JAVA PROGRAMING

매소드

반복적으로 사용될 것 같은 로직을 별도의 단위로 보관하기 위한 아이디어

일반적으로 '{ }'를 이용해서 로직의 경계선으로 설정

C언어 = 함수
Java = 메소드
클래스 내부에 정의된 함수=메소드이다. 자바는 클래스 외부에는 정의가 불가능하다.(참고)

Java에서 로직을 실행하기 위해서는?

Java에서 실행의 기본 단위는 '.class'파일
- 실행의 핵심: ClassLoader는 '.class'파일을 메모리상으로 로딩

즉 실행하려면 .class라는 '{ }'에 속해야만 한다.

Class 파일의 '{ }'는 묶음의 단위
- 로직 역시 동일하게 묶음의 단위

모든 묶음에는 찾아갈 수 있는 방법이 필요
- 가장 보편적인 방법: 네이밍

매소드

Java에서 로직을 구성하는 방법

단위가 클래스 이므로 로직이 들어가는 적절한 클래스를 선택한다.
- 쉬운 커뮤니케이션을 위해서라면 관련 있는 클래스의 이름을 선택

가장 먼저 결정해야 하는 것은 로직의 정확한 이름

로직을 의미하기 때문에 '~기능'에 맞는 동사로 시작

public void 로직의 이름( ) { }로 구성

로직의 파라미터(필요 데이터) 결정

로직의 실행 결과 타입 결정

매소드

로직의 파라미터(Parameter, Arguments)

매개변수, 인자, 인수 등의 이름으로 통용됨

설계 기준

- 매번 로직을 실행할 때마다 변경될 만한 데이터
  - 사용자가 결정하는 부분은 거의 파라미터로 처리된다.

미리 결정할 수 없는 데이터
  - 다른 외부에서의 결과 데이터의 경우는 거의 대부분 파라미터로 처리된다.

- 파라미터를 여러 개 사용하는 경우에는 ‘,’를 사용

```
class mainClass {
    public static void main(String []args){
        printChar('^', 7);
        printChar('$', 4);
        printChar('!', 10);
    }
    static void printChar(char ch, int sum){
        for(int i=0;i<sum;i++)
            System.out.print(ch);
        System.out.println(); //한줄건너띄기
    }
}
```

```
^^^^^^^
$$$$
!!!!!!!!!!
```

실행흐름

1. 4번째줄 실행-
1-1. 함수검사(컴파일러: 으음.. 함수이름은 printChar 이고, '^'는 char형이고, 7은 int 형이군?)
1-2. 1번째파라미터로 char 형, 2번째로 int 형 파라미터를 받는 메소드를 찾는다 (반환형은 따지지 않는다)
1-3. 찾았으면 메소드 본체에 파라미터인 ch에 '^' 대입, sum 에 7 대입
1-4. 메소드 몸체 실행(sum 만큼 ch를 출력하고 한줄 건너띄기)
1-5. 더이상 내용이 없으면 함수호출을 완료하고 호출된 곳으로 돌아감
2. 5번째 줄 실행
....

매소드

실행된 결과 – 리턴 타입

로직의 실행 결과물이 어떤 타입의 변수인지 말해 주는 것
- 편리하게 해석해서 실행 결과를 반환한다고 해석

return 이라는 키워드는 실행의 제어를 반납한다

연산자와 달리 리턴 타입은 결과를 반드시 변수로 다시 받을 필요가 없다는 것의 의미는?

키워드	int	double	char	String	void
반환하는것	int형 정수	double 형 실수	char 형 문자	문자열	반환하지않음

* 처음 배울때 하는 오해
1. 함수정의에서의 파라미터는 초기화 안하나요?
 - 해도되고 안해도 이상없음
2. 파라미터 변수의 이름이랑 main함수에서 파라미터로 넘겨받는 변수랑 이름 같아도 되나요?
 - 상관없음 ex)print(a) .... static void print(int a)
3. 함수 정의를 먼저 해야 하나요?
 - C언어는 그래야 하지만 자바는 상관 X (객체지향)

매소드

예시)

```
public class SumMachine {
	public void makeSum(){
		System.out.println("make sum.......");
	}

	public static void main(String[] args) {
		SumMachine m = new SumMachine();
		m.makeSum();
	}
}
```

```
make sum.......
```

