

COMP217

JAVA Programming

Summer 2018

Day 12

Interface, polymorphism

Textbook:


Power Java 2: Chapter 12

Goals

- 이번 주에 배우게 될 내용 :
 - 다형성
 - 인터페이스
- 파워 자바 12장(인터페이스와 다형성)

두산백과

다형성

[polymorphism , 多形性]

요약 같은 종의 생물이지만, 모습이나 고유한 특징이 다양한 성질을 말한다.

생물은 본래 동일종이라도 완전히 일치하는 개체는 거의 없으므로, 이 말은 상대적으로 현저한 차이가 있을 경우에 한해서 사용한다. 다만, 암수의 성별에 따른 2차성징(二次性徵)의 차이에는 쓰지 않는다. 사회성곤충에서 많은 계급적인 다형성이 보이고, 또 곤충의 변태단계는 발생적 다형성이라고 할 수 있다.

업 캐스팅(예제)

```
class Pet{...}
```

```
class Dog extends Pet{...} //강아지 is a 애완동물
```

```
class Cat extends Pet{...} //고양이 is a 애완동물
```

```
Pet p1 = new Pet(); // 가능
```

```
Pet p2 = new Dog(); // 강아지도 애완동물이기 때문에 가능
```

```
Pet p3 = new Cat(); // 고양이도 애완동물이기 때문에 가능
```

```
Cat p4 = new Pet(); // ?
```

업 캐스팅

- 업 캐스팅

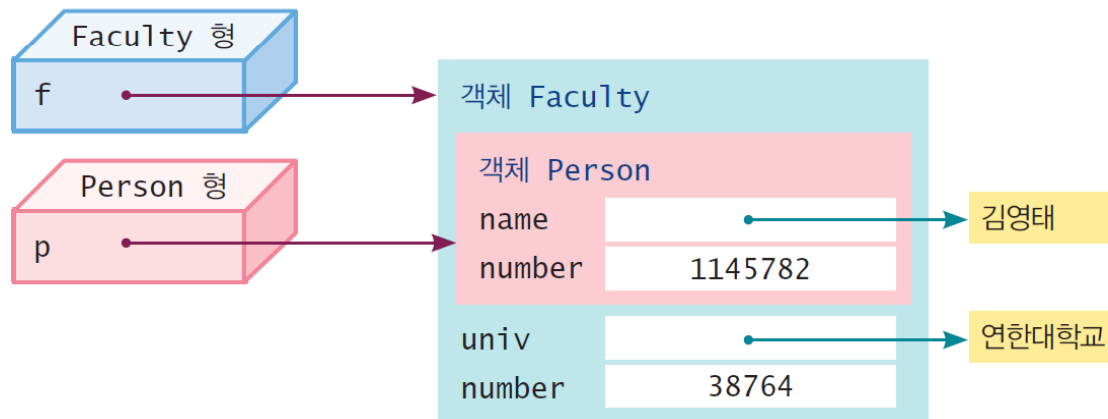
- 하위 객체는 상위 클래스형 변수에 대입이 가능, 상위로의 자료형 변환
 - 업 캐스팅은 하위인 교직원은 상위인 사람이라는 개념이 성립

- 업 캐스팅의 제약

- 업 캐스팅된 변수로는 하위 객체의 멤버를 참조할 수 없는 제약
 - Faculty 형 변수 f로는 접근 지정자만 허용하면 모든 멤버를 접근 가능
 - Person 형 변수 p로는 Person의 멤버인 name과 number만 접근이 가능

클래스자료형 변수 = 하위_클래스_자료형의_객체_또는_변수;

```
Faculty f = new Faculty("김영태", 1145782, "연한대학교", 38764);  
Person p = f;
```



클래스 상속 관계

```
class Person  
name: 이름  
number: 주민번호  
↓  
class Faculty  
univ: 대학이름  
number: 대학ID번호  
↓  
class Staff  
division: 소속부서
```

업 캐스팅 예제

UpCasting.java

```
01 package inheritance.typecast;
02
03 public class UpCasting {
04     public static void main(String[] args) {
05         Person she = new Person("이소라", 2056432);
06         System.out.println(she.name + " " + she.number);
07
08         Faculty f = new Faculty("김영태", 1145782, "연한대학교", 38764);
09         Person p = f;           //업캐스팅
10         System.out.print(p.name + " " + p.number + " ");
11         //System.out.print(p.univ); //참조 불가능
12         System.out.println(f.name + " " + ((Person) f).number);
13         System.out.println(f.univ + " " + f.number);
14
15         Staff s = new Staff("김상기", 1187543, "강서대학교", 3456);
16         s.division = "교학처";
17         Person pn = s;          //업캐스팅
18         Faculty ft = s;         //업캐스팅
19         System.out.print(pn.name + " " + pn.number + " ");
20         System.out.print(ft.univ + " " + ft.number + " ");
21         System.out.println(s.division);
22     }
23 }
```

