자바1

(클래스와 객체)

Chapter 07
클래스와 객체

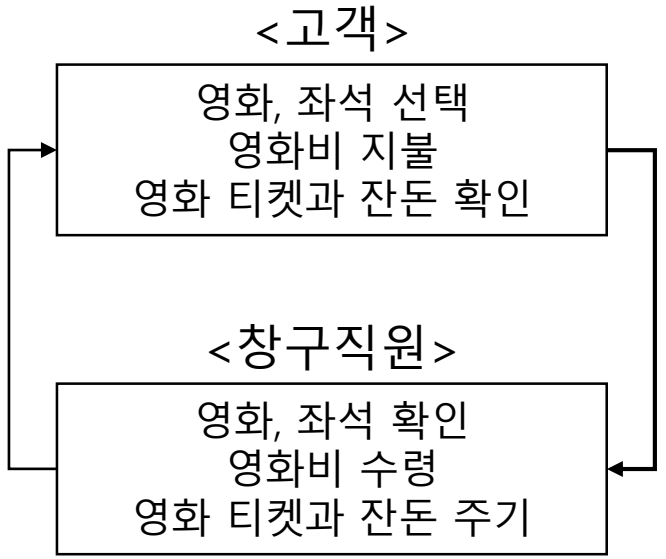
Chapter07의 학습목표

- 객체지향 프로그래밍의 특징을 알아본다.
- 클래스, 객체, 인스턴스의 의미를 이해한다.
- 클래스를 생성하고 사용하는 방법을 익힌다.
- 인스턴스 변수와 인스턴스 메소드에 대해 학습한다.

객체지향 프로그래밍

객체지향 프로그래밍 (OOP : Object-Oriented-Programming)

- 모든 데이터를 오브젝트(물체)로 취급하여 프로그래밍 하는 방법으로,
  처리 요구를 받은 객체가 자기 자신의 안에 있는 내용을 가지고 처리하는 방식
- C, Pascal, BASIC 등과 같은 절차형 언어(procedure-oriented programming)가
  크고 복잡한 프로그램을 구축하기 어렵다는 문제점을 해결하기 위해 탄생

절차적 프로그래밍	객체지향 프로그래밍
고객은 영화, 좌석 선택 ↓ 고객은 영화비 지불 ↓ 창구 직원은 영화, 좌석 확인 ↓ 고객에게 영화 티켓과 잔돈 주기 ↓ 고객은 영화 티켓과 잔돈 확인	

객체지향 프로그래밍

객체지향 프로그래밍의 특징

	자료 추상화	상속	캡슐화	다형성
설명	불필요한 정보는 숨기고 중요한 정보만을 표현하는 것	새로운 클래스가 기존의 클래스의 자료와 연산을 이용할 수 있게 하는 것	데이터(속성)와 데이터를 처리하는 함수를 하나로 묶는 것	어떤 한 요소에 여러 개념을 넣어 놓는 것
특징	객체의 속성들 중에서 가장 필요한 정보들만 간추려 구성하여 복잡한 문제를 다룰 때 사용함	상위 클래스의 모든 속성과 연산을 물려받아 자신의 속성으로 사용이 가능함	객체가 갖고 있는 속성과 데이터를 숨기며, 연산만을 이용해 접근을 허용함	한 가지 형태로 여러가지 기능을 수행할 수 있게 하여 함수와 변수를 상황에 따라 사용할 수 있게 함
장점	유사 모델을 만들어 테스트하기 좋고 시스템의 구조 및 구성을 알아보기 좋음	상위와 하위 클래스의 공유로 인해 코드 및 프로그램 재사용율이 증가	각 객체의 데이터가 외부에 은폐 되어있어 오류 발생이 적음.(정보 은닉)	같은 이름의 함수를 여러 개 정의하고 매개 변수나 타입만을 다르게 호출하는 것이 가능함

클래스와 객체

객체(object) ≒ 인스턴스(instance)