1. Scanner s, SumMachine m과 같이 선언의 앞글자가 대문자
2. new Scanner(System.in); new SumMachine();처럼 new라는 키워드가 사용됨

매소드

예시)

```
public class SumMachine {
      public void makeSum(int startValue, int endValue){
            int start = startValue;
            int end = endValue;
            int sum = 0;

            for(int i = start; i <= end ; i++){
                  sum = sum + i;
            }//end for

            System.out.println("시작값: " + start);
            System.out.println("종료값: " + end);
            System.out.println("총 합: "+ sum);
      }

      public static void main(String[] args) {
            SumMachine m = new SumMachine();
            m.makeSum(1,100);
            m.makeSum(20,200);
            m.makeSum(30,300);
      }
}
```

1. 파라미터 선언을 할때는 ,로 구분

매소드

예시)

```
public int makeSum(int startValue, int endValue){
      int start = startValue;
      int end = endValue;
      int sum = 0;

      for(int i = start; i <= end ; i++){
            sum = sum + i;
      }//end for
      return sum;
}
```

1. return을 이용해서 반환값 처리

매소드

사용자의 체중, 신장을 입력받아 카우프 지수를 구하시오.

조건1. 사용자의 데이터를 입력받는 메소드와, 카우프 지수를 구하는 메소드를 사용할 것

카우프 지수 공식

$$\text{카우프 지수} = \frac{\text{체중(kg)}}{\{\text{신장(m)}\}^2}$$

지수가 30 이상	비만
24~29	과체중
20~24	정상
20 미만	저체중
13~15	여윔
10~13	영양 실조증
10	이하는 소모증

객체지향

프로그램의 발전 순서

1. 동일한 로직을 여러 번 사용하는 것보다 하나의 묶음으로 사용하고 싶습니다.
 - 자바에서는 메소드라는 용어로 표현

2. 메소드가 많아지니까 사이에 오가는 데이터가 많아져 불편해 졌습니다.

3. 그래서 호출받은 쪽에서 데이터를 보관했음 좋을 듯 합니다.

4. 그 후 함수와 데이터를 하나의 덩어리로 묶는 것이 좋다고 생각합니다.

5. 문제는 이렇게 묶은 덩어리를 여러 명이 사용하면 기존 데이터가 사라지는 문제가 발생합니다. 따라서 필요한 만큼 덩어리를 만들어서 사용할 수 있음 좋겠다고 생각합니다.

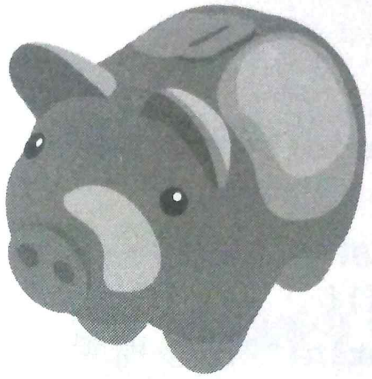
그래서 만들어 진 것이 클래스 입니다.

클래스는 복사본을 만들기 위한 원형(Prototype)과 같습니다.
 - 각자 따로 보관하고 싶어하는 데이터 혹은 속성들을 정의
 - 데이터에 관계없이 제공되는 같은 기능이나 로직

클래스의 복사본을 객체(Object, Instance)라고 합니다.
 - 틀(클래스)을 선언한 다음에는 필요할 때마다 복사본(Instance)를 만들어서 필요에 맞게 사용합니다.

객체지향

저금통을 클래스로 만들고 객체로 만들기



- 저금통이 있어서 나는 매번 금액이 어떻게 되는지 알 필요가 없습니다.
- 저금통에 돈을 넣으면 알아서 금액 데이터가 수정됩니다.
- 나중에 저금통을 깨면 전체 금액을 알 수 있게 됩니다.

저금통의 데이터

- 금액 데이터

저금통의 기능

- 돈을 넣는다.
- 모든 금액을 찾는다.