결과

이소라 2056432

김영태 1145782 연한대학교 38764

김상기 1187543 강서대학교 3456 교학처

다운 캐스팅

- 상위 클래스 형을 하위 클래스 형으로 변환
- 다운 캐스팅은 반드시 명시적인 **형변환 연산자 (하위 클래스)**가 필요
 - 만일 형변환 연산자가 없으면 컴파일 시간에 오류 발생

클래스자료형 변수 = (클래스자료형) 상위_클래스_자료형의_객체_또는_변수;

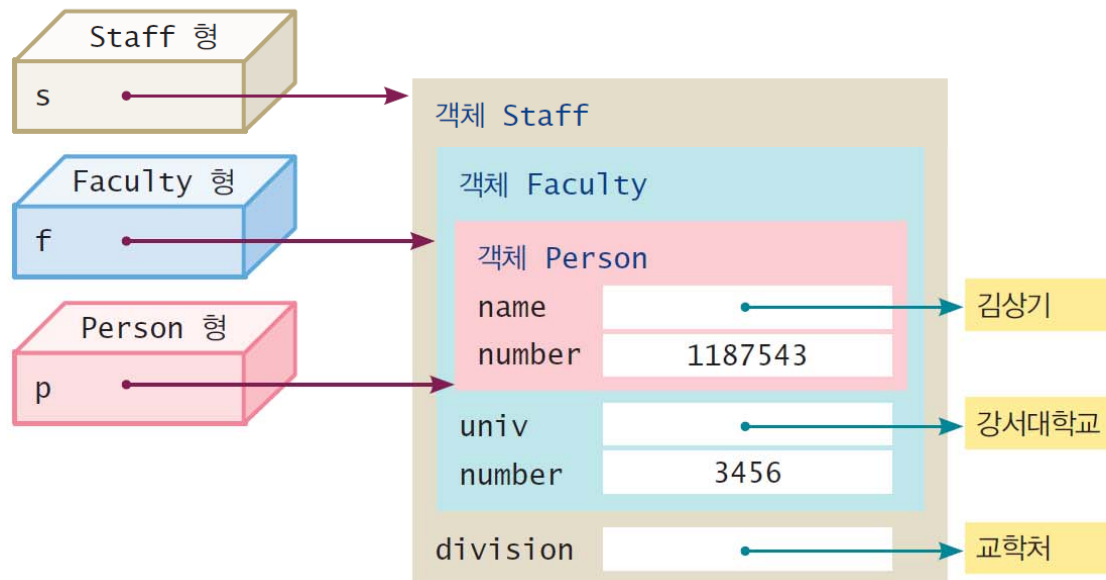
```
Person p = new Staff("김상기", 1187543, "강서대학교", 3456);
```

```
Staff s = (Staff) p;
```

```
Staff s = p;
```

Type mismatch cannot convert from Person to Staff

```
s.division = "교학처";
```



클래스 개층 정보

```
class Person
```

name: 이름

number: 주민번호

```
class Faculty
```

univ: 대학 이름

number: 대학 ID 번호

```
class Staff
```

division: 소속부서

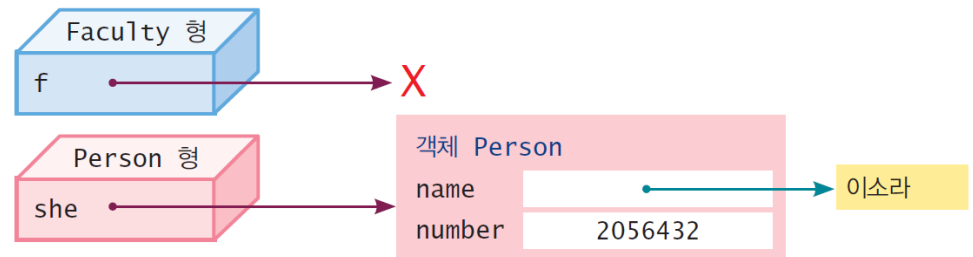
다운 캐스팅의 실행 오류

- 컴파일 시간
 - 상속 관계만 성립하면 다운 캐스팅은 가능
- 실행 시간
 - 실제 객체가 할당되지 않았다면 실행 시간에 오류가 발생

DownCasting.java

```
01 package inheritance.typecast;
02
03 public class DownCasting {
04     public static void main(String[] args) {
05         Person she = new Person("이소라", 2056432);
06         System.out.println(she.name + " " + she.number);
07         //Faculty f = she;           //컴파일 오류
08         //Faculty f1 = (Faculty) she; //실행 오류
09
10         Person p = new Staff("김상기", 1187543, "강서대학교", 3456);
11         //Staff s = p;               //컴파일 오류
12         Staff s = (Staff) p;
13         s.division = "교학처";
14         System.out.print(p.name + " " + p.number + " ");
15         System.out.print(s.univ + " " + s.number + " ");
16         System.out.println(s.division);
17     }
18 }
```

```
Person she = new Person("이소라", 2056432);
//Faculty f = she;           //컴파일 오류
//Faculty f = (Faculty) she; //실행 오류
```



