Come to HUFS Meet the World!

객체지향프로그래밍

24.10.16 실습

- 담당 교수 : 전 병 환 교수님
- 조교 : 윤 종 엄
- 조교 메일 : juyoon@hufs.ac.kr

10월 2일 실습 정답 코드

한국외국어대학교 HANKUK UNIVERSITY OF FOREIGN STUDIES

- 실습 1: 객체

```java
public class Point {
    int x, y;

    public Point() {
        x = 0;
        y = 0;
    }

    public Point(int a, int b) {
        x = a;
        y = b;
    }

    public double distance(Point p) {
        return Math.sqrt(Math.pow(x - p.x, 2) + Math.pow(y - p.y, 2));
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("첫 번째 좌표의 x, y 값을 입력하세요:");
        int x1 = sc.nextInt();
        int y1 = sc.nextInt();
        Point p1 = new Point(x1, y1);

        System.out.println("두 번째 좌표의 x, y 값을 입력하세요:");
        int x2 = sc.nextInt();
        int y2 = sc.nextInt();
        Point p2 = new Point(x2, y2);
        Point p3 = new Point();

        double d1 = p1.distance(p2);
        double d2 = p2.distance(p3);
        double d3 = p1.distance(p3);

        double minDistance = Math.min(d1, Math.min(d2, d3));

        System.out.println("가장 짧은 거리: " + minDistance);

        sc.close();
        }
    }
```

10월 2일 실습 정답 코드

한국외국어대학교 HANKUK UNIVERSITY OF FOREIGN STUDIES

- 실습 2: this

```java
class Car {
    String carName;
    int carYear;
    int maxSpeed;
    String carColor;

    public Car() {
        this.carName = "car";
        this.carYear = 0;
        this.maxSpeed = 0;
        this.carColor = "black";
    }

    public Car(String carName, int maxSpeed) {
        this.carName = carName;
        this.maxSpeed = maxSpeed;
    }

    public String getCarName() {
        return this.carName;
    }

    public int getMaxSpeed() {
        return this.maxSpeed;
    }


    public void setMaxSpeed(int maxSpeed) {
        this.maxSpeed += maxSpeed;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        Car[] cars = new Car[2];

        for (int i = 0; i < cars.length; i++) {
            System.out.println((i + 1) + "번째 자동차 정보를 입력하세요.");

            System.out.print("차 이름: ");
            String name = scanner.next();


            System.out.print("최대 속도: ");
            int speed = scanner.nextInt();

            cars[i] = new Car(name,  speed);
        }

        System.out.println("수정할 자동차 ID(0 또는 1)를 입력하세요:");
        int carId = scanner.nextInt();

        System.out.println("추가할 속도를 입력하세요:");
        int additionalSpeed = scanner.nextInt();

        cars[carId].setMaxSpeed(additionalSpeed);

        System.out.printf("수정된 차량 %s의 새로운 최고 속도는 %d 입니다.\n",
           cars[carId].getCarName(), cars[carId].getMaxSpeed());

        scanner.close();
    }
}
```

