

JAVA 8
대응

당장 현장에 투입된 **초보 개발자**를 위한

실무에서

바로 통하는 자바



실무에서 바로 통하는 자바

당장 현장에 투입된 초보 개발자를 위한

초판발행 2017년 7월 1일

지은이 다케다 하루키, 와타나베 유지, 사토 다이치, 다다 다케아키, 가미카와 노부히코

옮긴이 김성훈 / 편년이 김태현

펴낸곳 한빛미디어(주) / 주소 서울시 마포구 양화로7길 83 한빛미디어(주) IT출판부

전화 02-325-5544 / 팩스 02-336-7124

등록 1999년 6월 24일 제10-1779호 / ISBN 978-89-6848-862-7 93000

총괄 전대호 / 책임편집 김창수 / 기획·편집 이미연

디자인 표지·내지 최연희 / 교정·조판 김철수

영업 김형진, 김진불, 조유미 / 마케팅 박상용, 송경석, 변지영 / 제작 박성우, 김정우

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 [한빛미디어\(주\)의 홈페이지](#)거나 아래 이메일로 알려주십시오. 잘못된 책은 구입하신 서점에서 교환해 드립니다. 책값은 뒤표지에 표시되어 있습니다.

한빛미디어 홈페이지 www.hanbit.co.kr / 이메일 ask@hanbit.co.kr

即戦力にならないといけない人のためのJava入門(Java 8 対応)

(SokuSenryoku ni Naranaito Ikenai Hito no tameno Java Nyumon : 4407-8)

Copyright ©2016 by Haruki Takeda, Yuji Watanabe, Daichi Sato, Takeaki Tada, Nobuhiko Kamikawa.

Original Japanese edition published by SHOEISHA Co.,Ltd.

Korean translation rights arranged with SHOEISHA Co.,Ltd. through Botong Agency.

Korean translation copyright ©2017 by Hanbit Media, Inc.

이 책의 한국어판 저작권은 Botong Agency를 통한 저작권자와의 독점 계약으로 한빛미디어가 소유합니다.

신 저작권법에 의하여 한국 내에서 보호를 받는 저작물이므로 무단전재와 무단복제를 금합니다.

지금 하지 않으면 할 수 없는 일이 있습니다.

책으로 펴내고 싶은 아이디어나 원고를 메일(writer@hanbit.co.kr)로 보내주세요.

한빛미디어(주)는 여러분의 소중한 경험과 지식을 기다리고 있습니다.

JAVA8
매일

다케다 하루키,
와타나베 유지, 사토 다이치,
다다 다케야키,
가미카와 노부히코 지음

김성훈 옮김

당장 현장에 투입된
초보 개발자^{... ...} 위한

실무에서 빠로 통하는 자바



지은이·옮긴이 소개

지은이 **다케다 하루키**

비브레이크 시스템즈에 2014년 입사했으며 오늘도 현장에서 활약하는 프로그래머다. 자바뿐만 아니라 전자공작 등 하드웨어 쪽 기사도 집필한다. 최근에는 JavaFX를 공부 중이다.

지은이 **와타나베 유지**

비브레이크 시스템즈에 2015년 입사했으며 이 업계에 들어와 자바 외길만 걸었다. 취미는 요리이고 휴일에는 훈제를 만든다.

지은이 **사토 다이치**

비브레이크 시스템즈에 2011년 입사했으며 시스템 아키텍트를 목표로 공부 중인 프로그래마다. 좋아하는 음식은 라멘이다.

지은이 **다다 다케아키**

비브레이크 시스템즈에 2008년 입사했으며 프로그래머 겸 작가로 폭넓게 활동 중이다. 저서로 『누구나 하는 안드로이드 애플리케이션 제작』이 있다. 비브레이크 시스템즈 집필 팀을 모아 후진 양성에 힘쓰고 있다.

지은이 가미카와 노부히코

1997년 조치(上智)대학 대학원 수료 후 히타치제작소에 입사했으며 2002년 비브레이크 시스템즈 설립에 참여했다. 이후 자바 시스템 개발을 중심으로 여러 분야에 손대고 있다. 사내 연수와 사내 인프라 운용도 담당한다. 최근에는 비브레이크 시스템즈 설립 무렵 태어난 아들이 자바에 흥미를 보여 조금 흐뭇하다.

옮긴이 김성훈

첫 직장에서 과외 업무로 두꺼운 매뉴얼을 1년 내내 번역한 것을 시작으로, 지금은 IT 서적을 주로 번역하는 전문 번역가로 활동 중이다. 번역서로는『C가 보이는 그림책』,『프로그램이 보이는 그림책』등의 그림책 시리즈,『게임 프로그래밍의 정석』,『Objective-C 프로그래밍』,『구글을 지탱하는 기술』,『소프트웨어 설계 테크닉』,『UML 모델링의 본질』등이 있다.

지은이의 말

이 책을 선택해주셔서 감사합니다. 이 책은 신입사원이나 다른 언어만 경험해서 자바는 초보인데 자바를 사용하는 애플리케이션 개발에 곧바로 뛰어들어야 하는 분을 위한 책입니다.

자바 표준으로 개발할 수 있는 애플리케이션은 물론이고, 데이터베이스를 다루기 위한 SQL, 자주 사용하는 라이브러리 등 애플리케이션 개발에 필요한 기초 지식 전반을 학습하는 것이 목표입니다.

이 책은 비브레이크 시스템즈 직원들이 꾸린 집필 팀에서 썼습니다. 집필 팀은 최신 기술을 연구하고 이를 알리는 데 목적을 두는 뜻있는 사람들의 그룹으로, 구성원 모두가 평등하고 독립적인 모임입니다.

집필 팀은 평소에는 각자 자유롭게 연구·집필하다가 관심사가 맞으면 팀으로 활동합니다. 그러다가 하세가와 토모유키 씨가 시작한 기획이 계기가 되어 이 책을 집필하였습니다. 하세가와 씨는 사정상 집필에 참여하지 못했지만, 그가 길을 닦아두었기에 이 책의 집필을 시작할 수 있었습니다.

원고를 쓰면서 출판사 여러분께 많은 도움을 받았습니다. 끝까지 함께해주어 감사합니다. 또한 홍보 팀의 협력을 비롯한 회사의 지원에 힘입어 이 책을 완성할 수 있었습니다. 협력해주신 모든 분께 감사드립니다.

특히 휴일에는 모든 일을 제쳐두고 책을 쓰는 데 몰두했습니다. 글이 생각대로 써지지 않아 번민했던 날도 있었습니다. 그런 저자들을 응원해주고 지켜준 가족과 친구들에게 진심으로 감사합니다. 정말 고마웠습니다.

저자 일동

옮긴이의 말

잘 만들어진 책은 대체로 잘 읽힙니다. 책을 읽기면서 내내 범위와 주제 선정이 잘 됐다고 생각했습니다. 이 책의 원제목은『즉전력이 돼야 하는 사람을 위한 Java 입문』으로, 자바로 곧장 애플리케이션 개발을 시작해야 하는 초보 개발자가 꼭 알아야 할 내용에 초점을 맞춘 입문서입니다. 중요한 문법부터 데이터베이스, 테스트, 성과물 관리까지 넓은 범위를 다루면서도 입문서로서 방향을 잡아주는 든든한 길잡이가 될 수 있도록 친절하고 이해하기 쉽게 설명한 것이 특징입니다.

이 책은 ‘개발 회사에 입사 후 현장에 배치되기 전에’ 읽어두면 도움이 될 만한 책입니다. 프로그래밍 경험이 전혀 없다면 어렵게 느껴지는 곳도 몇 군데 있습니다. 그럼에도 예제로 확인해가면서 이 책에서 다른 내용을 빼대로 삼아 깊이를 더하면 실무에서 바로 통하는 자바 개발자가 될 수 있습니다.

보통 옮긴이의 말을 쓰고 나면 옮긴이로서의 할 일이 대개 마무리되므로 비로소 원서의 책장을 덮게 됩니다. 그러면서 여러 생각이 듭니다. 특히 이 책은 개인적으로나 국가적으로나 많은 일이 있던 시기에 작업한 책이라서 더 오래 기억에 남을 것 같습니다. 어려운 시기에는 일에 몰두하는 것도 마음을 쉬게 하는 휴식이 되더군요. 아마도 이 책이 쉽게 학습할 수 있게 구성되어 있어 더 편하게 느껴진 것이 아닐까 하는 생각도 듭니다.

끝으로 번역 작업을 맡겨주신 한빛미디어와 번역 원고를 꼼꼼히 확인하고 매끄럽게 다듬어주신 편집자께 깊이 감사드립니다. 항상 하는 말이지만 부디 이 책을 선택해 주신 독자 여러분께 책의 가치가 잘 전달돼 원하는 바를 얻을 수 있게 된다면 옮긴이로서 무척 기쁘겠습니다.

김성훈

이 책에 대하여

대상독자

- 자바를 전혀 경험해보지 않은 사람
- 다른 언어는 다뤄봤지만 자바는 초보며, 자바로 애플리케이션을 개발하게 된 사람
- 시스템 개발 회사에서 프로그래밍을 업무로 할 예정인 초보자
- 위와 같은 사람을 지도할 시스템 개발 회사의 교육담당자
- 장래 시스템 개발을 희망하는 사람

실행 환경

이 책의 코드는 다음 환경에서 동작을 확인했습니다.

- 윈도우 10
- JDK 8
- Eclipse IDE for Java EE Developers(4.5.1)

본문에서 소개한 단축키와 운영체제 설정 등은 윈도우 환경을 기본으로 설명합니다.

예제

이 책의 예제 프로그램은 한빛미디어 웹 페이지에서 내려받을 수 있습니다.

<http://www.hanbit.co.kr/src/2862>

CONTENTS

지은이 · 옮긴이 소개	4
지은이의 말	6
옮긴이의 말	7
이 책에 대하여	8

1장 자바 기초 지식

1.1 자바 시작하기	18
1.1.1 자바란	18
1.1.2 왜 자바인가	19
1.1.3 자바 애플리케이션 작성	20
1.2 개발 환경 구축	21
1.2.1 필요한 환경	21
1.2.2 JDK 설치	21
1.2.3 이클립스 설치	26
1.3 자바의 기본	36
1.3.1 기본 문법	37
1.3.2 패키지	47
1.3.3 접근제한자	50
1.3.4 키워드	51
1.3.5 식별자	53
1.3.6 주석	54
1.3.7 static 변수와 static 메서드	56
1.4 기본적인 계산	59
1.4.1 자바의 주요 연산과 제어	59
1.4.2 산술 연산	59

CONTENTS

1.5 자료형	68
1.5.1 기본 자료형	69
1.5.2 참조형	72
1.5.3 인스턴스	74
1.5.4 래퍼 클래스	74
1.5.5 상수	75
1.6 비교 연산	76
1.6.1 주요 비교 연산자	76
1.6.2 참조형 비교	77
1.6.3 String(문자열) 비교	79
1.6.4 자료형 비교(instanceof 연산자)	80
1.6.5 논리 연산	82
1.7 조건 분기	84
1.7.1 if 문	84
1.7.2 switch 문	88
1.7.3 for 문	90
1.7.4 while 문과 do-while 문	93
1.7.5 break 문과 continue 문	95
1.7.6 return 문	101
1.8 클래스와 인터페이스	101
1.8.1 객체 지향	102
1.8.2 자바 언어에서의 캡슐화	108
1.8.3 자바 언어에서의 상속	108
1.8.4 상속과 생성자	113
1.8.5 자바 언어의 다형성	116
1.8.6 인터페이스	120
1.8.7 위임	122

1.8.8	상속과 위임의 단점	124
1.9	제네릭	127
1.9.1	제네릭이 등장하기 전에는	127
1.9.2	타입 세이프	130
1.9.3	제네릭을 클래스 정의에 사용한다	130
1.10	람다식	132
1.10.1	람다식 기본 구문	132
1.10.2	익명 클래스	133
1.10.3	람다식 사용	134

2장 프로그래밍 기초

2.1	문자열 조작	140
2.1.1	문자열 기초 지식	140
2.1.2	문자열 연결	141
2.1.3	문자열의 형식	144
2.2	날짜 및 시간 조작	151
2.2.1	날짜 및 시간을 다루는 클래스	152
2.2.2	현재 날짜를 이용한 데이터 조작	152
2.2.3	Date-Time API 기초 지식	160
2.3	집합체	168
2.3.1	배열	168
2.3.2	컬렉션	172
2.3.3	ArrayList 클래스(List 인터페이스 구현)	174
2.3.4	HashSet 클래스(Set 인터페이스 구현)	180
2.3.5	HashMap 클래스(Map 구현)	184

CONTENTS

2.4 Apache-Commons	189
2.4.1 외부 라이브러리를 이용하는 도구 Maven	189
2.4.2 Apache-Commons 설치	193
2.4.3 Apache-Commons 이용 방법	195

3장 데이터베이스

3.1 데이터베이스 기초	198
3.1.1 데이터베이스란 무엇인가	198
3.1.2 데이터베이스의 종류와 특징	199
3.1.3 관계형 데이터베이스	204
3.1.4 데이터를 파일로 관리할 때의 문제점	210
3.1.5 데이터베이스 설계	213
3.1.6 데이터베이스 조작	216
3.2 데이터베이스 환경 구축	218
3.2.1 PostgreSQL 다운로드	218
3.2.2 PostgreSQL 설치	221
3.2.3 PostgreSQL 동작 확인	226
3.2.4 환경 변수 설정	227
3.3 SQL의 기본	228
3.3.1 SQL 실행 방법	228
3.3.2 psql을 사용한 접속과 해제	228
3.3.3 데이터베이스	229
3.3.4 테이블	234
3.3.5 인덱스	239
3.3.6 롤과 권한	240
3.3.7 선택(SELECT)	243

3.3.8 조건 지정	246
3.3.9 삽입(INSERT)과 갱신(UPDATE)	251
3.3.10 삭제(DELETE, TRUNCATE)	253
3.3.11 결합	254
3.4 데이터베이스 접속	260
3.4.1 자바를 사용한 데이터베이스 접속	260
3.4.2 개발 환경 구축	262
3.5 트랜잭션	274
3.5.1 트랜잭션 관리	275
3.5.2 락	276
3.5.3 자바 애플리케이션의 트랜잭션 관리	279
3.6 파라미터 지정 SQL 처리	286
3.6.1 프리페어드 스테이트먼트	286
3.6.2 프리페어드 스테이트먼트 이용 사례	287
3.7 ORM으로 쾌적한 데이터베이스 프로그래밍	290
3.7.1 ORM	290
3.7.2 DAO와 DTO	291
3.7.3 JPA	296

4장 텍스트 입출력

4.1 텍스트 파일 읽기	316
4.1.1 파일의 문자를 한 문자씩 읽는 방법	316
4.1.2 텍스트를 한 줄씩 읽는 방법	318
4.1.3 텍스트를 한번에 모두 읽는 방법(1)	321
4.1.4 텍스트를 한번에 모두 읽는 방법(2)	323

CONTENTS

4.2 텍스트 파일 쓰기	324
4.2.1 FileWriter 클래스로 파일에 쓰기	325
4.2.2 BufferedWriter 클래스로 파일에 쓰기	327
4.2.3 Files 클래스로 파일에 쓰기	329
4.3 CSV 파일의 입출력	331
4.3.1 CSV 파일 읽기	331
4.3.2 CSV 파일에 쓰기	333
4.4 XML 다루기	336
4.4.1 XML이란	336
4.4.2 XML의 구조	336
4.4.3 XML 파일 읽기	338
4.5 로그 출력	344
4.5.1 자바의 로깅 API	344
4.5.2 로그 레벨	345
4.5.3 로그 출력 방식	346
4.5.4 로그 출력 형식	346
4.5.5 로그 출력을 제어하는 프로퍼티 파일	347
4.5.6 콘솔에 로그 출력	349
4.5.7 프로그램 내부에서 프로퍼티를 설정해 로그 출력	351
4.5.8 프로퍼티 파일을 사용해서 로그 출력	353

5장 스레드

5.1 멀티 스레드 처리	360
5.1.1 스레드	360
5.1.2 멀티 스레드	361
5.1.3 더 복잡한 멀티 스레드 제어 방법	363

5.2 스레드 세이프란	366
5.2.1 스레드 세이프하지 않은 경우의 사례	367
5.2.2 스레드 세이프한 프로그램을 만들자	369
5.3 Stream API의 병렬 처리	376
5.3.1 Stream API	376
5.3.2 Stream API와 람다식	380
5.3.3 Stream API를 사용할 때 주의할 점	381

6장 테스트

6.1 테스트 기초 지식	386
6.1.1 테스트 공정이란	386
6.1.2 단위 테스트	388
6.1.3 결합 테스트	393
6.1.4 통합 테스트(시스템 테스트)	393
6.1.5 그 밖의 테스트	394
6.1.6 단위 테스트 방법	395
6.2 JUnit	397
6.2.1 환경 구축	397
6.2.2 테스트 대상 클래스 작성	397
6.2.3 테스트 클래스 작성	400
6.2.4 테스트 실행과 결과를 읽는 방법	405
6.2.5 디버그 방법	407
6.2.6 어노테이션	409
6.2.7 Assert 클래스	412
6.2.8 Matcher 클래스	414

CONTENTS

6.3 자주 사용하는 테스트 도구	414
6.3.1 목 라이브러리(JMockit)	415
6.3.2 데이터베이스 관련 확장 라이브러리(DbUnit)	424
6.3.3 정적 테스트	434

7장 팀 개발

7.1 팀 개발이란	440
7.1.1 팀 개발이란 '여러 사람이 공동으로 성과물을 만들어내는 것'	440
7.1.2 팀 개발의 요점	441
7.2 성과물 관리 - 버전 관리	442
7.2.1 버전 관리 시스템 = 성과물 관리 수단	444
7.2.2 Subversion 조작	448
7.2.3 버전 관리 시스템 사용 시 주의할 점	452
7.2.4 버전 관리 시스템이 없는 경우	453
7.3 과정 공유 - 작업 관리	456
7.3.1 '과정 공유'를 위한 동기 부여	457
7.3.2 작업 관리 시스템 = 과정을 공유하기 위한 수단	458
7.3.3 Redmine 조작	459
7.3.4 작업 관리 시스템이 없는 경우	463
7.4 작업 자동화 - CI	464
7.4.1 자동화의 장점	465
7.4.2 CI 도구 = 작업을 자동화하는 수단	466
7.4.3 CI를 구현하는 시스템이 없는 경우	471
7.4.4 정리	473
찾아보기	474

1

장

자바 기초 지식

1.1 자바 시작하기

이 절에서는 자바가 무엇인지 간단히 소개합니다. 업무 애플리케이션 개발에 왜 자바를 많이 선택하는지 자바의 매력을 조금이라도 느껴보시기 바랍니다.

1.1.1 자바란

자바는 1995년에 썬 마이크로시스템즈(2010년에 오라클에 흡수 합병됨)에서 공개한 프로그래밍 언어입니다. 자바에는 몇 가지 특징이 있지만, 다음 2가지 특징을 손꼽을 수 있습니다.

- 가상 머신에서 동작한다.
- 객체 지향 언어다.

자바는 가상 머신에서 동작합니다. 가상 머신이란 소프트웨어로 제공되어 OS 상에서 가상으로 동작하는 머신을 말합니다. 특히 자바가 동작하는 가상 머신을 **JVM^{Java Virtual Machine}**(자바 가상 머신)이라고 부릅니다(그림 1-1).

그림 1-1 JVM은 OS 상에서 동작한다.



또한 자바는 **객체 지향 소프트웨어 개발** 기법을 바탕으로 설계됐습니다. 객체 지향이란 현실 세계의 사물에 비유해 프로그램을 작성하는 사고방식입니다.

NOTE_ 객체 지향이란?

객체 지향 object-oriented 은 대규모 소프트웨어 개발에서 생기는 불편함을 해결하기 위해 고안됐습니다. 대규모 개발에서는 많은 사람이 동시에 프로그램을 작성합니다. 객체 지향 이전의 프로그래밍 언어로 개발할 때는 사람마다 프로그래밍 경험이나 사고방식이 다르기 때문에 구현 방법이 다양해서 통일감이 없었고, 작성자만 알아볼 수 있는 프로그램이 만들어지기도 했습니다. 읽기 어려운 프로그램은 기능을 수정하거나 다시 이용하고자 할 때 아무래도 읽기 쉬운 프로그램보다 개발 효율이 떨어집니다.

객체 지향에서는 어느 정도까지 구현 방법을 통일할 수 있게 됐고, 작성자가 아니더라도 프로그램을 이해하기 쉬워졌습니다. 객체 지향을 도입하면 기능 변경이나 재사용이 쉬워져 개발 효율이 올라갑니다.

이런 장점 덕분에 요즘 프로그래밍 언어에는 대부분 객체 지향 사고방식이 들어 있습니다. 객체 지향 사고방식에 관해서는 1.8절에서 더 자세히 설명합니다.

1.1.2 왜 자바인가

앞서 소개한 자바의 특징이 업무 애플리케이션 개발에 어떤 도움을 줄까요? 가상 머신(JVM)에서 동작한다는 것과 객체 지향 언어라는 2가지 관점에서 소개합니다.

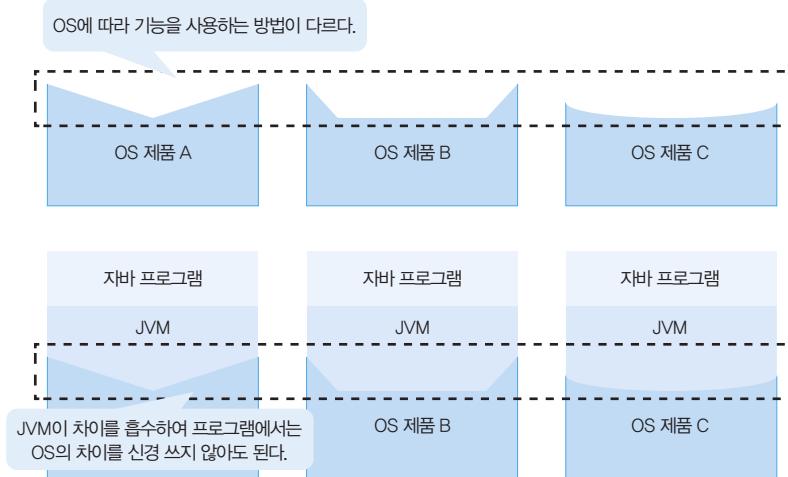
가상 머신의 장점

OS에는 윈도우, 리눅스, macOS 등 다양한 종류가 있고, 각 OS가 제공하는 기능을 사용하는 방법도 다릅니다.

예전에는 이 차이를 프로그램 쪽에서 흡수해야 했습니다. 즉, OS별로 프로그램을 따로 준비해야 했던 것입니다. 이 작업은 시간이 오래 걸리고, 업무 애플리케이션을 만드는 기업에는 큰 부담이 됐습니다.

이 문제를 자바에서는 JVM으로 해결했습니다. 다시 말해, OS의 차이를 프로그램이 아니라 JVM에서 흡수함으로써 개발자는 OS의 차이를 의식하지 않고 업무 애플리케이션을 작성할 수 있게 됐습니다(그림 1-2).

그림 1-2 JVM에서 OS의 차이를 흡수한다.



객체 지향 언어의 장점

객체 지향은 인원이 많이 투입되는 개발에 적합합니다. 하나의 기능을 몇 개의 작은 프로그램으로 나눌 수 있는 구조와 프로그램 작성 규칙을 어느 정도 강제로 지키게 하는 구조 등 동시에 많은 사람이 개발하는 데 편리한 기능이 마련되어 있습니다.

특히 업무 애플리케이션은 규모가 커서 개발에 인원이 많이 투입됩니다. 이때 객체 지향 언어라는 특징은 큰 장점이 됩니다.

