OOP

객체지향 프로그래밍이란?

객체지향의 개념은 **"실제 세계는 객체로 이루어져 있으며, 발생하는 모든 사건들은 객체간의 상호작용이다."**이다.
객체지향 프로그래밍은 실제 객체의 속성과 기능을 분석한 다음, 데이터와 함수로 정의함으로써 가상 세계를 구현하는 기법이다.

객체지향언어 특징

1. 코드의 재사용성이 높다.
2. 코드의 관리가 용이하다.
3. 신뢰성이 높은 프로그래밍을 가능하게 한다.

클래스와 객체

클래스는 3가지로 정의할 수 있는데 첫 번째로 **객체를 만들기 위한 설계도**이다.
여기서 객체는 실제 존재하는 것으로 Java에서는 **new 연산자**를 통해 객체를 생성한다.
클래스를 통해 만들어진 객체를 인스턴스라고 한다.
객체는 속성과 기능으로 이루어져 있으며 클래스에서 속성은 변수로 기능은 메서드로 구현한다.

두 번째로 **데이터 + 함수**로 이루어진 자료형이다. 데이터는 속성을 함수는 기능을 정의한다.
클래스의 대표적인 예로 문자열이 있다. C에서는 문자의 배열로 다루지만 왜 Java에서는 클래스로 정의했을까? 그 이유는 문자열과 문자열을 다루는데 필요한 함수들을 함께 묶기 위해서이다. 이렇게 하면 작업이 간단하고 명료해진다.

세 번째로 **사용자 정의 타입**이다. 예를 들어 아래와 같이 시간을 표현할 때 시, 분, 초를 하나로 묶어 사용할 수 있다.

```
class Time {
    int hour; // 시
    int minute; // 분
    int second; // 초
}
```

클래스의 구성

클래스는 필드와 생성자, 메서드로 구성되어있다.

```
class MyClass {
    // field: 클래스에 포함된 변수
    String name;

    // Constructor: 객체의 생성과 동시에 인스턴스변수를 원하는 값으로 초기화할 수 있는 특수한 메서드
```

```
    MyClass() {}

    // Method: 어떠한 특정 작업을 수행하기위한 명령문의 집합
    void print() { /* 출력 */ }
}
```

JVM의 메모리 구조

JVM의 메모리에는 3가지 영역인 메서드 영역, 호출 스택, 힙 영역이 있다.
메서드 영역은 클래스 정보와 클래스변수를 저장하는 영역이다.
호출 스택은 메서드의 작업 공간으로 호출되면 메모리 공간을 할당받고 종료되면 반환한다.
메서드는 1개만 실행하고 나머지는 대기중인데 호출 스택 제일 위에 있는 메서드가 현재 실행중인 메서드이며, 아래에 있는 메서드가 바로 위의 메서드를 호출한 메서드이다.
힙 영역은 인스턴스가 생성되는 공간이다.
new 연산자에 의해 생성되는 배열과 객체는 모두 여기에 생성이 된다.

변수의 종류

변수는 **선언 위치**에 따라 결정된다.

변수의 종류	선언 위치	생성 시기
클래스변수	클래스 영역	클래스가 메모리에 올라갈 때
인스턴스변수	클래스 영역	인스턴스 생성 시
지역변수	메서드 영역	변수 선언문 수행 시

클래스변수와 인스턴스변수를 멤버변수라고 하고 멤버변수를 제외한 나머지 변수들은 모두 지역변수이다.
멤버변수 중에서 **static**이 붙은 것은 클래스변수, 붙지 않은 것은 인스턴스변수이다.
여기서 주의할 점은 **클래스 영역에는 선언문만 가능하다.**

인스턴스변수는 먼저 인스턴스를 생성해야 사용할 수 있으며, 인스턴스마다 서로 다른 값을 가질 수 있다.
클래스변수는 모든 인스턴스가 공유한다.
그러므로 **공통적인 값을 유지해야하는 속성인 경우 클래스변수로, 서로 다른 값을 가져야한다면 인스턴스변수로 선언한다.**
인스턴스변수는 참조변수.인스턴스변수로 사용하고, 클래스변수는 클래스이름.클래스변수로 사용하자.

