

비전공자를 위한 자바 프로그래밍

강희은 지음

영어 독해하듯 코드를 읽고
그림으로 쉽게 배우는



비전공자를 위한 자바 프로그래밍

강희은 지음

영어 독해하듯 코드를 읽고
그림으로 쉽게 배우는



비전공자를 위한 자바 프로그래밍 영어 독해하듯 코드를 읽고 그림으로 쉽게 배우는

전자책 발행 2016년 2월 26일

1차 업데이트 2016년 4월 20일

지은이 강희은 / **펴낸이** 김태현

펴낸곳 한빛미디어(주) / **주소** 서울시 마포구 양화로 7길 83 한빛미디어(주) IT출판부

전화 02-325-5544 / **팩스** 02-336-7124

등록 1999년 6월 24일 제10-1779호

ISBN 978-89-6848-800-9 15000 / **정가** 12,800원

총괄 전태호 / **책임편집** 김창수 / **기획·편집** 정지연 / **교정** 이미연

디자인 표지/내지 여동일

마케팅 박상용, 송경석, 벤지영 / **영업** 김형진, 김진불, 조유미

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 **한빛미디어(주)**의 홈페이지나 아래 이메일로 알려주십시오.

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

Published by HANBIT Media, Inc. Printed in Korea

Copyright © 2016 강희은 & HANBIT Media, Inc.

이 책의 저작권은 강희은과 **한빛미디어(주)**에 있습니다.

저작권법에 의해 보호를 받는 저작물이므로 무단 복제 및 무단 전재를 금합니다.

지금 하지 않으면 할 수 없는 일이 있습니다.

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

한빛미디어(주)는 여러분의 소중한 경험과 지식을 기다리고 있습니다.

소개

지은이_ 강희은

경영학부를 졸업 후 IT 업계에 사무직으로 종사하다가 프로그래밍에 흥미를 느껴 컴퓨터 공학 대학원에 진학하였다. 아직 대학원생이고 많은 부족함이 있다고 생각하지만, 스스로 공부하며 체득한 컴퓨터 언어를 쉽게 이해하는 방법을 다른 사람들과도 공유할 기회를 얻고 싶어 책을 집필하게 되었다. 현재는 대학원의 한 연구실에서 경영학과 컴퓨터 공학 간의 융합에 대해 고민하고 연구하는 중이다. 한적하고 넓은 공간을 좋아하고 커피보다는 차를 선호한다. 스트레스를 받으면 매운 버섯 샤부샤부를 먹으며, 춤추기 좋은 노래를 즐겨 듣고 장래에는 1년 365일 따뜻한 곳에서 살고 싶은 꿈이 있다.

감수_ 박성용

서강대학교 컴퓨터공학과 교수로 재직 중이며, 연구 분야는 분산 처리와 클라우드 그리고 임베디드 시스템이다.

저자의 말

최근 융합 소프트웨어에 대한 관심이 높아지면서 유명 기업들이 이공계 학생에게는 인문학에 대한 소양을, 인문계 학생에게는 컴퓨터 프로그래밍에 대한 능력을 요구하고 있습니다. 이러한 흐름에 따라 많은 인문계 학생이 컴퓨터 프로그래밍을 배우기 위해 학원에 다니거나 독학하거나 학교 내 컴퓨터공학과 수업을 듣고 있습니다. 이 책은 이러한 상황에 놓인 인문계 학생을 대상으로 한 자바 프로그래밍 입문서입니다.

기존에도 프로그래밍 입문서는 다양하게 있었습니다. 하지만 초보자가 입문서를 보며 혼자 공부하는 것은 생각보다 쉽지 않습니다. 특히 컴퓨터 언어에 대해 생소하고, 이 공계 학생과는 공부 성향이 다른 인문계 학생에게 기존의 입문서는 너무 어렵게 다가올 수 있습니다. 저 역시 인문계 졸업생으로서 처음 프로그래밍 언어를 배울 때 그렇게 느꼈습니다. 그래서 프로그래밍의 재미를 느끼고 컴퓨터 공학 대학원에까지 진학하게 된 저의 경험을 바탕으로 어떻게 하면 인문계 학생에게 더 친근한 공부 방법을 알려 줄 수 있을지 고민해 보았습니다.

그렇게 고민한 끝에 생각한 공부 방법이 바로 영어를 공부하듯 자비를 공부하는 것입니다. 영어는 언어 중 하나고 우리는 영어권 사용자와 소통하기 위해 영어를 배웁니다. 마찬가지로 자바는 프로그래밍 언어 중 하나며 우리가 컴퓨터와 소통하기 위해 배우는 것입니다. 영어를 공부할 때 문법을 배우고 단어를 외우고 어순에 따라 해석하고 영작하듯, 자바를 공부할 때도 문법을 배우고 단어를 외우고 어순에 맞게 코드(프로그래밍 언어로 기술한 글)를 작성합니다. 이와 같은 점에 착안하여 이 책에서는 자바 코드를 설명할 때 영어를 번역하듯 설명하고, 주요 단어를 영어 단어 외우듯 공부함으로써 영작하는 것처럼 자바 프로그래밍을 할 수 있게 하였습니다. 또한, 자바를 조금 더 친숙하게 공부할 수 있도록 자바의 주요 개념을 일상생활에 빗대고, 조금 더 쉽게 이해할 수 있도록 그림을 곁들여 설명하였습니다. 이 책을 통해 프로그래밍을 공부하려는 인

문계 학생뿐만 아니라 처음 프로그래밍 언어를 접하는 분이 자바 프로그래밍에 대해 조금 더 쉽게 이해할 수 있기를 희망합니다.

마지막으로 집필에 많은 도움을 주신 한빛미디어의 김창수, 정지연, 김상민 님께 감사드립니다. 또한, 새로운 길을 감에 있어 어려움을 극복하고 잘 적응할 수 있도록 물심 양면 이끌어 주신 박성용 교수님과 연구실 식구들, 항상 응원하고 격려해 주시는 엄마, 아빠, 희진 언니, 형부와 비타민 채원, 채율 그리고 친구들에게 감사의 말을 전합니다.

감수자의 말

프로그래밍의 기본 개념과 자바의 개념을 일상생활에 비유하여 표현하고, 영어 독해 하듯 코드를 설명하여 프로그래밍을 처음 접하는 비전공자와 초급 사용자가 배우기 쉬운 좋은 책이라 생각합니다. 고급 프로그래밍을 하기 전에 이 책으로 기본 개념을 쌓고, 이후 고급 자바 책 등으로 심화학습을 하면 좋을 것입니다.



이 책은 기존에 프로그래밍을 접해보지 못한 독자를 대상으로 합니다. 또한, 프로그래밍을 배우려고 시도했지만, 복잡하고 어려워서 접근하기 어려웠던 분들이 보셔도 괜찮습니다. 컴퓨터나 프로그래밍에 대한 사전 지식과 경험이 없어도 해당 책을 읽는 데 큰 문제가 없습니다.

이 책의 예제는 32비트 운영체제인 Windows x86을 사용하는 컴퓨터에서 Java SE 8u45 버전의 JDK와 Eclipse IDE for Java EE Developers를 사용하여 작성되었으며, JDK와 Eclipse 버전에 따라 코드가 동작하지 않거나 책에 적힌 설명과 사용법이 다를 수도 있습니다.

이 책에서 사용한 코드는 다음에서 다운로드할 수 있습니다.

- <https://www.hanbit.co.kr/exam/2800/>

chapter 1 자바 프로그래밍을 시작하면서 —— 011

1.1 컴퓨터의 기초 —— 011
1.2 자바란 —— 014
1.3 자바 프로그램의 작동 원리 —— 015
1.3.1 컴파일과 실행 —— 015
1.3.2 자바 가상 머신 —— 016
1.4 자바 개발환경 구축하기 —— 017
1.4.1 JDK 설치 —— 017
1.4.2 이클립스 설치 —— 025
1.4.3 Welcome to Java World! —— 030

chapter 2 기초 문법 —— 037

2.1 기초 문법 —— 037
2.1.1 데이터 타입 —— 037
2.1.2 변수 —— 039
2.2 여러 가지 연산자 —— 043
2.2.1 산술 연산자 —— 043
2.2.2 비교 연산자 —— 047
2.2.3 논리 연산자 —— 048
2.2.4 데이터 타입의 변환 —— 050
2.2.5 연산자의 우선순위 —— 054
2.3 여러 가지 배열 —— 056
2.3.1 배열 —— 056
2.3.2 다차원 배열 —— 058
2.4 반복문 —— 062
2.4.1 for문 —— 062
2.4.2 while문 —— 066



2.5 조건문 ———	071
2.5.1 if문 ———	071
2.5.2 switch문 ———	075

chapter 3 **클래스와 객체** ——— 079

3.1 클래스와 객체의 개념 ———	079
3.2 클래스 선언하기 ———	083
3.3 객체의 생성과 사용 ———	084
3.4 메서드 ———	086
3.5 생성자 ———	090

chapter 4 **클래스의 상속** ——— 097

4.1 상속의 정의 ———	097
4.2 오버라이딩 ———	101
4.3 여러 가지 수식자 ———	104
4.3.1 final ———	105
4.3.2 static ———	106
4.4 추상 클래스 ———	110
4.5 인터페이스 ———	113
4.6 다형성 ———	117

chapter 5 **패키지와 접근제어** ——— 123

5.1 패키지 ———	123
5.2 접근제어 ———	127



chapter 6 입출력 —— 131

6.1	입출력이란	131
6.2	스트림	132
6.3	파일 쓰고 읽기	135
6.3.1	파일 쓰기	137
6.3.2	파일 읽기	141
6.4	키보드로 입력하기	144

chapter 7 객체의 직렬화 —— 151

7.1	직렬화와 역직렬화	151
7.2	직렬화 클래스 만들기	152
7.3	객체를 직렬화/역직렬화하는 방법	154

chapter 8 멀티스레드 —— 161

8.1	멀티스레드란	161
8.2	멀티스레드 작성 방법	163
8.3	스레드 간의 통신	169

chapter 9 네트워크 —— 177

9.1	네트워크의 기초	177
9.1.1	IP 주소와 포트 번호	180
9.1.2	DNS	183
9.1.3	클라이언트와 서버	184



9.2 네트워크 통신 프로그래밍	187
9.2.1 인터넷을 통해 데이터를 주고받을 때 필요한 것들	187
9.2.2 TCP/IP 프로그래밍	189
9.2.3 스레드를 이용한 채팅 프로그램	197

chapter 10 편리한 함수들 205

10.1 수학 관련 함수	205
10.2 문자 관련 함수	207
10.3 시간 관련 함수	209

마무리하며 213

부록 연습문제와 답 215

자바 프로그래밍을 시작하면서

1장에서는 자바로 프로그램을 만들기 전에 기본적으로 알아야 할 내용을 다룹니다. ‘1.1 컴퓨터의 기초’에서는 제가 처음 프로그래밍을 시작할 때 궁금했던 컴퓨터의 기초 지식에 대해 이야기합니다. ‘1.2 자바란’에서는 대표적인 프로그래밍 언어인 자바의 정의와 역사를 다룹니다. ‘1.3 자바프로그램의 작동 원리’에서는 자바로 작성한 프로그램이 어떻게 실행되는지 그 원리를 간단히 배웁니다. 자바로 프로그램을 작성하기 전에 필요한 기초 환경을 구성하는 방법은 ‘1.4 자바 개발환경 구축하기’에 있으니 참고하시기 바랍니다.

1.1 컴퓨터의 기초

요즘에는 어디서든 컴퓨터를 볼 수 있습니다. 집에도 있고 학교에도 있고 언제나 들고 다니는 스마트폰도 컴퓨터입니다.

그림 1-1 컴퓨터의 다양한 종류

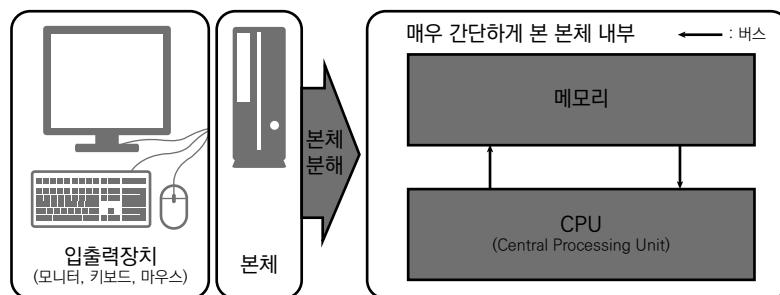


컴퓨터는 일종의 자동 계산기라고 볼 수 있습니다. 간단하게는 숫자 계산을 하고, 더 나아가서는 이메일을 보내고, 문서를 작성하고, 동영상을 보는 데 쓰기도 합니

다. 전기를 연결한 기계일 뿐인데 컴퓨터는 어떻게 이런 다양한 작업을 할 수 있을까요? 이를 알기 위해서는 컴퓨터의 구조를 먼저 보아야 합니다.

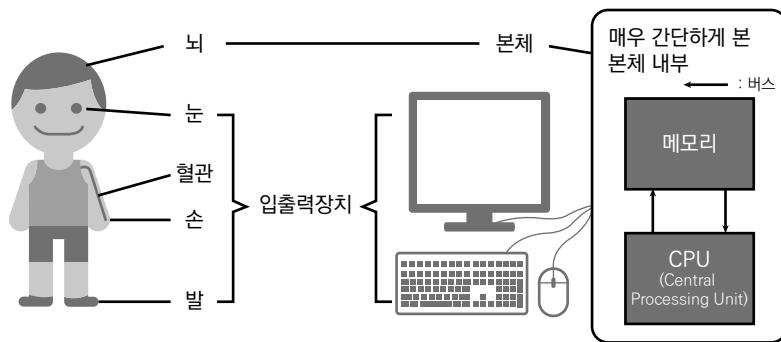
컴퓨터는 크게 본체와 각종 입출력장치로 이루어져 있습니다. 본체 내부를 살펴보면 여러 가지 장치가 있는데, 그중에서 핵심이 되는 것은 메모리와 CPU입니다. 메모리는 프로그램 및 데이터를 저장하는 역할을 하며, 보조 기억장치(하드디스크, RAM 등)가 그 종류입니다. CPU는 명령이 들어오는 경우 이를 계산하고 실행하는 역할을 합니다. 키보드, 마우스 등은 입력장치로 사용자가 데이터를 입력할 수 있는 도구입니다. 컴퓨터로 문서를 작성할 때 키보드로 글자를 입력하거나 윈도우 바탕화면에서 아이콘을 더블 클릭하는 것을 생각하면 쉽습니다. 모니터는 출력장치입니다. 컴퓨터의 본체에서 계산된 결과를 나타내 주는데, 문서를 작성할 때 키보드로 글자를 입력하면 그 내용을 모니터 화면으로 보거나 바탕화면 아이콘을 더블 클릭하면 해당 프로그램이 모니터 화면에 뜨는 것이 그 예입니다. 마지막으로 [그림 1-2]에서 화살표로 표시된 버스는 데이터와 명령이 컴퓨터 안의 다른 장치로 전달되는 통로 역할을 합니다.

그림 1-2 컴퓨터의 구조



컴퓨터는 사람과 매우 비슷한 구조를 가졌습니다. 컴퓨터를 사람의 신체에 비교하면, 본체는 뇌, 버스는 혈관, 입출력장치는 눈, 입, 손, 다리 등이 될 수 있습니다. 본체에 CPU와 메모리가 있는 것은 사람의 뇌에는 기억을 저장하는 부분과 생각하는 부분이 있는 것과 같습니다.

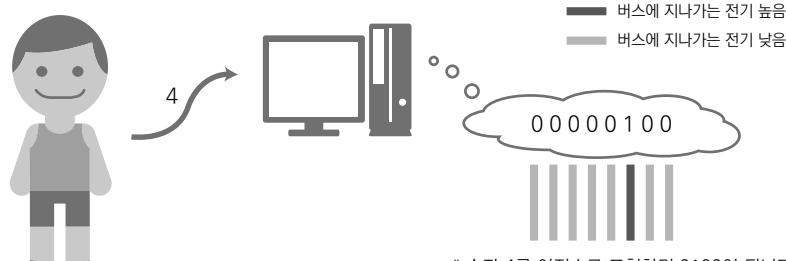
그림 1-3 컴퓨터와 사람의 유사성



사람은 산소나 음식을 섭취하여 영양소로 분해한 후 혈관을 통해 이들을 몸속 각 부분으로 전달합니다. 이와 비슷한 원리로, 컴퓨터도 전기를 공급받아서 버스를 통해 데이터나 명령을 각 장치에 전달합니다. 따라서 컴퓨터 내 데이터와 명령은 모두 전기로 표현되어야 합니다. 컴퓨터는 전기가 낮을 때를 0, 높을 때를 1로 인식합니다. 즉, 컴퓨터의 언어(기계어)는 0과 1로 나타나는 2진수가 됩니다.

그렇지만 0과 1로 표현되는 기계어를 사람들이 읽고 쓰는 것이 매우 불편하기 때문에, 사람들은 컴퓨터를 작동하기 위한 소프트웨어를 작성할 때 ‘프로그래밍 언어’를 사용합니다. 다양한 프로그래밍 언어 중 하나가 바로 자바입니다.

그림 1-4 컴퓨터 언어(기계어)

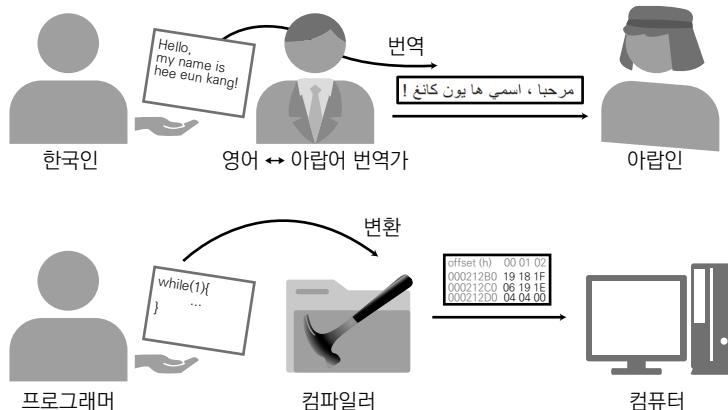


* 숫자 4를 이진수로 표현하면 0100이 됩니다.

프로그래밍 언어는 일상 언어, 특히 영어 단어를 주로 사용합니다. 언어가 단어와 문법을 가진 것처럼, 프로그래밍 언어 역시 단어와 문법으로 구성되어 있습니다.

프로그래밍 언어로 작성한 내용을 기계어로 변환하는 것을 ‘컴파일(Compile)’이라고 합니다. 기계어로의 번역이라고 생각하면 이해하기 쉽습니다. 사람이 프로그래밍 언어로 프로그램 코드를 작성하면, 이를 ‘컴파일러’라는 번역기가 0과 1로 구성된 기계어로 변환하고, 컴퓨터는 이 변환된 프로그램 코드를 실행합니다. 이 원리는 마치 한국 사람이 아랍어를 읽고 쓰는 것이 매우 어려워서 비교적 배우기 편한 영어로 하고 싶은 말을 글로 작성한 후에 영어 ⇔ 아랍어 번역가에게 부탁하여 이 글을 아랍어로 번역하는 것과 비슷합니다.

그림 1-5 프로그래밍 언어를 기계어로 변환하는 과정



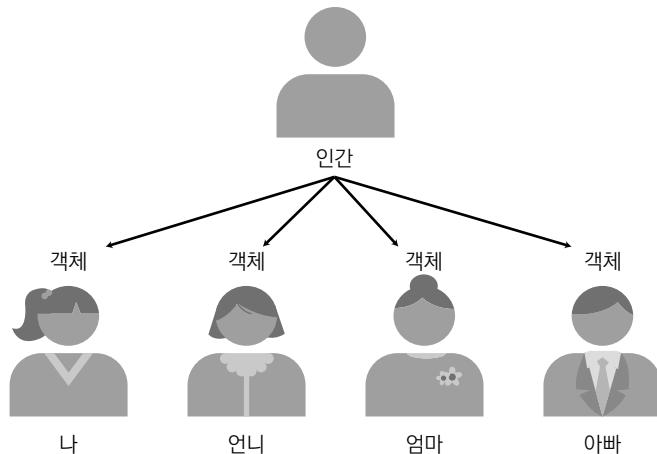
1.2 자바란

자바는 대표적인 객체 지향 프로그래밍 언어로, 지금은 Oracle에 인수된 Sun Microsystems라는 미국의 컴퓨터 회사에 의해 1995년에 발표되었습니다. 자바는 원래 가전제품에 들어갈 소프트웨어를 개발하기 위한 용도였지만, 지금은 인터넷 및 안드로이드 애플리케이션 개발에도 많이 사용하고 있습니다.

자바: 객체 지향 프로그래밍 언어

앞서 말했듯 프로그래밍 언어는 컴퓨터를 작동하기 위한 소프트웨어를 작성할 때 사용하는 언어를 말합니다. 그렇다면 ‘객체 지향’은 무엇일까요? 객체를 지향한다는 것은 프로그래밍할 때 각각 독립적으로 작동하는 객체를 만들고 이들의 관계를 설계해간다는 의미입니다. 객체란 실체입니다. 예를 들어, 나, 언니, 엄마, 아빠는 ‘인간’의 실체(객체)라 할 수 있습니다. 객체에 대한 내용은 3장에서 자세히 다루도록 하겠습니다.

그림 1-6 객체의 개념

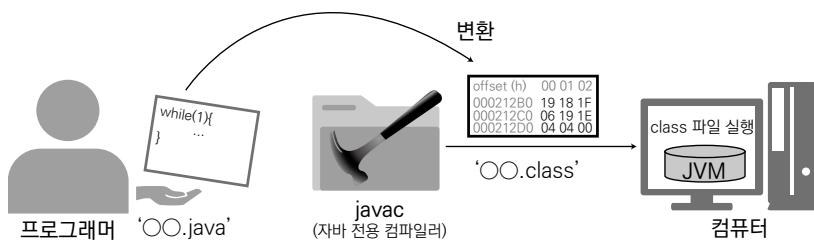


1.3 자바 프로그램의 작동 원리

1.3.1 컴파일과 실행

자바로 작성한 코드는 '.java'라는 확장자를 가지고 있습니다. 자바에는 '.java'로 끝나는 자바 코드를 컴퓨터가 이해할 수 있는 기계어로 변경해 주는 'javac'라는 이름의 전용 컴파일러가 있습니다. 'javac'로 프로그래밍 언어를 기계어로 변경하면 '.class'라는 확장자를 가진 파일이 생성됩니다. '.class' 파일은 자바 가상 머신, 즉 JVM^{Java Virtual Machine}을 통해 실행됩니다.

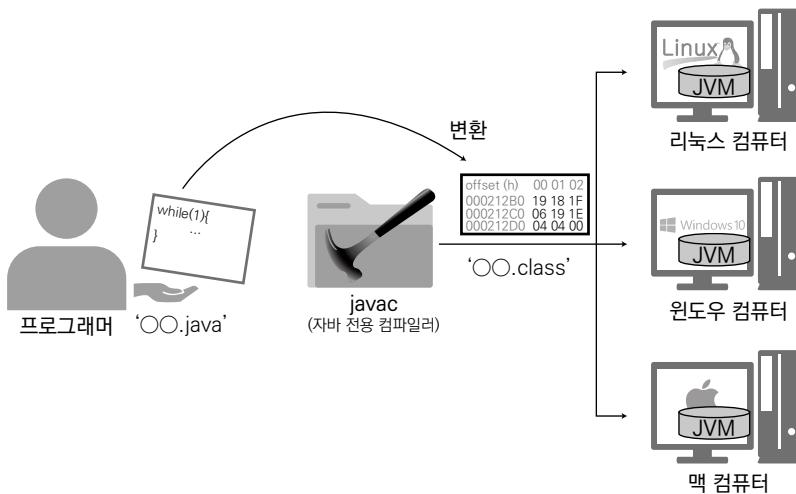
그림 1-7 자바 프로그램 컴파일과 실행



1.3.2 자바 가상 머신

JVM은 자바로 작성된 프로그램을 실행하기 위한 프로그램으로, 자바로 작성한 코드를 해당 컴퓨터의 명령어로 번역해 주는 역할을 합니다. 따라서 JVM만 있으면 기계 종류나 운영체제에 관계없이 자바 프로그램을 실행할 수 있습니다.

그림 1-8 자바가상머신(JVM)



이처럼 자바로 프로그램을 개발하기 위해서는 자바 전용 컴파일러 javac나 JVM 등이 필요합니다. 자바로 프로그램을 개발하는 데 필요한 것들을 준비하는 과정을 “자바 개발환경을 구축한다”라고 말합니다. 자바 개발환경을 구축하는 일은 시간이 조금 걸리지만 어렵지 않습니다. javac와 JVM 등 자바 프로그램 개발에 필요

한 모든 것들이 자바 개발 키트 JDK, Java Development Kit 안에 묶여 있고, 이 JDK를 인터넷에서 무료로 다운로드하여 설치만 하면 되기 때문입니다. 또한, 무료로 제공하는 이클립스 Eclipse라는 개발 툴을 활용하면 메모장보다 편리하게 자바 프로그램을 작성하고, 컴파일하고, 실행해 볼 수 있습니다.

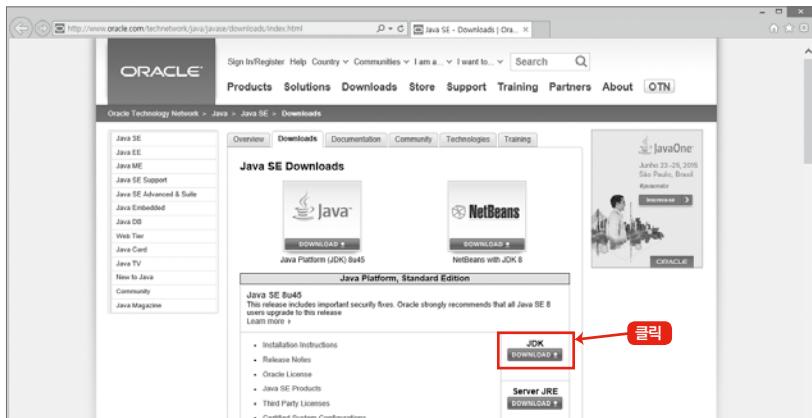
1.4 자바 개발환경 구축하기

자바 개발환경을 구축하기 위해서는 크게 JDK와 이클립스 Eclipse, 이 2가지가 필요합니다. JDK는 Java Development Kit(자바 개발 키트)의 약자로, javac나 JVM 등 자바 프로그램 개발에 필요한 모든 것을 담고 있습니다. 이클립스는 무료로 제공되는 개발 툴로, 자바 프로그램을 작성하고 컴파일하고 실행하는 것을 간편하게 도와줍니다.

1.4.1 JDK 설치

JDK를 다운로드하기 위해 <http://www.oracle.com/technetwork/java/javase/downloads/index.html>에 접속하여 [DOWNLOAD] 버튼을 클릭합니다(이 책을 집필하는 시점에서 최신 버전인 'Java SE 8u45'를 다운로드하였습니다).

그림 1-9 JDK 다운로드 과정 1



NOTE Java SE, Java EE, Java ME

Java SE, Java EE, Java ME는 각기 다른 용도를 가진 자바 플랫폼입니다. 플랫폼에 대한 정의는 많지만, 실질적으로는 다른 사람들이 구현해 놓은 함수를 쉽게 부를 수 있도록 함수명과 파라미터를 제공한 API(Application Program Interface) 모음이라고 볼 수 있습니다.

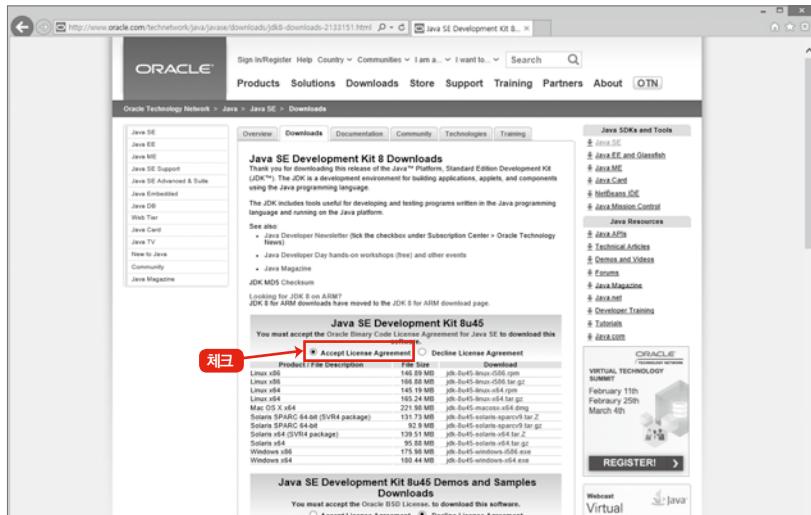
Java SE Standard Edition의 약자로, 가장 많이 사용하는 기본적인 플랫폼입니다.

Java EE Enterprise Edition의 약자로, Java SE에서 서버 소프트웨어 개발을 위한 API가 추가된 플랫폼입니다.

Java ME Micro Edition의 약자로, PDA나 휴대 전화 등 소형 기기용 소프트웨어 개발을 위한 API가 추가된 플랫폼입니다.

[DOWNLOAD] 버튼을 클릭하여 다음과 같이 화면이 바뀌면 'Accept License Agreement' 부분을 체크합니다.

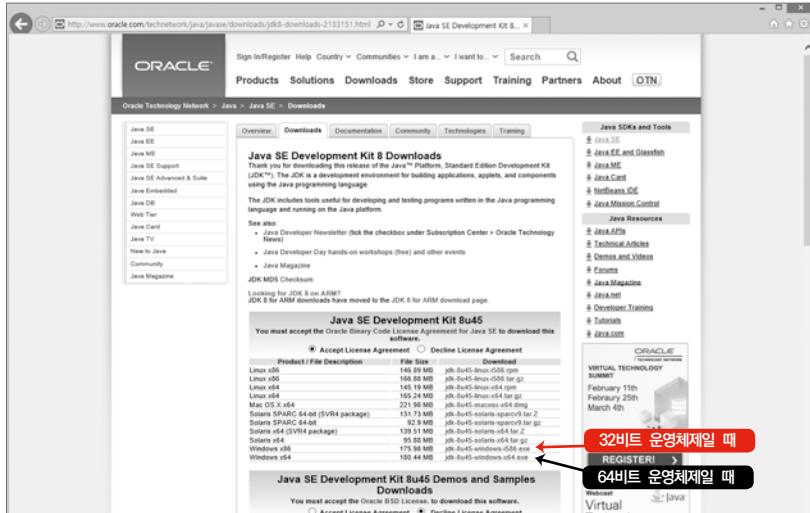
그림 1-10 JDK 다운로드 과정 2



컴퓨터 운영체제의 종류에 따라 다운로드할 수 있는 파일의 목록이 나와 있습니다. 이 책을 작성할 때 32비트 운영체제 Windows 7을 사용했으므로 Windows x86에 해당하는 'jdk-8u45-windows-i586.exe' 파일을 클릭해 다운로드 합니다. Windows를 기준으로 '제어판 → 시스템 및 보안 → 시스템'에 가면 몇

비트 운영체제인지 확인할 수 있습니다. 64비트 운영체제를 사용하고 있다면, Windows x64에 해당하는 ‘jdk-8u45-windows-x64.exe’ 파일을 다운로드 하면 됩니다.

그림 1-11 JDK 다운로드 과정 3



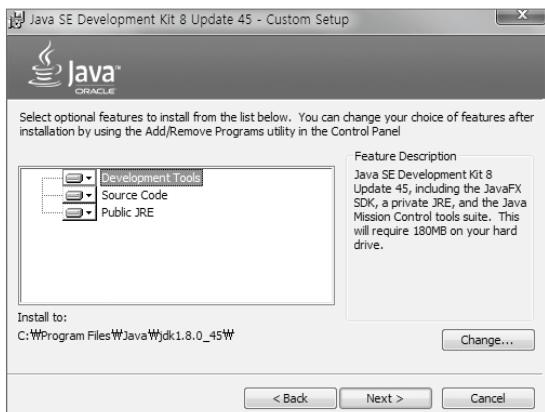
다운로드가 완료된 파일을 더블 클릭해 실행하면 다음과 같은 화면이 뜨는데, [Next] 버튼을 클릭합니다.

그림 1-12 JDK 설치 과정 1



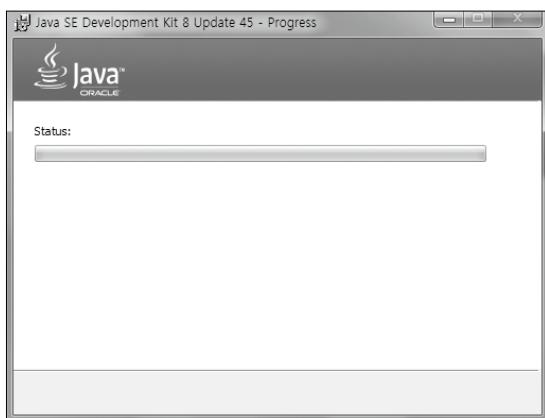
다음과 같은 화면이 뜨면 [Next] 버튼을 클릭합니다.

그림 1-13 JDK 설치 과정 2



다음과 같은 화면이 뜨면서 설치를 진행합니다.

그림 1-14 JDK 설치 과정 3



[그림 1-14] 화면 위에 [그림 1-15]와 같은 화면이 뜨면 자바를 설치할 폴더를 설정합니다. 자바를 다른 폴더에 설치하려면 [변경] 버튼을 누르고 설치할 폴더 위치를 설정합니다(자바를 설치한 폴더 위치를 기억해 두세요). 폴더 위치를 변경한 후나 변경하지 않을 경우 [다음] 버튼을 클릭합니다.

그림 1-15 JDK 설치 과정 4



[그림 1-16]과 같은 설치 진행 화면이 나오면서 설치를 진행합니다.

그림 1-16 JDK 설치 과정 5



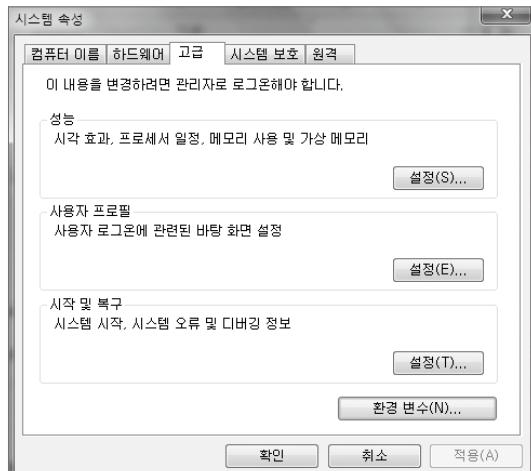
설치가 끝나고 [그림 1-17]의 화면이 나오면 [Close] 버튼을 눌러 해당 화면을 닫습니다. 이렇게 JDK 설치를 완료하면 설치한 폴더 내에 java.exe, javac.exe 같은 여러 프로그램과 기타 파일들이 생기게 됩니다. 어떤 폴더에서든지 단순한 명령어 하나로 이렇게 생성된 프로그램과 파일들에 접근하려면 컴퓨터에 이들의 위치 정보를 알려주어야 합니다. 이를 “환경 변수를 설정한다”라고 합니다.

그림 1-17 JDK 설치 과정 6



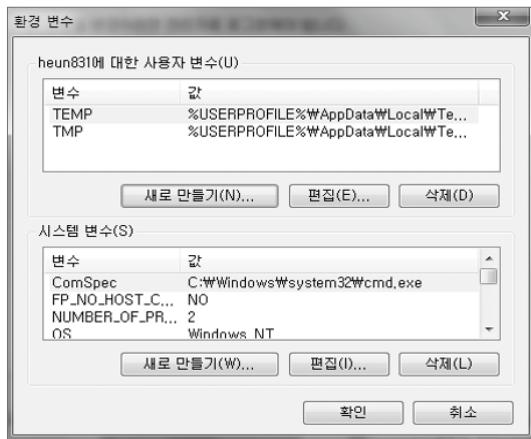
환경 변수를 설정하기 위해 [제어판 → 시스템 및 보안 → 시스템 → 고급 시스템 설정 → 고급 탭]이나 [시작 → 프로그램 및 파일 검색] 창에 ‘시스템 환경 변수 편집’을 검색한 후 실행하여 ‘시스템 속성’ 창을 열고, ‘고급’ 탭으로 이동합니다.

그림 1-18 환경 변수 설정 1



[환경 변수] 버튼을 클릭하면 [그림 1-19]와 같은 화면이 뜹니다.

그림 1-19 환경 변수 설정 2



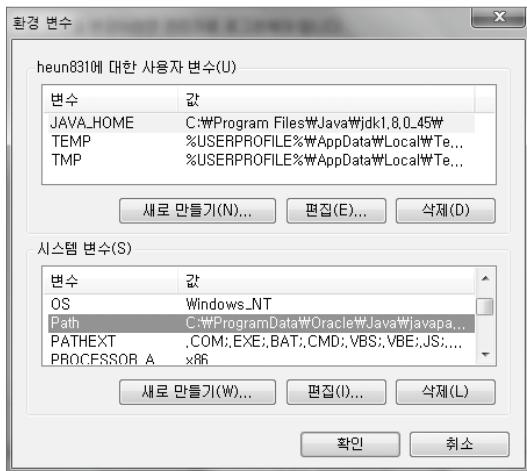
'사용자 변수' 부분에서 [새로 만들기] 버튼을 클릭하면 [그림 1-20]의 화면이 됩니다. 여기에 변수 이름은 'JAVA_HOME', 변수 값으로는 자바가 설치된 경로를 적어 줍니다. 이 책을 집필할 때 'C:\Program Files\Java\jdk1.8.0_45\'에 자바를 설치하였기에 이 주소를 적었습니다. [확인] 버튼을 클릭합니다.

그림 1-20 환경 변수 설정 3



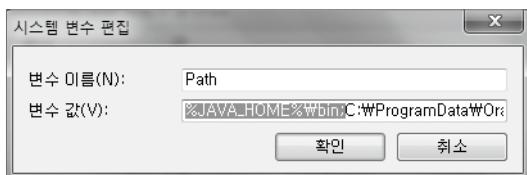
다시 '환경 변수' 창으로 돌아오면 사용자 변수 부분에 'JAVA_HOME'이 새로 추가된 것을 볼 수 있습니다. 이제 컴퓨터 내 어느 폴더에서나 JDK를 통해 설치한 각종 자바 프로그램(java.exe, javac.exe)을 명령어로 실행할 수 있게 설정할 차례입니다. 이번에는 '시스템 변수' 부분에서 'Path'를 선택하고 [편집] 버튼을 클릭합니다.

그림 1-21 환경 변수 설정 4



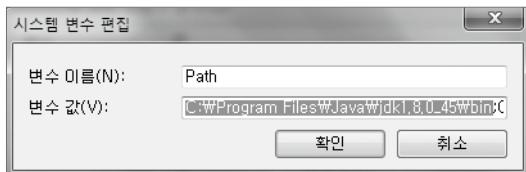
[그림 1-22]와 같이 'Path' 변수를 편집할 수 있는 '시스템 변수 편집' 창이 뜹니다. 변수 값의 가장 앞으로 마우스 커서를 이동한 후 바로 전 설정한 자비의 사용자 변수를 이용하여 '%JAVA_HOME%\bin;'이라고 적습니다. 이는 'JAVA_HOME'으로 설정된 사용자 변수의 주소 안 'bin'이라는 폴더를 가리킵니다.

그림 1-22 환경 변수 설정 5



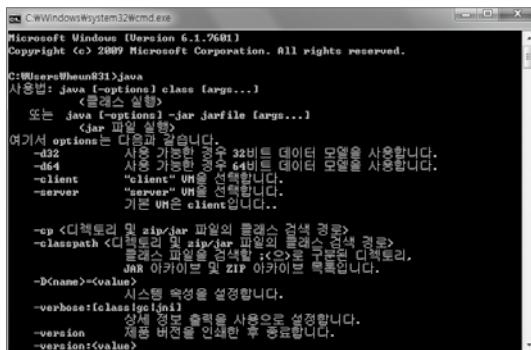
또는 [그림 1-23]처럼 주소를 바로 적을 수도 있습니다. 두 가지 방법 중 하나를 사용하여 시스템 변수를 편집 후 [확인] 버튼을 클릭합니다. '환경 변수' 창에서도 [확인] 버튼을, '시스템 속성' 창에서도 [확인] 버튼을 클릭하여 설정을 마무리합니다.

그림 1-23 시스템 변수 편집의 다른 예



이제 JDK가 잘 설치되고 환경 변수 설정도 정상적으로 하였는지를 확인하겠습니다. [시작 → 프로그램 및 파일 검색]에서 ‘cmd’를 검색해 도스 프롬프트 창을 엽니다. 도스 창에 ‘java’를 입력하고 Enter 키를 누릅니다. [그림 1-24]와 같은 화면이 뜨면 JDK 설치와 환경 변수 설정을 성공적으로 완료한 것입니다.

그림 1-24 JDK 설치와 환경 변수 설정 완료 확인



1.4.2 이클립스 설치

이클립스를 다운로드하기 위해 <http://www.eclipse.org/downloads/>에 접속합니다. JDK를 기본 버전인 JDK SE로 다운로드하였기 때문에 이클립스 역시 기본 버전인 ‘Eclipse IDE for Java Developers’를 다운로드하여도 됩니다. 하지만 기업 내 프로젝트를 수행할 때는 주로 ‘Eclipse IDE for Java EE Developers’를 사용하므로 이 버전을 다운로드하겠습니다. JDK를 다운로드할 때와 마찬가지로 자신의 운영체제 비트에 맞는 것을 선택합니다.

그림 1-25 이클립스 다운로드 과정 1

The screenshot shows the Eclipse website's download section. At the top, there are links for 'GETTING STARTED', 'MEMBERS', 'PROJECTS', and 'MORE'. A search bar at the top right includes a 'Google' button, a search icon, and a 'Log in' link. Below the main navigation, a black bar contains 'HOME / DOWNLOADS', 'Packages', and 'Developer Builds'. The main content area displays the 'Eclipse Mars 2 (4.5.2) Release for Windows' download, which is a 32-bit | 64-bit installer with 669,612 downloads. Below this, there are sections for 'Eclipse IDE for Java EE Developers' (275 MB, 52,822 downloads) and 'Eclipse IDE for Java Developers' (167 MB, 328,822 downloads). To the right, there are links for 'Eclipse Che', 'DevOps Services', and 'RELATED LINKS'.

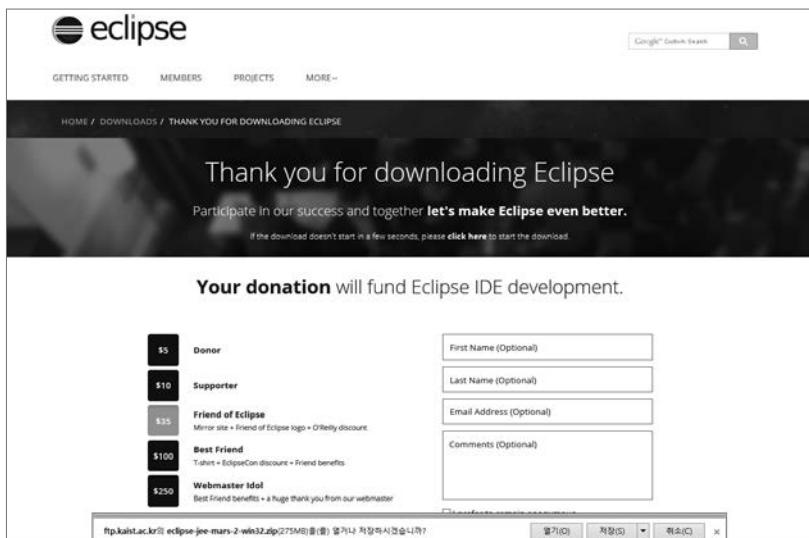
운영체제에 맞는 비트를 클릭하면 다음과 같이 다운로드할 미러 사이트들이 보이는데, [그림 1-26]에 표시된 두 곳 중 하나를 클릭합니다.

그림 1-26 이클립스 다운로드 과정 2

This screenshot shows the 'Eclipse downloads - Select a mirror' page. It starts with a note that all downloads are under the Eclipse Foundation Software User Agreement. Below this, there are two main sections: 'Get It Faster from our Members' and 'Choose a mirror close to you'. The 'Get It Faster from our Members' section lists 'IBM', 'Genuitec', 'Spring by Pivotal', and 'EclipseSource'. The 'Choose a mirror close to you' section lists 'China - Dalian Neusoft University of Information', 'Taiwan - Computer Center, Shu-Te University', 'Japan - Japan Advanced Institute of Science and Technology', 'Korea, Republic Of - KAIST' (highlighted with a red box), 'Japan - Yamagata University', 'Korea, Republic Of - Daum Kakao Corp.' (highlighted with a red box), 'Philippines - RISE', and 'Taiwan - Dept of Computer Science and Engineering, Yuan Ze University'. A red arrow points to the 'KAIST' entry, and another red arrow points to the 'Daum Kakao Corp.' entry. On the right side, there is a sidebar with 'OTHER OPTIONS FOR THIS FILE' (links to 'All mirrors (xml)' and 'Direct link to file'), 'RELATED LINKS' (links to 'Friends of Eclipse', 'Becoming a mirror site', 'Updating and installing Eclipse components', and 'Eclipse newsgroups'), and an advertisement for 'Bring Analytics to Your App Free BLU Report Server Download Now'.

링크를 클릭하면 다음과 같은 화면이 나오면서 다운로드를 시작합니다.

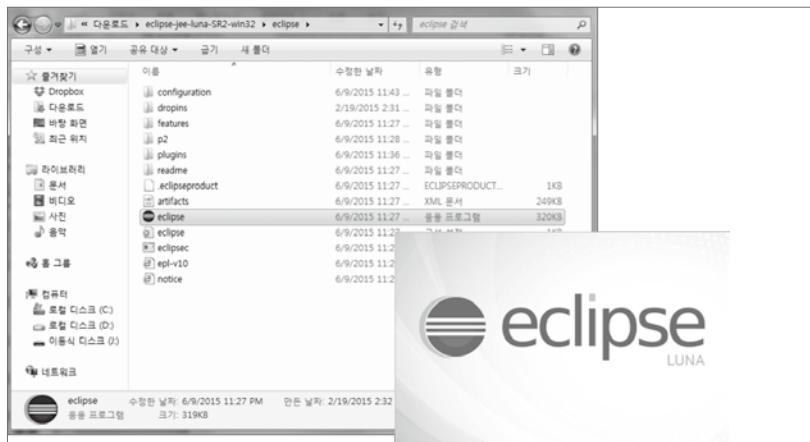
그림 1-27 이클립스 다운로드 과정 3



폴더를 지정하고 파일을 다운로드한 후 다운로드가 완료되면 파일의 압축을 풁니다. 압축을 풀고 해당 폴더에 들어가면 ‘eclipse.exe’ 프로그램이 있습니다. 이를 더블 클릭해 실행합니다.

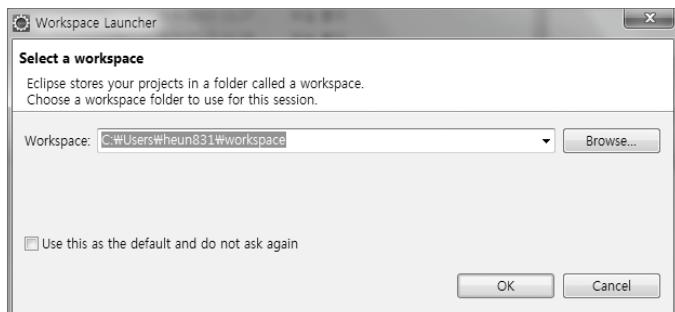
이클립스는 JDK와는 다르게 다운로드 파일의 압축을 풀면 설치하는 과정 없이 eclipse.exe 프로그램을 더블 클릭해 바로 실행할 수 있습니다. 이클립스를 사용할 때마다 이 EXE 파일을 클릭해 실행해야 하는데, 더 편리하게 사용하려면 해당 파일 클릭 후 마우스 오른 쪽 버튼을 눌러 [보내기 → 바탕화면에 바로가기 만들기]를 클릭하거나 ‘작업상태줄 고정’을 클릭하면 됩니다.

