프로그래밍을 시작하는 사람들을 위한

GOOD JAVA

우재남 지음

한빛아카데미 Hanbit Academy, Inc.

지은이 **우재남** 5288893@hanafos.com

서강대학교에서 정보시스템 전공으로 석사과정을 마친 후 다양한 IT 관련 분야에서 실전 업무를 경험했으며, 여러 대학에서 프로그래밍, 데이터베이스 등의 과목을 강의해왔다. 현재는 디티솔루션의 공간데이터베이스 연구소장으로 재직 중이며, 공간정보와 IT의 융합 학문인 유시티 분야의 공학박사 학위도 취득했다. 집필 및 강의의 모토는 다양한 IT 실무 경험과 관련 지식을 독자와 수강생에게 최대한 쉽고 빠르게 전달하는 것이다.

저서로는 『뇌를 자극하는 Redhat Fedora : 리눅스 서버 & 네트워크』(2005), 『뇌를 자극하는 SQL Server 2005』(2006), 『IT CookBook for Beginner, C 언어 기초』(2008), 『뇌를 자극하는 SQL Server 2008』(2009), 『뇌를 자극하는 Redhat Fedora : 리눅스 서버 & 네트워크(개정판)』(2010), 『뇌를 자극하는 Windows Server 2008』(2011), 『IT CookBook 안드로이드 프로그래밍』(2012), 『뇌를 자극하는 SQL Server 2012(1권 : 기본편)』(2013), 『뇌를 자극하는 SQL Server 2012(2권 : 관리/응용편)』(2014), 『IT CookBook, C 언어 for Beginner(개정판)』(2014), 『뇌를 자극하는 Redhat Fedora : 리눅스 서버 & 네트워크(3판)』(2014), 『IT CookBook, Eclipse를 활용한 안드로이드 프로그래밍』(2015), 『IT CookBook, Android Studio를 활용한 안드로이드 프로그래밍』(2015), 『이것이 리눅스다』(2015), 『뇌를 자극하는 Windows Server 2012 R2』(2016) 등이 있으며, 『Head First HTML and CSS(개정판)』(2013)를 번역했다.

GOOD JAVA

초판발행 2016년 2월 12일

지은이 우재남 / **펴낸이** 김태헌
펴낸곳 한빛아카데미(주) / **주소** 서울시 마포구 잔다리로7길 16 한빛아카데미(주)
전화 02-336-7195 / **팩스** 02-336-7199
등록 2013년 1월 14일 제2013-000013호 / **ISBN** 979-11-5664-244-2 93000

총괄 배용석 / **책임편집** 김현용 / **기획** 김현용 / **편집** 김지선
디자인 표지 더 그라프, 내지 여동일
영업 이윤형, 길진철, 유제욱, 김태진, 주희 / **마케팅** 김호철

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 아래의 홈페이지나 이메일로 알려주십시오.
잘못된 책은 구입하신 서점에서 교환해 드립니다. 책값은 뒤표지에 표시되어 있습니다.
홈페이지 www.hanbit.co.kr / 이메일 question@hanbit.co.kr

Published by HANBIT Academy, Inc. Printed in Korea

지금 하지 않으면 할 수 없는 일이 있습니다.
책으로 펴내고 싶은 아이디어나 원고를 메일(writer@hanbit.co.kr)로 보내주세요.
한빛아카데미(주)는 여러분의 소중한 경험과 지식을 기다리고 있습니다.

한빛아카데미(주)는 한빛미디어(주)의 대학교재 출판 부문 자회사입니다.

C 언어를 배우지 않은,
프로그래밍 초보자를 위한 JAVA 책

이 책을 집필하면서 필자가 프로그래밍 언어를 처음 만났을 때를 다시 떠올려봤습니다. 난생처음 보는 이상한 문장들을 마주하고는 '뭐 이렇게 어려운 게 다 있지?'라고 생각했던 기억이 납니다. 지금 여러분도 이 책을 훑어보면서 똑같은 생각을 했을지도 모르겠습니다.

하지만 IT 입문자가 반드시 거쳐야 하는 것이 JAVA나 C와 같은 프로그래밍 언어입니다. JAVA나 C는 모든 운동선수가 필수적으로 갖춰야 할 '체력'과 같습니다. 체력이 좋은 선수는 어떤 종목을 하더라도 금방 적응해서 좋은 결과를 얻을 수 있듯이, 프로그래밍 언어에 익숙해진다면 나중에 어떤 IT 관련 분야를 접하더라도 쉽고 빠르게 익힐 수 있습니다.

필자 역시 몇 줄 안 되는 프로그램 코드가 오류 없이 돌아가고, 빈 화면에 결과가 나타나는 경험을 수없이 반복하고 나니, 이제는 데이터베이스, 알고리즘, 운영체제, 네트워크 등 IT 분야의 어떤 종목도 별로 어렵지 않고 처음 보는 내용도 쉽게 느껴집니다.

이 책은 처음 프로그래밍 언어를 접하는 독자를 위해 필자의 초보 시절을 떠올리며 이야기를 풀어나가듯 집필했습니다. 꼭 알아야 할 개념을 엄선하여 예제와 함께 구성했으며, 중요한 내용은 약간의 응용과 퀴즈 형식을 통해 계속 등장시켜 독자의 뇌리에 새겨질 수 있도록 했습니다. 그래서 머리만 복잡하게 만드는 것들은 되도록 간략하게 다루거나 과감히 생략했습니다.

최대한 쉽고 재미있게 쓰고자 했으니, 모든 JAVA 문법을 배워야만 프로그램을 짤 수 있다는 생각은 버리고 일단 프로그램을 입력하고 실행해보는 것부터 시작하면 됩니다. 필자가 그랬던 것처럼 여러분도 프로그래밍의 재미에 푹 빠지기를 간절히 기대합니다. 나아가 그러한 경험을 토대로 진정한 IT 전문가로 거듭난다면 제게는 더없는 기쁨과 보람이 될 것입니다.

필자의 『IT CookBook, C 언어 for Beginner』(초판 2008, 개정판 2014)는 많은 교수님과 학생들에게 관심과 사랑을 받았습니다. 그리고 JAVA 프로그래밍 책을 집필해달라는 많은 독자들의 꾸준한 요청으로 이 책을 출간하게 되었습니다. 대부분의 JAVA 책이 C 언어나 다른 언어에 어느 정도 익숙한 사람을 대상으로 한다면, 이 책은 다른 프로그래밍 언어를 배우지 않고 처음 프로그래밍을 학습하는 독자를 위한 것입니다. 따라서 책의 앞부분에서는

프로그래밍 입문자가 가장 어려워하는 객체지향의 개념을 최대한 배제하고 입문용 프로그래밍 언어 관점에서 구성했습니다.

이미 JAVA를 공부해본 독자라면 JAVA의 가장 큰 특징인 객체지향의 색깔이 나타나지 않아서 의아해할 수 있겠지만, 프로그래밍 언어를 처음 접하는 사람에게는 프로그래밍의 개념을 잡는 데 좋은 학습 방법이 될 것입니다. 기본적인 프로그래밍 언어에 어느 정도 익숙해진 다음, 책의 중반부 이후에 객체지향의 특징과 개념을 소개함으로써 자연스럽게 JAVA를 익힐 수 있도록 했습니다. 그리고 이 책을 학습한 프로그래밍 입문자가 충분히 완성할 수 있는 수준의 '실전 프로젝트'를 마지막에 실어 본문의 세분된 내용을 종합적으로 응용해볼 수 있도록 했습니다. 이를 통해 독자들은 JAVA뿐 아니라 다른 프로그래밍 언어도 어렵지 않게 익힐 수 있는 능력을 지니게 될 것입니다.

책이 나오기까지 물심양면으로 지원해주신 한빛아카데미(주) 임직원 여러분께 감사의 말씀을 드립니다. 특히 배용석 이사님과 김현용 팀장님, 그리고 책을 개발·편집해준 김지선 대리님께 감사드립니다. 그리고 제가 열심히 강의하고 집필할 수 있도록 조언과 충고를 해주시는 주위의 교수님들께도 이 자리를 빌려 다시 한 번 고마운 마음을 전합니다. 끝으로 함께 하는 것만으로도 즐겁고 행복한 가족에게 무한한 사랑을 보냅니다.

2016년 초 어느 깊은 밤 연구실에서

저자 우재남

이 책의 사용 설명서

무엇을 다루는가

❶ JAVA 언어 맛보기(1~2장)

- JAVA의 개요와 특징
- JAVA 프로그래밍 개발 환경
- JAVA 맛보기 프로그램 만들기

❷ JAVA 언어 기본(3~10장)

- 변수와 데이터 형식
- 연산자, 조건문, 반복문, 배열
- 문자열과 메소드, 예외 처리

❸ JAVA 고급 개념(11~14장)

- 객체지향 프로그래밍의 개념
- 클래스, 상속, 인스턴스, 메소드
- GUI 프로그래밍

❹ JAVA 실전 프로젝트(15장)

- 친구 연락처 관리 프로그램
- 사진 처리 프로그램

예제 소스

실습에 필요한 자료는 다음 주소에서 내려받을 수 있습니다.

http://www.hanbit.co.kr/exam/4244

실습 환경

이 책의 실습에 필요한 환경은 다음과 같습니다.

- **운영체제 : Windows 7**(Windows 7 이후라면 모두 사용 가능)
- **개발도구 : JDK 8**(JDK 7도 가능), **이클립스**

학습에 유용한 사이트

- **JAVA 개발 관련 공식 홈페이지 : http://www.oracle.com/technetwork/java/index.html**

  JAVA 최신 개발 도구 및 관련 정보를 얻을 수 있습니다.

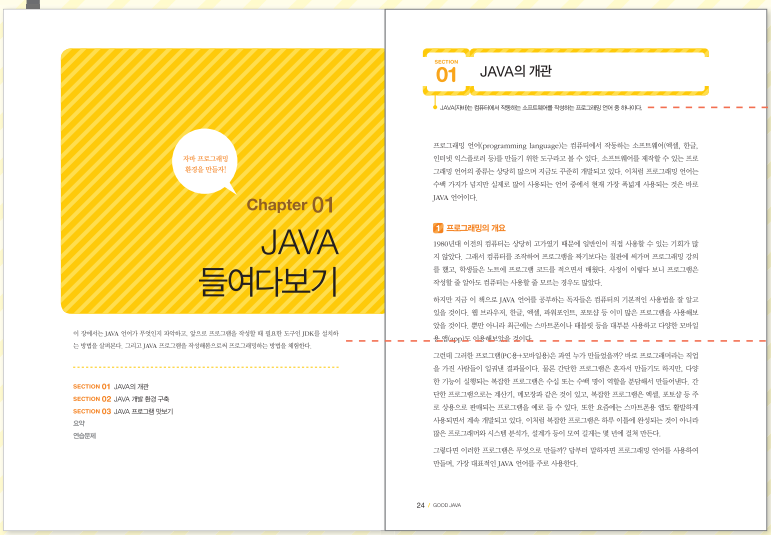
- **이것이 자바다 카페 : http://cafe.naver.com/thisisjava**

  JAVA와 관련된 스터디 카페로 동영상 강좌를 비롯해 다양한 정보를 얻을 수 있습니다

이 책의 구성

이 책은 프로그래밍을 처음 접하는 독자를 위한 기본서입니다. 좀 더 쉽게 학습 방향을 잡고 기본을 다지며 실력을 기를 수 있도록 다양한 학습 장치로 구성되어 있습니다. 이 책이 제안하는 다음과 같은 4단계 학습법을 통해 학습 능률을 배가하길 바랍니다.

1 단계 워밍업 본격적으로 학습을 시작하기 전에 그 장에서 배울 내용을 알려주고 학습 방향을 제시합니다.



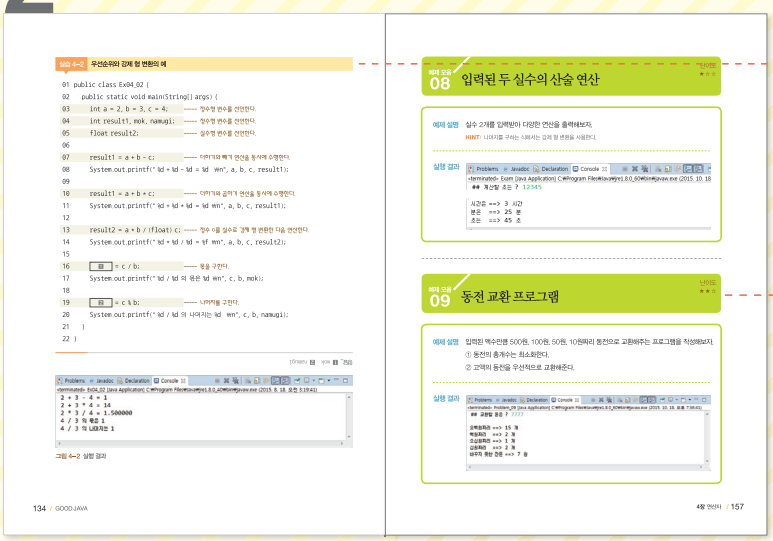
핵심 내용

그 절에서 배우는 내용을 간략하게 정리하여 핵심을 파악할 수 있습니다.

