ICTO 초기전력화 교육 과제 1일차

객체 지향 문법 정리

1. 객체 지향 문법

객체란?

객체란 물리적으로 존재하거나 추상적으로 생각할 수 있는 것 중에서 자신의 속성을 가지고 있고 다른 것과 식별 가능한 것.

객체의 상호작용

하나의 객체가 다른 객체와 상호작용을 할 때의 수단은 메소드이다. 즉 객체가 다른 객체의 기능을 이용하는 것이 바로 메소드이다. (객체명.메소드명) 의 형식으로 호출이 가능하고, 결과를 리턴함으로 상호작용한다. 메소드를 통한 호출 시 매개변수를 입력할 수도 있다.

객체 간의 관계

객체끼리는 사용관계, 집합관계, 상속관계를 가질 수 있다. 그 중, 상속관계는 부모 객체를 기반으로 자식 객체를 생성하는 관계이다.

1. 객체 지향 프로그래밍의 특징

캡슐화

객체의 필드, 메소드를 하나로 묶고, 실제 구현 내용을 감추는 것을 말한다. 외부 객체는 객체 내부의 구조를 알지 못하며 객체가 노출해서 제공하는 필드와 메소드만 이용할 수 있다.

상속

부모가 가지고 있는 재산을 자식에게 물려주는 것을 말한다. 즉 부모 객체가 가지고 있는 필드와 메소드를 자식 객체가 사용할 수 있도록 해주는 것이다. 이를 통해 하위 객체를 쉽고 빨리 설계할 수 있고, 반복된 코드의 중복을 줄여준다.

다형성(Polymorphism)

같은 타입이지만 실행 결과가 다양한 객체를 이용할 수 있는 성질을 말한다. 코드 측면에서 보면 다형성은 하나의 타입에 여러 객체를 대입함으로써 다양한 기능을 이용할 수 있도록 해준다.

1. 생성자

기본 생성자

모든 클래스는 생성자가 반드시 존재하며, 하나 이상을 가질 수 있다. 클래스 내부에 생성자 선언을 생략했다면 컴파일러가 기본 생성자(Default 생성자)를 자동 추가시킨다. 기본 생성자를 통해 객체를 생성시킬 수 있다. Car myCar = new Car();

생성자 선언

생성자는 메소드와 비슷한 형태를 지니고 있으나 리턴 타입이 없고 클래스 이름과 동일하다. 생성자 블록 내부에는 객체 초기화 코드가 작성되는데, 일반적으로 필드에 초기값을 저장하거나 메소드를 호출하여 객체 사용 전에 필요한 준비를 한다.

생성자 오버로딩

매개 변수를 달리하는 생성자를 여러 개 선언하는 것을 말한다. 생성자 오버로딩 시 주의할 점은 매개 변수의 타입과 개수 그리고 선언된 순서가 똑같을 경우 매겨 변수 이름만 바꾸면 에러가 난다는 것이다.

다른 생성자 호출(this)

생성자 오버로딩이 많아질 경우 중복된 코드가 발생할 수 있는데, this를 사용해서 중복을 최소화할 수 있다. 한 생성자에 변수 하나만 추가할 경우 추가 할 생성자의 블록 내부에 this(매개변수들)의 형태로 기존 생성자를 호출한 후, 추가할 내용만 적으면 된다.

1. VO, DAO

VO

변수 및 getter, setter, toString 등의 메소드를 포함하는 클래스를 말한다. 이 클래스를 생성함으로써 매 클래스마다 변수를 생성하고 접근하는 코드를 줄일 수 있다.

DAO

VO에 선언된 변수들에 접근 해 값을 변화시켜주는 클래스이다. 사용자(main메소드)는 DAO를 호출함으로써 VO를 조작하기 때문에 VO를 보지 않고도 변수들을 조작할 수 있다.

1. Interface

단지 부모 역할만 하는 껍데기 클래스이다. 다형성의 사용을 위해 자식들이 공통적으로 호출할 메소드의 이름만 정의해 놓는다.

1. 예제코드

// interface 클래스

**public** **interface** TV {

**public** **void** powerOn();

**public** **void** powerOff();

**public** **void** volumeUp();

**public** **void** volumeDown();

}

// 삼성TV 클래스 – TV 인터페이스를 구현하므로 네 개의 메소드를 가지며 메소드명이 동일함

**public** **class** SamsungTV **implements** TV{

**public** **void** powerOn(){

System.*out*.println("SamsungTV => 전원 켜기");

}

**public** **void** powerOff(){

System.*out*.println("SamsungTV => 전원 끄기");

}

**public** **void** volumeUp(){

System.*out*.println("SamsungTV => 소리 높이기");

}

**public** **void** volumeDown(){

System.*out*.println("SamsungTV => 소리 낮추기");

}

}

// 엘지TV 클래스 – TV 인터페이스를 구현하므로 네 개의 메소드를 가지며 메소드명이 동일함

**public** **class** LgTV **implements** TV{

**public** **void** powerOn(){

System.*out*.println("LgTV => 전원 켜기");

}

**public** **void** powerOff(){

System.*out*.println("LgTV => 전원 끄기");

}

**public** **void** volumeUp(){

System.*out*.println("LgTV => 소리 높이기");

}

**public** **void** volumeDown(){

System.*out*.println("LgTV => 소리 낮추기");

}

}

// 팩토리 클래스 – 클래스 내용이 변해도 해당 클래스만 수정하면 되도록 하는 클래스

**public** **class** tvFactory {

//어떤 객체를 사용할 지를 결정

**public** Object getBean(String tv){

Object result = **new** SamsungTV();

**if**(tv.equals("samsung")){

**return** **new** SamsungTV();

}

**else** **if**(tv.equals("lg")){

**return** **new** LgTV();

}

**return** result;

}

}

// 사용자 클래스 – 팩토리 클래스를 호출해 객체를 생성한 후 생성된 객체에 접근한다

**public** **class** TVUser {

**public** **static** **void** main(String[] args) {

tvFactory factory = **new** tvFactory();

TV tv = (TV)factory.getBean("samsung");

tv.powerOn();

tv.volumeUp();

tv.volumeDown();

tv.powerOff();

}

}