[5주차] Chapter 06 클래스

1. 객체 지향 프로그래밍의 개념과 클래스

> 현실 세계의 객체를 소프트웨어 객체로 설계하는 것을 모델링이라고 한다.

```
class Person {
    // filed 필드
    public String name;
    public int age;
    // method 메소드
    public void eating(){

    }
}
```

객체 지향 프로그램은 객체가 다른 객체와 서로 상호작용 하면서 동작한다. 객체들 사이의 상호작용 수단은 메소드이다. 객체가 다른 객체의 기능을 이용할 때 이 메소드를 호출한다.

객체 간의 관계에는 집합 관계, 사용 관계, 상속 관계가 있다. 자동차 객체 안에 타이어 핸들 등의 객체가 포함되는 것을 집함, 사람이라는 객체 안에서 사용되는 자동차 객체는 사용, 자동차 객체가 기계라는 부모 객체를 상속받아 사용하고 있다면 상속 관계가 된다.

- 객체 지향 프로그래밍의 특징

캡슐화 캡슐화란 객체 내부 구조를 숨기고 접근 제한자로 제공되는(public) 필드와 메소드만 사용 가능한 것을 말한다. 객체가 손상되지 않게 하기 위함이다.

상속 코드의 재사용성을 높이고 유지 보수 시간을 최소화하기 위해 사용한다.

다형성 상속과 인터페이스 구현을 통해 다양한 실행 결과가 나올 수 있도록 하는 것이다. 빨간옷 객체와 파란옷 객체가 있을 때 여자사람 객체한테 빨간옷 객체를 입힐 수도, 파란옷 객체를 입힐 수도 있는 것을 말한다.

2. 클래스

> 클래스로부터 생성된 객체를 해당 클래스의 인스턴스라고 부른다. 그리고 클래스로부터 객체를 만드는 과정을 인스턴스화라고 한다. 동일한 클래스로부터 여러 개의 인스턴스를 만들 수 있다.

- 특징

1. 클래스명 첫 문자는 대문자로 한다.
2. 캐멀 스타일로 작성한다.
3. 특수문자($, _)와 숫자를 포함할 수 있다.
4. 하나의 파일에 복수개의 클래스를 선언하면 컴파일 된 바이트 코드 파일(.class) 은 클래스 수만큼 생성된다.
5. 한 파일에 복수 개의 클래스가 존재할경우 파일명과 동일한 클래스만 공개 클래스로 선언할 수 있다. 일반적으로 한 파일당 클래스 하나를 사용한다.

```java
// Person.java
package test;

public class Person {
    // 생성자를 정의하지 않았다면 컴파일러가 자동으로 기본 생성자를 생성한다.
}

// Run.java
package test;

public class Run {
    // new 연산자 디음에는 생성자 호출 코드가 온다.
    Person person = new Person();
}
```

new 연산자 뒤에는 Person 클래스의 생성자를 호출하여 person 이라는 객체를 생성하고 초기화한다. 객체는 힙 메모리에 할당되며, person 변수는 그 객체를 참조한다. 변수에 할당된 객체를 생성하면 해당 객체의 주소(참조)가 변수에 저장되므로, 이후에 변수를 통해 객체에 접근할 수 있다.

생성자 선언과 호출

- 클래스 안에 생성자가 없을 경우 컴파일 할 때 자동 생성된다.
- 리턴 타입이 없다.
- 클래스와 이름과 접근 제한자가 동일하다.

> new 연산자는 객체를 생성한 후 생성자를 호출하여 객체를 초기화하는 역할을 한다. 객체 초기화란 필드 초기화를 하거나 메소드를 호출해서 객체를 사용할 준비를 하는 것을 말한다.

- 기본 생성자 모든 클래스는 생성자가 존재하며, 하나 이상을 가질 수 있다. 클래스와 생성자의 접근 제한자는 동일하다. 클래스가 public class 로 선언되면 생성자로 public 이 붙는다. 생성자가 존재하지 않으면 컴파일러가 자동으로 생성하며, 그렇기 때문에 new 연산자 뒤에 기본 생성자를 호출할 수 있다.

```java
// 소스 파일 (Test.java)
public class Test {
    클래스 변수 = new 클래스();
}

// 바이트 코드 파일 (.class)
public class Test {
    public Test() {} // 자동 추가
}
```

- 생성자 초기화

1. 생성자를 사용한 초기화

> 생성자 메서드는 객체를 생성할 때 호출된다.

```
class Person {
    String name;
    int age;

    // 생성자를 정의하여 초기화
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}

// 생성자를 사용하여 객체 초기화
Person person = new Person("John", 30);
```

2. 초기화 블록을 사용한 초기화

> 생성자 외에 객체 초기화 코드를 추가하고 싶을 때 사용한다.

```
class Person {
    String name;
    int age;

    // 인스턴스 초기화 블록
    {
        name = "John";
        age = 30;
    }
}

// 초기화 블록을 사용하여 객체 초기화
Person person = new Person();
```

