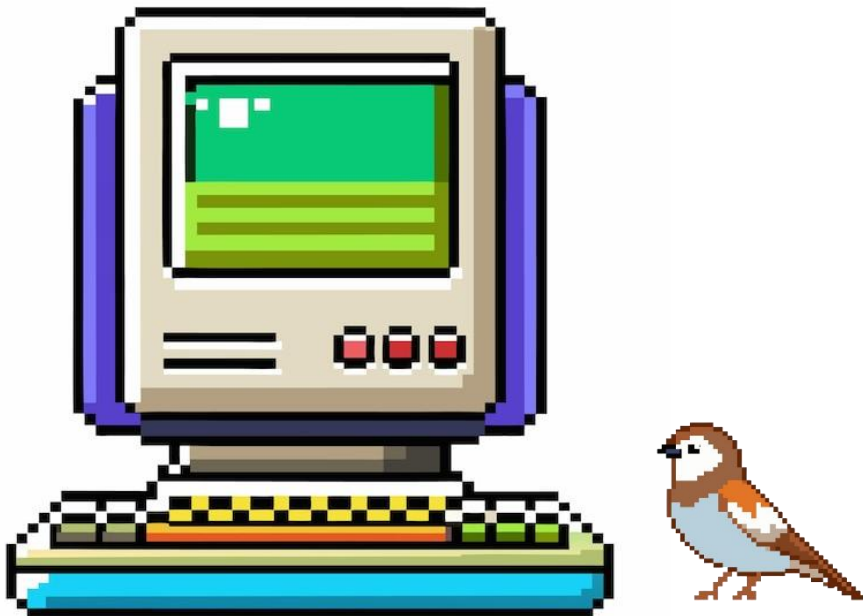


06. 클래스 심층 이해



강동기

접근 제어

- 클래스 안에 변수나 메소드들을 누구나 사용할 수 있게 하면?
 - (예) 내 주민등록번호와 스마트폰번호 정보가 들어가 있는 객체라면?



접근 제어

- 접근 제어 (access control)
 - 다른 클래스가 특정한 필드나 메소드에 접근하는 것을 관리



접근 제어

	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	World
<code>public</code>	+	+	+	+	+
<code>protected</code>	+	+	+	+	
<i>no modifier</i>	+	+	+		
<code>private</code>	+				

예제: 접근 제어 구현

- AccessControl.java 소스코드



예제: 접근 제어 구현 [2]

AccessControl.java

```
package main;

class MyInfo{
    private int stdNumber;
    int age;
    public String name;
}

public class AccessControl {
    public static void main(String args[]) {
        MyInfo myInfo = new MyInfo();
        // myInfo.stdNumber;
        myInfo.age = 21;
        myInfo.name = "Cha Eun Woo";
    }
}
```

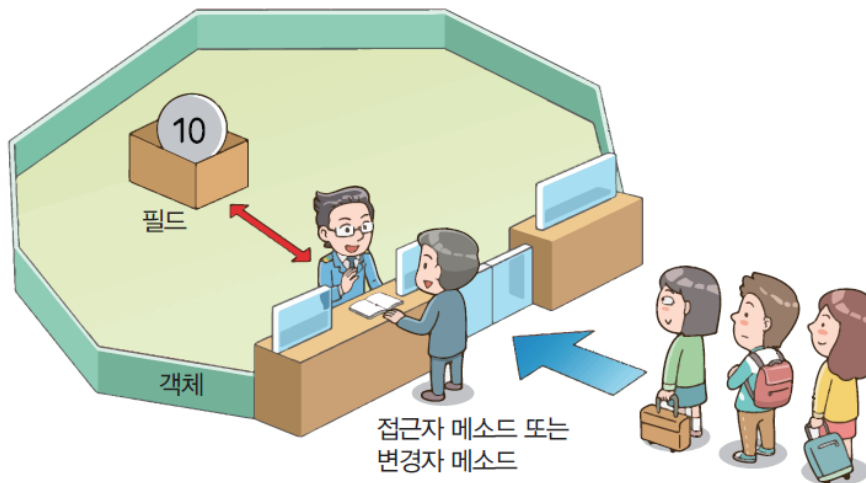
setters 와 getters

설정자 (setters)

- 필드의 값을 설정하는 메소드
- setXXX() 형식

접근자 (getters)

- 필드의 값을 반환하는 메소드
- getXXX() 형식

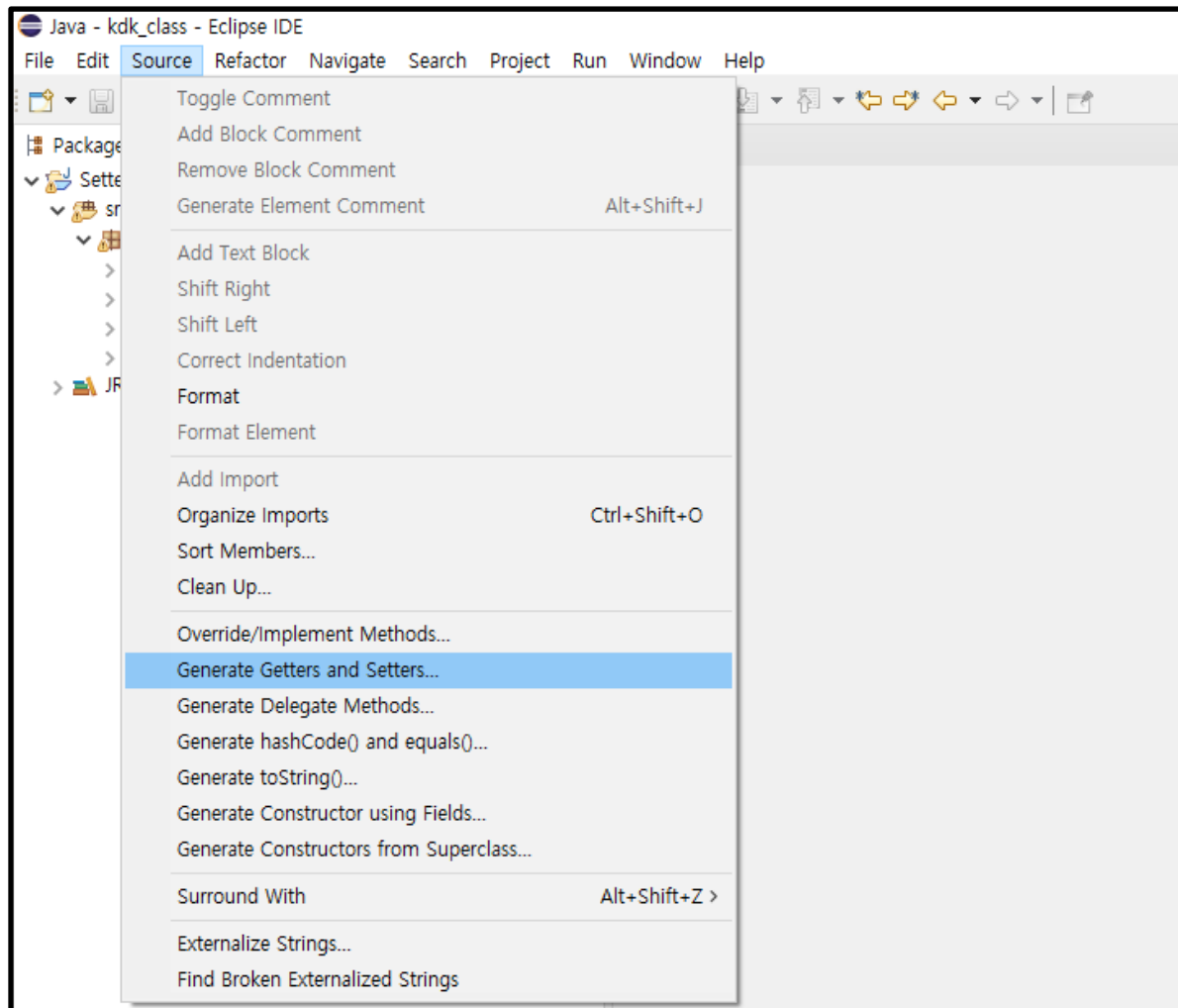


접근자와 설정자 메소드
만을 통하여 필드에
접근하여야 합니다.



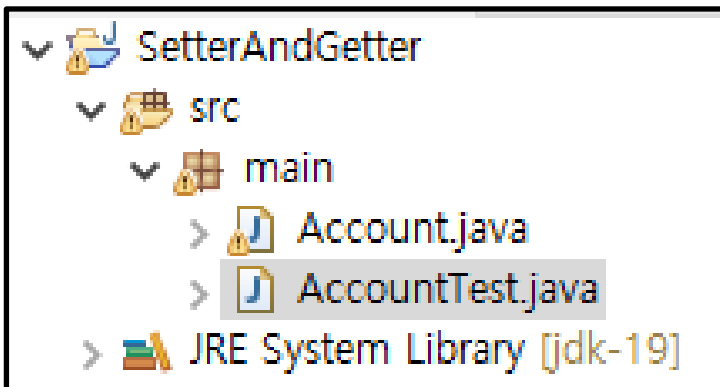
setters 와 getters

[Source] – [Generate Getters and Setters] 로 자동 생성 가능



예제: setters 와 getters 활용 (1)

프로젝트 구조



예제: setters 와 getters 활용 (2)

Account.java

```
package main;

public class Account {
    private int regNumber;
    private String name;
    private int balance;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    public int getBalance() {
        return balance;
    }
    public void setBalance(int balance) {
        this.balance = balance;
    }
}
```

예제: setters 와 getters 활용 (3)

AccountTest.java

```
package main;

public class AccountTest {
    public static void main(String[] args) {
        Account obj = new Account();
        obj.setName("Tom");
        obj.setBalance(100000);
        System.out.println("The name is " + obj.getName() +
            ", the balance is " + obj.getBalance());
    }
}
```

예제: setters 와 getters 활용 (4)

실행 결과

```
The name is Tom, the balance is 100000
```

setters 와 getters 사용 이유

클래스 멤버를 아무나 수정하지 못하도록 필터링 할 수 있음

- getter 만을 제공하면 읽기만 가능한 필드 구성 가능

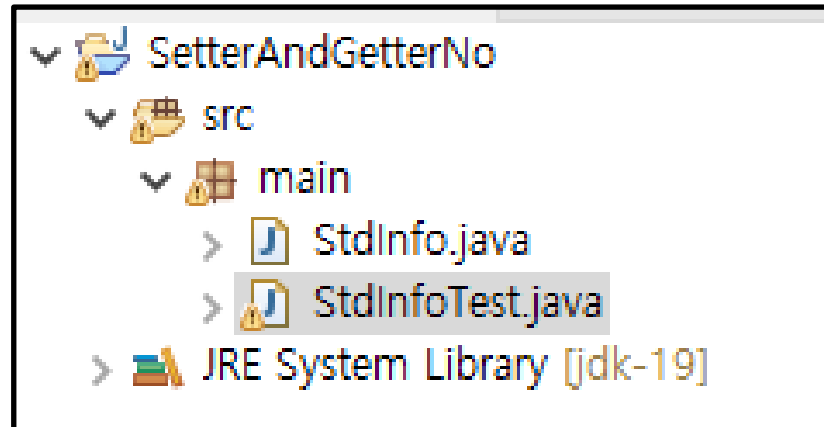
잘못된 파라미터가 넘어오는 경우, 이를 일괄 차단

- 클래스 제작자가 직접 설정 규칙을 정할 수 있음

- 추후 소스 코드를 변경 및 확장할 때 편리함

예제: setters 와 getters 사용 이유 (1)