1.1.3 자바 애플리케이션 작성

자바로 만들어진 애플리케이션은 하나 이상의 클래스 파일로 구성됩니다.

클래스 파일이란 컴퓨터가 처리하기 쉬운 형식으로 기술된 파일로, 인간이 이해할 수 있는 형태로 기술된 소스 파일을 바탕으로 만들어집니다. 클래스 파일처럼 컴퓨터가 처리할 수 있는 형식의 코드를 가리켜 **바이트 코드**라고 부릅니다.

소스 파일을 클래스 파일로 만들기 위해선 컴파일 과정을 거쳐야 합니다. 컴파일이란 인간이 이해할 수 있는 소스 파일을 컴퓨터가 처리할 수 있는 바이트 코드로 번역하는 것입니다. 이 번역 작업을 하는 애플리케이션이 바로 **자바 컴파일러**입니다(그림 1-3).

그림 1-3 클래스 파일 만들기



1.2 개발 환경 구축

이 절에서는 자바 업무 애플리케이션 개발에 필요한 환경을 준비합니다. 자바 애플리케이션은 대부분 통합 개발 환경(IDE)에서 만들어집니다. 통합 개발 환경이란 애플리케이션 개발을 지원하는 환경으로, 이 책에서는 많은 협장에서 사용하는 **이클립스 Eclipse**를 선택했습니다.

지금부터 이클립스를 설치해 자바 개발 환경을 갖추고, 예제 애플리케이션을 만들어 실행해보겠습니다.

1.2.1 필요한 환경

이 책에서는 윈도우에서 애플리케이션을 개발한다는 전제로 설치 과정을 설명합니다. 설치할 것은 다음 2가지 소프트웨어입니다.

- JDK 8
- Eclipse IDE for Java EE Developers (4.5.1)

이 책에서는 이클립스 영문 버전을 사용합니다. 이클립스 한글 버전을 사용할 경우에도 메뉴나 버튼 등의 명칭만 한글화되어 있을 뿐 이 책을 읽는 데는 지장이 없습니다. 문헌(특히 해외 문헌) 등을 읽을 때 메뉴나 버튼의 영문 표기가 자주 등장하므로 영문 표기에 익숙해지도록 영문 버전을 사용한 것뿐입니다.

1.2.2 JDK 설치

자바로 개발을 하려면 **JDK** Java Development Kit라는 자바 애플리케이션 개발 도구 모음

을 설치해야 합니다. JDK는 자바로 기술된 소스 파일을 컴파일하는 등 자바 애플리케이션을 만들 때 사용합니다. JDK는 JDK 인스톨러로 간단히 설치할 수 있습니다.

또한 JDK 인스톨러는 JRE^{Java Runtime Environment}라는 환경도 동시에 설치합니다. JRE에는 자바 애플리케이션을 작성하는 기능은 없고, 실행하는 기능만 있습니다.

JDK 인스톨러를 실행할 때는 관리자 권한을 가지고 있는지 확인해주세요.

[1] JDK 다운로드

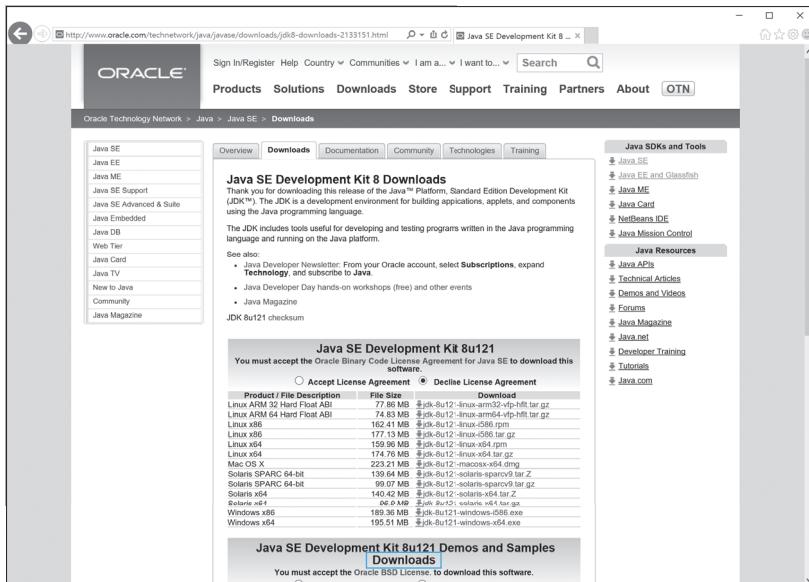
JDK를 설치하려면 우선 JDK 인스톨러를 다운로드해야 합니다. JDK 인스톨러 다운로드 사이트 URL은 다음과 같습니다.

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

사이트가 연결되지 않을 때는 구글 등에서 'JDK 다운로드'로 검색하면 'Java SE Downloads - Oracle'처럼 오라클의 JDK 다운로드 페이지를 찾을 수 있습니다.

웹 브라우저 주소 입력창에 JDK 다운로드 사이트 URL을 입력하면 다음 그림과 같은 다운로드 페이지로 이동합니다. 여기서는 집필 시점 최신 버전인 JDK 8을 설치합니다.

그림 1-4 JDK 다운로드 페이지



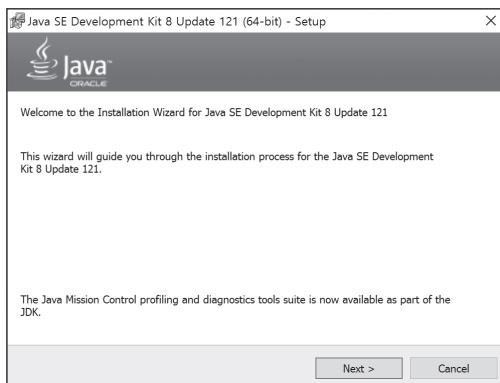
DOWNLOAD 버튼을 누르면 라이선스 동의 페이지가 나타납니다. Accept License Agreement를 선택해 라이선스에 동의한 뒤(라이선스에 동의하지 않으면 JDK 인스톨러를 다운로드할 수 없습니다) 자신의 컴퓨터 환경에 적합한 JDK의 인스톨러를 다운로드합니다. 예를 들어 컴퓨터의 운영체제가 윈도우 64비트 버전이면 Windows x64를, 윈도우 32비트 버전이면 Windows x86을 선택합니다.

[2] JDK 설치

인스톨러를 다운로드했으면 JDK를 설치합니다.

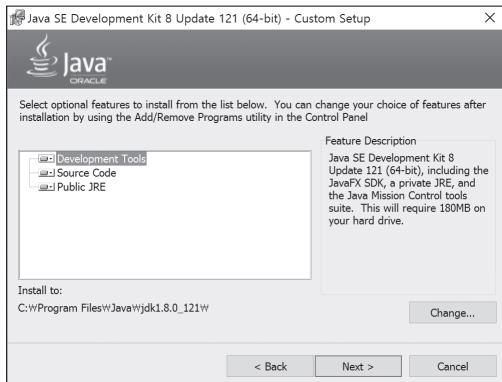
다운로드한 인스톨러를 실행하세요. 여기서는 특별한 설정을 하지 않고 기본값 그대로 설치합니다. 다음 그림과 같이 인스톨러가 시작되면 Next 버튼을 누릅니다.

그림 1-5 인스톨러 시작 화면



[그림 1-6]과 같은 창이 나타나면 설치 위치를 지정합니다. 이 책에서는 기본으로 설정된 폴더에 JDK를 설치합니다. Next 버튼을 눌러 설치를 시작하세요. 설치 위치를 바꿔도 상관없습니다. 설치 위치를 바꾸고 싶으면 Change 버튼을 누른 뒤 설치 위치를 지정하면 됩니다.

그림 1-6 설치 위치 지정



JDK 설치가 끝나면 JRE 설치가 이어집니다. JRE는 자바로 만들어진 애플리케이션을 실행하기 위한 환경입니다.

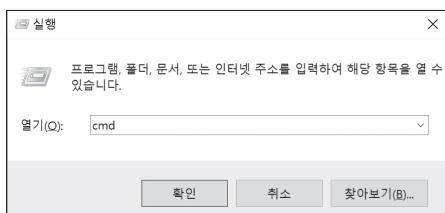
설치가 끝나면 Close 버튼을 눌러 인스톨러를 종료합니다.

[3] 설치 후 설정 확인

설치를 마쳤으면 자바 버전을 확인합니다.

윈도우 키와 R 키를 동시에 누르면 실행 창이 엽니다. ‘열기’에 ‘cmd’를 입력하고 확인 버튼을 누릅니다.

그림 1-7 명령어를 입력해서 실행



다음 그림과 같이 명령 프롬프트 창에 `java -version` 명령을 입력하고 실행하면 자바 버전을 확인할 수 있습니다.

그림 1-8 버전 표시

```
C:\>java -version
java version "1.8.0_121"
Java(TM) SE Runtime Environment (build 1.8.0_121-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)

C:\>
```

JDK 8을 설치한 경우 ‘java version 1.8.0’으로 표시되면 문제없이 설치된 것입니다. 업데이트 패치가 포함된 경우에는 ‘java version 1.8.0_xxx’로 표시됩니다.

NOTE 자바 버전 표시

JDK 8을 설치했는데 어째서 버전이 8이 아니라 ‘java version 1.8.0’처럼 1.8로 표시되는 걸까요.

자바 1.5 버전 이전에는 대규모 버전업이 있을 때 1.2나 1.3처럼 두 번째 숫자를 증가시켰지만 1.5 버전부터 1.5가 아니라 5로 표기하기 시작했습니다. 하지만 내부에서는 버전을 이전처럼 1.5로 관리했고, 그것이 현재도 이어지고 있습니다. 그래서 JDK 8을 설치했어도 버전은 1.8로 표시되는 것입니다. 자바 버전을 [표 1-1]에 정리했습니다.

표 1-1 자바 버전

이름	제품 버전	개발자 버전	공개 연도
JDK 1.0	1	–	1996
JDK 1.1	1.1	–	1997
J2SE 1.2	1.2	–	1998
J2SE 1.3	1.3	–	2000
J2SE 1.4	1.4	–	2002
J2SE 5.0	5	1.5	2004
Java SE 6	6	1.6	2006
Java SE 7	7	1.7	2011
Java SE 8	8	1.8	2014

1.2.3 이클립스 설치

자바 애플리케이션을 개발할 때 텍스트 파일을 편집하고, 명령줄에서 컴파일하고, 컴파일로 생성된 클래스 파일을 다시 실행하는 과정을 매번 수동으로 하기엔 시간이 오래 걸리고 개발 효율도 좋지 않습니다.

또한 업무 애플리케이션 개발에서는 규모가 커지면 작성하는 소스 파일이 계속 늘어나서 사람이 수동으로 작업하면 실수할 가능성이 커집니다. 예를 들어 소스 코드를 편집할 때 오자가 있어도 컴파일해서 오류가 날 때까지 그 실수를 알아채기 어렵거나, 소스 파일 수가 많아지면 전부 컴파일하는 데 상당한 노력이 들기도 합니다.

이러한 개발상에서의 번거로움을 개선하고자 자바에서는 통합 개발 환경(IDE)으로 불리는 개발 환경을 무료로 사용할 수 있습니다. 통합 개발 환경을 사용하면 소스 코드 작성 도중에 입력한 단어(클래스명 등)를 자동으로 완성해주거나 오지를 발견해 주기도 하고, 모든 소스 파일을 한번에 컴파일할 수 있는 등 개발 효율이 현격히 올라갑니다.

이 책에서는 통합 개발 환경 중에서도 현장에서 많이 사용하는 이클립스를 선택했습니다. 그럼 이클립스를 설치해봅시다.

[1] 이클립스 다운로드

이클립스는 다음 URL에서 다운로드할 수 있습니다.

<http://www.eclipse.org/downloads/>

이클립스에는 사용 목적에 따라 다양한 종류가 있지만, 여기서는 자바로 개발할 것 이므로 Eclipse IDE for Java EE Developers에서 윈도우 버전에 맞게 32비트인 경우에는 32bit, 64비트인 경우는 64bit 링크를 누릅니다. 그리고 다음 페이지에서 DOWNLOAD 버튼을 눌러 다운로드합니다.

[2] 이클립스 설치와 실행

이클립스는 다운로드한 압축 파일을 원하는 폴더에서 압축 해제해 `eclipse` 폴더를 두는 것만으로 설치가 끝납니다. `eclipse` 폴더 안에 있는 `eclipse.exe`를 실행하면 이클립스가 시작됩니다.

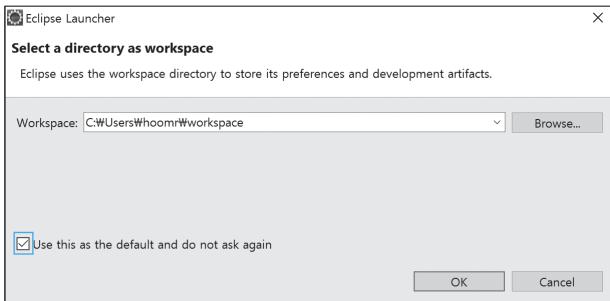
[3] 이클립스 초기 설정

이클립스를 처음 시작하면 이클립스가 사용할 워크스페이스 workspace를 어느 폴더로 할지 물어봅니다. 워크스페이스란 이클립스로 개발할 때 파일을 생성할 장소입니다.

경로에 공백이나 한글이 포함되어 있지 않으면 기본값을 그대로 사용해도 문제없습니다. 하지만 공백이나 한글이 포함된 경우에는 그런 문자가 포함되지 않은 다른 장소에 워크스페이스용 폴더를 만들고 그 폴더로 지정해주세요. 특히 사용자명에 한글이나 공백이 들어 있으면 워크스페이스에도 한글이나 공백이 포함되니 주의해야 합니다(이 경우 이클립스의 플러그인에 따라서는 지정한 폴더를 인식하지 못해 이클립스를 사용하지 못할 수도 있습니다).

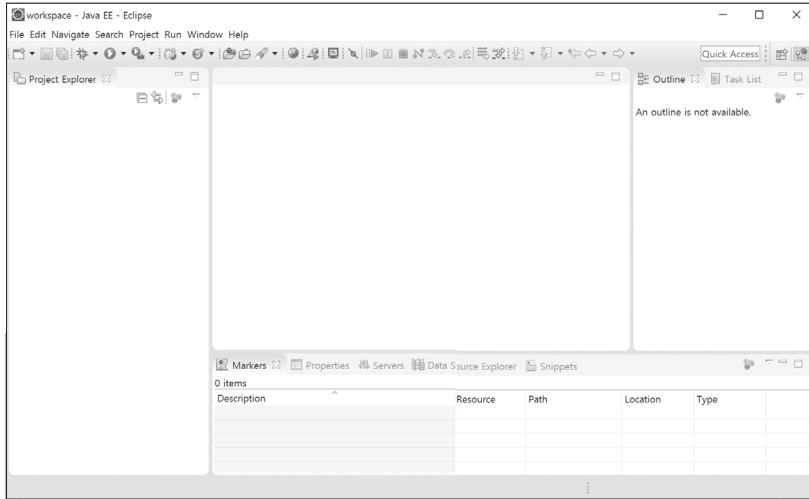
다음 그림과 같이 대화 창 왼쪽 아래에 있는 체크박스를 체크해두면 다음부터는 같은 워크스페이스를 사용하고 처음 시작할 때 대화 창을 표시하지 않습니다.

그림 1-9 워크스페이스 지정



이제 Welcome 페이지가 표시되는데, 이 페이지를 닫으면 드디어 이클립스로 개발 할 수 있게 됩니다.

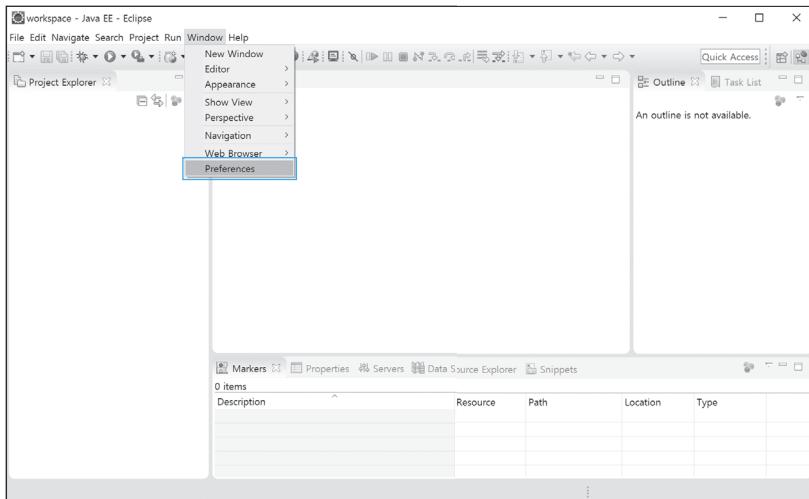
그림 1-10 이클립스 실행 화면



여기서 이클립스 설정을 확인해봅시다. 이 책에서는 자바 8을 사용할 것이므로 사용 할 JRE가 자바 8인지 확인합니다.

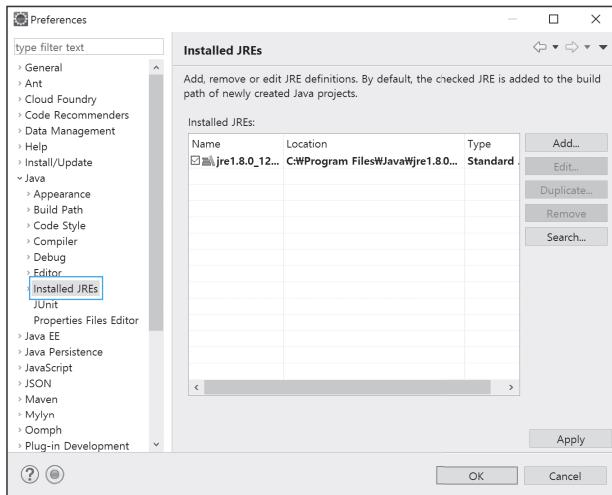
다음과 같이 이클립스 실행 화면에서 Window → Preferences 메뉴를 선택합니다.

그림 1-11 Preferences 창 열기



다음 그림과 같이 Preferences 창이 열리면 왼쪽 패널에서 Java → Installed JREs를 선택해서 어떤 JRE를 사용하는지 확인합니다. 여기서는 JRE 8을 사용하므로 문제가 없지만 만약 다른 JRE를 사용하는 경우에는 Add 버튼을 클릭한 뒤 사용하고자 하는 JRE를 지정합니다.

그림 1-12 사용할 JRE 설정 확인

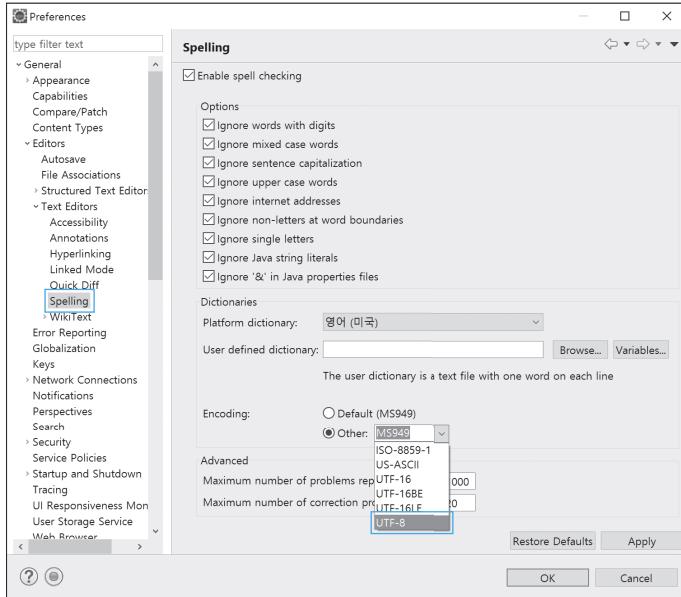


또한 이 책에서는 자바 소스 코드를 UTF-8로 프로그래밍하므로 이클립스의 기본 문자 코드¹를 UTF-8로 변경합니다.

기본 문자 코드를 변경하려면 Preferences 창에서 General → Editors → Text Editors → Spelling을 선택합니다. 그리고 창의 중간쯤에 있는 Encoding 라디오 버튼에서 Other를 클릭한 뒤 목록에서 UTF-8을 선택합니다.

1 컴퓨터상에서 문자를 이용하기 위해 할당하는 번호(코드)를 말합니다.

그림 1-13 문자 코드 지정

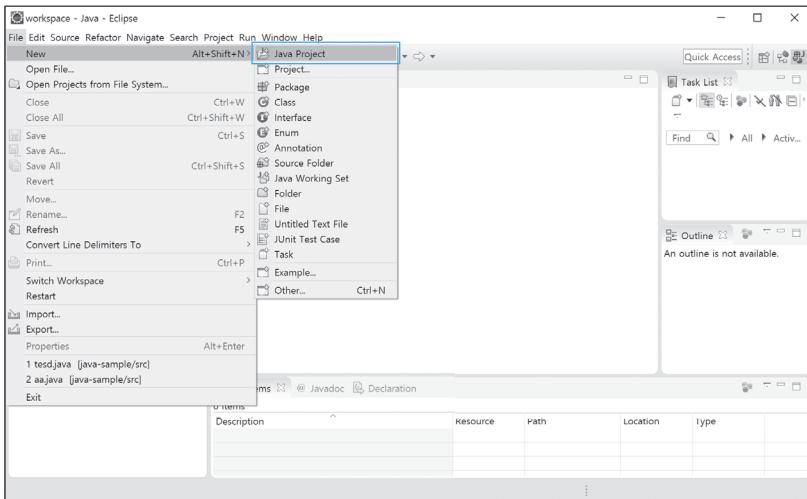


모두 설정했으면 Apply 버튼을 눌러 변경 사항을 등록하고, OK 버튼을 눌러 환경 설정을 마칩니다.

[4] 새 프로젝트 만들기

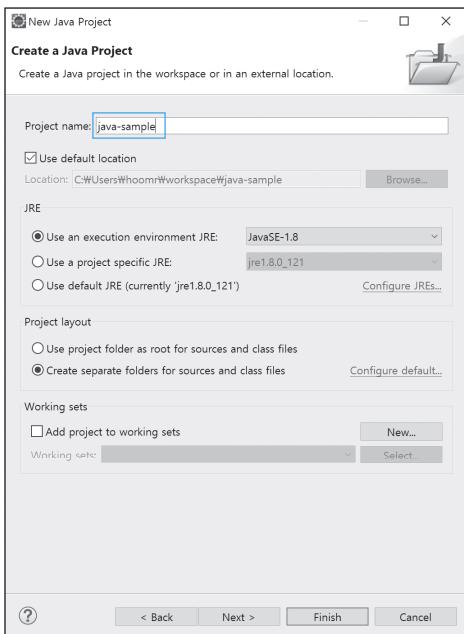
이클립스로 자바 프로그램을 개발하려면 우선 프로젝트를 생성해야 합니다. 다음 그림과 같이 메뉴에서 File → New → Java Project를 선택하거나 Ctrl+N을 누른 후 Java Project를 선택하면 New Java Project 마법사로 새 프로젝트를 생성할 수 있습니다.

그림 1-14 프로젝트 작성 메뉴 선택



New Java Project 창이 열리면 Project name에 'java-sample'이라고 입력하고(다른 설정은 그대로 둡니다) Finish 버튼을 누릅니다.