학습 목표

학습 방향을 제시함으로써 무엇을 왜 배워야 하는지 알려줍니다.

2 단계 기본기 다지기 실습을 통해 개념을 이해하고, 다양한 예제 모음을 통해 실력을 키웁니다.



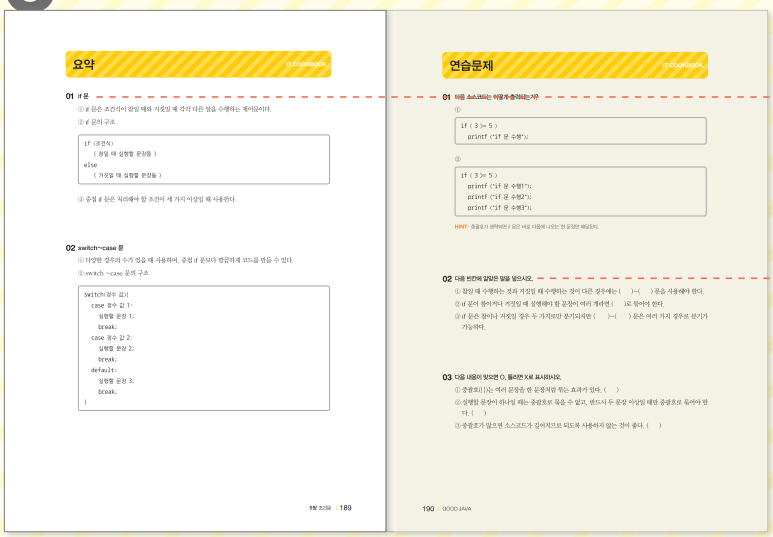
실습

본문에서 설명한 내용을 익힐 수 있는 실습 예제입니다. 코드에 대해 설명하고 실행화면을 보여주어 초보자도 쉽게 따라 할 수 있습니다.

예제 모음

본문 내용을 응용한 예제 모음입니다. 학습자가 반복 연습할 수 있는 다양한 예제로 구성되어 있습니다.

3단계 이해력 점검하기 장별 요약과 연습문제를 통해 배운 내용을 정리하고 문제 해결력을 기릅니다.



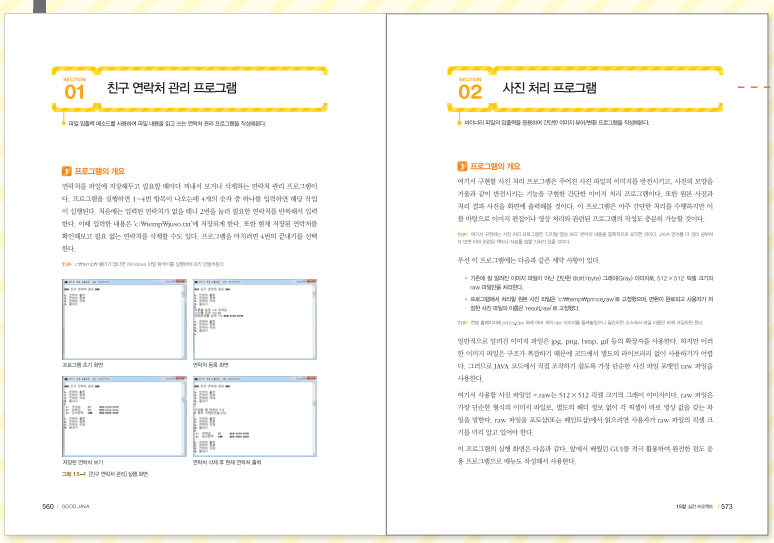
요약

각 장의 핵심 내용을 요약해서 정리했습니다. 세분된 지식을 정리, 종합하여 살펴볼 수 있습니다.

연습문제

핵심 내용을 문제 형식으로 제시했습니다. 본문에서 익힌 내용을 다시 확인함으로써 응용력을 배가할 수 있습니다.

4단계 응용력 기르기 앞에서 배운 내용을 종합적으로 적용해볼 수 있는 프로젝트를 통해 실전 감각을 기릅니다.



프로젝트

본문의 세분된 내용을 종합적으로 응용해볼 수 있는 프로젝트입니다. 코드가 좀 더 길고 추가로 알아야 할 내용도 있지만 앞서 배운 내용을 바탕으로 차근차근 따라 하면 큰 어려움 없이 완성할 수 있습니다.

목차

Chapter 01 JAVA 들여다보기 23

이 장에서는 JAVA 언어가 무엇인지 파악하고, 앞으로 프로그램을 작성할 때 필요한 도구인 JDK를 설치하는 방법을 살펴본다. 그리고 JAVA 프로그램을 작성해봄으로써 프로그래밍하는 방법을 체험한다.

실/습/목/차

Chapter 02 JAVA 프로그래밍 시작하기 51

1장에서 JAVA가 무엇인지 대략적으로 파악하고 컴파일러 프로그램을 설치해보았다. 또한 프로그램을 어떻게 작성하고 실행하는지 간략하게 다루었다. 이 장에서는 그보다 한발 더 나아가 어느 정도 완성된 프로그램을 만들어보자.

실/습/목/차

Chapter 03 변수와 데이터 형식 85

이 장에서는 JAVA의 문법을 차근차근 익혀보자. 필자 역시 문법에 치우치는 것을 그리 좋아하지 않으니 문법은 간단히 설명하고 예제를 통해 이해할 수 있도록 진행해나갈 것이다. 독자들도 직접 코딩을 하면서 따라오다 보면 고급 프로그래머에 한 발짝 다가가게 될 것이다.

실/습/목/차

Chapter 04 연산자 131

JAVA를 사용한다면 당연히 많은 계산을 하게 되는데, 이때 연산자를 이용한다. 이 장에서는 산술 연산자를 비롯해 증감 연산자, 관계 연산자, 논리 연산자, 비트 연산자를 살펴볼 것이다. 이러한 내용은 다른 장에서도 많이 사용되는 것이니 잘 익혀둔다.

실/습/목/차

Chapter 05 조건문 167

지금까지 배운 프로그램은 main( ) 메소드 내의 첫 줄부터 순차적으로 실행되었다. 만약 실행 순서를 바꾸거나 특정 부분을 반복하려면 어떻게 해야 할까? 이때는 조건문을 사용한다. 이 장에서는 조건문 가운데 if 문과 switch 문에 대해 알아보자.

실/습/목/차

Chapter 06 반복문의 기본, for 문 193

반복문은 동일한 기능이나 문장을 반복해서 실행하는 프로그램을 만들 때 사용하는 매우 효율적인 구문이다. JAVA에서 가장 많이 사용하는 구문 중 하나로서 특히 활용도가 높은 for 문의 동작과 사용법에 대해 살펴보자.

실/습/목/차

Chapter 07 반복문의 심화, while 문 233

6장에서 기본적인 반복문으로 for 문을 다루었다. 이제 for 문이 등장하는 프로그램에는 자신 있을 것이다. 이 장에서는 for 문과 비슷한 기능을 하는 while 문과 do~while 문에 대해 알아본다. 그리고 프로그램의 흐름을 조절하는 다양한 구문도 살펴볼 것이다.

실/습/목/차

Chapter 08 배열 263

배열은 실무에서 JAVA 프로그래밍을 할 때 필수적으로 사용되는 중요한 개념이다. 배열이 없었다면 반복적인 작업을 일일이 수행하는 수고를 피할 수 없었을 것이다. 배열은 JAVA 외의 다른 프로그래밍 언어에서도 중요하게 쓰이는 개념이므로 기초를 탄탄히 다져야 한다.

실/습/목/차

Chapter 09 문자열과 메소드 301

이 장에서는 JAVA에서 다루는 문자열과 문자열 메소드에 대해 학습한 뒤, JAVA뿐 아니라 대부분의 프로그래밍 언어에서 중요하게 사용되는 개념인 메소드(함수)에 대해 상세히 알아본다. 그리고 프로그램의 효율성을 높여주는 메소드의 개념을 파악하고, 사용 범위와 관계된 전역변수와 지역변수에 대해서도 살펴볼 것이다. 또한 메소드를 사용할 때 반드시 알아야 하는 반환 값과 매개변수도 다룬다.

실/습/목/차

Chapter 10 예외 처리와 파일 입출력 349

프로그램을 작성하다 보면 오류가 발생하고 오류를 수정해야만 프로그램이 정상적으로 작동할 수 있다. 이러한 오류를 JAVA나 운영체제가 아닌 프로그래머가 직접 처리하는 것을 예외 처리라고 한다. 이 장에서는 예외 처리에 대해 설명한 다음, 키보드와 화면에서 입력 및 출력을 하는 표준 입출력과 하드디스크에서 파일을 읽어오거나 저장해야 하는 파일 입출력을 살펴볼 것이다.

실/습/목/차

Chapter 11 객체지향 프로그래밍의 기초 389

JAVA 언어의 가장 큰 특징인 객체지향 프로그래밍에 대해 살펴보자. JAVA를 적극적으로 활용하려면 객체지향적인 특징을 잘 이해하고 프로그래밍에 적용해야 한다. 객체지향 프로그래밍은 대규모 소프트웨어 개발에 적합한 프로그래밍 기법으로, JAVA는 대표적인 객체지향 프로그래밍 언어 중 하나이다. 이는 단순한 개념이 아니라 초보자의 입장에서는 어렵게 느껴질 수도 있으므로 11장과 12장에 걸쳐 객체지향 프로그래밍의 개념과 기법을 차근차근 살펴볼 것이다.

실/습/목/차

Chapter 12 객체지향 프로그래밍의 응용 431

11장에서는 클래스의 개념을 파악하고 그 용도를 살펴보았다. 이 장에서는 클래스의 상속에 대해 알아본 다음 오버라이딩, 추상 클래스, 인터페이스 등 객체지향 프로그래밍을 위한 응용 학습을 할 것이다.

실/습/목/차

Chapter 13 GUI 프로그래밍 471

지금까지는 텍스트 환경에서 값을 입력하고 결과를 확인했는데, 이 장에서는 GUI 환경에서 프로그램을 작성하는 방법을 알아볼 것이다. 대표적인 GUI 응용 프로그램으로는 Windows의 메모장, 계산기, 그림판 등을 들 수 있으며, 스윙을 통해 이와 비슷한 Windows용 응용 프로그램을 제작하는 방법을 학습할 것이다. 이 장에서 다루는 레이아웃, 컴포넌트, 이벤트 처리, 메뉴, 툴바 등은 실무에서 응용할 수 있는 것이다.

실/습/목/차

Chapter 14 고급 프로그래머로 나아가기 521

이 장에서는 본격적으로 고급 프로그래밍을 배우기 위해 알아둬야 할 몇 가지 내용을 소개할 것이다. 초보자에게는 아직 어려울 수 있지만 실무에서는 자주 사용하는 개념이니 잘 이해해야 한다.

실/습/목/차

Chapter 15 실전 프로젝트 557

이 장에서는 지금까지 단편적으로 배웠던 내용을 종합하여 2개의 프로젝트를 수행할 것이다. 앞에서 다룬 실습보다 코드가 더 길지만 대부분 이미 배운 내용이므로 차근차근 살펴보면 문제없이 프로젝트를 수행할 수 있을 것이다.

JAVA의 꽃!
객체지향을
알아보자.

Chapter 11
객체지향 프로그래밍의 기초

JAVA 언어의 가장 큰 특징인 객체지향 프로그래밍에 대해 살펴보자. JAVA를 적극적으로 활용하려면 객체지향적인 특징을 잘 이해하고 프로그래밍에 적용해야 한다. 객체지향 프로그래밍은 대규모 소프트웨어 개발에 적합한 프로그래밍 기법으로, JAVA는 대표적인 객체지향 프로그래밍 언어 중 하나이다. 이는 단순한 개념이 아니라 초보자의 입장에서는 어렵게 느껴질 수도 있으므로 11장과 12장에 걸쳐 객체지향 프로그래밍의 개념과 기법을 차근차근 살펴볼 것이다.

