JAVA 기초입문과정

CHAPTER06 클래스와 객체1

Contents


PART
01
“
객체 지향 프로그래밍,
클래스와 인스턴스
”

객체 지향 프로그래밍과 클래스

1. 객체 지향 프로그래밍이란?

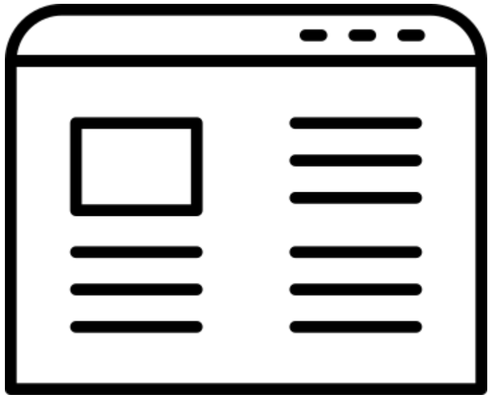
프로그래밍의 목적 -> 실세계에서 발생하는 일을 프로그램으로 만드는것 -> 실세계의 일을 모델링

물리적/추상적으로 존재하는 모든 객체



사람 자동차 책

모델링



객체의 고유한 특성

객체의 고유한 행동

< 현실 세계 >

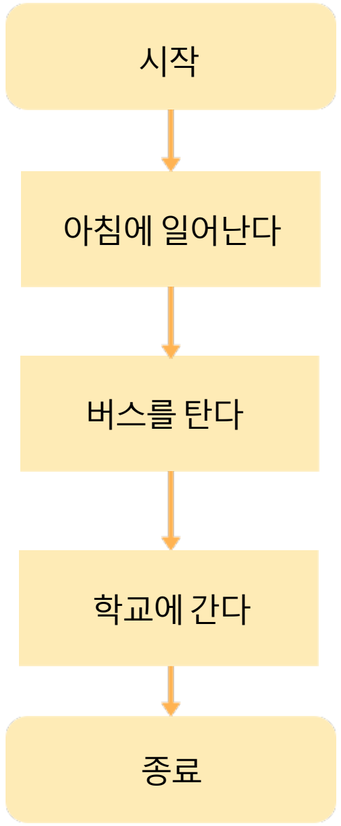
< 프로그램 >

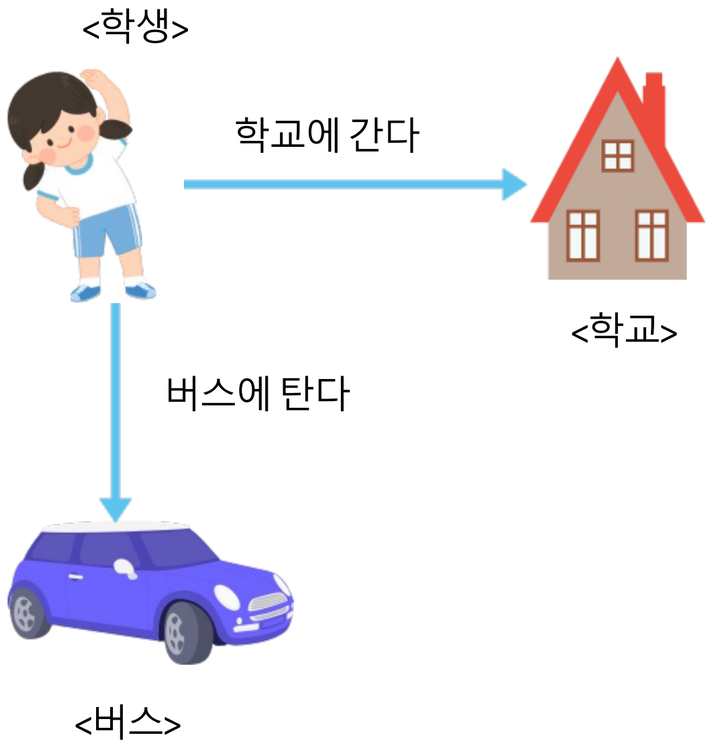
-> 객체지향프로그래밍은 객체를 기반으로 프로그래밍한다

객체 지향 프로그래밍

* 절차 지향 프로그래밍과 객체 지향 프로그래밍의 차이

실세계 모델링 종류





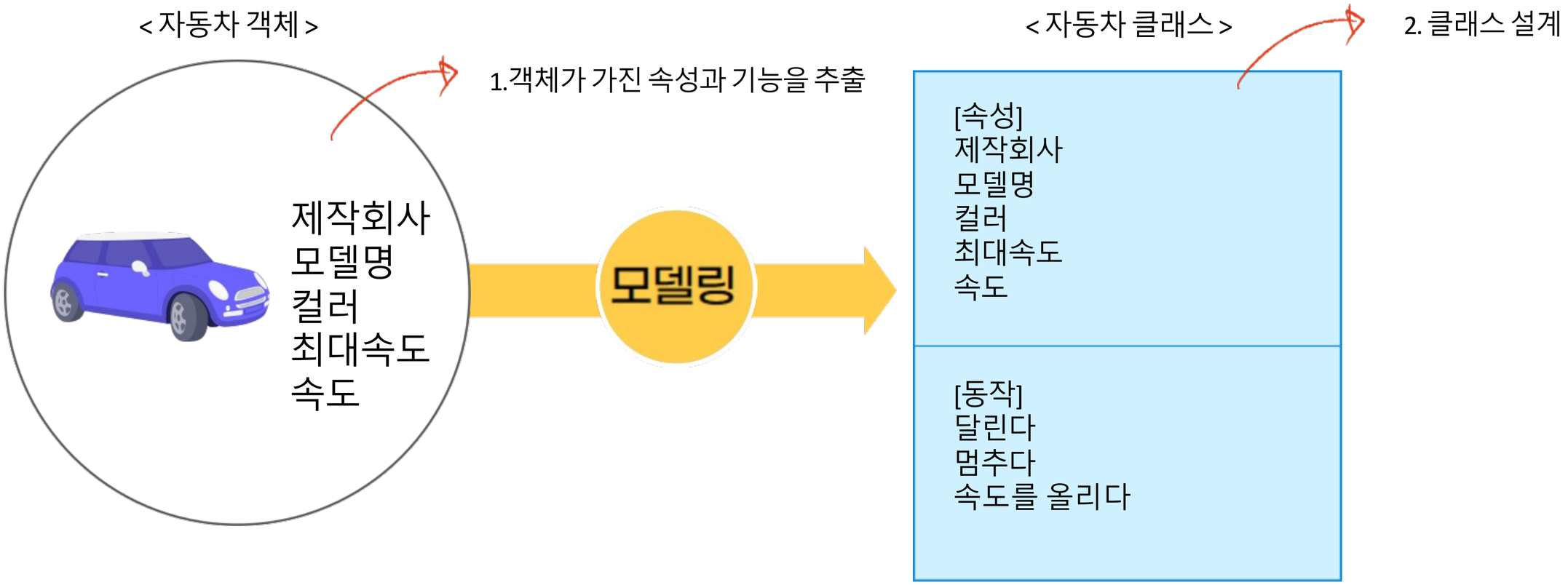
-> 수행작업을 순차적으로 표현

-> 수행작업을 객체들간의 상호작용으로 표현

2. 클래스란?

