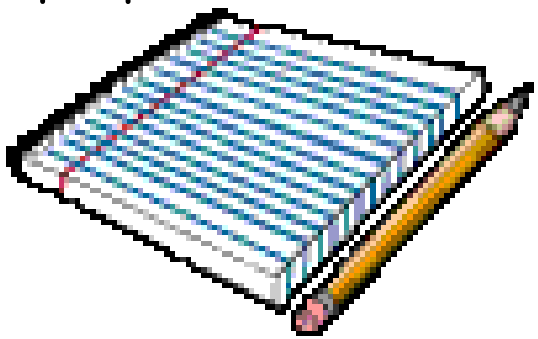
J10-12
인터페이스와 다형성

충남대학교
컴퓨터공학과

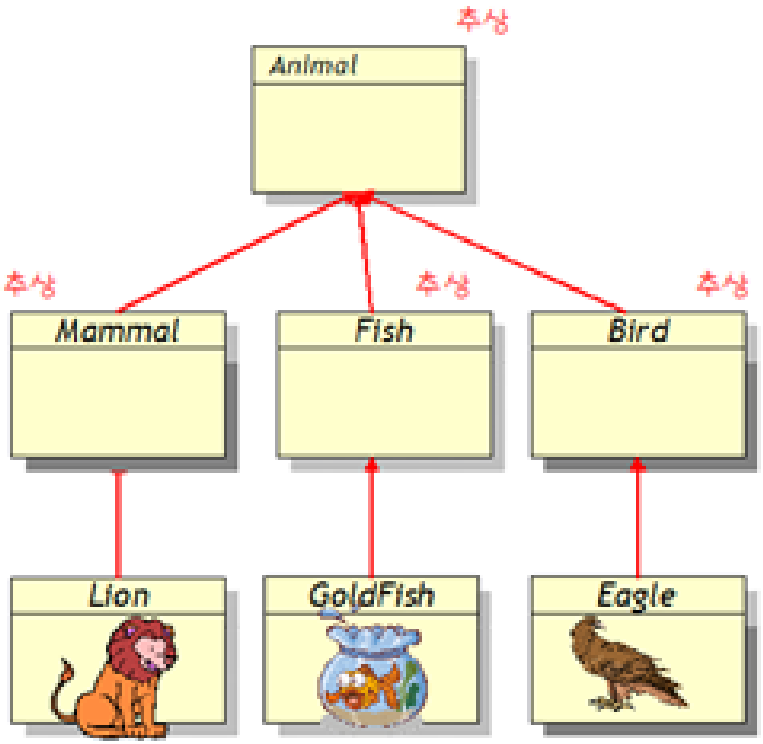


학습 내용

1. 추상 클래스
2. 인터페이스
3. 다형성
4. 내부 클래스

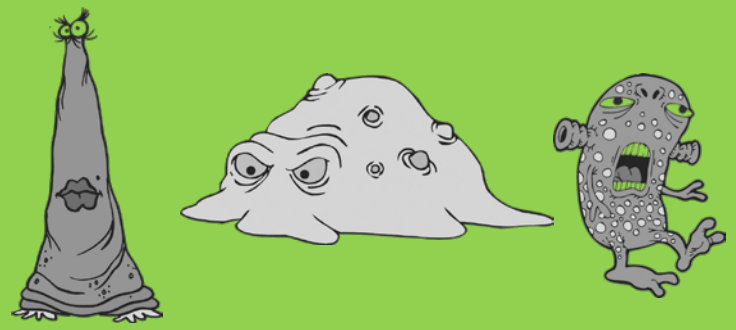
1. 추상 클래스(abstract class)

- 추상적인 개념을 표현하기 위한 목적으로 만든 클래스
- 추상 클래스 타입의 객체 인스턴스를 만들 수 없다.

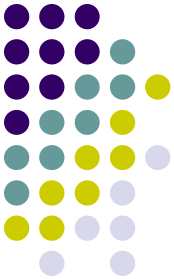



```
abstract class Animal {
    …
}
```

Animal 객체는 어떻게 생겼을까?



추상 메소드(abstract method)



- 몸체가 없는 메소드

```
public abstract void eat();
```

- 서브 클래스에서 반드시 재정의해야 함.
- 추상 메소드를 포함하는 클래스는 반드시 추상 클래스이어야 함
  - 하나 이상 abstract 메소드를 포함하는 클래스는 abstract로 선언되어야 한다.
  - 객체 인스턴스 생성은 불가!
- 추상 클래스가 아닌 클래스는 추상 메소드를 가질 수 없음

