09장 객체와 클래스

객체와 클래스

◈ 객체 지향 프로그래밍(OOP: Object Oriented Programming)

객체 지향 프로그래밍에서는 모든 데이터를 객체(Object)로 취급하며, 이러한 객체가 프로그램의 중심이 된다.

◈ 객체

객체란 어떤 대상이 상태(State)와 행위(Behavior)를 갖는 것

객체는 상태를 나타내기 위해 변수(variable)를 사용하며, 행위들은 함수(Method)로 표현된다.

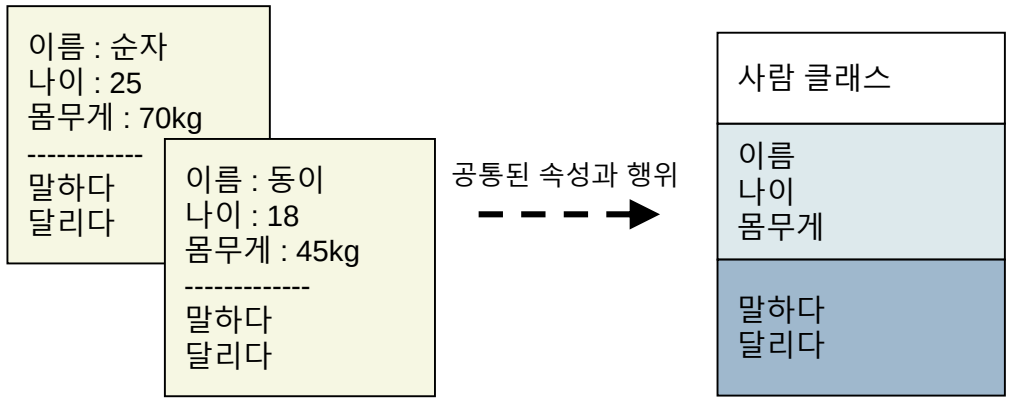
객체와 클래스

◈ 클래스(Class)

클래스는 상태와 행위가 같은 객체들을 대표하는 것

객체의 상태를 나타내는 필드(field)와 객체의 행위를 나타내는 메서드(method)로 구성된다.

객체를 통해서 구체화 된다.



객체와 클래스

◈ 클래스의 구성요소

클래스의 구성요소에는 필드(Field), 생성자(Constructor), 메서드(Method)가 있다.

```
public class Child {
    String name;
    int age;

    Child(String name, int age){
        this.name = name;
        this.age = age;
    }

    public void sayHello() {
        System.out.println(name + "가 인사합니다. 안녕하세요. ^o^//");
    }
}
```

필드

생성자

메서드

객체와 클래스

◈ 객체 생성

자바의 new 연산자를 사용하여 생성한다.

클래스명 객체명 = new 생성자(매개변수);

별도로 생성자를 정의하지 않은 클래스는 매개변수를 넣지 않으며,

그 경우 참조형 필드 값은 null로 초기화 되며, 숫자형 필드 값은 0으로 초기화 된다.

◈ 멤버 참조

멤버 참조는 . 연산자를 이용한다.

객체와 클래스

◈ 생성자

객체의 생성과 동시에 자동으로 호출되어 변수의 초기화 작업을 하는 메서드

생성자의 이름은 클래스 이름과 같아야 한다.

하나의 클래스가 여러 개의 생성자를 가질 수 있다.

◈ this

객체 내부에서 객체 자신을 참조하는 변수

지역변수와 전역변수를 구별할 때 사용하는 변수

```java
public class Child {
    String name;
    int age;

    Child(String name, int age){
        this.name = name;
        this.age = age;
    }

    Child(String name){
        this(name, 0);
    }

    Child(){

    }
```

객체와 클래스

◈ this()

현재 객체의 생성자를 의미한다.

반드시 생성자의 첫 행에 정의해야 한다.

특정 생성자에서 Overloading되어 있는 다른 생성자를 호출할 수 있다.

```
public class Child {
    String name;
    int age;

    Child(String name, int age){
        this.name = name;
        this.age = age;
    }

    Child(String name){
        this(name, 0);
    }

    Child(){

    }
```

객체와 클래스

◈ 필드

객체의 특징적인 속성을 변수와 상수로 정의한다.

◈ 캡슐화(Encapsulation)

캡슐화란 외부에서 객체 내부 속성에 직접 접근하거나 변경할 수 없게 하고, 외부에서 접근할 수 있도록 정의된 오퍼레이션을 통해서만 내부 속성에 접근할 수 있도록 하는 것이다.

```java
public class Account {
    private int budget;

    public int getBudget() {
        return budget;
    }

    public void setBudget(int budget) {
        this.budget = budget;
    }
}
```

캡슐화 →

```java
public static void main(String[] args){
    Account account = new Account();     // 캡슐화된 클래스

    account.budget = 1000000;            // 직접 변경 불가능
    account.setBudget(1000000);          // 오퍼레이션을 통해 변경

    int budget = account.budget;         // 직접 접근 불가능
    int budget = account.getBudget();    // 오퍼레이션을 통해 접근
}
```

객체와 클래스

◈ 클래스의 정적(static) 구성 요소

필드와 메서드를 클래스 자체에 속하는 구성 요소로 선언 시 필요

static 필드 및 메서드는 객체의 생성 없이도 사용이 가능하다.

◈ 정적 메서드

특정 객체에 종속되지 않고 기능을 수행하는 기능적 메서드에 유용

클래스 필드에 정의된 변수를 사용할 시, 정적 변수만 사용할 수 있다.

객체와 클래스

◈ 싱글톤(Singleton) 패턴

어떤 클래스가 최초 한번만 메모리를 할당하고, 그 메모리에 객체를 만들어 사용하는 디자인 패턴

해당 클래스의 생성자 호출이 반복되어도 최초 생성된 객체만 반환하도록 한다.

메모리 낭비를 방지할 수 있으며, 프로젝트 내에 단 하나의 객체로 존재해도 되는 클래스에 적용한다.

싱글톤 패턴 적용

```
public class ExampleClass {
    //Instance
    private static ExampleClass instance = new ExampleClass();

    //private 생성자
    private ExampleClass() {}

    public static ExampleClass getInstance() {
        return instance;
    }
}
```

```
public static void main(String[] args)  {
    ExampleClass exClass = ExampleClass.getInstance();

}
```

싱글톤 클래스 호출

객체와 클래스

◈ 클래스 리플렉션(Reflection)

객체를 통해 클래스의 정보를 분석하는 기법

클래스의 생성자, 필드, 메서드 정보를 알아낼 수 있다.

메서드	설명
getMethods()	클래스 내에 포함된 메서드 알아내기
getFields()	클래스 내에 포함된 필드 알아내기
getConstructors()	클래스 내에 존재하는 생성자 알아내기
getInterfaces()	클래스 내에 존재하는 인터페이스 알아내기
getSuperclass()	클래스 내에 존재하는 상위클래스 알아내기