2021.06.01 Java

```java
import java.util.ArrayList;
import java.util.Arrays;
public class _99th_Practice {
    public static void main(String[] args) {
        int[] array_numbers = {1, 2, 3, 4, 5, 6, 7};
//...
        ArrayList<int[]> arNum = new ArrayList<int[]>(Arrays.asList(array_numbers));
        System.out.println(arNum);
```

```
_99th_Practice
"C:\Program Files\Java\jdk-16\bin\java.exe" -javaagent:C:\Users\Samuel\AppData\Local\
[[I@4c873330]

Process finished with exit code 0
```

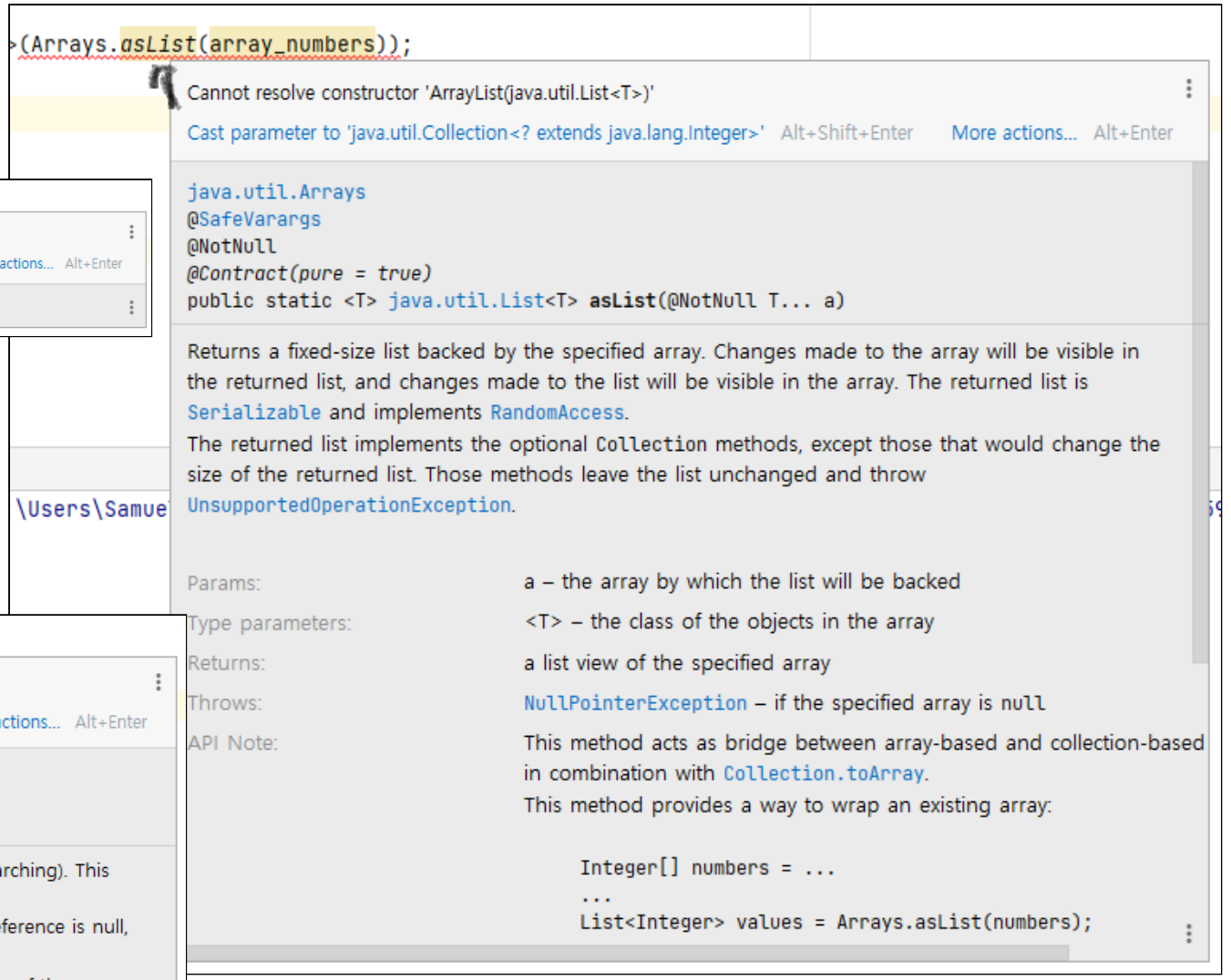
Q. arNum을 ArrayList로 변환했는데
왜 print기능이 적용이 안 되는지 궁금합니다.