그림 1-15 프로젝트명 설정



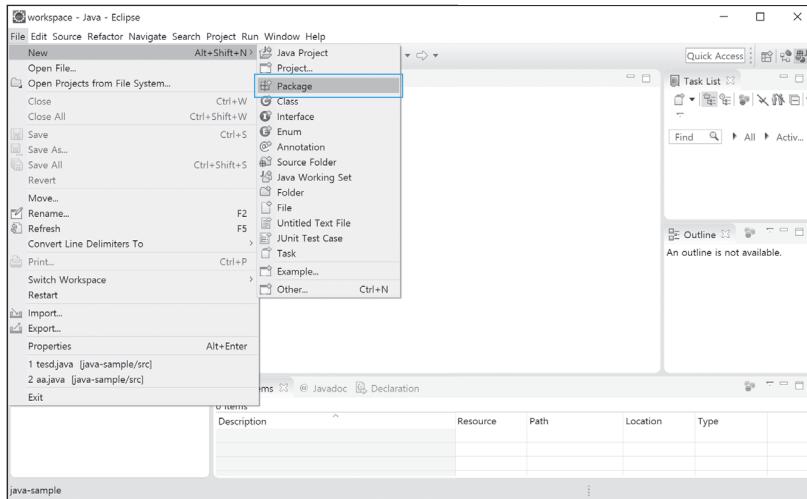
[5] 패키지 만들기

자바 애플리케이션을 작성할 때는 다른 자바 애플리케이션에 똑같은 이름의 클래스 파일이 있을 가능성이 크므로 서로의 클래스 파일을 구별하는 식별자로 패키지명을 사용합니다.

현장에서 업무 애플리케이션을 만들 때는 대부분 패키지로 클래스를 관리합니다. 패키지에 관한 자세한 설명은 나중에 하기로 하고, 여기서는 일단 패키지 만드는 방법을 경험해봅시다.

다음 그림과 같이 메뉴에서 File → New → Package를 선택하면 New Java Package 마법사로 새 패키지를 만들 수 있습니다.

그림 1-16 메뉴에서 New → Package 선택



[그림 1-17]과 같이 Name(패키지명)에 jp.co.bbreak.sokusen._1._2라고 입력합니다.

패키지를 생성했으면 이클립스를 시작할 때 지정한 워크스페이스에 있는 프로젝트의 src 폴더를 확인해보세요. src 폴더 아래에 jp\co\bbreak\sokusen_1_2라고 패키지명의 점(.)마다 계층화된 폴더가 만들어진 것을 확인할 수 있습니다. 이로써 패키지명은 루트가 되는 폴더에서 시작하는 폴더 계층을 나타낼 수 있습니다.

그림 1-17 패키지 작성



[6] 클래스 만들기

패키지를 만들었으면 다음은 자바 클래스를 만듭니다. 이클립스에서는 자바 클래스라고 하지만, 실제로는 .class 클래스 파일을 만들기 위한 .java 소스 파일이 만들어집니다.

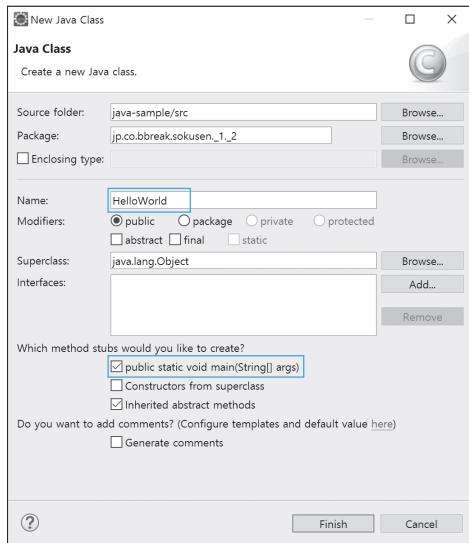
나중에 클래스에 관해 설명하겠지만, 여기서 말하는 클래스란 소스 파일에서 class로 선언한 것을 가리킵니다. 자바 애플리케이션은 클래스가 1개 이상 모여서 구축됩니다. 클래스란 자바 애플리케이션이 실제로 수행할 처리가 설정된 부품입니다.

확장자가 .class인 파일도 클래스로 불러 혼동하기 쉽지만 여기서는 소스 파일에서 class로 선언한 것을 ‘클래스’라 하고, 실제로 만들어진 확장자가 .class인 파일은 ‘클래스 파일’로 구별합니다.

그럼 자바 클래스를 만들어봅시다. 메뉴에서 File → New → Class를 선택하고 (Package Explorer의 빈 공간을 우클릭하면 나오는 콘텍스트 메뉴에서 New → Class를 선택해도 됩니다) New Java Class 마법사를 따라 새 클래스를 작성합니다.

다음 그림과 같이 Name에 ‘HelloWorld’라고 입력하고, **public static void main (String[] args)** 체크박스를 클릭하여 체크합니다. 나머지는 기본값 그대로 두고 Finish 버튼을 누릅니다.

그림 1-18 클래스명 지정



클래스가 만들어지면 클래스 소스 파일이 열립니다(예제 1-1). 소스 파일 첫 줄에 package jp.co.bbreak.sokusen._1._2;라고 패키지 선언이 추가되어 있습니다. 이 선언은 클래스가 어느 패키지에 속하는지 나타냅니다.

소스 파일에 [예제 1-1]의 굵은 글씨 부분을 추가합니다.

예제 1-1 HelloWorld.java

```
package jp.co.bbreak.sokusen._1._2;

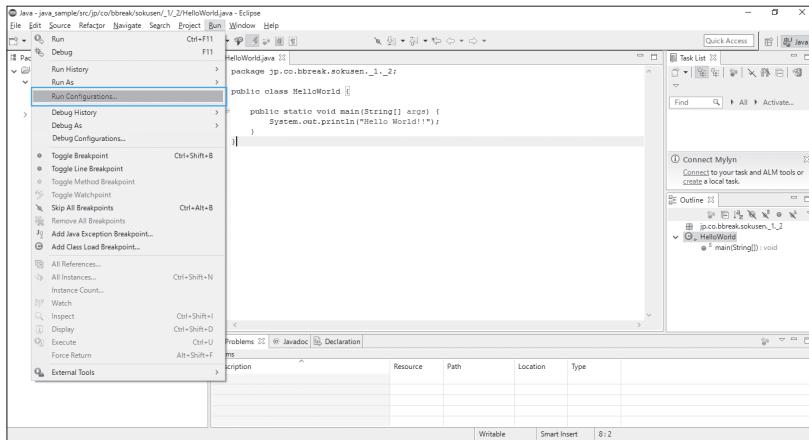
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

[7] 자바 애플리케이션 실행

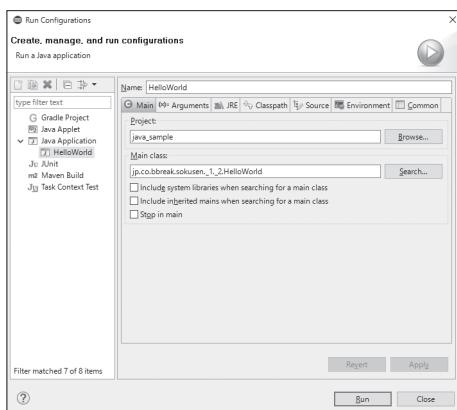
소스 파일을 완성했으면 자바 애플리케이션을 실행합니다. 자바 애플리케이션은 자바 클래스에 있는 `main` 메서드([예제 1-1]의 ① 부분)를 시작점으로 실행됩니다. 다음 그림과 같이 프로젝트를 선택하고, 메뉴에서 Run → Run Configurations...를 선택합니다.

그림 1-19 메뉴에서 Run → Run Configurations... 선택



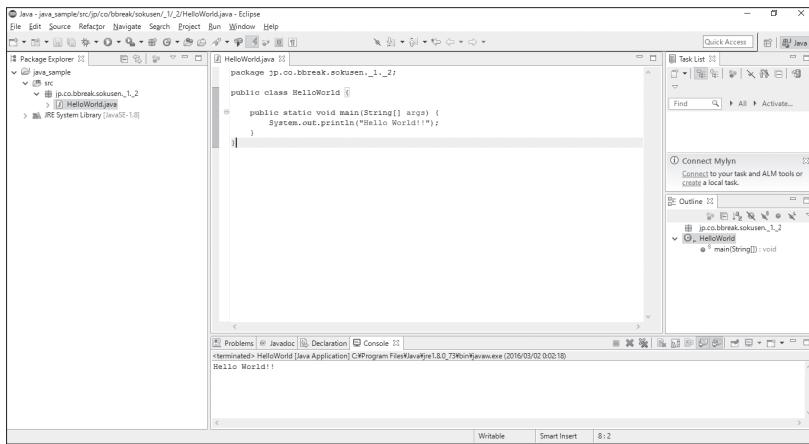
Run Configurations 창의 왼편에서 Java Application을 더블 클릭하면 지정된 클래스(`HelloWorld`)의 내용이 기술된 상태의 실행 구성(실행을 위한 설정)이 만들어집니다.

그림 1-20 실행 구성



메뉴에서 Run → Run을 선택해 다음 그림과 같이 콘솔 뷰에 처리 결과가 표시되면 예제를 올바로 따라한 것입니다.

그림 1-21 실행 결과



같은 자바 클래스를 실행하려면 녹색 Run 아이콘을 눌러 방금 작성한 HelloWorld를 선택합니다.

그림 1-22 Run 아이콘



1.3 자바의 기본

지금까지 자바로 개발하기 위한 개발 환경을 준비했습니다. 또한 환경이 제대로 구축됐는지 확인하기 위해 프로그램 언어 세계에서는 관례가 된 'Hello World!'를 표시하는 애플리케이션을 만들고 실행했습니다.

하지만 자바를 처음 공부하는 사람에게 이 소스 코드가 무엇을 나타내는지, 어떻게 해서 'Hello World!'를 표시하는지 궁금할 것입니다. 그러므로 이 절에서는 자바 프로그래밍의 기본 문법을 학습합니다.

1.3.1 기본 문법

자바에서는 소스 코드를 기술할 때 유의해야 할 몇 가지 규칙이 있습니다. 실제로 소스 코드를 읽거나 기술하기 전에 최소한 알아두는 편이 좋은 자바 형식을 살펴보겠습니다.

자바 형식

자바 소스 코드는 주로 다음 3가지 요소로 기술합니다.

- 자바에서 지정한 키워드
- 소스 코드 작성자가 직접 붙인 이름(식별자)
- 뭔가 역할을 가진 기호(연산자 등)

자바에서 지정한 키워드를 **예약어**라고 부릅니다. 예약어는 컴파일 시 프로그램을 분석하는 데 사용되는 단어 또는 `true`, `false`, `null` 등 이미 그 단어 자체에 프로그램 상의 의미가 있는 값을 나타냅니다. 그러므로 예약어와 똑같은 단어는 식별자로 지정할 수 없습니다.

식별자란 소스 코드를 기술하는 사람이 붙인 이름을 말합니다. 예를 들어 프로그램에서 ‘○○가 △△를 □□한다’와 같은 처리를 기술할 때 이들이 각각 무엇을 나타내는지 그리고 무엇을 하는지 등 그 의미를 사람이 읽고 이해할 수 있게 프로그램 언어에서 허용된 문자로 붙인 이름입니다.

또한 계산과 조건 판정 등에 사용하는 기호를 **연산자**라고 합니다. 따라서 기호는 _와 \$를 제외하고 식별자로 사용할 수 없습니다.

키워드, 식별자, 연산자를 자바 문법에 따라 서로 연결해 자바의 문(statement)을 만듭니다. 다시 몇 개의 문을 모아 블록을 만들고 자바 애플리케이션을 만듭니다.

문과 블록

자바 소스 코드는 문과 복수의 문을 모은 블록으로 구성됩니다.

문에서는 단어나 기호를 조합해 값을 설정하거나 처리를 호출합니다. 그리고 문의 맨 끝에는 세미콜론(;)을 붙입니다.

블록은 복수의 문을 묶어 순서대로 배치함으로써 하나의 큰 처리 흐름이나 문의 영향 범위를 나타냅니다. {와 }로 에워싼 범위가 블록입니다. 예를 들어 ‘사람’이라는 블록을 다음처럼 설정해봅시다.

- ‘이름’과 ‘취미’가 설정된 문
- ‘자기소개를 한다’와 ‘인사를 한다’를 나타내는 처리 블록

다음은 설정 예입니다.

예제 1-2 블록과 문

사람 {

 이름은 'OO';
 취미는 'XX';

 자기소개를 한다 {

 인사를 한다;
 이름을 말한다;
 취미를 말한다;

}

 인사를 한다 {

 인사말은 '안녕하세요';
 인사말을 한다;

}

}

블록 내의 문에는 각각 유효 범위가 설정되어 있습니다. 블록 내에서는 같은 계층 혹은 블록 밖에 있는 문이나 블록을 참조할 수 있지만, 블록 밖에서는 블록 내에 있는 문을 참조할 수 없습니다(그림 1-23).

그림 1-23 블록의 유효 범위

'사람'이 관리하는 문과 블록

- 이름
- 취미
- 자기소개를 한다
- 인사를 한다

'자기소개를 한다'가 관리하는 문. '사람'이 관리하는 아래 항목에 액세스 가능

- 인사를 한다
- 이름
- 취미

'인사말'은 '인사를 한다' 블록 내에서만 유효하므로 '사람'이나 '자기소개를 한다'에서 액세스 불가능

사람 {

 이름은 'OO';
 취미는 'XX';

자기소개를 한다 {

 인사를 한다;
 이름을 말한다;
 취미를 말한다;

} ✘

인사를 한다 {

 → 인사말은 '안녕하세요';
 인사말을 한다;

}

'자기소개를 한다'에서
이름, 취미에 액세스 가능'자기소개를 한다'에서
'인사를 한다'에 액세스 가능

소스 코드 구조

자바의 소스 코드는 단순하게 보면 다음과 같은 구조로 기술합니다.

- 소스 코드가 무엇을 나타내는지 선언
- 선언 블록 내에 값을 설정하기 위한 변수 정의
- 선언 블록 내에 변수값을 사용한 처리를 기술

변수란 프로그램 안에서 값을 전달하기 위한 상자입니다. 프로그램 안에서 계산을 하거나 가공한 값을 임시로 보관하는 데 사용합니다. 변수 선언이나 처리를 기술하지 않아도 소스 코드는 성립하지만 우선은 이 3가지로 구성되는 것이 기본입니다. 그리고 이번에 만든 '사람' 예제를 이 3가지 요소로 예를 들면 [표 1-2]와 같습니다.

표 1-2 사람 예제의 구성 요소

구성 요소	예제에서의 대상
소스 코드가 나타내는 것 선언	사람
변수(값을 가진 것) 선언	이름, 취미, 인사말
처리를 기술	자기소개를 한다, 인사를 한다

클래스 선언

자바에서는 소스 코드로 나타내고자 하는 것이 무엇인지 블록으로 감싸서 선언합니다. 구체적으로는 다음과 같은 것을 선언합니다.

- 클래스 `class`
- 인터페이스 `interface`
- 열거형 `enum`

여기서는 이 중에서 가장 자주 사용하는 클래스를 만드는 방법을 설명하겠습니다. 클래스를 학습하고 나면 인터페이스와 열거형도 이해하기 쉬워질 것입니다.

클래스는 다음 구문으로 선언합니다.

[구문] 클래스 선언

```
class 클래스명 {  
    ... 클래스 내용 ...  
}
```

클래스를 선언하는 블록에서는 `class` 키워드를 선두에 놓고 공백으로 구분한 뒤에 클래스명을 지정합니다.

이렇게 선언된 `class` 블록 아래에 클래스가 사용할 변수나 처리를 기술함으로써 클래스의 성질과 동작을 연결시킬 수 있습니다. 좀 전에 예로 든 ‘사람’ 클래스에는 ‘이름’과 ‘취미’라는 변수가 있었고, 처리로는 ‘자기소개를 한다’, ‘인사를 한다’라는 기능이 있었습니다.

변수 설정

자바 애플리케이션은 변수로 값을 유지합니다. 변수에 값을 설정하려면 `=` 기호를 사용하며, `=` 기호 왼쪽에는 변수명을, 오른쪽에는 설정할 값을 씁니다. 이것을 대입(혹은 할당)이라고 합니다.

다음은 변수를 선언하는 구문입니다.

[구문] 변수 선언(값을 대입하는 경우)

```
변수형 변수명 = 값;
```

변수형이란 그 변수에 어떤 값을 담을 수 있는지 나타내는 정보로, 수치, 문자, 클래스 등을 지정할 수 있습니다. 변수에는 선언한 자료형의 값만 설정할 수 있습니다.

변수에 값을 설정한 다음에는 변수를 지정해서 값을 가져올 수 있습니다.

다음과 같이 값을 대입하지 않고 변수를 선언할 수도 있습니다.

[구문] 변수 선언(값을 대입하지 않는 경우)

변수형 변수명:

이 경우 변수형의 기본값(아무것도 설정하지 않으면 자동으로 설정되는 정해진 값)이 대입됩니다.

좀 전에 만든 클래스를 예로 들면 ‘이름’과 ‘취미’에는 문자열²을 설정하므로 자바의 문자열 클래스인 `String`으로 형을 지정합니다. 문자열 값을 나타내려면 값의 앞뒤를 큰따옴표(")로 에워싸야 합니다.

위 설명을 바탕으로 ‘이름’ 변수를 다음과 같이 선언할 수 있습니다.

예제 1-3 ‘이름’ 변수 설정

`String 이름 = "oo";`

변수형은 처음에 변수명을 선언할 때만 지정하면 되고, 값을 변경할 때는 다음처럼 쓸 수 있습니다.

[구문] 변수의 값 변경

변수명 = 값;

값 부분에 계산식 등 어떤 처리를 넣은 경우 변수에는 처리 결과가 값으로 설정됩니다. 예를 들어 $1+1$ 의 계산 결과를 대입하고 싶을 때는 다음과 같이 기술합니다. 변수 형은 수치형 중 하나인 `int`로 합니다.

2 프로그래밍 세계에서는 한 글자만 나타낼 수 있는 형을 ‘문자’라고 하고, 몇 글자든 나타낼 수 있는 형을 ‘문자열’이라고 합니다.

예제 1-4 계산 결과를 수치형 int에 저장한다.

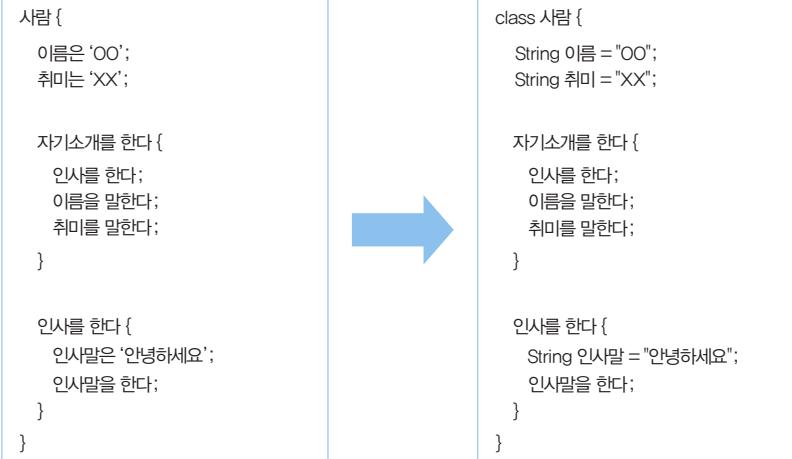
```
int 계산_결과 = 1 + 1;
```

지금까지 이해한 내용을 다시 ‘사람’ 클래스로 표현하면 다음을 문자열 변수로 지정 할 수 있습니다.

- 이름
- 취미
- ‘인사를 한다’ 안에 있는 인사말

이 문자열 변수를 자바 문법에 맞게 다시 쓰면 다음과 같이 됩니다.

그림 1-24 변수 부분을 자바 문법에 맞게 수정



예제에서 ‘이름’과 ‘취미’는 클래스의 상태를 나타내는 변수입니다. 이처럼 클래스의 블록에서 직접 선언한 변수를 필드 field라고 합니다.

메서드 기술

자바 애플리케이션에는 어떤 처리가 포함됩니다. 처리에는 1+1처럼 처리를 나타내는 기호를 사용해 표현하는 식 expression과 몇 개의 처리를 묶은 메서드 method가 있습니다.

2

장

프로그래밍 기초

2.1 문자열 조작

문자열 조작이란 문자 덩어리를 하나의 데이터로 다루는 처리를 말합니다. 문자열은 우리가 다루는 데이터 중에서 가장 기본이 됩니다. 프로그램을 작성할 때는 문자열을 나누거나 연결하거나 검색하는 등 다양한 방식으로 사용합니다. 여기서는 문자열을 다루는 방법을 학습합니다.

2.1.1 문자열 기초 지식

자바에서는 문자열 데이터를 `String` 클래스로 다루는 것이 기본입니다. 기본형인 `char`를 이용해 1문자씩 다룰 수도 있지만 이 방법은 불편해서 그다지 사용하지 않습니다. 일반적으로 문자열 조작에는 `String`형을 사용합시다.

`String` 클래스는 일반 클래스와 구별되어 특별 취급을 받습니다(자바 언어 명세에도 특별하게 다룬다고 명시되어 있습니다). 예를 들어 `new`를 사용하지 않고도 인스턴스화할 수 있습니다. `String` 객체를 생성하기 위해선 다음처럼 쓰기만 하면 됩니다.

```
String moonzayol = "이것은 String 클래스입니다.;"
```

물론 다른 클래스처럼 `new` 연산자로 선언할 수도 있습니다.

