클래스와 객체

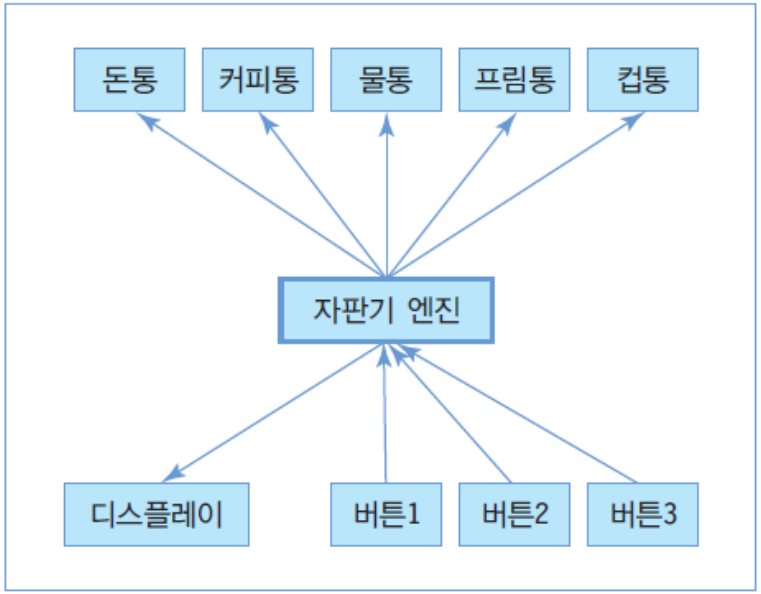
객체지향 언어의 목적

- 소프트웨어의 생산성 향상
  - 컴퓨터 산업 발전에 따라 소프트웨어의 생명 주기(life cycle) 단축
  - 객체 지향 언어는 상속, 다형성, 객체, 캡슐화 등 소프트웨어 재사용을 위한 여러 장치 내장
    - 소프트웨어의 재사용과 부분 수정을 통해 소프트웨어를 다시 만드는 부담을 대폭 줄임으로써 소프트웨어의 생산성이 향상
- 실세계에 대한 쉬운 모델링
  - 과거
    - 수학 계산/통계 처리를 하는 등의 처리 과정, 계산 절차가 중요
  - 현재
    - 컴퓨터가 산업 전반에 활용
    - 실세계에서 발생하는 일을 프로그래밍
      - 실세계에서는 절차나 과정보다 일과 관련된 물체(객체)등의 상호 작용으로 묘사하는 것이 용이
  - 실세계의 일을 보다 쉽게 프로그래밍하기 위한 객체 중심의 객체 지향 언어 탄생

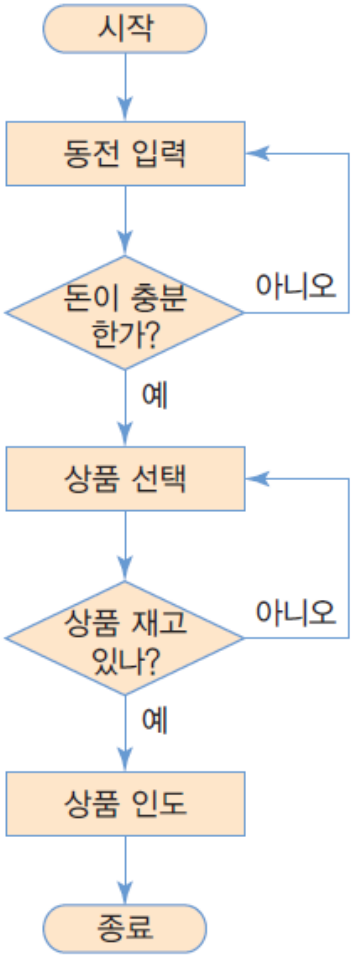
절차 지향 프로그래밍과 객체 지향 프로그래밍

- 절차 지향 프로그래밍
  - 작업 순서를 표현하는 컴퓨터 명령 집합
  - 함수들의 집합으로 프로그램 작성
- 객체 지향 프로그래밍
  - 프로그램을 실제 세상에 가깝게 모델링
  - 컴퓨터가 수행하는 작업을 객체들간의 상호 작용으로 표현
  - 클래스 혹은 객체들의 집합으로 프로그램 작성

