

# 루비페이퍼 조판 가이드북

VER 0.1

## 더코딩

꼭 필요한 것만 골라 배우는 인공지능 맞춤 수학

초판 1쇄 발행 2018년 11월 1일 / 2쇄 2018년 12월 20일

지은이 채규택 표선영

옮긴이 김갑자

편집 한창훈 / 교정교열 한창훈

디자인 이대범 / 표지일러스트 한창훈 / 디자인 이대범

펴낸이 한창훈

펴낸곳 루비페이퍼 / 등록 2013년 11월 6일(제 385-2013-000053 호)

주소 경기도 부천시 원미구 길주로 252 603호

전화 032-322-6754 / 팩스 031-8039-4526

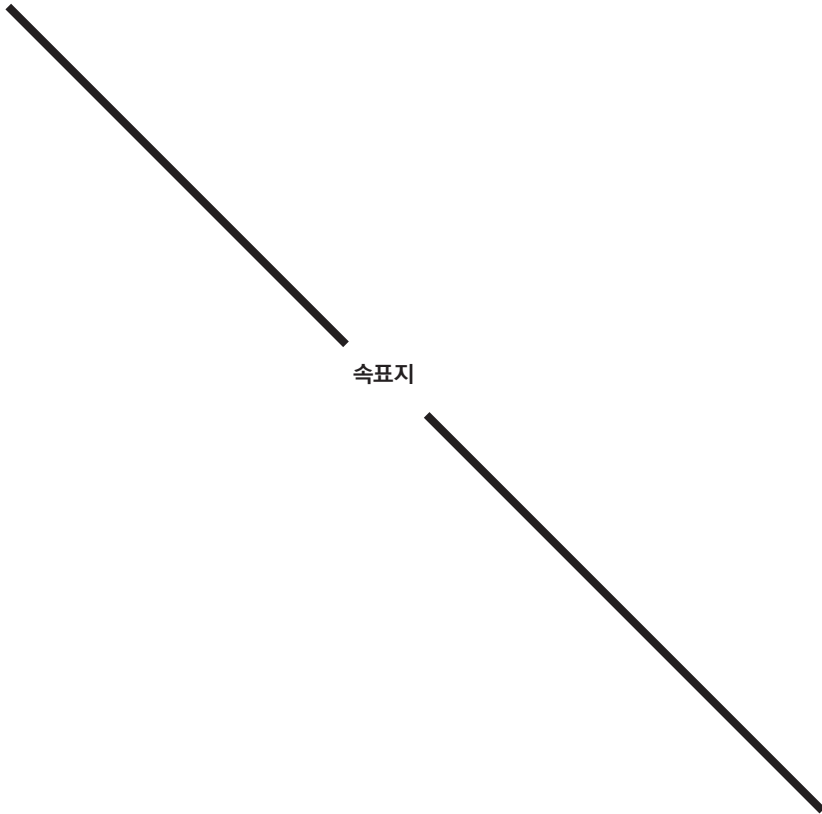
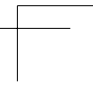
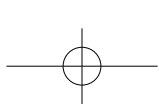
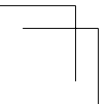
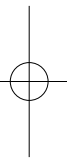
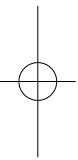
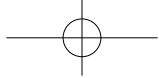
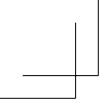
홈페이지 [www.RubyPaper.co.kr](http://www.RubyPaper.co.kr) / 인스타그램

ISBN 979-11-86710-37-1

- \* 이 책은 저작권법에 따라 보호받는 저작물이므로 무단 전재와 무단 복제를 금하며,  
이 책 내용의 전부 또는 일부를 이용하려면 저작권자와 루비페이퍼의 서면 동의를 받아야 합니다.
- \* 책값은 뒤표지에 있습니다.
- \* 잘못된 책은 구입처에서 교환해 드리며, 관련 법령에 따라서 환불해 드립니다.  
단 제품 훼손 시 환불이 불가능합니다.

〈스프링 부트〉는 ○○○의 ○○○가 루비페이퍼와 ○○○협업하여 ○○○을  
책으로 만들었습니다.

이 책의국립중앙도서관 출판예정도서목록(CIP)는서지정보유통지원시스템 홈페이지([seoji.nl.go.kr](http://seoji.nl.go.kr))와  
국가자료공동목록시스템([www.nl.go.kr/kolisnet](http://www.nl.go.kr/kolisnet))에서 이용하실 수 있습니다.  
CIP제어번호 : CIP2019123456



속표지

KOREKARA DATA BUNSEKI WO HAJIMETAHITO NO TAMENO HON

©TAKUYA KUDO 2013

Originally published in Japan in 2013 by PHP Institute, Inc., TOKYO,

Korean translation rights arranged with PHP Institute, Inc., TOKYO,

through TOHAN CORPORATION, TOKYO, and Danny Hong Agency, SEOUL..

Korean translation copyright ©2014 by RubyPaper

이 책의 한국어판 저작권은 대니홍 에이전시를 통해 저작권자와 독점 계약한  
루비페이퍼에 있습니다. 저작권법에 의하여 한국 내에서 보호를 받는 저작물이므로  
무단 전재와 무단 복제를 금합니다.

## 추천사

스프링 프레임워크가 뛰어난 것은 다른 프레임워크들과 경쟁이 아닌 통합을 지향하기 때문이다. 하지만 역설적이게도 스프링이 다양한 프레임워크와 기술들을 지원하면서 라이브러리와 설정도 복잡해졌다. 따라서 스프링을 처음 시작하는 사람들은 스프링의 복잡하고 방대한 설정에 대해 부담을 느끼게 되었으며, 복잡한 설정에서 발생한 오류를 수정하는 데에 점점 더 많은 시간과 노력이 필요하게 되었다. 스프링에서는 이런 문제들을 해결하기 위해 스프링 부트를 만들게 된 것이다.

라크랜드 이사 김춘식

스프링 프레임워크가 뛰어난 것은 다른 프레임워크들과 경쟁이 아닌 통합을 지향하기 때문이다. 하지만 역설적이게도 스프링이 다양한 프레임워크와 기술들을 지원하면서 라이브러리와 설정도 복잡해졌다.

월미도 횃집 대표 김봉식

스프링 프레임워크가 뛰어난 것은 다른 프레임워크들과 경쟁이 아닌 통합을 지향하기 때문이다. 하지만 역설적이게도 스프링이 다양한 프레임워크와 기술들을 지원하면서 라이브러리와 설정도 복잡해졌다.

월미도 횃집 대표 김봉식

## 감사의글

이 책을 완성하기 까지는 여러 사람들의 도움이 있었다. 우선 부족한 나에게 흔쾌히 출판을 제안해주고 직접 편집까지 책임져준 루비페이퍼의 한창훈 대표께 진심으로 감사 드린다. 그리고 17년 동안 IT 분야에서 강사의 길을 걷는 동안 고락을 같이하며 조언을 아끼지 않았던 멀티캠퍼스 전임 강사실의 여러 동료들에게도 부족하나마 이 책을 빌어 감사의 말을 전하고 싶다.

마지막으로 이 책을 집필하는 동안 아낌없는 응원과 격려를 보내준 아내 영옥과 주말에도 놀아달라고 조르지 않고 쿨하게 사무실로 보내준 아들 준석에게 아빠가 정말로 사랑한다고 말해주고 싶다.

2019년 5월 9일

채규태



Chae Kyoutae

스프링은 EJB(Enterprise Java Beans)의 무겁고 복잡한 플랫폼에서 벗어나 POJO(Plain Old Java Object)를 기반의 경량화된 개발환경을 제공하는 오픈소스 프레임워크다. 필자가 처음 스프링 프레임워크를 접했던 2007년만 해도 스프링은 애플리케이션 운용에 필요한 객체를 생성하고, 의존성 주입을 처리하는 단순한 컨테이너 기능만 제공했었다. 하지만 오늘날의 스프링은 SNS부터 빅데이터에 이르기까지 엔터프라이즈 시스템 개발에 필요한 모든 분야를 포괄하는 거대한 플랫폼이 되었다.

스프링 프레임워크가 뛰어난 것은 다른 프레임워크들과 경쟁이 아닌 통합을 지향하기 때문이다. 하지만 역설적이게도 스프링이 다양한 프레임워크와 기술들을 지원하면서 라이브러리와 설정도 복잡해졌다. 따라서 스프링을 처음 시작하는 사람들은 스프링의 복잡하고 방대한 설정에 대해 부담을 느끼게 되었으며, 복잡한 설정에서 발생한 오류를 수정하는 데에 점점 더 많은 시간과 노력이 필요하게 되었다. 스프링에서는 이런 문제들을 해결하기 위해 스프링 부트를 만들게 된 것이다.

