상 클래스

## ■ 추상 클래스 상속 예시 2

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

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

```
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");  
    }  
}
```

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

```
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");  
    }  
}
```

# 추상 클래스

## ■ 추상 클래스 상속 예시 2

```
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

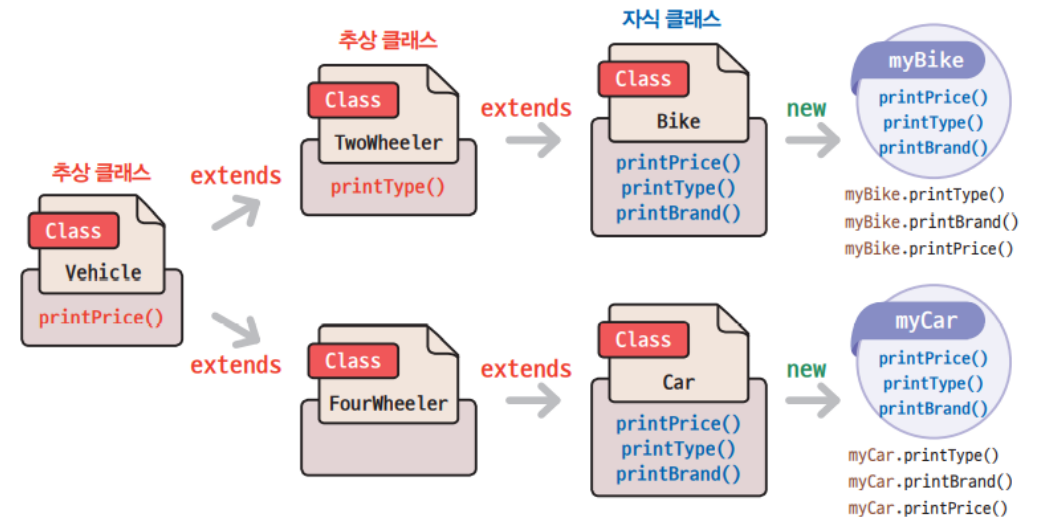
가격 : 150,000

-----

이것은 자동차입니다.

브랜드 : BMW

가격 : 50,000,000



# 추상 클래스

---

## ■ 예제 9-1. 추상 클래스를 상속하여 인스턴스 생성하기

```
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 }
```



# 추상 클래스

## ■ 예제 9-1. 추상 클래스를 상속하여 인스턴스 생성하기

```
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 }
```

# 추상 클래스

## ■ 예제 9-1. 추상 클래스를 상속하여 인스턴스 생성하기

```
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 }
```

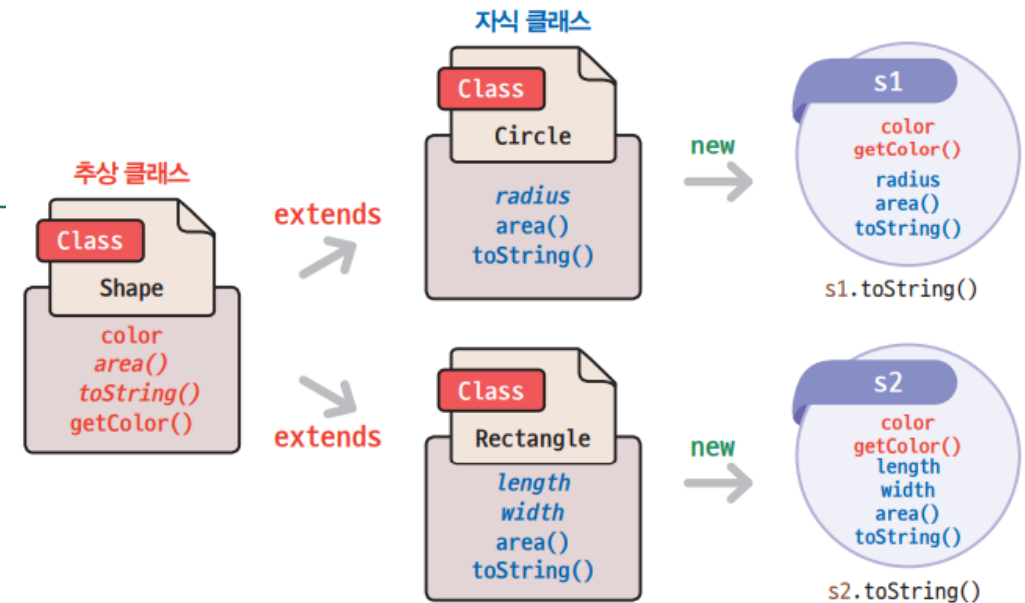
# 추상 클래스

## ■ 예제 9-1. 추상 클래스를 상속하여 인스턴스 생성하기

