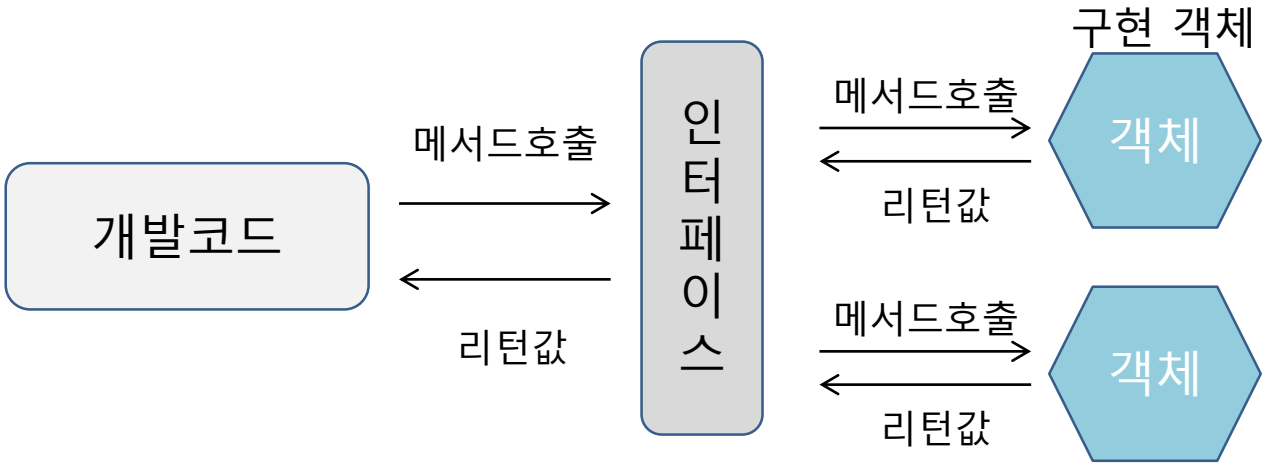
9장. 인터페이스

interface

인터페이스(Interface)

- **인터페이스란?**
  - 모든 메서드가 추상메서드(abstract method)로 이루어진 클래스이다.
  - **형식적인 선언만 있고 구현은 없다. *추상클래스처럼 상속 관계는 아님***
  - **인터페이스의 역할** : 클래스 혹은 프로그램이 제공하는 기능을 명시적으로 선언하는 역할을 한다. 즉 인터페이스만 봐도 어떤 매개변수가 사용되는지 또는 어떤 자료형이 반환되는지 알 수 있다.



인터페이스(Interface)

- **인터페이스 선언**

```
interface 인터페이스 이름{
    //상수
    상수이름 = 값;

    //추상메서드
    메서드 이름(매개변수, ...)
}
```

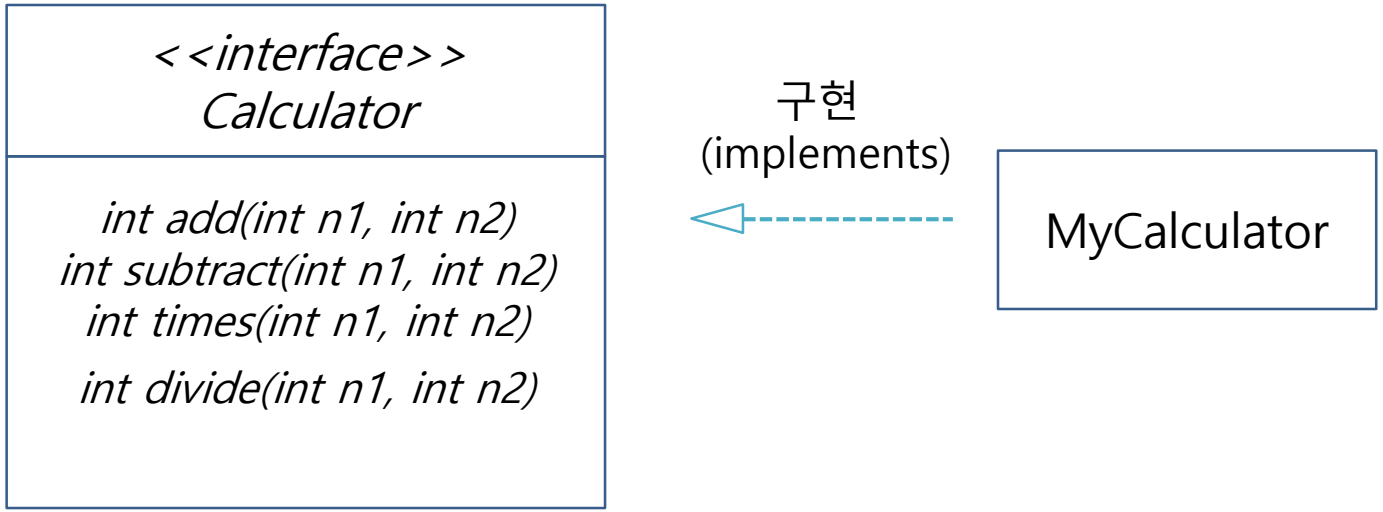
- **구현 클래스**

Implements 키워드 사용

```
class 구현클래스 이름 implements 인터페이스 이름{

    실체 메서드 구현
}
```

인터페이스(Interface)

- **정수형 계산기를 인터페이스로 구현하기**



인터페이스(Interface)

Calculator 인터페이스

```
package interfaceex.calculator;

public interface Calculator {
    //인터페이스에서 선언한 변수는 컴파일 과정에서 상수로 변환함.
    int ERROR = -99999;

    //abstract 예약어를 명시하지 않아도 컴파일 과정에서
    // 추상메서드로 변환됨
    int add(int n1, int n2);
    int subtract(int n1, int n2);
    int times(int n1, int n2);
    int divide(int n1, int n2);
}
```

