10. 인 터 페 이 스

인터페이스(interface)

인터페이스 란?(interface)

- 모든 메서드가 추상 메서드(abstract method)로 이루어진 클래스
- 형식적인 선언만 있고 구현은 없음

```
interface 인터페이스 이름{
    public static final float pi = 3.14f;
    public void add();
}
```

- 인터페이스에 선언된 모든 메서드는 public abstract 로 추상 메서드
- 인터페이스에 선언된 모든 변수는 public static final 로 상수

인터페이스 만들기

```
public interface Calc {
    double PI = 3.14;
    int ERROR = -999999999;

    int add(int num1, int num2);
    int substract(int num1, int num2);
    int times(int num1, int num2);
    int divide(int num1, int num2);
}
```

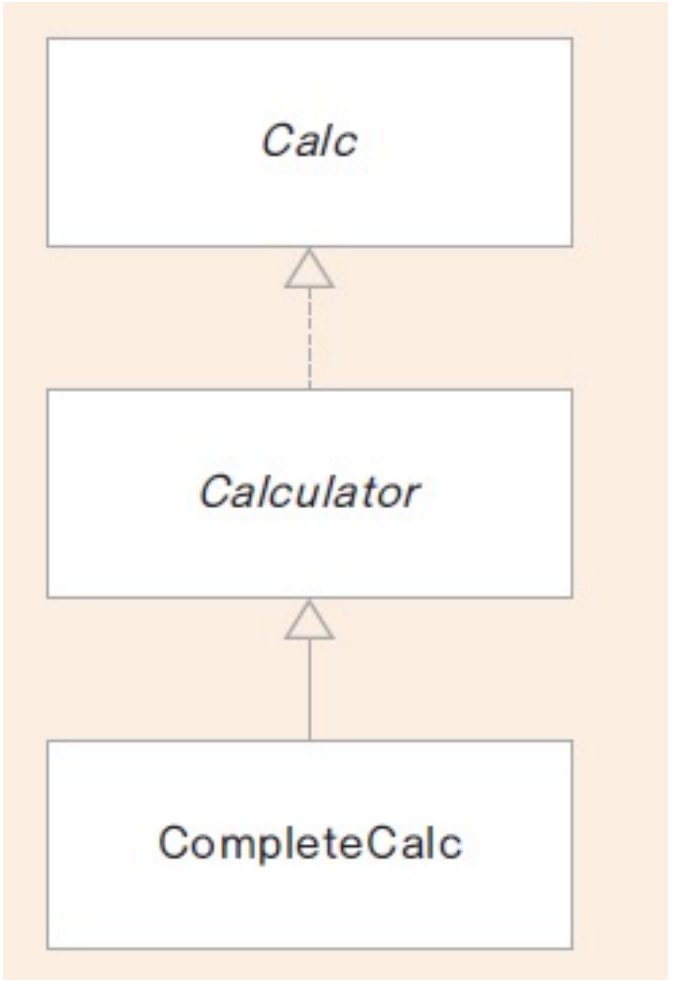
인터페이스에서 선언한 변수는 컴파일 과정에서 상수로 변환됨

인터페이스에서 선언한 메서드는 컴파일 과정에서 추상 메서드로 변환됨

클래스에서 인터페이스 구현하기

- Calc 인터페이스를
- Calculator 클래스에서 구현
- Calc의 모든 추상 메서드를 구현하지 않으면
- Calculator는 추상 클래스가 됨

- CompleteCalc클래스가 Calculator를
- 상속받은 후 모든 메서드를 구현
- CompleteCalc 는 생성가능한 클래스



calculator와 CompleteCalc 클래스

```
public abstract class Calculator implements Calc {  //추상 클래스
   @Override
   public int add(int num1, int num2) {
      return num1 + num2;
   }

   @Override
   public int substract(int num1, int num2) {
      return num1 - num2;
   }
}
```

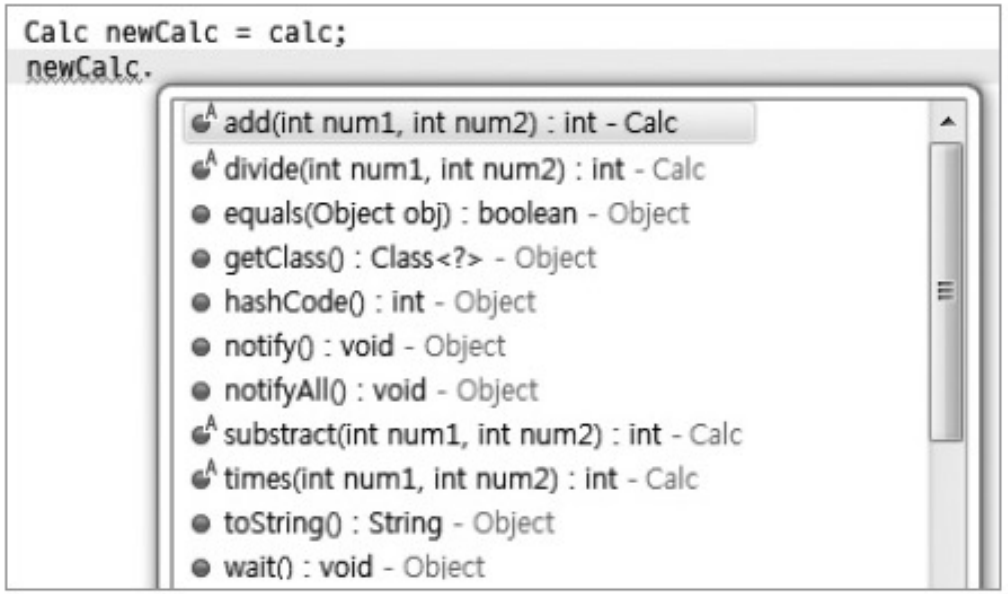
```
public class CompleteCalc extends Calculator {
   @Override
   public int times(int num1, int num2) {
      return num1 * num2;
   }

   @Override
   public int divide(int num1, int num2) {
      if(num2 != 0)
         return num1/num2;
      else
         return ICalc.ERROR;          //분모가 0인 경우에 대해 오류
   }
```