결과

이소라 2056432

김상기 1187543 강서대학교 3456 교학처

객체 확인 연산자 instanceof

InstanceOf.java

```
01 package inheritance.typecast;
02
03 public class Instanceof {
04     public static void main(String[] args) {
05         Person she = new Person("이소라", 2056432);
06         if (she instanceof Staff) {
07             Staff st1 = (Staff) she;
08         } else {
09             System.out.println("she는 Staff 객체가 아닙니다.");
10         }
11
12         Person p = new Staff("김상기", 1187543, "강서대학교", 3456);
13         if (p instanceof Staff) {
14             System.out.println("p는 Staff 객체입니다.");
15             Staff st2 = (Staff) p;
16         }
17     }
18 }
```

```
Person she = new Person("이소라", 2056432);
if (she instanceof Staff) {
    Staff st1 = (Staff) she;
} else {
    System.out.print("she는 Staff 객체가 아닙니다. ");
}
```

사용법: 객체변수 instanceof 클래스이름

상속관계가 없는데,
instanceof 연산자를 사용할
경우 컴파일 에러

결과 she는 Staff 객체가 아닙니다.
p는 Staff 객체입니다.

다형성(polymorphism)

- 하나의 객체가 여러 개의 자료 타입을 가질 수 있는 것
- 객체들의 타입이 다르면 똑같은 메시지가 전달되더라도 서로 다른 동작을 하도록 하는데 사용됨
- 메소드의 매개 변수로 수퍼 클래스 참조 변수를 이용한다.

-> 다형성을 이용하는 전형적인 방법

매개변수를 공으로 설정
공을 상속한 야구공, 축구공,
골프공 모두 인자로 받을 수
있음.



다형성을 이용하면
뭐든지 받을 수 있지

매개변수를 동물로 둔 경우
동물을 상속받은 고양이,
강아지 모두 speak 메서드의
인자로 받을 수 있다.



그림12-5. 다형성의 개념

예제

```
01 class Shape {
02     protected int x, y;
03     public void draw() {
04         System.out.println("Shape Draw");
05     }
06 }
07
08 class Rectangle extends Shape {
09     private int width, height;
10     public void draw() {
11         System.out.println("Rectangle Draw");
12     }
13 }
14
15 class Triangle extends Shape {
16     private int base, height;
17     public void draw() {
18         System.out.println("Triangle Draw");
19     }
20 }
21
```

각 도형들은 2차원 공간에서 도형의 위치를 나타내는 기준 점 (x, y)을 가진다. 이것은 모든 도형에 공통적인 속성이므로 부모 클래스인 Shape에 저장한다. 또한 각 도형들을 화면에 그리는 멤버 함수 draw()가 필요하다. 이것도 모든 도형에 필요한 기능이므로 부모 클래스 Shape에 정의하도록 하자. 하지만 아직 특정한 도형이 결정되지 않았으므로 draw()에서 하는 일은 없다.

이어서 Shape에서 상속받아서 사각형을 나타내는 클래스 Rectangle을 정의하여 보자. Rectangle은 추가적으로 width와 height 변수를 가진다. Shape 클래스의 draw()를 사각형을 그리도록 재정의한다. 물론 실제 그래픽은 아직까지 사용할 수 없으므로 화면에 사각형을 그린다는 메시지만을 출력한다.

서브 클래스인 Triangle을 Shape 클래스에서 상속받아 만든다.

예제

```
22 class Circle extends Shape {
23     private int radius;
24     public void draw() {
25         System.out.println("Circle Draw");
26     }
27 }
```

서브 클래스인 Rectangle을 Shape 클래스에서 상속받아 만든다.

```
28
29 public class ShapeTest {
30     private static Shape arrayOfShapes[];
31     public static void main(String arg[]) {
32         init();
33         drawAll();
34     }
```

클래스 Shape의 배열 arrayOfShapes[]를 선언한다.

```
35
36     public static void init() {
37         arrayOfShapes = new Shape[3];
38         arrayOfShapes[0] = new Rectangle();
39         arrayOfShapes[1] = new Triangle();
40         arrayOfShapes[2] = new Circle();
41     }
42 }
```

배열 arrayOfShapes의 각 원소에 객체를 만들어 대입한다. 다형성에 의하여 Shape 객체 배열에 모든 타입의 객체를 저장할 수 있다.

예제

```
43     public static void drawAll() {  
44         for (int i = 0; i < arrayOfShapes.length i++) {  
45             arrayOfShapes[i].draw();  
46         }  
47     }  
48 }
```

배열 arrayOfShapes[] 길이만큼 루프를 돌면서 각 배열 원소를 사용하여 draw() 메소드를 호출해본다. 어떤 draw()가 호출될까? 각 원소가 실제로 가리키고 있는 객체에 따라 서로 다른 draw()가 호출된다.