인터페이스(Interface)

MyCalculator 클래스

Calculator 인터페이스를 구현한 클래스이다.

```
public class MyCalculator implements Calculator{

    @Override
    public int add(int n1, int n2) {
        return n1 + n2;
    }

    @Override
    public int subtract(int n1, int n2) {
        return n1 - n2;
    }

    @Override
    public int times(int n1, int n2) {
        return n1 * n2;
    }

    @Override
    public int divide(int n1, int n2) {
        //return n1 / n2;  //분모가 0인경우 오류 발생!
        if(n2 != 0)
            return n1 / n2;
        else
            return Calculator.ERROR;
    }
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Chapter9/interfaces.calculator.MyCalculator.divide(MyCalculator.java:22)
        at Chapter9/interfaces.calculator.CalculatorTest.main(CalculatorTest.java:12)
```

인터페이스(Interface)

- CalculatorTest 클래스

```
public class CalculatorTest {

    public static void main(String[] args) {
        MyCalculator calc = new MyCalculator();
        int num1 = 10;
        int num2 = 0;

        System.out.println(calc.add(num1, num2));
        System.out.println(calc.subtract(num1, num2));
        System.out.println(calc.times(num1, num2));
        System.out.println(calc.divide(num1, num2));
    }
}
```

```
10
10
0
-99999
```

인터페이스(Interface)

- try ~ catch로 에러 처리

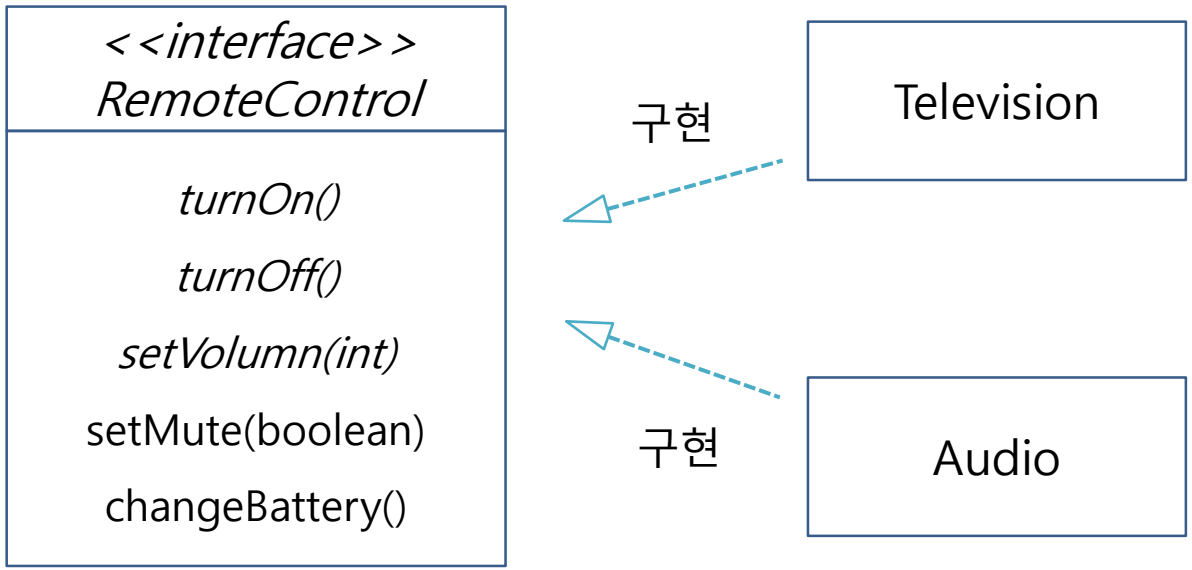
```
MyCalculator calc = new MyCalculator();
try {
    int num1 = 10;
    int num2 = 0;

    System.out.println(calc.add(num1, num2));
    System.out.println(calc.subtract(num1, num2));
    System.out.println(calc.times(num1, num2));
    System.out.println(calc.divide(num1, num2));
}catch(ArithmeticException e) {
    System.out.println("0으로 나눌수 없습니다.");
}
```

인터페이스 구성 요소

- **인터페이스의 요소**
  - **인터페이스 상수** : 인스턴스를 생성할 수 없으며 멤버 변수도 사용할 수 없다. -> 변수를 선언해도 오류가 나지 않는 이유는 상수로 변환됨
  - **추상메서드** : 구현부가 없는 추상메서드로 구성
  - **디폴트 메서드** : 기본 구현을 가지는 메서드, 구현 클래스에서 재정의 할수 있음 (자바 8부터 가능)
  - **정적 메서드** : 인스턴스 생성과 상관없이 인터페이스 타입으로 사용할 수 있는 메서드(자바 8부터 가능)

인터페이스

- **리모컨으로 TV와 오디오 구현하기**



인터페이스

- 리모컨 인터페이스

```
package interfaceex;
public interface RemoteControl {
    //인터페이스 상수
    public int MAX_VOLUME = 10;
    public int MIN_VOLUME = 0;

    //추상 메서드
    public void turnOn();
    public void turnOff();
    public void setVolume(int volume);

    //디폴트 메소드
    default void setMute(boolean mute) {
        if(mute) {
            System.out.println("무음 처리합니다.");
        }
        else {
            System.out.println("무음 해제합니다.");
        }
    }

    //정적 메서드
    static void changeBattery() {
        System.out.println("건전지를 교환합니다.");
    }
}
```

