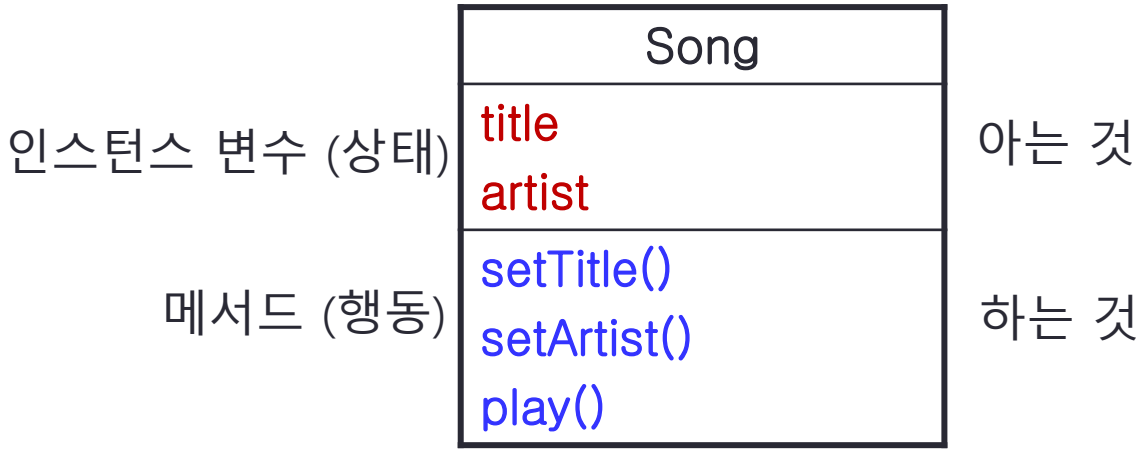
4장. 객체의 행동

- 인스턴스 변수와 메서드
- 매개변수와 리턴값
- 객체의 동치

객체의 행동

상태 – 인스턴스 변수
행동 – 메서드

객체의 상태와 행동



Q: 모든 객체의 인스턴스 변수(상태)는 달라질 수 있습니다. 그런데 메서드(행동)도 달라질 수 있을까요?

A: 메서드 자체는 똑같지만 그 메서드의 실행 결과는 인스턴스 변수 값에 따라서 달라질 수 있습니다.

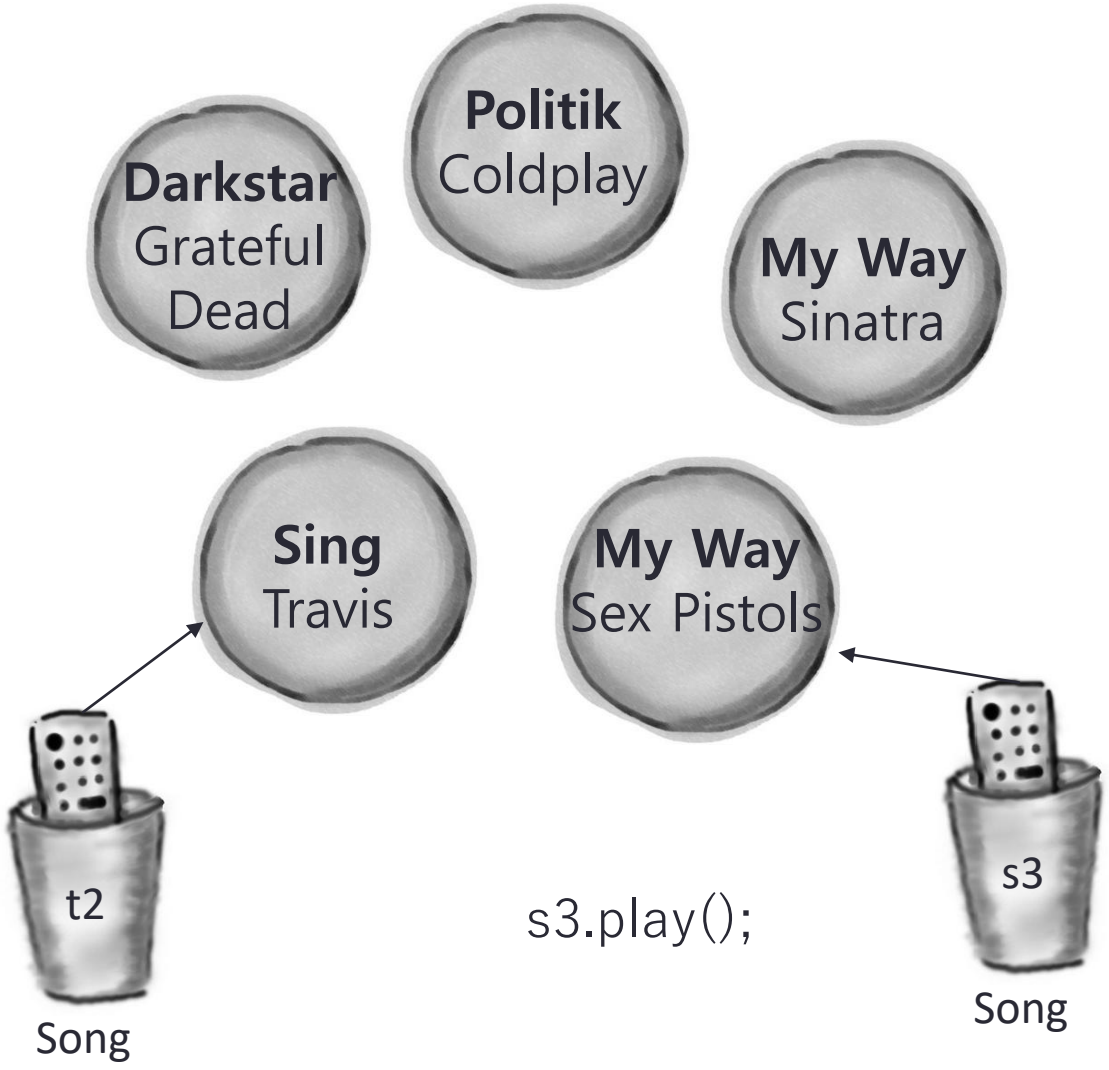
객체의 상태와 행동

```
void play() {
  soundPlayer.playSound(title);
}
```

```
Song t2 = new Song();
t2.setArtist("Travis");
t2.setTitle("Sing");

Song s3 = new Song();
s3.setArtist("Sex Pistols");
s3.setTitle("My Way");
```

t2.play();

