JAVA 기초

다형성(Polymorphism)

- method overriding/Dynamic Binding

학습목표

학/습/내/용

1. Method Overriding
2. 동적 바인딩

학/습/목/표

1. Method Overriding에 대하여 학습한다.
2. 동적 바인딩에 대하여 이해한다.

다형성(Polymorphism)

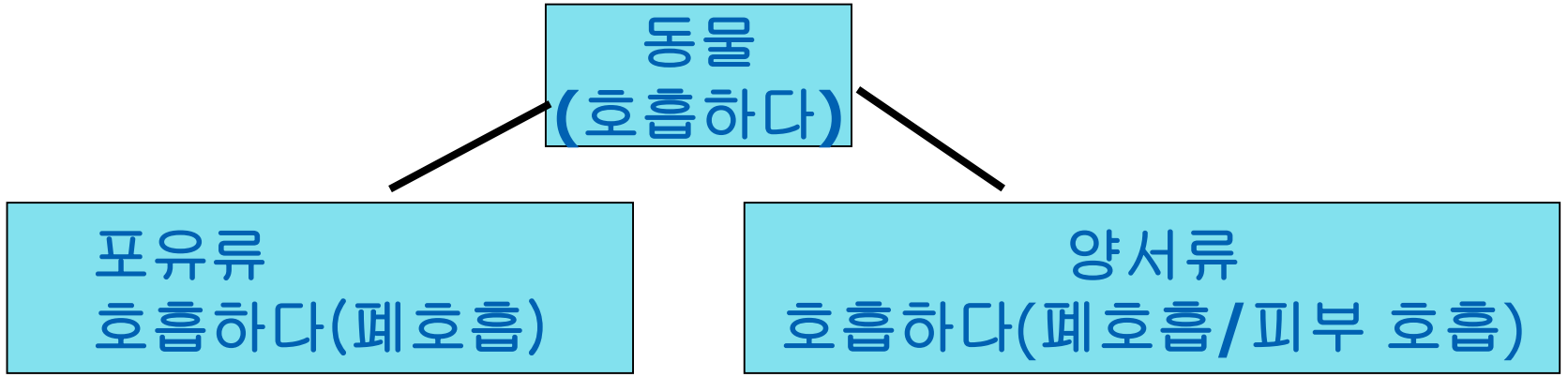
- 다른 분야에서의 다형성의 의미

생물학	동일종의 생물이면서 형태나 성질이 다르게 보이는 다양성. 생물은 본래 동일종이라도 완전히 일치하는 개체는 거의 없으므로, 이 말은 상대적으로 현저한 차이가 있을 경우에 한해서 사용한다. 다만, 암수의 성별에 따른 2차 성징(二次性徵)의 차이에는 쓰지 않는다.
화학	화학조성이 같은 물질로써 결정구조를 달리하는 것. 다형(多形), 동질이형(同質異形), 동질다상(同質多像), 동질이정(同質異晶)이라고도 한다. 동질다상은 결정구조의 수에 따라 동질이상(同質二像), 동질삼상(同質三像) 등으로 나뉜다.

- 객체지향 프로그래밍에서의 다형성이란 타입에 관계없이 동일한 방법으로 다룰 수 있는 능력을 말한다.

