어서와 Java는 처음이지!

제6장 클래스, 메소드 심층연구

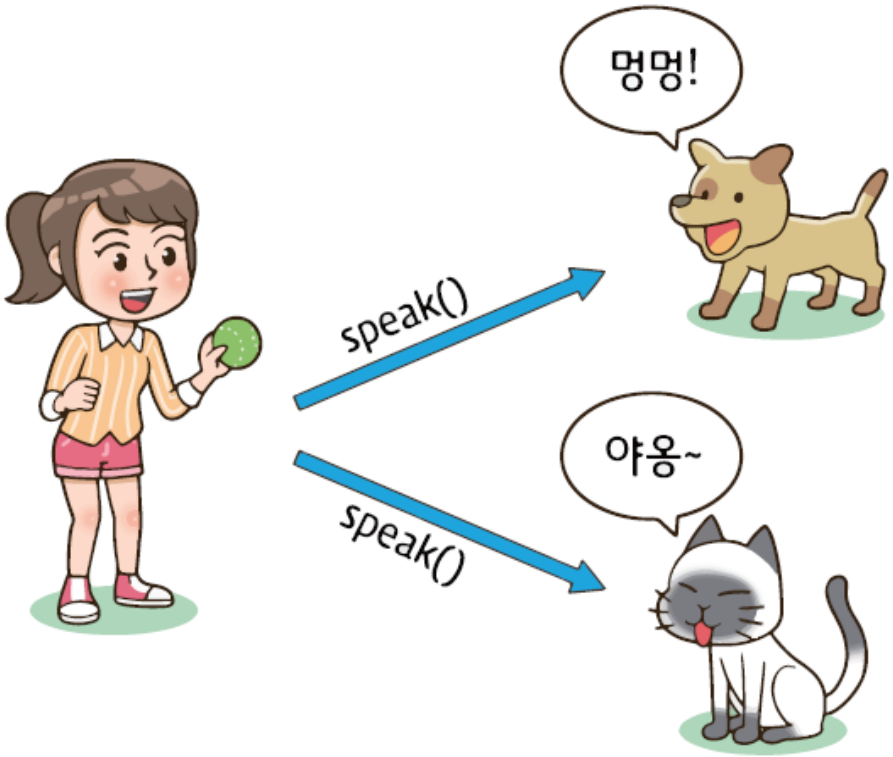
- 다형성
- 오버로딩
  - 메소드 오버로딩
  - 생성자 오버로딩

다형성

- 하나의 이름(방법)으로 많은 상황에 대처하는 기법
  - 개념적으로 동일한 작업을 하는 멤버 함수들에 똑같은 이름을 부여할 수 있다
  - 그러므로 코드가 더 간단해진다

다형성

- 하나의 이름(방법)으로 많은 상황에 대처하는 기법

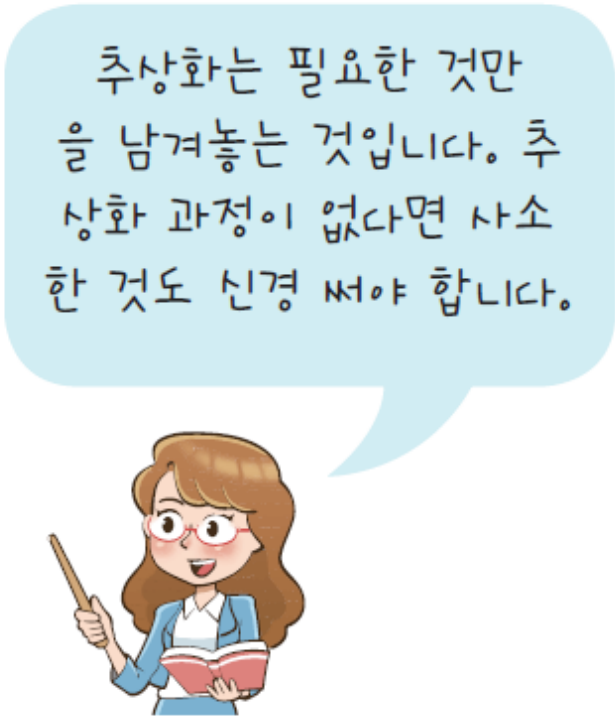


다형성은 객체의 동작이 상황에 따라서 달라지는 것을 말합니다. "speak"라는 메시지를 받은 객체들이 모두 다르게 소리를 내는 것이 바로 다형성입니다.