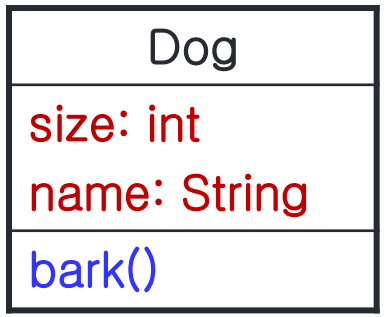
s3.play();

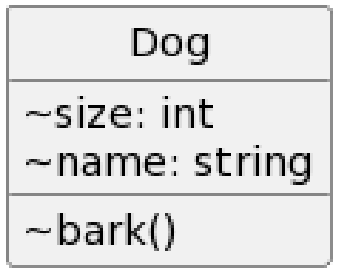


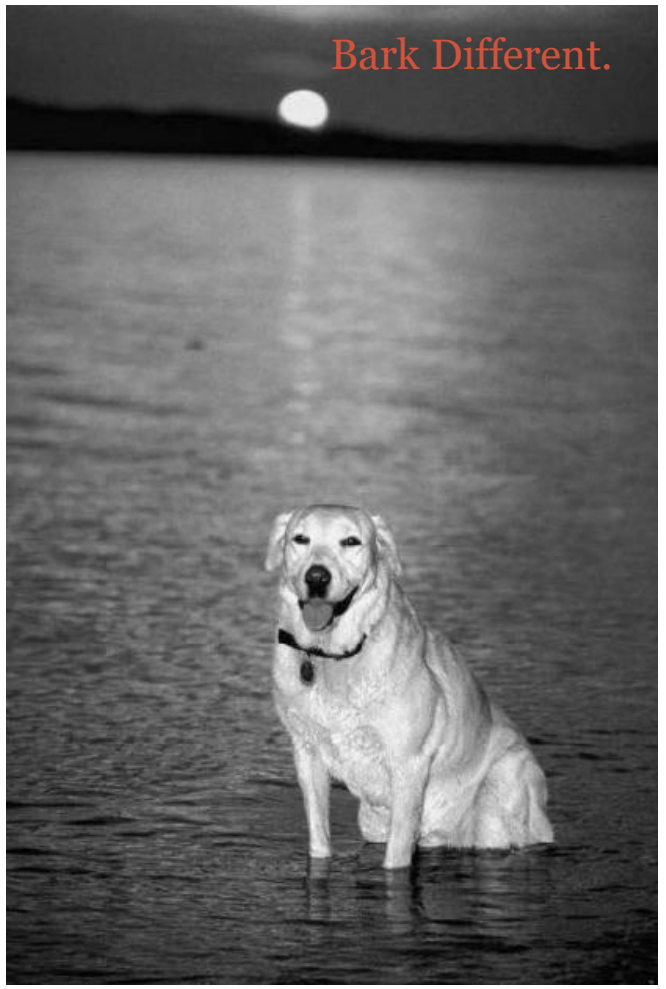
p.114

객체의 상태와 행동

```
class Dog {
  int size;
  String name;

  void bark() {
    if (size > 60) {
      System.out.println("Woof! Woof!");
    } else if (size > 14) {
      System.out.println("Ruff! Ruff!");
    } else {
      System.out.println("Yip! Yip!");
    }
  }
}
```








```
class DogTestDrive {
  public static void main(String[] args) {
    Dog one = new Dog();
    one.size = 70;
    Dog two = new Dog();
    two.size = 8;
    Dog three = new Dog();
    three.size = 35;

    one.bark();
    two.bark();
    three.bark();
  }
}
```

[실행 결과]
Woof! Woof!
Yip! Yip!
Ruff! Ruff!

p.115

코드를 직접 실행시켜봅시다.

Dog.java

```java
package cse.oop2.ch04.diff_behavior;

/**
 * TestDrive의 main() 메소드를 통합
 * @author Prof.Jong Min Lee
 */
public class Dog {
    int size;
    String name;

    void bark() {
        if (size > 60) {
            System.out.println("Woof! Woof!");
        } else if (size > 14) {
            System.out.println("Ruff! Ruff!");
        } else {
            System.out.println("Yip! YTip!");
        }
    }
```

```java
    /**
     * DogTestDrive 클래스의 main()과 동일
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        Dog one = new Dog();
        one.size = 70;
        Dog two = new Dog();
        two.size = 8;
        Dog three = new Dog();
        three.size = 35;

        one.bark();
        two.bark();
        three.bark();
    }

}
```

```
--- exec-maven-plugin:
Woof! Woof!
Yip! YTip!
Ruff! Ruff!
```

매개변수/인자*

- 자바에서도 메서드에 어떤 값을 전달할 수 있습니다.

- **매개변수 (parameter)**
  - 메서드 정의 시 선택적으로 정의하여 사용 가능
  - 메서드 안에서는 지역 변수와 동일: 값 할당도 가능
- **인수 (argument)**
  - 호출하는 쪽에서 전달하는 실제 값을 의미

- 메서드에 매개변수가 있으면 반드시 해당 유형의 값을 전달해야만 합니다.