커피 자판기



객체지향적 프로그래밍의 객체들의 상호 관련성

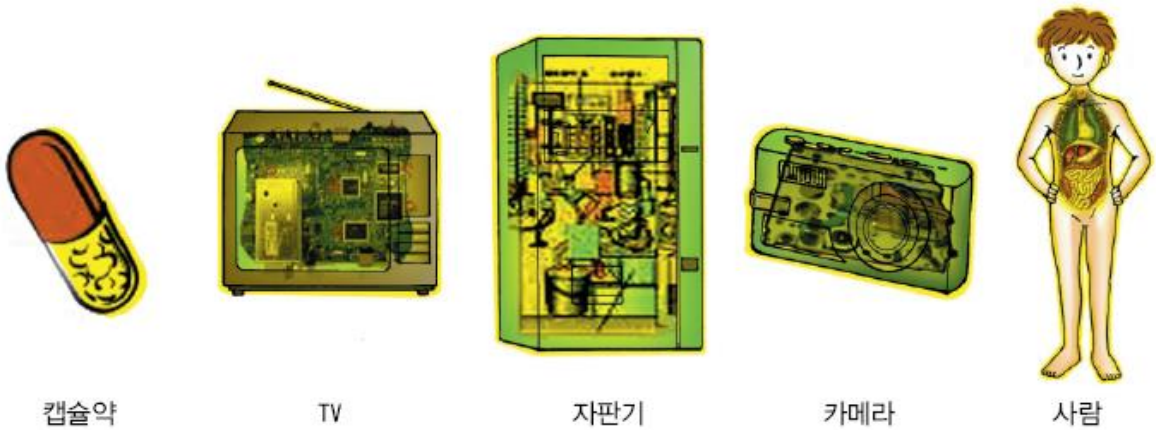


절차지향적 프로그래밍의 실행 절차

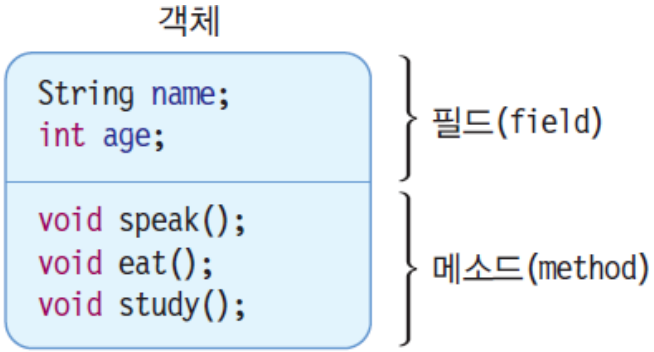
객체 지향 언어의 특성 : 캡슐화

- 캡슐화
  - 메소드(함수)와 데이터를 클래스 내에 선언하고 구현
  - 외부에서는 공개된 메소드의 인터페이스만 접근 가능
    - 외부에서는 비공개 데이터에 직접 접근하거나 메소드의 구현 세부를 알 수 없음
  - 객체 내 데이터에 대한 보안, 보호, 외부 접근 제한