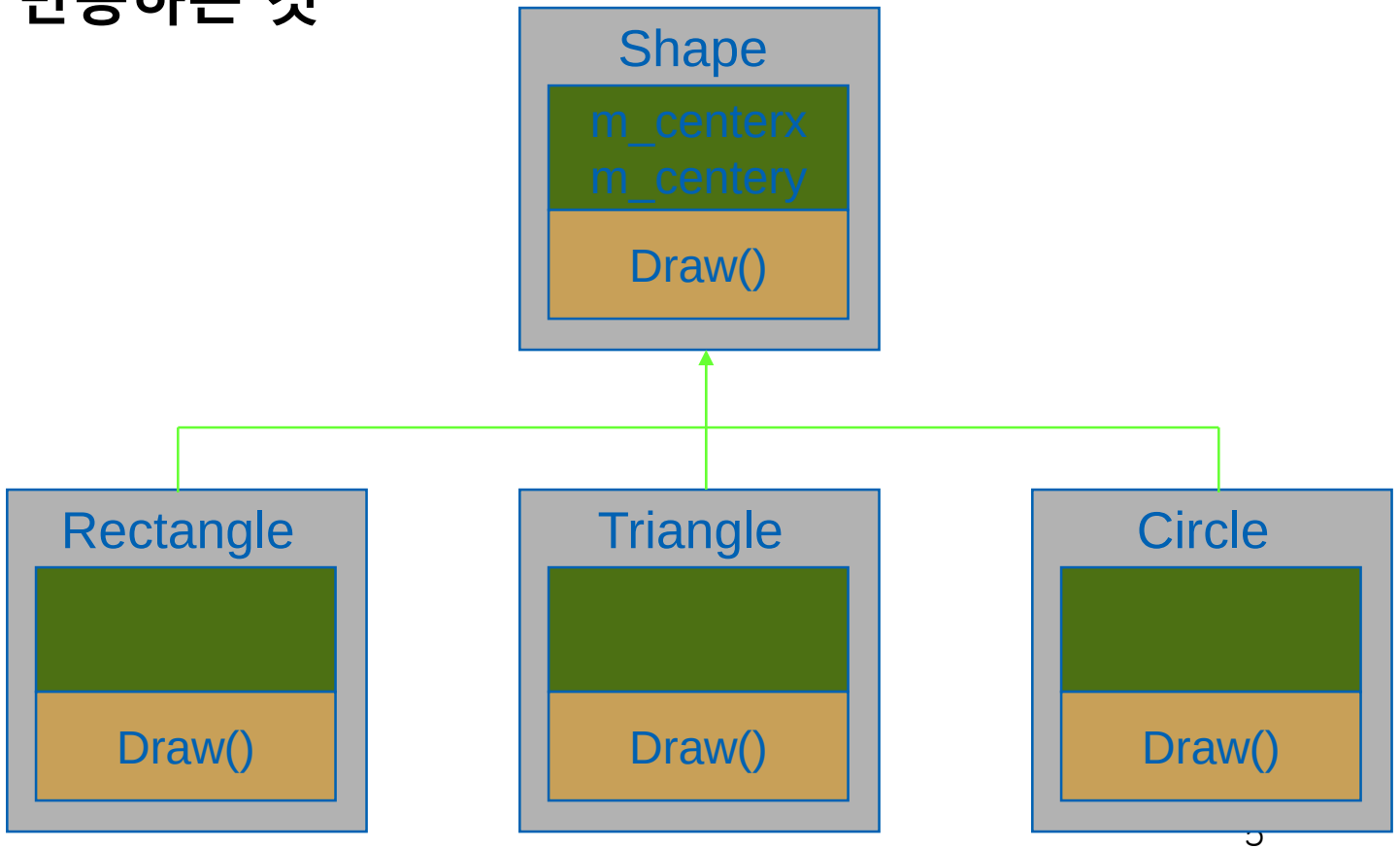
다형성 (Polymorphism)(1)

- 목적은 다르지만 연관성 있는 두 가지 이상의 용도로 하나의 이름을 사용할 수 있게 하는 성질
  - 하나의 인터페이스(Interface)에 여러 방법들
  - 함수의 재정의
    - method overloading(중복)
    - method overriding(재생 또는 치환)



다형성(Polymorphism) (2)