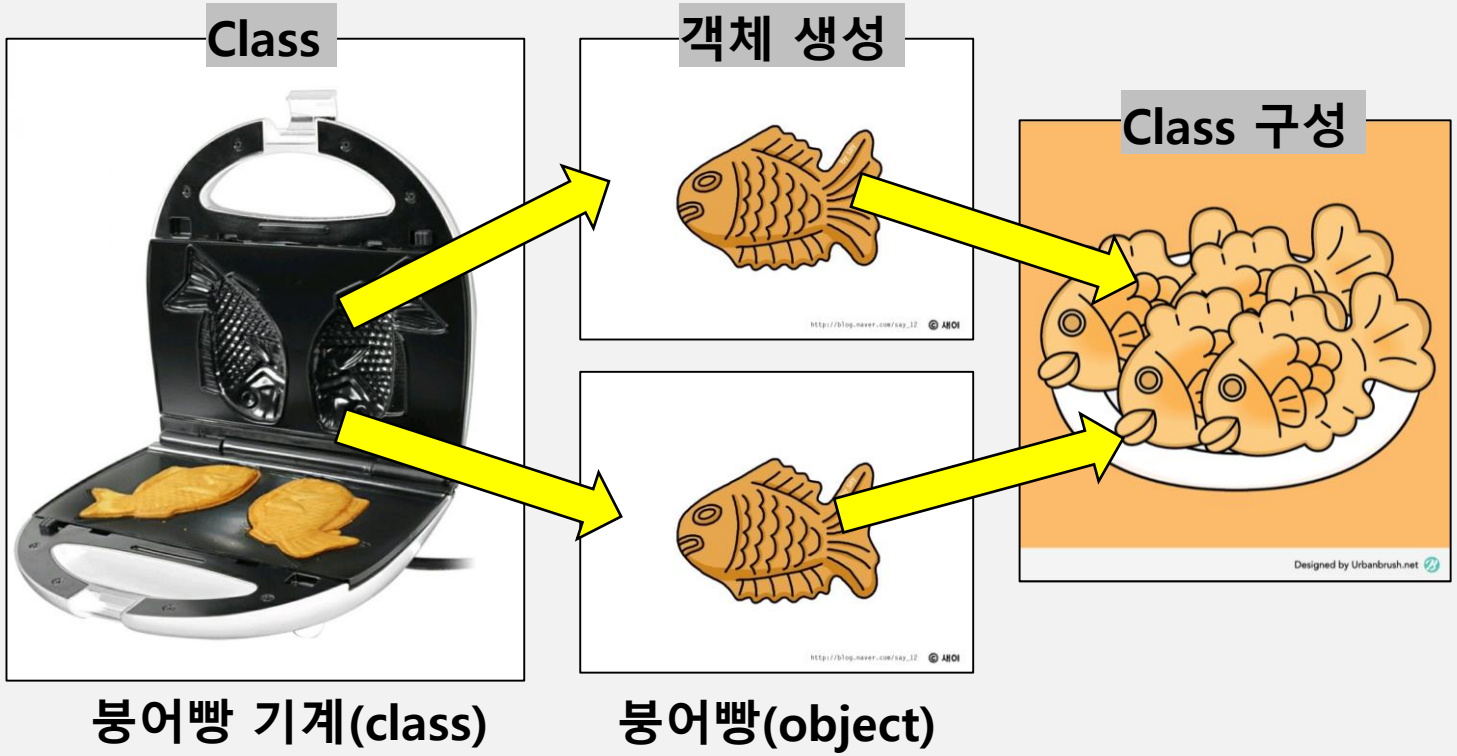
- 서로 다른 많은 데이터를 하나로 묶어서 표현한 것
- 객체 : 실제로 존재하는 것, 사물 또는 개념.
- 용도 : 객체가 가지고 있는 기능과 속성에 따라 다름

클래스(class)

- 객체를 만드는 도구
- 객체를 정의해 놓은 것

게시글

게시글 번호
제목
작성자
비밀번호
내용
최초작성일자
최종수정일자
조회수



클래스와 객체

클래스를 사용하는 이유

- 코드의 재사용성 : 새로운 코드를 작성할 때 기존의 코드를 이용
  - 코드의 가독성/확장성이 높아짐
  - 변수나 데이터의 관리나 접근성이 높아짐
- 신뢰성 높은 프로그래밍 : 제어자와 메소드를 사용해서 데이터를 보호
  - 코드의 중복을 제거하여 오동작을 방지

1) 전달해야 할 정보가 적을 때

```
String name = "emily";
String departemnt = "computer engineering";
String address = "seoul";
int age = 24;
```

2) 전달해야 할 정보가 많을 때

```
String name1 = "emily";
String departemnt1 = "computer engineering";
String address1 = "seoul";
int age1 = 24;

String name2 = "alice";
String departemnt2 = "chemical engineering";
String address2 = "busan";
int age2 = 21;
```

클래스와 객체

클래스의 정의와 객체 생성

- 클래스 정의 : 클래스를 작성하는 것
- class 키워드로 클래스를 정의
- 클래스 이름은 Upper Camel Case 규칙을 따름

클래스 정의

```
class 클래스명{
    본문
}
```

```
class Car{  #class 정의
    boolean powerOn;
    String colorString;
    int wheel;
    int speed;
    boolean wiperOn;
}
```

#클래스 안에서 변수를 생성

네이밍 규칙

종류	예시	특징
Snake Case	my_best_friend	언더바(_)로 연결
Lower Camel Case	myBestFriend	첫 글자는 소문자
Upper Camel Case	MyBestFriend	첫 글자도 대문자

클래스와 객체

클래스의 정의와 객체 생성

객체 생성

클래스명 객체 = new 클래스명()

클래스명 객체1 = new 클래스명()
클래스명 객체2 = new 클래스명()

객체 호출

객체명.변수명

객체명.메소드명()

```
class Car{
    boolean powerOn;
    String color;
    int wheel;
    int speed;
    boolean wiperOn;
}

public class Test {
    public static void main(String[] args) {
        // TODO Auto-generated method stub

        Car my_Car1 = new Car();
        Car my_Car2 = new Car();

        System.out.println(my_Car1.color);

        my_Car1.color = "red";
        my_Car2.color = "blue";

        System.out.println(my_Car1.color);
    }

}
```

#Car클래스의 객체 생성

#my_Car1 객체의 변수 출력

#my_Car1 객체의 변수에 값을 저장

메소드

메소드란?

