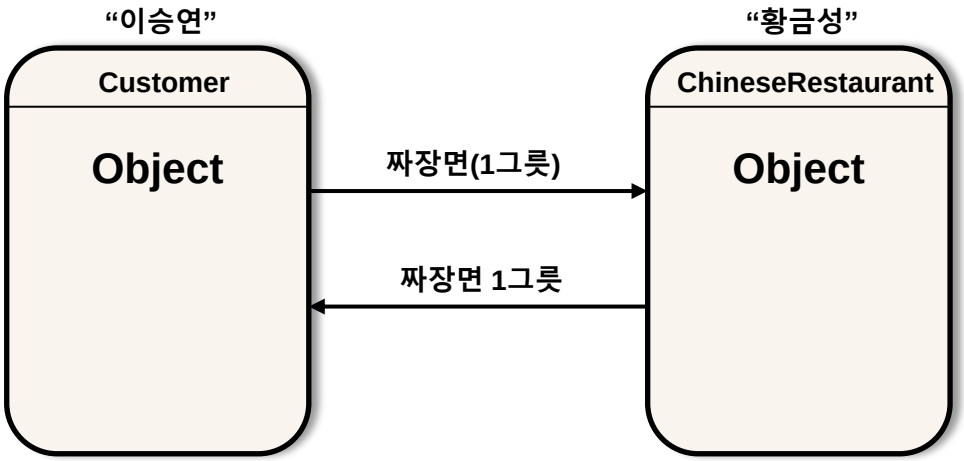
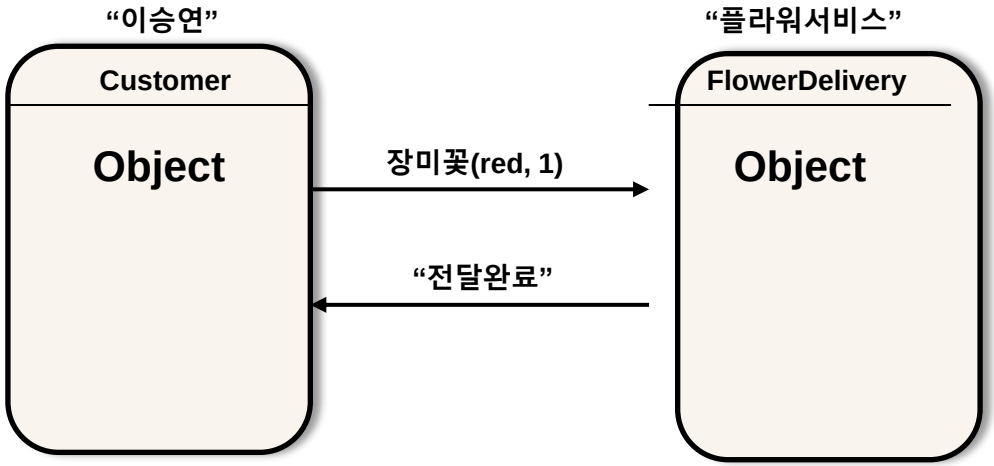
제 2 장
객체지향프로그래밍과 자바

현실세계의 작업

- 70% 이상이 **객체**들 간의 소통(요청과 응답)
  - 객체(object) : 세상에 존재하는 모든 것
- 예: 중국음식(짜장면)을 먹고자 한다
  - 가고자 하는 중국음식점을 찾는다
  - 메뉴를 살펴보고 원하는 음식(삼선짜장면)을 요청 – what
  - 응답으로 나온 삼선짜장면을 맛있게 먹는다
- 예: 부모님께 어버이날 꽃배달을 하고자 한다
  - 꽃배달 업체를 찾는다
  - 원하는 꽃(카네이션)의 배달을 요청 – what
  - 응답으로 배달 완료 메시지를 받는다
- 작업체를 찾고 - 작업을 요청 – 그 결과를 전달

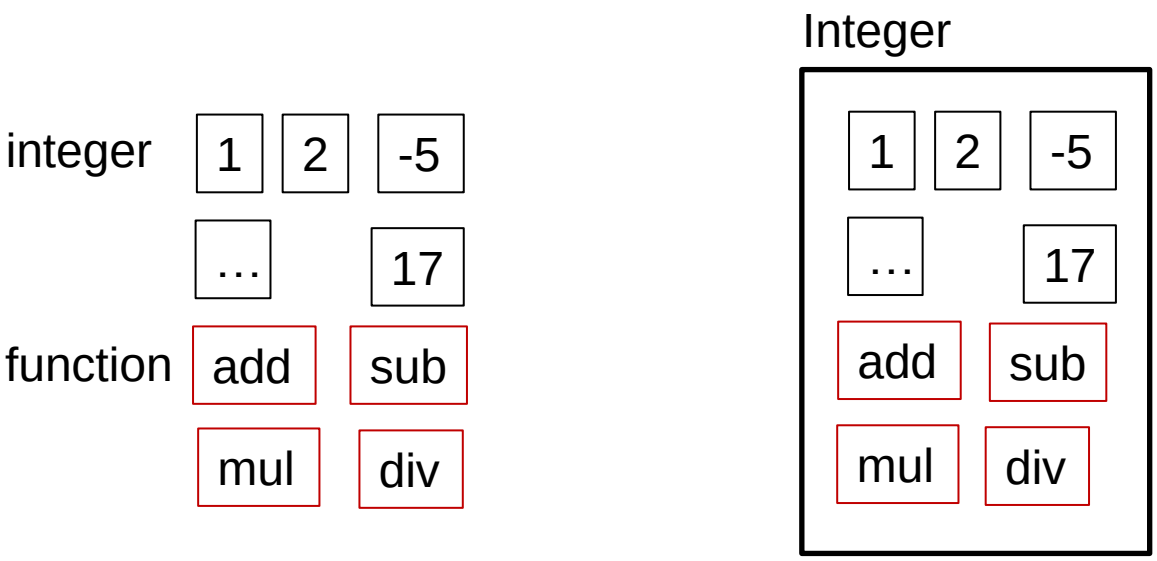

“이승연”
Customer
Object
짜장면(1그릇)
짜장면 1그릇
“황금성”
ChineseRestaurant
Object


“이승연”
Customer
Object
장미꽃(red, 1)
“전달완료”
“플라워서비스”
FlowerDelivery
Object

객체 개념과 자바프로그램 생김새

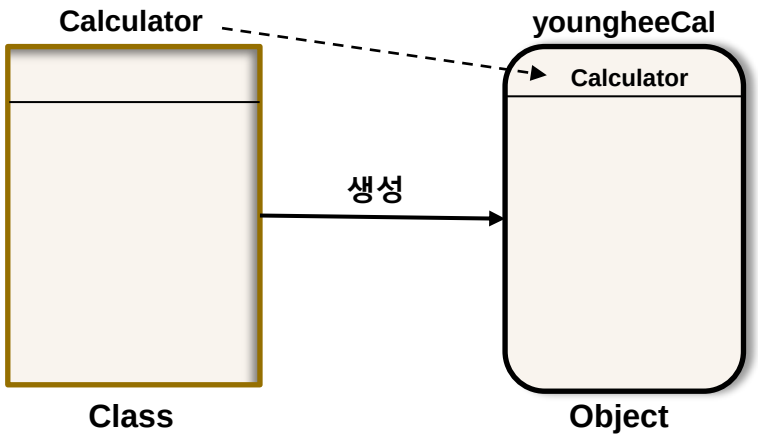
- 객체란?
  - 상태(혹은 속성)
    - 객체를 규정하는정적 특성을 나타내는 값
    - 동작을 지원하며, 동작에 의해 변경이 되기도 한다
  - 행동(동작 기능)
    - 상태를 기반으로 행해지는 동적 기능
    - 행동에 의해 상태가 변경되기도 한다
  - 예: 강아지
    - 상태 : 이름, 품종, 색깔, 꼬리모양, 굶주림정도 …
    - 행동 : 짓기, 꼬리흔들기, 달리기, …

- 객체는 하나의 명사처럼 생각되지만, 내부에 동적 기능을 내포하고 있다는 점을 알아야 한다
- 예: Integer(정수)를 생각해보자
  - Integer(정수)와 정수 기반의 동작(add, sub, div, mul 등)을 바라보는 관점
  - add_int, add_float → add
- 중국음식점과 꽃배달업체를 객체로 정의하면?