프로젝트 구조



예제: setters 와 getters 사용 이유 (2)

StdInfo.java

```
package main;

public class StdInfo {
    public String name;
    public int age;
    public String major;
}
```


예제: setters 와 getters 사용 이유 (3)

StdInfoTest.java (1/2)

```
package main;

import java.util.Scanner;

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

        Scanner input = new Scanner(System.in);

        StdInfo std1 = new StdInfo();
        System.out.print("Student 1 name: ");
        std1.name = input.next();
        System.out.print("Student 1 age: ");
        std1.age = input.nextInt();
        System.out.print("Student 1 major: ");
        std1.major = input.next();
        System.out.println();
    }
}
```

예제: setters 와 getters 사용 이유 [4]

StdInfoTest.java (2/2)

```
StdInfo std2 = new StdInfo();
System.out.print("Student 2 name: ");
std2.name = input.next();
System.out.print("Student 2 age: ");
std2.age = input.nextInt();
System.out.print("Student 2 major: ");
std2.major = input.next();
System.out.println();

System.out.println("The name of student 1 is " + std1.name +
    ", and the name of student 2 is " + std2.name + ".");
System.out.println(std1.name + " is " + std1.age +
    " years old, while " + std2.name + " is " + std2.age + " years old.");
System.out.println(std1.name + " loves the subject " + std1.major +
    ", but " + std2.name + " does not.");
}
```

예제: setters 와 getters 사용 이유 (5)

실행결과

```
Student 1 name: KDK  
Student 1 age: 60  
Student 1 major: CS
```

```
Student 2 name: Anonymous  
Student 2 age: 50  
Student 2 major: EE
```

```
The name of student 1 is KDK, and the name of student 2 is Anonymous.  
KDK is 60 years old, while Anonymous is 50 years old.  
KDK loves the subject CS, but Anonymous does not.
```

예제: setters 와 getters 사용 이유 (6)

나이에 음수를 입력하면???

```
Student 1 name: KDK
```

```
Student 1 age: -100
```

```
Student 1 major: CS
```

```
Student 2 name: Anonymous
```

```
Student 2 age: -99
```

```
Student 2 major: EE
```

The name of student 1 is KDK, and the name of student 2 is Anonymous.
KDK is -100 years old, while Anonymous is -99 years old.
KDK loves the subject CS, but Anonymous does not.

예제: setters 와 getters 사용 이유 (7)

나이에 음수를 입력하는 것을 막고 싶다면???

```
StdInfo std1 = new StdInfo();
System.out.print("Student 1 name: ");
std1.name = input.next();
while(true) {
    System.out.print("Student 1 age: ");
    std1.age = input.nextInt();
    if(std1.age >= 1)
        break;
}
System.out.print("Student 1 major: ");
std1.major = input.next();
System.out.println();

StdInfo std2 = new StdInfo();
System.out.print("Student 2 name: ");
std2.name = input.next();
while(true) {
    System.out.print("Student 2 age: ");
    std2.age = input.nextInt();
    if(std2.age >= 1)
        break;
}
System.out.print("Student 2 major: ");
std2.major = input.next();
System.out.println();
```

예제: setters 와 getters 사용 이유 (8)

StdInfo2.java (1/2)

```
package main;

import java.util.Scanner;

public class StdInfo2 {
    private String name;
    private int age;
    private String major;

    public void setStdInfo() {
        Scanner input = new Scanner(System.in);
        System.out.print("Student name: ");
        this.setName(input.next());
        while(true) {
            System.out.print("Student age: ");
            boolean flag = this.setAge(input.nextInt());
            if(flag == true) break;
        }
        System.out.print("Student major: ");
        this.setMajor(input.next());
        System.out.println();
    }
}
```

예제: setters 와 getters 사용 이유 (9)

StdInfo2.java (2/2)

```
public String getName() {  
    return name;  
}  
private void setName(String name) {  
    this.name = name;  
}  
public int getAge() {  
    return age;  
}  
private boolean setAge(int age) {  
    if(age < 0) {  
        return false;  
    }  
    this.age = age;  
    return true;  
}  
public String getMajor() {  
    return major;  
}  
private void setMajor(String major) {  
    this.major = major;  
}  
}
```

예제: setters 와 getters 사용 이유 (10)

StdInfoTest2.java

```
package main;

public class StdInfoTest2 {
    public static void main(String[] args) {

        StdInfo2 std1 = new StdInfo2();
        std1.setStdInfo();

        StdInfo2 std2 = new StdInfo2();
        std2.setStdInfo();

        System.out.println("The name of student 1 is " + std1.getName() +
            ", and the name of student 2 is " + std2.getName() + ".");
        System.out.println(std1.getName() + " is " + std1.getAge() +
            " years old, while " + std2.getName() + " is " + std2.getAge() + " years old.");
        System.out.println(std1.getName() + " loves the subject " +
            std1.getMajor() + ", but " + std2.getName() + " does not.");
    }
}
```


예제: setters 와 getters 사용 이유 [11]

실행결과

```
Student name: KDK  
Student age: -60  
Student age: 60  
Student major: CS
```

```
Student name: Anonymous  
Student age: -100  
Student age: 100  
Student major: EE
```

The name of student 1 is KDK, and the name of student 2 is Anonymous.
KDK is 60 years old, while Anonymous is 100 years old.
KDK loves the subject CS, but Anonymous does not.

메서드 오버로딩

메서드 시그니처 중 자료형을 다르게 하여 여러 개의 동일한 이름을 가진 메서드를 같은 공간에 정의하는 것

메서드 시그니처(Method Signature)

- 메서드 이름과 매개변수 자료형을 의미함
- JVM은 시그니처가 다르다면 다른 메서드로 인식함

메서드 오버로딩

입력매개변수에 따라 4개의 메서드로 오버로딩된 메서드의 예

```
public static void main(String[] ar) {
```

```
    print();
```

```
    print(3);
```

```
    print(5.8);
```

```
    print(2, 5);
```

```
}
```

```
public static void print() {  
    System.out.println("데이터가 없습니다");  
}
```

```
public static void print(int a) {  
    System.out.println(a);  
}
```

```
public static void print(double a) {  
    System.out.println(a);  
}
```

```
public static void print(int a, int b) {  
    System.out.println(a + ", " + b);  
}
```

메서드 오버로딩

메서드 시그니처가 다른 경우 (오버로딩 가능)

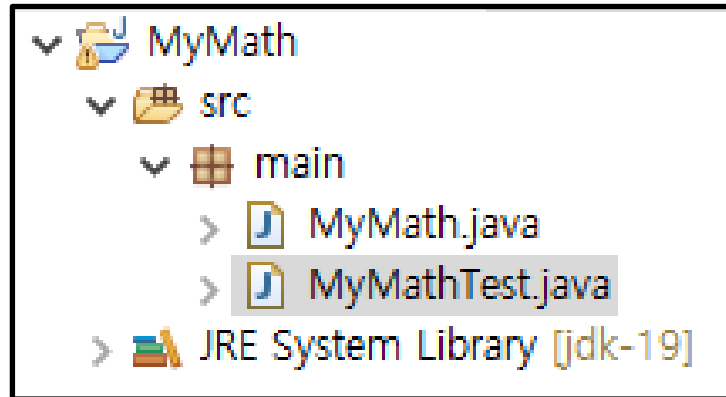
- public int getAge() { ... }
- public int getAge(int birthYear){ ... }

메서드 시그니처가 동일한 경우 (오버로딩 불가)

- public int getAge() { ... }
- public double getAge() { ... }

예제: 메서드 오버로딩 활용 (1)

프로젝트 구조



예제: 메서드 오버로딩 활용 (2)

MyMath.java

```
package main;

public class MyMath {
    int square(int i) {
        return i * i;
    }
    double square(double i) {
        return i * i;
    }
}
```

예제: 메서드 오버로딩 활용 (3)

MyMathTest.java

```
package main;

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));
    }
}
```

예제: 메서드 오버로딩 활용 (4)

실행결과

```
100  
9.8596
```

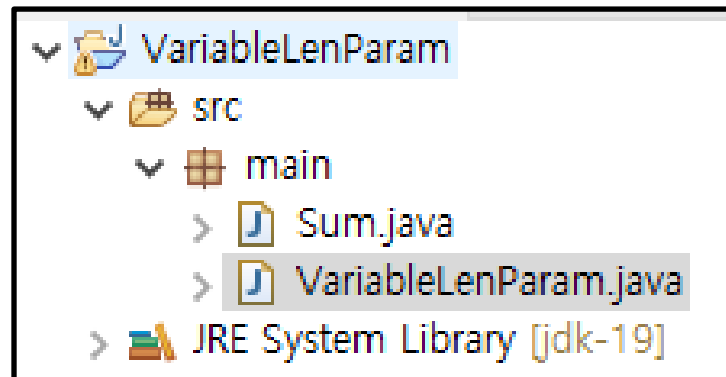

가변길이 매개변수 메서드

다양한 개수의 매개변수 처리가 가능한 메서드

- 자료형 바로 뒤에 점 세개(...)를 붙여서 표현
- 예: `public int sum(int... values) { }`

예제: 가변길이 매개변수 메서드 활용 (1)

프로젝트 구조



예제: 가변길이 매개변수 메서드 활용 (2)

Sum.java

```
package main;

public class Sum {
    public int getIntSum(int... values) {
        int total = 0;
        for(int element: values) {
            total += element;
        }
        return total;
    }

    public double getDoubleSum(double... values) {
        double total = 0;
        for(double element: values) {
            total += element;
        }
        return total;
    }
}
```

예제: 가변길이 매개변수 메서드 활용 (3)

VariableLenParam.java

```
package main;

public class VariableLenParam {
    public static void main(String[] args) {
        Sum sum = new Sum();
        int intSum = sum.getIntSum(1,2,3,4,5,6,7,8,9,10);
        double doubleSum = sum.getDoubleSum(3.3, -1.7, 5.0, 5.999);

        System.out.println(intSum);
        System.out.println(doubleSum);
    }
}
```

예제: 가변길이 매개변수 메서드 활용 (4)

실행결과

```
55  
12.599
```

생성자

constructor

객체가 생성될 때에 필드에게 초기값을 제공하고 그 외 필요한 초기화 절차를 실행하는 메소드



생성자

생성자(Constructor)의 특징

- 클래스 이름과 동일함
- 객체를 생성함
- 객체내의 필드 초기화를 수행함
- 리턴 타입이 없음

전체적인 구조



형식

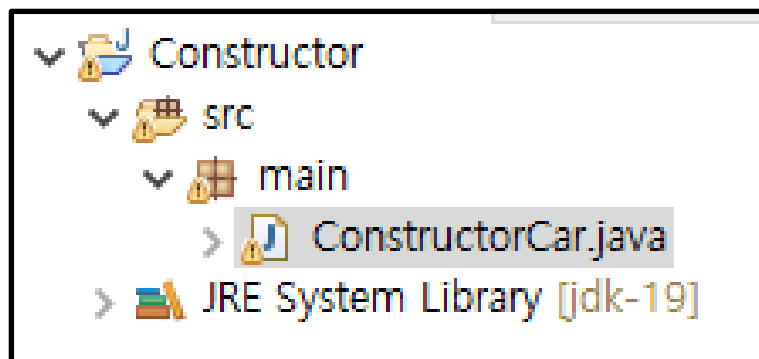
```
public class Car {
    Car() {
        ...
    }
}
```

클래스 이름과 동일한 메소드가 바로 생성자입니다. 여기서 객체의 초기화를 담당합니다.



