

6.0. 클래스의 발생배경

- 자동차 경주 게임을 만드는 것을 가정.
- 차량별 표현속성을 기존의 배열로 프로그램하기가 어려움

기존의 방법으로는 현실세계를 프로그램 하기가 어려움



speed [] = {100,90,80,150,... }
color [] = {red,yellow,blue,... }

속성	1	2	3	4	...	100
현재속도	100	90	80	150	...	120
색상	빨강	노랑	빨강	파랑	...	블랙
제조사	현대	기아	벤츠	쌍용	...	벤츠
모델	H10	K5	C20	SS1	...	C23
운전자	jin	kim	park	Jim	...	park
순간가속	5	3	7	1	...	4
현재순위	1	8	52	3	...	90
현재연료



1. 배열변수를 통해서 속성을 정의하는 것이 아닌 차ID를 통해서 속성을 입력하고, 갱신할 수는 없을까? (차량 ID 배열만으로 처리가능?) (횡이 아닌 열로써 관리가능?)
2. 차량 ID만 알면 그 차의 속성을 바로 확인할 수 있다면 좋겠는데...
3. 운전자의 액셀입력 동작으로 인해 현재 속도가 변하는데. 이를 처리할 수 있는 방법은?
4. 참가차량이 늘어나더라도 속성을 표시하는 배열변수 처리가 쉬웠으면?
5. 차량의 속성이 늘고, 줄어도 프로그램상 처리가 쉬웠으면...

6.1. 객체 및 객체지향 프로그램 ↔ 절차지향

- JAVA는 객체지향 프로그래밍을 지원하는 프로그램 언어임.
- 객체지향 프로그램이란 객체(주어)가 동작(서술어)하는 형태로 표시하는 것임(현실세계와 유사)

객체의 개념

객체

- 물리적으로 존재하거나 (사람, 학원, 책, 자동차 등) 추상적인 것 (날짜) 에서 자신의 속성과 동작을 가지는 것을 객체라 한다.
- 객체는 필드(속성)과 메소드(동작)로 모델링이 가능함.

현실 세계



[사람]

속성 : 이름, 나이
동작 : 걷다, 먹다



[컴퓨터]

속성 : 모델, 연식
동작 : 계산하다, 부팅하다

S/W 모델링

[사람 객체]

```
String name; int age;  
void walking() {...}  
void eating() {...}
```

[컴퓨터 객체]

```
String model; date year;  
int adding() {...}  
void booting() {...}
```

객체간 상호작용

자바는 객체간 서로 메소드를 호출하고, 결과를 리턴받음



[사람 객체]

계산 요청
(사람이 제공하는 데이터)
→
`person1.request(computer1)`

←
결과 회신
(컴퓨터가 제공하는 데이터)



[컴퓨터 객체]

```
int result = computer1.adding(10,20)
```

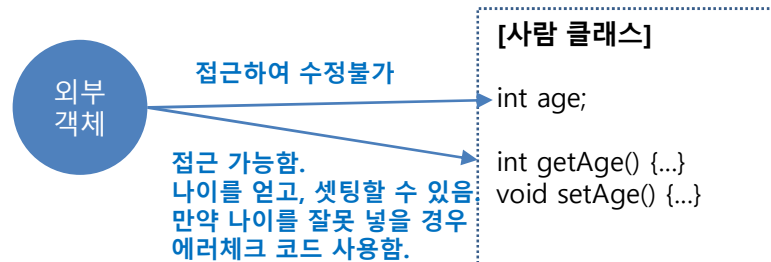
6.2. 객체지향 프로그램의 특징

- JAVA는 캡슐화, 상속, 다형성이란 주요한 특징이 있음.

객체지향 프로그램 특징

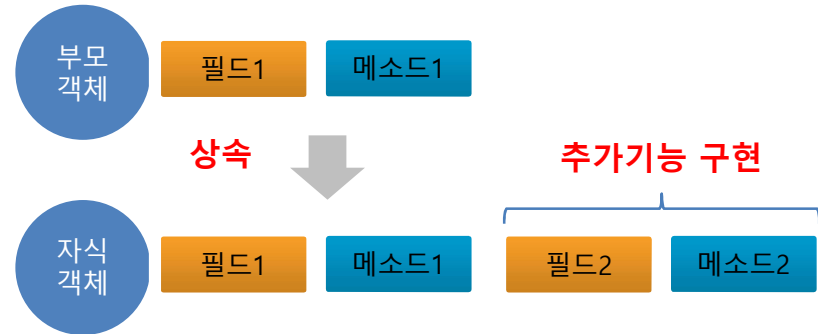
특징 1 캡슐화

- * 객체의 변수(필드) 및 행위(메서드)를 하나로 묶어내고, 실제 구현내용은 외부에 감추는 것을 의미함.
- * 외부의 잘못된 사용으로 객체가 손상되지 않도록 함
- * 캡슐화 된 멤버를 노출여부를 접근제한으로 관리함.



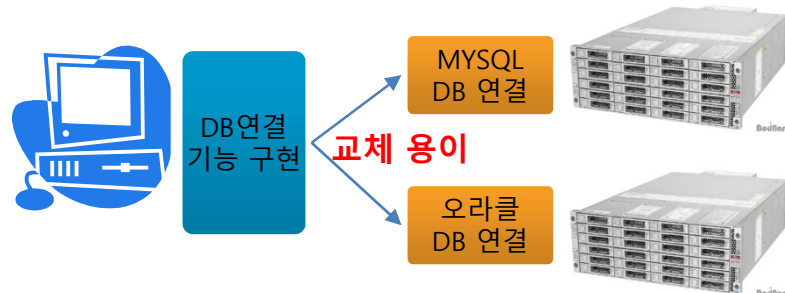
특징 2 상속

- * 부모가 가지고 있는 기능을 자식에게 물려줌
- * 코드 재활용 가능, 유지보수 편의성 제공



특징 3 다형성

- * 같은 타입이지만 실행결과가 다양한 객체를 대입할 수 있는 성질을 의미함.
- * 객체를 부품화할 수 있고, 유지보수가 용이함.