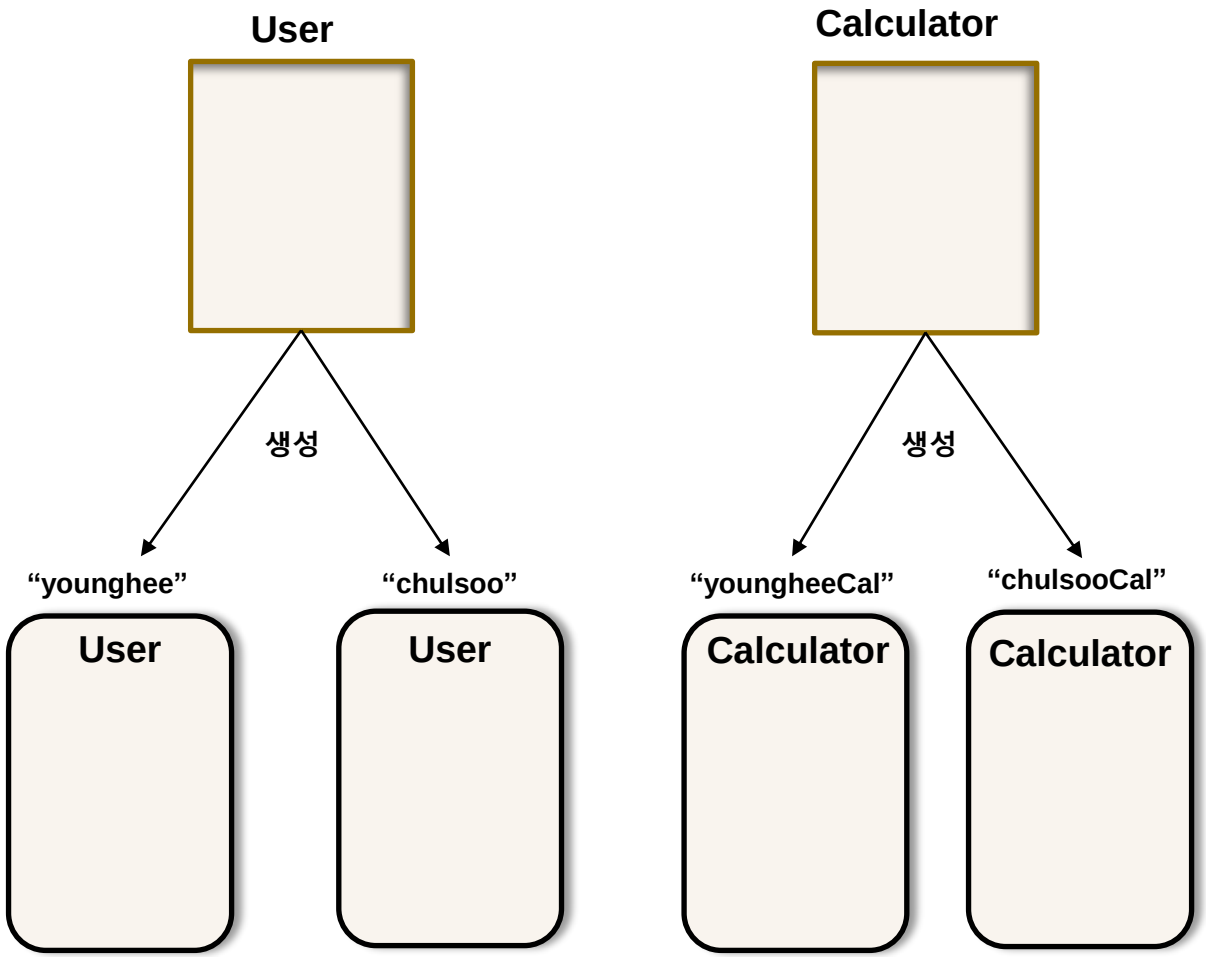
- 오버로딩(overloading)과 오버라이딩(overriding)
  - 동일한 이름으로 다른 동작을 수행할 수 있는 자바의 특성
  - 다형성(polymorphism)이라는 개념으로 이론적 정의
  - man-cry, dog-cry가 다른가? 모두 cry 아닌가?
- 캡슐화(encapsulation)와 정보은폐(information hiding)
  - 중국음식점에서 삼선짜장면이 "어떻게(how)" 만들어지는 주방이 따로 분리되어 있어(캡슐화) 고객에게는 보이지 않는다(정보은폐)
    - 어떤 재료를 사용하여 어떻게 만들어지는지 알 수 없음 - 다만 어떤 음식만 공개
  - 꽃배달 업체가 다른 장소에 분리되어 있으며, 꽃을 "어떻게(how)" 배달하는지 고객에게는 감추어져 있다
  - 데이터 추상화(abstraction)를 위한 기반 개념 (교재 p.21 정독)

- 클래스와 객체
  - 클래스
    - 상태와 동작으로 구성되는 객체 생성을 위한 틀(frame) 혹은 구성도
      - 필드(field)와 메소드(method)
    - 자바 프로그램을 구성하는 **단위 컴포넌트** → 자바프로그램: 클래스의 집합
  - 객체
    - 필드와 메소드로 구성된 추상데이터 형(abstract data type)
    - 모든 객체는 **클래스로부터 생성**된다

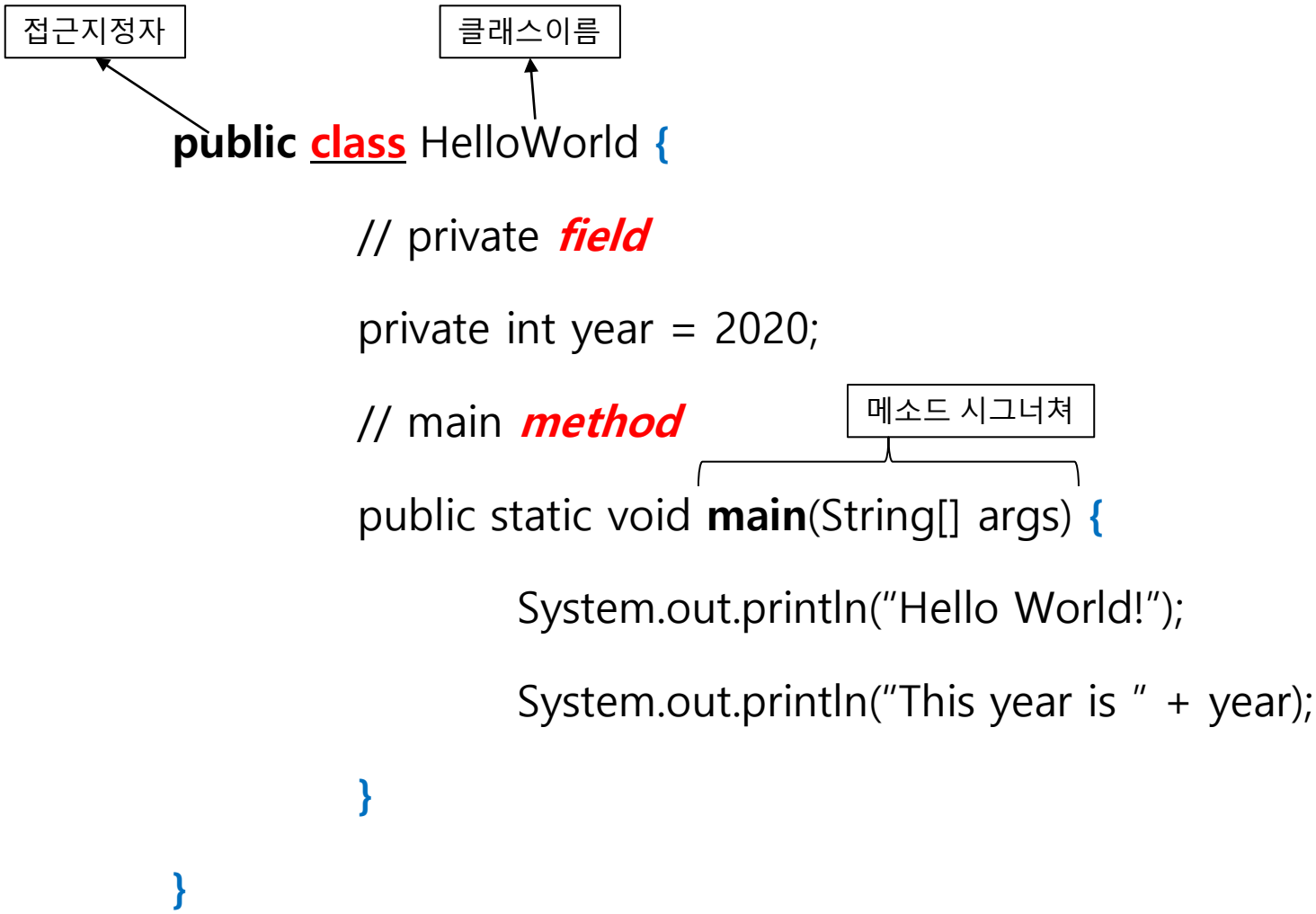


클래스

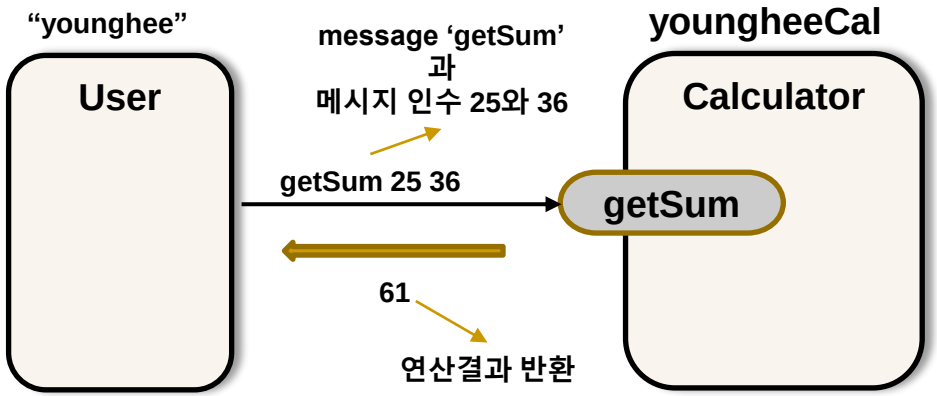
객체
(인스턴스)