스프링 부트는 스프링 프레임워크의 서브 프로젝트로 만들어졌다. 따라서 스프링 부트의 개념과 동작 원리를 정확하게 이해하기 위해서는 스프링에 대한 학습이 전제되어야 한다. 물론 스프링에 대한 학습이 되어있지 않다 하더라도 교재의 실습 코드를 타이핑하고 대략적인 내용을 이해하는 데는 크게 어려움이 없을 수도 있다. 하지만 가급적이면 스프링 문법을 먼저 학습하고 이 책을 보는 것을 권고한다. 특히 스프링 부트는 XML 기반의 설정이 아닌 어노테이션 기반의 설정을 지원하기 때문에 다양한 어노테이션의 사용법과 의미를 이해해야 한다.

이 책의 전체 소스는 하나의 게시판 프로그램을 완성하는 것을 목표로 한다. 게시판은 누구나 쉽게 이해하는 비즈니스지만 실제 대부분의 웹애플리케이션이 게시판 프로그램의 기능에서 크게 벗어나지 않는다는 점에서 가장 좋은 예라 생각했기 때문이다. 따라서 이 책을 참고하여 게시판의 기본 기능을 완성하고 댓글이나 페이징 처리 같은 기능들을 확장해나가면서 응용능력을 키우기 바란다.

그리고 이 책의 소스는 디자인과 관련된 부분을 최대한 배제했다. 물론 제이쿼리(jQuery)나 부트스트랩(Bootstrap)을 이용하여 빠르고 쉽게 원하는 화면을 만들 수도 있다. 특히 부트스트랩을 사용하면 모바일 기기를 고려한 반응형 웹 페이지를 만들 수도 있다. 하지만 이 책에서는 UI 화면에서 HTML 태그 외에는 사용하지 않음으로써 스프링 부트 기능 구현에 집중한다. 그리고 책의 소스를 타이핑할 때는 설명을 위한 소스와 실제 타이핑해야 하는 소스를 구분하여 표시했으므로 설명을 위한 소스를 타이핑하여 오류가 발생하지 않도록 주의하기 바란다.

이 책의 예제 소스는 다음 경로에서 내려받습니다.

 <https://github.com/username/myproject.git>

 <https://www.rubypaper.co.kr/category/자료실>



**PART I**  
**스프링 부트 기초**

011

Chapter 01	<b>스프링 부트 시작하기</b>	<b>11</b>
<b>1</b>	<b>스프링 부트(Spring Boot)의 등장</b>	<b>11</b>
— 1.1	스프링 프레임워크(Spring Framework)	123
— 1.1.1	스프링 부트 프로젝트 만들기	123
— 1.1.2	스프링 부트 프로젝트 구조 및 실행	999
— 1.1.3	스프링 부트 프로젝트 둘러보기	1
— 1.1.4	웹애플리케이션 작성하기	12
— 2.1	스프링 부트(Spring Boot)의 등장	23
— 2.1.1	스프링 부트 프로젝트 만들기	688
— 2.1.2	스프링 부트 프로젝트 구조 및 실행	788
— 2.1.3	스프링 부트 프로젝트 둘러보기	785
— 2.1.4	웹애플리케이션 작성하기	78
<b>2</b>	<b>실습 환경 설정</b>	<b>363</b>
— 2.1	JDK 설치하기	54
— 2.2	이클립스 설치하기	21
— 2.3	STS(Spring Tool Suite) 설치하기	21
<b>3</b>	<b>스프링 부트 Quick Start</b>	<b>25</b>
— 3.1	스프링 부트 프로젝트 만들기	33
— 3.2	스프링 부트 프로젝트 구조 및 실행	11
— 3.3	스프링 부트 프로젝트 둘러보기	258
— 3.4	웹애플리케이션 작성하기	954

**PART II**  
**스프링 부트 기초의 기초의**  
**기초**

011

**Chapter 02 | 의존성 관리와 자동 설정의** 211

**2.1 스프링 부트의 의존성 관리** 875

- 1.1 스타터로 의존성 관리하기 568
- 2.1 의존성 재정의하기 741

**2.2 스프링 부트의 자동 설정** 369

- 1.1 자동 설정 이해하기 785
- 2.1 사용자 정의 스타터와 자동 설정 만들기 968
- 3.1 자동 설정 재정의하기 599

**Chapter 03 | 테스트와 로깅, 빌드**

**3.1 스프링 부트 테스트** 12

- 1.1 스프링 부트에서 테스트하기 12
- 2.2 MockMvc 이용해서 컨트롤러 테스트하기 12
- 3.3 서비스 계층을 연동하는 컨트롤러 테스트하기 12

**3.2 스프링 부트 로깅(LOGGING)** 12

- 1. 스프링 부트 로깅 이해하기 696
- 2. 스프링 부트 로깅 수정하기 45

**3.3 독립적으로 실행 가능한 JAR** 898

- 1. 스프링 부트 빌드 이해하기 156
- 2. Runnable JAR 실행하기 121

Part

I

⋮

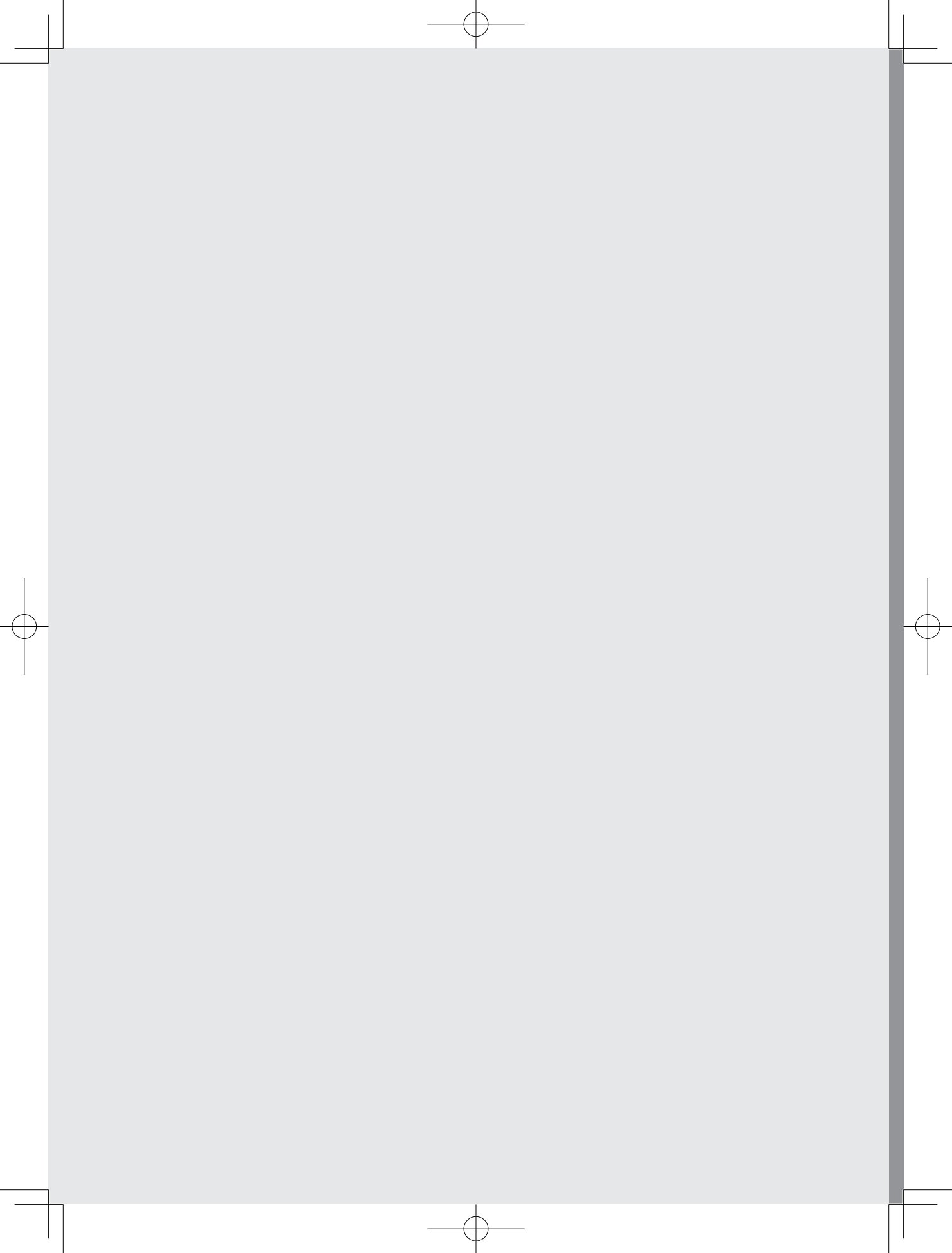
## 파트의 도입부와 제목

⋮

소스 구문이란 메서드 사용 예나 if문 while문 등의 정의를 지정한 것입니다. 보통 다음과 같이 한 줄이나 두 줄 정도 정의합니다. 코드 폰트보다 두껍게 사용하는 경우가 많습니다.