```
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



# **Section 03**

## **인터페이스**

# 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
- 느슨한 결합 관계를 형성할 수 있음

## ■ 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();  
}
```

# 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 public과 abstract이므로 명시적으로 선언하지 않아도 되며, 인터페이스 내에서 private 또는 abstract가 아닌 메서드를 정의하면 오류가 발생

### • 인터페이스 변수

- 인터페이스는 변수를 포함할 수 있으나 변수를 선언하지 않는 것이 좋음
- 인터페이스를 구현 하는(상속받는) 클래스에 동일한 변수가 있는 경우 인터페이스 내부에 변수를 선언하면 중복 되기 때문
- 인터페이스에 선언하는 모든 변수는 기본적으로 public, static, final이므로 명시 적으로 선언할 필요가 없음

# 인터페이스

---

## ■ 인터페이스 정의 예시

```
public interface Parent {  
    public void gender();  
  
    public void printInfo() {  
        System.out.println("부모입니다");  
    }  
}
```

```
public class Example04 {  
    public static void main(String[] args) {  
        Parent myObj = new Parent();  
    }  
}
```

실행 결과

오류 발생

# 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

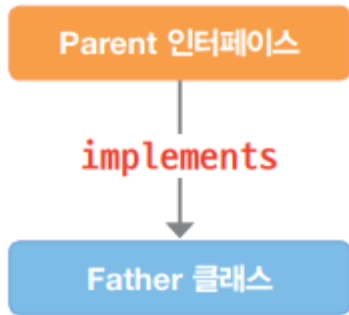


```
interface Parent {  
    // 추상 메서드 포함  
}  
  
class Father implements Parent {  
    // 추상 메서드 구현  
}
```



# 인터페이스

## ■ 인터페이스 정의 예시



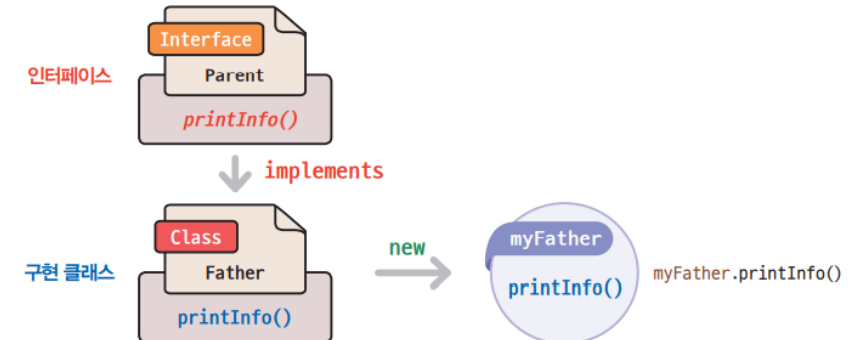
```
public interface Parent {  
    public void printInfo();  
}
```

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

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

실행 결과

아버지입니다.



# 인터페이스

---

## ■ 예제 9-2. 인터페이스 구현 클래스 만들기

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

```
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 }
```

# 인터페이스

---

## ■ 예제 9-2. 인터페이스 구현 클래스 만들기