- 서로 다른 객체가 같은 메시지에 대하여 다르게 반응하는 것



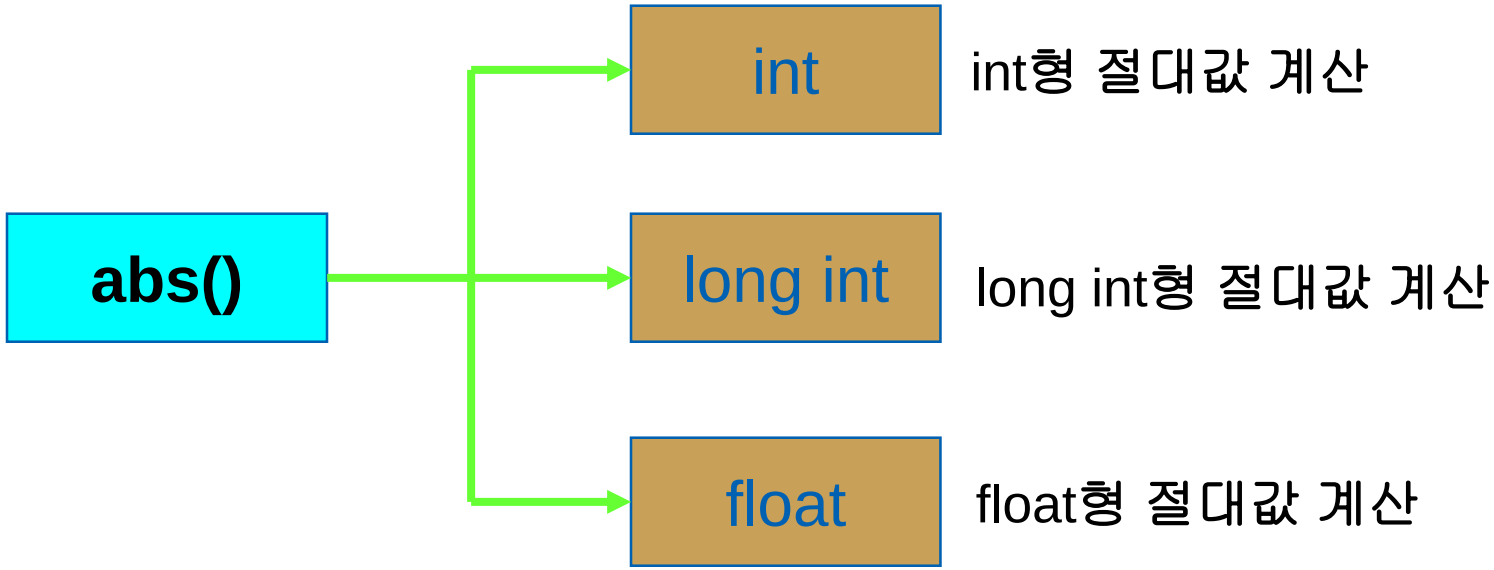
methond 중복 정의 (1)

- 함수 중복 정의(method Overloading)
  - 정의
    - 똑같은 하나의 클래스 안에서 동일한 이름을 가진 method를 여러 번 중복으로 정의하는 것
    - 메서드의 이름과 접근권한지정자가 모두 같음.
  - 중복된 method 구별 방법
    - 파라미터의 개수와 타입으로 구별
    - 파라미터의 개수가 같다면

      - 각 파라미터들의 타입으로 구별함.

methond 중복 정의 (1)

- Method 중복의 정의(method Overloading)