```
String moonzayol = new String("이것은 String 클래스입니다.");
```

하지만 이렇게 선언하는 것은 별로 의미도 없고 이중으로 인스턴스를 만들게 되므로 낭비입니다. 따라서 `String` 객체를 생성할 때는 첫 번째 방식을 사용하도록 합시다.

문자열을 다룰 때 또 한 가지 주의해야 할 점은 `String` 클래스는 인스턴스의 데이터가 변하지 않는 불변^{immutable} 클래스라는 점입니다. `String`형 데이터에 어떤 변경을 가하면 인스턴스의 문자열 데이터가 변하는 것이 아니라 인스턴스를 새로 생성합니다.

NOTE 불변이란?

생성된 인스턴스의 내용이 절대로 변하지 않는 클래스를 **불변 클래스**라고 합니다. 불변 클래스의 내용을 변경한다는 것은 인스턴스를 다시 생성한다는 것을 뜻합니다. 인스턴스를 생성할 때는 메모리 공간과 처리 능력을 소비하게 됩니다. 그러므로 불변 클래스를 이용할 때는 무턱대고 내용을 변경하지 않는지(인스턴스를 재생성하지 않는지) 주의해야 합니다.

2.1.2 문자열 연결

문자열의 기초를 이해했으니 기본 중의 기본인 문자열 연결부터 해봅시다.

예제 2-1 TextSample.java

```
package jp.co.bbreak.sokusen._2._1;

public class TextSample {
    public static void main(String[] args) {
        String text1 = "정말 ";
        String text2 = "감사합니다.";
        text1 = text1 + text2;
        System.out.println(text1);
    }
}
```

이 코드를 실행하면 ‘정말 감사합니다.’라는 문자열이 화면에 한 줄로 출력됩니다. 이처럼 문자열을 + 연산자로 연결해 2개의 문자열을 하나로 만들 수 있습니다.

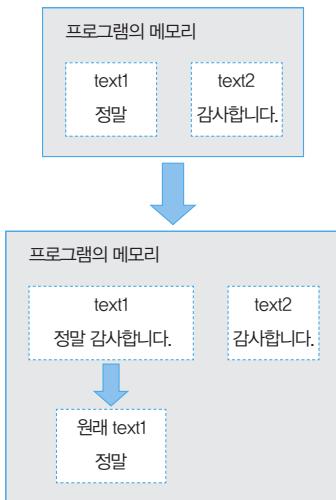
하지만 + 연산자를 사용하는 것은 그다지 좋은 방법이 아닙니다. 왜 그럴까요?

String 클래스는 영원히 변하지 않는다

그 이유는 `String`이 불변 클래스라는 것과 관련이 있습니다. 좀 전의 예에서는 `String` 클래스인 2개의 문자열을 + 연산자로 결합했습니다. 이때 문자열이 연결된다는 것에만 주목하기 십상이지만 사실은 프로그램 내부에서는 `text1`과 `text2` 그리고 결합 후의 `text1`이라는 3개의 `String` 인스턴스가 생성됩니다. 원래의 `text1` 인스턴스는 아무에게도 보이지 않는 상태지만 여전히 메모리상에 존재합니다.

무슨 말인지 잘 이해가 안 될 수도 있습니다. 다음 그림을 보세요.

그림 2-1 String 클래스의 메모리 상태 변화



`text1`과 `text2`를 연결할 때 `text1` 인스턴스에 문자열이 추가되는 것이 아닙니다. 그 이유는 지금까지 설명한 대로 `String`은 불변 클래스이기 때문입니다.

문자열을 연결할 때는 새로운 인스턴스가 만들어지고, 원래 참조하던 `text1` 인스턴스에서 새로운 인스턴스로 교체됩니다. 이전 `text1` 인스턴스는 어디에서도 참조되지 않는 상태지만 곧바로 메모리에서 없어지는 것은 아닙니다.

이번 사례처럼 간단한 경우는 아무 문제도 없지만 루프 안에서 문자열을 몇 번이고 연결하게 되면 반복할 때마다 인스턴스를 새로 만듭니다. 즉, 사용하지 않는 인스턴스가 계속 쌓여갑니다. 인스턴스 생성은 비용이 드는 처리라서, 티끌 모아 태산이라고 프로그램의 동작이 느려집니다. 심하면 메모리 자원을 고갈시켜 결국 프로그램이 멈출 수도 있습니다.

그럼 어떻게 하면 효율적으로 문자열을 연결할 수 있을까요?

NOTE_ 가비지 컬렉션

가비지 컬렉션이란 자동으로 메모리를 관리하는 기능입니다.

프로그래밍 언어에 따라서는 메모리 관리를 프로그래머가 직접 해야 하는 경우도 있지만 자바에서는 필요 없어진 메모리를 가비지 컬렉션으로 자동 해제합니다. 해제 시점은 자바 실행 환경이 결정 하므로 프로그래머는 메모리를 관리하지 않아도 된다는 장점이 있습니다.

그렇다고 해서 완전히 메모리를 고려하지 않아도 된다는 건 아닙니다. 처음부터 낭비되는 메모리가 없도록 고려해 프로그램을 작성하는 것이 프로그래머가 갖추어야 하는 자세입니다.

문자열 연결에는 **StringBuilder**를 사용하자

+ 연산자 대신 문자열 연결에 이용할 수 있는 클래스가 **StringBuilder**입니다.

StringBuilder 클래스는 **append** 메서드로 문자열을 축적하고, 마지막에 **toString** 메서드로 모아둔 문자열을 출력합니다. 좀 전의 예를 **StringBuilder**로 구현해봅시다.

예제 2-2 **StringBuilderSample.java**

```
package jp.co.bbreak.sokusen._2._1;

public class StringBuilderSample {
    public static void main(String[] args) {
        String text1 = "정말 ";
        String text2 = "감사합니다.";
        StringBuilder sb = new StringBuilder();
        sb.append(text1);
        sb.append(text2);
        String resultString = sb.toString();
        System.out.println(resultString);
    }
}
```

+ 연산자를 이용하면 문자열을 연결할 때마다 **String** 인스턴스가 만들어지지만 **StringBuilder**를 이용하면 인스턴스 내부에서 문자열을 연결할 수 있습니다(일일이 인스턴스가 생성되지 않습니다). 따라서 연결하고 싶은 문자열이 늘어나도 사용하는 메모리는 일정하게 억제할 수 있습니다.

또한 자바 8에는 **StringJoiner**라는 클래스도 추가됐습니다. **StringBuilder**보다 편리하므로 흥미가 있는 사람은 사용법을 알아보면 좋을 것입니다.

2.1.3 문자열의 형식

String 클래스는 불변이므로 가능한 한 효율적으로 이용하는 방법을 알아두는 것이 중요합니다. 여기서는 문자열에 형식을 지정하는 방법을 소개합니다.

문자열의 형식이란 어떤 것일까요? 예를 들어 가게에서 물건을 구입할 때 받는 영수증을 떠올려보세요. 영수증에는 다음과 같이 구매한 물건과 금액이 적혀 있습니다.

[영수증 상세 내역의 예]

쌀 5kg	15,000원
-------	---------

이런 문자열을 출력하려면 어떻게 하면 좋을까요?

영수증에서 한 줄의 데이터를 다음과 같이 클래스로 관리하고, 출력할 때는 List 객체(뒤에 설명)로 둑기로 하겠습니다.

예제 2-3 Detail.java

```
package jp.co.bbreak.sokusen._2._1;

import java.math.BigDecimal;

public class Detail {
    // 상품명
    private String itemName;
    // 금액
    private BigDecimal amount;

    // 게터와 세터는 생략
}
```

느낌으로는 상품명, 공백, 금액, 그리고 마지막에 ‘원’을 연결하면 될 것처럼 보입니다. 좀 전에 알아본 StringBuilder를 사용할 수 있을 것 같습니다.

예제 2-4 PrintReceipt1.java

```
package jp.co.bbreak.sokusen._2._1;

import java.math.BigDecimal;

public class PrintReceipt1 {
```

```

public static void main(String[] args) {
    // 구매 내역 데이터 작성
    Detail detail = new Detail();
    detail.setItemName("쌀5kg");
    detail.setAmount(new BigDecimal(15000));

    // 구매 내역 데이터 가공
    StringBuilder sb = new StringBuilder();
    sb.append(detail.getItemName());
    sb.append(" ");
    sb.append(detail.getAmount());
    sb.append("원");

    // 구매 내역 출력
    System.out.println(sb.toString());
}
}

```

이 코드를 실행하면 좀 전의 ‘영수증 상세 내역의 예’와 같은 결과가 출력됩니다. 단, 이 출력 방법에는 몇 가지 문제가 있습니다. 자신의 지갑에 있는 영수증을 꺼내서 모든 가능성을 상상해보세요.

아마도 다양한 의견이 나올 것으로 생각합니다. 몇 가지 가능성을 다음에 나열해보았습니다.

- 금액에 따라서는 위치가 어긋나서 엉성해 보인다.
- 상품명이 길면 점점 오른쪽으로 밀려 금액이 인쇄되지 않는다.
- 금액이 3자리마다 콤마로 구분되지 않는다.
- 여러 내역을 출력할 수 없다.
- 합계 행이 있어야 한다.
- 가게 이름과 전화번호가 없어 불친절하다.

이외에도 여러 의견이 있을 것입니다. 단, 이번에는 상세 내역 한 줄만 생각하기로 했으니 앞에 나열한 의견 중 처음 3개를 어떻게 표현할지 생각해봅시다.

문자열의 포맷

일정한 틀에 따라 문자열의 형식을 바꾸는 것을 포맷이라고 합니다. 여기서는 다음 규칙을 따라 포맷합니다.

- 상세 내역 1행 전체가 20칸
- 상품명과 금액이 각각 10칸씩 사용
- 상품명은 왼쪽 맞춤, 금액은 오른쪽 맞춤

또한 제대로 표시됐는지 확인하기 위해 상세 내역은 2개로 합니다. 그런데 이 규칙을 `StringBuilder`로 구현하기에는 조금 힘들어 보입니다.

그럼 어떻게 해야 할까요? 이런 경우에는 `String` 클래스의 정적 메서드 `format`을 사용하면 됩니다. `format` 메서드는 기본 틀이 되는 문자열을 첫 번째 인수로 설정하고, 두 번째 인수 이후에 지정된 값을 차례대로 서식 위치에 채워가는 기능을 제공합니다. 기본 틀은 미리 정해진 서식(작성 규칙)에 따라 기술해야 합니다. 지정하는 규칙은 다양하지만 기본은 [표 2-1]과 같이 2가지입니다.

표 2-1 `String` 클래스의 `format` 메서드에서 사용하는 서식의 종류

서식	의미
%s	여기에 문자열이 들어간다.
%d	여기에 수치가 들어간다.

또한 %와 문자 사이에 [표 2-2]의 서식을 추가해 더욱 세밀하게 서식을 지정할 수 있습니다.

표 2-2 `String` 클래스의 `format` 메서드의 상세 서식 지정 방법

서식	의미
%10s	문자열을 오른쪽 정렬하고 10자리로 한다. 남는 자리엔 공백이 채워진다.
%-10s	문자열을 왼쪽 정렬하고 10자리로 한다. 남는 자리엔 공백이 채워진다.

서식을 지정하면 영수증 내역의 한 줄을 정렬된 형태로 출력할 수 있습니다. 스스로 영수증의 한 줄을 표현할 수 있는 서식을 생각하고 다음 소스를 작성해봅시다. 지난 번과 같지만, 원하는 대로 정렬됐는지 확인하기 위해 2줄을 출력하느라 조금 길어졌습니다.

예제 2-5 PrintReceipt2.java

```
package jp.co.bbreak.sokusen._2._1;
```

```

import java.math.BigDecimal;

public class PrintReceipt2 {

    public static void main(String[] args) {
        // 구매 내역 데이터 작성
        Detail detail1 = new Detail();
        detail1.setItemName("쌀5kg");
        detail1.setAmount(new BigDecimal(15000));
        Detail detail2 = new Detail();
        detail2.setItemName("김9개");
        detail2.setAmount(new BigDecimal(9000));

        // 구매 내역 문자열 서식 정의
        String lineBase = "%-10s%10d원";

        // 문자열 데이터 가공
        String result1 = String.format(lineBase, detail1.getItemName(),
            detail1.getAmount().longValue());
        String result2 = String.format(lineBase, detail2.getItemName(),
            detail2.getAmount().longValue());

        // 구매 내역 출력
        System.out.println(result1);
        System.out.println(result2);
    }
}

```

어떤가요? 정렬된 형태로 영수증 내역이 2줄 표시됐을 것입니다.

‘어? 줄이 안 맞는데?’라는 목소리가 들리네요.

화면에 출력되는 글꼴의 폭은 실행 환경에 따라 다르므로 때에 따라서는 줄이 맞지 않는 사람도 있을 것입니다. 그런 사람은 일단 화면의 실행 결과를 복사해 에디터(메모장 등)에 붙여넣어 보세요. 그렇게 하면 제대로 맞게 출력됐음을 알 수 있습니다.

여기서는 최소한의 서식만 소개했습니다. 상세한 지정 방법을 알고자 하는 독자는 Javadoc¹의 `String` 클래스의 `format` 메서드 항목을 보면 서식 문자열을 소개하는 페이지 링크가 있으므로 읽어보세요.

1 자바에서 제공하는 표준 클래스에 관한 도큐먼트입니다. 왼쪽 아래의 클래스 목록에서 원하는 클래스를 선택하면 오른쪽 프레임에 해당 클래스에 관한 자세한 정보가 표시됩니다. <http://docs.oracle.com/javase/8/docs/api/index.html>을 참고하세요.

수치의 포맷

영수증에 구매 내역이 잘 출력되긴 했지만, 한 가지 문제가 아직 해결되지 않았습니다. 그렇습니다. 금액이 3자리마다 콤마로 구분되지 않는다는 문제가 여전히 남아 있습니다.

여기까지 공부한 분이라면 이 또한 `format` 메서드로 구현할 수 있을 것 같다는 생각이 들 것입니다. 어떤 서식을 지정하면 좋을지 우선 스스로 조사해서 구현 방법을 모색해보세요.

자, 방법을 찾으셨나요?

정답은 다음과 같습니다.

예제 2-6 수정 후의 서식

```
String lineBase = "%-10s%,10s원";
```

% 뒤에 콤마를 입력하면 3자리마다 콤마로 구분해줍니다. 간단하지요? 하지만 이것만으로 이 절을 마칠 순 없습니다.

같은 처리를 구현하는 데도 몇 가지 방법이 있는 법입니다. 프로그램에 따라서는 전체 문장을 한번에 만들지 않고, 수치만 변환해두고 다른 처리로 넘기는 경우도 있습니다. 또한 표준 서식 이외의 포맷을 구현하고 싶을 때는 `String`의 `format` 메서드로는 어려운 경우도 있습니다.

이러한 경우에 안성맞춤인 클래스가 `DecimalFormat`입니다. `DecimalFormat`은 수치를 서식에 따라 문자로 변환하는 클래스입니다. `DecimalFormat`의 서식은 `String` 클래스의 `format` 메서드에서 사용하는 것과는 다릅니다(표 2-3).

표 2-3 DecimalFormat의 서식

서식	의미
0	여기에 수치가 들어간다(설정한 자릿수에 모자라면 0으로 채운다).
#	여기에 수치가 들어간다(설정한 자릿수에 모자라도 아무것도 들어가지 않는다).
,	해당 위치에 콤마가 그대로 들어간다.
.	해당 위치에 도트가 그대로 들어간다.

`DecimalFormat`의 서식은 보이는 그대로이므로 직관적입니다. 예를 들어 다음과 같이 서식을 2개 정의합시다.

예제 2-7 DecimalFormat 서식 예

```
###,##0 // 서식 1  
000,000 // 서식 2
```

i) 포맷에 몇 가지 수치를 넘겨주면 [표 2-4]처럼 됩니다.

표 2-4 DecimalFormat 실행 예

입력값	서식 1의 결과	서식 2의 결과
12	12	000,012
1234	1,234	001,234
1234568	1,234,567	1,234,567

왜 이런 결과가 나왔는지 살펴봅시다.

서식 1은 첫 번째 자리는 0이지만 그 이상은 #이고, 3번째와 4번째 자리 사이에 콤마가 들어 있습니다. 따라서 3자리 이하는 콤마가 표시되지 않습니다. 일반적으로 금액은 3자리씩 콤마를 넣는데, 서식에서 처음에 오른쪽부터 3자리로 구분하면 그 이후도 3자리마다 구분해줍니다.

서식 2는 모든 자리가 0이므로 결과의 길이가 일정합니다. 입력값이 서식보다 모자란 부분은 0으로 채워줍니다. 또한 자릿수가 넘치는 부분도 출력됩니다.

어떠셨나요?

`DecimalFormat` 서식은 간단해서 바로 이해할 수 있을 것입니다. 그럼 이 클래스를 사용해서 좀 전의 예제에서 콤마로 자릿수가 구분된 금액을 출력해봅시다.

정답은 아래에 있지만 우선 스스로 생각해서 수정해보세요.

예제 2-8 PrintReceipt3.java

```
package jp.co.bbreak.sokusen._2._1;  
  
import java.math.BigDecimal;
```

```

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.List;

public class PrintReceipt3 {

    public static void main(String[] args) {
        // 구매 내역 데이터 작성
        Detail detail1 = new Detail();
        detail1.setItemName("쌀5kg");
        detail1.setAmount(new BigDecimal(15000));
        Detail detail2 = new Detail();
        detail2.setItemName("감9개");
        detail2.setAmount(new BigDecimal(9000));

        List<Detail> detailList = new ArrayList<>();
        detailList.add(detail1);
        detailList.add(detail2);

        // 구매 내역 문자열 서식 정의
        String lineBase = "%-10s%10s원"; // 포맷 문자 수정

        // 금액 표시용 서식 정의
        DecimalFormat df = new DecimalFormat("###,##0"); // 추가

        // 금액을 표시용으로 가공
        String dispAmount1 = df.format(detail1.getAmount().longValue());
        String dispAmount2 = df.format(detail2.getAmount().longValue());

        // 변환한 금액을 표시하도록 수정
        String result1 = String.format(lineBase, detail1.getItemName(), dispAmount1);
        String result2 = String.format(lineBase, detail2.getItemName(), dispAmount2);

        // 구매 내역 출력
        System.out.println(result1);
        System.out.println(result2);
    }
}

```

자, 자신의 생각대로 동작하나요?

화면에 다음처럼 출력됐다면 성공입니다.

실행 결과

쌀5kg	15.000원
감9개	9.000원

이것으로 문자열의 기초 지식 설명을 마칩니다. 이 절에서 학습한 내용은 다음과 같습니다.

- 문자열은 불변한다.
- 문자열의 연결은 `StringBuilder`를 사용한다.
- 문자열 서식은 `String` 클래스의 `format` 메서드를 사용한다.
- 수치 서식은 `DecimalFormat`이 편리하다.

이상의 내용이 이해되셨나요?

그 밖에도 문자열을 가공하는 방법은 여러 가지입니다. 예를 들어 파일 전체의 문자열을 포맷하기 위한 편리한 라이브러리인 Apache Velocity 등이 대표적입니다.

다음 절에서는 날짜와 시간을 조작해봅시다.

2.2 날짜 및 시간 조작

시스템에서 다루는 대표적인 데이터는 날짜와 시간입니다. 일반적으로 시간 정보는 ‘언제’ 무슨 일이 일어났는지 기록하기 위해 사용합니다. 또한 기록한 정보를 기간으로 추출할 때의 기준으로 사용합니다.

구체적으로 이야기해봅시다.

문자열 조작에서 예로 든 영수증을 떠올려보세요. 영수증에는 상품을 언제 구매했는지도 적혀 있습니다. 편의점이나 슈퍼마켓에서는 영수증에 기재된 정보를 데이터로 보존하는 것이 일반적입니다. 그 정보는 ‘언제 어떤 상품이 잘 팔리는 경향이 있는가’와 같은 경영 분석에 사용합니다. 그럼 이번에는 날짜와 시간 데이터를 어떻게 다루는지 학습합시다.

2.2.1 날짜 및 시간을 다루는 클래스

날짜 및 시간을 다루는 대표적인 클래스를 [표 2-5]에 나타냈습니다.

표 2-5 날짜와 시간을 다루는 데 사용하는 대표적인 클래스

클래스	용도
java.util.Date	특정 날짜를 저장한다.
java.util.Calendar	일시에 대한 다양한 조작을 한다.
java.text.SimpleDateFormat	지정한 서식에 따라 문자열을 Date 클래스로 변환하거나, 그 반대로 처리한다.

이 클래스들을 사용해 날짜와 시간을 다루는 방법을 배워봅시다.

2.2.2 현재 날짜를 이용한 데이터 조작

Date 클래스를 이용해 현재 날짜와 시간을 가져와서 콘솔 화면에 출력해봅시다. 다음과 같이 Date 클래스의 인스턴스를 생성하면 그 시점의 날짜와 시간이 인스턴스에 저장됩니다.

예제 2-9 DateSample.java

```
package jp.co.bbreak.sokusen._2._2;

import java.util.Date;

public class DateSample {
    public static void main(String[] args) {
        Date now = new Date();
        System.out.println(now);
    }
}
```

이 프로그램을 실행하면 콘솔에 날짜와 시간이 표시됩니다.

실행 결과