예제: 생성자 사용 (1)

프로젝트 구조



예제: 생성자 사용 [2]

ConstructorCar.java

```
package main;

public class ConstructorCar {
    public static void main(String[] args) {
        Car car = new Car();
    }
}

class Car {
    String myStat = "I was a car.";

    Car() {
        System.out.println(myStat);
    }
}
```

예제: 생성자 사용 (3)

실행결과

```
I was a car.
```

생성자 오버로딩

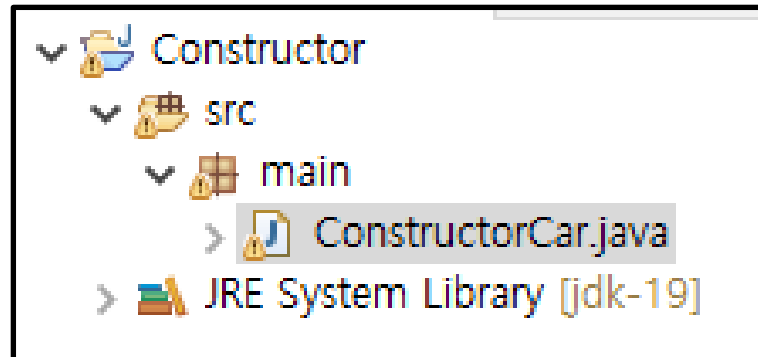
메소드처럼 생성자도 오버로딩될 수 있음
- 생성자도 메소드의 한 종류니까 ...

오버로딩을 통해 다양한 객체 생성 가능



예제: 매개변수를 가지는 생성자 (1)

프로젝트 구조



예제: 매개변수를 가지는 생성자 [2]

수정된 ConstructorCar.java

```
package main;

public class ConstructorCar {
    public static void main(String[] args) {
        Car car = new Car(3);
    }
}

class Car {
    String myStat = "I was a car.";

    Car() {
        System.out.println(myStat);
    }
    Car(int count) {
        System.out.println(myStat + " " + count + " times.");
    }
}
```

예제: 매개변수를 가지는 생성자 (3)

실행결과

```
I was a car. 3 times.
```

this

모든 메서드에는 자신이 포함된 클래스의 객체를 가리키는
this 참조 변수가 존재함

멤버변수의 경우

- this.멤버변수 형태로 사용

메서드의 경우

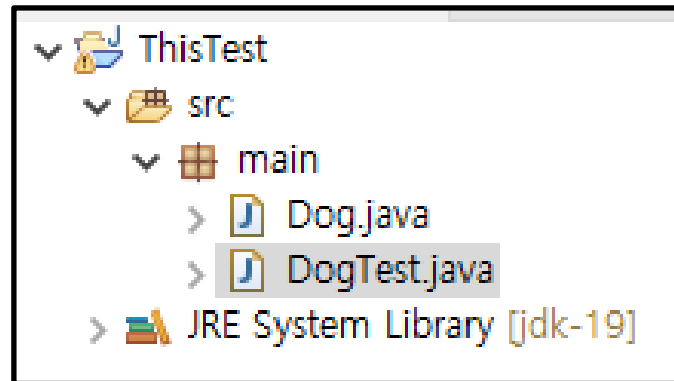
- this.메서드 형태로 사용

this를 생략해도 컴파일러가 자동으로 멤버앞에 this.를 추가함

- 단 지역변수는 멤버변수가 아니므로 this.가 앞에 붙지 않음

예제: this 활용 (1)

프로젝트 구조



예제: this 활용 (2)

Dog.java

```
package main;

public class Dog {
    String name;
    String breed;
    int age;

    Dog(){
        breed = "beagle";
    }

    public void compare(String paramBreed) {
        if(breed.equals(paramBreed) == true) {
            System.out.println("Yes, that's the dog breed I was referring to!");
        }
        else {
            System.out.println("No, that's not.");
        }
    }
}
```

예제: this 활용 (3)

DogTest.java

```
package main;

public class DogTest {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.compare("beagle");
    }
}
```

예제: this 활용 (4)

실행결과

```
Yes, that's the dog breed I was referring to!
```

예제: this 활용 (5)

수정된 Dog.java

– paramBreed 보기 싫어서 breed로 매개변수 이름을 바꾼다면??

```
package main;

public class Dog {
    String name;
    String breed;
    int age;

    Dog(){
        breed = "beagle";
    }

    public void compare(String breed) {
        if(breed.equals(breed) == true) {
            System.out.println("Yes, that's the dog breed I was referring to!");
        }
        else {
            System.out.println("No, that's not.");
        }
    }
}
```

예제: this 활용 [6]

수정된 Dog.java

- breed가 멤버인지 매개변수인지 어떻게 구분할까??
- 컴파일러는 둘 다 매개변수로 간주함

```
package main;

public class Dog {
    String name;
    String breed;
    int age;

    Dog(){
        breed = "beagle";
    }

    public void compare(String breed) {
        if(breed.equals(breed) == true) {
            System.out.println("Yes, that's the dog breed I was referring to!");
        }
        else {
            System.out.println("No, that's not.");
        }
    }
}
```



예제: this 활용 (7)

수정된 Dog.java

- 이 경우엔 this. 키워드를 명시적으로 넣어줘야 함

```
package main;

public class Dog {
    String name;
    String breed;
    int age;

    Dog(){
        breed = "beagle";
    }

    public void compare(String breed) {
        if(this.breed.equals(breed) == true) {
            System.out.println("Yes, that's the dog breed I was referring to!");
        }
        else {
            System.out.println("No, that's not.");
        }
    }
}
```



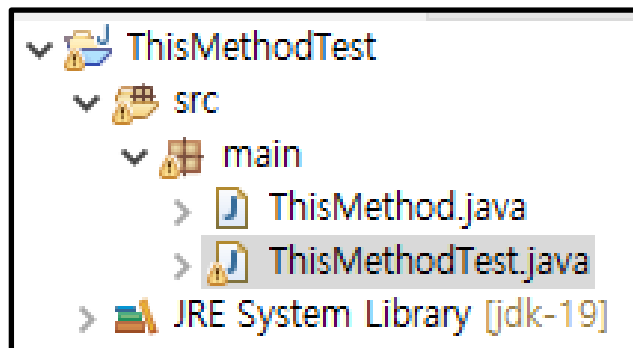
this() 메서드

같은 클래스 내에 다른 생성자를 호출함

- 매개변수를 달리하여 오버로딩된 생성자를 선택할 수 있음

예제: this() 메서드 활용 (1)

프로젝트 구조



예제: this() 메서드 활용 (2)

ThisMethod.java

```
package main;

public class ThisMethod {
    ThisMethod() {
        System.out.println("I like apples.");
    }

    ThisMethod(int i) {
        System.out.println("I like apples.");
        System.out.println("And I also like bananas.");
    }

    ThisMethod(int i, int j) {
        System.out.println("I like apples.");
        System.out.println("And I also like bananas.");
        System.out.println("How about you?");
    }

    ThisMethod(int i, int j, int k) {
        System.out.println("I like apples.");
        System.out.println("And I also like bananas.");
        System.out.println("How about you?");
        System.out.println("I don't like them. They are so expensive.");
    }
}
```

예제: this() 메서드 활용 (3)

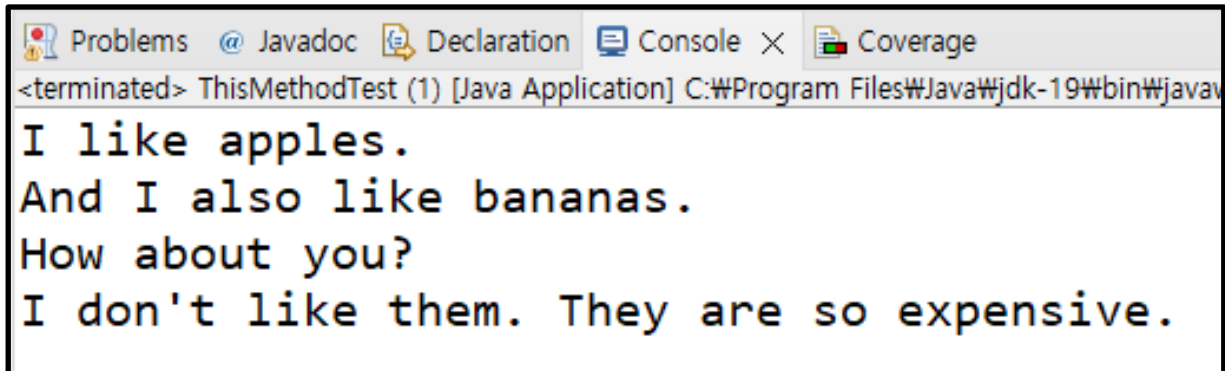
ThisMethodTest.java

```
package main;

public class ThisMethodTest {
    public static void main(String[] args) {
        ThisMethod thisMethod = new ThisMethod(1, 2, 3);
    }
}
```

예제: this() 메서드 활용 (4)

실행결과



The screenshot shows an IDE console window with the following tabs: Problems, Javadoc, Declaration, Console, and Coverage. The console output is as follows:

```
<terminated> ThisMethodTest (1) [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe  
I like apples.  
And I also like bananas.  
How about you?  
I don't like them. They are so expensive.
```

예제: this() 메서드 활용 (5)

ThisMethod2.java

```
package main;

public class ThisMethod2 {
    ThisMethod2() {
        System.out.println("I like apples.");
    }

    ThisMethod2(int i) {
        this();
        System.out.println("And I also like bananas.");
    }

    ThisMethod2(int i, int j) {
        this(i);
        System.out.println("How about you?");
    }

    ThisMethod2(int i, int j, int k) {
        this(i, j);
        System.out.println("I don't like them. They are so expensive.");
    }
}
```

예제: this() 메서드 활용 (6)

수정된 ThisMethodTest.java

```
package main;

public class ThisMethodTest {
    public static void main(String[] args) {
        ThisMethod thisMethod = new ThisMethod(1, 2, 3);

        System.out.println();

        ThisMethod2 thisMethod2 = new ThisMethod2(1, 2, 3);
    }
}
```

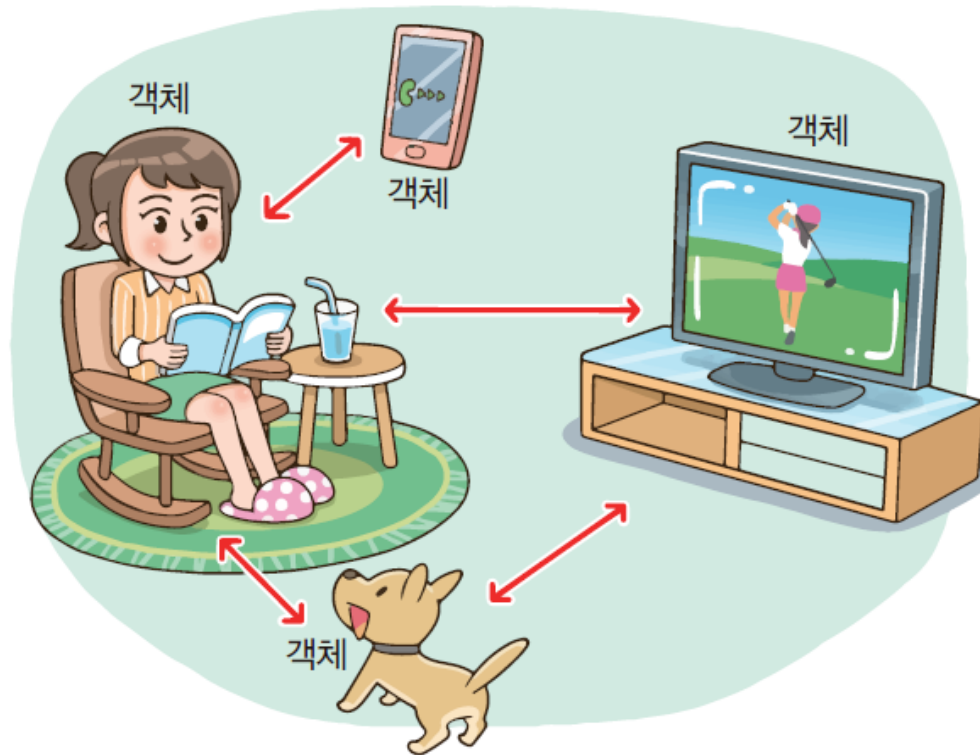
예제: this() 메서드 활용 (7)