그림 1-28 이클립스 다운로드 과정 4



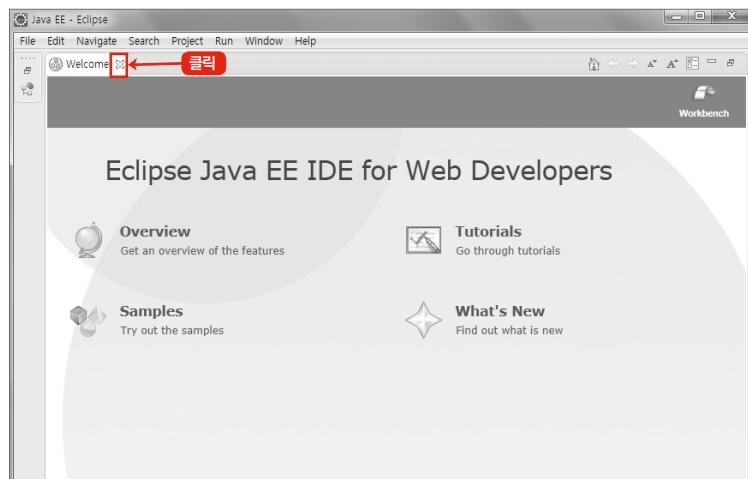
[그림 1-29]와 같은 화면이 나오면 작성한 자바 파일이 저장될 ‘Workspace’를 지정하는데, 보통 실행 시 자동으로 지정된 폴더를 쓰면 되므로 바로 [OK] 버튼을 누릅니다. workspace의 경로는 이클립스를 실행할 때마다 물어보며 필요에 따라 언제든 새 위치로 설정할 수 있습니다.

그림 1-29 이클립스 실행 1



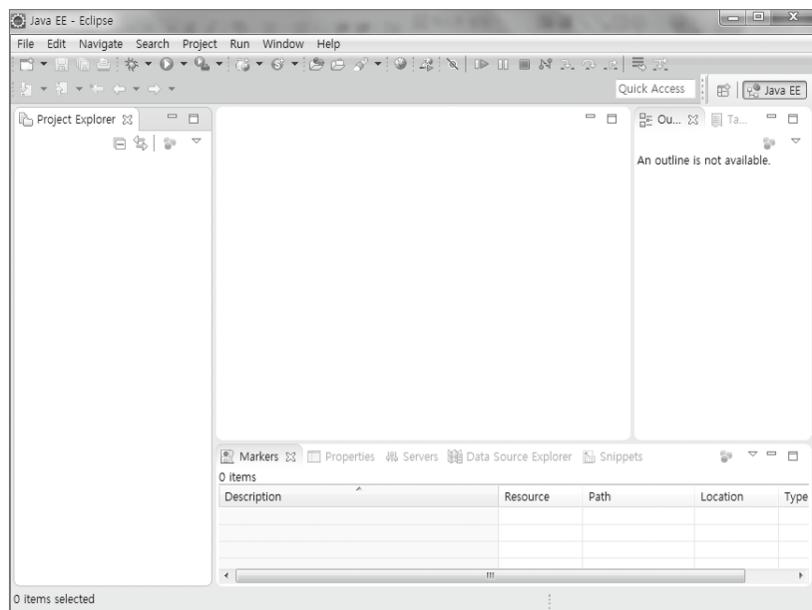
프로세스 진행 창이 잠시 나타난 후 다음과 같은 화면이 나타나면 [그림 1-30]처럼 닫기 버튼(x) 버튼을 클릭하여 ‘Welcome’ 창을 닫습니다.

그림 1-30 이클립스 실행 2



Welcome 창을 닫으면 다음과 같이 자바 프로그램을 작성할 수 있는 창을 볼 수 있습니다. 이 화면이 나오면 이클립스를 성공적으로 설치한 것입니다.

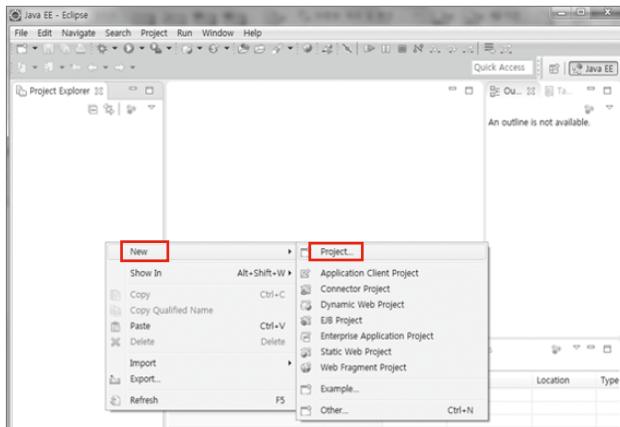
그림 1-31 이클립스 실행 3



1.4.3 Welcome to Java World!

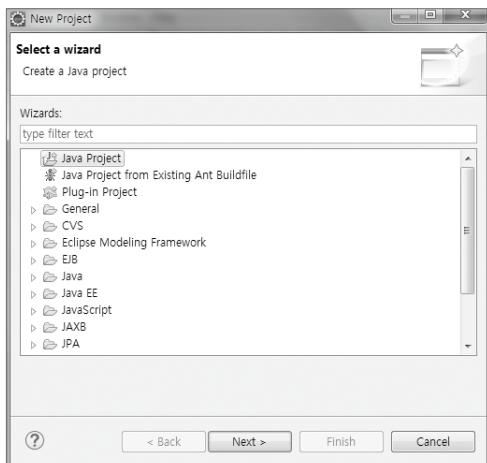
테스트로 자바 프로그램을 한번 작성해 보겠습니다. ‘Project Explorer’에서 마우스 오른쪽 버튼을 클릭하면 여러 가지 명령어의 목록이 뜹니다. 이중에서 [New → Project]를 클릭합니다.

그림 1-32 자바 프로그램 작성 1



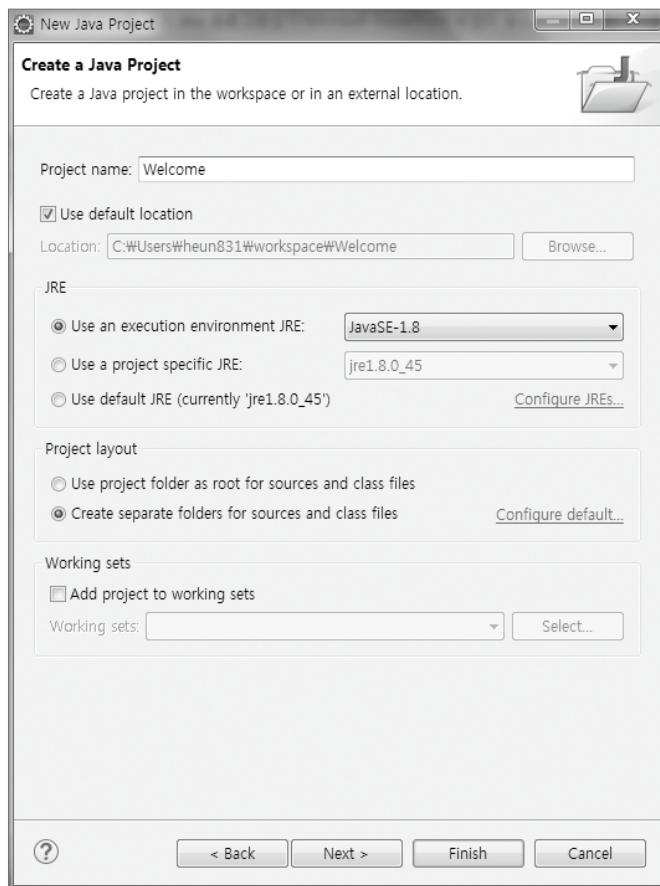
‘New Project’ 창이 뜨면 목록에서 ‘Java Project’를 선택한 후 [Next] 버튼을 클릭합니다.

그림 1-33 자바 프로그램 작성 2



[그림 1-34]와 같이 ‘New Java Project’ 창이 뜨면 ‘Project name’에 원하는 이름을 입력한 후(여기에서는 ‘Welcome’을 입력하였습니다) [Finish] 버튼을 클릭합니다. [Finish] 버튼을 클릭한 후 ‘Open Associated Perspective?’ 창이 뜨는 경우 [Yes]를 클릭하면 됩니다.

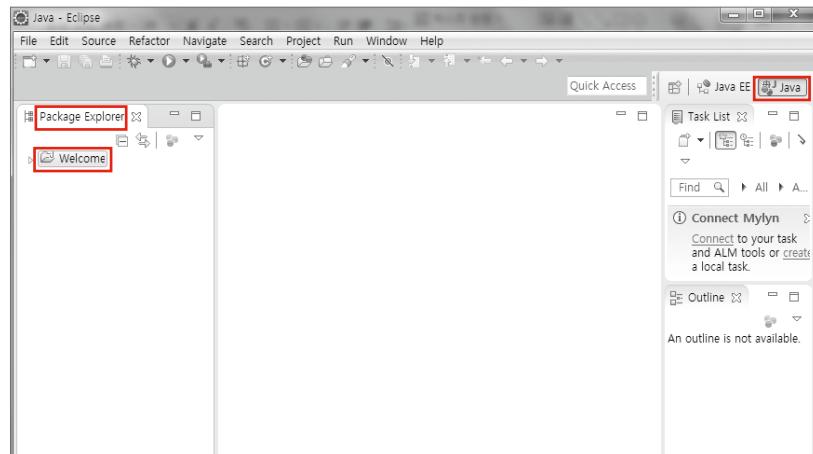
그림 1-34 자바 프로그램 작성 3



다시 원래 이클립스 화면으로 돌아가면 폴더 아이콘을 가진 새로운 프로젝트가 생겼고, 이클립스 오른쪽 상단이 ‘Java EE’에서 ‘Java’로 바뀌었음을 볼 수 있습니다. Java EE에서 Java로 바뀌면서 프로젝트와 파일들을 표시하는 뷰도 Project

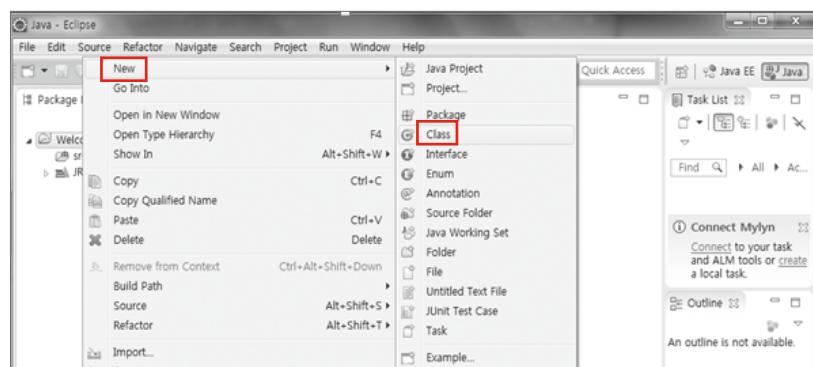
Explorer에서 'Package Explorer'로 나타나게 됩니다. Project Explorer와 Package Explorer는 매우 유사한데, Project Explorer가 웹 개발에 좀 더 최적화되어 있는 뷰이기 때문에 Java EE에서는 기본으로 Project Explorer를 사용합니다. 그 외에는 상황과 필요에 따라 Package Explorer를 사용합니다.

그림 1-35 자바 프로그램 작성 4



Project Explorer나 Package Explorer 어느 곳에서든 상관없이 새롭게 생긴 프로젝트를 클릭한 후 마우스 오른쪽 버튼을 누르면 여러 가지 명령어 목록이 뜹니다. 여기서 [New → Class]를 클릭합니다.

그림 1-36 자바 프로그램 작성 5



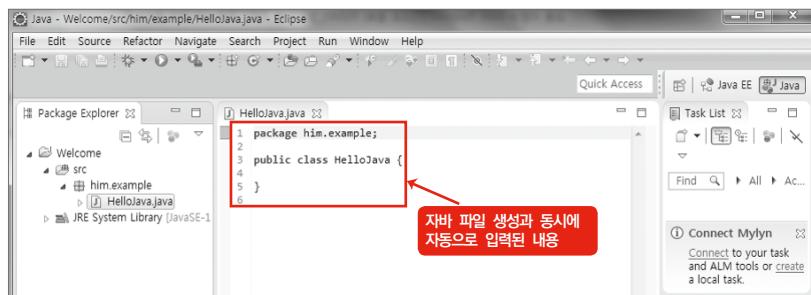
[그림 1-37]과 같이 ‘New Java Class’ 창이 뜨면 ‘Package’와 ‘Name’에 각각 알맞은 내용을 적습니다. ‘Package’에는 보통 회사 도메인이 들어가는데, 현재는 아무 것이나 들어가도 괜찮으므로 ‘him.example’이라고 입력하였습니다. ‘Name’에는 클래스 이름이 들어가야 하므로 ‘HelloJava’라고 적었습니다.

그림 1-37 자바 프로그램 작성 6



[Finish] 버튼을 누르면, [그림 1-38]과 같이 ‘HelloJava.java’ 파일이 만들어지고, 그 파일의 내용을 화면 중앙의 창에서 볼 수 있습니다. ‘HelloJava.java’ 화면에 적혀 있는 내용은 자바 파일 생성과 동시에 자동으로 입력된 것입니다.

그림 1-38 자바 프로그램 작성 7



"Welcome to Java World!"란 문장을 콘솔Console 화면에 띄우기 위해 자동으로
입력된 내용에 다음 코드를 추가합니다.

[Welcome to Java World 프로그램 코드]

```
package him.example;

public class HelloJava {

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

NOTE 기본적인 자바 클래스 파일 구조

자바 클래스 파일은 다음과 같은 기본 구조를 보입니다.

```
class 클래스명{
    public static void main(String[] args){
        처리할 명령어들
    }
}
```

이때 ;와 {} 입력에 주의합니다. 여기서 ;는 “이 명령어가 여기서 끝이다”를 의미하고, {}는 “공통의 목적을 가지고 여러 개의 명령문이 모였다”를 의미합니다. 다시 말하면, 우리가 글을 쓸 때 문장의 끝을 마침표(.)로 표현하고 들여쓰기를 통해

문단을 구분하듯이, 자바에서도 ;는 하나의 명령문의 끝을 표현하고, {}는 명령문들이 한 덩어리임을 표시해 줍니다.

NOTE public static void main(String[] args)

`public static void main(String[] args)`는 프로그램 시작 함수로 항상 이 형태로 구성되어 있습니다.

public 이 함수에 대한 접근이 어떤 패키지든 클래스든 가능하다는 것을 나타내주는 말인데, `main` 함수에 붙은 이유는 어떤 인터프리터(자바 같은 고급 언어를 한 줄씩 기계어로 변환해 주는 해석기)도 접근할 수 있게 하기 위함입니다.

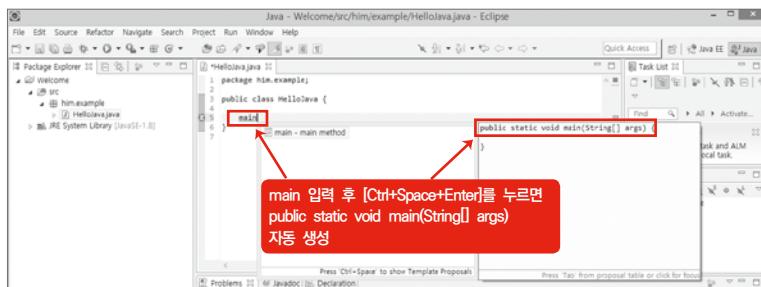
static 원래 같은 클래스에서 생성된 객체들과 값을 공유하게 하는 키워드입니다. 이 키워드가 붙은 것은 프로그램 시작 시 제일 먼저 메모리에 올라가게 됩니다. 따라서 메모리에 제일 먼저 올려 프로그램의 시작점인 이 함수를 쉽게 찾을 수 있도록 하기 위해 `static` 키워드가 `main` 함수에 붙은 것입니다.

void `main` 함수의 리턴값이 없다는 것을 의미합니다.

`String[] args` `main` 함수의 파라미터로, 프로그램 생성 시 파라미터로 받는 모두를 문자열로 인식하여 `args`라는 이름의 배열에 저장하겠다는 의미입니다. 즉, ‘120 Java 3.14’가 `main` 함수의 파라미터로 들어오면 `args[0] = “120”, args[1] = “Java”, args[2] = “3.14”`로 저장됩니다. `String[] args`로 적어도 되고, `String args[]`로 적어도 괜찮습니다.

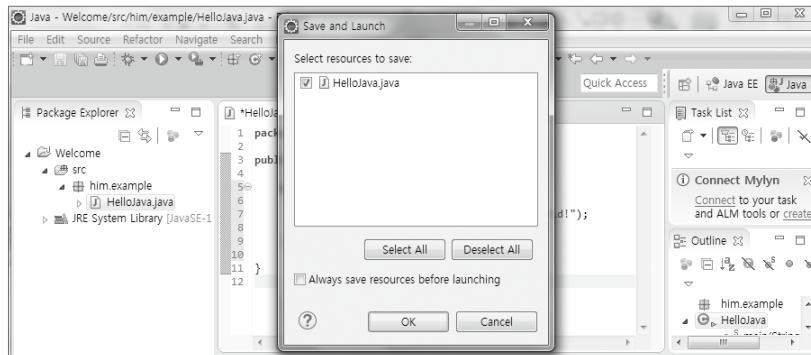
`public static void main(String[] args)`는 이클립스에서 코드를 입력할 때 `main`을 쓰고 Ctrl과 Space 키를 동시에 누른 후 엔터를 치면 자동으로 생깁니다. 이클립스에서 [Ctrl + Space]는 자동 완성키로, 다른 것도 완성할 수 있습니다. 다른 예로, `sysout`을 쓰고 [Ctrl + Space]하면 `System.out.println()`이 자동 완성됩니다.

그림 1-39 시작 함수 편하게 작성하는 법



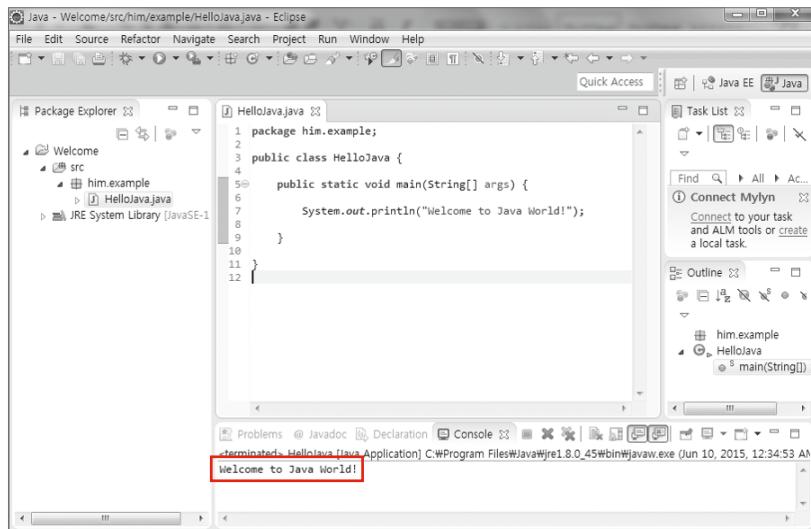
화면 상단의 실행 버튼(▷)이나 단축키 [Ctrl] + F11]을 누르면 ‘Save and Launch’ 창이 뜨고, 이 과정에서 작성한 코드는 자동으로 컴파일됩니다.

그림 1-40 Welcome to Java World 프로그램 실행



[OK] 버튼을 누르면 화면 맨 아래에 'Console' 창이 뜨면서 'Welcome to Java World' 문장이 출력된 것을 확인할 수 있습니다.

그림 1-41 Welcome to Java World 프로그램 실행 결과



앞으로도 예제 코드에 'class A {}'라는 문구가 나오면 특별한 지시가 없어도 [그림 1-35]~[그림 1-37]에서 실행한 것처럼 'A'라는 클래스 이름을 가진 클래스를 만든 후에 예제 코드를 작성하면 됩니다.

기초 문법

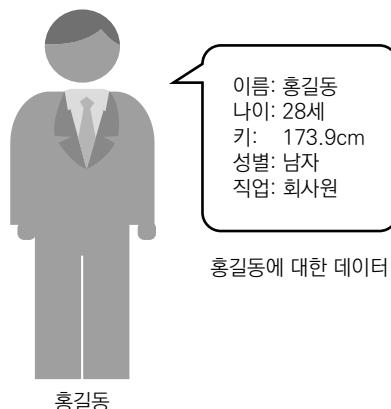
2장에서는 자바 프로그래밍 언어에서 알아야 할 기초 문법에 대해 다룹니다. 영어를 배울 때 명사, 동사, 형용사 등 품사와 1형식, 2형식 등 형식과 같은 기초 문법을 공부하듯이, 자바 프로그래밍 언어를 배울 때도 여러 가지 기초 문법을 공부할 필요가 있습니다. 언어는 보고 느끼는 것을 표현하기 위한 수단이고, 이를 위한 다양한 표현법이 있습니다. 프로그래밍 언어인 자바 역시 데이터를 표현하는 방법, 반복되는 것을 표현하는 방법 등 다양한 표현법이 있습니다. ‘2.1 기초 문법’에서는 데이터를 표현하고 저장하는 방법을 다루고, ‘2.2 연산자’에서는 수학의 사칙연산과 같이 데이터를 가지고 연산할 수 있는 연산자에 대해 배웁니다. ‘2.3 여러 가지 배열’에서는 여러 데이터를 좀 더 편리하게 저장하는 방법을 공부하겠습니다. ‘2.4 반복문’에서는 반복되는 처리 동작을 표현하는 방법을, ‘2.5 조건문’에서는 어떤 경우에만 특정 행동을 하게 조건을 거는 방법에 대해 공부하겠습니다.

2.1 기초 문법

2.1.1 데이터 타입

컴퓨터는 데이터를 가지고 연산합니다. 그럼 데이터란 무엇일까요? 데이터는 어떤 대상을 수치나 말로 묘사하여 표현한 것을 말합니다. 예를 들면, 어떤 사람에 대해서 표현할 때 ‘이름: 홍길동, 나이: 28세, 키: 173.9cm, 성별: 남자, 직업: 회사원’이라고 한다면, 이는 그 사람에 대한 데이터가 됩니다.

그림 2-1 데이터의 개념

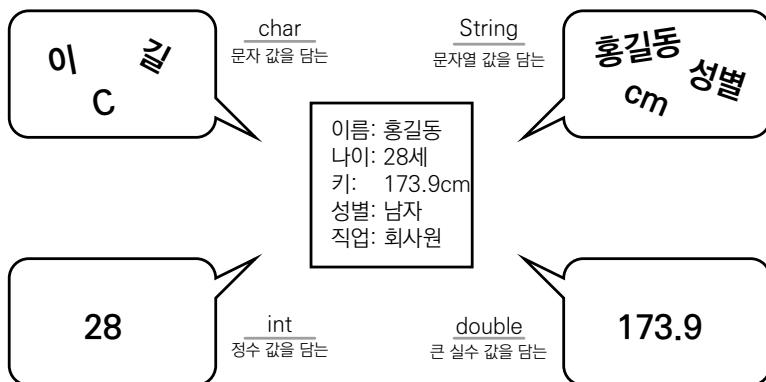


1, 2, 3 등의 숫자를 ‘정수’라고 하고, 3.4, 4.5 등의 숫자를 ‘실수’라고 하며, 아, a를 ‘문자’, 학교를 ‘문자열’이라고 그 종류를 분류하듯이, 컴퓨터 프로그래밍 언어에서도 여러 데이터 값의 종류를 지칭하는 ‘데이터 타입’이 있습니다. 자바에는 대표적으로 ‘char, String, int, long, float, double, boolean’ 등의 데이터 타입이 있습니다.

표 2-1 데이터 타입의 종류와 예시

데이터 타입	표현할 수 있는 값	예시
char	문자	'이', '길', '안', 'a', 'b'
String	문자열	“학교”, “Hi! I'm Hin!”
byte	매우 작은 정수(-128 ~ +127)	0, 30, 255
short	작은 정수(약 -3만 ~ +3만)	-3, 2, 100, 30000
int	정수(약 -21억~ +21억)	-1, -234, 3, 40, 100, 2000000000
long	큰 정수(약 -900경 ~ +900경)	-5, -1000, 1, 30, 9000000000000000
float	실수	-1.2, 222.31, 3.14
double	큰 실수	-111.85, 6.3, 5.783
boolean	true, false	true, false

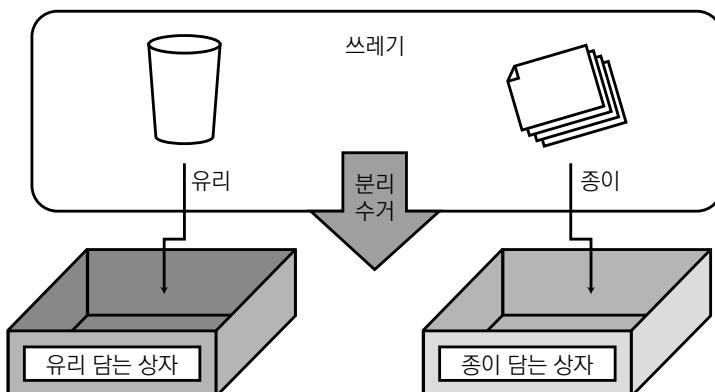
그림 2-2 데이터 타입의 종류

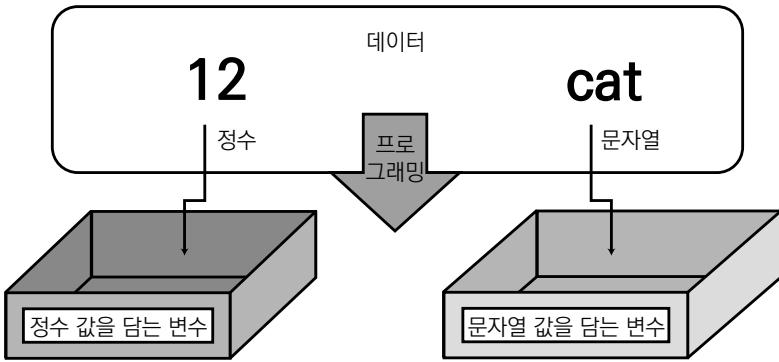


2.1.2 변수

변수란 특정 데이터 타입의 값을 담는 상자입니다. 쓰레기를 버릴 때 유리는 유리를 담는 상자에, 종이는 종이를 담는 상자에 분리해 넣는 것처럼 어떤 데이터를 가지고 프로그래밍할 때도 정수 같은 정수 값을 담는 변수에, 문자열 같은 문자열 값을 담는 변수에 넣어 주어야 합니다.

그림 2-3 변수의 개념





메모리 공간에 해당 변수를 저장할 공간을 만드는 것을 ‘변수를 선언한다’라고 합니다. 변수를 선언할 때는 (1) 해당 변수가 어떤 타입의 데이터를 담을 것인지 명시하고 (2) 변수의 이름을 적은 후 (3) ‘;’으로 변수 선언이 끝났다는 걸 표시합니다. 프로그래밍 언어에서 세미콜론(:)은 문장의 끝에 찍는 문장의 마침표(.)와 같다고 생각하면 됩니다.

[변수의 선언]

- | | |
|-------------------------------|-----------|
| (1) 변수가 어떤 타입의 데이터를 담을 것인지 명시 | -> int |
| (2) 변수의 이름을 적음 | -> int a |
| (3) 생분점(:)을 찍음 | -> int a; |

int a;

정수 값을 담는 a라는 이름의 변수를 선언한다

NOTE | 변수 선언 법칙

1. 변수명은 ‘영문, 숫자, _, \$’만 사용할 수 있다(ex. int a, int a2, int _a2, int \$a2).
2. 변수명에 공백문자와 특수문자를 사용할 수 없다(ex. int a+b는 안 됨).
3. 변수명의 첫 글자는 숫자가 오면 안된다(ex. int 22java는 안 됨).
4. 변수명은 영문의 대소문자를 구분한다. (ex. int ab와 int aB는 다른 변수).
5. 자바 예약어는 사용할 수 없다 (ex. int byte나 int class는 안 됨).

NOTE 자바 예약어

'자바 예약어'란 자바가 어떤 의미로 사용하겠다고 예약한 단어들을 말합니다. 그 종류로는 abstract, boolean, byte, break, catch, char, class, continue, case, do, double, default, enum, extends, else, final, finally, float, for, false, implements, import, instanceof, int, interface, if, long, new, null, native, private, protected, public, package, return, static, strictfp, super, switch, synchronized, short, throw, throws, transient, this, true, try, void, volatile, while 등이 있습니다. 각 예약어의 의미는 앞으로 공부하면서 자연스레 배울 수 있으니, 지금은 신경 쓰지 않아도 괜찮습니다.

선언한 변수에 등호(=)를 사용해 값을 저장할 수 있습니다.

[변수에 값 저장]

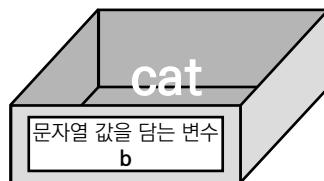
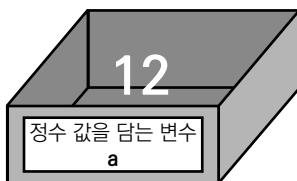
a = 1;

변수 a에 1을 저장한다

변수를 선언함과 동시에 값을 저장할 수도 있는데, 이를 '변수를 초기화한다'라고 합니다. 즉, 정수 값을 담는 변수 a를 선언하고 동시에 1을 저장하여 초기화하는 것을 int a = 1;처럼 쓸 수 있습니다. 이와 같은 방법을 사용하여 [그림 2-3]의 아랫부분을 프로그래밍 언어로 표현하면 [그림 2-4]와 같습니다.

그림 2-4 변수의 선언과 값 저장 응용

변수: 특정 타입의 데이터를 담는 상자



int a = 12;

정수 값을 담는 **a**라는 이름의 변수를 선언하고 12를 저장한다

String b = "cat";

문자열 값을 담는 **b**라는 이름의 변수를 선언하고 cat을 저장한다

이번 장에서 공부한 변수를 바탕으로 [코드 2-1]을 분석해 보면 문자열 값을 담는 String 변수 4개에 값을 대입하고 이 값을 출력하고 있습니다. 'Variables'라는 클래스 이름을 가진 새로운 클래스를 생성하여 [코드 2-1]을 작성해 봅시다.

[코드 2-1] 변수의 선언, 값 저장과 출력 1

class Variables {

public static void main(String[] args) {

String str1 = "Hello";

문자열 값을 담는 **str1**이라는 이름의 변수를 선언하고 Hello를 저장한다

String str2 = "Java";

문자열 값을 담는 **str2**라는 이름의 변수를 선언하고 Java를 저장한다

String str3 = "Nice to meet you";

문자열 값을 담는 **str3**이라는 이름의 변수를 선언하고 Nice to meet you를 저장한다

String str4 = "Bye!";

문자열 값을 담는 **str4**라는 이름의 변수를 선언하고 Bye!를 저장한다

System.out.println(str1);

str1 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다

System.out.println(str2);

str2 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다

System.out.println(str3);

str3 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다

System.out.println(str4);

str4 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다

}

}

```
Hello
Java
Nice to meet you
Bye
```

여기서 절낀! | char와 String의 차이

`char`과 `String`은 둘 다 문자를 담는 데이터 타입이지만, `char`는 오직 한 글자만 담을 수 있고 `String`은 여러 글자를 한 번에 담을 수 있습니다. 또한, `char`는 문자를 담을 때 해당 문자를 ''로 표시해야 하고(ex. `char c = 'k';`), `String`은 문자열을 표현하는 해당 문장을 ""로 표시해야 합니다(ex. `String str = "Hello!";`).

2.2 여러 가지 연산자

컴퓨터는 전기를 연결한 기계입니다. 전기가 낮을 때를 0, 높을 때를 1로 생각하여 모든 데이터나 명령어를 0과 1로 구성된 2진수로 나타냅니다. 따라서 컴퓨터 프로그램의 실행과 종료가 2진수의 수학 연산을 바탕으로 이루어집니다.

연산자는 영어로 ‘operator’라고 하며, 실행해야 하는 행동을 지정하는 것을 말합니다. 수학의 사칙 연산인 +, -, ×, ÷ 모두 연산자입니다. 프로그래밍 언어에서는 수학에서 사용하는 연산자 외에도 여러 가지 연산자가 있는데, 차근차근 살펴보겠습니다.

2.2.1 산술 연산자

산술이란 2개 이상의 수를 결합하는 모든 법칙을 아우르는 말로서, 쉽게는 ‘계산’이라고 볼 수 있습니다. 따라서 산술 연산자란 계산하는 연산자입니다. 그 종류는 다음과 같습니다.

표 2-2 산술 연산자의 종류

연산자	기능	사용법	해석
=	저장한다	$x = y$	변수 x에 변수 y의 값을 저장한다.
+	더한다 (+)	$x = y + z$	변수 x에 변수 y와 z를 더한 값을 저장한다.
-	뺀다 (-)	$x = y - z$	변수 x에 변수 y에서 z를 뺀 값을 저장한다.
*	곱한다 (x)	$x = y * z$	변수 x에 변수 y와 z를 곱한 값을 저장한다.
/	나눈다 (\div)	$x = y / z$	변수 x에 변수 y를 z로 나눈 값을 저장한다.
%	나눈 나머지를 구한다	$x = y \% z$	변수 x에 변수 y를 z로 나눠서 얻은 값을 저장한다.

산술 연산자를 사용한 예는 다음과 같습니다.

[코드 2-2] 산술 연산자를 이용한 예1

```
class Operator1 {  
    public static void main(String[] args){  
        System.out.println("8+3= "+(8+3));  
        "8+3= 11"을 시스템 콘솔에 출력하고 한 줄 띄운다  
  
        System.out.println("9-2= "+(9-2));  
        "9-2= 7"을 시스템 콘솔에 출력하고 한 줄 띄운다  
  
        System.out.println("10×5= "+(10*5));  
        "10×5= 50"을 시스템 콘솔에 출력하고 한 줄 띄운다  
  
        System.out.println("8÷2= "+(8/2));  
        "8÷2= 4"를 시스템 콘솔에 출력하고 한 줄 띄운다  
  
        System.out.println("9÷2의 나머지= "+(9%2));  
        "9÷2의 나머지= 1"을 시스템 콘솔에 출력하고 한 줄 띄운다  
  
    }  
}
```

실행 결과

8+3=11
9-2=7
10×5=50
8÷2=4

‘=’ 연산자와 각 연산자들은 함께 쓸 수도 있습니다.

표 2-3 ‘=’ 연산자와 합쳐진 연산자들

연산자	기능	사용법	해석
$+=$	더한다	$x += y$	변수 x 와 변수 y 를 더한 값을 x 에 저장한다. ($x = x + y$)
$-=$	뺀다	$x -= y$	변수 x 와 변수 y 를 뺀 값을 x 에 저장한다. ($x = x - y$)
$*=$	곱한다	$x *= y$	변수 x 와 변수 y 를 곱한 값을 x 에 저장한다. ($x = x * y$)
$/=$	나눈다	$x /= y$	변수 x 를 변수 y 로 나눈 값을 x 에 저장한다. ($x = x / y$)
$%=$	나눈 나머지를 구한다	$x \%= y$	변수 x 를 변수 y 로 나눈 나머지를 x 에 저장한다. ($x = x \% y$)

[코드 2-3] 산술 연산자를 이용한 예2

```
class Operator2 {
    public static void main(String[] args){
        int a = 10;
        정수 값을 담는 a라는 이름의 변수를 선언하고 10을 저장한다

        a += 2;
        변수 a가 담고 있는 값에 2를 더하고 그 값을 변수 a에 저장한다

        System.out.println(a);
        ()안의 내용을 시스템 콘솔에 출력하고 한 줄 띄운다 변수 a의 값

        a %= 5;
        변수 a가 담고 있는 값을 5로 나눈 나머지를 변수 a에 저장한다

        System.out.println(a);
        변수 a의 값을 시스템 콘솔에 출력하고 한 줄 띄운다

    }
}
```

12

2

+와 -에는 더 특별한 연산자가 존재합니다. 바로 값을 1씩 증가시킬 때 사용하는
++와 값을 1씩 감소시킬 때 사용하는 --입니다.

표 2-4 1씩 증감하는 특별한 연산자

연산자	기능	사용법	해석
++	1을 더한다	x++ 또는 ++x	변수 x에 1을 더한다($x = x + 1$)
--	1을 뺀다	x-- 또는 --x	변수 x에서 1을 뺀다($x = x - 1$)

[코드 2-4] 산술 연산자를 이용한 예3

```
class Operator3 {
    public static void main(String[] args){
        int a = 10, b = 10;
        정수 값을 담는 a라는 이름의 변수를 선언하고 10을 저장하며,
        정수 값을 담는 b라는 이름의 변수를 선언하고 10을 저장한다

        System.out.println(++a);
        변수 a의 값에 1을 더한 값을 시스템 콘솔에 출력하고 한 줄 띄운다

        System.out.println(b++);
        변수 b의 값에 1을 더한 값을 시스템 콘솔에 출력하고 한 줄 띄운다

        System.out.println(--a);
        변수 a의 값에서 1을 뺀 값을 시스템 콘솔에 출력하고 한 줄 띄운다

        System.out.println(b--);
        변수 b의 값에서 1을 뺀 값을 시스템 콘솔에 출력하고 한 줄 띄운다
    }
}
```

11
10
10
11

NOTE a++와 ++a의 차이점

`++`는 변수 `a`의 값에 1씩 더하라는 의미의 연산자입니다. 그런데 그 위치가 변수 뒤에 붙느냐 앞에 붙느냐에 따라 약간의 차이가 있습니다. `a++`와 같이 변수 뒤에 연산자가 붙는 경우 해당 변수의 값을 읽고 난 후 1의 값을 더합니다. 반면, `++a`와 같이 변수 앞에 연산자가 붙는 경우 해당 변수의 값에 1을 더하고 난 후 값을 읽습니다. 이 법칙은 --연산자에도 똑같이 적용됩니다.

2.2.2 비교 연산자

수학에서 2개의 숫자 값을 비교한다는 의미는 어느 숫자가 큰지, 작은지, 같은지, 같지 않은지 등을 판단한다는 의미입니다. 비교하는 기호에는 `>`, `<`, `\geq` , `\leq` , `=`, `\neq` 등의 연산자가 있습니다. 자바 프로그래밍 언어에서도 두 변수의 값을 비교하기 위해 다음과 같은 비교 연산자가 있습니다.

표 2-5 비교 연산자의 종류

연산자	기능	사용법	해석
<code>></code>	크다(<code>></code>)	<code>x > y</code>	변수 <code>x</code> 의 값이 변수 <code>y</code> 의 값보다 크다.
<code><</code>	작다(<code><</code>)	<code>x < y</code>	변수 <code>x</code> 의 값이 변수 <code>y</code> 의 값보다 작다.
<code>\geq</code>	크거나 같다(<code>\geq</code>)	<code>x \geq y</code>	변수 <code>x</code> 의 값이 변수 <code>y</code> 의 값보다 크거나 같다.
<code>\leq</code>	작거나 같다(<code>\leq</code>)	<code>x \leq y</code>	변수 <code>x</code> 의 값이 변수 <code>y</code> 의 값보다 작거나 같다.
<code>$=$</code>	같다(<code>$=$</code>)	<code>x == y</code>	변수 <code>x</code> 의 값이 변수 <code>y</code> 의 값과 같다.
<code>\neq</code>	같지 않다(<code>\neq</code>)	<code>x != y</code>	변수 <code>x</code> 의 값이 변수 <code>y</code> 의 값과 같지 않다.

자바에서는 비교 연산자로 조건식을 만들면, 그 결과 값으로 참(true), 거짓(false) 값이 나옵니다. 그래서 `while`문이나 `if`문 등에서 () 안에 비교 연산자를 사용한 조건식을 넣고, 그 값이 참일 동안에만 {} 안의 내용을 실행합니다. 예를 들면,

'while (a<5){ 처리할 명령 }'라고 쓰는 경우, a의 값이 5보다 작을 동안에만 {} 안의 내용을 실행합니다.

[코드 2-5] 비교 연산자를 이용한 예

```
class Operator4 {  
    public static void main(String[] args){  
  
        int a = 10, b = 15;  
        정수 값을 담는 a라는 이름의 변수를 선언하고 10을 저장하며,  
        정수 값을 담는 b라는 이름의 변수를 선언하고 15를 저장한다  
  
        System.out.println("a>b : " + (a>b));  
        "a>b :" 다음에 a가 b보다 크면 true, 아니면 false를 시스템 콘솔에 출력하고 한 줄 띠운다  
  
        System.out.println("a<b : " + (a<b));  
        "a<b :" 다음에 a가 b보다 작으면 true, 아니면 false를 시스템 콘솔에 출력하고 한 줄 띠운다  
  
        System.out.println("a==b : " + (a==b));  
        "a==b :" 다음에 a와 b가 같으면 true, 아니면 false를 시스템 콘솔에 출력하고 한 줄 띠운다  
  
        System.out.println("a!=b : " + (a!=b));  
        "a!=b :" 다음에 a와 b가 같지 않으면 true, 아니면 false를 시스템 콘솔에 출력하고 한 줄 띠운다  
  
    }  
}
```

실행 결과

```
a>b : false  
a<b : true  
a==b : false  
a!=b : true
```

2.2.3 논리 연산자

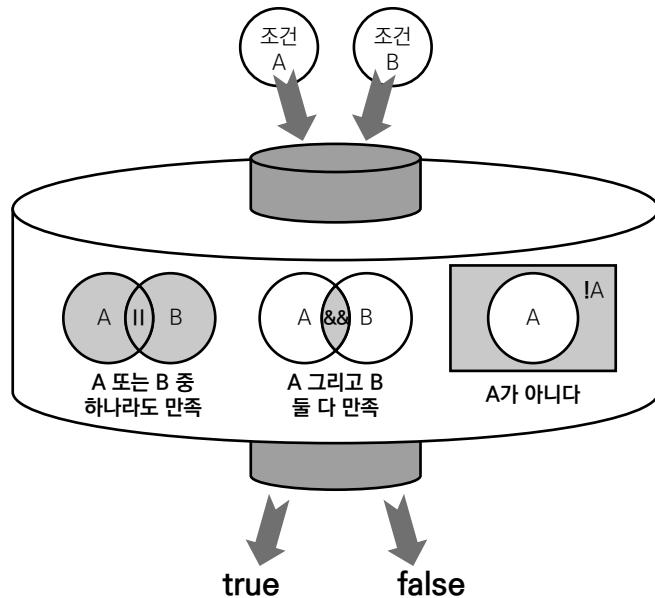
우리가 어떤 조건을 이야기하고 싶은데 그 조건이 한 개 이상일 때가 간혹 있습니다. 예를 들면, “내일 비가 오거나 눈이 오면 우린 밖에 나갈 수 없을 거야.”라든지 “해당 시험의 응시 자격 나이는 20세 이상 30세 이하입니다.”가 있습니다. 프로

그래밍 언어에서도 한 개 이상의 조건이 모여 새로운 조건을 만들어 낼 때 사용하는 것으로, ‘또는(or)’, ‘그리고(and)’, ‘~이 아님(not)’이란 뜻을 가진 총 3가지 논리 연산자가 있습니다.

표 2-6 논리 연산자의 종류

연산자	기능	사용법	해석
<code> </code>	또는	<code>(a==2) (a==10)</code>	변수 a의 값이 2 또는 10
<code>&&</code>	그리고	<code>(a>2) && (a<10)</code>	변수 a의 값이 2 초과 그리고 10 미만
<code>!</code>	~이 아닙니다	<code>!(a==2)</code>	변수 a의 값이 2가 아닙니다

그림 2-5 논리 연산자



논리 연산자를 활용한 예는 다음과 같습니다.

[코드 2-6] 논리 연산자를 이용한 예

```
class Operator5 {  
    public static void main(String[] args){
```

```
int a = 10, b = 10;
```

정수 값을 담는 **a**라는 이름의 변수를 선언하고 10을 저장하며,

정수 값을 담는 **b**라는 이름의 변수를 선언하고 10을 저장한다

```
System.out.println((a==10)&&(b==5));
```

변수 **a**의 값이 10이고 변수 **b**의 값도 5면 true, 아니면 false를 시스템 콘솔에 출력하고 한 줄 띄운다

```
System.out.println((a==10) || (b==5));
```

변수 **a**의 값이 10이거나 변수 **b**의 값이 5면 true, 아니면 false를 시스템 콘솔에 출력하고 한 줄 띄운다

```
System.out.println(!(a==10));
```

변수 **a**의 값은 10이 아니면 true, 10이면 false를 시스템 콘솔에 출력하고 한 줄 띄운다

```
}
```



```
}
```

[실행 결과]

```
false  
true  
false
```

2.2.4 데이터 타입의 변환

자바는 변수에 데이터를 담습니다. 앞에서 설명한 바와 같이 변수는 특정 데이터 타입의 값을 담는 상자를 말합니다. 그래서 정수 값을 담는 int라는 데이터 타입의 변수에는 2, 3, 4와 같은 정수 값을 대입하고, 실수 값을 담는 float라는 데이터 타입의 변수에는 1.5, 2.3과 같은 실수 값을 대입합니다. 그렇다면 산술 연산자 +, -, *, /를 사용하여 정수 값끼리 계산했을 때는 어떤 값이 나올까요?

[수학 연산 결과]

```
7 / 2 = 3.5
```

[자바 연산 결과]

```
7 / 2 = 3
```

우리가 배운 수학대로라면 7 나누기 2를 하면 3.5가 나옵니다. 그런데 자바로 프로그래밍하여 7 나누기 2를 하면 3이 나오게 됩니다. 자바에는 변수라는 개념이 있어서 같은 데이터 타입의 변수로 산술 연산을 하면 그 데이터 타입의 값이 나옵니다. 즉, int 변수끼리 연산하면 int 값이, float 변수끼리 연산하면 float 값이 나옵니다. 하지만 서로 다른 데이터 타입의 변수로 산술 연산을 하면 두 데이터 타입 중 나타낼 수 있는 데이터의 범위가 큰 타입으로 값이 나옵니다. 즉, int 변수와 float 변수끼리 연산하면 float 값이 나옵니다.

[코드 2-7] 데이터 타입별 연산

```
class DataType {  
    public static void main(String[] args){  
        System.out.println("7/2=" + 7/2);  
        "7/2=3"를 시스템 콘솔에 출력하고 한 줄 띄운다  
  
        System.out.println("7/2.0=" + 7/2.0);  
        "7/2=3.5"를 시스템 콘솔에 출력하고 한 줄 띄운다  
  
        System.out.println("7.0/2=" + 7.0/2);  
        "7/2=3.5"를 시스템 콘솔에 출력하고 한 줄 띄운다  
  
        System.out.println("7.0/2.0=" + 7.0/2.0);  
        "7/2=3.5"를 시스템 콘솔에 출력하고 한 줄 띄운다  
  