User
Calculator
생성
생성
“younghee”
“chulsoo”
“youngheeCal”
“chulsooCal”
User
User
Calculator
Calculator

- 클래스 = 필드 U 메소드
  - **필드(field)**
    - 객체의 정적 특성을 나타내는 상태 정보 변수
    - 클래스 내부에 정의된 모든 메소드들이 공유하는 정보
  - **메소드(method)**
    - 객체의 동적 기능을 기술한 명령어 블록
    - 메소드 구성의 헤더부분을 메소드 시그너쳐(method signature)하고 한다
      - 메소드이름(매개변수리스트)
    - 자바 프로그램을 구성하는 클래스 중 어느 한 클래스에는 main 메소드가 존재
  - 클래스 선언
    - *접근지정자* **class** *클래스이름* { … 클래스 몸체 = 필드 + 메소드 …}


접근지정자
클래스이름
public class HelloWorld {
// private field
private int year = 2020;
// main method
메소드 시그너쳐
public static void main(String[] args) {
System.out.println("Hello World!");
System.out.println("This year is " + year);
}
}

- 메시지와 메소드
  - 객체들 간의 메시지 전달을 통해 작업을 수행
  - 메시지를 기반으로 객체 내부의 메소드를 수행
  - 메시지 : 요청과 응답(request/response)
    - 메소드 이름 + 인수
    - 리턴 값
  - 메소드
    - 클래스 메소드 : 인스턴스의 생성 없이 클래스 이름으로 호출
    - 인스턴스 메소드 : 인스턴스가 생성 되어야만 호출 가능



API 클래스와 실행클래스

- API(Application Programming Interface)
  - 다른 클래스로부터 access 가능한 필드와 메소드들로 구성
  - 프로그램 실행의 진입점이 되는 main 메소드를 가지고 있지 않다
  - 독자적 실행이 불가능하다
  - C 프로그램의 라이브러리(library)에 해당한다

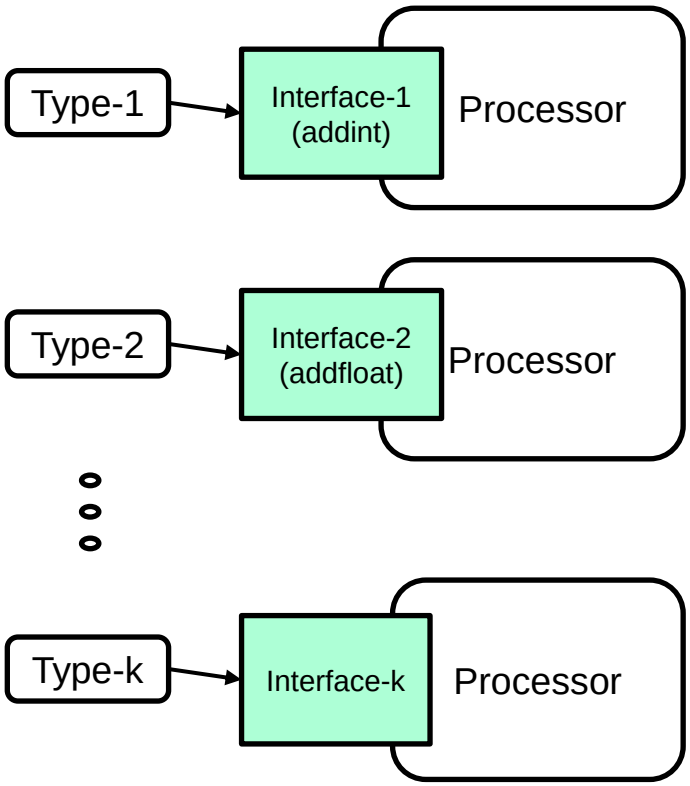
- 실행(execution) 클래스
  - 다른 클래스로부터 access 가능한 필드와 메소드들로 구성
  - 프로그램 실행의 진입점이 되는 main 메소드를 가지고 있다
  - 독자적 실행이 가능하다
  - main 함수를 포함하는 C 프로그램에 해당한다

자바 객체의 특성

- 추상화(abstraction)
  - 필요한 것만 남기고 불필요한 모든 것은 외부로부터 보이지 않게 감춘다
  - 캡슐화(encapsulation) + 정보은폐(information hiding)
  - 컴퓨터 프로그래밍
    - 데이터 추상화(data abstraction) 혹은 추상 데이터형(abstract data type)
    - 객체지향 프로그래밍 → 객체(object)
  - 추상화 레벨
    - 고수준 추상화 → 명확한 의미 전달이 어려우며 애매모호함이 증가
      - 예: 메뉴도 없는 중국집
    - 저수준 추상화 → 복잡도가 높아지며 이해도가 떨어지며 변형과 확장이 어려움
      - 예: 모든 재료 준비해 놓고 고객이 원하는 음식 스스로 요리하는 중국집

- 다형성(polymorphism)
  - Morphism : 기본 구성을 유지하며 다른 개체로의 변환
    - Monomorphism vs. Polymorphism
  - Monomorphism (단일 변환, 단변환성)
    - A → B, A → C 등 처럼 어떤 개체에서 다른 하나의 개체로의 변환
  - Polymorphism(다중 변환 혹은 다형성)
    - A → B, A→ C 등 처럼 어떤 개체에서 다른 1개 이상의 개체로의 변환

Monomorphism 개념도



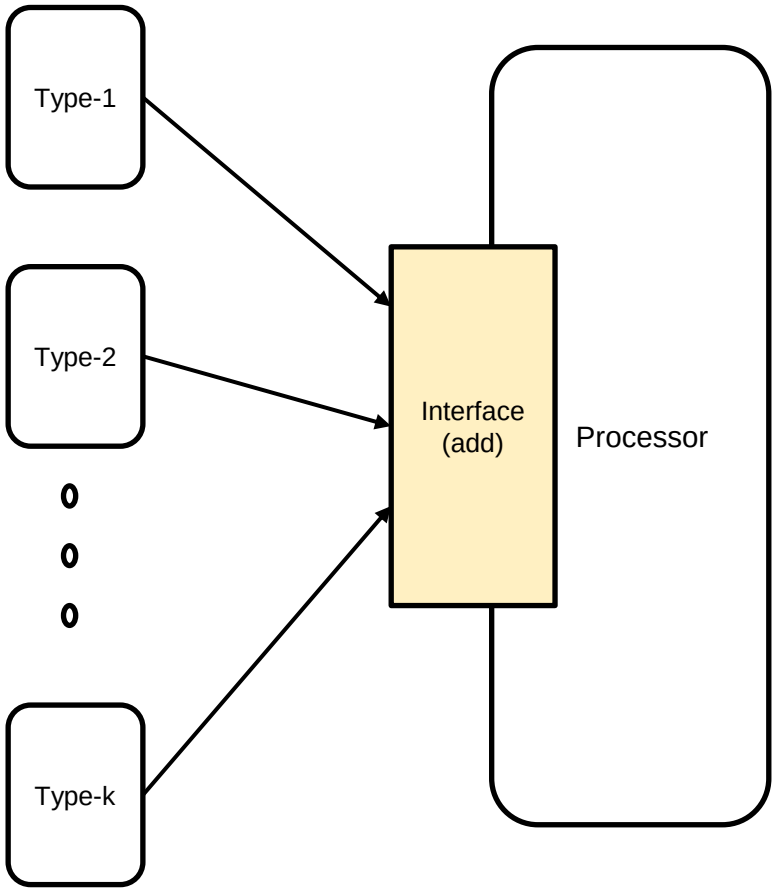
- Monomorphism
  - 정수들의 덧셈

```
int addint(int x, int y)
{
        return x+y ;
}
```

  - 실수들의 덧셈

```
int addfloat(float x, float y)
{
        return x+y ;
}
```

  - 동일한 이름으로는 다른 타입의 덧셈이 불가능



- Polymorphism
  - 정수들의 덧셈

```
int add(int x, int y)
{
        return x+y ;
}
```