인터페이스 구현과 형 변환

- 인터페이스를 구현한 클래스는 인터페이스 형으로 선언한 변수로 형 변환 할 수 있음
- 상속에서의 형 변환과 동일한 의미
- 단 클래스 상속과 달리 구현코드가 없기 때문에 여러 인터페이스를 구현할 수 있음
- 형 변환시 사용할 수 있는 메서드는 인터페이스에 선언된 메서드만 사용할 수 있음

```
Calc calc = new CompleteCalc( );
```

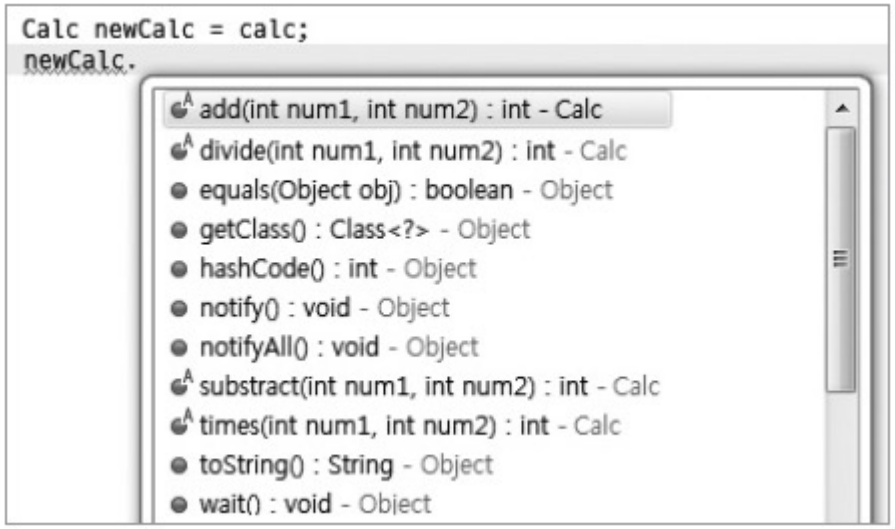
CompleteCalc 가 Calc 형 으로
대입된 경우 사용할 수 있는
메서드

