

## 4/27 스터디노트

### ● P249 익명구현객체

- ✓ 이전에 배운 인터페이스 사용하는 방법

#### (1) 인터페이스생성 - 추상메서드

```
public interface Printer {
    int INK = 100;
    void print();
}

public interface Scanner {
    void scan();
}

public interface Fax {
    String FAX_NUMBER = "02-1234-5678";
    void send(String tel);
    void receive(String tel);
}
```

#### (2) 구현클래스생성 - 메서드 오버라이딩

```
public class Complexer implements Printer, Scanner, Fax {
    @Override
    public void send(String tel){
        System.out.println(FAX_NUMBER+"에서 "+tel+"로 Fax 전송");
    }
    @Override
    public void receive(String tel){
        System.out.println(tel+"에서 "+FAX_NUMBER+"로 Fax 수신");
    }
    @Override
    public void scan(){
        System.out.println("스캔 실행");
    }
    @Override
    public void print(){

```

#### (3) 실행클래스생성(메인) - 객체생성,메서드실행

```
public class ComplexerMain {
    public static void main(String[] args) {
        Complexer com = new Complexer();

        System.out.println(Complexer.INK);
        System.out.println(Complexer.FAX_NUMBER);

        com.print();
        com.scan();
        com.send(tel: "02-8765-4321");
        com.receive(tel: "02-8765-4321");
    }
}
```

- ✓ 그러나 (2) 에 해당하는 구현클래스 만드는게 귀찮다면 → (2)를 안만들고 (3)실행하는 메인클래스에서 인터페이스의 객체로 (1)을 구현해버림  
= **익명 구현 객체** (진짜 인터페이스의 객체라기 보다는 인터페이스를 구현한 익명클래스의 객체 라는 의미)

#### (1) 인터페이스 생성: 추상메서드

```
public interface Printer {
    int INK = 100;
    void print();
}

public interface Scanner {
    void scan();
}

public interface Fax {
    String FAX_NUMBER = "02-1234-5678";
    void send(String tel);
    void receive(String tel);
}
```

#### (2) 메인클래스에서

익명구현객체 생성,  
메서드 오버라이딩

```
public class ComplexerMain2 {
    public static void main(String[] args) {
        Fax fax = new Fax(){
            @Override
            public void send(String tel) {
                System.out.println("여기는 익명 구현 객체의 send()");
            }
            @Override
            public void receive(String tel) {
                System.out.println("여기는 익명 구현 객체의 receive()");
            }
        };
    }
}
```

실행문이므로 ; 을 꼭 붙여야함!!

- P250 익명구현객체 하나로 여러 개의 인터페이스를 구현하는 방법

- ✓ 앞의 fax 익명구현객체 예제는 Printer, Fax, Scanner 3개의 인터페이스 중 Fax 인터페이스만 구현가능하다.
- ✓ (익명구현객체는 하나의 인터페이스만 구현 가능하다.)
- ✓ 그러면 익명구현객체 하나로 여러 개의 인터페이스를 구현하고 싶다면?
- ✓ 여러 개의 인터페이스를 상속받는 하나의 인터페이스를 만들고, 그 인터페이스의 익명구현객체를 생성한다

`public interface ComplexerInterface extends Printer, Scanner, Fax {  
}`

`public class ComplexerMain3 {  
 public static void main(String[] args) {  
 ComplexerInterface ci = new ComplexerInterface() {  
 @Override  
 public void send(String tel) { System.out.println("익명구현객체의 send()"); }  
  
 @Override  
 public void receive(String tel) { System.out.println("익명구현객체의 receive()"); }  
  
 @Override  
 public void print() { System.out.println("익명구현객체의 print()"); }  
  
 @Override  
 public void scan() { System.out.println("익명구현객체의 scan()"); }  
 };  
 }  
}`