매개변수/인자*

1. bark 메서드 호출 (인자로 3을 전달)

```
Dog d = new Dog();
d.bark(3);
```

인수

매개변수

```
void bark(int numOfBarks) {
  while (numOfBarks > 0) {
    System.out.println("ruff");
    numOfBarks = numOfBarks - 1;
  }
}
```

2. 3이라는 값을 나타내는 비트들이 bark 메서드로 전달됨

3. 그 비트들이 numOfBarks 매개변수에 들어감

4. numOfBarks 매개변수를 메서드 코드 내에서 변수로 사용

리턴값*

- 메서드로부터 값을 받을 수도 있습니다.

```
void go() {
}
```

```
int giveSecret() {
   return 42;
}
```

- 리턴 유형은 메서드를 선언할 때 지정하며, 리턴 유형이 정해져 있으면 반드시 그 유형의 값을 리턴해야만 합니다.

p.117

리턴값

```
int theSecret = life.giveSecret();
```

```
int giveSecret() {
   return 42;
}
```

두 개 이상의 매개변수

- 메서드에 두 개 이상의 인자를 전달할 수도 있습니다.
  - 각 인자는 **쉼표로 구분**합니다.
  - 메서드에 매개변수가 있을 때 반드시 인자를 전달해야 한다는 원칙은 여전히 적용됩니다.

p.118

두 개 이상의 매개변수

```
void go() {
  TestStuff t = new TestStuff();
  t.takeTwo(12, 34);
}
```

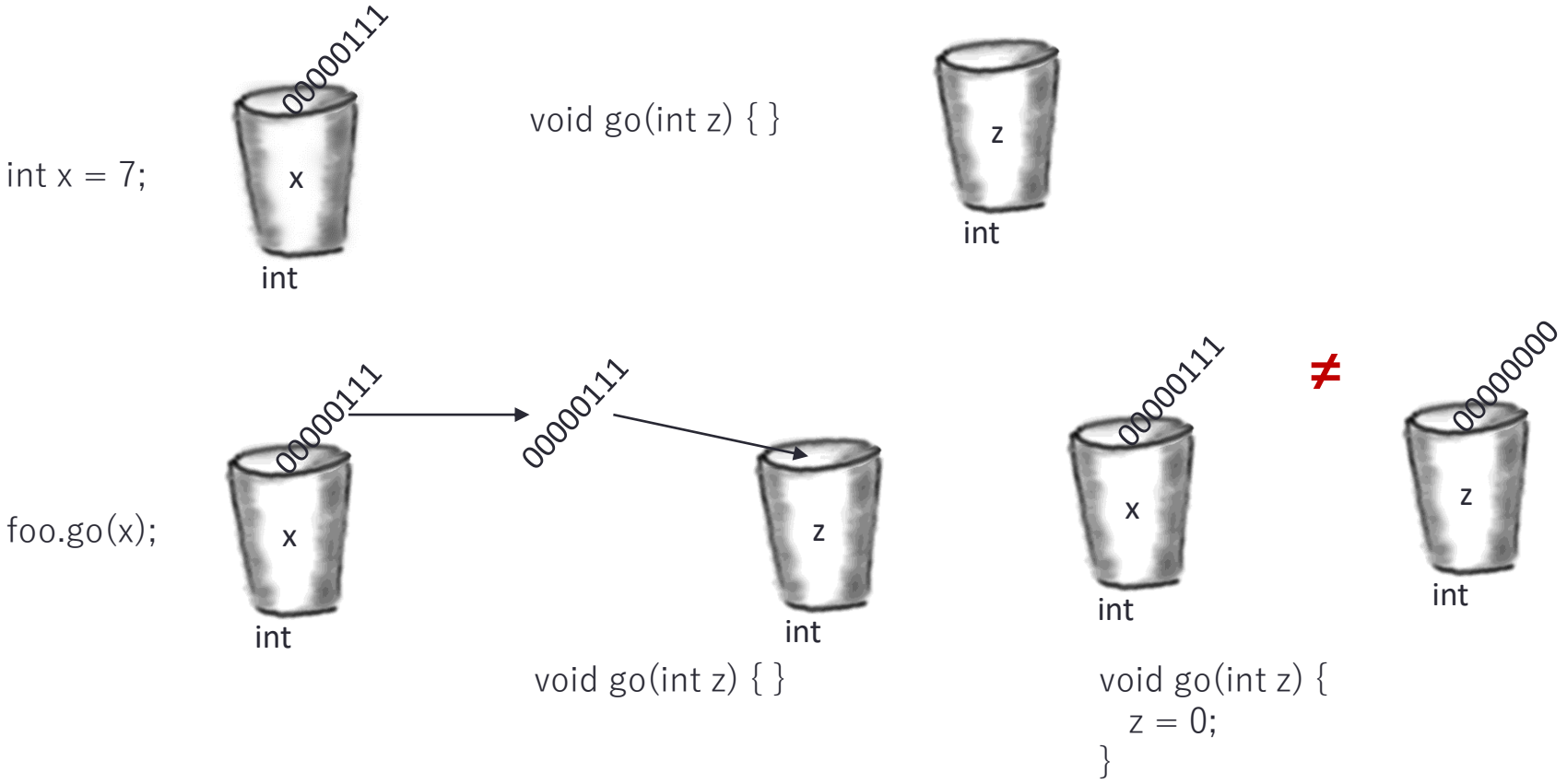
```
void go() {
  int foo = 7;
  int bar = 3;
  t.takeTwo(foo, bar);
}
```

```
void takeTwo(int x, int y) {
  int z = x + y;
  System.out.println("Total is " + z);
}
```

- 자바에서는 값으로 전달하는 방식을 사용합니다. (pass by value)

p.119

값으로 전달 (pass by value)