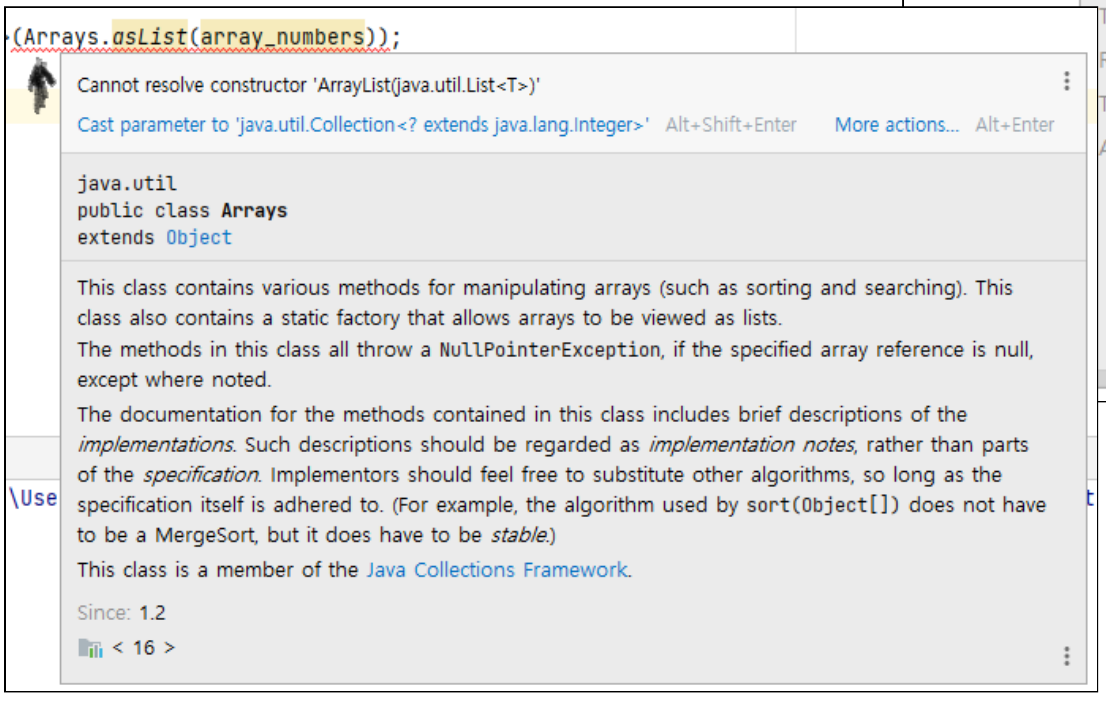
```java
class Market {
    private ArrayList<String> userBuyList;
    private ArrayList<Integer> userBuyListStock;

    private String[] marketSellList = {"선풍기", "키보드", "마우스", "모니터"};
    private int[] marketSellListPrice = {380000, 80000, 70000, 400000};

    private int myMoney;

    private Boolean continueShopping;
    Scanner scan;

    final int DEFAULT_IDX = 1;

    public Market () {
        userBuyList = new ArrayList<String>();
        userBuyListStock = new ArrayList<Integer>();
```

```java
ArrayList<Integer> arNum2 = new ArrayList<Integer>(Arrays.asList(array_numbers));
System.out.println(arNum);
```

Q. 두 개가 어떤 개념의 차이로 위의 code는 실행이 불가한지 궁금합니다.

위의 code에서 나타는 error들.