- 특정 객체/데이터 자료형이 가지고 있는 함수 (function)
- 데이터와 멤버 변수에 대한 접근 권한을 가짐
- 특정한 작업이나 논리를 구성하는 코드를 괄호로 묶어 놓은 것
- 입력 값을 받아서 내부에서 처리하여 결과를 출력/반환

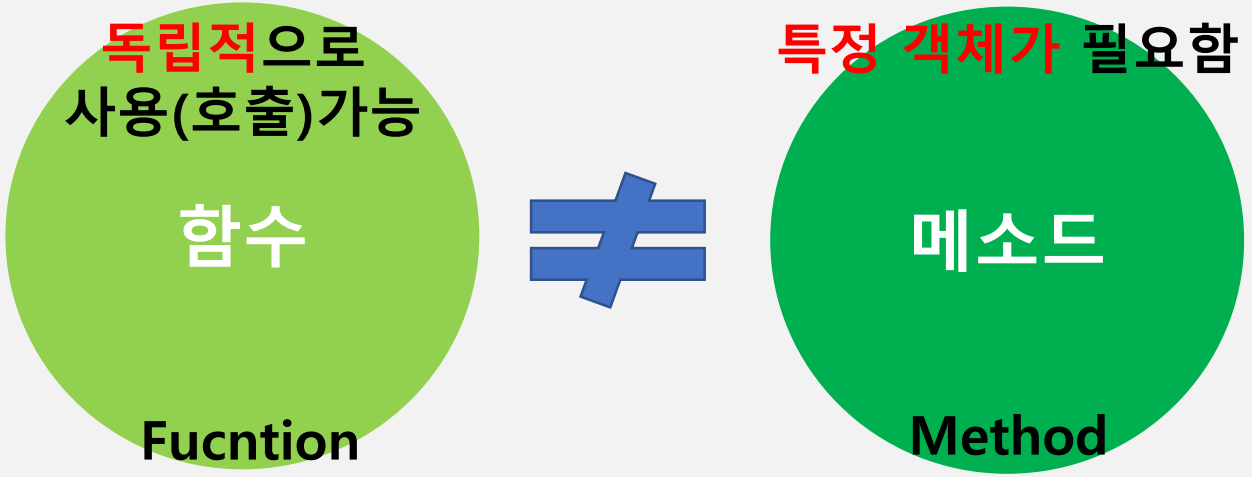
메소드 호출 방식

- 특정 객체를 통해서만 호출가능
- 객체 + ' . '(마침표)를 통해 호출가능

메소드 호출 예)

```
ArrayList <Integer> my_list = new ArrayList<>();
my_list.add(3)
my_list.clear()
```

함수와 메소드의 차이점?



문자열 클래스/메소드

String

- 문자열에 대한 처리를 위한 다양한 메소드가 정의 되어있는 클래스

메소드 명	기능
concat(문자열)	문자열을 연결
substring(index1,index2)	문자열을 index1부터 index2-1까지 잘라서 반환
toUpperCase() / toLowerCase()	문자열을 대문자로 변경 / 문자열을 소문자로 변경
charAt(index)	index위치의 문자를 반환
indexOf(문자열)	해당 문자열의 위치를 반환
equals(문자열)	문자열이 같은지 boolean값으로 반환
trim()	문자열 앞, 뒤에 있는 공백을 제거
replace(바꾸고 싶은 문자열, 바꿀 문자열)	문자열 내의 특정 부분을 다른 문자열로 변경
replaceAll(바꾸고 싶은 문자열, 바꿀 문자열)	문자열 내의 특정 부분을 다른 문자열로 변경 (문자열 내에서 해당하는 문자 전체)
split(기준문자열)	기준문자열을 기준으로 문자열을 나눠서 배열 형태로 반환
toCharArray()	문자열을 문자 1개씩 나눠 char형 배열형태로 반환

문자열 클래스/메소드

StringBuffer / StringBuilder

- 문자열에 대한 처리를 위한 다양한 메소드가 정의 되어있는 클래스
- StringBuffer는 StringBuilder와 사용과 메서드가 같지만 StringBuffer는 멀티 스레드에서 동기화 처리를 지원

메소드 명	기능
append(값)	매개변수로 입력된 값을 문자열로 바꾸어서 더해줌
reverse()	문자열의 순서를 반대로 함
insert(index, 값)	매개변수로 입력된 값을 문자열로 바꾸어서 index위치에 추가
deleteCharAt(index)	index위치의 문자를 제거
delete(index1, index2)	index1위치부터 index2위치까지의 문자를 제거
charAt(index)	index위치의 문자를 반환
indexOf(문자열)	해당 문자열의 위치를 반환
substring(index1,index2)	문자열을 index1부터 index2까지 잘라서 반환
length()	해당 문자열의 길이를 반환
replace(바꾸고 싶은 문자열, 바꿀 문자열)	문자열 내의 특정 부분을 다른 문자열로 변경 (문자열 중 가장 앞에 있는 문자 1개만)