SECTION 01

클래스

클래스는 설계도이며, 인스턴스(객체)는 설계도를 통해 제작된 실제 물건이다.

클래스(class)는 JAVA를 처음 공부하는 사람에게 가장 생소하게 느껴지는 개념 중 하나이다. 하지만 클래스의 개념을 알고 JAVA에서 실제로 클래스를 사용하다 보면 오히려 클래스가 없는 프로그래밍이 더 어색하게 느껴질 것이다. 먼저 실제 구현되는 코드를 가지고 클래스에 대해 이해해보자.

1 클래스의 개념

JAVA는 완전한 객체지향 프로그래밍 언어이며, 객체지향 프로그래밍 언어에서 가장 핵심적인 개념이 클래스라고 할 수 있다. 10장까지는 객체지향에 대한 개념을 최대한 배제하고 다른 프로그래밍 언어와 공통되는 내용으로만 구성했다. 하지만 이미 클래스를 계속 사용해왔는데 지금부터 클래스를 제대로 파헤쳐보자.

클래스의 형태

지금까지 작성한 모든 코드는 다음과 같은 형태였다. 즉 코드에 클래스의 개념이 이미 포함되어 있었던 것이다.

```
public class 클래스이름 {
   // 이  부분에 관련 코드 구현
}
```

위 코드에서 public을 지우면 다음과 같은 간단한 형태가 된다.

```
class 클래스이름 {
   // 이  부분에 관련 코드 구현
}
```

TIP/ public은 접근 제어 수식어(access control modifier) 또는 접근 제한자라고 일컫는다. 이에 대해서는 뒤에서 다시 설명할 것이다.

저자 한마디

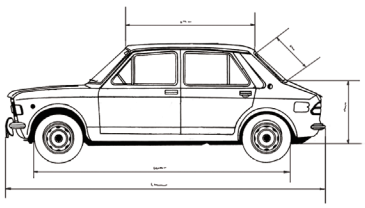
객체지향 프로그래밍의 특징

객체지향 프로그래밍을 지원하는 언어는 대표적으로 JAVA, C#, C++ 등이다. 객체지향 프로그래밍의 특징은 다음과 같이 몇 가지로 요약되는데, 어렵게 느껴진다면 일단 읽어보고 넘어가자.

- 추상화란 불필요한 정보의 노출을 최소화하고 꼭 필요한 정보만 노출하는 기법으로 캡슐화, 은닉화 등의 용어와 관련이 있다. 자료의 추상화를 위해 구현한 것이 클래스이다.
- 상속이란 기존에 만들어놓은 클래스의 기능을 그대로 물려받아서 사용하는 것을 말한다. 이렇게 하면 기존의 코드가 재사용되기 때문에 상당히 효율적인 프로그래밍이 가능하다.
- 다형성이란 같은 이름의 기능을 하는 요소를 여러 개 만드는 것을 말한다. 예를 들어 A라는 이름의 메소드 여러 개가 각각 다른 기능을 하도록 만들 수 있다.
- 동적 바인딩이란 실행할 시점에 동작이 변경될 수 있는 것을 의미하며, 컴파일할 때 동작이 결정되는 정적 바인딩과 반대되는 개념이다.

클래스 생성하기

클래스는 현실의 '사물'을 컴퓨터 안에서 구현하기 위해 고안된 개념이다. 예를 들어 자동차를 생각해보자. 자동차는 색상, 현재 속도 등의 상태(또는 속성)를 지니고 있으며 엑셀 밟기(속도 올리기), 브레이크 밟기(속도 내리기) 등의 기능도 할 수 있다. 이것을 클래스 형태로 표현해보자.



```
class 자동차 {
    // 자동차의 속성
    자동차 색상;
    자동차 속도;
    // 자동차의 기능
    속도 올리기();
    속도 내리기();
}
```

위와 같이 속성과 기능을 갖춘 자동차 클래스가 완성되었다. 이렇게 현실 세계의 다양한 사물을 클래스로 표현할 수 있다. 이제 자동차 클래스의 개념을 실제 코드로 구현해보자.

우선 자동차의 속성은 지금까지 사용했던 변수처럼 생성하면 되는데 이것을 필드(field)라고 한다. 또한 자동차의 기능은 지금까지 사용했던 메소드로 구현하면 된다.

```
class Car {
   // 자동차의 필드
   String 색상;
   int 현재 속도;
   // 자동차의 메소드
   void upSpeed(int 증가할_속도량) {
      // 현재 속도에서 증가할_속도량만큼 속도를 올리는 코드
   }
   void downSpeed(int 감소할_속도량) {
      // 현재 속도에서 감소할_속도량만큼 속도를 내리는 코드
   }
}
```

이번에는 자동차 클래스를 완전한 JAVA 코드로 만들어보자.

```
class Car {
   // 자동차의 필드
   String color;
   int speed;
   // 자동차의 메소드
   void upSpeed(int value) {
      speed = speed + value;
   }
   void downSpeed(int value) {
      speed = speed - value;
   }
}
```

필드의 이름과 메소드의 파라미터를 영문 변수명으로 변경했으며, 메소드는 자동차의 속도(speed)를 변경시키는 내용으로 완성했다. 이 코드는 잠시 후 실습에서 사용할 것이다.

메/멘/토 퀴/즈 자동차의 속성을 클래스에서는 □□라 하고, 자동차의 기능을 클래스에서는 □□□라 한다.

인스턴스 생성하기

앞에서 자동차 클래스를 완성했다. 하지만 자동차 클래스를 만들었다고 해서 자동차의 실체가 만들어진 것은 아니며 자동차의 '설계도'를 만든 것에 불과하다. 그렇다면 이 설계도를 바탕으

로 실제 자동차를 제작하는 작업을 해야 하는데, 이렇게 해서 생산되는 자동차를 '객체' 또는 '인스턴스'라고 한다.

TIP/ '객체'와 '인스턴스'는 동일한 용어라고 생각하면 되는데 이 책에서는 인스턴스라고 지칭할 것이다.

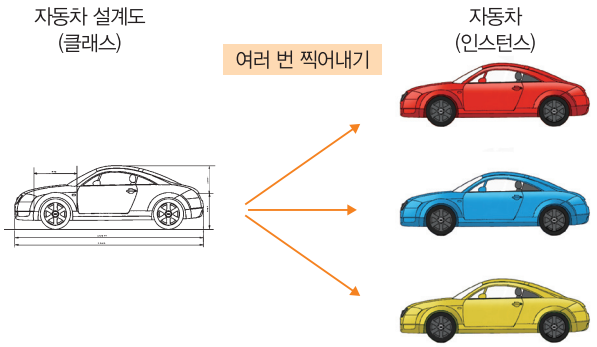


그림 11-1 클래스와 인스턴스의 개념

자동차 설계도가 완성되면 자동차를 여러 대 생산할 수 있듯이, 클래스를 만들고 나면 인스턴스를 여러 개 만들 수 있다. [그림 11-1]은 설계도(클래스)에 의해 여러 대의 자동차(인스턴스)를 생산해내는 개념을 나타낸 것이다.

다음으로 실제 클래스와 인스턴스의 형식을 살펴보자.

자동차 설계도(클래스)

```
class 자동차 {
    // 자동차의 속성
    자동차 색상;
    자동차 속도;
    //자동차의 기능
    속도 올리기();
    속도 내리기();
}
```

자동차(인스턴스)

```
자동차 1 = new 자동차( );
자동차 2 = new 자동차( );
자동차 3 = new 자동차( );
```

그림 11-2 클래스와 인스턴스의 코드 구성

자동차 3대의 인스턴스 생성을 실제 코드로 만들면 다음과 같다.

```
Car myCar1 = new Car();
Car myCar2 = new Car();
Car myCar3 = new Car();
```

또는 다음과 같이 인스턴스를 먼저 준비한 다음 객체를 대입해도 된다.

```
Car myCar1, myCar2, myCar3;
myCar1 = new Car();
myCar2 = new Car();
myCar3 = new Car();
```

자동차 1(myCar1), 자동차 2(myCar2), 자동차 3(myCar3)의 인스턴스를 생성했다. 이 3개의 인스턴스는 각각 자동차의 색상(color)과 속도(speed) 필드를 가지고 있다. 즉 자동차 1은 파란색에 현재 속도 30km, 자동차 2는 빨간색에 현재 속도 50km, 자동차 3은 노란색에 현재 속도 0km와 같이 완전히 별개의 자동차인 것이다.

인스턴스 필드에 값 대입하기

각 인스턴스는 독립적인 공간을 차지한다. 즉 각 인스턴스에는 별도의 필드가 존재하고 각각에 별도의 값을 대입할 수 있다.

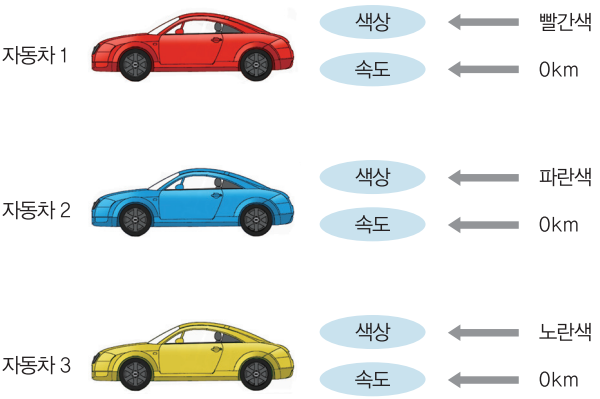


그림 11-3 인스턴스 필드에 값 대입하기

[그림 11-3]을 코드로 표현하면 다음과 같다.

```
myCar1.color = "빨간색";
myCar1.speed = 0;
myCar2.color = "파란색";
myCar2.speed = 0;
myCar3.color = "노란색";
myCar3.speed = 0;
```

TIP/ 처음에는 자동차가 멈춰 있기 때문에 속도를 0으로 초기화했다. 필요하다면 30, 50 등의 값을 대입해도 된다.

앞의 코드를 보면 각 인스턴스의 필드에 '인스턴스이름.필드이름' 형식으로 접근할 수 있다. 각 인스턴스마다 별도의 필드를 가지고 있음을 확인할 수 있다.

메소드 호출하기

Car 클래스에서 메소드는 upSpeed()와 downSpeed()를 만들어놓았다. 메소드도 각 인스턴스마다 별도로 존재한다고 생각하고 사용하면 된다. 메소드의 호출은 '인스턴스이름.메소드이름()' 형식을 사용한다.

```
myCar1.upSpeed(30);
myCar2.upSpeed(60);
```

myCar1은 30으로 속도를 증가시켰고, myCar2는 60으로 속도를 증가시켰다.

TIP/ 실제 메소드는 각 인스턴스별로 존재하지 않고 모든 인스턴스가 공유하는 개념이다. 즉 메소드는 인스턴스의 상태를 저장한 것이 아니라 동작을 표현한 것이기 때문에 공유해도 문제가 없다. 하지만 인스턴스마다 각각의 메소드를 가지고 있다고 생각해도 괜찮다.

2 클래스의 실제 코딩

부분적으로 작성했던 자동차 클래스의 코드를 완성해보자.

실습 11-1 자동차 클래스 생성 및 사용 예 1

```
01 class Car {                          ----- Car 클래스를 선언한다.
02    String color;                     ┐
03    int speed;                        ┘--- 자동차의 색상과 속도 필드를 정의한다.
04
05    void upSpeed(int value) {         ┐
06       speed = speed + value;         ├--- 파라미터로 추가 속도(value)를 받아서 현재 속도를 증가시킨다.
07    }                                 ┘
08
09    void downSpeed(int value) {       ┐
10       speed = speed - value;         ├--- 파라미터로 추가 속도(value)를 받아서 현재 속도를 감소시킨다.
11    }                                 ┘
12 }
13
14 public class Ex11_01 {
15    public static void main(String[] args) {
```

```
16      Car myCar1 = new Car();
17      myCar1.color = "빨강";
18      myCar1.speed = 0;
19
20      Car myCar2 = new Car();
21      myCar2.color = "파랑";
22      myCar2.speed = 0;
23
24      Car myCar3 = new Car();
25      myCar3.color = "노랑";
26      myCar3.speed = 0;
27
28      myCar1.upSpeed(30);
29      System.out.println("자동차1의 색상은 " + myCar1.color +
        "이며, 현재속도는 " + myCar1.speed + "km 입니다.");
30
31      myCar2.upSpeed(60);
32      System.out.println("자동차2의 색상은 " + myCar2.color +
        "이며, 현재속도는 " + myCar2.speed + "km 입니다.");
33
34      myCar3.upSpeed(0);
35      System.out.println("자동차3의 색상은 " + myCar3.color +
        "이며, 현재속도는 " + myCar3.speed + "km 입니다.");
36    }
37 }
```

myCar1 인스턴스를 생성하고 색상과 속도를 지정한다. 모든 속도는 초기에 0으로 설정한다.

myCar2 인스턴스를 생성하고 색상과 속도를 지정한다. 모든 속도는 초기에 0으로 설정한다.

myCar3 인스턴스를 생성하고 색상과 속도를 지정한다. 모든 속도는 초기에 0으로 설정한다.

myCar1의 속도를 30으로 변경하고 자동차의 색상과 속도를 화면에 출력한다.

myCar2의 속도를 60으로 변경하고 자동차의 색상과 속도를 화면에 출력한다.

myCar3의 속도를 0으로 변경하고 자동차의 색상과 속도를 화면에 출력한다.