인터페이스

- Television 클래스

```
public class Television implements RemoteControl{

    private int volume;

    @Override
    public void turnOn() {
        System.out.println("TV를 켭니다.");
    }

    @Override
    public void turnOff() {
        System.out.println("TV를 끕니다.");
    }

    @Override
    public void setVolume(int volume) {
        if(volume > RemoteControl.MAX_VOLUME) {
            this.volume = RemoteControl.MAX_VOLUME;
        }
        else if(volume < RemoteControl.MIN_VOLUME) {
            this.volume = RemoteControl.MIN_VOLUME;
        }
        else {
            this.volume = volume;
        }
        System.out.println("현재 TV 볼륨: " + this.volume);
    }
}
```

인터페이스

- Audio 클래스

```
public class Audio implements RemoteControl{

    private int volume;

    @Override
    public void turnOn() {
        System.out.println("오디오를 켭니다.");
    }

    @Override
    public void turnOff() {
        System.out.println("오디오를 끕니다.");
    }

    @Override
    public void setVolume(int volume) {
        if(volume > RemoteControl.MAX_VOLUME) {
            this.volume = RemoteControl.MAX_VOLUME;
        }
        else if(volume < RemoteControl.MIN_VOLUME) {
            this.volume = RemoteControl.MIN_VOLUME;
        }
        else {
            this.volume = volume;
        }
        System.out.println("현재 오디오 볼륨: " + this.volume);
    }
}
```

인터페이스

- 리모컨 테스트 클래스

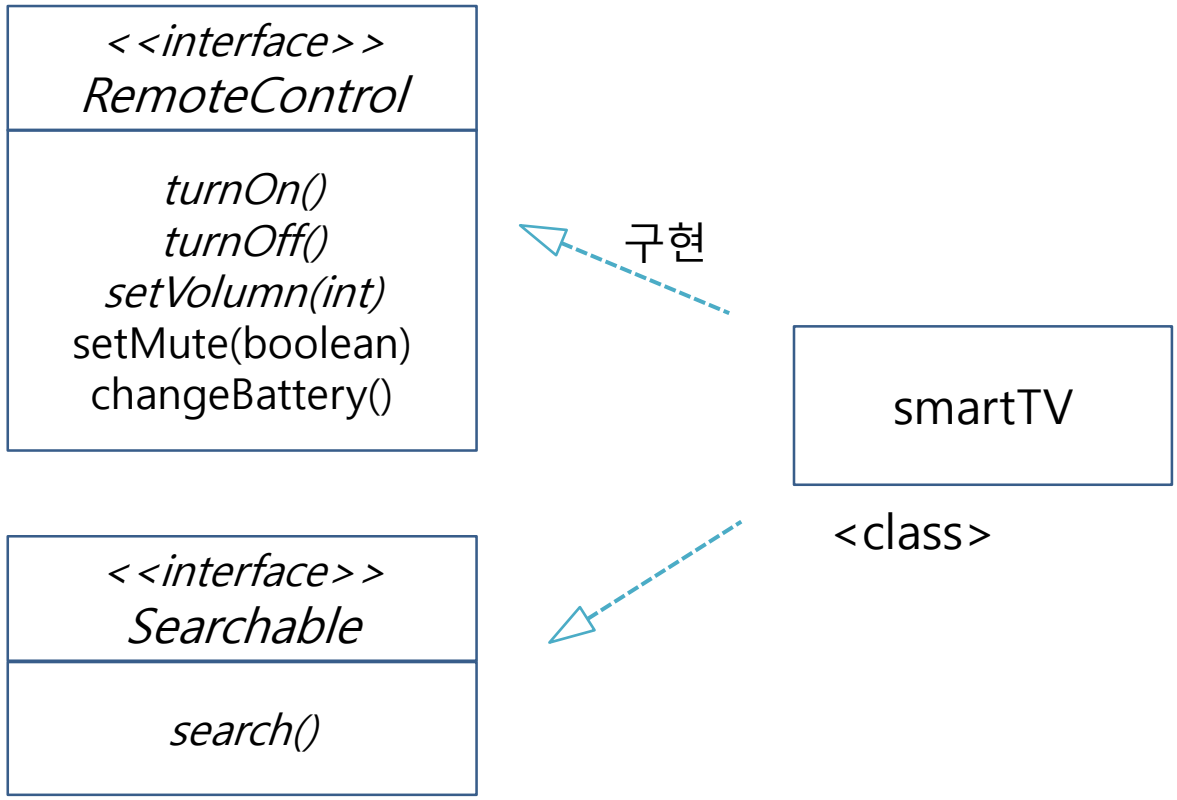
```
public class RemoteControlTest {
    public static void main(String[] args) {
        RemoteControl rcTV = new Television();
        RemoteControl rcAudio = new Audio();

        rcTV.turnOn();
        rcTV.setVolume(7);
        rcTV.setVolume(12);  //최대 볼륨값 - 10
        rcTV.setMute(true);
        rcTV.setMute(false);
        RemoteControl.changeBattery();
        rcTV.turnOff();

        System.out.println("============================");

        rcAudio.turnOn();
        rcAudio.setVolume(5);
        rcAudio.setVolume(-3);  //최소 볼륨값 - 0
        rcAudio.setMute(true);
        rcAudio.setMute(false);
        RemoteControl.changeBattery();
        rcAudio.turnOff();
    }
}
```

```
TV를 켭니다.
현재 TV 볼륨: 7
현재 TV 볼륨: 10
무음 처리합니다.
무음 해제합니다.
건전지를 교환합니다.
TV를 끕니다.
============================
오디오를 켭니다.
현재 오디오 볼륨: 5
현재 오디오 볼륨: 0
무음 처리합니다.
무음 해제합니다.
건전지를 교환합니다.
오디오를 끕니다.
```

다중 인터페이스 구현

인터페이스는 한 클래스가 여러 인터페이스를 다중 구현할 수 있다.