실행결과

Rectangle Draw
Triangle Draw
Circle Draw

다형성의 장점

- 만약 새로운 도형 클래스를 작성하여 추가한다고 해보자.

```
class Cylinder extends Shape {  
    public void draw(){  
        System.out.println("Cylinder Draw");  
    }  
};
```

- drawAll() 메소드는 수정할 필요가 없다.

```
public static void drawAll() {  
    for (int i = 0; i < arrayOfShapes.length; i++) {  
        arrayOfShapes[i].draw();  
    }  
}
```

동적 바인딩

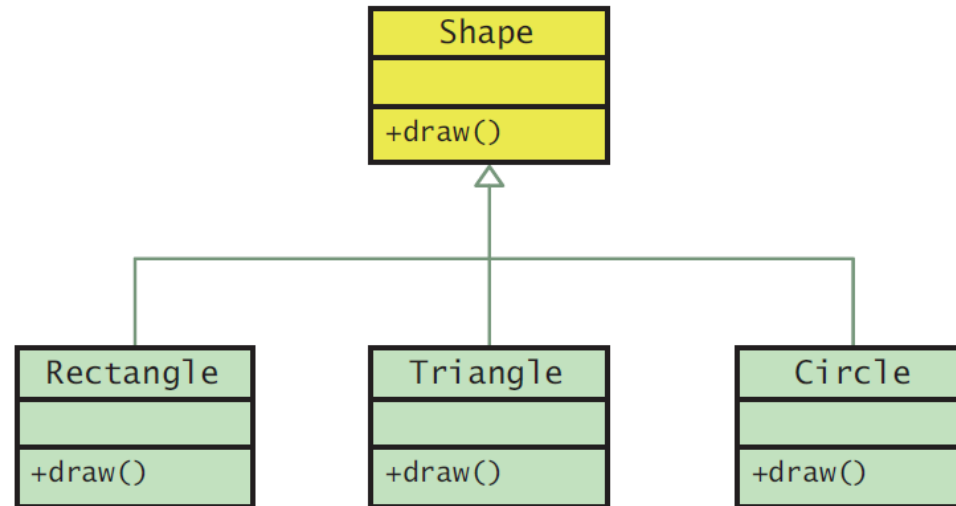


그림12-9. 도형의 UML

```
Shape s = new Rectangle(); // OK!
s.draw();                  // 어떤 draw()가 호출되는가?
```

Shape의 `draw()`가 호출되는 것이 아니라 **Rectangle**의 `draw()`가 호출된다. `s`의 타입은 **Shape**이지만 `s`가 실제로 가리키고 있는 객체의 타입이 **Rectangle**이기 때문이다.

중간 점검

1. 수퍼 클래스 참조 변수는 서브 클래스 객체를 참조할 수 있는가? 역은 성립하는가?
2. `instanceof` 연산자가 하는 연산은 무엇인가?
3. 다형성은 어떤 경우에 유용한가?
4. 어떤 타입의 객체라도 받을 수도 있게 하려면 메소드의 매개변수를 어떤 타입으로 정의하는 것이 좋은가?



equals() 메소드

```
class Car {  
    private String model;  
    public Car(String model) {        this.model= model;    }  
    public boolean equals(Object obj) {  
        if (obj instanceof Car)  
            return model.equals(((Car) obj).model);  
        else  
            return false;  
    }  
}
```

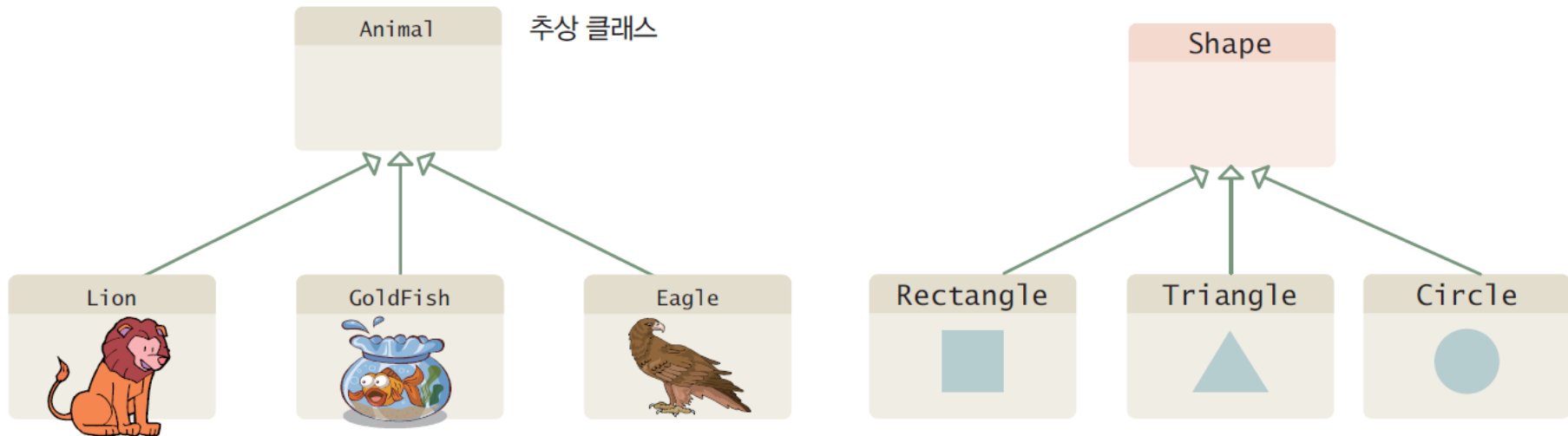
equals()를 재정의한다. String의
equals()를 호출하여서 문자열이 동
일한지를 검사한다.

```
public class CarTest {  
    public static void main(String[] args) {  
        Car firstCar = new Car("BMW520");  
        Car secondCar = new Car("BMW520");  
        if (firstCar.equals(secondCar)) {  
            System.out.println("동일한 종류의 자동차입니다.");  
        }  
        else {  
            System.out.println("동일한 종류의 자동차가 아닙니다.");  
        }  
    }  
}
```