필요한 메소드
- 돈을 넣는다 : deposit, 모든 돈을 찾는다 : withdraw

객체지향

```
public class PigSave {
      public int total;

      public void deposit(){
            System.out.println("저금통 입금");
      }

      public void withdraw(){
            System.out.println("저금통 배 따기");
      }
}
```

```
public class PigSaveTest {
      public static void main(String[] args) {
            PigSave save = new PigSave();
            System.out.println(save);
      }
}
```

```
PigSave@1fb8ee3 ← @뒤의 결과는 다를 수 있습니다.
```

PigSave save = new PigSave();
- new PigSave() : PigSave라는 클래스에서 새로(new) PigSave 객체를 생산해라
- PigSave save : save라는 이름의 상자를 하나 만든다. Save 상자에는 PigSave 클래스에서 생산된 객체의 리모컨(레퍼런스)이 들어갈 것이다.

객체지향

```
PigSave save = new PigSave();
save.deposit(100);
save.deposit(500);
System.out.println(save.total); // 직접 데이터에 접근
```

```
저금통 입금
저금통 입금
600
```

. 으로 접근할 수 있는 대상
- 객체가 가진 메소드
- 객체가 가진 데이터

* 만약 save.total = 10000;
라는 식으로 데이터를 몰래 바꿀 수 있으므로 정보 은닉이 중요하다.

반드시 객체의 데이터는 메소드를 통해 변경해야 함. 따라서 private를 사용하여
Private int total;
작성해야 한다. (private의 의미는 클래스의 {}를 벗어나서는 접근할 수 없다는 뜻이다.)

결과값을 얻기 위해서는 getTotal()같은 메소드를 만들어서 사용한다.

마지막으로 객체 안에 선언된 지역변수는 객체마다 데이터의 값이 다르게 유지되므로,
객체의 상태, 속성이라고 한다.

객체지향

```
public class FoodPrice {
    private int menuPrice;
    private int quantity;

    public FoodPrice(int menuPrice){
        this.menuPrice = menuPrice;
        this.quantity = 1;
    }

    public FoodPrice(int menuPrice, int quantity){
        this.menuPrice = menuPrice;
        this.quantity = quantity;
    }

    public int getTotalPrice(){
        return menuPrice * quantity;
    }
}
```

```
public FoodPrice(int menuPrice){
    this(menuPrice, 1);
}

public FoodPrice(int menuPrice, int quantity){
    this.menuPrice = menuPrice;
    this.quantity = quantity;
}
```

this는 self의 의미
코드는 위와 같이 고치는게 좋음

객체지향

저금통 프로그램을 만드시오.

조건1. 입금하는 방식은 돈만 넣는 방법, 돈과 메시지를 넣는 방법의 2가지이다.
조건2. 돈만 넣었을 경우 '딸그랑'이라는 메시지와 입금 액수 출력
조건3. 메시지와 같이 넣으면 메시지와 입금 액수 출력
조건4. 'crash'를 입력하면 총 액수 출력과 함께 프로그램 종료

객체지향

객체지향 분석 설계 (OOAD)

객체지향 설계를 위한 모델링 기법

일반적으로 요구사항 분석 -> 설계 -> 구현 -> 테스트

요구사항의 분석

고객이 시스템과 만나는 접점: 화면
- 보통은 화면에서 할 수 있는 기능을 설명

간단한 그림과 도표를 통해서 커뮤니케이션

여러 종류의 시스템 사용자들: 이해 관계자

클래스의 분석방식

일반적으로 Top – down 방식으로 구성
- 큰 모듈에서 작은 모듈로, 각 모듈별 클래스 구성

클래스 분석의 기초: 명사 분석
- 고객의 설명 중에 등장하는 명사, 혹은 기록되어 있는 데이터들을 위주로 분석하는 방식
- 도메인객체(Domain Object)

객체지향

로직 위주의 객체의 구분법

3- tier 방식이 가장 많이 사용됨
- Presentation Tier: 화면
- Business Tier: 회사마다 달라지는 로직
- Persistence Tier: 영속적인 데이터 전문 처리

현재 패스트푸드 업무 처리 시스템이 가장 적절한 예

사용자와 시스템의 만남 – 유스케이스

모든 업무 프로세스는 말로 표현하는 것이 가능하다.
시간의 순서대로 기록
주어와 동사로 기록
행위를 하는 주도적인 입장 – Actor
Actor는 사람이나 사물, 시스템, 데이터일 수 있다.