실세계의 캡슐화

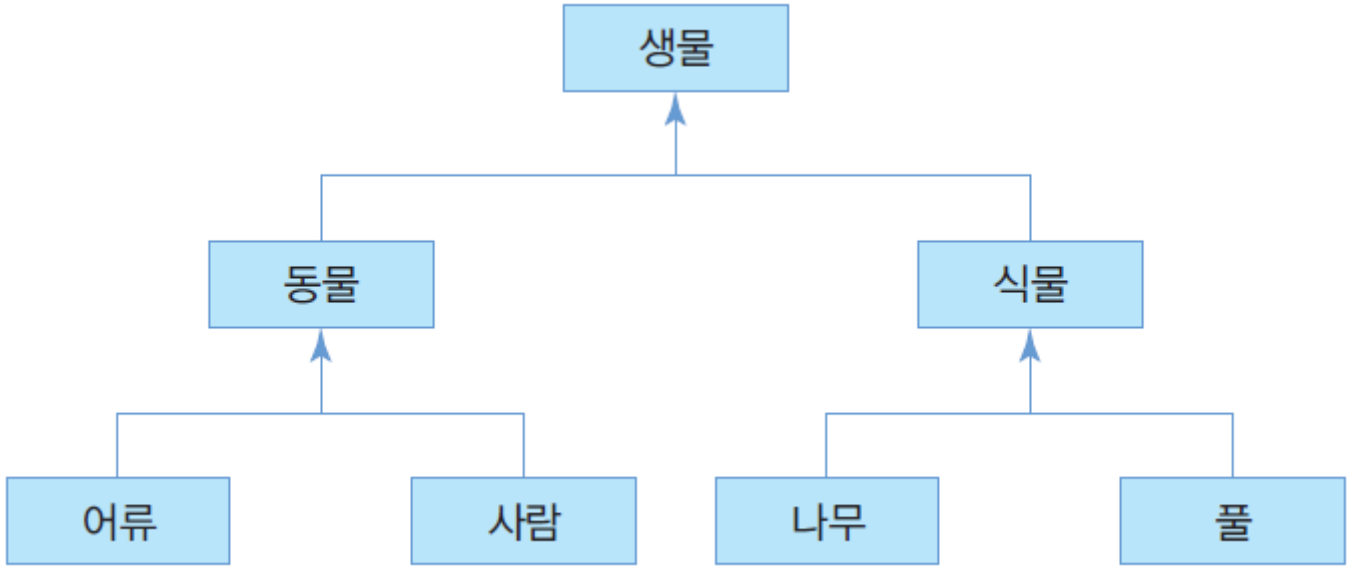


자바 객체의 캡슐화



객체 지향의 특성 : 상속

- 유전적 상속 관계 표현
  - 나무는 식물의 속성과 생물의 속성을 모두 가짐
  - 사람은 생물의 속성은 가지지만 식물의 속성은 가지고 있지 않음



객체 지향 언어에서의 상속


```
class Animal {
    String name;
    int age;
    void eat() {...}
    void sleep() {...}
    void love() {...}
}
```

상속

```
class Human extends Animal {
    String hobby;
    String job;
    void work() {...}
    void cry() {...}
    void laugh() {...}
}
```

Animal의 객체

```
String name;
int age;

void eat();
void sleep();
void love();