이 equals() 메소드를 사용하여
검사하는 다음과 같은 코드를
가정할 있다.

추상 클래스

- 추상 클래스(abstract class)의 의미
 - 클래스 계층구조에서 상위에 위치, 하위 클래스를 대표하는 클래스
 - 추상의 의미대로 "구체적"이지 않은 클래스
 - 보다 구체적인 하위 클래스를 대표하는 클래스



추상 클래스의 특징

- 다른 일반 클래스와 구별되는 특징
 - ① 추상 클래스는 직접 객체화(instantiation)될 수 없다.
 - 즉 생성자를 사용하여 객체를 생성할 수 없다.
 - ② 추상 클래스는 다른 클래스에 의하여 상속되어야 한다.
 - 즉 하위 클래스가 없는 추상클래스는 의미가 없다.
 - ③ 추상 클래스는 하위 클래스가 있어야 하므로
 - 추상 클래스 구현 시 클래스 앞에 키워드 final이 올 수 없다.
- 키워드 **abstract** 사용

```
public abstract class Shape {  
    protected double x, y;
```

```
    public Shape(double x, double y) {  
        this.x = x;  
        this.y = y;  
    }  
    ...  
}
```

```
Shape s = new Shape(3.1, 4.5); //객체화 오류  
cannot instantiate the type Shape
```

추상 클래스 내부의 추상 메소드

- 추상 메소드
 - 메소드 몸체가 없는 메소드
 - 추상 메소드 정의 시 반환형 앞에 키워드 `abstract`를 기술하고
 - 메소드 몸체 구현 없이 바로 세미콜론을 삽입
 - 추상 메소드는 `private`와 `final`이 사용될 수 없음
- 추상 클래스
 - 적어도 하나 이상의 추상 메소드를 가진 클래스는 반드시 추상이어야 함
 - 키워드 `final` 이용 불가능

```
public abstract class Shape {  
    protected double x, y;  
    ...  
    public void drawCenter() {           //일반 클래스  
        System.out.println("(x, y) = " + x + ", " + y);  
    }  
    public abstract void draw();         //추상 클래스  
}
```

클래스 Shape는 추상 메소드 draw()를 가지므로 반드시 추상 클래스여야 한다.

추상 메소드 draw()는 몸체가 구현되지 않은 메소드이다.

하위 클래스에서 추상 메소드의 구현

```
public abstract class Shape {  
    protected double x, y;  
    ...  
    public void drawCenter() { //일반  
        System.out.println("중심좌표 (x, y) = " + x + ", " + y);  
    }  
    public abstract void draw(); //추상  
}
```

하위 클래스에서 추상 메소드를 반드시 구현해야 한다. 아니면 다음과 같은 문법 오류가 발생한다.

The type Rectangle must implement the inherited abstract method Shape draw()

```
public class Rectangle extends Shape {  
    double width;  
    double height;  
    ...  
    public void draw() {  
        super.drawCenter();  
        System.out.printf("가로: %f, 세로: %f, ", width, height);  
        System.out.printf("사각형 면적: %f\n", width*height);  
    }  
}
```

```
public class Circle extends Shape {  
    double radius;  
    ...  
    public void draw() {  
        super.drawCenter();  
        System.out.printf("반지름: %f, ", radius);  
        System.out.printf("원 면적: %f\n", radius*radius*Math.PI);  
    }  
}
```

추상 클래스의 예

```
abstract class Shape {  
    int x, y;  
    public void move(int x, int y)  
    {  
        this.x = x;  
        this.y = y;  
    }  
    public abstract void draw();  
};
```

추상 클래스 Shape를 선언한다. 추상 클래스로는 객체를 생성할 수 없다.

추상 클래스라고 하더라도 추상 메소드가 아닌 보통의 메소드도 가질 수 있음을 유의하라.

추상 메소드를 선언한다. 추상 메소드를 하나라도 가지면 추상 클래스가 된다. 추상 메소드를 가지고 있는데도 abstract를 class 앞에 붙이지 않으면 컴파일 오류가 발생한다.

```
public class Rectangle extends Shape {  
    int width, height;  
    public void draw() { // 추상 메소드 구현  
        System.out.println("사각형 그리기 메소드");  
    }  
};
```

서브 클래스 Rectangle에서 슈퍼 클래스의 추상 메소드 draw()가 실제 메소드로 구현한다. 서브 클래스에서 추상 메소드를 구현하지 않으면 컴파일 오류가 발생한다.

```
class Circle extends Shape {  
    int radius;  
    public void draw() {  
        System.out.println("원 그리기 메소드");  
    }  
};
```

추상 메소드 draw()가 실제 메소드로 구현한다.