변수의 초기화

변수의 초기화는 변수를 선언하고 처음으로 값을 저장하는 것을 의미한다.
변수의 종류에 따라 초기화 방법이 다른데, 멤버변수는 초기화를 생략 가능하고, 지역변수는 반드시 직접 초기화를 해줘야 한다.
멤버변수 초기화를 생략하면 논리형은 false, 숫자형은 0, 문자형은 공백, 참조형은 null로 초기화된다.

멤버변수 초기화의 종류로는 3가지가 있는데 자동 초기화, 간단 초기화(명시적 초기화), 복잡 초기화가 있다.
첫 번째로 자동 초기화는 변수만 선언하면 된다. 위와 같이 논리형은 false, 숫자형은 0, 문자형은 공백, 참조형은 null로 초기화된다.
두 번째로 간단 초기화는 대입 연산자(=)를 이용해 초기화한다.

세 번째로 복잡 초기화는 클래스변수는 클래스 초기화블럭을 사용하고, 인스턴스변수는 생성자와 인스턴스 초기화블럭을 사용한다.

멤버변수 초기화 시점으로는 클래스변수는 클래스가 처음 로딩될 때 단 한번만 하고 인스턴스변수는 인스턴스가 생성될 때마다 초기화된다.
클래스변수가 초기화된 후 인스턴스변수가 초기화되고 자동 초기화, 간단 초기화, 복잡 초기화순으로 초기화된다.

1. 클래스변수 기본값
2. 클래스변수 명시적 초기화
3. 클래스변수 클래스 초기화 블럭
4. 인스턴스변수 기본값
5. 인스턴스변수 명시적 초기화
6. 인스턴스변수 인스턴스 초기화 블럭
7. 인스턴스변수 생성자

메서드

클래스영역에만 정의가능한 함수이다.
반복적으로 수행되는 여러 문장들을 메서드로 작성하고 **하나의 메서드는 하나의 기능만 수행하도록 작성한다.**

메서드를 사용하는 이유는 세 가지다.

1. 높은 재사용성
2. 코드 중복 제거
3. 프로그램의 구조화

main 메서드는 프로그램의 전체 흐름이 한눈에 들어올 정도로 단순하게 구조화하는 것이 좋다.
메서드에는 반드시 return문이 있어야 한다.

메서드는 참조변수.메서드이름();으로 호출한다.
메서드를 호출할 때 괄호()안에 지정해준 값들을 인자 또는 인수라고 한다.
인자의 타입은 매개변수의 타입과 일치하거나 자동 형변환이 가능한 것이어야 한다.

매개변수의 종류

매개변수는 메서드를 정의할 때 메서드의 괄호()안에 지정해준 값이다.

2가지의 종류가 있는데 기본형 매개변수와 참조형 매개변수가 있다.
기본형 매개변수는 오로지 변수의 값을 읽기만 할 수 있다.
참조형 매개변수는 변수의 값을 읽고 변경 가능하다.
왜 그럴까? 이유는 Java는 메서드를 호출할 때 인자를 메서드의 매개변수에 복사해서 넘겨주기 때문이다.

따라서 값을 변경해야한다면 참조형 매개변수를 사용하자.

메서드의 종류

인스턴스변수의 사용여부에 따라 나뉜다. 인스턴스변수를 사용하면 인스턴스메서드 그렇지 않다면 클래스메서드이다.
변수와 같이 클래스메서드는 인스턴스메서드에 static 키워드를 붙인다.

인스턴스메서드는 인스턴스 생성 후에 참조변수.메서드이름();으로 호출하고, 클래스메서드는 객체 생성없이 클래스이름.메서드이름();으로 호출한다.

클래스 메서드 내에서는 인스턴스멤버를 사용할 수 없다.
왜냐하면 인스턴스멤버는 인스턴스를 생성해야 사용할 수 있는데 클래스메서드를 호출할 때 인스턴스가 생성이 되었다고 보장을 할 수 없기때문이다.