- **리모컨, 검색 인터페이스를 구현한 스마트TV**



다중 인터페이스

- 리모컨, 검색 인터페이스를 구현한 스마트TV

Searchable 인터페이스

```
package interfaceex.remotecontrol;

public interface Searchable {

    void search(String url);  //추상 메서드

}
```

다중 인터페이스

```java
public class SmartTV implements RemoteControl, Searchable{

    private int volume;

    @Override
    public void turnOn() {
        System.out.println("TV를 켭니다.");
    }

    @Override
    public void turnOff() {
        System.out.println("TV를 끕니다.");
    }

    @Override
    public void setVolume(int volume) {
        //볼륨의 크기 제한
        if(volume > RemoteControl.MAX_VOLUME) {
            this.volume = RemoteControl.MAX_VOLUME;
        }else if(volume < RemoteControl.MIN_VOLUME) {
            this.volume = RemoteControl.MIN_VOLUME;
        }else {
            this.volume = volume;
        }
        System.out.println("현재 TV 볼륨: " + this.volume);
    }
```

```java
@Override
public void search(String url) {
    System.out.println(url + "을 검색합니다.");
}
```

다중 인터페이스

- 스마트TV 테스트

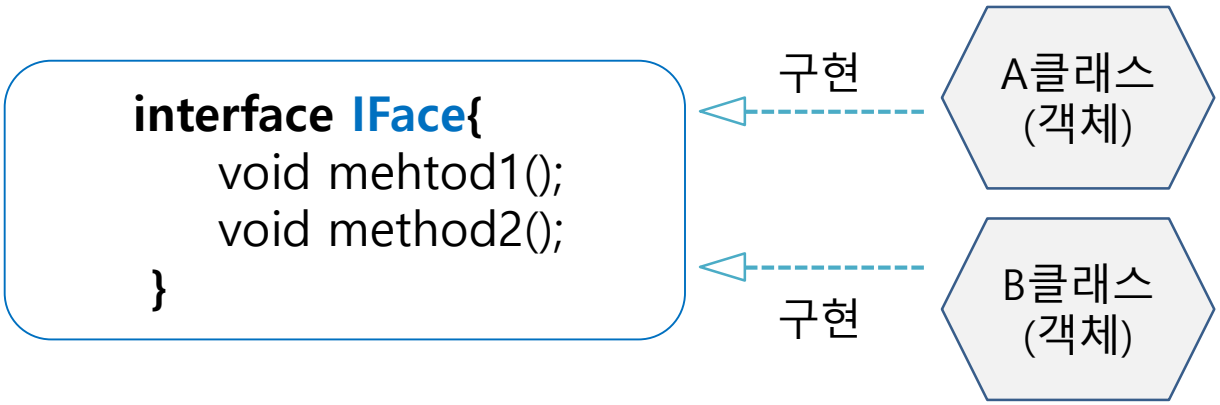
```
public class SmartTVTest {

    public static void main(String[] args) {

        SmartTV tv = new SmartTV();

        //구현 객체(tv)를 인터페이스에 타입에 대입
        RemoteControl rc = tv;
        Searchable searchable = tv;

        rc.turnOn();
        searchable.search("www.naver.com");
        rc.turnOff();
    }
}
```

인터페이스와 다형성

▪ 타입 변환과 다형성

인터페이스도 다형성을 구현하는 기술이 사용된다.

다형성은 하나의 타입에 대입되는 객체에 따라서 실행 결과가 다양한 형태로 나오는 성질을 말한다. 부모 타입에 어떤 지식 객체를 대입하느냐에 따라 실행 결과가 달라지듯이, 인터페이스 타입에 어떤 구현 객체를 대입하느냐에 따라 실행 결과가 달라진다.