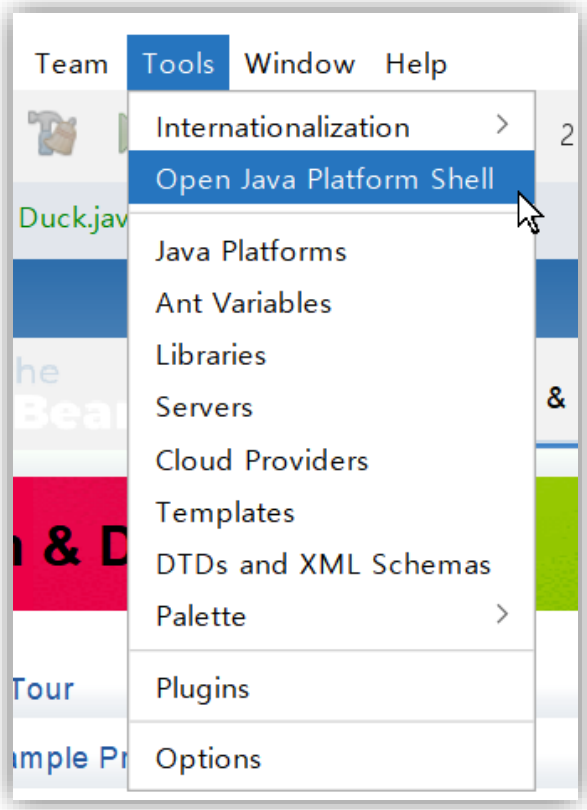
바보 같은 질문은 없습니다.

- 전달하려고 하는 인자가 원시변수가 아니고 객체인 경우에는 어떻게 되나요?
  - 객체 레퍼런스를 인자로 전달하는 경우에도 여전히 값으로만 전달됩니다. 중요한 것은 여기에서 "값"이라는 것이 "객체"가 아니라, 객체를 참조하는 레퍼런스라는 점입니다. 따라서 그 레퍼런스의 복사본이 전달됩니다.

- 메서드에서 리턴 값을 여러 개 선언할 수 있나요? 값을 두 개 이상 리턴하는 방법이 있나요?
  - 리턴값은 한 가지밖에 선언할 수 없습니다. 하지만 세 개의 int를 리턴하고 싶다면 리턴 유형을 int 배열로 선언하면 됩니다. 여러 유형의 값을 리턴하는 방법은 조금 더 복잡한데, 나중에 배우게 될 ArrayList라는 것을 사용하면 됩니다.
  - 서로 다른 자료형을 가지는 여러 개의 값을 반환하고자 할 경우 클래스를 이용하면 됩니다. 따라서 클래스는 자료형(+연산)임을 알 수 있습니다.

- 정확하게 처음에 선언한 유형으로만 리턴해야 하나요?
  - 자동으로 해당 유형으로 변환되는 것은 그냥 리턴해도 됩니다.
  - 예: int를 리턴하겠다고 선언한 경우에 byte를 리턴해도 됩니다.
  - 하지만 그렇지 않은 경우에는 강제로 형 변환(type casting)을 해야 합니다.
  - 예: return (int) (Math.random() * 10.0);

```
jshell> int a = 10; byte b = a;
a ==> 10
|  Error:
|  incompatible types: possible lossy conversion from int to byte
|   byte b = a;
|              ^

jshell> byte b = 10; int a = b;
b ==> 10
a ==> 10
```



  - 참고) 자바 쉘 실행 방법: 명령창(cmd.exe)에서 jshell.exe 실행해도 됨

- 메서드에서 리턴한 값으로 반드시 뭔가를 해야 하나요? 그냥 무시하면 안 되나요?
  - 자바에서는 **리턴 값의 사용 여부에는 전혀 신경을 쓰지 않습니다**. 따라서 리턴값을 꼭 어떤 변수에 대입한다거나 특정한 용도로 사용하지 않아도 됩니다. 예를 들어 리턴값은 별로 필요 없고 메서드로 어떤 작업을 하기만 되는 경우에는 리턴 값은 그냥 무시하고 메서드를 호출하기만 해도 됩니다.
  - 하지만 원래 **메서드 정의 시 반환 값을 정의해 놓은 목적을 생각해 보면 반드시 사용**하는 것이 좋습니다.

핵심 정리

- 클래스에서는 객체가 하는 것과 객체가 아는 것을 정의합니다.
- **인스턴스 변수(상태)**는 객체가 아는 것입니다.
- **메서드(행동)**는 객체가 하는 것입니다.
- 메서드에서 인스턴스 변수를 이용하여 같은 형식의 객체가 다른 식으로 행동하도록 할 수 있습니다.
- 메서드에서 매개변수를 사용할 수 있습니다. 즉 메서드에 한 개 이상의 값을 전달할 수 있습니다.

- 전달하는 값의 개수와 유형은 반드시 메서드를 선언할 때 지정한 것과 같아야 하며 그 순서도 같아야 합니다.
- 메서드 안팎으로 전달되는 값은 상황에 따라 자동으로 더 큰 유형으로 올라갈 수 있습니다. 더 작은 유형으로 바꿀 때는 강제로 캐스팅을 해야 합니다.
- 메서드에 인자를 전달할 때는 리터럴 값(2, ‘c’ 등)을 사용할 수도 있고 선언된 매개변수 유형의 변수(예를 들어 int 변수 x)를 사용할 수도 있습니다.
- 메서드를 선언할 때는 반드시 리턴 유형을 지정해야 합니다. 리턴 유형을 void로 지정하면 아무 것도 리턴하지 않아도 됩니다.
- 메서드를 선언할 때 void가 아닌 리턴 유형을 지정했을 때는 반드시 선언된 리턴 유형과 호환 가능한 값을 리턴해야 합니다.

게터(getter)와 세터(setter)

- 게터(getter)
  - 인스턴스 변수의 값을 알아내기 위한 메서드
  - 일반적으로 인스턴스 변수의 값을 리턴함
  - getBrand(), getNumOfPickups()…
- 세터(setter)
  - 인스턴스 변수의 값을 설정하기 위한 메서드
  - 전달된 값을 확인하고 인스턴스 변수의 값을 설정함
  - setBrand(), setNumOfPickups()…

게터와 세터

```
class ElectricGuitar {
  String brand;
  int numOfPickups;
  boolean rockStarUsesIt;

  String getBrand() {
    return brand;
  }

  void setBrand(String aBrand) {
    brand = aBrand;
  }

  int getNumOfPickups() {
    return numOfPickups;
  }

  void setNumOfPickups(int num) {
    numOfPickups = num;
  }

  boolean getRockStarUsesIt() {
    return rockStarUsesIt;
  }

  void setRockStarUsesIt(boolean yesOrNo) {
    rockStarUsesIt = yseOrNo;
  }
}
```

프로퍼티 (Property)*

- (참고) 객체지향설계 4장 강의 자료 중에서...