실행결과

```
I like apples.  
And I also like bananas.  
How about you?  
I don't like them. They are so expensive.  
  
I like apples.  
And I also like bananas.  
How about you?  
I don't like them. They are so expensive.
```



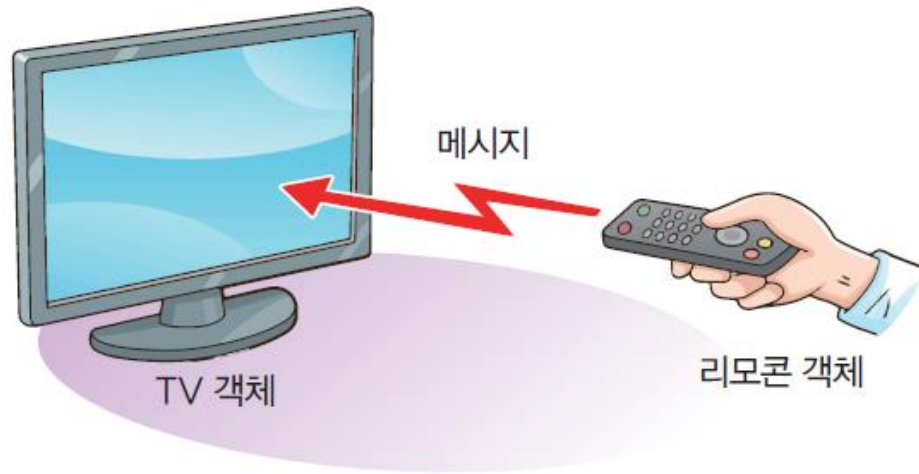
실제 세계는 객체로 이루어진다



실제 세계는 객체들로
이루어져 있죠!



객체와 메시지

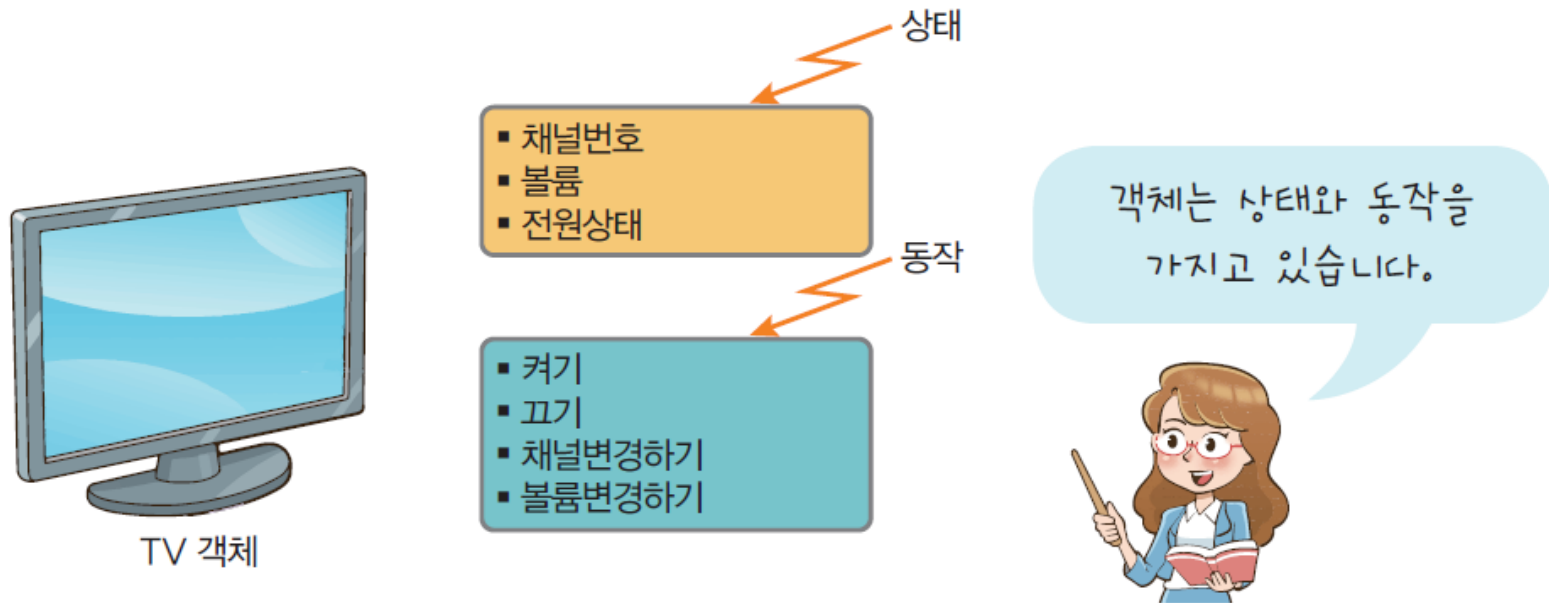


객체들은 메시지를
보내고 받으면서
상호 작용합니다.



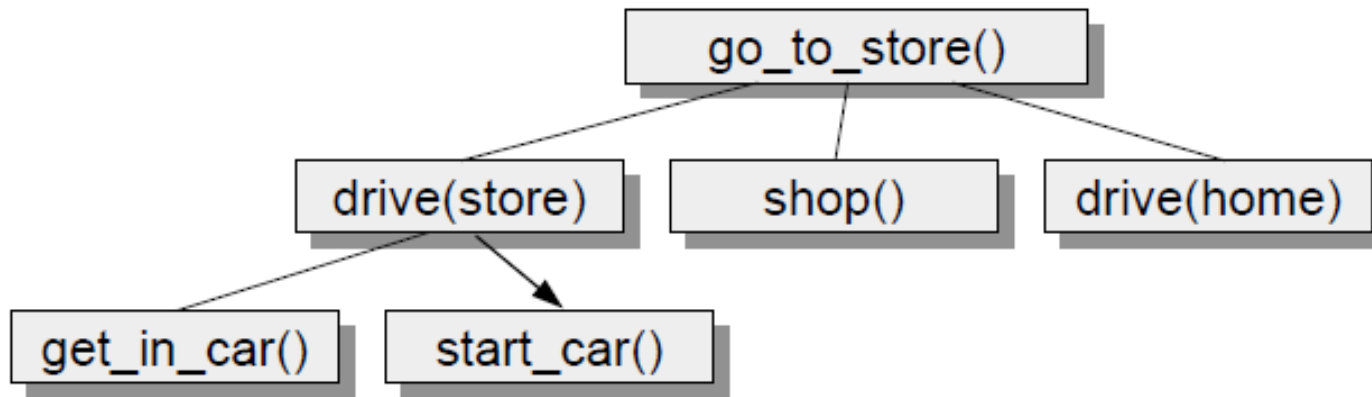
객체의 상태와 동작

- 객체(Object)는 상태(State)와 동작(Behavior)을 가지고 있다.



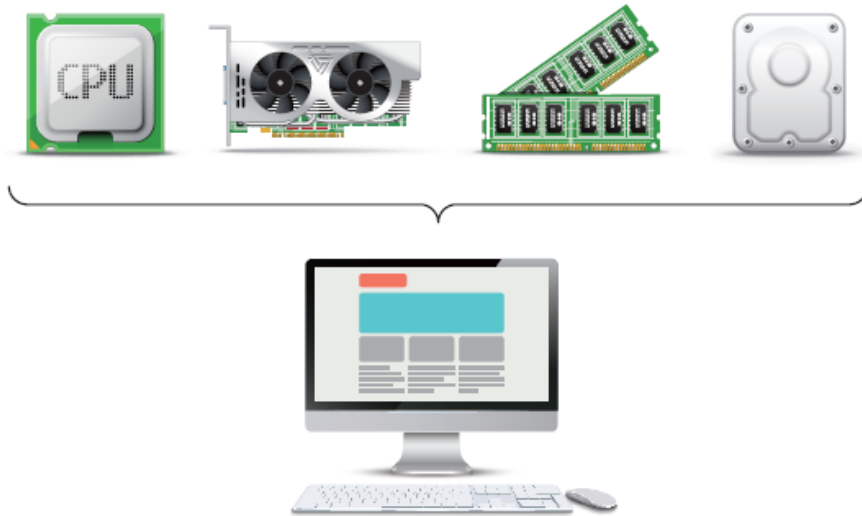
절차지향과 객체지향

- 절차 지향 프로그래밍(Procedural Programming)
 - 문제를 해결하는 순서를 중요하게 생각하는 방법이다.

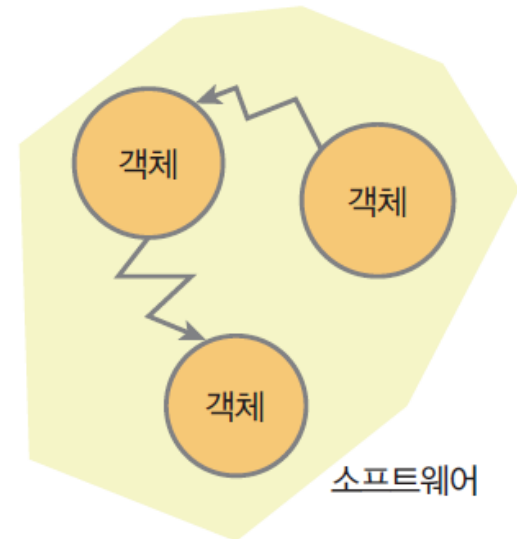


절차지향과 객체지향

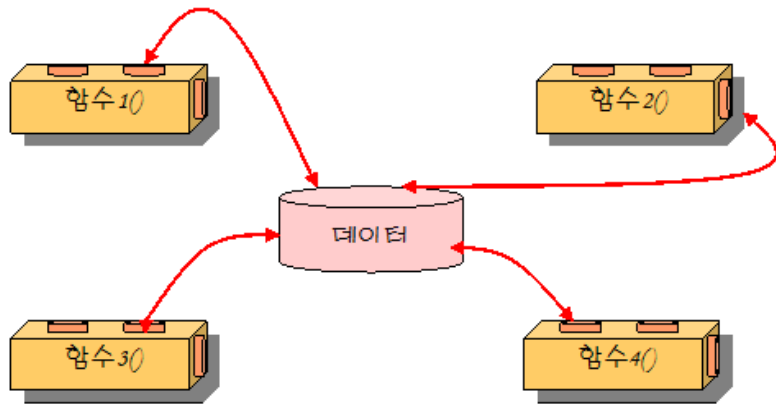
- 객체 지향 프로그래밍(Object-Oriented Programming)
 - 데이터와 절차를 하나의 덩어리(객체)로 묶어서 생각하는 방법이다.



