
JAVA
객체 지향 프로
그래밍

1 객체지향이란?

1. 객체

2. 메시지

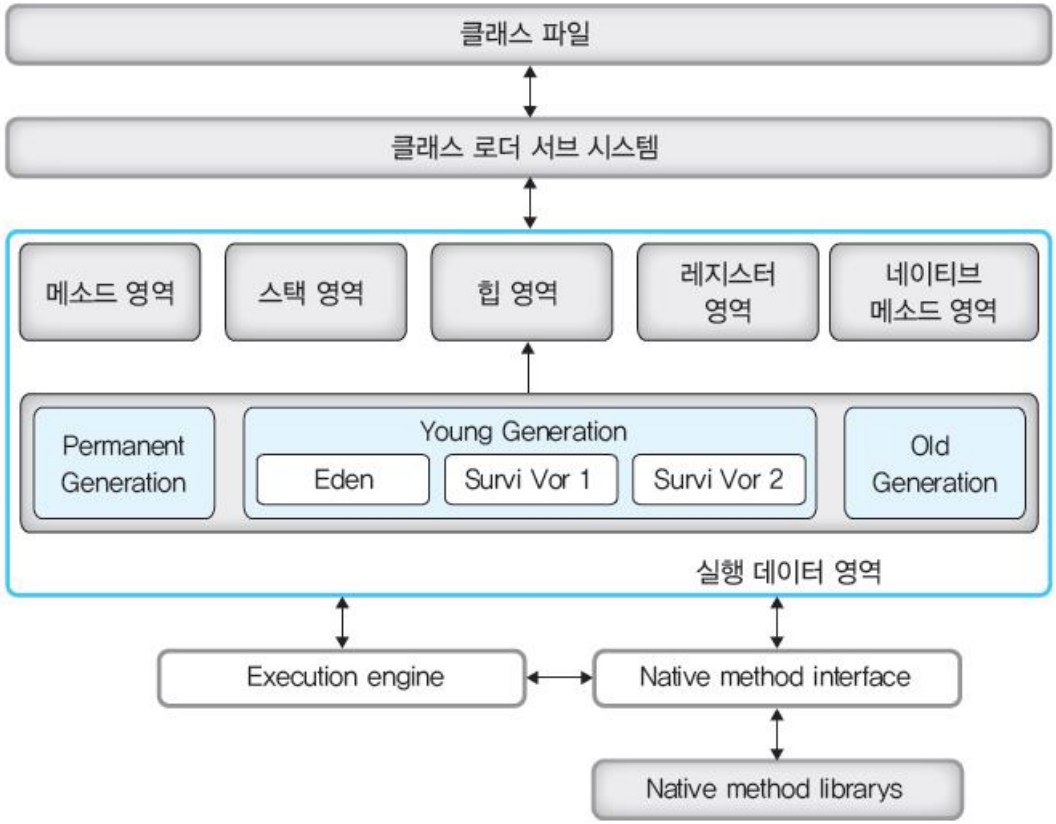
3. 클래스

4. 객체 지향의 장점

5. String 클래스

자바에서 JVM의 역할은 무엇일까?

구조로 알아보는 JVM의 동작 원리



- JVM의 내부 구조를 큰 형태로 분리

1. 클래스 로더 서브 시스템
2. 실행 데이터 영역
3. Execution Engine

자바에서 JVM의 역할은 무엇일까?

구조로 알아보는 JVM의 동작 원리

① 클래스 파일

- **개발자가 만드는 혹은 이미 만들어진 프로그램**
- 개발할 자바 프로그램은 파일 확장자가 '.java'
- '.java' 파일이 자바 컴파일러에서 컴파일 과정을 거치면 '.class' 파일이 생성
- 컴파일 과정을 거쳐 생성된 클래스 파일은 JVM에서 실행 가능
- 클래스 파일이 서로 유기적으로 동작하면서 프로그램은 자기의 기능을 수행

자바에서 JVM의 역할은 무엇일까?

구조로 알아보는 JVM의 동작 원리

② 클래스 로더 서브 시스템

- **자바 클래스 파일들은 OS에서 직접 동작하는 것이 아니라 JVM 위에서 동작**
- **JVM은 실행할 클래스 파일을 읽고, JVM 메모리에 올려놓는 과정이 필요**
- 이 과정을 클래스 로딩(Class loading)이라 하며 JVM의 클래스 로더 서브 시스템이 담당
- 클래스 로더 서브 시스템을 줄여서 클래스 로더라 부름
- 동적 로딩 : 프로그램을 실행하는 도중에 새로운 클래스를 로딩할 수 있음

1. 로드 타임 동적 로딩(Load time dynamic loading) : 프로그램 실행 초기에 클래스를 로딩하는 것
2. 런타임 동적 로딩(Runtime dynamic loading) : 프로그램 실행 중간에도 클래스를 로딩하는 것

자바에서 JVM의 역할은 무엇일까?

▶ 구조로 알아보는 JVM의 동작 원리

③ 실행 데이터 영역

- **클래스 로더로부터 분석된 데이터를 저장하고 실행 도중 필요한 데이터를 저장하는 영역**
- 메모리에 올라간 클래스, 객체, 변수들이 저장되는 곳
- 데이터의 목적과 종류에 따라서 메모리를 효율적으로 관리하기 위해 5개 영역으로 구분

④ 메소드 영역

- **클래스 로더에 의해서 로딩된 클래스가 저장되는 곳**
- JVM에서 클래스를 실행하면 메소드 영역에서 클래스 정보를 복사
- 메소드 영역은 JVM 메모리 영역 중 가장 먼저 데이터가 저장되는 영역

자바에서 JVM의 역할은 무엇일까?

구조로 알아보는 JVM의 동작 원리

⑤ 스택 영역

- **호출된(실행된) 메소드 정보가 저장되는 곳, 실행이 끝나면 저장된 정보는 삭제**
- 메소드가 실행될 때마다 **저장되는 메소드 정보에는 매개변수, 지역 변수** 등
- 이 영역에서는 스택을 데이터 관리 방법으로 사용
- **스택은 LIFO(Last In First Out) 방식으로 동작**
- 메소드의 호출 정보나 호출 순서 등을 추적하기 편리

자바에서 JVM의 역할은 무엇일까?

구조로 알아보는 JVM의 동작 원리

⑥ 힙 영역

- JVM의 실행 데이터 영역 중에서 가장 중요한 역할을 담당
- **객체는 클래스가 실행될 때 생성되어서 힙 영역에 저장**
- 힙 영역은 JVM에서 가장 중요한 데이터를 저장함과 동시에 세밀한 관리가 이뤄지는 곳
- 힙 영역은 동적으로 데이터가 생성되고 소멸되는 영역
- 클래스를 실행하면 데이터가 저장되는 영역, 프로그램 실행에 영향을 미칠 수도 있음

자바에서 JVM의 역할은 무엇일까?

구조로 알아보는 JVM의 동작 원리

⑦ 레지스터 영역

- 현재 JVM이 수행할 명령어의 주소를 저장하는 메모리 공간

⑧ 네이티브 메소드 스택 영역

- 네이티브 메소드 : OS의 시스템 정보, 리소스를 사용하거나 접근하기 위한 코드
- 이러한 매개변수나 지역 변수 등 네이티브 메소드를 위한 영역으로 이에 대한 정보가 저장
- 자바 프로그램과 OS 사이에 JVM이 존재, 자바 프로그램은 시스템에 직접 접근하기 어려움
- JNI(Java Native Interface) API를 사용하면 자바 프로그램에서 OS 시스템에 대한 접근이 가능

객체 지향 개발 방법론

프로그램 개발에 있어서 주요 관점은 개발과 유지 보수의 전 과정에서 비용을 줄이고 품질을 향상시키는데 있다.

" 코드의 재사용을 높이자"

프로그램 개발에서 코드의 재사용을 높이려면 코드가 읽기 쉽고 수정도 쉬워야 한다. 이런 코드가 유지 보수 비용을 줄일 수 있다. 이와 같이 재사용성을 고려한 개발 방법론이 객체지향 방법론(프로그램 개발 방법론이란 프로그램을 생산하는데 필요한 반복적인 과정을 정리한 것.)이다.

- 구조적 개발 방법론(절차적 개발 방법론)
- 구조적 개발 방법론의 설계 핵심은 기능을 수행하는 함수가 데이터를 조작하거나 사용하는 방법이다. 다양한 함수가 데이터를 공유하며 기능을 수행하기 때문에 데이터의 변경은 곧 데이터를 사용하는 코드(함수)의 변경으로 이어진다.

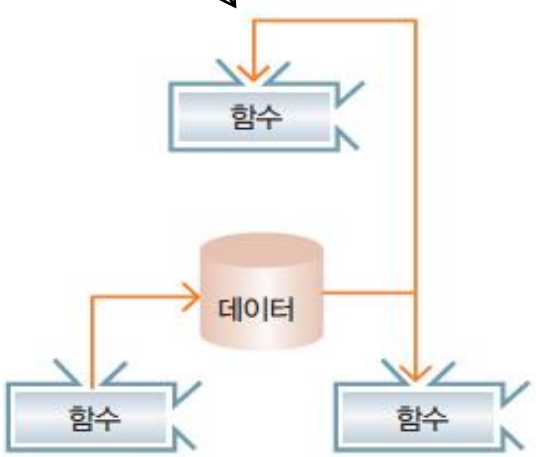
객체 지향 개발 방법론

하나 이상의 데이터와 기능을 묶어 객체라 하고, 데이터의 변경에 해당 객체만 영향을 받도록 분리하는 방법론이 바로 객체 지향 개발 방법론이다. 실행 흐름은 데이터와 기능을 함께 묶은 객체 간에 상호 작용하는 방식이다.

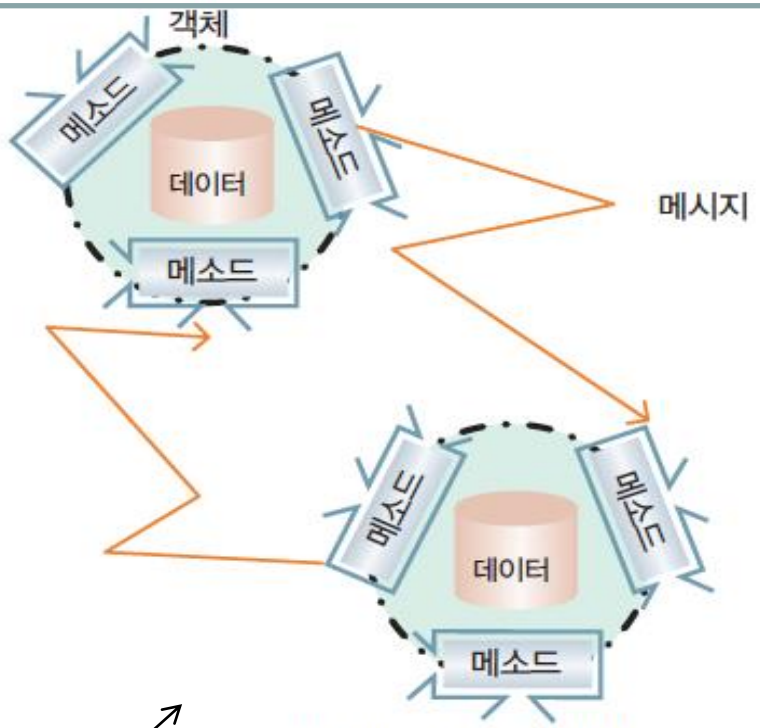
객체는 내부적으로 데이터를 가지며 다른 객체에서 데이터를 노출하지 않기 때문에 특정 객체가 가지고 있는 데이터에 변경이 있더라도 해당 객체로만 변경이 제한된다. 다른 객체에까지 데이터 변경이 전파되지 않아서 코드의 재사용에 있어서는 절차적 개발 방법론에 비해 효과적이다.