- 구성 요소
  - 애트리뷰트
  - getter/setter 메서드

- 종류
  - 단순 프로퍼티 (simple property)
  - 부울 프로퍼티 (boolean property)
  - 색인 프로퍼티 (indexed property)

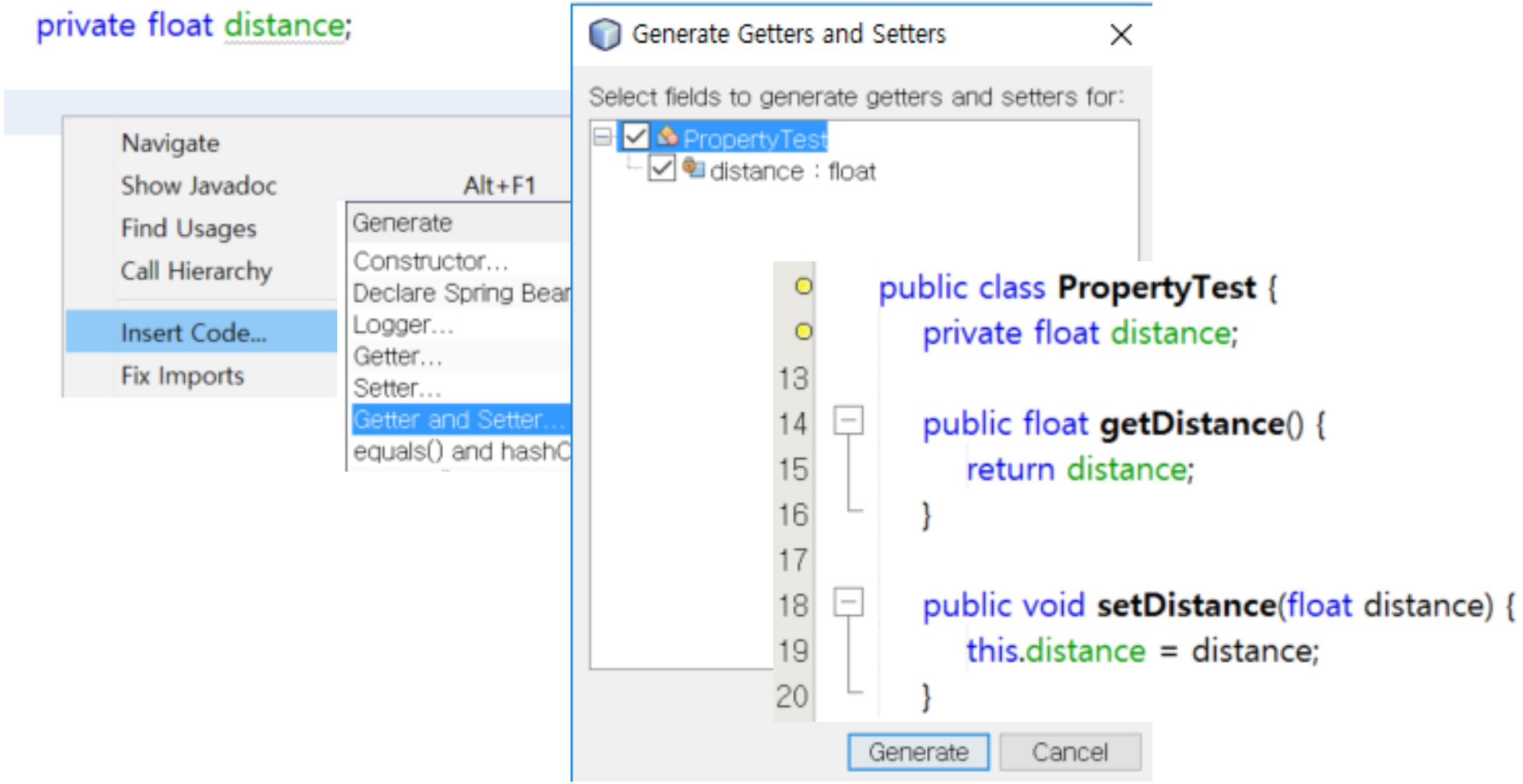
- 참고 문헌: JavaBeans Specification 1.01 (https://download.oracle.com/otndocs/jcp/7224-javabeans-1.01-fr-spec-oth-JSpec/)

단순 프로퍼티

단순 프로퍼티	
애트리뷰트	private <PropertyType> <propertyName>;
접근 메소드	public <PropertyType> get<PropertyName>(); public void set<PropertyName>(<PropertyType> a);

사용 예	private float distance;
	
	public float getDistance();
	public void setDistance(float distance);

단순 프로퍼티 생성 방법


private float distance;
Navigate
Show Javadoc Alt+F1
Find Usages
Call Hierarchy
Insert Code...
Fix Imports
Generate
Constructor...
Declare Spring Bean
Logger...
Getter...
Setter...
Getter and Setter...
equals() and hashC
Generate Getters and Setters
Select fields to generate getters and setters for:
PropertyTest
distance : float
public class PropertyTest {
private float distance;
public float getDistance() {
return distance;
}
public void setDistance(float distance) {
this.distance = distance;
}
Generate
Cancel