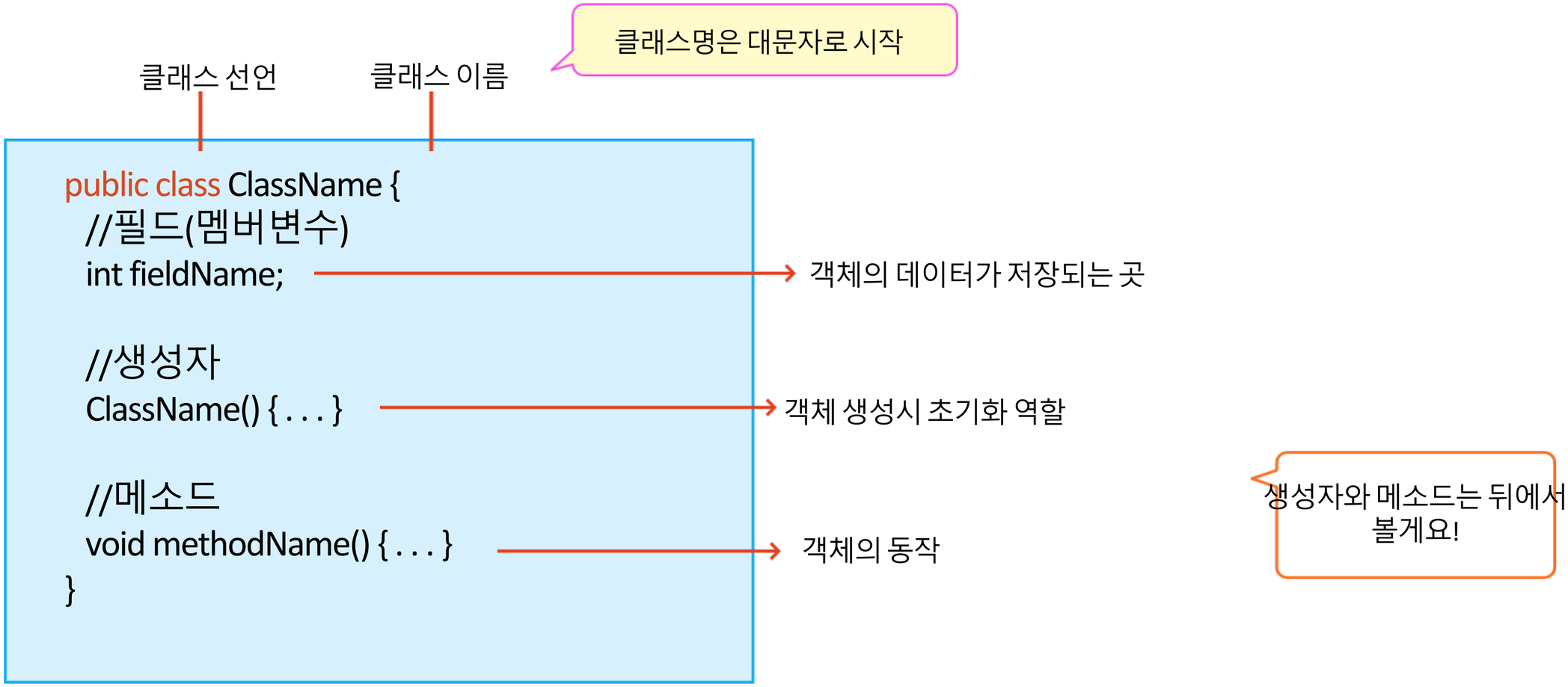
-> 객체의 속성과 기능을 코드로 구현한 것

객체지향 프로그램은 클래스를 기반으로 만든다



* 클래스 구성

- 구성요소: 멤버변수(속성), 메소드(기능), 생성자



3. 클래스 속성을 구현하는 멤버 변수


```
public class Car {
    String company;
    String model;
    String color;
    int maxSpeed;
    int speed;
}
```

* 멤버 변수 만들기
1. 속성에따라 알맞은 자료형을 사용
2. 속성을 표현할수 있는 변수명 선언

클래스와 인스턴스

* 클래스를 사용하는 방법

1 클래스 내부에 main함수를 넣는다

```
public class Car {

 public static void main(String[] args) {
  // 메소드 호출 및 테스트
  }
}
```

-> 이전까지 사용했던 방식
-> 이렇게 사용하면 모든 클래스에 메인함수를 넣어야한다.

이전에 인스턴스를 만들지 않았음에도
멤버변수와 메소드 사용이 가능했던 이유는
모두 static으로 선언했기 때문이다

2 클래스에서 메인함수를 분리한다 + 클래스 객체를 생성한다

```
public class Car {
   String company = "현대 자동차";
   String model = "그렌져";
   String color = "검정";
   int maxSpeed = 350;
   int speed;
}

public class CarExample {
   public static void main(String[] args) {
     Car myCar = new Car();
     myCar.speed = 60;
   }
}
```

객체를 생성

클래스와 인스턴스

1. 클래스 생성하기

설계 대상

설계도

클래스 객체 생성

< 객체 >

< 클래스 >

< 인스턴스 >

모델링

```
public class Car {
   String company = “현대 자동차”;
   String model = “그렌져”;
   String color = “검정”;
   int maxSpeed = 350;
   int speed;
}
```

```
Car myCar = new Car();
myCar.speed = 60;
```

설계도 만으로는 클래스를 사용할수없다!

2. 클래스와 인스턴스

설계도

< 클래스 >

```
public class Car {
   String company = “현대 자동차”;
   String model = “그렌져”;
   String color = “검정”;
   int maxSpeed = 350;
   int speed;
}
```