추상화

실제 객체

추상화된 객체



객체 지향의 장점

- 신뢰성있는 소프트웨어를 쉽게 작성할 수 있다.
- 코드를 재사용하기 쉽다.
- 업그레이드가 쉽다.
- 디버깅이 쉽다.

쉬운 디버깅

- 예를 들어서 절차 지향 프로그램에서 하나의 변수를 1000개의 함수가 사용하고 있다고 가정해보자.

- -> 하나의 변수를 1000개의 함수에서 변경할 수 있다.

쉬운 디버깅

- 객체 지향 프로그램에서 100개의 클래스가 있고 클래스당 10개의 메소드를 가정해보자.
- -> 하나의 변수를 10개의 메소드에서 변경할 수 있다.

- 어떤 방법이 디버깅이 쉬울까?

중간 점검 문제

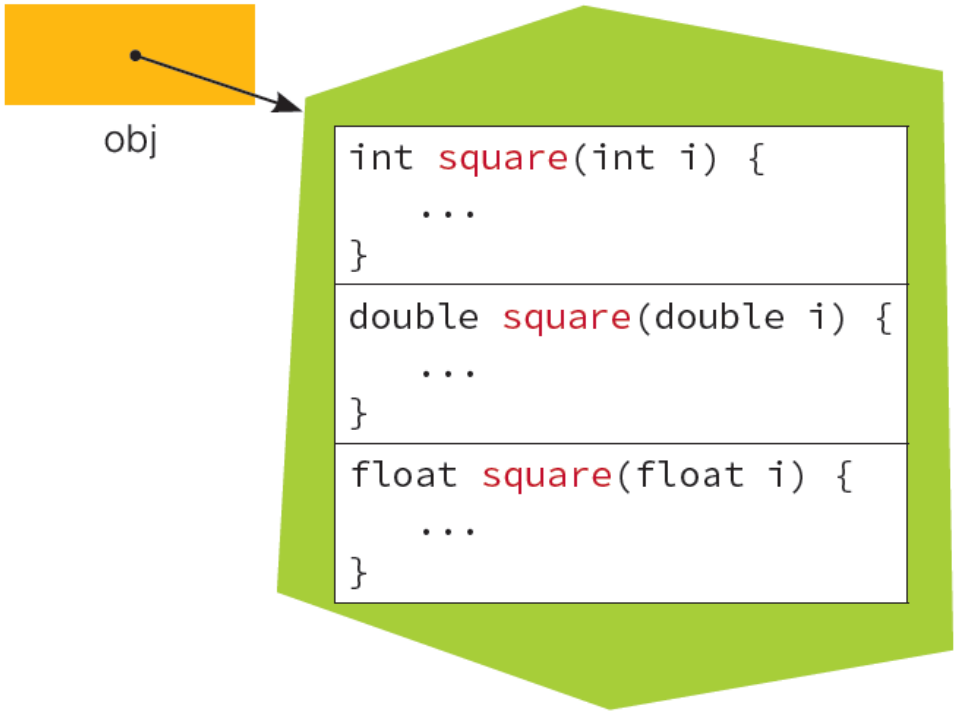
1. 자바에서 코드 재사용이 쉬운 이유는 관련된 _____와 _____이 하나의 덩어리로 묶여 있기 때문이다.

2. 정보 은닉이란 _____을 외부로부터 보호하는 것이다.

3. 정보를 은닉하면 발생하는 장점은 무엇인가?

메소드 오버로딩

- 자바에서는 같은 이름의 메소드가 여러 개 존재할 수 있다. 이것을 **메소드 오버로딩(method overloading)**이라고 한다.