```
Thu Mar 09 19:41:57 KST 2017
```

다만 이런 식으로 사용하는 것은 시스템 날짜를 가져올 때 정도뿐입니다.

3

장

데이터베이스

3.1 데이터베이스 기초

여기서는 자바에서 조금 벗어나 데이터베이스에 관해 설명합니다. 데이터베이스란 데이터를 보존하는 창고입니다. 우리가 만드는 프로그램은 대부분 데이터베이스를 조작하기 위해 존재한다고 해도 과언이 아닙니다. 시스템 전체가 열악해도 데이터베이스의 데이터에 문제가 없으면 나중에 복구할 수 있습니다. 그만큼 데이터베이스 설계와 조작은 매우 중요합니다.

현재 업무 애플리케이션에 사용하는 데이터베이스의 주류는 관계형 데이터베이스 Relational Database입니다. 그런 이유로 이 책에서도 관계형 데이터베이스만을 대상으로 설명합니다.

이 장에서는 데이터베이스 기초 지식부터 시작해 자바로 데이터베이스를 어떻게 조작하는지까지 설명합니다.

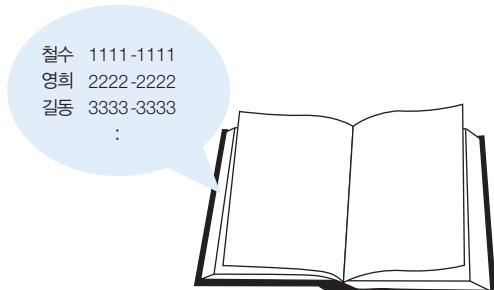
3.1.1 데이터베이스란 무엇인가

데이터베이스를 사용한 애플리케이션을 만들기 전에 ‘데이터베이스란 무엇인지’부터 간단히 설명하겠습니다.

데이터베이스란 데이터의 집합이다

데이터베이스란 특정한 주제로 데이터를 모아둔 것입니다. 주변에서 볼 수 있는 데이터베이스의 예로는 전화번호부를 들 수 있습니다. 전화번호부에는 다음 그림과 같이 전화번호와 이름 등이 실려 있습니다. 전화번호를 주제로 한 데이터의 집합이므로 어엿한 데이터베이스라고 할 수 있지요.

그림 3-1 전화번호부도 데이터베이스의 일종



데이터를 관리하는 애플리케이션

일반적으로 데이터베이스는 단독으로 사용되지 않습니다. 일반적으로 데이터를 관리하고 조작하는 애플리케이션과 함께 사용됩니다. 애플리케이션을 통해 데이터를 등록하거나 검색한 결과를 가공해서 뽑아내기도 합니다. 이처럼 데이터베이스를 관리하는 애플리케이션을 **데이터베이스 관리 시스템 Database Management System**이라고 합니다. 데이터베이스 관리 시스템에는 일반적으로 [표 3-1]과 같은 기능이 구현되어 있습니다.

표 3-1 데이터베이스 관리 시스템이 제공하는 기능

기능	설명
데이터 검색 기능	데이터베이스에서 원하는 데이터를 선택한다.
데이터 추가, 갱신 기능	데이터베이스에 임의의 데이터를 추가 또는 갱신한다.
추가, 갱신 데이터 확인 기능	추가 또는 갱신하려고 한 데이터가 데이터베이스의 규칙을 지키는지 확인한다.
보안 관리 기능	데이터베이스에 접근할 수 있는 사용자를 제한한다.
동시 접근 관리 기능	복수의 사용자가 데이터베이스에 접근할 때의 동작을 제어한다.
데이터베이스 크기 관리 기능	데이터베이스에 저장할 수 있는 데이터양을 관리한다.

데이터베이스와 데이터베이스 관리 시스템은 대부분 한 세트로 다루므로, 이 2가지를 한꺼번에 데이터베이스라고 부르기도 합니다.

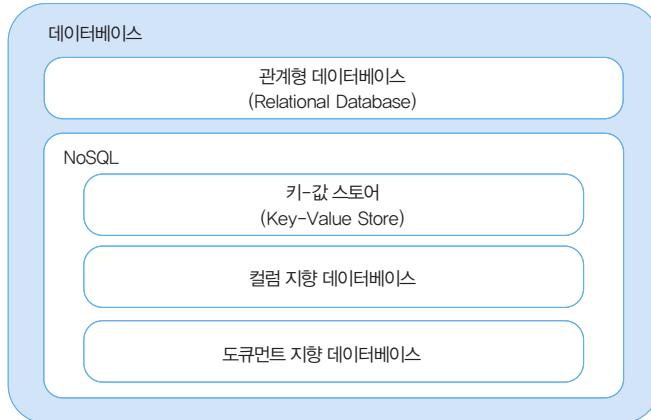
3.1.2 데이터베이스의 종류와 특징

데이터베이스는 데이터 저장 방식이나 추출 방식 등의 차이에 따라 몇 가지 종류가 있습니다. 이 절에서는 이들 데이터베이스가 데이터를 관리하는 방법을 알아봅시다.

데이터베이스 종류

현재 업무 시스템의 주류는 관계형 데이터베이스라고 불리는 방식을 사용합니다. 물론 용도에 따라 다른 방식의 데이터베이스가 사용되기도 합니다. 다음 그림은 관계형 데이터베이스와 그 외의 데이터베이스를 보여줍니다.

그림 3-2 관계형 데이터베이스와 NoSQL



관계형 데이터베이스

관계형 데이터베이스란 컬럼(열)과 레코드(행)로 구성된 여러 개의 테이블(표) 사이에 각각 어떤 관계를 맺게 한 데이터베이스입니다.

테이블과 테이블 사이의 관계를 이용하면 표로 저장된 데이터끼리 결합하거나, 레코드나 컬럼의 일부를 추출하거나, 테이블에 저장된 데이터를 필요한 형태로 가공해 추출할 수 있습니다(그림 3-3).

그림 3-3 테이블의 결합

전화번호 테이블		주소 테이블	
이름	전화번호	이름	주소
철수	1111-1111	철수	서울시 강남구
영희	2222-2222	영희	서울시 서대문구
길동	3333-3333	길동	서울시 서초구
호동	4444-4444	호동	서울시 마포구

결합

↓

전화번호 · 주소 테이블		
이름	전화번호	주소
철수	1111-1111	서울시 강남구
영희	2222-2222	서울시 서대문구
길동	3333-3333	서울시 서초구
호동	4444-4444	서울시 마포구

관계형 데이터베이스의 테이블과 테이블을 결합하는 예를 들어보겠습니다. 어떤 상점의 데이터베이스에 가격 테이블(표 3-2)과 재고 테이블(표 3-3)이 있다고 합시다. 가격 테이블에는 상품의 판매 가격과 매입 가격을 저장하고, 재고 테이블에는 재고 수량을 저장합니다.

표 3-2 가격 테이블

상품명	판매 가격	매입 가격
귤	30	20
사과	100	50
포도	500	100

표 3-3 재고 테이블

상품명	재고 수량
귤	12
사과	5
포도	8

가격 테이블과 재고 테이블 양쪽에 같은 상품명이 있으므로, 이 관계를 이용해 2개의 테이블을 [표 3-4]와 같이 결합할 수 있습니다.

표 3-4 가격 테이블과 재고 테이블

상품명	판매 가격	재고 수량
귤	30	12
사과	100	5
포도	500	8

2개의 테이블에서 상품명을 기준으로 가격과 재고가 추출됐습니다. 다음 그림은 이 과정을 보여줍니다.

그림 3-4 상품명이 일치하는 데이터를 결합 대상으로 한다.

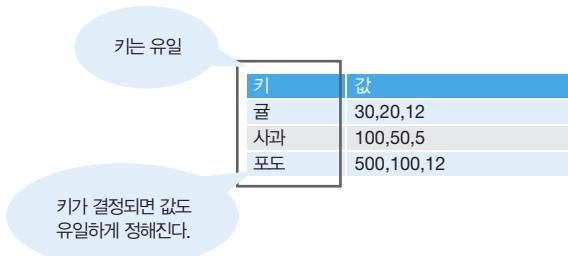


키-값 스토어

키-값 스토어 Key-Value Store란 키와 값을 쌍으로 저장하는 데이터베이스입니다. 키-값 스토어에서 키는 그 데이터 집합 안에서 중복되지 않습니다. 이러한 중복되지 않는 특성을 가리켜 ‘유일하다’고 합니다.

키가 유일하다는 말은 키를 지정하면 쌍으로 연결된 값도 유일하게 정해져 곧바로 추출할 수 있다는 것을 뜻합니다. 키-값 스토어는 결합이나 행, 열을 추출해서 원하는 값을 가져오는 관계형 데이터베이스와 달리 그 구조가 단순해서 값을 추출할 때 걸리는 시간이 단축됩니다(그림 3-5).

그림 3-5 키-값 스토어에서는 키와 값을 쌍으로 저장한다.



컬럼 지향 데이터베이스

컬럼 지향 데이터베이스는 레코드와 컬럼으로 구성된 테이블에 데이터를 컬럼 단위

로 묶어서 저장하는 데이터베이스입니다. 컬럼 단위로 집계하고 추출하는 데 적합합니다.

컬럼 지향 데이터베이스에 관해 알아보기 위해 [표 3-5]와 같은 벌딩의 입퇴실 기록 테이블이 있다고 합시다.

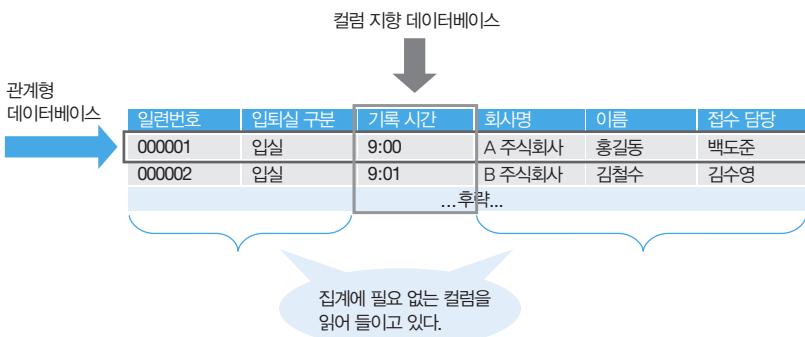
표 3-5 입퇴실 기록

일련번호	입퇴실 구분	기록 시간	회사명	이름	접수 담당
000001	입실	9:00	A 주식회사	홍길동	백도준
000002	입실	9:01	B 주식회사	김철수	김수영
... 중략 ...					
600000	퇴실	23:00	Z 주식회사	강영훈	이현수

이 테이블에는 60만 줄의 대량 데이터가 저장되어 있습니다.

이 테이블에서 시간대별 입퇴실 상황을 집계하고 싶을 때 컬럼 지향 데이터베이스에서는 데이터가 컬럼 단위로 저장되어 있어 집계와 관계없는 회사명이나 이름 등의 항목을 쉽게 배제하고 처리할 수 있으므로 관계형 데이터베이스보다 처리 속도가 빨라집니다(그림 3-6).

그림 3-6 컬럼 지향 데이터베이스에서는 컬럼 단위로 처리한다.



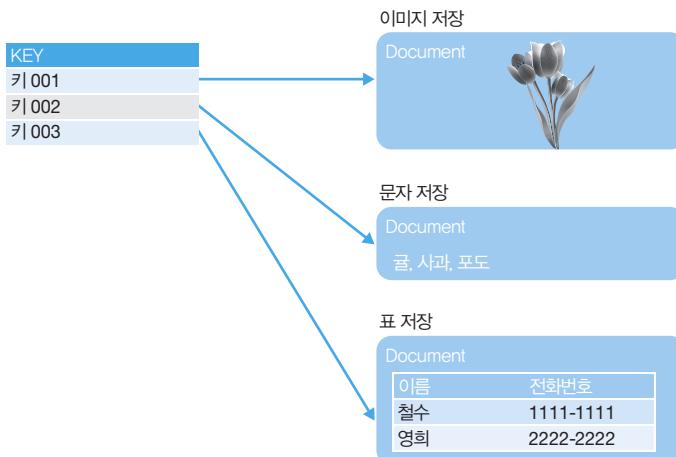
도큐먼트 지향 데이터베이스

도큐먼트 지향 데이터베이스는 키와 도큐먼트를 쌍으로 저장하는 데이터베이스입니다. 도큐먼트에는 임의의 데이터 구조로 된 데이터를 저장할 수 있습니다. 도큐먼트

마다 데이터 구조가 같을 필요는 없으며, 틀에 사로잡히지 않고 자유롭게 데이터를 저장할 수 있습니다.

관계형 데이터베이스에는 컬럼별로 저장할 수 있는 형이나 데이터 크기가 정해져 있지만, 도큐먼트 지향 데이터베이스에는 그런 제한이 없다는 특징이 있습니다(그림 3-7).

그림 3-7 도큐먼트 지향 데이터베이스에는 다양한 형식의 데이터가 저장된다.



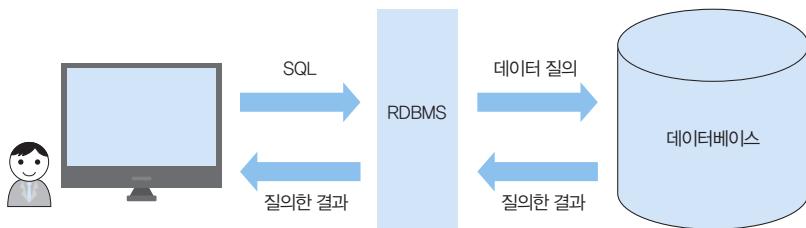
3.1.3 관계형 데이터베이스

이 절에서는 업무 애플리케이션에서 주로 사용하는 관계형 데이터베이스에 대해 자세히 설명하겠습니다.

데이터베이스

관계형 데이터베이스에서는 컬럼과 레코드로 구성된 테이블에 데이터를 저장합니다. 테이블에서는 SQL^{Structured Query Language}이라는 언어로 데이터를 추출하거나 저장할 수 있습니다. 데이터베이스의 데이터를 관리하는 애플리케이션으로 RDBMS^{Relational DataBase Management System}를 사용합니다(그림 3-8).

그림 3-8 데이터베이스에 질의할 때는 SQL을 사용한다.



테이블

앞서 이야기했듯이 관계형 데이터베이스의 테이블은 컬럼과 레코드로 이루어져 있습니다. 또한 모든 컬럼에는 저장할 수 있는 값의 종류가 정의되어 있어, 지정된 종류의 값만 저장할 수 있습니다. 정의에 따라 데이터를 저장해 한 줄의 데이터를 완성합니다.

열에 들어갈 수 있는 값의 정의를 정의역(도메인)이라고 부릅니다. 정의역에는 자바 언어와 유사한 데이터형을 지정할 수 있습니다(표 3-6).

표 3-6 데이터형

분류	데이터형
비트형	BIT, BIT VARYING
문자열형	CHARACTER, CHARACTER VARYING, NATIONAL CHARACTER, NATIONAL CHARACTER VARYING
정수형	INTEGER, DECIMAL, SMALLINT
소수형	DOUBLE PRECISION, FLOAT
날짜시간형	DATE, TIME, TIMESTAMP

사용할 RDBMS의 종류에 따라 이 밖에도 독자적인 데이터형이 정의된 것도 있습니다.

키

테이블에는 특정한 하나의 레코드를 지정할 수 있는 컬럼이 존재합니다. 이 컬럼을 키라고 합니다. 예를 들어 [표 3-7]과 같은 테이블이 있는 경우 키가 되는 컬럼은 무엇일까요?

표 3-7 상품 테이블

상품 코드	상품명	산지
001	사과	영주
002	사과	문경
003	사과	청송
101	복숭아	영주

이 상품 테이블에서 특정한 하나의 레코드를 지정할 수 있는 항목은 ‘상품 코드’ 컬럼이라는 것을 알 수 있습니다. 그러므로 ‘상품 코드’ 컬럼을 키라 할 수 있습니다.

키는 복수의 컬럼 조합으로도 사용할 수 있습니다. ‘상품명’ 컬럼과 ‘산지’ 컬럼의 조합으로도 테이블 중 한 레코드를 지정할 수 있으므로 ‘상품명’ 컬럼과 ‘산지’ 컬럼의 조합도 키라 할 수 있습니다.

앞에서 키가 되는 조건을 ‘테이블에서 특정한 하나의 레코드를 지정할 수 있는 컬럼’이라고 설명했지만, 구체적으로는 컬럼 안에서 값이 중복되지 않는 컬럼, 즉 유일성(유니크)을 만족하는 컬럼이 키가 됩니다.

또한 이번에 예로 든 상품 테이블처럼 복수의 키가 존재할 때는 어느 하나를 그 테이블을 대표하는 키로 사용합니다. 이때 대표가 되는 키를 **주키(기본키)**, 선택되지 않은 키를 **대체키**라고 합니다.

릴레이션십

관계형 데이터베이스에는 테이블과 테이블을 연결해 결합할 수 있는 기능이 있습니다. 이런 연결을 **릴레이션십**이라고 부릅니다. 다음 두 테이블을 이용해 릴레이션십을 설명합니다(표 3-8, 9).

표 3-8 상품 테이블

상품 코드	상품명	산지
001	사과	S01
002	사과	K01
003	사과	K02
101	복숭아	S01

표 3-9 산지 테이블

산지 코드	산지명
S01	영주
K01	문경
K02	청송

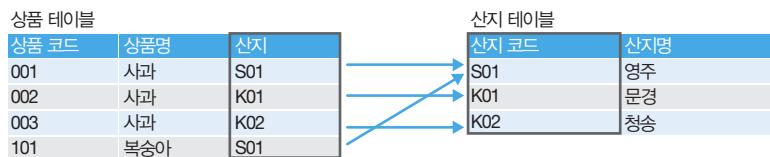
이 두 테이블을 상품 테이블의 ‘산지’와 산지 테이블의 ‘산지 코드’를 이용해 각 레코드를 결합하면 [표 3-10]처럼 됩니다.

표 3-10 상품 테이블과 산지 테이블을 결합한 결과

상품 코드	상품명	산지 코드	산지
001	사과	S01	영주
002	사과	K01	문경
003	사과	K02	청송
101	복숭아	S01	영주

상품 테이블의 ‘산지’가 산지 테이블의 ‘산지 코드’를 참조합니다. 이때 상품 테이블의 ‘산지’를 외래키라고 합니다(그림 3-9).

그림 3-9 다른 테이블의 컬럼을 참조하는 컬럼을 외래키라고 한다.



제약

각 컬럼에 저장되는 데이터에 규칙을 적용할 수 있습니다. 이 규칙을 제약이라고 합니다. 주요 제약을 [표 3-11]에 나타냈습니다.

표 3-11 주요 제약

제약	제약 내용
NOT NULL	열 값으로 NULL을 넣을 수 없다.
유일성 제약(UNIQUE 제약)	열 값으로 중복된 값을 넣을 수 없다.
주키 제약	열 값으로 NULL, 중복된 값을 넣을 수 없다.
CHECK 제약	사용자가 설정한 값 외에는 넣을 수 없다.

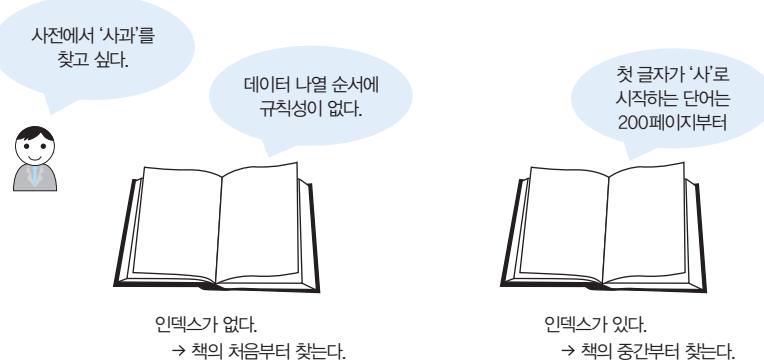
표를 보면 데이터베이스에는 NULL이란 값이 없음을 알 수 있습니다.

인덱스

RDBMS는 데이터베이스에 저장된 데이터를 빠르게 검색하는 인덱스 index라는 메커니즘을 제공합니다.

인덱스는 사전 등을 찾아볼 때 사용하는 색인에 해당하는 기능으로, 테이블에 저장된 데이터에 색인을 부여하는 것입니다. 가나다순으로 정렬된 사전에서 단어를 찾을 때 ‘사’로 시작하는 단어는 ‘ㅅ’ 색인이 붙은 곳부터 찾기 시작하면 빠르게 찾을 수 있습니다. 만약 인덱스가 없다면 처음부터 모조리 찾아봐야 하므로 시간이 더 걸리겠지요 (그림 3-10).

그림 3-10 색인이 있는 경우와 없는 경우의 페이지 탐색



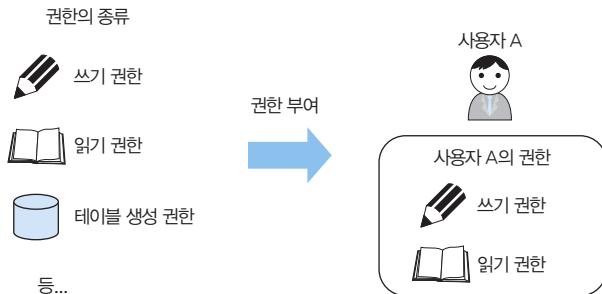
권한과 룰

데이터베이스에는 여러 가지 데이터가 저장됩니다. 저장되는 데이터 중에는 은행의

예금 정보, 온라인 쇼핑몰의 고객 정보처럼 기밀성이 높은 데이터도 있습니다. 이런 데이터를 아무나 검색할 수 있다거나 간신할 수 있다면 큰 문제가 되겠지요.

그래서 RDBMS에는 데이터베이스 접근을 제한하는 기능이 구현되어 있습니다. ID와 패스워드로 사용자를 인증하고, 제한된 사용자에게만 데이터베이스 접근을 허가합니다. 또한 사용자 단위로 테이블에 대해 검색, 데이터 삽입, 간신 등 어떤 조작을 할 수 있는지 설정할 수 있습니다. 이처럼 데이터베이스 접근과 테이블 검색, 삽입, 간신 등을 허가 또는 금지하는 기능을 권한이라고 합니다.

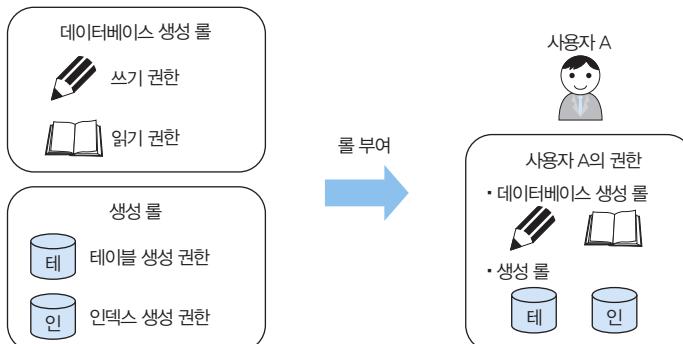
그림 3-11 권한 부여



권한을 몇 가지 모아서 하나의 셋으로 만든 것이 룰^{role}입니다(그림 3-12).

데이터베이스 상의 모든 테이블에 대한 검색 권한만을 모은 ‘열람’ 를 등을 미리 만들 어두면 사용자가 늘어났을 때 사용자마다 모든 테이블에 대한 권한을 일일이 설정할 필요가 없어 시간이나 실수를 줄일 수 있습니다.

그림 3-12 룰 생성



3.1.4 데이터를 파일로 관리할 때의 문제점

데이터를 데이터베이스와 파일로 저장했을 때 양쪽의 차이는 무엇일까요?

파일로 저장된 데이터를 다룰 경우를 예로 들어 관계형 데이터베이스와 어떤 차이가 있는지, 그리고 파일로 데이터를 관리하는 방식의 문제점을 살펴보겠습니다.

데이터 사이의 관계가 파괴될 가능성이 있다

어떤 상점의 데이터 관리 시스템에 [표 3-12]와 [표 3-13] 같은 데이터가 각각 다른 파일에 저장되어 있다고 합시다. 하나는 매입 가격과 판매 가격을 관리하는 상품 가격 파일이고, 다른 하나는 상품명을 관리하는 파일입니다.

표 3-12 상품 가격 파일

상품명	매입 가격	판매 가격
포테이토칩	50	100
다크초콜릿	100	150
롤리팝사탕	20	50
눈깔사탕	10	30

표 3-13 상품명 파일

상품명	상품 분류
포테이토칩	스낵
다크초콜릿	초콜릿
롤리팝사탕	사탕
눈깔사탕	사탕

어느 날 상품 구매처에서 상품명이 변경됐다는 연락이 왔습니다. 롤리팝사탕을 막대 사탕으로 변경했다고 합니다.

이 경우 상품명이 저장된 파일을 갱신할 필요가 있습니다. 파일을 확인해보면 상품명은 2개 파일에 저장되어 있습니다. 만약 갱신 담당자가 상품명과 관련된 것이 상품명 파일뿐이라고 착각하여 상품 가격 파일의 상품명을 갱신하지 않았을 때 파일의 데이터는 각각 [표 3-14]와 [표 3-15] 같은 상태가 됩니다.

표 3-14 상품 가격 파일

상품명	매입 가격	판매 가격
포테이토칩	50	100
다크초콜릿	100	150
롤리팝사탕	20	50
눈깔사탕	10	30

표 3-15 상품명 파일

상품명	상품 분류
포테이토칩	스낵
다크초콜릿	초콜릿
막대사탕	사탕
눈깔사탕	사탕

상품명 파일의 막대사탕과 상품 가격 파일의 롤리팝사탕은 원래 같은 상품을 가리켰지만, 한쪽 파일을 간접화하지 않아서 두 데이터 간의 관계가 사라져버렸습니다. 관계형 데이터베이스에서는 데이터끼리의 관계가 일치하도록 유지하는 기능이 있어 이러한 일이 발생하지 않습니다.

동시에 데이터를 간접화해서 데이터가 불일치할 가능성

파일로 데이터를 관리할 경우 데이터를 동시에 간접화하면 데이터의 불일치가 발생할 가능성이 있습니다. 예를 들어 다음과 같이 상품 재고를 관리하는 재고 파일이 있다 고 합시다.

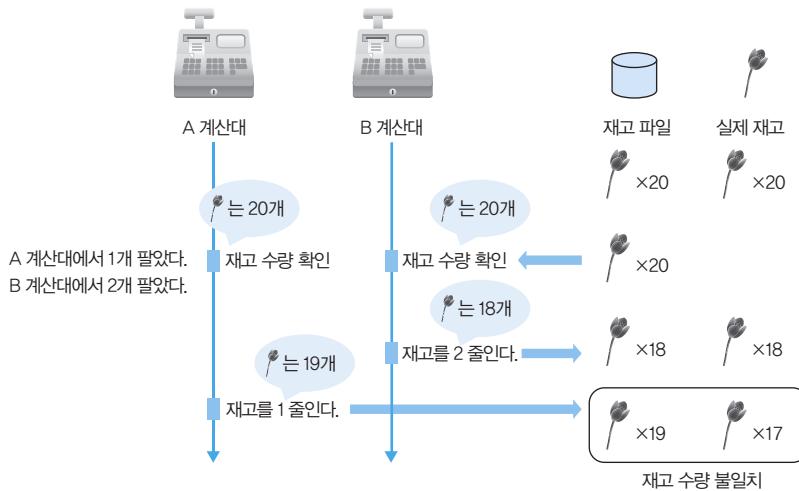
표 3-16 재고 파일

상품명	재고 수량
포테이토칩	10
다크초콜릿	5
막대사탕	20
눈깔사탕	20

어느 날 A 계산대에서 눈깔사탕을 1개 팔았고, 그와 동시에 B 계산대에서 같은 상품을 2개 팔았습니다. 이때 재고 파일에는 현재 재고 수량에서 팔린 수량만큼 빼는 처리가 진행되어야 합니다. 그러므로 실제 재고는 A 계산대와 B 계산대를 합하여 3개 팔았으므로 17이 되어야 합니다.

하지만 A 계산대와 B 계산대에서 동시에 눈깔사탕을 팔았기 때문에 재고 수량이 20 개임을 확인합니다. 이때 A 계산대에서는 재고 파일을 갱신하여 재고 수량이 19개라고 저장하고, B 계산대에서는 18개라고 저장하게 됩니다. 여기서 A 계산대가 B 계산대보다 조금 늦게 저장했다면 B 계산대에 의해 재고 수량이 18로 갱신됐다가 A 계산대에 의해 다시 19로 갱신됩니다. 파일상의 데이터와 실제 재고 수량에 불일치가 생기고 말았습니다(그림 3-13).

그림 3-13 실제 재고 수량과 파일상의 재고 수량이 일치하지 않는다.



관계형 데이터베이스에서는 복수의 사용자가 데이터를 갱신할 경우 한 사용자가 갱신하는 동안 다른 사용자의 갱신 처리를 일시적으로 대기하게 하는 메커니즘이 있습니다.

이런 사례 외에도 파일로 데이터를 관리하게 되면 같은 상품명의 데이터를 중복해서 저장하는 실수도 일어날 가능성이 있습니다. 파일을 이용한 데이터 관리는 비용이 저렴하다는 장점이 있지만 이런 문제점을 안고 있습니다.

4 장

텍스트 입출력

4.1 텍스트 파일 읽기

이 장에서는 텍스트 파일을 읽고 쓰는 방법을 학습합니다. 우선 읽기부터 시작합시다.

텍스트를 읽는 방법에는 다음처럼 몇 가지가 있습니다.

- 파일의 문자를 한 문자씩 읽는 방법(FileReader 클래스)
- 텍스트를 한 줄씩 읽는 방법(BufferedReader 클래스)
- 텍스트를 한번에 모두 읽는 방법(Scanner 클래스, Files 클래스)

지금부터 각 방법의 구체적인 코드를 살펴보면서 텍스트 읽는 방법을 알아보겠습니다.

4.1.1 파일의 문자를 한 문자씩 읽는 방법

FileReader 클래스는 텍스트 파일에서 문자 단위로 데이터를 읽어 들어 들이는 클래스입니다. 자바 1.1부터 사용했습니다. [표 4-1]은 FileReader 클래스의 생성자입니다.

표 4-1 FileReader 클래스의 생성자

생성자	개요
FileReader(File file)	File 객체를 지정해 새 FileReader를 생성한다.
FileReader(String fileName)	읽어 들일 파일명을 지정해 새 FileReader를 생성한다.

[표 4-2]는 FileReader 클래스의 주요 메서드입니다. 지정한 파일을 읽어올 때와 파일을 닫을 때 사용하는 메서드입니다.

표 4-2 FileReader 클래스의 주요 메서드

메서드	개요
int read()	입력 스트림에서 한 개의 문자를 읽어 반환한다.
void close()	입력 스트림을 닫고, 스트림에 관련된 모든 시스템 리소스를 해제한다.

다음과 같이 읽고 싶은 텍스트 파일을 대상으로 File 클래스의 오브젝트를 생성합니다. 그런 다음 File 오브젝트를 인수로 지정해 FileReader 클래스의 오브젝트를 생성합니다.

예제 4-1 FileReader 오브젝트 생성