- 과일, 사과, 배, 포도를 표현한 클래스를 만들고, 이들 간의 관계를 고려하여 하나의 클래스를 추상 클래스로 만들어 메소드 print()를 구현하고

```
Fruit f[] = {new Grape(), new Apple(), new Pear()}  
for(Fruit fr:f) fr.print();
```

의 결과가

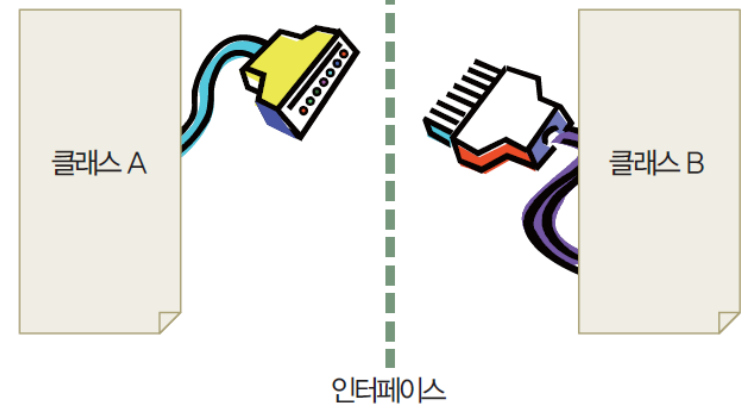
나는 포도

나는 사과

나는 배

가 되도록 클래스를 작성하라.

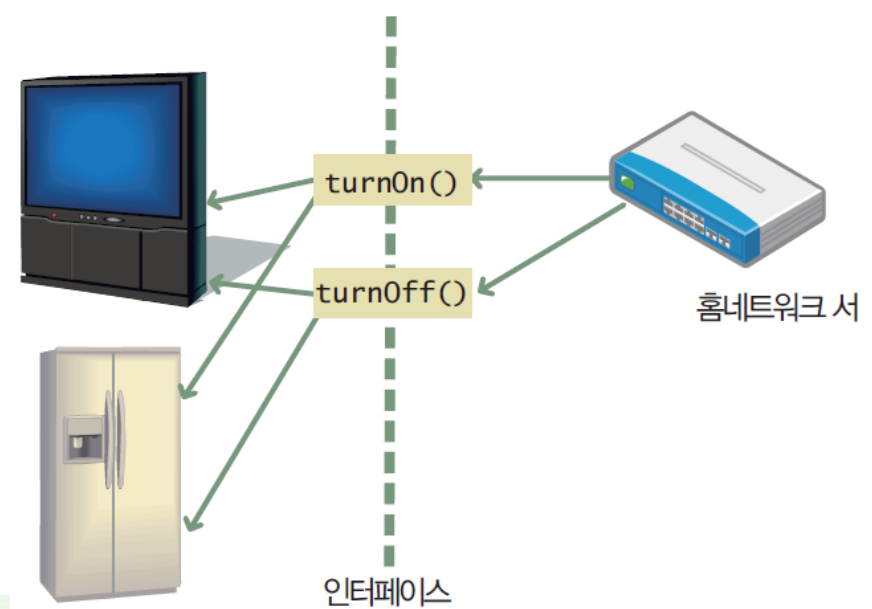
인터페이스



- IT에서 인터페이스
 - 인터페이스는 컴퓨터와 다른 주변기기를 연결하는 표준
 - usb, 병렬포트...
- 자바에서 인터페이스
 - 객체와 객체 사이의 상호작용을 위해 만들어 놓은 클래스
 - 해야 할 작업의 구체적 구현 없이 기능만 선언한 클래스
 - 하위 클래스가 수행해야 하는 메소드와 필요한 상수만을 미리 추상적으로 정의해 놓은 특별한 클래스
 - 인터페이스는 다중 상속(multiple inheritance)을 지원
 - 자바의 일반 클래스는 다중 상속을 지원하지 않음
 - 키워드 interface

인터페이스

- 인터페이스(interface)
 - 추상 메소드들로만 이루어진다.



```
public interface 인터페이스_이름 {  
    반환형 추상 메소드1(...);  
    반환형 추상 메소드2(...);  
    ...  
}
```

인터페이스 안에는 추상 메소드들이 정의된다.

```
public class 클래스_이름 implements 인터페이스_이름 {  
    반환형 추상 메소드1(...) {  
        .....  
    }  
    반환형 추상 메소드2(...) {  
        .....  
    }  
}
```

인터페이스를 구현하는 클래스는 추상 메소드의 몸체를 구현하여야 한다.