클래스 객체 생성

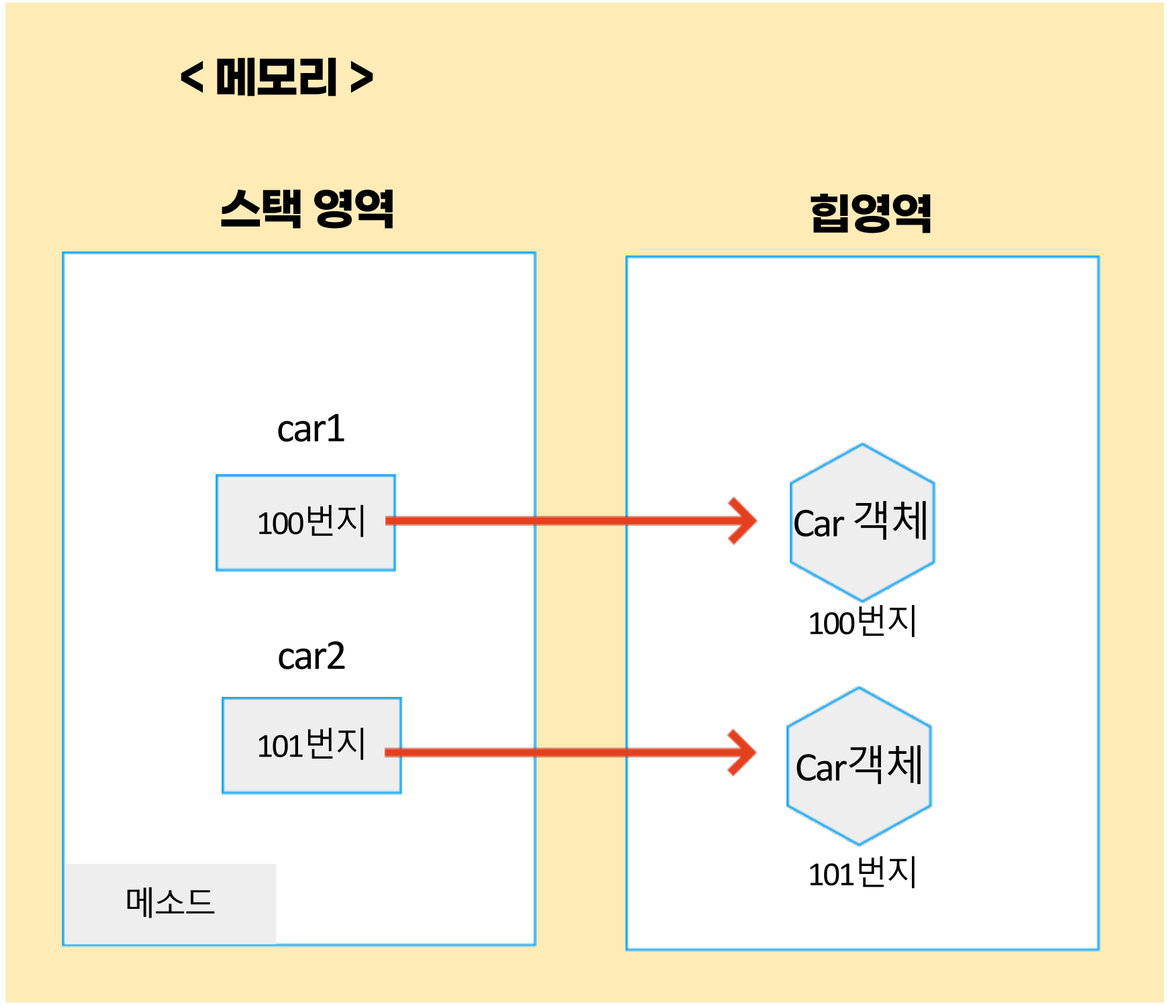
< 인스턴스 >

```
Car car1 = new Car();
car1. speed = 60

Car car2 = new Car();
car2.studentName = 100;
```

인스턴스는 여러개 만들 수 있다.
자동차1 자동차2 자동차3 ....

* 인스턴스와 힙 메모리



변수가 객체의 주소를 참조한다

```
Car car1 = new Car();
System.out.println( car1  )

Car2 car2 = new Car();
System.out.println( car2  )
```

[실행결과]
Car@15321353
Car@4542455

주소값

기초자료형은 변수에 값을 담지만, 참조자료형은 주소를 담는다

3. 참조변수 사용하기

1. 인스턴스를 생성한다

```
Car car1 = new Car();
```

2. 도트연산자를 사용하여 멤버변수와 메소드를 사용한다,

```
car1. setSpeed;
car1.run();
```

인스턴스를 참조하여 사용한다

PART
02

“생성자”

PART 02

생성자

1. 생성자란?

1) 클래스 객체를 생성한다
2) 멤버변수를 초기화한다
3) 무조건 한개 이상 가진다

```
public class  Student {
  int studentID;
  String studentName;
  int grade;
  String address;


}
```

디폴트 생성자

생성자 호출

```
Student studentAhn = new Student();
```

생성자 메소드이름은 클래스 이름과 같다

객체의 주소를 반환한다

디폴트 생성자 만든적 없는데??
자바가 자동으로 만들었음

2. 디폴트 생성자

- 생성자 이름은 클래스와 같다
- 반환 타입은 없다.
- 반환타입은 없는데, 생성한 객체의 주소를 반환한다

생성자의 목적은 무엇일까?
-> 객체생성 + 멤버변수 초기화

```
public class  Student {
  int studentID;
  String studentName;
  int grade;
  String address;


}
```

디폴트 생성자