```
File file = new File(file_name);
FileReader filereader = new FileReader(file);
```

다음과 같이 생성된 `FileReader` 오브젝트의 `read` 메서드로 파일에서 문자를 하나 읽어 들입니다.

예제 4-2 문자열을 한 문자씩 읽어 들이는 방법

```
int singleCh = 0;
while((singleCh = filereader.read()) != -1) {
    System.out.print((char)singleCh);
}
```

`read` 메서드는 문자를 하나씩 읽어 들이므로 `while` 루프로 읽기 처리를 반복합니다. 또한 이 메서드는 읽어 들인 값을 문자 코드(int형)로 반환하므로 문자로 사용할 때는 문자형으로 캐스트(형변환)할 필요가 있습니다. 반환값이 -1인 경우에는 파일 끝에 도달한 것이므로 루프를 종료합니다.

`FileReader` 오브젝트를 이용할 때는 지정한 파일이 없는 경우에 발생하는 `FileNotFoundException`과 `read` 메서드의 `IOException` 예외를 `try-catch` 문으로 처리해야 합니다.

다음은 이상의 내용을 정리한 코드입니다.

예제 4-3 ReadText1.java

```
package jp.co.bbreak.sokusen._4._1;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class ReadText1 {

    public static void main(String[] args) {

        try {
            // 파일 오브젝트 생성
            File file = new File("c:\\sokusen\\Sample.txt");

            // 입력 스트림 오브젝트 생성
            FileReader filereader = new FileReader(file);


```

```

int singleCh = 0;

// while 문을 이용해 파일을 읽어 들인다.
while ((singleCh = filereader.read()) != -1) {
    System.out.print((char) singleCh);
}

// 입력 스트림을 닫는다.
filereader.close();

} catch (FileNotFoundException e) {
    System.out.println(e);
} catch (IOException e) {
    System.out.println(e);
}
}
}
}

```

4.1.2 텍스트를 한 줄씩 읽는 방법

`FileReader`로 텍스트를 한 문자씩 읽는 것은 꽤 번거로운 작업입니다. 그러므로 한 줄 단위로 텍스트를 읽어 들이는 `BufferedReader` 클래스를 이용해 효율적으로 텍스트를 읽어봅시다.

[표 4-3]은 `BufferedReader` 클래스의 생성자입니다.

표 4-3 `BufferedReader` 클래스의 생성자

생성자	개요
<code>BufferedReader (Reader in)</code>	기본 크기의 버퍼로 버퍼링된 문자열 입력 스트림 생성
<code>BufferedReader (Reader in, int sz)</code>	지정한 크기의 버퍼로 버퍼링된 문자열 입력 스트림 생성

[표 4-4]는 주요 메서드입니다.

표 4-4 `BufferedReader` 클래스의 주요 메서드

메서드	개요
<code>String readLine()</code>	텍스트를 한 줄씩 읽어 들인다. 단, 행의 마지막 문자(개행 문자)는 포함하지 않는다. 스트림의 끝에 도달하면 <code>null</code> 을 반환한다.
<code>Stream<String> lines()</code>	텍스트를 한 줄씩 읽어 들인다. 파일의 마지막 행인지 체크했던 <code>null</code> 체크가 필요 없어졌다(자바 8에서 추가된 메서드).

`readLine` 메서드에서는 읽어 들인 텍스트를 한 줄씩 처리하므로 다음과 같이 `while` 루프로 읽기 처리를 반복해야 합니다. `readLine` 메서드의 반환값이 `null`이 아닌 경우 조건식의 값은 `true`가 되어 다음 행이 존재한다고 판정합니다.

예제 4-4 문자열을 한 줄씩 읽어 들이는 방법(1)

```
BufferedReader br = new BufferedReader(new FileReader('텍스트의 경로'));
String line = br.readLine();
while (line != null) {
    System.out.println(line);
}
```

또는 다음과 같이 판정 처리를 `while` 블록에 넣어도 상관없습니다.

예제 4-5 문자열을 한 줄씩 읽어 들이는 방법(2)

```
String line "";
while((line = br.readLine()) != null) {
    System.out.println(line);
}
```

다음은 지금까지 설명한 내용을 정리한 코드입니다.

예제 4-6 ReadText2.java

```
package jp.co.bbreak.sokusen._4._1;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

public class ReadText2 {

    public static void main(String[] args) {
        try {
            // 파일 오브젝트 생성
            File file = new File("c:\\sokusen\\Sample.txt");
            FileReader fRd = new FileReader(file);
            BufferedReader bufRd = new BufferedReader(fRd);

            String line = "";
```

```
// while 문을 이용해 파일을 읽어 들인다.  
while ((line = bufRd.readLine()) != null) {  
    System.out.println(line);  
}  
  
// 입력 스트림을 닫는다.  
bufRd.close();  
  
} catch (FileNotFoundException e) {  
    System.out.println(e);  
} catch (IOException e) {  
    System.out.println(e);  
}  
}  
}  
}
```

위 예제 코드에서는 파일을 조작한 후 `close` 메서드로 `BufferedReader` 오브젝트를 닫았습니다.

하지만 자바 7부터 추가된 `try-with-resources` 문을 사용하면 `close` 메서드로 명시적으로 `BufferedReader` 오브젝트를 닫을 필요가 없습니다. `java.lang.AutoCloseable` 인터페이스를 구현한 클래스는 `try` 블록을 빠져나온 시점에 자동으로 `close`되기 때문입니다.

다음은 `ReadText2.java`를 `try-with-resources` 문으로 바꿔 쓴 것입니다.

예제 4-7 try-with-resources 문을 사용한 예

```
public static void main(String[] args) {  
  
    // 파일 오브젝트 생성  
    File file = new File("c:\\sokusen\\Sample.txt");  
  
    // FileReader와 BufferedReader를 try 뒤에 선언한다.  
    try(FileReader fRd = new FileReader(file);  
        BufferedReader bufRd = new BufferedReader(fRd)) {  
  
        String line = "";  
  
        // while 문을 이용해 파일을 읽어 들인다.  
        while ((line = bufRd.readLine()) != null) {  
            System.out.println(line);  
        }  
    }  
}
```

```
// 여기서 실행했던 close 처리는 생략할 수 있다.

} catch (FileNotFoundException e) {
    System.out.println(e);
} catch (IOException e) {
    System.out.println(e);
}
}
```

`try-with-resources` 문을 사용할 때는 `try` 바로 뒤에 자동으로 닫을 리소스 오브젝트를 선언합니다. 위에서처럼 복수의 리소스가 있을 때는 세미콜론(;)으로 구분합니다.

4.1.3 텍스트를 한번에 모두 읽는 방법(1)

텍스트를 한번에 모두 읽는 방법으로 `Scanner` 클래스를 사용할 수 있습니다. [표 4-5]는 `Scanner` 클래스에서 이용할 수 있는 생성자와 메서드입니다.

표 4-5 `Scanner` 클래스의 생성자와 메서드

메서드	개요
<code>Scanner(File source)</code>	지정된 파일에서 입력받을 <code>Scanner</code> 를 생성한다 (생성자).
<code>boolean hasNextLine()</code>	다음 행이 있을 때는 <code>true</code> 를 반환한다.
<code>boolean useDelimiter(String pattern)</code>	스캐너에서 이용할 구분 문자를 지정된 패턴으로 설정한다.
<code>String next()</code>	스캐너에서 다음 토큰을 가져온다.
<code>String nextLine()</code>	스캐너에서 다음 행을 가져온다.

`nextLine` 메서드는 읽어 들인 텍스트를 한 줄씩 처리하므로 다음과 같이 `while` 루프로 읽는 처리를 반복합니다. `hasNextLine` 메서드는 다음 행이 존재하는지 판정해 `boolean`형 값을 반환합니다. `true`인 경우에는 다음 행에 문자가 있다는 뜻입니다.

예제 4-8 문자열을 한 줄씩 가져온다.

```
while (scan.hasNextLine()) {
    System.out.println(scan.nextLine());
}
```

Scanner 클래스에는 루프를 사용하지 않고 텍스트 내용을 가져오는 방법도 있습니다. 정규 표현식을 조합해 마지막 행까지의 데이터를 가져오면 됩니다. 다음 코드는 \z로 파일 끝을 판정하고 그때까지의 텍스트를 모아서 가져옵니다.

예제 4-9 텍스트의 마지막 줄까지 한번에 가져오는 방법

```
System.out.println(scan.useDelimiter("\z").next());
```

다음은 지금까지 설명한 내용을 정리한 코드입니다.

예제 4-10 ReadText3.java

```
package jp.co.bbreak.sokusen._4._1;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class ReadText3 {

    public static void main(String[] args) {
        try {
            // 파일 오브젝트 생성
            File file = new File("c:\\sokusen\\Sample.txt");

            // Scanner 오브젝트 생성
            Scanner scan = new Scanner(file);

            // while 문으로 파일을 읽어 들인다.
            System.out.println(scan.useDelimiter("\z").next());

            // 입력 스트림을 닫는다.
            scan.close();

        } catch (FileNotFoundException e) {
            System.out.println(e);
        }
    }
}
```

4.1.4 텍스트를 한번에 모두 읽는 방법(2)

파일의 내용을 모아서 읽는 방법에는 Scanner 클래스 외에도 Files 클래스를 사용하는 방법이 있습니다. Files 클래스는 자바 7부터 도입됐습니다.

이 클래스는 static 메서드만으로 구성되어 있으므로 생성자는 없습니다. [표 4-6]은 Files 클래스의 주요 메서드입니다.

표 4-6 Files 클래스의 주요 메서드

메서드	개요
boolean isReadable(Path path)	파일을 읽을 수 있는지 판정한다.
byte[] readAllBytes(Path path)	파일의 모든 바이트를 읽고, 바이트 배열로 반환한다(파일 크기가 2GB를 넘을 경우에는 OutOfMemoryError 예외를 던진다).
List<String> readAllLines(Path path, Charset cs)	지정한 문자셋으로 파일의 모든 행을 String으로 가져온다. 자바 8부터는 문자셋을 생략할 수 있다(생략 시 기본값은 UTF-8).
Stream<String> lines(Path path, Charset cs)	지정한 문자셋으로 파일의 모든 행을 Stream으로 가져온다. 자바 8부터는 문자셋을 생략할 수 있다(생략 시 기본값은 UTF-9).

Files 클래스에서 대상이 되는 파일과 폴더는 Path와 Paths 클래스로 지정할 수 있습니다. [표 4-7]은 Paths 클래스의 주요 메서드입니다.

표 4-7 Paths 클래스의 주요 메서드

메서드	개요
Path get(String first, String... more)	지정한 경로 문자열을 연결해 파일 경로를 반환한다. get("C:\\\\", "temp", "aaa") 일 때 반환값은 C:\\temp\\aaa다.

다음은 이상의 메서드를 조합해 텍스트를 읽어 들이는 처리를 작성한 코드입니다.

예제 4-11 ReadText4.java

```
package jp.co.bbreak.sokusen._4._1;

import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
```

```
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;

public class ReadText4 {

    public static void main(String[] args) {

        // 파일 오브젝트 생성
        Path path = Paths.get("c:\\sokusen\\Sample.txt");

        // 문자셋을 지정한다.
        Charset cs = StandardCharsets.UTF_8;
        List<String> list = new ArrayList<String>();

        try {
            list = Files.readAllLines(path, cs);

        } catch (IOException e) {
            e.printStackTrace();
        }

        // 가져온 텍스트 내용을 출력한다.
        for (String readLine : list) {
            System.out.println(readLine);
        }
    }
}
```

4.2 텍스트 파일 쓰기

텍스트 파일을 읽을 때처럼 파일에 쓰는 방법도 몇 가지 있습니다.

- `FileWriter` 클래스로 파일에 쓰기
- `BufferedWriter` 클래스로 파일에 쓰기
- `Files` 클래스로 파일에 쓰기

여기서는 각 방법에 대해 구체적인 코드를 보면서 이해해봅시다.

5장

스레드

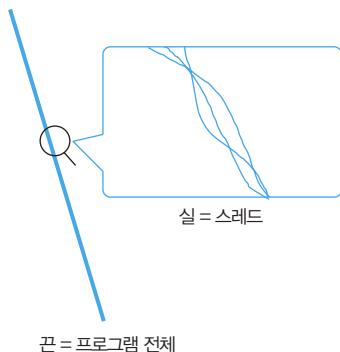
5.1 멀티 스레드 처리

처음 프로그래밍을 접하는 사람에겐 멀티 스레드 처리가 생소한 용어일지도 모릅니다. 이 장에서는 지금까지와는 달리 프로그램을 어떻게 동작시키는지 생각해봅니다. 지금까지는 프로그램이 코드의 위에서 아래로 흘러가면서 처리됐습니다. 여기서는 병렬로 처리하는 방법을 학습합니다. 우선 멀티 스레드란 말의 뜻부터 알아봅시다.

5.1.1 스레드

스레드 thread를 사전에서 찾아보면 ‘끈을 구성하는 실’이라는 뜻도 있습니다. 프로그래밍 용어의 스레드도 거기서 유래했고, 다음 그림이 보여주는 것처럼 스레드를 엮어서 하나의 프로그램을 조립할 수 있습니다.

그림 5-1 스레드와 실의 관계



단, 어디까지나 ‘할 수 있다’는 것이지, 스레드를 엮어서 프로그램을 만들려면 자바의 구조를 따라 작성해야 합니다.

지금까지 이 책에서 학습한 프로그램은 여러 스레드를 엮어서 만들지 않았습니다. 하나의 스레드(싱글 스레드)로 프로그램이 동작했지요. 그래도 아무런 문제가 없었습니다. 이처럼 프로그램이 예상한 대로 잘 동작하는데 왜 멀티 스레드가 필요한 걸까요?

NOTE_용어의 의미를 찾아보는 것도 중요하다

프로그래밍 용어의 사전적 의미를 알아보는 것도 중요합니다. 프로그래밍뿐만 아니라 컴퓨터 용어는 대부분 영어권에서 왔습니다. 그러므로 사전적인 의미를 알아두면, 그 용어가 원래 가리키는 의미를 알 수 있고 올바른 이해로 이어집니다.

5.1.2 멀티 스레드

멀티 스레드란 이름 그대로 복수의 스레드로 하나의 프로그램을 실행하는 기술입니다.

왜 멀티 스레드를 사용하는 걸까요? 멀티 스레드를 사용하는 이유는 처리를 빠르게 하기 위함입니다. 멀티 스레드 사용과 프로그램의 실행 속도가 어떻게 관련되는지 알려면 프로그램이 가진 성질을 먼저 이해해야 합니다.

프로그램에서는 주로 논리적인 조작을 하면서 동시에 외부와 데이터를 주고받는 처리를 하게 됩니다. 그런데 외부와의 연계에서 대기 시간이 발생하는 경우가 있습니다. 기다리는 동안 다른 일을 해두면 전체적인 처리 시간이 짧아지겠지요. 집안일을 예로 들면 세탁기가 돌아가는 동안에 청소기로 방을 청소하는 것입니다.

단, 멀티 스레드로 처리한다고 뭐든지 빨라지는 것은 아닙니다. 프로그램을 실행하는 컴퓨터의 CPU 코어 수가 적으면 병렬 처리 스레드를 그만큼 만들 수 없기 때문에 생각보다 빨라지지 않습니다. 또한 처리하는 데이터양이 적은 경우도 마찬가지로 속도 향상을 기대할 수 없습니다. 그러므로 이용 상황을 잘 고려해서 꼭 멀티 스레드로만 들어야 할지 검토할 필요가 있습니다.

NOTE_빨리 동작하는 것의 의의

왜 빠르게 동작시키는 기술이 필요한 것일까요? 프로그램의 구성을 보는 관점에는 기능 요건과 비기능 요건이라는 분류가 있습니다. 기능 요건은 원하는 동작을 하길 바라는 행위 요건입니다. 반면 비기능 요건은 '이 프로그램은 몇 초 내로 처리하고 싶다'와 같은 기능 이외의 사용성에 관련된 요건을 가리킵니다.

양쪽 모두 시스템 개발을 진행하는 데 있어 고려해야 하는 중요한 요건입니다. 멀티 스레드는 비기능 요건을 만족시키고자 만든 선인들의 지혜입니다.

다음은 멀티 스레드의 간단한 예를 보여줍니다.

예제 5-1 MultiThreadSample.java

```
package jp.co.bbreak.sokusen._5._1;

/**
 * 멀티 스레드 샘플
 */
public class MultiThreadSample implements Runnable {

    /** 출력 메시지 템플릿 */
    private static final String MSG_TEMPLATE = "출력 중입니다.[%s][%d회째]";

    /** 스레드명 */
    private final String threadName;

    public MultiThreadSample(String threadName) {
        this.threadName = threadName;
    }

    public void run() {
        for (int i = 1; i < 100; i++) {
            System.out.println(String.format(MSG_TEMPLATE, threadName, i));
        }
    }

    public static void main(String[] args) {
        MultiThreadSample runnable1 = new MultiThreadSample("thread1");
        MultiThreadSample runnable2 = new MultiThreadSample("thread2");
        MultiThreadSample runnable3 = new MultiThreadSample("thread3");

        Thread thread1 = new Thread(runnable1);
        Thread thread2 = new Thread(runnable2);
        Thread thread3 = new Thread(runnable3);

        thread1.start();
        thread2.start();
        thread3.start();
    }
}
```

실행 결과

```
출력 중입니다.[thread3][1회째]
출력 중입니다.[thread1][1회째]
출력 중입니다.[thread2][1회째]
출력 중입니다.[thread1][2회째]
... 후략 ....
```

몇 번 실행해보면 알 수 있지만 출력 순서가 매번 달라집니다. 그 이유는 처리가 동시에 실행되기 때문입니다.

이번 예제에서는 멀티 스레드로 처리할 클래스에 `Runnable` 인터페이스를 구현했습니다. 스레드로 처리할 내용은 `run` 메서드에 기술합니다.

`Runnable` 구현 클래스의 인스턴스를 바탕으로 `Thread` 인스턴스를 생성함으로써 스레드를 생성할 수 있습니다. 그리고 `start` 메서드로 스레드를 시작합니다. 단순히 `Runnable` 인터페이스를 구현하고 `run` 메서드를 실행해서는 멀티 스레드로 실행되지 않으니 주의하세요.

또한 멀티 스레드를 구현하는 데는 `Thread` 클래스를 상속하는 방법도 있습니다. 하지만 클래스의 구조를 단순하게 유지할 수 있으므로 일반적으로는 `Runnable` 인터페이스를 구현하는 방법을 사용합니다.

5.1.3 더 복잡한 멀티 스레드 제어 방법

앞에서 설명한 멀티 스레드 작성 방법은 원시적인 형태라고 할 수 있습니다. 멀티 스레드 프로그램이 어떤 것인지 알기에는 충분하지만 실무에서 사용하기에는 문제가 있습니다.

좀 전의 예는 3개의 스레드만 만든 것이라서 문제가 없었습니다. 하지만 스레드가 몇 개 만들어질지 정해지지 않은 프로그램은 너무 많은 스레드가 한번에 실행될 가능성이 있습니다. 그 결과 동작 중인 컴퓨터의 메모리 자원을 다 써버려서 처리를 계속할 수 없게 됩니다.

이런 경우 멀티 스레드 프로그램을 어떻게 만들면 좋을까요?

스레드 풀

자바에서는 스레드 풀 Thread Pool이라는 기능을 제공합니다. 스레드 풀이란 사용할 스레드를 제한된 수만큼 만들어두고 일정한 규칙에 따라 실행하는 기능으로, `java.util.concurrent` 패키지에서 제공합니다.

[표 5-1]은 `java.util.concurrent` 패키지에 속하는 대표적인 인터페이스입니다.

표 5-1 스레드 풀 기능을 제공하는 인터페이스

인터페이스	설명
ExecutorService	스레드 수를 제한하는 등 일정한 제한 아래에서 멀티 스레드 처리를 실행하기 위한 인터페이스
ScheduledExecutorService	일정 시간 후에 시작하고 일정 제한 아래에서 멀티 스레드 처리를 실행하기 위한 인터페이스

이 인터페이스를 구현한 오브젝트는 마찬가지로 `java.util.concurrent` 패키지에서 제공하는 `Executors` 클래스를 거쳐서 가져옵니다. 대표적인 메서드는 [표 5-2]와 같습니다.

표 5-2 Executors 클래스의 주요 메서드

인터페이스	설명
<code>newSingleThreadExecutor</code>	싱글 스레드로 동작하는 <code>ExecutorService</code> 의 인스턴스를 반환한다.
<code>newFixedThreadPool</code>	지정한 최대 동시 실행 스레드 수로, <code>ExecutorService</code> 의 인스턴스를 반환한다.
<code>newScheduledThreadPool</code>	지정한 최대 동시 실행 스레드 수로, <code>ScheduledExecutorService</code> 의 인스턴스를 반환한다.

다음은 앞서 만든 프로그램을 스레드 풀로 동작하도록 수정한 것입니다. 코드의 어디가 바뀌었는지 주목하세요. 실행 결과는 변하지 않습니다.

예제 5-2 ThreadPoolSample.java