## ● P251 인터페이스의 다형성

- ✓ 교재 P221 클래스 다형성 예제와 동일하지만, class -> interface 만 변경되었음 (따라서 extends 도 implements 로 바뀌었음)

```
public class GraphicCard {
    int memory;
    public void process(){
        System.out.println("그래픽 처리");
    }
}
```

```
public interface GraphicCard {
    String MEMORY = "2G";
    public void process();
}
```

```
public class Amd extends GraphicCard {
    public void process(){
        System.out.println("AMD 그래픽 처리");
    }
}
```

```
public class Nvidia implements GraphicCard {
    public void process(){
        System.out.println("Nvidia 그래픽 처리");
    }
}
```

```
public class Nvidia extends GraphicCard{
    public void process(){
        System.out.println("Nvidia 그래픽 처리");
    }
}
```

```
public class Amd implements GraphicCard {
    public void process(){
        System.out.println("AMD 그래픽 처리");
    }
}
```

main 클래스인 computer 클래스에서  
객체 gc 를 부모(GraphicCard)자료형으로  
생성했기 때문에  
자식인 Amd, Nvidia의 새 인스턴스를  
대입하여 바뀌가며 메서드 사용가능.

```
public class Computer {
    public static void main(String[] args) {
        GraphicCard gc = new GraphicCard();
        gc.process();    // 원래 그래픽카드

        gc = new Amd();    // Amd 객체를 부모클래스 자료형에 대입
        gc.process();    // Amd 메서드 실행됨

        gc = new Nvidia();    // 이번에는 Nvidia 객체를 대입
        gc.process();    // 그러면 Nvidia 메서드가 실행됨
    }
}
```

```
public class Computer {
    public static void main(String[] args) {
        GraphicCard gc = new Amd();
        System.out.println("메모리: "+gc.MEMORY);
        // Amd 로 생성

        gc = new Amd();    // 자식 -> 부모 자동형변환
        gc.process();    // Amd 메서드 실행됨
        // Nvidia로 교체

        gc = new Nvidia();    // 자식 -> 부모 자동형변환
        gc.process();    // Nvidia 메서드 실행됨
    }
}
```

### 클래스 상속과 다른점,

상위클래스 GraphicCard를  
인터페이스로 만들면  
하위클래스에서  
모든 메서드를 꼭 구현하도록  
강제할 수 있음.  
(상속은 모든 메서드 강제x)

## ● P254 매개변수 다형성

- ✓ 교재p223 메서드의 매개변수를 부모클래스자료형으로 정의하여 자식클래스자료형이 매개변수로 들어와도 메서드가 처리되게끔 할 수 있었음.
- ✓ 인터페이스 구현시에도 동일하게 가능.

```
public interface GraphicCard {
    String MEMORY = "2G";
    public void process();
}
```

공통의 메서드 (추상메서드)를 가지고 있는 인터페이스

- process ( );

```
public class Amd implements GraphicCard {
    public void process(){
        System.out.println("AMD 그래픽 처리");
    }
}
```

```
public class Nvidia implements GraphicCard {
    public void process(){
        System.out.println("Nvidia 그래픽 처리");
    }
}
```

공통의 메서드 (추상메서드)를

구현하는

Amd, Nvidia 클래스

```
public class Game {
    void display(GraphicCard gc){ gc.process(); }
}
```

인터페이스 자료형을 매개변수로 받아 공통의 메서드 (추상메서드)가 실행되도록 하는 메서드 생성

- display( );

```
public class Computer2 {
    public static void main(String[] args) {
        Game g = new Game();
        Amd gc = new Amd();
        g.display(gc);

        Nvidia gc2 = new Nvidia();
        g.display(gc2);
    }
}
```