추상 클래스 예

- Shape 상속 계층

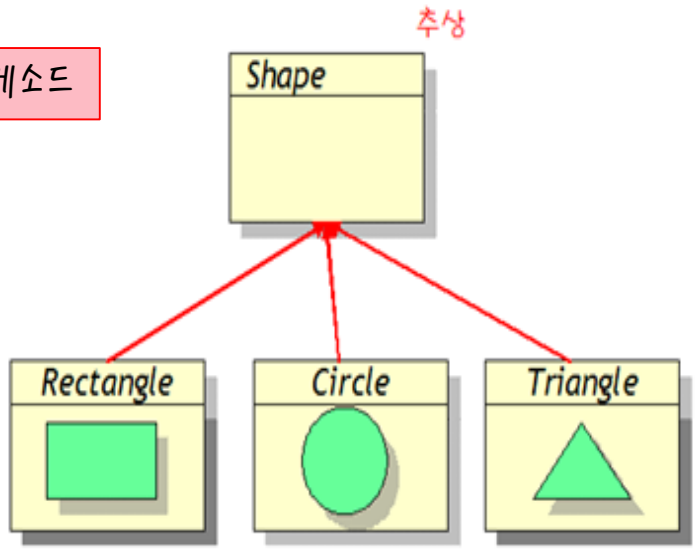
```
abstract class Shape {
    int x, y;
    void move(int x, int y) {
        this.x = x;
        this.y = y;
    }
    abstract void draw();
}
```

추상 클래스는
추상 메소드를 갖는다.

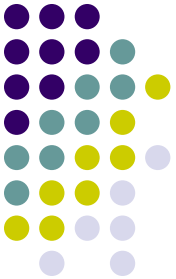
추상 클래스는
필드를 가질 수 있다.

추상 클래스는
일반적인 메소드를 가질 수 있다.

추상 메소드



Shape 상속 계층(Cont')



```
class Rectangle extends Shape {
   int width, height;
   void draw() {
      System.out.println("사각형그리기");
   }
}
class Circle extends Shape {
   int radian;
   void draw()  {
      System.out.println("원그리기");
   }
}
```

추상클래스 상속

추상메소드 재정의

```
public class ATest {
   public static void main() {
      Rectangle r = new Rectangle();
      Circle    c = new Circle();
      r.draw();
      c.draw();
   }
}
```

2. 인터페이스(Interface)

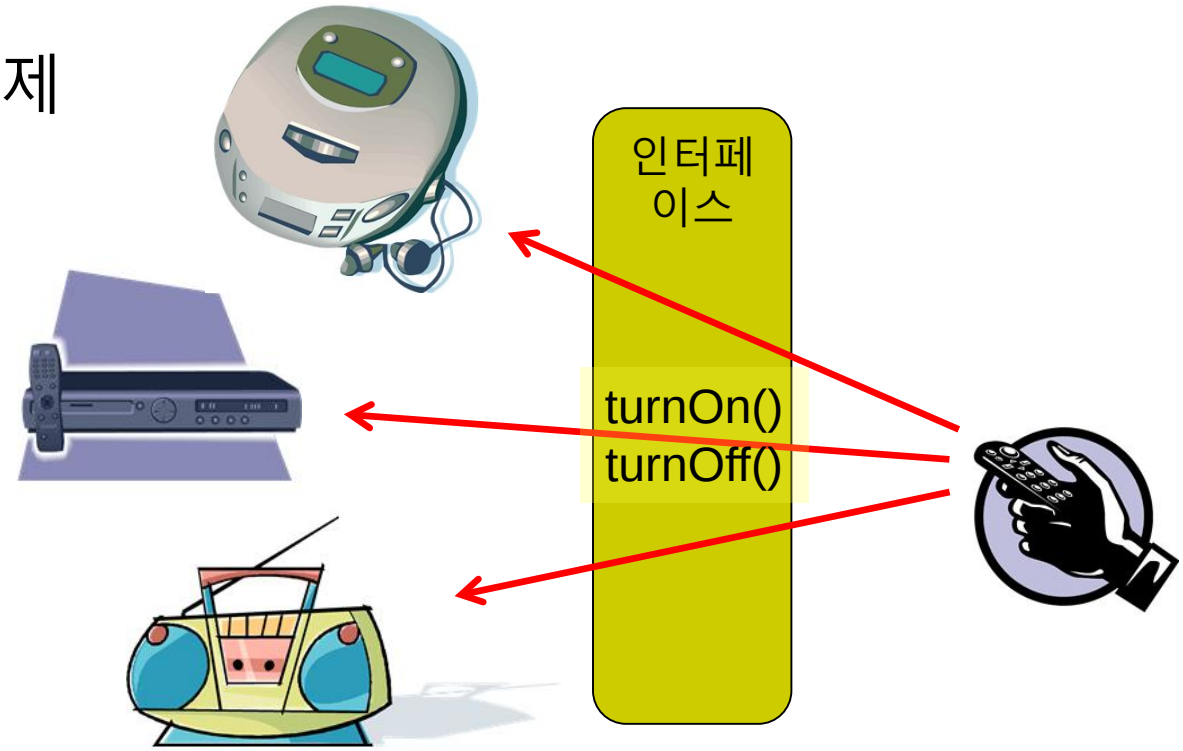
- 구현되지 않은 추상 메소드들의 집합
  - 인터페이스 안의 추상 메소드는 모두 public, abstract임 (생략 가능)
- 원하는 메소드들을 가진 객체들의 타입으로 사용 가능
- 개별적인 연산은 각 클래스에서 구현

```
public interface 인터페이스_이름 {
    반환형   추상메소드1(...);
    반환형   추상메소드2(...);
    ...
}
```

```
public class 클래스_이름 implements 인터페이스_이름 {
    반환형   추상메소드1(...) {
             .....
    }
    반환형   추상메소드2(...) {
             .....
    }
}
```