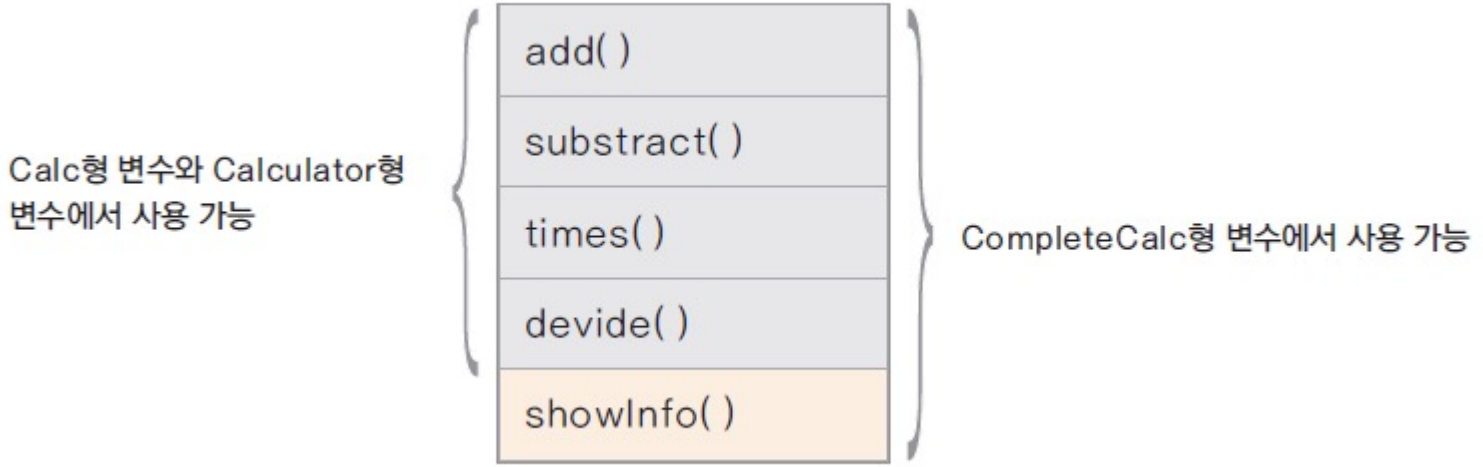


인터페이스 구현과 형 변환

```
Calc calc = new CompleteCalc( );
```

CompleteCalc 가 Calc 형 으로 대입된 경우 사용할 수 있는 메서드

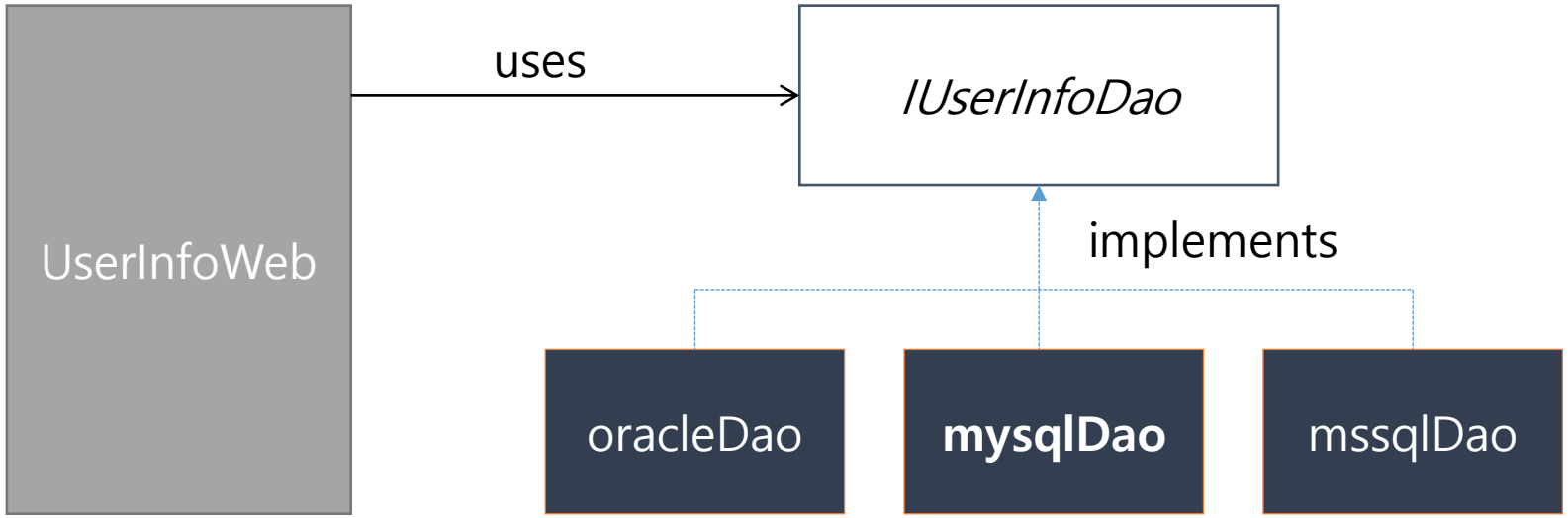




인터페이스와 다형성

- 인터페이스는 “Client Code” 와 서비스를 제공하는 “객체” 사이의 약속이다.
- 어떤 객체가 어떤 interface 타입이라 함은 그 interface가 제공하는 메서드를 구현했다는 의미임
- Client 는 어떻게 구현되었는지 상관없이 interface의 정의만을 보고 사용할 수 있음 (ex: JDBC)
- 다양한 구현이 필요한 인터페이스를 설계하는 일은 매우 중요한 일임

왜 인터페이스를 사용하는가?



UserInfoWeb 은 IUserInfoDao 에 정의된 메소드 명세만 보고 Dao를 사용할 수 있고, Dao 클래스들은 IUserInfoDao 에 정의된 메소드를 구현할 책임이 있다.

인터페이스의 요소

- 상수 : 모든 변수는 상수로 변환 됨

```
public interface Calc {
   double PI = 3.14;
   int ERROR = -999999999;
   ...
}
```

- 추상 메서드 : 모드 메서드는 추상 메서드로 구현코드가 없음
- 디폴트 메서드 : 기본 구현을 가지는 메서드, 구현 클래스에서 재정의 할 수 있음
- 정적 메서드 : 인스턴스 생성과 상관 없이 인터페이스 타입으로 사용할 수 있는 메서드
- private 메서드 : 인터페이스를 구현한 클래스에서 사용하거나 재정의 할 수 없음. 인터페이스 내부에서만 기능을 제공하기 위해 구현하는 메서드

디폴트 메서드 재정의

- Calc 인터페이스에 디폴트 메서드 정의

```
public interface Calc {
   ...
   default void description( ) {
      System.out.println("정수 계산기를 구현합니다");
   }
}
```

- CompleteCalc 에서 재정의 하기

```
public class CompleteCalc extends Calculator {
   ...
   @Override
   public void description( ) {
     //TODO Auto-generated method stub
      super.description( );
   }
}
```