display( ); 메서드를 가지는 Game 클래스의 객체 g 생성

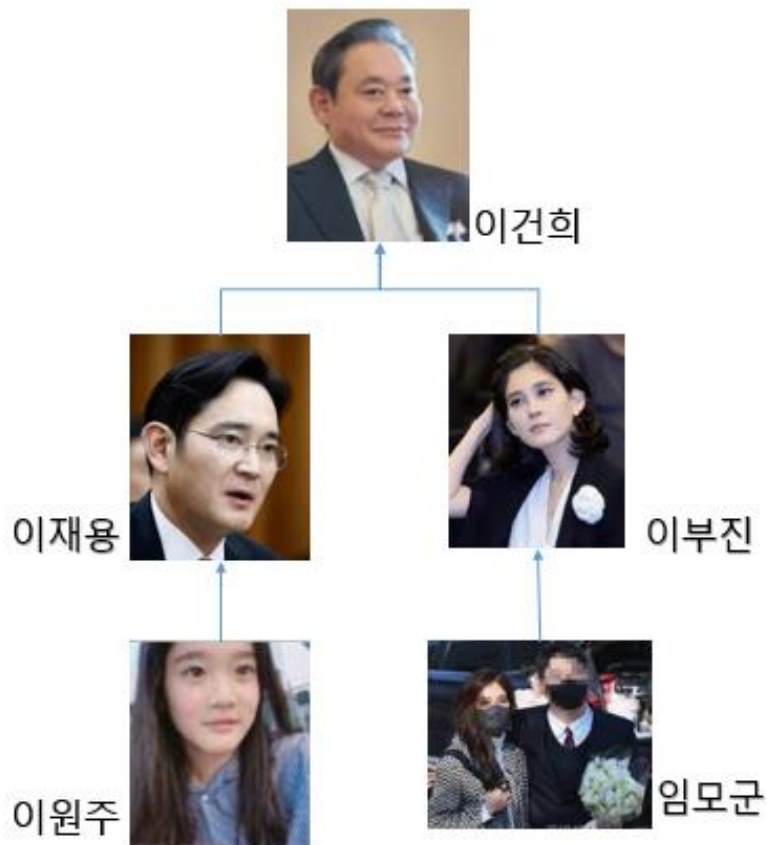
AMD 그래픽 처리  
Nvidia 그래픽 처리

## ● P256 instanceof

- ✓ 인터페이스 간에도 상속이 가능함
- ✓ 상속 관계에서는 형변환이 가능한데, 상속이 복잡하게 되어있다면 -> 이 두 클래스 간 형변환이 가능한지 헷갈릴 때가 있을 수 있음
- ✓ 관계를 알고 싶을 때 (=형변환이 가능한지 확인할때) 사용하는 함수 : instanceof
- ✓ instanceof = 앞의 객체가 뒤쪽 클래스의 객체인가? -> 결과: True(예) / False(아니오)
- ✓ 예시 : 삼성가

상속관계에서의 형변환 다시 정리하기

- 자식은 부모가 될 수 있고 (자동형변환)
- 부모는 자식이 될 수 없다 (강제형변환도 불가)
- 부모가된 자식은 다시 자식이 될 수 있다 (강제형변환)



이원주는 이건희가 될 수 있다. (자식 -> 부모)

이재용은 이부진이 될 수 있다. (자식 -> 부모)

이원주는 이부진은 될 수 없다. (자식 -> 이모(?))

java 코드로 쓰면 (클래스는 성+이름, 객체는 이름으로 작성함)

```
이건희 건희 = new 이원주( );
```

```
이재용 재용 = new 이원주( );
```

```
이원주 원주 = new 이원주( );
```

```
이건희 건희2 = new 임모군( );
```

```
이부진 부진 = new 임모군( );
```

```
임모군 모군 = new 임모군( );
```

instanceof 로 확인을 해보면

```
System.out.println(건희 instanceof 이원주); => true
```

```
System.out.println(재용 instanceof 이원주); => true
```

```
System.out.println(원주 instanceof 이부진); => false
```

- P259 default 메서드와 static 메서드

- 인터페이스에는 default 메서드와 static 메서드가 있다 (자바8버전부터)

- ✓ default 메서드는 구현부가 없는 추상메서드로서, 인터페이스를 구현하는 클래스가 꼭 구현해야함(재정의, 오버라이딩)
- ✓ static 메서드는 구현부가 있는 메서드
- ✓ 사용하는 방법

#### <interface1>

```
public interface Myinterface1 {  
  
    //JAVA8버전 이후 부터 default, static 메서드를 지원함으로써  
    //Field(속성)을 제외한 기능(Method)에 대해 다중상속의 효과를 낼 수 있다.  
  
    //default 접근 제어 속성만 가능하며  
    // 접근범위  
    // 모든 클래스 : x  
    // 상속 : o  
    // 동일클래스 : o  
    // 동일패키지 : o  
    default void defaultMethod(){  
        System.out.println("Myinterface1의 default 메서드");  
    }  
    // 인터페이스의 static 메서드 사용법 : [인터페이스명].[메서드명]  
    static void staticMethod(){  
        System.out.println("Myinterface2의 static 메서드");  
    }  
}
```