- 인터페이스 내부의 추상메소드에서 접근지정자 public abstract 생략 가능

홈네트워킹 예제

```
public interface RemoteControl {  
    // 추상 메소드 정의  
    public void turnOn();        // 가전 제품을 켜다.  
    public void turnOff();       // 가전 제품을 끄다.  
}
```

```
public class Television implements RemoteControl {  
    public void turnOn()  
    {  
        // 실제로 TV의 전원을 켜기 위한 코드가 들어 간다.  
        ...  
    }  
    public void turnOff()  
    {  
        // 실제로 TV의 전원을 끄기 위한 코드가 들어 간다.  
        ...  
    }  
}
```

```
Television t = new Television();  
t.turnOn();  
t.turnOff();
```

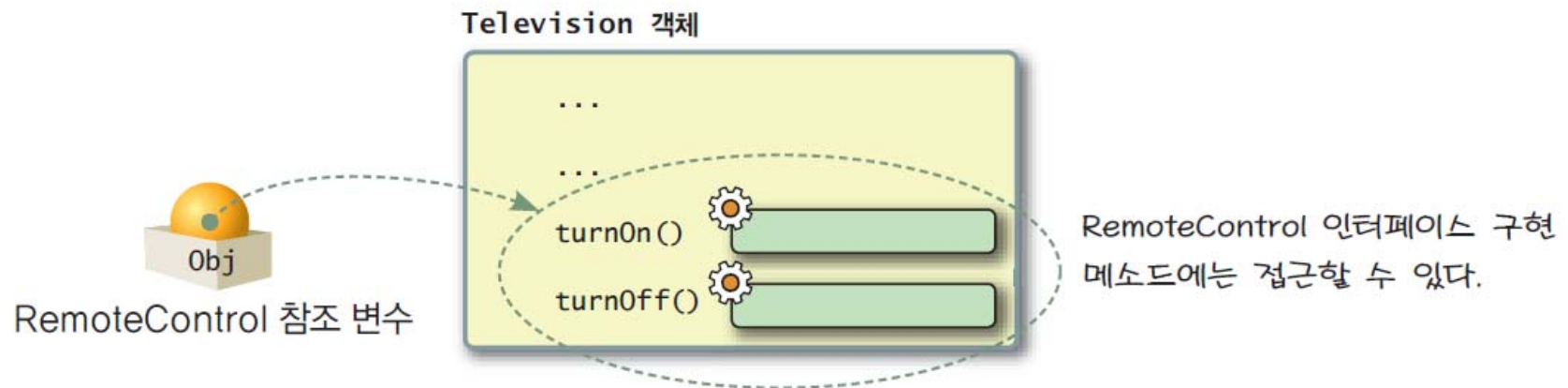
```
Refrigerator r = new Refrigerator();  
r.turnOn();  
r.turnOff();
```

인터페이스와 타입

```
RemoteControl obj = new Television();  
obj.turnOn();  
obj.turnOff();
```

Television 객체이지만 RemoteControl 인터페이스를
구현하기 때문에 RemoteControl 타입의 변수로 가리
킬 수 있다.

obj를 통해서 RemoteControl 인터페이스에 정
의된 메소드만을 호출할 수 있다.



예제

```
public interface Comparable {  
    // 이 객체가 다른 객체보다 크면 1, 같으면 0, 작으면 -1을 반환한다.  
    int compareTo(Object other);  
}
```

```
public class Box implements Comparable {  
    private double volume = 0;  
    public Box(double v) {  
        volume = v;  
    }  
    public int compareTo(Object otherObject) {  
        Box other = (Box) otherObject;  
        if (this.volume < other.volume) return -1;  
        else if (this.volume > other.volume) return 1;  
        else return 0;  
    }  
    public static void main(String[] args) {  
        Box b1 = new Box(100);  
        Box b2 = new Box(85.0);  
        if (b1.compareTo(b2) > 0)  
            System.out.println("b1이 b2보다 더 크다");  
        else  
            System.out.println("b1이 b2와 같거나 작다");  
    }  
}
```

Box 클래스는 Comparable 인터페이스를
구현하고 있기 때문에 비교 가능하다.


Comparable 인터페이스의 메소드
compareTo()를 구현한다. otherObject를 형
변환하여서 Box 참조 변수로 바꾼다.

여러 인터페이스를 동시에 구현

```
public interface SerialCommunication {  
    void send(byte[] data);    // 시리얼 포트에 데이터를 전송한다.  
    byte[] receive();         // 시리얼 포트에서 데이터를 받는다.  
}
```

```
public class Television implements RemoteControl, SerialCommunication  
{  
    ...  
    // RemoteControl과 SerialCommunication의 메소드를 동시에 구현하여야 한다.  
    public void turnON() { ... }  
    public void turnOFF() { ... }  
    public void send(byte[] data) { ... }  
    public byte[] receive() { ... }  
}
```

2개의 인터페이스를 동시에
구현한다는 의미이다.