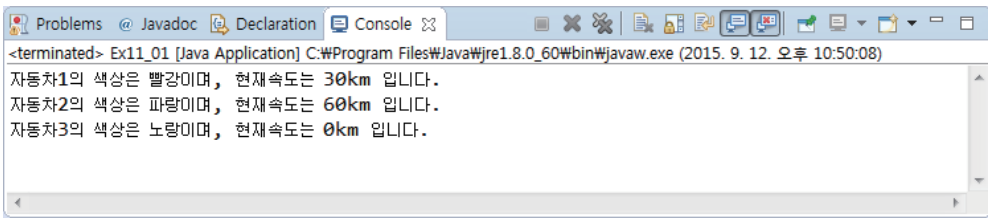


그림 11-4 실행 결과

[실습 11-1]은 단순하지만 클래스의 생성과 사용이 잘 표현되어 있다. 코드 자체는 어렵지 않지만 클래스를 처음 접하는 경우 혼란스러울 수도 있을 것이다. 다음과 같은 순서로 클래스를 사용한다고 기억해두자.

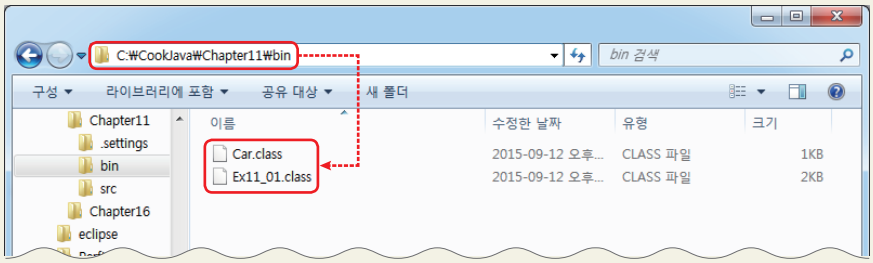
단계	작업	형식	예
1단계	클래스 생성하기	class 클래스이름 { // 필드 선언 // 메소드 선언 }	class Car { String color; void upSpeed ( ) { ~~~ }
2단계	인스턴스 생성하기	클래스이름 변수; 변수 = new 클래스이름();	Car myCar1; myCar1 = new Car( );
3단계	인스턴스 필드에 값 대입하기 또는 메소드 호출하기	변수.필드이름 = 값; 변수.메소드( );	myCar1.color = "빨강"; myCar1.upSpeed( );

앞으로 다양한 클래스를 만들고 사용할 테지만 위의 순서 및 큰 틀과 많이 다르지 않을 것이다.

저자 한마디

class 파일

[실습 11-1]은 소스 파일은 Ex11_01.java 1개이지만 그 안에 Car 클래스와 Ex11_01 클래스 2개가 있다. 그래서 이 파일을 실행하면 2개의 바이트코드 파일, 즉 Car.class와 Ex11_01.class가 생성된다. class 개수만큼 파일이 생성되는 것인데, 단 클래스 안에 클래스를 작성하는 내부 클래스는 외부 클래스 1개만 *.class로 생성된다. 내부 클래스는 12장에서 다시 살펴볼 것이다.



직접 풀어보기 11-1

[실습 11-1]은 속도에 제한이 없다. 만약 속도가 200km를 넘으면 최고 200km가 유지되도록 upSpeed() 메소드를 수정해보자.

HINT/ if 문으로 속도를 증가시킨 값이 200을 넘는 경우 200을 대입하면 된다.

3 필드와 메소드에 대한 접근 제한

앞의 실습에서 클래스를 이용하여 인스턴스를 만들었다. 그리고 필드나 메소드를 사용하려면 '변수.필드이름'이나 '변수.메소드()'로 접근이 가능했는데 이에 대해 좀 더 자세히 살펴보자.

필드와 메소드에 대한 접근

[실습 11-1]의 29, 32, 35행에서는 직접 myCar1.color나 myCar1.speed에 접근하여 색상과 속도를 출력했다. 이번에는 Car 클래스에 getColor()와 getSpeed() 메소드를 추가하여 이 메소드들이 현재 색상과 속도를 반환하도록 해보자. 코드가 길어지므로 인스턴스는 myCar1 하나만 사용한다.

TIP/ 동일한 이름의 클래스는 같은 패키지 안에 하나만 있어야 한다(패키지는 14장에서 다룰 것이다). 그러므로 다음 실습을 실행하려면 [실습 11-1] 전체를 주석(/* ~~ */)으로 묶는 것이 좋다.

실습 11-2 자동차 클래스 생성 및 사용 예 2

```
01 class Car {
02    String color;
03    int speed;
04
05    void upSpeed(int value) {
06       speed = speed + value;
07    }
08
09    void downSpeed(int value) {
10       speed = speed - value;
11    }
12
13    String getColor() {
14       return color;
15    }
16
17    int getSpeed() {
18       return speed;
19    }
20 }
21
22 public class Ex11_02 {
```

Car 클래스를 선언한다.

현재 자동차의 색상을 반환한다.

현재 자동차의 속도를 반환한다.

```
23    public static void main(String[] args) {
24       Car myCar1 = new Car();
25       myCar1.color = "빨강";
26       myCar1.speed = 0;
27
28       myCar1.upSpeed(30);
29       System.out.println("자동차1의 색상은 " + myCar1.color + "이며, 현재속도는 " + myCar1.
         speed + "km 입니다."); ----- myCar1 자동차의 색상과 속도를 화면에 출력한다.
30       System.out.println("자동차1의 색상은 " + myCar1.getColor() + "이며, 현재속도는 " +
         myCar1.getSpeed() + "km 입니다."); ----- myCar1 자동차의 색상과 속도를 화면에 출력한다.
31    }
32 }
```

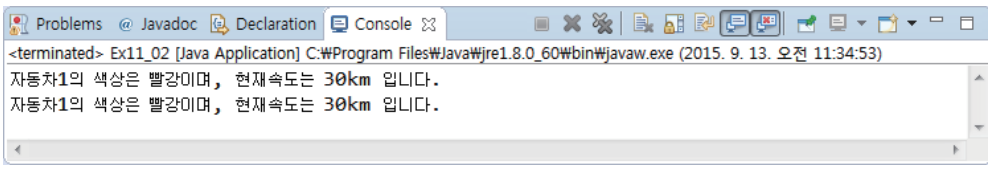


그림 11-5 실행 결과

[실습 11-2]는 [실습 11-1]에 Car 클래스의 메소드를 2개 추가했을 뿐이다. 13행의 getColor() 메소드는 현재 인스턴스에 설정된 색상을 반환하고, 17행의 getSpeed() 메소드는 현재 인스턴스에 설정된 속도를 반환한다. 주의 깊게 살펴볼 부분은 29행과 30행이다. 모두 myCar1의 색상과 속도를 출력하지만, 29행은 직접 color와 speed 필드에 접근했고, 30행은 getColor()와 getSpeed() 메소드를 호출함으로써 간접적으로 color와 speed 필드에 접근했다. 이 두 가지 중에서는 30행의 간접적인 접근 방식을 사용하는 것이 바람직하다. 29행처럼 직접 접근하면 실수로 필드 값을 변경하는 경우 원치 않은 결과가 나올 수 있기 때문이다.

private 접근 제어 수식어

JAVA에서는 필드에 직접 접근하지 못하도록 private 접근 제어 수식어를 제공한다. 필드 앞에 private를 붙이면 클래스 안의 메소드에서는 접근이 가능하지만, 인스턴스를 통해 직접 필드에 접근할 수는 없다.

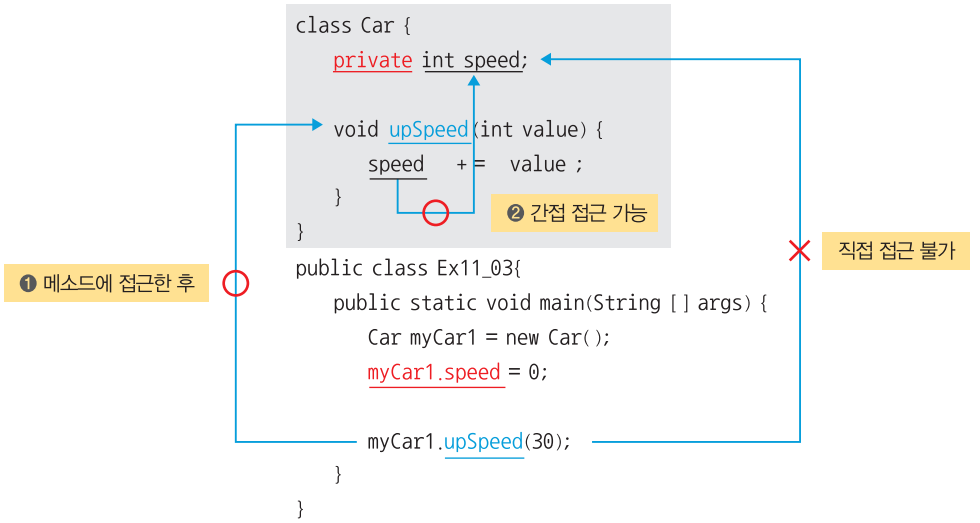


그림 11-6 필드에 private 접근 제어 수식어를 붙이는 경우

[그림 11-6]에서 speed 필드에 private 접근 제어 수식어를 붙이면 인스턴스인 myCar1을 통해서는 speed 필드에 직접 접근할 수 없다. 이는 프로그램의 오류를 방지하기 위한 코딩 방법으로 권장된다. 하지만 private 필드도 Car 클래스 내부의 upSpeed() 메소드에서는 접근이 가능하므로, main() 함수 본체에서는 myCar1.upSpeed(30)과 같이 메소드에 접근한 후 메소드를 통해 speed 필드에 간접 접근하는 방식으로 코드를 작성하면 된다.

TIP/ 일반적으로 필드에 private을 붙이는 방법을 권장하지만 예외적인 경우도 종종 있다. 이는 뒤에서 자세히 설명할 것이다.

[실습 11-2]의 필드에 private 접근 제어 수식어를 붙여서 코드를 수정해보자.

실습 11-3 private 접근 제어 수식어 사용 예

```
01 class Car {
02     [1] String color;          ---- 필드에 private 접근 제어 수식어를 붙인다.
03     private int speed;         ----
04
05     // [실습11-2]의 upSpeed(), downSpeed(), getColor(), getSpeed()와 동일
06                                          [실습 11-2]의 메소드와 동일하다.
07     void setColor(String color) {
08         this.color = color;    ---- color 필드의 값을 변경시켜 주는 메소드이다.
09     }                          ----
10
```

```
11    void setSpeed(int speed) {
12       this.speed = speed;
13    }
14 }
15
16 public class Ex11_03 {
17    public static void main(String[] args) {
18       Car myCar1 = new Car();
19       myCar1.[ 2 ]("빨강"); // myCar1.color는 오류
20       myCar1.setSpeed(0); // myCar1.speed는 오류
21
22       myCar1.upSpeed(30);
23       System.out.println("자동차1의 색상은 " + myCar1.getColor() + "이며, 현재속도는 " +
         myCar1.getSpeed() + "km 입니다.");
24    }
25 }
```

speed 필드의 값을 변경시켜 주는 메소드이다.

myCar1 자동차의 색상과 속도를 화면에 출력한다.