인터페이스의 기능

- 인터페이스는 **객체와 객체 사이의 상호 작용**을 나타내기 위한 방법이다.
- 홈 네트워킹 예제



인터페이스가 구현된 클래스의 객체들

인터페이스와 타입

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

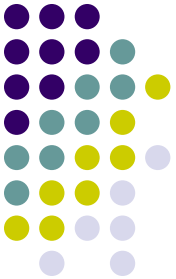
```
Refrigerator r = new Refrigerator();
r.turnOn();
r.turnOff();
```

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

Television 객체이지만 RemoteControl 인터페이스를 구현하기 때문에 RemoteControl 타입의 변수로 가리킬 수 있다.

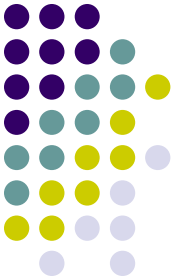
obj를 통해서는 RemoteControl 인터페이스에 정의된 메소드 만을 호출할 수 있다.

인터페이스 사용 예 - compareTo



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

compareTo - Box

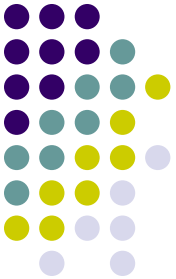


```
public class Box implements Comparable {
    private double volume = 0;
    public Box(double v) {
        volume = v;
    }
    public int compareTo(Object otherObject) {
        Box other = (Box) otherObject;
        if (this.volume < other.volume)  return -1;
        else if (this.volume > other.volume) 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("b1가 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.

compareTo – Student (gpa)



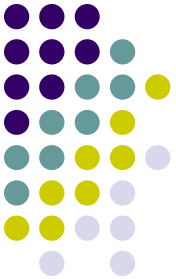
```
class Student implements Comparable {
    private String name;            // 이름
    private double gpa;             // 평점

    public Student(String n, double g) {
        name = n;
        gpa = g;
    }

    public String getName() { return name; }
    public double getGPA() { return gpa; }
    public int compareTo(Object obj) {
        Student other = (Student) obj;
        if (gpa < other.gpa)
            return -1;
        else if (gpa > other.gpa)
            return 1;
        else
            return 0;
    }
}
```

```
public class StudentTest {
    public static void main(String[] args) {

        Student[] students = new Student[3];
        students[0] = new Student("홍길동", 3.39);
        students[1] = new Student("임꺽정", 4.21);
        students[2] = new Student("황진이", 2.19);

        Arrays.sort(students);
        for (Student s : students)
                System.out.println("이름=" + s.getName() + " 평점=" + s.getGPA());
    }
}
```

Arrays 클래스의 정적 메소드 sort()는 Comaparable 인터페이스를 구현한 원소로 이루어진 배열을 정렬한다.

for-each 구문이다. 배열 안의 모든 배열 원소가 s에 대입되면서 반복된다.

여러 인터페이스 implements 하기



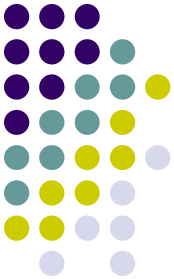
```
public interface RemoteControl {
    public void turnON();        // 가전 제품을 켠다.
    public void turnOFF();       // 가전 제품을 끈다.
}
```

```
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.

인터페이스의 상속



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

기능 추가

다시 정의

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

또는 상속을 이용

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

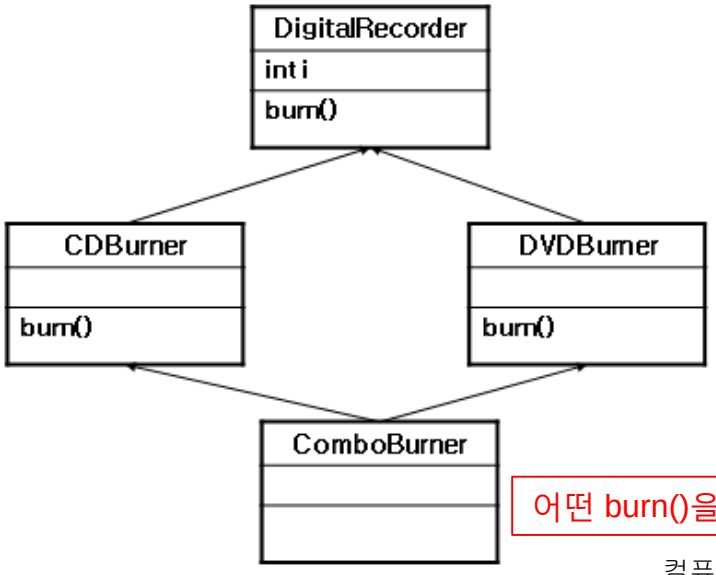
인터페이스도 다른 인터페이스를 상속받을 수 있다.

다중 상속과 인터페이스

다중 상속의 문제점

```
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 수퍼 클래스의 x를 참조하는가?
```

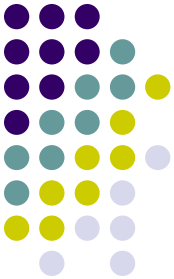
Java는 다중 상속을 허용하지 않음



어떤 burn()을 실행하는가?