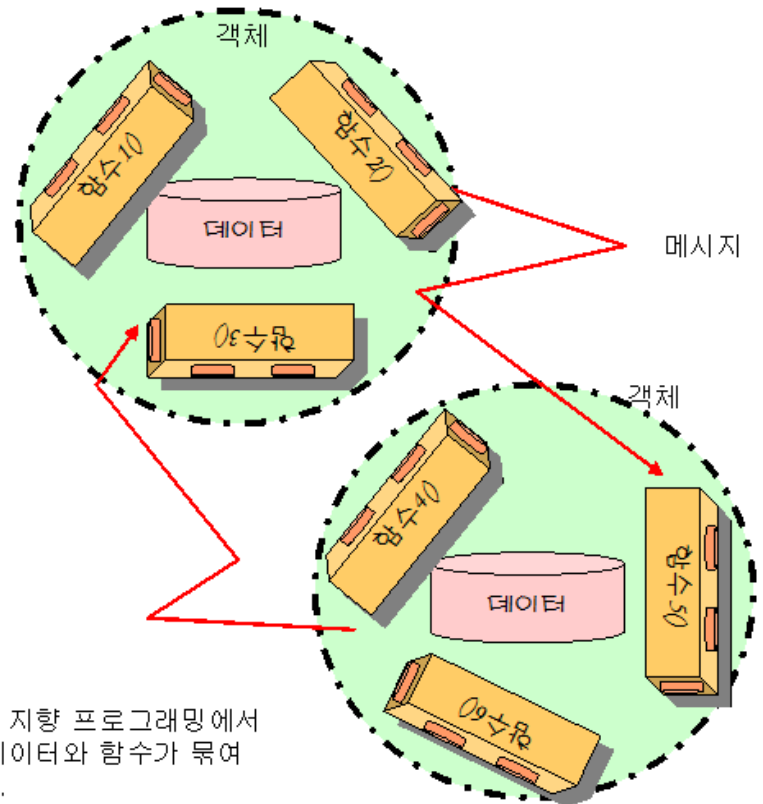
부품을 조립하여 제품을 만들듯이
객체를 조합하여 소프트웨어를 만든다



절차지향과 객체지향



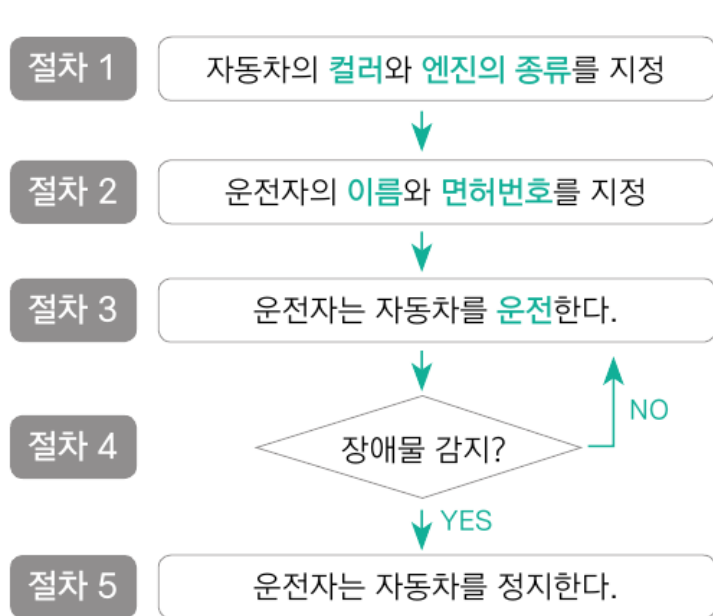
절차 지향 프로그래밍에서
는 데이터와 함수가 묶여
있지 않다.



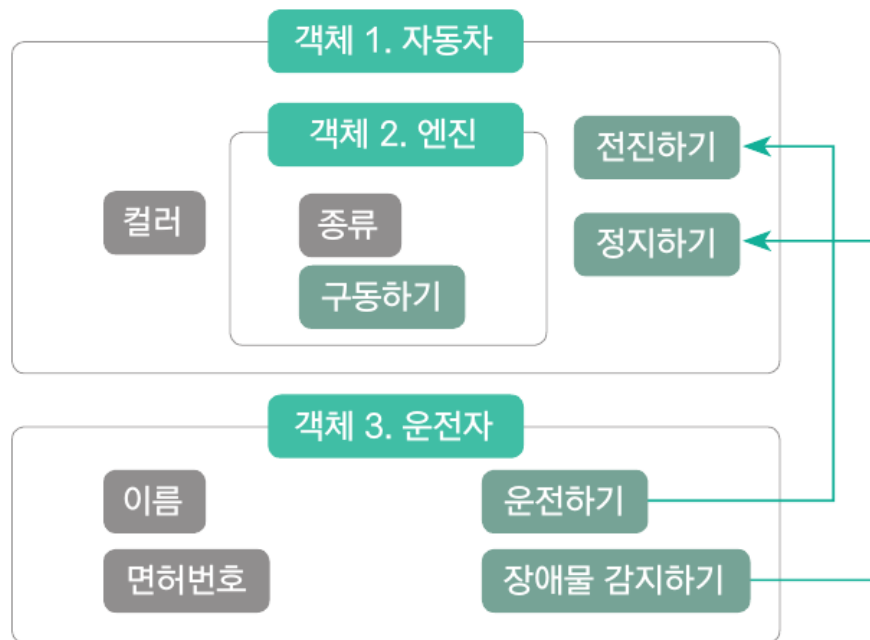
객체 지향 프로그래밍에서
는 데이터와 함수가 묶여
있다.

절차지향과 객체지향

- 자동차를 운전하는 프로그램



절차지향형 프로그램



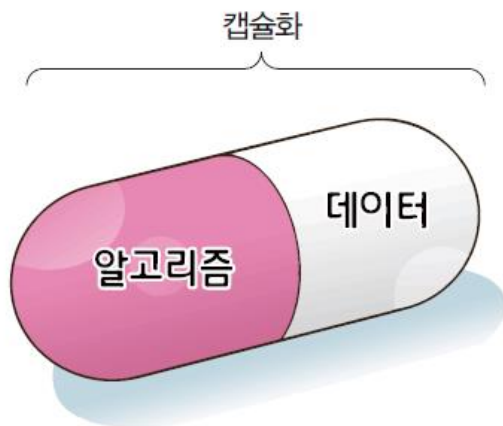
객체지향형 프로그램

객체 지향의 특징

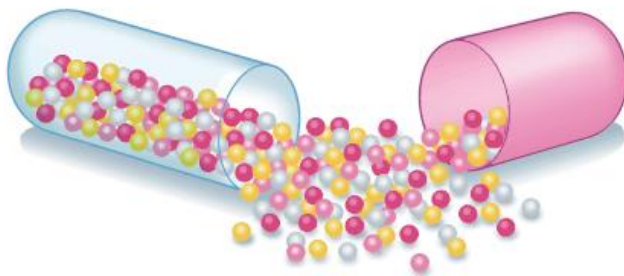
- 캡슐화(Encapsulation)
- 상속(Inheritance)
- 다형성(Polymorphism)
- 추상화(Absraction)

캡슐화(Encapsulation)

- 관련된 데이터와 코드(알고리즘)이 하나의 묶음으로 정리되어 있는 것



캡슐화는 데이터와 알고리즘을 하나로 묶는 것입니다.



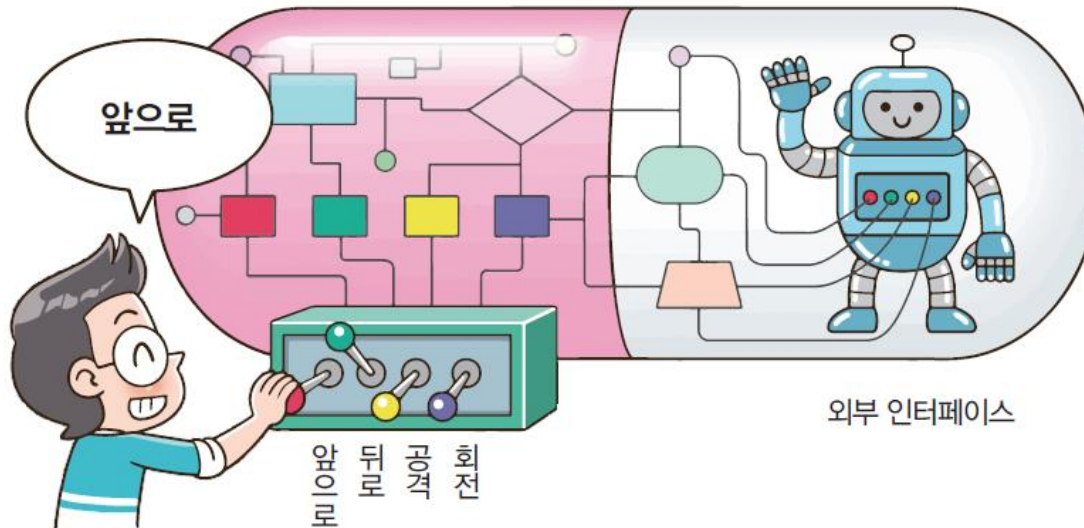
캡슐화 되어 있지 않은 데이터와 코드는 사용하기 어렵겠죠!



캡슐화와 정보 은닉

- 정보 은닉(Information Hiding)

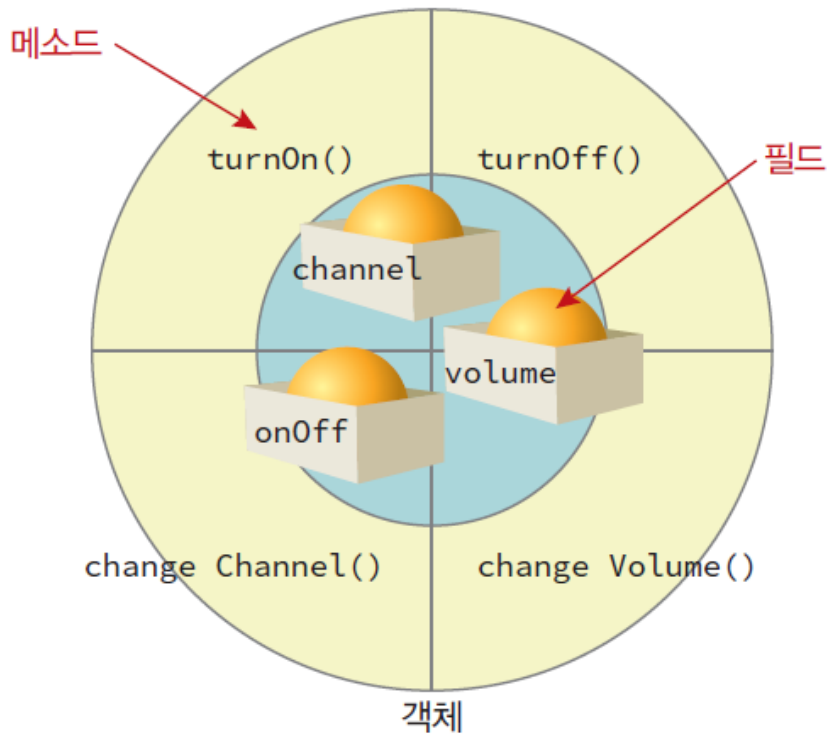
- 객체를 캡슐과 같이 싸서 객체의 내부(상세한 구현 내용)를 숨긴다.
- 별도의 인터페이스를 통해 객체를 접근하고 사용한다.



객체는 공개된
인터페이스를
통하여 사용
하여야 합니다.