Method Overriding

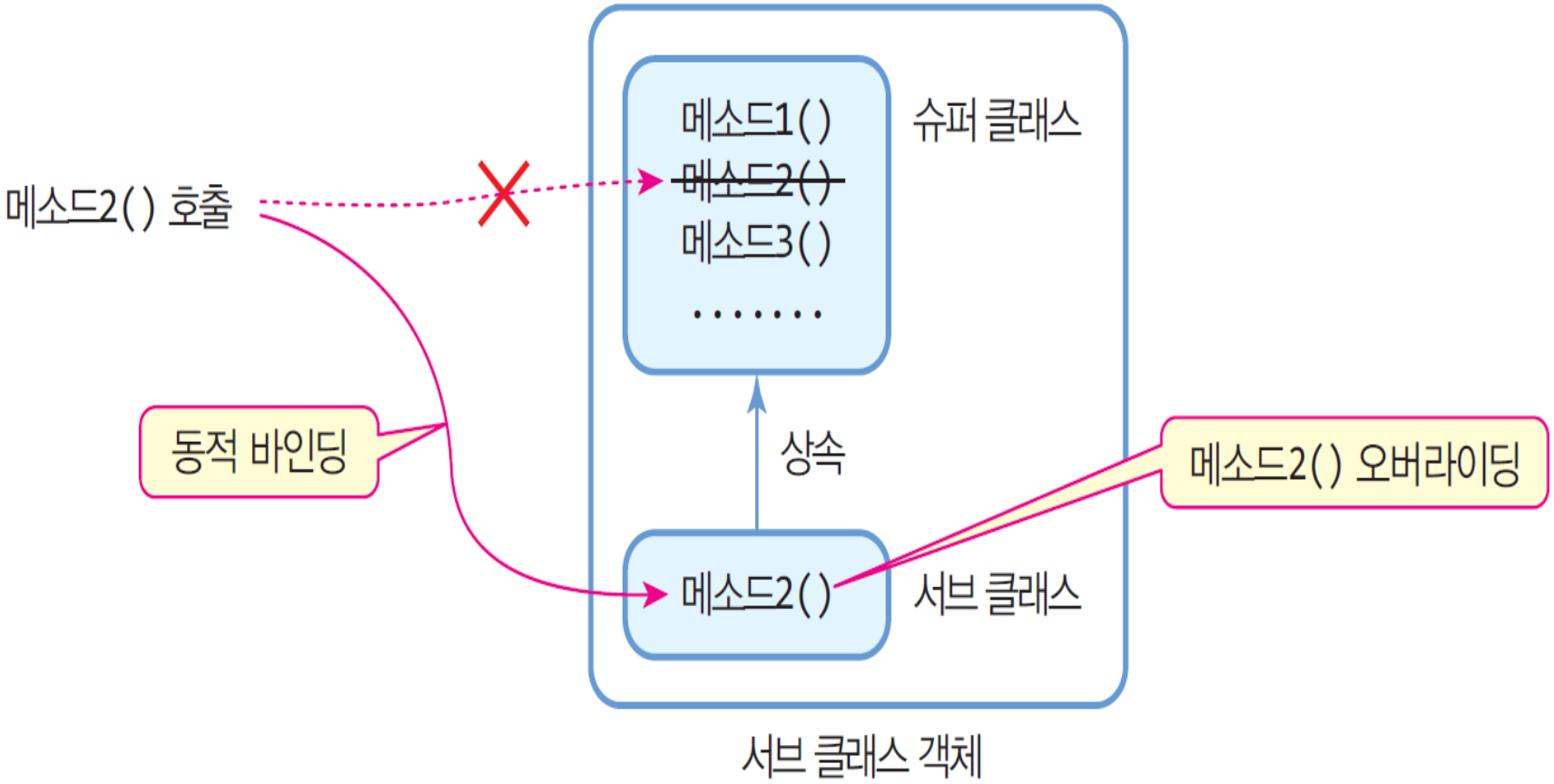
- 메소드 오버라이딩(Method Overriding, 재생)
  - 슈퍼 클래스의 메소드를 서브 클래스에서 재정의
    - 슈퍼 클래스 메소드의 이름, 매개변수 타입 및 개수, 리턴 타입 등 모든 것 동일하게 작성
  - 메소드 무시하기, 덮어쓰기, 재생(再生)로 번역되기도 함
  - 동적 바인딩 발생
    - 서브 클래스에 오버라이딩된 메소드가 무조건 실행되는 동적 바인딩

Method Overriding

- 재생(Overriding)된 메서드의 특징
  - *중복(overloading)된 메서드는* 매개변수의 형(type)과 갯수가 달라야 하지만,
  - *재생된 (overriding) 함수*는 매개변수의 형과 갯수, 그리고 반환값의 형(type)이 정확히 같아야 한다.
  - 실시간 다형성(real-time polymorphism)을 구현함.