디폴트 메서드 description( )을 CompleteCalc 클래스에서 원하는 기능으로 재정의

정적 메서드 사용하기

- **static 키워드로 정적 메서드 구현**

```
public interface Calc {
   ...

   static int total(int[] arr) {
      int total = 0;

      for(int i : arr) {
         total += i;
      }
      return total;
   }
}
```

인터페이스에 정적 메서드 total( ) 구현

- **인터페이스 이름으로 정적 메서드 호출**

```
int[] arr = {1, 2, 3, 4, 5};
System.out.println(Calc.total(arr));
}
```

정적 메서드 사용하기

private 메서드

- 인터페이스 내부에 private
- 혹은 private static 으로
- 선언한 메서드 구현
- private static은 정적 메서드에서
- 사용가능

```
public interface Calc {
 ...
 default void description( ) {
  System.out.println("정수 계산기를 구현합니다");
 myMethod( );
 }

 static int total(int[] arr) {
    int total = 0;

    for(int i: arr){
    total += i;
    }
   myStaticMethod( );
    return total;
 }

 private void myMethod( ) {
    System.out.println("private 메서드입니다.");
 }

 private static void myStaticMethod( ) {
    System.out.println("private static 메서드입니다.");
 }
}
```

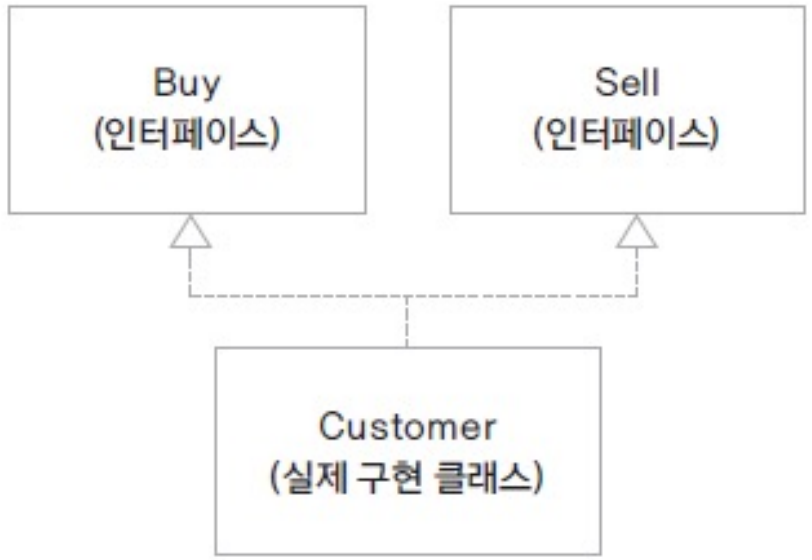
디폴트 메서드에서 private 메서드 호출

정적 메서드에서 private static 메서드 호출

private 메서드

private static 메서드

두 개의 인터페이스 구현하기




```
package interfaceex;

public interface Buy {
   void buy( );
}
```

```
package interfaceex;

public interface Sell {
   void sell( );
}
```

두 인터페이스를 구현한 클래스

```
package interfaceex;

public class Customer implements Buy, Sell {
   @Override
   public void sell( ) {
      System.out.println("구매하기");
   }

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

Customer 클래스는 Buy와 Sell 인터페이스를 모두 구현함

구현한 클래스 사용하기

```
public class CustomerTest {
   public static void main(String[] args) {
      Customer customer = new Customer( );

      Buy buyer = customer;
      buyer.buy( );

      Sell seller = customer;
      seller.sell( );

      if(seller instanceof Customer) {
         Customer customer2 = (Customer)seller;
         customer2.buy( );
         customer2.sell( );
      }
   }
}
```

Customer 클래스형인 customer를 Buy 인터페이스형인 buyer에 대입하여 형 변환. buyer는 Buy 인터페이스의 메서드만 호출 가능

Customer 클래스형인 customer를 Sell 인터페이스형인 seller에 대입하여 형 변환. seller는 Sell 인터페이스의 메서드만 호출 가능

seller를 하위 클래스형인 Customer로 다시 형 변환

두 인터페이스의 디폴트 메서드가 중복되는 경우

```java
package interfaceex;

public interface Buy {
   void buy( );

   default void order( ) {
      System.out.println("구매 주문");
   }
}
```

```java
package interfaceex;

public interface Sell {
   void sell( );