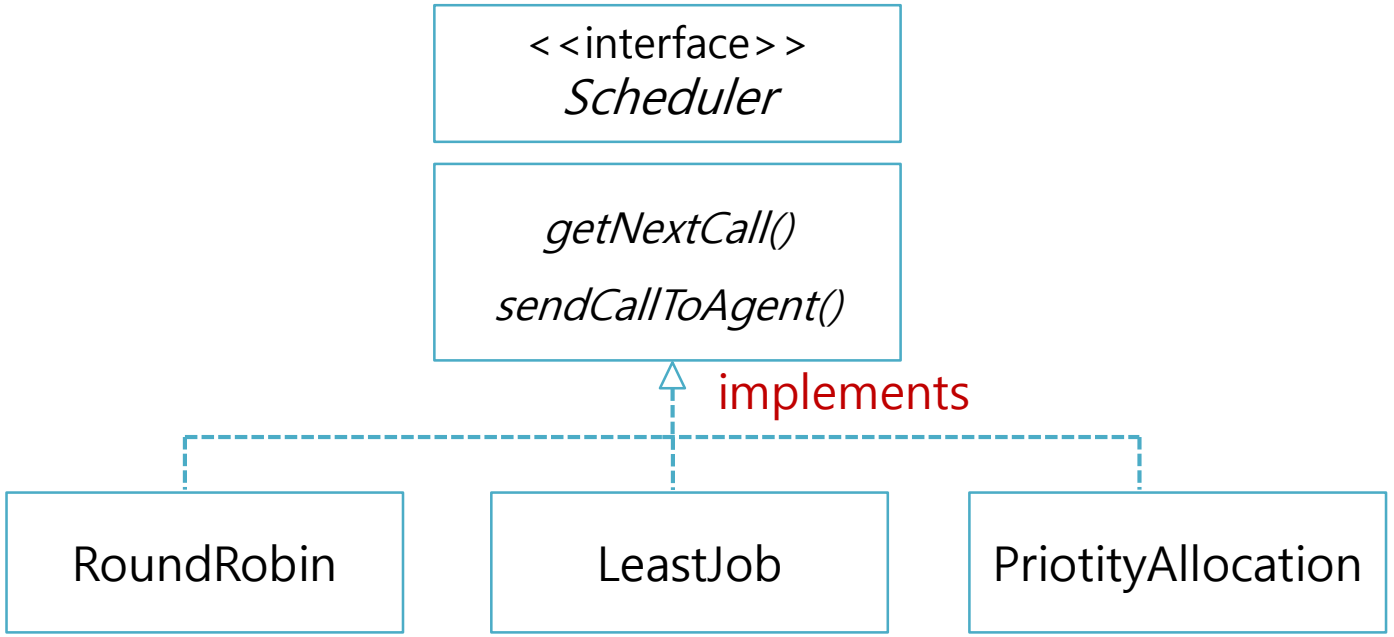
인터페이스와 다형성

- **고객 상담 전화 배분 프로그램**

예제 시나리오

고객 센터에는 전화 상담을 하는 상담원들이 있습니다. 일단 고객센터로 전화가 오면 대기열에 저장됩니다. 상담원이 지정되기 전까지는 대기 상태가 됩니다.

1. 순서대로 배분하기 - RoundRobin
2. 짧은 대기열 찾아 배분하기 - LeastJob
3. 우선순위에 따라 배분하기 - PriorityAllocation



인터페이스와 다형성

- **Scheduler 인터페이스**

```java
public interface Scheduler {
    //다음 전화를 가져오기
    public void getNextCall();

    //상담원에게 전화를 배분하기
    public void sendCallToAgent();
}
```

구현

Scheduler 인터페이스를 구현한 LeastJob 클래스

```java
public class LeastJob implements Scheduler{
    @Override
    public void getNextCall() {
        System.out.println("상담 전화를 순서대로 대기열에서 가져오기");
    }

    @Override
    public void sendCallToAgent() {
        System.out.println("현재 상담 업무가 없거나 대기가 가장 적은 상담원에게 할당합니다.");
    }
}
```

인터페이스와 다형성

RoundRobin 클래스

```
public class RoundRobin implements Scheduler{
    @Override
    public void getNextCall() {
        System.out.println("상담 전화를 순서대로 대기열에서 가져오기");
    }

    @Override
    public void sendCallToAgent() {
        System.out.println("다음 순서 상담원에게 배분합니다.");
    }
}
```

PriorityAllocation 클래스

```
public class PriorityAllocation implements Scheduler{
    @Override
    public void getNextCall() {
        System.out.println("고객 등급이 높은 고객의 전화를 먼저 가져옵니다.");
    }

    @Override
    public void sendCallToAgent() {
        System.out.println("업무 skill이 높은 상담원에게 우선 배분합니다.");
    }
}
```

인터페이스와 다형성

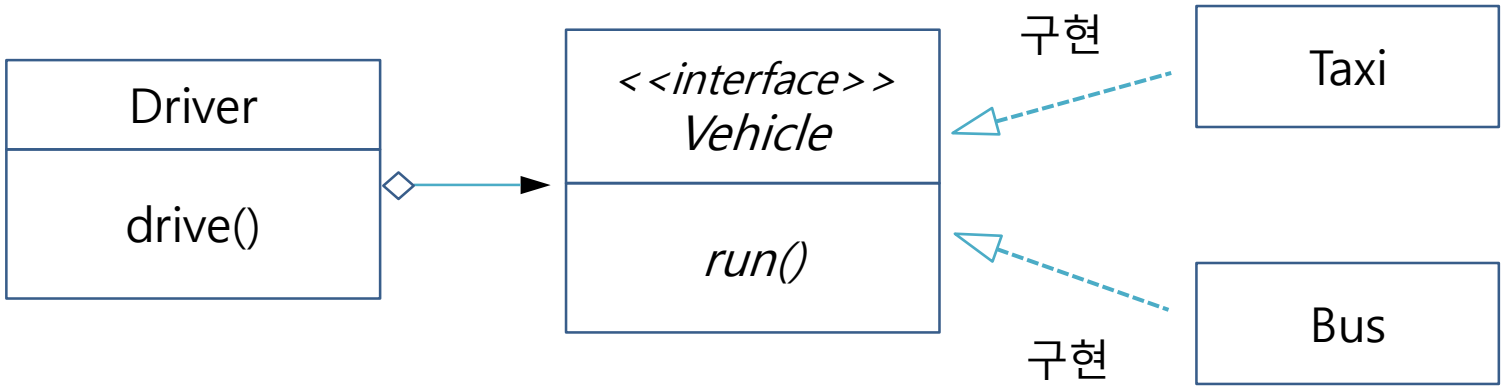
```
public class SchedulerTest {
    public static void main(String[] args) throws IOException { //예외 처리
        System.out.println("전화 상담 배분 방식을 선택하세요.");
        System.out.println("R : 한명씩 차례로 배분");
        System.out.println("L : 쉬고 있거나 대기가 가장 적은 상담원에게 배분");
        System.out.println("P : 우선 순위가 높은 고객 먼저 할당");

        int ch = System.in.read();  //할당 방식을 입력받아 ch에 대입
        Scheduler scheduler = null;

        if(ch=='R' || ch=='r') {  //입력받은 값이 'R' 이나 'r'이면
            scheduler = new RoundRobin();  //다형성으로 생성
        }
        else if(ch=='L' || ch=='l') {
            scheduler = new LeastJob();
        }
        else if(ch=='P' || ch=='p') {
            scheduler = new PriorityAllocation();
        }
        else {
            System.out.println("지원되지 않는 기능입니다.");
            return;
        }

        scheduler.getNextCall();       //입력 받은 정책의 메서드 호출
        scheduler.sendCallToAgent();
    }
}
```