```
package jp.co.bbreak.sokusen._5._1;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

/**
 * 스레드 풀 샘플
 */
public class ThreadPoolSample implements Runnable {

    /** 출력 메시지 템플릿 */
    private static final String MSG_TEMPLATE = "출력 중입니다. [%s][%d회째]";

    /** 스레드명 */
    private final String threadName;
```

```

public ThreadPoolSample(String threadName) {
    this.threadName = threadName;
}

public void run() {
    for (int i = 1; i < 100; i++) {
        System.out.println(String.format(MSG_TEMPLATE, threadName, i));
    }
}

public static void main(String[] args) {
    ThreadPoolSample runnable1 = new ThreadPoolSample("thread1");
    ThreadPoolSample runnable2 = new ThreadPoolSample("thread2");
    ThreadPoolSample runnable3 = new ThreadPoolSample("thread3");

    // 스레드 동시 실행 수는 3
    ExecutorService executorService = Executors.newFixedThreadPool(3);
    executorService.execute(runnable1);
    executorService.execute(runnable2);
    executorService.execute(runnable3);

    executorService.shutdown();
    try {
        if (!executorService.awaitTermination(5, TimeUnit.MINUTES)) {
            // 타임아웃 후에도 아직 실행이 끝나지 않았다.
            executorService.shutdownNow();
        }
    } catch (InterruptedException e) {
        // 종료 대기 시에 뭔가 오류가 발생했다.
        e.printStackTrace();
        executorService.shutdownNow();
    }
}
}

```

스레드 풀을 `newFixedThreadPool` 메서드로 생성하고 있습니다. 이번에는 3개의 스레드를 동시에 실행하므로 인수(동시 실행 스레드 최대 수)에도 이 값을 지정합니다.

또한 `newFixedThreadPool` 메서드의 인수를 1로 지정하면 동시 실행 수가 1이 되어 최초의 스레드부터 차례로 실행됩니다. 이 경우는 `getSingleThreadExecutor` 메서드로 `ExecutorService`의 인스턴스를 가져오는 것과 같습니다.

스레드는 `execute` 메서드로 시작하고 `shutdown` 메서드로 처리를 종료합니다. 단,

그 자리에서 종료하는 것은 아니고, 실행 중인 스레드가 끝나야 비로소 종료 상태가 됩니다.

그래서 `awaitTermination` 메서드로 모든 스레드가 종료될 때까지 대기 상태로 둡니다. 이 예제에서는 첫째 인수로 5를 지정하고, 둘째 인수로 `TimeUnit.MINUTES`를 지정했으므로 5분이 경과하면 타임아웃(시간 종료)됩니다.

타임아웃되면 `awaitTermination` 메서드가 `false`를 반환합니다. 예제에서는 타임아웃된 시점에 `shutdownNow` 메서드를 호출해서 실행 중인 메서드가 있어도 스레드 전체를 강제 종료합니다.

단, 일반적인 애플리케이션에서는 (단순히 강제 종료하는 것이 아니라) 어떻게 대처할지 검토한 다음에 이상 시 처리를 기술할 필요가 있습니다.

또한 `InterruptedException`이 발생했을 때의 처리가 예외 클래스의 `printStackTrace` 메서드 호출과 `shutdownNow` 메서드 호출로 되어 있습니다. 이는 오류 정보를 출력하고 스레드를 강제 종료한다는 의미입니다. 이런 예외 처리에서도 타임아웃과 마찬가지로 이상 시 처리를 검토할 필요가 있습니다.

5.2 스레드 세이프란

스레드 세이프 Thread Safe (스레드 안전성)란 멀티 스레드로 동작하는 프로그램에서 복수의 스레드로부터 호출되더라도 기대대로 동작하는 것을 가리킵니다.

멀티 스레드를 설명할 때 세탁과 청소를 동시에 하는 예를 들었습니다. 세탁과 청소라면 사용하는 기계가 다르니 동시에 해도 지장이 없겠지요.

하지만 티셔츠와 스웨터를 동시에 세탁하는 경우를 생각해봅시다. 일반적으로 스웨터와 티셔츠는 따로 세탁해야 합니다. 스레드 세이프하지 않은 프로그램의 경우 이를 무시하고 티셔츠를 세탁하는 도중에 스웨터를 던져 넣습니다(집안일을 잘하는 사람에게는 있을 수 없는 일이지만).

이처럼 동시에 뭔가 하려고 했는데 기대한 결과가 돌아오지 않는 프로그램은 스레드 세이프하지 않은 것입니다. 멀티 스레드 프로그램에서는 복수의 스레드가 인스턴스

를 공유할 가능성이 있으므로 인스턴스가 어떤 때라도 기대한 대로 처리해줄 필요가 있습니다.

다시 세탁기를 예로 들어봅시다. 세탁 도중에 뭔가 던져 넣었을 때 세탁조에 빨래가 떨어지기 전에 확보하고, 현재 처리 중인 세탁이 끝나기를 기다렸다가 나중에 넣은 빨래를 세탁하는 기능이 있다면 그 세탁기는 스레드 세이프라고 할 수 있습니다(이런 기능을 탑재한 세탁기는 비싸겠지만요).

5.2.1 스레드 세이프하지 않은 경우의 사례

슬슬 세탁기를 예로 드는 것도 질렸을 것입니다. 이제 프로그램에선 어떻게 되는지 알고 싶을 것입니다.

그럼 실제로 다음 프로그램을 작성해봅시다. 이 프로그램은 스레드 세이프하지 않습니다. 스레드 세이프하지 않은 프로그램은 어떤 문제를 일으키는지 실행 결과를 살펴봅시다.

예제 5-3 UnsafeSample.java

```
package jp.co.bbreak.sokusen._5._2;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;

public class UnsafeSample {
    public static void main(String[] args) {
        // SimpleDateFormat 클래스는 스레드 세이프하지 않다.
        DateFormat unsafeDateFormat = new SimpleDateFormat("yyyy/MM/dd");
        // 날짜 1은 1989/03/10이다.
        Calendar cal1 = Calendar.getInstance();
        cal1.set(1989, Calendar.MARCH, 10);
        Date date1 = cal1.getTime();
        // 날짜 2는 2020/06/20이다.
        Calendar cal2 = Calendar.getInstance();
        cal2.set(2020, Calendar.JUNE, 20);
        Date date2 = cal2.getTime();

        Thread thread1 = new Thread(() -> {
            for (int i = 0; i < 100; i++) {
```

```

        try {
            String result = unsafeDateFormat.format(date1);
            System.out.println("Thread1: " + result);
        } catch (Exception e) {
            e.printStackTrace();
            break;
        }
    }
});;

Thread thread2 = new Thread(() -> {
    for (int i = 0; i < 100; i++) {
        try {
            String result = unsafeDateFormat.format(date2);
            System.out.println("Thread2: " + result);
        } catch (Exception e) {
            e.printStackTrace();
            break;
        }
    }
});;

System.out.println("스레드 세이프하지 않은 프로그램의 검증을 시작합니다.");
thread1.start();
thread2.start();
}
}

```

이 프로그램에서는 하나의 `SimpleDateFormat` 클래스를 2개의 스레드에서 동시에 사용합니다. 스레드 1에서는 오직 1989년 3월 10일을 출력하고, 스레드 2에서는 2020년 6월 20일을 출력합니다. 프로그램을 실행하면 아래와 같은 결과를 얻을 수 있습니다.

실행 결과

```

스레드 세이프하지 않는 프로그램의 검증을 시작합니다.
Thread2: 2020/06/10
Thread1: 1989/03/10
Thread2: 2020/06/20
Thread1: 1989/03/10
Thread2: 2020/06/20
Thread1: 1989/03/10
... 후략 ...

```

6 장

테스트

6.1 테스트 기초 지식

이 장에서는 소프트웨어 개발 중에서 후반 공정에 해당하는 테스트 공정에 관해 설명합니다.

6.1.1 테스트 공정이란

소프트웨어 개발 공정은 크게 아래 4가지로 나눌 수 있습니다.

- **요건 정의**: 소프트웨어에 요구하는 기능이나 성능을 정의한다.
- **설계**: 요건 정의에서 정의한 기능과 성능을 만족하는 소프트웨어 사양을 정의한다.
- **제조**: 설계에서 정의한 사양을 바탕으로 프로그램을 작성한다.
- **테스트**: 제작한 프로그램이 요건 정의와 설계 사양을 만족하는지 확인한다.

그림 6-1 소프트웨어 개발 공정

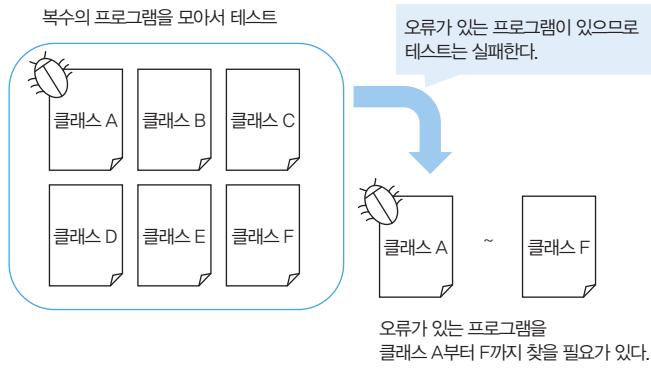


테스트 공정의 세분화

테스트 공정은 릴리스 전의 중요한 프로세스입니다. 제품을 고객에게 넘겨줄 때는 각종 테스트를 거쳐 오류가 포함되지 않았음을 증명해야 합니다. 많은 프로젝트에서 테스트 대상이 되는 프로그램의 개수와 환경에 따라 테스트 공정을 다시 세분화하여 테스트를 시행합니다.

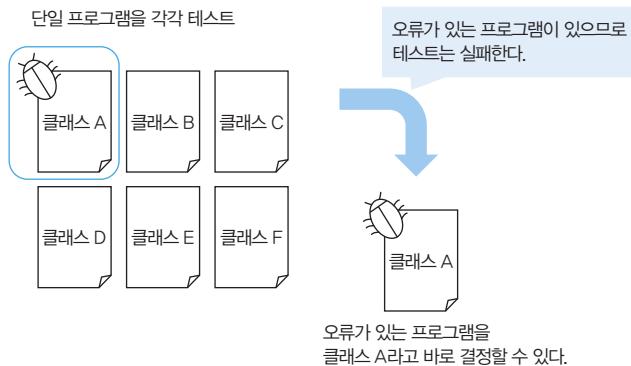
테스트의 종류

일반적인 현장에서 다루는 소프트웨어의 클래스 수와 스텝 수는 개인이 만드는 소프트웨어와 비교가 안 될 정도로 많습니다. 프로그램 가운데 하나라도 오류(버그)가 포함된 경우 방대한 클래스 중에서 그 원인을 밝혀내야 합니다. 제한된 기간과 인원으로 개발해야 하는 프로젝트에서는 그렇게 많은 시간을 들일 수 없습니다(그림 6-2).

그림 6-2 모아서 테스트하는 경우

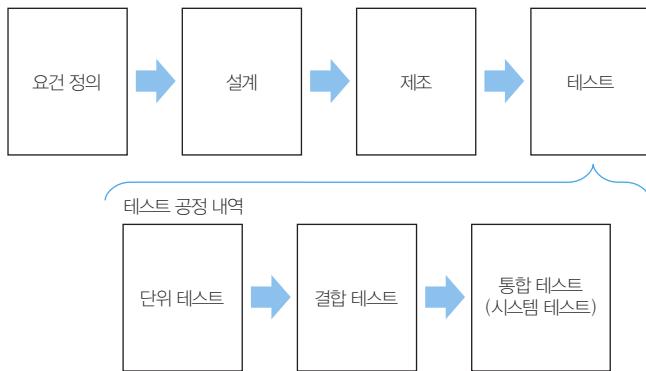
그러므로 모든 클래스를 사용해서 테스트하지 말고 우선은 클래스 하나하나를 테스트하도록 합니다. 그렇게 하면 오류가 발생해도 테스트 대상 클래스만 확인하면 되므로 원인을 빠르게 찾아낼 수 있습니다.

다음으로 단위 테스트가 완료된 클래스끼리 결합해 테스트를 시행하도록 합니다. 이와 같이 작은 규모부터 척실히 검증을 진행함으로써 모든 클래스를 결합했을 때 테스트에 걸리는 시간을 줄일 수 있습니다(그림 6-3).

그림 6-3 하나씩 테스트하는 경우

한마디로 테스트라고 해도, 테스트 방식에 따라 다양한 종류의 테스트가 존재합니다. 클래스 하나를 검증하는 단위 테스트, 검증이 끝난 클래스끼리 연결해 검증하는 결합 테스트, 모든 클래스를 연결해 검증하는 통합 테스트(시스템 테스트)가 있습니다(그림 6-4).

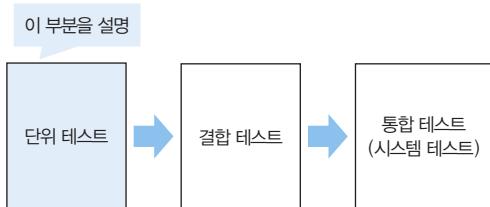
그림 6-4 테스트 공정의 세분화



6.1.2 단위 테스트

단위 테스트는 클래스를 만든 후 처음으로 실시하는 테스트입니다. 1개의 클래스를 명령 단위로 해서 동작을 확인합니다(그림 6-5).

그림 6-5 단위 테스트는 테스트 공정의 첫 번째 단계

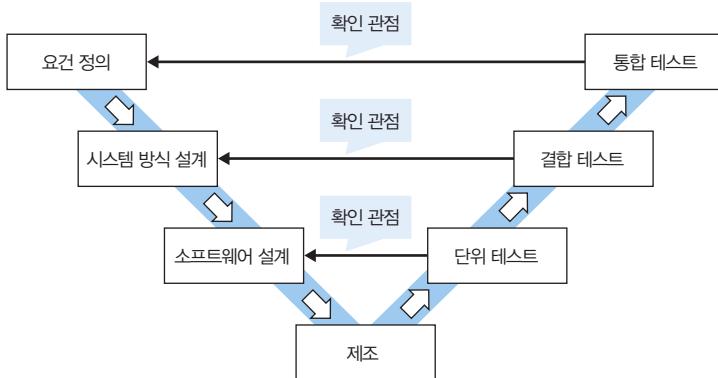


단위 테스트에서는 1개의 클래스를 검증한다

단위 테스트에서는 하나의 클래스를 대상으로 검증을 합니다. 클래스 내에 정의된 메서드를 실행하고 결과가 예상대로 나오는지 확인합니다.

어떤 동작이 예상대로 나오는지는 설계 공정에 정의되어 있습니다. 테스트 공정과 설계 공정의 관계는 다음 그림과 같이 나타낼 수 있습니다.

그림 6-6 V 모델



이 그림은 V 모델입니다. 그림을 보면 단위 테스트에서는 소프트웨어 설계(상세 설계라고 부르기도 합니다)의 내용을 확인 관점에서 테스트하고 있습니다.

소프트웨어 설계란 클래스가 동작하는 흐름을 메서드 단위로 상세히 기재한 것입니다. 이 설계에는 분기 조건과 반복 등 실제 프로그램과 가까운 내용이 기재됩니다. 단위 테스트에서는 설계에서 기재한 대로 프로그램이 동작하는지 확인합니다.

테스트에 사용하는 입력값 선택 방법

실제 클래스를 단위 테스트로 검증할 때는 클래스의 메서드 동작을 확인하는 것이 대부분입니다. 메서드에는 항상 같은 값을 출력하는 것도 있고, 입력한 값에 따라 출력이 바뀌는 것도 있습니다. 이렇게 출력이 바뀌는 메서드를 테스트할 때는 무수히 많은 입력 후보 중에서 적절한 값을 선택해 테스트할 필요가 있습니다.

입력값을 선택하는 방법은 크게 2가지가 있고, 각각 블랙박스 테스트와 화이트박스 테스트로 불립니다.

블랙박스 테스트

블랙박스 테스트에서는 메서드 입력과 출력을 검사해 테스트 케이스를 작성합니다. 구체적인 소프트웨어 설계와 프로그램을 예로 들어 설명하겠습니다. 다음은 윤년 판정 메서드입니다.

예제 6-1 윤년 판정 메서드

```
package jp.co.bbreak.sokusen._6._1;

public class LeapYear {
    public boolean isLeapYear(int year) {
        // 판정 결과
        boolean result;

        // 기원전일 때는 콘솔에 출력
        if (year < 0) {
            System.out.println("기원전입니다! ");
        }