#### <interface2>

```
public interface Myinterface2 {  
    default void defaultMethod(){  
        System.out.println("MyInterface2의 default 메서드");  
    }  
    static void staticMethod(){  
        System.out.println("MyInterface2의 static 메서드");  
    }  
}
```

#### <interface1, 2 를 모두 구현하는 Child라는 클래스>

```
public class Child implements Myinterface1, Myinterface2 {  
    @Override  
    public void defaultMethod() {  
        System.out.println("Child 클래스의 default 메서드");  
        Myinterface1.super.defaultMethod();  
        Myinterface2.super.defaultMethod();  
    }  
}
```

#### <메인 클래스>

```
public class DefaultStaticEx {  
    public static void main(String[] args) {  
        Child c = new Child();  
        c.defaultMethod();  
  
        Myinterface1.staticMethod();  
        Myinterface2.staticMethod();  
    }  
}
```

Child 클래스의 객체 c 생성

Child 클래스의 defaultMethod 수행

default 메서드를 수행하려면

객체를 생성해야함~

: 문자열 프린트+interface1과 2의 default method 를 수행함

interface1과 2의 static method도 각각 수행함 (static 은 객체없이 바로 수행)

- P261-P264 어노테이션 : 스프링에서 배우는 것으로 합시다.

- P265 9장 연습문제

- ✓ 1번 : 클래스가 인터페이스 구현할 때 사용하는 예약어: implements
- ✓ 2번 : 인터페이스에 대한 설명으로 옳지 않은 것 ③ 구현객체를 인터페이스로 형변환하려면 강제 형변환을 해야한다 : X 자동 형변환 가능함

- ✓ 3번 : 아래의 Player 라는 인터페이스를 이용해 출력결과가 오른쪽 처럼 나올 수 있도록 Player인터페이스를 상속받는 BaseBallPlayer와 FootBallPlayer 클래스를 정의하세요

야구선수가 야구를 합니다.  
축구선수가 축구를 합니다.

```
package chapter09;

interface Player {

    // 추상 메서드
    void play();

}

public class Exercise3 {

    public static void main(String[] args) {

        Player p1 = new BaseBallPlayer();
        Player p2 = new FootBallPlayer();

        playGame(p1);
        playGame(p2);

    }

    public static void playGame(Player p) {
        p.play();
    }

}
```

```
class BaseBallPlayer implements Player{

    @Override
    public void play() {
        System.out.println("야구선수가 야구를 합니다.");
    }

}

class FootBallPlayer implements Player{

    @Override
    public void play() {
        System.out.println("축구선수가 축구를 합니다.");
    }

}
```

- ✓ 4번 : 아래 Tv라는 인터페이스를 만들고 Excercise4 클래스의 main( ) 메서드에서 Tv 인터페이스의 익명 구현 객체를 생성해 실행 결과가 동일하게 출력되도록 코드를 완성하세요.

```
interface Tv{
    void turnOn();
}

public class Excercise4 {
    public static void main(String[] args){
        Tv p1 = new Tv() {
            @Override
            public void turnOn() {
                System.out.println("TV를 켭니다.");
            }
        };
        p1.turnOn();
    }
}
```

TV를 켭니다.



## Chapter10 내부클래스(중첩클래스)

### ● P268 내부(중첩)클래스

#### ➤ 중첩클래스의 구조

- ✓ 밖에 있는 걸 외부클래스, outer class / 안에 있는 걸 내부클래스, inner class 라고 부름

#### ➤ 내부클래스의 종류

- **멤버클래스** : 클래스의 멤버로 정의됨 = 객체로 생성된 후 어디서든 다시 사용 가능

두가지로 나뉨 - **static inner** 클래스 : 내부클래스의 객체를 외부클래스에 대해 독립적으로 생성함.