```

Human의 객체

```
String name;
int age;

void eat();
void sleep();
void love();

String hobby;
String job;

void work();
void cry();
void laugh();
```

상속

- 상위 클래스의 특성을 하위 클래스가 물려받음
  - 상위 클래스 : 수퍼 클래스, 하위 클래스 : 서브 클래스
- 서브 클래스
  - 수퍼 클래스 코드의 재사용
  - 새로운 특성 추가 가능
- 자바는 클래스 다중 상속 없음
  - 인터페이스를 통해 다중 상속과 같은 효과 얻음

객체 지향의 특성 : 다형성

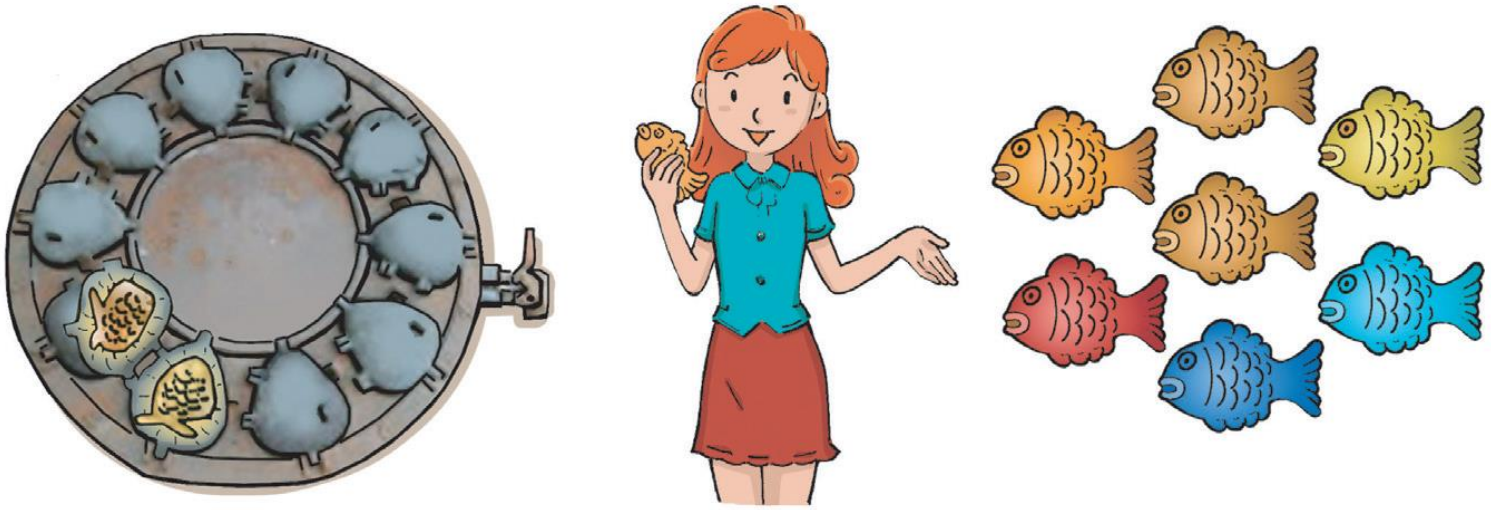
- 다형성
  - 동일한 이름의 기능이 서로 다르게 작동하는 현상
  - 자바의 다형성 사례
    - 슈퍼 클래스의 메소드를 서브 클래스마다 다르게 구현하는 메소드 오버라이딩
    - 한 클래스 내에 구현된 동일한 이름이지만 다르게 작동하는 여러 메소드