시스템 구성- 클래스 설계

데이터 부분은 별도 설계
화면
비즈니스 로직
데이터 저장
각 각의 테스트

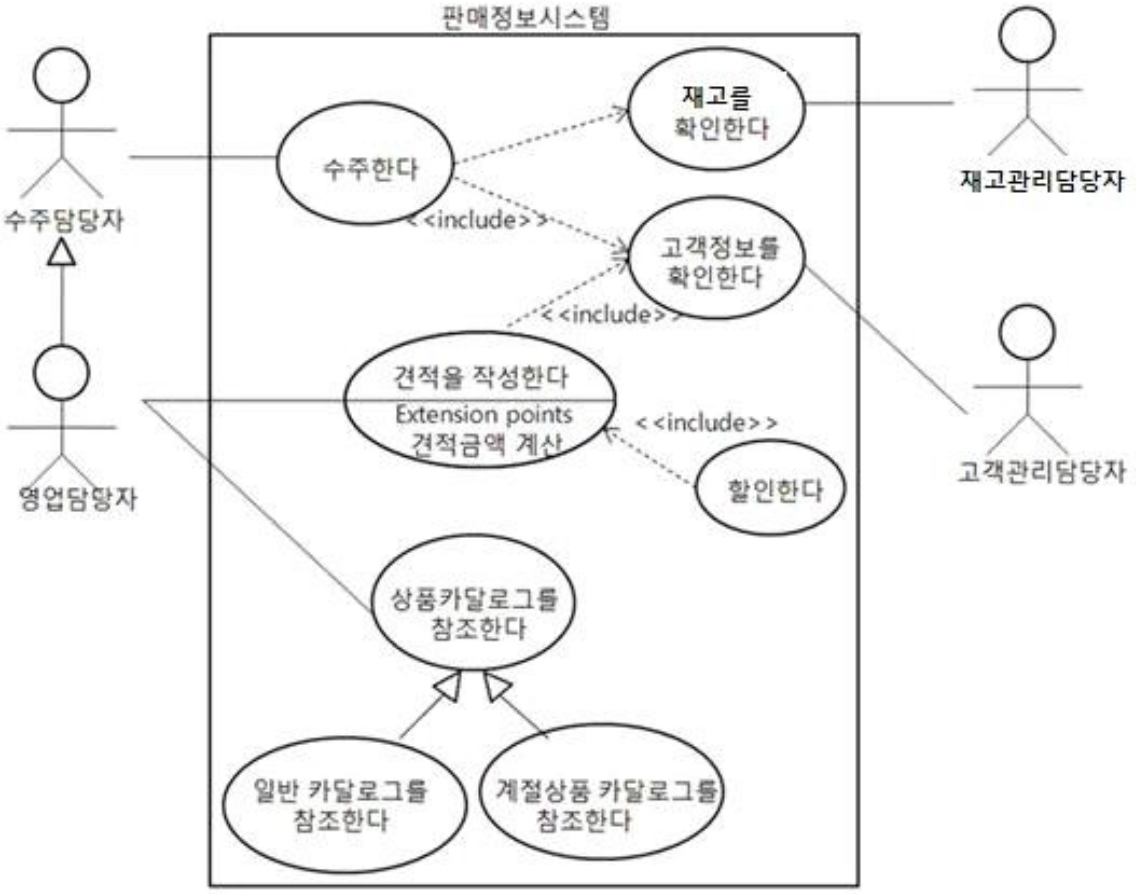
객체지향

유스케이스 예시

1. 액터의 후보를 밝힌다.
: 수주담당자, 영업담당자
2. 추출한 액터별로 유스케이스를 생각한다. (시스템의 기능을 검토한다)
: 수주담당자 -> 수주한다
3. 포함된 유스케이스를 추출한다. (기능과 관련한 외부시스템을 검토한다)
: "수주한다" 유스케이스 -> "고객정보를 확인한다", "재고를 확인한다" 유스케이스를 포함한다
4. 외부시스템을 유스케이스를 추출한다. (유저의 추상화, 구체화를 검토한다)
: 외부시스템의 "고객관리시스템", "재고관리시스템" 액터를 추출하고 유스케이스 관계를 기술한다
5. 영업담당자와 수주담당자의 일반화 관계를 정의한다. (시스템 기능을 검토한다)
6. 영업담당자의 유스케이스를 추출한다. (기능의 포함관계를 검토한다)
7. "견적을 작성한다"로 부터 고객정보를 확인한다로 포함관계를 표시한다. (기능의 확장을 검토한다)
8. "견적을 작성한다" 유스케이스를 확장한다. (기능의 추상화, 구체화를 검토한다)
: "견적을 작성한다" 유스케이스를 확장하는 "할인한다" 유스케이스의 확장관계를 기술한다
9. "상품카탈로그를 참조한다"를 특화시킨 유스케이스 수출.