인터페이스를 정의하고 구현해서 해결한다.

다중 상속과 인터페이스 예



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

```
public class Circle extends Shape implements Drawable {
    int   radius;
    public void draw() {
        System.out.println(“Circle Draw”);
    }
}
```

인터페이스와 추상 클래스

```
class Shape {
    protected int x, y;
}
```

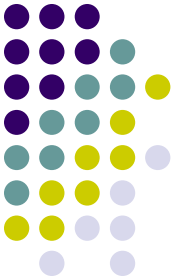
```
abstract class MyComparable
{
    public abstract int compareTo(Object other);
}
```

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

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

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

인터페이스와 상수 정의



- 인터페이스는 여러 클래스에서 사용되는 상수를 정의하는데 사용된다.

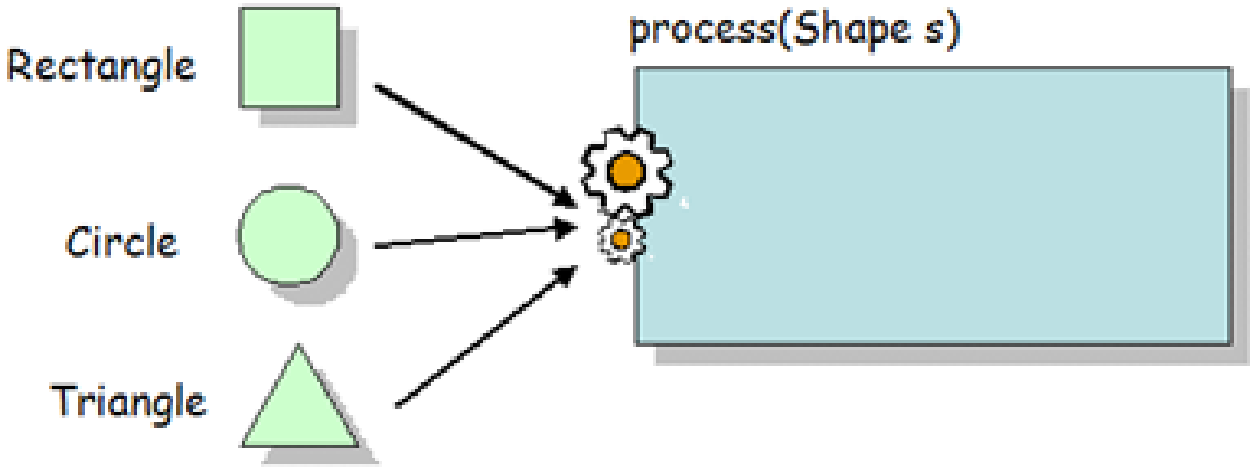
```
interface Days {
    public static final int SUNDAY = 1, MONDAY = 2, TUESDAY = 3,
           WEDNESDAY = 4, THURSDAY = 5, FRIDAY = 6, SATURDAY = 7;
}
```

상수가 정의된 인터페이스이다.
상수는 대개 정적 변수로 선언된다.

```
public class DayTest implements Days
{
    public static void main(String[] args)
    {
        System.out.println("일요일: " + SUNDAY);
    }
}
```

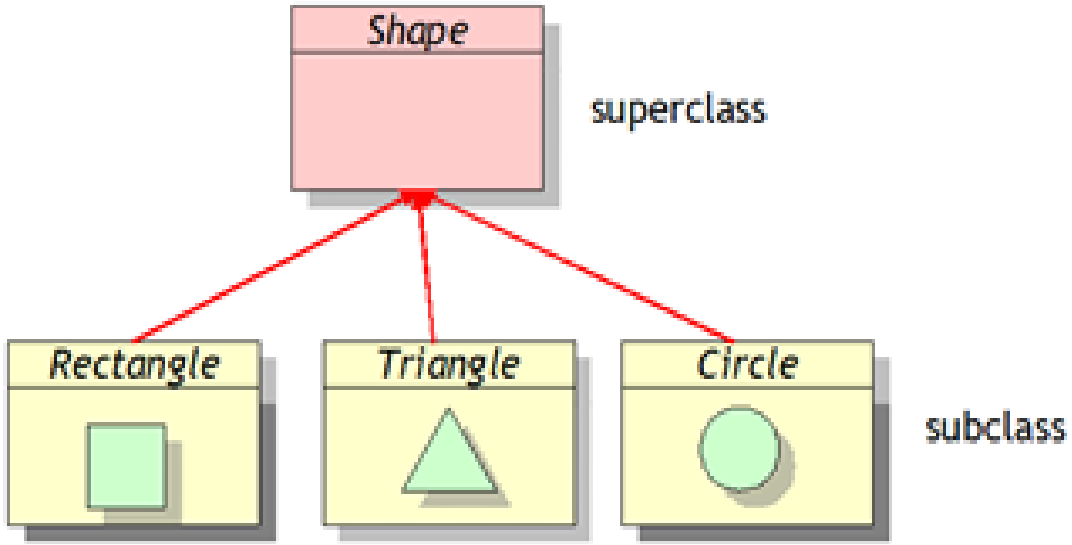
3. 다형성(Polymorphism)