※ 만약 내부클래스내에 static메서드가 있으면 해당 내부클래스는 반드시 static으로 선언해야함

-> 다른 곳에서 static inner에 접근할 때 객체를 생성하는 방법 : `Outer.Inner inn = new Outer.Inner();`

- **인스턴스 inner** 클래스 : static 키워드가 없는 내부클래스

-> 다른 곳에서 인스턴스 inner에 접근할 때 객체 생성 : 외부클래스의 객체를 먼저 생성하고

그 객체변수를 이용해서 내부클래스의 객체 생성

```
Outer out = new Outer();  
Outer.Inner_nonstatic inn = out.new Inner_nonstatic;
```

- **로컬 클래스** : 메서드 내부에 정의됨 = 메서드 내에서만 사용됨

#### ➤ 이너클래스를 쓰는 이유는

##### 1. 로직에 밀접한 영향을 주지만 간단한 내용의 클래스는

다른 파일(외부)에 있는 것보다 내부에서 정의할 때(=inner class) 코드 실행문 부분과 인접해있어서 가독성이 좋기 때문이다.

##### 2. 동일한 클래스 안에 정의가 되기 때문에 외부/내부 클래스간 필드/메서드 참조 및 호출이 자유롭다.

#### ➤ 이너클래스의 단점 (: 선생님은 간단한 클래스여도 외부에서 작성하는 것을 추천합니다.)

##### 1. 다른 클래스에서 쉽게 재 사용할 수 없다.

##### 2. 구조화된 프로그램 개발이 어려울 수 있다.

##### 3. 복잡한 클래스는 오히려 외부로 정의하는 것이 유리하다.

➤ P269 이너클래스 예제

```
public class LocalInnerEx {    외부클래스
    int i = 10;
    void outerMethod(){        외부클래스의 메소드

        class Inner {        메소드 내부에 선언된 내부클래스 (=로컬클래스)
            int x=20;
            int i=30;

            void innerMethod(){
                System.out.println(x);
                System.out.println(i);
                System.out.println(this.i);
                System.out.println(LocalInnerEx.this.i);
            }

        }

        Inner inn = new Inner();    외부클래스 메소드의 실행문
        inn.innerMethod();          : 로컬클래스의 객체를 생성하고, 로컬클래스의 메서드를 수행함

    }

    public static void main(String[] args) {
        LocalInnerEx lic = new LocalInnerEx();
        lic.outerMethod();
    }
}
```

로컬클래스가 가지는 메서드

로컬클래스의 변수 x

로컬클래스의 변수 i

로컬클래스의 변수 i

외부클래스의 변수 i

메인클래스

: 로컬클래스의 메서드를 수행하기 위해서는

해당 로컬클래스를 가지고 있는 외부클래스의 메서드를 수행해야함.

따라서 외부클래스의 객체 생성 후 외부클래스의 메서드 시행

20  
30  
30  
10

➤ P271 예제

```
class A {    외부클래스 A

    public A() { System.out.println("A 객체 생성"); }    클래스 A 생성자

    static class B{    내부 static 클래스 B
        public B() { System.out.println("B 객체 생성"); }    클래스 B 생성자
        int var1;
        static int var2;    클래스 B 필드
        void method1(){ System.out.println("static 내부 클래스의 method1()");}    클래스 B 메서드1
        static void method2(){ System.out.println("static 내부 클래스의 static method2()");}    클래스 B 메서드2 (static)
    }

    public class C {    내부 인스턴스 클래스 C
        public C() { System.out.println("C 객체 생성");}
        int var1;
        void method1() { System.out.println("인스턴스 내부 클래스의 method1()"); }
    }

    void method(){    외부클래스 A 의 메서드
        class D{    외부클래스 A 의 메서드 안에 있는 클래스 D (=로컬클래스)
            public D() { System.out.println("D 객체 생성"); }
            int var1;
            void method1(){ System.out.println("로컬 내부 클래스의 method1()"); }
        }
        D d = new D();    외부클래스 A 의 메서드의 실행문
        d.var1 = 3;
        d.method1();
    }
}
```

```

public class Amain {    메인클래스 Amain
    public static void main(String[] args) {
        A a = new A();

        //외부클래스인 A클래스의 static 멤버 클래스인 클래스 B의 객체 생성
        A.B b = new A.B();    내부 클래스 중 static 내부 클래스는 new 생성자만 이용해서 바로 생성가능
        b.var1 = 3;
        b.method1();           내부 클래스의 static이 아닌 메서드는 내부클래스 객체를 이용해서 수행
        A.B.var2 = 3;
        A.B.method2();         내부 클래스의 static 메서드는 객체를 통하지 않고 바로 수행 : 사용방법은 [외부클래스].[내부클래스].[메서드명]

        //외부클래스인 A클래스의 인스턴스 내부 클래스인 클래스 C의 객체 생성
        A.C c = a.new C();    내부 클래스 중 인스턴스 내부 클래스는 new 생성자와 외부클래스 객체를 이용해서 생성
        c.var1 = 3;
        c.method1();

        //외부클래스인 A클래스의 메서드 안에 있는 로컬 클래스인 클래스 D의 객체 생성 : = 메서드 호출
        a.method();           내부 클래스 중 로컬 클래스는 외부 클래스의 메서드를 사용하여 생성
    }
}