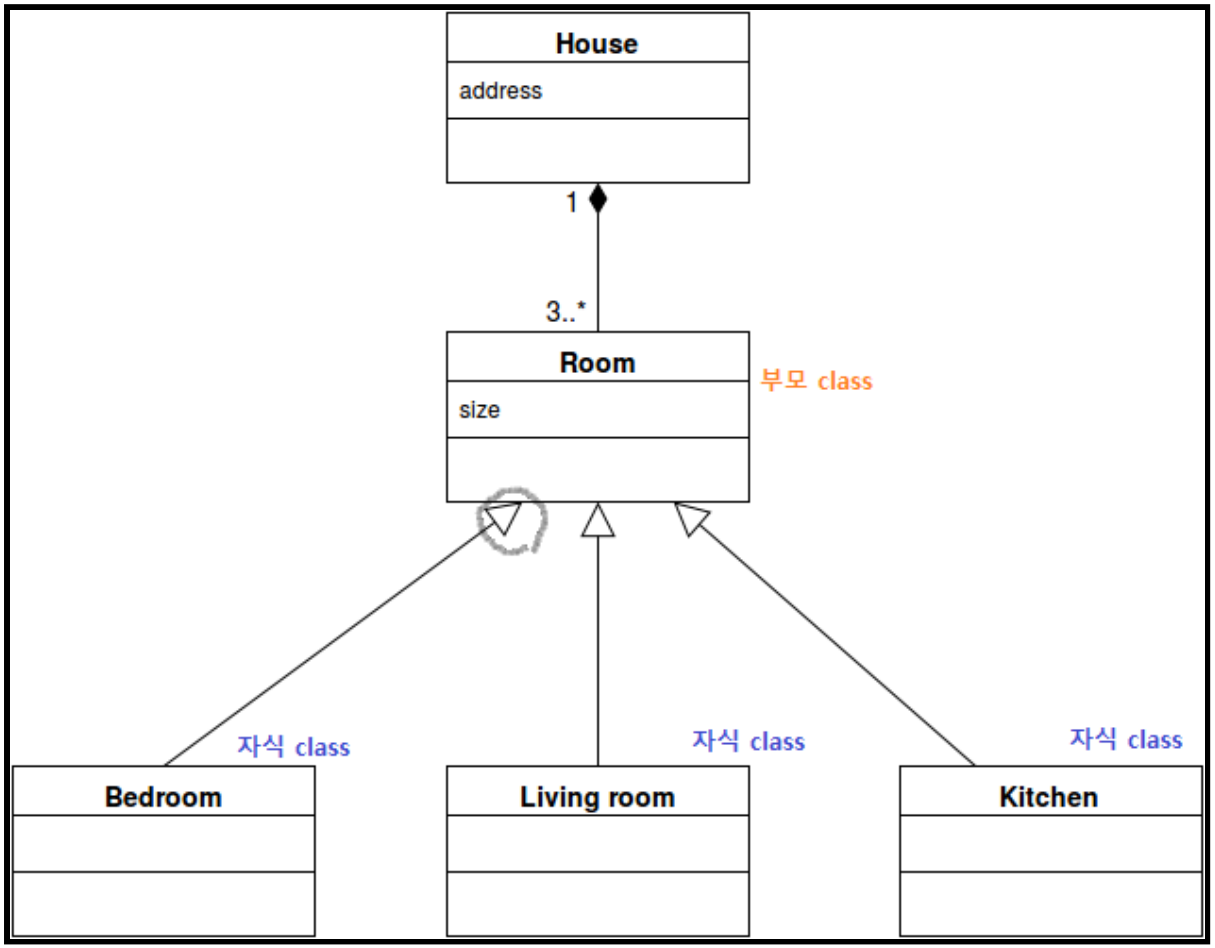
〈상속, Inheritance〉

상속: 어떤 객체가 있을 때 그 객체의 필드(변수)와 method를 다른 객체가 물려 받을 수 있는 기능.

그리고 필요에 따라서 새로운 변수 또는 새로운 method를 추가하거나 기존의 변수나 method를 변경해서 사용할 수 있도록 하는 것.

ex) 1. 본인이 어떤 class를 이용하는데 추가적인 method가 필요하다. 이 class를 본인이 만들지 않았기 때문에 소스를 변경할 수 없을 때.
또는 변경을 할 수있다고 하더라도 추가하려는 method와 기존의 method의 어떠한 이유에 의한 충돌이 될 수도 있다. 이럴 때 상속을 활용.

2. class가 다양한 곳에서 활용되고 있을 때 특수한 경우에만 사용되는 method를 추가하면 다른 사용자 입장에서는 불필요한 기능까지 추가적으로 학습해야 하는 경우가 생기기 때문에. 이럴 때 상속을 활용.