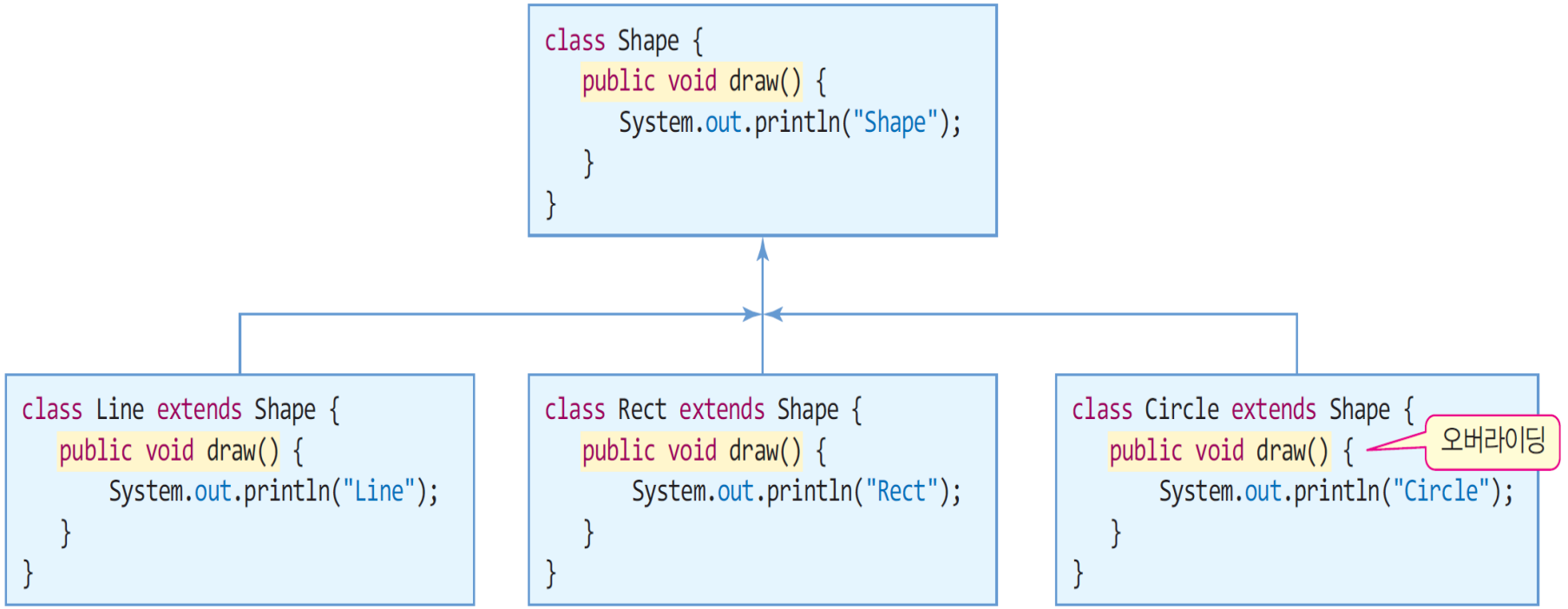
Method Overriding

▪ 메소드 오버라이딩(Method Overriding)