```

- 선생님 예제 : 학생 점수 관리 프로그램 1) 중첩클래스를 이용하지 않고 다른 파일에 클래스 생성 (코드 해석은 다음페이지)

```
public class Student {  
    private String name;  
    private int[] grades;  
  
    public Student(String name, int[] grades) {  
        this.name = name;  
        this.grades = grades;  
    }  
  
    public String getName() { return name; }  
  
    public double getAverageGrade(){  
        int sum = 0;  
        for (int grade : grades){ sum+=grade; }  
        return (double) sum/grades.length;  
    }  
}
```

|           |
|-----------|
| 홍길동: 90.0 |
| 김철수: 75.0 |
| 오명달: 95.0 |

```
public class GradeBook {  
    private Student[] students;  
    private int numStudents;  
  
    public GradeBook(int maxNumStudents){  
        students = new Student[maxNumStudents];  
        numStudents = 0;  
    }  
  
    public void addStudent(String name, int[] grades){  
        if (numStudents < students.length){  
            students[numStudents] = new Student(name, grades);  
            numStudents++;  
        }  
    }  
  
    public void printGradeReport(){  
        for (int i = 0; i < numStudents; i++) {  
            System.out.println(students[i].getName()+": "+students[i].getAverageGrade());  
        }  
    }  
  
    public static void main(String[] args) {  
        GradeBook book = new GradeBook( maxNumStudents: 3);  
  
        book.addStudent( name: "홍길동", new int[]{90, 85, 95});  
        book.addStudent( name: "김철수", new int[]{80, 75, 70});  
        book.addStudent( name: "오명달", new int[]{95, 90, 100});  
  
        book.printGradeReport();  
    }  
}
```

```

public class Student {
    private String name;
    private int[] grades;

    public Student(String name, int[] grades) {
        this.name = name;
        this.grades = grades;
    }

    public String getName() { return name; }

    public double getAverageGrade(){
        int sum = 0;
        for (int grade : grades){ sum+=grade; }
        return (double) sum/grades.length;
    }
}

```

## Student 클래스

: 학생이름과 학생의 점수들(배열)로 이루어진 객체

: 학생이름을 리턴해주는 getter 함수

: 학생의 점수들의 평균값을 구해주는 메서드 생성 getAverageGrade( );

sum 이라는 변수 선언 후

for 반복문 사용 → sum 에다가 grade 배열의 점수값 하나하나를 더해서 대입

```

public class GradeBook {
    private Student[] students;
    private int numStudents;

    public GradeBook(int maxNumStudents){
        students = new Student[maxNumStudents];
        numStudents = 0;
    }

    public void addStudent(String name, int[] grades){
        if (numStudents < students.length){
            students[numStudents] = new Student(name, grades);
            numStudents++;
        }
    }

    public void printGradeReport(){
        for (int i = 0; i < numStudents; i++) {
            System.out.println(students[i].getName()+" : "+students[i].getAverageGrade());
        }
    }

    public static void main(String[] args) {
        GradeBook book = new GradeBook( maxNumStudents: 3);

        book.addStudent( name: "홍길동", new int[]{90, 85, 95});
        book.addStudent( name: "김철수", new int[]{80, 75, 70});
        book.addStudent( name: "오명달", new int[]{95, 90, 100});

        book.printGradeReport();
    }
}