절차 지향과 객체 지향

기본적인 구조(순차, 반복, 선택)만을 이용하여 순차적으로 프로그램하는 방식
프로그램을 독립적인 모듈(module)이라는 단위로 나누어서 작성하는 기법
데이터보다는 주로 문제를 해결하는 알고리즘(절차)에 집중하는 방법



절차 지향 프로그래밍에서는
데이터와 알고리즘이
묶여있지 않다.



객체 지향 프로그래밍에서는
데이터와 알고리즘이
묶여있다.

데이터와 알고리즘을 하나의 단위로 묶어서 생각하는 방법

객체 지향 개발 방법론

객체 지향 개발 방법론의 핵심은 객체이다. 모든 관점을 객체라는 단위로 본다. 객체는 하나의 역할(기능)과 그 역할을 수행할 때 필요한 데이터들로 구성된다. 하나의 기능을 수행하도록 요청을 하게 되는데 예를 들어 영희(객체)의 기능인 달려라()를 요청하는 방식을 나타내면 다음과 같다.

구조적 개발 방법론	달려라(영희)
객체지향 개발 방법론	영희.달려라();

객체 간의 상호 작용은 메시지를 통해 다른 객체의 함수를 호출함으로서 이루어진다. 그리고 객체지향 개발 방법론에서는 **객체 간에는 연관성이 낮아야 하고, 객체 내 요소(데이터와 기능) 간에는 연관성이 높아야 한다.** 다시 말해 객체 응집도는 높아야 하고 객체 간의 결합도는 낮추어야 한다.

결합도

응집도

객체 지향 프로그래밍

❖ 객체 지향 프로그래밍

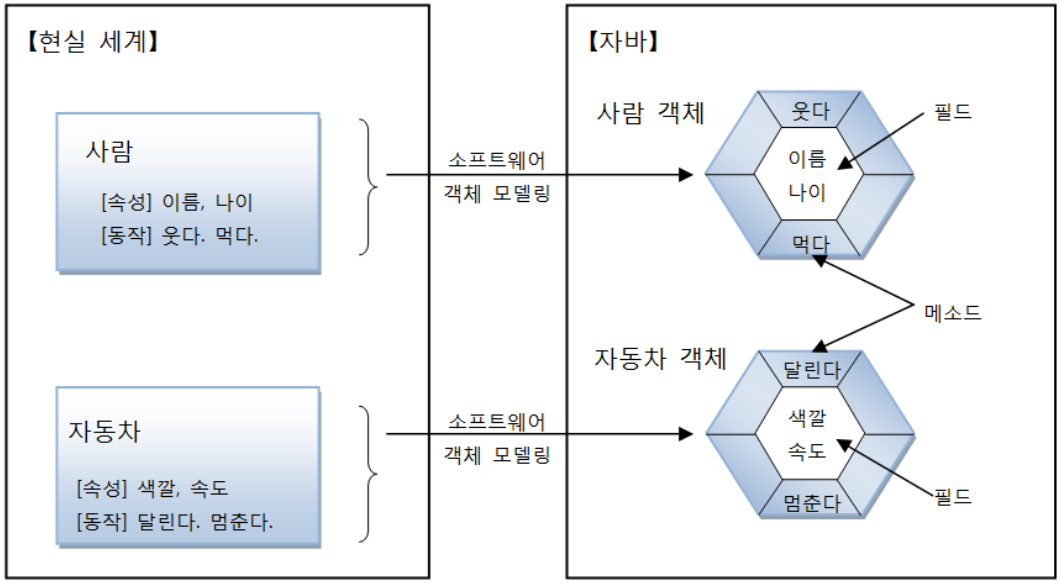
- OOP: Object Oriented Programming
- 현실 세계에서 어떤 제품을 만들 때 부품을 먼저 개발하고 이 부품들을 하나씩 조립해서 완성된 제품을 만들 듯이, 소프트웨어를 개발할 때에도 부품에 해당하는 객체들을 먼저 만들고 이것들을 하나씩 조립해서 완성된 프로그램을 만드는 기법 (모든 관점을 객체라는 단위로 바라본다)

- 자바는 완벽한 객체지향 언어이다. 객체는 Object로 우리 주변 존재하는 물건(연필, 공책, 지각, 돈 등)이나 대상(철수, 영희, 친구 등) 전부를 의미한다.
- 객체지향프로그래밍은 현실에 존재하는 사물과 대상 그리고 그에 따른 행동을 있는 그대로 실체화시키는 프로그래밍이다.

객체 지향 프로그래밍

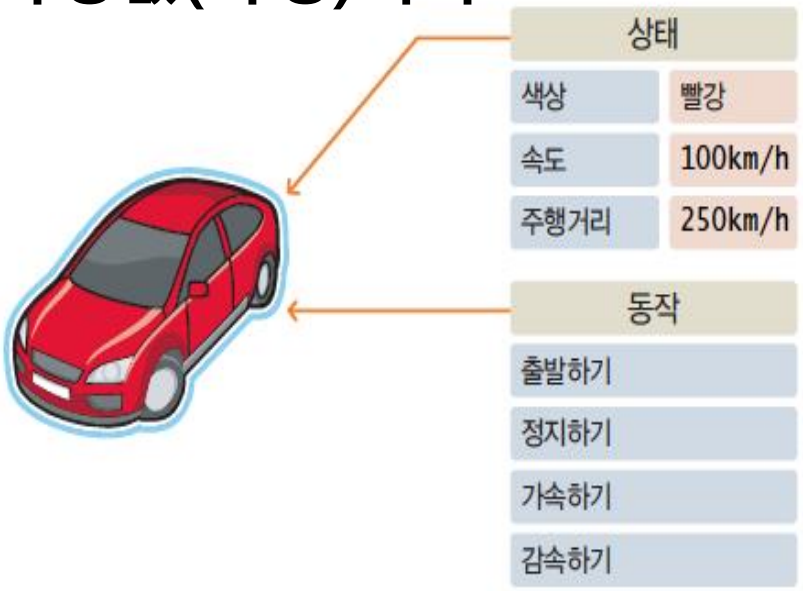
객체(Object)란?

- 물리적으로 존재하는 것 (자동차, 책, 사람)
- 추상적인 것(회사, 학과) 중에서 자신의 속성과 동작을 가지는 모든 것
- 사람은 이름, 나이 등의 속성과 웃다, 걷다 등의 동작이 있고, 자동차는 색상, 모델명 등의 속성과 달린다, 멈춘다 등의 동작이 있다.
- 현실 세계의 객체를 소프트웨어 객체로 설계하는 것을 객체 모델링 (Object Modeling)이라고 한다. 현실 세계 객체의 속성과 동작을 추려내어 소프트웨어 객체의 필드와 메서드로 정의하는 과정이라고 볼 수 있다.



객체란?

- 객체는 현실 세계에서 존재하는 객체들을 소프트웨어 상에서 구현한 것이다.
- 객체(Object)는 상태와 동작을 가지고 있다.
- 객체의 상태(state)는 객체의 특징값(속성)이다.
- 객체의 동작(behavior) 또는 행동은 객체가 취할 수 있는 동작을 의미한다.



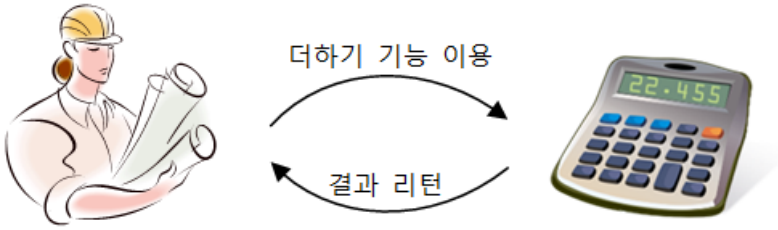
객체 = 자신의 데이터를 사용해 하나의 역할(기능들)을 수행하는 독립된 단위.

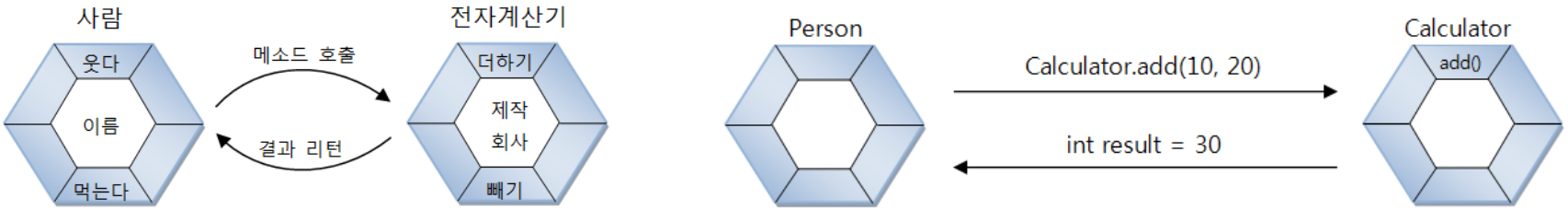
객체 = 데이터들 + 기능(메서드)들
객체 지향 프로그램들 = 객체 + 객체...