10월 2일 실습 정답 코드

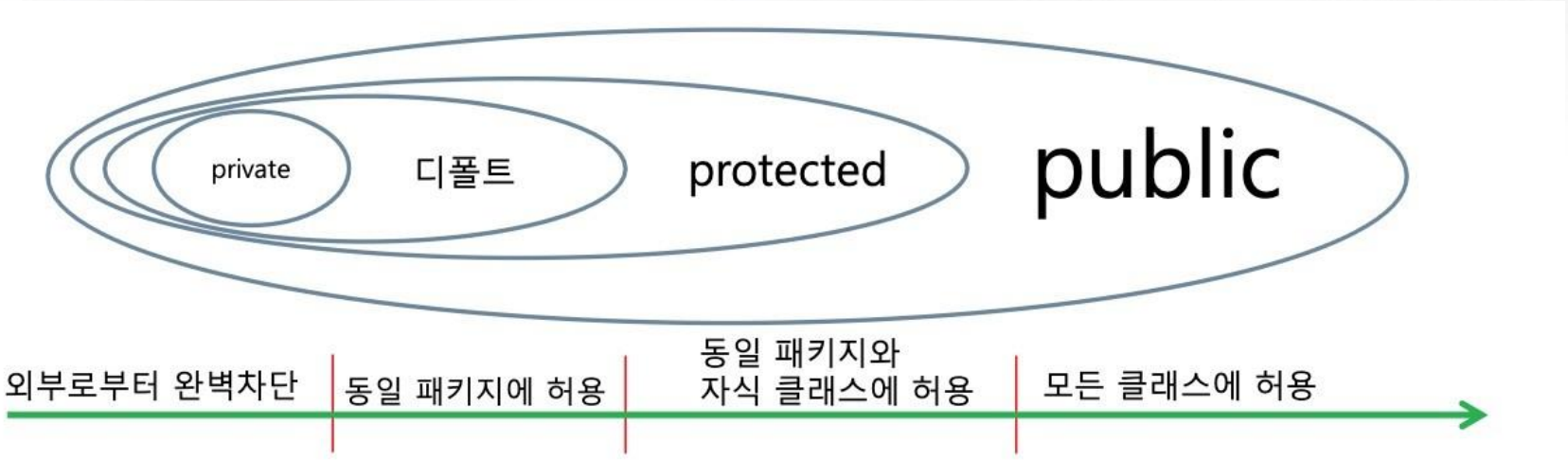
한국외국어대학교 HANKUK UNIVERSITY OF FOREIGN STUDIES

- 실습 3: 메소드 오버로딩

```java
public class Figure {
    String name;
    double radius;
    double length;
    double width;
    double area;

    public Figure(String name, double radius) {
        this.name = name;
        this.radius = radius;
    }

    public Figure(String name, double length, double width) {
        this.name = name;
        this.length = length;
        this.width = width;
    }

    public void getArea(double radius) {
        this.area = 3.14 * radius * radius;
    }

    public void getArea(double length, double width) {
        this.area = length * width;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Figure[] f = new Figure[4];

        for (int i = 0; i < f.length; i++) {
            String name = scanner.next();
            if (name.equals("Circle")) {
                double radius = scanner.nextDouble();
                f[i] = new Figure(name, radius);
                f[i].getArea(radius);
            } else {
                double length = scanner.nextDouble();
                double width = scanner.nextDouble();
                f[i] = new Figure(name, length, width);
                f[i].getArea(length, width);
            }
        }

        for (int i = 0; i < f.length; i++) {
            System.out.printf("%s의 넓이: %.2f\n", f[i].name, f[i].area);

        scanner.close();
        }
    }
}
```

접근 지정자

- 자바의 접근 지정자는 4가지
  - private, protected, public, default(접근지정자 생략)
- 접근 지정자의 목적
  - 클래스나 일부 멤버를 공개하여 다른 클래스에서 접근하도록 허용
  - 객체 지향 언어늬 캡슐화 정책은 멤버를 보호하는 것
    - 접근 지정은 캡슐화에 묶인 보호를 일부 해제할 목적