정답 1 private 2 setColor

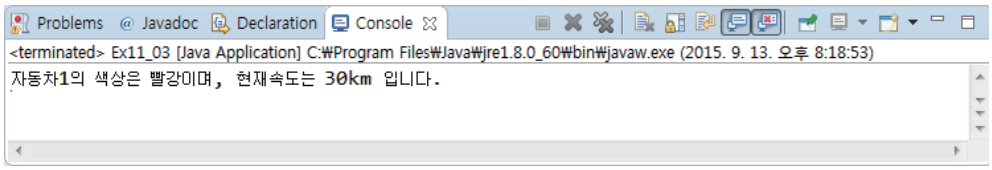


그림 11-7 실행 결과

2, 3행에서 private 예약어를 사용했다. 필드에 private을 붙이면 클래스 내부의 메소드에서는 접근할 수 있지만 그 외에서는 접근할 수 없다. 이런 경우에는 7행과 11행처럼 set○○○(변경할 값)과 같은 이름으로 값을 변경하는 메소드와 5행의 get○○○()과 같은 이름으로 값을 반환하는 메소드를 만드는 것이 일반적인 코딩 방법이다. 본체 프로그램에서 19, 20행과 같이 '인스턴스이름.set○○○(변경할 값)' 형식으로 메소드를 통해 값을 변경하고, 23행과 같이 '인스턴스이름.get○○○()' 형식으로 값을 얻어오면 된다. 8행과 12행의 this는 자신의 클래스를 가리킨다. 다음 그림을 살펴보자.

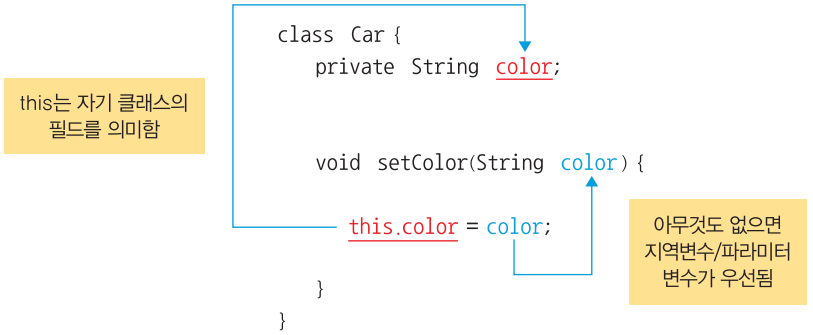


그림 11-8 this의 의미

setColor() 메소드 안에는 color라는 변수가 2개 있다. 파라미터 변수 color와 Car 클래스의 필드 color인데, 이 둘은 이름이 같기 때문에 구분하기 위해 Car 클래스의 필드에는 자신의 클래스를 의미하는 this를 붙였다. 아무것도 붙이지 않고 color라고 하면 파라미터 변수를 가리킨다.

public 접근 제어 수식어

public 접근 제어 수식어는 private과 반대로 외부(모든 클래스)에서 접근이 가능하도록 하는 예약어이다. 일반적으로 private은 필드 앞에 붙여서 사용하고, public은 메소드 앞에 붙여서 사용한다. 따라서 필드는 외부에서 함부로 변경할 수 없도록 하고, 외부에 공개된 메소드를 통해 접근하도록 하는 것이다. 앞으로 코딩을 할 때는 다음과 같은 기본적인 원칙을 기억하자.

저자 한마디

default와 protected 접근 제어 수식어

자주 사용하지는 않지만 default와 protected 접근 제어 수식어도 있다. 둘 다 같은 패키지 안에서는 접근이 가능하지만, default의 경우 하위 클래스는 접근하지 못하고 protected는 서브 클래스에서 접근이 가능하다. 특별히 필드나 메소드에 아무것도 붙이지 않으면 default가 기본으로 설정된다. 즉 같은 패키지 안에서는 접근이 가능하다. (서브 클래스는 12장에서, 패키지는 14장에서 살펴볼 것이다.)

public, private, default, protected, 4개의 접근 제어 수식어(또는 접근 제한자)를 통해 객체지향 프로그래밍의 특징인 캡슐화와 정보 은닉 등이 구현된다. 참고로 접근 제어 수식어는 public > protected > default > private의 순서로 공개적이다.

지금까지의 내용을 표로 요약하면 다음과 같다.

접근 제어 수식어	같은 클래스	같은 패키지	하위 클래스	외부 클래스
public	접근 ○	접근 ○	접근 ○	접근 ○
protected	접근 ○	접근 ○	접근 ○	접근 ×
default	접근 ○	접근 ○	접근 ×	접근 ×
private	접근 ○	접근 ×	접근 ×	접근 ×

```
필드 : private
메소드 : public
```

[실습 11-3]을 수정하여 Car 클래스 안의 6개 메소드에 public을 모두 붙여서 다시 실행해보자. [실습 11-3]과 동일하게 작동할 것이다.

실습 11-4 public 접근 제어 수식어 사용 예

```
01 class Car {
02     // [실습 11-3]의 6개 메소드 앞에 public을 붙임 ----- 6개 메소드에 public 접근 제어 수식어를 붙인다.
13     }
14 }
15
16 public class Ex11_04 {
17     public static void main(String[] args) {
18      // [실습 11-3]의 main() 메소드와 동일
24     }
25 }
```

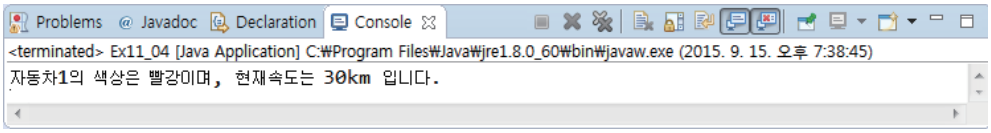


그림 11-9 실행 결과

2행에서 모든 메소드에 public을 붙였으므로 모든 클래스에서 이 6개의 메소드에 접근할 수 있게 되었다.

▶ **직접 풀어보기 11-2**

[실습 11-4]의 6개 메소드 앞에 protected를 붙여서 실행해보자.

private, public 접근 제어 수식어의 활용

private으로 필드에 대한 접근을 제한하면 실수로 잘못된 값을 대입하는 것을 막을 수 있다. 예를 들어 speed 필드에 private 지정자가 없다면 자동차의 속도가 200km를 넘거나 0km 미만이 되는 것을 방지할 수 없다. 하지만 upSpeed()나 downSpeed() 메소드에서 값을 확인하면 이러한 오류를 피할 수 있다.

실습 11-5 private, public 접근 제어 수식어 활용 예

```
01 class Car {
02    private int speed = 0;
03
04    public void upSpeed(int value) {
05       if (speed + value > 200)
06          speed = 200;
07       else
08          [1]
09
10       System.out.println("현재 속도 ==>" + getSpeed());
11    }
12
13    public void downSpeed(int value) {
14       if (speed - value < 0)
15          speed = 0;
16       else
17          [2]
18
19       System.out.println("현재 속도 ==>" + getSpeed());
20    }
21
22    public int getSpeed() {
23       return speed;
24    }
25 }
26
27 public class Ex11_05 {
28    public static void main(String[] args) {
29       Car myCar1 = new Car();
30
31       myCar1.upSpeed(100);
32       myCar1.upSpeed(150);
33
34       myCar1.downSpeed(50);
35       myCar1.downSpeed(160);
36    }
37 }
```

----- speed 필드에 private을 붙이고 초깃값으로 0을 할당한다.

속도를 올리는 메소드로 최대 200km로 제한한다.

속도를 내리는 메소드로 최소 0km로 제한한다.

처음 0에서 100km로 속도를 올리고, 추가로 150km를 더 올린다.

속도를 50km 내린 다음 추가로 160km를 더 내린다.

정답_ [1] speed += value; [2] speed -= value;

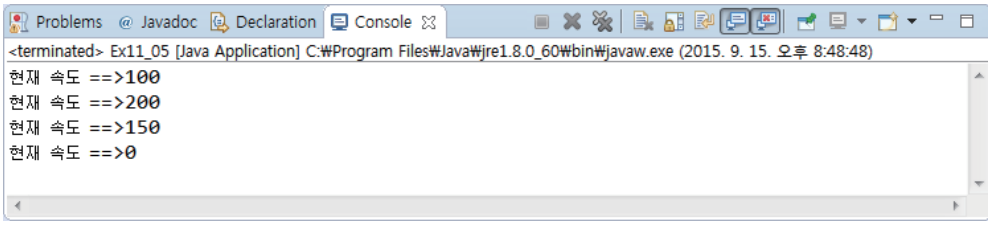


그림 11-10 실행 결과

[실습 11-5]는 private과 public의 좋은 예를 보여준다. 2행에서 speed에 private 접근 제어 수식어를 지정했기 때문에 27~37행 Car 클래스의 myCar1 인스턴스에서 직접 speed 변수를 수정할 수 없다. 그러므로 200km를 넘는 값이나 마이너스 속도를 speed에 입력하는 실수를 미연에 막아준다. speed를 수정하려면 upSpeed()나 downSpeed()를 사용해야 하며, 두 메소드 내부에서 0~200km의 속도만 유지하도록 되어 있다.

SECTION 02

생성자

생성자는 클래스의 이름과 동일한 메소드를 말하며, 주로 초기화할 때 사용된다.

[실습 11-1]의 16~18행을 다시 살펴보자.

```
16      Car myCar1 = new Car();
17      myCar1.color = "빨강";
18      myCar1.speed = 0;
```

16행에서 myCar1 인스턴스를 생성한 후, 17행에서 색상을 빨강으로 초기화하고, 18행에서는 속도를 0으로 초기화했다. 그런데 16행에서 인스턴스를 생성하면서 동시에 빨강과 0으로 초기화하면 어떨까? 코드가 훨씬 간결해지고, 인스턴스를 생성하면서 값을 초기화하기 때문에 필드에 초깃값을 대입하는 것을 잊어버리는 일도 없을 것이다.

1 생성자의 기본

먼저 생성자의 기본적인 형태를 살펴보자.

```
class 클래스이름 {
    클래스이름() {  // 생성자
       // 이 부분에 초기화할 코드를 입력
    }
}
```

생성자는 클래스와 이름이 동일하다. 예를 들면 Car 클래스는 다음과 같이 생성자를 만들 수 있다.

```
class Car {
   String color;
   int speed;
   Car( ) {  // 생성자
      color = "빨강";
```

```
        speed = 0;
    }
}
```

이제부터는 [실습 11-1]의 16~18행 중에서 16행만 있으면 된다. 즉 인스턴스를 생성하면 자동으로 생성자가 호출된다.

```
Car myCar1 = new Car();
```

메/멘/토 퀴/즈 Car 클래스의 생성자 이름은 □□□()이다.

기본 생성자

생성자를 사용한 완전한 코드를 살펴보자.

실습 11-6 생성자 사용 예 1

```
01 class Car {
02    private String color;
03    private int speed;
04
05    Car() {
06       color = "빨강";
07       speed = 0;
08    }
09
10    public String getColor() {
11       return color;
12    }
13
14    public int getSpeed() {
15       return speed;
16    }
17 }
18
19 public class Ex11_06 {
20    public static void main(String[] args) {
21       Car myCar1 = new Car();
22       Car myCar2 = new Car();
23
```

생성자를 작성하고 색상과 속도의 초깃값을 설정한다.

myCar1과 myCar2 인스턴스를 만든다. 자동으로 생성자가 호출된다.

```
24        System.out.println("자동차1의 색상은 " + myCar1.getColor() +
          "이며, 현재속도는 " + myCar1.getSpeed() + "km 입니다.");
25        System.out.println("자동차2의 색상은 " + myCar2.getColor() +
          "이며, 현재속도는 " + myCar2.getSpeed() + "km 입니다.");
26    }
27 }
```

생성자에 의해 색상과 속도가 모두 초기화되어 있다.

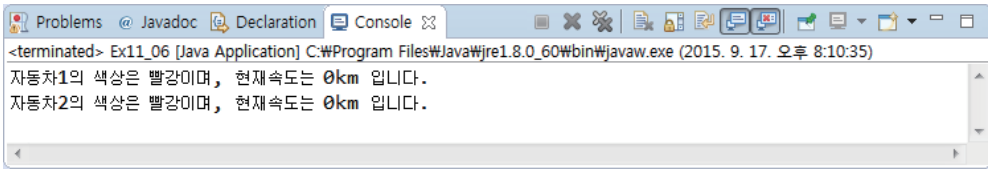


그림 11-11 실행 결과

5행의 생성자에 의해 21, 22행에서 자동으로 값이 초기화되었다. 하지만 myCar1과 myCar2가 모두 빨강과 0으로 초기화되었는데 두 차는 다르기 때문에 다른 초기화가 필요하다.

파라미터가 있는 생성자

생성자도 다른 메소드처럼 파라미터를 사용할 수 있다. [실습 11-6]을 수정하여 파라미터가 있는 생성자를 사용해보자. 그리고 인스턴스를 만들 때 초깃값을 파라미터로 넘기는 방법을 사용해보자.