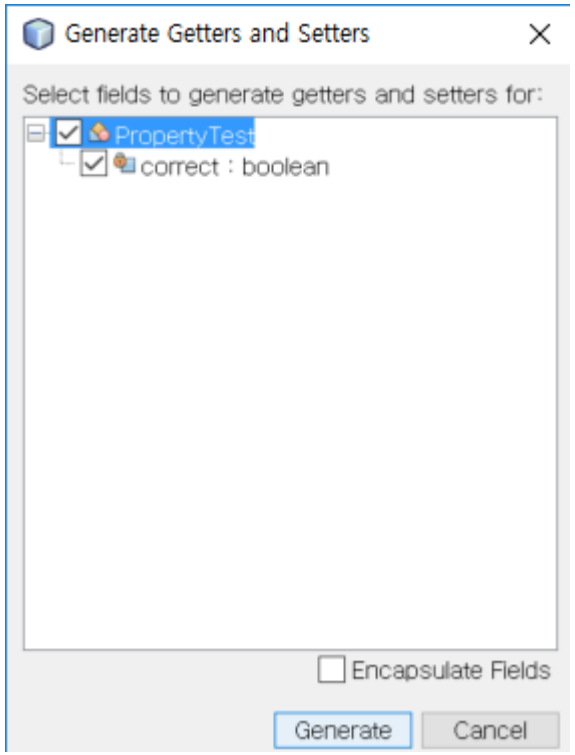
부울 프로퍼티

부울 프로퍼티	
애트리뷰트	private boolean <propertyName>;
접근 메소드	public boolean is<PropertyName>(); public void set<PropertyName>(boolean a);

사용 예	private boolean correct;
	
	public boolean isCorrect();
	public void setCorrect(boolean correct);

부울 프로퍼티 생성 방법

- MRB (Mouse Right Button) > Insert Code... > Getter and Setter... 선택




```
public class PropertyTest {
    private boolean correct;

    public boolean isCorrect() {
        return correct;
    }

    public void setCorrect(boolean correct) {
        this.correct = correct;
    }
```

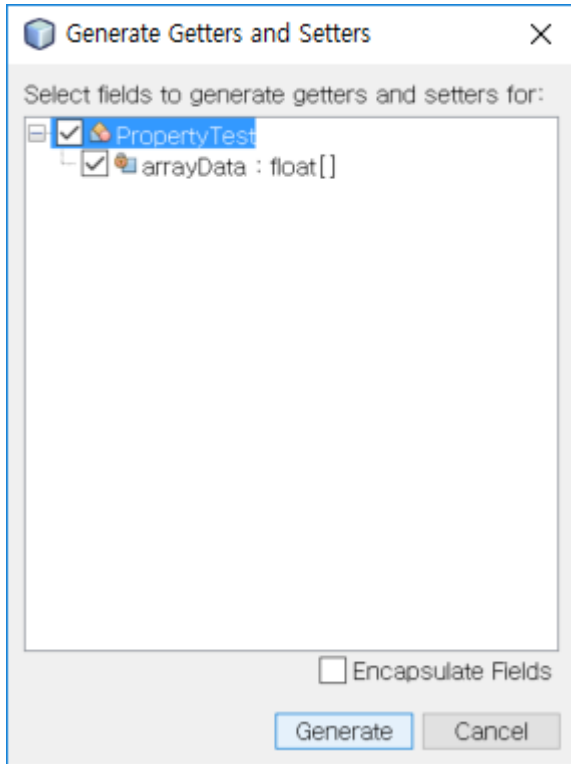
색인 프로퍼티

색인 프로퍼티	
애트리뷰트	private <PropertyElement>[] <propertyName>;
접근 메소드	public <PropertyElement>[] get<PropertyName>(); public void set<PropertyName>(<PropertyElement>[] a); public <PropertyElement> get<PropertyName>(int a); public void set<PropertyName>(int a, <PropertyElement> b);

사용 예	private float[] arrayData = new float[10];
	
	public float[] getArrayData();
	public void setArrayData(float[] arrayData);
	public float getArrayData(int index);
	public void setArrayData(int index, float data);

색인 프로퍼티 생성 방법

- MRB > Insert Code... > Getter and Setter... 선택




```
public class PropertyTest {
    private float[] arrayData = new float[10];

    public float[] getArrayData() {
        return arrayData;
    }

    public void setArrayData(float[] arrayData) {
        this.arrayData = arrayData;
    }
```

캡슐화(encapsulation)*

- 데이터 노출!!!
  - 지금까지 우리가 만든 프로그램에는 데이터가 완전히 노출되어있다는 심각한 문제가 있었습니다. 즉 아무나 인스턴스 변수를 마음대로 보고 건드릴 수 있었습니다.
  - theCat.height = 27;
  - theCat.height = 0;

이렇게 theCat이라는 객체의 height 변수의 값을 마음대로 0으로 바꿀 수 있으면 안 됩니다.

캡슐화

- 이런 문제를 어떻게 해결할 수 있을까요?
  - 세터 메서드를 쓰면 됩니다.

~~theCat.height = 0;~~

```
public void setHeight(int ht) {
  if (height > 9) {
    height = ht;
  }
}
```

세터 메서드를 사용하면 이렇게 인스턴스 변수의 값이 합당한지 검사할 수도 있습니다.

캡슐화

- 그렇다면 데이터를 직접 건드릴 수 없도록 하려면 어떻게 해야 할까요?

- 액세스 변경자(access modifier; public, private, protected, default) 사용
  - 인스턴스 변수: private으로 선언
  - 게터 및 세터 메서드: public으로 선언

- 참고: UML에서 가시성(액세스 변경자) 표현 방법
  - public (공개 가시성): +
  - private (비공개 가시성): -
  - protected (보호 가시성): #
  - default (패키지 가시성): ~ (package-private라고도 함)

캡슐화 실습

GoodDog
-size: int
+getSize(): int +setSize(in size: int) : void ~bark(): void

```
1  package cse.oop2.ch04.encapsulation;
2
3  /**
4   * p.116 실습:  캡슐화 개념
5   *
6   * @author Prof.Jong Min Lee
7   *
8   */
9  public class GoodDog {
10
11     private int size;
12
13     public int getSize() {
14         return size;
15     }
16
17     public void setSize(int size) {
18         this.size = size;
19     }
```

```
21     void bark() {
22         if (size > 60) {
23             System.out.println("Wooof! Wooof!");
24         } else if (size > 14) {
25             System.out.println("Ruff! Ruff!");
26         } else {
27             System.out.println("Yip! Yip!");
28         }
29     }
30 }
```

116 페이지에 있는 예제를 직접 실행시켜봅시다.