인터페이스 상속하기

```
public interface RemoteControl {  
    public void turnON();        // 가전 제품을 켜다.  
    public void turnOFF();       // 가전 제품을 끈다.  
}
```

```
public interface RemoteControl {  
    public void turnON();        // 가전 제품을 켜다.  
    public void turnOFF();       // 가전 제품을 끈다.  
    public void volumeUp();      // 가전제품의 볼륨을 높인다.  
    public void volumeDown();    // 가전제품의 볼륨을 낮춘다.  
}
```

```
public interface AdvancedRemoteControl extends RemoteControl { SerialCommunication  
    public void volumeUp();      // 가전제품의 볼륨을 높인다.  
    public void volumeDown();    // 가전제품의 볼륨을 낮춘다.  
}
```

인터페이스도 다른 인터페이스를 상속받을 수 있다.

인터페이스는 다중 상속 가능

중간 점검

1. 추상 클래스의 주된 용도는 무엇인가?
2. 추상 클래스는 일반 메소드를 포함할 수 있는가?
3. 추상 클래스를 상속받으면 반드시 추상 메소드를 구현하여야 하는가?

1. 인터페이스의 주된 용도는 무엇인가?
2. 하나의 클래스가 두 개의 인터페이스를 구현할 수 있는가?
3. 인터페이스 안에 인스턴스 변수를 선언할 수 있는가?

인터페이스와 다중 상속

```
class SuperA { int x; }  
class SuperB { int x; }  
class Sub extends SuperA, SuperB    // 만약에 다중 상속이 허용된다면  
{  
    ...  
}  
Sub obj = new Sub();  
obj.x = 10;                          // obj.x는 어떤 슈퍼 클래스의 x를 참조하는가?
```

```
class Sub extends Super implements Interface1, Interface2 {
```

```
    // 클래스의 정의
```

```
}
```

Super 클래스를 상속받으면서 동시에 Interface1, Interface2를 구현하는 클래스를 정의한다.

인터페이스와 다중 상속

```
class Shape {  
    protected int x, y;  
}  
  
interface Drawable {  
    void draw();  
};  
  
public class Rectangle extends Shape implements Drawable {  
    int width, height;  
    public void draw() {  
        System.out.println("Rectangle Draw");  
    }  
};
```

```
abstract class MyComparable  
{  
    public abstract int compareTo(Object other);  
}
```

```
public class Rectangle extends Shape, MyComparable // 컴파일 오류!!  
{  
    ...  
}
```

```
public class Rectangle extends Shape implements MyComparable // OK  
{  
    ...  
}
```


상수 정의

```
interface Days {
```

```
    public static final int SUNDAY = 1, MONDAY = 2, TUESDAY = 3,  
        WEDNESDAY = 4, THURSDAY = 5, FRIDAY = 6, SATURDAY = 7;
```

```
}
```

상수가 정의된 인터페이스이다.

상수는 대개 정적 변수로 선언된다.

```
public class DayTest implements Days
```

```
{
```

```
    public static void main(String[] args)
```

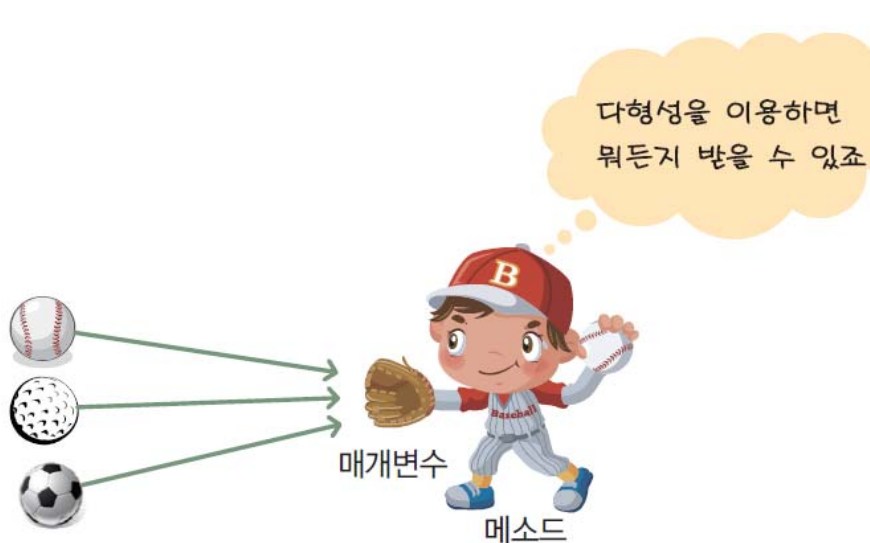
```
    {
```

```
        System.out.println("일요일: " + SUNDAY);
```

```
    }
```

```
}
```

상속과 interface 구현하는 이유



```
class Baseball{...}  
class Golfball{...}  
class Soccerball{...}
```

```
void catch(Baseball b){...}  
void catch(Golfball b){...}  
void catch(Soccerball b){...}
```

=====

```
class Baseball extends Ball{...}  
class Golfball extends Ball{...}  
class Soccerball extends Ball{...}
```

```
void catch(Ball b){...}
```

그림12-8. 다형성을 이용하는 메소드의 매개변수