**16회차(1)**

< 다형성 >

- 메소드에 대한 다형성

- 오버로딩(Overloading)

- 오버라이딩(Overriding)

\*이름이 매우 유사하기 때문에 헷갈리지 말아야한다

1) 오버로딩

- 상속과는 무관한 개념

- 하나의 메소드가 여러가지 형태를 가질 수 있는 것

- 오버로딩의 조건 => 두 조건 다 만족해야 함!

1. 오버로딩하는 메소드의 이름은 반드시 동일

2. 오버로딩하는 메소드의 매개변수는 반드시 달라야 한다

- 매개변수의 개수와 타입 둘 중에 하나는 반드시 달라야 한다.

ex) 예를 들면

- 덧셈을 하는 메소드가 있는데, 반드시 매개변수가 정수이거나 2개일 필요는 없다

- 두 수의 합을 구하는 메소드와 세 수의 합을 구하는 메소드가 필요하다면?

1-1) 생성자 오버로딩

- 객체를 생성할 때, 다양한 방법으로 객체를 생성할 수 있다.

2) 오버라이딩

- 메소드를 재정의한다

- 부모 클래스로부터 물려받은 메소드를 그대로 사용하지 않고, 자식 클래스에서 재정의해서 사용

- 메소드 덮어쓰기(?)로 이해

- 오버라이딩의 조건

1. 반드시 상속관계에서만 성립

2. 재정의하려는 메소드와 물려받은 메소드의 이름이 반드시 동일해야 한다

3. 리턴 타입, 매개변수의 개수, 타입이 모두 일치해야 한다.

**16회차(2)**

package overloading;

public class Overloading {

// 오버로딩이 아닌 경우: 하나의 메소드에서 두 수 또는 세 수의 합을 모두 처리할 수는 없다.

// 하나의 메소드에서 매개변수의 개수를 가변적으로 할 수 없다. (가변인자)

// 두 수의 합을 구하는 메소드

int two\_add(int a, int b) {

return a + b;

}

// 세 수의 합을 구하는 메소드

int three\_add(int a, int b, int c) {

return a + b + c;

}

// 오버로딩 : 메소드명 동일하게, 매개변수 개수나 이름 둘 중 하나 다르게!

// 이렇게 하면(오버로딩) 메소드를 사용할 때 구분할 필요가 없이 편하게 사용할 수 있다.

int add(int a, int b) {

return a + b;

}

int add(int a, int b, int c) {

return a + b + c;

}

}

package overloading;

public class Main {

public static void main(String[] args) {

// 오버로딩이 아니라면 매개변수에 따라서 매번 메소드를 다르게 호출

Overloading obj = new Overloading(); // 객체 생성

int ret = obj.two\_add(10, 20);

System.out.println(ret);

ret = obj.three\_add(10, 20, 30);

System.out.println(ret);

// 오버로딩

// 오버로딩이 되면 같은 메소드의 이름으로 다양하게 사용할 수 있다.

Overloading obj2 = new Overloading(); // 객체 생성

int ret2 = obj2.add(10, 20);

System.out.println(ret2);

ret2 = obj2.add(10, 20, 30);

System.out.println(ret2);

}

}

**16회차(3)**

package overloading;

public class Overloading1 {

private String name;

private String telecom;

private int price;

// 기본 생성자 : 아무것도 하지 않음.

Overloading1(){}

// 생성자 오버로딩 : 이름 동일, 매개변수 다름

Overloading1(String name, String telecom, int price) {

this.name = name;

this.telecom = telecom;

this.price = price;

}

String get\_name() {

return this.name;

}

void set\_name(String name) {

this.name = name;

}

void info() {

// 객체의 속성을 전부 출력할 수 있도록

System.out.println("이름 : " + this.name);

System.out.println("통신사 : " + this.telecom);

System.out.println("가격 : " + this.price);

}

}

package overloading;

public class Main1 {

public static void main(String[] args) {

// 객체를 생성할 때 생성자에서 정의된 매개변수를 반드시 입력

// 기본 생성자가 실행됨

Overloading1 obj = new Overloading1();

obj.info();

// 생성자 오버로딩을 이용하면 다양한 형태로 객체 생성 가능

// 오버로딩된 생성자, 메소드 호출

Overloading1 obj1 = new Overloading1("Iphone", "SKT", 100000);

obj1.info();

}

}

**16회차(4)**

package overriding;

public class Parent {

// 멤버 메소드

public void method() {

// 부모 클래스임을 알 수 있는 출력문

System.out.println("부모 클래스");

}

}

package overriding;