객체 지향 프로그래밍

❖객체의 상호 작용

- 객체들은 서로 간에 기능(동작)을 이용하고 데이터를 주고 받음



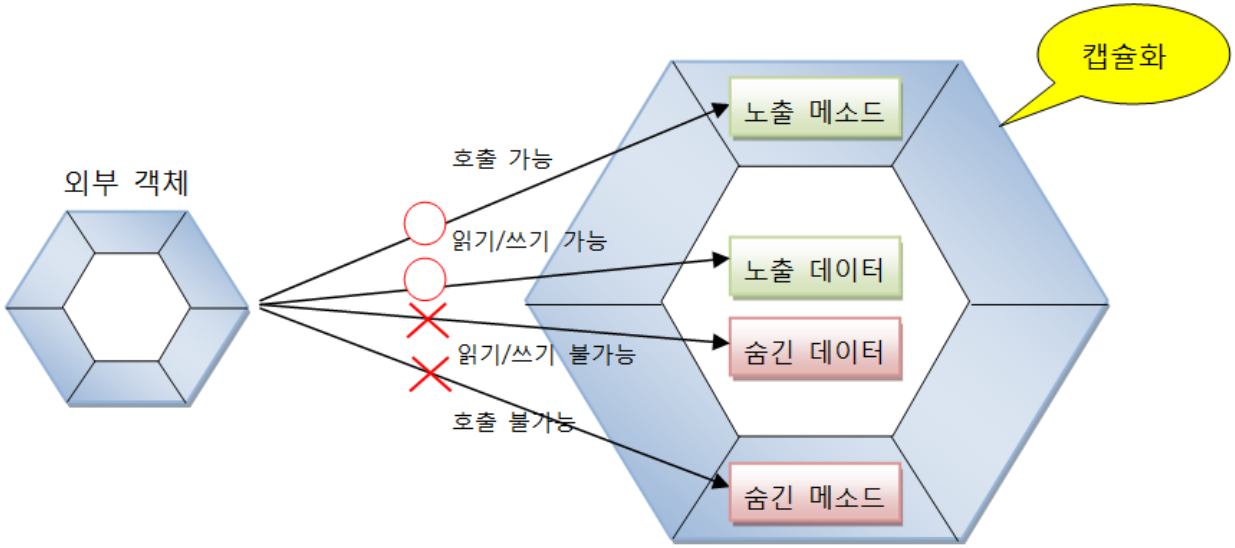


객체 지향 프로그래밍

❖객체 지향 프로그래밍의 특징

▪ 캡슐화

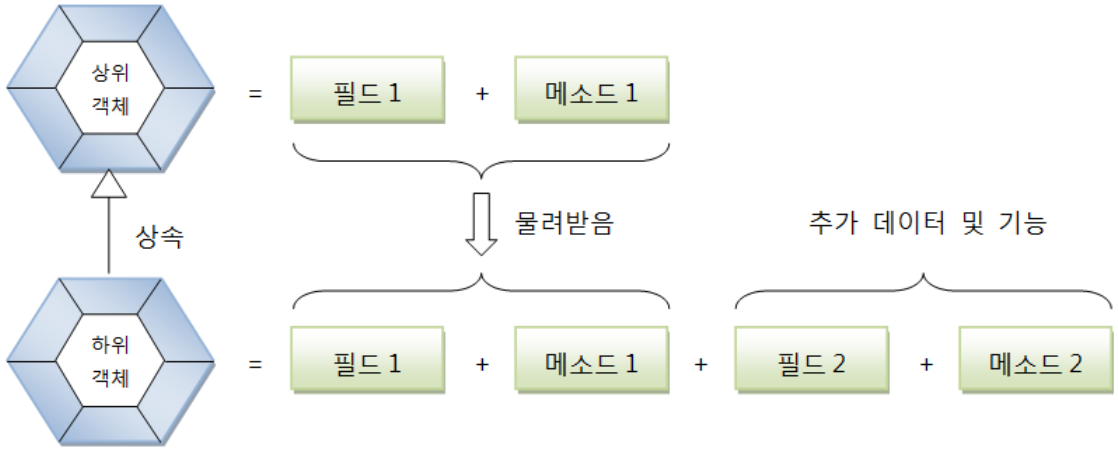
- 객체의 필드, 메소드를 하나로 묶고, 실제 구현 내용을 감추는 것
- 외부 객체는 객체 내부 구조를 알지 못하며 객체가 노출해 제공하는 필드와 메소드만 이용 가능
- 필드와 메소드를 캡슐화하여 보호하는 이유는 외부의 잘못된 사용으로 인해 객체가 손상되지 않도록 하는 것이다.
- 자바 언어는 캡슐화된 멤버를 노출시킬 것인지 숨길 것인지 결정하기 위해 접근 제한자(Access Modifier) 사용



객체 지향 프로그래밍

❖객체 지향 프로그래밍의 특징

- 상속
  - 상위(부모) 객체의 필드와 메소드를 하위(자식) 객체에게 물려주는 행위
  - 하위 객체는 상위 객체를 확장해서 추가적인 필드와 메소드를 가질 수 있음
  - 상속 대상: 필드와 메소드
  - 상속의 효과
    - 상위 객체를 재사용해서 하위 객체를 빨리 개발 가능
    - 반복된 코드의 중복을 줄임
    - 유지 보수의 편리성 제공
    - 객체의 다형성 구현



객체 지향 프로그래밍

❖객체 지향 프로그래밍의 특징

- 다형성 (Polymorphism)
  - 하나의 타입에 여러 가지 객체 대입해 다양한 실행 결과를 얻는 것
    - 부모 타입에는 모든 자식 객체가 대입
    - 인터페이스 타입에는 모든 구현 객체가 대입
  - 효과
    - 객체를 부품화시키는 것 가능
    - 유지보수 용이

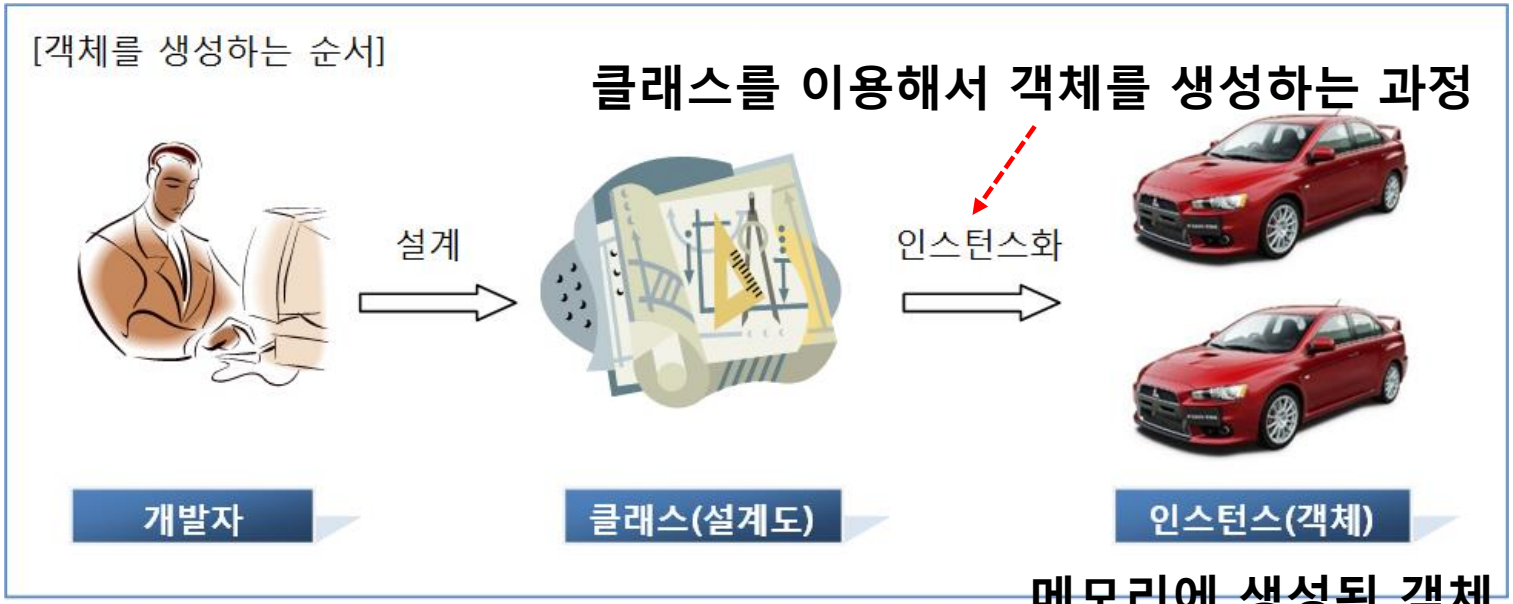
객체와 클래스

❖ 객체(Object)와 클래스(Class)

- 현실세계: 설계도 → 객체
- 자바: 클래스 → 객체

프로그램에서는 반드시 클래스를 사용해서 객체를 메모리에 생성해야만 실행할 수 있다.

- 클래스에는 객체를 생성하기 위한 필드와 메소드가 정의
- 클래스로부터 만들어진 객체를 해당 클래스의 인스턴스(instance)
- 하나의 클래스로부터 여러 개의 인스턴스를 만들 수 있음



클래스

- 클래스(class): 모든 객체에게 공통인 변수와 메소드를 정의하는 틀(template)를 의미.
- 클래스로부터 만들어지는 각각의 객체를 특별히 그 클래스의 인스턴스(instance)라고도 한다.
- 클래스의 이름(자바 식별자 작성 규칙에 맞게 지정)

번호	작성 규칙	예
1	하나 이상의 문자로 이루어져야 한다.	Car, SportsCar
2	첫 번째 글자는 숫자가 올 수 없다.	Car, 3Car(x)
3	'$', '_' 외의 특수 문자는 사용할 수 없다.	$Car, _Car, @Car(x), #Car(x)
4	자바 키워드는 사용할 수 없다.	int(x), for(x)

- 한글 이름도 가능하나, 영어 이름으로 작성
- 알파벳 대소문자는 서로 다른 문자로 인식
- 첫 글자와 연결된 다른 단어의 첫 글자는 대문자로 작성하는 것이 관례

클래스 구성

접근 권한 | 클래스 선언 | 클래스 이름

```
public class Circle {
   public int radius; // 원의 반지름 필드
   public String name; // 원의 이름 필드

   public Circle() { // 원의 생성자 메소드
   }
   public double getArea() { // 원의 면적 계산 메소드
      return 3.14*radius*radius;
   }
}
```

필드(변수)

메소드

클래스 구성 설명

클래스 선언, class Circle
- class 키워드로 선언
- 클래스는 {로 시작하여 }로 닫으며 이곳에 모든 필드와 메소드 구현
- class Circle은 Circle 이름의 클래스 선언
- 클래스 접근 권한, public
  - 다른 클래스들에서 Circle 클래스를 사용하거나 접근할 수 있음을 선언

필드와 메소드
- 필드 (field) : 객체 내에 값을 저장하는 멤버 변수
- 메소드 (method) : 함수이며 객체의 행동(행위)를 구현

필드의 접근 지정자, public
- 필드나 메소드 앞에 붙어 다른 클래스의 접근 허용을 표시
- public 접근 지정자 : 다른 모든 클래스의 접근 허용

생성자
- 클래스의 이름과 동일한 특별한 메소드
- 객체가 생성될 때 자동으로 한 번 호출되는 메소드
- 개발자는 객체를 초기화하는데 필요한 코드 작성