따라서 메서드내에서 인스턴스멤버를 사용하지 않는다면 클래스메서드를 고려하자.

메서드 오버로딩

한 클래스 내에 같은 이름의 메서드를 여러 개 정의하는 것을 의미한다.
메서드의 이름을 절약할 수 있고 메서드의 이름을 외우기 쉽다는 장점이 있다.

오버로딩은 다음과 같은 조건을 만족해야한다.

1. 메서드 이름이 같아야 한다.
2. 매개변수의 개수 또는 타입이 달라야 한다.

가능하면 가변인자를 사용한 메서드는 오버로딩하지 않는 것이 좋다.

생성자

인스턴스 초기화 메서드이다. 메서드라서 오버로딩이 가능하다.

생성자의 조건은 다음과 같다.

1. 생성자의 이름은 클래스의 이름과 같아야 한다.
2. 생성자는 리턴 값이 없다.

모든 클래스는 반드시 하나 이상의 생성자가 있어야 한다.
클래스에 생성자가 하나라도 없으면 컴파일러가 기본 생성자를 추가해준다.

생성자에서 다른 생성자를 호출 가능한데 두 조건을 만족시켜야 한다.

1. 생성자의 이름으로 클래스이름 대신 this를 사용한다.
2. 한 생성자에서 다른 생성자를 호출할 때는 반드시 첫 줄에서만 호출이 가능하다.

왜 반드시 첫 줄에서만 호출해야 될까? 다른 생성자를 호출하기 이전의 초기화 작업이 무의미해질 수 있기 때문이다.

생성자 this()는 같은 클래스의 다른 생성자를 호출할 때 사용한다.
참조변수 this는 인스턴스 자신을 가리키는 참조변수로 인스턴스변수와 지역변수가 이름이 같을 때 구별하기 위해서 사용한다.
인스턴스 자신을 가리키는 참조변수이므로 클래스메서드에선 사용할 수 없다.

생성자를 이용해서 인스턴스를 복사할 수도 있다.

```
MyClass(MyClass c) {
    this(c.name);
}
```

```
MyClass(String name) {
    this.name = name;
}
```

클래스간의 관계

객체지향적으로 프로그래밍을 짜려면 가능한 많은 관계를 맺어줘야한다.

클래스간의 관계는 2가지가 있는데 상속관계와 포함관계이다.

상속관계는 두 클래스를 조상과 자손으로 관계를 맺어주는 것이며 자손은 조상의 모든 멤버만 상속받고 생성자와 초기화 블럭은 상속되지 않는다.
포함관계는 클래스의 멤버로 참조변수를 선언하는 것이다.

Java에서는 단일 상속만 지원하기 때문에 비중이 높은 클래스 하나만 상속하고 나머지는 포함관계로 관계를 맺어준다.
그러므로 어떤 클래스를 상속으로 맺어줄 지 골라야한다.

Tv로 예를 들면 smart Tv는 TV이다. 그리고 smart Tv는 전원 버튼을 가지고 있다.
여기서 ~는 ~이다와 같은 관계를 상속관계로, ~는 ~를 가지고 있다는 포함관계로 맺어준다.

Object 클래스는 모든 클래스의 조상으로 모든 클래스는 11개의 메서드를 상속받는다.

메서드 오버라이딩

조상 클래스로부터 상속받은 메서드의 내용을 변경하는 것을 의미한다.

오버라이딩이 성립하기 위해서는 다음 조건을 만족해야 한다.

1. 메서드의 선언부가 같아야 한다.
2. 접근 제어자는 조상 클래스의 메서드보다 좁은 범위로 변경할 수 없다.
3. 조상 클래스의 메서드보다 많은 수의 예외를 선언할 수 없다.
4. 인스턴스메서드를 static메서드로 또는 그 반대로 변경할 수 없다.