캡슐화와 정보 은닉

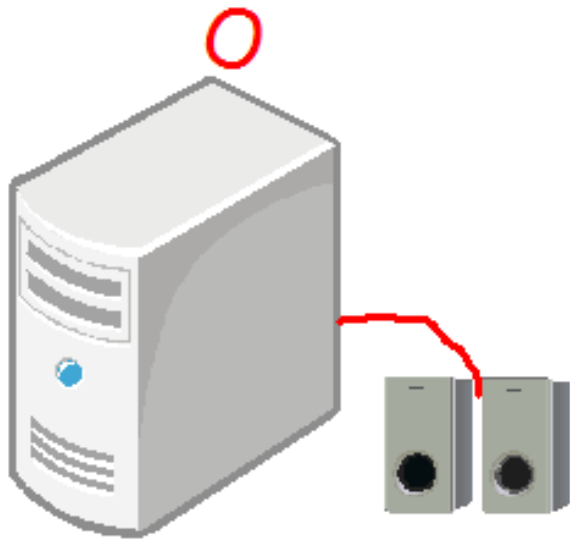


보통은 데이터들은 공개되지 않고 몇 개의 메소드만이 외부로 공개됩니다.

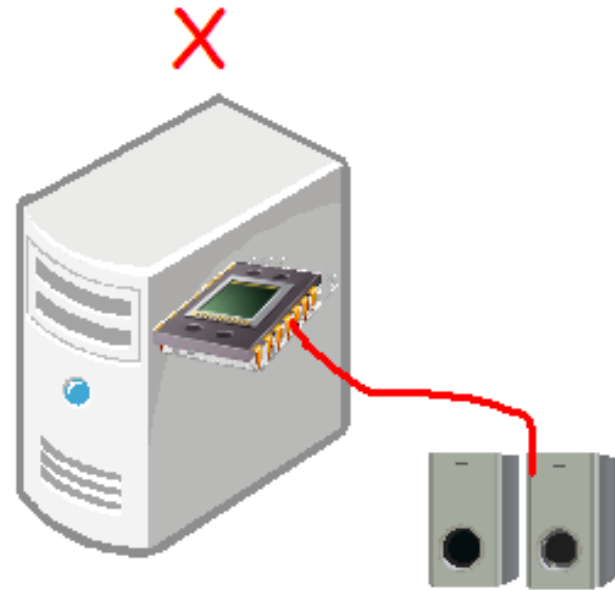


캡슐화로 인한 객체의 분리

- 객체가 서로 분리되므로 독자적인 업그레이드가 가능하다.



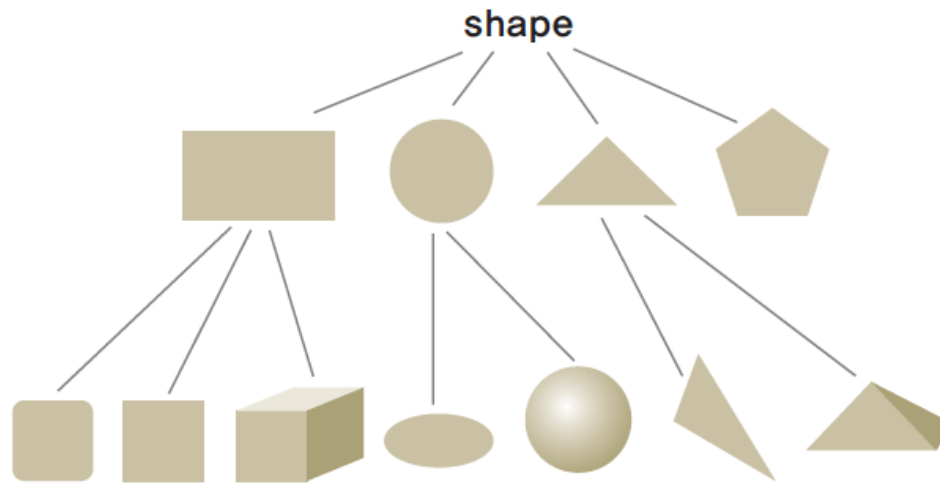
만약 외부의 표준 오디오 단자를 이용하였으면 내부의 사운드 카드를 변경할 수 있다.



만약 내부의 오디오 제어 칩의 단자에 연결하였으면 내부의 사운드 카드를 변경할 수 없다.

상속(Inheritance)

- 이미 작성된 클래스를 이어받아서 새로운 클래스를 생성하는 기법
 - 이미 작성된 클래스를 "부모 클래스" 라고 한다.
 - 새로운 클래스를 "자식 클래스" 라고 한다.
- 기존에 만든 코드를 재활용 할 수 있다.

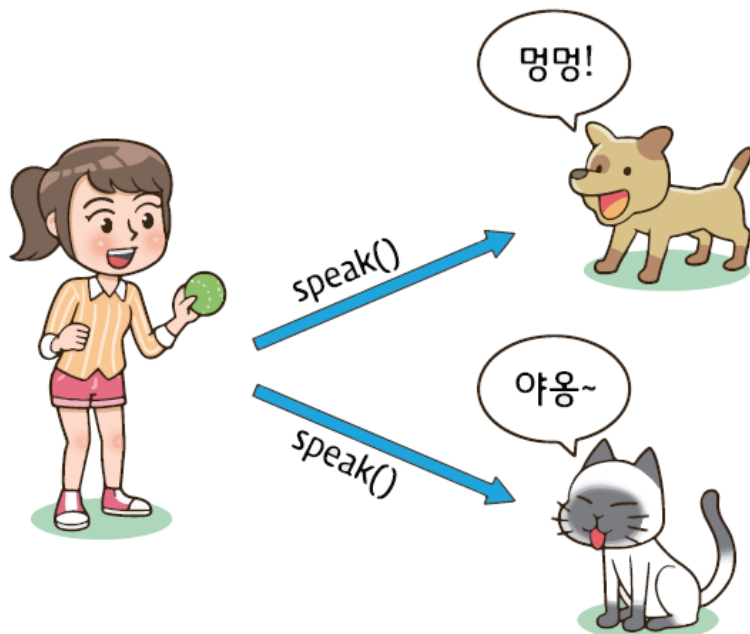


상속은 기존에 만들어진 코드를 이어받아서 보다 쉽게 코드를 작성하는 기법입니다.



다형성(Polymorphism)

- 하나의 객체가 여러 가지 타입을 가질 수 있다.
- 부모 클래스 객체로 자식 클래스 객체를 참조할 수 있다.



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



추상화(Abstraction)

- 클래스들이 공통적으로 가지고 있는 특성을 추출하여 상위 클래스를 정의하는 것
- 자바에서는 클래스외에 인터페이스의 개념을 활용하여 추상화를 구현한다.



실제 객체



추상화된 객체

추상화는 필요한 것만을 남겨놓는 것입니다. 추상화 과정이 없다면 사소한 것도 신경 써야 합니다.



클래스

- 클래스(Class)란?
 - 객체를 만들기 위한 설계도를 의미한다.
- 클래스 관점에서의 객체란?
 - 메모리에 적재하여, 코드로서 사용할 수 있는 클래스의 실체가 된다.
 - 클래스로부터 만들어지는 각각의 객체를 클래스의 인스턴스라고 한다.



클래스는 왜 사용할까?

- 학생 정보를 저장하고 관리하는 프로그램 만들기
 - 학생의 신상과 여러 정보를 저장할 변수가 필요하다.
 - 정보를 출력 및 관리할 수 있는 함수가 필요하다.

클래스는 왜 사용할까?

- NoClassEx.java
 - 클래스를 사용하지 않고 짤다면

```
package ch05;

public class NoClassEx {
    public static void main(String[] args) {
        // 학생 1
        String name1 = "Alice";
        int age1 = 20;
        double grade1 = 3.8;

        // 학생 2
        String name2 = "Bob";
        int age2 = 22;
        double grade2 = 3.5;

        double total = grade1 + grade2;
        double avg = total / 2;

        System.out.println(name1 + " (" + age1 + "세), 학점: " + grade1);
        System.out.println(name2 + " (" + age2 + "세), 학점: " + grade2);
        System.out.println("학점 평균: " + avg);
    }
}
```

클래스는 왜 사용할까?

- NoClassEx.java
 - 클래스를 사용하지 않고 짤다면

```
// 학생 1
String name1 = "Alice";
int age1 = 20;
double grade1 = 3.8;

// 학생 2
String name2 = "Bob";
int age2 = 22;
double grade2 = 3.5;

// 학생 3
String name3 = "Faker";
int age3 = 21;
double grade3 = 4.5;

// 학생 4
String name4 = "Chovy";
int age4 = 19;
double grade4 = 4.3;
```

...

학생 수가 늘어나면 변수도 같이 늘려야 함. 변수이름을 짓고 관리하기 어려움.

클래스는 왜 사용할까?

- NoClassEx.java
 - 배열을 활용한다면

```
package ch05;

public class NoClassEx {
    public static void main(String[] args) {
        String[] names = {"Alice", "Bob", "Faker", "Chovy", ...};
        int[] ages = {20, 21, 21, 19, ...};
        double[] grades = {3.8, 3.5, 4.5, 4.2, ...};

        for(int i=0; i<names.length; i++) {
            double total_ages += ages[i];
            double total_grades += grades[i];
        }

        double avg_ages = total_ages / ages.length;
        double avg_grades = total_grades / grades.length;

        for(int i=0; i<names.length; i++) {
            System.out.println(names[i] + " (" + ages[i] + "세), 학점: " + grades[i]);
        }

        System.out.println("나이 평균: " + avg_ages);
        System.out.println("학점 평균: " + avg_grades);
    }
}
```

같은 자료형의 변수들을 1개의 새로운 자료형으로 묶을 수 있어 사용과 관리가 편함.
그러나 여전히 데이터와 함수가 따로 놀기 때문에 프로그램 덩치가 커질수록 확장 및 디버깅이 어려움.

클래스는 왜 사용할까?

- YesClassEx.java
 - 클래스를 활용한다면

```
package ch05;

class Student {
    String name;
    int age;
    double grade;

    Student(String name, int age, double grade) {
        this.name = name;
        this.age = age;
        this.grade = grade;
    }

    void printInfo() {
        System.out.println(name + " (" + age + "세), 학점: " + grade);
    }
}

public class YesClassEx {
    public static void main(String[] args) {
        Student s1 = new Student("Alice", 20, 3.8);
        Student s2 = new Student("Bob", 22, 3.5);
        Student s3 = new Student("Faker", 22, 3.5);
        Student s4 = new Student("Chovy", 22, 3.5);

        ...

        s1.printInfo();
        s2.printInfo();
        s3.printInfo();
        s4.printInfo();
    }
}
```

데이터와 함수(메서드)가 같은 그룹으로 묶이므로 코드 관리, 확장, 디버깅이 용이해짐.
코드가 길어지고 메모리 사용량은 늘어나지만 최신 컴퓨터 스펙에서는 큰 문제가 안됨.