예제

```
public class MyMath {
	// 정수값을 제곱하는 메소드
	int square(int i) {
		return i * i;
	}
	// 실수값을 제곱하는 메소드
	double square(double i) {
		return i * i;
	}
}
```

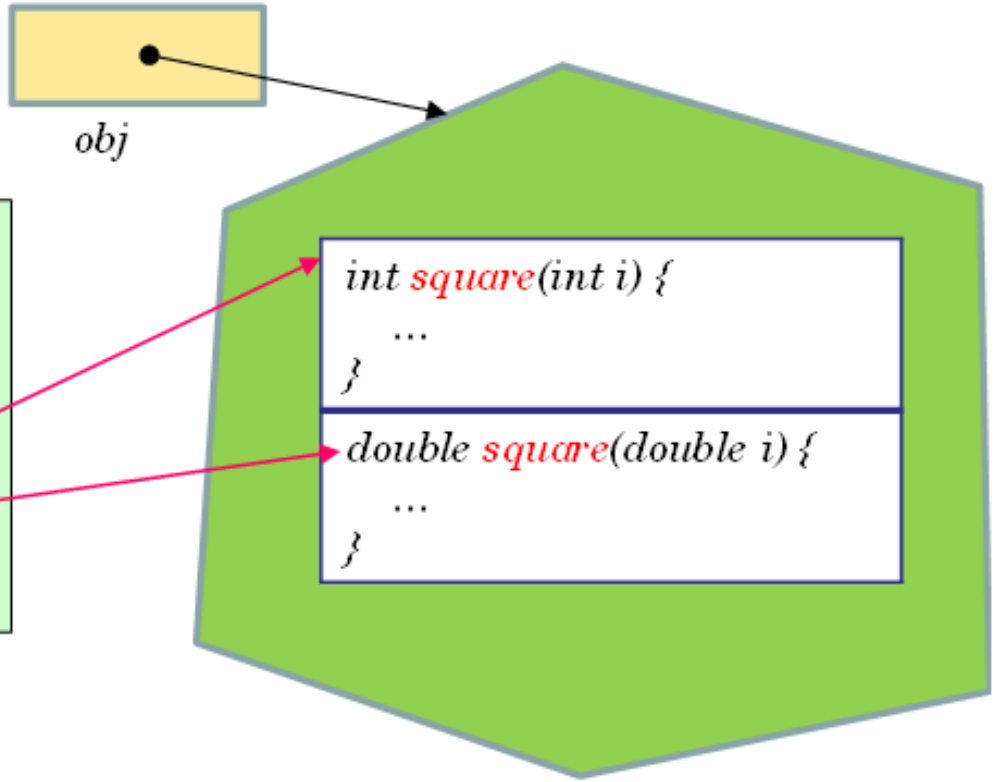
예제

```
public class MyMathTest {
	public static void main(String args[]) {
		MyMath obj = new MyMath();
		System.out.println(obj.square(10));
		System.out.println(obj.square(3.14));
	}
}
```


```
100
9.8596
```

예제 설명

```
MyMath obj = new MyMath();

System.out.println(obj.square(10));
System.out.println(obj.square(3.14));
```



생성자 오버로딩

- 메소드처럼 생성자도 오버로딩될 수 있다.