배열 클래스/메소드

ArrayList/LinkedList

- ArrayList 배열 처리를 위한 다양한 메소드가 정의되어있는 클래스

메소드 명	기능
add(값)	마지막 요소로 값을 추가
add(인덱스, 값)	인덱스 위치에 값을 추가
clear()	배열요소들을 초기화
clone()	배열을 복사
size()	배열의 크기를 반환
contains(값)	배열에 값이 존재하는지 검사
get(인덱스)	배열의 index요소를 반환
insert(index, 값)	list의 index위치에 값/객체를 추가
indexOf(값)	해당 값이 존재하는 배열의 index위치를 반환
remove(index)	해당 index위치의 값을 배열에서 제거
set(index, 값)	해당 index위치의 값을 해당 값으로 변경
sort(null)	list를 오름차순으로 정렬

Wrapper 클래스/메소드

Boolean/Character/Byte/Integer/Long/Double/Float

- 기본 자료형들을 객체로 다룰 때 지원하는 메소드
- 객체로 다루어지지 않는 기본 자료형들을 객체로 다루기 위해 도입한 클래스

메소드 명	기능
자료형이름+Value()	Wrapper 클래스 객체를 기본 자료형 형태로 변경
toString()	해당 객체의 값을 String으로 변경
insert(index, 값)	매개변수로 입력된 값을 문자열로 바꾸어서 index위치에 추가
valueOf(문자열)	문자열을 해당 객체 타입으로 변경
max(숫자, 숫자)	두 숫자 중 가장 큰 값을 반환
min(숫자, 숫자)	두 숫자 중 가장 작은 값을 반환
parse+자료형이름(문자열)	해당 문자열을 기본 자료형 형태로 변경
sum(숫자, 숫자)	두 숫자의 합을 반환
compareTo(값)	해당 객체와 값을 비교해 동일한 경우 0, 객체의 값이 클 경우 1, 매개변수 값이 클 경우 -1을 반환

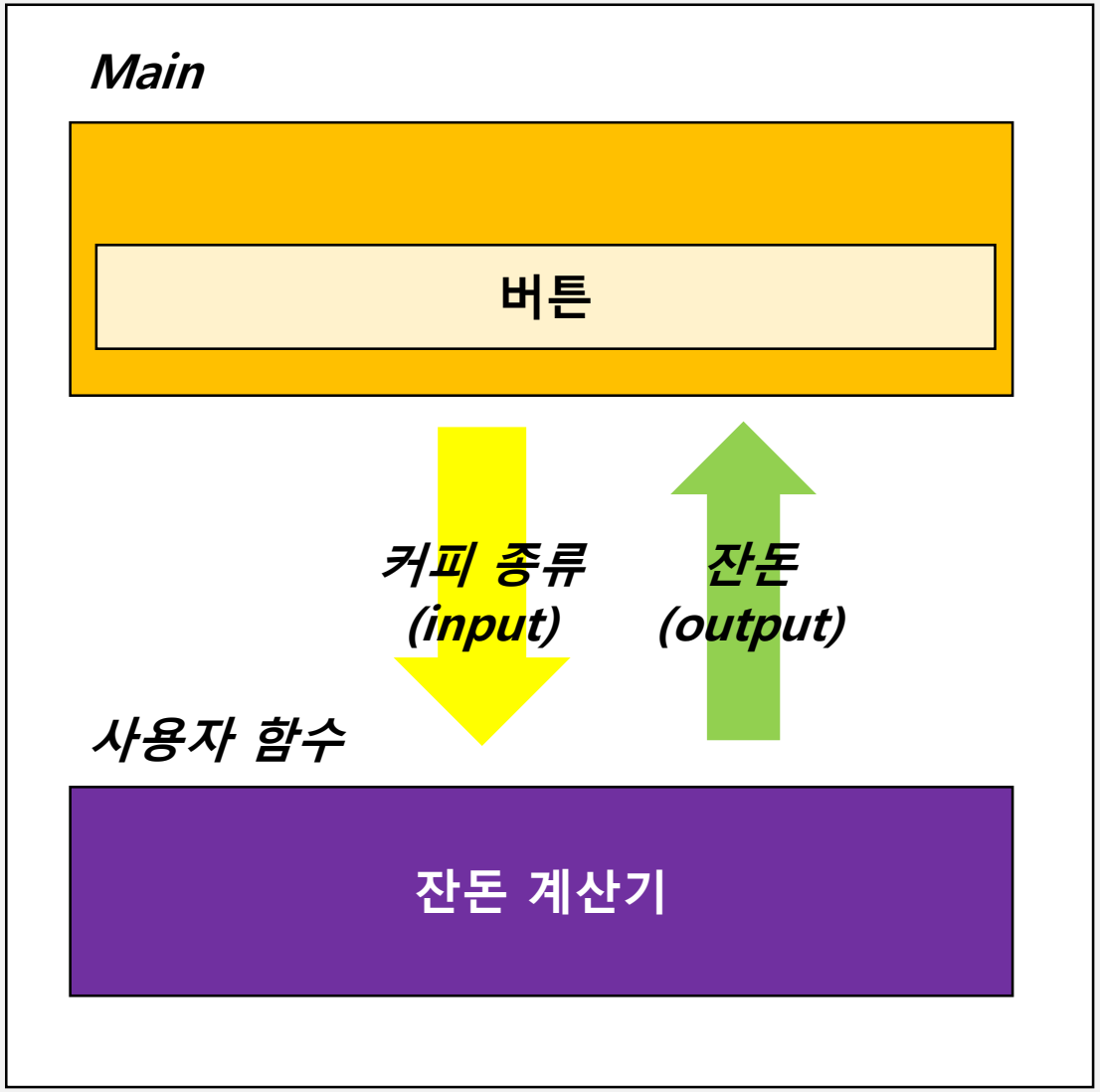
사용자 메소드

사용자 메소드 (사용자 함수)