클래스

- 클래스 선언과 컴파일
  - 소스 파일 생성: 클래스이름.java (대소문자 주의)
  - 소스 작성

```
public class 클래스명{
   ...
}
```

컴파일 → javac.exe → 클래스이름.class

  - 소스 파일당 하나의 클래스를 선언하는 것이 관례
    - 두 개 이상의 클래스도 선언 가능
    - 소스 파일 이름과 동일한 클래스만 public으로 선언 가능
    - 선언한 개수(클래스수)만큼 바이트 코드 파일이 생성

Car.java

```
public class Car{
}

class Tire {
}
```

컴파일 → javac.exe → Car.class, Tire.class

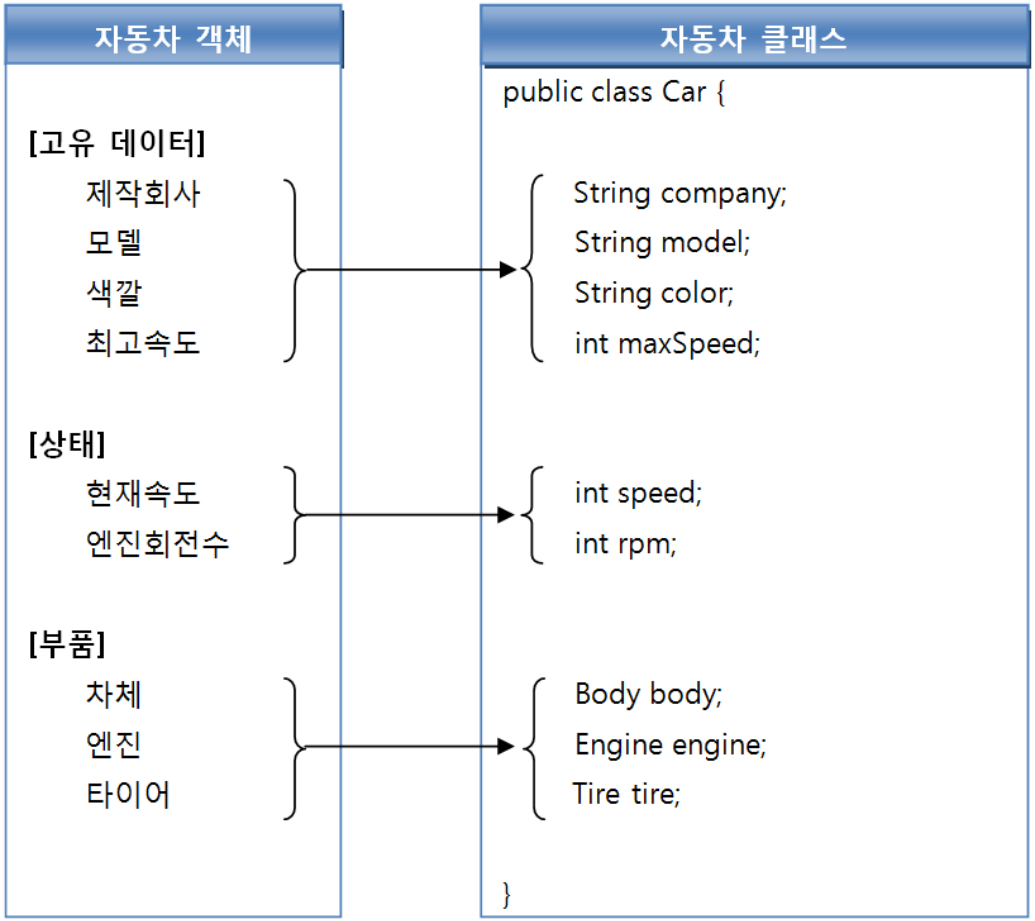
클래스

- 클래스의 용도
  - 라이브러리(API: Application Program Interface) 용
    - 자체적으로 실행되지 않음
    - 다른 클래스에서 이용할 목적으로 만든 클래스
  - 실행용
    - main() 메소드를 가지고 있는 클래스로 실행할 목적으로 만든 클래스

1개의 애플리케이션 = (1개의 실행클래스) + (n개의 라이브러리 클래스)

클래스

- 필드의 내용
  - 객체의 고유 데이터
  - 객체가 가져야 할 부품 객체
  - 객체의 현재 상태 데이터



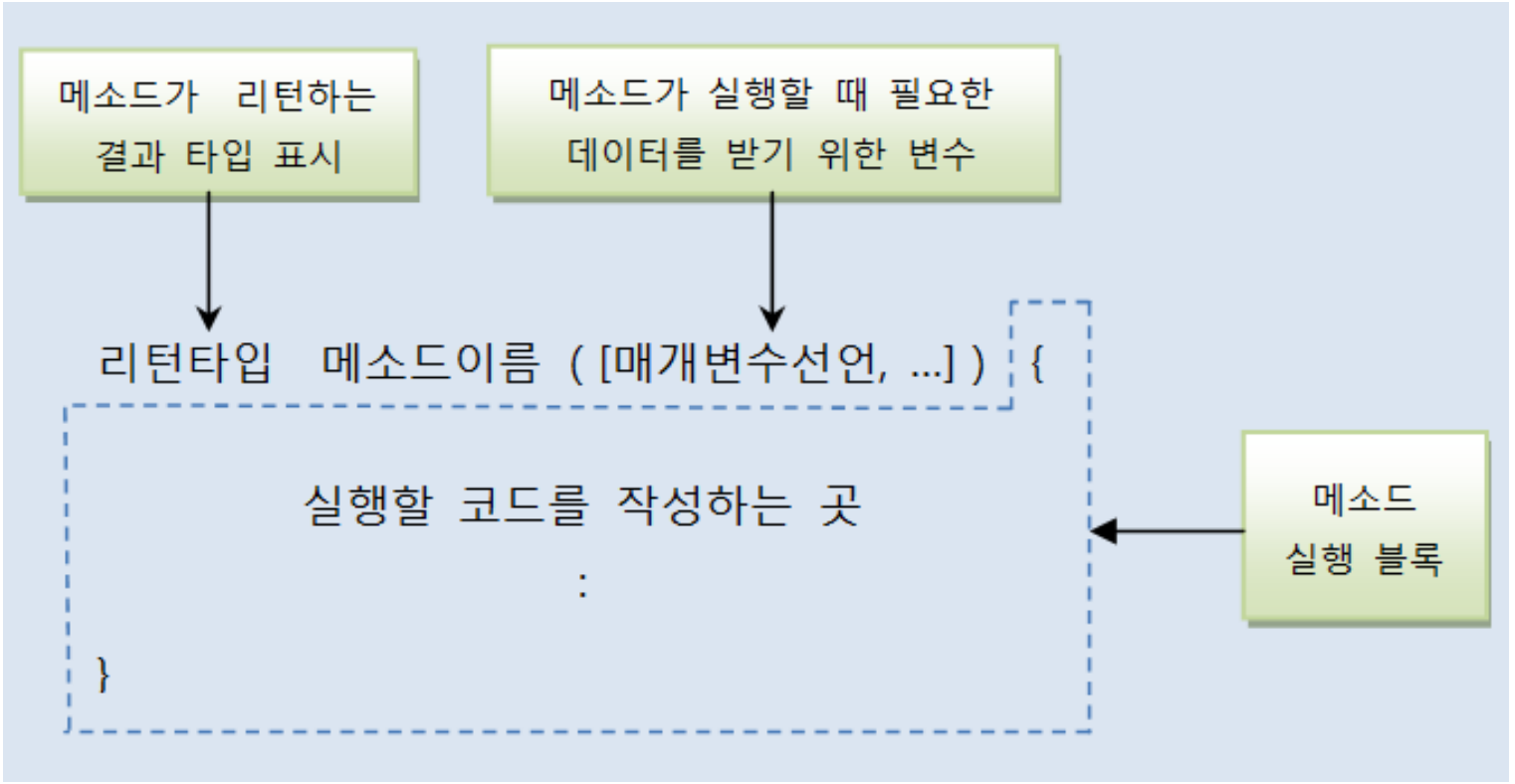
- 필드 선언

```
타입 필드 [ = 초기값] ;
```

```
String company = "현대자동차";
String model = "그랜저";
int maxSpeed = 300;
int productionYear;
int currentSpeed;
boolean engineStart;
```

클래스

- **메소드란?**
  - 객체의 동작(기능)
  - 호출해서 실행할 수 있는 중괄호 { } 블록
  - 메소드 호출하면 중괄호 { } 블록에 있는 모든 코드들이 일괄 실행



클래스

- 메소드 리턴 타입
  - 메소드 실행된 후 리턴하는 값의 타입
  - 메소드는 리턴값이 있을 수도 있고 없을 수도 있음

[메소드 선언]

```
void powerOn(){ }
double divide(int x, int y){ }
```

[메소드 호출]

```
powerOn();
double result = divide(10, 20);
```

- 메소드 매개변수 선언
  - 매개변수는 메소드를 실행할 때 필요한 데이터를 외부에서 받기 위해 사용
  - 매개변수도 필요 없을 수 있음

[메소드 선언]

```
void powerOn(){…}
double divide(int x, int y) {…}
```

[메소드 호출]

```
powerOn();
double result = divide(10, 20);
```

클래스

1) 클래스 선언

클래스 이름

클래스는 다음과 같은 구조로 선언한다.

```
class 클래스명{
        // 필드정의
        int field1;
        int field2;

        // 메소드 정의
        void method(parameter) {
                ....
        }
}
```

필드정의: 객체의 속성을 나타낸다

메소드 정의: 객체의 동작을 나타낸다

클래스

2) 자동차 클래스 예제

Car 클래스는 자동차에 관련된 속성과 동작을 묶은 것이다.
자동차의 속성과 동작 중에서 색상, 현재 속도, 현재 기어 단수 만을 기술한다.
자동차는 수동 기어를 장착했다고 가정한다. 불필요한 속성을 제거하는 과정을 추상화(abstraction)라고 한다.

```
class Car {
        public String color; // 모델색상
        public int speed;    // 현재 속도
        public int gear;     // 현재 기어 단수

        public void printData() {
                System.out.println("(" + color + ", " + speed
                                + ", " + gear + ")");
        }
}
```

클래스

2-1) 이름과 세 과목의 점수를 이용해서 클래스를 작성해 보자.

이름	국어	영어	수학
홍길동	90	100	95
김철수	90	80	95

```
class Score {
        public String name;
        public int kor;
        public int eng;
        public int mat;
        public void printData(){
                System.out.println(name +" " + kor + " "
                    + eng + " " + mat);
        }
}
```

클래스

3) 기초 변수와 참조 변수

자바에서 변수를 크게 분류하면 기초 변수(primitive variable)와 참조 변수(reference variable)로 나눌 수 있다.

- 기초 변수는 int, float, char 등의 기초 자료형으로 선언된 변수이다. 기초 변수는 실제 데이터값이 저장된다.