    }  
}
```

실행 결과

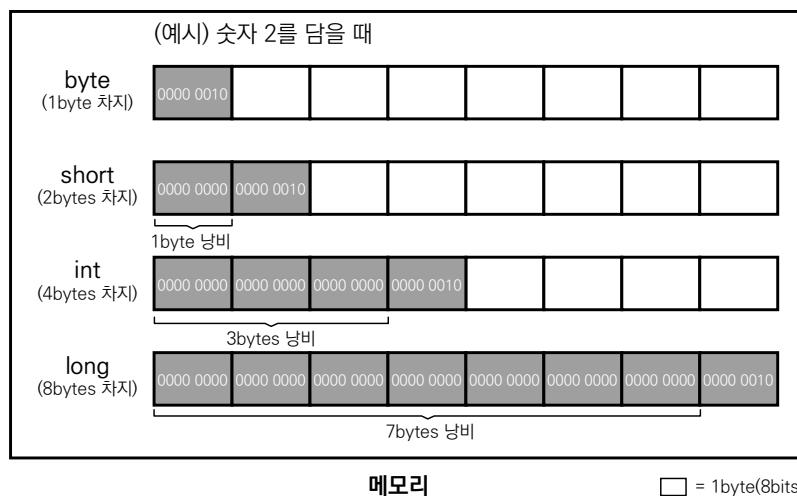
```
7/2=3  
7/2.0=3.5  
7.0/2=3.5  
7.0/2.0=3.5
```

[코드 2-7]에서 볼 수 있듯이 연산 시 정수와 실수를 섞어서 계산하면, 정수는 실수로 변환되어 계산합니다. 표현할 수 있는 수의 범위에 따라 데이터 타입의 종류

도 여러 가지가 있습니다. [표 2-1]에서 확인한 바와 같이 정수를 표현하는 데이터 타입에는 byte, short, int, long 등이 있고, 실수를 표현하는 데이터 타입에는 float과 double이 있습니다.

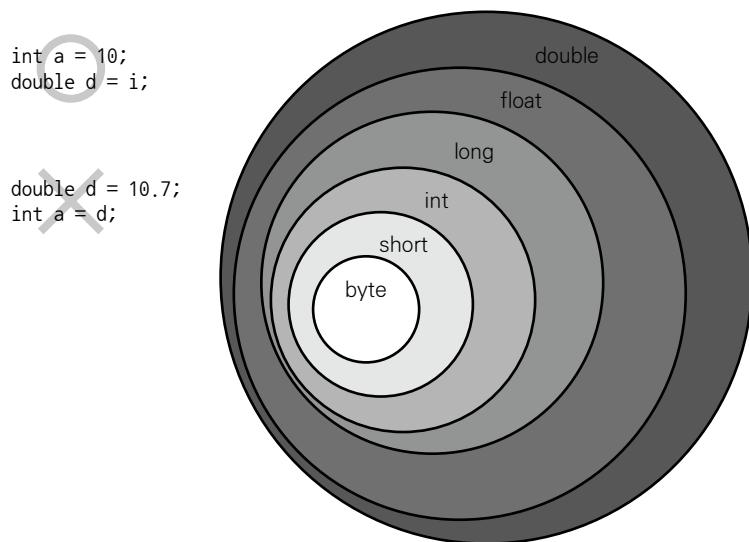
똑같이 정수를 표현하고, 실수를 표현하는 데도 데이터 타입이 여러 개인 이유는 바로 메모리 때문입니다. 자바에서 변수를 선언하면 메모리 공간에 해당 변수의 데이터 타입만큼 자리가 잡힙니다. 예를 들어, 프로그래머가 1~100000 사이의 정수만 사용하여 연산할 것이라면 90000000000 이상의 정수를 담을 수 있는 long형 변수보다는 int형 변수를 사용하는 것이 더 경제적입니다. 따라서 메모리 공간을 더 효율적으로 이용하기 위해, 사용할 숫자의 크기에 따라 여러 종류의 데이터 타입을 정의하여 사용하는 것입니다.

그림 2-6 정수 데이터 타입별 차지하는 메모리 공간



차지하는 메모리 공간의 크기가 작은 데이터 타입에서 큰 데이터 타입으로는 데이터 타입이 다른 변수끼리여도 대입할 수 있지만, 그 반대는 불가능합니다. XS 사이즈의 옷을 입는 여성이 L 사이즈의 옷을 입을 수는 있지만, L 사이즈의 옷을 입는 남성이 XS 사이즈의 옷을 입으면 몸에 맞지 않는 것과 같습니다.

그림 2-7 담을 수 있는 값의 크기 비교



()를 사용하여 '(데이터 타입)'이라고 값이나 변수 앞에 쓰면, 해당 데이터 타입으로 강제 변환할 수 있습니다. 이를 '캐스트^{cast}'라고 하며, 이때 ()를 '캐스트 연산자'라고 부릅니다.

[코드 2-8] 데이터 타입 변환 연산

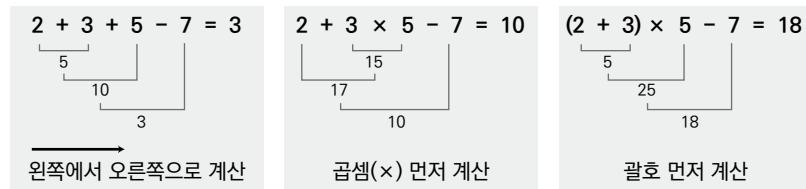
```
class DataTypeCast {  
    public static void main(String[] args){  
        System.out.println((float)7/2);  
        float 형으로 변환된 7을 2로 나눈 값을 시스템 콘솔에 출력하고 한 줄 띠운다  
  
        System.out.println(7/(float)2);  
        7을 float 형으로 변환된 2로 나눈 값을 시스템 콘솔에 출력하고 한 줄 띠운다  
  
        System.out.println((float)(7/2));  
        7을 2로 나눈 값을 float 형으로 변환해 시스템 콘솔에 출력하고 한 줄 띠운다  
    }  
}
```

3.5
3.5
3.0

2.2.5 연산자의 우선순위

수학식을 풀 때 보통은 왼쪽에서 오른쪽 방향(→)으로 계산합니다. 그리고 곱셈(*)이나 나눗셈(/)이 있다면 이것들을 먼저 계산합니다. 또, 괄호가 있으면 그 괄호 안의 식을 제일 먼저 계산합니다. 즉, 연산의 우선순위로 보면 괄호가 제일 먼저 고, 곱셈과 나눗셈이 그다음, 덧셈과 뺄셈이 마지막이 됩니다.

그림 2-8 수학식의 연산 순서



프로그래밍 언어에서의 연산자도 수학 연산자와 마찬가지로 연산의 우선순위가 있습니다. 수학 연산자의 우선순위와 프로그래밍 언어 연산자의 우선순위는 상당히 비슷합니다. 다만 프로그래밍 언어에는 추가된 연산자(논리 연산자 등)가 있기 때문에 이들의 우선순위를 알아두는 것이 좋습니다. 프로그래밍 언어 연산자의 우선순위를 나타내면 다음 표와 같습니다.

표 2-7 연산자의 우선순위

우선순위	연산자
1	[] ()
2	! ++ -- (캐스트 연산자)
3	* / %
4	+ -

우선순위	연산자
5	<code><< >></code>
6	<code>< >= <= == !=</code>
7	<code>&&</code>
8	<code> </code>
9	<code>= += -= *= /= %= <<= >>=</code>

[코드 2-9] 우선순위 적용의 예

```
class Priority {
    public static void main(String[] args){
        System.out.println("3+2*8-4=" + (3+2*8-4));
        "3+2×8-4=15"를 시스템 콘솔에 출력하고 한 줄 띄운다

        System.out.println("(3+2)*(8-4)=" + (3+2)*(8-4));
        "(3+2)×(8-4)=20"를 시스템 콘솔에 출력하고 한 줄 띄운다

        System.out.println("3<<2=" + (3<<2));
        "3<<2=12"를 시스템 콘솔에 출력하고 한 줄 띄운다

        System.out.println("8>>3=" + (8>>3));
        "8>>3=1"를 시스템 콘솔에 출력하고 한 줄 띄운다
    }
}
```

실행 결과

3+2*8-4=15
 $(3+2) \times (8-4) = 20$
 $3 \ll 2 = 12$
 $8 \gg 3 = 1$

NOTE 시프트 연산자(<<, >>)

컴퓨터는 모든 데이터를 0과 1로 표현하는 2진수를 사용합니다. 1bit는 0과 1, 두 가지 값 중 하나를 가질 수 있습니다. 예를 들어, byte a = 2; 라고 선언을 하면 0000 0010으로 메모리에 저장됩니다.

영어로 'shift(시프트)'는 '옮기다'라는 뜻입니다. 프로그래밍 언어에서의 시프트 연산자 역시 옮기는 역할을 하는데, 바로 bit 값을 옮기는 역할을 합니다. '<<'는 bit를 원쪽으로 이동시키고, '>>'는 bit를 오른쪽으로 이동시킵니다.

앞서 나온 `byte a = 2;`를 이용하여 시프트 연산자의 예를 살펴보면 다음과 같습니다.

```
result = a << 2; // a의 bit 값을 원쪽으로 2칸 이동해라(0000 0010 => 0000 1000)
                // result에 대입된 값은 8이 됨
a = result >> 3; // result의 bit 값을 오른쪽으로 3칸 이동해라(0000 1000 => 0000 0001)
                  // a에 대입된 값은 1이 됨
```

정리하면, '<<'는 원쪽으로 bit를 이동시키고, 이동시킨 bit 수만큼 2배의 값이 됩니다. '>>'는 오른쪽으로 bit를 이동시키고, 이동시킨 bit 수만큼 1/2배의 값이 됩니다.

2.3 여러 가지 배열

2.3.1 배열

변수가 특정 타입의 데이터를 담는 상자였다면, 배열은 이런 변수들의 묶음입니다. 배열을 선언할 때는 `new` 연산자를 사용합니다.

배열 요소 하나하나는 '배열 이름[요소 번호]'로 나타낼 수 있습니다. 주의할 점은 배열의 요소 번호(인덱스)는 1이 아닌 0부터 시작한다는 것입니다. 따라서 [그림 2-9] 상단의 `arr` 배열의 첫 번째 요소를 참조하고 싶다면 '`arr[0]`'이라고 나타냅니다.

그림 2-9 배열의 개념 및 선언

배열: 특정 타입의 데이터를 담는 상자(변수)들의 묶음



```
int arr[] = new int[3];
```

정수 값을 담는 arr이라는 이름의 배열을 3개 크기만큼 새로 선언한다



```
String arr2[] = new String[4];
```

문자열 값을 담는 arr2라는 이름의 배열을 4개 크기만큼 새로 선언한다

일반 변수에 값을 저장하는 것처럼 배열 요소 하나하나에도 값을 저장할 수 있습니다.

[배열 요소에 값 저장 1]

```
int arr[] = new int[3];
```

정수 값을 담는 arr이라는 이름의 배열을 3개 크기만큼 새로 선언한다

```
arr[0] = 1;
```

배열 요소 arr[0]에 1을 저장한다

```
arr[1] = 2;
```

배열 요소 arr[1]에 2를 저장한다

```
arr[2] = 3;
```

배열 요소 arr[2]에 3을 저장한다

더 간편하게 다음처럼 배열의 선언과 동시에 배열 요소에 값을 저장할 수도 있습니다.

[배열 요소에 값 저장 2]

```
int arr[] = new int[]{1,2,3};
```

정수 값을 담는 arr이라는 이름의 배열을 새로 선언하고 각 요소에 1,2,3을 저장한다

↓ new 생략 가능

```
int arr[] = {1,2,3};
```

정수 값을 담는 arr이라는 이름의 배열을 선언하고 각 배열 요소에 1,2,3을 저장한다

[코드 2-1]에서는 문자열 값을 담는 String 변수 4개에 값을 저장하고 이 값을 출력하기 위하여 비슷한 문구가 여러 번 반복되어 나타났습니다. 그러나 [배열 요소에 값 저장 2]을 활용하면 [코드 2-10]처럼 간편하게 나타낼 수 있습니다. ‘Array’라는 이름의 클래스를 만들어 [코드 2-10]을 입력해 봅시다.

[코드 2-10] 변수의 선언, 값 저장과 출력 2

```
class Array {  
    public static void main(String[] args) {  
        String str[] = {"Hello", "Java", "Nice to meet you", "Bye!"};  
        문자열 값을 담는 str이라는 이름의 배열을 선언하고 각 배열 요소에 Hello, Java, Nice to meet you, Bye를 저장한다  
        System.out.println(str[0]);  
        str[0] 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다  
        System.out.println(str[1]);  
        str[1] 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다  
        System.out.println(str[2]);  
        str[2] 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다  
        System.out.println(str[3]);  
        str[3] 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다  
    }  
}
```

실행 결과

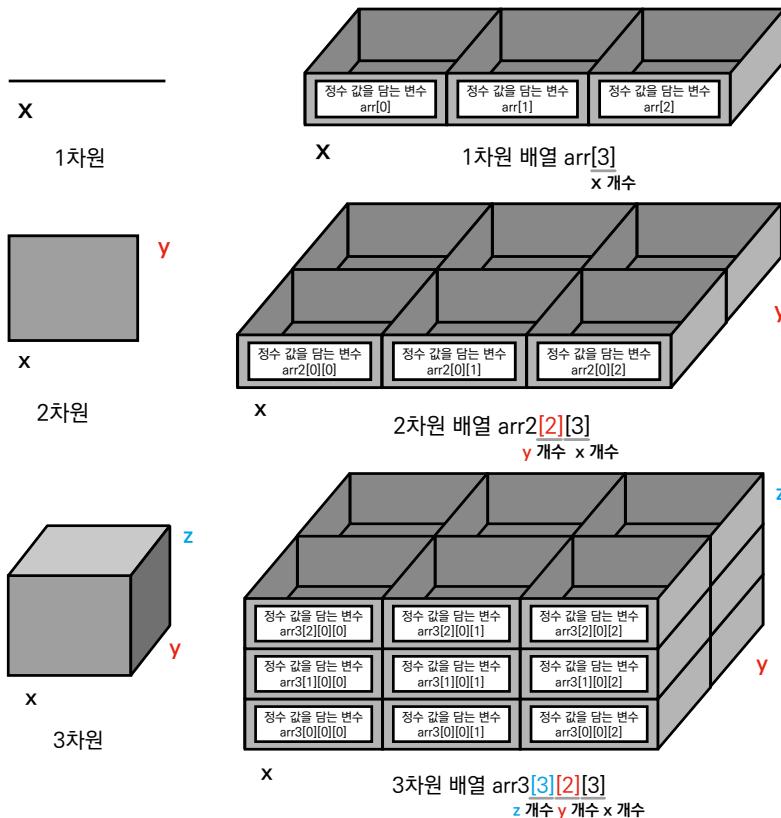
```
Hello  
Java  
Nice to meet you  
Bye
```

2.3.2 다차원 배열

보통 1차원이 아닌 2차원, 3차원 등을 통틀어 다차원이라고 합니다. 앞에서 배열을 특정 데이터 타입을 담는 상자(변수)들의 끝음이라고 정의하였는데, 이때 배열

은 1차원 배열을 말합니다. 1차원 배열은 개념적으로 상자들이 일렬로 나열되어 있습니다. 반면, 다차원 배열은 이런 상자 묶음들이 2차원, 3차원처럼 나열되어 있는 것을 말합니다.

그림 2-10 다차원 배열의 개념



다차원 배열은 1차원 배열을 생성하는 것과 매우 유사합니다. 다만 배열 표시인 '[]'의 개수를 몇 차원인지에 따라 늘려 주면 됩니다.

[다차원 배열의 선언]

- (1) 배열이 어떤 타입의 데이터를 담을 것인지 명시 → `int`
- (2) 배열의 이름을 적음 → `int arr2`
- (3) 몇 차원 배열인지 적음(차원 수만큼 []를 적음) → `int arr2[][]`

- (4) 등호(=)를 적고, new 연산자를 적음
 (5) 배열이 담을 데이터 타입과 배열의 크기를 적음
 (6) 쌍반점(:)을 찍음
- int arr2[][] = new int[2][3];
 → int arr2[][] = new int[2][3];
 → int arr2[][] = new int[2][3];

int arr2[][] = new int[2][3];

정수 값을 담는 arr2라는 이름의 2차원 배열을 2행 3열의 크기만큼 새로 선언한다

다음 [다차원 배열의 선언]에서 생성한 2차원 배열 arr2에 값을 저장하는 것도 1 차원 배열에 값을 저장하는 것과 유사합니다. 자주 사용되는 2차원 배열은 표(또는 엑셀 시트)와 비슷한 형태로 생각하면 됩니다.

[다차원 배열 요소에 값 저장 1]

int arr2[][] = new int[2][3];

정수 값을 담는 arr2라는 이름의 2차원 배열을 2행 3열의 크기만큼 새로 선언한다

1		

arr2[0][0] = 1;

배열 요소 arr2[0][0]에 1을 저장한다

1		

arr2[0][1] = 2;

배열 요소 arr2[0][1]에 2를 저장한다

1	2	

arr2[0][2] = 3;

배열 요소 arr2[0][2]에 3을 저장한다

1	2	3

arr2[1][0] = 4;

배열 요소 arr2[1][0]에 4를 저장한다

1	2	3
4		

arr2[1][1] = 5;

배열 요소 arr2[1][1]에 5를 저장한다

1	2	3
4	5	

arr2[1][2] = 6;

배열 요소 arr2[1][2]에 6을 저장한다

1	2	3
4	5	6

다음처럼 다차원 배열의 선언과 동시에 배열 요소에 값을 저장할 수도 있습니다.

이때, '[']' 개수만큼 '{ }' 개수도 늘어나야 합니다.

[다차원 배열 요소에 값 저장 2]

int arr2[][] = {{1,2,3},

정수 값을 담는 arr2라는 이름의 2차원 배열을 선언하고 1행 각 배열 요소에 1,2,3을,

{4,5,6}};

2행 각 배열 요소에 4,5,6을 저장한다

이번 장에서 공부한 다차원 배열을 활용한 ‘MultidimensionalArray’라는 클래스를 만들어 봅시다. [코드 2-11]을 보면 정수 값을 담는 int 형 2차원 배열에 값을 저장하고 이 값을 출력하고 있습니다.

[코드 2-11] 다차원 배열의 선언, 값 저장과 출력

```
class MultidimensionalArray {  
    public static void main(String[] args) {
```

int arr2[][] = {{1,2,3},

정수 값을 담는 arr2라는 이름의 2차원 배열을 선언하고 1행 각 배열 요소에 1,2,3을,

{4,5,6}};

2행 각 배열 요소에 4,5,6을 저장한다

```
System.out.println("arr2[0][0]: " + arr2[0][0]);
```

문자열 “arr2[0][0]:” 다음에 arr2[0][0] 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다

```
System.out.println("arr2[0][1]: " + arr2[0][1]);
```

문자열 “arr2[0][1]:” 다음에 arr2[0][1] 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다

```
System.out.println("arr2[0][2]: " + arr2[0][2]);
```

문자열 “arr2[0][2]:” 다음에 arr2[0][2] 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다

```
System.out.println("arr2[1][0]: " + arr2[1][0]);
```

문자열 “arr2[1][0]:” 다음에 arr2[1][0] 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다

```
System.out.println("arr2[1][1]: " + arr2[1][1]);
```

문자열 “arr2[1][1]:” 다음에 arr2[1][1] 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다

```
System.out.println("arr2[1][2]: " + arr2[1][2]);
```

문자열 “arr2[1][2]:” 다음에 arr2[1][2] 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다

```
}
```

실행 결과

```
arr2[0][0]: 1
```

```
arr2[0][1]: 2  
arr2[0][2]: 3  
arr2[1][0]: 4  
arr2[1][1]: 5  
arr2[1][2]: 6
```

NOTE

자바에서 '+' 연산자 앞에 문자열이 있는 경우, 문자열과 '+' 연산자 뒤의 값이 접합되어 하나의 문자열이 됩니다. 따라서 [코드 2-11]에서 `System.out.println("arr2[0][0]: " + arr2[0][0]);`'의 실행 결과로 'arr2[0][0]'이란 문자열에 'arr2[0][0]' 변수의 값인 1이 접합되어 'arr2[0][0]: 1'이 출력된 것입니다. 이뿐만 아니라 두 개의 문자열을 합칠 때도 '+' 연산자를 사용할 수 있습니다. 예를 들어, `String str = "안녕"+" 좋은 아침!";`'처럼 '+' 연산자로 두 문자열을 접합한 값을 변수 str에 저장하고, `System.out.println(str);`'을 실행하면 '안녕? 좋은 아침!'이 출력됩니다.

2.4 반복문

[코드 2-11]을 보면 `System.out.println()` 구문이 6번 반복됩니다. `System.out.println`은 간단히 말해서 '화면에 팔호 안의 내용을 찍어라'라는 명령인데, 화면에 6개의 내용을 찍기 위해 일일이 같은 코드를 반복하여 적은 것입니다. 6번 정도는 직접 적을 수도 있지만 그 횟수가 늘어나면 상황은 달라집니다. 예를 들어, 100개의 내용을 반복하고 싶다면 직접 같은 코드 100번을 적어 주어야 하는데 많이 힘들겠죠? 이렇게 같은 처리를 여러 번 반복해야 할 때 `for`문이나 `while`문 같은 반복문을 쓸 수 있습니다.

2.4.1 for문

영어에서 `for`가 가진 의미 중에는 '~(기간) 동안'이라는 의미가 있습니다. 자바에서도 `for`는 '~동안'이라는 뜻으로 사용합니다. 유의할 점은 `for`문에는 몇 번을 반복할 것인지에 대한 조건이 필요하다는 점입니다.

[영어의 for]

I haven't spoken to him for three months.

나는 말하지 않고 있다 그에게(그와) 동안 3개월

[자바의 for]

for(int i=0; i<4; i++) {

정수 값을 담는 **i**라는 변수에 0을 대입하고 **i**를 1씩 증가시키면서 **i**가 4보다 작을 동안 {} 안을 반복해라

처리할 명령들

}

또한, 같은 처리를 여러 번 반복할 때 언제부터 시작하여 언제 끝낼 것인지 얼마나 자주 실행할 것인지에 대한 기준이 필요합니다. 이를 위해 for문에는 '카운터'라는 것이 있습니다. for문의 선언 방법은 다음과 같습니다.

[for문의 선언]

(1) for를 적음

→ **for**

(2) () 안에 반복에 대한 조건을 적음

→ **for()**

카운터를 선언하고 초기화함

→ **for(int i=0;**

반복을 계속하기 위한 조건을 적음

→ **for(int i=0; i<4;**

카운터의 증가/감소 방법을 적음

→ **for(int i=0; i<4; i++)**

(3) {} 안에 반복하여 처리할 명령들을 적음

→ **for(int i=0; i<4; i++) {**

처리할 명령들

}

for(int i=0; i<4; i++) {

정수 값을 담는 **i**라는 변수에 0을 대입하고 **i**를 1씩 증가시키면서 **i**가 4보다 작을 동안 {} 안을 반복해라

처리할 명령들

}

[코드 2-10]에서 나온 것처럼 System.out.println()을 4번 반복하도록 for 문을 사용하고 싶을 때는 [코드 2-12]와 같이 코드를 작성하면 됩니다.

```

class Array {
    public static void main(String[] args) {
        String str[] = {"Hello", "Java", "Nice to meet you", "Bye!"};
        문자열 값을 담는 str이라는 이름의 배열을 선언하고 각 배열 요소에 Hello, Java, Nice to meet you, Bye를 저장한다

        for(int i=0; i<4; i++) {
            정수 값을 담는 i라는 변수에 0을 저장하고 i를 1씩 증가시키면서 i가 4보다 작을 동안 {} 안을 반복해라

            System.out.println(str[i]);
            str[i] 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다
        }
    }
}

```

실행 결과

```

Hello          // i가 0일 때
Java          // i가 1일 때
Nice to meet you  // i가 2일 때
Bye!          // i가 3일 때

```

이 프로그램에서는 for문의 {} 안에 있는 처리 명령문에서 카운터 i를 1차원 배열 str의 인덱스(배열 요소 번호)로 사용하였습니다. 이를 심화하면, 2차원 배열의 경우 이중 for문을 사용하여 그 인덱스를 표현할 수 있습니다. [코드 2-11]의 2 차원 배열을 출력하는 코드에 for문을 사용하면 [코드 2-13]과 같습니다.

```

class MultidimensionalArray {
    public static void main(String[] args) {
        int arr2[][] = {{1,2,3},
        정수 값을 담는 arr2라는 이름의 2차원 배열을 선언하고 1행 각 배열 요소에 1,2,3을,
        {4,5,6}};
        2행 각 배열 요소에 4,5,6을 저장한다
    }
}

```

```
for(int i=0; i<2; i++) {
```

정수 값을 담는 i라는 변수에 0을 저장하고 i를 1씩 증가시키면서 i가 2보다 작을 동안 {} 인을 반복해라

```
    for(int j=0; j<3; j++) {
```

정수 값을 담는 j라는 변수에 0을 저장하고 j를 1씩 증가시키면서 j가 3보다 작을 동안 {} 인을 반복해라

```
        System.out.println("arr2["+i+"]["+j+"]: " + arr2[i][j]);
```

문자열 "arr2" 다음에 변수 i 값, 다음에 문자열 "]" , 다음에 변수 j 값, 다음에 문자열 "]" , 다음에 arr2[i][j] 변수 값을 시스템 콘솔에 출력하고 한 줄 띄운다

```
}  
}  
}  
}
```

실행 결과

```
arr2[0][0]: 1      // i가 0, j가 0일 때  
arr2[0][1]: 2      // i가 0, j가 1일 때  
arr2[0][2]: 3      // i가 0, j가 2일 때  
arr2[1][0]: 4      // i가 1, j가 0일 때  
arr2[1][1]: 5      // i가 1, j가 1일 때  
arr2[1][2]: 6      // i가 1, j가 2일 때
```

[코드 2-13]에서 나온 것처럼 이중 for문이란 for문 안에 for문이 있는 구조를 말합니다. 이중 for문에서는 안쪽에 있는 for문이 바깥쪽 for 문의 실행 횟수만큼 실행됩니다. 즉, [코드 2-13]에서 바깥쪽 for문의 카운터 i가 0에서 2 미만이 될 때까지 1씩 증가하므로 i=0일 때와 i=1일 때 각각 안쪽 for문을 실행합니다. 즉, 안쪽 for문은 총 2번 실행이 됩니다. 이중 for문 뿐만 아니라 삼중 for문, 사중 for문도 가능하지만 프로그램 실행 속도가 그만큼 느려져서 잘 사용하지는 않습니다.

연습 2-1

구구단을 for문을 사용하여 계산해 보세요(프로그래밍 언어에서는 곱셈을 '*'로 표현합니다).

1X1 = 1

1X2 = 2

...

9X9 = 81

2.4.2 while문

영어에서 while은 ‘~하는 동안’이라는 뜻의 단어인데, 자바에서도 while은 ‘~하는 동안’이라는 의미로 사용합니다. 몇 번 반복할 것인지를 정하고 시작하는 for 문과는 달리, while문은 반복 횟수를 정함이 없이 특정 조건이 충족되는 동안 계속 반복 처리합니다.

[영어의 while]

Make hay while the sun shines.

만들어라 건초를 **동안** 해가 **비치는**

[자바의 while]

while(i == 0) {

변수 i의 값이 0과 같을 **동안** { } 안을 반복해라

처리할 명령들

}

NOTE =와 ==의 차이

수학에서 '='은 '같다'라는 의미입니다. 그런데 프로그래밍 언어에서 '='은 '값을 저장한다'라는 의미로 쓰이고 '='의 왼쪽에는 변수, 오른쪽에는 왼쪽 변수에 저장할 값을 적습니다(예: int a = 1;).

반면, 프로그래밍 언어에서는 '같다'라는 의미로 '=='를 사용합니다. 따라서 변수 a에 1을 저장한다는 'int a = 1'이라고 표현하고, 변수 a의 값이 1과 같은가를 표현할 때는 'a == 1'이라고 적습니다.

즉, 반복 횟수가 얼마나 필요한지 모를 때 while문을 사용합니다. 예를 들어, 택배기사님이 오늘 집에 방문한다고 생각해 봅시다. 경비실에서 택배를 맡아주지 않아서 할 수 없이 집에서 종이학을 접으며 택배기사님이 올 때까지 기다리기로 하였습니다. 그렇지만 택배기사님이 언제 올지 모르기 때문에 내가 종이학을 접는 횟수는 몇 번이라고 꼬집어 말할 수 없습니다. 이런 상황을 while문으로 적으면 다음과 같이 표현할 수 있겠죠?

```
while(i == 0){ // 0은 택배 아저씨가 안 왔음을 의미
    종이학을 접는다
}
```

while문의 선언 방법은 다음과 같습니다.

[while문의 선언]

```
(1) while을 적음                                -> while
(2) () 안에 반복에 대한 조건을 적음          -> while(i==0)
(3) {} 안에 반복하여 처리할 명령들을 적음      -> while(i==0) {
                                                처리할 명령들
}
```

```
while(i == 0) {
    변수 i의 값이 0과 같을 동안 {} 안을 반복해라
    처리할 명령들
}
```

다시 택배기사님을 기다리는 상황으로 돌아가 봅시다. 택배기사님이 깜빡하고 우리집에 방문하지 않으면 나는 6시간이고, 12시간이고 종이학을 무한 번 접는 상황에 처할 수 있습니다. while문에서도 () 안의 조건이 항상 성립하도록 하면 {} 안의 명령들을 무한 번 수행합니다. 이런 것을 ‘무한 루프’라고 하며, 이를 피하기 위해 {} 안에는 () 안의 조건을 변경하거나 루프를 빠져나갈 수 있는 명령어를 적어 주어야 합니다. 예를 들면 다음과 같이 표현할 수 있습니다.

```
while(i == 0){ // 0은 택배 아저씨가 안 왔음을 의미  
    종이학을 접는다  
    if(택배 아저씨가 옴) {  
        i = 1; // 1은 택배 아저씨가 왔음을 의미  
    }  
}
```

[코드 2-10]에서 `System.out.println()`이 4번 반복된 것을 `while`문을 사용해 표현하려면 다음과 같이 코드를 작성하면 됩니다.

[코드 2-14] 코드 2-10에 `while`문 적용

```
class Array {  
    public static void main(String[] args) {  
        String str[] = {"Hello", "Java", "Nice to meet you", "Bye!"};  
        문자열 값을 담는 str이라는 이름의 배열을 선언하고 각 배열 요소에 Hello, Java, Nice to meet you, Bye를 저장한다  
        int i=0;  
        정수 값을 담는 i라는 변수를 선언하고 0을 저장한다  
        while(i<4) {  
            변수 i의 값이 4보다 작을 동안 {} 안을 반복해라  
            System.out.println(str[i]);  
            str[i] 변수 값을 시스템 콘솔에 출력하고 한 줄 띄운다  
            i++;  
            변수 i의 값을 1씩 증가시켜라  
        }  
    }  
}
```

실행 결과

```
Hello          // i가 0일 때  
Java           // i가 1일 때  
Nice to meet you // i가 2일 때  
Bye!           // i가 3일 때
```

NOTE | 코드에 주석 넣기

글을 작성할 때 내용 흐름과는 관계없지만 부가 설명을 하고 싶다면 주석을 닙니다. 코드를 작성할 때도 이 코드가 어떤 의미인지 주석을 달아 설명하고 싶을 때가 있습니다. 자바에서 주석 표시는 '//' 와 '/* */'가 있습니다. //는 한 줄만 주석으로 처리하고, /* */는 여러 줄을 주석으로 처리합니다.

주석 처리 방법은 다음과 같습니다.

1) 한 줄만 주석 처리할 때

```
class Array {  
    public static void main(String[] args) {  
        String str[] = { "Hello", "Java", "Nice to meet you", "Bye!" };  
        int i = 0;  
        while (i < 4) {      //배열 str의 내용 출력  
            System.out.println(str[i]);  
            i++;  
        }  
    }  
}
```

2) 여러 줄을 주석 처리할 때

```
class Array {  
    public static void main(String[] args) {  
        /* 배열에 담긴 내용을 while문을 이용하여 출력하는 프로그램  
         작성일: 2015-11-10  
         작성자: hin  
        */  
        String str[] = { "Hello", "Java", "Nice to meet you", "Bye!" };  
        int i = 0;  
        while (i < 4) {  
            System.out.println(str[i]);  
            i++;  
        }  
    }  
}
```

//와 /* */로 주석 처리된 부분은 컴파일러가 읽지 않기 때문에 코드에 대한 부가 설명뿐만 아니라 코드 실행 시 어떤 부분을 생략하고 실행하고 싶을 때에도 //와 /* */를 사용합니다.

`while`문의 응용으로 `do~while`문이 있습니다. `while`문은 반복 처리를 위한 조건이 충족해야만 {} 안의 내용을 처리하는 반면, `do~while`문은 반복 처리 조건에 상관없이 일단 한 번은 수행하고, 그다음 반복 처리하기 위한 조건을 고려합니다. `do~while`문의 선언 방법은 다음과 같습니다.

[`do~while`문의 선언]

- (1) do를 적음 → do
- (2) {} 안에 반복하여 처리할 명령들을 적음 → do {
 처리할 명령들
}
- (3) while을 적고, 그 옆 () 안에 반복에 대한 조건을 적음 → do {
 처리할 명령들
} `while(i==0);`

do{

일단 {} 안의 명령을 수행해라

처리할 명령들

} `while(i == 0);`

변수 i의 값이 0과 같을 동안 {} 안을 반복해라

연습 2-2

`do~while`문을 사용하여 [코드 2-10]의 결과와 똑같이 출력되도록 프로그래밍해 보세요.

NOTE 반복문의 흐름을 바꾸는 `break`와 `continue`

`for`문과 `while`문은 조건이 맞지 않을 때까지 반복 처리를 하는데, 이 반복 처리의 흐름을 바꾸어 주는 `break`와 `continue`가 있습니다.

- **break** 반복문을 종료합니다.

```
int a = 1;  
while(true){  
    if(a == 3){  
        break;
```

```
    }  
    a++;  
}  
System.out.println(a);
```

앞의 코드를 보면, while문 내에서 a를 1씩 더하다가 a의 값이 30이 되면 break 명령어로 while문을 멈추고 System.out.println(a); 명령어를 실행합니다. 따라서 이 코드의 실행 결과는 30이 됩니다.

- **continue** 반복문 내에서 continue 아래 남겨진 명령어들은 생략하고, 다음 반복으로 넘어갑니다.

```
for(int i=0; i<4; i++){  
    if(i==2){  
        continue;  
    }  
    System.out.print(i);  
}
```

앞의 코드를 보면, if문 내에서 i를 1씩 더하면서 그 값을 출력하다가 i의 값이 2가 되면 continue 명령어로 그 아래 System.out.print(i);는 생략하고, 다음 반복으로 넘어갑니다. 그리고 System.out.print()는 System.out.println()과 다르게 시스템 콘솔에 출력 후 줄바꿈을 하지 않습니다. 따라서 이 코드의 실행 결과는 0130이 됩니다.

2.5 조건문

우리는 일상생활에서 조건문을 생각보다 자주 사용합니다. 예를 들어, “내일 비가 오면 집에 있을 거야!”라든가 “대학생이 되면 유럽여행을 갈 거야!”와 같이 ‘~라면 (조건) ~할 것이다’라는 표현을 자주 씁니다. 프로그래밍 언어에서도 어떤 경우엔 이런 처리를, 다른 경우엔 저런 처리를 해주고 싶을 때 if문이나 switch문 같은 조건문을 사용합니다.

2.5.1 if문

영어에서 if는 ‘(만약) ~이면’이라는 뜻입니다. 자바 프로그래밍 언어의 if도 이와 같이 조건에 따라 처리를 다르게 할 때 사용합니다.

[영어의 if]

If it rains, I will stay at home.
만약 비가 온다면 나는 집에 있을 것이다

[자바의 if]

if(i<=5) {

변수 i의 값이 5보다 작거나 같으면 {} 안을 처리해라

처리할 명령들

}

if문의 선언 방법은 다음과 같습니다.

[if문의 선언]

(1) if를 적음

-> if

(2) () 안에 조건을 적음

-> if(i<=5)

(3) {} 안에 처리할 명령들을 적음

-> if(i<=5) {

처리할 명령들

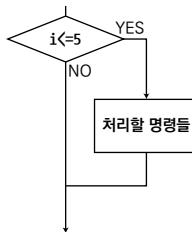
}

if(i <= 5) {

변수 i의 값이 5보다 작거나 같으면 {} 안을 처리해라

처리할 명령들

}



“내일 비가 오면 집에 있고, 그렇지 않으면 소풍을 갈 거야.”라는 말처럼 종종 조건이 성립하면 무엇을 하고, 성립하지 않으면 다른 것을 한다고 말하곤 합니다. 이를 자바 프로그래밍 언어에서는 if문의 심화로 if~else문을 이용하여 표현할 수 있으며, ‘(만약)~라면 A를 하고, 그렇지 않으면 B를 한다’는 뜻입니다. if~else문의 선언 방법은 다음과 같습니다.

- (1) if를 적음
 (2) () 안에 조건을 적음
 (3) { } 안에 처리할 명령들을 적음
 (4) else를 적고 { } 안에 처리할 명령들을 적음

-> if
 -> if(*i*<=5)
 -> if(*i*<=5) {
 처리할 명령들 A
 }
 -> if(*i*<=5) {
 처리할 명령들 A
 }
 else {
 처리할 명령들 B
 }

if(*i* <= 5) {
 변수 *i*의 값이 5보다 작거나 같으면 A를 처리해라

처리할 명령들 A

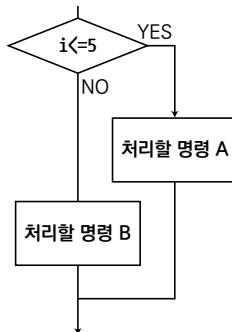
}

else {

그렇지 않으면 B를 처리해라

처리할 명령들 B

}



'Iftest'라는 이름의 클래스를 만들어 if~else문을 사용해 봅시다. if~else문을 사용한 예는 다음과 같습니다.

[코드 2-15] if~else문의 사용

```
class Iftest {
    public static void main(String[] args) {
```

int i = 2;

정수 값을 담는 *i*라는 이름의 변수를 선언하고 2를 저장한다

if(*i*<=5) {

변수 *i*의 값이 5보다 작거나 같으면

System.out.println(*i*+"는 5보다 작거나 같다");

"2(변수 *i*의 값)는 5보다 작거나 같다"를 시스템 콘솔에 출력하고 한 줄 띄운다

```
 }else {
```

그렇지 않으면

```
 System.out.println(i+"는 5보다 크다");
```

"2(변수 i의 값)는 5보다 크다"를 시스템 콘솔에 출력하고 한 줄 띄운다

```
 }
```

```
}
```

```
}
```

실행 결과

2는 5보다 작거나 같다

앞에서 본 if문과 if~else문에서는 조건이 1개만 있었습니다. 그런데 조건이 여러 개 있을 때는 어떻게 사용할까요? 예를 들어, “비가 오면 집에 있고, 눈이 오면 눈싸움을 하고, 구름이 끼면 자전거를 탄다.”라고 말할 때 조건은 총 3개(비 옴, 눈 옴, 구름 끼)가 있습니다. 이럴 때 프로그래밍 언어에서는 if~else if문을 사용할 수 있습니다. if~else if문의 선언 방법은 다음과 같습니다.

[if~else if문의 선언]

(1) if를 적음

-> if

처리할 명령이 1개일 때는
{ }를 생략할 수 있습니다.

(2) () 안에 조건을 적음

-> if(i<=5)

(3) { } 안에 처리할 명령들을 적음

-> if(i<=5)

처리할 명령 A

(4) else if(조건)을 적고 { } 안에 처리할 명령들을 적음

-> if(i<=5)

처리할 명령 A

else if(i>5 && i<=10)

처리할 명령 B

(5) else를 적고 { } 안에 처리할 명령들을 적음

-> if(i<=5)

처리할 명령 A

else if(i>5 && i<=10)

처리할 명령 B

else

처리할 명령 C

if(i <= 5) {
변수 i의 값이 5보다 작거나 같으면 A를 처리해라

처리할 명령들 A

}else if(i > 5 && i <= 10) {
변수 i의 값이 5보다 크고 그리고 10보다 작거나 같으면 B를 처리해라

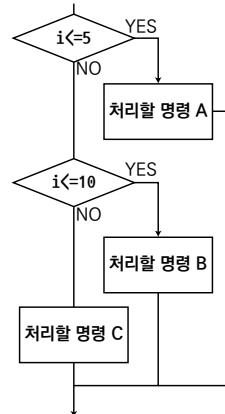
처리할 명령들 B

}else {

그렇지 않으면 C를 처리해라

처리할 명령들 C

}



연습 2-3

교수님이 기말고사 성적에 따라 학점을 매기려고 합니다. 성적이 80 이상이면 A, 60 이상이면 B, 40 이상이면 C, 40 미만이면 D라는 학점 기준이 있습니다. 강희은 학생이 45점을 맞았을 때, 어떤 학점을 받는지 출력하는 프로그램을 작성해 봅시다.

2.5.2 switch문

if~else if문처럼 조건이 여러 가지 있을 때 그중 하나를 선택해 처리하는 것으로 switch문이 있습니다. ‘switch’라는 단어는 영어에서 ‘바꾸다, 전환하다’라는 뜻으로, 프로그래밍 언어에서도 ‘조건에 따라 프로그램의 실행 흐름을 바꾸다, 전환하다’라는 의미로 사용합니다.

[영어의 switch]

You'd better switch to night mode.
너는 야간 모드로 전환하는 게 더 좋을 거야

[자바의 switch]

switch(i) {
(i 안의 값에 따라 프로그램의 흐름을 전환해라) 변수 i

case 0: 처리할 명령 A;
0일 때 처리할 명령 A를 실행하고

break;
여기서 멈춘다

case 1: 처리할 명령 B;
1일 때 처리할 명령 B를 실행하고

break;
여기서 멈춘다

case 2: 처리할 명령 C;
2일 때 처리할 명령 C를 실행하고

break;
여기서 멈춘다

default: 처리할 명령 D;
아무것도 아닐 때 처리할 명령 D를 실행한다

break;
여기서 멈춘다

}

switch문을 선언하는 방법은 다음과 같습니다.

[switch문의 선언]

(1) switch를 적음
(2) () 안에 상수, 변수, 식을 적음
(3) { } 안에 case별로 처리할 명령들을 적고, break를 적음
*break가 없는 경우 그 밑의 case에 처리할 명령까지 실행된다!
ex) case 0: 처리할 명령 A;
 case 1: 처리할 명령 B;
 break;
이 예제의 경우 처리할 명령 A, B 모두 실행된다.
(4) case로 적을 것이 없으면 그 외 처리를 위해
 default를 적고 처리할 명령과 break를 적음

-> switch
 -> switch(i)
 -> switch(i) {
 case 0: 처리할 명령 A;
 break;
 case 1: 처리할 명령 B;
 break;
 case 2: 처리할 명령 C;
 break;
 default: 처리할 명령 D;
 break;
 }
}

```
switch(i) {  
    변수 i의 값에 따라 프로그램의 흐름을 전환해라
```

case 0: 처리할 명령 A;

0일 때 처리할 명령 A를 실행하고

break;

여기서 멈춘다

case 1: 처리할 명령 B;

1일 때 처리할 명령 B를 실행하고

break;

여기서 멈춘다

case 2: 처리할 명령 C;

2일 때 처리할 명령 C를 실행하고

break;

여기서 멈춘다

default: 처리할 명령 D;

아무것도 아닐 때 처리할 명령 D를 실행한다

break;

여기서 멈춘다

}

switch문을 사용한 예는 다음과 같습니다.

[코드 2-16] switch문의 사용

```
class Switchtest {  
    public static void main(String[] args) {
```

int i = 2;

정수 값을 담는 i라는 이름의 변수를 선언하고 2를 저장한다

switch(i*5) {

변수 i의 값×5의 값에 따라 프로그램의 흐름을 전환해라

case 0: System.out.println(i+"*5=0");

0일 때 “2(변수 i의 값)×5=0”을 시스템 콘솔에 출력하고 한 줄 띄운다



break;

여기서 멈춘다

case 10: System.out.println(i+"*5=10");

10일 때 “2(변수 i의 값)×5= 10”를 시스템 콘솔에 출력하고 한 줄 띄운다

break;

여기서 멈춘다

case 15: System.out.println(i+"*5=10");

15일 때 “2(변수 i의 값)×5= 15”를 시스템 콘솔에 출력하고 한 줄 띄운다

break;

여기서 멈춘다

default: System.out.println("모르겠음");

아무것도 아닐 때 “모르겠음” 시스템 콘솔에 출력하고 한 줄 띄운다

break;

여기서 멈춘다

```
}
```

실행 결과

2*5 = 10

연습 2-4

과일가게에서 사과는 개당 1,000원, 배는 개당 2,000원, 딸기는 개당 500원, 귤은 300원에 판매하고 있습니다. 어떤 손님이 귤 8개를 산다고 했을 때 얼마를 내야 하는지 출력하는 프로그램을 작성해 봅시다.

클래스와 객체

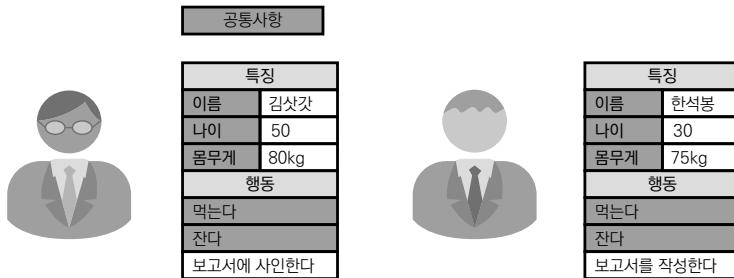
3장에서는 자바의 핵심 개념인 클래스와 객체에 대해서 배웁니다. 3.1장에서 클래스와 객체가 무엇인지 공부하고, 3.2장에서는 클래스를 선언하는 방법을 다룹니다. 3.3장에서는 선언된 클래스로 객체를 생성하고 사용하는 방법에 대해 배우고, 3.4장에서는 클래스의 처리동작을 나타내는 메서드에 대해 공부합니다. 마지막으로 3.5장에서는 클래스를 이용해 객체를 생성할 때 좀 더 편하게 초기 데이터 값을 설정할 수 있도록 도와주는 생성자에 대해 살펴보겠습니다.

3.1 클래스와 객체의 개념

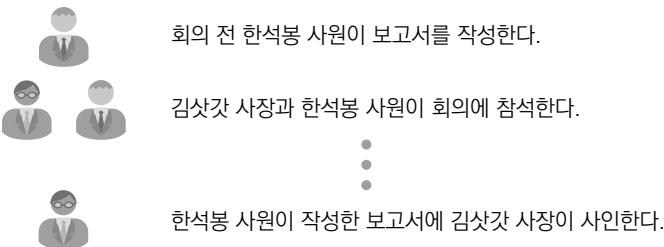
자바는 객체 지향 프로그래밍 언어 중 하나입니다. 여기서 객체란 현실 세계에 존재하는 물체(사람, 책, 집, 자동차 등)를 의미합니다. 사람이 이름, 나이, 키, 몸무게 등의 개별적인 특징이 있고 먹고 자고 말하는 등의 행동들을 하듯, 객체는 각자 특징들을 가지고 어떤 행동들을 합니다. 이런 아이디어를 바탕으로 프로그래밍의 요소 요소를 객체로 표현하고, 이를 엮어 프로그래밍하는 것이 객체 지향 프로그래밍이고, 이를 쉽게 표현할 수 있게 하는 언어가 객체 지향 프로그래밍 언어입니다.

그림 3-1 객체들의 상호작용

객체의 예 - 사람

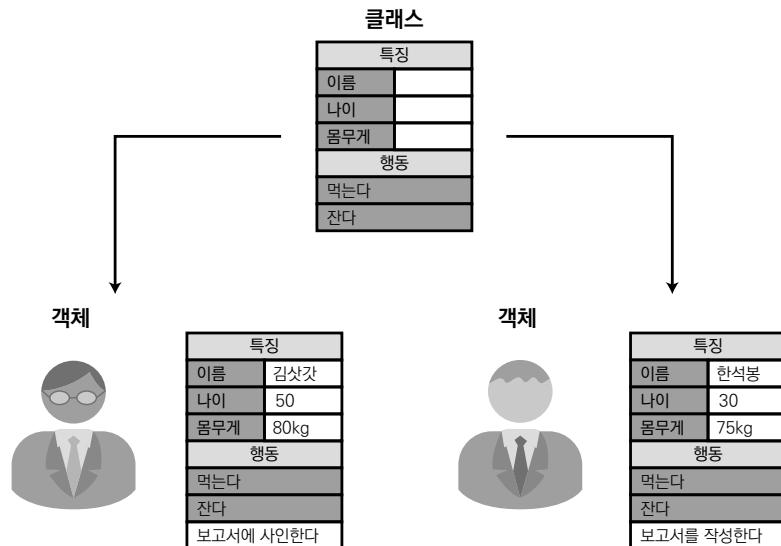


객체들의 상호작용의 예 - 회의



[그림 3-1]에 나타난 것처럼 현실 세계의 사장 김삿갓 씨와 사원 한석봉 씨를 객체, 이 두 명이 회의하는 것을 객체들이 상호작용하는 것으로 생각하고 이를 컴퓨터 육성 게임으로 제작한다고 가정해 보겠습니다. 게임 개발자는 김삿갓과 한석봉이라는 캐릭터의 특징과 행동들을 정의하고, 두 캐릭터가 회의하는 상황을 프로그래밍하고자 합니다. 이때 객체인 김삿갓과 한석봉은 공통점이 있습니다. 우선 개별적인 데이터는 다르지만 어쨌든 이름과 나이, 몸무게 등의 특징으로 표현되고, 먹고 자는 등의 공통적인 행동을 합니다. 게임 개발자가 생각해 보니 김삿갓과 한석봉을 따로따로 생성하는 것보다는 이 공통점을 모아서 하나의 틀을 만들어 두고, 그 틀로 객체들을 만들 때 다른 점만 표현하면 더 쉽고 빠르게 캐릭터를 생성할 수 있을 것이라 판단하였습니다. 따라서 게임 개발자는 사람에 속하는 객체들의 공통점을 모아 ‘클래스’라고 불리는 하나의 사람 틀을 만들고 이 틀을 바탕으로 각 사람 캐릭터를 찍어내듯 프로그래밍하였습니다.

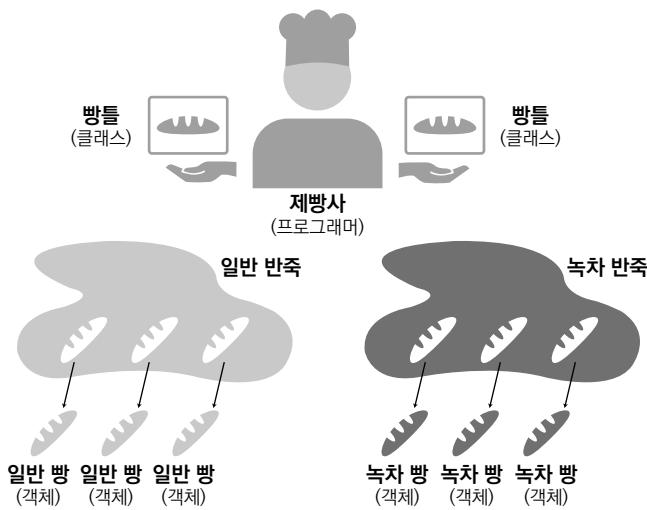
그림 3-2 클래스로 찍어낸 객체들



앞에서 나온 클래스를 다시 정의하면, 클래스 Class는 데이터와 이 데이터를 처리하기 위한 동작이 모여 있는 틀입니다. 객체 Object는 클래스로 찍어낸 실체입니다. 클래스는 틀에 불과하기 때문에 이 자체로는 사용할 수 없고 이를 이용하여 만든 객체만 사용할 수 있습니다. 클래스와 객체의 관계는 마치 빵틀과 이로 제작된 빵의 관계와 같습니다.

그림 3-3 클래스와 객체의 관계

제빵사가 빵틀로 여러 개의 빵을 만든다
= 프로그래머가 클래스로 여러 개의 객체를 만든다



클래스는 해당 클래스로 찍어낸 객체들의 공통된 데이터 구조와 처리동작을 대표하고 있습니다. 클래스의 데이터와 처리동작은 각각 필드 Field와 메서드 Method로 말하기도 합니다.

그림 3-4 클래스의 개념

특징	
이름	
나이	
몸무게	
행동	
먹는다	
잔다	

클래스

} 데이터 = 필드

} 처리동작 = 메서드

연습 3-1

은행에서 개설할 수 있는 계좌의 특징과 행동을 생각해 보고 [그림 3-4]와 같이 표현해 보세요.

3.2 클래스 선언하기

클래스를 프로그래밍 언어로 작성하는 것을 ‘클래스 선언Class Declaration’이라고 합니다. 앞에 나온 [그림 3-4]의 클래스를 자바로 프로그래밍해보면 다음과 같습니다. 관례상 클래스의 이름을 쓸 때는 첫 글자를 대문자로 씁니다.

[코드 3-1] 클래스의 선언