기초 문법에는 한글에도 볼드를 주는 경우가 많습니다.

하지만 먼저 JSP를 이용해서 간단하게 화면을 구성하고, 이후에 JSP 기반의 화면을 타임리프로 변경한다. JSP를 먼저 적용하는 가장 큰 이유는 JSP는 자바의 표준이며, 여전히 JSP로 화면을 제공하는 시스템들도 존재하기 때문이다.



# 01

## 스프링 부트 시큐리티 시큐리티

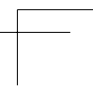
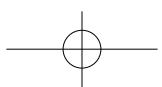
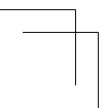
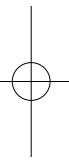
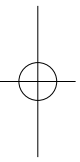
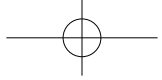
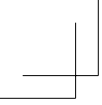
소스 구문이란 메서드 사용 예나 if문 while 문 등의 정의를 지정한 것입니다. 보통 다음과 같이 한 줄이나 두 줄 정도 정의합니다. 코드 폰트 보다 두껍게 사용하는 경우가 많습니다.

기초 문법에는 한글에도 볼드를 주는 경우가 많습니다.

우리는 최종적으로는 스프링 부트가 제공하는 템플릿 엔진 중에서 타임리프를 적용할 것이다. 하지만 먼저 JSP를 이용해서 간단하게 화면을 구성하고, 이후에 JSP 기반의 화면을 타임리프로 변경한다. JSP를 먼저 적용하는 가장 큰 이유는 JSP는 자바의 표준이며, 여전히 JSP로 화면을 제공하는 시스템들도 존재하기 때문이다.

우리는 최종적으로는 스프링 부트가 제공하는 템플릿 엔진 중에서 타임리프를 적용할 것

이다. 하지만 먼저 JSP를 이용해서 간단하게 화면을 구성하고, 이후에 JSP 기반의 화면을 타임리프로 변경한다. JSP를 먼저 적용하는 가장 큰 이유는 JSP는 자바의 표준이며, 여전히 JSP로 화면을 제공하는 시스템들도 존재하기 때문이다.



## 01 \*

컴퓨터는 애초에 거대한 연산 기계로 태어났습니다. 사람 수십 명이 장시간에 걸쳐 계산해야 할 것을 단시간에 정확하게 처리하기 위해 만들어졌죠. 그런 만큼, 컴퓨터를 다루기 위한 프로그래밍의 가장 기본 요소가 연산자(Operator)인 것은 당연한 일입니다.

연산자라는 말을 들었을 때 가장 먼저 떠오르는 것은 아마도 사칙 연산일 겁니다. 수학을 조금 더 열심히 배운 사람이라면 제곱이나 미적분, 극한 등을 위한 연산 정도까지 떠올릴지도 모르겠군요.

하지만 프로그래밍에서는 계산을 위한 산술 연산자 뿐만 아니라 훨씬 다양한 역할의 연산자들이 사용됩니다. 값의 크고 작음을 비교하기 위한 비교 연산자, 변수에 값을 대입하는 데 사용하는 대입 연산자, 산술 연산자와 대입 연산자가 합쳐진 증감 연산자, 참 거짓의 집합 연산 결과를 처리하는 논리 연산자 등은 모두 프로그래밍에서 연산자의 범주에 포함되는 것들입니다.

## 1.1 스프링 부트 테스트

가장 먼저 배워볼 것은 대입 연산자입니다. 사실 이 연산자는 여러분이 이미 알고 있을 텐데요, 바로 변수에 값을 대입할 때 사용한 '='입니다. '연산자 오른쪽에 있는 값을 왼쪽의 변수에 대입한다'라는 의미를 가지는 연산자로, 수학 시간에 사용하는 등호(=)와 모양이 같지만 의미가 전혀 다르므로 사용에 주의해야 한다고 설명한 기억이 나는군요.

대입 연산자의 사용방법을 다시 한번 확인해 봅시다. 오른쪽의 값을 왼쪽에 대입하는 것이 원칙입니다.

객체 개념을 지원하지 않는 언어에서는 속성과 기능을 함께 정의할 수 없습니다. 속성은 속성끼리만 묶을 수 있고 기능은 기능끼리만 묶을 수 있을 뿐, 속성은 기능을 포함할 수 없으며 기능 또한 속성을 소유할 수 없습니다. 따라서 이런 류의 언어에서는 항상 속성과 기능을 나누어 정의해야 합니다. 속성은 변수로, 기능은 함수로 말합니다. 반면 객체를 다루는 언어에서는 속성과 기능을 함께 다룰 수 있습니다.

객체를 지원한다 하더라도 각 언어마다 객체를 정의하는 방법은 다릅니다. 자바스크립트에서는 위와 같은 방식으로 중괄호 블록 안에 [속성 : 값], 또는 [함수명 : 내용]의 짝으로 이루어진 데이터를 나열하는 방법을 사용하죠. 예제에서 obj 변수는 객체로 정의되었으며 name, age, height이라는 속성을 가집니다.

대입 연산자의 사용방법을 다시 한번 확인해 봅시다. 오른쪽의 값을 왼쪽에 대입하는 것이 원칙입니다.

객체 개념을 지원하지 않는 언어에서는 속성과 기능을 함께 정의할 수 없습니다. 속성은 속성끼리만 묶을 수 있고 기능은 기능끼리만 묶을 수 있을 뿐, 속성은 기능을 포함할 수 없으며 기능 또한 속성을 소유할 수 없습니다. 따라서 이런 류의 언어에서



는 항상 속성과 기능을 나누어 정의해야 합니다. 속성은 변수로, 기능은 함수로 말합니다. 반면 객체를 다루는 언어에서는 속성과 기능을 함께 다룰 수 있습니다.

## 1.2 산술 연산자

산술 연산자는 대입 연산자와 함께 프로그래밍에서 가장 많이 사용되는 연산자입니다. 값을 더하거나 빼고, 곱하거나 나누고 제곱을 계산하는 등 수학 계산을 처리하는데 사용하죠. 가장 잘 알려진 산술 연산자로는 사칙 연산자가 있습니다. 아마도 다들 익숙할 겁니다. 초등학교 이후로 많이 나열이니까요.

1. 무작위로  $x$  몇 개를 선택
2. 1에서 정한  $x$ 값에 대응하는  $y$ 값을 계산하고, 그중 큰  $y$ 값 몇 개를 선택
3. 2에서 선택한  $y$ 값 근처에 최댓값이 있다고 가정하므로, 해당  $y$ 값에 대응하는  $x$ 값 근처를 탐색해  $x$ 값을 새로 선택
4. commando 정한  $x$ 값에 대응하는  $y$ 값을 계산
5. commando 찾을 때까지 3~4 과정을 반복 실행해  $y$ 값을 계산

산술 연산자는 대입 연산자와 함께 프로그래밍에서 가장 많이 사용되는 연산자입니다. 값을 더하거나 빼고, 곱하거나 나누고 제곱을 계산하는 등 수학 계산을 처리하는데 사용하죠.

가장 잘 알려진 산술 연산자로는 사칙 연산자가 있습니다. 아마도 다들 익숙할 겁니다. 초등학교 이후로 많이 사용해 왔을 테니까요.

1. **라이브러리**: 프로그래밍할 때 자주 사용하는 기능을 모아 미리 만들어 놓은 코드를 뜻합니다.
2. **실행 환경**: 프로그램을 실제로 동작시키는 환경을 뜻합니다. 프로그램을 원활하게 실행한다.
3. **commando**: 프로그래밍할 때 자주 사용하는 기능을 모아 미리 만들어 놓은 코드를 뜻합니다.
4. **Rambo**: 프로그램을 실제로 동작시키는 환경을 뜻합니다. 프로그램을 원활하게 실행한다.