        // 윤년 판정
        if (year % 400 == 0) {
            // 400으로 나누어떨어졌으므로 윤년
            result = true;
        } else if (year % 100 == 0) {
            // 400으로 나누어떨어지지 않고, 100으로 나누어떨어지므로 윤년이 아니다.
            result = false;
        } else if (year % 4 == 0) {
            // 400, 100으로 나누어떨어지지 않지만, 4로 나누어떨어지므로 윤년
            result = true;
        } else {
            // 윤년이 아니다.
            result = false;
        }
        return result;
    }
}
```

[표 6-1]은 윤년 판정 메서드의 소프트웨어 설계서입니다.

표 6-1 윤년 판정 메서드의 소프트웨어 설계서

No.	입력 사양	출력 사양
1	0이하	기원전으로 표시
2	400으로 나누어떨어진다.	윤년
3	No. 2에 해당하지 않지만 100으로 나누어떨어진다.	윤년이 아님
4	No. 3에 해당하지 않지만 4로 나누어떨어진다.	윤년
5	No. 4에 해당하지 않는다.	윤년이 아님

이런 설계서와 프로그램이 있다고 할 때 블랙박스 테스트 관점에서는 아래와 같은 사양을 만족하는지 확인할 수 있는 값을 입력값으로 선택합니다.

- 400으로 나누어떨어지는 경우는 윤년
- 100으로 나누어떨어지는 경우는 윤년이 아님 등

입력값을 고를 때는 한계치 분석과 동치 분할 등 2가지 방식이 주로 사용됩니다.

- 한계치 분석: 출력이 바뀌는 경계의 양쪽 값을 입력값으로 선택한다.
- 동치 분할: 그 값이 출력되는 대표적인 값을 입력값으로 선택한다.

No. 1 사양을 한계치 분석으로 테스트할 경우는 0이하의 경계인 0과 1을 입력값으로 합니다. No. 1의 사양을 동치 분할로 테스트할 경우는 0이하의 값 -100과 0이상의 값 200을 입력값으로 선택합니다.

화이트박스 테스트

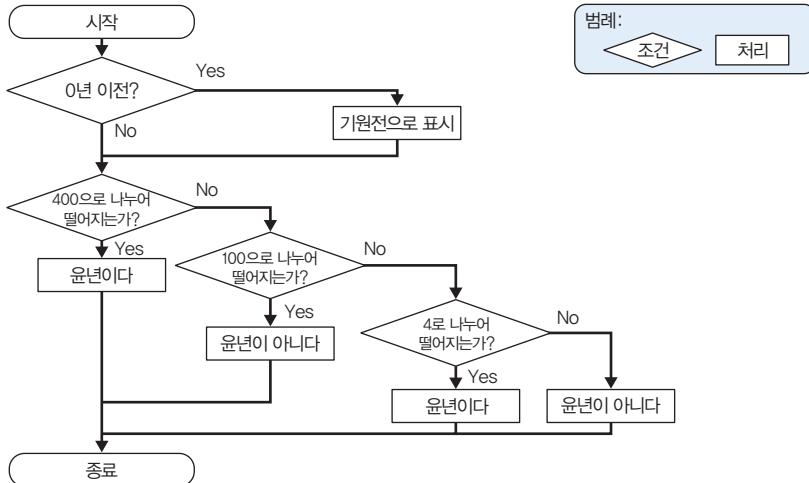
화이트박스 테스트에서는 처리 내부 구조를 해석해 테스트 케이스를 작성합니다.

블랙박스 테스트의 예와 같은 메서드로 설명하겠습니다. 화이트박스 테스트 관점에서는 프로그램의 어느 단계가 실행됐는지 의식해서 입력값을 선택합니다. 입력값을 고를 때는 아래와 같은 방식이 주로 사용됩니다.

- 명령 망라: 메서드 내의 명령을 모두 실행하도록 입력값을 선택한다.
- 판정조건 망라: 메서드 내의 조건 분기의 모든 분기가 실행되도록 입력값을 선택한다.
- 조건 망라: 메서드 내의 조건 분기의 모든 분기의 조합이 실행되도록 입력값을 선택한다.

판정조건 망라와 조건 망라는 설명만으로는 차이를 이해하기 어려울지도 모릅니다. 하지만 그림으로 보면 차이점을 쉽게 알 수 있습니다. 다음 그림은 윤년을 구하는 메서드의 동작 흐름을 나타낸 플로차트입니다.

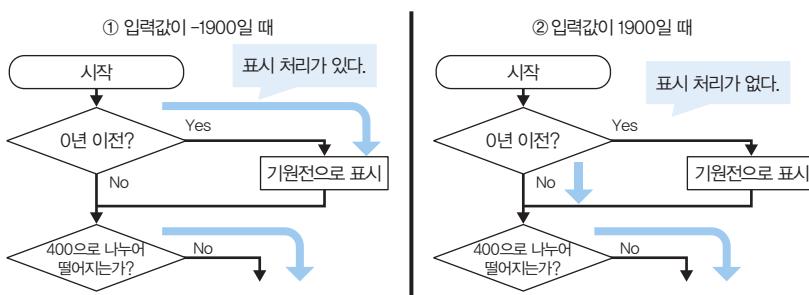
그림 6-7 윤년을 판정하는 메서드의 동작 흐름



판정조건 망라 테스트를 할 경우 -1, 2000, 1900, 2004, 2003 등 5가지 입력값을 선택합니다. 모든 값을 하나하나 그림에 적용해보면 조건의 마름모꼴에서 나온 화살표를 모두 한 번은 지나간다는 것을 알 수 있습니다.

조건 망라 테스트를 할 경우 2000, 1900, 2004, 2003, -2000, -1900, -2004, -2003 등 8가지 입력값을 선택합니다. 값을 그림에 적용해보면 시작부터 끝까지 화살표 탐색 경로의 모든 패턴이 망라되어 있다는 것을 알 수 있습니다(그림 6-8).

그림 6-8 판정조건 망리와 조건 망리의 차이



하나의 조건에 초점을 맞춰 분기를 망라하는 것이 판정조건 망리고, 시작부터 종료까지의 동작을 망라하는 것이 조건 망리입니다.

7

장

팀 개발

7.1 팀 개발이란

이 장에서는 팀 개발에 필요한 기초 지식을 설명합니다. 팀 개발은 어디에서나 하고 있지만 어디에서나 똑같은 팀 개발은 존재하지 않습니다. 성과물이나 개발 규칙, 멤버 구성부터 개발에 사용하는 도구에 이르기까지 같은 것을 찾는 것이 힘들 정도입니다.

그러므로 이 장에서는 특정 개발 규칙이나 개발 도구의 자세한 사용법이 아니라 줄기가 되는 사고방식을 설명합니다. 이 사고방식을 이해해두면 다양한 개발 현장에서 여러 상황을 만나더라도 즉시 활용할 수 있을 것입니다.

7.1.1 팀 개발이란 ‘여러 사람이 공동으로 성과물을 만들어내는 것’

자, ‘팀 개발’이란 무엇일까요?

‘우리 현장은 애자일이 아니니 팀 개발과 무관하다’거나 ‘혼자서 한 개 모듈(혹은 하나의 시스템)을 담당해서 팀 개발하고 관계없다’라고 생각해 그만 책을 덮으려는 독자가 있을지도 모릅니다.

잠깐 기다리세요. 조금만 더 읽어보세요.

이 책에서는 팀이 개발을 하는 것, 즉 복수의 사람이 공동으로 성과물을 만들어내는 것을 팀 개발이라고 부릅니다. 이런 의미에서는 기본적으로 ‘업무 애플리케이션을 포함해 시스템 개발은 모두 팀 개발’이라고 할 수 있겠지요. 오늘날 여러 사람이 관계되지 않은 시스템은 존재하지 않기 때문입니다.

그렇다면 혼자서 하나의 시스템을 담당하는 경우는 어떨까요? 이렇게 한번 생각해 보세요. 과거의 자신은 타인, 미래의 자신도 타인입니다. 앞으로도 계속 그 시스템에 자기만 관여한다는 것은 취미로 만든 프로그램이 아닌 한 현실적으로 있을 수 없습니다. 요컨대 혼자서 개발하더라도 ‘팀 개발’을 의식하는 편이 더 낫습니다.

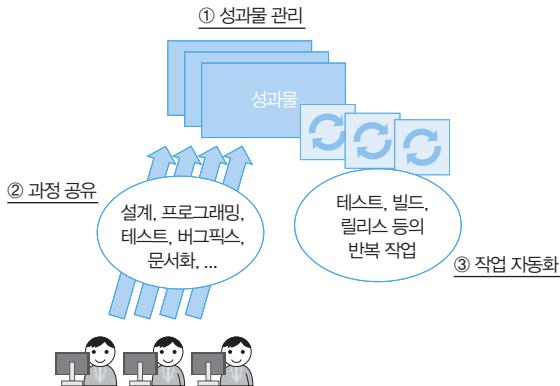
NOTE 필자가 경험한 이야기

예전에 구현 작업을 하면서 프로그램 소스를 읽다가 ‘이거 참 주석을 잘 달아놨네’라고 감탄한 적이 있었습니다. 그런데 변경 이력을 보면서 주석을 기술한 사람을 따라가 봤더니 필자 자신이었습니다. 자신이 작성한 주석을 기억하지 못했던 것입니다. 물론 그 반대도 있습니다(누가 이렇게 코드를 자자분하게 작성했냐고 화를 냈는데 정작 본인이 작성한 프로그램이었습니다).

7.1.2 팀 개발의 요점

그렇다면 팀 개발에 필요한 기초 지식은 무엇일까요? 그 기초 지식은 바로 팀 개발에서 알아둬야 하는 요점을 이해하는 것입니다. 구체적으로는 다음 그림과 같습니다.

그림 7-1 팀 개발에서 알아둬야 하는 3가지 포인트



이제부터 이 3가지에 대해 설명합니다.

성과물 관리

팀 개발의 목적은 대부분 어떤 성과물을 만들어내는 것입니다. 대표적인 성과물은 프로그램 소스와 설명서 등의 도큐먼트입니다.

성과물은 여러 사람이 동시에 편집하는 경우도 있고, 버전이 올라가면서 내용이 변경되는 경우도 있습니다. 따라서 이런 동시 편집과 내용 변경을 포함해서 성과물을 관리해야 합니다.

과정 공유

개발을 진행하기 위해서는 성과물이 완성될 때까지의 태스크를 명확히 하고 태스크를 하나하나 확실하게 진행할 필요가 있습니다. 일반적으로 이 태스크들은 WBS^{Work Breakdown Structure} (작업 분할 구조도)로 정리되어 각각의 태스크가 담당자에 의해 진행됩니다. 팀 개발에서는 태스크 진척 상황뿐만 아니라 ‘언제, 누가, 어떻게 진행했는지’를 공유하는 것이 중요합니다.

- 어떤 우선순위로 진행할 것인가
- 개개의 태스크가 각각 어떤 성황인가
- 진행한 결과가 어떻게 됐는가

개발을 진행하다 보면 계획된 태스크 이외에 해결해야 할 과제나 장애가 되는 문제점 등이 발생해 원래 계획하지 않았던 태스크도 생깁니다. 이런 태스크에 대해서도 위와 같은 과정을 관리하고 공유하는 것이 중요합니다.

작업 자동화

개발을 진행하는 과정에선 반복해서 하는 작업이 자주 발생합니다. 예를 들어 프로그램 소스를 빌드하고 테스트 환경에 릴리스하는 작업은 대부분의 개발 현장에서 반복적으로 일어납니다.

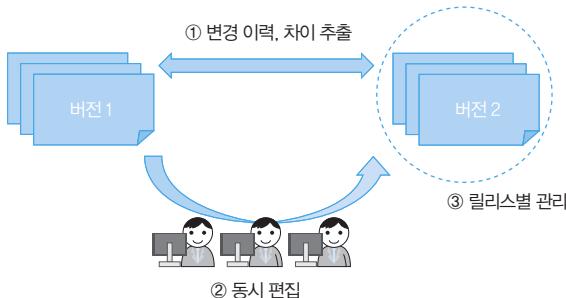
반복 작업을 할 때 중요한 점은 더 효율적으로 작업을 할 수 있게 하는 것입니다. 작업 품질을 유지한 채로 효율화해야 하는 것은 당연해서 말할 필요도 없습니다.

그러기 위해서는 특정인에 대한 의존을 배제하는 형태로 작업 절차를 정리해두고 그 절차가 항상 최신이 되게 할 필요가 있습니다. 그리고 효율화의 궁극의 형태는 바로 자동화입니다.

7.2 성과물 관리 - 버전 관리

여기서는 성과물 관리에 대해 설명합니다. 도대체 성과물 관리란 무엇을 하는 것일까요? 무엇을 할 수 있어야 성과물이 관리된다고 할 수 있는지 알아보겠습니다. 성과물이 관리되고 있는지는 다음 그림에 나타낸 3가지를 할 수 있느냐에 따라 결정됩니다.

그림 7-2 성과물 관리에서 해야 할 3가지



변경 이력, 차이 추출

성과물을 만들어내는 과정에서 보통은 성과물에 이런저런 변경을 가하게 됩니다. 그래서 언제, 누가, 어떻게 손을 댔는지 정확하게 또한 기계 처리 가능한 형태로 남겨두는 것이 중요합니다. 특히 중요한 것은 어떻게 변경했는지 정확하게 또한 기계 처리 가능한 형태로 남기는 일입니다. 그렇게 하면 차이를 자동으로 추출할 수 있다는 커다란 장점이 생깁니다.¹

최신 버전과 직전 버전의 차이는 물론이고, 1년 전 버전과의 차이나 1단계의 납품물과 2단계의 납품물의 차이 등도 간단히 볼 수 있습니다. 또한 변경 이력과 차이 정보가 컴퓨터로 처리할 수 있는 형식이라면 프로그램 소스에 자동으로 반영할 수도 있습니다.

최근에는 어느 개발 현장에서나 버전 관리 시스템을 도입함으로써 변경 이력과 차이를 정확하게 또한 기계 처리 가능한 형태로 남길 수 있게 되었습니다.

동시 편집

시스템을 개발할 때는 여러 사람이 팀을 이뤄 작업합니다. 물론 UI 부분이나 데이터 액세스 부분 등으로 계층마다 담당을 할당하기도 해서, 하나의 시스템이라고 하지만 패키지와 클래스가 분리되어 완전히 똑같은 클래스(파일)를 동시에 편집하는 일은 많지 않을지도 모릅니다. 하지만 그렇다고 해서 하나의 파일을 여러 사람이 동시에 편집할 수 없으면 편집이 가능할 때까지 기다려야 하겠지요. 또한 동시에 편집해버린 탓에 재작업이 발생하거나 하면 개발 효율 저하를 피할 수 없습니다.

최악의 상황은 동시 편집이 원인이 되어 내용 자체가 엉망이 되는 것입니다. 팀 규모와 시스템 규모가 커지면 영향 정도도 더욱 커집니다. 그래서 개발 현장에서는 클래스와 파일을 여러 사람이 동시에 편집할 수 있도록 버전 관리 시스템을 도입해 성과물을 관리합니다.

릴리스별 관리

현재 PC는 파일 단위로 데이터를 관리하는 구조로 되어 있고, 우리도 그런 구조에 익숙합니다. 예를 들어 변경 이력을 파일 단위로 남기거나 파일명에 버전 번호를 붙

¹ 표현의 차이긴 하지만, 실제로는 차이를 추출한 형식으로 변경 이력을 남기는 경우가 많겠지요.

이는데, 어느 쪽이든 파일별로 관리하는 것이 일반적입니다.

하지만 시스템 개발에서 중요한 것은 파일이 아니라 릴리스물입니다. 예를 들어 개발 현장에서는 파일별 변경 이력이나 차이점을 보려는 것이 아니라 지난주 릴리스물과 금주 릴리스물의 차이를 관리하고 싶어 합니다.

또한 시스템을 개발할 때 복수의 버전을 동시에 개발하는 사례도 적지 않습니다. 예를 들어 이미 가동 중인 버전의 버그픽스와 다음 버전의 기능 확장을 같은 기간에 개발합니다. 이때에도 가동 중인 버전과 기능 확장 버전의 일관성을 유지하고 성과물의 내용을 적절하게 관리할 수 있어야 합니다.

버전 관리 시스템은 복수의 파일을 모아 릴리스별로 관리할 수 있고, 복수 버전의 관리를 도와주는 기능도 제공합니다. 많은 개발 현장에서는 버전 관리 시스템을 도입해 릴리스를 관리합니다.

7.2.1 버전 관리 시스템 = 성과물 관리 수단

이 절에서는 버전 관리 시스템으로 어떻게 성과물을 관리하는지 구체적으로 살펴보겠습니다. 그에 앞서 버전 관리 시스템에서 사용되는 일반적인 용어를 설명합니다.

리포지터리

‘저장소’라는 의미입니다. 버전 관리 시스템에서는 프로그램 소스 등 데이터를 저장해둔 장소와 영역을 의미합니다. 리포지터리에는 파일 자체는 물론이고 과거 변경 이력이 모두 축적됩니다.

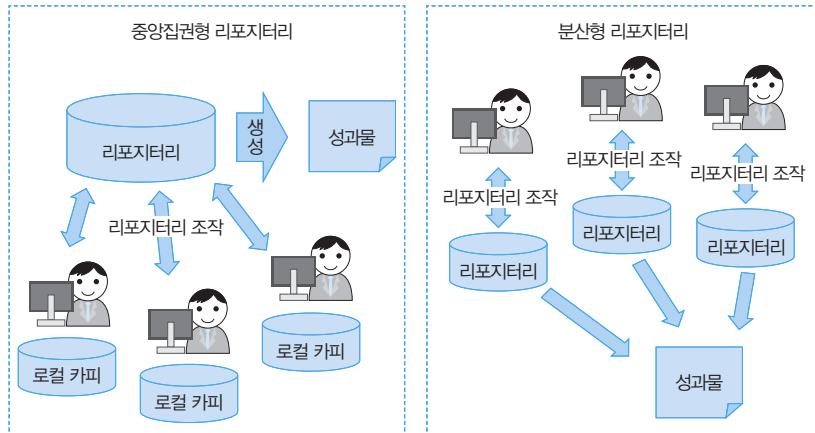
버전 관리 시스템은 중앙집권형 리포지터리와 분산형 리포지터리로 분류할 수 있습니다.

Subversion²으로 대표되며, 서버상에 하나의 리포지터리를 가지고 개발자가 그 리포지터리를 조작하는 것이 중앙집권형입니다. 한편 최근 빠르게 보급되는 Git³은 분산형입니다. 분산형은 각 개발자가 자신의 리포지터리를 가지며 그 리포지터리를 조작해 개발을 진행합니다(그림 7-3).

² <https://subversion.apache.org/>

³ <http://git-scm.com/>

그림 7-3 중앙집권형 리포지터리와 분산형 리포지터리

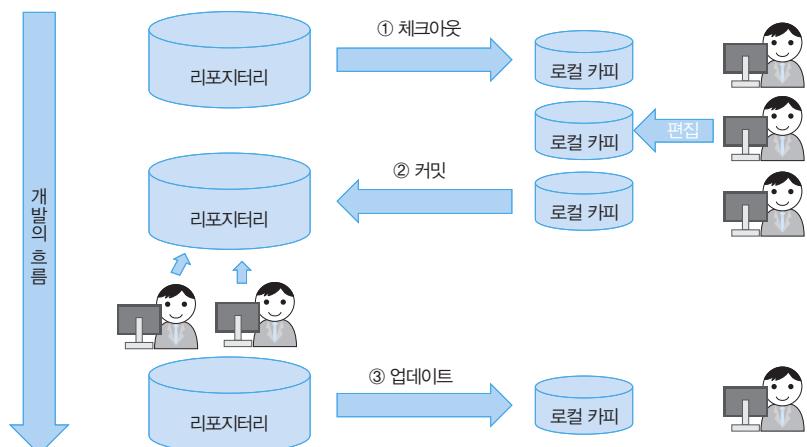


체크아웃, 커밋, 업데이트

모두 Subversion의 리포지터리 조작입니다. 체크아웃은 리포지터리의 내용을 로컬 PC에 다운로드하는 것입니다. 체크아웃된 로컬 PC의 데이터는 로컬 카피로 불립니다. 커밋은 로컬 카피의 내용을 리포지터리에 반영합니다. 업데이트는 리포지터리의 내용을 로컬 카피에 반영합니다.

다음 그림은 리포지터리를 조작하는 일련의 흐름을 보여줍니다.

그림 7-4 리포지터리 조작의 흐름



그런데 여러 사람이 같은 파일의 로컬 카피를 각각 변경하고 커밋했다면 어떻게 될까요? 이 경우 모든 사람의 변경 내용이 제대로 커밋됩니다.

그 이유는 버전 관리 시스템이 머지 모델을 채용했기 때문입니다. 머지 모델은 로컬 카피와 리포지터리 정보에서 차이를 추출해 로컬 카피의 변경분만 리포지터리에 반영합니다(그림 7-5).

반면 락 모델은 같은 파일을 복수의 사람이 동시에 편집할 수 없습니다. 이때는 로컬 카피를 변경하기 전에 미리 락을 얻어 편집하고, 커밋과 동시에 락을 해제하는 조작이 필요합니다(그림 7-6).

그림 7-5 머지 모델

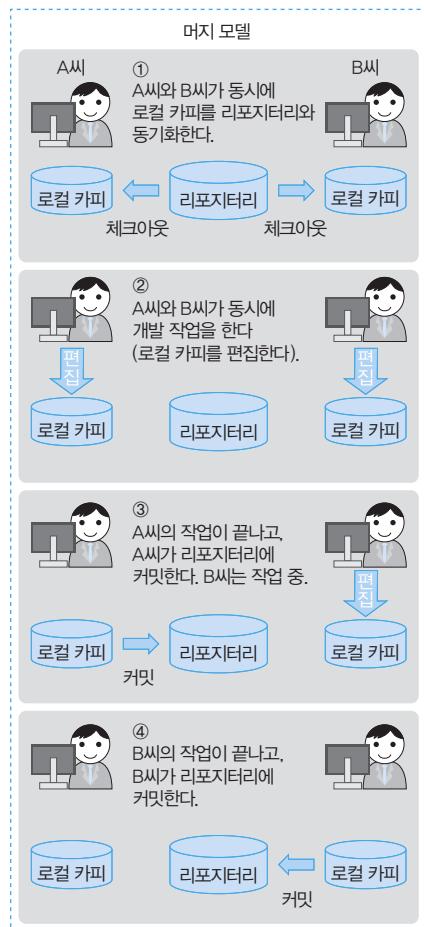


그림 7-6 락 모델



리비전

커밋 등으로 리포지터리가 변경될 때마다 변경 후 상태에 이력 번호가 붙습니다. 이를 이력 번호를 리비전이라고 합니다.

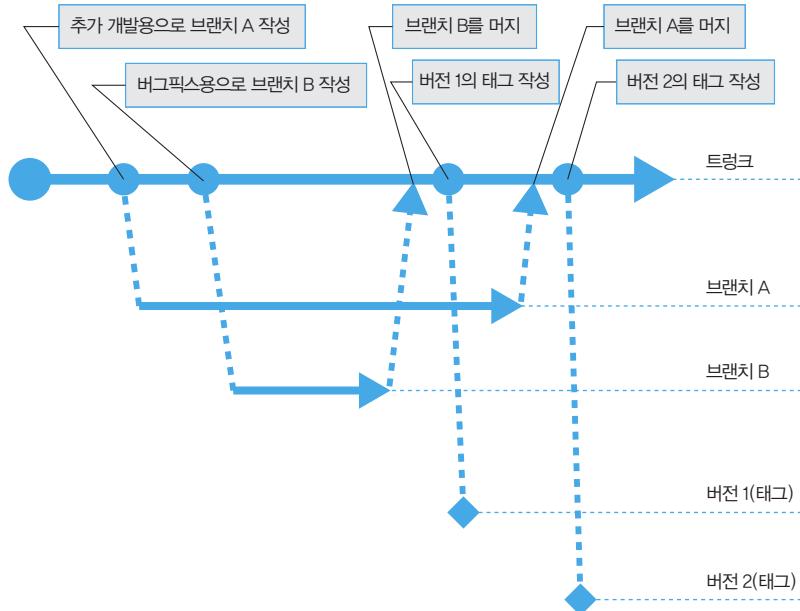
트렁크, 브랜치, 태그

트렁크 trunk는 ‘원줄기’에 해당하는 브랜치입니다. 원줄기란 현재 메인 개발이 진행되는 브랜치, 최신 버전 브랜치라는 뜻입니다.

브랜치^{branch}는 ‘가지’라는 뜻입니다. 트렁크와 별개로 개발을 진행하고 싶은 경우 트렁크에서 브랜치를 만듭니다.

태그^{tag}는 어떤 시점의 브랜치 상태를 스냅샷처럼 찍어두기 위한 표시입니다. 예를 들어 버전 1을 릴리스한 경우에는 릴리스한 브랜치에 대해 태그를 잘라두는 식으로 사용합니다(그림 7-7).

그림 7-7 브랜치와 태그 사용법



7.2.2 Subversion 조작

여기서는 Subversion을 조작하는 구체적인 예를 들어보겠습니다. 앞에서도 언급한 것처럼 Subversion은 중앙집권형 리포지터리의 디폴트 표준이고, 다수의 현장에서 채용되고 있습니다.

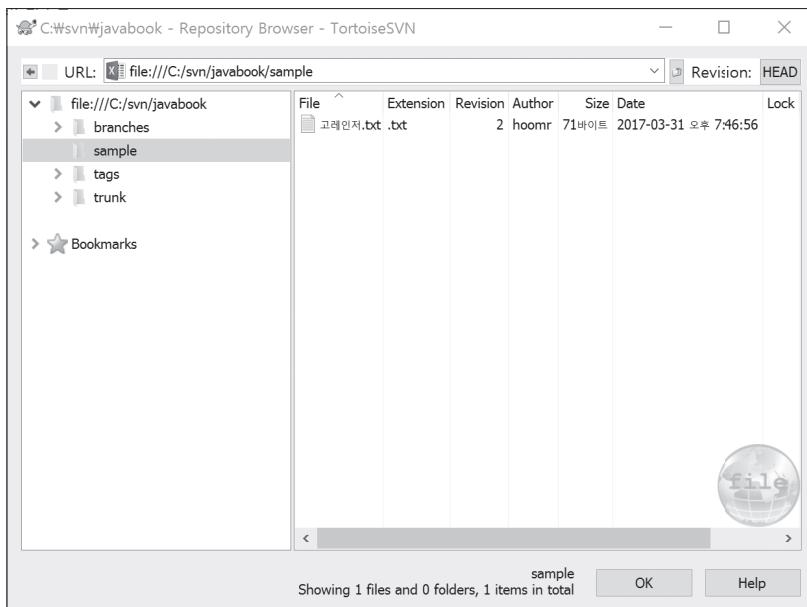
중앙집권형 리포지터리는 분산형 리포지터리보다 자유도가 낮지만, 최소한의 운영 규칙으로 간편하게 성과물을 관리할 수 있고 비교적 얻을 수 있는 이익도 커서 개발 현장에서 중요시되고 있습니다.

예제에서는 Subversion의 클라이언트로서 TortoiseSVN⁴을 사용합니다. 개발 환경에서는 Subversive⁵와 같은 이클립스에서 동작하는 Subversion 클라이언트도 많이 사용합니다.

우선 리포지터리를 참조해봅시다. TortoiseSVN의 리포지터리 브라우저 기능을 사용하면 리포지터리 안의 파일을 직접 참조할 수 있습니다.

다음 그림과 같이 리포지터리 내 sample이라는 폴더를 체크아웃하면 sample 폴더 아래의 내용을 로컬 파일로 참조할 수 있게 됩니다.

그림 7-8 리포지터리 참조

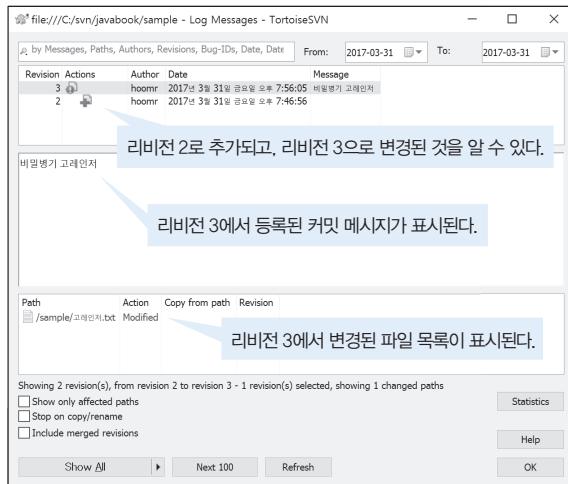


4 <https://tortoisessvn.net/>

5 <http://www.eclipse.org/subversive/>

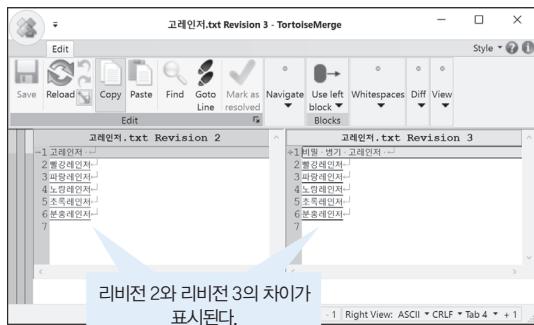
다음 그림에서 리포지터리 상의 파일 이력을 보면 파일이 만들어지고 1회 갱신됐습니다. 이번 갱신에서는 하나의 파일만 갱신되었습니다(동시에 여러 갱신이 커밋된 경우에는 동시에 변경된 파일도 파악할 수 있게 되어 있습니다).

그림 7-9 파일 이력(변경 전 상태)



다음 그림과 같이 변경된 각 파일에서 어떻게 변경됐는지 그 내용을 참조할 수도 있습니다.

그림 7-10 파일의 변경 내역



로컬 파일에서 편집 작업이 끝나면 리포지터리에 커밋합니다. 이때 다음 그림과 같이 커밋 메시지 입력을 요구합니다.

기초부터 테스트, 팀 개발까지 실무에 최적화한 자바 입문서

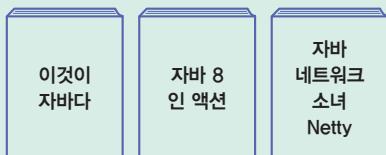
이 책은 현장 엔지니어로 활동하며 신입사원을 교육한 저자들이 실무에 꼭 필요한 내용만 담아낸 자바 입문서입니다. 이 책의 대상 독자는 다음과 같습니다.

- 자바를 전혀 다뤄보지 않은 사람
- 다른 프로그래밍 언어는 다뤄봤지만 자바는 다뤄보지 않은 사람
- 자바 초보인데 자바로 애플리케이션 개발 업무를 맡게 된 사람

책에서 다루는 내용



관련 도서



프로그래밍 언어 / 자바



예제 소스 <http://www.hanbit.co.kr/src/2862>

정가 28,000원