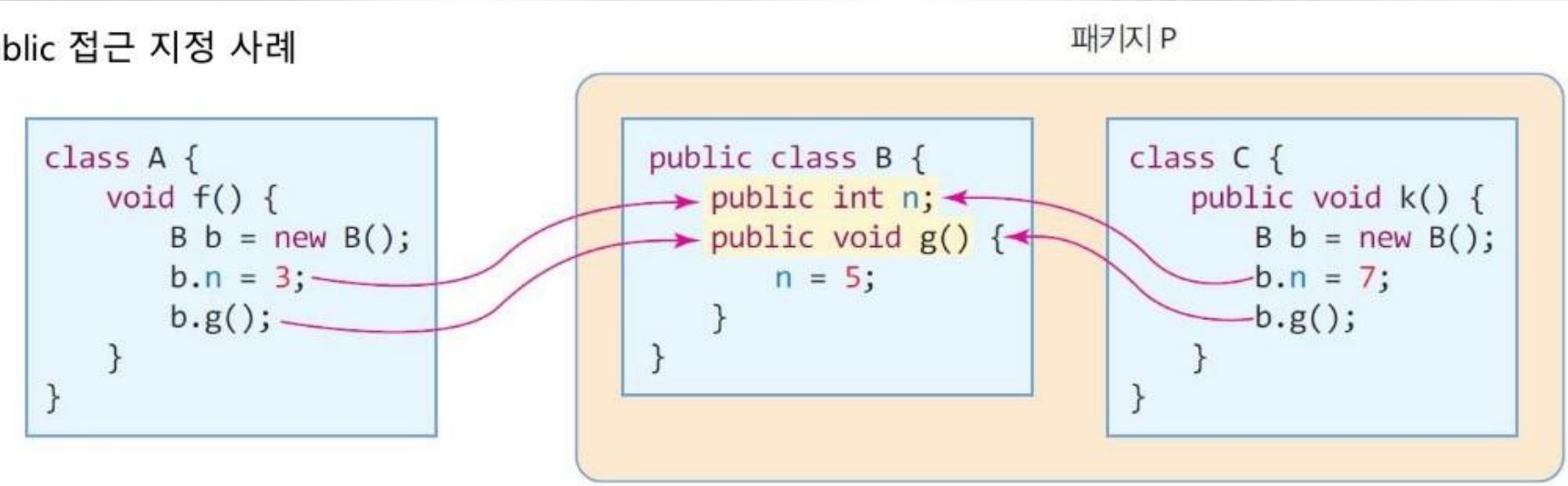
멤버 접근 지정

- public
  - 패키지에 관계없이 모든 클래스에게 접근 허용
- private
  - 동일 클래스 내에서만 접근 허용
  - 상속 받은 서브 클래스에서 접근 불가
- protected
  - 같은 패키지 내의 다른 모든 클래스에게 접근 허용
  - 상속 받은 서브 클래스는 다른 패키지에 있어도 접근 가능
- default(접근지정자 생략)
  - 같은 패키지 내의 다른 클래스에게 접근 허용

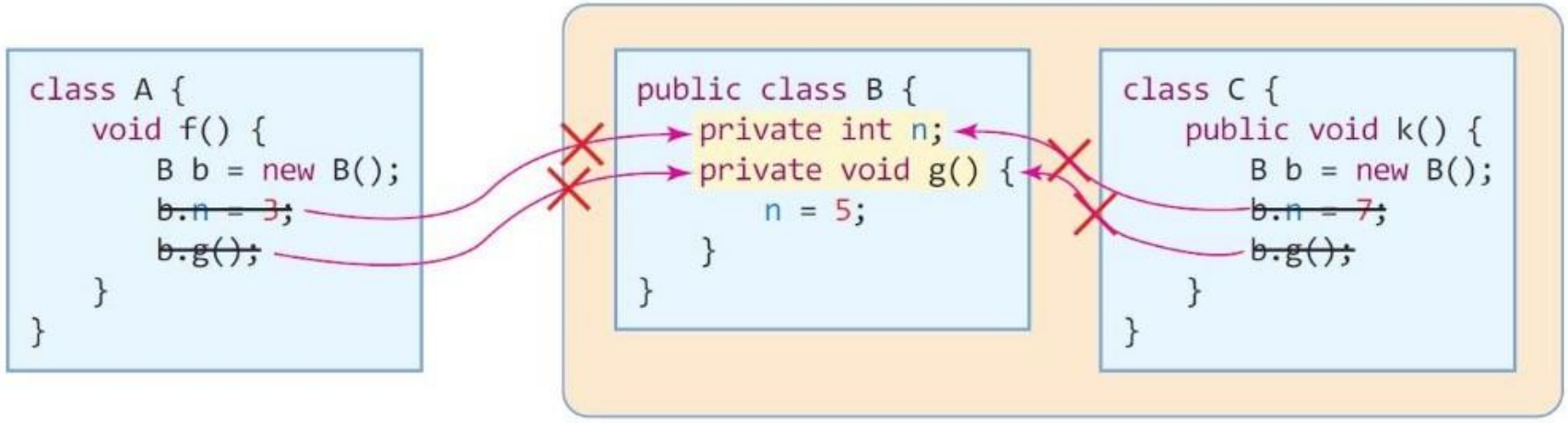
멤버에 접근하는 클래스	멤버의 접근 지정자			
	private	디폴트 접근 지정	protected	public
같은 패키지의 클래스	×	○	○	○
다른 패키지의 클래스	×	×	×	○
접근 가능 영역	클래스 내	동일 패키지 내	동일 패키지와 자식 클래스	모든 클래스