6.3. 객체와 클래스간의 관계

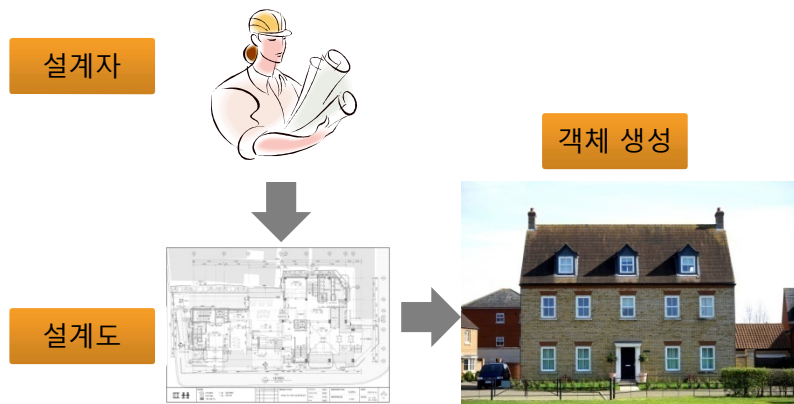
- 클래스는 객체를 생성하기 위한 설계도와 비슷한 의미임.
- 클래스에는 객체를 표현하는 필드와 객체의 동작을 표현하는 메서드로 이루어짐

객체와 클래스간의 관계

❖ 객체와 클래스간의 관계

- 현실세계 : 설계도를 통해서 자동차, 건물, 기계 등의 객체가 생성됨.
- JAVA : 클래스는 객체의 설계도임.
- 클래스에는 객체를 표현하기 위한 필드와 동작을 정의하는 메소드가 정의되어 있음
- 클래스로부터 만들어진 객체를 해당클래스의 인스턴스라고 함.
- 하나의 클래스로부터는 여러 개의 인스턴스를 만들 수 있다.

현실세계



JAVA



6.4. 객체의 생성

- 객체의 생성은 new 연산자를 통해서 객체가 생성된다.
- new 연산자로 생성된 객체는 참조변수로 관리된다.

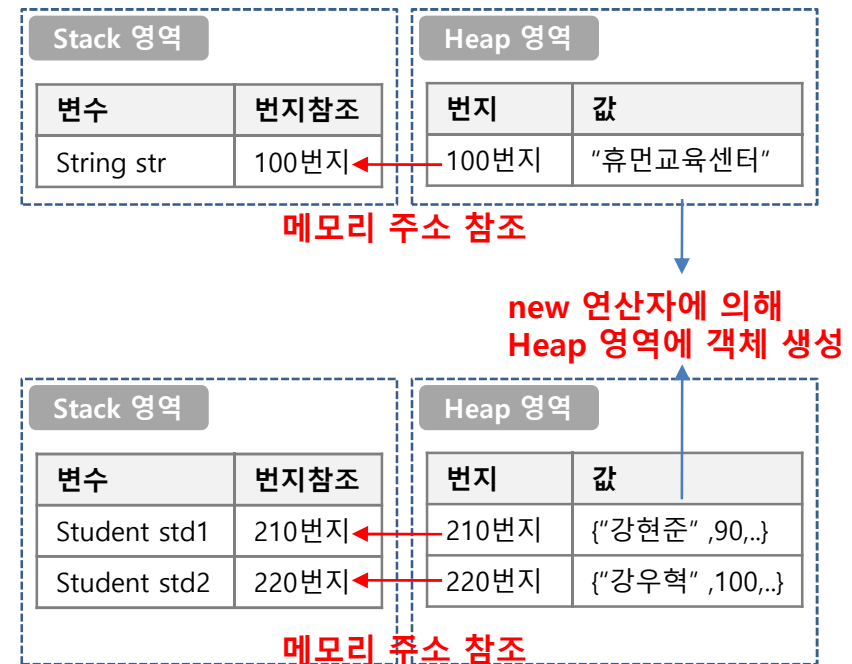
객체의 생성

문법

```
클래스 인스턴스변수 = new 클래스( );  
// (객체)
```

```
ex1) String str = new String("휴먼교육센터");
```

```
ex2)  
Student std1 = new Student( );  
Student std2 = new Student( );
```



6.5. 클래스의 구성멤버

- 클래스의 구성멤버는 멤버변수(필드), 생성자, 메서드로 구성됨
- 각 구성멤버는 객체를 구체화하는데 사용함

클래스의 구성멤버

구성멤버

```
public class Human {  
    int field;           // 멤버변수 (필드)  
    Human () { ..... } // 생성자  
    void method () {....} // 메서드  
}
```

멤버변수(필드)

Exam_02

- ❖ 객체의 고유데이터를 의미함.
- ❖ 객체가 가져야 할 제약사항
- ❖ 객체의 현재상태의 데이터
 . 생성자에 의해 초기화, 메서드에 의해 업데이트 되는 데이터

Ex. 자동차 클래스

```
- String company = "현대자동차" // 객체 고유데이터  
- String model = "그랜저"       // 객체 고유데이터  
- int maxSpeed = 300             // 객체의 제약 사항  
- int currentSpeed = 80          // 객체의 현재 상태
```

생성자

Exam_03

- ❖ new 연산에 의해 초기화되면서 객체가 생성됨 .
- ❖ **클래스명과 동일함.**
- ❖ 모든 클래스는 반드시 존재하며 한 개 이상이 존재함

Ex. 자동차 클래스

```
class Car {  
    Car () {....} // 생성자  
    Car car1 = new Car(); // car1이란 객체는 생성자를 통해 생성됨  
    Car car2 = new Car(); // car2이란 객체는 생성자를 통해 생성됨  
}
```

메서드

Exam_04

- ❖ 생성된 객체의 동작(기능)을 말함.

Ex. 자동차 클래스

```
class Car {  
    Car () {....} // 생성자  
    void speedUp() { currentSpeed++; } // 현재의 속도를 올림  
    Car car1 = new Car(); // car1이란 객체는 생성자를 통해 생성됨  
    car1.speedUp(); // currentSpeed가 80 → 81로 증가함.  
}
```

6.5.1. 클래스의 구성멤버 (생성자)

- 생성자는 객체를 생성하는 것으로 초기값을 생성
- 클래스명과 동일한 함수가 바로 생성자임.