```
class Person {
```

객체를 정의/생성하는 툴인 Person이라는 클래스를 만든다

```
String name;
```

문자열 값을 담는 name이라는 변수를 선언한다

데이터
= 필드

```
int age;
```

정수 값을 담는 age라는 변수를 선언한다

```
int weight;
```

정수 값을 담는 weight라는 변수를 선언한다

```
String eat(){
```

문자열을 리턴값으로 받는 eat라는 메서드를 선언한다

```
    return "eat";
```

이 메서드를 호출한 곳에 “eat”를 리턴한다

```
}
```

처리동작
= 메서드

```
String sleep(){
```

문자열을 리턴값으로 받는 sleep이라는 메서드를 선언한다

```
    return "sleep";
```

이 메서드를 호출한 곳에 “sleep”을 리턴한다

```
}
```

```
}
```

연습 3-2

[연습 3-1]에서 작성했던 것을 토대로 Account 클래스를 작성해보세요.

3.3 객체의 생성과 사용

[코드 3-1]에서 선언한 Person 클래스를 사용하여 [그림 3-2]의 김삿갓과 한석봉 객체를 생성해 보겠습니다. 클래스는 객체를 생성하는 틀인 동시에 String, int, double 등과 같이 변수가 어떤 데이터를 담는가를 표현하는 데이터 타입이기도 합니다. 다시 말하면, 어떤 클래스는 변수가 해당 클래스로 찍어낸 객체를 담는 것을 나타내는 사용자 정의 데이터 타입이라고 볼 수 있습니다. 따라서 문자열 값을 담는 변수 str을 String str;이라고 표현하듯, Person 클래스의 객체를 담는 변수 personObj1은 Person personObj1;이라고 표현합니다.

personObj1은 단순히 데이터를 담는 상자인 변수일 뿐이고 그 안에 데이터를 넣기 위해서는 객체를 생성해야 합니다. 객체를 생성하기 위해서는 new라는 연산자를 사용합니다. new 연산자는 ‘새롭게 ~을 생성한다’라는 의미로 생각하면 됩니다.

[객체의 생성 1]

Person personObj1;

Person이라는 클래스의 객체를 담는 personObj1이라는 변수를 선언한다

personObj1 = new Person();

personObj1 변수에 Person 클래스의 객체를 새로 생성하여 저장한다

[객체의 생성 1]처럼 객체를 담는 변수를 먼저 선언한 후에 그 변수에 객체를 담을 수도 있고, [객체의 생성 2]처럼 한번에 기술할 수도 있습니다.

[객체의 생성 2]

Person personObj = new Person();

Person이라는 클래스의 객체를 담는 personObj이라는 변수를 선언하고

Person 클래스의 객체를 새로 생성하여 저장한다

객체가 생성되었으면 그 객체에 있는 데이터와 메서드에 접근하여 이를 참조하거나 값을 저장하여 사용해야 합니다. 이때는 ‘.’을 사용하여 데이터와 메서드의 소

속을 밝혀 줍니다. 예를 들어, 앞서 생성한 `personObj1`이라는 변수에 저장된 객체의 데이터나 메서드에 접근하고 싶은 경우에는 ‘`personObj1`’ 다음에 데이터 또는 메서드명을 적으면 됩니다. 이와 같은 원리를 바탕으로 [그림 3-2]의 김삿갓 캐릭터를 생성하여 그 특징과 행동들을 기술하면 [코드 3-2]와 같습니다. [코드 3-1]에서 작성한 `Person` 클래스는 그대로 두고, `PersonExample` 클래스를 추가하여 다음과 같이 작성해 봅시다.

[코드 3-2] 객체 생성 후 데이터 및 메서드 사용

```
class PersonExample{
```

객체를 정의/생성하는 툴인 `PersonExample`이라는 클래스를 만든다

```
public static void main(String[] args){
```

```
    Person personObj1;
```

Person이라는 클래스의 객체를 담는 `personObj1`이라는 변수를 선언한다

```
    personObj1 = new Person();
```

`personObj1` 변수에 `Person` 클래스의 객체를 새로 생성하여 저장한다

```
    personObj1.name = "김삿갓";
```

`personObj1` 변수 안의 `name` 변수에 “김삿갓”을 저장한다

```
    personObj1.age = 50;
```

`personObj1` 변수 안의 `age` 변수에 50을 저장한다

```
    personObj1.weight = 80;
```

`personObj1` 변수 안의 `weight` 변수에 80을 저장한다

```
    System.out.println(personObj1.name);
```

`personObj1` 변수 안의 `name` 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다

```
    System.out.println(personObj1.eat());
```

`personObj1` 변수 안의 `eat` 메서드를 호출하여 시스템 콘솔에 출력하고 한 줄 띄운다

```
}
```

```
}
```

〈`personObj1`〉

특징	
이름	
나이	
몸무게	
행동	
먹는다	
잔다	

〈`personObj1`〉

특징	
이름	김삿갓
나이	50
몸무게	80kg
행동	
먹는다	
잔다	

실행 결과

김삿갓

eat

연습 3-3

2015년 8월 31일, 흥길동 씨가 OO은행에서 100,000원으로 계좌를 개설하였습니다. 계좌번호는 1002-03-0001000이었습니다. 흥길동 씨가 개설한 계좌의 객체를 [연습 3-2]에서 작성한 Account 클래스를 사용하여 생성하고, 다음과 같은 실행 결과가 나오게 코드를 작성하세요.

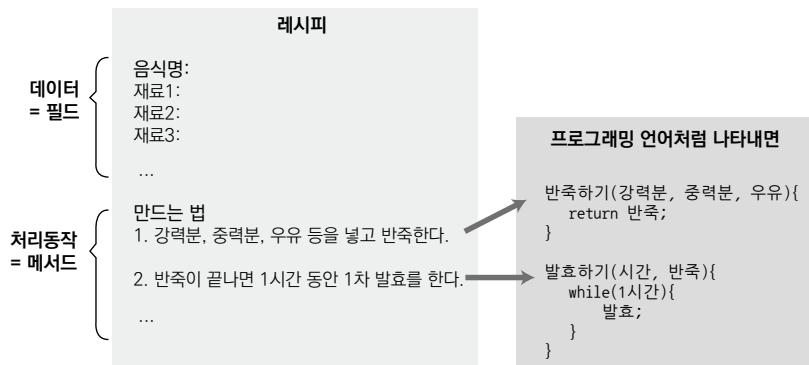
실행 결과

[계좌번호:1002-03-0001000] 잔액:100000

3.4 메서드

레시피를 보고 빵을 만든다고 했을 때, 만드는 과정에 따라 필요한 재료들이 있습니다. 예를 들어 강력분, 중력분, 우유 등을 넣고 반죽한다고 했을 때, 반죽하는 행동에 필요한 재료는 강력분, 중력분, 우유가 되고, 반죽하는 행동의 결과로 반죽이 생깁니다. 또는 캡슐 커피 머신에 캡슐 커피를 넣으면 커피가 나오는 것을 생각해 볼 수 있습니다. 이처럼 자바의 메서드는 프로그래머가 준 값을 재료로 어떤 행동을 하고, 그 결과로 생긴 것을 돌려주는 처리상자입니다.

그림 3-5 현실세계의 행동(처리동작)과 자바의 메서드



재료가 되는 값을 '파라미터(인수)'라 하고, 이를 사용하여 처리한 결과 값을 '리턴 값(반환값)'이라고 부릅니다. 그리고 처리상자인 메서드가 그 내부에서 어떤 처리

를 할지 적는 것을 ‘메서드를 정의한다’라고 하는데, 메서드를 정의하는 방법은 다음과 같습니다.

[메서드를 정의하는 방법]

- (1) 클래스를 선언함
→ class Calc{
- (2) 리턴값의 데이터 타입을 적음
→ class Calc{
 int
- *void는 리턴값이 없다는 것을 의미
 }
- (3) 메서드명을 적음
→ class Calc{
 int plus
- (4) 메서드명 옆 () 안에 필요한 파라미터를 적음
→ class Calc{
 int plus(int a, int b)
- **파라미터 역시 변수이므로 데이터 타입과 변수명을 함께 적음
 }
- (5) { } 안에 처리할 동작을 적음
→ class Calc{
 int plus(int a, int b){
 int k = a+b;
- (5)) 메서드를 호출한 곳에 보낼 리턴값을 적어줌
→ class Calc{
 int plus(int a, int b){
 int k = a+b;
 return k;
- }

class Calc{

객체를 정의/생성하는 툴인 Calc라는 클래스를 만든다

int plus(int a, int b){

정수 값을 리턴값으로 담는 plus라는 메서드를 선언하고, 정수 값을 담는 a와 b라는 파라미터 변수를 인자로 받는다

int k = a + b;

정수 값을 담는 변수 a와 변수 b를 더한 값을 저장한다

return k;

이 메서드를 호출한 곳에 변수 k값을 리턴한다

}

NOTE void

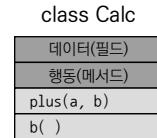
'int plus(int a, int b)'라는 메서드 처럼 해당 메서드의 처리 결과 값의 데이터 타입을 메서드명 앞에 적어줍니다. 보통 int, String 등의 데이터 타입 값이 오지만, 때로는 처리 결과로 아무것도 리턴할 게 없을 때도 있습니다. 그럴 땐 '리턴값이 없다'는 의미로 void를 적어줍니다.

정의한 메서드를 호출하는 방법은 두 가지로 나누어집니다. 우선, 해당 메서드를 정의한 클래스 내에서 그 메서드를 호출할 때는 메서드명만을 사용하여 호출합니다. 그리고 메서드를 정의한 클래스 외부에서 그 메서드를 호출할 때는 메서드가 정의된 클래스의 객체를 만들고 '객체.메서드명()'을 작성하여 호출합니다.

[메서드를 정의한 클래스 내에서 호출할 때]

```
class Calc{
    int plus(int a, int b){
        int k = a + b;
        return k;
    }
    void b(){
        System.out.println(plus(1,3));
    }
}
```

메서드 이름으로 바로 호출

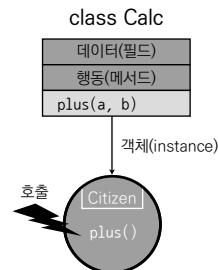


[메서드를 정의한 클래스 밖에서 호출할 때]

```
class Calc{
    int plus(int a, int b){
        int k = a + b;
        return k;
    }
}

class CalcTest{
    public static void main(String[] args){
        Calc calc = new Calc();
        System.out.println(calc.plus(1,3));
    }
}
```

메서드를 정의한 클래스의 객체를 생성한 후
'객체.메서드명()'으로 호출

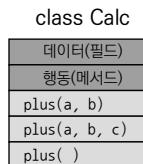


한 클래스 내에서 같은 이름의 메서드를 파라미터와 처리동작, 리턴값을 다르게 하여 여러 번 정의할 수도 있습니다. 이를 ‘오버로딩’이라고 합니다. 오버로딩으로 여러 번 정의된 메서드는 호출 시, 파라미터의 수와 데이터 타입이 일치하는 메서드가 호출됩니다.

오버로딩: 한 클래스 내에서 같은 이름의 메서드를 파라미터와 처리동작, 리턴값을 다르게 하여 여러 번 정의하는 것

[오버로딩]

```
class Calc{  
    int plus(int a, int b){  
        int k = a + b;  
        return k;  
    }  
    double plus(double a, double b, double c){  
        return a+b+c;  
    }  
    void plus(){  
        System.out.println("더하기!");  
    }  
}
```



[오버로딩]에서 정의한 Calc 클래스를 이용하여 오버로딩한 메서드를 호출하는 프로그램을 작성하면 다음과 같습니다.

[코드 3-3] 오버로딩한 메서드 호출 프로그램

class CalcTest{

객체를 정의/생성하는 툴인 CalcTest라는 클래스를 만든다

```
public static void main(String[] args){  
    Calc calc = new Calc();
```

Calc라는 클래스의 객체를 담는 calc라는 변수를 선언하고 Calc 클래스의 객체를 새로 생성하여 저장한다

```
calc.plus();
```

calc 변수 안의 plus() 메서드를 호출한다

```
System.out.println(calc.plus(1,3));
```

calc 변수 안의 plus 메서드를 1과 3 파라미터 값으로 호출하여 시스템 콘솔에 출력하고 한 줄 띄운다

```
System.out.println(calc.plus(1.5,2.5,3.5));
```

calc 변수 안의 plus 메서드를 1.5, 2.5, 3.5 파라미터 값으로 호출하여 시스템 콘솔에 출력하고 한 줄 띄운다

```
}
```

실행 결과

더하기!

4

7.5

3.5 생성자

[코드 3-2]에서 객체를 생성할 때 일일이 데이터(필드)에 값을 저장한 부분이 있습니다. 이 작업을 2번, 3번 반복할 경우 많이 복잡하겠죠? 이러한 복잡한 작업 대신 객체 생성과 동시에 데이터 값은 설정할 수 있습니다.

[객체의 데이터(필드) 값을 설정하는 2가지 방법]

```
class PersonExample {
    public static void main(String args[]){
        Person PersonObj1;
        PersonObj1 = new Person();
        PersonObj1.name = "김삿갓";
        PersonObj1.age = 50;
        PersonObj1.weight = 80;

        System.out.println(PersonObj1.name);
        System.out.println(PersonObj1.eat());
    }
}
```

간편화
객체 생성과 동시에 데이터 값을 설정하니
코드가 5줄에서 1줄로 줄어듭니다!

```
class PersonExample {
    public static void main(String args[]){
        Person PersonObj1 = new Person("김삿갓",50,80);
    }
}
```

1. 객체 생성 후 일일이 데이터(필드)에 값을 저장

2. 객체 생성과 동시에 데이터(필드) 값 설정

```
        System.out.println(PersonObj1.name);
        System.out.println(PersonObj1.eat());
    }
}
```

}

앞에서 볼 수 있듯이, 객체 생성과 동시에 데이터(필드) 값을 설정하면 코드도 짧아지고 매우 간편합니다. 그런데 이렇게 객체 생성과 동시에 데이터 값을 설정하려면 사전에 할 일이 있습니다. 바로 객체를 생성하는 툴인 클래스를 작성할 때 그 안에 생성자를 만들어주는 작업입니다.

생성자(Constructor)는 객체가 생성과 동시에 객체 안의 데이터 값을 설정(초기화)하는 코드입니다. 생성자의 이름은 반드시 클래스의 이름과 같아야 하고, 리턴 타입이 없습니다. 생성자를 작성하는 방법은 다음과 같습니다.

[생성자를 정의하는 방법]

- (1) 생성자는 클래스명과 같음 -> Person
(2) 객체 생성 시 초기화 값을 파라미터로 받음 -> Person(String name, int age, int weight)
(3) {} 안의 객체 내 데이터에 파라미터를 저장 -> Person(String name, int age, int weight){
* this는 객체 자신을 가리키는 키워드 this.name = name;
 this.age = age;
 this.weight = weight;
 }
}

class Person {

 String name;
 int age;
 int weight;

 Person(String name, int age, int weight){

 Person 클래스의 객체 필드를 초기 설정하는 생성자를 만들고,

 문자열 값을 담는 name 파라미터 변수와 정수 값을 담는 age, weight 파라미터 변수를 인자로 받는다

this.name = name;

 이 클래스로 생성된 객체 안의 name 변수에 name 파라미터 변수를 저장한다

this.age = age;

 이 클래스로 생성된 객체 안의 age 변수에 age 파라미터 변수를 저장한다

```
this.weight = weight;  
이 클래스로 생성된 객체 안의 weight 변수에 weight 파라미터 변수를 저장한다  
}  
String eat(){  
    return "eat";  
}  
String sleep(){  
    return "sleep";  
}  
}
```

자바 프로그래머가 직접 클래스 안에 생성자를 기술하지 않았을 경우에 'Person personObj1 = new Person();'이라고 하면 자바는 다음과 같은 '디폴트 생성자'를 자동으로 생성하여 실행합니다.

[디폴트 생성자]

```
Person(){}
```

하지만 직접 작성한 생성자가 있는 경우에 자바는 디폴트 생성자를 만들지 않으므로 Person personObj1 = new Person();이라고 객체를 생성하면 오류가 발생합니다.

생성자는 메서드처럼 오버로딩이 가능합니다. 즉, 같은 이름의 생성자를 파라미터와 처리동작을 다르게 하여 여러 번 정의할 수도 있습니다.

[코드 3-4] 생성자 오버로딩

```
class Person {  
    String name;  
    int age;  
    int weight;
```

```

Person(String name, int age, int weight){
    this.name = name;
    this.age = age;
    this.weight = weight;
}

Person(String name, int age){
    this.name = name;
    this.age = age;
}

Person( ){
}

String eat(){
    return "eat";
}

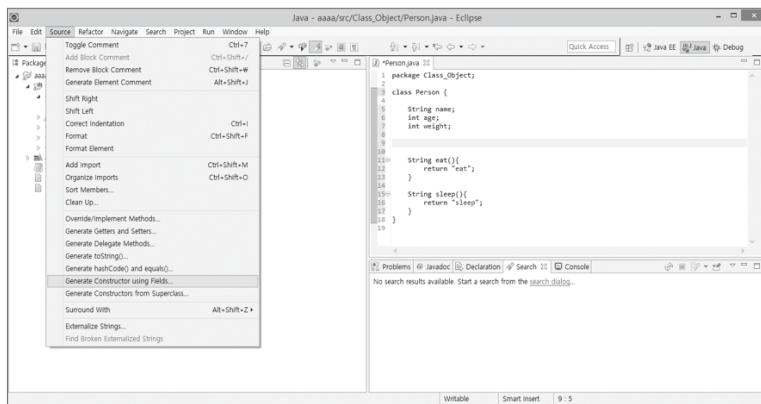
String sleep(){
    return "sleep";
}

```

NOTE Eclipse – 생성자 자동으로 만들기

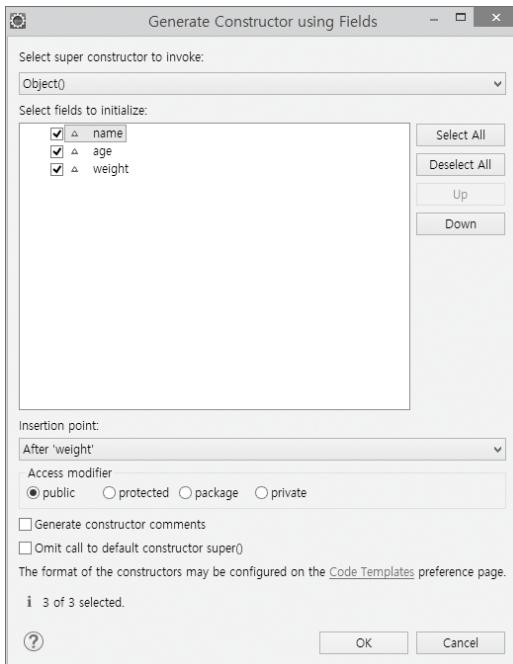
이클립스에서는 생성자를 자동으로 만드는 방법을 제공합니다. 이를 알아보기 위해 전체 코드에서 생성자 코드를 만들기 원하는 위치에 마우스 커서를 놓은 후 이클립스 상단의 [Source → Generate Constructor using Fields···]를 클릭합니다.

그림 3-6 이클립스에서 생성자 자동으로 만들기 1



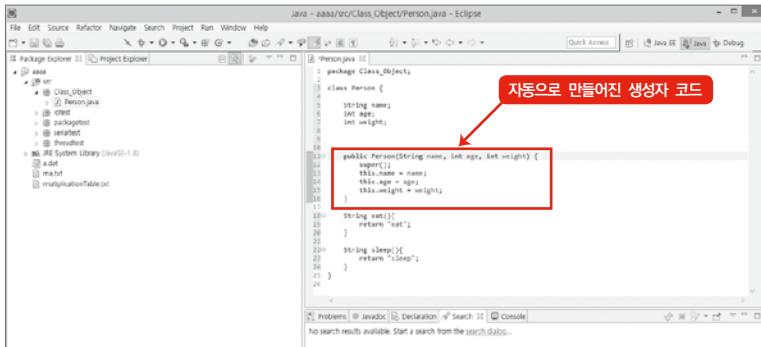
그리면 [그림 3-7]과 같이 'Generate Constructor using Fields' 창이 뜹니다. 모든 필드를 다 선택한 후 [OK] 버튼을 클릭하면 모든 필드를 객체 생성과 동시에 설정할 수 있는 생성자가 생기고, 모든 필드를 하나도 체크하지 않고 [OK] 버튼을 누르면 디폴트 생성자가 생깁니다.

그림 3-7 이클립스에서 생성자 자동으로 만들기 2



모든 필드를 선택하고 [OK] 버튼을 누르면 [그림 3-8]과 같은 코드가 이클립스에 자동으로 만들어집니다.

그림 3-8 이클립스에서 생성자 자동으로 만들기 3



[코드 3-4]에서 작성한 Person 클래스로 객체를 생성하면 다음과 같습니다.

[코드 3-5] 여려 생성자로 객체 초기화하기

```
class ConstructorTest {
```

객체를 정의/생성하는 틀인 ConstructorTest라는 클래스를 만든다

```
public static void main(String args[]){
```

```
    Person p1 = new Person();
```

Person이라는 클래스의 객체를 담는 p1이라는 변수를 선언하고 Person이라는 클래스의 객체를 새로 생성하여 저장한다

```
    p1.name = "hin";
```

p1 변수 안의 name 변수에 "hin" 파라미터 변수를 저장한다

```
    Person p2 = new Person("John", 19);
```

Person 클래스의 객체를 담는 p2라는 변수를 선언하고

Person 클래스의 "John"과 19로 초기화한 객체를 새로 생성하여 저장한다

```
    Person p3 = new Person("Miranda", 24, 50);
```

Person 클래스의 객체를 담는 p3라는 변수를 선언하고

Person 클래스의 "Miranda", 24, 50으로 초기화한 객체를 새로 생성하여 저장한다

```
    System.out.println(p1.name);
```

p1 변수 안의 name 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다

```
    System.out.println(p2.name);
```

p2 변수 안의 name 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다

```
    System.out.println(p3.name);
```

p3 변수 안의 name 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다

```
}
```

```
}
```

실행 결과

hin

John

Miranda

연습 3-4

[연습 3-2]에서 작성한 Account 클래스에 계좌번호, 예금주, 잔액을 선언과 동시에 초기화할 수 있는 생성자와 디폴트 생성자를 만드세요.

연습 3-5

다음과 같은 실행 결과가 나오도록 Account 클래스를 사용하여 5개의 계좌 객체를 생성하고, 그 정 보를 출력하는 코드를 작성하세요.

TIP 배열과 반복문을 사용하면 코드가 짧아집니다.

실행 결과

```
===== 당일 개설 계좌 명단 =====
계좌번호: 1002-03-0001000, 예금주: 홍길동, 잔액: 100000원
계좌번호: 1002-06-0701421, 예금주: 박채윤, 잔액: 20000000원
계좌번호: 1002-01-9891123, 예금주: 이영희, 잔액: 360433원
계좌번호: 1002-05-0801000, 예금주: 강진철, 잔액: 50000원
계좌번호: 1002-02-0150802, 예금주: 임성수, 잔액: 1000000000원
=====
```

클래스의 상속

4장에서는 어떤 클래스를 상속해 또 다른 클래스를 만드는 방법에 대해 배웁니다. 우리가 현실 세계에서 ‘유산을 상속받다, 유전자를 상속받다’라고 말할 때의 ‘상속’이란 ‘물려받다’라는 의미입니다. 클래스도 어떤 클래스의 특성을 물려받아 새로운 클래스를 만들 수 있는데, 이를 ‘클래스의 상속’이라고 합니다.

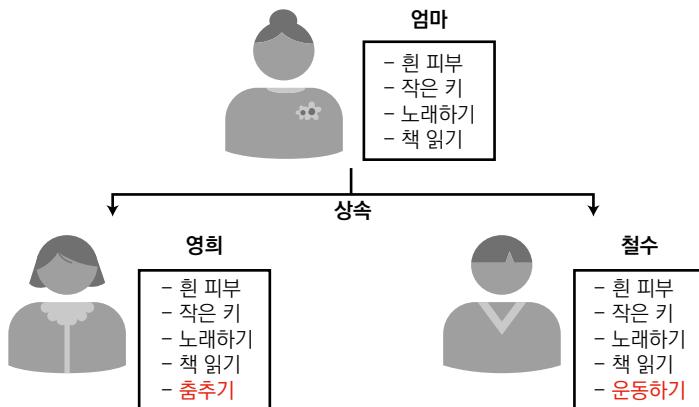
4.1장에서는 상속의 정의와 선언하는 법에 대해 공부합니다. 4.2장에서는 상속받은 클래스에서 상속해 준 클래스의 메서드를 재정의하는 방법에 대해 배웁니다. 4.3장에서는 클래스를 상속할 때 어떤 데이터 또는 메서드를 변경하고 싶지 않거나 모두와 공유하고 싶은 경우 등 이런 사항을 표시할 수 있는 수식자에 대해 배웁니다. 4.4장에서는 여러 클래스를 상속해 만들 것이라 예상될 경우, 이들의 공통 점만 뽑아 추상화한 추상 클래스를 다룹니다. 4.5장에서는 클래스와 유사하지만 조금 다른 개념인 인터페이스를 공부하겠습니다.

4.1 상속의 정의

어떤 집안이 있습니다. 이 집안 사람들은 전통적으로 피부가 하얗고, 키가 작으며, 책 읽기를 좋아합니다. 그리고 노래를 잘 부르는 유전자를 가지고 있습니다. 이 집안에서 태어난 영희와 철수는 자신의 어머니로부터 이 집안 유전자를 고스란히 상속받아 피부가 하얗고, 키가 작고, 책 읽기를 좋아하며, 노래를 잘 부릅니다. 그렇지만 영희는 몸치인 어머니와 다르게 춤을 잘 추고, 철수는 운동을 싫어하는 어머

니와 다르게 운동을 좋아합니다. 이처럼 상속을 받으면 상속받은 것을 기초로 되 그에 더해진 특징을 갖게 됩니다.

그림 4-1 유전자 상속



자바에서도 ‘상속’은 ‘물려받다’는 의미로 사용하고, 어떤 클래스 A가 가진 것을 다른 클래스 B에게 상속해 줄 수 있습니다. 즉, 클래스 A로부터 상속을 받은 클래스 B는 클래스 A가 가진 모든 필드(데이터)와 메서드(처리동작)를 기본으로 가지고 있고, 자신 특유의 필드와 메서드 역시 갖고 있습니다. 이때, 상속해 주는 클래스 A를 슈퍼 클래스^{Super Class}라고 하고, 상속받는 클래스 B를 서브 클래스^{Sub Class}라고 합니다. 슈퍼 클래스로 서브 클래스를 여러 개 만들 수 있지만, 서브 클래스가 상속받을 수 있는 슈퍼 클래스는 하나뿐입니다(단일 상속). 부모는 여러 자식을 낳아 여러 명에게 유전자를 상속해 줄 수 있지만, 자식은 여러 부모가 아닌 한 부모로부터만 유전자를 물려받을 수 있는 것처럼 말입니다.

클래스를 상속하려면 ‘확장하다’라는 의미의 ‘extends’라는 단어를 사용합니다. 즉, 슈퍼 클래스를 확장해 서브 클래스를 만든다고 생각하면 됩니다. [그림 4-2]는 클래스 Person을 상속한 클래스 Student와 클래스 Worker를 보여줍니다.

그림 4-2 클래스 상속



[그림 4-2]를 자바로 프로그래밍해 보면 다음과 같습니다.

[코드 4-1] 상속받은 클래스들의 선언

class Student extends Person {

Person이라는 클래스를 확장(상속)하여 객체를 정의/생성하는 데 사용하는 Student라는 클래스를 만든다

int stuNo;

정수형 값을 담는 stuNo라는 변수를 선언한다

String study();

문자열을 리턴값으로 받는 study라는 메서드를 선언한다

return "study";

이 메서드를 호출한 곳에 "study"를 리턴한다

}

}

class Worker extends Person {

Person이라는 클래스를 확장(상속)하여 객체를 정의/생성하는 데 사용하는 Worker라는 클래스를 만든다

참고

```
class Person {  
    String name;  
    int age;  
    int weight;  
  
    String eat(){  
        return "eat";  
    }  
    String sleep(){  
        return "sleep";  
    }  
}
```

String position;

문자열 값을 담는 **position**이라는 변수를 선언한다

String goToWork(){

문자열을 리턴값으로 받는 **goToWork**라는 메서드를 선언한다

return "go to work";

이 메서드를 호출한 곳에 “**go to work**”를 리턴한다

}

String attendMeeting(){

문자열을 리턴값으로 받는 **attendMeeting**라는 메서드를 선언한다

return "attend a meeting";

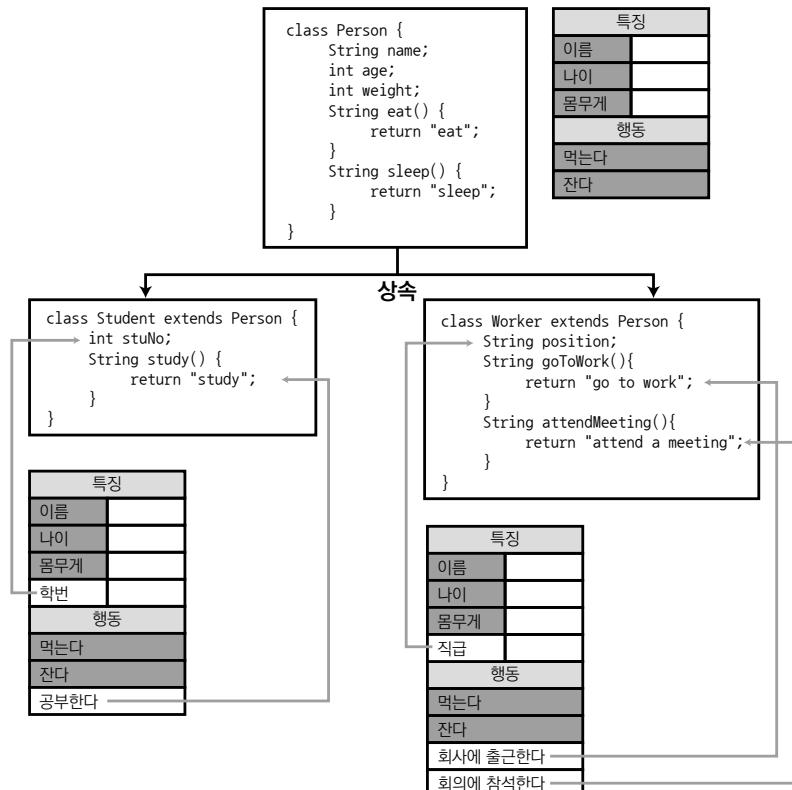
이 메서드를 호출한 곳에 “**attend a meeting**”을 리턴한다

}

}

클래스 Person을 확장(상속)하여 만들어진 클래스 Student와 클래스 Worker는
클래스 Person의 필드와 메서드 내용은 이미 가지고 있습니다. 따라서 클래스 선
언 시 그들만이 새롭게 가진 필드와 메서드만 적어 주면 됩니다.

그림 4-3 상속받은 클래스들의 계층구조



4.2 오버라이딩

영어로 ‘override(오버라이드)’는 ‘~을 타고 넘다, ~에 우선하다’라는 의미의 단어입니다. 자바에서의 오버라이딩 Overriding은 슈퍼 클래스의 메서드를 서브 클래스에서 다시 정의하는 것을 말합니다. 즉, 서브 클래스에서 다시 정의된 메서드가 슈퍼 클래스의 메서드보다 우선한다고 생각하면 됩니다.

[코드 4-2] 오버라이딩

```
class Subject{
```

객체를 정의/생성하는 툴인 `Subject`라는 클래스를 만든다

String name;

문자열 값을 담는 name이라는 변수를 선언한다

void printInfo(){

리턴값이 없는 printInfo라는 메서드를 선언한다

```
System.out.println("과목명:" + name);
```

“과목명:” 다음에 name 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다

```
}
```

```
}
```

오버라이딩

class MajorSubject extends Subject {

Subject라는 클래스를 확장(상속)하여 객체를 정의/생성하는 데 MajorSubject라는 클래스를 만든다

String targetGrader;

문자열 값을 담는 targetGrader라는 변수를 선언한다

void printInfo(){

리턴값이 없는 printInfo라는 메서드를 선언한다

```
super.printInfo();
```

슈퍼 클래스 안의 printInfo 메서드를 호출한다

```
System.out.println("대상학년:" + targetGrader);
```

“대상학년:” 다음에 targetGrader 변수의 값을 시스템 콘솔에 출력하고 한 줄 띄운다

```
}
```

```
}
```

NOTE super

서브 클래스 내에서 super는 슈퍼 클래스를 의미합니다. 즉, [코드 4-2]를 보면 클래스 MajorSubject에서 super.printInfo();라고 적힌 부분은 MajorSubject가 상속받은 Subject 클래스의 printInfo 메서드를 실행하라는 의미입니다. Subject 클래스의 name 변수를 MajorSubject 클래스 내부에서 참조하고 싶다면 super.name이라고 표현하면 됩니다.

슈퍼 클래스의 메서드를 서브 클래스에서 오버라이딩할 때는 해당 메서드의 이름과 인자, 리턴값은 그대로 선언하고 {} 안의 처리 명령만 변경하여 작성하면 됩니다.

[코드 4-2]에서 정의한 클래스들을 사용하여 객체를 만드는 프로그램을 작성하면 다음과 같습니다.

[코드 4-3] 오버라이딩한 메서드 실행

```
class SubjectExample {  
    객체를 정의/생성하는 데 인 SubjectExample이라는 클래스를 만든다  
    public static void main(String args[]){  
        MajorSubject sub = new MajorSubject();  
        MajorSubject라는 클래스의 객체를 담는 sub 변수를 선언하고  
        MajorSubject 클래스의 객체를 새로 생성하여 저장한다  
        sub.name = "자바프로그래밍";  
        sub 변수 안의 name 변수에 "자바 프로그래밍"을 저장한다  
        sub.targetGrader = "2학년";  
        sub 변수 안의 targetGrader 변수에 "2학년"을 저장한다  
        sub.printInfo();  
        sub 변수 안의 printInfo 메서드를 호출한다  
    }  
}
```

실행 결과

과목명: 자바프로그래밍

대상학년: 2학년

NOTE 오버라이딩 vs 오버로딩

오버라이딩 슈퍼 클래스를 상속받은 서브 클래스에서 슈퍼 클래스의 메서드를 다시 정의하는 것

오버로딩 하나의 클래스 내에서 동일한 이름의 메서드를 파라미터, 리턴 타입 등을 다르게 하여 여러 개 정의하는 것

연습 4-1

김진영 씨는 회사에 입사 후 한도가 3,000만 원까지 가능한 마이너스 통장을 은행에서 개설하였습니다. 다음 코드를 실행했을 때 코드 아래의 실행 결과가 나오도록 [연습 3-2]에서 작성한 Account 클래스를 상속받아 MinusAccount 클래스를 작성해 보세요.

```
public class MinusAccountExample {  
    public static void main(String[] args) {  
        MinusAccount maccount = new MinusAccount();  
  
        maccount.account_num = "1002-03-0001050";  
        maccount.depositor = "김진형";  
        maccount.date = "2013-01-04";  
        maccount.balance = 100000;  
  
        maccount.withdraw(40000);  
        maccount.withdraw(40000);  
        maccount.withdraw(40000);  
        maccount.withdraw(40000);  
    }  
}
```

실행 결과

```
[계좌번호:1002-03-0001050] -40000, 잔액:60000, 남은 한도:30000  
[계좌번호:1002-03-0001050] -40000, 잔액:20000, 남은 한도:30000  
[계좌번호:1002-03-0001050] -40000, 잔액:0, 남은 한도:10000  
[계좌번호:1002-03-0001050] 출금 실패! 마이너스 통장의 한도가 부족합니다. 남은 한도:  
10000
```

4.3 여러 가지 수식자

자바에는 필드와 메서드, 클래스에 특정한 성질을 부여하기 위한 여러 가지 수식자가 있습니다. final, static, abstract가 그 예인데, abstract는 4.4 추상 클래스에서 더 자세하게 다루기로 하고 여기서는 final과 static에 대해 공부해 보겠습니다.

4.3.1 final

영어에서 ‘final’은 ‘최종의, 마지막의’라는 의미로 ‘final outcome’는 ‘최종 결과’, ‘final round’는 ‘결승전’이라는 뜻이 됩니다. 최종 결과는 바꿀 수 없고, 결승전에서도 승부가 나면 돌이킬 수 없습니다. 이처럼 final이란 단어는 ‘바꿀 수 없는, 변경할 수 없는’이란 의미도 그 안에 내포하고 있습니다.

자바에서도 이와 같은 의미로 사용하여 변수나 클래스 앞에 final을 붙일 수 있습니다. 변수 앞에 final이 붙으면 선언문이나 생성자 안에서 반드시 처음 값을 지정해야 하고, 그 변수의 값을 변경할 수 없습니다. 즉, final 변수에는 새로운 값을 저장할 수 없습니다. 이때 처음 값을 설정하는 것을 ‘초기화’라고 합니다.

[선언문에서 초기화]

final String name = "Hee Eun";

변경할 수 없는, 문자열 값을 담는 name이라는 변수를 선언하고 “Hee Eun”을 저장한다

그림 4-4 final 변수



[생성자 안에서 초기화]

class Subject{

객체를 정의/생성하는 틀인 Subject라는 클래스를 만든다

final String name;

변경할 수 없는, 문자열 값을 담는 name이라는 변수를 선언한다

Subject(String name) {

Subject 클래스의 객체 필드를 초기 설정하는 생성자를 만들고, 문자열 값을 담는 name 파라미터 변수를 선언한다

this.name = name;

이 클래스로 생성된 객체 안의 name 변수에 name 파라미터 변수를 저장한다

}

}

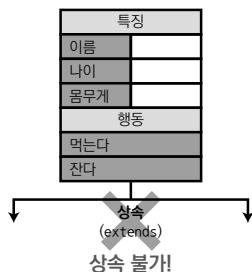
또한, 클래스 앞에 final이 붙으면 해당 클래스는 상속할 수 없습니다. 즉, final 클래스를 확장하여 다른 클래스를 만들 수 없습니다.

[final 클래스]

final class Person{

변경할 수 없는, 객체를 정의/생성하는 틀인 Person이라는 클래스를 만든다

그림 4-5 final 클래스



4.3.2 static

영어에서 ‘static’은 ‘정적인, 정지된’이라는 의미로 ‘static place’는 ‘정지된 장소’라는 뜻이고, 정지된 장소란 한 곳에 있다는 말입니다. 자바에서 ‘static’은 정지된 메모리’, 즉 하나의 메모리 주소에 계속 저장한다는 의미로 사용합니다.

변수나 메서드 앞에 static을 붙일 수 있는데, 변수 앞에 static이 붙으면 그 변수는 같은 메모리 주소에 값을 계속 저장합니다. 해당 클래스로 생성된 객체들의

`static` 변수는 모두 같은 메모리 주소에 값을 저장하므로 결과적으로는 객체들끼리 그 값을 공유할 수 있습니다.

[static 변수]

```
class Book{
```

```
    String name;
```

```
    static int price;
```

같은 클래스에서 생성된 객체들과 값을 공유하고, 정수 값을 담는 `price`라는 변수를 선언한다

```
    ...
```

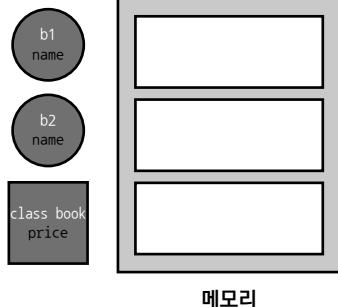
```
}
```

```
...
```

```
Book b1 = new Book();
```

```
Book b2 = new Book();
```

```
...
```



[코드 4-4] static 변수를 이용한 프로그램

```
class Book{
```

객체를 정의/생성하는 툴인 `Book`이라는 클래스를 만든다

```
    String name;
```

문자열 값을 담는 `name`이라는 변수를 선언한다

```
    static int price;
```

같은 클래스에서 생성된 객체들과 값을 공유하고, 정수 값을 담는 `price`라는 변수를 선언한다

```
    void printInfo(){
```

리턴값이 없는 `printInfo`라는 메서드를 선언한다

```
        System.out.println(name + ":" + price);
```

`name` 변수의 값 다음에 `:` 다음에 `price` 변수의 값을 시스템 콘솔에 출력하고 한 줄 띠운다

```
    }
```

```
}
```

class BookTest{

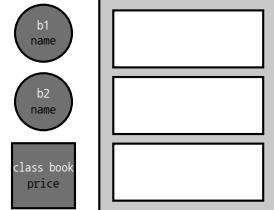
객체를 정의/생성하는 데 **BookTest**라는 클래스를 만든다

```
public static void main(String args[]){
```

Book b1 = new Book();

Book 클래스의 객체를 담는 b1 변수에

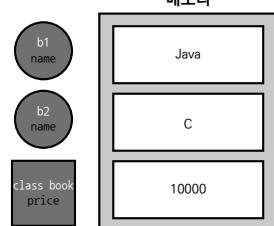
Book 클래스의 객체를 새로 생성하여 저장한다



Book b2 = new Book();

Book 클래스의 객체를 담는 b2 변수에

Book 클래스의 객체를 새로 생성하여 저장한다



b1.name = "Java";

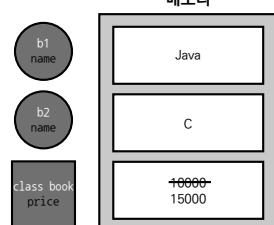
b1 변수 안의 name 변수에 "Java"를 저장한다

b2.name = "C";

b2 변수 안의 name 변수에 "C"를 저장한다

b1.price = 10000;

b1 변수 안의 price 변수에 10000을 저장한다



b2.price = 15000;

b2 변수 안의 price 변수에 15000을 저장한다

b1.printInfo();

b1 변수 안의 printInfo() 메서드를 호출한다

b2.printInfo();

b2 변수 안의 printInfo() 메서드를 호출한다

```
}
```

실행 결과

Java : 15000

C : 15000

static은 메서드 앞에도 붙일 수 있습니다. static이 붙은 메서드는 클래스 메서드로 불리며, 한 메모리 주소에 메서드 코드가 저장됩니다. 따라서 클래스와 그 클래스로 생성한 객체 모두가 메서드를 공유하고, 객체를 생성하지 않아도 클래스

이름으로 바로 참조할 수 있습니다. 또한, static 메서드는 오버라이딩이 불가능합니다.

[코드 4-5] static 메서드를 이용한 프로그램

```
class Hello{
```

객체를 정의/생성하는 둘인 Hello라는 클래스를 만든다

```
    static void greeting(){
```

같은 클래스에서 생성된 객체들과 값을 공유하고, 리턴값이 없는 greeting이라는 메서드를 선언한다

```
        System.out.println("Hello, everyone!");
```

"Hello, everyone!"을 시스템 콘솔에 출력하고 한 줄 띄운다

```
}
```

```
}
```

```
class HelloTest{
```

객체를 정의/생성하는 둘인 HelloTest라는 클래스를 만든다

```
    public static void main(String args[]){
```

```
        Hello h1 = new Hello();
```

Hello 클래스의 객체를 담는 h1 변수에 Hello 클래스의 객체를 새로 생성하여 저장한다

```
        Hello h2 = new Hello();
```

Hello 클래스의 객체를 담는 h2 변수에 Hello 클래스의 객체를 새로 생성하여 저장한다