```
public Student( ){ }
```

자바가 자동으로 생성하는 생성자

3. 생성자 만들기

- 개발자가 직접 생성자를 구현할 수 있다
- 멤버변수의 값을 초기화 하고 싶을때 직접 만든다

```
public class  Student {
  int studentID;
  String studentName;
  int grade;
  String address;

public Student( Stri
  studentName = name;
  }
}
```

사람이름을 매개변수로 입력받아서
Student 클래스를 생성하는 생성자

```
// 생성자 테스트하기

Student student1 = new Student("홍길동"); //정상

Student student2 = new Student();
```

오류발생!!!

생성자를 직접 만들어서 사용했기 때문에
이제 자바가 디폴트 생성자를 만들지 않는다

4. 생성자 오버로드

- 생성자가 두 개 이상 제공되는 경우를 말한다
- 같은 이름의 메소드를 여러개 선언하는 것
- 단, 매개변수의 타입, 개수, 순서 중 하나가 달라야함

생성자 메소드 뿐만 아니라 모든 메소드는 오버로드 할 수 있다

```
// 1번 생성자 만들기
public Student( ){ }

// 2번 생성자 만들기
public Student(int id){
  studentId = Id;
}

// 3번 생성자 만들기
public Student(int id, String name){
  studentId = Id;
  studentName = name;
}
```

```
// 1번 생성자 테스트하기
Student student1 = new Student();

// 2번 생성자 테스트하기
Student student2 = new Student( 20201255 );

// 3번 생성자 테스트하기
Student student3 = new Student( 20201255 , "홍길동" );
```

입력한 매개변수에따라 생성자를 호출한다


PART
03
“
메소드
”

PART 03

메소드

1. 함수

-> 하나의 기능을 수행하는 코드

어떤 기능을 미리 구현해 놓고 필요할 때마다 호출하여 사용 할 수 있다.