   default void order( ) {
      System.out.println("판매 주문");
   }
}
```

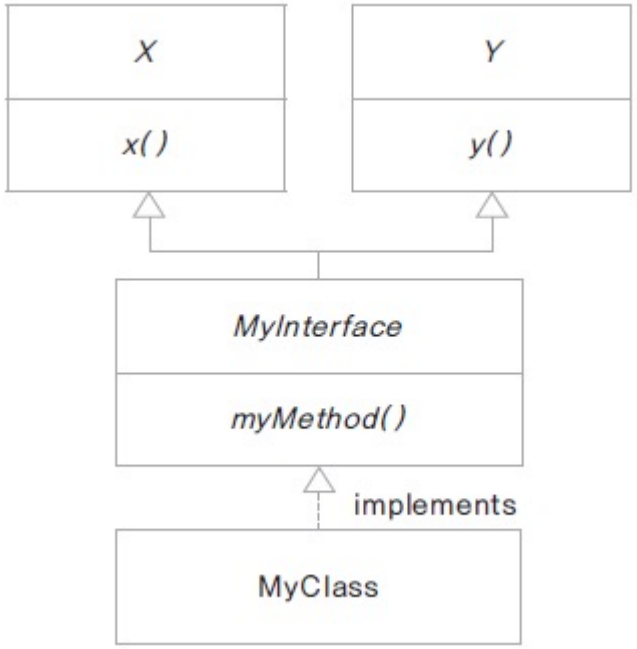
구현한 클래스에서 중복된 디폴트 메서드를 재정의 함

```java
public class Customer implements Buy, Sell {
   ...
   @Override
   public void order( ) {
      System.out.println("고객 판매 주문");
   }
}
```

디폴트 메서드 order( )를
Customer 클래스에서 재정의함

인터페이스 상속

- 인터페이스 간에도 상속이 가능
- 구현코드의 상속이 아니므로 형 상속(type inheritance) 라고 함




```
public interface MyInterface extends X, Y {
   void myMethod( );
}
```

인터페이스 여러 개를 상속받을 수 있음

인터페이스 상속

- 여러 인터페이스를 상속한 인터페이스(MyInterface)를 구현하는 클래스는 선언된 모든 추상 메서드를 구현 해야 함

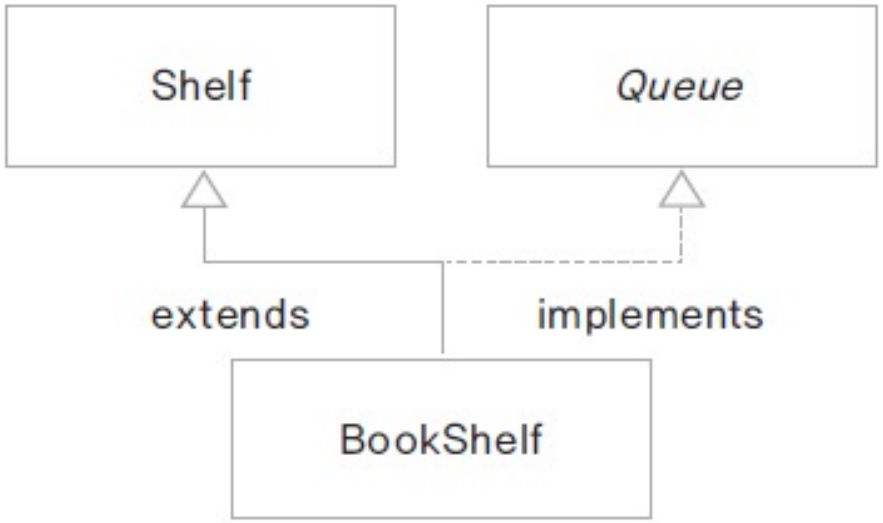
```
public class MyClass implements MyInterface {
    @Override
    public void x( ) {
        System.out.println("x( )");
    }

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

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

X 인터페이스에서 상속받은 x( ) 메서드 구현

Y 인터페이스에서 상속받은 y( ) 메서드 구현

MyInterface 인터페이스의 myMethod( ) 메서드 구현

인터페이스 구현과 클래스 상속 함께 사용하기



실제 프레임 워크(스프링, 안드로이드)를
사용하면 클래스를 상속 받고
여러 인터페이스를 구현하는 경우가
종종 있음

```
public class BookShelf extends Shelf implements Queue {
    @Override
    public void enQueue(String title) {
        shelf.add(title);
    }

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

    @Override
    public int getSize( ) {
        return getCount( );
    }
}
```