클래스와 객체

- 클래스
  - 객체의 속성과 행위 선언
  - 객체의 설계도 혹은 틀
- 객체
  - 클래스의 틀로 찍어낸 실체
    - 메모리 공간을 갖는 구체적인 실체
    - 클래스를 구체화한 객체를 인스턴스(instance)라고 부름
    - 객체와 인스턴스는 같은 뜻으로 사용
- 사례
  - 클래스: 소나타자동차, 객체: 출고된 실제 소나타 100대
  - 클래스: 벽시계, 객체: 우리집 벽에 걸린 벽시계들
  - 클래스: 책상, 객체: 우리가 사용중인 실제 책상들

클래스와 객체와의 관계

붕어빵 틀은 클래스이며, 이 틀의 형태로 구워진 붕어빵은 바로 객체입니다. 붕어빵은 틀의 모양대로 만들어지지만 서로 조금씩 다릅니다.
치즈붕어빵, 크림붕어빵, 앙코붕어빵 등이 있습니다.
그래도 이들은 모두 붕어빵입니다.



사람을 사례로 든 클래스와 객체 사례

클래스: 사람

이름, 직업,나이, 성별, 혈액형
밥 먹기, 잠자기, 말하기, 걷기

이름 최승희
직업 의사
나이 45
성별 여
혈액형 A

객체 : 최승희

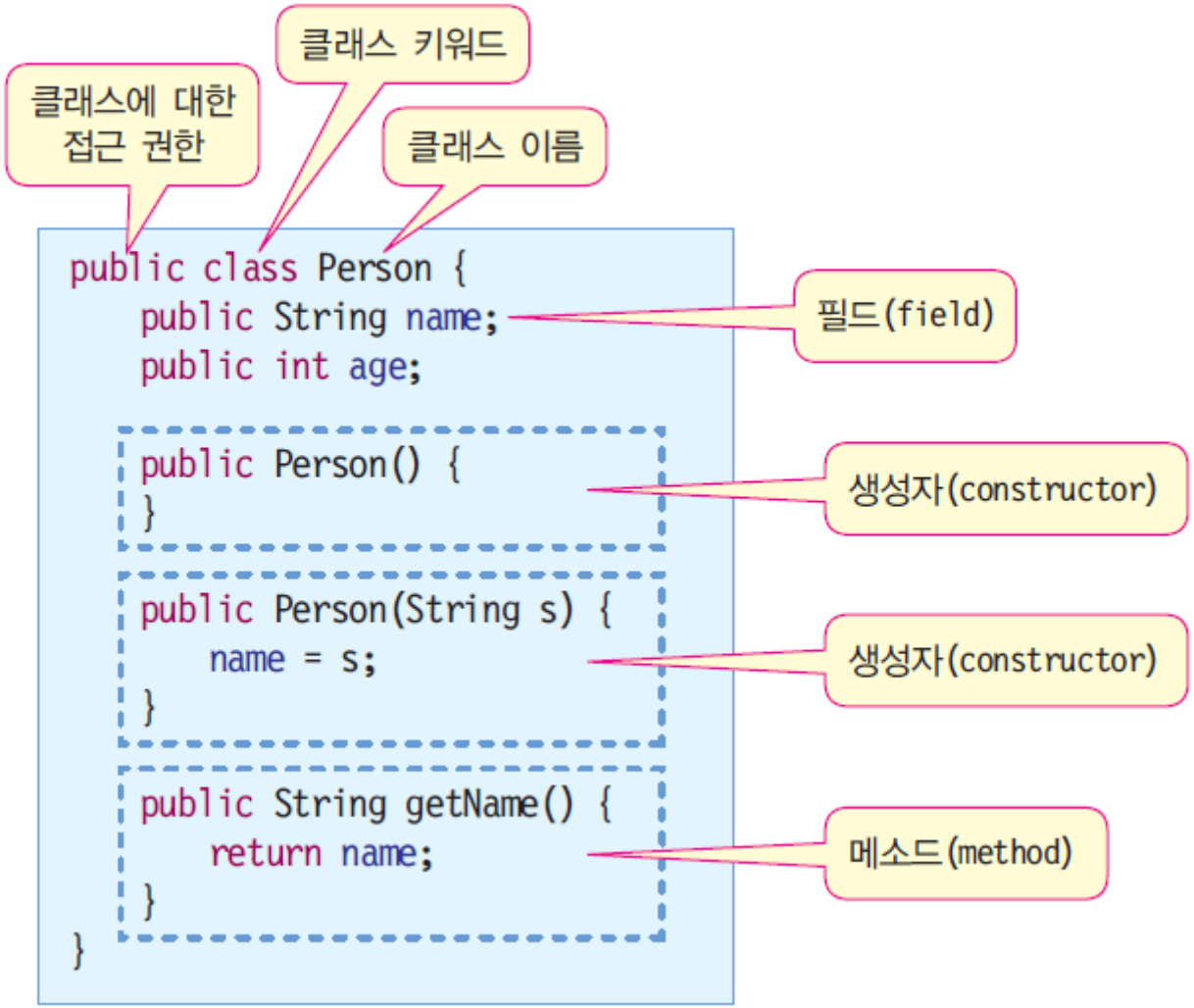
이름 이미녀
직업 골프선수
나이 28
성별 여
혈액형 O

객체 : 이미녀

이름 김미남
직업 교수
나이 47
성별 남
혈액형 AB

객체:김미남

클래스 구성


클래스 키워드
클래스에 대한 접근 권한
클래스 이름
public class Person {
public String name;
public int age;
public Person() {
}
public Person(String s) {
name = s;
}
public String getName() {
return name;
}
}
필드(field)
생성자(constructor)
생성자(constructor)
메소드(method)

클래스 선언

- 클래스 접근 권한, public
  - 다른 클래스들에서 이 클래스를 사용하거나 접근할 수 있음을 선언
- class Person
  - Person이라는 이름의 클래스 선언
  - 클래스는 {로 시작하여 }로 닫으며 이곳에 모든 필드와 메소드 구현
- 필드(field)
  - 값을 저장할 멤버 변수
    - 멤버 변수 혹은 필드라고 함
  - 필드의 접근 지정자 public
    - 필드를 다른 클래스의 메소드에서 접근할 수 있도록 공개한다는 의미
- 생성자(constructor)
  - 클래스의 이름과 동일한 메소드
  - 클래스의 객체가 생성될 때만 호출되는 메소드
- 메소드(method)
  - 메소드는 함수이며 객체의 행위를 구현
  - 메소드의 접근 지정자 public
    - 메소드를 다른 클래스의 메소드에서 호출할 수 있도록 공개한다는 의미