- 참조 변수는 객체를 참조할 때 사용되는 변수이다. 참조 변수에는 객체의 주소가 들어 있어서 객체가 있는 곳을 가리킨다.

클래스

4) 자동차 클래스 예제

```
public class Car {
        // 필드 정의
        public int speed;       // 속도
        public int gear;        // 기어 단수
        public String color;    // 색상
        // 메소드 정의
        public void speedUp() {  // 속도 증가 메소드
                speed += 10;   }
        public void speedDown() {  // 속도 감소 메소드
                speed -= 10;    }
        public void printData() {
            System.out.println("속도: "+speed+" 기어: "+gear+
                                " 색상: " + color); }
        public String toString() {
            return "속도: "+speed+" 기어: "+gear+" 색상: "+color; }
}
```

인스턴스화

클래스를 이용해서 실제인 객체를 생성하는 과정을 인스턴스화(instantiate)라고 한다. 그리고 메모리에 생성된 객체, 즉 실체를 인스턴스(instance)라고 한다.

객체의 생성 객체의 사용 객체의 소멸

인스턴스는 클래스라는 틀을 기반으로 실체화 된 대상이라는 뜻을 담고 있다. 이처럼 클래스라는 틀을 기반으로 실체화되었음을 강조할 때에는 객체를 대신해서 인스턴스라는 단어를 사용한다.

인스턴스화

1) 객체의 생성

```
Car myCar;
myCar = new Car();
```

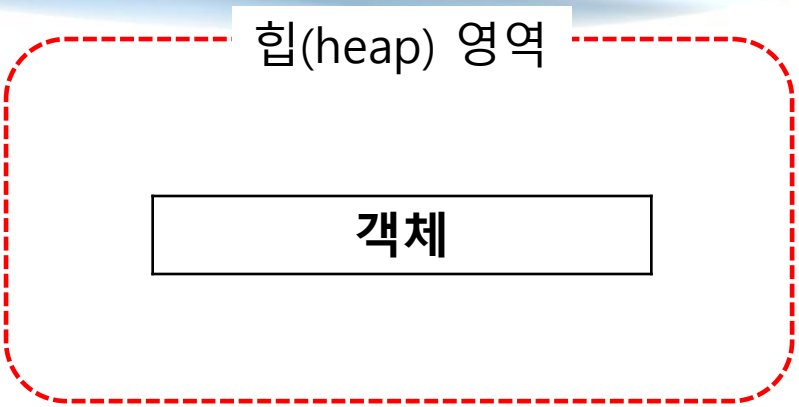
- 참조 변수 선언 - Car타입의 객체를 참조할 수 있는 변수 myCar를 선언한다.
- 객체 생성 – new 연산자를 이용하여 객체를 생성하고 객체 참조값을 반환한다. (클래스의 구성을 참조해 속성과 메서드에 대한 정보를 메모리에 할당하는 작업)
- 참조 변수와 객체의 연결 – 생성된 새로운 객체의 참조값을 myCar라는 참조 변수에 대입한다.

객체 생성과 클래스 변수

❖new 연산자

- **객체 생성 역할**

```
new 클래스명();
```



  - **클래스()는 생성자를 호출하는 코드**
  - **생성된 객체는 힙 메모리 영역에 생성**

- **new 연산자는 객체를 생성 후, 객체 생성 번지 리턴**

※ new 연산자는 다음과 같은 작업을 수행한다.

① 객체를 저장할 힙 메모리의 할당
② 생성자 호출
③ 할당하고자 하는 객체의 힙 주솟값을 참조값으로 변환 후 반환.

인스턴스(instance): 클래스를 new 명령문으로 메모리에 생성된 객체.

객체 생성

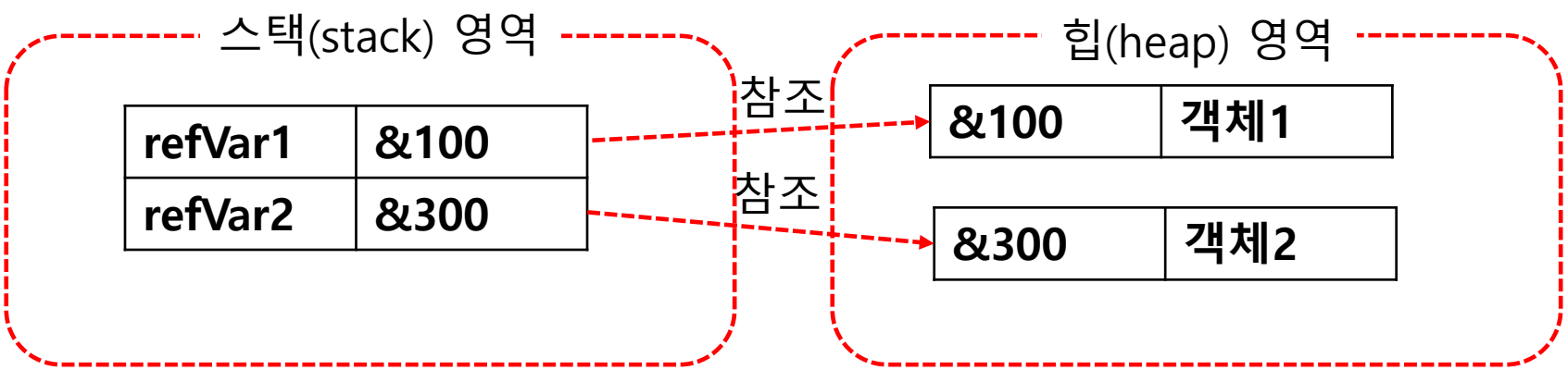
❖참조 변수

- new 연산자에 의해 리턴 된 객체의 번지 저장 (참조 타입 변수)

- 힙 영역의 객체를 사용하기 위해 사용

클래스 변수; 변수 = new 생성자();	클래스 변수 = new 생성자();

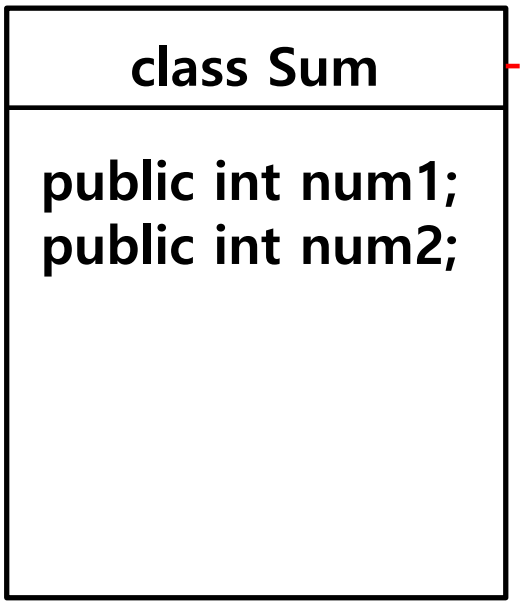
ObjectClass refVar1; refVar1 = new ObjectClass();	ObjectClass refVar2 = new ObjectClass();