Method Overriding 사례

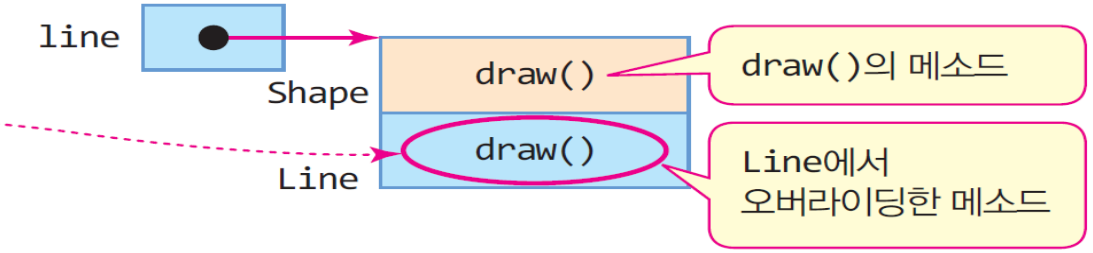
- Shape 클래스의 draw() 메소드를 Line, Rect, Circle 클래스에서 각각 오버라이딩한 사례



Overriding에 의해 서브 클래스의 메소드 호출

(1) 서브 클래스 레퍼런스로 오버라이딩된 메소드 호출

```
Line line = new Line();
line.draw();
```

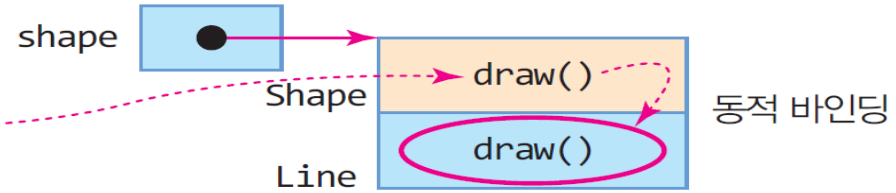


실행 결과

```
Line
```

(2) 업캐스팅에 의해 슈퍼 클래스 레퍼런스로 오버라이딩된 메소드 호출(동적 바인딩)

```
Shape shape = new Line();
shape.draw();
```



실행 결과

```
Line
```

Overriding 의 목적 : 다형성 실현

- Overriding(재생, 재정의)
  - 슈퍼 클래스에 선언된 메소드를 각 서브 클래스들이 자신만의 내용으로 새로 구현하는 기능
  - 상속을 통해 '하나의 인터페이스(같은 이름)에 서로 다른 내용 구현'이라는 객체 지향의 다형성 실현
    - Line 클래스에서 draw()는 선을 그리고
    - Circle 클래스에서 draw()는 원을 그리고
    - Rect 클래스에서 draw()는 사각형 그리게 됨.

Overriding 의 목적 : 다형성 실현

- **Overriding은 실행 시간 다형성 실현**
  - 실시간 다형성(real-time polymorphism)
    - 동적 바인딩을 통해 실행 중에 다형성 실현
      - Overloading은 컴파일 타임시에 다형성 실현

- *중복함수(overloading)*는 매개변수의 형(type)과 개수가 달라야 하지만, *재생된 (overriding) 메소드는* 매개변수의 형과 수, 그리고 반환값의 형(Type)이 정확히 같아야 함.

예제 : 메소드 오버라이딩으로 다형성 실현

```
class Shape { // 슈퍼 클래스
    public Shape next;
    public Shape() { next = null; }

    public void draw() {
        System.out.println("Shape");
    }
}

class Line extends Shape {
    public void draw() { // 메소드 오버라이딩
        System.out.println("Line");
    }
}

class Rect extends Shape {
    public void draw() { // 메소드 오버라이딩
        System.out.println("Rect");
    }
}

class Circle extends Shape {
    public void draw() { // 메소드 오버라이딩
        System.out.println("Circle");
    }
}
```