SchedulerTest 클래스

```
전화 상담 배분 방식을 선택하세요.
R : 한명씩 차례로 배분
L : 쉬고 있거나 대기가 가장 적은 상담원에게 배분
P : 우선 순위가 높은 고객 먼저 할당
L
상담 전화를 차례대로 대기열에서 가져옵니다.
현재 상담업무가 없거나 대기가 가장 적은 상담원에게 배분합니다.
```

인터페이스와 다형성

- **운전자가 차량을 운전하는 인터페이스 예제**



운전자가 drive할때 run()을 사용함.
다이아몬드는 포함관계임.

인터페이스와 다형성

- 운전자가 차량을 운전하는 인터페이스 예제

Vehicle 인터페이스

```
package vehicle;

public interface Vehicle {

    public void run();

}
```

Bus 클래스

```
public class Bus implements Vehicle{

    @Override
    public void run() {
        System.out.println("버스가 달립니다.");
    }
}
```

Taxi 클래스

```
public class Taxi implements Vehicle{

    @Override
    public void run() {
        System.out.println("택시가 달립니다.");
    }
}
```

인터페이스와 다형성

- **매개변수의 다형성**

매개변수를 인터페이스 타입으로 선언하고 호출할 때에는 구현 객체를 대입한다.

매개변수의 다형성

```
public class Driver {

    public void drive(Vehicle vehicle) {
        vehicle.run();
    }
}
```

```
public class DriverTest {

    public static void main(String[] args) {
        Driver driver = new Driver();

        Bus bus = new Bus();
        Taxi taxi = new Taxi();

        driver.drive(bus);
        driver.drive(taxi);
    }
}
```

```
버스가 달립니다.
택시가 달립니다.
```

인터페이스와 강제 타입 변환

- **객체 타입 확인**

강제 타입 변환은 구현 객체가 인터페이스 타입으로 변환되어 있는 상태에서 가능하다.

```
package vehiclecasting;

public class Bus implements Vehicle{

    @Override
    public void run() {
        System.out.println("버스가 달립니다.");
    }

    public void checkFare() {
        System.out.println("승차 요금을 체크합니다.");
    }
}
```

인터페이스와 강제 타입 변환

- 강제 타입 변환

```
public class VehicleTest {

    public static void main(String[] args) {
        //인터페이스에서 매개 변수의 다형성
        Driver driver = new Driver();

        driver.drive(new Bus());
        driver.drive(new Taxi());

        //강제 타입 변환 - 인터페이스 타입으로 객체 생성
        Vehicle vehicle = new Bus();
        vehicle.run();

        if(vehicle instanceof Bus) {
            Bus bus = (Bus)vehicle;
            bus.checkFare();
        }
    }
}
```

```
버스가 달립니다.
버스가 달립니다.
승차 요금을 체크합니다.
```

인터페이스와 강제 타입 변환

- 강제 타입 변환

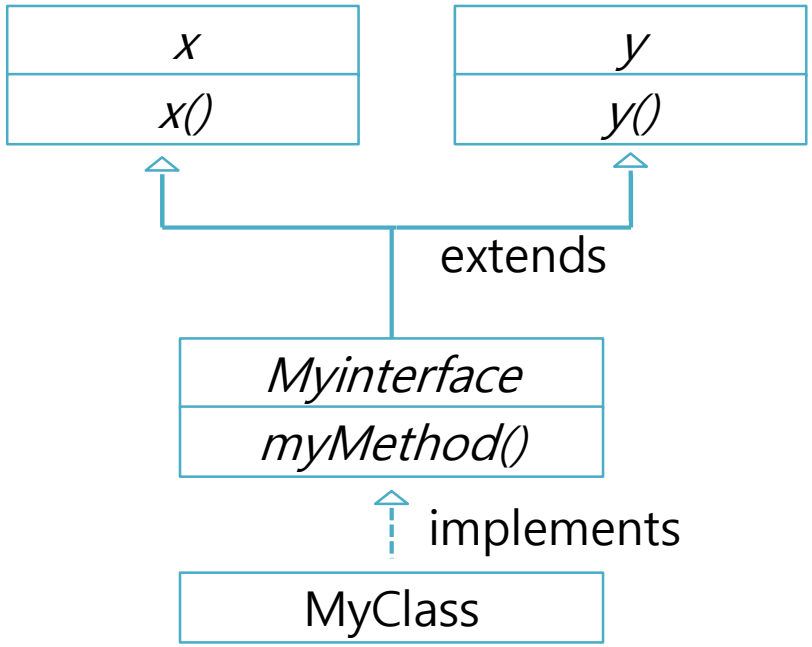
```
public class Driver {

    public void drive(Vehicle vehicle) {
        //vehicle이 Bus의 객체라면
        if(vehicle instanceof Bus) {
            Bus bus = (Bus)vehicle;
            bus.checkFare();
        }
        vehicle.run();
    }
}
```

```
public class DriverTest {

    public static void main(String[] args) {
        //driver 객체 생성
        Driver driver = new Driver();

        Bus bus = new Bus();
        Taxi taxi = new Taxi();

        driver.drive(bus);
        driver.drive(taxi);
    }
}
```

인터페이스 상속

- **인터페이스 상속하기**

  인터페이스간에도 상속이 가능하다. 구현 코드를 통해 기능을 상속하는 것이 아니므로 **형 상속(type inheritance)**라고 한다. 클래스의 경우는 단일 상속이지만, 인터페이스는 다중 상속이 가능하다.