- 사용자가 직접 만든 함수
- 내장함수 외의 기능 혹은 자신이 원하는 기능을 작성

1. 커피자판기에 돈을 넣는다
2. 마시고 싶은 커피의 종류를 골라 버튼을 누른다
3. 커피자판기는 선택된 커피와 경우에 따라 잔돈을 준다

1. 메소드 정의 : 사용자 메소드를 새로 만드는 것
2. 인수 : 사용자 함수에 전달할 입력(input)
   argument라고도 함
3. 매개변수 : 인수를 받아서 저장하는 변수
   parameter라고도 함
4. 반환값 : 사용자 함수의 출력(output)
   return이라고도 함
5. 함수 호출 : 만들어진 사용자 함수를 사용하는 것



사용자 메소드

사용자 메소드의 정의

- return문은 반환 값이 없을 경우에는 작성하지 않음
- 메소드의 반환 타입이 void일 경우를 제외하고는 반드시 메소드 내부에 return문이 있어야 함
- 메소드 이름은 사용자가 지정

```
반환타입 메소드명(자료형타입 매개변수, 자료형타입 매개변수, ...){
    본문
    return 반환값;
}
```

메소드 호출

- 정의된 메소드를 사용하는 것(정의 후 사용)

```
참조변수.메소드명(인수);
```

1) 인수: X, 반환값: X

객체명.메소드이름()

```java
class House{
    void welcome() {
        System.out.println("Hello Java!");
    }
}

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

        House my_house = new House();
        my_house.welcome();
    }

}
```

사용자 메소드

2) 인수: O, 반환값: X

객체명.메소드이름(인수)

```
class House{
    void welcome(String s) {
        System.out.println(s);
    }
}

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

        House my_house = new House();
        my_house.welcome("Hello Java!");
    }

}
```

3) 인수: X, 반환값: O

자료형 변수명 = 객체명.메소드이름(인수)

```
class House{
    String welcome() {
        return "Hello Java!";
    }
}

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

        House my_house = new House();
        String word = my_house.welcome();
        System.out.println(word);
    }

}
```

사용자 메소드

4) 인수: O, 반환값: O

자료형 변수명 = 객체명.메소드이름(인수)

```
class Calc{
    int result(int number) {
        int new_number = number + 10;
        return new_number;
    }
}

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

        Calc clcu1 = new Calc();
        int num1 = clcu1.result(3);
        int num2 = clcu1.result(5);
        System.out.println(num1);
        System.out.println(num2);
    }

}
```

사용자 메소드

인수와 매개변수

- 함수로 전달되는 인수(argument)를 저장하는 변수를 매개변수(parameter)라고 함

```
class Person{
    void introduce(String name, int age) {
    System.out.println("내 이름은 " + name);
    System.out.println("나이는 " + age);
    }
}

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

    Person new_person = new Person();
    new_person.introduce("james", 25);
    }
}
```

```
//객체 생성
Person new_person = new Person();

//메소드 정의
void introduce(String name, int age)

//인수를 매개변수에 전달
new_person.introduce("james", 25);

//introduce함수 실행
new_person.introduce("james", 25);
```

사용자 메소드

반환값

- 메소드의 결과 값을 돌려주는 것을 반환이라고 함
- return을 만나면 즉시 함수를 빠져나옴

```
class Person{
    String address() {
    String str = "우편번호 12345\n";
    str += "대구시 중구 반월당";
    return str;
    }
}

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

    Person new_person = new Person();
    System.out.println(new_person.address());
    }
}
```

#address함수에 반환값 str가 반환

함수의 종료를 위한 return

- 값을 반환하지 않고 함수가 종료
- 함수의 특정 부분에서 함수를 빠져나오고 싶을 때 사용

```
class Energy{
    void charge(int energy) {
    if(energy < 0) {
        System.out.println("0보다 작은 에너지는 충전 불가능");
        return;
        }
    System.out.println("에너지가 충전됨");
    }
}

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

    Energy charger = new Energy();
    charger.charge(1);
    charger.charge(-1);
    }
}
```

사용자 메소드 예제

커피 자판기 프로그램 작성

1. 커피 자판기에 돈과 주문할 커피를 전달합니다.
2. 주문할 수 있는 커피의 종류와 가격은 다음과 같습니다.
- 아메리카노 : 1000원
- 카페라떼 : 1500원
- 카푸치노 : 2000원
3. 없는 커피를 주문할 경우 입력한 돈을 그대로 반환합니다.
4. 구매 금액이 부족하면 입력한 돈을 그대로 반환합니다.
5. 정상 주문이면 주문한 커피와 잔돈을 반환합니다.