```
        h1.greeting();
```

h1 변수 안의 greeting() 메서드를 호출한다

```
        h2.greeting();
```

h2 변수 안의 greeting() 메서드를 호출한다

```
        Hello.greeting();
```

Hello 클래스 안의 greeting() 메서드를 호출한다

```
}
```

```
}
```

실행 결과

Hello, everyone!

Hello, everyone!

Hello, everyone!

4.4 추상 클래스

'추상Abstract'이란 '여러 가지 사물이나 개념에서 공통되는 특성이나 속성 따위를 추출하여 파악하는 작용'입니다. 예를 들면, 지구상에 존재하는 나무, 새, 고양이, 물고기 등 모든 생물을 크게 2가지로 구분하면, 동물과 식물로 나눌 수 있습니다. 동물은 이동할 수 있고 다른 생물로부터 양분을 얻어 살아간다는 특성을 공통으로 가지고 있으며, 식물은 이동할 수 없고 광합성을 통해 스스로 양분을 생산한다는 특성을 공통으로 가지고 있습니다.

그림 4-6 객체와 추상



이러한 추상의 개념을 적용하여 자바에서도 추상 클래스를 만들 수 있습니다. '추상 클래스'란 다른 클래스들의 공통이 되는 변수나 메서드의 이름과 형태만 기술해 놓았을 뿐 구체적인 내용이 없는 클래스입니다. 메서드에서 무엇을 하는가는 추상 클래스를 상속받은 서브 클래스에서 오버라이딩을 통해 정의합니다. 따라서

추상클래스는 다른 클래스의 템플릿으로 사용됩니다. 예를 들어, 개와 고양이 클래스를 만든다고 할 때 이 둘은 동물이므로 ‘소리 내 운다’는 공통점이 있습니다. 이럴 때 동물에 관한 추상 클래스를 만들어 공통된 행동인 ‘운다’를 메서드로 적어 두면, 개 클래스와 고양이 클래스는 추상 클래스를 상속하여 ‘운다’ 메서드에 개와 고양이가 각각 어떻게 우는지만 다시 정의하면 됩니다.

이처럼 추상 클래스는 코드의 확장성을 높여서 유연한 프로그램을 만들 수 있는 장점이 있고, 큰 프로그램일수록 추상 클래스가 중요하게 사용됩니다. 추상 클래스는 `abstract`라는 수식자를 사용하여 다음과 같은 방법으로 선언합니다.

[코드 4-6] 추상 클래스의 선언

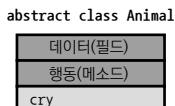
abstract class Animal{

구체적인 내용 없이 공통점만 뽑아낸, 객체를 정의/생성하는 틀인 `Animal`이라는 클래스를 만든다

abstract void cry();

구체적인 내용 없이 공통점만 뽑아낸, 리턴값이 없는 `cry`라는 메서드를 선언한다

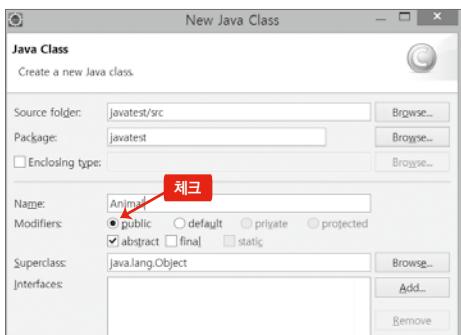
}



NOTE Eclipse – 추상 클래스 자동으로 만들기

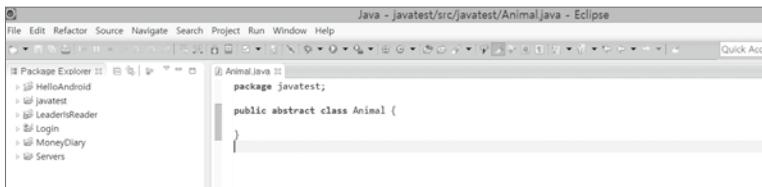
Eclipse에서는 클래스 생성 창에서 [Modifiers → abstract]를 체크하면 자동으로 추상 클래스 선언 코드를 생성할 수 있습니다. 클래스 생성 창은 Project Explorer나 Package Explorer에서 마우스 오른쪽 버튼을 클릭한 후 [New → Class]를 누르면 띄울 수 있습니다.

그림 4-7 이클립스의 클래스 생성 창에서 abstract 클래스 만드는 법



이클립스가 자동으로 생성한 추상 클래스는 [그림 4-8]과 같습니다. `public abstract class Animal`에서 `public`은 접근 제어 수식어로, 어떤 제한도 없이 다른 클래스에서 해당 클래스를 이용할 수 있다는 의미입니다. 접근 제어 수식어에 관한 내용은 [5장](#)에서 자세히 다루겠습니다.

그림 4-8 이클립스에서 자동으로 만든 abstract 클래스 선언 코드



구체적인 내용이 없기 때문에 추상 클래스로는 객체를 만들 수 없습니다. 마치 동물을 그리라고 할 때 어떤 동물을 그려야 할지 막막한 것처럼 말입니다. 추상 클래스를 이용하는 방법은 다음과 같습니다.

[코드 4-7] 추상 클래스의 이용

class Dog extends Animal{

Animal이라는 클래스를 확장(상속)하여

객체를 정의/생성하는 데인 Dog라는 클래스를 만든다

void cry(){

리턴값이 없는 cry라는 메서드를 선언한다

System.out.println("멍멍");

"멍멍"을 시스템 콘솔에 출력하고 한 줄 띄운다

```
}
```

class Cat extends Animal{

Animal이라는 클래스를 확장(상속)하여

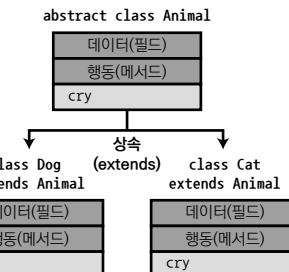
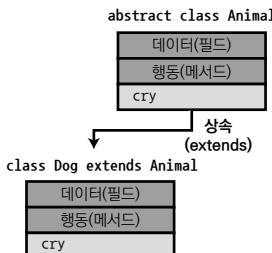
객체를 정의/생성하는 데인 Cat이라는 클래스를 만든다

void cry(){

리턴값이 없는 cry라는 메서드를 선언한다

System.out.println("야옹");

"야옹"을 시스템 콘솔에 출력하고 한 줄 띄운다



```
}
```

추상 클래스를 상속받은 서브 클래스에 관한 코드 예제를 살펴보겠습니다.

[코드 4-8] 추상 클래스를 상속받은 서브 클래스

```
class AnimalTest{
```

객체를 정의/생성하는 룰인 `AnimalTest`라는 클래스를 만든다

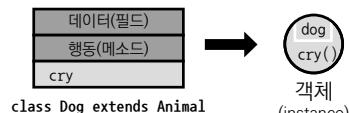
```
public static void main(String args[]){
```

모든 패키지와 클래스에서 접근 가능하고, 같은 클래스에서 생성된 객체들과 값을 공유하며, 리턴값이 없는 `main()`이라는 함수를 선언하고, 문자열 값을 담는 `args[]`라는 배열을 인자로 받는다

```
Dog dog = new Dog();
```

`Dog`라는 클래스의 객체를 담는 `dog`라는 변수에

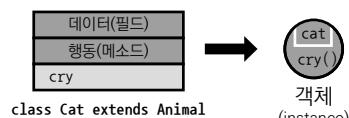
`Dog`라는 클래스의 객체를 새로 생성하여 저장한다



```
Cat cat = new Cat();
```

`Cat`이라는 클래스의 객체를 담는 `cat`이라는 변수에

`Cat`이라는 클래스의 객체를 새로 생성하여 저장한다



```
dog.cry();
```

`dog` 변수 안의 `cry()` 메서드를 호출한다



```
cat.cry();
```

`cat` 변수 안의 `cry()` 메서드를 호출한다



```
}
```

실행 결과

멍멍

야옹

4.5 인터페이스

영어로 ‘interface(인터페이스)’는 ‘결합부’라는 뜻으로, 서로 다른 두 시스템이나

장치, 사람 등을 이어주는 부분을 말합니다. 예를 들어, 어떤 사람이 컴퓨터를 켜려고 전원 버튼을 눌렀다면 그 버튼이 어떤 사람과 컴퓨터 전원 시스템 사이의 인터페이스가 됩니다.

자바에서도 인터페이스는 클래스와 외부 세계(ex. 개발자, 다른 클래스)를 이어주는 역할을 합니다. 인터페이스는 내부에 추상 메서드를 가지고 있으며, 인터페이스를 받은 클래스에서는 해당 메서드를 오버라이딩으로 다시 정의하여 사용합니다. 인터페이스를 이용하면 메서드의 선언과 구현을 분리할 수 있어서 인터페이스를 상속받아 메서드를 구현하는 클래스와 그 메서드를 호출하는 클래스를 서로 독립적으로 개발하거나 관리할 수 있는 장점이 있습니다.

[코드 4-9] 인터페이스의 선언

interface Money{

공통 기능을 추출한 **Money**라는 인터페이스를 선언한다

```
abstract void give(int money, String date);  
abstract void receive(int money, String date);  
}
```

인터페이스는 메서드만 있음!

행동(메서드)
give(int, String)
receive(int, String)

interface Work{

공통 기능을 추출한 **Work**라는 인터페이스를 선언한다

```
abstract void work(int salary, String date);  
}
```

interface Work

행동(메서드)
work(int, String)

부모는 여러 자식에게 유전자를 상속할 수 있지만 자식은 한 부모로부터만 유전자를 물려받을 수 있는 것처럼, 자바에서 각 클래스가 상속받을 수 있는 클래스는 최대 1개였습니다(단일상속). 하지만 낳아준 부모 외에도 어떤 친구들과 어울리느냐에 따라 다른 모습을 보일 수 있듯, 자바의 클래스는 슈퍼 클래스를 상속받는 것 외에도 여러 개의 인터페이스로부터 메서드를 받아올 수 있습니다(다중상속).

인터페이스를 받아 클래스를 만들 때는 `implements`를 사용하고, 여러 개의 인터페이스를 상속하는 경우 `comma(,)`를 이용해 인터페이스 명을 구분합니다. 또한, 상속을 위해 `extends`와 함께 쓰일 때는 항상 `extends`가 `implements`보다 먼저 오게 됩니다. 인터페이스를 이용하여 클래스를 만든 코드를 살펴보겠습니다.

[코드 4-10] 인터페이스의 이용

```
class Citizen extends Person implements Money, Work{
```

Person이라는 클래스를 확장(상속)하여, 공동 기능을 주출한 Money와 Work라는 인터페이스를 담는 객체를 정의/생성하는 데 Citizen이라는 클래스를 만든다

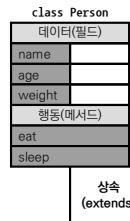
```
public void give(int money, String date){
```

모든 패키지와 클래스에서 접근 가능하고, 리턴값이 없는 give라는 함수를 선언하고,
정수 값을 담는 money와 문자열 값을 담는 date 파라미터 변수를 인자로 받는다

```
System.out.println(date + "] - " + money);
```

date 변수 값 다음에 "] - " 다음에 money 변수 값을 시스템 콘솔에 출력하고 한 줄 띄운다

```
}
```



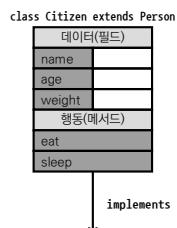
```
public void receive(int money, String date){
```

모든 패키지와 클래스에서 접근 가능하고, 리턴값이 없는 receive라는 함수를 선언하고,
정수 값을 담는 money와 문자열 값을 담는 date 파라미터 변수를 인자로 받는다

```
System.out.println(date + "] + " + money);
```

date 변수 값 다음에 "] + " 다음에 money 변수 값을 시스템 콘솔에 출력하고 한 줄 띄운다

```
}
```



```
public void work(int salary, String date){
```

모든 패키지와 클래스에서 접근 가능하고, 리턴값이 없는 work라는 함수를 선언하고,
정수 값을 담는 salary와 문자열 값을 담는 date 파라미터 변수를 인자로 받는다

```
System.out.println(date + "] + " + salary);
```

date 변수 값 다음에 "] + " 다음에 salary 변수 값을 시스템 콘솔에 출력하고 한 줄 띄운다

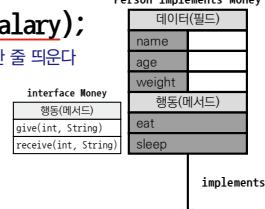
```
}
```

```
}
```

```
class Life{
```

객체를 정의/생성하는 데 Life라는 클래스를 만든다

```
public static void main(String args[]){
```



모든 패키지와 클래스에서 접근 가능하고,
 같은 클래스에서 생성된 객체들과 값을 공유하며,
 리턴값이 없는 main이라는 함수를 선언하고,
 문자열 값을 담는 args[]라는 배열을 인자로 받는다

citizen c = new Citizen();

Citizen 클래스의 객체를 담는 c라는 변수에

Citizen 클래스의 객체를 새로 생성하여 저장한다

c.give(1000, "2015-05-01");

c 변수 안의 give() 메서드를

1000과 “2015-05-01”을 파라미터 값으로 하여 호출한다

c.receive(25000, "2015-05-02");

c 변수 안의 receive() 메서드를 25000과 “2015-05-02”를 파라미터 값으로 하여 호출한다

c.work(300000, "2015-05-25");

c 변수 안의 work() 메서드를 300000과 “2015-05-25”를 파라미터 값으로 하여 호출한다

}

}

실행 결과

2015-05-01] - 1000

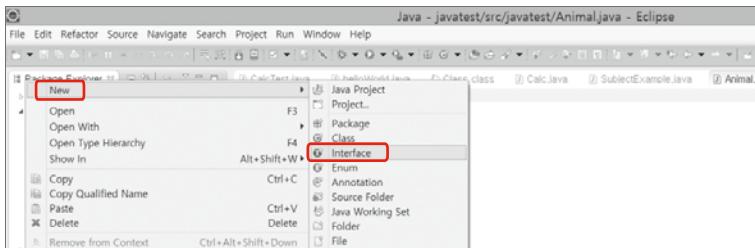
2015-05-02] + 25000

2015-05-25] + 300000

NOTE Eclipse – 인터페이스 자동으로 만들기

이클립스의 Project Explorer나 Package Explorer에서 마우스 오른쪽 버튼을 클릭한 후 [New → Interface]를 누르면 자동으로 인터페이스 선언 코드를 생성할 수 있습니다.

그림 4-9 이클립스에서의 인터페이스 생성



인터페이스 생성 창에 인터페이스 이름을 넣고 [Finish] 버튼을 누르면 인터페이스 선언 코드가 자동으로 생성됩니다.

그림 4-10 인터페이스 생성 창

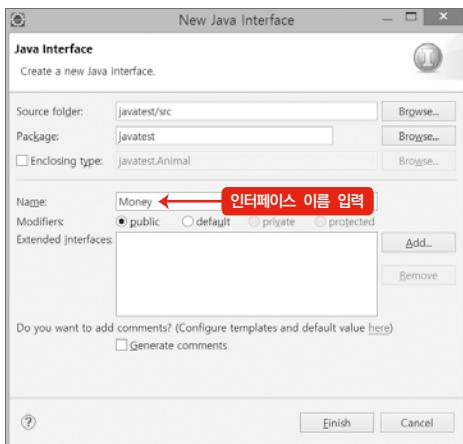
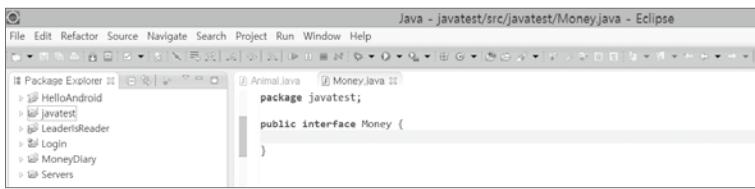


그림 4-11 이클립스에서 자동으로 만든 인터페이스 선언 코드



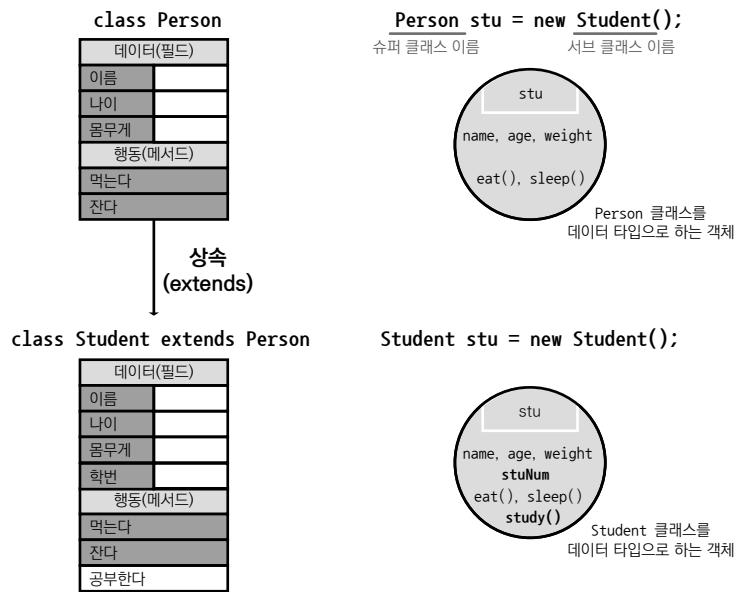
4.6 다형성

다형성 Polymorphism 이란 다양한 형태를 가지는 성질을 말합니다. 자바에서의 다형성이란 슈퍼 클래스를 상속받은 서브 클래스의 객체가 슈퍼 클래스의 객체로도 사용되고 서브 클래스의 객체로도 사용되는 등 다양한 형태를 취할 수 있는 것을 말합니다. 예를 들면, Person 클래스(슈퍼 클래스)를 상속받은 Student 클래스(서브 클래스)의 객체 stu는 원래는 Student 클래스의 객체지만, Person 클래스의 객체로도 다룰 수 있습니다. 마치 흥길동 학생은 사람이므로 흥길동 학생을 지칭할 때

사람이라고도 표현할 수 있는 것처럼 말입니다. “홍길동 학생이 밥을 먹고 있다.”라는 문장을 “홍길동이란 사람이 밥을 먹고 있다.”라고 표현해도 이상하지 않죠?

[그림 4-12]의 오른쪽에 만들어진 두 객체를 비교해 보면, 두 객체 모두 Student 클래스를 이용하여 객체로 만들어졌습니다. 하지만 Person 클래스를 데이터 타입으로 하는 객체는 Person 클래스에 사용한 데이터(필드)와 행동(메서드)만을 가지고 있는 반면, Student 클래스를 데이터 타입으로 하는 객체는 Student 클래스에 사용한 학번(stuNum)이나 공부한다(study) 등의 필드와 메서드를 가지고 있습니다.

그림 4-12 어떤 클래스를 데이터 타입으로 담는가의 차이



서브 클래스의 객체는 슈퍼 클래스의 객체가 될 수 있지만, 슈퍼 클래스의 객체는 서브 클래스의 객체가 될 수 없습니다. 모든 학생은 사람이라고 볼 수 있지만, 그렇다고 모든 사람이 학생이라고 볼 수는 없는 것과 같습니다.

그림 4-13 슈퍼 클래스 객체가 서브 클래스로 쓰일 수 없는 이유

모든 학생은 사람이다. (O)

Person Stu = new Student(); (O)

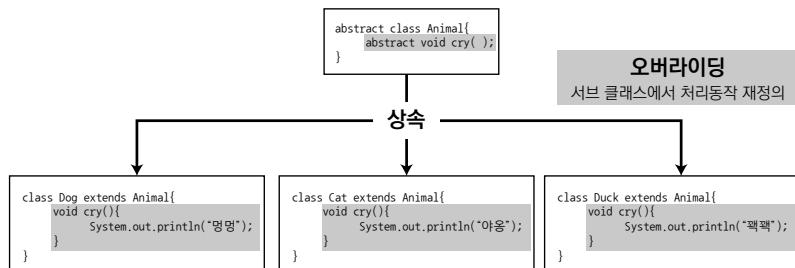
모든 사람은 학생이다. (X)

Student Stu = new Person(); (X)

⇒ 모든 사람이 학생인 것은 아니다.

자바에서 다형성이 유용하게 쓰이는 이유는 코드를 편리하게 작성할 수 있기 때문입니다. 예를 들어, Animal 클래스와 이를 상속받은 Dog 클래스, Cat 클래스 그리고 Duck 클래스를 생각해 봅시다. 같은 동물이지만 개는 “멍멍”, 고양이는 “야옹”, 오리는 “꽥꽥” 운다고 가정하고, Animal 클래스의 cry() 메서드를 각 클래스에서 오버라이딩하면 [그림 4-14]와 같습니다.

그림 4-14 슈퍼 클래스의 메서드를 서브 클래스에서 오버라이딩



이 클래스들을 바탕으로 다형성을 이용하여 객체를 생성해 cry() 메서드를 호출하는 프로그램을 작성하면 [코드 4-11]과 같습니다.

[코드 4-11] 다형성을 이용한 프로그램

class PolyTest{

 객체를 정의/생성하는 둘인 PolyTest라는 클래스를 만든다

 public static void main(String args[]){

 모든 패키지와 클래스에서 접근 가능하고, 같은 클래스에서 생성된 객체들과 값을 공유하며, 리턴값이 없는 main이라는 함수를 선언하고, 문자열 값을 담는 args[]라는 배열을 인자로 받는다

 Animal animals[] = {new Dog(), new Cat(), new Duck()};

 Animal이라는 클래스의 객체를 담는 animals라는 배열을 선언하고

Dog라는 클래스의 객체를 새로 생성하여 0번째 요소에,
Cat이라는 클래스의 객체를 새로 생성하여 1번째 요소에
Duck이라는 클래스의 객체를 새로 생성하여 2번째 요소에 저장한다

```
for(int i=0; i<animals.length; i++){
    정수 값을 담는 i라는 변수에 0을 저장하고 i를 1씩 증가시키면서 i가 animals 배열의 크기(3)보다 작을
    동안 {} 안을 반복해라

    animals[i].cry();
        animals 배열의 i번째 요소에 저장된 값 안의 cry() 메서드를 호출한다

}
```

실행 결과

멍멍
야옹
꽥꽥

NOTE 배열명.length

배열명.length를 사용하면 배열의 요소(혹은 배열의 크기)가 몇 개인지 알 수 있습니다. [코드 4-11]에서도 배열 animals의 요소 개수를 구하려고 animals.length를 사용했고 그 결과 3을 얻을 수 있었습니다.

다형성을 이용한 [코드 4-11]과 다형성을 이용하지 않은 코드를 비교하면 다음과 같이 확연히 차이가 납니다. 이때 객체의 수가 많아질수록 코드의 길이도 차이가 크게 나겠죠?

[다형성을 사용하지 않았을 때]

```
class NonPolyTest{
    public static void main(String args[]){
        Dog dog = new Dog();
        Cat cat = new Cat();
        Duck duck = new Duck();
```

[다형성을 사용했을 때]

```
class PolyTest{
    public static void main(String args[]){
        Animal animals[] =
            {new Dog(), new Cat(), new Duck()};
        for(int i=0; i<animals.length; i++){
```

```
    dog.cry();
    cat.cry();
    duck.cry();
}
}
```

실행 결과

멍멍
야옹
꽥꽥

```
    animals[i].cry();
}
}
}
```

실행 결과

멍멍
야옹
꽥꽥

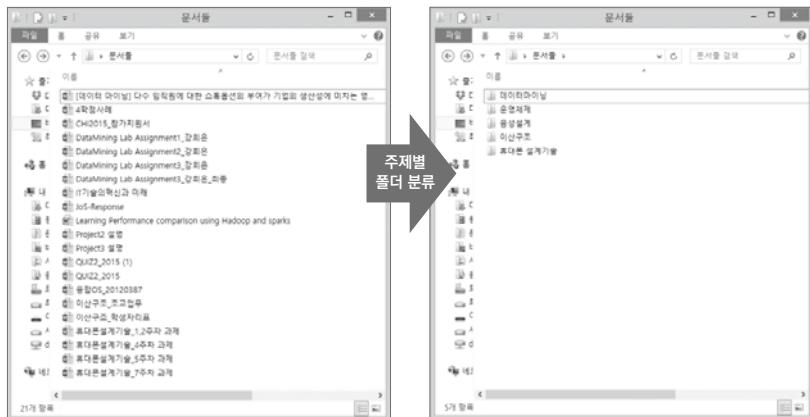
패키지와 접근제어

규모가 큰 자바 프로그램을 작성하다 보면 클래스와 인터페이스가 많아집니다. 이를 효율적으로 분류하고 관리하기 위하여 연관성 있는 클래스와 인터페이스를 모아 둔 것이 바로 ‘패키지’입니다. 그리고 어떤 클래스를 다른 패키지의 클래스가 상속하지 못하게 하는 등 경우에 따라 클래스나 인터페이스를 사용하는 것을 제한할 필요가 있습니다. 이를 위해 사용하는 것이 `public`과 같은 ‘접근 제어 수식어’입니다. 5장에서는 패키지와 접근 제어에 대해 공부할 것입니다. 5.1장에서는 패키지가 무엇인지 공부하고, 다른 패키지의 클래스나 인터페이스를 이용하는 방법을 다룹니다. 5.2장에서는 접근 제어 수식어를 이용하여 클래스와 인터페이스별로 경우에 따라 접근을 제한하는 법에 대해 공부하겠습니다.

5.1 패키지

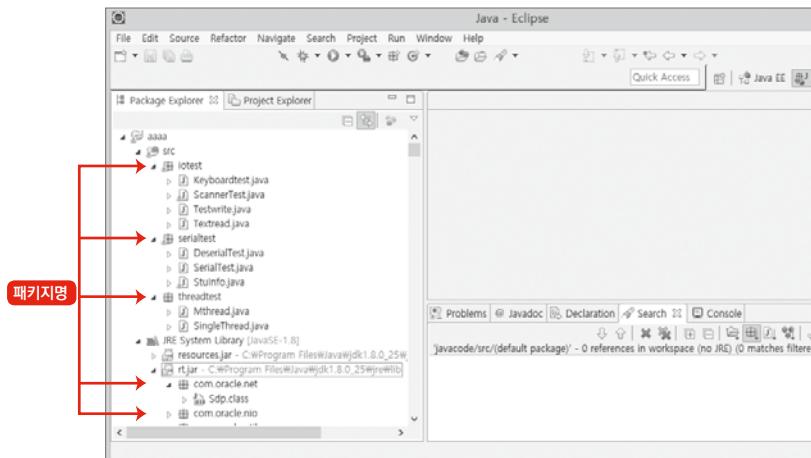
컴퓨터에 여러 가지 문서를 분류하지 않고 그냥 저장해 두면 어떤 문서가 무엇인지 찾기도 힘들고 보기에도 불편합니다. 이 문서들을 관련 주제별로 폴더를 만들어 그 안에 분류하면 이후에는 찾기도 쉽고 보기에도 좋습니다.

그림 5-1 윈도우 파일탐색기에서 본 파일 관리 전과 후



자바에서도 프로그램을 작성하다 보면 많은 클래스와 인터페이스가 생기는데, 이를 풀더에 넣어 두는 것처럼 패키지에 담아 관리합니다.

그림 5-2 이클립스에서 본 패키지의 모습



[코드로 패키지를 선언하는 방법]처럼 프로그램 코드의 맨 위에 패키지를 선언하면 해당 코드의 클래스는 그 패키지에 속하게 됩니다.

클래스 선언 위에 package를 붙임

package iotest;

클래스와 인터페이스를 분류별로 모아 관리하는 **iotest**라는 패키지를 만든다

```
class StuInfo{
```

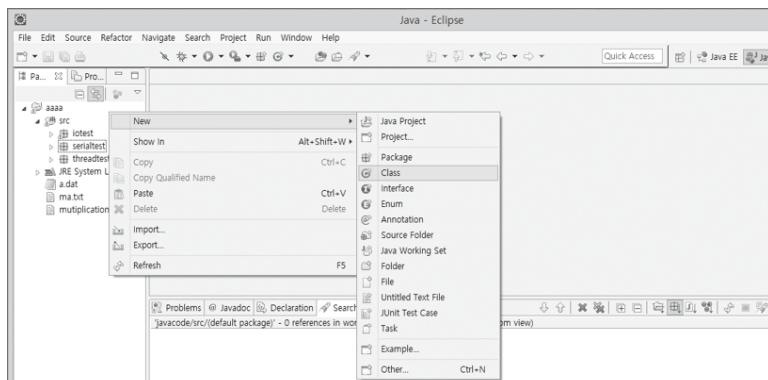
...

}

NOTE Eclipse – 패키지 자동으로 만들기

이클립스에서 자바 코드를 작성하는 경우에는 이클립스가 패키지 선언문을 프로그램 코드의 맨 위에 자동으로 넣어 줍니다. 이를 위해 이클립스에서 클래스를 만들면서 패키지도 함께 선언하는 방법을 공부해 보겠습니다. 이클립스에서 [그림 5~3]처럼 Package Explorer나 Project Explorer 둘 중 아무 곳이나 마우스 오른쪽 버튼을 클릭하고 [New → Class]를 눌러 클래스 선언 창을 띄웁니다.

그림 5~3 이클립스에서 클래스 생성 창을 띄우는 법



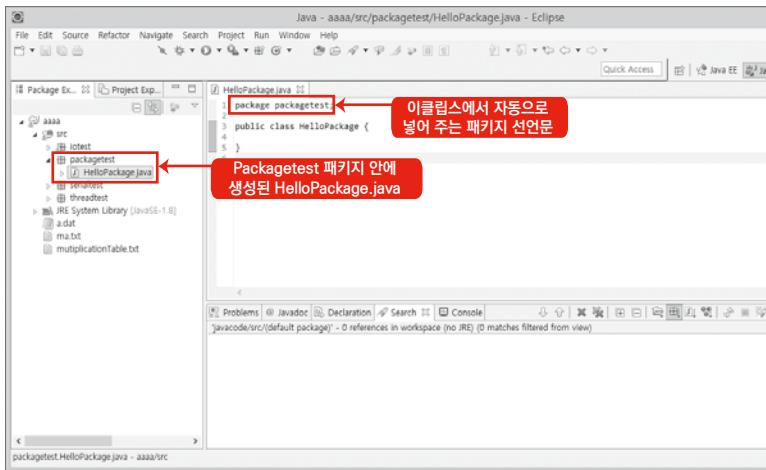
[그림 5~4]와 같이 클래스 설정 창이 뜹니다. 'Package:'라고 표시된 부분에 패키지명을 입력하고 'Name:'이라고 표시된 부분에 클래스명을 입력합니다. 여기에서는 'packagetest'라는 패키지를 선언하고 'HelloPackage'라는 이름의 클래스를 만들겠습니다.

그림 5-4 클래스 생성 창



패키지명과 클래스명을 입력한 후 [Finish] 버튼을 누르면 [그림 5-5]와 같은 화면이 나옵니다. 'Package.' 부분에 아무것도 입력하지 않으면 'default package'라는 이름의 임시 패키지가 생기고 그 안에 클래스가 생성됩니다.

그림 5-5 패키지와 클래스 생성 화면



또한, ‘import’ 수식어를 사용하여 다른 패키지의 클래스를 현재 작성하고 있는 코드에서 사용할 수 있습니다. 예를 들면, 자바를 설치할 때 받았던 JDK에는 미리 자바에서 제공하는 각종 기능을 가진 클래스들의 패키지가 담겨 있는데, 이를 import 수식어를 사용하여 내가 작성하는 코드에서 사용할 수 있습니다. 파일의 내용을 읽고 쓸 때 사용하는 입출력 클래스들의 패키지인 `java.io` 패키지를 이용하여 파일 입출력을 하고 싶다면, 프로그램 코드 맨 위에 `import java.io.*;` 를 넣어 주면 됩니다. `java.io.*`에서 *은 ‘해당 패키지 내에 있는 모든 클래스’를 의미합니다. import 수식어를 사용하여 다른 패키지의 클래스를 현재 코드에서 사용하는 방법은 다음과 같습니다.

[다른 패키지의 클래스를 현재 코드에서 사용하는 방법]

다른 패키지를 import한다

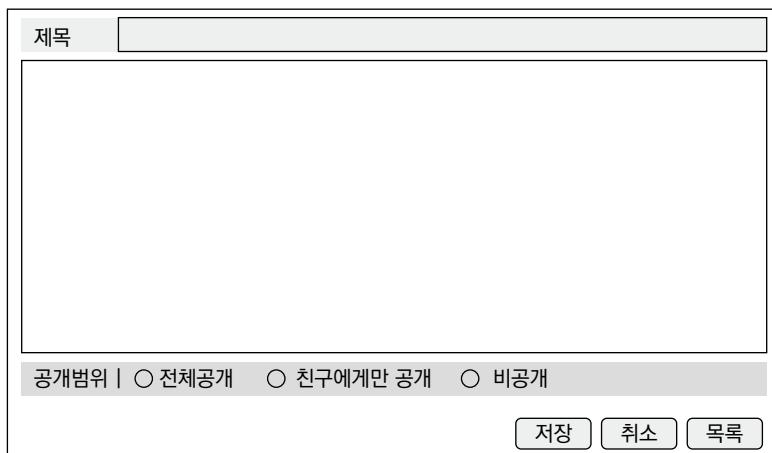
```
import java.io.*;
java.io 패키지의 모든 클래스를 여기서도 사용한다

class StuInfo{
    ...
}
```

5.2 접근제어

어떤 인터넷 커뮤니티에 글을 게시한다고 생각해 봅시다. 나만 읽을 수 있도록 ‘비공개’로 설정할 수도 있고, 다른 사람과 해당 내용을 공유할 수 있도록 ‘공개’로 설정할 수도 있습니다. ‘공개’ 설정 시에도 공개 범위를 ‘친구에게만 공개’로 설정하거나 친구가 아닌 모든 사람과 공유할 수 있는 ‘전체 공개’로 설정할 수도 있습니다.

그림 5-6 인터넷 커뮤니티 글 공개 범위 설정



이와 같은 공개 범위의 원리가 자바 프로그램 코드를 작성할 때도 똑같이 적용됩니다. 클래스와 그 필드, 메서드에 대해 누구한테까지 접근을 허용할 것인지를 ‘접근 제어 수식어’를 사용하여 설정할 수 있습니다. 접근 제어 수식어에는 public, protected, private 그리고 아무것도 설정하지 않는 것을 의미하는 default 총 4가지가 있습니다.

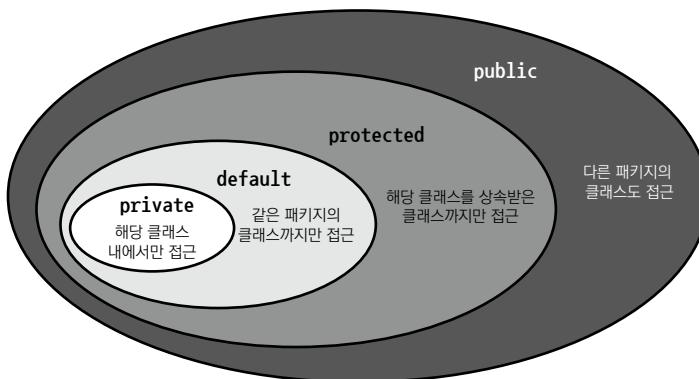
접근 제어 수식어별 접근 허용 범위를 살펴보면 [표 5-1]과 같습니다. 여기서 ‘접근한다’의 의미는 다른 클래스에서 해당 클래스의 필드나 메서드를 호출하거나 사용할 수 있다는 의미입니다.

표 5-1 접근 제어 수식어별 접근 허용 범위

구분	해당 클래스	같은 패키지의 클래스	해당 클래스를 상속받은 클래스	다른 패키지의 클래스
public	○	○	○	○
protected	○	○	○	×
private	○	×	×	×
default	○	○	×	×

이 4가지 수식어의 접근 허용 범위는 ‘public > protected > default > private’ 순입니다. public이 붙은 필드, 메서드, 클래스는 해당 클래스가 소속된 패키지를 import만 해 주면 어디서든 사용할 수 있지만, private이 붙은 필드, 메서드, 클래스는 해당 클래스 내에서만 사용할 수 있습니다. 참고로 private 클래스는 nested class, 즉 클래스 안에 클래스가 정의된 경우에만 만들 수 있습니다.

그림 5-7 접근 제어 수식어별 접근 허용 범위



주의해야 할 것은 public이 붙은 클래스는 하나의 소스 코드 파일당 하나만 있을 수 있다는 점입니다. 또한, 소스 코드 파일명과 public이 붙은 클래스명이 동일 해야 하고 main 함수는 public이 붙은 클래스 안에 있어야 합니다.

그림 5-8 public 접근 제어 수식어 사용시 주의할 점

The screenshot shows the Eclipse IDE interface. In the Package Explorer, there is a project named 'aaaa' with a 'src' folder containing 'Class Object' and 'PersonExample.java'. The code editor displays the following Java code:

```
1 //PersonExample.java
2 package Class.Object;
3
4 public class PersonExample{
5     public static void main(String[] args) {
6         Person personObj1 = new Person("张昊昊",50,80);
7         System.out.println(personObj1.name);
8         System.out.println(personObj1.eat());
9     }
10 }
11
12 class Person{
13     String name;
14     int age;
15     int weight;
16
17     public Person(String name, int age, int weight) {
18         super();
19         this.name = name;
20         this.age = age;
21         this.weight = weight;
22     }
23
24     String eat(){
25         return "eat";
26     }
27
28     String sleep(){
29         return "sleep";
30     }
31 }
32
33 }
```

Annotations in the code editor highlight the package declaration and the main method. A red box with an arrow points to the package declaration with the text "소스 코드 파일명과 public이 붙은 클래스명 일치". Another red box with an arrow points to the main method with the text "main 함수는 public이 붙은 클래스 안에 위치". The status bar at the bottom shows the output of the run command: 'terminated- PersonExample [Java Application] C:\wProgram Files\Java\jdk1.8.0_25\bin\javaw.exe (2015. 10. 1. 오후 5:29:26)'.

입출력

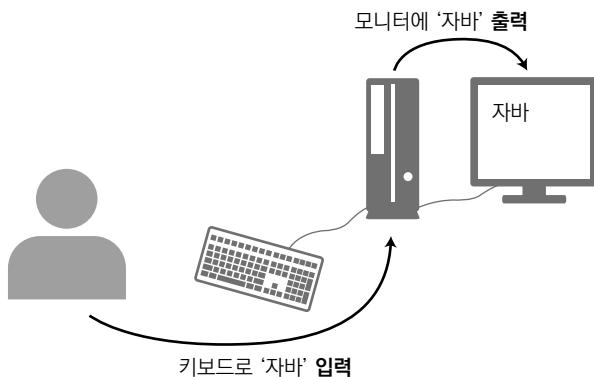
우리가 컴퓨터를 사용하여 문서 작성이나 인터넷 검색 등을 할 때 가장 기본이 되는 것이 입력과 출력입니다. 많은 사람이 사용할 수 있는 자바 프로그램을 작성하기 위해 6장에서는 자바에서의 입력과 출력을 다루려고 합니다.

6.1장에서는 입력과 출력의 정의를, 6.2장에서는 자바 내에서 데이터를 입출력할 때 이동하는 통로인 스트림에 대해 공부합니다. 6.3장에서는 자바 프로그램에서 파일을 읽고 쓰는 방법을 배우고, 6.4장에서는 키보드로 직접 데이터를 입력하는 방법에 대해 다루겠습니다.

6.1 입출력이란

일상 생활에서 어떨 때 컴퓨터를 사용하나요? 보통 인터넷 검색, 온라인 쇼핑, 문서 작성, 이메일 확인과 전송, 게임 등을 하기 위해 사용합니다. 컴퓨터로 이런 작업들을 하기 위해서는 어떤 정보가 필요한지 컴퓨터에 입력하고, 그 결과를 컴퓨터로부터 출력받아야 합니다. 예를 들어, 인터넷 검색을 위해 포털사이트의 검색창에 키보드로 키워드를 입력하면 컴퓨터 모니터로 해당 키워드에 맞는 검색 결과가 출력된 것을 볼 수 있습니다. 보고서를 작성하기 위해 컴퓨터에서 문서 작성 프로그램을 실행하고 키보드로 글자를 입력하는 경우, 글자를 입력하는 즉시 모니터로 입력한 글자가 출력되는 것을 확인할 수 있습니다.

그림 6-1 컴퓨터 입출력의 예

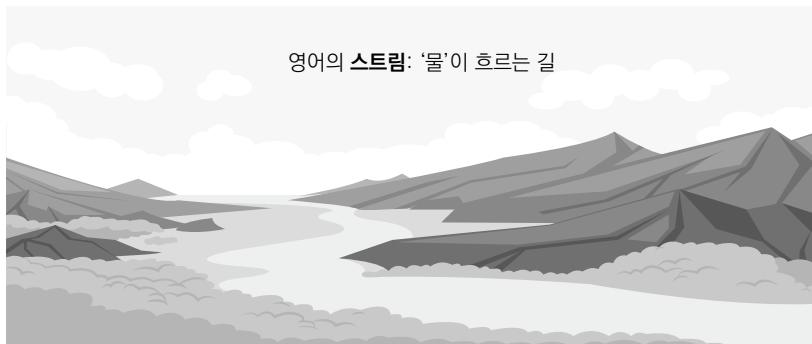


이처럼 입력이란 마우스나 키보드와 같은 입력 장치를 통해 컴퓨터에 데이터를 넣는 것이고, 출력이란 모니터나 프린터와 같은 출력 장치를 통해 컴퓨터로부터 데이터를 끄집어내는 것을 말합니다.

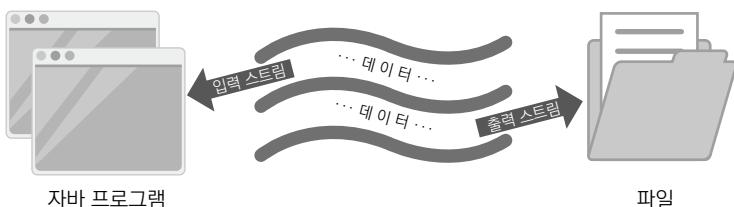
6.2 스트림

영어로 ‘Stream(스트림)’은 ‘강’입니다. 강은 ‘물이 흐르는 길’이라는 의미로, 의미 그대로 물의 흐름을 나타냅니다. 자바에서의 스트림 역시 데이터가 흐르는 길을 의미하며, 프로그램에 입출력되는 데이터의 흐름을 나타냅니다. 즉, 스트림은 데이터의 이동 통로이고, 프로그램에서 입출력되는 데이터가 스트림을 통해 이동하게 됩니다. 강에서 물은 순류와 역류가 동시에 존재하지 않고 한 방향으로 흐릅니다. 마찬가지로 자바의 스트림에서도 데이터는 한 방향으로만 흐르기 때문에 입력과 출력 시 사용하는 스트림이 구분되어 있습니다. 프로그램에 입력되는 데이터의 길을 ‘**입력 스트림**’^{Input Stream}이라고 하고, 프로그램에서 출력되는 데이터의 길을 ‘**출력 스트림**’^{Output Stream}이라고 합니다.

그림 6-2 스트림



자바의 스트림: '데이터'가 흐르는 길



자바 프로그램에 입출력하는 곳은 다양합니다. “Hello Java”라는 내용이 있는 텍스트 파일(.txt)에서 그 내용을 읽어올 수도 있고(입력), 자바 프로그램에서 새로운 텍스트 파일을 생성하고 어떤 내용을 저장할 수도 있습니다(출력). 또는 자바 프로그램이 실행 중일 때는 키보드를 통해 바로바로 데이터를 입력하고 그 내용을 모니터를 통해 확인(출력)할 수 있습니다.

원래대로라면 장치별로 데이터를 입력하고 출력하는 방법이 각기 다릅니다. 하지만 자바에서는 상황별로 입출력을 위한 데이터 이동 통로로 스트림을 클래스로 만들어 JDK 안에 제공하기 때문에 프로그래머는 장치별로 세부적인 입출력이 어떻게 되는지 신경 쓰지 않아도 됩니다.

프로그래머가 입출력을 위해 할 일은 자바에서 미리 제공해 준 스트림에 대한 클래스가 있는 `java.io` 패키지를 쓰겠다는 말(`import java.io.*`)을 프로그램 코드

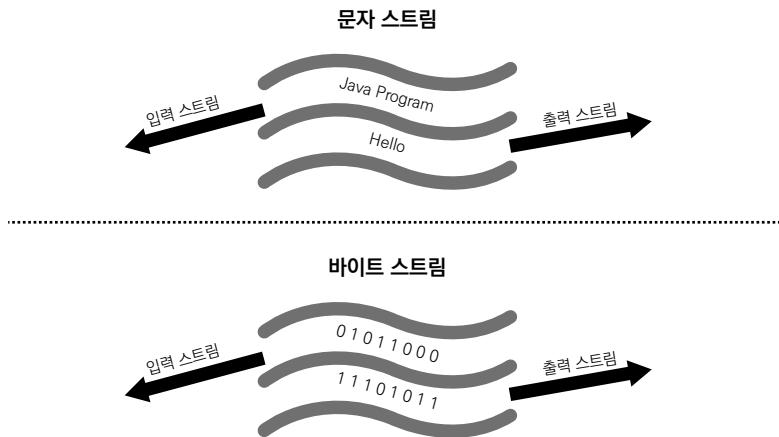
가장 위에 적고, 데이터가 흘러갈 수 있게끔 스트림을 열고, 데이터를 스트림을 통해 흘려보낸 후 다시 스트림을 닫는 것입니다.

스트림 사용법

- ① 'import java.io.*'를 프로그램 코드의 가장 위에 적는다.
- ② 스트림을 연다(스트림 객체를 생성한다).
- ③ 스트림을 통해 데이터를 흘려보낸다.
- ④ 스트림을 닫는다.

데이터가 흐르는 길인 스트림은 흐르는 데이터의 종류에 따라 문자 스트림[Character Stream]과 바이트 스트림[Byte Stream]으로 나누어집니다. 문자 스트림은 사람이 이해할 수 있는 언어로 표현된 문자 데이터가 이동하는 길이고, 바이트 스트림은 동영상, 이미지, 음악 등 0과 1로 표현된 데이터가 이동하는 길입니다.

그림 6-3 스트림의 종류



스트림의 종류에 따라 스트림을 다루는 클래스명도 다릅니다. JDK에 있는 스트림을 다루는 클래스는 여러 가지가 있습니다. 하지만 이들은 대표적인 4가지의 추상 클래스에서 파생된 서브 클래스입니다.

표 6-1 대표적인 4가지 스트림 추상 클래스

구분	입력 스트림	출력 스트림
문자 스트림	Reader	Writer
바이트 스트림	InputStream	OutputStream

따라서 어떤 스트림 클래스의 이름이 `FileReader`면, 해당 클래스는 문자를 파일로부터 스트림을 통해 입력받아 읽어 오는 클래스임을 알 수 있습니다. `FileOutputStream`의 경우 바이트를 스트림을 통해 출력하여 파일에 쓰는 클래스임을 알 수 있습니다.

6.3장과 6.4장에서는 다양한 스트림 클래스를 실제로 활용하는 방법과 그 예시를 살펴보겠습니다.

6.3 파일 쓰고 읽기

파일은 데이터의 모음을 말하며 문서, 동영상, 그림, 소리 등이 모두 파일이 될 수 있습니다. 문서 파일, 동영상 파일, 그림 파일, 소리 파일 등으로 부르기도 합니다. 자바 프로그램에서 파일을 작성하는 과정을 쉽게 생각해 보기 위해서, 파일 중 하나인 문서를 작성할 때의 과정을 생각해 봅시다. 우선 아래 한글이나 MS Word 등의 문서 작성 프로그램을 열고 ‘새로 만들기’를 클릭하여 새 문서를 엽니다. 그리고 새로운 문서에 내용을 작성한 후, 저장하고 해당 문서를 닫는 과정을 거치게 됩니다. 또, 이미 작성된 문서를 읽을 때도 먼저 문서를 열고, 읽고, 다 읽으면 문서를 닫는 과정을 거칩니다. 자바 프로그램에서도 파일을 읽고 쓸 때는 우선 파일을 열고, 파일에 내용을 적거나 읽고, 마지막으로 파일을 닫는 과정을 거칩니다.

그림 6-4 파일을 쓰거나 읽는 순서

컴퓨터에서 문서를 쓰거나 읽는 경우



① 새 문서를 연다.

② 새 문서에 내용을 쓰거나 읽는다.

③ 새 문서를 닫는다.

자바 프로그램에서 파일을 쓰거나 읽는 경우

① 파일을 연다.

② 파일에 내용을 쓰거나 읽는다.

③ 파일을 닫는다.

앞에서 설명한 바와 같이, 자바에서 스트림은 데이터의 종류에 따라 문자열 스트림과 바이트 스트림으로 나누어져 있었습니다. 파일 역시 그 안의 내용이 무엇이냐에 따라 ‘텍스트 파일’과 ‘바이너리 파일’로 나누어집니다. 파일 안의 내용이 문자열로 되어 있으면 ‘텍스트 파일’이라고 하고, 바이트(0이나 1로만 구성)로 되어 있으면 ‘바이너리 파일’이라고 합니다. 바이너리 파일의 종류로는 사진 파일, 동영상 파일 등이 있습니다.

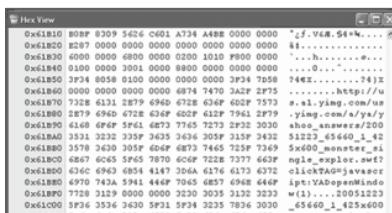
문자열 스트림을 다루는 클래스는 입력할 때 Reader 클래스를, 출력할 때 Writer 클래스를 상속받고, 바이트 스트림을 다루는 클래스는 입력할 때 InputStream 클래스를, 출력할 때 OutputStream 클래스를 상속받습니다. 마찬가지로 자바 프로그램에서 파일을 읽고 쓸 때는 파일의 종류가 텍스트 파일인지 바이너리 파일인지에 따라 사용하는 스트림 클래스가 달라집니다.

그림 6-5 파일의 종류

텍스트 파일(문자열로 구성된 파일)



바이너리 파일(0과 1로 구성된 파일)



이번 장에서는 우리가 자주 사용하는 텍스트 파일을 대상으로 파일을 쓰고 읽는 방법에 대해 공부해 보겠습니다.

6.3.1 파일 쓰기

자바에서는 텍스트 파일을 쓸 때 `FileWriter`라는 스트림 클래스의 객체를 만들어 사용합니다. 해당 객체는 파일에 내용을 쓸 때 사용하는 `write()` 메서드와 파일을 닫을 때 사용하는 `close()` 메서드를 가지고 있습니다.

텍스트 파일을 쓸 때 사용하는 메서드

- ① 파일명의 파일이 없으면 새로 만들고, 있으면 해당 파일을 연다.