인터페이스 상속

X 인터페이스

```
package interfaceinherit;

public interface X {

    void x();

}
```

Y 인터페이스

```
public interface Y {

    void y();

}
```

MyInterface 인터페이스

```
public interface MyInterface extends X, Y{

    void myMethod();

}
```

인터페이스 상속

```
public class MyClass implements MyInterface{

    @Override
    public void x() {
        System.out.println("x()");
    }

    @Override
    public void y() {
        System.out.println("y()");
    }

    @Override
    public void myMethod() {
        System.out.println("myMethod()");
    }
```

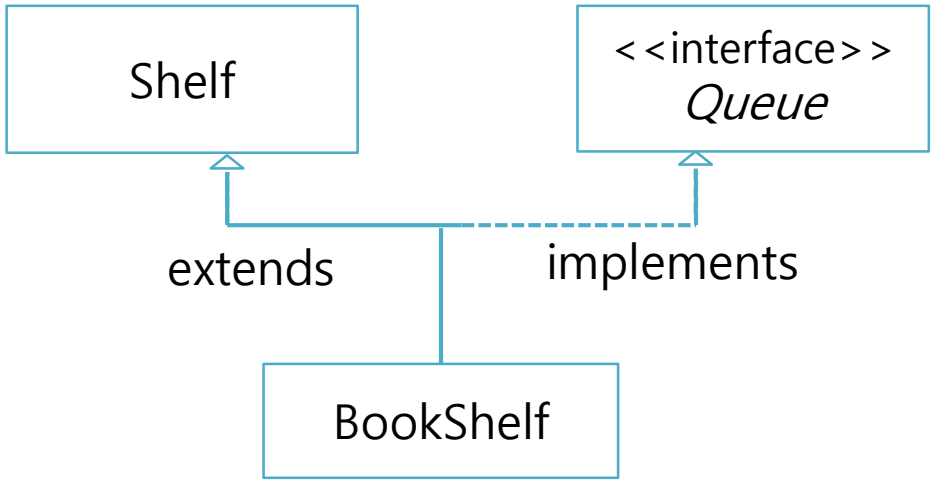
인터페이스 상속

```
public static void main(String[] args) {
    //자동 타입 변환
    MyClass myClass = new MyClass();
    X x = myClass;
    x.x();

    Y y = myClass;
    y.y();

    //X와 Y를 상속한 iClass 객체 생성
    System.out.println("** 다중 상속한 iClass 출력 **");
    MyInterface iClass = myClass;
    iClass.myMethod();
    iClass.x();
    iClass.y();
}
```

인터페이스 활용

- **인터페이스 구현과 클래스 상속 함께 사용하기**

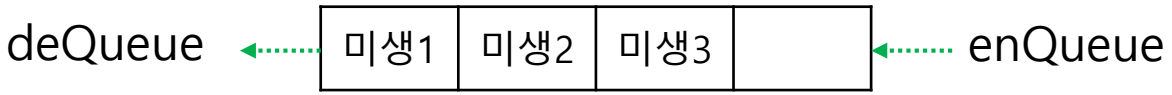
  Queue 인터페이스를 구현하고 shelf 클래스를 상속받는 BookShelf 클래스



인터페이스 활용

☆ 큐(Queue) 자료 구조

Queue : 선입선출(먼저 들어온 자료가 먼저 나옴) -> 선착순, 지하철 타기




```
package bookshelf;

public interface Queue {

    void enQuee(String title); //리스트의 맨 마지막에 추가

    String deQueue();    //리스트의 맨 처음 항목 반환

    int getSize();       //현재 Queue에 있는 개수 반환
}
```

인터페이스 활용

Shelf 클래스

```
public class Shelf {
    //문자열을 저장할 리스트 생성
    protected ArrayList<String> shelf;

    public Shelf() {
        shelf = new ArrayList<>();
    }

    public ArrayList<String> getShelf(){
        return shelf;
    }
}
```

인터페이스 활용

BookShelf 클래스

```
public class BookShelf extends Shelf implements Queue{

    @Override
    public void enQueue(String title) {
        shelf.add(title);  //ArrayList의 객체인 shelf를 상속받음
    }

    @Override
    public String deQueue() {
        return shelf.remove(0);
    }

    @Override
    public int getSize() {
        return shelf.size();
    }
}
```