생성자

기본생성자

Exam_01

- ❖ 생성인자를 전달하지 않는 생성자를 의미하며, 생략 가능함. (모든 생성자는 1개 이상을 가져야 함)

Car.java

```
class Car {  
    int speed = 100;  
    void driving () {...}  
}
```

* 생성자가 없는 클래스

Car.class

```
class Car {  
    int speed = 100;  
    public Car() { }  
    void driving () {...}  
}
```

* 컴파일시 자동생성됨

필드초기화

Exam_05

- ❖ 생성자 호출시 인자를 전달하면서 필드를 초기화 함.

Car.java

```
class Car {  
    int speed;  
    public Car (speed) {  
        this.speed = speed;  
    }  
  
    Car car1 = new Car(100);  
    Car car2 = new Car(120);  
}
```

- car1은 speed가 100으로 객체 생성
- car2은 speed가 120으로 객체 생성

생성자 오버로딩

Exam_06

- ❖ 오버로딩은 개발자가 함수를 재정의하는 것을 의미함
- ❖ 생성자 오버로딩은 외부에서 어떠한 인자로 생성자를 만들지 모르기 때문에 다양한 생성자를 준비하는 것이 좋음

```
class Car {  
    int speed;  
    String color;
```

```
    public Car () {
```

```
        public Car (speed) {  
            this.speed = speed;  
        }
```

```
        public Car (speed, color) {  
            this.speed = speed;  
            this.color = color;  
        }
```

```
        Car car3 = new Car(120, "RED");  
        Car car2 = new Car(100);  
        Car car1 = new Car();  
    }
```

인자의 개수에 맞추어
생성자를 생성함.

6.5.1. 클래스의 구성멤버 (생성자)

- 생성자는 객체를 생성하는 것으로 초기값을 생성
- 생성자를 생성시에는 멤버변수를 전체 초기화하는 것이 좋음

다른 생성자 호출

생성자 오버로딩의 문제

- ❖ 생성자 오버로딩시 멤버변수를 모두 초기화해야 하는데 중복코드가 많이 발생함.

```
class Car {  
    int speed;  
    String color;  
  
    public Car () {  
        this.speed = 100;  
        this.color = "GREEN";  
    }  
  
    public Car (speed) {  
        this.speed = speed;  
        this.color = "BLUE";  
    }  
  
    public Car (speed, color) {  
        this.speed = speed;  
        this.color = color;  
    }  
  
    Car car3 = new Car(120, "RED");  
    Car car2 = new Car(100);  
    Car car1 = new Car();  
}
```

중복코드 발생

중복코드 발생

중복코드 발생

this의 사용

Exam_07

- ❖ this는 자신의 객체를 의미하는 것으로 this를 사용하면 코드를 간결화 할 수 있다.

```
class Car {  
    int speed;  
    String color;  
  
    public Car () {  
        this (100, "GREEN");  
    }  
  
    public Car (speed) {  
        this(speed, "BLUE");  
    }  
  
    public Car (speed, color) {  
        this.speed = speed;  
        this.color = color;  
    }  
  
    Car car3 = new Car(120, "RED");  
    Car car2 = new Car(100);  
    Car car1 = new Car();  
}
```

인자에 맞추어
재호출하여
생성자 초기화 함.

6.5.2. 클래스의 구성멤버 (메서드)

- 메서드는 객체의 동작(기능)을 정의함.
- 메서드는 객체의 호출에 의해 동작하며, 일부는 return값을 회신하여 객체의 요구에 부응함

메서드

메서드 문법

Exam_08

- ❖ 메서드와 클래스명과 다른 함수부분을 메서드라 함
- ❖ 메서드는 객체의 동작을 정의하는 것으로써 객체의 상태를 표현하거나 객체의 현재상태를 변경하는 역할을 함

Ex. 자동차 클래스

```
class Car {  
    int currentSpeed;  
  
    public Car (currentSpeed) {    // 클래스 명과 동일. → 생성자임  
        this.currentSpeed = currentSpeed  
    }  
    public void speedUp() {  
        currentSpeed++;           // 현재의 속도를 1 증가함  
    }  
    Car car1 = new Car(80);  
    Car car2 = new Car(120);  
  
    car1.speedUp();               // 현재의 속도를 1 증가함  
    // car1의 currentSpeed가 80 → 81로 증가함.  
}
```

리턴 타입

Exam_08

- ❖ 메서드는 리턴타입을 갖는다.
리턴타입은 데이터 타입과 비슷한 맥락을 갖으며 메서드 명 앞에 표시한다

리턴타입	예시
void	리턴타입 없음
기본타입	- int : 정수형 데이터로 리턴 - double : 실수형 데이터로 리턴 - boolean : 논리형 데이터로 리턴
객체타입	- String : 문자열형 객체로 리턴 - Human : Human 객체로 리턴 * 객체타입에 맞추어 리턴해야 함

Ex. 자동차 클래스의 메서드

```
public void speedUp() {  
    currentSpeed++;           // 현재의 속도를 1 증가함  
}  
  
public int getCurrentSpeed() {  
    return currentSpeed;  
    // 현재의 속도를 정수형 데이터로 리턴함  
}  
  
car1.sppedUp();               // 리턴값이 없는 메서드  
car1.Speed = car1.getCurrentSpeed();  
// 현재의 속도를 car1_Speed 변수에 저장함.
```

6.5.2. 클래스의 구성멤버 (메서드)

- 메소드 이름과 인자의 개수 및 타입이 동일해야만 호출이 됨
- 여러 상황을 대응하기 위해 메서드도 오버로딩이 가능함.

메서드 호출

메서드 오버로딩

Exam_09

- ❖ 클래스내 같은 이름의 메서드를 여러 개 선언하는 것을 메서드 오버로딩이라 함
- ❖ 인자의 개수 및 인자의 타입에 따라 여러 개 선언 가능




```
class Calc{
    public int add (int x, int y) {
        return (x+y);
    }
    public double add (int x, double y) {
        return (x+y);
    }
    public int add (int x) {
        return (x+10);
    }
}

Calc calc = new Calc()
System.out.println( calc.add(10) );           // 20 이 출력됨
System.out.println( calc.add(10, 10.3) );     // 20.3이 출력됨
System.out.println( calc.add(10, 100) );      // 110이 출력됨
```

인자의
개수 및 타입에
맞추어
메서드 호출함

[6-3] 다음과 같은 멤버변수를 갖는 Student클래스를 정의하시오.