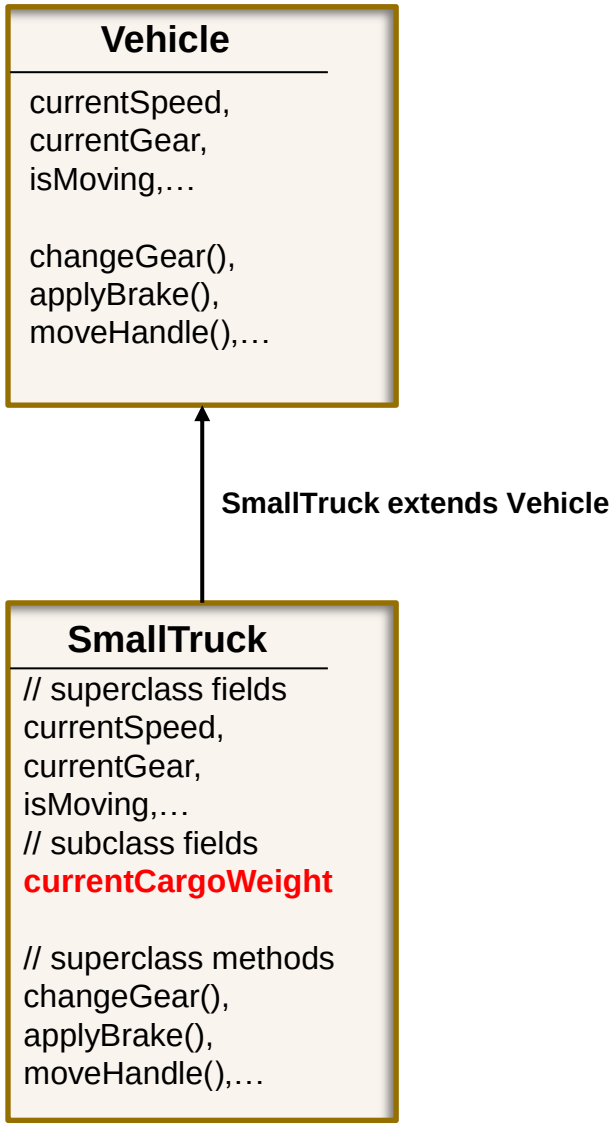
  - 실수들의 덧셈

```
int add(float x, float y)
{
        return x+y ;
}
```

  - 동일한 이름으로는 다른 타입의 덧셈이 가능
  - 다른 말로 **method overloading**(혹은 operator overloading)

상속(inheritance)

- 여러 다른 객체들이 동일한 멤버들을 중복 정의할 필요없이 공통된 멤버들을 어떤 객체로부터 계승 받아 사용할 수 있는 기능
- 어떤 클래스가 다른 클래스의 멤버(필드와 메소드)를 그대로 계승하는 기능
- 수퍼클래스(혹은 부모 클래스) : 멤버를 상속하는 클래스
- 서브클래스(혹은 자식 클래스) : 멤버를 상속받는 클래스
- 키워드 “extends”를 사용하여 상속을 정의한다
- class A extends B
  - 클래스 A가 클래스 B로부터 모든 멤버를 상속 받음
  - A : 서블래스, B : 수퍼클래스