산술 연산자는 대입 연산자와 함께 프로그래밍에서 가장 많이 사용되는 연산자입니다. 값을 더하거나 빼고, 곱하거나 나누고 제곱을 계산하는 등 수학 계산을 처리하는데 사용하죠. 가장 잘 알려진 산술 연산자로는 사칙 연산자가 있습니다. 아마도 다들 익숙할 겁니다. 초등학교 이후로 많이 사용해 왔을 테니까요.

- **초기 개체 생성**: 처음에 x값을 선택
- **선택 연산**: 큰 y값을 많이 선택하는 연산 등으로 큰 y값만 남김
- **교차 연산**: '선택' 과정에서 계산한 y값에 대응하는 x값 근처를 탐색해 새로운 x값을 선택(부모가 되는 x값 2개를 비트열로 삼아 다음 x값을 생성)
- **돌연변이 연산**: 초기 개체 생성에서 선택한 x값과 다른 부분 영역에 해당하는 x값을 선택.

기존 연산이 국숫값에 수렴하는 것을 막는 효과가 있음

산술 연산자는 대입 연산자와 함께 프로그래밍에서 가장 많이 사용되는 연산자입니다. 값을 더하거나 빼고, 곱하거나 나누고 제곱을 계산하는 등 수학 계산을 처리하는데 사용하죠. 가장 잘 알려진 산술 연산자로는 사칙 연산자가 있습니다. 아마도 다들 익숙할 겁니다. 초등학교 이후로 많이 사용해 왔을 테니까요.

### 1.2.1 스프링 부트에서 테스트하기

객체를 지원한다 하더라도 각 언어마다 객체를 정의하는 방법은 다릅니다. 자바스크립트에서는 위와 같은 방식으로 중괄호 블록 안에 [속성 : 값], 또는 [함수명 : 내용]

의 쪽으로 이루어진 데이터를 나열하는 방법을 사용하죠. 예제에서 obj 변수는 객체로 정의되었으며 name, age, height이라는 속성을 가집니다.

대입 연산자의 사용방법을 다시 한번 확인해 봅시다. 오른쪽의 값을 왼쪽에 대입하는 것이 원칙입니다.

객체 개념을 지원하지 않는 언어에서는 속성과 기능을 함께 정의할 수 없습니다. 속성은 속성끼리만 묶을 수 있고 기능은 기능끼리만 묶을 수 있을 뿐, 속성은 기능을 포함할 수 없으며 기능 또한 속성을 소유할 수 없습니다. 따라서 이런 류의 언어에서는 항상 속성과 기능을 나누어 정의해야 합니다. 속성은 변수로, 기능은 함수로 말합니다. 반면 객체를 다루는 언어에서는 속성과 기능을 함께 다룰 수 있습니다.

#### **〔1〕 테스트 환경 만들기**

대입 연산자의 사용방법을 다시 한번 확인해 봅시다. 오른쪽의 값을 왼쪽에 대입하는 것이 원칙입니다.

객체 개념을 지원하지 않는 언어에서는 속성과 기능을 함께 정의할 수 없습니다. 속성은 속성끼리만 묶을 수 있고 기능은 기능끼리만 묶을 수 있을 뿐, 속성은 기능을 포함할 수 없으며 기능 또한 속성을 소유할 수 없습니다. 따라서 이런 류의 언어에서는 항상 속성과 기능을 나누어 정의해야 합니다.

#### **〔2〕 @AutoConfigureMockMvc 사용하기**

대입 연산자의 사용방법을 다시 한번 확인해 봅시다. 오른쪽의 값을 왼쪽에 대입하는 것이 원칙입니다.

### 1.3 산술 연산자

산술 연산자는 대입 연산자와 함께 프로그래밍에서 가장 많이 사용되는 연산자입니다. 값을 더하거나 빼고, 곱하거나 나누고 제곱을 계산하는 등 수학 계산을 처리하는데 사용하죠. 가장 잘 알려진 산술 연산자로는 사칙 연산자가 있습니다. 아마도 다들 익숙할 겁니다. 초등학교 이후로 많이 나열이니깐요.

1. 무작위로 x 몇 개를 선택
2. 1에서 정한 x값에 대응하는 y값을 계산하고, 그중 큰 y값 몇 개를 선택
3. 2에서 선택한 y값 근처에 최댓값이 있다고 가정하므로, 해당 y값에 대응하는 x값 근처를 탐색해 x값을 새로 선택
4. commando 정한 x값에 대응하는 y값을 계산
5. commando 찾을 때까지 3~4 과정을 반복 실행해 y값을 계산

산술 연산자는 대입 연산자와 함께 프로그래밍에서 가장 많이 사용되는 연산자입니다. 값을 더하거나 빼고, 곱하거나 나누고 제곱을 계산하는 등 수학 계산을 처리하는데 사용하죠.

가장 잘 알려진 산술 연산자로는 사칙 연산자가 있습니다. 아마도 다들 익숙할 겁니다. 초등학교 이후로 많이 사용해 왔을 테니까요.

1. **라이브러리**: 프로그래밍할 때 자주 사용하는 기능을 모아 미리 만들어 놓은 코드를 뜻합니다.
2. **실행 환경**: 프로그램을 실제로 동작시키는 환경을 뜻합니다. 프로그램을 원활하게 실행한다.
3. **commando**: 프로그래밍할 때 자주 사용하는 기능을 모아 미리 만들어 놓은 코드를 뜻합니다.
4. **Rambo**: 프로그램을 실제로 동작시키는 환경을 뜻합니다. 프로그램을 원활하게 실행한다.

산술 연산자는 대입 연산자와 함께 프로그래밍에서 가장 많이 사용되는 연산자입니다. 값을 더하거나 빼고, 곱하거나 나누고 제곱을 계산하는 등 수학 계산을 처리하는데 사용하죠. 가장 잘 알려진 산술 연산자로는 사칙 연산자가 있습니다. 아마도 다들

익숙할 겁니다. 초등학교 이후로 많이 사용해 왔을 테니까요.

- **초기 개체 생성:** 처음에 x값을 선택
- **선택 연산:** 큰 y값을 많이 선택하는 연산 등으로 큰 y값만 남김
- **교차 연산:** '선택' 과정에서 계산한 y값에 대응하는 x값 근처를 탐색해 새로운 x값을 선택(부모가 되는 x값 2개를 비트열로 삼아 다음 x값을 생성)
- **돌연변이 연산:** 초기 개체 생성에서 선택한 x값과 다른 부분 영역에 해당하는 x값을 선택.

기존 연산이 국숫값에 수렴하는 것을 막는 효과가 있음

산술 연산자는 대입 연산자와 함께 프로그래밍에서 가장 많이 사용되는 연산자입니다. 값을 더하거나 빼고, 곱하거나 나누고 제곱을 계산하는 등 수학 계산을 처리하는데 사용하죠. 가장 잘 알려진 산술 연산자로는 사칙 연산자가 있습니다. 아마도 다들 익숙할 겁니다. 초등학교 이후로 많이 사용해 왔을 테니까요.

- **라이브러리:** 프로그래밍할 때 자주 사용하는 기능을 모아 미리 만들어 놓은 코드를 뜻합니다.
- **실행 환경:** 프로그램을 실제로 동작시키는 환경을 뜻합니다. 프로그램을 원활하게 실행한다.
- **commando:** 프로그래밍할 때 자주 사용하는 기능을 모아 미리 만들어 놓은 코드를 뜻합니다.
- **Rambo:** 프로그램을 실제로 동작시키는 환경을 뜻합니다. 프로그램을 원활하게 실행한다.

### 1.3.1 소스 코드

객체를 지원한다 하더라도 각 언어마다 객체를 정의하는 방법은 다릅니다. 자바스크립트에서는 위와 같은 방식으로 중괄호 블록 안에 [속성 : 값], 또는 [함수명 : 내용]의 짝으로 이루어진 데이터를 나열하는 방법을 사용하죠. 예제에서 obj 변수는 객체로 정의되었으며 name, age, height이라는 속성을 가집니다.

#### BoardServiceImpl.java

```
package com.rubypaper.service;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import org.springframework.stereotype.Service;

import com.rubypaper.domain.BoardVO;

@Service
public class BoardServiceImpl implements BoardService {

    public String hello(String name) {
        return "Hello : " + name;
    }

    public BoardVO getBoard() {
        BoardVO board = new BoardVO();
        board.setSeq(1);
        board.setTitle("테스트 제목...");
        board.setWriter("테스터");
        board.setContent("테스트 내용입니다.....");
        board.setCreateDate(new Date());
        board.setCnt(0);
        return board;
    }
}
```

#### 실행 결과

```
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
```

숫자와 문자열을 + 연산자로 연결하는 경우에는 어떻게 될까요? 프로그래밍에서는 데이터 타입이 다른 경우 원칙적으로 연산이 불가능합니다. 다만 한 쪽 데이터를 다른 쪽의 데이터 타입으로 바꿀 수 있다면 해당 타입으로 연산이 처리됩니다. 바꿀 수 없다면요? 오류죠.