인스턴스화

1) 객체의 생성

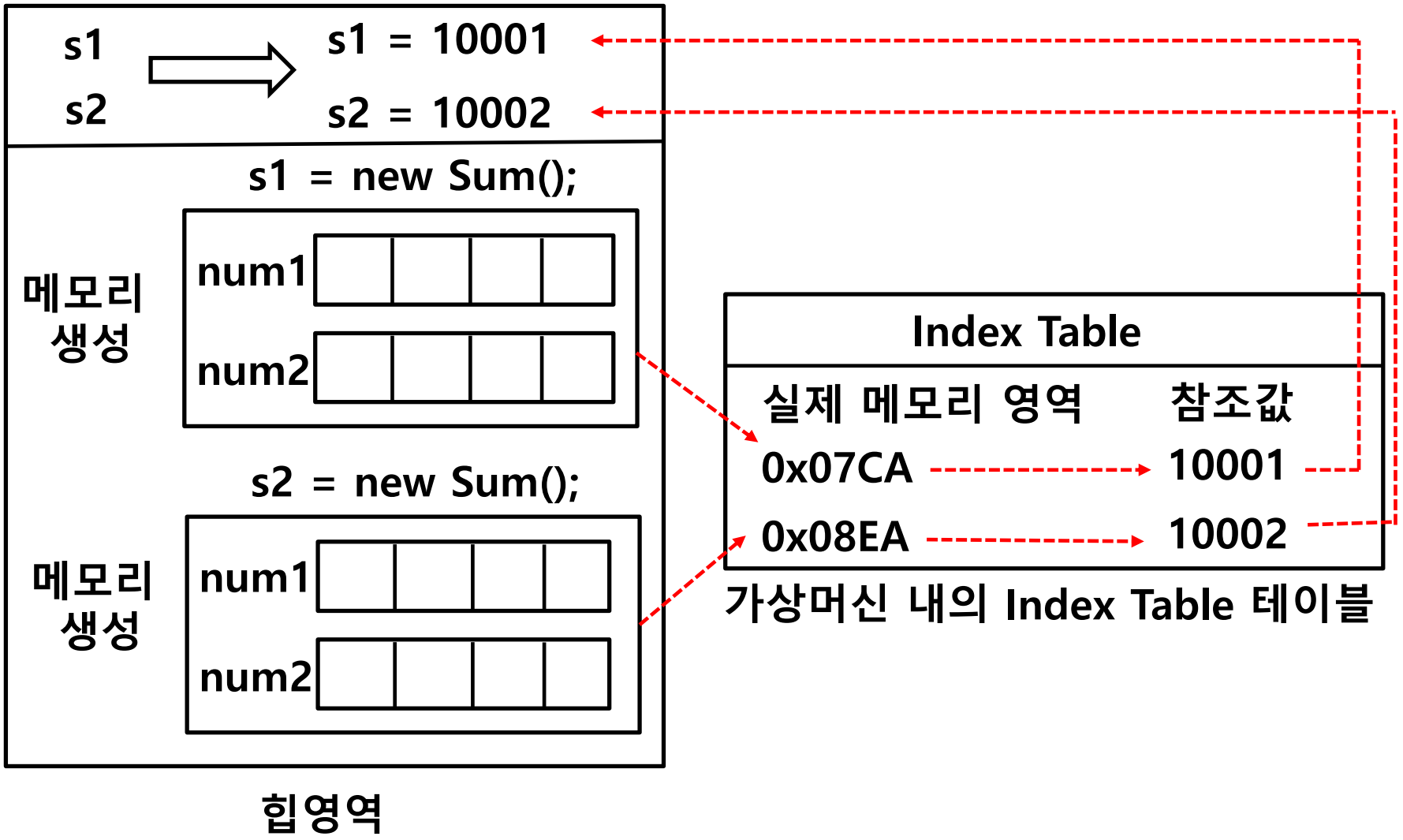
클래스 설계



객체 변수 선언

Sum s1;
Sum s2;
처음 선언 될 때는
s1 = null
s2 = null

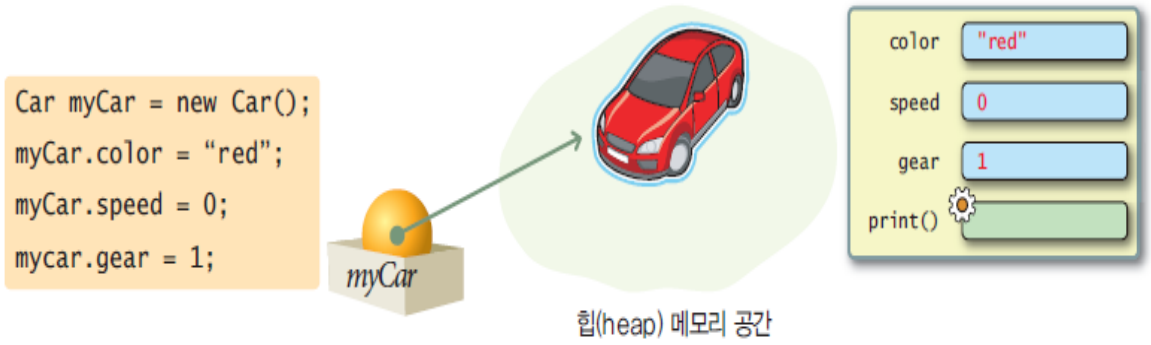
인스턴스화


s1
s2
s1 = 10001
s2 = 10002
s1 = new Sum();
메모리 생성
num1
num2
s2 = new Sum();
메모리 생성
num1
num2
힙영역
Index Table
실제 메모리 영역
참조값
0x07CA
10001
0x08EA
10002
가상머신 내의 Index Table 테이블

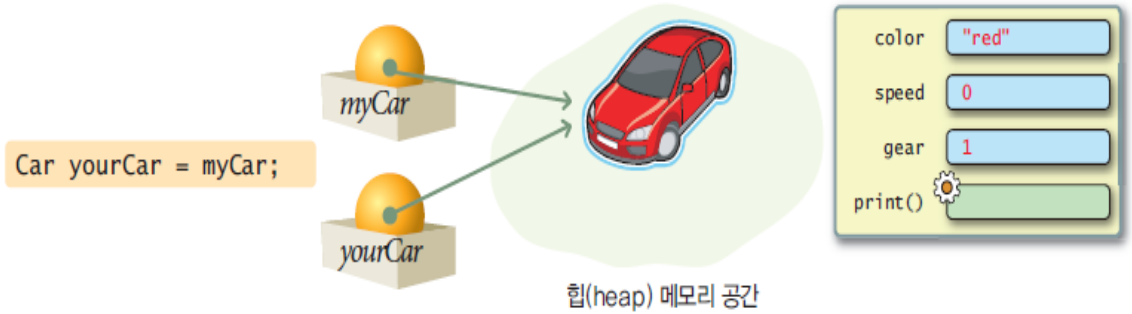
인스턴스화

2) 참조 변수와 대입 연산

```
Car myCar = new Car() ;
```




```
Car yourCar = new Car() ;
yourCar = myCar;
```



인스턴스화

따라서 다음의 문장은 yourCar가 가리키는 객체의 speed 값을 120으로 설정한다.

```
yourCar.speed = 120 ;
```

실제로 위의 문장을 실행한 후에 myCar과 yourCar가 가리키는 객체의 speed값을 출력하여 보면 모두 120임을 알 수 있다.

```
System.out.println(myCar.speed);
System.out.println(yourCar.speed);
```

인스턴스화

```
Car myCar = new Car();
```

- myCar는 객체 참조 변수이다.
- 실제 객체의 생성은 new 연산자가 담당한다.
- new 다음에는 클래스이름과 동일한 생성자를 써준다.
- 이 문장이 실행되면 myCar라는 객체가 실제로 생성된다.
- 즉 물리적인 실체를 가지게 되는 것이다. 이처럼 실제로 생성된 객체를 클래스의 인스턴스(instance)라고도 부른다. 이는 객체(object)라는 용어가 많은 의미를 가지고 있으므로 그 의미를 더 확실히 하기 위해서이다.

인스턴스화

3) 객체의 필드와 메소드 접근

객체의 외부에서 객체에 포함된 필드를 참조하려면 참조 변수에 도트 연산자(.)을 사용하면 된다.

color라는 필드에 접근

```
myCar.color= "red";
```

myCar가 참조하는 객체로부터

객체 지향의 개념에서 보면 필드를 외부에서 직접 접근하는 것은 바람직하지 않다. 객체 내의 필드는 메소드를 통해서 간접적으로 접근하는 것이 바람직하다.

```
myCar.printData();
```

필드(field)

❖ 필드 사용

- 필드 값을 읽고, 변경하는 작업을 말한다.
- 필드 사용 위치
  - 클래스 내부: "필드이름" 으로 바로 접근
  - 클래스 외부: "변수.필드이름"으로 접근

Person 클래스

```
void method(){
   // Car 객체 생성
   Car myCar = new Car();
   // 필드 사용
   myCar.speed = 60;
}
```

값변경

Car 클래스

```
// 필드
int speed;

// 생성자
Car(){
   speed = 0;
}

// 메서드
void method(...){
   speed = 10;
}
```

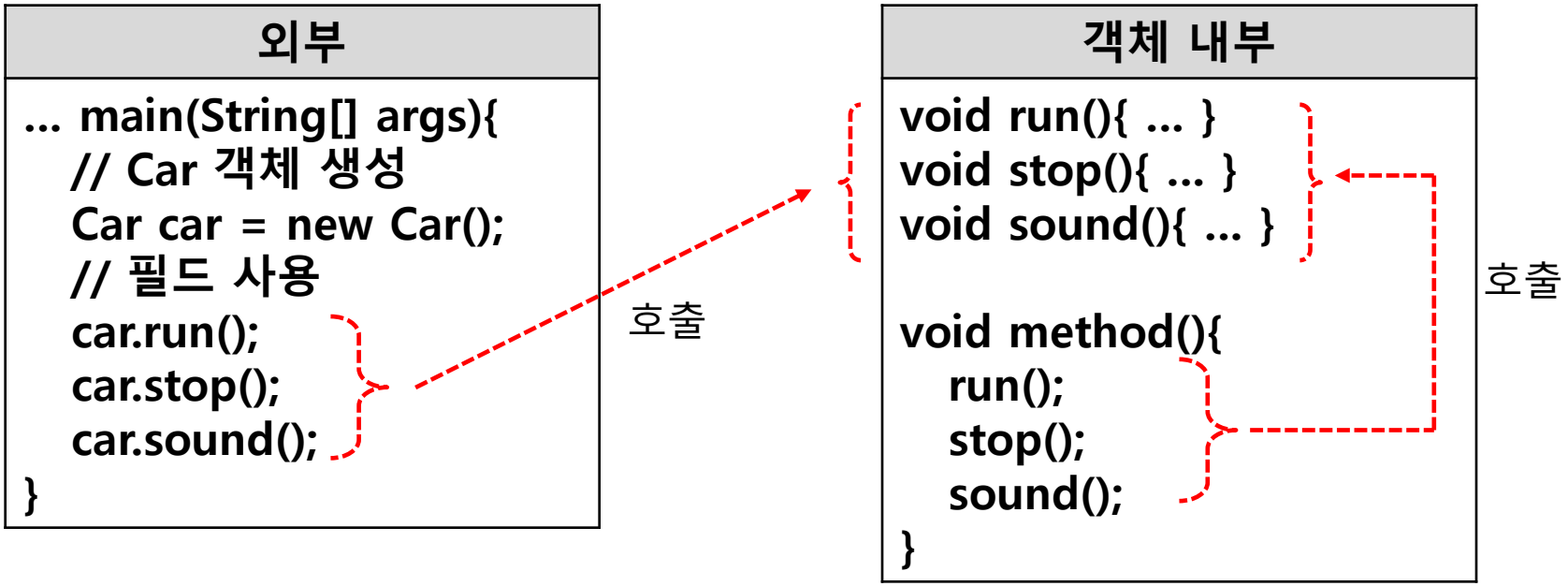
값변경

값변경

메소드(method)

❖메소드 호출

- 메소드는 클래스 내·외부의 호출에 의해 실행
  - 클래스 내부: 메소드 이름으로 호출
  - 클래스 외부: 객체 생성 후, 참조 변수를 이용해 호출



클래스

4) 자동차 객체의 사용

```
public class CarTest {
        public static void main(String[] args) {
                Car myCar = new Car();   // 첫 번째 객체 생성

                myCar.speed = 0;      // 객체의 필드 변경
                myCar.gear = 1;       // 객체의 필드 변경
                myCar.color = "red";  // 객체의 필드 변경

                myCar.speedUp(); // 객체의 메소드 호출
                 myCar.printData();
                System.out.println(myCar.toString());
        }
}
```

객체의 소멸과 가비지 컬렉션

객체 소멸

- new에 의해 할당된 객체 메모리를 자바 가상 기계의 가용 메모리로 되돌려 주는 행위

자바 응용프로그램에서 임의로 객체 소멸할 수 없음

- 객체 소멸은 자바 가상 기계의 고유한 역할
- 자바 개발자에게는 매우 다행스러운 기능
  - C/C++에서는 할당받은 객체를 개발자가 되돌려 주어야 함
    - C/C++ 프로그램 작성을 어렵게 만드는 요인

가비지

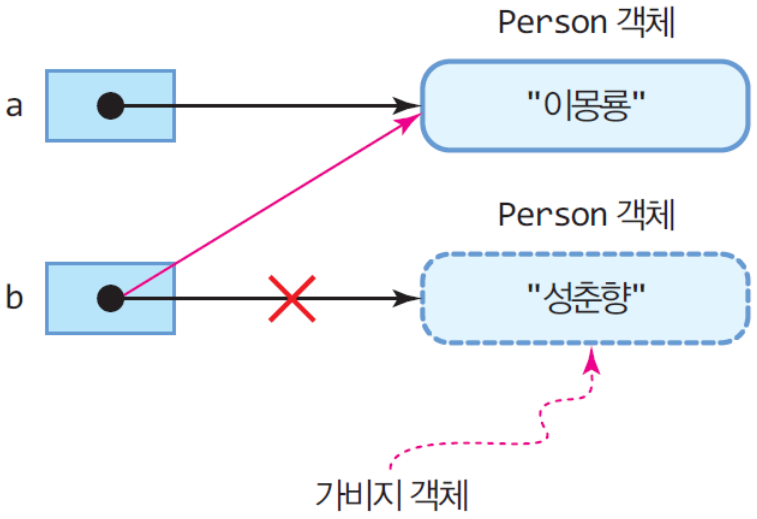
- 가리키는 레퍼런스가 하나도 없는 객체
  - 누구도 사용할 수 없게 된 메모리