상속

구현

인터페이스 활용

BookShelfTest 클래스

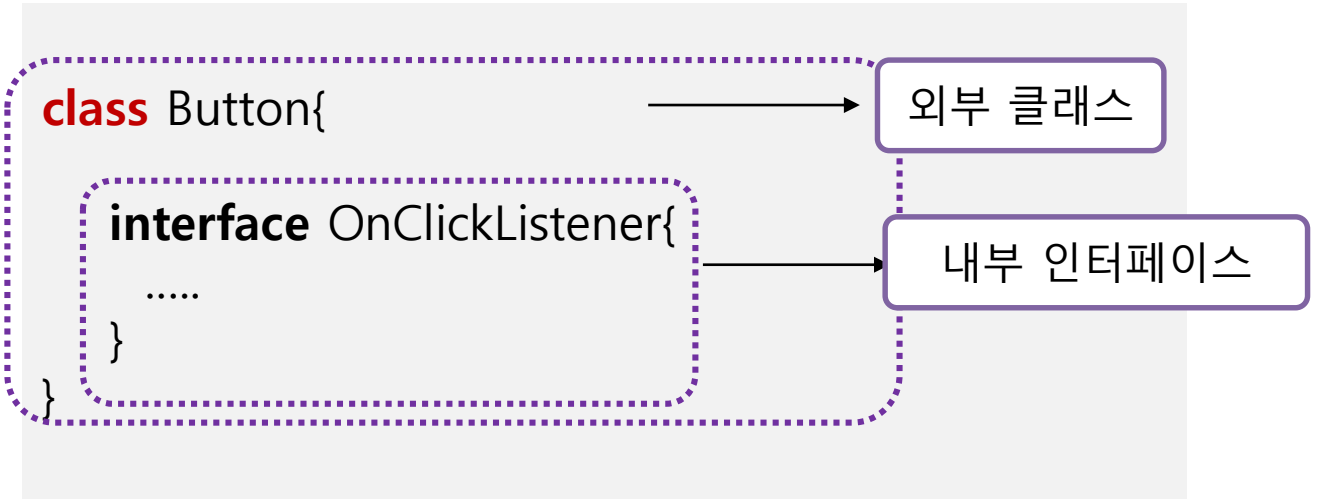
```
public class BookShelfTest {
    public static void main(String[] args) {
        //인터페이스 타입으로 객체 생성
        Queue shelfQueue = new BookShelf();

        //자료 삽입
        shelfQueue.enQueue("반응형 웹");
        shelfQueue.enQueue("혼공 Java");
        shelfQueue.enQueue("스프링부트");

        //자료의 개수
        System.out.println("현재 리스트의 개수 " + shelfQueue.getSize());

        //자료 출력
        System.out.println(shelfQueue.deQueue());
        System.out.println(shelfQueue.deQueue());
        System.out.println(shelfQueue.deQueue());
    }
}
```

내부(중첩) 인터페이스

내부 인터페이스

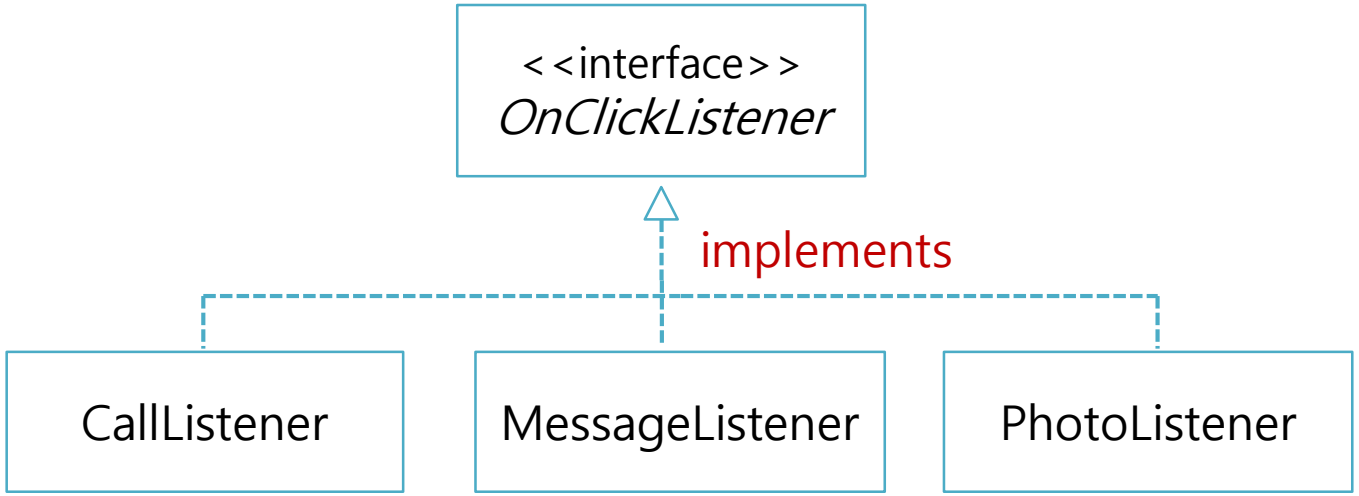
클래스의 멤버로 선언된 인터페이스를 중첩 인터페이스라 한다.

인터페이스를 클래스 내부에 선언하는 이유는 해당 클래스와 긴밀한 관계를 맺는 구현 클래스를 만들기 위함이다.



내부(중첩) 인터페이스

- 버튼을 클릭했을때 이벤트를 처리하는 객체 만들기



내부(중첩) 인터페이스

내부 인터페이스 사용 예제

```
package innerinterface;

public class Button {

    private OnClickListener listener; //인터페이스형 멤버 변수(필드)

    interface OnClickListener{ //내부 인터페이스
        public void onClick();
    }

    public void setListener(OnClickListener listener) {
        //OnClickListener 객체를 매개변수로 전달 받음
        this.listener = listener;
    }

    public void touch() {
        listener.onClick();
    }
}
```

내부(중첩) 인터페이스

내부 인터페이스 – 구현 클래스 만들기

```
public class CallListener implements Button.OnClickListener{
    //Button 클래스의 OnClickListener에 접근 -> 구현 클래스 만들기
    @Override
    public void onClick() {
        System.out.println("전화를 겁니다.");
    }

}
```

```
public class MessageListener implements Button.OnClickListener{
    //Button 클래스의 OnClickListener에 접근
    @Override
    public void onClick() {
        System.out.println("문자를 보냅니다.");
    }
}
```

내부(중첩) 인터페이스

내부 인터페이스 테스트 – 익명 객체로 구현하기

```
public class ButtonTest {

    public static void main(String[] args) {

        Button button = new Button();

        //CallListener 객체 생성
        button.setListener(new CallListener());
        button.touch();

        //MessageListener 객체 생성
        button.setListener(new MessageListener());
        button.touch();
    }
}
```

```
전화를 겁니다.
문자를 보냅니다.
사진을 찍습니다.
```