더하기 기능을 만들어보자 (두 수를 더하여 결과를 보여주는 기능)

입력
숫자, 성자, 거리 등

더하기 함수

출력
숫자의 합, 성적의 합, 거리의 합 등

입력
num1, num2

add 함수
result = num1 + num2

출력
result

* 함수 만들기
1. 입력값 정의
2. 기능을 수행할 코드 만들기
3. 반환값 정의

java 에서는 함수를 메소드라고 부른다

2. 메소드

객체의 동작(기능)을 담당

리턴타입 메소드이름 매개변수

```
public int add ( int num1, int num2 ) {
    . . .
return result;
}
```

기능을 수행할 코드

리턴타입과 반환값의 타입이 일치해야함

반환값

3. 매개변수와 리턴값

리턴타입이 void가 아니면 반드시 블록안에서 retrun 해야한다

타입: 모든 기초자료형, 참조자료형, void

타입: 모든 기초자료형, 참조자료형, 빈값
개수: 0개~n개

```
public int function ( int num1, int num2 ) {
. . .
return result;
}
```

*의미
1. 결과를 반환한다
2. 메소드를 종료한다

* 리턴타입 void의 의미

```
public void function () {
  return ;
}
```

-> return을 생략해도된다
-> 결과값이 없다

4. 메소드 사용하기

메인함수

```
// 메소드를 호출하는 쪽
public static void main(){
  int nun1 = 10, num2 = 20;
  int result = add ( num1, num2 );
}
```