3. 메서드를 사용한 초기화

> 초기화 로직이 복잡하거나 초기화 값을 동적으로 설정해야 할 때 유용하다.

```
class Person {
    String name;
    int age;

    // 초기화 메서드
    public void initialize(String name, int age) {
        this.name = name;
        this.age = age;
    }
}

// 메서드를 사용하여 객체 초기화
```

```
Person person = new Person();
person.initialize("John", 30);
```

- 필드 초기화

> 객체마다 동일한 값이라면 초기값을 대입하고, 객체마다 다른 값이라면 생성자에서 필드를 초기화한다.

```
// 다른 이름 사용
public class Person {
    // 필드 선언
    String nation = "kr";
    String name;
    int age;

    // 생성자 선언
    public Person(String n, int g){
        name = na;
        age = g;
    }
}
```

```
// 동일한 이름 사용
public class Person {
    // 필드 선언
    String nation = "kr";
    String name;
    int age;

    // 생성자 선언
    public Person(String name, int age){
        this.name = name;
        this.age = age;
    }
}
```

초기화는 동일한 이름을 사용하는 것이 일반적이다. 매개변수명이 필드명과 동일한 경우 필드 구분을 위해 this 키워드를 앞에 붙여서 사용한다. this는 현재 객체를 말하며, this.name은 객체의 필드 이름을 말한다.

- 생성자 오버로딩

> 생성자는 여러개의 매개변수를 사용하기 위해 오버로딩할 수 있다. 생성자 오버로딩은 매개변수의 타입, 개수, 순서가 다르게 동일한 생성자를 호출하는 것을 말한다. 단, 이름만 바꾸는 것은 오버로딩 되지 않는다.

```
public class Person {
    String name;
    String age;
```

```
    public Person(){}

    public Perseon(Map map){
        // ...
    }

    public Perseon(String name, String age){
        // ...
    }
}
```

- 다른 생성자 호출

1. this(매개값...)은 생성자의 첫 줄에 작성되며 다른 생성자를 호출한다.
2. this(매개값...)은 호출되는 생성자 실행이 끝나면 다시 원래 생성자로 돌아와 다음 실행문을 실행한다.

```
public Person(){
    this(Map map);
    // 추가 실행문
}
```

생성자 오버로딩이 많아질 경우 중복된 코드가 발생된다. 이 경우 공통 코드를 한 생성자에 작성하고 나머지는 this() 를 사용하여 공통 코드를 가지고 있는 생성자를 호출하는 방식을 사용한다.

```
// 중복 코드 발생
Person(Map map){
    this.map = map;
}

Person(Map map, String name){
    this.map = map;
    this.name = name;
}

Person(Map map, String name, String age){
    this.map = map;
    this.name = name;
    this.age = age;
}

// 공통 코드를 가지고 있는 생성자 호출
Person(Map map){
    this(map, "", "");
}

Person(Map map, String name){
    this(map, name, "");
}
```

```
Person(Map map, String name, String age){
    this.map = map;
    this.name = name;
    this.age = age;
}
```

메소드 선언과 호출

> 메소드는 생성자와 다른 메소드 내부에서 호출될 수 있고, 객체 외부에서도 호출될 수 있다.

```
타입 변수 = 메소드();
```

> 메소드는 함수 또는 서브루틴이라고도 부른다. 메소드 선언은 메소드의 시그니처(이름, 매개변수 목록, 반환타입)와 메소드의 동작을 정의하는 코드 블록으로 구성된다. 메소드를 선언하면 다른 곳에서 메소드를 호출하여 사용할 수 있다. 메소드 선언은 재사용성 촉진하고 코드를 구조화하고 유지보수하기 용이하다.

- 리턴 타입 메소드 실행 후 호출한 곳으로 전달하는 결과값의 타입 리턴값이 없다면 void로 작성한다.

```
void run(){}
```

- 메소드명 첫 문자를 소문자로 시작하하고 캐멀 스타일을 사용하는 것이 관례이다.

```
String getName(){}
```

- 매개변수 메소드 호출시 전달받으며, 전달할 매개값이 없다면 매개변수는 생략할 수 있다.
- 가변길이 매개변수 메소드 매개변수는 가변길이를 사용할 수 있다. 매개값들은 자동으로 배열 항목으로 변환되어 메소드에서 사용되기 때문에 호출 시 직접 배열을 매개값으로 전달해도 된다.

```
// 가변길이 매개변수 선언
int sum(int ... values) {}

int result = sum(1, 2);
int result = sum(1, 2, 3, 4, 5);

int[] values = { 1, 2, 3 };
int result = sum(values);
int result = sum(new int[] {1, 2, 3});
```

- 메소드 오버로딩

> 매개변수의 타입, 개수, 순서가 다른 같은 이름의 메소드를 여러개 선언하는 것을 말한다. 대표적인 메소드 오버로딩으로 System.out.println() 메소드가 있다. 호출할 때 주어진 매개값의 타입에 따라서 오버로딩된 println() 메소드 중

> 하나를 실행한다.

```
void println(){}
// return type도 변경할 수 있다.
String println(String x){}
```

인스턴스 멤버