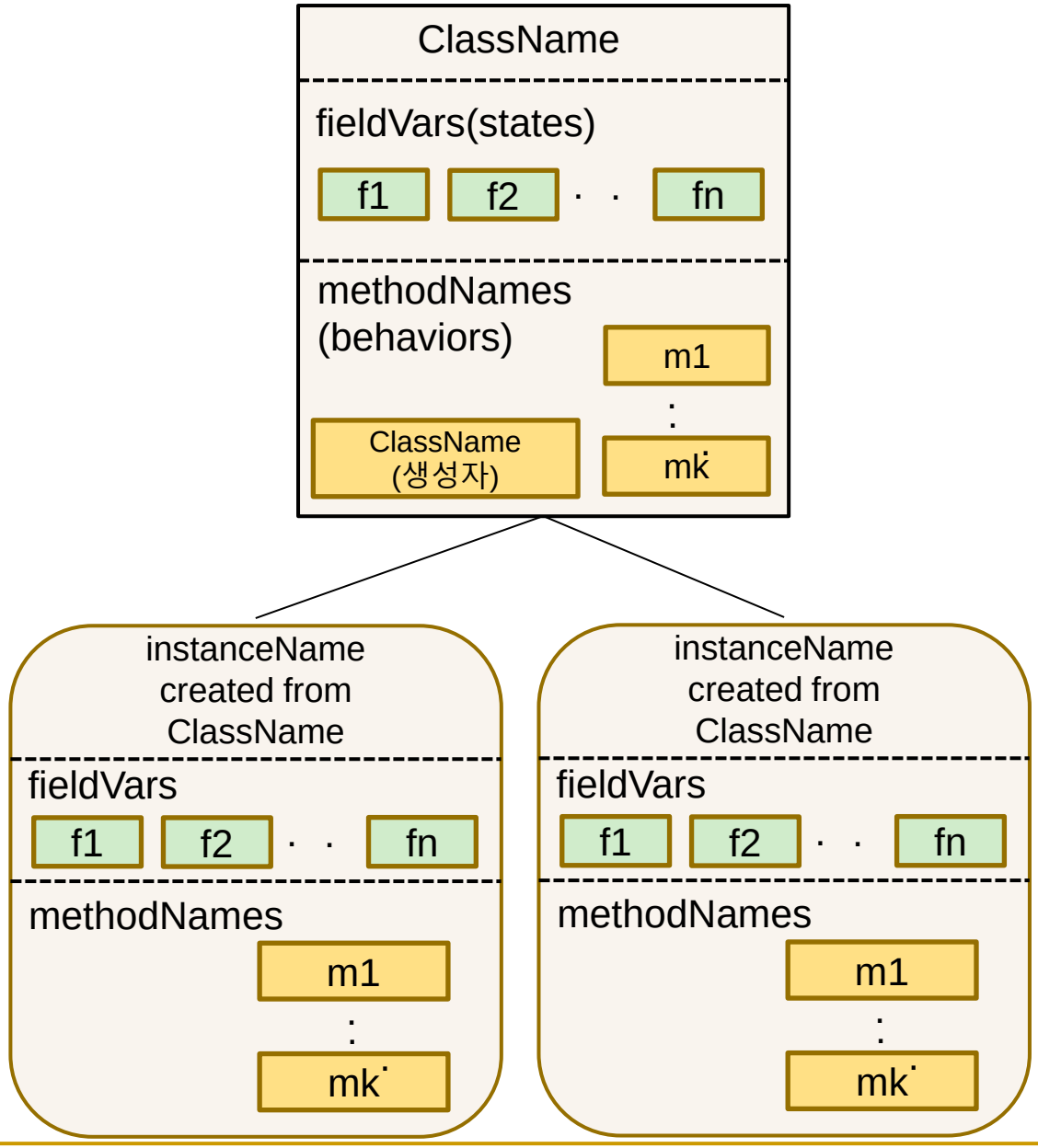


자바 프로그래밍 예

- 시작하기 전에 2개의 작업을 생각해 보자
  - 1. “영희가 계산을 한다“
  - 2. “영희가 사칙연산 계산기를 사용하여 계산을 한다“

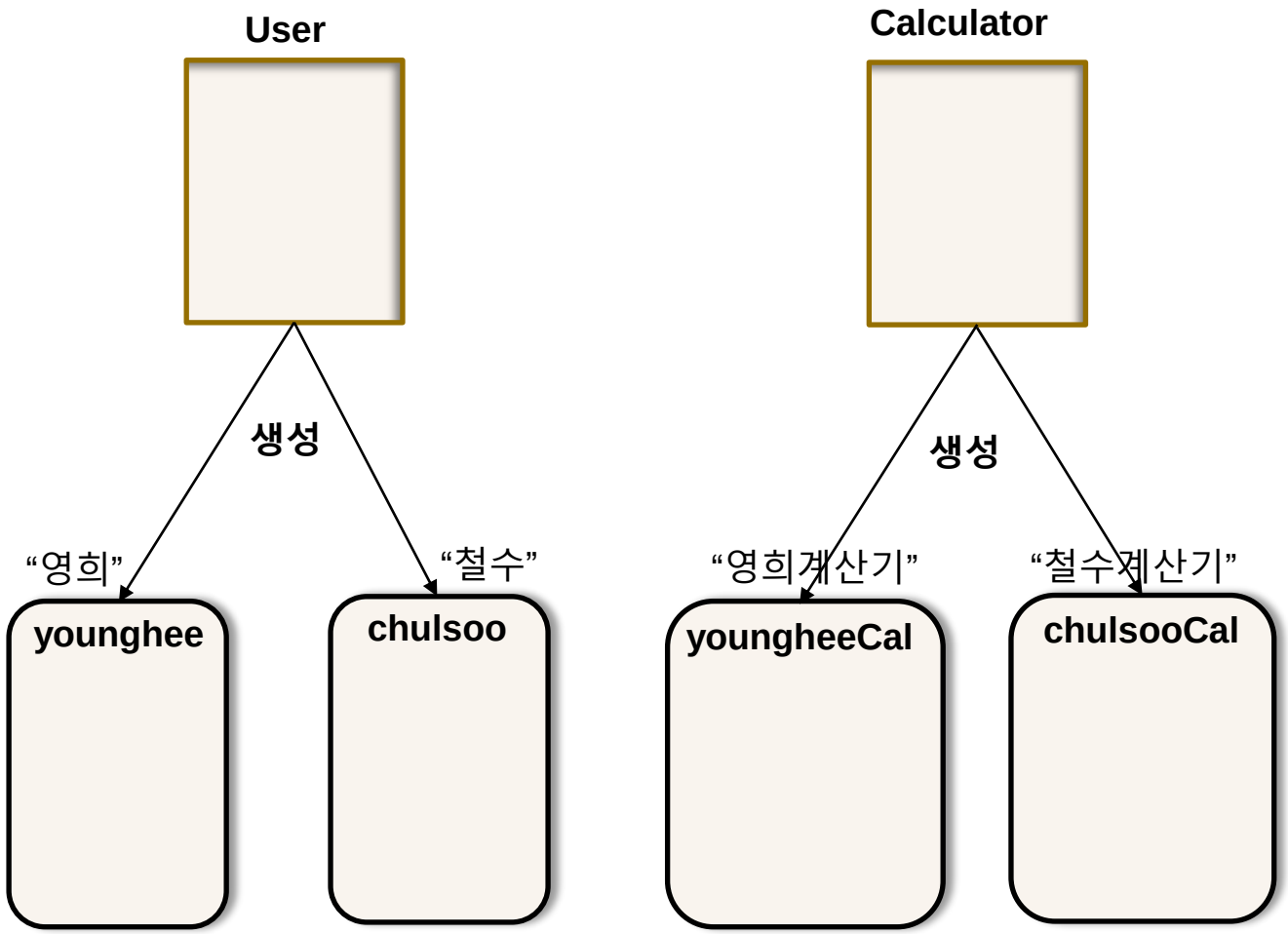
  - 1번 작업과 2번 작업의 차이점을 생각해 보자
    - 추상화 수준
    - **객체의 발견**

  - 2번 작업을 기반으로 자바 프로그래밍 단계를 기술해보자

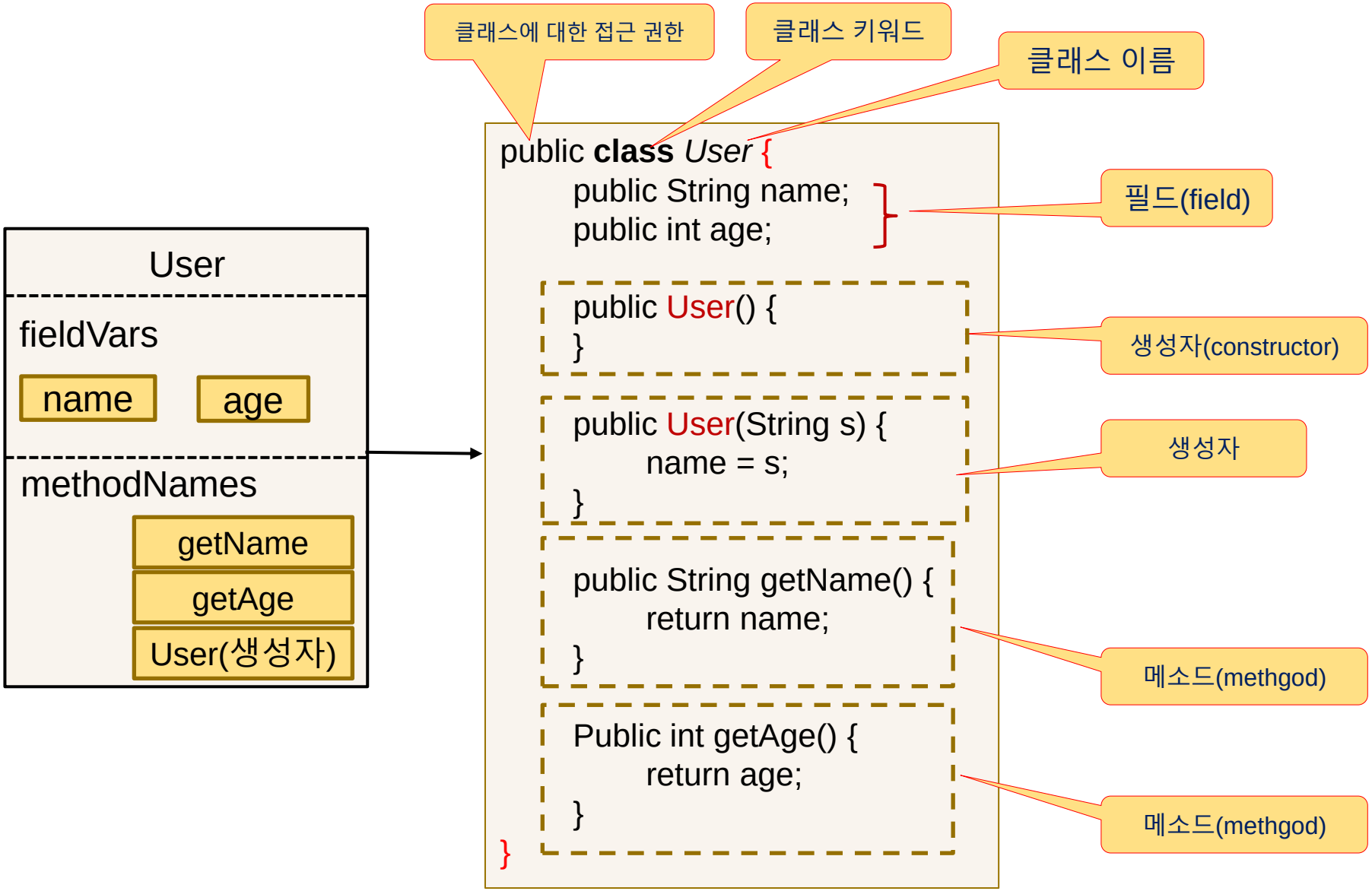

ClassName
fieldVars(states)
f1
f2
fn
methodNames
(behaviors)
m1
mk
ClassName
(생성자)
instanceName
created from
ClassName
fieldVars
f1
f2
fn
methodNames
m1
mk
instanceName
created from
ClassName
fieldVars
f1
f2
fn
methodNames
m1
mk

- 프로그래밍 단계
  - 작업 설정은 2번 작업과 유산한 “영희와 철수가 각자의 계산기로 사칙연산을 수행한다”를 기반으로 프로그래밍 단계를 기술하기로 한다
  - 객체의 발견과 분류
  - 클래스 설정
  - 클래스 설계
  - 실행 클래스와 인스턴스의 생성

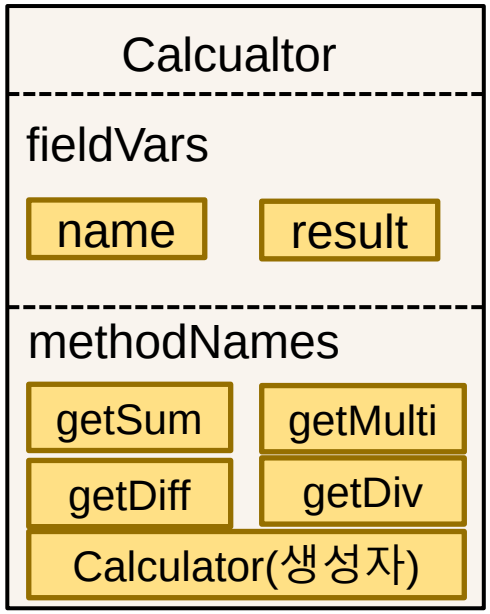
- 1. 객체의 발견과 분류
  - 존재하는 객체 :
    - “영희”, “철수”, “영희계산기”, “철수계산기”
  - 객체의 분류
    - 발견한 객체들을 분석하여 동일한 부류에 속하는 것들을 집단화
    - “영희”와 “철수”가 동일한 부류
    - “영희계산기”와 “철수계산기”가 동일한 부류
- 2. 클래스 인지
  - 2개의 클래스를 인지할 수 있다
    - “영희”와 “철수” 집단을 사용자 → User라고 하자
      - “영희”와 “철수”는 User로부터 생성되어야 할 인스턴스 객체
    - “영희계산기”와 “철수계산기” 집단을 계산기 → Calculator라고 하자
      - “영희계산기”와 “철수계산기”는 Calculator로부터 생성되어야 할 인스턴스 객체