부모 class / 자식 class

상위(super) class / 하위 (sub) class

기초(base) class / 유도(derived)class

〈〈 class diagram에서는 이런식으로 표현 (빈 화살표)

상속의 장점

1. 중복의 제거
2. 재활용성 높음
   (부모class 것을 자식class가 활용)
3. 유지보수 편리
   (부모class 의 개선) 자식class도 개선)
4. 가독성 증가
   (부모class에 있으면 자식class에서 중복x)

› 하위 class가 호출될 때(하위 class의 객체화) 자동으로 상위 class의 기본 생성자(매개변수 사용하지 않는 생성자)를 호출하게 된다.

(하위 class의 생성자가 있는 경우에 하위 class의 생성자도 호출. // 생성자는 상위 class의 생성자를 따라간다는 의미가 아니니 주의)

그런데 상위class에 매개변수가 있는 생성자가 있다면 자바는 자동으로 상위 class의 기본 생성자를 만들어주지 않는다.

이런 경우에는 존재하지 않는 생성자를 호출하게 되는 것이기 때문에 에러가 발생한다. 이 문제를 해결하기 위해서는 상위 class에 기본 생성자를 추가하면 된다.

```
class father{
    public father(int a, int b) { System.out.println(a + b); }
}

class son extends father{
    public son(int a, int b){
        System.out.println(a + b);
    }
}

public class _99th_practice {
    public static void main(String[] args) {
        son lee = new son( a: 5, b: 10);
    }
}
```

상위 class가 이미 int a, b를 사용하는 생성자가 있기 때문에
하위 class의 객체를 생성했을 때 이런 오류가 생김.

```
D:\java_work\LectureContents\java\JuhyungLee\20210601_Java\src\_99th_practice.java:8:29
java: constructor father in class father cannot be applied to given types;
  required: int,int
  found:    no arguments
  reason: actual and formal argument lists differ in length
```

<<< class명 앞 대분마 잊지말자 Father / Son

```
class father{
    public father(){
        System.out.println("이제 된다");
    }
    public father(int a, int b) { System.out.println(a + b); }
}

class son extends father{
    public son(int a, int b) { System.out.println(a + b); }
}

public class _99th_practice {
    public static void main(String[] args) {
        son lee = new son( a: 5, b: 10);
    }
}
```

상위 class에서 기본 생성자 생성 후 error 해결.

```
"C:\Program File
이제 된다
15
```

```java
class father{
    public father(int a, int b) { System.out.println(a + b); }
}
class son extends father{
    public son(int a, int b) { super(a, b); }
}
public class _99th_practice {
    public static void main(String[] args) {
        son lee = new son( a: 5, b: 10);
    }
```

```
"C:\Program F
15

Process finis
```

만약 상위 class의 생성자와 하위 class의 생성자가 같은 로직이 들어가 있다면(중복).
이런식으로도 활용 가능)) 이 경우에는 상위 class의 기본 생성자가 필요 없게된다.

```java
class father{
    public father(int a, int b) { System.out.println(a + b); }
}
class son extends father{
    public son(int a, int b){
        System.out.println(a - b);
        super(a, b);
    }
}
public class _99th_practice {
    public static void main(String[] args) {
        son lee = new son( a: 5, b: 10);
    }
```

<< 하위 class의 생성자에 추가적인 코드가 있을 때 이런 error가 나는 이유는

```java
class father{
    public father(int a, int b) { System.out.println(a + b); }
}
class son extends father{
    public son(int a, int b){
        super(a, b);
        System.out.println(a - b);
    }
}
public class _99th_practice {
    public static void main(String[] args) {
        son lee = new son( a: 5, b: 10);
    }
```