클래스를 포함하는 자바 소스 파일 구조

- 패키지 (Package)
 - 여러개의 클래스를 묶은 라이브러리 집합
- 임포트 (Import)
 - 다른 패키지의 클래스를 사용하고 싶을 때 작성하는 키워드

A.java

```
package ...;           // ① 패키지
import ...;            // ② 임포트
class 클래스명 {...}    // ③ 외부 클래스
```

클래스의 밖에 올 수 있는 3가지

파일명과 동일해야 함.

```
public class A {
    int a = 3;           // ① 필드
    double abc() {...}   // ② 메서드
    A() {...}            // ③ 생성자
    class 클래스명 {...}  // ④ 이너 클래스
}
```

클래스의 안에 올 수 있는 4가지

클래스 내부 구성 요소

- 필드 (Field)
 - 클래스내에 포함된 변수 (예: human class 의 나이 변수 int age)
- 메서드 (Method)
 - 클래스내에 포함된 함수 (예: human class 의 먹기 함수 eat())
- 생성자 (Constructor)
 - 클래스와 똑같은 이름의 메서드. 객체를 생성하는 역할 수행
- 이너 클래스 (Inner Class)
 - 클래스 내부에 포함된 클래스

객체 생성하기

- new 키워드로 객체 생성하기
 - 모든 클래스는 객체를 new 키워드로 생성함
 - 메서드로 객체를 생성하는 경우에도 해당 메서드 내부에서 new 키워드를 사용함

클래스의 객체 생성

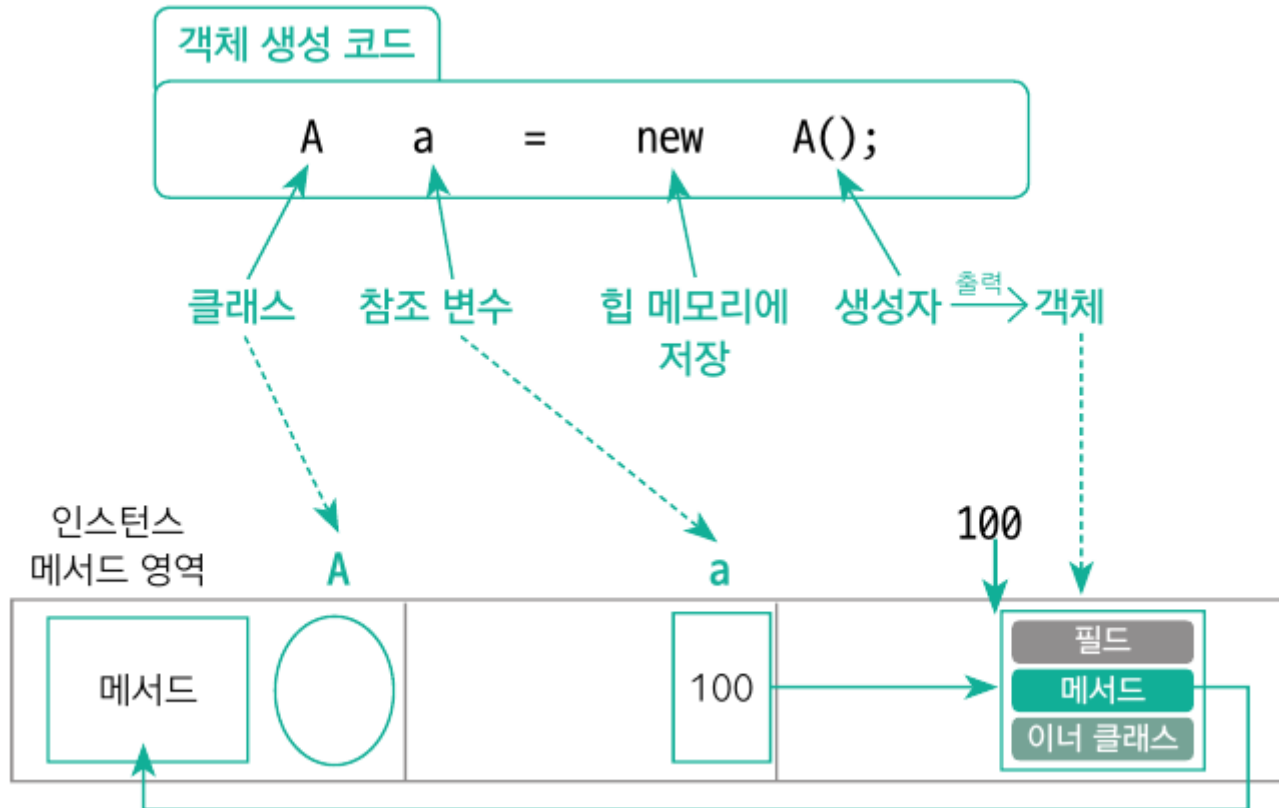
```
클래스명 참조 변수명 = new 생성자();
```

예

```
A a = new A();
```

객체 생성하기

- 객체 생성에 따른 메모리 구조
 - 객체 참조변수는 스택영역에 저장됨
 - 객체(필드, 메서드, 이너클래스 포함)는 힙영역에 저장됨
 - 메서드 구현코드는 인스턴스 메서드영역(클래스영역)에 저장됨



객체 활용하기

- 포인트 연산자 사용하기
 - 소스코드로 힙영역에 직접 접근할수는 없음
 - 포인트 연산자를 통해 참조 변수를 이용하여 객체에 접근함

필드와 메서드의 활용

참조 변수명.필드명

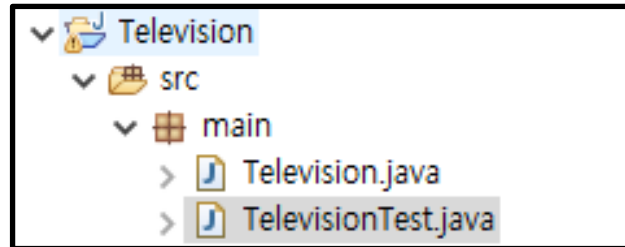
참조 변수명.메서드명()

필드와 메서드의 활용 예

```
A a = new A();  
System.out.println(a.m);    // 필드 활용  
a.print();                  // 메서드 활용
```

예제: 텔레비전 클래스

- 프로젝트 구조



예제: 텔레비전 클래스

- Television.java

```
package main;

public class Television {
    int channel;
    int volume;
    boolean onOff;
}
```

예제: 텔레비전 클래스

- TelevisionTest.java

```
package main;

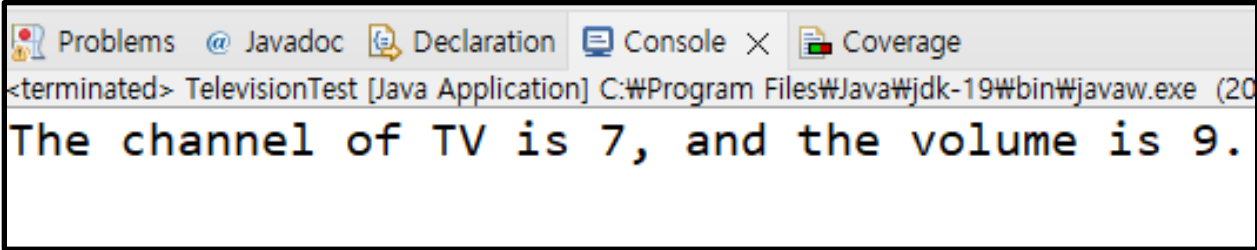
public class TelevisionTest {
    public static void main(String[] args) {
        Television tv = new Television();

        tv.channel = 7;
        tv.volume = 9;
        tv.onOff = true;

        System.out.println("The channel of TV is " + tv.channel +
            ", and the volume is " + tv.volume + ".");
    }
}
```

예제: 텔레비전 클래스

- 실행결과



The screenshot shows an IDE's console window with the following tabs: Problems, Javadoc, Declaration, Console, and Coverage. The Console tab is active, displaying the output of a Java application. The text in the console is: `<terminated> TelevisionTest [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (20` followed by a new line and the output string `The channel of TV is 7, and the volume is 9.`

```
<terminated> TelevisionTest [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (20
The channel of TV is 7, and the volume is 9.
```

예제: 텔레비전 클래스

- TelevisionTest.java 변경

```
package main;

public class TelevisionTest {
    public static void main(String[] args) {
        Television myTv = new Television();
        myTv.channel = 7;
        myTv.volume = 9;
        myTv.onOff = true;

        Television yourTv = new Television();
        yourTv.channel = 9;
        yourTv.volume = 12;
        yourTv.onOff = true;

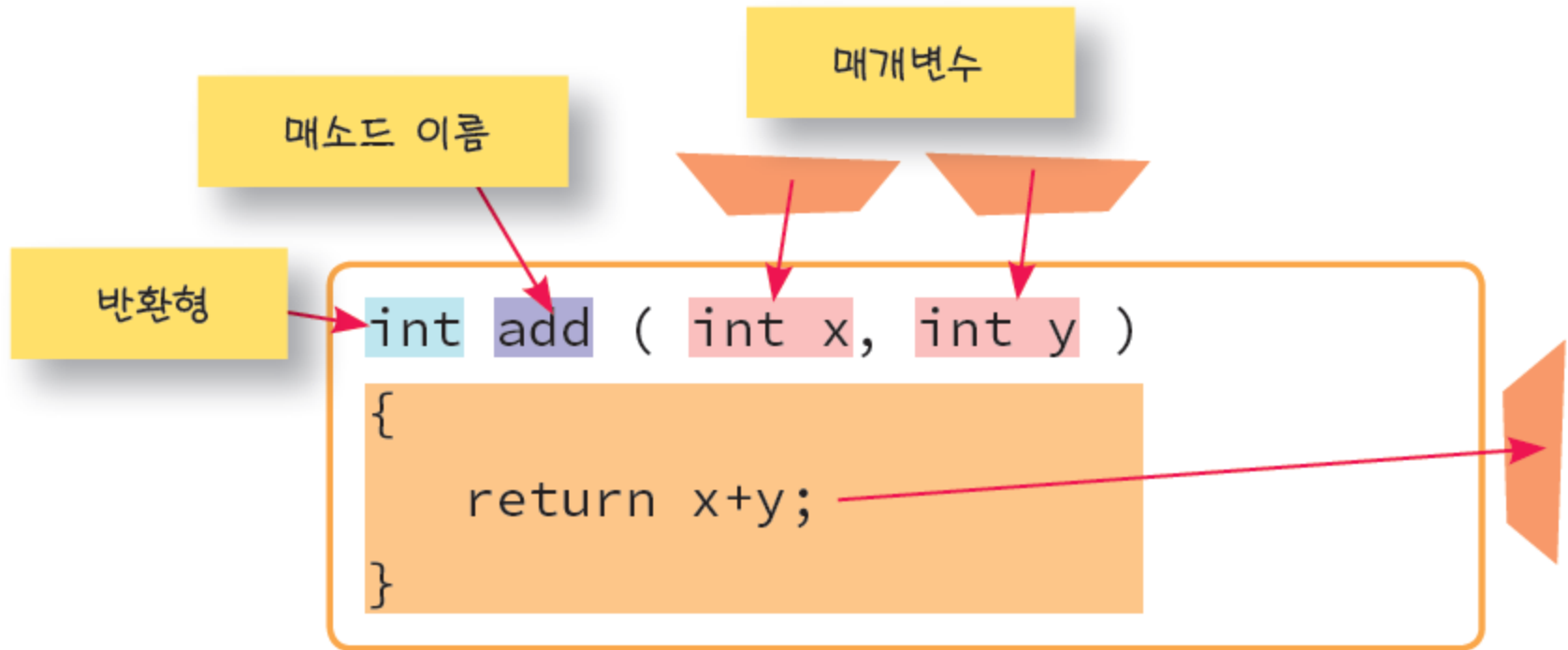
        System.out.println("The channel of myTV is " +
            myTv.channel + ", and the volume is " + myTv.volume + ".");
        System.out.println("The channel of yourTV is " +
            yourTv.channel + ", and the volume is " + yourTv.volume + ".");
    }
}
```