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

        Coffee_machine machine = new Coffee_machine();
        Scanner input = new Scanner(System.in);

        System.out.println("커피를 선택하세요. (아메, 카페, 카푸) >>>");
        String order = input.next();
        System.out.println("얼마를 내시나요? >>>");
        int pay = input.nextInt();

        int change = machine.calc_Change(pay, order);
        System.out.println("잔돈은 "+change+"원입니다.");
    }
}
```

사용자 메소드 예제

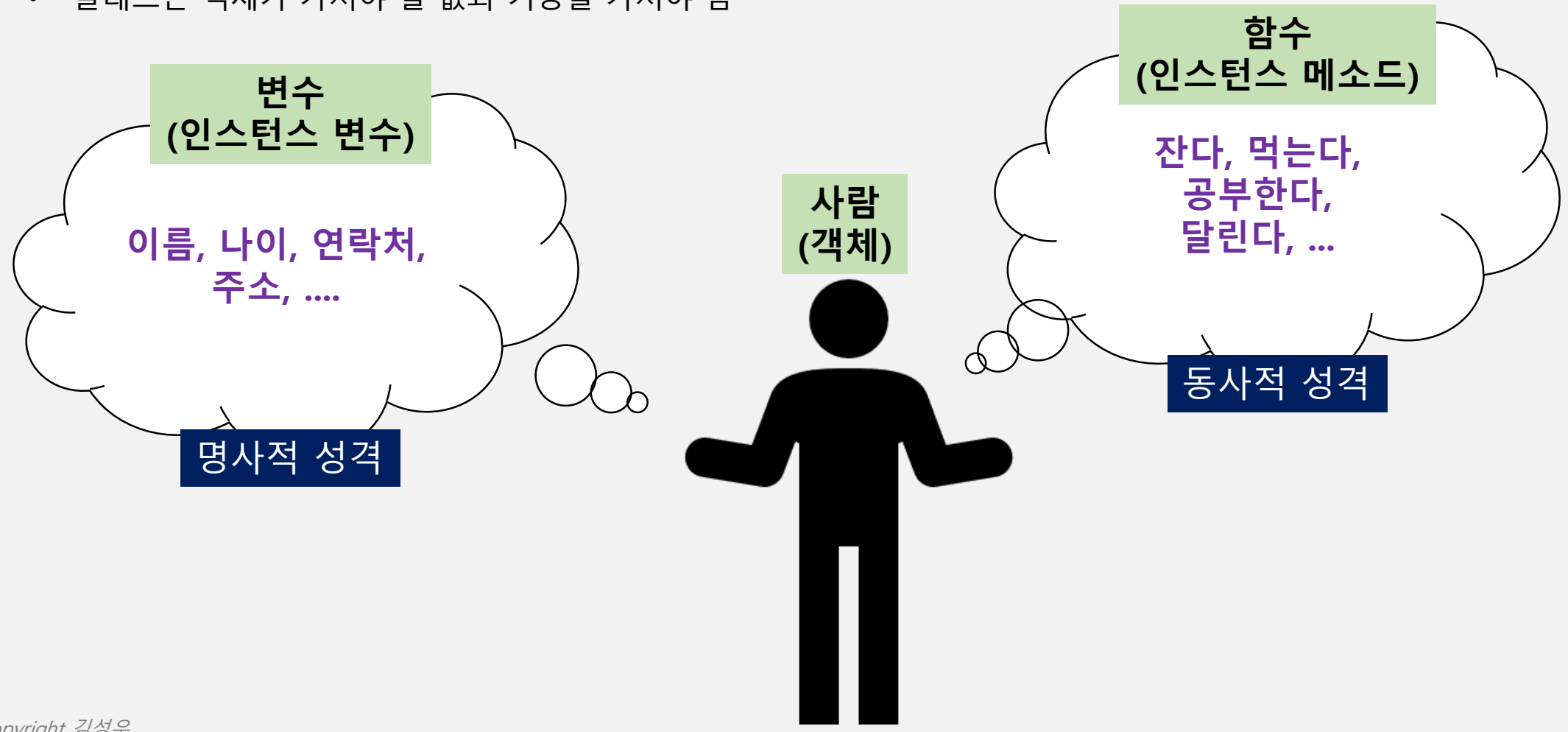
커피 자판기 프로그램 작성

```
class Coffee_machine{
    int calc_Change(int money, String pick) {
        System.out.println(money+"원에 "+pick+"를 선택하셨습니다.");

        int cost = 0;

        switch (pick) {
            case("아메리카노"):
                cost = 1000;
                break;
            case("카페라떼"):
                cost = 1500;
                break;
            case("카푸치노"):
                cost = 2000;
                break;
        default:
            System.out.println(pick+"는 판매하지 않습니다.");
            return money;
        }

        if (cost > money) {
            System.out.println("금액이 부족합니다.");
            return money;
        }
        else {
            System.out.println(pick+"가 나옵니다.");
            return money - cost;
        }
    }
}
```

클래스의 구성

클래스의 구성

- 클래스는 객체가 가져야 할 값과 기능을 가져야 함



클래스의 구성

인스턴스 변수와 인스턴스 메소드

인스턴스 변수

1) 클래스를 기반으로 만들어지는 모든 인스턴스들이 각각 따로 저장하는 변수
   2) 객체마다 가지는 고유한 변수
2) 인스턴스를 생성할 때 만들어짐
3) 인스턴스마다 다른 값을 가져야 할 경우 인스턴스 변수로 선언

인스턴스 메소드

1) 객체를 생성해야 호출이 가능한 메소드
2) 객체가 생성되어야 메모리에 올라감
3) 메소드 앞에 static이 붙어있지 않음

Car 클래스 구현