타입	변수명	설명
String	name	학생이름
int	ban	반
int	no	번호
int	kor	국어점수
int	eng	영어점수
int	math	수학점수

 Exercise_01
 >  Student.java
 >  StudentExam.java

[6-4] 문제6-3에서 정의한 Student클래스에 다음과 같이 정의된 두 개의 메서드 getTotal()과 getAverage()를 추가하시오.

1. 메서드명 : getTotal
 기능 : 국어(kor), 영어(eng), 수학(math)의 점수를 모두 더해서 반환한다.
 반환타입 : int
 매개변수 : 없음
2. 메서드명 : getAverage
 기능 : 총점(국어점수+영어점수+수학점수)을 과목수로 나눈 평균을 구한다.
 소수점 둘째자리에서 반올림할 것.
 반환타입 : float
 매개변수 : 없음

[연습문제]/ch6/Exercise6_4.java

```

class Exercise6_4 {
    public static void main(String args[]) {
        Student s = new Student();
        s.name = "홍길동";
        s.ban = 1;
        s.no = 1;
        s.kor = 100;
        s.eng = 60;
        s.math = 76;

        System.out.println("이름:"+s.name);
        System.out.println("총점:"+s.getTotal());
        System.out.println("평균:"+s.getAverage());
    }
}

class Student {
    /*
    (1) 알맞은 코드를 넣어 완성하시오.
    */
}
  
```

생성자를 인자로 주어서 값을 만드는 것을 권함.

getTotal()/ getAvg()
메서드로 int/double형으로 반환

[실행결과]

이름:홍길동
 총점:236
 평균:78.7

[6-5] 다음과 같은 실행결과를 얻도록 Student 클래스에 생성자와 info()를 추가하시오.

【연습문제】/ch6/Exercise6_5.java

```
class Exercise6_5 {
    public static void main(String args[]) {
        Student s = new Student("홍길동",1,1,100,60,76);

        System.out.println(s.info());
    }
}

class Student {
    /*
    (1) 알맞은 코드를 넣어 완성하시오.
    */
}
```

【실행결과】

홍길동,1,1,100,60,76,236,78.7

대각선 길이
= $\sqrt{(5-1)*(5-1) + (4-1)*(4-1)}$

1,1

[6-7] 문제6-6에서 작성한 클래스에서 getDistance()를 MyPoint 클래스의 인스턴스에서 드로 정의하시오.

【연습문제】/ch6/Exercise6_7.java

```
class MyPoint {
    int x;
    int y;

    MyPoint(int x, int y) {
        this.x = x;
        this.y = y;
    }

    /*
    (1) 인스턴스에서 getDistance를 작성하시오.
    */
}

class Exercise6_7 {
    public static void main(String args[]) {
        MyPoint p = new MyPoint(1,1);

        // p와 (2,2)의 거리를 구한다.
        System.out.println(p.getDistance(2,2));
    }
}
```

5,4

System.out.println (Math.sqrt(4));

【실행결과】

1.4142135623730951

- . 객체 : new를 통해서 생성된 인스턴스, 객체지향프로그램의 대상
- . 클래스 : 객체를 설계한 설계도
- . 인스턴스 : new 키워드를 통해 클래스를 메모리에 생성한 상태
- . 멤버변수 : 클래스의 속성, 특성
- . 메서드 : 멤버변수를 이용하여 클래스의 기능을 구현한 함수
- . 참조변수 : 메모리에 생성된 인스턴스를 가리키는 변수
- . 참조값 : 생성된 인스턴스의 메모리 주소값

. 키(height)가 180이고,
몸무게(weight)가 78kg인
남성(gender)이 있음.
이름(name)은 tomas이고,
나이(age)는 37세.

```
Class Person {  
    int height;  
    int weight;  
    String gender;  
    String Name;  
    int age;  
}
```

. 음식점에 배달주문이 들어왔고,
주문내용을 출력하는 프로그램
==> 클래스 : Delivery

- . 주문접수번호(receiptNumber) : 202011020003
- . 주문핸드폰번호(콜) : 01023450001
- . 주문집주소(address) : 서울시 강남구 역삼동 111-333
- . 주문날짜(receiptDate) : 20201102
- . 주문시간(receiptTime) : 130258
- . 주문가격(price) : 35000
- . 메뉴번호(menuNumber) : 0003

추가 : 객체간의 협력

고객의 요구사항은 아래와 같습니다.

저는 XX운수의 관리부장입니다.
저희 교통수단은 버스, 지하철, 택시가 있고,
버스는 1~3번까지 3대,
지하철은 1~2호선이 있는데. 호선별 각각 1대가 있습니다.
택시는 3대가 있는데. 1,2,3 호기라고 부릅니다.

우리는 학생만을 대상으로 운송사업을 하고 있으며,
우리가 원하는 것은 각 교통수단별로 실시간 승객수와 실시간 수입의
자료를 온라인으로 얻고 싶습니다.
참고적으로 버스는 1200원, 지하철은 1450원, 택시는 5000원의 요금이 필요합니다.

그리고, 무임승차로 인한 손실이 크기 때문에 학생이 돈이 없을 때는
교통수단을 이용할 수 없도록 막아주셨으면 합니다.

우선 위의 내용을 정리하면

1. 교통수단은 버스, 지하철, 택시가 있고.
2. 학생만 이용합니다.
3. 버스는 3대가 있고, 지하철은 호선별로 구분되고, 택시는 호기로 구분됩니다.
4. 학생이 교통수단에 탑승할 때는 돈을 지불해야 하고.
5. 돈이 부족할때는 탑승을 거부할 수 있는 기능 필요
6. 시스템을 통해서 얻고자 하는 데이터는 교통수단별 실시간 승객과 실시간 수입입니다.