가비지 켈렉션

- 자바 가상 기계의 가비지 컬렉터가 자동으로 가비지 수집 반환

가비지 사례

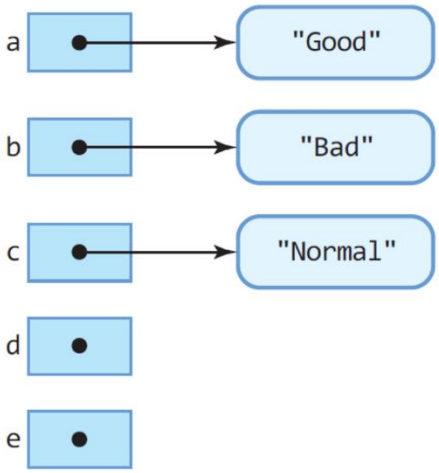
```
Person a, b;
a = new Person("이몽룡");
b = new Person("성춘향");
b = a; // b가 가리키던 객체는 가비지가 됨
```



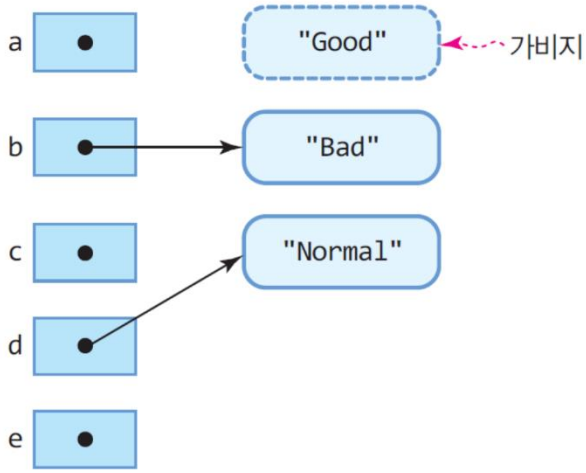
가비지의 발생

다음 코드에서 언제 가비지가 발생하는지 설명하라.

```
public class GarbageEx {
    public static void main(String[] args) {
        String a = new String("Good");
        String b = new String("Bad");
        String c = new String("Normal");
        String d, e;
        a = null;
        d = c;
        c = null;
    }
}
```



(a) 초기 객체 생성 시(라인 6까지)



(b) 코드 전체 실행 후

가비지 컬렉션

가비지 컬렉션

- 자바에서 가비지를 자동 회수하는 과정
  - 가용 메모리로 반환
- 가비지 컬렉션 스레드에 의해 수행

개발자에 의한 강제 가비지 컬렉션

- System 또는 Runtime 객체의 gc() 메소드 호출

```
System.gc(); // 가비지 컬렉션 작동 요청
```

  - 이 코드는 자바 가상 기계에 강력한 가비지 컬렉션 요청
    - 그러나 자바 가상 기계가 가비지 컬렉션 시점을 전적으로 판단

객체의 소멸

참조가 끊긴 인스턴스는 더 이상 사용할 수가 없다. 사용할 수 없는 인스턴스가 메모리에 계속 남아 있으면 메모리
만 낭비할 뿐이다. 그래서 참조가 끊긴 인스턴스들을 '가비지(garbage)', 즉 쓰레기라고 한다.

이처럼 가비지가 된 인스턴스들을 JVM의 '가비지 콜렉션(Garbage Collection)에 의해 메모리에서 삭제된다. 즉 힙에 생성된 필드들은 참조가 끊기면 가비지 콜렉션 작업이 일어날 때 삭제된다.

JVM은 메모리가 부족한 상황이 발생하면 메모리에서 더 이상 사용하지 않는 객체들을 제거하여 사용할 수 있는 메모리를 확보하는데 이러한 작업을 가비지 콜렉션이라고 한다.

객체의 소멸

- 객체는 생성되어서 사용되다가 결국은 소멸 단계를 거쳐 객체가 점유하고 있었던 기억공간을 반환된다. 객체를 소멸한다는 말은, 객체에 할당되어 있던 메모리를 회수한다는 뜻이다.
- 자바에서는 이 문제를 쓰레기 수집(garbage collection)이라는 기능을 추가함으로써 해결하였다. 즉 자바의 실행 환경이 끝난 객체를 스스로 판별하여 소멸시키고 그 객체가 가지고 있던 자원을 시스템에 반납시킨다. 만약 객체를 가리키는 참조 변수가 하나도 남아 있지 않다면 객체의 사용이 끝났다고 판단할 수 있다.

중간 점검

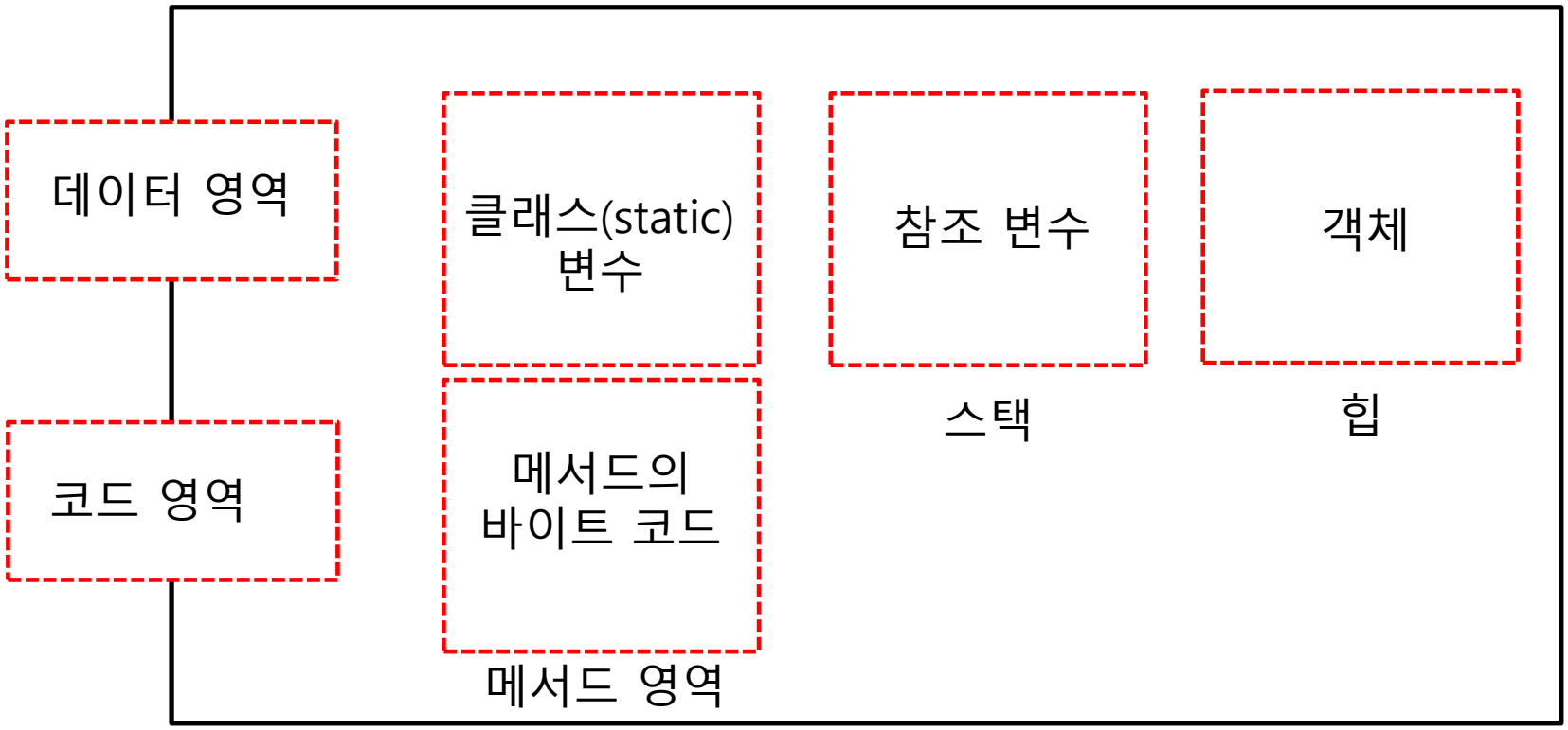
1. 객체를 형성하기 위한 틀로써 _____를 생성한다.
2. 같은 종류의 객체가 여러 개 생성될 때 각 객체의 필드와 메소드는 공유되는가? 아니면 각 객체마다 별도로 만들어지는가?
3. 클래스 선언 시에 클래스 안에 포함되는 것은 _____과 _____이다.

중간 점검

4. 객체의 멤버에 접근하는데 사용되는 연산자는 _____이다.
5. 상품을 나타내는 클래스(Product)를 작성하여 보자. 클래스 안에 상품번호, 상품명, 재고수량이 필드로 저장되고 재고를 증가, 감소하는 메소드를 작성하여 보라.

JVM의 메모리 구조

프로그램은 반드시 주기억장치(메모리)에 적재되어야 실행된다. 프로그램의 코드와 데이터는 역할에 맞게 분산해서 적재된다. JVM에서는 메서드 영역과 힙 영역, 스택 영역으로 나누어 적재된다. 메서드 영역은 데이터와 코드를 모두 저장하는 영역이다.