#### BoardServiceImpl.java

```
01 : package com.rubypaper.service;
02 :
03 : import java.util.ArrayList;
04 : import java.util.Date;
05 : import java.util.List;
06 :
07 : import org.springframework.stereotype.Service;
08 :
09 : import com.rubypaper.domain.BoardVO;
10 :
11 : @Service
12 : public class BoardServiceImpl implements BoardService {
13 :
14 :     public String hello(String name) {
15 :         return "Hello : " + name;
16 :     }
17 :
18 :     public BoardVO getBoard() {
19 :         BoardVO board = new BoardVO();
20 :         board.setSeq(1);
21 :         board.setTitle("테스트 제목...");
22 :         board.setWriter("테스터");
23 :         board.setContent("테스트 내용입니다.....");
24 :         board.setCreateDate(new Date());
25 :         board.setCnt(0);
26 :         return board;
27 :     }
```

연산 구문을 보여줍니다.

#### 실행 결과

```
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
```

덧셈, 혹은 결합 연산에서 데이터를 섞어 쓰는 경우 혼란스러울 수 있습니다. 데이터 형이 혼재되어 있을 때에는 순서에 볼드와 바탕 강조

#### BoardServiceImpl.java

```
01 : package com.rubypaper.service;
02 :
03 : import java.util.ArrayList;
04 : import java.util.Date;
05 : import java.util.List;
06 :
07 : import org.springframework.stereotype.Service;
08 :
09 : import com.rubypaper.domain.BoardVO;
10 :
11 : @Service
12 : public class BoardServiceImpl implements BoardService {
13 :
14 :     public String hello(String name) {
15 :         return "Hello : " + name;
16 :     }
17 :
18 :     public BoardVO getBoard() {
19 :         BoardVO board = new BoardVO();
20 :         board.setSeq(1);
21 :         board.setTitle("테스트 제목...");
22 :         board.setWriter("테스터");
23 :         board.setContent("테스트 내용입니다.....");
24 :         board.setCreateDate(new Date());
25 :         board.setCnt(0);
26 :         return board;
27 :     }
```

덧셈, 혹은 곱셈 연산에서 데이터를 섞어 쓰는 경우 혼란스러울 수 있습니다. 데이터 형이 혼재되어 있을 때에는 순서에 민감하기 원번호 설명.

#### BoardServiceImpl.java

```
01 : package com.rubypaper.service;
02 :
03 : import java.util.ArrayList;
04 : import java.util.Date;
05 : import java.util.List;
06 :
07 : import org.springframework.stereotype.Service;
```



```

08 :
09 : import com.rubypaper.domain.BoardVO;
10 :
11 : @Service
12 : public class BoardServiceImpl implements BoardService { ❶
13 :
14 :     public String hello(String name) { ❷
15 :         return "Hello : " + name; ❸
16 :     } ❹

```

❶ 스프링 부트에서 사용한 경우는 소스 구문에 어울리지는 않으나 형태에 큰 차이가 없어 소스 구문으로 만듭니다. ❷ 소스 구문이란 메서드 사용 예나 if문 while문 등의 정의를 지정한 것입니다. ❸ 스프링 부트에서 사용한 경우는 소스 구문에 어울리지는 않으나 형태에 큰 차이가 없어 소스 구문으로 만듭니다 ❹ 소스 구문이란 메서드 사용 예나 if문 while문 등의 정의를 지정한 것입니다.

❶ 스프링 부트에서 사용한 경우는 소스 구문에 어울리지는 않으나 형태에 큰 차이가 없어 소스 구문으로 만듭니다.

❷ 소스 구문이란 메서드 사용 예나 if문 while문 등의 정의를 지정한 것입니다.

❸ 스프링 부트에서 사용한 경우는 소스 구문에 어울리지는 않으나 형태에 큰 차이가 없어 소스 구문으로 만듭니다

❹ 소스 구문이란 메서드 사용 예나 if문 while문 등의 정의를 지정한 것입니다.

스프링 부트는 템플릿 엔진을 이용한 화면 처리를 지원한다. 스프링 부트가 지원하는 템플릿 엔진은 타임리프(Thymeleaf)를 비롯하여 Freemarker, Mustache, Groovy Templates이 있다. 템플릿 엔진을 이용하면 데이터와 완벽하게 분리된 화면을 개발할 수 있기 때문에 순수하게 HTML 파일만을 이용한 화면 개발이 가능하고, 운영

과정에서 쉽게 화면을 변경할 수도 있다.

```
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
</dependency>

<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
</dependency>
```

우리는 최종적으로는 스프링 부트가 제공하는 템플릿 엔진 중에서 타임리프를 적용할 것이다. 하지만 먼저 JSP를 이용해서 간단하게 화면을 구성하고, 이후에 JSP 기반의 화면을 타임리프로 변경한다. JSP를 먼저 적용하는 가장 큰 이유는 JSP는 자바의 표준이며, 여전히 JSP로 화면을 제공하는 시스템들도 존재하기 때문이다.

### 1.3.2 구문 강조 구문 강조

소스 구문이란 메서드 사용 예나 if문 while문 등의 정의를 지정한 것입니다. 보통 다 음과 같이 한 줄이나 두 줄 정도 정의합니다. 코드 폰트보다 두껍게 사용하는 경우가 많습니다.

기초 문법에는 한글에도 볼드를 주는 경우가 많습니다.

```
for (초기치; 조건문; 증가치)
```

고급 문법에는 영문만 들어가는 경우

```
private async void startButton_Click(object sender, RoutedEventArgs e)
{
    await DoSomethingAsync();
}
```

이렇게 긴 경우도 있습니다. 일반적으로 특정 단어가 두껍게 처리됩니다.

```
switch (비교 대상) {
    case 비교값 :
        실행할 내용
        break;
    case 비교값 :
        실행할 내용
        break;
    ...
    default :
        실행할 내용
        break;
}
```

스프링 부트는 템플릿 엔진을 이용한 화면 처리를 지원한다. 스프링 부트가 지원하는 템플릿 엔진은 타임리프(Thymeleaf)를 비롯하여 Freemarker, Mustache, Groovy Templates이 있다. 템플릿 엔진을 이용하면 데이터와 완벽하게 분리된 화면을 개발할 수 있기 때문에 순수하게 HTML 파일만을 이용한 화면 개발이 가능하고, 운영 과정에서 쉽게 화면을 변경할 수도 있다.

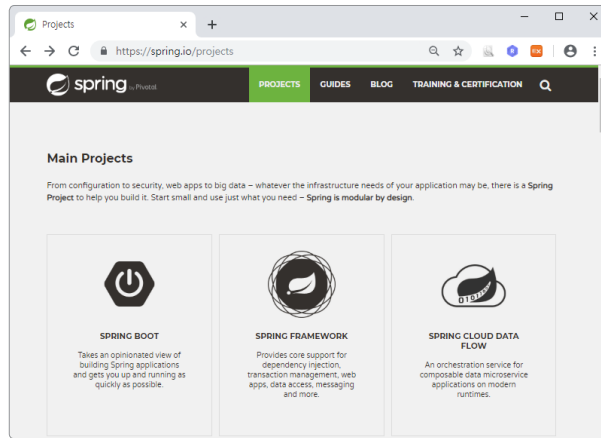
## 1.4 그림그림그림 산술 연산자

숫자 1과 2, 그리고 문자 "3"의 더하기 연산은 순서에 따라 다른 결과를 만들어 냅니다. 1행과 2행은 앞에서부터 순서대로 처리되어 "33"이라는 결과를 만들지만, 마지막 3행은 괄호로 인해 뒤쪽부터 연산이 수행됩니다. 우선 숫자 2와 문자열 3이 결합하여 문자열 "23"이 만들어지고, 여기에 다시 숫자 1이 결합하면서 "123"이 되죠. 이처럼 데이터 형이 섞여 있을 경우 연산 순서에 따라 결과가 달라지기 때문에 주의해야 합니다.