배열에 요소 추가

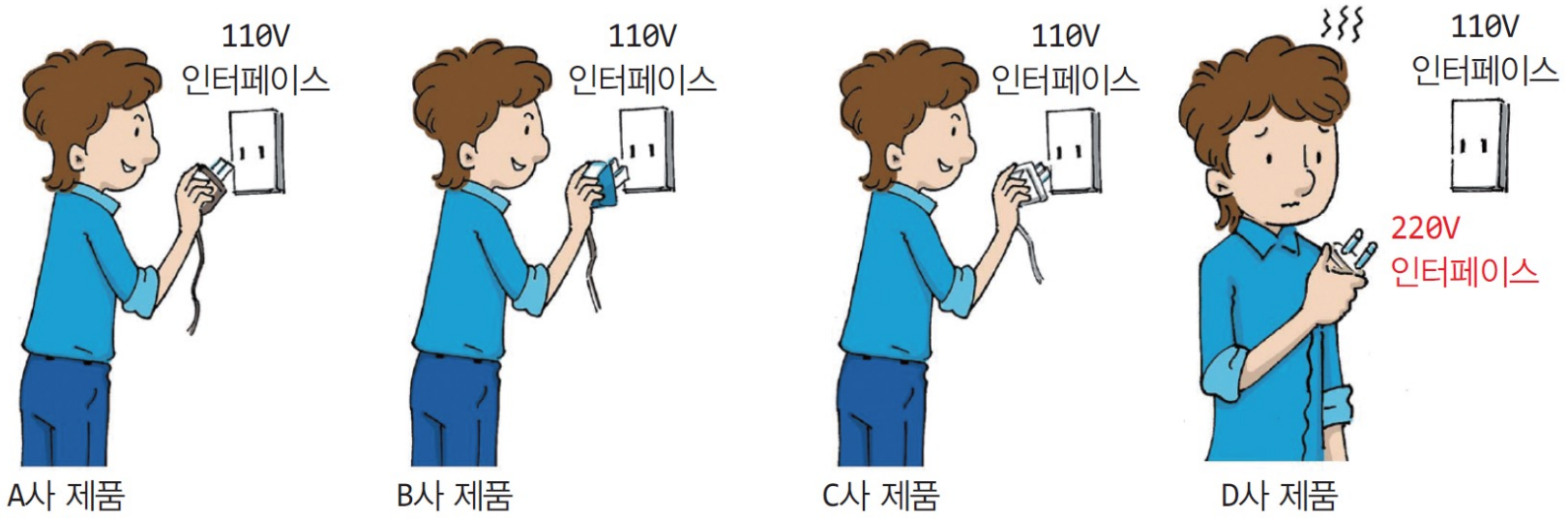
맨 처음 요소를 배열에서 삭제하고 반환

배열 요소 개수 반환

감사합니다.

끝

실세계의 인터페이스



정해진 규격(인터페이스)에 맞기만 하면 연결 가능.
각 회사마다 구현 방법은 다름

정해진 규격(인터페이스)에 맞지 않으면 연결 불가

자바의 인터페이스

- **자바의 인터페이스**
  - 클래스가 구현해야 할 메소드들이 선언되는 추상형
  - 인터페이스 선언
    - **interface** 키워드로 선언
    - Ex) public **interface** SerialDriver {…}
- **자바 인터페이스에 대한 변화**
  - Java 7까지
    - 인터페이스는 상수와 추상 메소드로만 구성
  - Java 8부터
    - 상수와 추상메소드 포함
    - default 메소드 포함 (Java 8)
    - private 메소드 포함 (Java 9)
    - static 메소드 포함 (Java 9)
  - 여전히 인터페이스에는 **필드(멤버 변수) 선언 불가**

자바 인터페이스 사례

```
interface PhoneInterface { // 인터페이스 선언
    public static final int TIMEOUT = 10000; // 상수 필드 public static final 생략 가능
    public abstract void sendCall(); // 추상 메소드 public abstract 생략 가능
    public abstract void receiveCall();  // 추상 메소드 public abstract 생략 가능
    public default void printLogo() { // default 메소드 public 생략 가능
        System.out.println("** Phone **");
    };  // 디폴트 메소드
}
```

인터페이스의 구성 요소들의 특징

- **인터페이스의 구성 요소들**
  - 상수
    - public만 허용, public static final 생략
  - 추상 메소드
    - public abstract 생략 가능
  - default 메소드
    - 인터페이스에 코드가 작성된 메소드
    - 인터페이스를 구현하는 클래스에 자동 상속
    - public 접근 지정만 허용. 생략 가능
  - private 메소드
    - 인터페이스 내에 메소드 코드가 작성되어야 함
    - 인터페이스 내에 있는 다른 메소드에 의해서만 호출 가능
  - static 메소드
    - public, private 모두 지정 가능. 생략하면 public

자바 인터페이스의 전체적인 특징

- **인터페이스의 객체 생성 불가**


```
new PhoneInterface(); // 오류. 인터페이스 PhoneInterface 객체 생성 불가
```

- **인터페이스 타입의 레퍼런스 변수 선언 가능**

```
PhoneInterface galaxy; // galaxy는 인터페이스에 대한 레퍼런스 변수
```

- **인터페이스 구현**
  - 인터페이스를 상속받는 클래스는 인터페이스의 모든 추상 메소드 반드시 구현
- **다른 인터페이스 상속 가능**
- **인터페이스의 다중 상속 가능**

인터페이스 구현

- **인터페이스의 추상 메소드를 모두 구현한 클래스 작성**
  - implements 키워드 사용
  - 여러 개의 인터페이스 동시 구현 가능
- **인터페이스 구현 사례**
  - PhoneInterface 인터페이스를 구현한 SamsungPhone 클래스

```
class SamsungPhone implements PhoneInterface { // 인터페이스 구현
   // PhoneInterface의 모든 메소드 구현
   public void sendCall() { System.out.println("띠리리리링"); }
   public void receiveCall() { System.out.println("전화가 왔습니다."); }

   // 메소드 추가 작성
   public void flash() { System.out.println("전화기에 불이 켜졌습니다."); }
}
```