숙제) 도서 대여점의 유스케이스를 작성하시오.

객체지향


판매정보시스템
수주담당자
영업담당자
수주한다
재고를
확인한다
재고관리담당자
<<include>>
고객정보를
확인한다
<<include>>
견적을 작성한다
Extension points
견적금액 계산
<<include>>
할인한다
고객관리담당자
상품카달로그를
참조한다
일반 카달로그를
참조한다
계절상품 카달로그를
참조한다

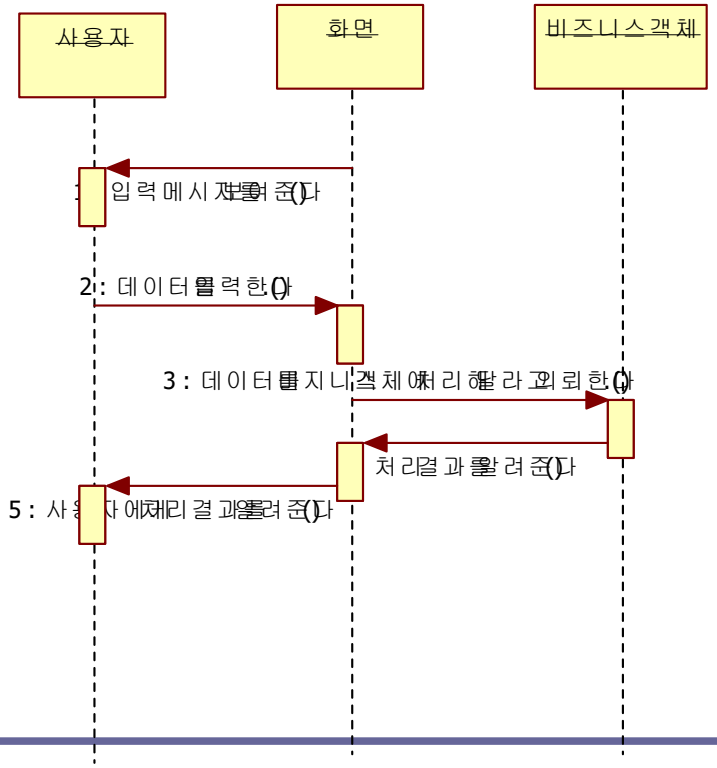
객체지향

초급자에게 적합한 설계 가이드

시스템의 뒤쪽에 신경쓰지 말고, 화면에 집중하라.
사용자와 화면 간에 이루어지는 일에만 집중하라.
성공하는 경우만 집중하라.
한번에 완성할 생각은 버려라.
그림으로 표현이 가능한지 살펴보라.
다 완성되면 정확한 시스템의 스펙(기능명세)이 완성되어야 한다.

Sequence 다이어그램

객체 사이에 이루어지는
커뮤니케이션을 그림으로 표현



객체지향

시스템의 내부의 설계

데이터를 찾아내고, 분석한다.

로직과 저장을 분리하는 설계로 작성한다.