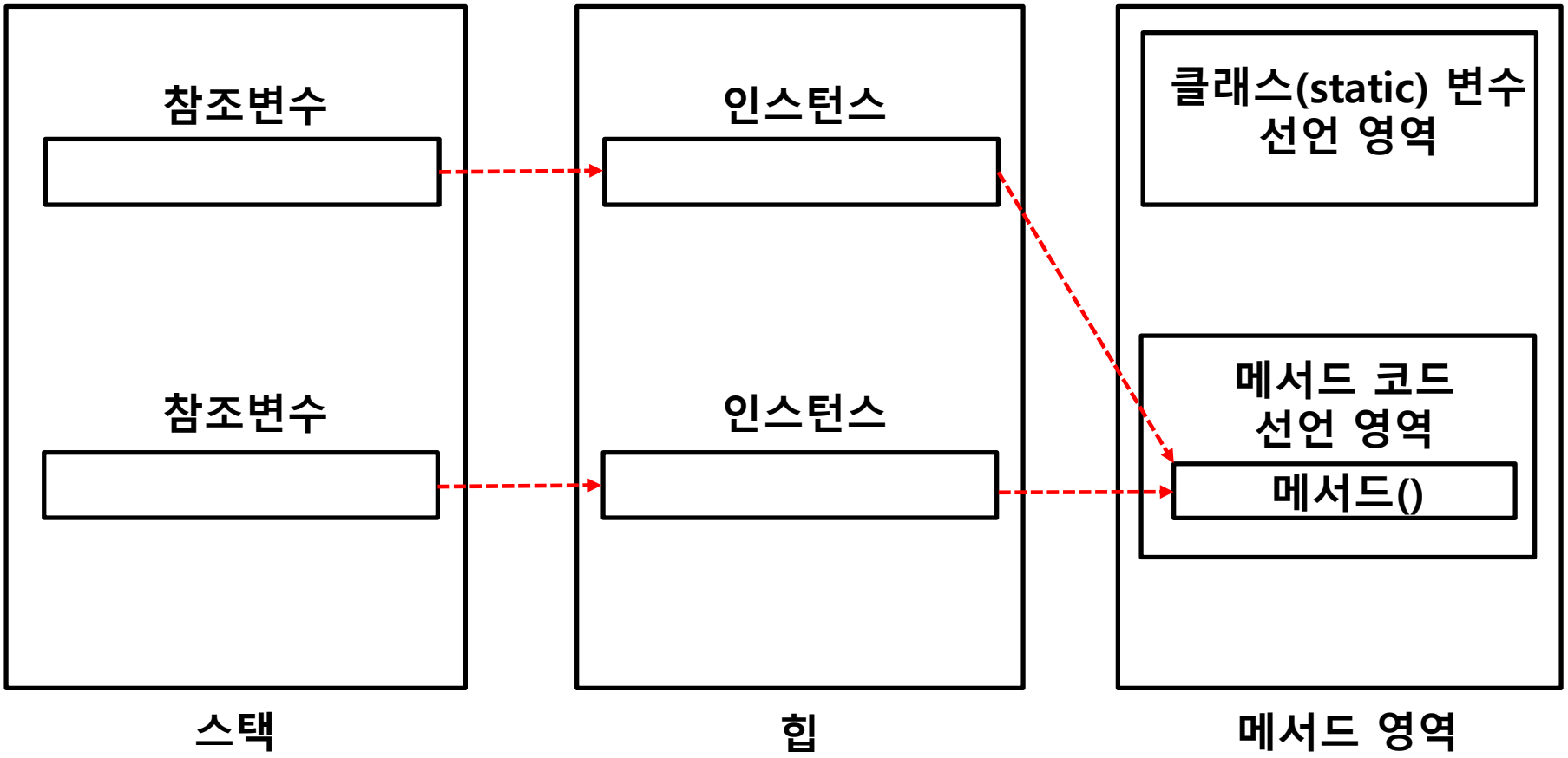
변수의 종류에 따라 메모리 관리

	정적(메서드 영역)	스택	힙
사용용도	static으로 선언. shared(공유)라고도 불린다	선입후출(FILO) 구조 객체 참조 변수 저장 메서드 변수 저장 메서드 매개변수 저장 메서드 내의 { }블록 변수 저장	선입선출(FIFO) 구조 멤버 필드 저장 인스턴스 저장 참조형 저장 [사용법] 참조변수.* 으로 사용
사용기간	객체가 생성 전부터 모든 객체가 없어질 때까지나 프로그램이 종료할 때까지, 명시적으로 null을 선언할 때까지. 가비지 컬렉션의 대상이 된다.	블록 {} 이나 메서드가 끝날 때까지.	객체가 더 이상 사용되지 않거나 명시적으로 null을 선언할 때까지. 가비지 컬렉션의 대상이 된다.
사용범위	같은 형(type)으로 만들어진 곳.	메서드 내에서 {} 블록으로 지정된 곳.	자신 클래스의 모든 곳.

메모리 관리

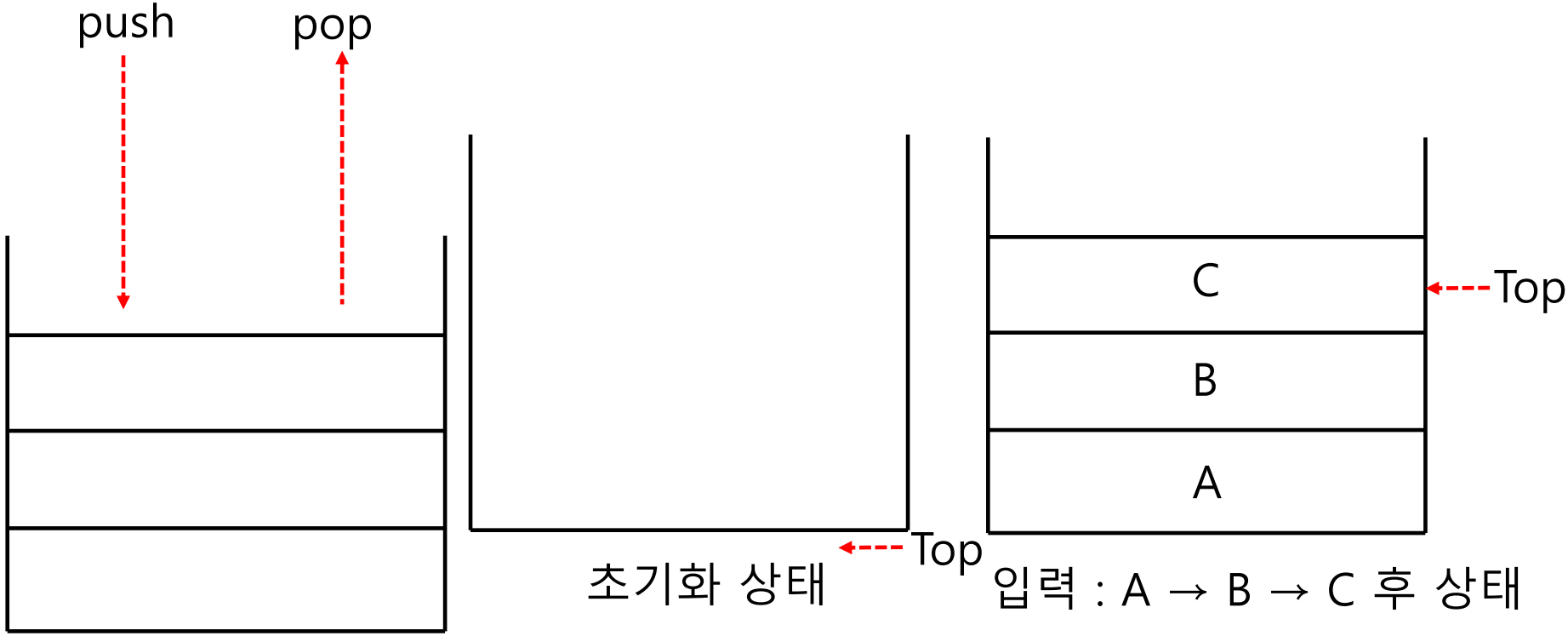
메서드(멤버함수)는 반복적으로 실행하고자 하는 명령어들의 집합이다.

한번 선언으로 여러 번 호출해 사용할 수 있어서 코드의 재사용을 높여준다. 메서드는 메서드 영역이라 불리는 특정 메모리 영역에 선언하고 객체 간에 코드를 공유해 사용한다. 이때 객체에는 메서드의 주소값만 저장한다.



스택

스택은 메모리 측면에서 중앙처리장치와 붙여있는 램(RAM)의 일부분이다. 또한 스택은 자료구조 측면에서 한쪽 끝이 막혀있고 다른 쪽 끝에서만 입력과 출력을 하는 구조이다. LIFO(Last In First Out: 후입선출) 방식으로 데이터의 삽입과 삭제가 이루어진다.

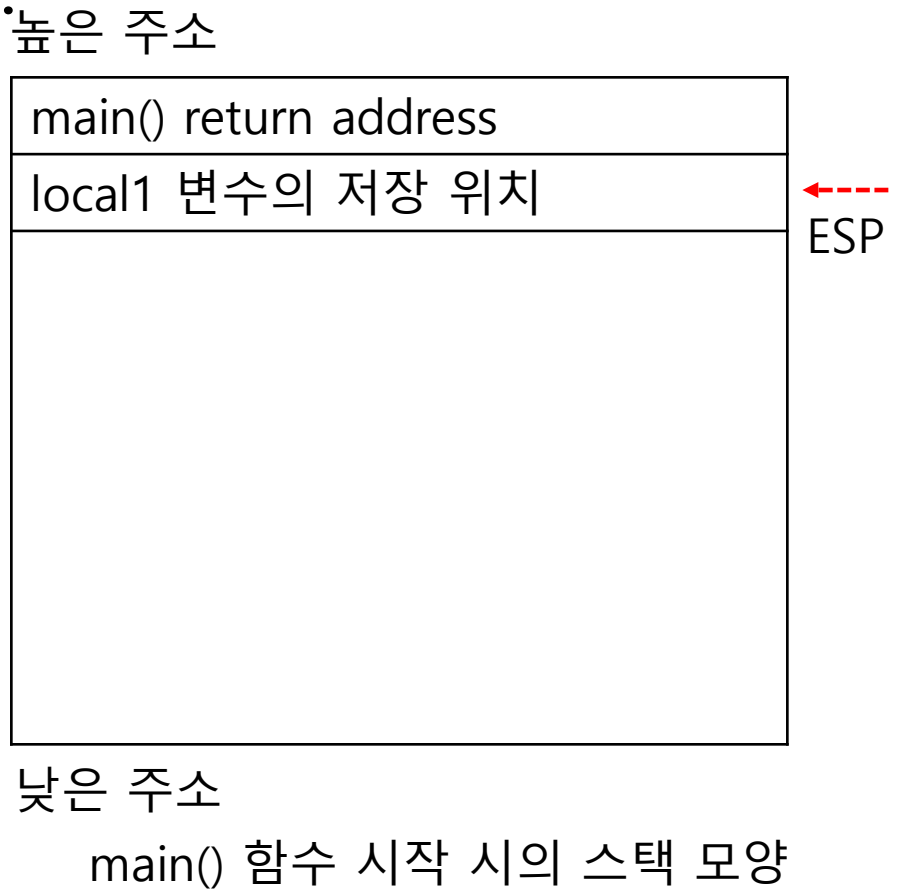


스택

스택은 다음과 같은 상황에서 효율적으로 사용된다.
함수의 지역 변수를 임시로 저장한다.
함수를 호출하면서 매개 변수를 전달한다.
함수 종료 후 복귀한 주소를 저장한다.

```
main(){
    char local1;
    ...
    aMethod();
}
```

※ ESP는 스택의 현재 포인터(주솟값)로, 이 포인터의 위치는 항상 변한다.



main() 함수 시작 시의 스택 모양

스택

```
aMethod(){
    char local2, local3;
    …
    bMethod();
    cMethod();
}
```

높은 주소

main() return address
local1 변수의 저장 위치
aMethod() return address
local2 변수의 저장 위치
local3 변수의 저장 위치 ← ESP

낮은 주소

aMethod 함수 시작 시의 스택 모양


Thank You