```

## GradeBook 클래스

→ 학생이름과 성적배열로 이루어진 Student 배열 생성,  
학생수라는 변수 생성

→ GradeBook 클래스 생성

int 매개변수를 입력받으면 그 크기만큼의 길이를 가지는 student 배열이며  
numStudent 변수 초기값은 0 명

→ 학생수를 추가해주는 메서드 생성 addStudent( );

if 문 사용 : numStudent 변수가 student 배열의 길이보다 작으면  
String 학생이름과 int[] 성적배열 을 입력받아서 배열에 넣고  
학생수 변수(numStudents)값을 1 더해줌

→ 평균성적값을 출력해주는 메서드 생성 printGradeReport( );

for 문 사용 :

학생이름과 성적들로 이루어진 Student 배열의 0 번째(i) 값부터  
numStudents 까지 i 를 1 씩 더해가면서, 배열의 다음값을 출력

→ 메인메서드

GradeBook 클래스의 book 객체 생성하고, 학생수는 3 임

➔ addStudent 메서드로 Student 배열의 길이인 3 될때까지 학생추가  
첫번째학생은 홍길동, 그다음 학생은 김철수, 마지막은 오명달

➔ 세 학생의 학생별 평균 성적값 출력하는 메서드 시행

- P273-P275 내부 인터페이스 : 생략

- P276 10장 연습문제

- ✓ 1번 : 내부 클래스에 대한 설명 중 옳지 않은 것은? ① 로컬 클래스는 한번 생성하면 다른 메서드내에서도 사용가능  
: X 로컬클래스는 메서드 내에 정의되는 클래스로, 해당 메서드 내에서만 사용가능하다.

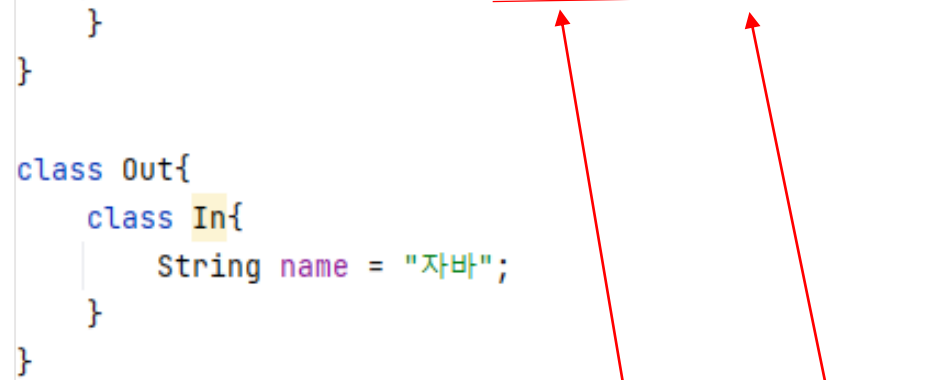
- ✓ 2번 : 아래 중첩 클래스 In 클래스의 name 필드를 출력하는 코드를 Excercise2 클래스의 main 메서드에 작성하세요

[A 방법]

```
public class Excercise2 {  
    public static void main(String[] args) {  
        Out out = new Out();  
        Out.In in = out.new In();  
        System.out.println(in.name);  
    }  
}  
  
class Out{  
    class In{  
        String name = "자바";  
    }  
}
```

[B 방법]

```
public class Excercise2 {  
    public static void main(String[] args) {  
        System.out.println(new Out().new In().name);  
    }  
}  
  
class Out{  
    class In{  
        String name = "자바";  
    }  
}
```



→ 풀이 ;

다른 곳에서 특정 클래스의 내부 클래스에 접근하려면  
외부클래스인 해당 특정클래스의 객체를 먼저 만들고  
그 외부클래스의 객체를 이용해서 내부클래스의 객체를 생성.