실습 11-7 생성자 사용 예 2

```
01 class Car {
02    private String color;
03    private int speed;
04
05    Car(String color, int speed) {
06       this.color = color;
07       this.speed = speed;
08    }
09
10    // [실습 11-6]의 getColor() , getSpeed()  메소드와 동일
11    }
12 }
13
14 public class Ex11_07 {
15    public static void main(String[] args) {
```

생성자에 2개의 파라미터를 받는다. this를 붙여서 필드와 파라미터 변수를 구분한다.

```
16      Car myCar1 = new Car("빨강", 0);
17      Car myCar2 = new Car("파랑", 30);
18
19      System.out.println("자동차1의 색상은 " + myCar1.getColor() + "이며, 현재속도는 " +
        myCar1.getSpeed() + "km 입니다.");
20      System.out.println("자동차2의 색상은 " + myCar2.getColor() + "이며, 현재속도는 " +
        myCar2.getSpeed() + "km 입니다.");
21    }
22 }
```

인스턴스를 생성할 때 2개의 파라미터를 넘긴다.

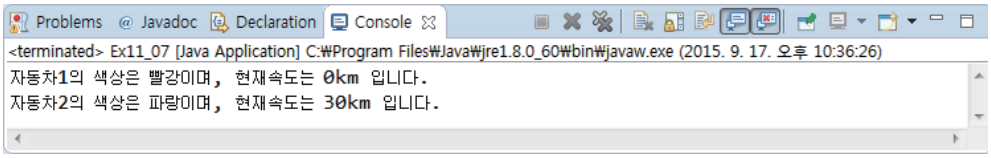


그림 11-12 실행 결과

5행의 생성자에서 2개의 파라미터를 받도록 설정해놓았다. 그리고 넘겨받은 파라미터 값으로 색상과 속도를 초기화했다. 16, 17행에서 보듯이 Car 인스턴스를 생성할 때는 2개의 파라미터를 넘겨야 하며, 파라미터 없이 Car 인스턴스를 생성하면 오류가 발생한다.

2 메소드 오버로딩

메소드 오버로딩(overloading)은 같은 클래스 내에서 메소드의 이름이 같아도 파라미터의 개수나 데이터 형식만 다르면 여러 개를 선언할 수 있는 것을 말한다. 생성자도 메소드의 일종이므로 메소드 오버로딩을 할 수 있다. 즉 Car 클래스에 다음과 같이 여러 개의 생성자를 만들어 놓을 수 있다.

```
Car() {
}

Car(String color) {
  this.color = color;
}

Car(String color, int speed) {
  this.color = color;
  this.speed = speed;
}
```

이제는 인스턴스를 생성할 때 필요한 생성자를 호출해서 생성하면 된다. 3개의 생성자를 차례로 사용하려면 다음과 같이 한다.

```
Car myCar1 = new Car();
Car myCar2 = new Car("빨강");
Car myCar3 = new Car("파랑", 30);
```

전체 코드를 통해 이해해보자.

실습 11-8 메소드 오버로딩 1

```
01 class Car {
02    private String color;
03    private int speed;
04
05    Car() {
06    }
07
08    Car(String color) {
09       this.color = color;
10    }
11
12    Car(String color, int speed) {
13       this.color = color;
14       this.speed = speed;
15    }
16
17    // [실습 11-6]의 getColor(), getSpeed() 메소드와 동일
18    }
19 }
20
21 public class Ex11_08 {
22    public static void main(String[] args) {
23       Car myCar1 = new Car();
24       Car myCar2 = new Car("빨강");
25       Car myCar3 = new Car("파랑", 30);
26
```

파라미터 개수가 다른, 동일한 이름의 생성자 3개를 만든다.

인스턴스를 만들 때 모두 다른 생성자를 호출한다.

```
27      System.out.println("자동차1의 색상은 " + myCar1.getColor() + "이며, 현재속도는 " +
        myCar1.getSpeed() + "km 입니다.");
28      System.out.println("자동차2의 색상은 " + myCar2.getColor() + "이며, 현재속도는 " +
        myCar2.getSpeed() + "km 입니다.");
29      System.out.println("자동차3의 색상은 " + myCar3.getColor() + "이며, 현재속도는 " +
        myCar3.getSpeed() + "km 입니다.");
30    }
31 }
```

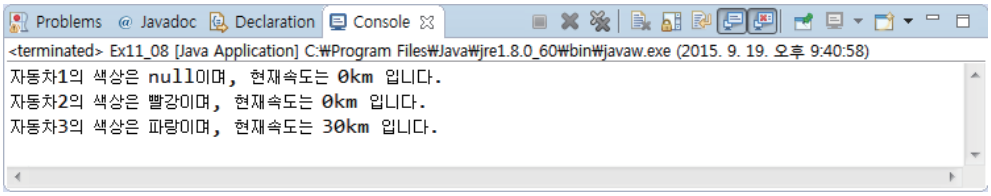


그림 11-13 실행 결과

27행의 myCar1이 출력된 결과를 살펴보면 String 데이터 형식은 별도로 초기화하지 않으면 null(널) 값이 들어가고, int형은 0 값으로 별도로 초기화하지 않아도 0 값이 들어가는 것을 확인할 수 있다.

메/멘/토 퀴/즈 동일한 이름의 메소드를 여러 개 만드는 것을 메소드 □□□□이라고 한다.

생성자뿐 아니라 일반 메소드도 오버로딩을 할 수 있다. 다음 실습은 간단한 더하기 메소드인데 파라미터의 데이터 형식에 따라서 다른 메소드가 호출된다.

실습 11-9 메소드 오버로딩 2

```
01 class Calc {
02    void addValue ( 1 ) {
03       System.out.println("double값 계산 ==> " + (v1 + v2));
04    }
05    void addValue ( 2 ) {
06       System.out.println("int값 계산 ==> " + (v1 + v2));
07    }
08 }
09
10 public class Ex11_09 {
```

double형 2개의 파라미터를 받아서 더하는 메소드이다.

int형 2개의 파라미터를 받아서 더하는 메소드이다.

```
11    public static void main(String[] args) {
12       Calc myCalc = new Calc();
13
14       myCalc.addValue(1.0, 1.0);
15       myCalc.addValue(1, 1);
16    }
17 }
```

double형 및 int형 파라미터를 가지고 addValue() 메소드를 호출한다.

정답 1 double v1, double v2 2 int v1, int v2

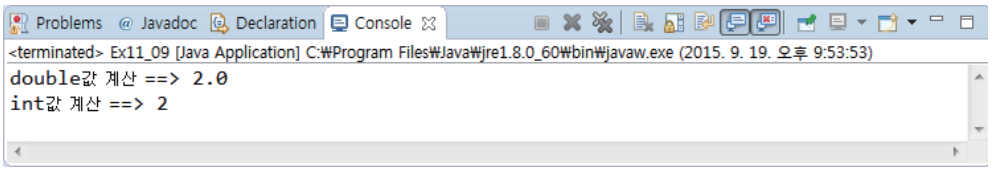


그림 11-14 실행 결과

2행과 5행에 같은 이름의 addValue() 메소드가 있지만 파라미터의 형식이 다르기 때문에 여러 개를 만들 수 있다.

▶ **직접 풀어보기 11-3**

[실습 11-9]를 수정하여 파라미터가 short형 및 float형을 갖는 addValue() 메소드를 추가해보자.

SECTION
03 인스턴스 변수와 클래스 변수

인스턴스 변수는 인스턴스를 생성해야 공간이 할당되고, 클래스 변수는 클래스 자체에 변수의 공간이 할당되어 있다.

1 인스턴스 변수

지금까지 사용한 필드는 모두 인스턴스 변수로, 앞에서 살펴본 Car 클래스의 color나 speed 필드도 인스턴스 변수이다. 인스턴스 변수는 인스턴스를 생성해야 비로소 사용할 수 있는 변수인데, 다음 예를 통해 인스턴스 변수의 개념을 파악해보자.

```
class Car {
  String color;  // 필드 : 인스턴스 변수
  int speed;     // 필드 : 인스턴스 변수
}
```

Car 클래스의 2개 필드는 모두 인스턴스 변수이며, 이 두 필드는 클래스 안에 존재하는 것으로 아직 실제 공간이 할당되지는 않았다. 클래스는 설계도에 해당하므로 설계도의 자동차 색상이나 속도가 실제로 존재하지 않는 것과 같은 개념이다.

그렇다면 설계도(클래스)를 이용하여 main() 메소드에서 자동차(인스턴스)를 만들어보자.

```
Car myCar1 = new Car();
Car myCar2 = new Car();
```

이제 myCar1은 실체가 있는 자동차가 되고 myCar1 안에 color와 speed의 공간이 만들어진다. 마찬가지로 myCar2도 별도로 color와 speed의 공간이 만들어진다.

[그림 11-15]를 보면 클래스(설계도)에는 실제 인스턴스 변수의 공간이 할당되어 있지 않으며, 인스턴스(자동차)로 만들어져야 인스턴스 변수에 공간이 할당되고 myCar1.color 등으로 사용이 가능해진다. [실습 11-1]부터 [실습 11-9]까지 모두 이런 방식으로 사용한 것이다.

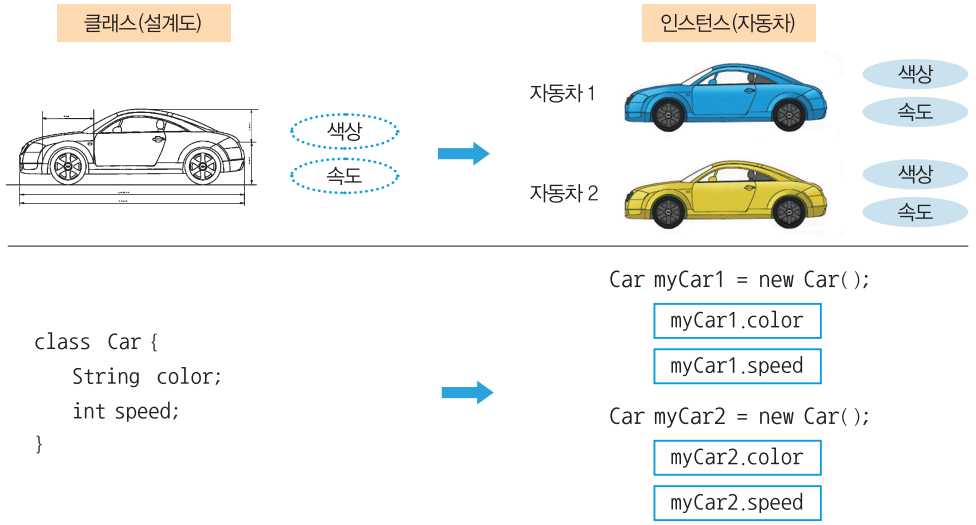


그림 11-15 인스턴스 변수의 개념

2 클래스 변수

클래스 변수는 클래스 안에 공간이 할당된 변수를 말한다. 그래서 클래스 변수는 인스턴스에는 별도의 공간이 할당되지 않고 여러 인스턴스가 클래스 변수의 공간을 같이 사용한다. 다음 그림을 살펴보자.

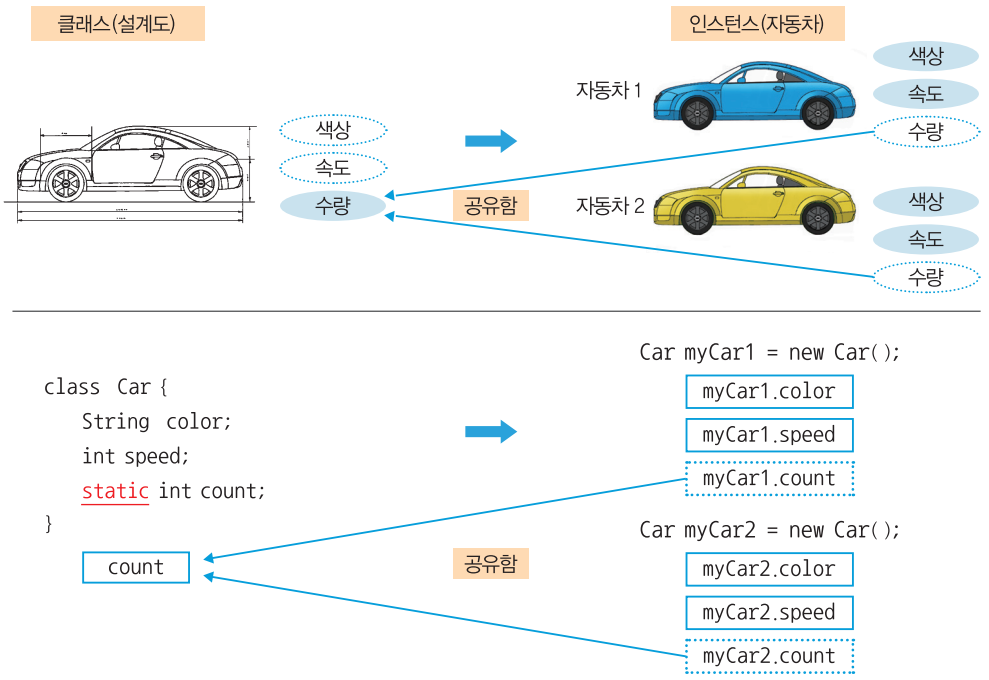


그림 11-16 클래스 변수의 개념

클래스 변수를 만들기 위해서는 필드 앞에 'static' 키워드를 붙이기만 하면 된다. 그러면 그 필드는 클래스 자체에 공간이 생기며, 인스턴스를 생성해도 추가로 공간을 할당하지 않고 클래스에 이미 생성된 공간을 공유한다.