- Polymorphism의 어원
  - Greek: "*having many forms*"
  - OOP : *many methods with the same signature*
- 다양한 객체들을 하나의 코드로 처리하는 기술
  - 문장은 같은데 결과는 다르다



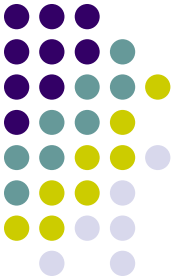
상향 형변환

- 수퍼 클래스 타입으로 선언된 참조변수를 이용하여 서브 클래스 타입 객체 참조 가능

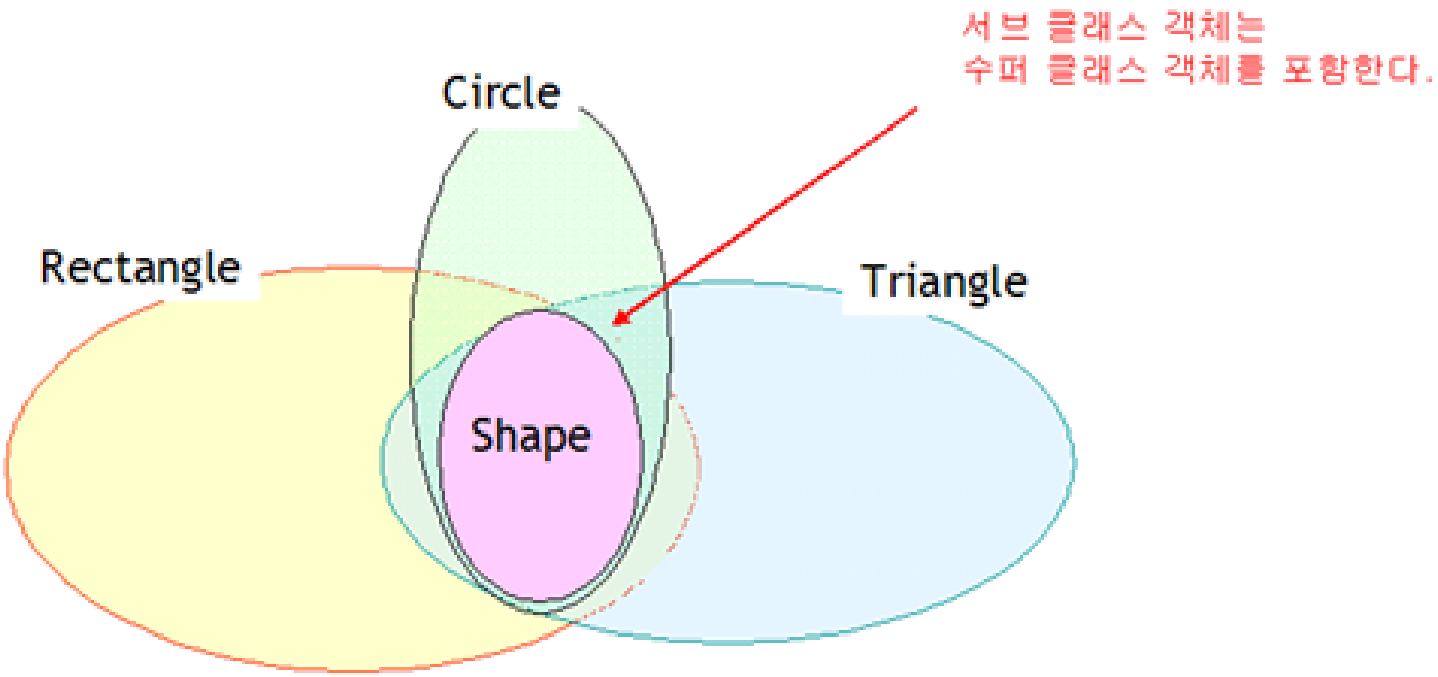



```
Shape s1 = new Rectangle();  // OK!
Shape s2 = new Triangle();   // OK!
Shape s3 = new Circle();     // OK!
```

수퍼클래스 참조 변수로 서브클래스 객체 참조가 가능한 이유

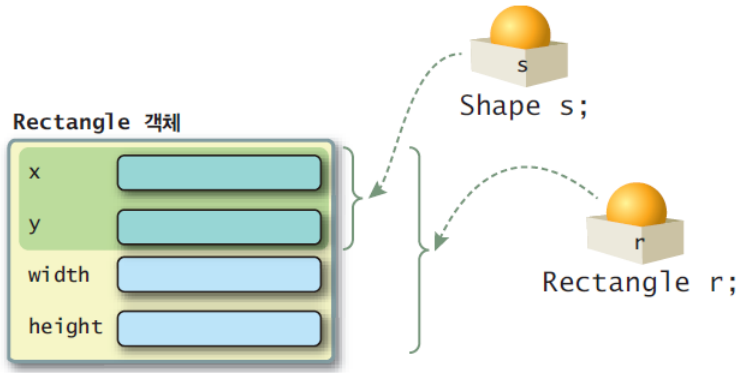


- 서브클래스 객체는 수퍼클래스 객체를 포함하고 있기 때문
- Subclass is-a superclass 관계이기 때문



상향 형변환 예

```
01  class Shape {
02      int x, y;
03  }
04
05  class Rectangle extends Shape {
06      int width, height;
07  }
08
09  public class ShapeTest {
10      public static void main(String arg[]) {
11          Shape s;
12          Rectangle r = new Rectangle();
13          s = r;
14          s.x = 0;
15          s.y = 0;
16          s.width = 100;
17          s.height = 100;
18      }
19  }
```



수퍼 클래스의 참조변수로 서브 클래스의 객체를 가리키는 것은 합법적이다.