- 선생님 클래스 기초문제 : 전화번호부 만들기

AddressBook 클래스를 생성하고, 다음 메소드를 구현하여 Contact를 추가하고, 주소록을 출력하는 프로그램을 작성.

AddressBook(int maxNumContacts):

지정된 최대 연락처 수를 사용하여 AddressBook 객체를 초기화합니다.

void addContact(String name, String phone, String email):

이름, 전화번호 및 이메일을 사용하여 새 Contact 객체를 만들고 AddressBook에 추가합니다.

void printAddressBook():

AddressBook 내의 모든 Contact를 출력합니다.

최대 연락처 수는 5입니다.

출력 예시:

이름: John Smith, 전화번호: 123-456-7890, 이메일: john@example.com

이름: Jane Doe, 전화번호: 234-567-8901, 이메일: jane@example.com

이름: Bob Johnson, 전화번호: 345-678-9012, 이메일: bob@example.com

이름: Emily Davis, 전화번호: 456-789-0123, 이메일: emily@example.com

이름: David Lee, 전화번호: 567-890-1234, 이메일: david@example.com

1단계 : Contact 클래스 생성 : private 변수인 이름, 전화번호, 이메일 필드를 가지며 getter 함수 필요

```
public class Contact {  
    private String name;  
    private String phone;  
    private String email;  
  
    public Contact(String name, String phone, String email) {  
        this.name = name;  
        this.phone = phone;  
        this.email = email;  
    }  
  
    public String getName() { return name; }  
    public String getPhone() { return phone; }  
    public String getEmail() { return email; }  
}
```

2단계 : AddressBook 클래스 : Contact 클래스의 객체인 contacts 배열을 가지며,

배열에 사람을 추가하는 메서드인 addContact, 배열을 출력해주는 메서드인 printAddressBook이 있음

```
public class AddressBook {  
  
    private Contact[] contacts;    → 이름,번호,메일로 이루어진 contacts 배열 생성, 연락처수 numContacts 라는 변수 생성  
    private int numContacts;  
  
    public AddressBook(int maxnumContacts) {    → AddressBook 클래스 생성  
        contacts = new Contact[maxnumContacts];  
        numContacts = 0;  
    }  
    → int 매개변수를 입력받으면 그 크기만큼의 길이를 가지는 contacts 배열이며 numContacts 변수 초기값은 0 명  
  
    public void addContact(String name, String phone, String email){    → 연락처를 추가해주는 메서드 생성 addContact( );  
        if (numContacts < contacts.length){    if 문 사용 : numContacts 변수가 contacts 배열의 길이보다 작으면  
            contacts[numContacts] = new Contact(name, phone, email);    이름,번호,메일을 입력받아서 배열에 넣고  
            numContacts++;    연락처수 변수(numContacts)값을 1 더해줌  
        }  
    }  
  
    public void printAddressBook(){    → 연락처 정보를 출력해주는 메서드 생성 printAddressBook( );  
        for (int i = 0; i < numContacts; i++) {  
            System.out.println("이름:"+contacts[i].getName() + ", 전화번호: "+contacts[i].getPhone()+", 이메일: "+contacts[i].getEmail());  
        }  
    }  
    for: contacts 배열의 0 번째(i) 값부터 numContacts 까지 i 를 1 씩 더해가면서, 배열값을 출력  
  
    public static void main(String[] args) {    → 메인메서드  
        AddressBook book = new AddressBook( maxnumContacts: 5);    AddressBook 클래스의 book 객체 생성하고, 연락처수는 5 임  
  
        book.addContact( name: "John Smith", phone: "123-456-7890", email: "john@ex.com");  
        book.addContact( name: "Jane Doe", phone: "234-567-8901", email: "jane@ex.com");  
        book.addContact( name: "Bob Johnson", phone: "345-678-9012", email: "bob@ex.com");    → addContact 메서드 이용하여 연락처 정보 추가  
        book.addContact( name: "Emily Davis", phone: "456-789-0123", email: "emily@ex.com");  
        book.addContact( name: "David Lee", phone: "567-890-1234", email: "david@ex.com");  
  
        book.printAddressBook();    → printAddressBook 메서드 이용하여 연락처 정보 출력  
    }  
}
```