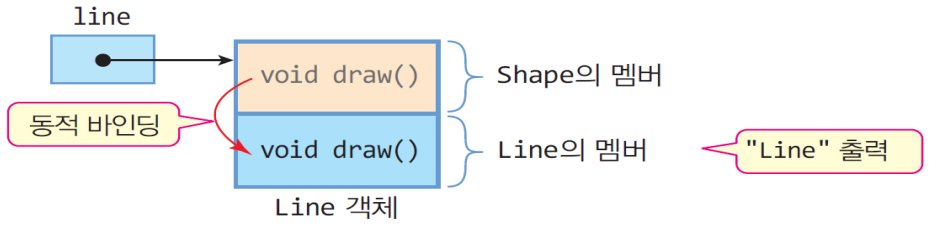
```
public class MethodOverridingEx {
    static void paint(Shape p) {
        p.draw(); // p가 가리키는 객체 내에 오버라이딩된 draw() 호출.
                  // 동적 바인딩
    }

    public static void main(String[] args) {
        Line line = new Line();
        paint(line);
        paint(new Shape());
        paint(new Line());
        paint(new Rect());
        paint(new Circle());
    }
}
```

실행결과

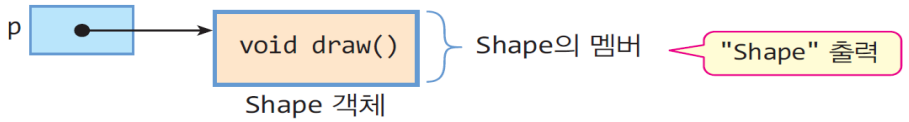
```
Line
Shape
Line
Rect
Circle
```

예제 실행 과정

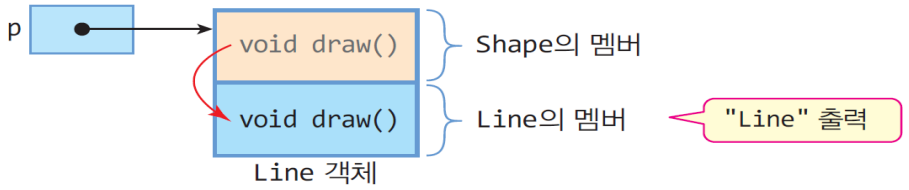
```
Line line = new Line()
paint(line);
```



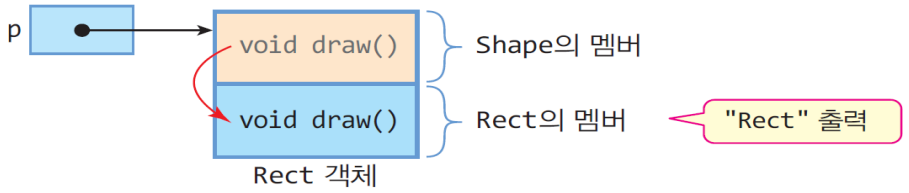

```
paint(new Shape());
```



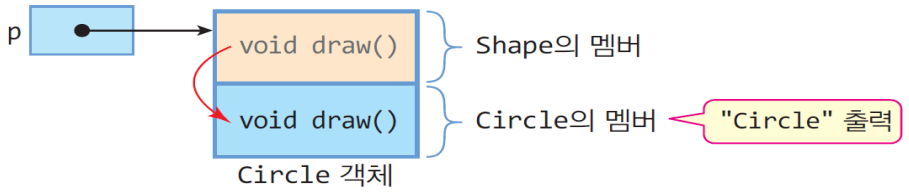

```
paint(new Line());
```




```
paint(new Rect());
```




```
paint(new Circle());
```



동적 바인딩(Dynamic Binding)

- **실행할 메소드를**
  Overriding(재정의)안 한 경우

```
public class Shape {
    protected String name;
    public void paint() {
        draw();
    }
    public void draw() {
        System.out.println("Shape");
    }
    public static void main(String [] args) {
        Shape a = new Shape();
        a.paint();
    }
}
```