Shape 클래스의 필드와 메소드에 접근하는 것은 OK

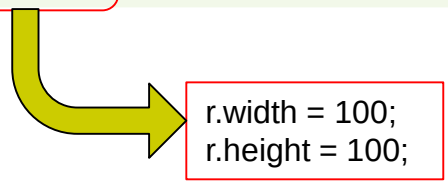
컴파일 오류가 발생한다. s를 통해서는 Rectangle 클래스의 필드와 메소드에 접근할 수 없다.

하향 형변환

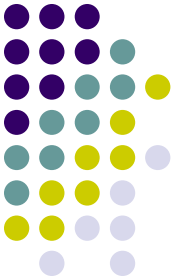
- 서브 클래스 객체인데 형변환에 의하여 일시적으로 수퍼 클래스 참조 변수에 의해 참조되고 있는 경우만 가능

```
Rectangle r;
Shape s;
s = new Rectangle();

r = (Rectangle)s;
r->width = 100;
r->height = 100;
```

하향 형변환: r을 통하여 다시 Rectangle 클래스의 모든 멤버에 대한 접근이 가능하다.



하향 형변환과 instanceof



- 하향 형변환을 하려면 subclass의 type을 정확히 알아야 한다.
  - Instanceof를 사용한다.

```
public void shapeType(Shape s) {
    if (s instanceof Rectangle) {
        r = (Rectangle) s;
        System.out.println("Shape is Rectangle");
    } else if (s instanceof Circle) {
        c = (Circle) s;
        System.out.println("Shape is Circle");
    } else
        System.out.println("Shape is Unknown");
}
```

다형성의 이용

- Method의 매개변수로 super class 참조변수를 이용한다.

```
public static double calcArea(Shape s) {
    double area = 0.0;
    if (s instanceof Rectangle) {
        int w = ((Rectangle) s).getWidth();
        int h = ((Rectangle) s).getHeight();
        area = (double) (w * h);
    }
    ...          // 다른 도형들의 면적을 구한다.
    return area;
}
```

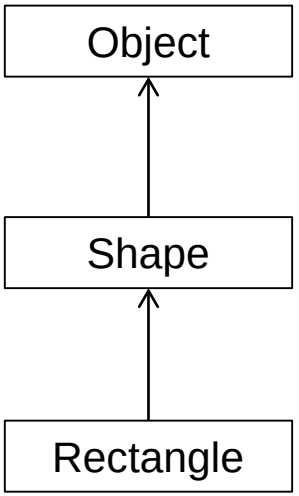
Shape에서 파생된 모든 클래스의 객체를 다 받을 수 있다.

만약 Rectangle 객체이면 (가로*세로)로 면적을 구한다.

상향 형변환

하향 형변환

```
Rectangle r = new Rectangle(…);
… calcArea(r) …
```




```
public void print(Object obj) {
        ...
}
```

모든 타입의 객체를 전부 받을 수 있다. 필요하면 다른 타입으로 형변환해서 사용하면 된다.

다형성 – 동적 바인딩

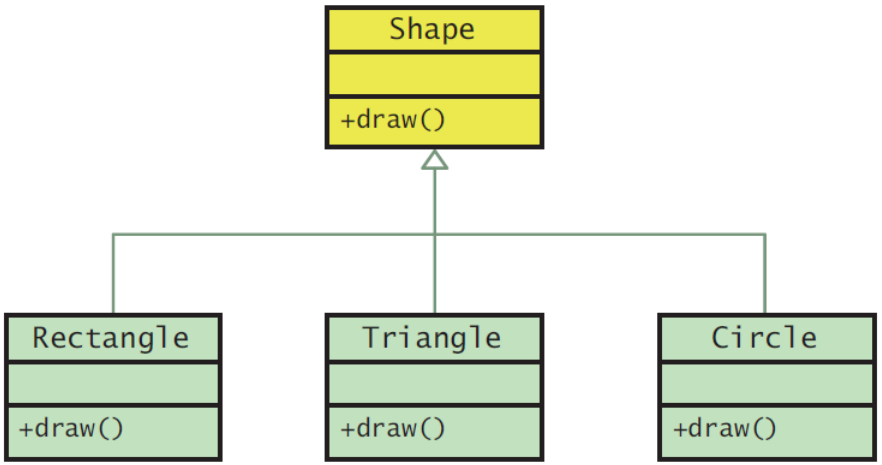
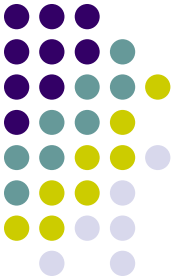


그림12-9. 도형의 UML

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

Shape의 draw()가 호출되는 것이 아니라 Rectangle의 draw()가 호출된다. s의 타입은 Shape이지만 s가 실제로 가리키고 있는 객체의 타입이 Rectangle이기 때문이다.