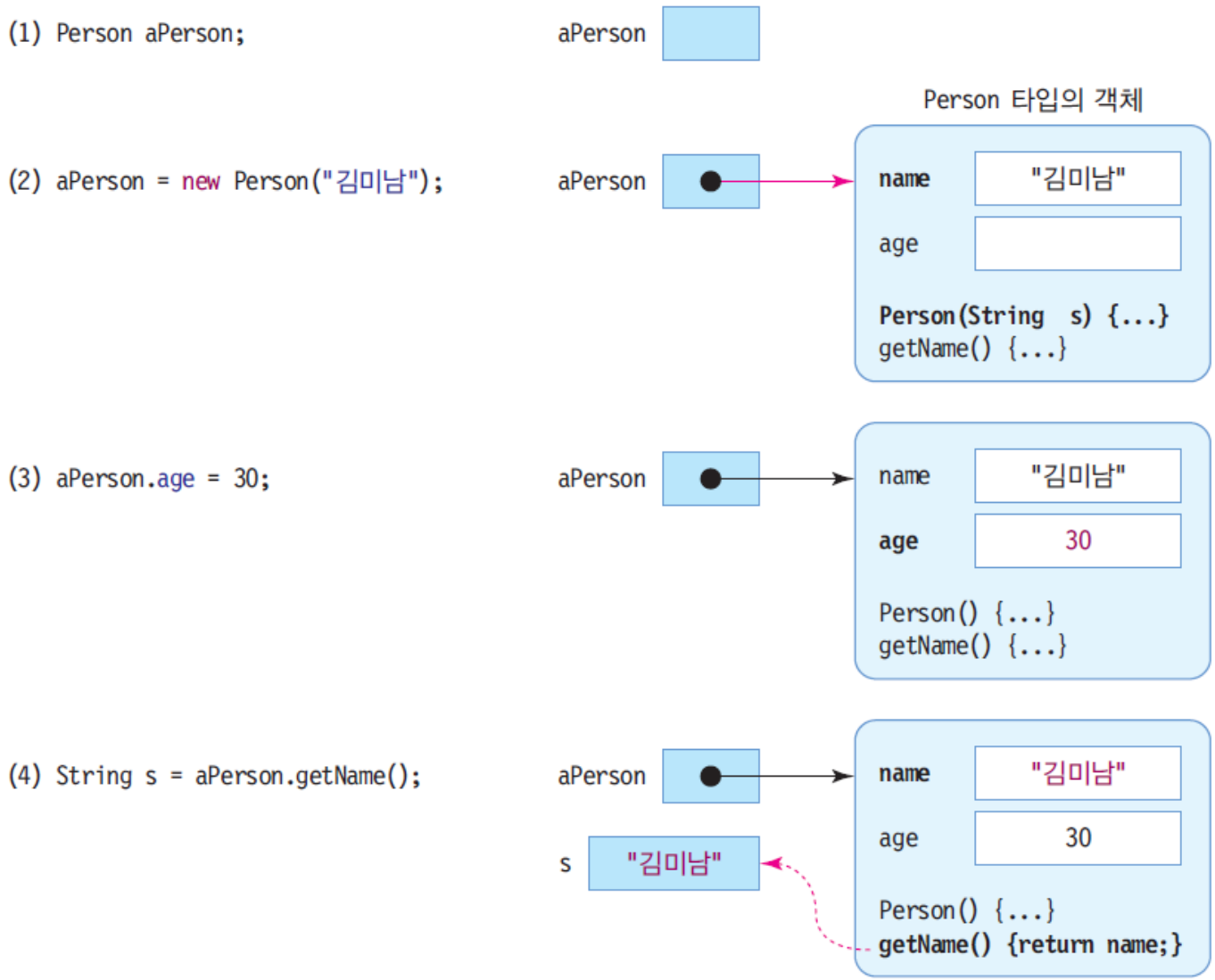
객체 생성

- 객체 생성
  - new 키워드를 이용하여 생성
    - new는 객체의 생성자 호출
- 객체 생성 과정
  1. 객체에 대한 레퍼런스 변수 선언
  2. 객체 생성

```
public static void main (String args[]) {
        Person aPerson;                    // 1. 레퍼런스 변수 aPerson 선언
        aPerson = new Person("김미남");     // 2. Person 객체 생성

        aPerson.age = 30;                  // 객체 멤버 접근
        int i = aPerson.age;               // 30
        String s = aPerson.getName();      // 객체 메소드 호출
}
```

객체 생성 및 사용 예


(1) Person aPerson;
aPerson
Person 타입의 객체
(2) aPerson = new Person("김미남");
aPerson
name
"김미남"
age
Person(String s) {...}
getName() {...}
(3) aPerson.age = 30;
aPerson
name
"김미남"
age
30
Person() {...}
getName() {...}
(4) String s = aPerson.getName();
aPerson
s
"김미남"
name
"김미남"
age
30
Person() {...}
getName() {return name;}