조상 클래스에 접근

super()는 조상의 생성자를 호출할 때 사용한다.
Object클래스를 제외한 모든 클래스의 생성자 첫 줄에는 생성자를 호출해야 한다.
같은 클래스의 다른 생성자나 조상의 생성자를 호출한다.
같은 클래스는 this()로 호출하고 조상의 생성자는 super()로 호출한다.
생성자를 호출하면 모든 조상 클래스의 생성자가 순서대로 호출된다.

super는 인스턴스 자신을 가리키는 참조변수로서 this와의 차이는 super는 조상 클래스를 가리키는 참조변수고 this는 클래스 자신을 가리키는 참조변수이다.
this와 같이 조상의 멤버와 자신의 멤버를 구별할 때 사용한다.

패키지

서로 관련된 클래스와 인터페이스들의 묶음이다.
모든 클래스들은 반드시 하나의 패키지에 속해야 한다.

패키지는 소스파일에 첫 번째 문장으로 단 한번 선언한다.
프로젝트를 할 때 미리 패키지를 구성하여 적용하도록 한다.

클래스패스는 컴파일러나 JVM 등이 클래스의 위치를 찾는데 사용되는 경로이다.
구분자는 ;으로 하고 클래스패스를 지정할 때 현재 폴더(.)도 함께 추가를 한다.

import문은 사용할 클래스가 속한 패키지를 지정하는 데 사용한다.
클래스를 사용할 때 패키지명을 생략할 수 있다는 장점이 있다.
패키지문과 클래스 선언의 사이에 선언한다.
컴파일시에 처리하므로 프로그램 성능에 아무런 영향을 안 미친다.

static import문은 static멤버를 사용할 때 클래스 이름을 생략가능하게 해준다.

제어자

제어자는 접근 제어자와 그 외 제어자로 나뉜다.
접근 제어자는 단 하나만 사용할 수 있고, 그 외 제어자는 여러 개의 제어자를 사용할 수 있다.

제어자의 종류는 다음과 같다.

종류	제어자
접근 제어자	public, (default), protected, private
그 외 제어자	static, final, abstract, native, transient, synchronized, volatile, strictfp

접근 제어자의 접근 범위는 다음과 같다.

제어자	같은 클래스	같은 패키지	자손 클래스	전체
public	O	O	O	O
protected	O	O	O	
(default)	O	O		
private	O			

접근 제어자를 왜 사용할까? 그 이유는 캡슐화를 하기 위해서이다.
캡슐화는 외부로부터 데이터를 보호하고 내부를 감추는 장점이 있다.
그러므로 접근 범위를 최소화하도록 노력해야한다.

생성자에도 접근 제어자를 사용할 수 있는데 인스턴스의 생성을 제한할 때 사용한다.
대신 인스턴스를 생성해서 반환해주는 public메서드를 제공해야 하는데 이 메서드는 public인 동시에 static이여야 한다.

타입마다 사용 가능한 제어자가 다르다.

타입	제어자
클래스	public, (default), final, abstract
메서드	모든 접근 제어자, final, abstract, static

타입	제어자
멤버변수	모든 접근 제어자, final, static
지역변수	final

제어자의 조합은 다음 조건에 만족해야 한다.

1. 메서드에 static과 abstract를 함께 사용할 수 없다.
2. 클래스의 abstract와 final을 동시에 사용할 수 없다.
3. abstract메서드의 접근 제어자가 private이면 안 된다.
4. 메서드에 private와 final을 같이 사용할 필요가 없다.

다형성

Java에서는 한 타입의 참조변수로 여러 타입의 객체를 참조할 수 있도록 해서 다형성을 구현한다.
다시 말하자면, 조상클래스 타입의 참조변수로 자손클래스의 인스턴스를 참조할 수 있도록 하였다.
반대로 자손타입의 참조변수로 조상타입의 인스턴스를 참조할 수 없다.

참조변수의 형변환을 이용해서 사용할 수 있는 멤버의 갯수를 조절하여 다형성을 구현하는데 서로 상속관계에 있는 타입간의 형변환만 가능하다.
형변환할 때 주의점은 실제 인스턴스가 무엇인지에 따라 멤버의 개수가 달라진다.
그러므로 실제 인스턴스의 멤버 개수보다 많은 클래스로 형변환을 하면 안 된다.