클래스 변수에 접근하려면 다른 인스턴스 변수처럼 myCar1.count로 접근한다. 하지만 인스턴스 내부의 공간이 아닌 클래스의 공간을 사용하므로 myCar1.count와 myCar2.count는 동일한 값을 갖는다.

메/멘/토 퀴/즈 클래스 변수 앞에는 □□□□□□ 키워드를 붙인다.

실습을 통해 이를 확인해보자. [실습 11-10]은 자동차가 몇 대 생산되었는지를 확인하는 프로그램이다.

실습 11-10 클래스 변수 활용 예

```
01 class Car {
02    String color;
03    int speed;
04    static int count = 0;
05
06    Car() {
07       count++;
08    }
09 }
10
11 public class Ex11_10 {
12    public static void main(String[] args) {
13       Car myCar1 = new Car();
14       System.out.println("현재 생산된 자동차 숫자 ==> " + myCar1.count);
15
16       Car myCar2 = new Car();
17       System.out.println("현재 생산된 자동차 숫자 ==> " + myCar2.count);
18
19       Car myCar3 = new Car();
20       System.out.println("현재 생산된 자동차 숫자 ==> " + Car.count);
21    }
22 }
```

04행: 클래스 변수를 선언하고 0으로 초기화한다. count는 생산된 자동차의 총대수를 나타낸다.

06~08행: 생성자에서 count를 1씩 증가시킨다.

13~20행: 자동차 인스턴스를 1개씩 만들고, 현재 생산된 자동차의 총대수를 출력한다.

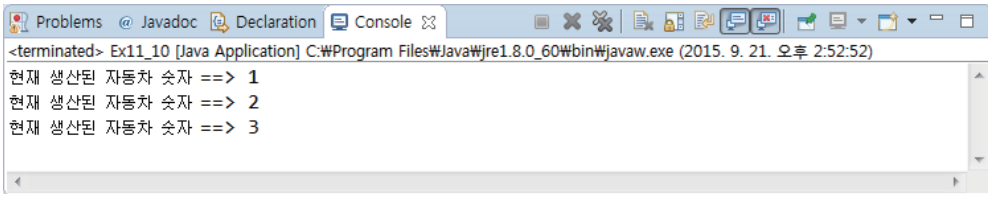


그림 11-17 실행 결과

4행에서 클래스 변수 count를 선언하면 이 변수는 클래스 자체에 공간이 할당된다. 그리고 13, 16, 19행에서 인스턴스를 생성해도 count는 인스턴스에 속하지 않고 클래스에 남아 있는다. 13, 16, 19행에서 인스턴스를 생성할 때 6~8행의 생성자가 호출되며, 생성자에는 count(생산된 자동차의 총대수)를 1씩 증가시킨다.

14행과 17행에서 '인스턴스이름.count'로 클래스 변수에 접근하여 현재의 총대수를 출력했다. 관심 있게 볼 부분은 20행의 '클래스이름.count'이다. 즉 클래스 변수 count에 접근하기 위해 '인스턴스이름.count' 또는 '클래스이름.count' 모두 사용할 수 있다.

▶ 직접 풀어보기 11-4

[실습 11-10]에 클래스 변수 CAR_TYPE="승용차"를 추가하고 main() 메소드에서 출력해보자.

3 인스턴스 메소드와 클래스 메소드

필드가 인스턴스 변수와 클래스 변수로 나누어지듯이 메소드도 인스턴스 메소드와 클래스 메소드로 나누어진다. 인스턴스 메소드는 인스턴스를 먼저 생성한 다음 '인스턴스이름.메소드이름()' 방식으로 호출하는 것을 말하며, 지금까지 사용한 메소드는 모두 인스턴스 메소드였다.

클래스 메소드는 메소드 이름 앞에 'static' 키워드를 붙이면 된다. 클래스 메소드는 인스턴스를 생성하지 않고 메소드를 사용하는 데 주로 쓰인다. 즉 인스턴스 메소드는 인스턴스를 생성한 이후에야 호출이 가능하지만, 클래스 메소드는 인스턴스를 생성하기 전에도 '클래스이름.메소드이름()' 형식으로 사용할 수 있다. 실습을 통해 클래스 메소드의 용도를 살펴보자.

실습 11-11 클래스 메소드 활용 예

```
01 class Car {
02    String color;
03    int speed;
04    static [1] int count = 0;
```

----- 클래스 변수를 선언하고 0으로 초기화한다. 직접 접근하지 못하도록 private을 붙인다.

```
05
06    Car() {
07       count++;
08    }
09
10       2    int getCount() {
11       return count;
12    }
13 }
14
15 public class Ex11_11 {
16    public static void main(String[] args) {
17
18       System.out.println("현재 생산된 자동차 숫자 ==> " + Car.getCount());
19
20       Car myCar1 = new Car();
21       System.out.println("현재 생산된 자동차 숫자 ==> " + myCar1.getCount());
22    }
23 }
```

static 키워드를 붙여서 클래스 메소드를 생성한다.

클래스 이름을 이용하여 클래스 메소드를 호출한다.

인스턴스 이름을 이용하여 클래스 메소드를 호출한다.

정답_ 1 private 2 static

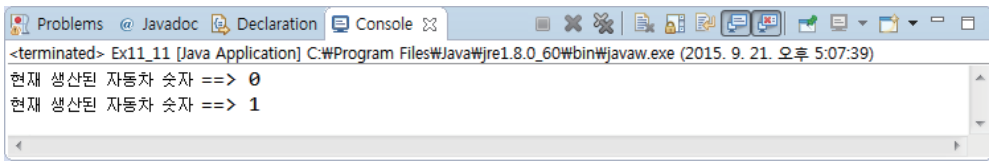


그림 11-18 실행 결과

count에 직접 접근하지 못하도록 4행에서 private을 붙였다. 이제 count는 6행의 생성자와 10행의 getCount() 메소드에서만 접근이 가능하다. 10행에서는 static 키워드를 붙여서 getCount() 메소드를 클래스 메소드로 지정했다. 그러므로 인스턴스가 생성되기 전에도 18행에서 getCount() 메소드를 '클래스이름.메소드이름()' 형식으로 호출할 수 있게 되었다. 21행에서는 클래스 메소드를 '인스턴스이름.메소드이름()'으로도 호출이 가능하다는 것을 확인했다.

메/멘/토 퀴/즈 클래스 메소드는 클래스 이름 앞에 □□□□□□ 키워드를 붙인다.

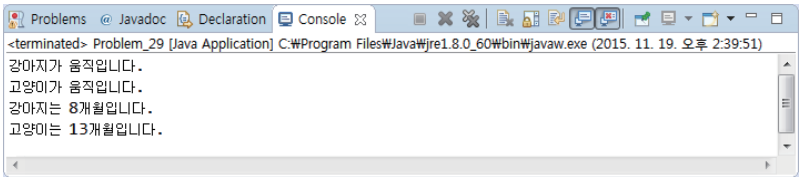
예제 모음 29

클래스의 기본

난이도 ★☆☆

예제 설명 애완동물(Pet) 클래스를 만들어보자.

실행 결과



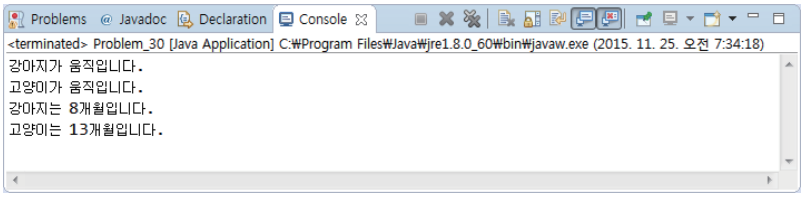
예제 모음 30

클래스의 기본–접근 제어 수식어 활용

난이도 ★★☆

예제 설명 애완동물(Pet) 클래스를 만들어보자. 접근 제어 수식어를 활용하여 속성은 외부에서 접근할 수 없도록 하고 메소드에서만 속성에 접근하게 한다.

실행 결과



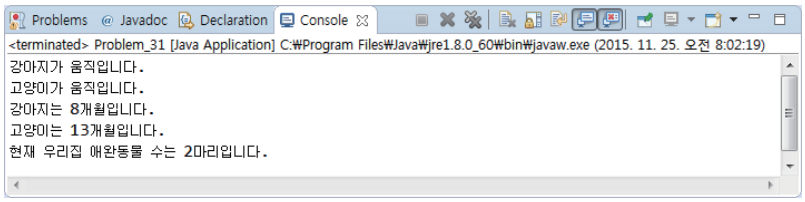
예제 모음 31

클래스의 기본–생성자, 클래스 변수, 클래스 메소드 활용

난이도 ★★★

예제 설명 애완동물(Pet) 클래스를 만들어보자. 초기에 생성자에서 속성 값을 설정하는 방법을 사용한다.

실행 결과



예제 모음 코드 설명

29

```
01 class Pet {
02    String type; // 애완동물 종류
03    int age; // 애완동물 개월 수
04
05    void move() {
06       System.out.println(this.type + "가 움직입니다.");
07    }
08
09    int getAge() {
10       return this.age;
11    }
12 }
13
14 public class Problem_29 {
15
16    public static void main(String[] args) {
17       Pet pet1 = new Pet();
18       pet1.type = "강아지";
19       pet1.age = 8;
20
21       Pet pet2 = new Pet();
22       pet2.type = "고양이";
23       pet2.age = 13;
24
25       pet1.move();
26       pet2.move();
27
28       System.out.println(pet1.type + "는 " + pet1.age + "개월입니다.");
29       System.out.println(pet2.type + "는 " + pet2.age + "개월입니다.");
30    }
31 }
```

- 인스턴스 변수로 애완동물의 종류와 개월 수를 선언한다.
- 애완동물의 움직임을 설정하는 메소드를 정의한다.
- 애완동물의 개월 수를 반환하는 메소드를 정의한다.
- 애완동물 클래스를 정의한다.
- 애완동물 인스턴스 1을 생성하고 종류에 '강아지', 개월 수에 '8'을 대입한다.
- 애완동물 인스턴스 2를 생성하고 종류에 '고양이', 개월 수에 '13'을 대입한다.
- 메소드를 호출한다.
- 인스턴스 변수의 내용을 출력한다.

30

```
01 class Pet {
02    private String type;
03    private int age;
04
05    public void move() {
06       System.out.println(this.type + "가 움직입니다.");
07    }
08
09    public void setType(String type) {
10       this.type = type;
11    }
12
13    public void setAge(int age) {
14       this.age = age;
15    }
16
17    public String getType() {
18       return this.type;
19    }
20
21    public int getAge() {
22       return this.age;
23    }
24 }
25
26 public class Problem_30 {
27
28    public static void main(String[] args) {
29       Pet pet1 = new Pet();
30       pet1.setType("강아지");
31       pet1.setAge(8);
32
33       Pet pet2 = new Pet();
34       pet2.setType("고양이");
35       pet2.setAge(13);
36
37       pet1.move();
38       pet2.move();
```

인스턴스 변수로 애완동물의 종류와 개월 수를 선언한다.

애완동물의 움직임을 설정하는 메소드를 정의한다.

애완동물의 종류를 설정하는 메소드를 정의한다.

애완동물의 개월 수를 설정하는 메소드를 정의한다.

애완동물의 종류를 반환하는 메소드를 정의한다.

애완동물의 개월 수를 반환하는 메소드를 정의한다.

애완동물 클래스를 정의한다.