```
FileWriter 객체명 = new FileWriter("파일명");
```

- ▶ “파일명”에 경로 없이 이름만 쓰면(ex. a.txt) 이클립스 실행 시 지정하는 workspace 폴더에 해당 파일이 저장됩니다.
- ▶ “파일명”에 이름과 경로를 함께 쓰면(ex. c://a.txt) 그 경로에 해당 파일이 저장됩니다.

- ② 파일 내용을 쓴다.

```
객체명.write();
```

- ③ 파일을 닫는다.

```
객체명.close();
```

텍스트 파일에 내용을 쓸 때 특정한 이름의 파일이 없는 경우라면 새로 생성하고, 있는 경우라면 해당 파일을 열어 그 내용을 수정합니다. 이 메서드를 기본으로 텍스트 파일을 쓰는 방법을 살펴보면 다음과 같습니다.

- (1) 새로 만들거나 열 파일명으로 `FileWriter` 스트림 클래스의 객체를 생성한다.
 -> `FileWriter fw = new FileWriter("a.txt");`
- (2) `write` 메서드를 사용하여 파일에 내용을 쓴다. -> `FileWriter fw = new FileWriter("a.txt");
 fw.write("Hello!\n");`
- (3) `close` 메서드를 사용하여 파일을 닫는다. -> `FileWriter fw = new FileWriter("a.txt");
 fw.write("Hello!\n");
 fw.close();`

FileWriter fw = new FileWriter("a.txt");

문자열로 된 파일(텍스트 파일)을 쓰는 스트림 클래스의 객체를 담는 `fw`라는 변수에

"a.txt"란 파일을 쓰는 `FileWriter` 클래스의 객체를 새로 생성하여 저장한다

fw.write("Hello!\n");

`fw` 안의 `write()` 메서드를 호출하여 내용을 쓴다 Hello! 줄바꿈

fw.close();

`fw` 안의 `close()` 메서드를 호출하여 파일을 닫는다

이제 실제로 텍스트 파일을 작성하는 자바 프로그램 예제를 살펴보겠습니다. 실제 프로그램 코드에서는 `try~catch` 구문을 사용하여 파일이 정상적으로 작성되지 않는 등의 에러 상황이 발생했을 때 프로그램이 수행을 멈추지 않고 에러 메시지를 리턴하도록 합니다.

NOTE try~catch 구문

자바에서는 프로그램 코드를 실행하다가 오류가 나면 예외(Exception, 예외)이라는 에러 메시지를 발생시키고, 해당 프로그램의 실행을 멈춥니다. 그런데 프로그램의 실행이 갑자기 중단되어 버리면 해당 프로그램의 사용자는 당황스러울 수 있습니다. 이런 상황을 방지하기 위해 자바는 `try~catch` 구문을 제공합니다.

프로그램 코드에서 미리 오류가 발생할 수도 있을 것이라 예상되는 부분의 코드를 `try{}` 블록 안에 넣고, `catch{}` 블록에는 예외가 발생했을 때 처리할 명령문 코드를 적어 둡니다. 보통은 어떤 오류가 발생했는지 출력하는 명령문을 적습니다. `try{}` 블록 안에 기술된 코드를 실행 중에 오류가 나서 예외가 발생하게 되면 이는 `catch{}` 블록으로 던져집니다. `catch{}` 구문에서는 예외를 잡아서 그 예외가 발생했을 때 처리할 코드를 실행합니다.

예외의 종류는 매우 다양하기 때문에 종류별로 `catch{}` 구문을 만들어 둘 수도 있습니다. `try~catch` 구문의 선언 방법은 다음과 같습니다.

```
try{
    오류가 발생할 수도 있을 것이라 예상되는 코드
} catch(Exception e){
    해당 오류 발생 시 어떻게 대처할 것인지 기술한 코드
}
```

오류가 발생하는 것과는 별개로 try 블록에 들어간 코드 후에 실행되어야 하는 코드가 있다면, finally 블록을 사용할 수 있습니다. try~catch~finally 구문의 선언 방법은 다음과 같습니다.

```
try{
    오류가 발생할 수도 있을 것이라 예상되는 코드
} catch(Exception e){
    해당 오류 발생 시 어떻게 대처할 것인지 기술한 코드
} finally{
    오류 발생 여부와는 상관없이 실행되어야하는 코드
}
```

[코드 6-1] 5단부터 시작하는 구구단을 텍스트 파일에 쓰는 프로그램

import java.io.*;

java.io 패키지의 모든 클래스를 여기서 사용한다

class Testwrite{

객체를 정의/생성하는 틀인 Testwrite라는 클래스를 만든다

public static void main(String[] args) {

모든 패키지와 클래스에서 접근 가능하고 같은, 클래스에서 생성된 객체들과 값을 공유하며, 리턴값이 없는 main이라는 함수를 선언하고, 문자열 값을 담는 args라는 배열을 인자로 받는다

try {

{ } 안을 시도해 보고 에러가 뜨면 Exception 메시지를 catch 블록으로 던진다

FileWriter fw = new FileWriter("multiplicationTable.txt");

문자열로 된 파일(텍스트 파일)을 쓰는 스트림 클래스의 객체를 담는 fw라는 변수에

"multiplicationTable.txt"라는 파일을 쓰는 FileWriter 클래스의 객체를 새로 생성하여 저장한다

for(int i=5; i<10; i++){

정수 값을 담는 i라는 변수에 5를 저장하고 i를 1씩 증가시키면서 i가 10보다 작을 동안 {} 안을 반복해라

for(int j=5; j<10; j++){

정수 값을 담는 j라는 변수에 5를 저장하고 i를 1씩 증가시키면서 j가 10보다 작을 동안 {} 안을 반복해라

```

fw.write(i+"x"+j+"="+(i*j)+" ");
fw 안의 write() 메서드를 호출하여 내용을 쓴다 i 변수 값×j 변수 값=i 변수 값×j 변수 값
}

fw.write("\n");
fw 안의 write() 메서드를 호출하여 내용을 쓴다 줄바꿈

}

fw.close();
fw 안의 close() 메서드를 호출하여 파일을 닫는다

} catch(IOException e) {
Exception 메시지를 받은 경우 {} 안을 수행한다 입출력 에러 타입의 e 변수

System.out.println(e);
e 변수 값을 시스템 콘솔에 출력하고 한 줄 띄운다

}
}
}

```

실행 결과

이를 실행하면 이클립스 콘솔 창에는 아무것도 뜨지 않습니다. 하지만 이클립스 실행 시 설정한 workspace 폴더에 가 보면 프로그램으로 작성한 파일이 생성된 것을 볼 수 있습니다.

연습 6-1

[연습 3-5]의 실행 결과를 account.txt 파일에 써보세요.

연습 3-5의 실행 결과

```
=====
당일 개설 계좌 명단 =====
계좌번호: 1002-03-0001000, 예금주: 홍길동, 잔액: 100000원
계좌번호: 1002-06-0701421, 예금주: 박채윤, 잔액: 20000000원
계좌번호: 1002-01-9891123, 예금주: 이영희, 잔액: 360433원
```

계좌번호: 1002-05-0801000, 예금주: 강진철, 잔액: 50000원
계좌번호: 1002-02-0150802, 예금주: 임성수, 잔액: 1000000000원

6.3.2 파일 읽기

자바에서는 텍스트 파일을 읽을 때 `FileReader`라는 스트림 클래스의 객체를 만들어 사용합니다. 해당 객체는 파일을 읽을 때 사용하는 `read()` 메서드와 파일을 닫을 때 사용하는 `close()` 메서드를 가지고 있습니다.

텍스트 파일을 읽을 때 사용하는 메서드

- ① 파일을 연다.
-

```
FileReader 객체명 = new FileReader("파일명");
```

- ② 파일 내용을 읽는다
-

```
int 변수명 = 객체명.read();
```

▶ `read()` 메서드는 읽어 온 문자를 `int` 형으로 반환하고, 읽어 온 데이터가 없으면 `-1`을 리턴합니다.

- ③ 파일을 닫는다.
-

```
객체명.close();
```

내용을 읽을 파일이 있는 경우 정상적으로 실행되지만, 없는 경우에는 오류가 나서 예외가 발생합니다. 이 메서드를 기본으로 텍스트 파일을 읽는 방법을 살펴보면 다음과 같습니다.

(1) 열 파일명을 가지고 FileReader 스트림 클래스의 객체를 생성한다.

```
-> FileReader fr = new FileReader("a.txt");
```

(2) while문을 돌면서 read 메서드를 사용하여 파일에 내용이 더는 없을 때까지 계속 읽는다.

```
-> FileReader fr = new FileReader("a.txt");
    int c;
    while((c=fr.read())!= -1){
    }
```

(3) 읽어 온 내용을 char형으로 변환하고 이를 출력한다.

```
-> FileReader fr = new FileReader("a.txt");
    int c;
    while((c=fr.read())!= -1){
        char c = (char)i;
        System.out.print(c);
    }
```

(3) close 메서드를 사용하여 파일을 닫는다.

```
-> FileReader fr = new FileReader("a.txt");
    int c;
    while((c=fr.read())!= -1){
        char c = (char)i;
        System.out.print(c);
    }
    fr.close();
```

FileReader fr = new FileReader("a.txt");

문자열로 된 파일(텍스트 파일)을 읽는 스트림 클래스의 객체를 담는 fr이라는 변수에

"a.txt"란 파일을 읽는 FileReader 클래스의 객체를 새로 생성하여 저장한다

int i;

정수 값을 담는 i라는 변수를 선언한다

while((i=fr.read())!= -1) {

fr 안의 read() 메서드를 호출하여 파일을 읽고, 그 값을 변수 i에 저장하고 변수 i가 -1이 아닌 동안 {} 안을 반복해라

char c = (char)i;

문자 값을 담는 c라는 변수를 선언하고 변수 i를 문자형으로 변환하여 저장한다

System.out.println(c);

c 변수 값을 시스템 콘솔에 출력하고 한 줄 띄운다

}

fr.close();

fr 안의 close() 메서드를 호출하여 파일을 닫는다

[코드 6-1]에서 작성한 ‘multiplicationTable.txt’ 파일을 읽어 오는 프로그램 예제를 살펴보겠습니다. 여기서도 마찬가지로 try~catch 구문을 사용하여 파일이 없는 등의 에러 상황이 발생했을 때 프로그램이 수행을 멈추지 않고 에러 메시지를 리턴하도록 합니다.

[코드 6-2] 코드 6-1의 텍스트 파일을 읽는 프로그램

import java.io.*;

java.io 패키지의 모든 클래스를 여기서 사용한다

class Testread{

객체를 정의/생성하는 툴인 Testread라는 클래스를 만든다

public static void main(String[] args) {

모든 패키지와 클래스에서 접근 가능하고, 같은 클래스에서 생성된 객체들과 값을 공유하며, 리턴값이 없는 main이라는 함수를 선언하고, 문자열 값을 담는 args라는 배열을 인자로 받는다

try {

{ } 안을 시도해 보고 에러가 뜨면 Exception 메시지를 catch 블록으로 던진다

FileReader fr = new FileReader("multiplicationTable.txt");

문자열로 된 파일(텍스트 파일)을 읽는 스트림 클래스의 객체를 담는 fr이라는 변수에

“multiplicationTable.txt”란 파일을 읽는 FileReader 클래스의 객체를 새로 생성하여 저장한다

int i;

정수 값을 담는 i라는 변수를 선언한다

while((i=fr.read())!=-1) {

fr 안의 read() 메서드를 호출하여 파일을 읽고, 그 값을 변수 i에 저장하여 변수 i가 -1이 아님 동안 {} 인을 반복해라

char c = (char)i;

문자 값을 담는 c라는 변수를 선언하고 변수 i를 문자형으로 변환하여 저장한다

System.out.println(c);

c 변수 값을 시스템 콘솔에 출력하고 한 줄 띄운다

}

fr.close();

fr 안의 close() 메서드를 호출하여 파일을 닫는다

} catch(IOException e) {

Exception 메시지를 받은 경우 {} 인을 수행한다 입출력 에러 타입의 e 변수

```
System.out.println(e);  
e 변수 값은 시스템 콘솔에 출력하고 한 줄 띄운다  
}  
}  
}
```

실행 결과

```
5x5=25 5x6=30 5x7=35 5x8=40 5x9=45  
6x5=30 6x6=36 6x7=42 6x8=48 6x9=54  
7x5=35 7x6=42 7x7=49 7x8=56 7x9=63  
8x5=40 8x6=48 8x7=56 8x8=64 8x9=72  
9x5=45 9x6=54 9x7=63 9x8=72 9x9=81
```

연습 6-2

[연습 6-1]에서 작성했던 account.txt 파일을 읽는 프로그램을 작성해보세요.

6.4 키보드로 입력하기

컴퓨터 내 저장된 파일에서 데이터를 읽어 오는 것이 아니라 자바프로그램이 실행 중일 때 키보드로 직접 데이터를 입력받고 싶을 때가 있습니다. 이때는 Scanner 클래스의 객체를 만들어 사용합니다. 해당 객체는 읽어 올 데이터의 타입에 따라 문자열이면 next(), 정수면 nextInt(), 실수면 nextDouble() 등의 메서드를 따로 사용합니다. 이 메서드들은 Space 키로 띄어쓰기하기 전까지 입력된 데이터를 읽어 오지만, nextLine() 메서드는 Enter 키로 줄바꿈하기 전까지 입력된 데이터를 읽어 올 수 있습니다. 즉, 문장을 읽어 올 수 있습니다.

표 6-2 Scanner 클래스가 제공하는 메서드

메서드	설명
next()	Space키로 띄어쓰기하기 전까지 입력된 문자들을 읽어온다.
nextInt()	Space키로 띄어쓰기하기 전까지 입력된 정수들을 읽어온다.
nextDouble()	Space키로 띄어쓰기하기 전까지 입력된 실수들을 읽어온다.
nextLine()	Enter키로 줄바꿈하기 전까지 입력된 문자들을 읽어온다.

Scanner 클래스의 객체를 사용하여 키보드로 입력받는 방법은 다음과 같습니다.

[Scanner 클래스를 사용하여 키보드로 입력받는 방법]

(1) Scanner 클래스의 객체를 생성한다. → `Scanner sc = new Scanner(System.in);`

(2) 읽을 데이터의 형에 따라 메서드를 써서 데이터를 읽어 온다.

```
→ Scanner sc = new Scanner(System.in);
    String s = sc.next();
    int i = sc.nextInt();
    Double d = sc.nextDouble();
    String ss = sc.nextLine();
```

Scanner sc = new Scanner(System.in);

입력 클래스의 객체를 담는 `sc`라는 변수에, 키보드에서 입력받는 Scanner 클래스의 객체를 새로 생성하여 저장한다

String s = sc.next();

문자열 값을 담는 `s`라는 변수에 `sc` 안의 `next()` 메서드를 호출하여 문자열로 다음 값을 읽어 와서 저장한다

int i = sc.nextInt();

정수 값을 담는 `i`라는 변수에 `sc` 안의 `nextInt()` 메서드를 호출하여 정수로 다음 값을 읽어 와서 저장한다

Double d = sc.nextDouble();

실수 값을 담는 `d`라는 변수에 `sc` 안의 `nextDouble()` 메서드를 호출하여 실수로 다음 값을 읽어 와서 저장한다

String ss = sc.nextLine();

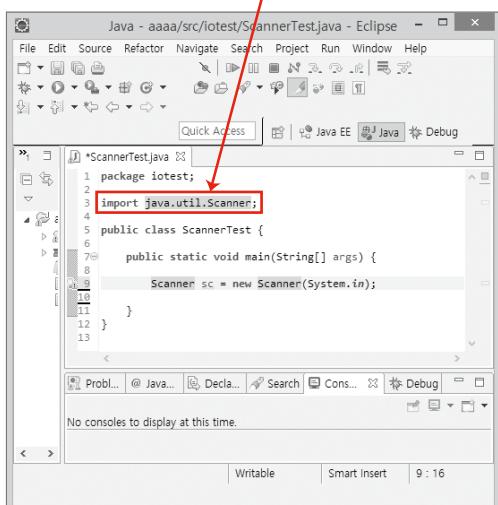
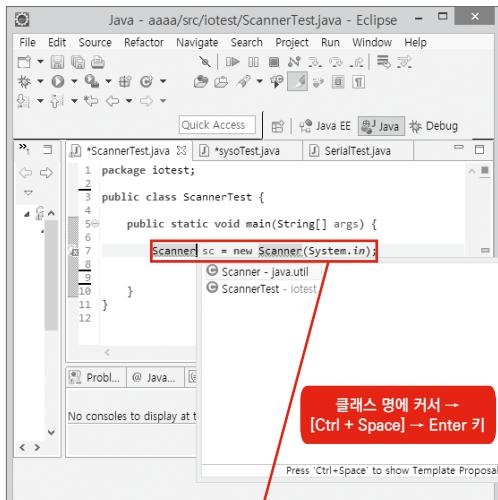
문자열 값을 담는 `ss`라는 변수에 `sc` 안의 `nextLine()` 메서드를 호출하여 문자열로 다음 값을 읽어 와서 저장한다

Scanner 클래스를 사용할 때 주의할 점은 기존의 스트림 클래스를 사용할 때 임포트했던 `java.io.*`가 아니라 `java.util.Scanner`를 임포트해야 한다는 것입니다.

NOTE Eclipse – 임포트 자동으로 하기

이클립스에서 사용할 클래스에 마우스 커서를 두고 자동 완성 단축키인 [Ctrl + Space]를 누르면 자동으로 해당 클래스를 사용할 때 필요한 라이브러리를 임포트하는 코드가 생성됩니다. 임포트할 수 있는 대상이 여러 개가 있다면 옆에 조그만 창이 뜨고 그 중에서 적절한 것을 선택한 후 Enter 키를 누르면 됩니다.

그림 6-6 이클립스에서 자동으로 임포트하는 방법



그럼 Scanner 클래스를 사용하여 키보드로부터 입력을 받아 오는 예제 프로그램을 살펴보겠습니다.

[코드 6-3] 키보드로부터 데이터를 입력받는 프로그램

```
import java.util.Scanner;
```

java.util.Scanner 패키지의 클래스들을 여기서 사용한다

```
class TestScanner{
```

객체를 정의/생성하는 툴인 TestScanner라는 클래스를 만든다

```
public static void main(String[] args) {
```

모든 패키지와 클래스에서 접근 가능하고, 같은 클래스에서 생성된 객체들과 값을 공유하며, 리턴값이 없는 main이라는 함수를 선언하고, 문자열 값을 담는 args라는 배열을 인자로 받는다

```
Scanner sc = new Scanner(System.in);
```

입력 클래스의 객체를 담는 sc라는 변수에, 키보드에서 입력받는 Scanner 클래스의 객체를 새로 생성하여 저장한다

```
System.out.print("name:");
```

"name:" 을 시스템 콘솔에 출력하고 한 줄 띄운다

```
String name = sc.nextLine();
```

문자열 값을 담는 name라는 변수에 sc 안의 nextLine() 메서드를 호출하여 문자열로 다음 값을 읽어 와서 저장한다

```
System.out.print("age:");
```

"age:" 을 시스템 콘솔에 출력하고 한 줄 띄운다

```
int age = sc.nextInt();
```

정수 값을 담는 age라는 변수에 sc 안의 nextInt() 메서드를 호출하여 정수로 다음 값을 읽어 와서 저장한다

```
System.out.println("*****회원정보*****");
```

"*****회원정보*****" 을 시스템 콘솔에 출력하고 한 줄 띄운다

```
System.out.println("name:" + name);
```

"name: " 다음에 name 변수 값을 시스템 콘솔에 출력하고 한 줄 띄운다

```
System.out.println("age:" + age);
```

"age: " 다음에 age 변수 값을 시스템 콘솔에 출력하고 한 줄 띄운다

```
System.out.println("*****");
```

"*****" 을 시스템 콘솔에 출력하고 한 줄 띄운다

```
}
```

```
}
```

[코드 6-3]을 실행합니다. 키보드로부터 데이터를 입력받기 위한 코드이므로 실행 후 name: 외에 다른 항목은 아직 나타나지 않습니다. name과 age에 원하는 값을 입력하면 다음과 같이 입력한 결과가 출력된 결과를 얻을 수 있습니다.

실행 결과

name: Kang Hee Eun

age: 27

*****회원정보*****

name: Kang Hee Eun

age: 27

연습 6-3

실행 결과처럼 '1'을 누르면 계좌를 생성하고, '2'를 누르면 현재 개설된 계좌 정보들을 출력하고, '3'을 누르면 프로그램을 종료하는 은행 시스템을 작성해봅시다.

실행 결과

원하는 서비스의 번호를 입력하세요. 1)계좌 개설, 2)현재 개설된 계좌 정보 보기, 3)시스템 종료

1

예금주:

이지현

예금할 금액:

120040

예금주 이지현 님 앞으로 생성된 계좌 번호는 1002-72-1879206입니다.

원하는 서비스의 번호를 입력하세요. 1)계좌 개설, 2)현재 개설된 계좌 정보 보기, 3)시스템 종료

1

예금주:

임지수

예금할 금액:

65000

예금주 임지수 님 앞으로 생성된 계좌 번호는 1002-59-8189877입니다.

원하는 서비스의 번호를 입력하세요. 1)계좌 개설, 2)현재 개설된 계좌 정보 보기, 3)시스템 종료

2

***** 개설된 계좌 정보 *****

계좌번호:1002-72-1879206, 예금주:이지현, 잔액:120040

계좌번호:1002-59-8189877, 예금주:임지수, 잔액:65000

원하는 서비스의 번호를 입력하세요. 1)계좌 개설, 2)현재 개설된 계좌 정보 보기, 3)시스템 종료

3

객체의 직렬화

자바 프로그램에 입출력되는 데이터는 스트림이라는 데이터 통로를 통해 이동했습니다. 그런데 문자열이나 정수형 같은 데이터와는 달리 객체는 바이트 형이 아니어서 스트림을 통해 파일에 저장하거나 네트워크로 전송할 수 없습니다. 따라서 객체를 스트림을 통해 입출력하려면 바이트 배열로 변환하는 것이 필요한데, 이를 ‘직렬화’라고 합니다. 반대로 스트림을 통해 받은 직렬화된 객체를 원래 모양으로 만드는 과정을 ‘역직렬화’라고 합니다. 7장에서는 객체의 직렬화에 대한 공부하겠습니다. 7.1장에서는 직렬화와 역직렬화가 무엇인지 알아보고, 7.2장에서는 직렬화 클래스를 만드는 방법을 다루겠습니다. 마지막으로 7.3장에서는 객체를 직렬화와 역직렬화하는 방법에 대해 공부하겠습니다.

7.1 직렬화와 역직렬화

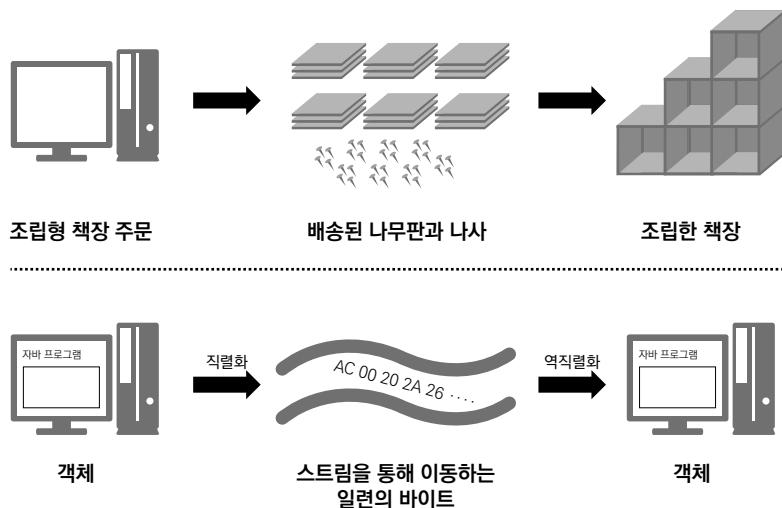
몇 년 전 책장이 필요하여 조립형 책장을 인터넷으로 주문한 적이 있었습니다. 6개의 나무상자로 된 것이었는데, 인터넷에서 사진으로 보았을 때는 이동이 간편하고 상자를 어떻게 구성해 놓느냐에 따라 모양도 다양하게 만들 수 있을 거란 생각이 들었습니다. 그런데 배송된 것을 보니, 6개의 나무상자는 없고 36개의 나무판과 몇 개의 나사가 들어 있었습니다. 완성된 나무상자로 배송하려면 부피가 커서 불편하니까 나무판과 나사만 배송해 주고, 물건을 받은 사람이 직접 상자를 조립하여야 하는 것이었죠.

그림 7-1 책장 → 나무판과 나사 → 책장



자바에서의 객체 직렬화와 역직렬화도 책장을 나무판과 나사로 배송하는 원리와 비슷합니다. ‘객체의 직렬화’란 객체를 스트림을 통해 파일에 저장하거나 네트워크로 전송하려고 바이트 형태로 만드는 것입니다. 반대로 전송받은 직렬화된 객체를 다시 원래의 객체로 만드는 것을 ‘역직렬화’라 합니다.

그림 7-2 객체 → 직렬화/역직렬화 → 객체



7.2 직렬화 클래스 만들기

객체를 직렬화/역직렬화하기 위해서는 우선 객체 자체가 직렬화가 가능한 클래스로부터 생성된 객체여야 합니다. 하트 모양의 빵을 만들려면 하트 모양의 빵틀에

반죽을 넣어야 하는 것처럼 직렬화 클래스에서 나온 객체여야 직렬화될 수 있습니다. 직렬화 클래스는 `java.io.Serializable` 인터페이스를 받아서 만들 수 있습니다.

[직렬화된 클래스를 만드는 방법]

```
class StuInfo implements java.io.Serializable{
```

공통 기능을 추출한 `java.io` 패키지의 `Serializable`이라는 인터페이스를 받는

객체를 정의/생성하는 툴인 `StuInfo`이라는 클래스를 만든다

...

```
}
```

유의할 점은 클래스 내 정의된 생성자나 메서드는 직렬화 대상이 되지 않고 데이터, 즉 필드만 직렬화 대상이 된다는 점입니다. 직렬화 클래스 예제를 살펴봅시다.

[코드 7-1] 직렬화 클래스 예제

```
import java.io.*;
```

`java.util` 패키지의 모든 클래스를 여기서도 사용한다

```
class StuInfo implements java.io.Serializable{
```

공통 기능을 추출한 `java.io` 패키지의 `Serializable`이라는 인터페이스를 받는

객체를 정의/생성하는 툴인 `StuInfo`이라는 클래스를 만든다

`String name;`

문자열 값을 담는 `name`이라는 변수를 선언한다

필드

`int age;`

정수 값을 담는 `age`라는 변수를 선언한다

직렬화 ○

`StuInfo(String name, int age){`

문자열 값을 담는 `name` 파라미터와 정수 값을 담는 `age` 파라미터 변수를 인자로 받는

`StuInfo` 클래스의 객체 필드를 초기 설정하는 생성자를 만든다

생성자

직렬화 ×

`this.name = name;`

이 클래스로 생성된 객체 안의 `name` 변수에 `name` 파라미터 변수를 저장한다

`this.age = age;`

이 클래스로 생성된 객체 안의 `age` 변수에 `age` 파라미터 변수를 저장한다

```
}
```

```

void printInfo(){
    리턴 값이 없는 printInfo라는 메서드를 선언한다
    메서드

    System.out.println("name:"+name);
    "name:" 다음에 name 변수 값을 시스템 콘솔에 출력하고 한 줄 띄운다
    직렬화 X

    System.out.println("age:"+age);
    "age:" 다음에 age 변수 값을 시스템 콘솔에 출력하고 한 줄 띄운다

}
}

```

연습 7-1

[연습 3-4]에서 작성한 Account 클래스를 직렬화 클래스로 만들어 보세요.

7.3 객체를 직렬화/역직렬화하는 방법

객체를 직렬화 또는 역직렬화하기 위해서는 자바의 JDK 라이브러리 중 `java.io.ObjectOutputStream`과 `java.io.ObjectInputStream`이라는 패키지를 사용합니다. `java.io` 아래의 모든 패키지를 의미하는 `java.io.*`를 임포트하면 해당 프로그램에서 두 패키지의 클래스들을 사용할 수 있습니다. 객체를 직렬화하는 방법은 다음과 같습니다

[객체를 직렬화하는 방법]

- (1) 출력을 위해 `OutputStream` 클래스를 상속받은
클래스의 객체를 생성한다.
`-> FileOutputStream fos = new FileOutputStream("a.dat");`
`*여기서는 바이너리 파일을 쓰기 위한 FileOutputStream 클래스를 생성하였다.`
- (2) 객체를 직렬화해 쓰는 클래스인
 `ObjectOutputStream`의 객체를 생성하고
파라미터에 출력 스트림 객체를 넣어
직렬화하는 스트림을 만든다.
`-> FileOutputStream fos = new FileOutputStream("a.dat");`
`ObjectOutputStream out = new ObjectOutputStream(fos);`
- (3) 스트림을 통해 출력할 객체를 생성한다.
`-> FileOutputStream fos = new FileOutputStream("a.dat");`

```
ObjectOutputStream out =  
    new ObjectOutputStream(fos);  
StuInfo s1 = new StuInfo("hin",27);
```

(4) writeObject 메서드를 사용하여

객체를 직렬화한 뒤 스트림으로 흘려 보낸다.

```
-> FileOutputStream fos =  
    new FileOutputStream("a.dat");  
ObjectOutputStream out =  
    new ObjectOutputStream(fos);  
StuInfo s1 = new StuInfo("hin",27);  
out.writeObject(s1);
```

(5) close 메서드를 사용하여 파일을 닫는다.

```
-> FileOutputStream fos =  
    new FileOutputStream("a.dat");  
ObjectOutputStream out =  
    new ObjectOutputStream(fos);  
StuInfo s1 = new StuInfo("hin",27);  
out.writeObject(s1);  
fos.close();  
out.close();
```

FileOutputStream fos = new FileOutputStream("a.dat");

바이트로 된 파일(바이너리 파일)을 사용하는 스트림 클래스의 객체를 담는 fos 변수에

“a.dat”란 파일을 쓰는 FileOutputStream 클래스의 객체를 새로 생성하여 저장한다

ObjectOutputStream out = new ObjectOutputStream(fos);

객체를 직렬화하여 쓰는 스트림 클래스의 객체를 담는 out 변수에

fos 변수에 직렬화 객체를 쓸 수 있게 ObjectOutputStream 클래스의 객체를 새로 생성하여 저장한다

StuInfo s1 = new StuInfo("hin",27);

StuInfo 클래스의 객체를 담는 s1이라는 변수를 선언하고

“hin”과 27로 초기화한 StuInfo 클래스의 객체를 새로 생성하여 저장한다

out.writeObject(s1);

out 안의 writeObject() 메서드를 호출하여 객체 s1을 직렬화하여 쓴다

fos.close();

fos 안의 close() 메서드를 호출하여 파일을 닫는다

out.close();

out 안의 close() 메서드를 호출하여 파일을 닫는다

파라미터로 객체를 넘기는 경우 new 연산자를 사용하여 바로 쓸 수도 있습

니다. 예를 들어, [객체를 직렬화하는 방법]에서 `ObjectOutputStream` 객체의 파라미터로 `FileOutputStream` 객체를 미리 생성하고 그 객체명을 파라미터로 주었던 것을(`ObjectOutputStream out = new ObjectOutputStream(fos)`), `ObjectOutputStream` 객체의 파라미터에 `new` 연산자를 사용하여 바로 적어도 됩니다(`ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream())`). 이를 응용한 방법으로 객체를 직렬화하여 파일에 저장하는 방법을 살펴보겠습니다.

[코드 7-2] 객체를 직렬화하여 파일에 저장하는 프로그램

```
import java.io.*;
```

java.util 패키지의 모든 클래스를 여기서도 사용한다

```
class SerialTest {
```

객체를 정의/생성하는 틀인 `SerialTest`라는 클래스를 만든다

```
public static void main(String[] args) {
```

모든 패키지와 클래스에서 접근 가능하고, 같은 클래스에서 생성된 객체들과 값을 공유하며, 리턴값이 없는 `main`이라는 함수를 선언하고, 문자열 값을 담는 `args`라는 배열을 인자로 받는다

```
try {
```

{ } 안을 시도해 보고 에러가 뜨면 Exception 메시지를 catch 블록으로 던진다

```
ObjectOutputStream out =
```

객체를 직렬화하여 쓰는 스트림 클래스의 객체를 담는 `out` 변수에

```
new ObjectOutputStream(new FileOutputStream("a.dat"));
```

직렬화 객체를 쓸 수 있게 "a.dat"란 파일을 쓰는 `FileOutputStream` 클래스의 객체를 새로 생성하고

이를 인자로 `ObjectOutputStream` 클래스의 객체를 새로 생성하여 저장한다

```
StuInfo s1 = new StuInfo("hin", 27);
```

`StuInfo` 클래스의 객체를 담는 `s1`이라는 변수를 선언하고

"hin"과 27로 초기화한 `StuInfo` 클래스의 객체를 새로 생성하여 저장한다

```
StuInfo s2 = new StuInfo("John", 25);
```

`StuInfo` 클래스의 객체를 담는 `s2`라는 변수를 선언하고

"John"과 25로 초기화한 `StuInfo` 클래스의 객체를 새로 생성하여 저장한다

```
out.writeObject(s1);
```

`out` 안의 `writeObject()` 메서드를 호출하여 객체 `s1`을 직렬화하여 쓴다

```

out.writeObject(s2);
    out 안의 writeObject() 메서드를 호출하여 객체 s2를 직렬화하여 쓴다

out.close();
    out 안의 close() 메서드를 호출하여 파일을 닫는다

} catch (Exception e) {
    Exception 메시지를 받은 경우 {} 안을 수행한다 모든 에러 타입 e 변수

System.out.println(e);
    e 변수 값을 시스템 콘솔에 출력하고 한 줄 띄운다

}
}
}

```

실행 결과

실행 결과에는 아무것도 나오지 않습니다. 그러나 저장된 a.dat 파일을 열어보면 [그림 7-3]처럼 사람이 읽을 수 없는 바이트 형식으로 객체가 표현되어 있음을 확인할 수 있습니다.

그림 7-3 코드 7-2에서 파일에 저장한 직렬화된 객체



연습 7-2

다음과 같은 정보를 가진 Account 클래스의 객체를 직렬화하여 account2.dat 파일에 저장해보세요.

계좌번호: 1002-47-2881299, 예금주: 김성현, 잔액: 240000원
 계좌번호: 1002-21-5440400, 예금주: 이시원, 잔액: 56000원
 계좌번호: 1002-64-1090347, 예금주: 홍진표, 잔액: 9400000원

바이트로 표현된 객체를 다시 원래대로 읽어 오기 위해서는 객체를 역직렬화하는 과정이 필요합니다. 직렬화된 객체를 역직렬화를 하여 원래대로 만드는 방법은 다음과 같습니다.

[객체를 역직렬화하는 방법]

- (1) 입력을 위해 `InputStream` 클래스를 상속받은
클래스의 객체를 생성한다.

*여기서는 바이너리 파일을 쓰기 위한 `FileInputStream` 클래스를 생성하였다.

- (2) 객체를 역직렬화하는 클래스인

`ObjectInputStream`의 객체를 생성하고
파라미터에 입력 스트림 객체를 넣어
역직렬화하는 스트림을 만든다.

- (3) `readObject` 메서드를 사용하여

스트림에서 객체를 역직렬화해 읽어 오고
이 값을 미리 선언한 알맞은 객체 변수에 저장한다.

- (4) `close` 메서드를 사용하여 파일을 닫는다.

-> `FileInputStream fis =
new FileInputStream("a.dat");`

-> `FileInputStream fis =
new FileInputStream("a.dat");`

`ObjectInputStream in =
new ObjectInputStream(fis);`

-> `FileInputStream fos =
new FileInputStream("a.dat");`

`ObjectInputStream in =
new ObjectInputStream(fis);`

`StuInfo s1 = (StuInfo) in.readObject();`

-> `FileInputStream fis =
new FileInputStream("a.dat");`

`ObjectInputStream in =
new ObjectInputStream(fis);`

`StuInfo s1 = (StuInfo) in.readObject();`

`fis.close();`

`in.close();`

`FileInputStream fis = new FileInputStream("a.dat");`

바이트로 된 파일(바이너리 파일)을 사용하는 스트림 클래스의 객체를 담는 `fis` 변수에

"a.dat" 파일을 읽는 `FileInputStream` 클래스의 객체를 새로 생성하여 저장한다

`ObjectInputStream in = new ObjectInputStream(fis);`

객체를 역직렬화하여 읽는 스트림 클래스의 객체를 담는 `in` 변수에

`fis` 변수에 직렬화 객체를 쓸 수 있게 `ObjectInputStream` 클래스의 객체를 새로 생성하여 저장한다

`StuInfo s1 = (StuInfo) in.readObject();`

`StuInfo` 클래스의 객체를 담는 `s1`이라는 변수를 선언하고

`in` 안의 `readObject()` 메서드를 호출하여 객체를 역직렬화해 읽어 온 값을 `StuInfo` 형으로 변환하여 저장한다

`fis.close();`

`fis` 안의 `close()` 메서드를 호출하여 파일을 닫는다

`in.close();`

`in` 안의 `close()` 메서드를 호출하여 파일을 닫는다

다음은 [코드 7-2]에서 파일에 바이트로 저장한 직렬화된 객체를 역직렬화해 읽어 오는 프로그램입니다. 6.3장에서 FileReader 클래스를 이용하여 파일을 읽어 올 때는 더는 읽을 데이터가 없으면 -1을 리턴하여 while문을 멈췄습니다. 반면에, ObjectInputStream 클래스를 이용하여 파일에서 객체를 역직렬화해 읽어 올 때는 더는 읽을 데이터가 없으면 EOFException이 발생하게 됩니다. 따라서 try~catch 구문을 이용하여 EOFException을 잡는 catch 블록을 만들고, 파일을 끝까지 읽었을 때 어떤 처리를 할 것인지 기술해 줍니다.

[코드 7-3] 파일에서 객체를 역직렬화해 읽어 오는 프로그램

```
import java.io.*;
```

java.io 패키지의 모든 클래스를 여기서 사용한다

```
class DeSerialTest {
```

객체를 정의/생성하는 틀인 DeSerialTest라는 클래스를 만든다

```
public static void main(String[] args) {
```

모든 패키지와 클래스에서 접근 가능하고, 같은 클래스에서 생성된 객체들과 값을 공유하며, 리턴값이 없는 main이라는 함수를 선언하고, 문자열 값을 담는 args라는 배열을 인자로 받는다

```
try {
```

{ } 안을 시도해 보고 에러가 뜨면 Exception 메시지를 catch 블록으로 던진다

```
ObjectInputStream in =
```

객체를 역직렬화하여 읽는 스트림 클래스의 객체를 담는 in 변수에 저장한다

```
new ObjectInputStream(new FileInputStream("a.dat"));
```

직렬화 객체를 쓸 수 있게 "a.dat"란 파일을 쓰는 FileInputStream 클래스의 객체를 새로 생성하고 이를 인자로 ObjectInputStream 클래스의 객체를 새로 생성하여 저장한다

```
StuInfo s;
```

StuInfo 클래스의 객체를 담는 s이라는 변수를 선언한다

```
while((s=(StuInfo)in.readObject())!=null)
```

in 안의 readObject() 메서드를 호출하여 객체를 역직렬화해 읽어 온 값을

StuInfo 형으로 변환하여 저장한 s 변수가 null이 아닌 동안 {} 안을 반복해라

```
s.printInfo();
```

s 안의 printInfo(): 메서드를 호출한다

```
in.close();
```

in 안의 close() 메서드를 호출하여 파일을 닫는다

```
} catch (EOFException e) {
```

Exception 메시지를 받은 경우 {} 안을 수행한다 파일의 끝이라고 알려주는 에러 타입 e 변수

```
System.out.println("더는 읽을 데이터가 없습니다");
```

"더는 읽을 데이터가 없습니다"를 시스템 콘솔에 출력하고 한 줄 띄운다

```
} catch (Exception e) {
```

Exception 메시지를 받은 경우 {} 안을 수행한다 모든 에러 타입 e 변수

```
System.out.println(e);
```

e 변수 값은 시스템 콘솔에 출력하고 한 줄 띄운다

```
}
```

```
}
```

실행 결과

```
name:hin  
age:27  
name:John  
age:25  
더는 읽을 데이터가 없습니다
```

NOTE null

우리가 흔히 아무것도 없는 상태를 표현하기 위한 말로 '무'라는 말을 사용하지요. "무(無)에서 유(有)를 창조한다"고 할 때처럼 말입니다. 자바에서도 "값이 아무것도 없다"는 표현하는 말로 'null'을 사용합니다. 이는 숫자 0과는 다른 개념입니다. null은 숫자도 문자도 아닌 그냥 '아무것도 없는 상태'를 지칭하기 때문입니다.

연습 7-3

[연습 7-2]에서 작성한 account2.dat 파일을 읽어 Account 클래스의 객체들을 출력해보세요.

출력 예시

```
[계좌번호:1002-47-2881299] 예금주:잔액:2400000
```

```
[계좌번호:1002-21-5440400] 예금주:잔액:560000
```

```
[계좌번호:1002-64-1090347] 예금주:잔액:9400000
```

```
더 이상 읽을 데이터가 없습니다
```

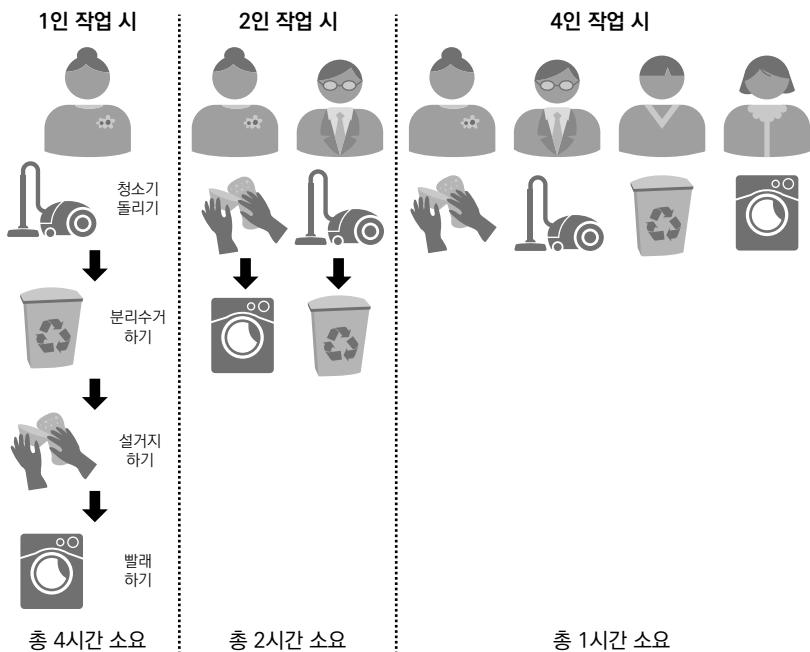
멀티스레드

스마트폰을 이용할 때, 노래를 들으면서 동시에 문자를 보내고 인터넷 검색도 할 수 있습니다. 그런데 이처럼 컴퓨터에서 동시에 여러 가지 작업을 하려면 내부적으로 멀티스레드 프로그래밍을 해주어야 합니다. 8장에서는 이 멀티스레드에 대한 전반적인 내용을 다룹니다. 8.1장에서는 멀티스레드가 무엇인지 공부하고, 8.2장에서는 멀티스레드를 작성하는 방법을 공부합니다. 마지막으로 8.3장에서 여러 개의 스레드가 있을 때 이들이 데이터를 주고받는 통신 방법에 대해 알아보겠습니다.

8.1 멀티스레드란

어머니가 집안일을 할 때를 생각해 봅시다. 청소기도 돌리고, 분리 수거도 하고, 다림질도 하고, 밀린 빨래도 해야 하죠. 어머니가 혼자서 한 가지 일을 끝내고 다른 일을 하는 식으로 하다 보면 시간이 오래 걸립니다. 한 가지 일에 1시간씩만 계산해도 4시간이 걸립니다. 그런데 일찍 퇴근한 아버지가 어머니의 집안일을 도와주게 되었습니다. 어머니가 설거지를 하는 동안 아버지는 청소기를 돌리면 그만큼 집안일을 끝내는 시간이 단축되겠죠? 아들, 딸도 집에 일찍 들어와 집안일을 돋는다고 하면 총 4명이 동시에 일을 하게 되어 집안일을 더욱 빨리 끝낼 수 있을 것입니다.

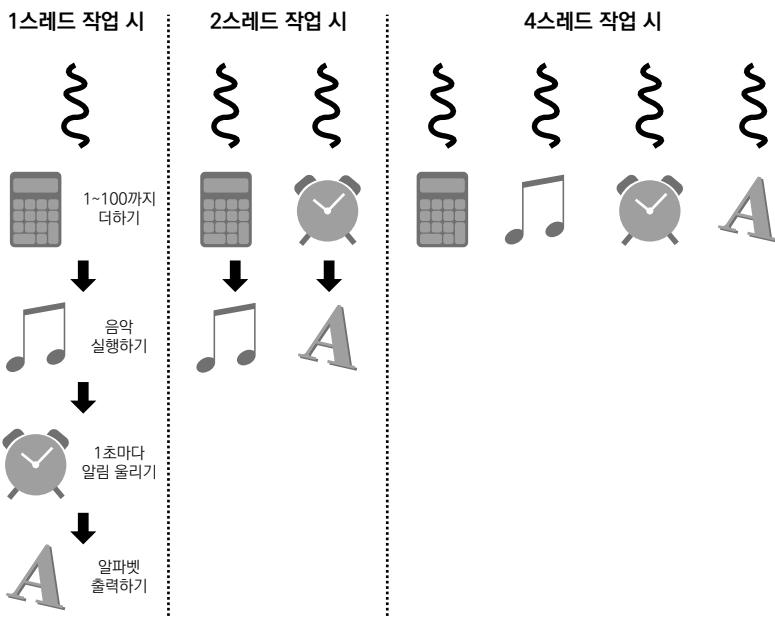
그림 8-1 집안일을 분담할 때의 시간 단축



이와 같은 원리가 스레드에서도 똑같이 적용됩니다. ‘스레드 Thread’란 자바 프로그램의 명령문을 위에서부터 아래로 차례대로 실행해 주는 일꾼입니다. 영어로 ‘Thread(스레드)’는 ‘실’이란 뜻인데, 자바에서 스레드는 프로그램의 명령문을 실행하는 흐름이 실처럼 연결된다고 하여 붙여진 이름입니다.

프로그램에 하나의 스레드만 있는 것을 싱글스레드 Single-thread라고 하는데, 이 경우에는 한번에 하나의 작업만 가능합니다. 반면, 멀티스레드 Multi-thread는 한 프로그램 내에 여러 스레드를 사용하는 것이어서 동시에 여러 가지 작업을 할 수 있습니다. 예를 들어, 컴퓨터가 처리해야 하는 일이 1부터 100까지 더하기, 음악 실행하기, 1초에 한 번씩 알림 올리기, 알파벳 출력하기가 있다면 싱글스레드 프로그램에서는 이 4가지 일을 차례로 하나씩 수행해야 합니다. 하지만 4개의 스레드를 만들어서 프로그램 내에서 작업을 할 일꾼을 늘려 놓으면, 각 스레드가 하나의 일을 맡아서 수행하면 되기 때문에 동시에 4가지 작업을 할 수 있습니다.

그림 8-2 멀티스레드의 동시 작업



이처럼 멀티스레드란 한 프로그램 내에서 둘 이상의 스레드가 작업을 하는 것을 말합니다. 다음 장에서는 멀티스레드를 구현하는 방법에 대해서 공부하겠습니다.

8.2 멀티스레드 작성 방법

프로그램에서 `main()` 메서드를 실행하는 스레드 외에 추가적인 스레드를 사용하여 다른 작업을 할 수 있는 멀티스레드를 구현하려면, 우선 `Thread` 클래스를 상속받은 클래스를 선언해야 합니다. 그리고 `Thread` 클래스의 `run()` 메서드를 오버라이딩하여 그 안에 해당 스레드가 어떤 일을 해야 하는지 명령문을 작성합니다. 그 후 `main()` 메서드에서 클래스의 객체를 생성한 후 `start()` 메서드를 이용하여 스레드를 실행하면 그 시점부터 한 프로그램 내에서 두 개의 스레드가 동시에 동작하게 됩니다.

(1) Thread 클래스를 상속받는다.

→ `class Mthread extends Thread{
}`

(2) Thread 클래스의 run() 메서드를 오버라이딩하여
그 안에 처리할 일을 적는다.

→ `class Mthread extends Thread{
 public void run(){
 처리할 명령
 }
}`

(3) main() 메서드에서 해당 클래스의 객체를 생성하고 → class Mthread extends Thread{
start() 메서드로 스레드를 실행시킨다.

`public void run(){
 처리할 명령
}
public static void main(String[] args){
 Mthread t = new Mthread();
 t.start();
}`

class Mthread extends Thread{

Thread라는 클래스를 확장(상속)하여 객체를 정의/생성하는 데 사용되는 Mthread라는 클래스를 만든다

public void run(){

모든 패키지와 클래스에서 접근 가능하고, 리턴값이 없는 run이라는 메서드를 선언한다

처리할 명령

}

public static void main(String[] args){

모든 패키지와 클래스에서 접근 가능하고, 같은 클래스에서 생성된 객체들과 값을 공유하며, 리턴값이 없는 main이라는 함수를 선언하고, 문자열 값을 담는 args라는 배열을 인자로 받는다

Mthread t = new Mthread();

Mthread 클래스의 객체를 담는 t라는 변수에 Mthread 클래스의 객체를 새로 생성하여 저장한다

t.start();

t 안의 start() 메서드를 호출하여 스레드를 실행한다

}

}

이 방법을 응용하여 1부터 100까지 더하기, 1초마다 알림 울리기, 알파벳 출력하기 등등을 동시에 작업하는 멀티스레드 프로그램 예제를 살펴보겠습니다.

[코드 8-1] 멀티스레드 예제

class Athread extends Thread{

Thread라는 클래스를 확장(상속)하여 객체를 정의/생성하는 데 Athread라는 클래스를 만든다

public void run(){

모든 패키지와 클래스에서 접근 가능하고, 리턴값이 없는 run이라는 메서드를 선언한다

int i = 0;

정수 값을 담는 i라는 이름의 변수를 선언하고 0을 저장한다

try {

{ } 안을 시도해 보고 에러가 뜨면 Exception 메시지를 catch 블록으로 던진다

while(true){

1초마다 알림 울리기

true일 동안 {} 안을 반복해라(while(true) 무한반복)

Thread.sleep(1000);

Thread 클래스 안의 sleep() 메서드를 호출하여 1000밀리 초(1초) 간 이 스레드를 정지한다.

i++;

변수 i의 값을 1씩 증가시켜라

if(i==10) {

변수 i의 값이 10과 같으면 {} 안을 처리해라

System.out.println("10초!");

"10초!"를 시스템 콘솔에 출력하고 한 줄 띄운다

break;

반복문을 멈춘다(= while문을 중단한다)

}else{

그렇지 않으면

System.out.print(i+"초");

i 변수 값 다음에 "초"를 시스템 콘솔에 출력한다

}

}

} catch (Exception e) {

Exception 메시지를 받은 경우 {} 안을 수행한다 모든 에러 타입 e 변수