하위 class의 생성자에 추가적인 코드를
상위 class의 생성자 보다 먼저 사용하면 안된다.
무조건 상위 class의 생성자(= super)부터 먼저.
하위 class가 instance화 된다는 것은 상위 class가 이미 instance화 되었다는 것을 의미. 즉 상위 class의 초기화가 진행된 후에 하위 class의 초기화가 가능하다.

```java
class A {
    int a = 10;
    void b () { System.out.println("A"); }
}
//...
class AA extends A{
    int a = 20;
    void b() { System.out.println("AA"); }
    void c() { System.out.println("C"); }
}
public class _1st_ExtendsTest {
    public static void main(String[] args) {
        A a = new A();
        a.b();
        System.out.println("A a:" +a.a);

        AA aa = new AA();
        aa.b();
        aa.c();
        System.out.println("AA aa: "+ aa.a);

        A a1 = new AA(); //A class type의 AA객체를 만듬.
        a1.b();
        System.out.println("A a1: " + a1.a);
        a1.a = 1000;
        System.out.println("A a1: " + a1.a);
        System.out.println("AA aa: "+ aa.a);
        // new의 대상은 AA()
        // 접근 데이터는 데이터타입 A를 참조한다.
        // method는 new된 타입을 따라가고
        // data는 datatype class에   .....
```

```
"C:\Program File
A
A a:10
AA
C
AA aa: 20
AA
A a1: 10
A a1: 1000
AA aa: 20
```

<<< 상속의 개념에 관한 간단한 예제들

추가적인 개념>>

- 상속 > 상속 > 상속 > 상속은 가능
- 한 번에 두 개 이상의 class를 상속 받는 것은 불가능 = multiple inheritance
- 상속 받을 때 선택적으로 항목을 선택해서 상속 받는 것은 불가능(빚도 상속되는 것과 같은 이치)

```java
class Car {
    private float rpm;
    private float fuel;
    private float pressure;
    private String color;

    public float getRpm() { return rpm; }
    public void setRpm(float rpm) { this.rpm = rpm; }
    public float getFuel() { return fuel; }
    public void setFuel(float fuel) { this.fuel = fuel; }
    public float getPressure() { return pressure; }
    public void setPressure(float pressure) { this.pressure = pressure; }
    public String getColor() { return color; }
    public void setColor(String color) { this.color = color; }
}
//...
class SportsCar extends Car {
    private boolean booster;

    public boolean isBooster() { return booster; }
    public void setBooster(boolean booster) { this.booster = booster; }

    @Override
    public String toString() {
        // super의 경우에는 상속해준 상속자를 직접 호출한다.
        // 아래는 super 빼도 됨. // 강사님 예제에서는 빼고 함.
        return "SportsCar{" +
                "rpm=" + super.getRpm() +
                ", fuel=" + super.getFuel() +
                ", pressure=" + super.getPressure()+
                ", color" + super.getColor()+
                ", booster=" + booster +
                '}';
```

```java
public class _1st_CarTest {
public static void main(String[] args) {

    SportsCar audi_r8 = new SportsCar();

    audi_r8.setRpm(150);
    audi_r8.setFuel(2.5f);
    audi_r8.setPressure(1.0f);
    audi_r8.setColor("silver");
    audi_r8.setBooster(false);
    System.out.println(audi_r8);

    Car matiz = new SportsCar();

    matiz.setRpm(60);
    matiz.setFuel(1.2f);
    matiz.setPressure(0.6f);
    matiz.setColor("yellow");
    matiz.setBooster(true);
    System.out.println(matiz);
}
```

Car type의 SportsCar의 객체.

Car type이기 때문에
Car class의 method가 아닌
setBooster()는 사용 불가

error나는 부분 주석처리하고 실행

```
SportsCar{rpm=150.0, fuel=2.5, pressure=1.0, colorsilver, booster=false}
SportsCar{rpm=60.0, fuel=1.2, pressure=0.6, coloryellow, booster=false}
```