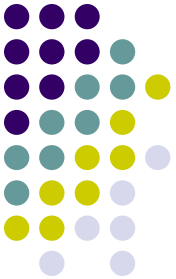
다양한 도형 그리기: 다형성을 사용하지 않은 경우



```
class  Shape {
   protected int  x, y;
}

class Rectangle extends Shape {
   private int width, height;
   public void draw()  {
      System.out.println("Rectangle Draw");
   }
}

class Triangle extends Shape {
   private int base, height;
   public void draw()  {
      System.out.println("Triangle Draw");
   }
}

…Circle, Pentagon, Hexagon, …
```

```
public class ShapeTest {
   private static Rectangle  r;
   private static Triangle  t;
   ……
   public static void main(String[ ] args) {
      init();
      drawAll();
   }

   public static void init() {
      r = new Rectangle();
      t = new Triangle();
      … Circle, Pentagon, Hexagon, …
   }

   public static void drawAll()  {
      r.draw();
      t.draw();
      … Circle, Pentagon, Hexagon, …
   }
}
```

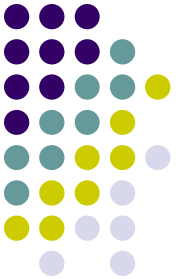
다양한 도형 그리기: 다형성을 사용하는 경우



```
class  Shape {
   protected int  x, y;
   public void draw() {
      System.out.println("Shape Draw");
   }
}

class Rectangle extends Shape {
   private int width, height;
   public void draw()  {
      System.out.println("Rectangle Draw");
   }
}

class Triangle extends Shape {
   private int base, height;
   public void draw()  {
      System.out.println("Triangle Draw");
   }
}

… Circle, Pentagon, Hexagon, …
```

```
public class ShapeTest {
   protected static Shape[ ]  arrayOfShapes;
   public static void main(String[ ] args) {
      init();
      drawAll();
   }

   public static void init() {
      arrayOfShapes = new Shape[9];
      arrayOfShapes[0] = new Rectangle();
      arrayOfShapes[1] = new Triangle();
      … Circle, Pentagon, Hexagon, …
   }

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

다형성의 장점



- Shape 계층에 새로운 도형 클래스 추가할 경우

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

- drawAll() 메소드는 수정할 필요가 없다.

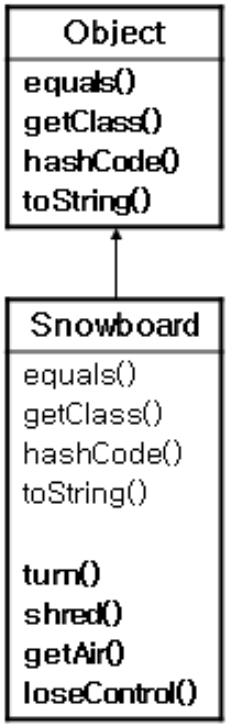
```
public static void drawAll() {
    for (int i = 0; i < arrayOfShapes.length; i++) {
        arrayOfShapes[i].draw();
    }
}
```

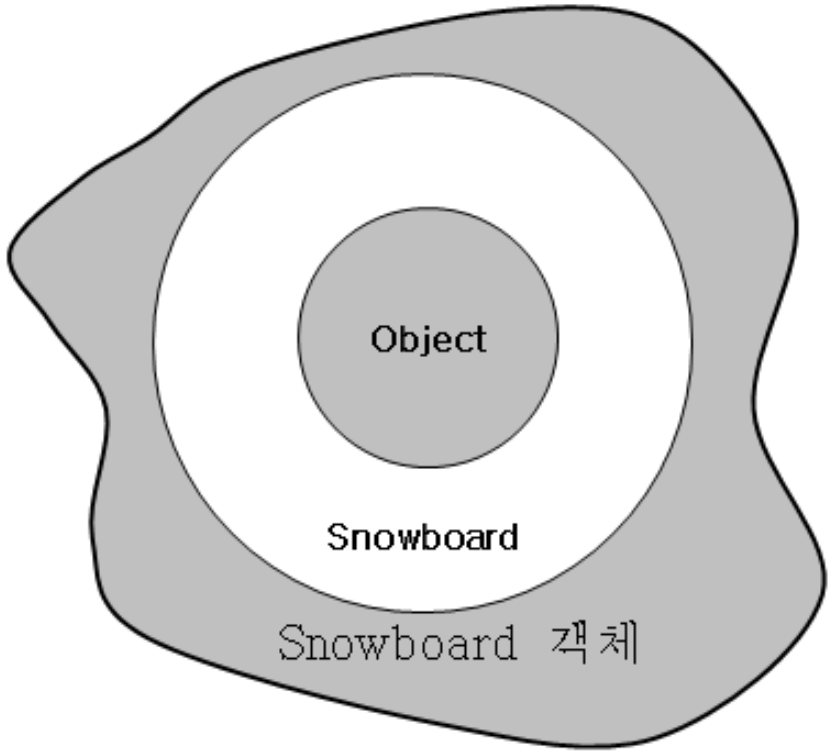
다형적인 인자, 반환 값

- 메소드의 인자와 반환 값에도 다형성 적용 가능
  - 메소드의 매개 변수로 수퍼 클래스 참조 변수를 이용
  - 다형성을 이용하는 전형적인 방법

Object 클래스와 다형성

- Object 클래스는 모든 자바 클래스의 수퍼클래스

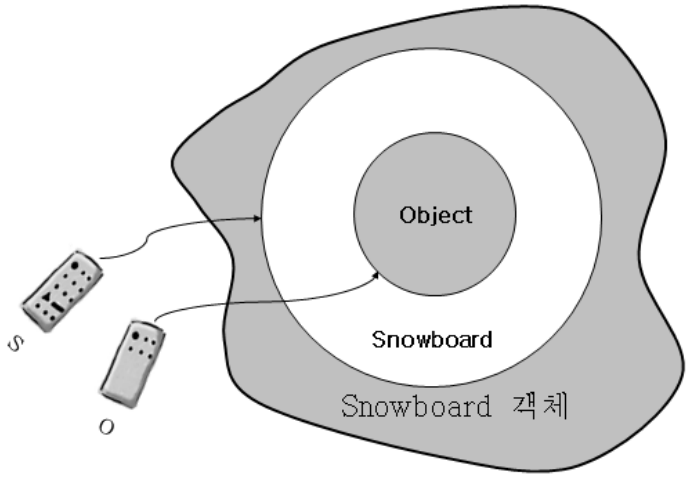




Object 타입 참조변수

- Object 클래스 타입 참조변수로 모든 클래스 타입 객체를 참조할 수 있다.