```
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

# 인터페이스

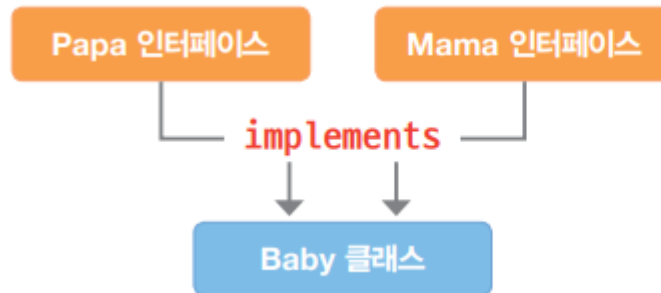
## ■ 인터페이스의 상속

- 다중 상속

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



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



# 인터페이스

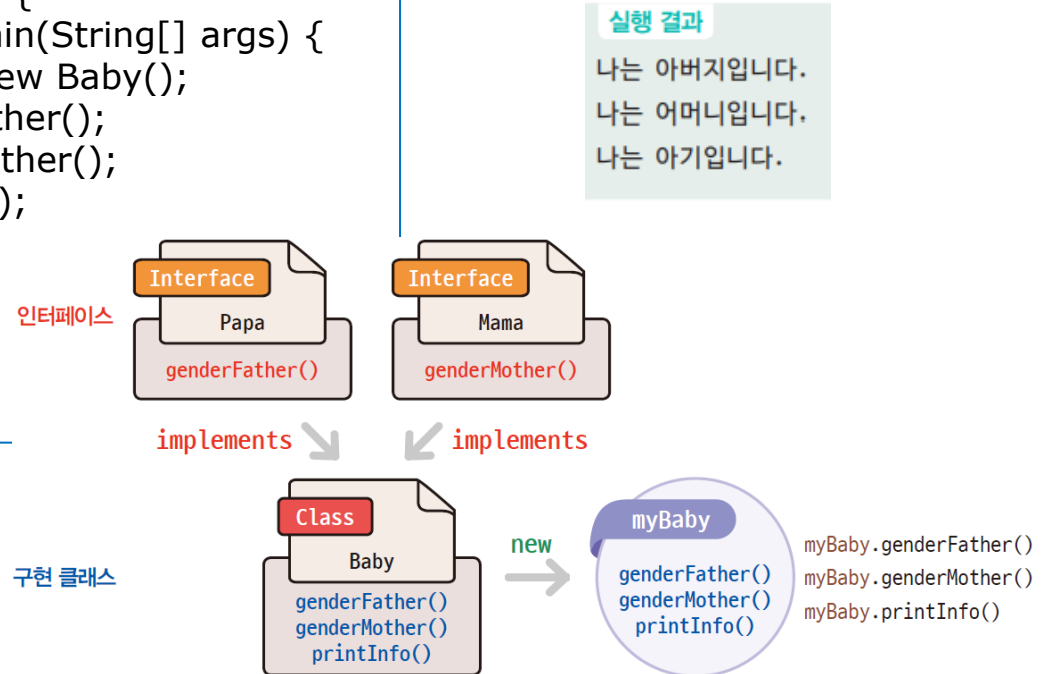
## ■ 인터페이스 정의 예시

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

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

```
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.");  
    }  
}
```

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



# 인터페이스

## ■ 예제 9-3. 인터페이스 다중 상속의 구현 클래스 만들기

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

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

```
01 public class Chicken implements Fly, Walk {  
02  
03     public void fly() {  
04         System.out.println("닭은 날 수 있다.");  
05     }  
06  
07     public void walk() {  
08         System.out.println("닭은 걸을 수 있다.");  
09     }  
10 }
```

```
01 public class Bird implements Fly {  
02  
03     public void fly() {  
04         System.out.println("새는 날 수 있다.");  
05     }  
06 }
```

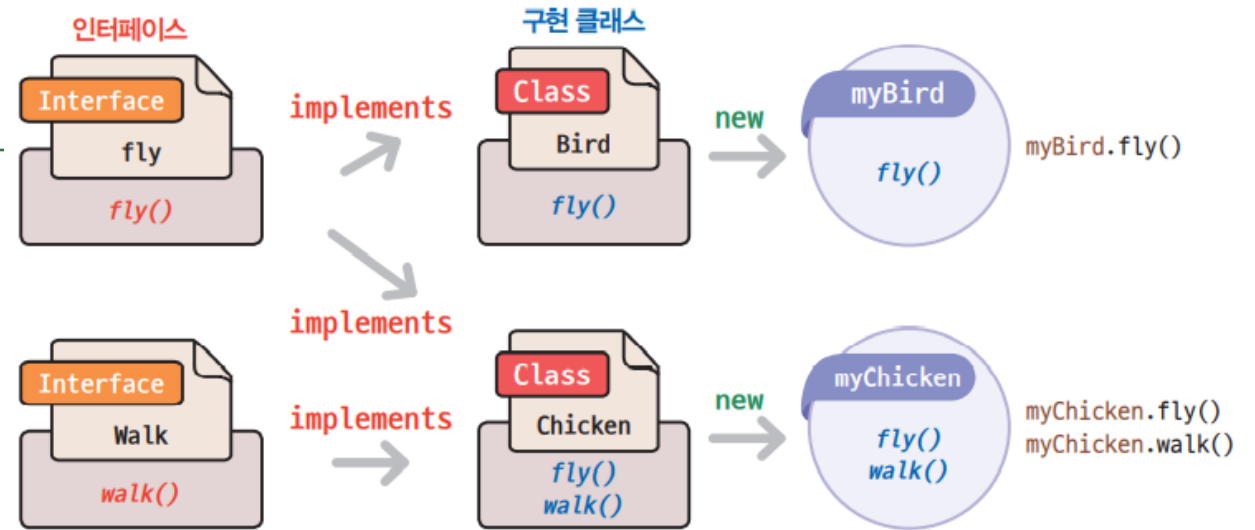
# 인터페이스

## ■ 예제 9-3. 인터페이스 다중 상속의 구현 클래스 만들기

```
01 public class Interface02 {  
02     public static void main(String[] args) {  
03         Chicken myChicken = new Chicken();  
04         Bird myBird = new Bird();  
05  
06         myChicken.fly();  
07         myChicken.walk();  
08         myBird.fly();  
09     }  
10 }
```

### 실행 결과

닭은 날 수 있다.  
닭은 걸을 수 있다.  
새는 날 수 있다.



프로그래밍기초

Copyright © Lee Seungwon Professor  
All rights reserved.

<Q&A : [lsw@kopo.ac.kr](mailto:lsw@kopo.ac.kr)>