멤버 접근 지정자의 이해

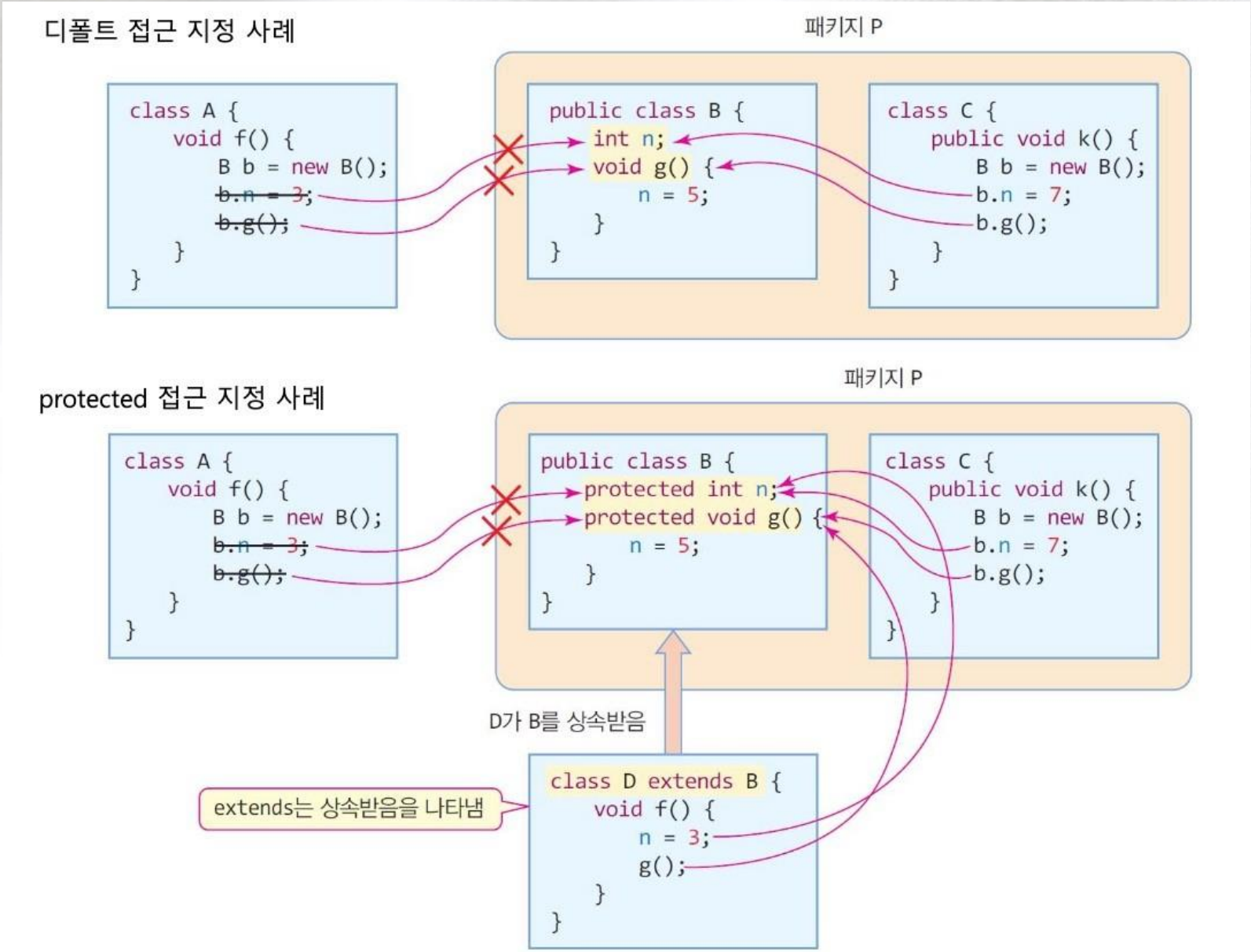
public 접근 지정 사례



private 접근 지정 사례



멤버 접근 지정자의 이해


디폴트 접근 지정 사례
패키지 P
class A {
void f() {
B b = new B();
b.n = 3;
b.g();
}
}
public class B {
int n;
void g() {
n = 5;
}
}
class C {
public void k() {
B b = new B();
b.n = 7;
b.g();
}
}
protected 접근 지정 사례
패키지 P
class A {
void f() {
B b = new B();
b.n = 3;
b.g();
}
}
public class B {
protected int n;
protected void g() {
n = 5;
}
}
class C {
public void k() {
B b = new B();
b.n = 7;
b.g();
}
}
D가 B를 상속받음
extends는 상속받음을 나타냄
class D extends B {
void f() {
n = 3;
g();
}
}

실습 1: 계좌 관리 프로그램 작성

한국외국어대학교 HANKUK UNIVERSITY OF FOREIGN STUDIES

문제 설명:
은행 계좌를 관리하는 Account 클래스를 작성하고자 한다. 아래의 조건을 고려하여 클래스를 작성하시오.

필드 (멤버 변수)
- 계좌 소유자의 이름 (name),
- 계좌 번호 (account_num),
- 잔액 (balance).
- 이 필드들은 외부에서 직접 접근할 수 없음

생성자
- 기본 생성자: 는 이름을 빈 문자열로, 계좌 번호와 잔액을 0으로 설정
- 인자가 있는 생성자: 이름, 계좌 번호, 잔액을 매개변수로 받아 초기화할 수 있는 생성자를 추가로 작성

메서드
계좌 소유자의 이름을 반환하는 getName() 메서드
계좌 번호를 반환하는 getAccNo() 메서드.잔액을 반환하는 getBalance() 메서드
입출금을 처리하는 transaction(int value) 메서드 (잔액에 매개변수로 받은 value를 더함, value는 +, -값을 가질 수 있음)

요구 사항
- 접근 제어자를 활용하여 각 필드를 적절히 보호하세요.
- transaction(int value) 메서드를 통해 잔액을 수정할 수 있도록 하고, 잔액이 음수가 될 경우 경고 메시지를 출력