* 매개변수, 리턴값 타입과 개수가 일치해야함

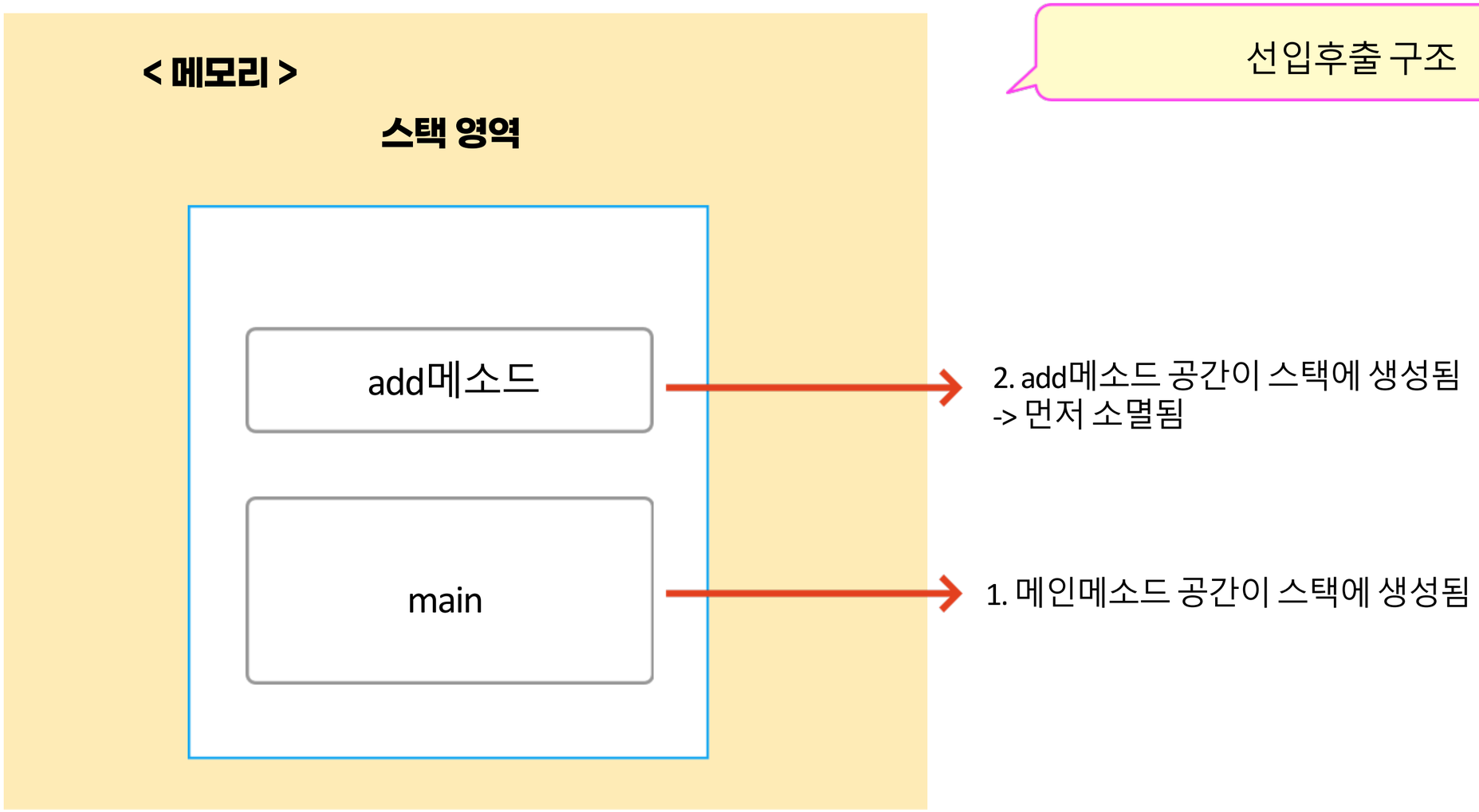
메소드이름 (매개변수) ; 으로 호출

ADD함수

```
// 구현 메소드
public static int add(int n1, n2){
  int result = n1 + n2;
  return result;
}
```

* 리턴타입과 리턴값의 자료형이 일치해야함

* 함수호출과 스택 메모리


< 메모리 >
스택 영역
add메소드
main
선입후출 구조
2. add메소드 공간이 스택에 생성됨
-> 먼저 소멸됨
1. 메인메소드 공간이 스택에 생성됨

PART
04

“인스턴스 멤버, 정적 멤버”

PART 04

this 예약어

1. this

- this.멤버 형태로 멤버변수에 접근할 때 사용한다

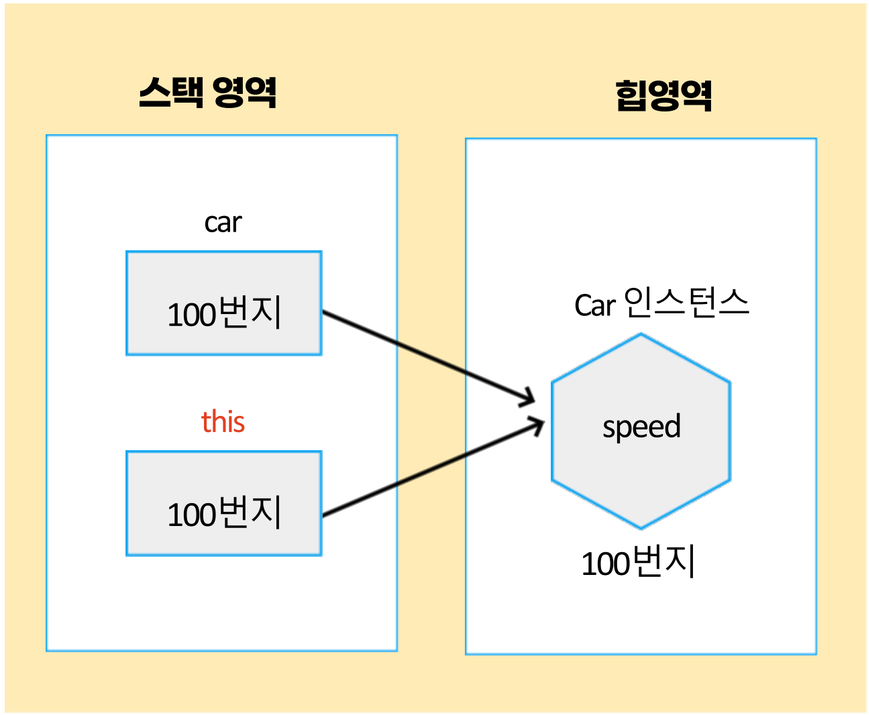
* 사용방법

```
public class Car {
   int speed;


 public void setSpeed ( int speed ) {
   this.speed = speed;
  }
}
```

인스턴스 자신을 가리킨다

메모리 구조



PART 04

static 변수

1. staitc 변수

- 인스턴스는 각자 고유의 값을 가진다
- 클래스에서 공통으로 사용하는 멤버는 static 변수로 선언한다

* static 변수 선언하기

```
public class Calculator {
  static double pi = 3.14159;
  String color;
}
```