booster 부분은 이전에 setter로 setting된 값이 불러와지게 된 것 참고.

```java
class Vehicle {
    private float rpm;
    private float fuel;
    private float pressure;
    private String color;

    public Vehicle(float rpm, float fuel, float pressure, String color) {
        this.rpm = rpm;
        this.fuel = fuel;
        this.pressure = pressure;
        this.color = color;
    }

    @Override
    public String toString() {
        return "Vehicle{" +
                "rpm=" + rpm +
                ", fuel=" + fuel +
                ", pressure=" + pressure +
                ", color='" + color + '\'' +
                '}';
```

```java
class Airplane extends Vehicle {
    private float aileron;
    private float pitch;
    private float rudder;

    public Airplane(float rpm, float fuel, float pressure, String color,
                    float aileron, float pitch, float rudder) {
        // super()는 무엇이 되었든 상속자인 부모를 호출한다.
        // super()만 적혀 있으니 생성자를 호출하게 된다.
        super(rpm, fuel, pressure, color);
        this.aileron = aileron;
        this.pitch = pitch;
        this.rudder = rudder;
    }

    @Override
    public String toString() {
        return "Airplane{" +
                // super.toString()은 부모 클래스의 toString()을 호출한 것이다.
                "super.Vehicle()=" + super.toString() +
                ", aileron=" + aileron +
                ", pitch=" + pitch +
                ", rudder=" + rudder +
                '}';
```

```java
public class _2nd_InheritanceWithSuper {
    public static void main(String[] args) {
        Vehicle v = new Vehicle( rpm: 200,  fuel: 1.2f,  pressure: 1.0f,  color: "Red");
        System.out.println(v);
        Airplane a = new Airplane(
                rpm: 1000,  fuel: 112.5f,  pressure: 12.3f,  color: "White",
                aileron: 77.3f,  pit
        System.out.println(a);
```

```
"C:\Program Files\Java\jdk-16\bin\java.exe" -javaagent:C:\Users\Samuel\AppData\Local\JetBrains\Toolbox\apps\IDEA-C\ch-0\211.71
Vehicle{rpm=200.0, fuel=1.2, pressure=1.0, color='Red'}
Airplane{super.Vehicle()=Vehicle{rpm=1000.0, fuel=112.5, pressure=12.3, color='White'}, aileron=77.3, pitch=0.02, rudder=33.9}
```

〈InheritanceWithSuper〉

〈Quiz.53〉

상속 관련 간단하게 개념 잡는 quiz

```java
class LeeGunHee{
    int age;
    String sex;
    String residence;
    String belong_to;

    public LeeGunHee(int age, String sex, String residence, String belong_to) {
        this.age = age;
        this.sex = sex;
        this.residence = residence;
        this.belong_to = belong_to;
    }
    @Override
    public String toString() {...}
}
class LeeJaeYong extends LeeGunHee{
    String spouse;
    String position;
    // alt + insert > constructor
    public LeeJaeYong(int age, String sex, String residence, String belong_to,
                      String spouse, String position) {
        super(age, sex, residence, belong_to);
        this.spouse = spouse;
        this.position = position;
    }

    @Override
    public String toString() {...}
```

```java
public class _2nd_.Quiz53 {
    public static void main(String[] args) {
        // 클래스 이건희를 만들고 클래스 이재용을 만든다

        LeeGunHee lee1 = new LeeGunHee( age: 87, sex: "male",  residence: "용산", belong_to: "삼성");

        LeeJaeYong lee2 = new LeeJaeYong( age: 55,  sex: "male",  residence: "용산",
                belong_to: "삼성",  spouse: "임세령(이혼)",  position: "부사장");

        System.out.println(lee1);
        System.out.println(lee2);
```

```
"C:\Program Files\Java\jdk-16\bin\java.exe" -javaagent:C:\Users\Samuel\AppData\Local\JetBrains\Toolbox\ap
LeeGunHee{age=87, sex='male', residence='용산', belong_to='삼성'}
LeeJaeYong{age=55, sex='male', residence='용산', belong_to='삼성', spouse='임세령(이혼)', position='부사장'}
```

```java
interface RemoteControl{
    public void turnOn();
    public void turnOff();
}
    /*...*/
class Abstract{ // abstract = 추상화
    RemoteControl rc_car = new RemoteControl() {
        @Override
        public void turnOn() { System.out.println("RC 자동차용 리모컨 ON. RF 송수신기 활성화"); }
        @Override
        public void turnOff() { System.out.println("RC 자동차용 리모컨 OFF. RF 송수신기 비활성화"); }
    };

    RemoteControl radio = new RemoteControl() {
        @Override
        public void turnOn() { System.out.println("RADIO ON. 주파수 채널 매칭 START"); }
        @Override
        public void turnOff() { System.out.println("RADIO OFF. 주파수 채널 매칭 TERMINATE"); }
    };

    public void testMethod(){
        RemoteControl tv = new RemoteControl() {
            @Override
            public void turnOn() { System.out.println("TV ON, AM/FM 신호 수신"); }
            // 원래 같은 return type, 같은 이름, 같은 입력형태이면 error나야되는데 안 남
            // 인터페이스라서 가능.
            // rc_car_turnOn / tv_turnOn 이런 식으로 따로 만들 필요 없다는 얘기.
            @Override
            public void turnOff() { System.out.println("TV OFF, AM/FM 신호 차단"); }
        };
        tv.turnOn();
        radio.turnOff();
    }
    public void testMethod2 () {
        rc_car.turnOn();
        radio.turnOff();
    }
```