- 두 개의 Account 객체를 생성하고, 입출금을 수행한 후 각 계좌의 잔액을 출력하는 메인 메서드를 작성

아래 코드를 기반으로 동작하도록 작성

한국외국어대학교 HANKUK UNIVERSITY OF FOREIGN STUDIES

```java
class Account {

    //멤버 변수 선언

    //기본 생성자 작성

    //인자가 있는 생성자 작성


    //메서드
    public String getName() {

    }

    public int getAccNo() {

    }

    public int getBalance() {

    }

    public void transaction(int balance) {

    }


    //클라이언트 코드
    public static void main(String[] args) {
        Account account1 = new Account("A", 0, 100000);
        Account account2 = new Account("B", 0, 100000);

        account1.transaction(-150000);
        account1.transaction(+5000);
        account1.transaction(-2000);
        account1.transaction(-30000);

        account2.transaction(+5000);
        account2.transaction(+5000);
        account2.transaction(+5000);

        System.out.println(account1.getBalance());
        System.out.println(account2.getBalance());

    }
}
```

```
잔액이 부족합니다.
73000
115000
```

static 멤버와 non-static 멤버

- static 멤버란?
  - 객체마다 생기는 것이 아님
  - 클래스 당 하나만 생성됨 (클래스 멤버라고도 부름)
  - 객체를 생성하지 않고 사용가능

	non-static 멤버	static 멤버

선언	class Sample {
	  int n;
	  void g() {…}
	}
		class Sample {
		  static int m;
		  static void f() {…}
		}

