

인터페이스와 추상클래스

17-1.

인터페이스의 기본과 그 의미

추상 메소드만 담고 있는 인터페이스

```
interface Printable {  
    public void print(String doc);    // 추상 메소드  
}
```

인터페이스의 정의! 메소드의 몸체를 갖지 않는다.

따라서 인스턴스 생성 불가! 참조변수 선언 가능!

```
class Printer implements Printable {  
    public void print(String doc) {  
        System.out.println(doc);  
    }  
}
```

인터페이스를 구현하는 클래스!

구현하는 메소드와 추상 메소드 사이에도 메소드 오버라이딩 관계 성립, 따라서 @Override 붙일 수 있음

인터페이스형 참조변수 선언 가능

```
Printable prn = new Printer();  
prn.print("Hello");
```

상속과 구현

```
class Robot extends Machine implements Movable, Runnable {...}
```

Robot 클래스는 Machine 클래스를 상속한다.

이렇듯 상속과 구현 동시에 가능!

Robot 클래스는 Movable과 Runnable 인터페이스를 구현한다.

이렇듯 둘 이상의 인터페이스 구현 가능!

인터페이스의 본질적 의미



출력 가능



삼성 프린터

```
class SPrinterDriver implements Printable {  
    @Override  
    public void print(String doc) {...}  
}
```

출력 가능



LG 프린터

```
class LPrinterDriver implements Printable {  
    @Override  
    public void print(String doc) {...}  
}
```

MS에서 제공하는 인터페이스

```
interface Printable {  
    public void print(String doc);  
}
```

Printer Driver 관련 예제

```
interface Printable { // MS가 정의하고 제공한 인터페이스
    public void print(String doc);
}
```

```
class SPrinterDriver implements Printable {
    @Override
    public void print(String doc) {
        System.out.println("From Samsung printer");
        System.out.println(doc);
    }
}
```

```
class LPrinterDriver implements Printable {
    @Override
    public void print(String doc) {
        System.out.println("From LG printer");
        System.out.println(doc);
    }
}
```

```
public static void main(String[] args) {
    String myDoc = "This is a report about...";

    // 삼성 프린터로 출력
    Printable prn = new SPrinterDriver();
    prn.print(myDoc);
    System.out.println();

    // LG 프린터로 출력
    prn = new LPrinterDriver();
    prn.print(myDoc);
}
```

17-2. 인터페이스의 문법 구성과 추상 클래스

인터페이스에 선언되는 메소드와 변수

```
interface Printable {  
    public void print(String doc);    // 추상 메소드  
}
```

```
interface Printable {  
    public static final int PAPER_WIDTH = 70;  
    public static final int PAPER_HEIGHT = 120;  
    public void print(String doc);  
}
```


인터페이스간 상속: 문제 상황의 제시



출력 가능



삼성 프린터

```
class SPrinterDriver implements Printable {  
    @Override  
    public void print(String doc) {...}  
} // 이 클래스에서 printCMYK 메소드 구현해야 함
```

출력 가능



LG 프린터

```
class LPrinterDriver implements Printable {  
    @Override  
    public void print(String doc) {...}  
} // 이 클래스에서 printCMYK 메소드 구현해야 함
```

```
interface Printable {  
    void print(String doc);  
    void printCMYK(String doc);  
}
```

} 컬러 출력 위한 메소드 추가되면?
시스템 전체에 문제 발생

인터페이스를 구현하는 클래스는 해당 인터페이스의 모든 추상 메소드를 구현해야 한다. 그래야 인스턴스 생성 가능!

제시한 문제의 해결책: 인터페이스의 상속

```
interface Printable {  
    void print(String doc);  
}  
  
    인터페이스간 상속도 extends로 표현  
interface ColorPrintable extends Printable {  
    void printCMYK(String doc);  
}  
  
class SPrinterDriver implements Printable {  
    ...  
}    // 기존 클래스 수정할 필요 없음
```

```
class Prn909Drv implements ColorPrintable {  
    @Override  
    public void print(String doc) {    // 흑백 출력  
        System.out.println("black & white ver");  
        System.out.println(doc);  
    }  
  
    @Override  
    public void printCMYK(String doc) {    // 컬러 출력  
        System.out.println("CMYK ver");  
        System.out.println(doc);  
    }  
}
```



컬러 프린터
드라이버

인터페이스의 디폴트 메소드: 문제 상황의 제시

<< interface >>
... i1

<< interface >>
... i2

<< interface >>
... i3

...

<< interface >>
... i256

총 256개의 인터페이스가 존재하는 상황에서 모든 인터페이스에 다음 추상 메소드를 추가해야 한다면?

```
void printCMYK(String doc);
```

인터페이스간 상속?

물론 인터페이스간 상속으로 문제 해결 가능하다. 다만 인터페이스의 수가 256개 늘어날 뿐이다.

문제 상황의 해결책: 인터페이스의 디폴트 메소드

<< interface >>
... i1

<< interface >>
... i2

<< interface >>
... i3

. . .