사용자가 알아야 할 것

프로그래머들이 해야되는 것

〈Interface / 추상화〉

/* 리모콘 제조사가 수십만 개

public void companyATurnOn();

public void companyBTurnOn();

...

public void companyZTurnOn();

public void companyAATurnOn();

...

public void companyAZTurnOn();

public void companyZZTurnOn();

.....

TurnOn method만 수십만 개 있을 수 있음.

*/

그러나 사용자들은 그냥 turnOn / turnOff만 알면 된다.

다른건 프로그래머들이 집 못 가면서 다 해줌...

OOP(객체지향)에서 제일 중요시 여기는 것이 바로 추상화.

추상화란 궁극적으로 무엇을 추구하는것인가 ?

굳이 모든 내용을 알지 않더라도 쉽게 실행 할 수 있도록 하는 것.

복잡하고 어렵고 토나오는것은 프로그래머가.

//eX) 자바 라이브러리 개발자 진영 및 스프링 프레임워크 개발 진영:

"API는 우리가 영혼 갈아서 만들어줄테니까"

"라이브러리 사용자들은 편하게 API 사용해서 개발만 해라"

// 입력 ----〉 Black Box ---〉 출력

사용자는 BlackBox 내용 알 필요 없음.

⟨Quiz.54⟩

interface

개념잡기 quiz

인터페이스 작성) interface 인터페이스명 { method prototype }

(method prototype: "public void methodname()" 이런 형태)

```
interface LightControl{
    public void turn_on();
    public void turn_off();
    public void blink_on();
}
class Lamp{
    LightControl lamp_control = new LightControl() {
        @Override
        public void turn_on() { System.out.println("<램프>에 전기신호를 일정하게 주고 지속적으로 켜져있게 한다"); }
        @Override
        public void turn_off() { System.out.println("<램프>에 전기신호를 차단한다"); }
        @Override
        public void blink_on() { System.out.println("<램프>에 전기신호를 일정한 주기로 주었다가 껐다를 반복하여 깜빡이게 한다."); }
    };
}
class Led{
    LightControl led_control = new LightControl() {
        @Override
        public void turn_on() { System.out.println("<led>에 전기신호를 일정하게 주고 지속적으로 켜져있게 한다"); }
        @Override
        public void turn_off() { System.out.println("<led>에 전기신호를 차단한다"); }
        @Override
        public void blink_on() { System.out.println("<led>에 전기신호를 일정한 주기로 주었다가 껐다를 반복하여 깜빡이게 한다."); }
    };
}
class Street_lamp{
    LightControl Street_lamp_control = new LightControl() {
        @Override
        public void turn_on() { System.out.println("<Street Lamp>에 전기신호를 일정하게 주고 지속적으로 켜져있게 한다"); }
        @Override
        public void turn_off() { System.out.println("<Street Lamp>에 전기신호를 차단한다"); }
        @Override
        public void blink_on() { System.out.println("<Street Lamp>에 전기신호를 일정한 주기로 주었다가 껐다를 반복하여 깜빡이게 한다."); }
    };
```

```java
public class _3rd_Quiz54 {
    public static void main(String[] args) {
        // Interface문제
        // Lamp Class / Led Class / StreetLamp Class작성
        // 3개는 모두 lightOn, LightOff 기능이 있다.
        // 세부사항은 본인 마음대로 사용 목적에 걸맞게 작업

        Lamp lp = new Lamp();
        lp.lamp_control.turn_on();
        lp.lamp_control.turn_off();
        lp.lamp_control.blink_on();
        System.out.println(); // 띄어쓰기용

        Led ld = new Led();
        ld.led_control.turn_on();
        ld.led_control.turn_off();
        ld.led_control.blink_on();
        System.out.println(); // 띄어쓰기용

        Street_lamp sl = new Street_lamp();
        sl.Street_lamp_control.turn_on();
        sl.Street_lamp_control.turn_off();
        sl.Street_lamp_control.blink_on();
    }
```

```
"C:\Program Files\Java\jdk-16\bin\java.exe" -javaagent:C:\Users\Samuel
<램프>에 전기신호를 일정하게 주고 지속적으로 켜져있게 한다
<램프>에 전기신호를 차단한다
<램프>에 전기신호를 일정한 주기로 주었다가 껐다를 반복하여 깜빡이게 한다.

<led>에 전기신호를 일정하게 주고 지속적으로 켜져있게 한다
<led>에 전기신호를 차단한다
<led>에 전기신호를 일정한 주기로 주었다가 껐다를 반복하여 깜빡이게 한다.

<Street Lamp>에 전기신호를 일정하게 주고 지속적으로 켜져있게 한다
<Street Lamp>에 전기신호를 차단한다
<Street Lamp>에 전기신호를 일정한 주기로 주었다가 껐다를 반복하여 깜빡이게 한다.
```