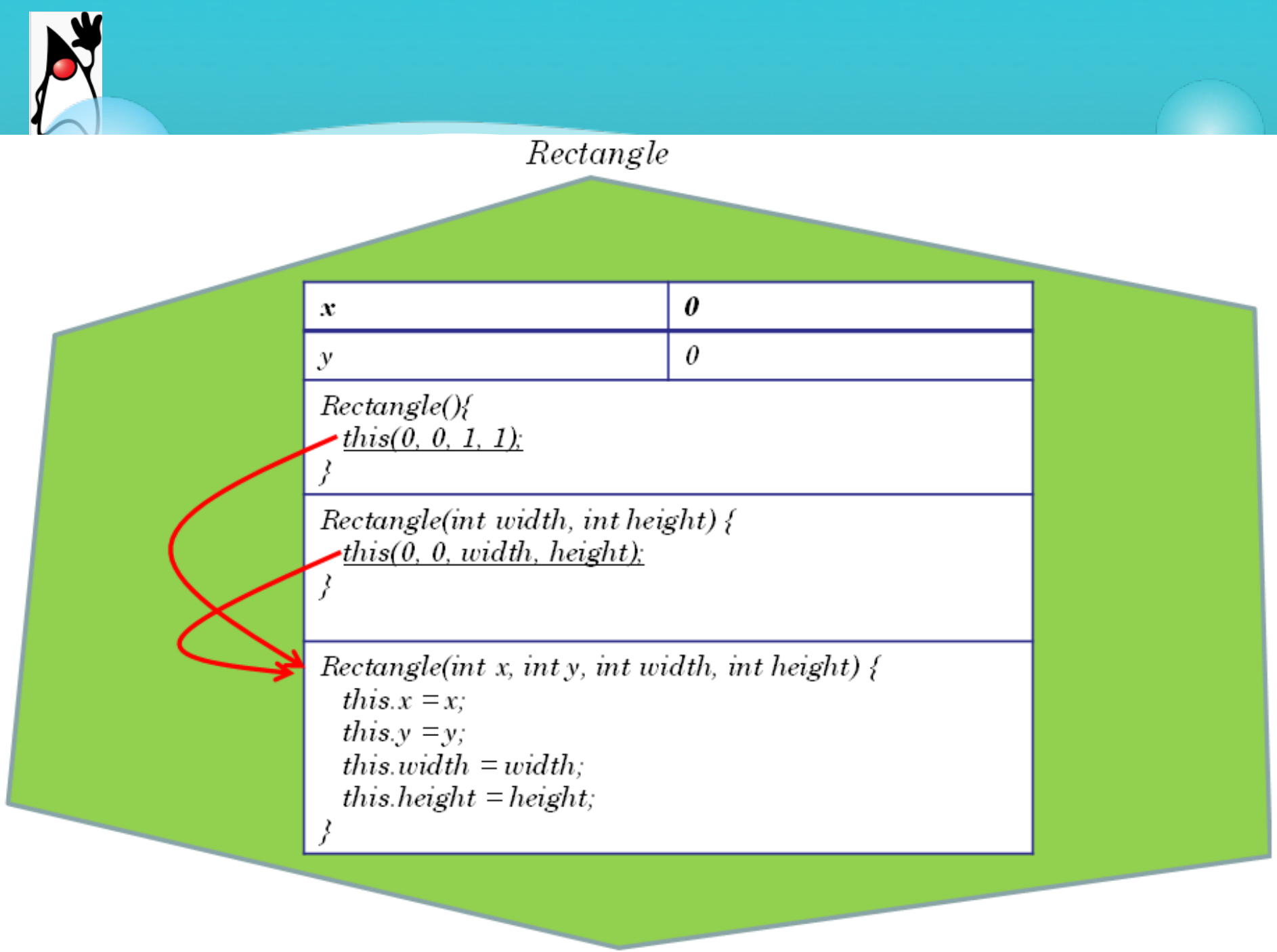
```java
public class Student {
	private int number;
	private String name;
	private int age;
	Student() {
		number = 100;
		name = "New Student";
		age = 18;
	}
	Student(int number, String name, int age) {
		this.number = number;
		this.name = name;
		this.age = age;
	}
	@Override
	public String toString() {
		return "Student [number=" + number +
				", name=" + name +
				", age=" + age + "]";
	}
}
```

this로 현재 객체 나타내기

- 메소드나 생성자에서 this는 현재 객체를 나타낸다.

```
public class Point {
    public int x = 0;
    public int y = 0;

    // 생성자
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

```
public class Rectangle {
	private int x, y;
	private int width, height;

	Rectangle() {
		this(0, 0, 1, 1);
	}

	Rectangle(int width, int height) {
		this(0, 0, width, height);
	}

	Rectangle(int x, int y, int width, int height) {
		this.x = x;
		this.y = y;
		this.width = width;
		this.height = height;
	}
	// ...
}
```


Rectangle
x
0
y
0
Rectangle(){
this(0, 0, 1, 1);
}
Rectangle(int width, int height) {
this(0, 0, width, height);
}
Rectangle(int x, int y, int width, int height) {
this.x = x;
this.y = y;
this.width = width;
this.height = height;
}

LAB 4-3: 날짜를 나타내는 Date 클래스

- 다음의 필드를 가진 Date 클래스에서 가능한 모든 생성자들을 작성해 보자

```
public class Date {
        private int year;
        private String month;
        private int day;

        }
```

Q & A