예제: 텔레비전 클래스

- 실행결과

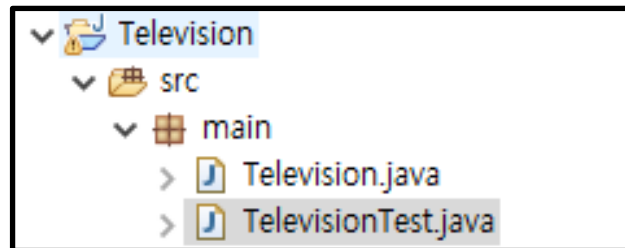
```
The channel of myTV is 7, and the volume is 9.  
The channel of yourTV is 9, and the volume is 12.
```

클래스의 메서드



예제: 텔레비전 메서드

- 프로젝트 구조



예제: 텔레비전 메서드

- Television.java

```
package main;

public class Television {
    int channel;
    int volume;
    boolean onOff;

    void print() {
        System.out.println("The channel is " +
            channel + ", and the volume is " + volume);
    }
}
```

예제: 텔레비전 메서드

- TelevisionTest.java

```
package main;

public class TelevisionTest {
    public static void main(String[] args) {
        Television myTv = new Television();
        myTv.channel = 7;
        myTv.volume = 9;
        myTv.onOff = true;
        myTv.print();

        Television yourTv = new Television();
        yourTv.channel = 9;
        yourTv.volume = 12;
        yourTv.onOff = true;
        yourTv.print();
    }
}
```

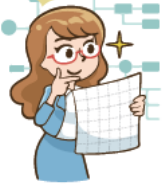
예제: 텔레비전 메서드

- 실행결과

```
The channel is 7, and the volume is 9  
The channel is 9, and the volume is 12
```

클래스 메서드의 반환값

전체적인 구조



형식

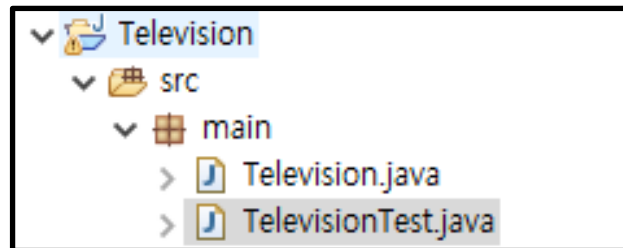
```
return 반환값;
```

return 뒤에 수식을 적으면
수식의 값이 반환됩니다.



예제: 텔레비전 메서드의 반환값

- 프로젝트 구조



예제: 텔레비전 메서드의 반환값

- Television.java

```
package main;

public class Television {
    int channel;
    int volume;
    boolean onOff;

    void print() {
        System.out.println("The channel is "
            + channel + ", and the volume is " + volume);
    }

    int getChannel() {
        return channel;
    }
}
```

예제: 텔레비전 메서드의 반환값

- TelevisionTest.java

```
package main;

public class TelevisionTest {
    public static void main(String[] args) {
        Television myTv = new Television();
        myTv.channel = 7;
        myTv.volume = 9;
        myTv.onOff = true;

        int ch = myTv.getChannel();
        System.out.println("The current channel is " + ch);
    }
}
```

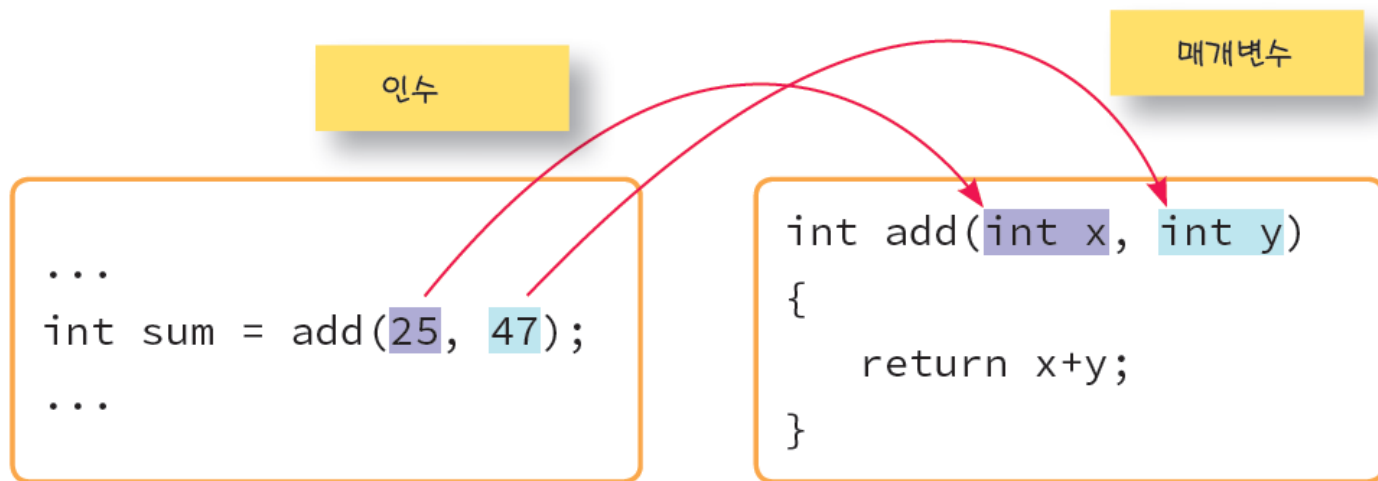

예제: 텔레비전 메서드의 반환값

- 실행결과

```
The current channel is 7
```

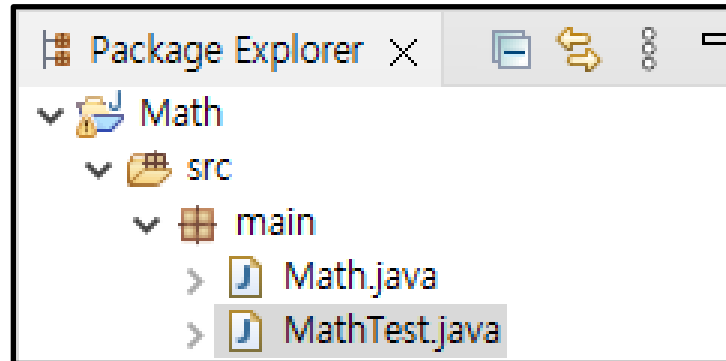
인수(Argument)와 매개변수(Parameter)

- 메서드 호출 시 전달하는 값을 인수(Argument)라 함
- 메서드에서 값을 받을 때 사용하는 변수를 매개변수(Parameter)라 함



예제: 수학 클래스와 덧셈 메서드

- 프로젝트 구조



예제: 수학 클래스와 덧셈 메서드

- Math.java

```
package main;

public class Math {
    int add(int x, int y) {
        return x + y;
    }
}
```

예제: 수학 클래스와 덧셈 메서드

- MathTest.java

```
package main;

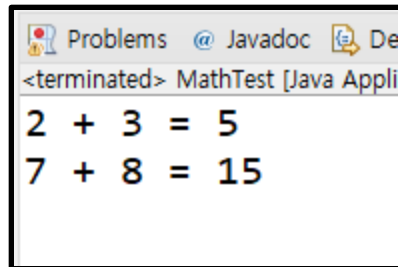
public class MathTest {
    public static void main(String[] args) {
        int sum;
        Math obj = new Math();

        sum = obj.add(2, 3);
        System.out.println("2 + 3 = " + sum);

        sum = obj.add(7, 8);
        System.out.println("7 + 8 = " + sum);
    }
}
```

예제: 수학 클래스와 덧셈 메서드

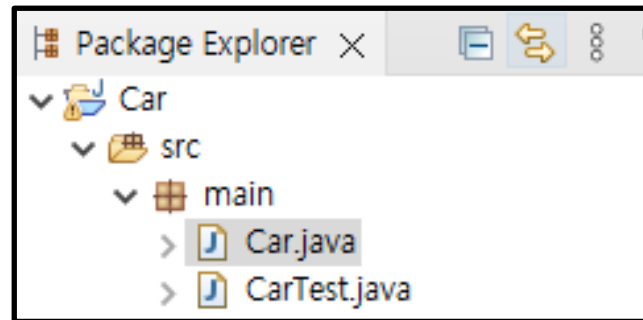
- 실행결과



```
Problems @ Javadoc De
<terminated> MathTest [Java Appli
2 + 3 = 5
7 + 8 = 15
```

예제: 자동차 클래스

- 프로젝트 구조



예제: 자동차 클래스

- Car.java

```
package main;

public class Car {
    String color;
    int speed;
    int gear;

    @Override
    public String toString() {
        return "Car [color=" + color + ", speed="
            + speed + ", gear=" + gear + "];"
    }

    void changeGear(int g) {
        gear = g;
    }

    void speedUp() {
        speed = speed + 10;
    }

    void speedDown() {
        speed = speed - 10;
    }
}
```


예제: 자동차 클래스

- CarTest.java

```
package main;

public class CarTest {
    public static void main(String[] args) {
        Car myCar = new Car();

        myCar.changeGear(1);
        myCar.speedUp();

        System.out.println(myCar);
    }
}
```

예제: 자동차 클래스

- 실행결과

```
Car [color=null, speed=10, gear=1]
```

연습문제 1

- 텔레비전 클래스
 - myTv, yourTv 외에 다른 TV 를 더 만들어보자

```
package main;

public class TelevisionTest {
    public static void main(String[] args) {
        Television myTv = new Television();
        myTv.channel = 7;
        myTv.volume = 9;
        myTv.onOff = true;

        Television yourTv = new Television();
        yourTv.channel = 9;
        yourTv.volume = 12;
        yourTv.onOff = true;

        ??????

        System.out.println("The channel of myTV is " +
            myTv.channel + ", and the volume is " + myTv.volume + ".");
        System.out.println("The channel of yourTV is " +
            yourTv.channel + ", and the volume is " + yourTv.volume + ".");
        ??????
    }
}
```

연습문제 2

- 텔레비전 클래스
 - Television 클래스에 print() 메서드 외에 다른 메서드를 넣고 호출해보자

```
package main;

public class Television {
    int channel;
    int volume;
    boolean onOff;

    void print() {
        System.out.println("The channel is " +
            channel + ", and the volume is " + volume);
    }

    ???
}
```

연습문제 3

- 텔레비전 클래스
 - 멤버 변수 onOff 정보를 반환하는 메서드를 추가하고 호출해보자

```
package main;

public class Television {
    int channel;
    int volume;
    boolean onOff;

    void print() {
        System.out.println("The channel is "
            + channel + ", and the volume is " + volume);
    }

    int getChannel() {
        return channel;
    }

    ????? ?????(){
        ?????
    }
}
```

연습문제 4

- 수학 클래스와 덧셈 메서드
 - 곱셈과 나눗셈도 구현하고 호출해보자

```
package main;

public class Math {
    int add(int x, int y) {
        return x + y;
    }
    ??? ???(???, ???){
        ???
    }
    ??? ???(???, ???){
        ???
    }
}
```

연습문제 5

- 자동차 클래스
 - 자동차 핸들 방향 및 감빡이와 관련된 멤버 변수를 넣고 메서드를 추가하자
 - 택시, 버스 클래스를 만들자
 - 클래스를 추가로 만들때 어떤 문제점이 있다고 생각되는가?

```
package main;

public class Car {
    String color;
    int speed;
    int gear;

    @Override
    public String toString() {
        return "Car [color=" + color + ", speed="
            + speed + ", gear=" + gear + "]";
    }
    void changeGear(int g) {
        gear = g;
    }
    void speedUp() {
        speed = speed + 10;
    }
    void speedDown() {
        speed = speed - 10;
    }
}
```

감사합니다! XD