<< interface >>
... i256

총 256개의 인터페이스가 존재하는 상황에서 모든 인터페이스에 다음 추상 메소드를 추가해야 한다면?

```
void printCMYK(String doc);
```

다음 디폴트 메소드로 이 문제를 해결하면 인터페이스의 수가 늘어나지 않는다.

```
default void printCMYK(String doc) { }    // 디폴트 메소드
```

디폴트 메소드의 효과

```
interface Printable {  
    void print(String doc);  
}
```

인터페이스의 교체

```
interface Printable {  
    void print(String doc);  
    default void printCMYK(String doc) { }  
}
```

```
class SPrinterDriver implements Printable {  
    @Override  
    public void print(String doc) {...}  
}
```

기존에 정의된 클래스:

인터페이스 교체로 인해 코드 수정 필요 없다.

```
class Prn909Drv implements Printable {  
    @Override  
    public void print(String doc) {...}  
  
    @Override  
    public void printCMYK(String doc) {...}  
}
```

새로 정의된 클래스

인터페이스의 static 메소드

“인터페이스에도 static 메소드를 정의할 수 있다.”

“그리고 인터페이스의 static 메소드 호출 방법은 클래스의 static 메소드 호출 방법과 같다.”

```
interface Printable {  
    static void printLine(String str) {  
        System.out.println(str);  
    }  
  
    default void print(String doc) {  
        printLine(doc); // 인터페이스의 static 메소드 호출  
    }  
}
```

인터페이스 대상의 instanceof 연산

```
if(ca instanceof Cake) ....
```

Cake는 클래스의 이름도, 인터페이스의 이름도 될 수 있다.

ca가 참조하는 인스턴스를 Cake형 참조변수로 참조할 수 있으면 true 반환!

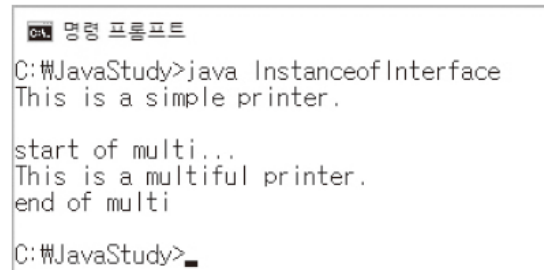
교재에서는 이를 다음과 같이 설명하고 있다. (같은 의미, 표현만 다를 뿐)

ca가 참조하는 인스턴스가 Cake를 직접 혹은 간접적으로 구현한 클래스의 인스턴스인 경우 true 반환!

인터페이스 대상 instanceof 연산의 예

```
interface Printable {  
    void printLine(String str);  
}  
  
class SimplePrinter implements Printable {  
    public void printLine(String str) {  
        System.out.println(str);  
    }  
}  
  
class MultiPrinter extends SimplePrinter {  
    public void printLine(String str) {  
        super.printLine("start of multi...");  
        super.printLine(str);  
        super.printLine("end of multi");  
    }  
}
```

```
public static void main(String[] args) {  
    Printable prn1 = new SimplePrinter();  
    Printable prn2 = new MultiPrinter();  
  
    if(prn1 instanceof Printable)  
        prn1.printLine("This is a simple printer.");  
    System.out.println();  
  
    if(prn2 instanceof Printable)  
        prn2.printLine("This is a multiful printer.");  
}
```



A screenshot of a Windows command prompt window titled "명령 프롬프트". The window shows the execution of a Java program. The command entered is "C:\JavaStudy>java InstanceofInterface". The output displayed is "This is a simple printer.", followed by a blank line, then "start of multi...", "This is a multiful printer.", and "end of multi". The prompt "C:\JavaStudy>" is visible at the bottom.

인터페이스의 또 다른 용도: Maker 인터페이스

```
interface Upper { } // 마커 인터페이스
interface Lower { } // 마커 인터페이스
```

```
interface Printable {
    String getContents();
}
```

Lower로 표시해 둔다면?

```
class Report implements Printable, Upper {
    String cons;

    Report(String cons) {
        this.cons = cons;
    }
    public String getContents() {
        return cons;
    }
}
```

```
public void printContents(Printable doc) {
    if(doc instanceof Upper) {
        System.out.println((doc.getContents()).toUpperCase());
    }
    else if(doc instanceof Lower) {
        System.out.println((doc.getContents()).toLowerCase());
    }
    else
        System.out.println(doc.getContents());
}
```

클래스에 특정 표시를 해 두기 위한 목적으로 정의된 인터페이스를 마커 인터페이스라 한다. 마커 인터페이스에는 구현해야 할 메소드가 없는 경우가 흔하다.

추상 클래스

```
public abstract class House {    // 추상 클래스
    public void methodOne() {
        System.out.println("method one");
    }

    public abstract void methodTwo();    // 추상 메소드
}
```

하나 이상의 추상 메소드를 지니는 클래스를 가리켜 추상 클래스라 한다.

그리고 추상 클래스를 대상으로는 인스턴스 생성이 불가능하다. 물론 참조변수 선언은 가능하다.