### 1.4.1 그림 종류

숫자 1과 2, 그리고 문자 "3"의 더하기 연산은 순서에 따라 다른 결과를 만들어 냅니다. 1행과 2행은 앞에서부터 순서대로 처리되어 "33"이라는 결과를 만들지만, 마지막 3행은 괄호로 인해 뒤쪽부터 연산이 수행됩니다. 우선 숫자 2와 문자열 3이 결합하여 문자열 "23"이 만들어지고, 여기에 다시 숫자 1이 결합하면서 "123"이 되죠. 이처럼 데이터 형이 섞여 있을 경우 연산 순서에 따라 결과가 달라지기 때문에 주의해야 합니다.

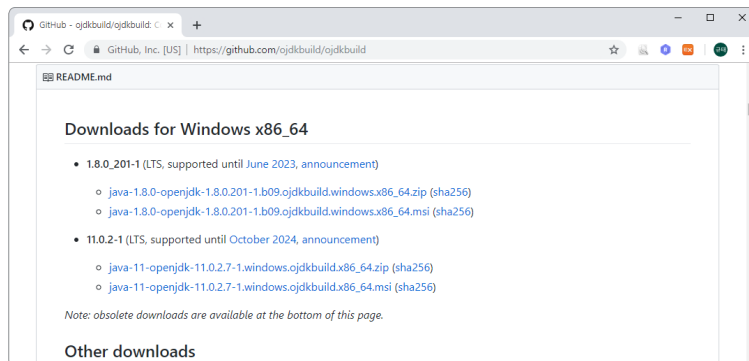
## ■ 캡션번호 있는 그림



[그림 3-1] 스프링 프레임워크 홈페이지

## ■ 캡션 번호 없는 그림

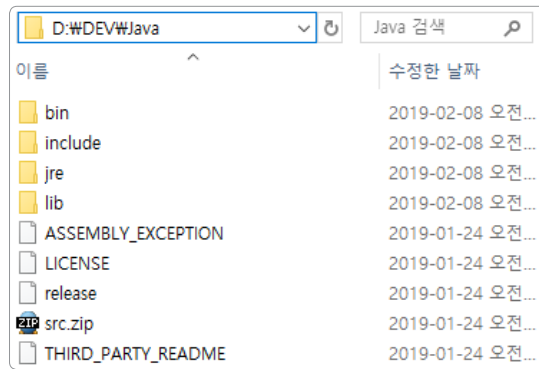
숫자 1과 2, 그리고 문자 "3"의 더하기 연산은 순서에 따라 다른 결과를 만들어 냅니다. 1행과 2행은 앞에서부터 순서대로 처리되어 "33"이라는 결과를 만들지만, 마지막 3행은 괄호로 인해 뒤쪽부터 연산이 수행됩니다.



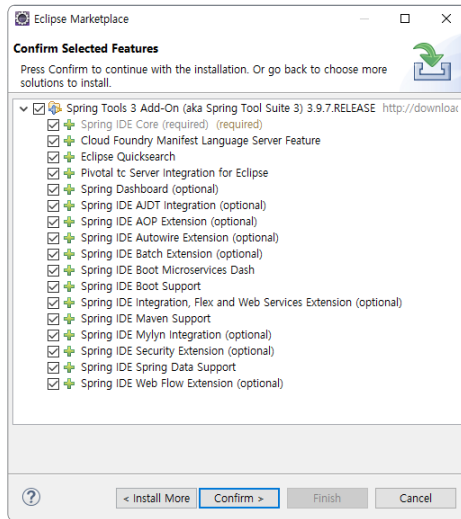
[그림] OpenJDK 다운로드

## ■ 캡션 그림

숫자 1과 2, 그리고 문자 "3"의 더하기 연산은 순서에 따라 다른 결과를 만들어 냅니다. 1행과 2행은 앞에서부터 순서대로 처리되어 "33"이라는 결과를 만듭니다



## ■ 흐르는 그림



그리고 앞에서 설명했지만 만약에 이 클립스에 플러그인을 설치하려고 하는데 설치할 플러그인이 검색되지 않으면, 현재 설치한 이클립스와 호환되는 플러그인이 아직 개발되지 않았다는 것이다. 따라서 플러그인 검색이 실패한 경우에는 직접 플러그인 URL을 입력하여 다운로드하거나 현재 설치한 이클립스 버전을 낮춰야 한다.

STS 설치 화면으로 이동하면 설치할

기능을 선택해야 하는데, STS의 모든 기능을 설치해도 설치 시간은 거의 비슷하므로 모두 선택하고 〈Confirm〉 버튼을 누른다.

STS 설치 화면으로 이동하면 설치할 기능을 선택해야 하는데, STS의 모든 기능을 설치해도 설치 시간은 거의 비슷하므로 모두 선택하고 〈Confirm〉 버튼을 누른다.

숫자 1과 2, 그리고 문자 "3"의 더하기 연산은 순서에 따라 다른 결과를 만들어 냅니다. 1행과 2행은 앞에서부터 순서대로 처리되어 "33"이라는 결과를 만들지만, 마지막 3행은 괄호로 인해 뒤쪽부터 연산이 수행됩니다. 우선 숫자 2와 문자열 3이 결합하여 문자열 "23"이 만들어지고, 여기에 다시 숫자 1이 결합하면서 "123"이 되죠. 이처럼 데이터 형이 섞여 있을 경우 연산 순서에 따라 결과가 달라지기 때문에 주의해야 합니다.

#### 1.4.2 표 종류

캡션 번호 있는 표

숫자 1과 2, 그리고 문자 "3"의 더하기 연산은 순서에 따라 다른 결과를 만들어 냅니다. 1행과 2행은 앞에서부터 순서대로 처리되어 "33"이라는 결과를 만들지만, 마지막 3행은 괄호로 인해 뒤쪽부터 연산이 수행됩니다. 우선 숫자 2와 문자열 3이 결합하여 문자열 "23"이 만들어지고, 여기에 다시 숫자 1이 결합하면서 "123"이 되죠. 이처럼 데이터 형이 섞여 있을 경우 연산 순서에 따라 결과가 달라지기 때문에 주의해야 합니다.

프로젝트 생성 시에 입력할 정보들은 다음과 같다.

항목	설명	설정 값
Name	생성할 프로젝트의 이름	Chapter01
Type	라이브러리 관리 도구 설정	Maven
Packaging	패키징 파일의 형식 지정	Jar
Group	프로젝트를 만들고 관리할 단체나 회사 정보(도메인 이름)	com.rubypaper
Package	프로젝트 생성 시 기본적으로 생성할 패키지 경로 지정	com.rubypaper

[표 3-2] 프로젝트 설정 정보

캡션 번호 없는 표

프로젝트 생성 시에 입력할 정보들은 다음과 같다.

속성	의미
properties	테스트가 실행되기 전에 테스트에 사용할 프로퍼티들을 'key=value' 형태로 추가하거나 properties 파일에 설정된 프로퍼티를 재정의한다.
classes	테스트할 클래스들을 등록한다. 만일 classes 속성을 생략하면 애플리케이션에 정의된 모든 빈을 생성한다.
webEnvironment	애플리케이션이 실행될 때, 웹과 관련된 환경을 설정할 수 있다.

[표] @SpringBootTest의 속성과 의미

머리 상단 표

메소드	설명
isOk()	응답 상태 코드가 정상적인 처리에 해당하는 200인지 확인한다.
isNotFound()	응답 상태 코드가 404 Not Found인지 확인한다.
isMethodNotAllowed()	응답 상태 코드가 메소드 불일치에 해당하는 405인지 확인한다.
isInternalServerError()	응답 상태 코드가 예외 발생에 해당하는 500인지 확인한다.
is(int status)	몇 번 응답 상태 코드가 설정되어 있는지 확인한다. 예) is(200), is(404), is(405), is(500)



## 머리 오른 표

[illegible]

## 1.5 노트주의 그리고 참고

노트와 주의 1~6줄 정도 사용하는 것으로 정의했으면 합니다. 참고와 더 알아보기는 6줄 이상으로 그 안에 그림과 코드가 들어간 경우로 정의합니다.

두 가지다 제목 있는 경우와 제목 없는 경우 두 가지 스타일이 되어야 할 것 같습니다.

그리고 노트와 주위는 글자가 아니라 아이콘을 사용하여 뭘 넣어도 크게 이상하지 않은 것이 있으면 합니다.

## 1.5.1 노트와 주의는 같지만 다르다

### (1) 노트 주의 제목 유



#### 이럴때 노트다!

노트와 주의 1~6줄 정도 사용하는 것으로 정의했으면 합니다. 참고와 더 알아보기는 6줄 이상으로 그 안에 그림과 코드가 들어간 경우로 정의합니다.