Abstract: 추상/추상화

〉 공통 특성/공통 속성을 추출하여 실제 사물을 단순화하는 작업.

〉 추상 class를 상속받아 기능을 이용하고 확장시키는 것.

〉 상속을 강제하는 일종의 규제

〉 즉. abstract class나 method를 사용하기 위해서는 반드시 상속해서 사용하도록 강제하는 것이 abstract.

어떤 객체(class)가 있고 그 객체가 특정한 interface를 사용(implements)한다면 그 객체는 반드시 interface의 method들을 구현해야한다.

〉 만약 interface에서 강제하고 있는 method를 구현하지 않으면 compile 조차 되지 않는다.

	추상 클래스(abstract)	추상 메소드(abstract method)	인터페이스(Interface)
개념	존재하지 않는 혹은 하나 이상의 추상 메소드를 가지고 있는 것	내부가 구현되어 있지 않은 abstract으로 정의된 메소드	- 내부가 비어 있는 메소드들의 형태(추상 메소드)들만 써놓은 것 - 인터페이스를 상속하는 클래스들이 반드시 구현해야 하는 메소드들의 집합
목적	상속	-	구현 객체가 같은 동작을 한다는 것을 보장
키워드	extends	-	implements
멤버	일반 메소드, 추상 메소드 O 일반 변수 X	일반 메소드, 일반 변수 O	- 모든 메소드는 추상 메소드 - 모든 변수들은 정적(static) - 일반 변수, 일반 메소드 X
생성자 여부	추상 클래스를 상속받은 클래스를 통한 인스턴스화 가능 추상 클래스 명 test = new 클래스 명( )	-	인터페이스를 구현한 클래스를 통한 인스턴스화 가능 인터페이스 명 test = new 클래스 명( )
접근 지정자	모두 가능	-	default, public, abstract
속도	추상 클래스 〉 인터페이스		

《 추가 개념 》

추상 클래스의 상속

- 클래스에서 추상 클래스를 상속: 존재하는 추상 메소드를 모두 구현
- 추상 클래스에서 추상 클래스를 상속: 모두 구현하지 않아도 됨

인터페이스의 키워드 implements

- 직역하면 '실행', '실천'이라는 뜻이므로 인터페이스 내의 메소드를 구현하여 실행한다고 생각하면 좋을 듯.

추상 메소드만 선언 〉 인터페이스 사용

일반 메소드, 필드와 함께 선언 〉 추상 클래스 사용

자바는 다중 상속(multiple inheritance)을 지원하지 않는다.

상속은 상위 클래스의 기능을 이용하거나 확장하기 위해 사용되며 다중 상속의 모호성을 막고자 한 개의 상위 클래스만 상속받을 수 있다.

그러나 자바에서 여러 개의 인터페이스를 구현할 수 있어 마치 여러 개의 클래스를 상속받는 것처럼 보인다.

하지만 이것이 인터페이스의 존재 이유는 아니다.

위 표에도 적혀 있듯이, 인터페이스는 해당 인터페이스를 구현한 객체들에 대해서 동일한 동작을 약속하기 위해 존재한다.