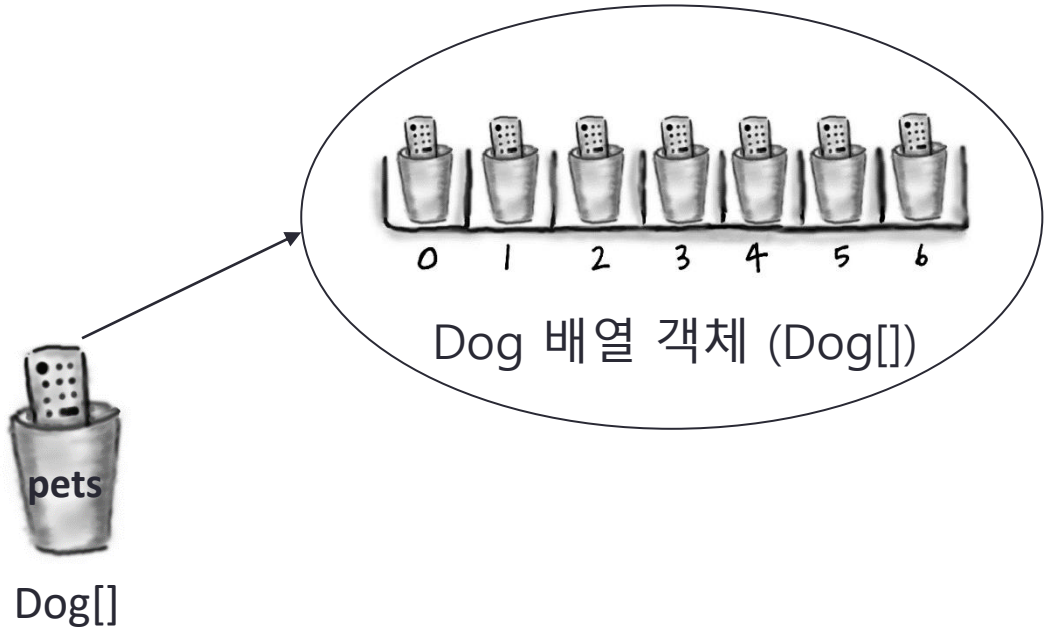
```
1  package cse.oop2.ch04.encapsulation;
2
3  /** p ...6 lines */
9  public class GoogDogTestDrive {
10
11     public static void main(String[] args) {
12         GoodDog one = new GoodDog();
13         one.setSize(70);
14         GoodDog two = new GoodDog();
15         two.setSize(8);
16         GoodDog three = new GoodDog();
17         three.setSize(35);
18         System.out.println("Dog one: " + one.getSize());
19         System.out.format("Dog two: %d%n", two.getSize());
20         one.bark();
21         two.bark();
22     }
23
24 }
```

```
--- exec-maven-plugin:
Dog one: 70
Dog two: 8
Wooof! Wooof!
Yip! Yip!
```