두 가지다 제목 있는 경우와 제목 없는 경우 두 가지 스타일이 되어야 할 것 같습니다.

그리고 노트와 주의는 글자가 아니라 아이콘을 사용하여 뭉뚱 놀아도 크게 이상하지 않은 것이 있으면 합니다.

숫자 1과 2, 그리고 문자 “3”의 더하기 연산은 순서에 따라 다른 결과를 만들어 냅니다. 1행과 2행은 앞에서부터 순서대로 처리되어 “33”이라는 결과를 만들지만, 마지막 3행은 괄호로 인해 뒤쪽부터 연산이 수행됩니다.



노트와 주의 1~6줄 정도 사용하는 것으로 정의했으면 합니다. 참고와 더 알아보기는 6줄 이상으로 그 안에 그림과 코드가 들어간 경우로 정의합니다.

두 가지다 제목 있는 경우와 제목 없는 경우 두 가지 스타일이 되어야 할 것 같습니다.

그리고 노트와 주의는 글자가 아니라 아이콘을 사용하여 뭉뚱 놀아도 크게 이상하지 않은 것이 있으면 합니다.

우선 숫자 2와 문자열 3이 결합하여 문자열 “23”이 만들어지고, 여기에 다시 숫자 1이 결합하면서 “123”이 되죠. 이처럼 데이터 형이 섞여 있을 경우 연산 순서에 따라 결과가 달라지기 때문에 주의해야 합니다.

## (2) 노트 주의 제목

문제가 발생된 이유는 메모리에 같은 타입의 빈이 두 개가 등록되기 때문에 충돌이 발생되었기 때문이다.

### 주의해야 할 라크랜드 44 스카이라인

문제가 발생된 이유는 메모리에 같은 타입의 빈이 두 개가 등록되기 때문에 충돌이 발생되었기 때문이다. 이 문제를 해결하려면 이전에 생성된 빈을 새롭게 생성한 빈이 덮어쓸 수 있도록 처리해주면 된다. src/main/resources 소스 폴더에 있는 application.properties 파일에 다음 설정을 추가한다.

이 문제를 해결하려면 이전에 생성된 빈을 새롭게 생성한 빈이 덮어쓸 수 있도록 처리해주면 된다. src/main/resources 소스 폴더에 있는 application.properties 파일에 다음 설정을 추가한다.



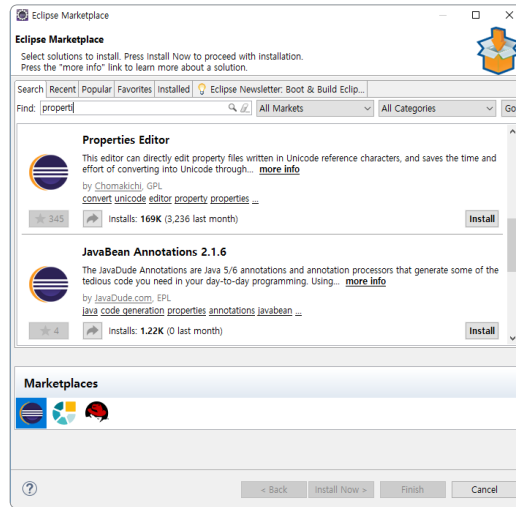
문제가 발생된 이유는 메모리에 같은 타입의 빈이 두 개가 등록되기 때문에 충돌이 발생되었기 때문이다. 이 문제를 해결하려면 이전에 생성된 빈을 새롭게 생성한 빈이 덮어쓸 수 있도록 처리해주면 된다. src/main/resources 소스 폴더에 있는 application.properties 파일에 다음 설정을 추가한다.

숫자 1과 2, 그리고 문자 “3”의 더하기 연산은 순서에 따라 다른 결과를 만들어 냅니다. 1행과 2행은 앞에서부터 순서대로 처리되어 “33”이라는 결과를 만들지만, 마지막 3행은 괄호로 인해 뒤쪽부터 연산이 수행됩니다. 우선 숫자 2와 문자열 3이 결합하여 문자열 “23”이 만들어지고, 여기에 다시 숫자 1이 결합하면서 “123”이 되죠. 이처럼 데이터 형이 섞여 있을 경우 연산 순서에 따라 결과가 달라지기 때문에 주의해야 합니다.



## Properties Editor 플러그인 설치

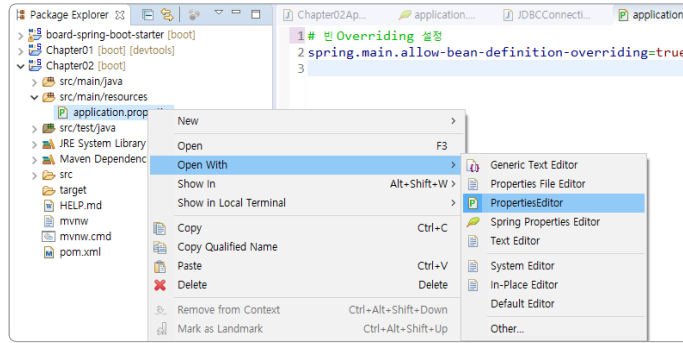
application.properties 파일에 작성한 한글이 정상적으로 보이지 않으면 'Properties Editor' 플러그인을 설치하면 된다. Help 메뉴에서 [Eclipse Marketplace...]를 선택하고, 'properties' 라는 키워드로 Properties Editor를 검색한다.



[그림] Properties Editor 검색

검색한 Properties Editor를 [install] 버튼을 눌러 설치한다.

Properties Editor가 설치가 종료되고 이클립스를 재구동하면 application.properties 파일을 선택하고 오른쪽 마우스를 클릭한다. 그리고 [Open With] → [PropertiesEditor]를 선택하 에디터를 변경하면 한글이 깨지지 않고 정상적으로 보일 것이다.



[그림] Properties Editor 적용

숫자 1과 2, 그리고 문자 "3"의 더하기 연산은 순서에 따라 다른 결과를 만들어 냅니다. 1행과 2행은 앞에서부터 순서대로 처리되어 "33"이라는 결과를 만들지만, 마지막 3행은 괄호로 인해 뒤쪽부터 연산이 수행됩니다. 우선 숫자 2와 문자열 3이 결합하여 문자열 "23"이 만들어지고, 여기에 다시 숫자 1이 결합하면서 "123"이 되죠. 이처럼 데이터 형이 섞여 있을 경우 연산 순서에 따라 결과가 달라지기 때문에 주의해야 합니다.

### [3] 참고 소스 있는 경우

숫자 1과 2, 그리고 문자 "3"의 더하기 연산은 순서에 따라 다른 결과를 만들어 냅니다. 1행과 2행은 앞에서부터 순서대로 처리되어 "33"이라는 결과를 만들지만, 마지막 3행은 괄호로 인해 뒤쪽부터 연산이 수행됩니다. 우선 숫자 2와 문자열 3이 결합하여 문자열 "23"이 만들어지고, 여기에 다시 숫자 1이 결합하면서 "123"이 되죠. 이처럼 데이터 형이 섞여 있을 경우 연산 순서에 따라 결과가 달라지기 때문에 주의해야 합니다.



## 셀렉터

Selector는 본래 오브젝티브-C에서 클래스 메소드의 이름을 가리키는 데 사용되는 참조 타입입니다. 동적 호출 등의 목적으로 @selector() 어트리뷰트에 메소드 이름을 인자값으로 넣어 전달하면 이를 내부적으로 정수값으로 매핑해서 처리하는 형태였죠. 이것이 스위프트로 넘어오면서 구조체 형식으로 정의되고, #selector() 구문을 사용하여 해당 타입의 값을 생성할 수 있게 되었습니다. 단, 스위프트에서 일급 함수로서 메소드 자체를 전달하는 것과 Selector 타입은 용법에 차이가 있음을 이해해야 합니다. Selector 타입으로 전달하는 것은 메소드의 이름이지, 메소드 자체가 아닙니다.

만약 해당 클래스가 NSObject나 UIViewController 등 오브젝티브-C 기반의 클래스를 상속받지 않았다면, #selector()의 인자값으로 사용하기 위해서 메소드 정의에 @objc 어트리뷰트를 추가해 주어야 합니다.