User
Calculator
생성
생성
“영희”
“철수”
“영희계산기”
“철수계산기”
younghee
chulsoo
youngheeCal
chulsooCal

- 3. 클래스 설계
  - 클래스의 정적 속성 설계 → 필드 정의
    - User 클래스와 Calculator 클래스
  - 클래스의 동적 기능 설계 → 메소드 정의
    - User 클래스와 Calculator 클래스
  - User 클래스
    - 정적 속성 설계
      - 사용자 이름과 나이를 정적 속성으로 설정한다 → 2개의 필드
      - 이름에 대해 name 필드를 선언하며 타입은 String으로 선언한다
      - 나이에 대해서는 age라는 필드를 선언하며 int 타입으로 선언한다
      - 이름은 초기값으로 “younghee”와 “chulsoo”를 지정한다
    - 동적 기능 설계
      - 동적 기능으로 “이름얻기(getName)”과 “나이얻기(getAge)”를 설정
    - 생성자(Constructor)
      - 자신의 인스턴스 객체의 생성을 위한 필수 메소드
      - 클래스 이름과 동일한 이름을 가지는 메소드로 리턴값이 없다
      - 오버로딩이 가능하다(2개 이상의 생성자 메소드를 가질 수 있다)


클래스에 대한 접근 권한
클래스 키워드
클래스 이름
public class User {
public String name;
public int age;
public User() {
}
public User(String s) {
name = s;
}
public String getName() {
return name;
}
Public int getAge() {
return age;
}
}
필드(field)
생성자(constructor)
생성자
메소드(methgod)
메소드(methgod)
User
fieldVars
name
age
methodNames
getName
getAge
User(생성자)

- Calculator 클래스
  - 정적 속성 설계
    - 소유자 이름과 연산결과를 저장하는 저장소(변수)를 정적 속성으로 설정한다
    - 소유자이름에 대해 name 필드를 선언하며 타입은 String으로 선언한다
    - 연산결과 저장변수로 result라는 필드를 선언하며 int 타입으로 선언한다
    - 이름은 초기값으로 “youngheeCal”와 “chulsooCal”을 지정한다
  - 동적 기능 설계 → 사칙연산 기능
    - 동적 기능으로 “getSum”, “getDiff”, getMulti”, 그리고 “getDiv”를 설정
    - 누구의 계산기인지 알기 위해서 “getName”을 추가 설정한다
  - 생성자(Constructor)
    - 자신의 인스턴스 객체의 생성을 위한 필수 메소드
    - 클래스 이름과 동일한 이름을 가지는 메소드로 리턴값이 없다
    - 오버로딩이 가능하다(2개 이상의 생성자 메소드를 가질 수 있다)



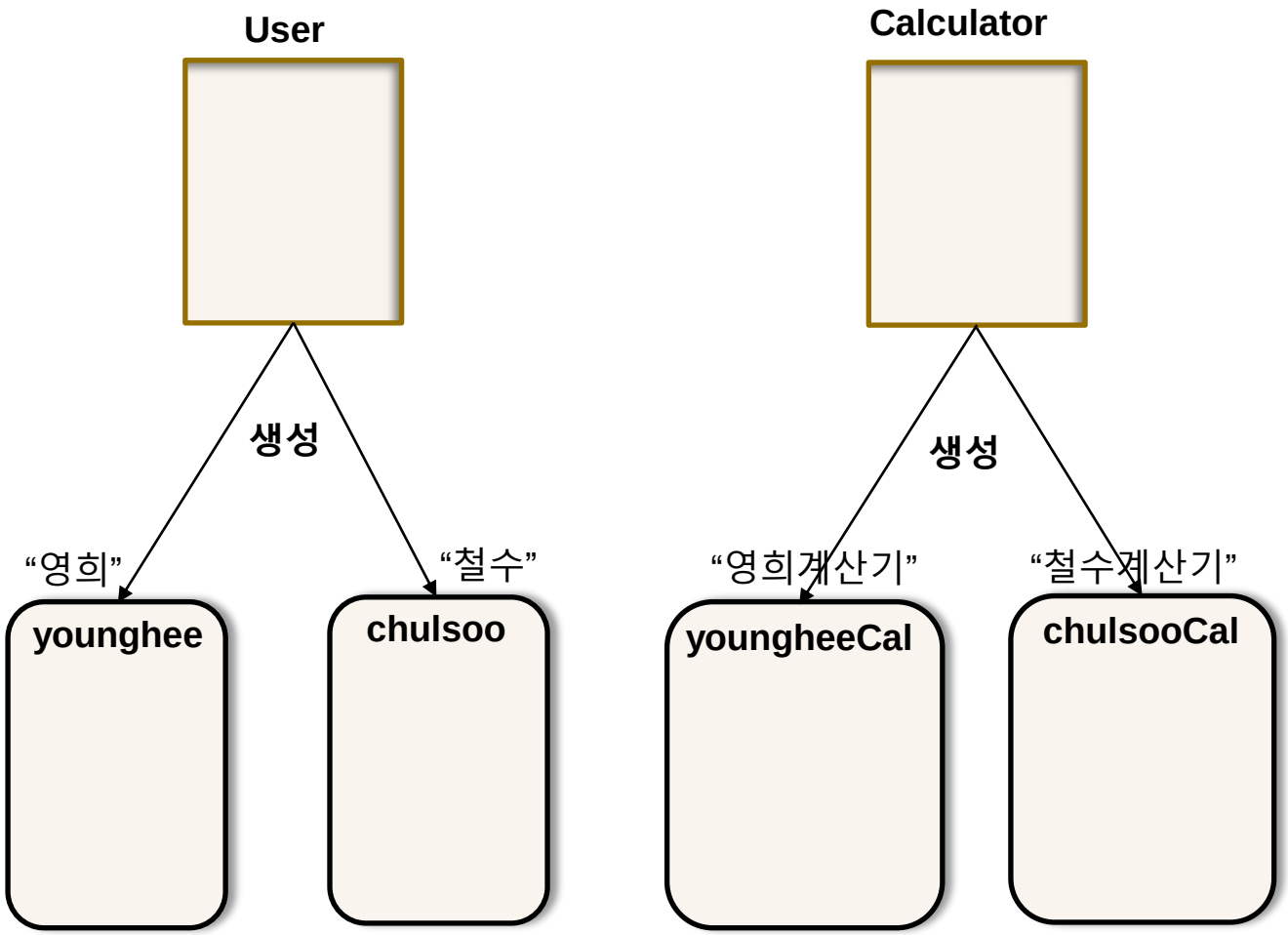

```
public class Calculator {
        public String name;
        public int result;

        public Calculator() {
        }
        public Calculator(String s) {
                name = s;
        }
        public String getName() {
                return name;
        }
        public int getSum(int x, int y) {
                result = x + y;
                return result;
        }
        public int getDiff(int x, int y) {
                result = x - y;
                return result;
        }
        public int getMulti(int x, int y) {
                result = x * y;
                return result;
        }
        public int getDiv(int x, int y) {
                result = x / y;
                return result;
        }

}
```