데이터 저장을 담당하는 클래스 설계
비즈니스 로직을 담당하는 클래스 설계
화면을 담당하는 클래스 설계

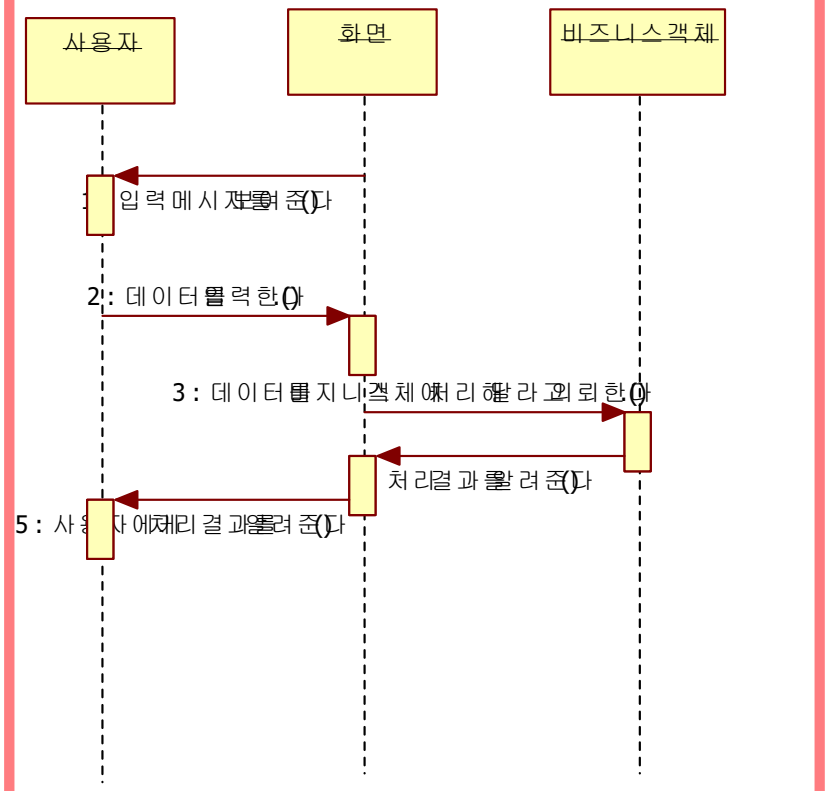
인사관리 시스템의 설계

사용자가 화면(시스템)을 통해서 원하는 기능 파악
- C (Create)
- R (Read)
- U (Update)
- D (Delete)

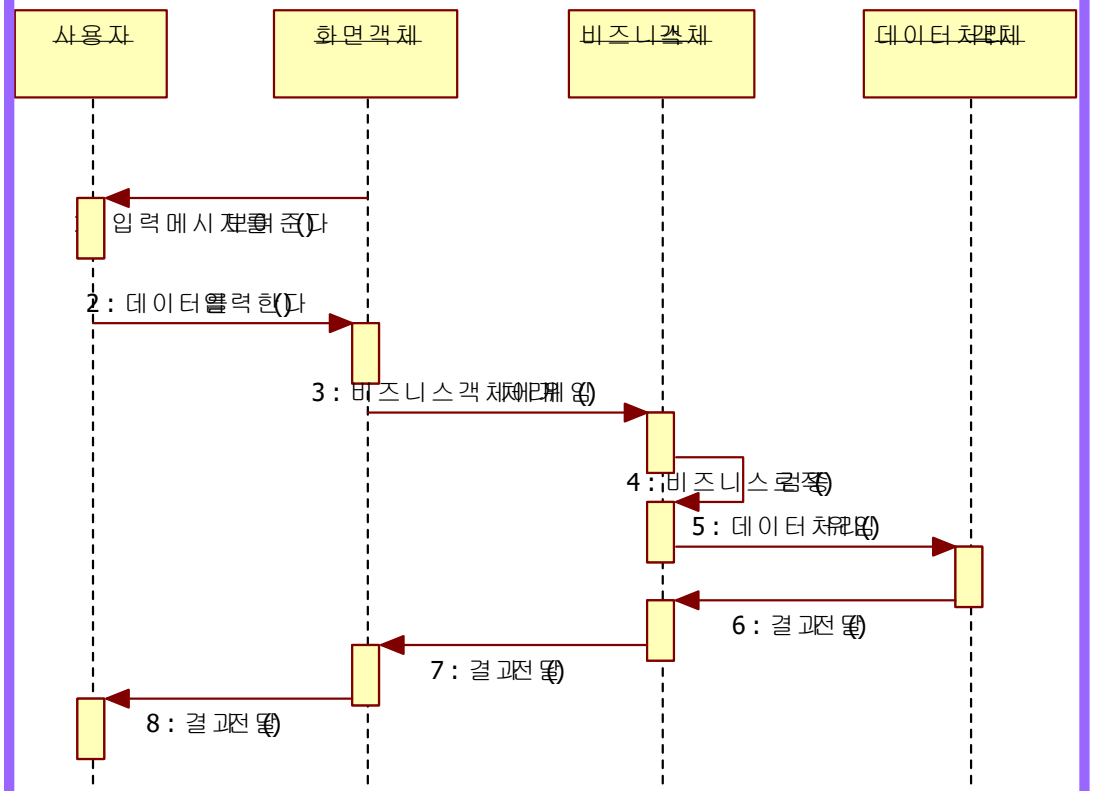
데이터의 분석
- Domain Object(Value Object)의 설계
- 생성자
- getter/setter