공간적 특성	멤버는 객체마다 별도 존재 • 인스턴스 멤버라고 부름	멤버는 클래스당 하나 생성 • 멤버는 객체 내부가 아닌 별도의 공간(클래스 코드가 적재되는 메모리)에 생성 • 클래스 멤버라고 부름
시간적 특성	객체 생성 시에 멤버 생성됨 • 객체가 생길 때 멤버도 생성 • 객체 생성 후 멤버 사용 가능 • 객체가 사라지면 멤버도 사라짐	클래스 로딩 시에 멤버 생성 • 객체가 생기기 전에 이미 생성 • 객체가 생기기 전에도 사용 가능 • 객체가 사라져도 멤버는 사라지지 않음 • 멤버는 프로그램이 종료될 때 사라짐
공유의 특성	공유되지 않음 • 멤버는 객체 내에 각각 공간 유지	동일한 클래스의 모든 객체들에 의해 공유됨

static의 활용

1) 전역 변수와 전역 함수를 만들 때

2) 공유 멤버를 작성할 때

static 메소드의 제약 조건

한국외국어대학교 HANKUK UNIVERSITY OF FOREIGN STUDIES

- static 메소드는 non-static 멤버 접근할 수 없음
  - 객체가 생성되지 않은 상황에서도 static 메소드는 실행될 수 있기 때문에, non-static 메소드와 필드 사용 불가
  - 반대로, non-static 메소드는 static 멤버 사용 가능

```
class StaticMethod {
    int n;
    void f1(int x) {n = x;} // 정상
    void f2(int x) {m = x;} // 정상

    static int m;
    static void s1(int x) {n = x;} // 컴파일 오류. static 메소드는 non-static 필드 사용 불가
    static void s2(int x) {f1(3);} // 컴파일 오류. static 메소드는 non-static 메소드 사용 불가

    static void s3(int x) {m = x;} // 정상. static 메소드는 static 필드 사용 가능
    static void s4(int x) {s3(3);} // 정상. static 메소드는 static 메소드 호출 가능
}
```

오류

오류

static 메소드의 제약 조건

- static 메소드는 this 사용불가
  - static 메소드는 객체가 생성되지 않은 상황에서도 호출이 가능하므로, 현재 객체를 가리키는 this 레퍼런스 사용할 수 없음

```
class StaticAndThis {
    int n;
    static int m;
    void f1(int x) {this.n = x;}
    void f2(int x) {this.m = x;} // non-static 메소드에서는 static 멤버 접근 가능
    static void s1(int x) {this.n = x;} // 컴파일 오류. static 메소드는 this 사용 불가
    static void s2(int x) {this.m = x;} // 컴파일 오류. static 메소드는 this 사용 불가
}
```

오류

오류

final 필드

- final 필드, 상수 선언
  - 상수를 선언할 때 사용

```
class SharedClass {
        public static final double PI = 3.14;
}
```

  - 상수 필드는 선언 시에 초기 값을 지정하여야 한다.
  - 상수 필드는 실행 중에 값을 변경할 수 없다.


```
public class FinalFieldClass {
        final int ROWS = 10; // 상수 정의, 이때 초기 값(10)을 반드시 설정

        void f() {
                int [] intArray = new int [ROWS]; // 상수 활용
                ROWS = 30; // 컴파일 오류 발생, final 필드 값을 변경할 수 없다.
        }
}
```