```
class Car{
    int speed = 0;

    void speed_up() {
        speed++;
    }
}

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

        Car car1 = new Car();
        Car car2 = new Car();

        car1.speed_up();
        car2.speed_up();
    }
}
```

클래스의 구성

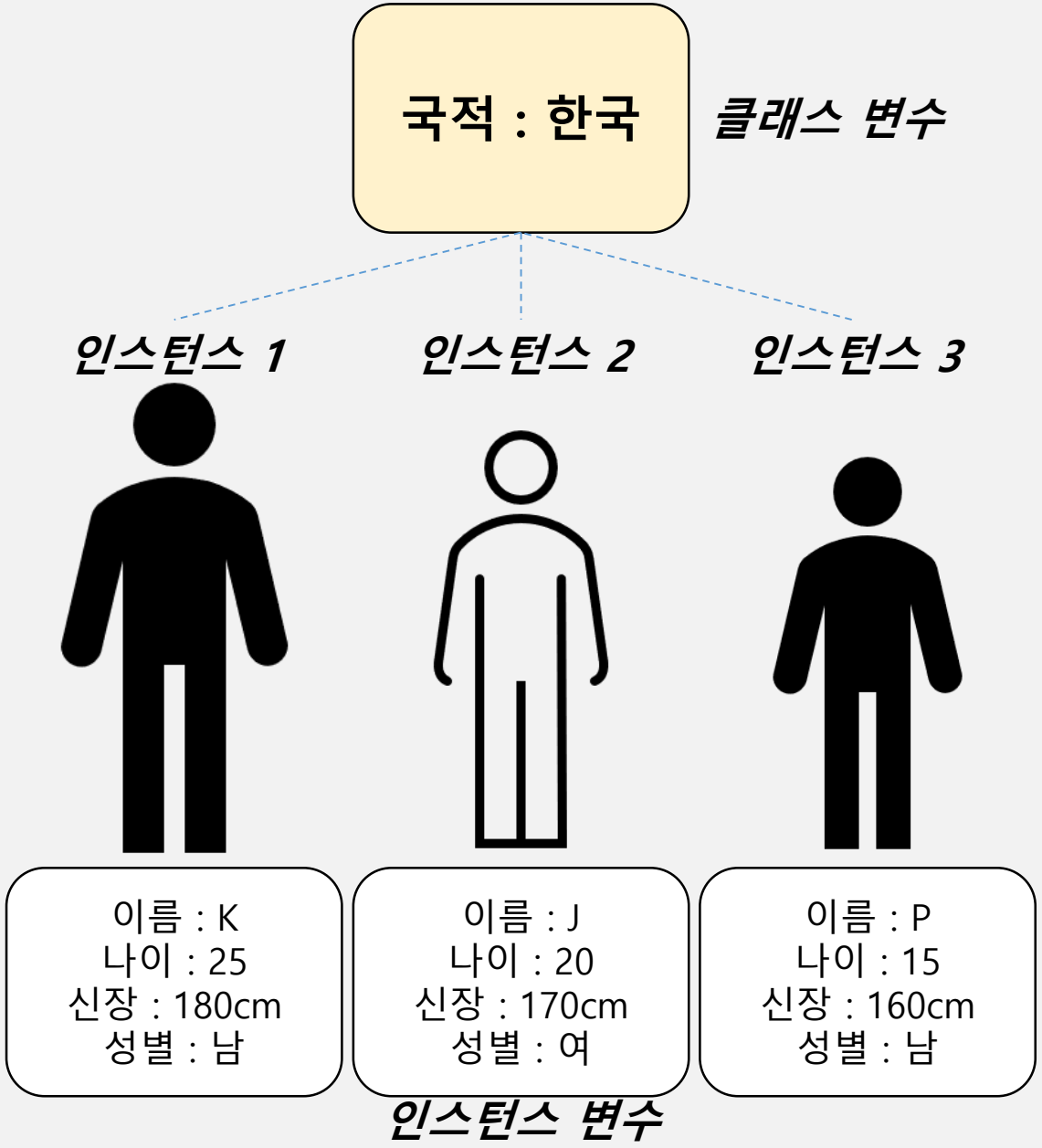
클래스 변수와 클래스 메소드

클래스 변수

1) 모든 인스턴스가 동일한 값을 사용할 때 정의함
2) 모든 인스턴스들이 공유하도록 처리
3) 메모리 공간을 절약할 수 있음
4) 클래스가 처음 메모리에 로딩될 때 생성
5) 변수 선언 시 static 키워드를 사용

클래스 메소드

1) 객체를 생성하지 않아도 호출이 가능
2) 메소드 선언 시 static 키워드를 사용



클래스의 구성

클래스 변수와 클래스 메소드