위의 내용을 기반으로 모델링을 한다면

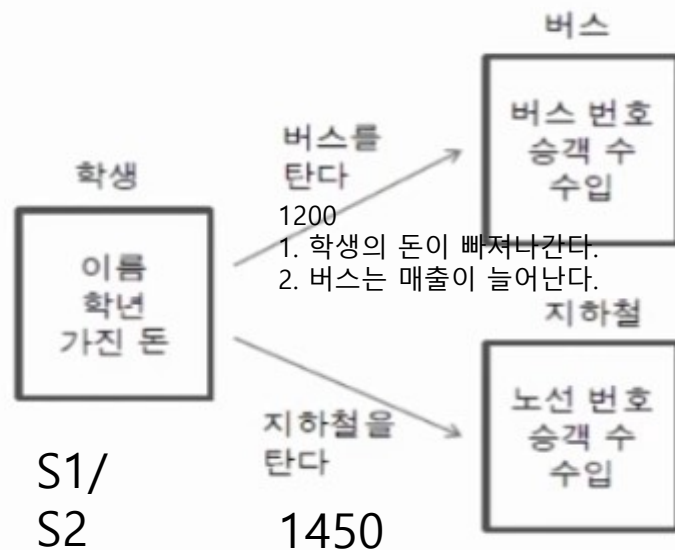
1. 주어가 되는 부분을 뽑는다면. 학생, 교통수단, 버스, 지하철, 택시.
2. 학생의 객체별 구분을 위해서는 이름.
3. 버스의 객체별 구분을 위해서는 버스번호
4. 지하철의 객체별 구분을 위해서는 노선번호
5. 택시의 객체별 구분을 위해서는 택시호기
6. 버스의 경우 버스번호별 승객수와 수입 데이터가 관리되어야 하고.
7. 지하철의 경우 지하철 노선번호별 승객수와 수입데이터가 관리되어야 하고.
8. 택시의 경우 택시호기별의 수입데이터가 관리되어야 함.(택시의 경우 승객수는 필요없음)

13. 객체 간의 협력 (collabration)

객체 지향 프로그래밍에서의 협력

- 객체 지향 프로그램에서 객체 간에는 협력이 이루어짐
- 협력을 위해서는 필요한 메시지를 전송하고 이를 처리함
- 매개 변수로 객체가 전달되는 경우가 발생
- 객체 협력의 예

현실세계 모델링 가능



sw1/
sw2

```
Problems Javadoc Declaration Console
<terminated> TakeTransTest (1) [Java Application] C:\Users\WAd
Edward님의 남은 돈은 10000원 입니다
잘 간다 운수택시 수입은 10000원입니다.
```

6.6.1. 정적 멤버 / static

- 정적(static)이란 키워드는
- 객체와 상관없이 클래스 차

정적 멤버

정적 멤버변수 사용사례

- ❖ 사번을 부여하는 프로그램을 작성할 때 사번이라 함은 입사순으로 사번을 갖게 될 것인데. 이 때 정적멤버로

사원

```
public class Employee {
    int employee_id; // 사번
    static int serialNum = 1000;

    public Employee() {
        serialNum++;
        employee_id = serialNum;
    }

    Employee p1 = new Employee();
    Employee p2 = new Employee();
    Employee p3 = new Employee();

    .....
    Employee p9 = new Employee();
}
```

클래스

Static 은 클래스 차원의 변수.

1번객체
- 멤버변수가 초기화

4번객체
- 멤버변수가 초기화

2번객체
- 멤버변수가 초기화

3번객체
- 멤버변수가 초기화

객체와는 무관하게 정의되는 멤버변수임.

- ❖ 이의 코드에서는 Employee 객체가 생성될 때마다 serialNum은 증가된 형태로 저장하게 된다.
- ❖ static으로 정의되는 정적 멤버는 객체가 생성될 때가 아닌 클래스가 로딩되는 시점에 먼저 메모리에 반영이 됨.
- ❖ 클래스 차원의 멤버변수이므로 객체외부에서 접근할 때는 "클래스.멤버"로 접근해야 함. (ex. Employ.serialNum)
- ❖ 이 외에도 객체와는 무관하게 처리해야 할 경우에는 static으로 사용할 것을 권장함.

6.6.2. 정적 멤버 / static 초기화 및 주의사항

- 정적(static) 멤버는 클래스 차원의 변수이므로 초기화가 중요함.
- 정적(static) 멤버는 객체를 통한 접근이 허용되지 않고, 클래스명을 통해 접근이 가능함

정적 멤버 초기화 및 주의사항

정적 멤버 초기화

Exam_11

- ❖ static 블록은 클래스가 로딩될 때 우선하여 실행됨
- ❖ static 블록은 static 변수를 초기화할 때 사용함
- ❖ 단, 인스턴스 필드는 안됨. (객체를 통해 초기화되는 것)
➔ Static이 수행될 시점은 아직 객체가 미생성됨.

예시

```
public class Car {
    static String company = "현대";
    static String model = "그랜저";
    static String info;

    static {
        info = company + "_" + model;
    }
}
```

Static 사용 주의사항

Exam_12

- ❖ static 블록내에서는 정적멤버만 사용가능함.
- ❖ 자기자신의 객체 참조인 this 사용시 에러.
(클래스 차원의 변수이기 때문임)
- ❖ static 메서드에서는 객체를 생성한 후 메서드 접근가능

```
public class Car {
    static int maxSpeed; // 정적멤버
    int speed;           // 기본타입변수

    public void speedUp() {
        this.speed++; // (O) 가능
        this.maxSpeed++; // (X) this 사용 불가
    }

    static {
        maxSpeed = 300;
        speed = 100; // (X) 일반변수 초기화 불가
        speedUp(); // (X) 일반메서드 호출불가
    }

    static void driving () {
        this.speedUp(); // (X) this 사용불가.
        // new 연산자를 통한 객체 생성이 우선되어야 함.
        maxSpeed = 250; // (O) 접근 가능
    }
}
```

6.6.3. 싱글톤

- 하나의 어플리케이션에서는 하나의 객체만 사용. (new 연산자로 신규로 객체 생성 하지 않음)
- getInstance() 함수를 통해서 객체의 참조값을 받아와서 사용함.