모든 객체가 공유하는 변수이다

* static 변수 사용하기

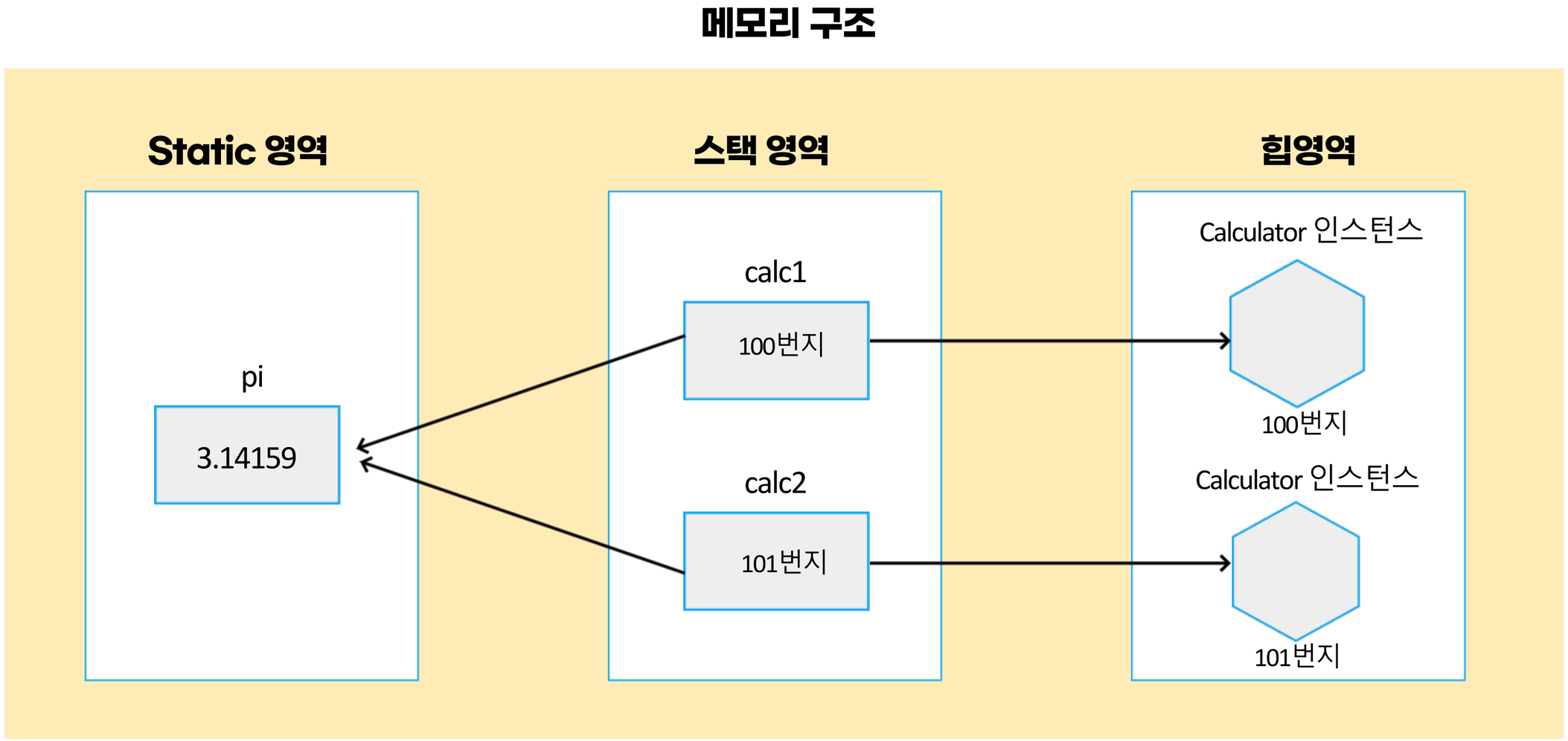
```
public staitc void main(){
   Calculator.pi;
}
```

-> 출력

-> 클래스이름.변수명 으로 접근

static 변수

* 메모리 구조에서 static 변수



static 변수

2. static 메소드

- static 메소드에서는 static변수만 사용할 수 있다.
- statis 메소드에서는 인스턴스변수는 사용할 수 없다.

* static 메소드 선언하기

```
public class Student {
   static int studentCount = 1000;
   int id;
   String name;

   public static int setStudentCount(int cnt){
   studentCount  = cnt;
   id = 11;
  }
}
```

static 메소드에서 인스턴스멤버는 사용할수없다

* static 메소드 사용하기

```
public staitc void main(){
Student.setStudentCount(10);
}
```

-> 클래스이름.메소드로 사용

PART 04

final 필드와 상수

final 타입 필드 [=초기값];

- 초기값이 저장되면 최종적인 값이 되어 실행 중에 수정할 수 없다.
  - 필드 선언시 초기값 대입
  - 생성자에 초기값 대입

static final 타입 상수 [=초기값]

- 객체 마다 저장할 필요가 없고, 여러 개의 값을 가져도 안됨.