애완동물 인스턴스 1을 생성하고 종류에 '강아지', 개월 수에 '8'을 대입한다.

애완동물 인스턴스 2를 생성하고 종류에 '고양이', 개월 수에 '13'을 대입한다.

애완동물의 움직임을 설정하는 메소드를 호출한다.

```
39
40      System.out.println(pet1.getType() + "는 " + pet1.getAge() + "개월입니다.");
41      System.out.println(pet2.getType() + "는 " + pet2.getAge() + "개월입니다.");
42   }
43 }
```

인스턴스 변수의 내용을 출력한다(메소드를 통해 접근).

31

```
01 class Pet {
02    private String type; // 애완동물 종류
03    private int age; // 애완동물 개월수
04    static int count = 0; // 애완동물 수
05
06    Pet(String type, int age) {
07       this.type = type;
08       this.age = age;
09    }
10
11    static int getCount() {
12       return count;
13    }
14
15    public void move() {
16       System.out.println(this.type + "가 움직입니다.");
17    }
18
19    public String getType() {
20       return this.type;
21    }
22
23    public int getAge() {
24       return this.age;
25    }
26 }
27
28 public class Problem_31 {
```

인스턴스 변수로 애완동물의 종류와 개월 수를 선언한다.

클래스 변수로 애완동물의 수를 선언한다.

애완동물의 종류와 개월 수를 설정하는 생성자를 정의한다.

애완동물의 수를 출력하는 클래스 메소드를 정의한다.

애완동물의 움직임을 설정하는 인스턴스 메소드를 정의한다.

애완동물의 종류를 반환하는 인스턴스 메소드를 정의한다.

애완동물의 개월 수를 반환하는 메소드를 정의한다.

애완동물 클래스를 정의한다.

```
29
30    public static void main(String[] args) {
31        Pet pet1 = new Pet("강아지", 8);
32        Pet.count++;
33        Pet pet2 = new Pet("고양이", 13);
34        Pet.count++;
35
36        pet1.move();
37        pet2.move();
38
39        System.out.println(pet1.getType() + "는 " + pet1.getAge() + "개월입니다.");
40        System.out.println(pet2.getType() + "는 " + pet2.getAge() + "개월입니다.");
41        System.out.println("현재 우리집 애완동물 수는 " + Pet.getCount() + "마리입니다.");
42    }
43 }
```

31행 ----- 애완동물 인스턴스 1을 생성하고 종류에 '강아지', 개월 수에 '8'을 대입한다.

32행 ----- 애완동물의 수(클래스 변수)를 증가시킨다.

33행 ----- 애완동물 인스턴스 2를 생성하고 종류에 '고양이', 개월 수에 '13'을 대입한다.

34행 ----- 애완동물의 수(클래스 변수)를 증가시킨다.

36~37행 ---- 애완동물의 움직임을 설정하는 메소드를 호출한다.

39~40행 ---- 인스턴스 변수의 내용을 출력한다(인스턴스 메소드를 통해 접근).

41행 ---- 클래스 변수의 내용을 출력한다(클래스 메소드를 통해 접근).

요약

IT COOKBOOK

01 클래스의 기본(자동차의 예)

① 클래스 구현

```
class Car {
   // 자동차의 필드
   String color;
   int speed;
   // 자동차의 메소드
   void upSpeed(int value) {
      speed = speed + value;
   }
}
```

② 인스턴스 생성

```
Car myCar1 = new Car();
Car myCar2 = new Car();
Car myCar3 = new Car();
```

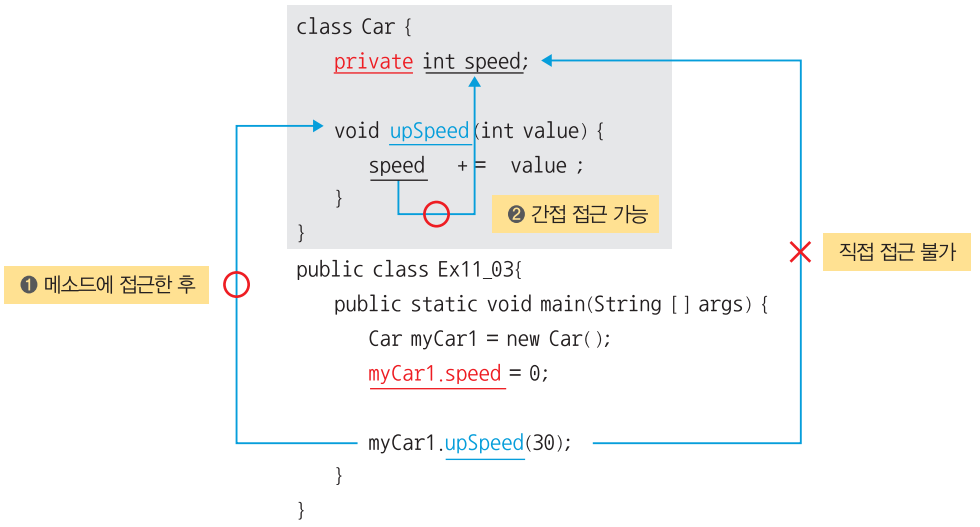
③ 필드에 값 대입

```
myCar1.color = "빨간색";
myCar1.speed = 0;
myCar2.color = "파란색";
myCar2.speed = 0;
myCar3.color = "노란색";
myCar3.speed = 0;
```

④ 메소드 호출

```
myCar1.upSpeed(30);
myCar2.upSpeed(60);
```

02 private 접근 제어 수식어



03 접근 제어 수식어

일반적으로 필드는 private 접근 제어 수식어를, 메소드는 public 접근 제어 수식어를 붙여서 사용한다.

04 생성자(자동차 클래스의 예)

① 기본 형식

```
class Car {
  String color;
  int speed;
  Car( ) {  // 생성자
    color = "빨강";
    speed = 0;
  }
}
```

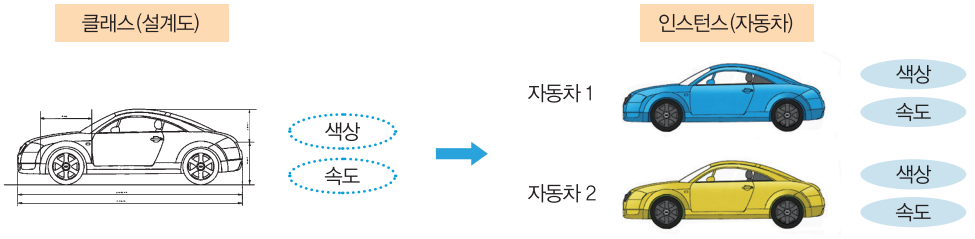
② 생성자는 파라미터를 가질 수 있다.

05 메소드 오버로딩

같은 클래스 내에서 메소드의 이름이 같아도 파라미터의 개수나 데이터 형식만 다르면 여러 개를 선언할 수 있는 것을 말한다. 생성자도 메소드의 일종이므로 메소드 오버로딩을 할 수 있다.

06 변수의 종류

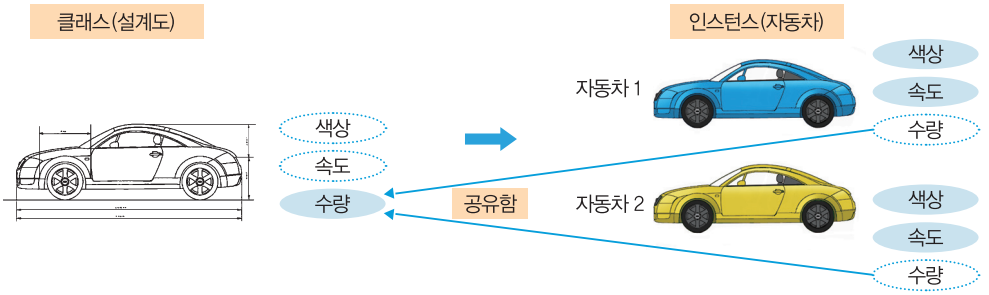
① 인스턴스 변수




```
class Car {
    String color;
    int speed;
}
```

```
Car myCar1 = new Car();
myCar1.color
myCar1.speed
Car myCar2 = new Car();
myCar2.color
myCar2.speed
```

② 클래스 변수




```
class Car {
    String color;
    int speed;
    static int count;
}
```

count

공유함

```
Car myCar1 = new Car();
myCar1.color
myCar1.speed
myCar1.count
Car myCar2 = new Car();
myCar2.color
myCar2.speed
myCar2.count
```

07 메소드의 종류

인스턴스 메소드는 인스턴스를 먼저 생성한 다음 '인스턴스이름.메소드이름()' 방식으로 호출하는 것이며, 클래스 메소드는 메소드 이름 앞에 'static' 키워드를 붙이면 된다. 클래스 메소드는 인스턴스를 생성하지 않고도 메소드를 사용하기 위한 용도로 주로 쓰인다. 즉 인스턴스 메소드는 인스턴스를 생성한 이후에야 호출이 가능하지만, 클래스 메소드는 인스턴스를 생성하기 전에도 '클래스이름.메소드이름()' 형식으로 사용할 수 있다.

연습문제

IT COOKBOOK

01 다음은 자동차 클래스를 구현한 코드이다. 코드를 보고 빈칸에 알맞은 말을 넣으시오.

```
class Car {
   String color;
   int speed;
   void upSpeed(int value) {
      speed = speed + value;
   }
}
```

① 속성을 나타내는 color, speed 등을 [ ]라 하고, 기능을 나타내는 upSpeed(), downSpeed()를 [ ]라 한다.

② 다음은 자동차 인스턴스를 생성하는 코드이다.

```
Car thisCar = [ ] ;
```

③ 다음은 앞에서 생성한 인스턴스의 필드에 값을 대입하는 코드이다.

```
[ ] = "RED";
[ ] = 100;
```

④ 다음은 앞에서 생성한 자동차의 속도를 50km 가속하는 코드이다.

```
[ ](50) ;
```

02 클래스의 생성과 사용 단계를 차례대로 나열하시오.

① 인스턴스의 생성

② 필드, 메소드의 활용

③ 클래스 선언

03 다음 4행의 코드를 3행의 코드로 변경하시오(인스턴스 선언과 동시에 객체를 생성한다).

```
Car myCar1, myCar2, myCar3;
myCar1 = new Car( );
myCar2 = new Car( );
myCar3 = new Car( );
```

04 다음 코드는 오류가 발생한다. 오류가 발생하는 행 번호를 밝히고 그 이유를 간단히 설명하시오.

```
01 class  Car {
02    private  int  speed;
03
04    void  upSpeed(int value ) {
05       speed  +=  value ;
06    }
07 }
08
09 public class Exam{
10    public static void main(String[] args) {
11       Car myCar1 = new Car( );
12       myCar1.speed = 0;
13
14       myCar1.upSpeed(30);
15    }
16 }
```

05 다음은 인스턴스 변수에 파라미터로 받은 값을 대입하는 내용이다. 빈칸에 알맞은 말을 넣으시오.

```
class  Book {
  private  Integer  page;

  void setPage(Integer  page) {
     [          ] = page;
  }
}
```

06 다음 접근 제어 수식어 중에서 서브 클래스에서 접근이 가능한 것을 2개 고르시오.

① public ② protected ③ default ④ private

07 다음은 생성자가 포함된 클래스이다. 틀린 부분을 찾고 그 이유를 간단히 설명하시오.

```
class Book {
  String color;
  int Book( ) {
    color = "빨강";
    return 0;
  }
}
```

08 생성자에 대한 설명 중 틀린 것을 고르시오.

① 생성자는 여러 개 있어도 된다. 단, 파라미터의 개수나 데이터 형식이 달라야 한다.

② 생성자는 return 문을 포함할 수 없다.

③ 생성자는 직접 호출할 수 없다.

④ 생성자는 메소드 오버로딩을 구현할 수 없다.

09 다음 빈칸에 알맞은 말을 넣으시오.

① [ ]은 같은 클래스 내에서 메소드의 이름이 같아도 파라미터의 개수나 데이터 형식만 다르면 여러 개를 선언할 수 있는 것을 말한다.

② [ ]는 인스턴스를 생성해야 비로소 사용할 수 있는 변수이고, [ ]는 클래스 안에 공간이 할당된 변수를 말한다. 그래서 클래스 변수는 인스턴스에는 별도의 공간이 할당되지 않고 여러 인스턴스가 클래스 변수의 공간을 같이 사용한다.

10 다음 중 인스턴스를 생성하지 않아도 접근이 가능한 것을 고르시오.

① 클래스 변수 ② 클래스 메소드 ③ 인스턴스 변수 ④ 인스턴스 메소드