싱글톤

싱글톤 만드는 방법

- ❖ 외부에서 new 연산자로 객체 생성을 막음
 - private 접근제한자 : 클래스 외부에서 접근 불가함.
- ❖ 클래스 자신의 타입으로 정적 필드로 선언.
자신의 객체를 생성해 초기화 함.
- ❖ 외부에서 호출할 수 있는 정적메서드인 getInstance()를 선언함

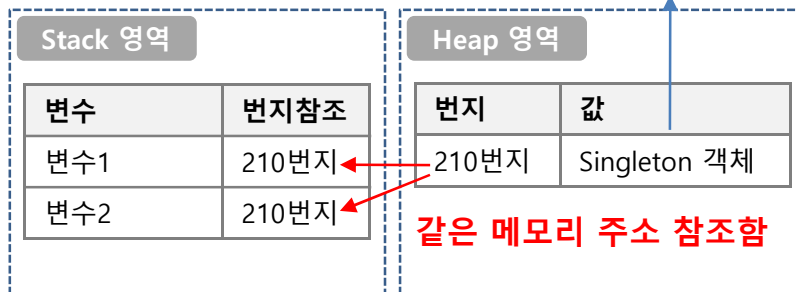
Singleton 생성 문법

```
public class 클래스 {  
    // 정적 필드 (singleton은 객체명임)  
    private static 클래스 singleton = new 클래스();  
  
    // 생성자  
    private 클래스() {}  
  
    // 정적 메서드  
    static 클래스 getInstance() {  
        return singleton;  
    }  
}
```

싱글톤 얻는 방법

Exam_13

- ❖ 클래스 변수1 = 클래스.getInstance();
클래스 변수2 = 클래스.getInstance();
→ 위와 같이 할 경우 같은 객체가 참조됨.



예시

```
public class Singleton{  
    private static Singleton singleton = new Singleton();  
    private Singleton() {}  
    static Singleton getInstance() { return singleton; }  
    Singleton obj1 = Singleton.getInstance();  
    Singleton obj2 = Singleton.getInstance();  
    Singleton obj3 = new Singleton(); // 컴파일 에러 남  
}
```

obj1과 obj2는 같은 객체임.

6.7. final 필드 및 상수(static final)

- final은 최종의 의미를 지니는 것으로 한번 정의되면 변경이 불가능한 필드를 의미함
- static final은 불변의 의미를 지니는 상수를 의미함.

final 필드 및 static final(상수) 사용법

final 필드

Exam_14

- ❖ final 필드는 딱 한번의 초기값 지정이 가능함.
 - 필드 선언시 초기값 지정가능
 - 생성자에서 초기값 지정 가능

예시

```
public class Car {  
    final String company = "현대"; // 필드 선언시 초기화 가능  
    final String carId;  
    String model;  
    public Car (String carId, String model) {  
        this.car_id = carId; // 생성자에서 초기화 가능  
        this.model = model;  
    }  
  
    Car car1 = new Car ("2022-200917", "그랜저");  
    car1.carId = "2021-000000" // (X) 변경불가함  
}
```

static final (상수)

5Exam_15

- ❖ 상수란 절대 불변의 값을 의미함.
- ❖ 원주율 등 절대적으로 정의되어 있는 수에 활용
- ❖ 전부 대문자로 작성하는 것이 관례임
- ❖ 다른 단어가 결합될 경우 "_"로 연결하는 것이 관례

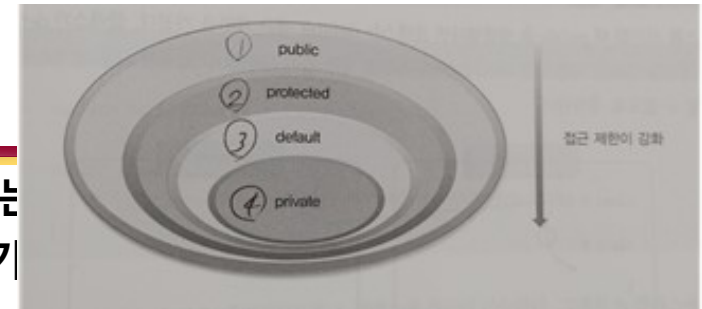
예시

```
public class Calc{  
    static final double PI= 3.141592;  
    // 절대 불변의 값을 의미함.  
    double circle(double r) {  
        return (r*r*PI); // 원의 넓이 반환  
    }  
    Calc calculator = new Calc();  
    System.out.print( calculator.circle(10) ); // 원의 넓이 출력  
}
```

Package 설명.
책으로.
Pack.t1 패키지에서 설명.

6.8. 접근제한자

- 접근제한자는 클래스, 필드, 생성자, 메서드의 접근을 제한하는
- 접근 제한자의 종류는 public, protected, default, private 4개



접근제한자

❖ 클래스, 필드, 생성자, 메서드의 접근제한을 통해 데이터 보호 및 안정적인 관리를 함

- 클래스 제한 : 다른 패키지에서 클래스를 사용하지 못하도록 막음
- 생성자 제한 : 클래스로부터 객체를 생성하지 못하게 함.
- 필드 및 메서드 제한 : 특정 필드와 메서드를 바로 접근하지 못하게 한다.

getter / setter를 통한 접근할 수 있음

접근제한	적용 대상	설명
public	클래스, 필드, 생성자, 메서드	모든 클래스가 접근 가능함
protected	필드, 생성자, 메서드	자식클래스만 접근 가능함
default	클래스, 필드, 생성자, 메서드	같은 패키지에 소속된 클래스만 접근 가능함
private	필드, 생성자, 메서드	접근 불가. 생성자를 통해서 생성이 된 객체를 통해서만 접근 가능

6.8.1. 접근제한자 (클래스)

- 클래스의 접근제한자는 public, default가 있음
- 접근제한의 기준은 동일 Package 여부에 따름

접근제한자

사용문법

❖ 접근 제한자 : public, default

- **public** : 모든 외부 클래스에서 접근 가능
라이브러리의 용도로는 public으로 선언
- **default** : 동일 Package에서만 접근가능.
접근 제한자는 생략하여 사용.