실행 결과

```
Shape
```

동적 바인딩

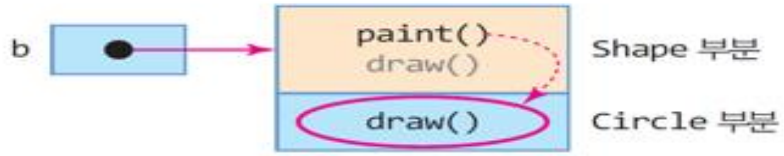
- 실행할 메소드를 실행 시(run time)에 결정
  - Overriding 된 메소드가 항상 호출

```
class Shape {
    protected String name;
    public void paint() {
        draw();
    }
    public void draw() {
        System.out.println("Shape");
    }
}
public class Circle extends Shape {
    @Override
    public void draw() {
        System.out.println("Circle");
    }
    public static void main(String [] args) {
        Shape b = new Circle();
        b.paint();
    }
}
```

동적 바인딩

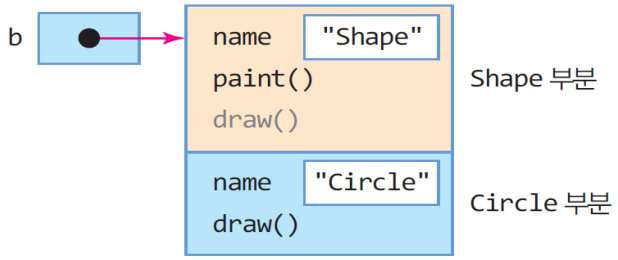
실행 결과

```
Circle
```



Overriding과 super 키워드

- super는 슈퍼 클래스의 멤버를 접근할 때 사용되는 레퍼런스
- 서브 클래스에서만 사용
- 슈퍼 클래스의 메소드 호출
- 컴파일러는 super의 접근을 정적 바인딩으로 처리




```java
class Shape {
    protected String name;
    public void paint() {
        draw();
    }
    public void draw() {
        System.out.println(name);
    }
}
public class Circle extends Shape {
    protected String name;
    @Override
    public void draw() {
        name = "Circle";
        super.name = "Shape";
        super.draw();
        System.out.println(name);
    }
    public static void main(String [] args) {
        Shape b = new Circle();
        b.paint();
    }
}
```

정적 바인딩

실행 결과

```
Shape
Circle
```

오버라이딩 vs. 오버로딩

비교 요소	메소드 오버로딩	메소드 오버라이딩
선언	같은 클래스나 상속 관계에서 동일한 이름의 메소드 중복 작성	서브 클래스에서 슈퍼 클래스에 있는 메소드와 동일한 이름의 메소드 재작성
관계	동일한 클래스 내 혹은 상속 관계	상속 관계
목적	이름이 같은 여러 개의 메소드를 중복 작성하여 사용의 편리성 향상. 다형성 실현	슈퍼 클래스에 구현된 메소드를 무시하고 서브 클래스에서 새로운 기능의 메소드를 재정의하고자 함. 다형성 실현
조건	메소드 이름은 반드시 동일하고, 매개변수 타입이나 개수가 달라야 성립	메소드의 이름, 매개변수 타입과 개수, 리턴 타입이 모두 동일하여야 성립
바인딩	정적 바인딩. 호출될 메소드는 컴파일 시에 결정	동적 바인딩. 실행 시간에 오버라이딩된 메소드 찾아 호출

핵심정리

- 메소드 오버라이딩(Method Overriding, 재생)
  - 슈퍼 클래스의 메소드를 서브 클래스에서 재정의

    -슈퍼 클래스 메소드의 이름, 매개변수 타입 및 개수, 리턴 타입 등 모든 것 동일하게 작성
  - 메소드 무시하기, 덮어쓰기, 재생(再生)로 번역되기도 함
  - 동적 바인딩 발생

    서브 클래스에 오버라이딩된 메소드가 무조건 실행되는 동적 바인딩
- Overriding은 실행 시간 다형성 실현
  - 실시간 다형성(real-time polymorphism)
- Overloading은 컴파일 타임시에 다형성 실현