```
Snowboard s = new Snowboard();
Object o = s;
```



- o를 통하여 Snowboard 클래스의 필드와 메소드를 사용하고자 할 때는? ⇒ 형 변환 필요
  - 예: ((Snowboard) o).turn();

다형적인 Object 타입 참조변수

- 다형적인 인자/반환형으로 사용 가능

```
public void go() {
    Dog aDog = new Dog();
    Dog sameDog = getObject(aDog);
}
public Object getObject(Object o) {
    return o;
}
```

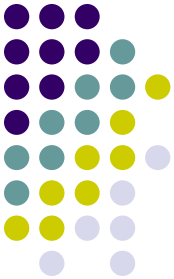
오류! : 반환유형이 Object 타입이므로 Object 타입으로 받아야 함

Dog도 Object의 서브클래스이므로 Object 타입 매개변수로 받을 수 있음

- 참조변수 형 변환 필요

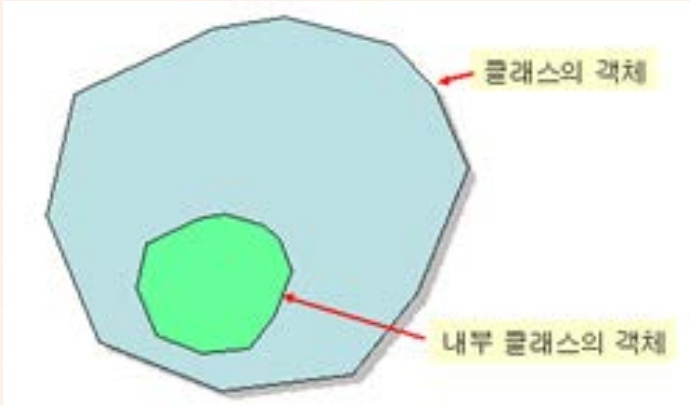
```
public void go() {
    Dog aDog = new Dog();
    Dog sameDog = (Dog) getObject(aDog);
}
```

4. 내부 클래스(Inner Class)



- 클래스 안에 다른 클래스를 정의한 것

```
public class OuterClass {
    // 클래스의 필드와 메소드 정의
    ...
    private class InnerClass {
        // 내부 클래스의 필드와 메소드 정의
        ...
    }
}
```

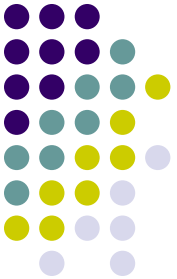


- 내부 클래스 사용 목적
  - 특정 멤버 변수를 private로 유지하면서 자유롭게 사용할 수 있다.
  - 특정한 곳에서만 사용되는 클래스들을 모을 수 있다.
  - 보다 읽기 쉽고 유지 보수가 쉬운 코드가 된다.

내부 클래스 사용 예

```
class OuterClass {
        private String secret = "Time is money";

        public OuterClass() {
                InnerClass obj = new InnerClass();
                obj.method();
        }

        private class InnerClass {
                public InnerClass() {
                        System.out.println("내부 클래스 생성자입니다.");
                }

                public void method() {
                        System.out.println(secret);
                }
        }

}

public class OuterClassTest {
        public static void main(String args[]) {
                new OuterClass();
        }
}
```

내부 클래스



수고했습니다!!!