A Package

default

class A

- A : 자기자신, 접근 가능
- B : 동일 패키지. 접근가능
- C : 다른 패키지 및
Default로 접근 불가
- D : public. 접근 가능

public

class B

B Package

default

class C

- A : 다른 패키지 및
Default로 접근 불가
- B : public. 접근 가능
- C : 동일 패키지. 접근가능
- D : 자기자신. 접근 가능

public

class D

사용사례 및 주의사항

Exam_16

```
package A;
class A {
    A a = new A();
    B b = new B();
    C c = new C();
    D d = new D();
}
```

// default 접근제한자
// 접근가능. 자기자신
// 접근가능. 동일 패키지
// 에러발생. 접근불가
// 접근가능. public 접근제한자

```
package A;
public class B {
}
```

// public 접근제한자

```
package B;
class C {
}
```

// default 접근제한자

```
package B;
public class D {
    A a = new A();
    B b = new B();
    C c = new C();
    D d = new D();
}
```

// public 접근제한자
// 에러발생. 접근불가
// 접근가능. public 접근제한자
// 접근가능. 동일 패키지
// 접근가능. 동일패키지

6.8.2. 접근제한자 (생성자)

- 생성자의 접근제한자는 public, protected, default, private가 있음

접근제한자

사용문법

접근제한	설 명
public	- 모든 package에서 생성자 생성 가능
protected	- 동일 package에서 생성자 생성 가능 - 단, 다른 package라도 동일 package 안의 클래스를 상속받았을 때는 생성자 생성 가능
default	- 동일 package에서 생성자 생성 가능 - 접근제한자 생략한 형태로 사용함
private	- 동일 class 내에서만 접근 가능. - 외부에서는 접근 불가

A Package

class A

- public 생성자
- private 생성자
- default 생성자

class B

- A의 public 생성자 생성가능
- A의 private 생성자 생성불가
- A의 default 생성자 생성가능

B Package

class C

- A의 public 생성자 생성가능
- A의 private 생성자 생성불가
- A의 default 생성자 생성불가

사용사례 및 주의사항

Exam_17

```
package A;
public class A {
    public A(boolean b){...}    // public 접근제한자의 생성자
    private A(int i){...}      // private 접근제한자의 생성자
    A(double d){...}           // default 접근제한자의 생성자

    A a1 = new A(true);        // (O)public 이므로 접근가능
    A a2 = new A(10);           // (O) private이나 동일클래스. 접근 가능
    A a3 = new A(10.3);         // 동일 package이므로 접근 가능
}
```

```
package A;    // 접근하려는 클래스와 동일 package
public class B {
    A a1 = new A(true);        // (O) public 이므로 접근가능
    A a2 = new A(10);           // (X) private이므로 접근 불가
    A a3 = new A(10.3);         // (O)동일 package이므로 접근 가능
}
```

```
package B;    // 접근하려는 클래스와 다른 package
public class C {
    A a1 = new A(true);        // (O) public 이므로 접근가능
    A a2 = new A(10);           // (X) private이므로 접근 불가
    A a3 = new A(10.3);         // (X) 다른 package이므로 접근불가
                                // protected는 동일 package에서만 접근가능
}
```

6.8.3. 접근제한자 (필드/메서드)

- 필드 및 메서드의 접근제한자는 public, protected, default, private가 있음 (생성자와 동일 기준)

접근제한자

. Package A . Class A

❖ 동일 클래스안에서는 접근제한자에 상관없이 모두 사용가능

```
package A;
public class A {
    public int field1;           // public 접근제한자의 필드
    private int field2;         // private 접근제한자의 필드
    int field3;                 // default 접근제한자의 필드

    public void method1() {...} // public 접근제한자의 메서드
    private void method2() {...} // private 접근제한자의 메서드
    void method3() {...}       // default 접근제한자의 메서드

    public A() {
        field1 = 1;             // 동일 클래스내이므로 접근가능
        field2 = 1;             // 동일 클래스내이므로 접근가능
        field3 = 1;             // 동일 클래스내이므로 접근가능

        method1();              // 동일 클래스내이므로 접근가능
        method2();              // 동일 클래스내이므로 접근가능
        method3();              // 동일 클래스내이므로 접근가능
    }
}
```

. Package A . Class B

❖ 동일 package / 다른 클래스의 사용사례

```
package A;
public class B {
    A a = new A();
    a.field1 = 1; // 접근가능. public
    a.field2 = 1; // 접근불가. private은 동일클래스만 접근 가능
    a.field3 = 1; // 접근가능. default는 동일 package 접근 가능
    a.method1(); // 접근가능
    a.method2(); // 접근불가. private은 동일클래스만 접근 가능
    a.method3(); // 접근가능. default는 동일 package 접근 가능
}
```

. Package B . Class C

❖ 다른 package / 다른 클래스의 사용사례

```
package B;
public class C {
    A a = new A();
    a.field1 = 1; // 접근가능. public
    a.field2 = 1; // 접근불가. private은 동일클래스만 접근 가능
    a.field3 = 1; // 접근불가. default는 동일 package 접근 가능
    a.method1(); // 접근가능
    a.method2(); // 접근불가. private은 동일클래스만 접근 가능
    a.method3(); // 접근불가. default는 동일 package 접근 가능
}
```


6.8.4. Getter / Setter 메서드

- 데이터의 무결성을 위해서는 멤버변수는 **private**로 설정하여 외부에서 직접 접근 차단
- 외부에서 접근하려면 객체를 생성한 후 객체를 통해 데이터를 접근함

Getter / Setter

- ❖ 데이터 무결성을 위해 Class 선언시 필드는 일반적으로 **private**로 설정함.
 - Database에서 관리하는 읽기전용 필드는 함부로 변경하면 안됨. (Getter 메서드로 처리)
 - 외부에서 잘못된 값이 체크없이 업데이트 될 수 없도록 함. (Setter 메서드로 처리)

❖ Getter

- **private** 필드의 값을 리턴함.
- 필요시 값을 가공하여 리턴함.
- 접두어로 **get-** 및 **is-**를 쓰는 것이 관례

❖ Setter

- 외부에서 값을 입력할 때 사용함.
- 값의 유효성을 점검한 후 입력함
- 접두어로 **set-**을 쓰는 것이 관례

Car class

```
public class Car {
    private int currentSpeed;

    Car (int currentSpeed) {
        this.currentSpeed = currentSpeed;
    }

    void speedUp() {
        this.currentSpeed ++;      // 속도 1증가 시킴
    }

    int getCurrentSpeed () {      // 현재의 속도를 확인함
        return this.currentSpeed;
    }

    void setCurrentSpeed (int currentSpeed) { // 현재속도 셋팅
        this.currentSpeed = currentSpeed;
    }
}
```