  - SamsungPhone 클래스는 PhoneInterface의 default 메소드상속

예제 8 인터페이스 구현

PhoneInterface 인터페이스를 구현하고 flash() 메소드를 추가한 SamsungPhone 클래스를 작성하라.

```
** Phone **
띠리리리링
전화가 왔습니다.
전화기에 불이 켜졌습니다.
```

```
interface PhoneInterface { // 인터페이스 선언
    final int TIMEOUT = 10000; // 상수 필드 선언
    void sendCall(); // 추상 메소드
    void receiveCall(); // 추상 메소드
    default void printLogo() { // default 메소드
        System.out.println("** Phone **");
    }
}
class SamsungPhone implements PhoneInterface { // 인터페이스 구현
    // PhoneInterface의 모든 추상 메소드 구현
    @Override
    public void sendCall() {
        System.out.println("띠리리리링");
    }
    @Override
    public void receiveCall() {
        System.out.println("전화가 왔습니다.");
    }

    // 메소드 추가 작성
    public void flash() { System.out.println("전화기에 불이 켜졌습니다."); }
}
public class InterfaceEx {
    public static void main(String[] args) {
        SamsungPhone phone = new SamsungPhone();
        phone.printLogo();
        phone.sendCall();
        phone.receiveCall();
        phone.flash();
    }
}
```

인터페이스 상속

- **인터페이스가 다른 인터페이스 상속**
  - extends 키워드 이용

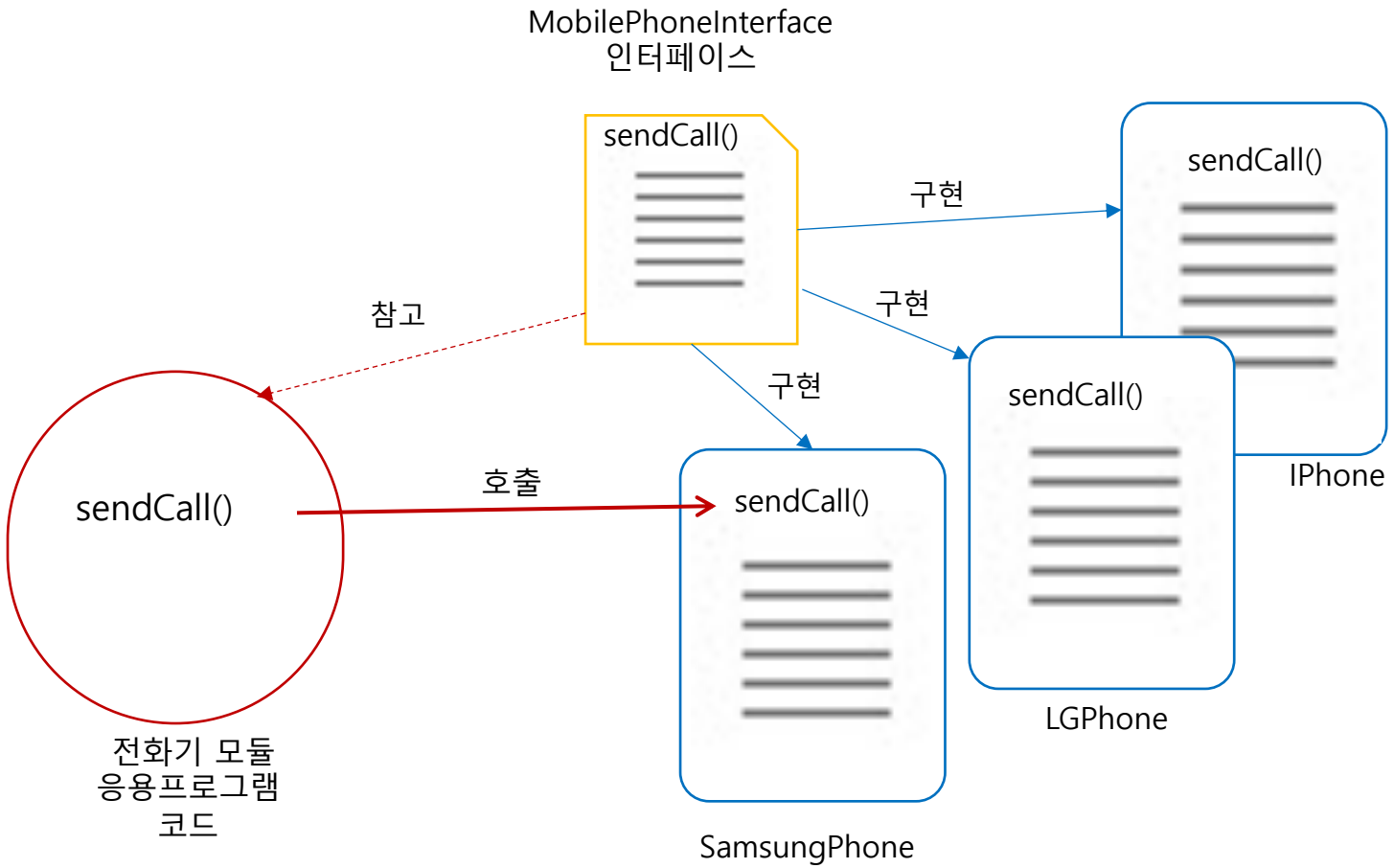
```
interface MobilePhoneInterface extends PhoneInterface {
   void sendSMS();      // 새로운 추상 메소드 추가
   void receiveSMS();   // 새로운 추상 메소드 추가
}
```