객체지향

사용자 – 화면 - 비즈니스객체(서비스)



사용자-화면-비즈니스객체- 데이터처리객체



객체지향

객체를 만들어야 하는 기준?

객체지향 프로그래밍이 데이터를 하나의 구조로 보관하기 위한 템플릿으로 클래스를 사용

실제의 개별적으로 묶이는 데이터는 클래스의 인스턴스(객체)를 생성하는 방식으로 사용

함수 VS 메소드

특정한 로직을 처리하는 데 사용한다는 점은 동일

메소드 – 각 객체마다 가지는 데이터를 이용하는 로직

함수 – 개별적인 데이터의 보관이 없는 로직

Java에서의 static

특정 로직에 static 키워드가 붙게 되면 전혀 다른 방식으로 동작

static의 의미는 '고정된, 변하지 않는'

'전역'이라는 의미와 static

객체지향

패키지?

클래스의 모음

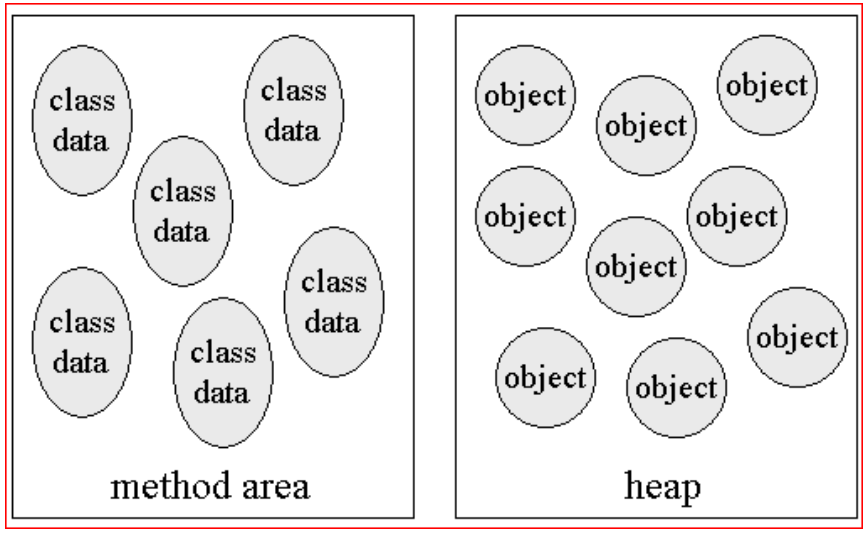
import를 사용해서 선언해줘야 함.

java.lang패키지는 자동으로 임포트됨.

자신만의 패키지를 만들 때
- 패키지의 시작은 회사의 도메인
- 패키지의 중간은 약어 혹은 모듈의 이름
- 패키지의 맨 뒤는 패키지 안의 클래스들의 역할