```
        System.out.println(e);  
    }  
}  
}  
}
```

class Mthread extends Thread{

Thread라는 클래스를 확장(상속)하여 객체를 정의/생성하는 데 사용하는 Mthread라는 클래스를 만든다

public void run(){

모든 패키지와 클래스에서 접근 가능하고, 리턴값이 없는 run이라는 메서드를 선언한다

```
for(char alp='A'; alp<='Z'; alp++){
```

알파벳 출력하기

문자 값을 담는 alp라는 변수에 A를 저장하고

alp의 값을 1씩 증가시키면서 alp가 Z보다 작거나 같을 동안 {} 안을 반복해라

```
System.out.print(alp+" ");
```

alp 변수의 값+“공백”을 시스템 콘솔에 출력한다

try {

{ } 안을 시도해 보고 예외가 뜨면 Exception 메시지를 catch 블록으로 던진다

```
Thread.sleep(500);
```

Thread 클래스 안의 sleep() 메서드를 호출하여 500밀리 초(0.5초)간 이 스레드를 정지한다.

} catch (Exception e) {

Exception 메시지를 받은 경우 {} 안을 수행한다 모든 예외 타입 e 변수

```
System.out.println(e);
```

e 변수 값을 시스템 콘솔에 출력하고 한 줄 띄운다

}

}

}

public static void main(String[] args) {

모든 패키지와 클래스에서 접근 가능하고, 같은 클래스에서 생성된 객체들과 값을 공유하며, 리턴값이 없는

main이라는 함수를 선언하고, 문자열 값을 담는 args라는 배열을 인자로 받는다

Mthread mt = new Mthread();

Mthread 클래스의 객체를 담는 mt라는 변수를 선언하고 Mthread 클래스의 객체를 새로 생성하여 저장한다

Athread at = new Athread();

Athread 클래스의 객체를 담는 at라는 변수를 선언하고 Athread 클래스의 객체를 새로 생성하여 저장한다

mt.start();

mt 인의 start() 메서드를 호출하여 스레드를 실행한다

at.start();

at 인의 start() 메서드를 호출하여 스레드를 실행한다

int sum = 0;

1~100까지 더하기

정수 값을 담는 sum이라는 이름의 변수를 선언하고 0을 저장한다

for(int i=1; i<=100; i++)

문자 값을 담는 i라는 변수에 1을 대입하고

i의 값을 1씩 증가시키면서 i가 100보다 작거나 같을 동안 반복해라

sum += i;

sum 변수에 i 값을 더하고 그 값을 sum 변수에 저장한다

System.out.println("1부터 100까지의 합=" +sum);

"1부터 100까지의 합=" 다음에 sum 변수 값을 시스템 콘솔에 출력하고 한 줄 띄운다

```
}
```

```
}
```

실행 결과 ※ 어느 스레드가 먼저 실행되느냐에 따라 실행 결과가 바뀔 수 있습니다.

1부터 100까지의 합=5050

A B C 1초D E 2초F G 3초H 4초I J K 5초L 6초M N O 7초P Q 8초R S 9초T U 10초!
V W X Y Z

[코드 8-1]에서는 총 3개의 스레드가 각자 업무를 맡아 작업하고 있습니다. Mthread는 A~Z까지 0.5초 간격으로 출력합니다. Athread는 1초 간격으로 시간을 알려 주고 10초가 되면 카운트를 멈춥니다. main() 메서드의 스레드는 start() 메서드로 Mthread와 Athread의 객체들을 실행시킨 후 1부터 100까지의 합을 계산해 출력합니다. 실행 결과를 보면 1부터 100까지의 합을 계산하는 것은 얼마 걸리지 않기 때문에 제일 먼저 출력되고, A~Z 출력과 초 단위로 시간을 알려 주는 것이 뒤섞여 출력됨을 알 수 있습니다. 이 경우 프로그램이 종료되기 까지 대략 13초가 걸립니다.

이 3개의 업무를 하나의 스레드에서 처리한다면, 다음과 같은 코드와 결과값이 나오고 프로그램이 종료되기까지 대략 24초가 걸립니다.

[코드 8-2] 코드 8-1을 싱글스레드로 구현한 프로그램

```
class SingleThread {
    public static void main(String[] args) {

        int i=0;
        for(char alp='A'; alp<='Z'; alp++){
            System.out.print(alp+" ");
            try {
                Thread.sleep(500);
            } catch (Exception e) {
                System.out.println(e);
            }
        }

        try {
            while(true){
                Thread.sleep(1000);
                i++;
                if(i==10) {
                    System.out.println("10초!");
                    break;
                }else{
                    System.out.print(i+"초");
                }
            }
        } catch (Exception e) {
            System.out.println(e);
        }

        int sum=0;
        for(i=1; i<=100; i++)
            sum+=i;

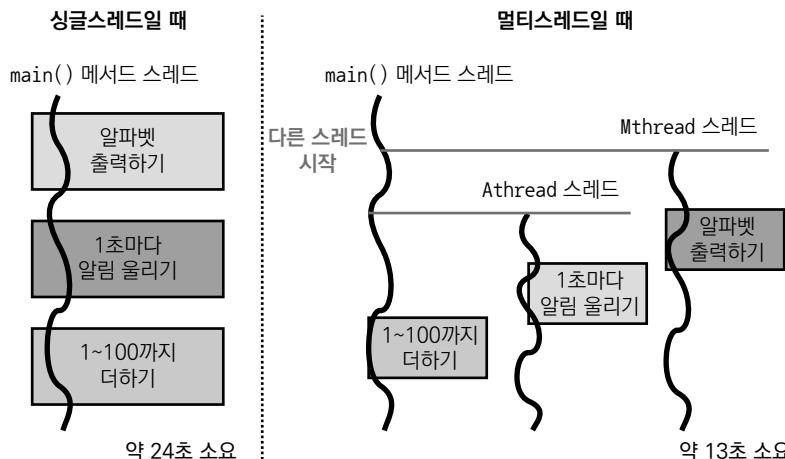
        System.out.println("1부터 100까지의 합="+sum);
    }
}
```

실행 결과

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 1초2초3초4초5초6초7초8초9초10초!
1부터 100까지의 합=5050

[코드 8-1]과 [코드 8-2]에 나타난 싱글스레드와 멀티스레드를 간단하게 비교해 보면 다음과 같습니다.

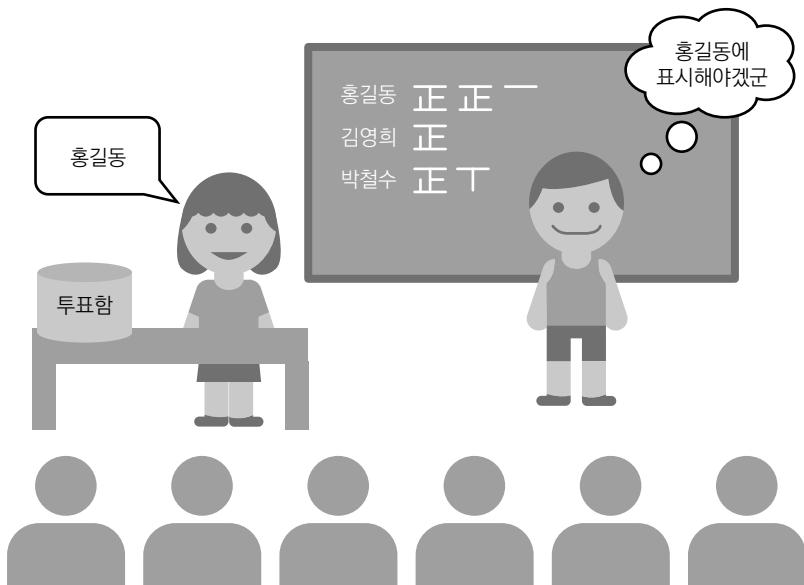
그림 8-3 [코드 8-1]과 [코드 8-2]의 싱글스레드와 멀티스레드 비교



8.3 스레드 간의 통신

사람들이 어떤 일을 할 때를 생각해 보면, 작업 내내 다른 사람과 상관없이 자기 일만 해도 되는 경우도 있지만 때로는 다른 사람과 정보를 주고받아야만 일이 진행되는 경우도 있습니다. 예를 들면, 나는 그림을 그리고 상대방은 노래를 부르는 경우는 정보를 주고받아야 할 일이 크게 없겠지요. 하지만 학교에서 반장 선거할 때를 생각해 봅시다. 한 친구는 접힌 투표지를 펼쳐 후보자의 이름을 발표하고, 다른 한 친구는 그 발표를 들은 후 칠판에 표시하는 등 서로 정보를 주고받아야만 일이 진행이 됩니다.

그림 8-4 반장 선거에서의 업무 분담

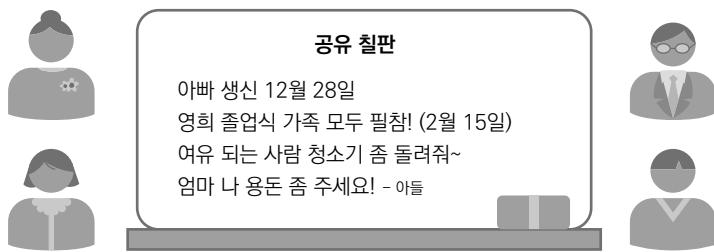


이와 같이 자바 프로그래밍에서도 여러 개의 스레드가 서로 다른 업무를 독립적으로 처리하는 경우, 서로 다른 스레드가 데이터를 주고받아서 업무를 처리하는 경우가 있습니다. 후자의 경우에 스레드끼리 데이터를 주고받는 것을 “스레드가 통신한다”라고 말합니다.

말, 문서 등 사람들이 정보를 주고받는 방법이 여러 개인 것처럼, 스레드 간에 서로 데이터를 주고받는 방법 역시 여러 가지가 있습니다. 이 중에서도 서로 다른 스레드들이 특정 메모리 영역을 같이 공유해 사용하는 방법인 ‘공유 메모리^{Shared Memory}’에 대해 공부해 보겠습니다.

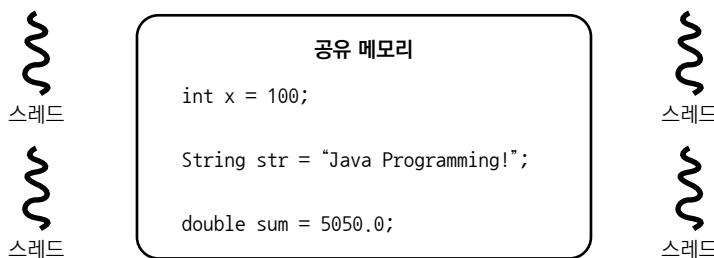
공유 메모리를 현실 세계에 빗대어 생각하면 공유 칠판으로 볼 수 있습니다. 가족끼리 공유해야 하는 내용을 적어 놓은 화이트보드를 거실에 걸어 놓는다고 생각해 봅시다.

그림 8-5 가족의 공유 칠판



이렇게 거실에 가족들이 모두 공유할 수 있는 화이트보드를 걸어 놓으면, 가족 중 누군가에게 메시지를 남기거나 공유할 내용을 적어 두는 용도로 사용할 수 있겠죠. 가족이라면 누구나 내용을 추가하거나 수정할 수 있습니다. 자바의 공유 메모리도 스레드들이 공유할 내용을 적어 두고, 또 적혀 있는 내용을 읽을 수 있는 칠판과 같은 역할을 합니다.

그림 8-6 스레드의 공유 메모리



공유 메모리를 선언하는 방법은 다음과 같습니다.

[공유 메모리를 선언하는 방법]

- (1) 공유할 데이터를 저장해 둘 수 있는

공유 메모리 클래스를 만든다.

class SharedMemory

데이터(필드)	
x	0

class SharedMemory{

int x = 0;

}

- (2) Thread 클래스를 상속받은 클래스들의 필드로 공유 메모리 클래스 객체를 선언하고

class PlusThread extends Thread {
 SharedMemory shmem;

run() 메서드를 오버라이딩하여
그 안에 처리할 일을 적는다.

class PlusThread
extends Thread



class MinusThread
extends Thread



public void run(){

처리할 내용

}

class MinusThread extends Thread {

SharedMemory shmem;

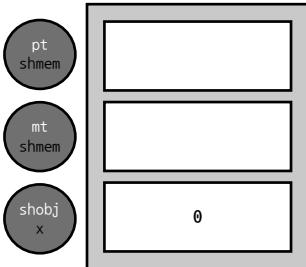
public void run(){

처리할 내용

}

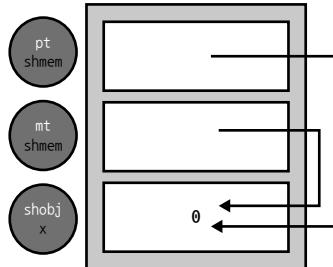
- (3) main() 메서드에서 스레드 클래스들과
공유 메모리 클래스의 객체를 생성한다.

메모리



- (4) 스레드 객체들의 공유 메모리 변수에
main() 메서드에서 선언한
공유 메모리 클래스의 객체를 저장한다.

메모리



- (5) start() 메서드로 스레드를 실행시킨다.

class SharedMemoryTest{

public static void main(String[] args){

PlusThread pt = new PlusThread();

MinusThread mt = new MinusThread();

SharedMemory shobj = new SharedMemory();

pt.shmem = shobj;

mt.shmem = shobj;

pt.start();

mt.start();

}

공유 메모리를 사용하여 컴퓨터와 간단한 게임을 하는 프로그램 코드를 살펴보겠습니다. 이 게임에서 컴퓨터는 0.3초마다 1씩 값을 증가시키고, 게임 참가자는 엔터를 누를 때마다 1씩 값을 감소시킬 수 있습니다. 게임 참가자가 엔터를 빨리 눌러서 값을 0으로 만들면 이깁니다.

[코드 8-3] 공유 메모리를 이용한 0 만들기 게임

import java.util.Scanner;

java.util.Scanner 패키지의 클래스들을 여기서 사용한다

class SharedMemoryTest{

객체를 정의/생성하는 툴인 SharedMemoryTest라는 클래스를 만든다

public static void main(String[] args) {

모든 패키지와 클래스에서 접근 가능하고, 같은 클래스에서 생성된 객체들과 값을 공유하며, 리턴값이 없는 main이라는 함수를 선언하고, 문자열 값을 담는 args라는 배열을 인자로 받는다

PlusThread pt = new PlusThread();

PlusThread라는 클래스의 객체를 담는 pt라는 변수를 선언하고

PlusThread 클래스의 객체를 새로 생성하여 저장한다

MinusThread mt = new MinusThread();

MinusThread라는 클래스의 객체를 담는 mt라는 변수를 선언하고

MinusThread 클래스의 객체를 새로 생성하여 저장한다

SharedMemory shobj = new SharedMemory();

SharedMemory라는 클래스의 객체를 담는 shobj라는 변수를 선언하고

SharedMemory 클래스의 객체를 새로 생성하여 저장한다

pt.shmem = shobj;

pt 안의 shmem이라는 변수에 shobj를 저장한다

mt.shmem = shobj;

mt 안의 shmem이라는 변수에 shobj를 저장한다

System.out.println("Press Enter until x is 0");

"Press Enter until x is 0"를 시스템 콘솔에 출력하고 한 줄 띄운다

pt.start();

pt 안의 start() 메서드를 호출하여 스레드를 실행한다

mt.start();

mt 안의 start() 메서드를 호출하여 스레드를 실행한다

```
}
```

class SharedMemory{

객체를 정의/생성하는 툴인 SharedMemory라는 클래스를 만든다

```
int x = 5;
```

정수 값을 담는 x라는 변수를 선언하고 5를 저장한다

```
boolean isQuit = false;
```

true나 false 둘 중 한 값을 담는 isQuit라는 변수를 선언하고 false를 저장한다

```
}
```

class PlusThread extends Thread{

Thread라는 클래스를 확장(상속)하여 객체를 정의/생성하는 툴인 PlusThread라는 클래스를 만든다

```
SharedMemory shmem;
```

SharedMemory라는 클래스의 객체를 담는 shmem이라는 변수를 선언한다

```
public void run(){
```

모든 패키지와 클래스에서 접근 가능하고, 리턴값이 없는 run이라는 메서드를 선언한다

```
try {
```

{ } 안을 시도해 보고 예외가 뜨면 Exception 메시지를 catch 블록으로 던진다

```
while(!shmem.isQuit){
```

shmem의 isQuit 변수 값이 true가 아닌(즉, false인) 동안 {} 안을 반복해라

```
System.out.println("x= "+shmem.x);
```

"x =" 다음에 shmem 객체 변수의 x 값을 시스템 콘솔에 출력하고 한 줄 띄운다

```
Thread.sleep(300);
```

Thread 클래스 안의 sleep() 메서드를 호출하여 300밀리 초(0.3초) 간 이 스레드를 정지한다.

```
shmem.x++;
```

shmem의 변수 x의 값을 1씩 증가시킨다

```
}
```

```
} catch (Exception e) {
```

Exception 메시지를 받은 경우 {} 안을 수행한다 모든 예외 타입 e 변수

```
System.out.println(e);
```

e 변수 값을 시스템 콘솔에 출력하고 한 줄 띄운다

```

    }
}

}

class MinusThread extends Thread{
    Thread라는 클래스를 확장(상속)하여 객체를 정의/생성하는 툴인 MinusThread라는 클래스를 만든다

    SharedMemory shmem;
    SharedMemory라는 클래스의 객체를 담는 shmem이라는 변수를 선언한다

    public void run(){
        모든 패키지와 클래스에서 접근 가능하고, 리턴값이 없는 run이라는 메서드를 선언한다

        Scanner sc = new Scanner(System.in);
        입력 클래스의 객체를 담는 sc라는 변수에, 키보드에서 입력받는

        Scanner 클래스의 객체를 새로 생성하여 저장한다

        while(shmem.x!=0){
            shmem의 x 변수 값이 0이 아닌 동안 [] 안을 반복해라

            sc.nextLine();
            sc의 nextLine() 메서드를 호출하여 문자열을 읽어 온다

            shmem.x--;
            shmem의 변수 x의 값을 1씩 감소시킨다

        }

        shmem.isQuit = true;
        shmem의 isQuit 변수 값에 true를 저장한다

        System.out.println("Now x is 0. You win!");
        "Now x is 0. You win!"을 시스템 콘솔에 출력하고 한 줄 띄운다

    }

}

```

실행 결과 ※ 사용자가 엔터를 치는 속도에 따라 실행 결과가 다를 수 있습니다.

```

Press Enter until x is 0
x=5
x=5
x=3

```

x=2

Now x is 0. You win!

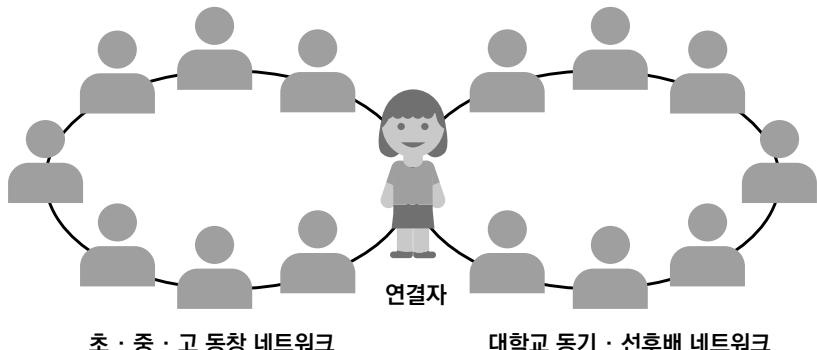
네트워크

인터넷으로 영화도 예매하고 강의도 보고 다양한 정보를 검색하기도 하는 현대 사회에서 네트워크는 매우 중요한 요소입니다. 자바로 프로그래밍을 할 때도 서로 다른 프로그램 또는 컴퓨터가 데이터를 주고받을 수 있게 하는 것이 중요한데, 이를 ‘네트워크 프로그래밍’이라 합니다. 9장에서는 이 네트워크 프로그래밍을 하는 방법을 알아봅니다. 9.1장에서 네트워크에 대한 기초 지식을 다루고, 9.2장에서는 자바에서 네트워크 프로그래밍을 하는 방법에 대해 공부하겠습니다.

9.1 네트워크의 기초

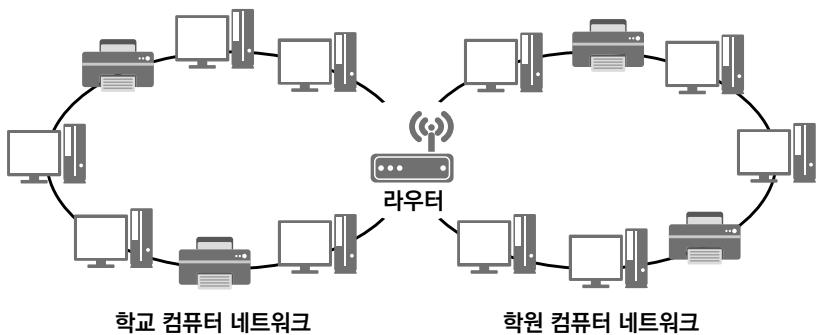
인적 네트워크라는 말을 들어보셨나요? 한국어로는 인맥 또는 인간 관계라고 하여 사람들과 관계를 형성하는 것을 말합니다. 예를 들어, 저의 경우 주로 학교나 학원, 회사 등 활동했던 곳을 토대로 인적 네트워크가 형성되었습니다. 초·중·고 동창 네트워크, 대학교 동기와 선후배 네트워크, 회사 동기와 선후배 네트워크 이렇게 말입니다. 어떤 경우에는 서로 다른 네트워크의 사람들인데 서로 아는 경우도 있고, 서로 다른 네트워크의 사람들을 연결해 주기도 합니다. 대학교 선배가 알고 보니 중학교 친구의 누나였던 적도 있고, 다른 인적 네트워크에 속해 있었지만 져를 통해 서로 알게 되어 연인이 된 친구들도 있었습니다.

그림 9-1 인적 네트워크



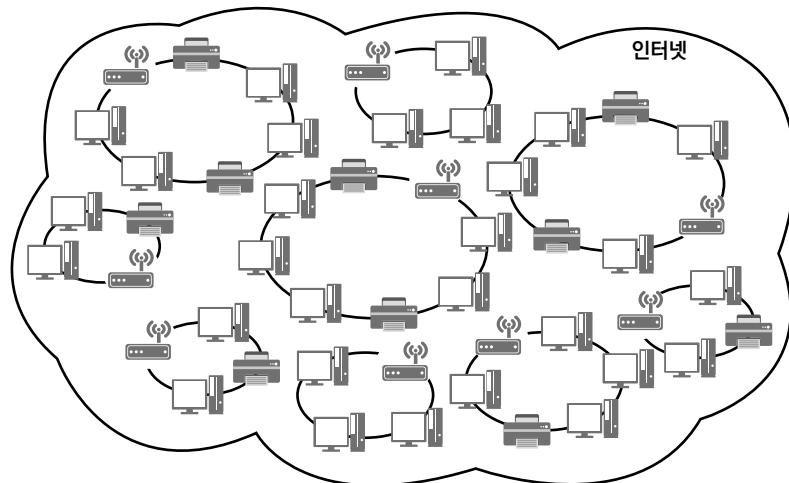
IT에서의 ‘네트워크’란 컴퓨터들 사이의 관계를 말합니다. 실제로 들여다보면 네트워크는 여러 기계 장치들이 ‘랜^{LAN}’이라는 물리적 선으로 연결되어 있는 것입니다. 네트워크 내에 있을 수 있는 기계 장치의 종류에는 컴퓨터뿐만 아니라 프린터나 라우터 등이 있습니다. ‘라우터’는 소개팅 주선자처럼 서로 다른 네트워크를 연결해 주는 연결고리 같은 장치입니다. 라우터는 데이터에 담긴 목적지 컴퓨터의 IP 주소를 읽고, 가장 적절한 네트워크로 전송하는 역할을 합니다.

그림 9-2 컴퓨터 네트워크



인터넷은 이러한 네트워크들이 모여 있는 것을 말합니다. 컴퓨터들이 연결된 네트워크가 전세계적으로 수없이 많은데 이를 연결해 둔 것이니, 인터넷은 전세계의 많은 컴퓨터가 연결된 집합체라고 할 수 있습니다.

그림 9-3 네트워크의 집합체인 인터넷



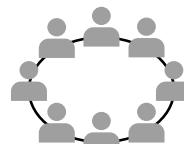
컴퓨터 네트워크를 인적 네트워크로 비교하면 컴퓨터는 한 개인, 네트워크는 집이나 동호회 또는 학교나 회사, 인터넷은 지구촌이라고 볼 수 있습니다.

그림 9-4 인적 네트워크와 컴퓨터 네트워크 비유

인적 네트워크



개인



집·동호회·학교·회사

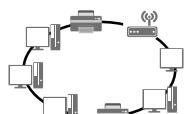


지구촌

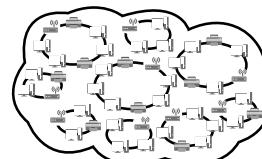
컴퓨터 네트워크



컴퓨터



네트워크



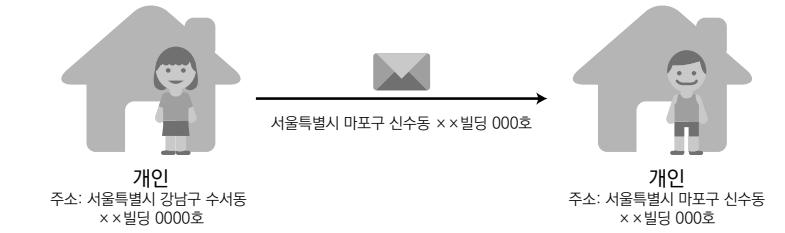
인터넷

9.1.1 IP 주소와 포트 번호

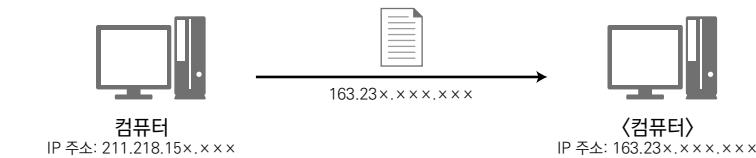
인터넷에 연결된 컴퓨터들끼리 서로 데이터를 주고받기 위해서는 각 컴퓨터를 구별할 수 있는 방법이 필요합니다. 그래서 각 컴퓨터는 인터넷 통신 서비스 회사에서 할당받은 ‘IP 주소’를 가지고 있습니다. 마치 집집마다 주소를 가지고 있는 것처럼 말이죠. 예를 들어, 어떤 친구가 저에게 우편을 보내려면 우편봉투에 제 주소인 ‘서울특별시 마포구 신수동 ××빌딩 000호’를 적어 보내면 되는 것처럼, 내 컴퓨터에서 인터넷에 연결된 전세계의 수많은 컴퓨터 중 한 컴퓨터로 데이터를 전송하고 싶으면 그 컴퓨터의 IP 주소(ex.163.23×.×××.×××)로 보내면 됩니다.

그림 9-5 주소로 데이터 전송하기

인적 네트워크



컴퓨터 네트워크



즉, ‘IP^{Internet Protocol} 주소’란 인터넷에 연결된 컴퓨터가 다른 컴퓨터와 구별될 수 있도록 갖는 고유한 주소입니다. IP 주소는 211.218.15×.×××와 같이 ‘.’을 기준으로 네 개의 도막으로 나누어진 최대 12자리의 번호로 구성되어 있습니다. ‘프로그램 및 파일 검색’ 창에 ‘cmd’를 입력한 후 엔터를 치면 윈도우의 명령어 프롬프트 창이 뜨는데, 여기에 ‘ipconfig’를 입력하면 내 컴퓨터의 IP 주소를 확인할 수 있습니다.

그림 9-6 윈도우 명령어 프롬프트 실행



그림 9-7 내 컴퓨터의 IP 주소 확인

A screenshot of a command prompt window titled 'cmd.exe'. The command 'ipconfig' is entered. The output shows network configuration for '무선 LAN 어댑터' and '이더넷 어댑터 이더넷'. A red box highlights the IPv4 주소 '192.168.1.1' under the '이더넷 어댑터 이더넷' section. Another red box highlights the same address with the text '내 컴퓨터의 IP 주소'.

```
C:\WINDOWS\system32>ipconfig
Windows IP 구성

무선 LAN 어댑터 로컬 영역 연결 11:
  미디어 상태 . . . . . : 미디어 연결 끊김
  연결별 DNS 접미사 . . . . . :

이더넷 어댑터 이더넷:
  연결별 DNS 접미사 . . . . . :
  IPv4-문제 IPv6 주소 . . . . . : fe80::1fd8%1
  IPv4 주소 . . . . . : 192.168.1.1
  서브넷 마스크 . . . . . : 255.255.255.0
  기본 게이트웨이 . . . . . : 192.168.1.1
```

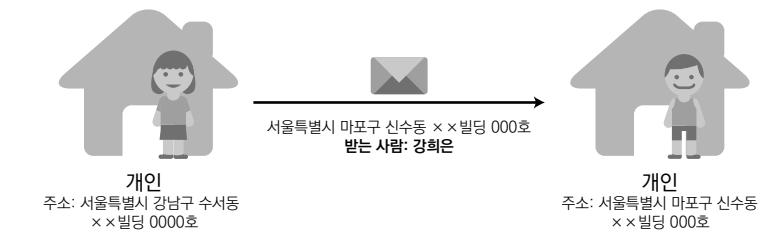
그런데 어떤 주소로 우편을 배달하였을 때 정작 ‘누구에게’가 빠져 있다면 그 주소에서 일하거나 살고 있는 사람 중에 누구에게 우편이 전달되어야 할지 몰라 우편을 받아야 하는 사람에게 잘 전달되지 않을 수 있습니다. 예를 들어, 엄마, 아빠, 딸 이렇게 세 사람이 한 집에 살고 있다면 받는 사람이 적혀 있지 않은 우편은 세 사람 중 누구의 우편인지 알 수가 없습니다. 그래서 우편봉투에는 받는 사람의 주소와 함께 받는 사람의 이름도 적습니다. 컴퓨터에서 다른 컴퓨터에게 데이터를

전송할 때 역시 마찬가지입니다. 데이터를 전달받을 대상 컴퓨터의 IP 주소와 함께 그 컴퓨터 내에서 실행되는 많은 프로그램 중 어떤 프로그램에 전달되어야 할지를 함께 적어 주어야 합니다.

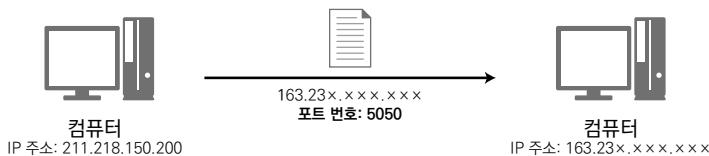
그렇다면 컴퓨터 네트워크에서 받는 사람의 이름에 해당하는 것은 무엇일까요? 프로그램은 ‘포트 Port’라는 네트워크를 통해 데이터를 주고받는 가상의 문을 가지고 있고, 운영체제는 포트마다 0~65535 범위의 정수 중 하나를 골라 번호를 붙여 관리합니다. 따라서 프로그램은 각각 고유한 포트 번호를 가지고 있습니다. 이를 이용하여 컴퓨터 간 데이터 전송을 할 때는 IP 주소(어떤 컴퓨터로 보낼지)와 포트 번호(그 컴퓨터 내 어떤 프로그램으로 보낼지)를 적어 줍니다.

그림 9-8 주소와 받는 사람을 적어 데이터 전송하기

인적 네트워크



컴퓨터 네트워크



따라서 자바로 컴퓨터 간 데이터를 주고받을 수 있는 프로그램을 작성할 때도 그 프로그램을 식별할 수 있게 해당 프로그램의 포트 번호를 정해주어야 합니다. 포트 번호는 0~65535 범위의 정수를 사용하지만 그 중 0~1023까지는 운영체제 시스템이 사용하는 포트 번호이므로 응용 프로그램은 1024 이상의 숫자를 포트 번호로 사용할 수 있습니다. 기존에 사용되고 있는 포트 번호와 겹치지 않기 위해

원도우 운영체제의 경우 명령어 프롬프트에서 ‘netstat’를 쳐서 현재 사용 중인 포트 번호를 확인한 후 새로운 포트 번호를 사용하는 것이 좋습니다.

그림 9-9 사용 중인 포트 번호 확인

프로토콜	로컬 주소	외부 주소	상태
TCP	127.0.0.1:19872	hin-windows8:49288	ESTABLISHED
TCP	127.0.0.1:49288	hin-windows8:19872	ESTABLISHED
TCP	127.0.0.1:49381	hin-windows8:49382	ESTABLISHED
TCP	127.0.0.1:49382	hin-windows8:49381	ESTABLISHED

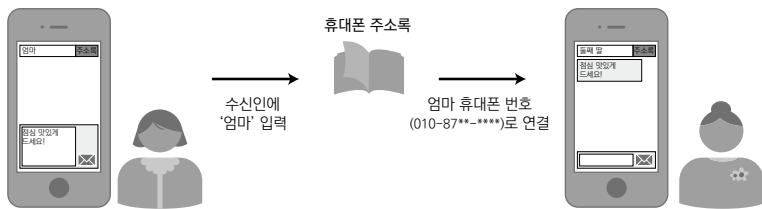
9.1.2 DNS

IP 주소는 최대 12개의 숫자로 구성되어 있기 때문에 여러 컴퓨터의 IP주소를 외우기는 어렵습니다. 휴대폰 전화번호 역시 일일이 다 기억하기 어려우므로 주로 휴대폰의 전화번호부에 이름마다 전화번호를 저장해 두고 사용합니다. 이와 같은 원리로 IP 주소도 사람이 비교적 쉽게 기억할 수 있는 문자 이름을 가질 수 있는데, 이를 ‘도메인’(ex. www.google.com)이라 합니다. 그리고 IP 주소와 도메인을 함께 저장해 둔 IP 주소록 역할을 하는 DNS^{Domain Name System (IP 주소를 도메인으로 표시한 체계)} 서버가 있습니다. 나라마다 도메인을 관리하는 곳이 있는데, 우리나라의 경우 '.kr'을 관리하는 한국인터넷진흥원^{KISA}이 있습니다.

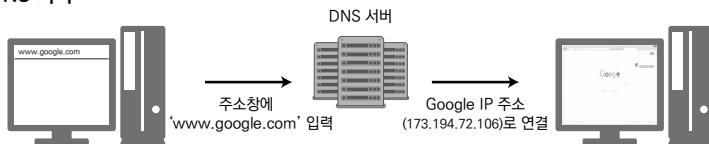
DNS 서버는 도메인과 IP 주소를 가지고 있고, 특정 컴퓨터가 인터넷 탐색기의 주소 창에 도메인을 입력하면 이를 실제 IP 주소와 매칭하여 연결하는 역할을 합니다. 휴대폰으로 누군가에게 메시지를 보낼 때 그 사람의 전화번호를 직접 써 넣지 않아도 수신인 자리에 이름을 입력하면 자동으로 그 이름의 전화번호로 메시지가 전송되는 것과 같은 원리입니다.

그림 9-10 휴대폰 주소록과 DNS 서버

휴대폰 주소록



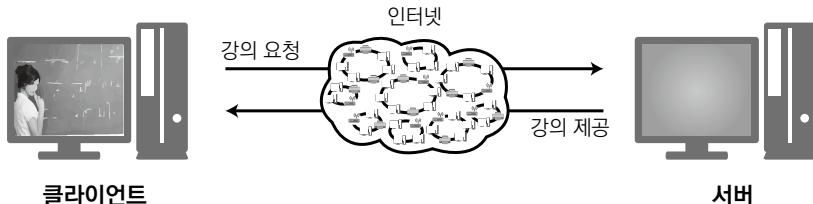
DNS 서버



9.1.3 클라이언트와 서버

우리가 인터넷으로 영화 예매를 할 수 있는 것은 영화관 사이트에서 예매 서비스를 제공하기 때문이고, 인터넷으로 강의를 볼 수 있는 것은 강의 사이트에서 강의 동영상을 제공하기 때문에 가능한 일입니다. 이때, 영화 예매 서비스나 강의 동영상을 제공하는 사이트의 프로그램 코드를 가진 컴퓨터를 ‘서버’라고 하고, Internet Explorer나 Chrome과 같은 인터넷 탐색기로 해당 사이트에 접속하여 그 사이트가 제공하는 서비스를 이용하는 컴퓨터를 ‘클라이언트’라고 합니다.

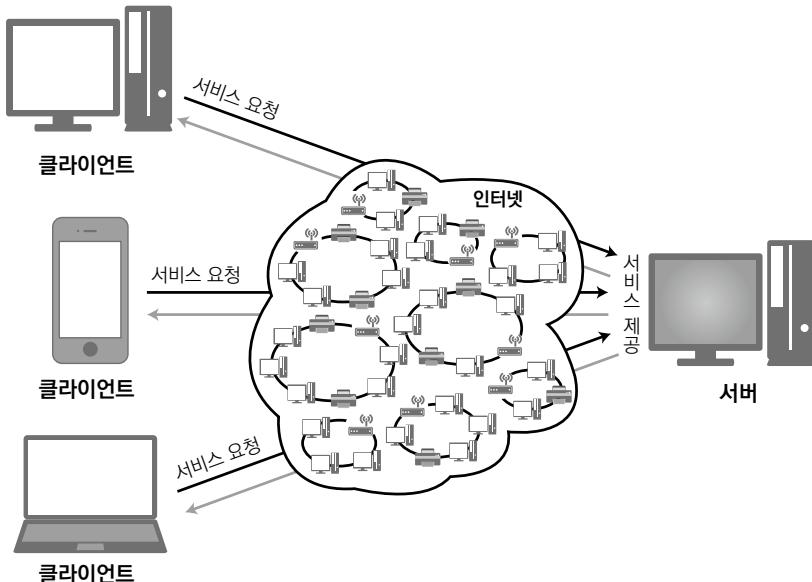
그림 9-11 클라이언트와 서버



다시 말해, 서버는 다른 컴퓨터나 프로그램에 서비스를 제공하는 쪽을 말하고, 클라이언트는 서버에 원하는 서비스를 요청하여 이를 제공받아 사용하는 쪽을 말합

니다. 보통 하나의 서버에 여러 대의 클라이언트가 서비스를 요청하고, 그 서비스를 제공받는 구조를 띠고 있습니다. 여러 명의 친구가 동시에 같은 인터넷 강의 사이트에 접속하여 같은 동영상을 볼 수 있는 것처럼요.

그림 9-12 다수의 클라이언트와 하나의 서버



일반 개인용 컴퓨터가 모니터와 본체를 갖추고 있는 것과는 달리, 서버는 보통 본체만 가지고 있는 경우가 많습니다. 물론 일반 개인용 컴퓨터도 서버로 사용할 수는 있지만, 수많은 클라이언트의 요청을 처리해야 하는 서버는 대부분 다수의 본체를 모아 놓은 형태로 만들어져 있습니다.

그림 9-13 서버의 다양한 형태



때때로 어떤 사이트에 접속했을 때 서비스가 느리거나 아예 접속이 안 되는 현상은 동시에 많은 클라이언트가 서버에 서비스를 요청했을 때 서버가 이를 다 처리하느라 바쁘기 때문에 발생한 것입니다. 보통 웹 서버는 접속하는 클라이언트마다 스레드를 만들어 요청을 처리하도록 대응하는데, 클라이언트 수가 많아지면 많아질수록 서버에서 돌아가는 스레드의 개수도 증가하게 됩니다. 이렇게 되면 고정되어 있는 컴퓨터 자원(CPU 등)을 여러 개의 스레드가 나누어 사용하기 때문에 속도가 자연스레 느려지게 됩니다. 그래서 많은 소비자에게 서비스를 제공해야 하는 경우 그만큼 서버도 여러 대 구성해 놓습니다.

많은 컴퓨터가 한곳에 모여 있으면 이것들로부터 열기가 많이 발생하는데, 이를 적정 온도와 습도로 유지할 수 있도록 시설을 갖춘 것이 ‘데이터 센터’입니다. 즉, 데이터 센터는 수백, 수만 개의 서버 컴퓨터와 네트워크 회선을 관리하기 위한 시설입니다. 큰 회사의 경우에는 자체적으로 데이터 센터를 만들기도 하지만 작은 규모의 회사는 데이터 센터에 요금을 내고 서버 공간을 임대하기도 합니다.

그림 9-14 데이터 센터의 모습⁰¹

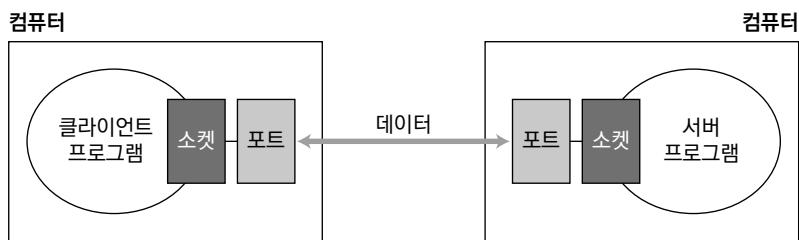


01 구글 데이터 센터(출처: <https://goo.gl/tu29lc>)

9.2 네트워크 통신 프로그래밍

네트워크를 통해 컴퓨터 간에 데이터를 주고받을 때는 서비스를 요청하는 클라이언트 컴퓨터와 이 요청을 받고 서비스를 제공해 주는 서버 컴퓨터로 역할이 나누어집니다. 자바로 네트워크 프로그래밍을 할 때도 2개의 프로그램, 즉 서버 프로그램과 클라이언트 프로그램 두 개의 코드를 작성해야 합니다. 서버 컴퓨터에서는 서버 프로그램을 돌리고, 클라이언트 컴퓨터에서는 클라이언트 프로그램을 돌려야만 두 컴퓨터가 서로 데이터를 주고받을 수 있습니다.

그림 9-15 클라이언트 프로그램과 서버 프로그램



이렇게 두 컴퓨터가 통신할 때 필요한 기본적인 요소들을 살펴보겠습니다.

9.2.1 인터넷을 통해 데이터를 주고받을 때 필요한 것들

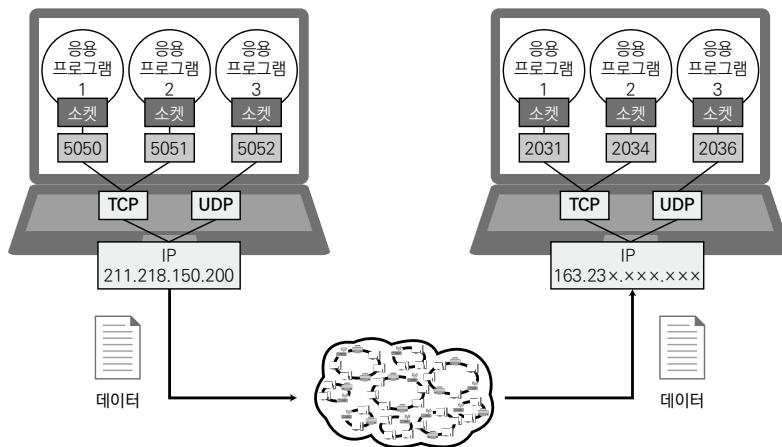
지리적으로 가까운 친구에게 편지를 전해 줄 때는 주소나 받는 사람 등을 적지 않고 바로 건네주면 됩니다. 하지만 멀리 사는 친구에게 편지를 주고 싶을 때는 우편배달부가 잘 전달해 줄 수 있게 주소와 받는 사람을 꼭 적어야 합니다. 이와 같은 원리로 클라이언트 컴퓨터와 서버 컴퓨터가 직접 연결되어 있다면 문제가 없지만, 인터넷을 통해 데이터를 전송하려면 클라이언트 컴퓨터는 서버 컴퓨터의 'IP 주소'와 그 위에서 돌고 있는 서버 프로그램의 '포트 번호'를 알아야만 합니다.

또한, 우편을 보낼 때 받는 사람이 잘 받았는지 확인하는 등기 우편과 확인하지 않는 일반 우편이 있듯이, 두 컴퓨터가 인터넷으로 데이터를 주고받는 방식에도 TCP^{Transmission Control Protocol}와 UDP^{User Datagram Protocol}가 있습니다.

TCP는 데이터를 주고받기 전에 서버와 클라이언트가 미리 연결을 맺은 후 등기 우편처럼 상대 컴퓨터에 데이터가 잘 전달되었는지 확인하고 혹시 문제가 있으면 데이터를 재전송하는 방식입니다. 주로 이메일이나 파일 다운로드 등 데이터가 정확히 전달되어야 하는 경우에 TCP를 씁니다.

UDP는 데이터를 주고받기 전 서버와 클라이언트가 미리 연결을 맺지 않고, 일반 우편처럼 상대 컴퓨터가 데이터를 잘 전달받았는지 확인하지 않습니다. 도중에 문제가 발생해도 모르기 때문에 당연히 데이터를 재전송하지도 않습니다. 주로 동영상 스트리밍이나 화상 채팅 같이 데이터가 조금 손실되어도 괜찮은 경우에 UDP를 사용합니다.

그림 9-16 데이터의 전달 과정



우편을 보낼 때를 다시 생각해 봅시다. 우편배달부가 받는 사람까지 어떤 경로를 통해 우편을 배달하는지 알 필요 없이 그냥 우체국에 우편을 맡기기만 하면 됩니다. 컴퓨터에서 다른 컴퓨터로 데이터를 전송할 때도 데이터 송수신 시 필요한 하드웨어, 소프트웨어 관점의 세세한 처리 작업을 프로그래머가 알 필요 없습니다. 운영체제에서 네트워크 연결을 위한 소프트웨어 장치를 제공해 주기 때문인데, 이를 ‘소켓Socket’이라고 합니다.

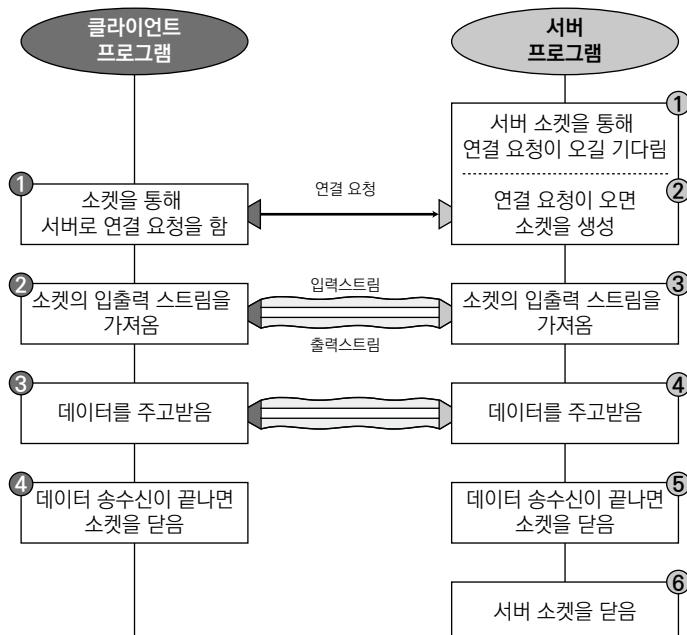
소켓은 담당하는 업무에 따라 서버 소켓과 일반 소켓으로 나누어집니다. 서버 소켓은 클라이언트의 연결 요청을 기다리고, 요청이 오면 연결을 맺는 역할을 담당합니다. 자바에서는 JDK의 라이브러리 중 java.net 패키지 안에 `ServerSocket` 이란 이름의 클래스로 제공됩니다. 일반 소켓은 서버 프로그램으로 연결을 요청하고 데이터를 전송하는 일을 하며 java.net 패키지 안에 `Socket`이란 이름의 클래스로 제공됩니다. 프로그래밍 시에는 '`java.net.*`'을 임포트하고 이 클래스의 객체를 생성하는 방식으로 서버 소켓과 클라이언트 소켓을 만들어 줄 수 있습니다.

이번 장에서는 데이터 통신을 할 때 미리 연결을 맺는 방식인 TCP 방식으로 IP 주소, 포트 번호, 소켓을 이용하여 통신하는 방법을 배우겠습니다.

9.2.2 TCP/IP 프로그래밍

TCP 방식으로 소켓을 통해 데이터를 주고받는 절차는 [그림 9-17]과 같습니다.

그림 9-17 TCP 방식으로 소켓을 통해 데이터를 주고받는 절차



[그림 9-17]의 절차대로 프로그램 코드를 작성하는 방법을 서버 프로그램과 클라이언트 프로그램으로 나누어서 살펴보겠습니다. 먼저, 서버 프로그램을 작성하는 방법은 다음과 같습니다.

[서버 프로그램을 작성하는 방법]

- (1) 해당 프로그램의 포트 번호를 선택하고 서버 소켓을 생성한다. ①