// Child는 Parent에게서 상속받는다

public class Child extends Parent {

// 오버라이딩을 통해서 method()를 재정의

// 이때 Child 클래스는 정의하지 않는다

}

package overriding;

public class Main {

public static void main(String[] args) {

Child c = new Child();

c.method();

}

}

**16회차(5)**

package overriding1;

public class Mobile {

private String name;

private String telecom;

private int price;

// 부모 클래스에 기본 생성자를 오버로딩 해주면 문제가 해결된다

Mobile(){};

Mobile(String name, String telecom, int price) {

this.name = name;

// 이렇게 작성하는 것도 가능하다

// this.set\_name(name);

this.telecom = telecom;

// set\_price을 총해서 가격 설정 <- 작성 전에는 처음 가격이 40만원 아래여도 그대로 출력된다

// this.price = price;

this.set\_price(price);

}

// getter

// public : 접근 상관없이 언제든지 메소드를 통해 값에 접근할 수 있다

public String get\_name() {

return this.name;

}

public String get\_telecom() {

return this.telecom;

}

public int get\_price() {

return this.price;

}

// setter

// 같은 클래스 내에서는 호출 가능

// < 휴대폰의 이름은 한 번 설정하면 절대 변경 불가 >

// 1. set\_name을 지운다

// 2. set\_name 앞에 private을 붙인다

private void set\_name(String name) {

this.name = name;

}

public void set\_telecom(String telecom) {

this.telecom = telecom;

}

public void set\_price(int price) {

if (price < 400000) {

System.out.println("휴대폰 최소가격은 400,000");

this.price = 400000; // 지역변수!

return;

}

this.price = price;

}

// member method

void info() {

System.out.println("이름 : " + this.name);

System.out.println("통신사 : " + this.telecom);

System.out.println("가격 : " + this.price);

}

}

package overriding1;

public class MobileEx extends Mobile {

private String card;

private int month;

// 이때, 생성자는 기준의 속성과 추가된 속성 모두를 전달받을 수 있도록 한다

// 부모 클래스에 기본 생성자가 없다면? 자식 클래스는 반드시 super()를 사용

public MobileEx(String name, String telecom, int price, String card, int month) {

// 부모 클래스에 private 속성은 상속되지 않는다

// System.out.println(this.name); -> error

// this.name = name; -> error

// => 부모 클래스의 생성자를 임의로 호출해준다 : super()

// super()를 통해서 부모 클래스의 생성자를 호출하는 경우

// 반드시 제일 첫 번째 줄에 작성해야 한다!!

// 그 전에 다른 코드들이 작동되어서는 안 된다.

super(name, telecom, price);

// 하지만 card와 month는 가능하다! 상속받은 것이 아니다.

this.card = card;

this.month = month;

}

// 메소드 오버라이딩을 통해서 제휴카드와 약정기간도 출력될 수 있도록 한다

void info() {

// 기존의 부모 클래스의 info()를 그대로 사용하는 것도 가능

// 이때, 부모 클래스의 info()를 따로 호출해줘야 한다 <= super()이용

super.info();

System.out.println("제휴카드 : " + this.card);

System.out.println("약정기간 : " + this.month + "개월");

}

// getter와 setter를 추가해서 수시로 설정 및 확인이 가능하도록 한다

String get\_card() {

return this.card;

}

void set\_card(String card) {

this.card = card;

}

int get\_month() {

return this.month;

}

void set\_month(int month) {

this.month = month;

}

}

package overriding1;

// 기존의 Mobile 클래스를 이용해서 상속을 실습 // Mobile 클래스를 통해서 휴대전화를 관리

// - 시간이 지나서 항목이 추가되었다고 가정

// - Mobile 클래스를 수정하지 않고, 상속을 통해서 추가된 항목을 저장할 수 있도록 만들어보자

// Mobile 클래스 상속받는 MobileEx

// [추가된 항목] : String 제휴카드(수시로 설정, 확인 가능), int 약정기간(월 단위)

public class Main {

public static void main(String[] args) {

MobileEx obj = new MobileEx("iphone", "SKT", 1000000, "삼성카드", 24);

// 부모 클래스의 private 속성(name, telecom, price)은 상속되지 않는다.

// 그렇다면 상속되지 않은 부모 클래스의 속성은 어떻게 확인?

// getter를 통해서 확인

System.out.println(obj.get\_name());

// info()를 통해서 확인하는 것도 가능 but 새로 추가된 내용은 나오지 않는다

// info()는 자식에서 추가된 속성까지는 출력되지 않는다 => 오버라이딩 필요

// obj.info(); => 미완료

// => 오버라이딩 필요 : Mobile(){};

obj.info();

}

}