  - 다중 인터페이스 상속

```
interface MP3Interface {
   void play(); // 추상 메소드
   void stop(); // 추상 메소드
}

interface MusicPhoneInterface extends MobilePhoneInterface, MP3Interface {
   void playMP3RingTone(); // 새로운 추상 메소드 추가
}
```

인터페이스의 목적

인터페이스는 스펙을 주어 클래스들이 그 기능을 서로 다르게 구현할 수 있도록 하는 클래스의 규격 선언이며, 클래스의 다형성을 실현하는 도구이다



다중 인터페이스 구현

클래스는 하나 이상의 인터페이스를 구현할 수 있음

```
interface AIInterface {
    void recognizeSpeech(); // 음성 인식
    void synthesizeSpeech(); // 음성 합성
}

class AIPhone implements MobilePhoneInterface, AIInterface { // 인터페이스 구현
    // MobilePhoneInterface의 모든 메소드를 구현한다.
    public void sendCall() { ... }
    public void receiveCall() { ... }
    public void sendSMS() { ... }
    public void receiveSMS() { ... }

    // AIInterface의 모든 메소드를 구현한다.
    public void recognizeSpeech() { ... } // 음성 인식
    public void synthesizeSpeech() { ... } // 음성 합성

    // 추가적으로 다른 메소드를 작성할 수 있다.
    public int touch() { ... }
}
```

클래스에서 인터페이스의 메소드를 구현할 때 public을 생략하면 오류 발생

예제 9 : 인터페이스를 구현하고 동시에 클래스를 상속받는 사례

```
interface PhoneInterface { // 인터페이스 선언
    final int TIMEOUT = 10000; // 상수 필드 선언
    void sendCall(); // 추상 메소드
    void receiveCall(); // 추상 메소드
    default void printLogo() { // default 메소드
        System.out.println("** Phone **");
    }
}
interface MobilePhoneInterface extends PhoneInterface {
    void sendSMS();
    void receiveSMS();
}
interface MP3Interface { // 인터페이스 선언
    public void play();
    public void stop();
}
class PDA { // 클래스 작성
    public int calculate(int x, int y) {
        return x + y;
    }
}
// SmartPhone 클래스는 PDA를 상속받고,
// MobilePhoneInterface와 MP3Interface 인터페이스에 선언된 추상 메소드를 모두 구현한다.
class SmartPhone extends PDA implements
MobilePhoneInterface, MP3Interface {
    // MobilePhoneInterface의 추상 메소드 구현
    @Override
    public void sendCall() {
        System.out.println("따르릉따르릉~~");
    }
    @Override
    public void receiveCall() {
        System.out.println("전화 왔어요.");
    }
    @Override
    public void sendSMS() {
        System.out.println("문자갑니다.");
    }
    @Override
    public void receiveSMS() {
        System.out.println("문자왔어요.");
    }
    // MP3Interface의 추상 메소드 구현
    @Override
    public void play() {
        System.out.println("음악 연주합니다.");
    }
    @Override
    public void stop() {
        System.out.println("음악 중단합니다.");
    }
    // 추가로 작성한 메소드
    public void schedule() {
        System.out.println("일정 관리합니다.");
    }
}
public class InterfaceEx {
    public static void main(String [] args) {
    SmartPhone phone = new SmartPhone();
    phone.printLogo();
    phone.sendCall();
    phone.play();
    System.out.println("3과 5를 더하면 " +
        phone.calculate(3,5));
    phone.schedule();
    }
}
```

```
** Phone **
따르릉따르릉~~
음악 연주합니다.
3과 5를 더하면 8
일정 관리합니다.
```

추상 클래스와 인터페이스 비교

- **유사점**
  - 객체를 생성할 수 없고, 상속을 위한 슈퍼 클래스로만 사용
  - 클래스의 다형성을 실현하기 위한 목적
- **다른 점**

비교	목적	구성
추상 클래스	추상 클래스는 서브 클래스에서 필요로 하는 대부분의 기능을 구현하여 두고 서브 클래스가 상속받아 활용할 수 있도록 하되, 서브 클래스에서 구현할 수밖에 없는 기능만을 추상 메소드로 선언하여, 서브 클래스에서 구현하도록 하는 목적(다형성)	• 추상 메소드와 일반 메소드 모두 포함 • 상수, 변수 필드 모두 포함
인터페이스	인터페이스는 객체의 기능을 모두 공개한 표준화 문서와 같은 것으로, 개발자에게 인터페이스를 상속받는 클래스의 목적에 따라 인터페이스의 모든 추상 메소드를 만들도록 하는 목적(다형성)	• 변수 필드(멤버 변수)는 포함하지 않음 • 상수, 추상 메소드, 일반 메소드, `default` 메소드, `static` 메소드 모두 포함 • `protected` 접근 지정 선언 불가 • 다중 상속 지원