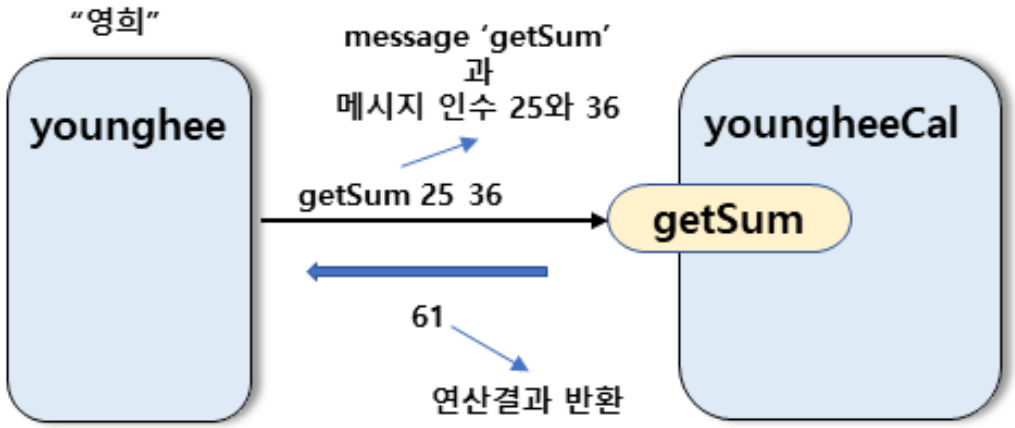
- 4. 실행클래스 설정과 인스턴스 생성
  - 클래스는 프레임(틀)
  - 실제 동작은 클래스로부터 생성되는 인스턴스를 통해 이루어진다
  - 어떤 인스턴스가 존재하는지 파악
    - [STEP-1]에서 ‘younghee’, ‘chulsoo’, ‘youngheeCal’ 그리고 ‘chulsooCal’ 4개의 객체가 필요
    - ‘younghee’와 ‘chulsoo’는 ‘User’ 클래스로부터 생성, ‘youngheeCal’과 ‘chunsooCal’은 ‘Calculator’ 클래스로부터 생성
  - **누가 이러한 인스턴스 생성을 주도할 것인가**, 이는 실행 클래스를 결정하는 문제
  - User 클래스인가? Calculator 클래스인가?
    - 계산기를 사용하는 사용자가 주도해야 하는 것이 합리적 → main 메소드의 선언되는 클래스 설정 문제
    - User 클래스에 main 메소드가 정의되어야 할 것이며 모든 동작의 주도는 User 클래스에서 일어나게 된다.
    - User 클래스는 실행 클래스이며 Calculator 클래스는 API 클래스가 된다
  - 생성자 메소드 설정

```
public static void main (String args[]) {
        User  younghee, chulsoo;

        // 사용자이름 선언
        younghee = new User("영희");

        // User 객체 생성
        chulsoo = new User("철수");

        //계산기이름 선언
        Calculator  youngheeCal, chulsooCal;

        //계산기 객체 생성
        youngheeCal = new Calculator("영희");
        chulsooCal = new Calculator("철수");
}
```

```
public class User {
        public String name;
        public int age;

        public User() {
        }

        public User(String s) {
                name = s;
        }

        public String getName() {
                return name;
        }

        Public int getAge() {
                return age;
        }

        public static void main (String args[]) {

                User   younghee, chulsoo;

                younghee = new User("영희"); //영희 인스턴스
                chulsoo = new User("철수");    //철수 인스턴스

                Calculator   youngheeCal, chulsooCal;
                youngheeCal = new  Calculator("영희");
                chulsooCal = new Calculator("철수");

        }
```

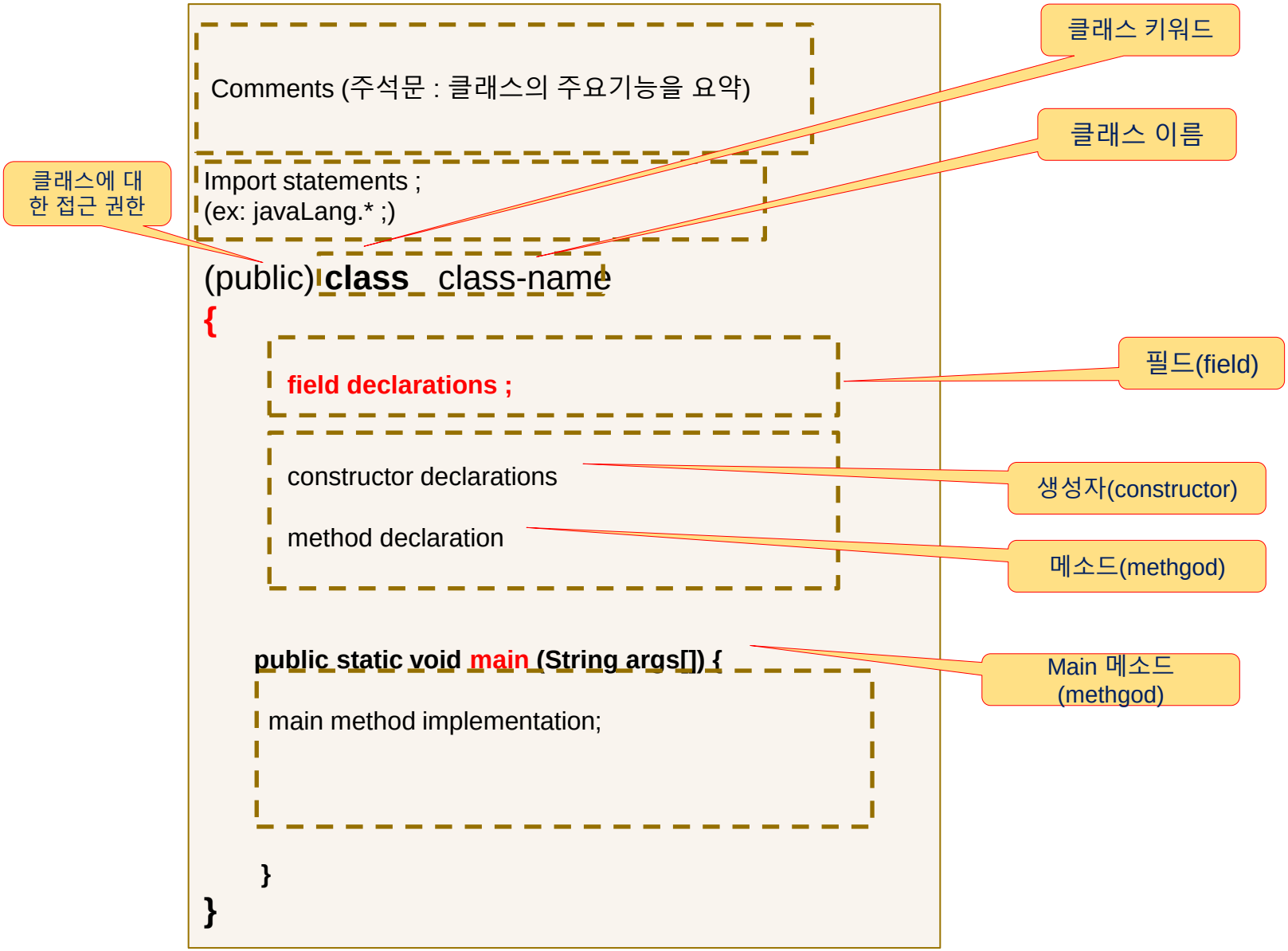
필드 선언


User
생성
“영희”
“철수”
younghee
chulsoo
Calculator
생성
“영희계산기”
“철수계산기”
youngheeCal
chulsooCal



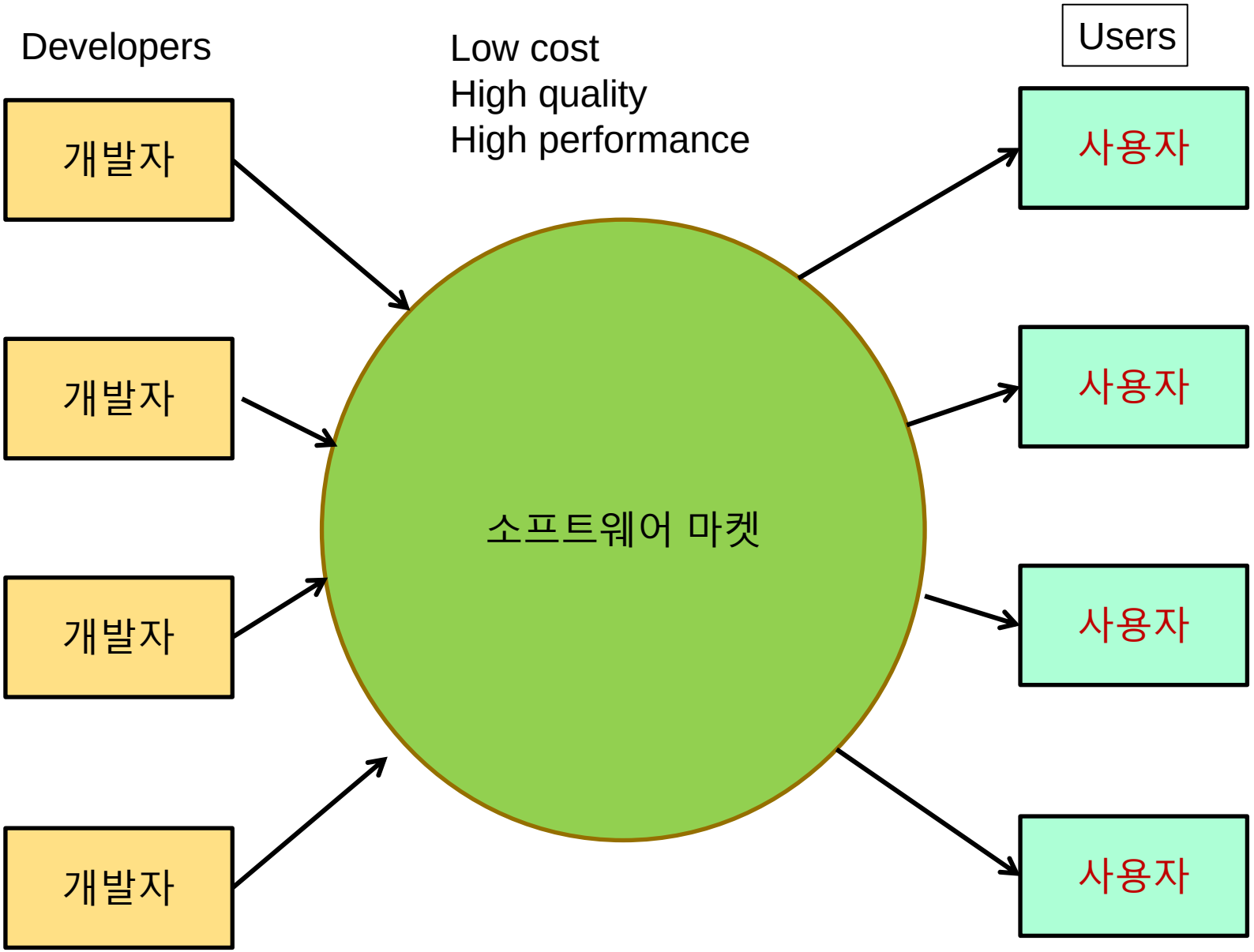

```
public static void main (String args[]) {
        User younghee, chulsoo;
        younghee = new User("영희");
        chulsoo = new User("철수");

        Calculator youngheeCal, chulsooCal;
        youngheeCal = new Calculator("영희");
        chulsooCal = new Calculator("철수");

        int youngheeSum = youngheeCal.getSum(25, 36);
        System.out.println("영희계산기합 = " + youngheeSum);
}
```