외부 class

```
Car car = new Car (100);      // Car 객체 생성
car.speedUp();                // Car 속도 1증가시킴
car.setCurrentSpeed(140);     // Car 속도 140으로 세팅
int speed = car.getCurrentSpeed( ); // Car 속도 확인
```

6.8.5. 참조자료형 정의의 사용

09. 참조 자료형 변수

참조 자료형

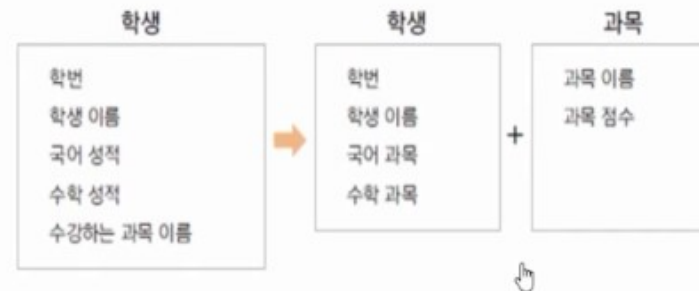
- 변수의 자료형



- 클래스형으로 변수를 선언
- 기본 자료형은 사용하는 메모리의 크기가 정해져 있지만, 참조 자료형은 클래스에 따라 다름
- 참조 자료형을 사용 할 때는 해당 변수에 대해 생성하여야 함
(String 클래스는 예외적으로 생성하지 않고 사용할 수 있음)

참조 자료형 정의하여 사용하기

- 학생이 수강한 과목들에 대한 성적을 산출하기 위한 경우 학생 클래스 속성에 과목이 모두 있으면 불합리
- 학생(Student)과 과목(Subject)에 대한 클래스를 분리하여 사용하고 Subject 클래스를 활용하여 수강한 과목들의 변수의 타입으로 선언



- 선언된 Subject 변수는 생성된 인스턴스가 아니므로, Student의 생성자에서 생성하여 사용

속성	학생1	학생2	학생3	학생4
학번	K01	K02	K03	K04
이름	강현준	강우혁	강주혁	김태희
전공	산업공	컴퓨터공	신문방송학	교육학
수강과목	경제, 물류, 수학, 컴퓨터, 영어	컴퓨터, 수학, 영어, 네트워크, 그래픽	방송, 신문, 영어, 컴퓨터, 심리	교육, 심리, 철학, 영어, 아동

6.8.5. 참조자료형 정의의 사용

참조 자료형 정의하여 사용하기

- 학생이 수강한 과목들에 대한 성적을 산출하기 위한 경우 학생 클래스 속성에 과목이 모두 있으면 불합리
- 학생(Student)과 과목(Subject)에 대한 클래스를 분리하여 사용하고 Subject 클래스를 활용하여 수강한 과목들의 변수의 타입



- 선언된 Subject 변수는 생성된 인스턴스가 아니므로, Student의 생성자에서 생성하여 사용



클래스 설계를 어떻게??

1. 첫번째 방법 : Exam21-01
2. 두번째 방법 : Exam21-02
(단, 인자를 배열로 처리하는 것은 사전 실습)

속성	학생1	학생2	학생3	학생4
학번	K01	K02	K03	K04
이름	강현준	김석훈	박선희	김태희
전공	산업공	컴퓨터공	신문방송학	교육학
수강과목	경제, 물류, 수학, 컴퓨터, 영어	컴퓨터, 수학, 영어, 네트워크, 그래픽	방송, 신문, 영어, 컴퓨터, 심리	교육, 심리, 철학, 영어, 아동
경제	100			
물류	100			
수학	100	100		
컴퓨터	100	100	100	
영어	100	100	100	100
네트워크		100		
그래픽		100		
방송			100	
신문			100	
심리			100	100
교육				100
철학				100
아동				100

[6-21] Tv클래스를 주어진 로직대로 완성하시오. 완성한 후에 실행해서 주어진 실행결과와 일치하는지 확인하라.

[참고] 코드를 단순히 하기 위해서 유효성검사는 로직에서 제외했다.

[연습문제]/ch6/Exercise6_21.java

```
class MyTv {
    boolean isPowerOn;
    int     channel;
    int     volume;

    final int MAX_VOLUME = 100;
    final int MIN_VOLUME = 0;
    final int MAX_CHANNEL = 100;
    final int MIN_CHANNEL = 1;

    void turnOnOff() {
        // (1) isPowerOn의 값이 true면 false로, false면 true로 바꾼다.
    }
}
```

```
void volumeUp() {
    // (2) volume의 값이 MAX_VOLUME보다 작을 때만 값을 1증가시킨다.
}

void volumeDown() {
    // (3) volume의 값이 MIN_VOLUME보다 클 때만 값을 1감소시킨다.
}

void channelUp() {
    // (4) channel의 값을 1증가시킨다.
    // 만일 channel이 MAX_CHANNEL이면, channel의 값을 MIN_CHANNEL로 바꾼다.
}

void channelDown() {
    // (5) channel의 값을 1감소시킨다.
    // 만일 channel이 MIN_CHANNEL이면, channel의 값을 MAX_CHANNEL로 바꾼다.
}
} // class MyTv

class Exercise6_21 {
    public static void main(String args[]) {
        MyTv t = new MyTv();

        t.channel = 100;
        t.volume = 0;
        System.out.println("CH:"+t.channel+", VOL:"+ t.volume);

        t.channelDown();
        t.volumeDown();
        System.out.println("CH:"+t.channel+", VOL:"+ t.volume);

        t.volume = 100;
        t.channelUp();
        t.volumeUp();
        System.out.println("CH:"+t.channel+", VOL:"+ t.volume);
    }
}
```

[실행결과]

```
CH:100, VOL:0
CH:99, VOL:0
CH:100, VOL:100
```

Annotation 설명.
책으로.