\* Objective-C 기반의 클래스를 상속받은 경우

```
import Foundation
class Foo : NSObject {
    func exec() {
        print("Do exec()")
    }
}
let sel01 = #selector(Foo.exec)
```

\* Objective-C 기반의 클래스를 상속받지 않은 경우

```
import Foundation
class Boo {
    @objc
    func exec() {
        print("Do exec()")
    }
}
let sel02 = #selector(Boo.exec)
```

Selector 타입에 대한 자료를 인터넷에서 찾아보면 #selector()의 인자값에 문자열 리터럴을 사용하는 예를 많이 볼 수 있다.


## 1.5.2 다운로드 경로

실수값을 표현할 때 소수점의 위치를 정확하게 고정시켜 표현하는 방식을 고정 소수점 방식\*이라고 합니다. 값을 그대로 사용하여 '10.245' 처럼 저장하는 것을 말합니다. 정확하게 값을 표현할 수는 있지만, 저장에 필요한 메모리 한계 때문에 넓은 범위의 값을 다룰 수는 없습니다. 나노 단위의 계산식이나 수십 제곱 이상의 값을 다루는 천문학 등에서는 사용하기 어려운 방식이죠.

경로는 다운로드와 강 주소를 아이콘으로 구분한다.

 <https://www.eclipse.org/downloads/packages/release/photon/r>

 <https://sports.news.naver.com/kbaseball/index.nhn>

 실수값을 표현할 때 소수점의 위치를 정확하게 고정시켜 표현하는 방식을 고정 소수점 방식이라고 합니다. 값을 그대로 사용하나요?

10.245처럼 저장하는 것을 말합니다. 정확하게 값을 표현할 수는 있지만, 저장에 필요한 메모리 한계 때문에 넓은 범위의 값을 다룰 수는 없습니다. 나노 단위의 계산식이나 수십 제곱 이상의 값을 다루는 천문학 등에서는 사용하기 어려운 방식이죠.

참고 정확하게 값을 표현할 수는 있지만, 저장에 필요한 메모리 한계 때문에 넓은 범위의 값을 다룰 수는 없습니다. 나노 단위의 계산식이나 수십 제곱 이상의 값을 다루는 천문학 등에서는 사용하기 어려운 방식이죠.

## 퀴즈 정답

프로그래밍 역시 같은 이유로 대부분 부동 소수점 방식을 사용합니다. 하지만 이 방식은 유효 숫자의 표시 방식 때문에 값이 근사적으로 표현된다는 단점이 있습니다. 앞서 수식의 결과값이 우리의 예상과 달랐던 것은 연산이 부동 소수점 방식으로 처리되었기 때문입니다. 근사값으로 계산하다보니 1.2999999999처럼 예상치 못한 결과가 나온 거죠.

**01-1** 정수 오버플로우는 C 언어 표준에 정의되어 있나요?

**01-2** `stdint.h`에 `int_least32_t`, `int_fast32_t`도 정의되어 있는데 이 자료형은 무엇인가요?

**01-3** 2-3 다음 소스에서 선언할 수 있는 식별자의 개수는 몇 개인가요?

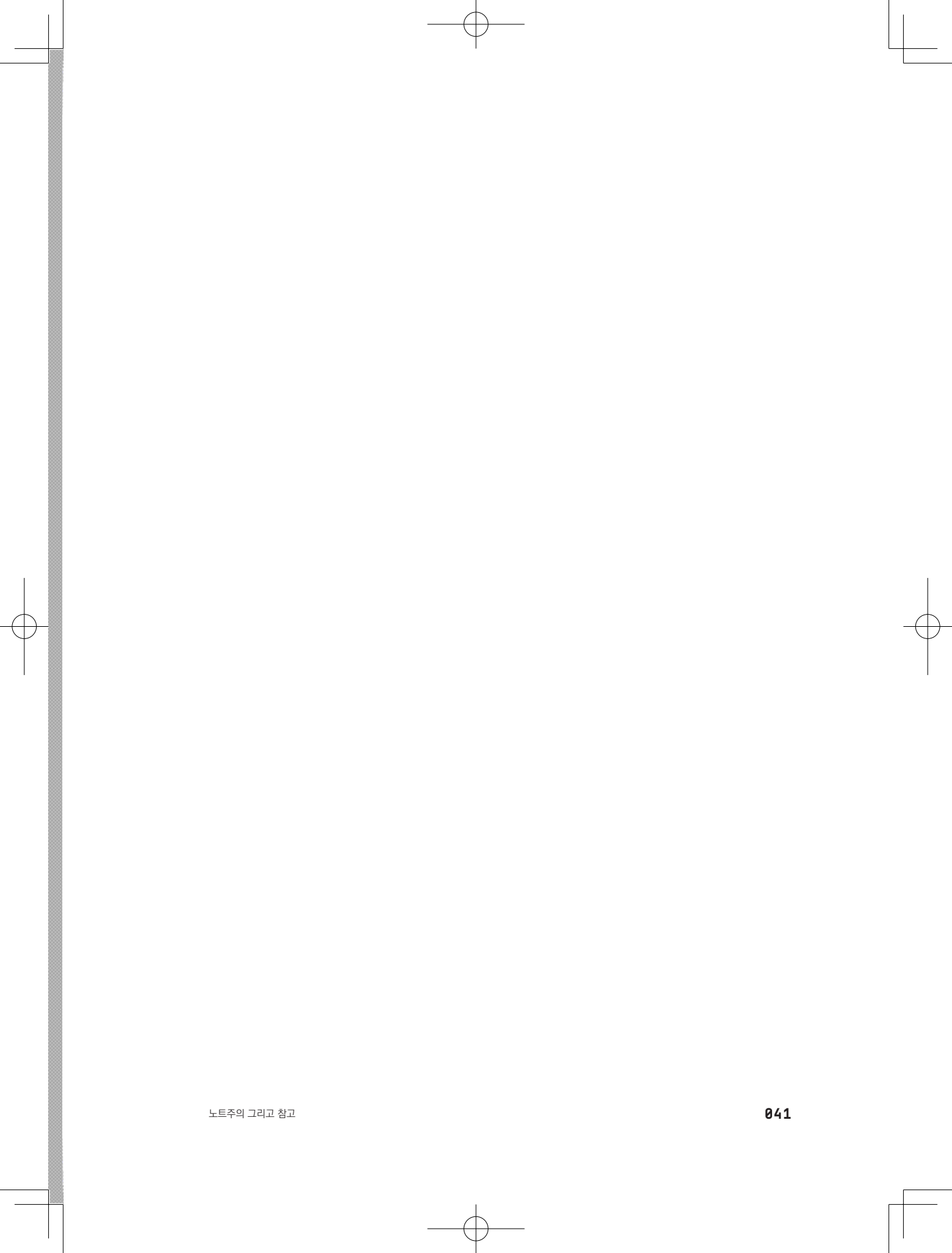
```
override func viewDidLoad() {
    ... (중략) ...
    // 자동갱신 레이블을 생성하고 루트 뷰에 추가한다.
    let lblUpdate = UILabel()
    lblUpdate.frame = CGRect(x: 30, y: 150, width: 100, height: 30)
    lblUpdate.text = "자동갱신"
    lblUpdate.font = UIFont.systemFont(ofSize: 14)

    self.view.addSubview(lblUpdate)

    // 갱신주기 레이블을 생성하고 루트 뷰에 추가한다.
    let lblInterval = UILabel()
    lblInterval.frame = CGRect(x: 30, y: 200, width: 100, height: 30)
    lblInterval.text = "갱신주기"
    lblInterval.font = UIFont.systemFont(ofSize: 14)

    self.view.addSubview(lblInterval)
}
```





## ㄱ

ILSVRC(ImageNet Large Scale Visual Recognition Challenge) 280

## L

L1 노름 ..... 135

L2 노름 ..... 136

Lasso 회귀 ..... 235

## M

mecab-ko ..... 253

MNIST ..... 282

## ㄴ~ㅅ

N-gram 분석 ..... 252

## O

One-vs-Rest ..... 255

## ㄱ

ILSVRC(ImageNet Large Scale Visual Recognition Challenge) 280

## L

L1 노름 ..... 135

L2 노름 ..... 136

Lasso 회귀 ..... 235

## M

mecab-ko ..... 253

MNIST ..... 282

## ㄴ~ㅅ

N-gram 분석 ..... 252

## O

One-vs-Rest ..... 255

ILSVRC(ImageNet Large Scale Visual Recognition Challenge) 280

## L

L1 노름 ..... 135

L2 노름 ..... 136

Lasso 회귀 ..... 235

## M

mecab-ko ..... 253

MNIST ..... 282

## ㄴ~ㅅ

N-gram 분석 ..... 252

## O

One-vs-Rest ..... 255