```
class Car{
    static int speed = 0;

    static void speed_up() {
        speed++;
        System.out.println(speed);
    }
}

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

        Car car1 = new Car();
        Car car2 = new Car();

        car1.speed_up();
        car2.speed_up();

        Car.speed++;
        Car.speed_up();
    }
}
```

Car 클래스 구현

```
class Car{
    static int speed = 0;

    static void speed_up() {
        speed++;
        System.out.println(speed);
    }
}

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

        Car car1 = new Car();
        Car car2 = new Car();

        car1.speed_up();
        car2.speed_up();

        Car.speed++;
        Car.speed_up();
    }
}
```

#클래스 변수 speed

#클래스 메소드 speed_up()

#Car 객체 생성

#Car 객체에서 speed_up 호출

#클래스에서 바로 변수명과 메소드명으로 호출

클래스 예제

CPU, RAM, VGA, SSD를 구성요소로 가지고 있는 Computer 클래스를 이용해서 인스턴스를 생성하는 프로그램을 구현하세요. 값을 저장할 수 있는 set_spec(), 값을 출력하는 hardware_info() 메소드로 구성

함수 정의

- class이름 : Computer
- 인스턴스 메소드 :
  set_spec()
  hardware_info()
- 인스턴스 변수 :
  cpu, ram, vga, ssd

```java
public class Test {
    public static void main(String[] args) {
        Computer com1 = new Computer();
        Computer com2 = new Computer();
        com1.set_spec("i7", "16GB", "GTX1060", "512GB");
        com1.hardware_info();

        com2.set_spec("i5", "8GB", "MX300", "256GB");
        com2.hardware_info();

    }
}
```

```java
class Computer{
    String cpu;
    String ram;
    String vga;
    String ssd;

    void set_spec(String _cpu, String _ram, String _vga, String _ssd) {
        cpu = _cpu;
        ram = _ram;
        vga = _vga;
        ssd = _ssd;
    }

    void hardware_info() {
        System.out.println("CPU = "+cpu);
        System.out.println("RAM = "+ram);
        System.out.println("VGA = "+vga);
        System.out.println("SSD = "+ssd);
    }
}
```

실습 문제

1. 다음 지시사항을 읽고 책 제목과 저자 정보를 저장할 수 있는 Book 클래스를 생성하세요.

지시사항

1. 책 제목과 책 저자 정보를 출력하는 print_info() 메소드를 구현하세요.
2. 다음과 같은 방법으로 book1과 book2 인스턴스를 생성하세요.

```
//book1, book2 인스턴스 생성
Book book1 = new Book();
Book book2 = new Book();
```

```
//book1, book2 제목과 저자 정보 저장
book1.set_info("파이썬", "민경태");
book2.set_info("어린왕자", "생택쥐페리");
```

3. 생성된 인스턴스 boo1과 book2의 책 제목과 책 저자 정보를 아래와 같이 출력하세요.

```
책 제목 : 파이썬
책 저자 : 민경태
책 제목 : 어린왕자
책 저자 : 생텍쥐페리
```

실습 문제

2. 다음 지시사항을 읽고 노래 제목과 장르 정보를 저장할 수 있는 Song 클래스와 가수 이름과 대표곡 정보를 저장할 수 있는 Singer 클래스를 구현하세요.

지시사항

1. 다음과 같은 방법으로 song과 singer 인스턴스를 생성하고 필요한 정보를 저장한 뒤 그 정보를 출력하세요.
2. Song 클래스는 다음 메소드를 반드시 가지고 있어야 합니다.
   set_song() 메소드 : 노래 제목과 장르를 저장하는 메소드
   print_song() 메소드 : 노래 제목과 장르를 출력하는 메소드
3. Singer 클래스는 다음 메소드를 반드시 가지고 있어야 합니다.
   set_singer() 메소드 : 가수 이름을 저장하는 메소드
   hit_song() 메소드 : 대표곡을 저장하는 메소드
   print_singer() 메소드 : 가수 이름과 대표곡 제목과 장르를 출력하는 메소드

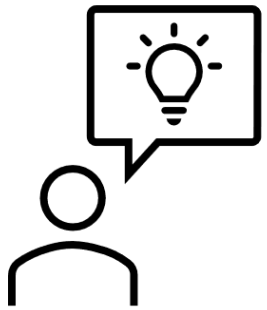
```
//song 인스턴스 생성
Song song = new Song();
song.set_Song("취중진담", "발라드");

///singer 인스턴스 생성
Singer singer = new Singer();
singer.set_Singer("김동률");

//singer의 대표곡 지정
singer.hit_Song(song);

//singer 정보 출력
singer.print_singer();
```

```
가수이름: 김동률
노래제목: 취중진담(발라드)
```

수고하셨습니다.

Chapter07에서 배운 내용

- 클래스
- 객체
- 인스턴스 변수
- 인스턴스 메소드



Chapter08에서 배울 내용

- 클래스와 객체2
- 생성자
- 상속