실습 2: 계좌와 연동된 신용카드 클래스 작성

한국외국어대학교 HANKUK UNIVERSITY OF FOREIGN STUDIES

문제 설명:
은행 계좌와 연동된 여러 개의 신용카드를 관리하는 Card 클래스를 작성하려고 한다. 법인카드의 경우 하나의 은행계좌에 여러 개의 카드가 연동되어 사용할 수 있다. 이때, static 변수를 이용하여 하나의 은행 계좌가 여러 카드에 공유되는 방식으로 구현하고자 한다. (문제 1에서 작성한 Account 클래스를 은행계좌로 활용)

요구 사항:
- Card 클래스는 Account 객체를 static 변수로 선언하여 모든 카드 객체가 같은 계좌를 공유하도록 함
- Card 클래스는 각 카드의 이름과 연동할 은행계좌를 인자를 받아 초기화할 수 있어야 함
- 각 카드 객체는 은행계좌의 잔액에 대하여 입출금(transaction)과 조회(inquiry)를 할 수 있어야 함
- 한 계좌에 연동된 여러 개의 카드가 존재할 수 있음.

클래스 요구 사항:
- static 변수 Account account를 통해 계좌를 공유
- 각 카드의 이름(cardName)을 저장할 수 있는 변수를 선언
- 은행 계좌와 카드의 이름을 초기화하는 생성자 작성
- transaction(int value) 메서드를 통해 계좌에 입출금 처리
- inquiry() 메서드를 통해 현재 계좌의 잔액을 확인
- getCardName() 메서드를 통해 카드의 이름을 반환
- getAccountName() 메서드를 통해 현재 연동된 계좌의 소유자 이름 반환

아래 코드를 기반으로 동작하도록 작성

한국외국어대학교 HANKUK UNIVERSITY OF FOREIGN STUDIES

```java
public class Card {

    static Account account;
    private String cardName;

    Card(Account a){
        this(a, "");
    }

    Card(Account a, String name){

    }

    public String getCardName() {

    }

    public void transaction(int value) {

        //account의 함수를 활용
    }

    public int inquiry() {
        //account의 함수를 활용
    }

    public String getAccountName() {
        //account의 함수를 활용
    }
```

```java
    public static void main(String[] args) {
        // TODO Auto-generated method stub

        //법인
        Account accHUFS = new Account("HUFS", 0, 100000);
        Account accYonsei = new Account("Yonsei", 0, 100000);

        //HUFS계좌에 연동된 여러 개 카드
        Card a = new Card(accHUFS, "a");
        Card b = new Card(accHUFS, "b");
        Card c = new Card(accHUFS, "c");

        a.transaction(10000);
        b.transaction(-3000);
        c.transaction(-5000);
        a.transaction(20000);

        System.out.println(a.getCardName()+"님 카드, " + a.getAccountName() + "의 현재 잔액: " + a.inquiry());
        System.out.println(b.getCardName()+"님 카드, " + b.getAccountName() + "의 현재 잔액: " + b.inquiry());
        System.out.println(c.getCardName()+"님 카드, " + c.getAccountName() + "의 현재 잔액: " + c.inquiry());

        //Yonsei 계좌에 연동된 여러 개 카드
        Card d = new Card(accYonsei, "d");
        Card e = new Card(accYonsei, "e");
        Card f = new Card(accYonsei, "f");

        d.transaction(20000);
        e.transaction(-5000);
        f.transaction(-6000);
        d.transaction(10000);

        System.out.println(d.getCardName()+"님 카드, " + d.getAccountName() + "의 현재 잔액: " + d.inquiry());
        System.out.println(e.getCardName()+"님 카드, " + e.getAccountName() + "의 현재 잔액: " + e.inquiry());
        System.out.println(f.getCardName()+"님 카드, " + f.getAccountName() + "의 현재 잔액: " + f.inquiry());

    }

}
```

```
a님 카드, HUFS의 현재 잔액: 122000
b님 카드, HUFS의 현재 잔액: 122000
c님 카드, HUFS의 현재 잔액: 122000
d님 카드, Yonsei의 현재 잔액: 119000
e님 카드, Yonsei의 현재 잔액: 119000
f님 카드, Yonsei의 현재 잔액: 119000
```