이걸 방지하기 위해서 형변환을 수행하기 전에 instanceof 연산자를 사용해서 형변환 가능한지 확인 후에 형변환을 해야한다.

멤버 중 인스턴스메서드는 참조변수의 타입에 관계없이 항상 실제 인스턴스의 메서드가 호출되고, 변수의 경우는 참조변수의 타입에 따라 달라진다.
하지만 static메서드는 참조변수 타입에 따라 달라지므로 static메서드는 클래스이름.메서드();로 호출해야 한다.

다형성을 쓰는 이유 중 가장 큰 이유는 매개변수의 다형성이다.
매개변수의 다형성의 장점은 다형적 매개변수를 가질 수 있고, 하나의 배열로 여러 종류 객체를 다룰 수 있다.
참조형 매개변수는 메서드 호출 시, 자신과 같은 타입 또는 자손타입의 인스턴스를 넘겨줄 수 있다.

추상클래스

추상메서드를 포함하고 있는 클래스이며 미완성 설계도이다.
추상메서드는 메서드에 abstract 키워드를 붙이며 선언부만 있고 구현부가 없는 메서드다.
완성된 설계도가 아니므로 인스턴스를 생성할 수 없으며 대부분 다른 클래스를 작성하는 데 도움을 줄 목적으로 작성한다.
추상클래스로부터 상속받는 자손클래스는 오버라이딩을 통해 조상인 추상클래스의 추상메서드를 모두 구현해 주어야 하거나 하나라도 구현하지 않는다면 추상클래스로 지정해 주어야 한다.
클래스들간의 공통점을 찾아내서 추상메서드로 작성해주는데 자손클래스에서 반드시 구현하도록 강요하기 위해서 추상메서드로 선언한다.

추상메서드

선언부만 있고 구현부가 없는 메서드다.
꼭 필요하지만 자손마다 다르게 구현될 것으로 예상되는 경우에 사용된다.

인터페이스

추상메서드의 집합이며 실제 구현된 것이 전혀 없는 기본 설계도이다.
그렇다면 왜 사용할까? 표준을 제시하기 위해서 사용을 한다.
인스턴스를 생성할 수 없고 추상메서드와 상수, 디폴트 메서드, static메소드를 멤버로 가질 수 있다.
추상클래스는 인스턴스변수를 가질 수 있고 인터페이스는 인스턴스변수를 가질 수 없다는 차이가 있다.

인터페이스의 멤버들은 다음과 같은 제약사항이 있다.

1. 모든 멤버변수는 public static final이어야 하며, 생략가능하다.
2. 모든 메서드는 public abstract이어야 하며, 생략가능하다.

클래스와 달리 다중상속이 가능하며 Objext클래스와 같은 최고 조상이 없다.
인터페이스를 구현할 때 주의할 점은 멤버들의 접근 제어자를 반드시 public으로 해야한다.

인터페이스도 매개변수로 사용할 수 있는데 메서드 호출 시 해당 인터페이스를 구현한 클래스의 인스턴스를 매개변수로 제공해야한다는 것이다.
이와 같이 리턴타입으로 인터페이스를 지정하면 해당 인터페이스를 구현한 클래스의 인스턴스를 반환해야한다는 것이다.

인터페이스의 장점은 이와 같다.

1. 개발시간을 단축시킬 수 있다.
2. 표준화가 가능하다.
3. 서로 관계없는 클래스들에게 관계를 맺어줄 수 있다.
4. 독립적인 프로그래밍이 가능하다.

내부 클래스

클래스 안에 선언된 클래스이다.

내부클래스의 장점으로는 외부 클래스의 멤버들을 쉽게 접근할 수 있고, 캡슐화를 할 수 있다. 내부클래스의 유효범위와 성질은 변수와 유사하다.
static클래스만 static멤버를 가질 수 있으며 외부클래스의 지역변수는 상수만 접근이 가능하다.

익명 클래스

정의와 생성을 동시에 하는 오직 하나의 객체만 생성할 수 있는 일회용 클래스이다.
이름이 없기 때문에 생성자를 가질 수 없다.