PART
05

“
정보은닉
”

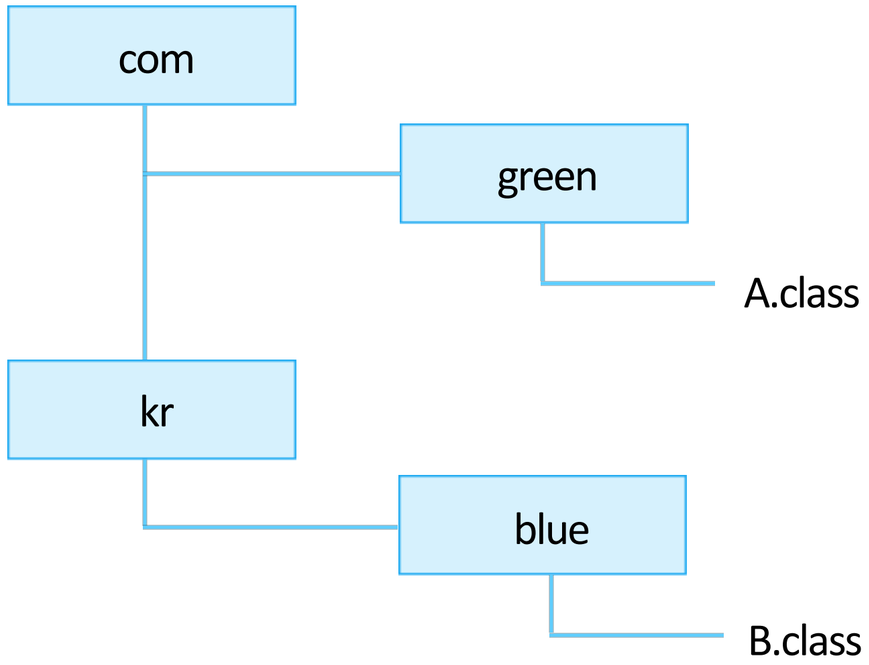
1. 패키지

패키지

- 소스파일의 폴더 개념
- 소스파일의 이름이 동일하더라도 패키지가 다르면 다른 클래스로 인식

※ 패키지 구조

상위패키지.하위패키지.클래스



2. 접근제한자

- 클래스 및 멤버를 공개/비공개 처리한다.
- 이를 통해 다른클래스에서의 접근을 제한한다.

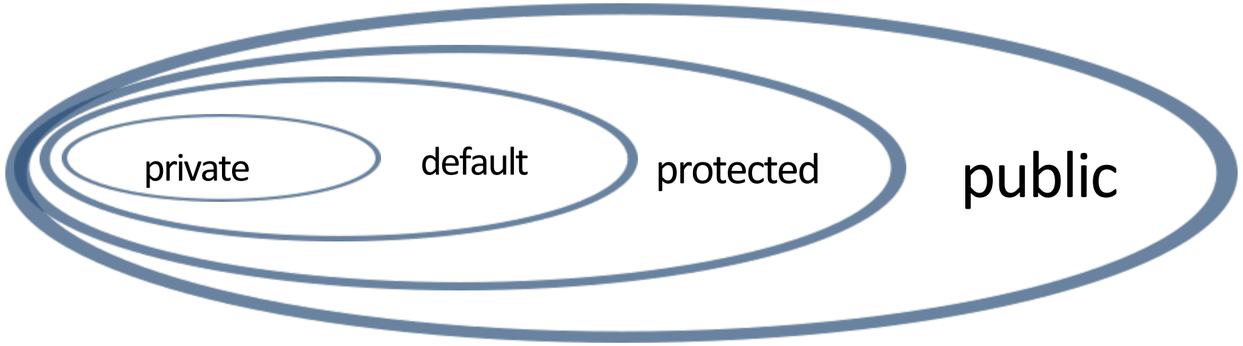
● 예시

```
public class Student {
   int studentID;
   private String studentName;
   int grade;
   String address;
}
```

```
public class StudentTest {
   public static void main(String[] args) {
      Student studentLee = new Student();
      studentLee.studentName = "안연수";
   }
}
```

-> 에러발생

※공개범위



접근제한자 종류	사용범위
private	오직 해당 클래스 내
default	동일 패키지 내 클래스
protected	동일 패키지 내 클래스 또는 상속 클래스
public	모든 타 클래스

3. 정보은닉

-> 변수나 메소드를 private로 선언하여 외부에서 직접 접근하지 못하도록 하는 것

그럼 어떻게 변수에 접근하지?

```
public class MyDate {
private int day;

public void setDay(int day){
    this.day = day;
}
 public int getDay(int day){
    return day;
}
}
```

public 메소드를 통해 클래스 멤버에 접근한다

```
public class MyDateTest {
 public static void main(String[] args) {
    MyDate date = new MyDate();
    date.setDay(10);
 }
}
```

멤버변수별로 get/set 메소드를 만든다

* 수동으로 만들지 말고,
이클립스의 generate 기능을 사용하자

3. 정보은닉

-> 변수나 메소드를 private로 선언하여 외부에서 직접 접근하지 못하도록 하는 것

그럼 어떻게 변수에 접근하지?

```
public class MyDate {
private int day;

public void setDay(int day){
    this.day = day;
}
 public int getDay(int day){
    return day;
}
}
```

public 메소드를 통해 클래스 멤버에 접근한다

```
public class MyDateTest {
 public static void main(String[] args) {
    MyDate date = new MyDate();
    date.setDay(10);
 }
}
```

멤버변수별로 get/set 메소드를 만든다

* 수동으로 만들지 말고,
이클립스의 generate 기능을 사용하자

Thank You!