배열 안에 있는 객체

1. Dog 배열 변수 선언

```
// Dog 배열 선언
Dog[] pets;
// 7개의 Dog 객체 참조가 들어갈 수 있는 공간을 만들어 pets 변수에 연결
pets = new Dog[7];
```



p.125

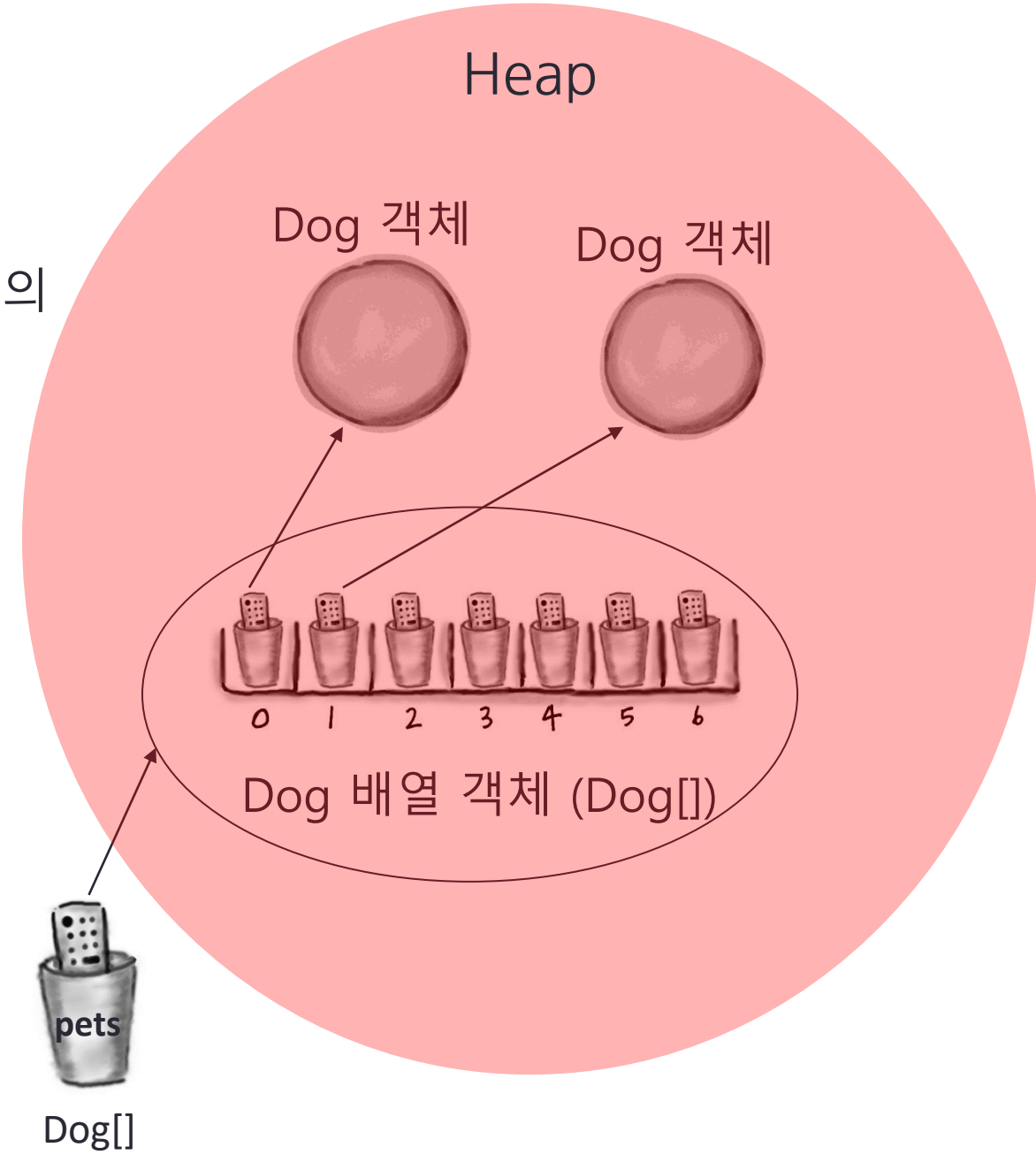
배열 안에 있는 객체

2. 두 개의 새로운 Dog 객체를 만들고 두 개의 배열 원소 대입

```
pets[0] = new Dog();
pets[1] = new Dog();
```

3. Dog 객체 두 개에 대해 메서드 호출

```
pets[0].setSize(30);
int x = pets[0].getSize();
pets[1].setSize(8);
```



인스턴스 변수 초기화

- 인스턴스 변수 선언
  - int size;
  - String name;

- 인스턴스 변수 선언 및 초기화
  - int size = 420;
  - String name = “Donny”;

인스턴스 변수의 기본값

```
6   package cse.oop2.ch04.encapsulation;
7
8   /**
9    * p.118 실습: 인스턴스 변수 초기화 X
10   * @author jongmin
11   */
12  public class PoorDog {
13
14      private int size;      // 기본 값: 0
15      private String name; // 기본 값: null
16
17      public int getSize() {
18          return size;
19      }
20
21      public String getName() {
22          return name;
23      }
```

```
25      public static void main(String[] args) {
26          PoorDog one = new PoorDog();
27          System.out.format("Dog size is %d₩n", one.getSize());
28          System.out.format("Dog name is %s₩n", one.getName());
29      }
30  }
```

실행 결과는?

```
--- exec-maven-plugin:
Dog size is 0
Dog name is null
```

p.126

인스턴스 변수의 기본값*

- 인스턴스 변수에는 항상 기본값이 들어갑니다.

- 각 유형별 기본값
  - 정수(byte, int, short, long) 0
  - 부동소수점 수(float, double) 0.0
  - 부울(Boolean) false
  - 객체 참조(object reference) null

인스턴스 변수와 지역 변수*

- 선언되는 위치
  - 인스턴스 변수 – 클래스 내에서
  - 지역 변수 – 메서드 내에서
  - 매개 변수 - 메서드 정의 시
- 초기화
  - 인스턴스 변수 – 기본 초기값이 **있음**
  - 지역 변수 – 기본 초기값이 **없음**
  - 매개 변수 - 메서드 호출 시 **값 복사**
    (메서드 호출 시 인자 없으면 컴파일 오류 발생)

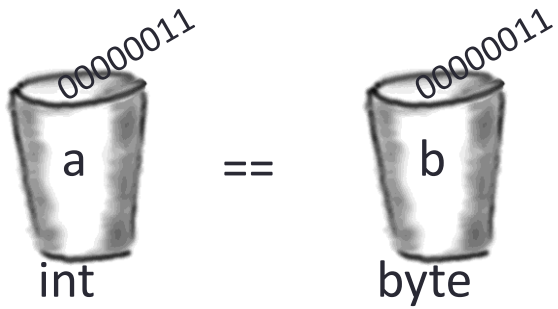
p.127

```
class Horse {
  private double height = 15.2;
  private String breed;  // default: null
  // 나머지 코드
}

class Foo {
  public void go() {
    int x;  // x 값은?
    int z = x + 3;  // z값은?
  }
}

class AddThing {
  ……
  public int add() {
    int total = a + b; // a,b: 인스턴스 변
    수
    return total;
  }
}
```

변수 비교

- == 연산자
  - 임의의 유형의 두 변수를 비교하기 위한 연산자
  - 비트 패턴을 비교합니다.

  - 원시값
    - 두 값을 직접 비교합니다.
  - 레퍼런스
    - 두 변수가 같은 객체를 참조하는지를 비교합니다. (참고) equals()




```
int a = 3;
byte b = 3;
if (a == b) { // 참 }
```

```
Foo a = new Foo();
Foo b = new Foo();
Foo c = a;
if (a == b) { // 거짓 }
if (a == c) { // 참 }
if (b == c) { // 거짓 }
```

p.128

연필을 깎으며*

```
int calcArea(int height, int width) {
  return height * width;
}
```

메서드를 제대로 호출한 부분은?

code review, informal review, or walk-through

```
int a = calcArea(7, 12);                O
short c = 7;
calcArea(c, 15);                        O
int d = calcArea(57);                   X(1)
calcArea(2,3);                          O
long t = 42;
int f = calcArea(t, 17);                X(2)
int g = calcArea();                     X(3)
calcArea();                             X(4)
byte h = calcArea(4, 20);               X(5)
int j = calcArea(2, 3, 5);              X(6)
```

p.129

요약

- 클래스의 인스턴스 변수와 메서드
- 매개 변수와 인자
- Getter & Setter 메서드
- 캡슐화
- 접근 제한자 (UML의 가시성): public, private, protected, package-default
- 배열: 원시 자료형 배열 vs 객체 (참조) 배열
- 인스턴스 변수, 지역 변수, 매개 변수
- 인스턴스 변수와 지역 변수의 초기화
- 변수 비교 (==): 원시 자료형 변수 vs 객체 참조 변수