객체지향

클래스의 메모리 영역과 객체의 메모리 영역



static 메소드

객체 영역이 아닌 클래스들의 메모리 영역에 존재

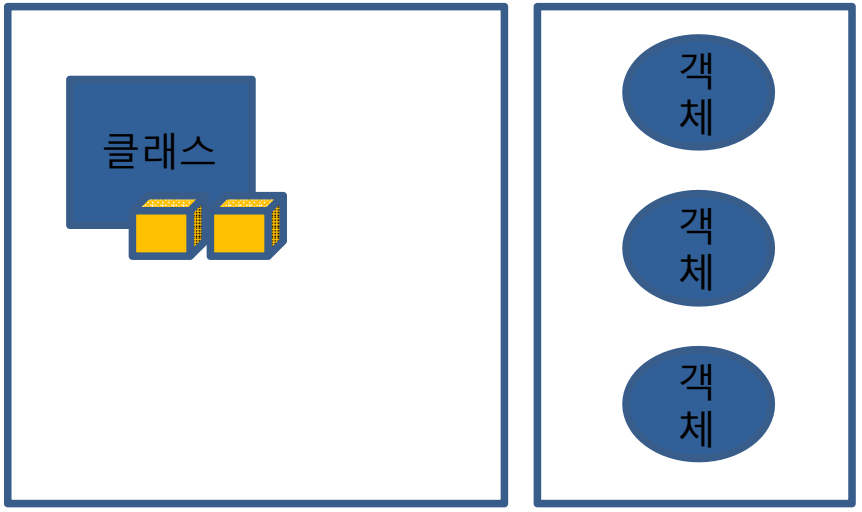
따라서 접근 방식 역시 객체를 통한 접근을 하지 않는 방식으로 사용

클래스의 이름.static 메소드

객체지향

static 메소드 VS 객체의 메소드

메모리 구조의 이해를 통한 접근 방식의 차이



객체들은 클래스에 대한 접근 가능하므로 static 메소드 접근가능

반면 클래스는 어떤 객체를 실행해야 하는지 판단할 수 없기 때문에 객체쪽으로 접근 불가

static메소드와 static 변수

.class 의 '{ }'로 묶는다 - 메모리상의 공간
.class 안의 선언된 static 변수
static 변수 - 클래스의 변수

객체지향

static 블록

클래스가 로딩된 직후에 실행되는 블록

클래스 로딩과 관련되서 실행되기 때문에 단 한번만 동작

When use static ?

객체는 '데이터를 보관한다' 그렇다면 각각의 데이터를 보관할 필요가 없다면?

객체에 상관없이 공개되면서 공유되어야 하는 기능과 데이터인가?

static 용도(1) – 객체간의 공유 변수

static 변수는 객체에 상관없이 클래스 정보를 통해서 사용하는 공유되는 변수를 의미한다.

객체들의 결과를 누적하거나 객체들의 메소드 제어시에 사용할 수 있다.

객체지향

static 용도(2) – 객체생성없는 함수

별도의 객체 생성없이 로직만을 이용하고 싶은 경우
데이터를 개별 보관하지 않는다.

Java코드에서 공통점을 찾아볼 수 있다.
- Math.random( );
- Integer.parseInt( );
- String.valueOf

static 용도(3) – 클래스당 한번만 실행

static { }

클래스 로딩 후 한번 실행

객체와 무관하게 클래스 로딩 후 실행

실행의 우선 순위가 가장 높은 코드

객체지향

예시)

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

        OrderUtil u1 = new OrderUtil();
        OrderUtil u2 = new OrderUtil();

        u1.pressButton();
        u2.pressButton();
}
```

```
고객님의 번호는 1 입니다.
고객님의 번호는 1 입니다.
```

```
private static int count = 0;

public void pressButton(){
        count++;
        System.out.println("고객님의 번호는 " + count+" 입니다.")
}

public static void main(String[] args) {

        OrderUtil u1 = new OrderUtil();
        OrderUtil u2 = new OrderUtil();
        u1.pressButton();
        u2.pressButton();
}
}
```

객체지향

객체의 메모리 문제

클래스의 각 객체마다 다른 데이터를 보관

따라서 자바에서 객체의 생성문제는 메모리를 많이 사용하게 되는 문제가 발생

static의 메모리 문제

객체영역이 아닌 클래스가 로딩되는 영역에 생성되므로 메모리의 소비를 줄일 수 있음
(이는 속도와도 관련)

하지만 static은 가비지 컬랙션의 대상이 아님
(메모리 회수가 일어나지 않음)
 - 즉, 장기간 프로그램이 가동되면 점차 느려지는 현상이 나타나게 됨

따라서 static의 사용은 조심해야 함

* java.lang.Object 클래스에 protected void finalize() throws Thowable 라는 소멸자가 있으므로 필요시 강제 소멸을 시킬 수 있음
단, 즉시 소멸되는 것이 아니라 가비지 컬랙터에 우선순위를 등록시키는 것임
(자바에서 가비지 컬랙터는 시스템에서 임의로 수행함)