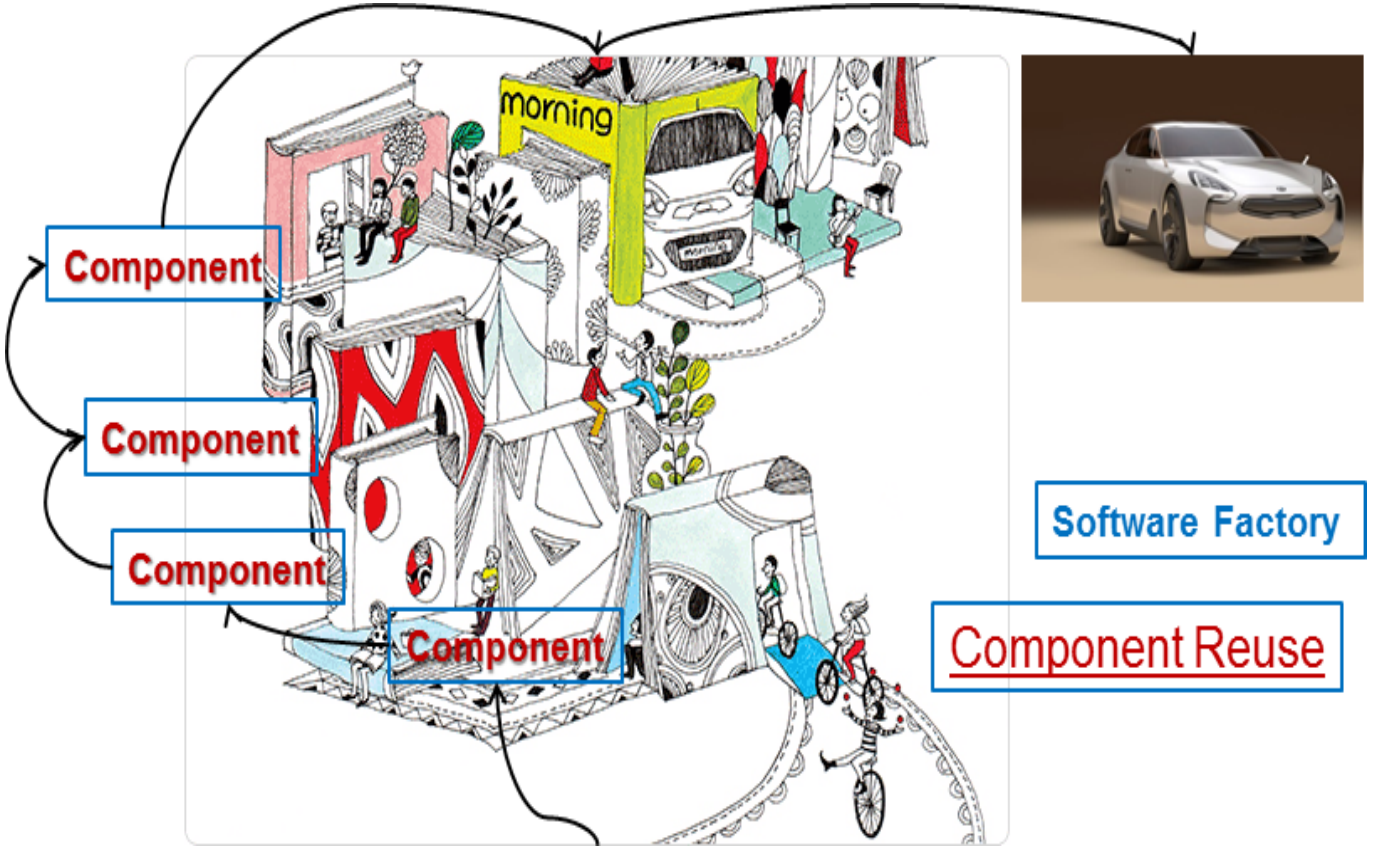
- 메인 메소드의 역할과 자바프로그램의 구조
  - 자신의 작업을 위한 환경구축
  - 객체들의 생성이 완료되면 그 객체들을 사용하여 자신이 하고자 하는 작업을 정상적으로 완료하는 것
  - main 메소드는 구성된 클래스 중 어느 한 클래스에 설정되며 이 클래스가 실행 클래스가 되며, 나머지 클래스들은 API 클래스로서 실행 클래스의 작업을 지원하는 역할
  - 그러므로 마지막 단계가 실행 클래스를 결정하고 실행 클래스 내에 main() 메소드(알고리즘)를 구현하는 것이다.

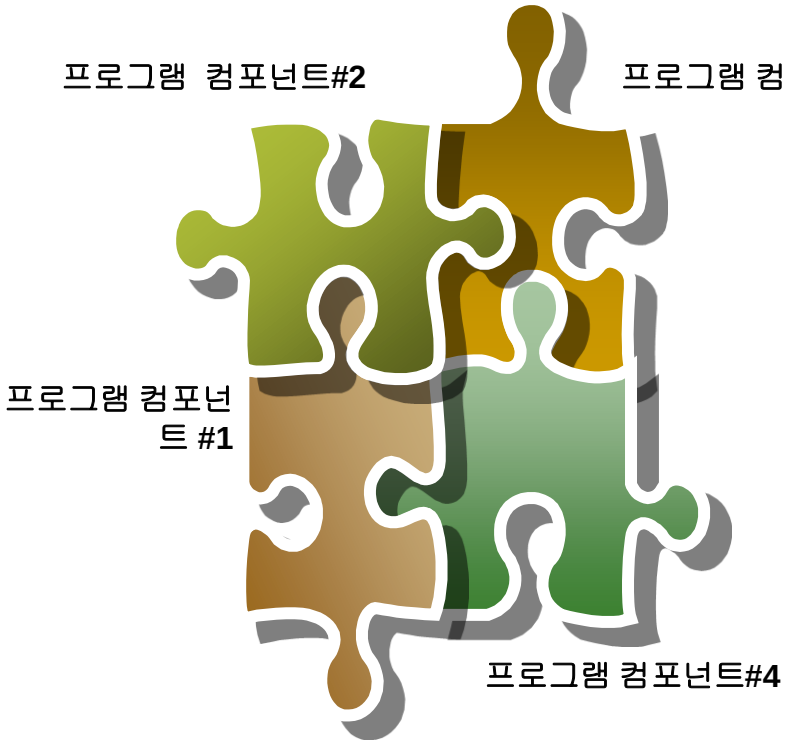

Comments (주석문 : 클래스의 주요기능을 요약)
Import statements ;
(ex: javaLang.* ;)
(public) class class-name
{
field declarations ;
constructor declarations
method declaration
public static void main (String args[]) {
main method implementation;
}
}
클래스 키워드
클래스 이름
클래스에 대한 접근 권한
필드(field)
생성자(constructor)
메소드(methgod)
Main 메소드 (methgod)

왜 객체지향 프로그래밍?

- 소프트웨어를 바라보는 2개의 눈
  - 개발자
    - Low Cost → 개발 기간 단축, 적은 man-year
    - 고 생산성(high productivity)을 추구
  - 사용자
    - 저가의(Low Price), 고성능(High Performance)
  - 궁극적으로
    - 저비용, 고성능을 추구


Developers
Low cost
High quality
High performance
Users
개발자
개발자
개발자
개발자
소프트웨어 마켓
사용자
사용자
사용자
사용자


Component
Component
Component
Component
morning
Software Factory
Component Reuse


프로그램 컴포넌트#2
프로그램 컴포넌트#3
프로그램 컴포넌트 #1
프로그램 컴포넌트#4