```
-> ServerSocket serverSocket = new ServerSocket(포트 번호)
```

- (2) accept() 메서드를 사용하여 클라이언트로부터 연결 요청을 기다리고 요청이 오면 연결을 맺은 후 소켓을 생성한다. ②

```
-> ServerSocket serverSocket = new ServerSocket(포트 번호)  
Socket socket = serverSocket.accept();
```

- (3) 소켓의 입력 스트림을 가져온다. ③

```
-> ServerSocket serverSocket = new ServerSocket(포트 번호)  
Socket socket = serverSocket.accept();  
BufferedReader reader = new BufferedReader(  
    new InputStreamReader(socket.getInputStream()));
```

* 이를 통해 클라이언트가 보낸 데이터를 읽어 올 수 있다.

* getInputStream() 메서드로 소켓의 입력 스트림을 가져온 후 이를 InputStreamReader 클래스의 객체에 넘겨 주어 바이트 스트림을 문자 스트림으로 바꾸어 준다. 그리고 이것을 BufferedReader 클래스의 객체를 통해 행 단위로 읽어 온다.

- (4) 소켓의 출력 스트림을 가져온다. ④

```
-> ServerSocket serverSocket = new ServerSocket(포트 번호)  
Socket socket = serverSocket.accept();  
BufferedReader reader = new BufferedReader(  
    new InputStreamReader(socket.getInputStream()));  
PrintWriter writer = new PrintWriter(socket.getOutputStream());
```

* 이것을 통해 클라이언트로 데이터를 보낼 수 있다.

* getOutputStream() 메서드로 소켓의 출력 스트림을 가져온 후 이를 PrintWriter 클래스의 객체에 넘겨 주어 문자 스트림을 바이트 스트림으로 바꾸어 준다.

- (5) BufferedReader 객체의 readLine() 메서드로 읽어 온 한 행의 문자열 데이터를 출력한다. ⑤

```
-> ServerSocket serverSocket = new ServerSocket(포트 번호)  
Socket socket = serverSocket.accept();  
BufferedReader reader = new BufferedReader(  
    new InputStreamReader(socket.getInputStream()));  
PrintWriter writer = new PrintWriter(socket.getOutputStream());  
System.out.println(reader.readLine());
```

- (6) PrintWriter 객체의 println() 메서드로 한 행의 바이트 데이터를 쓴다. ⑥

```
-> ServerSocket serverSocket = new ServerSocket(포트 번호)
    Socket socket = serverSocket.accept();
    BufferedReader reader = new BufferedReader(
        new InputStreamReader(socket.getInputStream()));
    PrintWriter writer = new PrintWriter(socket.getOutputStream());
    System.out.println(reader.readLine());
    writer.println("Hello, I'm server!");
    writer.flush();

* println( )메서드는 버퍼가 다 찬 후 한꺼번에 데이터를 보내기 때문에 이를 기다리지 않고 바로 보내기 위해
flush( ) 메서드를 사용합니다.
```

(6) 소켓을 닫는다. ⑥

```
-> ServerSocket serverSocket = new ServerSocket(포트 번호)
    Socket socket = serverSocket.accept();
    BufferedReader reader = new BufferedReader(
        new InputStreamReader(socket.getInputStream()));
    PrintWriter writer = new PrintWriter(socket.getOutputStream())
    System.out.println(reader.readLine());
    writer.println("Hello, I'm server!");
    writer.flush();
    socket.close();
```

(7) 서버 소켓을 닫는다. ⑦

```
-> ServerSocket serverSocket = new ServerSocket(포트 번호)
    Socket socket = serverSocket.accept();
    BufferedReader reader = new BufferedReader(
        new InputStreamReader(socket.getInputStream()));
    PrintWriter writer = new PrintWriter(socket.getOutputStream())
    System.out.println(reader.readLine());
    writer.println("Hello, I'm server!");
    writer.flush();
    socket.close();
    serverSocket.close();
```

실제로는 Socket을 생성할 때 오류가 발생할 수도 있으므로 try~catch 구문으로 해당 부분을 감싸주어야 합니다. try 블록에서 선언한 변수는 catch 블록 등 외부 블록에서 사용할 수 없으므로 ServerSocket과 Socket 객체 변수는 try 블록 밖에서 생성하고, 아직 그 변수에 “아무것도 없다”는 의미로 null을 넣어 줍니다. [코드 9-1]에서 서버 프로그램 예제를 살펴보겠습니다.

```

import java.io.*;
java.io 패키지의 모든 클래스를 여기서 사용한다

import java.net.*;

class ServerTest{
    public static void main(String[] args) {

        ServerSocket serverSocket = null;
        연결을 담당하는 서버 소켓을 생성하는 ServerSocket 클래스의 객체를 담는 serverSocket 변수를 선언하고
        아무 것도 없다는 의미로 null을 저장한다

        Socket socket = null;
        연결 요청과 데이터 송수신을 담당하는 일반 소켓을 생성하는 Socket 클래스의 객체를 담는 socket 변수를 선언하고
        아무 것도 없다는 의미로 null을 저장한다

        try {
            serverSocket = new ServerSocket(5050);
            serverSocket 변수에 해당 프로그램의 포트 번호를 5050으로 설정한
            ServerSocket 클래스의 객체를 새로 생성하여 저장한다

            System.out.println("ready to connect...");
            socket = serverSocket.accept();
            socket 변수에 serverSocket의 accept() 메서드를 호출하여
            클라이언트로부터 연결 요청을 기다리고 요청이 오면 연결을 맺은 후 생성된 소켓 객체를 저장한다

            System.out.println("connected!");

            BufferedReader reader = new BufferedReader(
            행 단위로 데이터를 읽어 오는 BufferedReader 클래스의 객체를 담는 reader 변수에
            BufferedReader 클래스를 새로 생성하여 저장한다

            new InputStreamReader(socket.getInputStream());
            socket 변수의 getInputStream() 메서드를 호출하여 가져온 소켓의 입력 스트림을 파라미터로,
            InputStreamReader 클래스의 객체를 새로 생성하여 문자 스트림으로 바꿔준다

            PrintWriter writer =
            문자 스트림을 바이트 스트림으로 바꿔 주는 PrintWriter 클래스의 객체를 담는 writer 변수에

            new PrintWriter(socket.getOutputStream());
            socket 변수의 getOutputStream() 메서드를 호출하여 가져온 소켓의 출력 스트림을
            바이트 스트림으로 바꿔 주는 PrintWriter 클래스의 객체를 새로 생성하여 저장한다
        }
    }
}

```

```
System.out.println(reader.readLine());
reader의 readLine() 메서드를 호출하여 한 행의 문자열 데이터를 읽어 와서 시스템 콘솔에 출력하고 한 줄 띄운다

writer.println("Hi, I'm server!");
writer의 println() 메서드를 호출하여, "Hi, I'm server!"를 입력한다

writer.flush();
writer의 flush() 메서드를 호출하여 버퍼에 있는 데이터를 스트림으로 흘려보낸다

}catch(Exception e){
    System.out.println(e);
}finally{
    try{
        Socket.close();
        Socket의 close() 메서드를 호출하여 소켓을 닫는다
    }catch(Exception e){ }
    try{
        serverSocket.close();
        serverSocket의 close() 메서드를 호출하여 소켓을 닫는다
    }catch(Exception e){ }
}
}
```

실행 결과

ready to connect...

아직 클라이언트 프로그램을 작성하여 실행하지 않아서 서버 프로그램의 실행 결과는 “ready to connect...”만 뜨게 됩니다. 이번에는 클라이언트 프로그램을 작성하는 방법을 알아보겠습니다.

(1) 연결을 요청할 서버의 IP 주소와 포트 번호를 가지고 소켓을 생성한다. ①

```
-> Socket socket = new Socket("IP 주소", 포트 번호);
```

(2) 소켓의 입력 스트림을 가져온다. ②

```
-> Socket socket = new Socket("IP 주소", 포트 번호);
BufferedReader reader = new BufferedReader(
    new InputStreamReader(socket.getInputStream()));
```

* 이를 통해 클라이언트가 보낸 데이터를 읽어 올 수 있다.

* getInputStream() 메서드로 소켓의 입력 스트림을 가져온 후 이를 InputStreamReader 클래스의 객체에 넘겨 주어 바이트 스트림을 문자 스트림으로 바꾸어준다. 그리고 이것을 BufferedReader 클래스의 객체를 통해 행 단위로 읽어 온다.

(3) 소켓의 출력 스트림을 가져온다. ③

```
-> Socket socket = new Socket("IP 주소", 포트 번호);
BufferedReader reader = new BufferedReader(
    new InputStreamReader(socket.getInputStream()));
PrintWriter writer =
    new PrintWriter(socket.getOutputStream());
```

* 이것을 통해 서버로 데이터를 보낼 수 있다.

* getOutputStream() 메서드로 소켓의 출력 스트림을 가져온 후 이를 PrintWriter 클래스의 객체에 넘겨 주어 문자 스트림을 바이트 스트림으로 바꾸어 준다.

(4) PrintWriter 객체의 println() 메서드로 한 행의 바이트 데이터를 입력한다. ③

```
-> Socket socket = new Socket("IP 주소", 포트 번호);
BufferedReader reader = new BufferedReader(
    new InputStreamReader(socket.getInputStream()));
PrintWriter writer =
    new PrintWriter(socket.getOutputStream());
writer.println("Hi, I'm client!");
writer.flush();
```

* println() 메서드는 버퍼가 다 찬 후 한꺼번에 데이터를 보내기 때문에 이를 기다리지 않고 바로 보내기 위해 flush() 메서드를 사용한다.

(5) BufferedReader 객체의 readLine() 메서드로 읽어온 한 행의 문자열 데이터를 출력한다. ③

```
-> Socket socket = new Socket("IP 주소", 포트 번호);
BufferedReader reader = new BufferedReader(
    new InputStreamReader(socket.getInputStream()));
PrintWriter writer =
    new PrintWriter(socket.getOutputStream());
writer.println("Hi, I'm client!");
writer.flush();
System.out.println(reader.readLine());
```

(6) 소켓을 닫는다. ④

```
-> Socket socket = new Socket("IP 주소", 포트 번호);
    BufferedReader reader = new BufferedReader(
        new InputStreamReader(socket.getInputStream()));
    PrintWriter writer =
        new PrintWriter(socket.getOutputStream());
    writer.println("Hi, I'm client!");
    writer.flush();
    System.out.println(reader.readLine());
    socket.close();
```

이 방법을 이용하여 작성한 클라이언트 프로그램 예제는 [코드 9-2]와 같습니다. 여기서 유의할 점은 서버 프로그램이 동작하고 있어야만 클라이언트 프로그램이 제대로 작동할 수 있다는 점입니다. 이 예제는 한 컴퓨터에서 서버 프로그램과 클라이언트 프로그램을 같이 테스트하였기 때문에 자기 자신을 가리키는 IP인 '127.0.0.1'을 서버 IP 주소로 주었습니다. 사용 가능한 컴퓨터가 2대 있다면 클라이언트 프로그램을 다른 컴퓨터에서 실행해 보는 것도 좋습니다. 이때, 서버의 IP 주소는 서버 프로그램을 작성한 컴퓨터의 IP 주소를 적어 주면 됩니다.

[코드 9-2] 클라이언트 프로그램 예제

import java.io.*;

java.io 패키지의 모든 클래스를 여기서 사용한다

import java.net.*;

class ClientTest{

public static void main(String[] args) {

Socket socket = null;

연결 요청과 데이터 송수신을 담당하는 일반 소켓을 생성하는 Socket 클래스의 객체를 담는 socket 변수를 선언하고

아무 것도 없다는 의미로 null을 저장한다

try {

socket = new socket("127.0.0.1", 5050);

socket 변수에 서버의 IP 주소 127.0.0.1와 포트 번호 5050을 가진

socket 클래스의 객체를 새로 생성하여 저장한다

BufferedReader reader = new BufferedReader(

행 단위로 데이터를 읽어 오는 BufferedReader 클래스의 객체를 담는 reader 변수에

BufferedReader 클래스를 새로 생성하여 저장한다

new InputStreamReader(socket.getInputStream()));

socket 변수의 getInputStream() 메서드를 호출하여 가져온 소켓의 입력 스트림을 파라미터로,

InputStreamReader 클래스의 객체를 새로 생성하여 문자 스트림을 바꿔 준다

PrintWriter writer =

문자 스트림을 바이트 스트림으로 바꿔 주는 PrintWriter 클래스의 객체를 담는 writer 변수에

new PrintWriter(socket.getOutputStream());

socket 변수의 getOutputStream() 메서드를 호출하여 가져온 소켓의 출력 스트림을

바이트 스트림으로 바꿔 주는 PrintWriter 클래스의 객체를 새로 생성하여 저장한다

writer.println("Hi, I'm client!");

writer의 println() 메서드를 호출하여 "Hi, I'm client!"를 입력한다

writer.flush();

writer의 flush() 메서드를 호출하여 버퍼에 있는 데이터를 스트림으로 흘려보낸다

System.out.println(reader.readLine());

reader의 readLine() 메서드를 호출하여 한 행의 문자열 데이터를 읽어 와서 시스템 콘솔에 출력하고 한 줄 띄운다

} catch(Exception e){

 System.out.println(e);

} finally{

 try{

 socket.close();

socket의 close() 메서드를 호출하여 소켓을 닫는다

 } catch(Exception e){ }

}

}

실행 결과

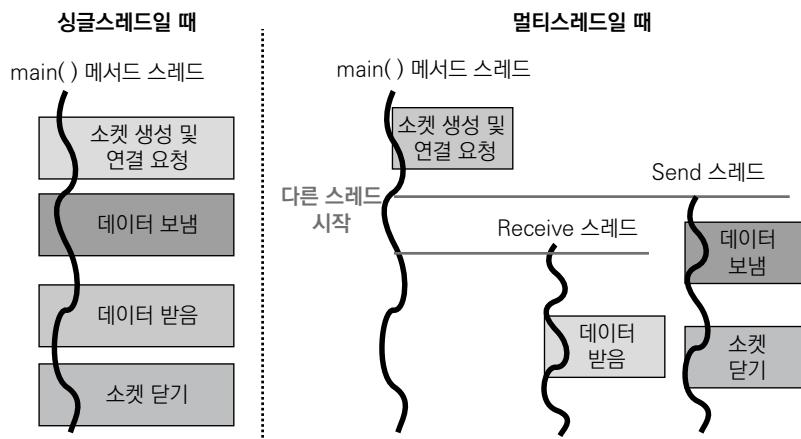
Hello, I'm server!

이때 [코드 9-1]에서 작성한 서버 프로그램의 실행 결과에는 “connected!”가 뜹니다.

9.2.3 스레드를 이용한 채팅 프로그램

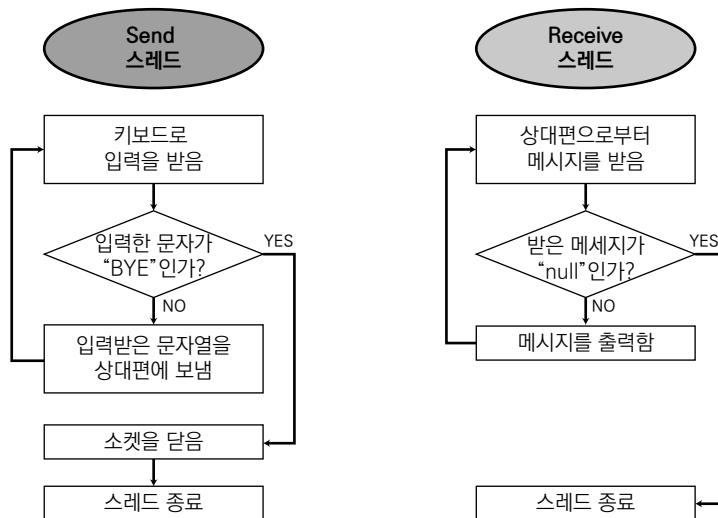
8장의 멀티스레드를 기억하시나요? 스레드를 여러 개 사용하면 한 프로그램 내에서 여러 가지 일을 동시에 할 수 있습니다. 네트워크 통신 프로그래밍에서도 스레드를 활용하면 데이터 전송과 수신을 동시에 할 수 있는 채팅 프로그램을 만들 수 있습니다.

그림 9-18 클라이언트 프로그램이 싱글스레드일 때와 멀티스레드일 때 비교



일반적인 채팅 프로그램의 기능을 생각해 보면 종료 버튼을 누를 때까지 하고 싶은 말을 키보드로 계속 입력할 수 있고 상대방이 보낸 메시지도 실시간으로 볼 수 있습니다. 이번 장에서 작성할 채팅 프로그램에서는 데이터를 보내는 Send 스레드가 “BYE”란 말을 입력받을 때까지 키보드로 하고 싶은 말을 사용자로부터 입력 받아 상대편 컴퓨터로 보내는 역할을 하고, 데이터를 받는 Receive 스레드가 상대편 컴퓨터로부터 받아온 메시지를 화면에 출력해 주는 역할을 할 것입니다. 중요한 점은 이 스레드들이 main 스레드에서 생성한 소켓을 통해 작업해야 하기 때문에 main 메서드를 작성한 후 해당 스레드들을 실행해야 한다는 점입니다.

그림 9-19 Send 스레드와 Receive 스레드의 역할



Send 스레드와 Receive 스레드는 클라이언트 프로그램과 서버 프로그램 둘 다에서 공통으로 사용할 수 있기 때문에 이 두 스레드 코드를 먼저 다룬 후, 클라이언트 프로그램의 main과 서버 프로그램의 main을 보겠습니다. 먼저 Send 스레드의 코드는 [코드 9-3]과 같습니다.

[코드 9-3] 메시지를 보내는 Send 스레드 예제

```
import java.io.*;
import java.net.*;

class SendThread extends Thread{
```

Socket socket;

연결 요청과 데이터 송수신을 담당하는 일반 소켓을 생성하는 Socket 클래스의 객체를 담는 socket 변수를 선언한다

SendThread(Socket socket){

SendThread 클래스의 객체 필드를 초기 설정하는 생성자를 만들고

Socket 클래스의 객체를 담는 socket 파라미터 변수를 선언한다

this.socket = socket;

이 클래스로 생성된 객체 안의 socket 변수에 socket 파라미터 변수를 저장한다

}

```
public void run(){
    try{
        BufferedReader reader = new BufferedReader(
            new InputStreamReader(System.in));
        키보드에서 입력받는다

        PrintWriter writer = new PrintWriter(
            socket.getOutputStream())

        while(true){
            true일 동안 {} 안을 반복해라 → while(true)는 무한반복

            String str = reader.readLine();
            if(str.equals("BYE")){
                str의 equals() 메서드를 호출하여 str의 값이 "BYE" 문자열과 같은지 확인하여
                같으면 true, 다르면 false를 리턴한다

                break;
                반복문을 여기서 멈춘다(while문을 중단한다)

                writer.println(str);
                writer.flush();
            }
        }catch(Exception e){
            System.out.println(e);
        }finally{
            try{
                System.out.println("****채팅방을 나갑니다****");
                socket.close();
            }catch(Exception e){}
        }
    }
}
```

[코드9-3]에서 나온 equals() 메서드는 String 클래스가 제공하는 메서드로 문자열 값이 같은지 다른지를 확인해 줍니다. 더 자세한 내용은 ‘11.2장 문자 관련 함수’에서 다루겠습니다. 이번에는 Receive 스레드의 예제 코드를 살펴보겠습니다.

[코드 9-4] 메시지를 받는 Receive 스레드 예제

```
import java.io.*;
import java.net.*;

class ReceiveThread extends Thread{

    Socket socket;
    연결 요청과 데이터 송수신을 담당하는 일반 소켓을 생성하는 Socket 클래스의 객체를 담는 socket 변수를 선언한다

    ReceiveThread(Socket socket){
        ResceiveThread 클래스의 객체 필드를 초기 설정하는 생성자를 만들고
            Socket 클래스의 객체를 담는 socket 파라미터 변수를 선언한다

        this.socket = socket;
        이 클래스로 생성된 객체 안의 socket 변수에 socket 파라미터 변수를 저장한다
    }

    public void run(){
        try{
            BufferedReader reader = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));

            while(true){
                true일 동안 {} 안을 반복해라 -> while(true)는 무한반복

                String str = reader.readLine();

                if(str == null)
                    str 변수의 값이 아무것도 없다는 의미인 null과 같으면
                        break;
                    반복문을 여기서 멈춘다(while문을 종단한다)

                System.out.println("[상대방]"+str);
        }
    }
}
```

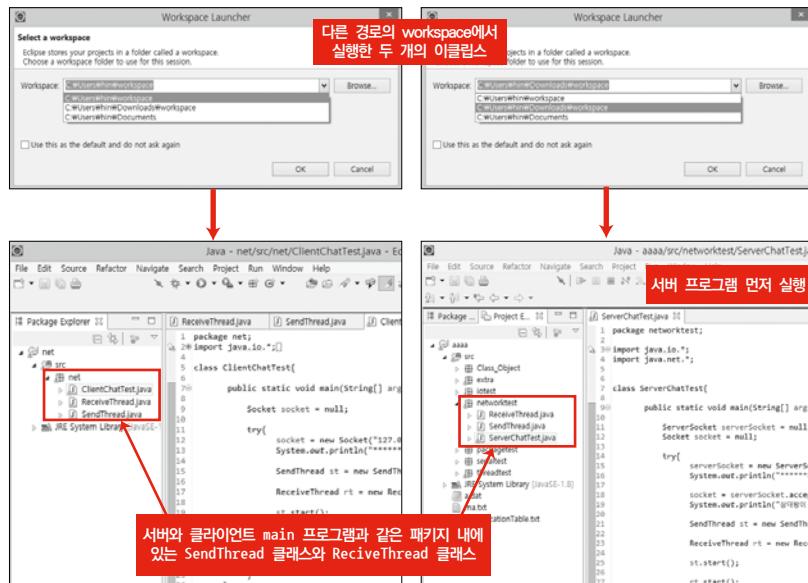
```

    }catch(Exception e){
    }
}
}

```

이제 서버 프로그램과 클라이언트 프로그램의 main을 작성해 보겠습니다. SendThread 클래스와 ReceiveThread 클래스의 코드를 서버 프로그램과 클라이언트 프로그램의 main을 작성한 클래스와 같은 패키지 안에 각각 복사하여 넣어 주어야 합니다. 두 개의 컴퓨터가 있다면, 각 컴퓨터에서 클라이언트 프로그램과 서버 프로그램을 하나씩 실행하면 좋습니다. 여의치 않는 경우, 이클립스를 하나 더 실행합니다. 두 개의 이클립스에 각각 다른 경로의 workspace를 설정한 후 한 이클립스에는 서버 프로그램을, 다른 이클립스에서는 클라이언트 프로그램을 실행합니다. 이 경우 클라이언트 프로그램 main의 IP 주소에 ‘127.0.0.1’을 넣어서 자신의 컴퓨터 IP를 가리켜 줍니다.

그림 9-20 이클립스를 두 개 실행하여 서버와 클라이언트 프로그램 돌리기



항상 서버 프로그램이 클라이언트 프로그램보다 먼저 실행되어야 한다는 점에 유의합니다. 먼저 서버 프로그램의 main을 살펴보겠습니다.

[코드 9-5] 서버 프로그램의 main 예제

```
import java.io.*;
import java.net.*;

class ServerChatTest{
    public static void main(String[] args) {
        ServerSocket serverSocket = null;
        Socket socket = null;

        try{
            serverSocket = new ServerSocket(7050);
            System.out.println("****채팅방이 열렸습니다****");

            socket = serverSocket.accept();
            System.out.println("상대방이 입장했습니다!");

            SendThread st = new SendThread(socket);
            SendThread라는 클래스의 객체를 담는 st라는 변수를 선언하고
            SendThread 클래스의 필드를 socket으로 초기화한 객체를 새로 생성하여 저장한다

ReceiveThread rt = new ReceiveThread(socket);
            ReceiveThread라는 클래스의 객체를 담는 rt라는 변수를 선언하고
            ReceiveThread 클래스의 필드를 socket으로 초기화한 객체를 새로 생성하여 저장한다

            st.start();
            st의 start() 메서드를 호출하여 스레드를 실행한다

            rt.start();
            rt의 start() 메서드를 호출하여 스레드를 실행한다
        }catch(Exception e){
            System.out.println(e);
        }finally{
            try{
                serverSocket.close();
            }
        }
    }
}
```

```
        }catch(Exception e){ }
    }
}
```

다음은 클라이언트 프로그램의 main 예제입니다.

[코드 9-6] 클라이언트 프로그램의 main 예제

```
import java.io.*;
import java.net.*;

class ServerChatTest{
    public static void main(String[] args) {
        Socket socket = null;
        try{
            serverSocket = new ("***.***.***.***",7050);
            System.out.println("****채팅방에 들어왔습니다****");
            SendThread st = new SendThread(socket);
            SendThread 클래스의 객체를 담는 st 변수를 선언하고
SendThread 클래스의 필드를 socket으로 초기화한 객체를 새로 생성하여 저장한다
            ReceiveThread rt = new ReceiveThread(socket);
            ReceiveThread 클래스의 객체를 담는 rt 변수를 선언하고
ReceiveThread 클래스의 필드를 socket으로 초기화한 객체를 새로 생성하여 저장한다
            st.start();
            st의 start() 메서드를 호출하여 스레드를 실행한다
            rt.start();
            rt의 start() 메서드를 호출하여 스레드를 실행한다
        }catch(Exception e){
            System.out.println(e);
        }
    }
}
```

서버 프로그램을 실행한 후 클라이언트 프로그램을 실행한 결과는 다음과 같습니다.

서버 프로그램(ServerChatTest) 실행 결과

****채팅방이 열렸습니다****

상대방이 입장했습니다!

Hi

[상대방]Hi

How are you?

[상대방]I'm fine, and you?

I'm fine, too.

BYE

****채팅방을 나갑니다****

클라이언트 프로그램(ClientChatTest) 실행 결과

****채팅방에 들어왔습니다****

[상대방]Hi

Hi

[상대방]How are you?

I'm fine, and you?

[상대방]I'm fine, too.

BYE

****채팅방을 나갑니다****

편리한 함수들

10장에서는 자바의 JDK 라이브러리 안에 구현된 여러 가지 유용한 클래스를 살펴보겠습니다. 10.1장에서는 제곱근이나 지수를 구할 수 있는 메서드를 포함한 `Math` 클래스를, 10.2장에서는 문자의 길이를 구하거나 문자열을 모두 대문자로 변경해 주는 등의 메서드를 가진 `String` 클래스를 공부해 보겠습니다. 마지막으로 10.3장에서는 현재 시각을 구할 수 있는 `Date`와 `Calendar` 클래스에 대해 다루겠습니다.

10.1 수학 관련 함수

수학의 사칙 연산인 ‘+, -, ×, ÷’는 자바에서 연산자로 구현되어 있기 때문에 2 더하기 3 등의 간단한 연산은 자바 프로그램 코드로 ‘`2+3`’으로 쉽게 표현할 수 있습니다. 그런데 수학 수업 시간에 배우는 제곱근, 지수, 사인, 코사인 같은 것을 구하는 프로그램을 자바로 직접 짜려하면, 막상 어떻게 해야 할지 감이 오지 않습니다. 이때 JDK 라이브러리 안의 `Math` 클래스를 이용하면 손쉽게 해결됩니다.

`Math` 클래스의 메서드는 전부 정적 메서드라서 객체를 생성하지 않고 클래스명으로 바로 메서드를 호출할 수 있습니다([4.3.2 static 참고](#)). `Math` 클래스의 메서드는 다음과 같습니다.

표 10-1 Math 클래스의 메소드

메소드 원형	설명	사용 예시
double pow(double a, double b)	a^b (a의 b승)	double z = Math.pow(2.0, 3.0);
double sqrt(double a)	\sqrt{a} (a의 제곱근)	double z = Math.sqrt(100.0);
int abs(int a)	$ a $ (a의 절대값)	int z = Math.abs(-17);
long abs(long a)		
float abs(float a)		
double abs(double a)		
int max(int a, int b)	a와 b 중 큰 값	int z = Math.max(10,5);
long max(long a, long b)		
float max(float a, float b)		
double max(double a, double b)		
int min(int a, int b)	a와 b 중 작은 값	double z = Math.min(10.0,5.0);
long min(long a, long b)		
float min(float a, float b)		
double min(double a, double b)		
double exp(double a)	e^a (e 지수)	double z = Math.exp(2.0);
double log(double a)	$\ln a$ (자연로그 a)	double z = Math.log(10.0);
double log10(double a)	$\log a$ (상용로그 a)	double z = Math.log10(10.0);
double toRadians(double a)	a 각도를 라디안으로 변경	double rad = Math.toRadians(30.0);
double toDegrees(double a)	a 라디안을 각도로 변경	double deg = Math.toDegrees(rad);
double sin(double a)	sin a(사인 a)	double s = Math.sin(rad);
double cos(double a)	cos a(코사인 a)	double c = Math.cos(rad);
double tan(double a)	tan a(타angent a)	double t = Math.tan(rad);
double random()	0.0~0.999990... 사이의 난수 리턴	double z = Math.random();

Math 클래스의 필드는 다음과 같습니다.

표 10-2 Math 클래스의 필드

필드명	설명	사용 예시
E	자연상수 e의 근사값(2.718···)	double z = Math.log(Math.E);
PI	원주율(3.14···)	double z = Math.PI

Math 클래스의 random 메서드를 사용한 프로그램 코드를 살펴보겠습니다.

[코드 10-1] 1~45 사이의 난수 발생시키기

class MathTest{

객체를 정의/생성하는 툴인 MathTest라는 클래스를 만든다

```
public static void main(String[] args) {
```

모든 패키지와 클래스에서 접근 가능하고, 같은 클래스에서 생성된 객체들과 값을 공유하며, 리턴값이 없는

main이라는 함수를 선언하고, 문자열 값을 담는 args라는 배열을 인자로 받는다

```
int rand = (int)(Math.random()*45)+1;
```

정수 값을 담는 rand라는 변수를 선언하고 Math의 random() 메서드를 호출하여 받은 값에 45를 곱한 후
(0.0~45.x의 값 중 리턴) int 형으로 바꾸고 그 값에 1을 더하여 저장한다(1~45사이의 난수가 만들어짐)

```
System.out.println(rand);
```

rand 변수의 값을 시스템 콘솔에 출력하고 한 줄 띠운다

```
}
```

```
}
```

실행 결과 ※ 난수가 무엇이 나오느냐에 따라 결과값이 다를 수 있습니다.

14

10.2 문자 관련 함수

문서를 작성하다 보면 문장 뒤에 문장을 붙이고 싶을 때도 있고, 이 문장의 길이가 얼마만큼인지 궁금할 때도 있습니다. 자바에서 문자열을 표현하는 클래스인

String에는 이와 같은 기능을 가진 메서드가 있습니다. String 클래스의 메서드는 다음과 같습니다.

표 10-3 String 클래스의 메서드

메소드 원형	설명	사용 예시
String substring(int start, int end)	해당 문자열의 start 위치부터 end 위치까지 문자열을 자른다.	String str = "Hello Java!"; String str2 = str.substring(2,5);
int length()	해당 문자열의 길이를 구한다.	String str = "Hello Java!"; int leng = str.length();
String toUpperCase()	해당 문자열을 모두 대문자로 변경한다.	String str = "Hello Java!"; String Ustr = str.toUpperCase();
String toLowerCase()	해당 문자열을 모두 소문자로 변경한다.	String str = "Hello Java!"; String Lstr = str.toLowerCase();
int indexOf(int ch)	해당 문자열에서 특정 문자(ch)의 위치를 찾아 리턴한다(문자 위치는 0부터 시작, 특정 문자 없으면 -1 리턴).	String str = "Hello Java!"; int idx = str.indexOf('H');
char charAt(int index)	해당 문자열에서 특정 위치 (index)에 해당하는 문자를 리턴한다(문자 위치는 0부터 시작).	String str = "Hello Java!"; char ch = str.charAt(6);
boolean equals(String str)	해당 문자열과 주어진 문자열(str)을 비교하여 일치하면 true, 아니면 false를 리턴한다.	String str = "Hello Java!"; boolean result = str.equals("Hi");

String 클래스의 equals() 메서드를 사용하여 키보드로 입력받은 값이 특정 문자열과 일치할 때까지 계속 키보드 입력을 받는 프로그램 코드 예제는 다음과 같습니다. 키보드 입력을 받을 때 사용하는 Scanner 클래스를 이용하려면 코드 상단에 ‘import java.util.Scanner;’를 써주어야 합니다.

[코드 10-2] 특정 문자열이 입력될 때까지 계속 키보드로 입력받는 프로그램

```
import java.util.Scanner;  
java.util.Scanner 패키지의 클래스들을 여기서 사용한다
```

class StringTest{

객체를 정의/생성하는 틀인 StringTest라는 클래스를 만든다

```
public static void main(String[] args) {
```

모든 패키지와 클래스에서 접근 가능하고, 같은 클래스에서 생성된 객체들과 값을 공유하며, 리턴값이 없는 main이라는 험수를 선언하고, 문자열 값을 담는 args라는 배열을 인자로 받는다

```
Scanner sc = new Scanner(System.in);
```

입력 클래스의 객체를 담는 sc라는 변수에, 키보드에서 입력받는 Scanner 클래스의 객체를 새로 생성하여 저장한다

```
System.out.println("insert QUIT...");
```

“Insert QUIT...”를 시스템 콘솔에 출력하고 한 줄 띄운다

```
while(!(sc.nextLine()).equals("QUIT")){
```

sc의 nextLine() 메서드를 호출하여 문자열로 읽어 온 다음 값이 “QUIT” 문자열과 같은지 확인하고
같지 않을 동안 반복한다(같으면 true, 다르면 false를 리턴한다)

```
System.out.println("Not match");
```

“Not match”를 시스템 콘솔에 출력하고 한 줄 띄운다

```
}
```

```
}
```

```
}
```

↑
키보드로 입력받은 값이 “QUIT”일 때 while문 종료!

실행 결과 ※ 원하는 값을 입력해 보고 마지막에는 QUIT를 입력해 보세요.

```
insert QUIT...
hi
Not match
quit
Not match
QUIT
```

10.3 시간 관련 함수

JDK 라이브러리의 Date 클래스와 Calendar 클래스를 이용하면 자바 프로그램에서 현재 일시를 가져올 수 있습니다. 이 두 클래스를 이용하려면 ‘import java.util.*’를 임포트해야 합니다. 우선 Date 클래스를 이용하여 현재 시각을 구하는 예제를 살펴보겠습니다.

```
import java.util.*;
```

java.util 패키지의 모든 클래스를 여기서 사용한다

```
class DateTest{
```

```
    public static void main(String[] args) {
```

```
        Date date = new Date();
```

Date 클래스의 객체를 담는 date라는 변수에 Date 클래스의 객체를 새로 생성하여 저장한다

```
        System.out.println(date.toString());
```

date의 toString() 메서드를 호출하여 date 객체를 문자열 형식으로 바꿔 시스템 콘솔에 출력하고 한 줄 띄운다

```
    }  
}
```

실행 결과 ※ 실행하는 컴퓨터 날짜에 따라 다르게 찍힙니다.

Mon Oct 12 21:38:58 KST 2015

Date 클래스의 객체를 생성하고, 이를 `toString()` 메서드를 사용하여 문자열로 바꾼 후 출력하면 현재 시간을 알 수 있습니다. 하지만 올해가 몇 년인지 또는 오늘이 며칠인지 따로따로 알고 싶을 때가 있습니다. 이럴 때 사용하는 것이 `Calendar` 클래스입니다. `Calendar` 클래스의 필드를 살펴보면 다음과 같습니다.

표 10-4 `Calendar` 클래스의 필드

필드명	설명	사용 예시
YEAR	연도	<code>Calendar cal = Calendar.getInstance(); cal.get(Calendar.YEAR));</code>
MONTH	월(0~11) * 0은 1월, 11은 12월	<code>Calendar cal = Calendar.getInstance(); cal.get(Calendar.MONTH)+1);</code>
DATE	일	<code>Calendar cal = Calendar.getInstance(); cal.get(Calendar.DATE));</code>
AM_PM	오전/오후 * 0은 오전, 1은 오후	<code>Calendar cal = Calendar.getInstance(); cal.get(Calendar.AM_PM));</code>

필드명	설명	사용 예시
HOUR	시	Calendar cal = Calendar.getInstance(); cal.get(Calendar.HOUR));
MINUTE	분	Calendar cal = Calendar.getInstance(); cal.get(Calendar.MINUTE));
SECOND	초	Calendar cal = Calendar.getInstance(); cal.get(Calendar.SECOND));
MILLISECOND	밀리초	Calendar cal = Calendar. getInstance();cal.get(Calendar. MILLISECOND));

Calendar 클래스를 이용하여 현재 시간을 구하려면 우선 getInstance() 메서드로 컴퓨터 시스템으로부터 현재 시간에 대한 객체를 얻어 와서 Calendar 클래스 타입을 담는 객체 변수에 저장해야 합니다. 그 후 Calendar 클래스의 객체에서 get() 메서드에 원하는 데이터 종류를 파라미터로 넣어 사용할 수 있습니다. Calendar 클래스를 사용하여 연, 월, 일을 구하는 예제는 다음과 같습니다.

[코드 10-4] Calendar 클래스를 이용하여 현재 연, 월, 일을 구하는 프로그램

```
import java.util.*;
java.util 패키지의 모든 클래스를 여기서 사용한다

class CalendarTest{
    public static void main(String[] args) {
        Calendar cal = Calendar.getInstance();
        Calendar 클래스의 객체를 담는 cal이라는 변수에
        Calendar 클래스의 getInstance() 메서드로 시스템의 시간 객체를 얻어 와 저장한다

        System.out.println("연도:"+cal.get(Calendar.YEAR));
        "연도:" 다음에 cal의 get() 메서드로 Calendar.YEAR 형식의 값을 가져와 시스템 콘솔에 출력하고 한 줄 띄운다

        System.out.println("월:"+(cal.get(Calendar.MONTH)+1));
        "월:" 다음에 cal의 get() 메서드로 Calendar.MONTH 형식의 값을 가져와 1을 더한 후
        시스템 콘솔에 출력하고 한 줄 띄운다

        System.out.println("일:"+cal.get(Calendar.DATE));
        "일:" 다음에 cal의 get() 메서드로 Calendar.DATE 형식의 값을 가져와 시스템 콘솔에 출력하고 한 줄 띄운다
```

}

실행 결과 ※ 실행하는 컴퓨터 날짜에 따라 다르게 찍힙니다.

연도:2015

월:10

일:12

마무리하며

보통 혼자서 컴퓨터 프로그래밍에 입문하는 사람들이 어렵다고 느끼고 포기하는 첫 번째 지점은 프로그래밍 언어를 배울 때입니다. 이를 잘 이겨내고 프로그래밍 언어를 배운 후에는 프로그램을 만드는 재미를 느낄 수도 있습니다. 마치 영어 단어를 외우고 문법을 배울 때는 힘이 들지만 이 과정이 지나면 영어 원서를 읽는 재미도 있고 외국인과 영어로 대화하는 즐거움도 있는 것처럼 말이죠.

이 책은 인문계 학생들이 혼자서 처음 자바 프로그래밍을 접할 때 어렵다고 포기하지 않고 쉽게 배울 수 있도록 하는 것을 목표로 쓰였습니다. 제가 자바를 처음 접했을 때 궁금했었던 컴퓨터의 기본 구조나 자바의 정의 및 작동 원리 등을 다루는 것을 시작으로, 변수, 배열, 클래스, 객체 등 자바의 기초 문법과 개념들을 공부하고 더 나아가 스레드나 네트워크 등의 상대적으로 깊이 있는 내용을 다루어 1대 1 채팅 프로그램까지 작성할 수 있도록 하였습니다.

하지만 이 책에서 다른 내용만으로는 실제 소비자가 사용할 수 있는 프로그램을 만드는데 부족한 면이 많습니다. 영어 단어와 문법을 배운 후에 영어 독해나 영작을 할 수 있어도, 비즈니스 문서나 영어 에세이를 쓰는 데에는 또 다른 노력이 필요한 것처럼 말입니다.

홈페이지나 그림을 사용한 사용자 인터페이스를 가진 프로그램 등을 작성하기 위해선 GUI 프로그래밍이나 데이터베이스 연동 프로그래밍(JDBC), 애플릿^{Applet} 프로그래밍 등을 공부해야 합니다. 정식으로 컴퓨터 공학에 대해 공부하고 싶다

면 하나의 프로그래밍 언어에 치중된 것이 아니라 원론적인 공부를 할 수도 있습니다.

이 책을 통해 자바를 배운 분들이 자바의 기본기를 익히는 데에 그치지 않고 프로그래밍에 재미를 느껴 프로그래밍을 계속해나가면 좋겠습니다.

연습문제와 답

연습 2-1

구구단을 for문을 사용하여 계산해 보세요(프로그래밍 언어에서는 곱셈을 '*'로 표현합니다).

```
1X1 = 1
1X2 = 2
...
9X9 = 81
```

답안

```
public class Gugudan {
    public static void main(String args[]) {
        for (int i = 1; i <= 9; i++) {
            System.out.println("*****" + i + "단 *****");
            for (int j = 1; j <= 9; j++) {
                System.out.println(i + "x" + j + "=" + i * j);
            }
        }
    }
}
```

연습 2-2

do~while문을 사용하여 [코드 2-2]의 결과와 똑같이 출력되도록 프로그래밍해 보세요.

답안

```
public class Dowhile {  
    public static void main(String[] args) {  
        String str[] = {"Hello", "Java", "Nice to meet you", "Bye!"};  
        int i=0;  
        do{  
            System.out.println(str[i]);  
            i++;  
        }while(i<4);  
    }  
}
```

연습 2-3

교수님이 기말고사 성적에 따라 학점을 매기려고 합니다. 성적이 80 이상이면 A, 60 이상이면 B, 40 이상이면 C, 40 미만이면 D라는 학점 기준이 있습니다. 강희은 학생이 45점을 맞았을 때, 어떤 학점을 받는지 출력하는 프로그램을 작성해 봅시다.

답안

```
public class Ifelse {  
    public static void main(String args[]){  
        int i=45;  
        if(i>=80)  
            System.out.println("A");  
        else if(i<80&&i>=60)  
            System.out.println("B");  
        else if(i<60&&i>=40)  
            System.out.println("C");  
        else  
            System.out.println("D");  
    }  
}
```

연습 2-4

과일가게에서 사과는 개당 1,000원, 배는 개당 2,000원, 딸기는 개당 500원, 귤은 300원에 판매하고 있습니다. 어떤 손님이 귤 8개를 산다고 했을 때 얼마를 내야 하는지 출력하는 프로그램을 작성해 봅시다.

답안

```
public class Switch {
    public static void main(String[] args) {
        String fruit = "귤";
        int price, amount = 8;

        switch(fruit){
            case "사과" : price = 1000; break;
            case "배"   : price = 2000; break;
            case "딸기" : price = 500; break;
            case "귤"    : price = 300; break;
            default      : price = 0; break;
        }

        System.out.println("지불하실 금액은 총 "+(amount*price)+"원 입니다.");
    }
}
```

연습 3-1

은행에서 개설할 수 있는 계좌의 특징과 행동을 생각해 보고 [그림 3-4]와 같이 표현해 보세요.

답안

특징	
계좌번호	
예금주	
개설일자	
잔액	
행동	
입금한다	
출금한다	
잔액을 조회한다	

연습 3-2

[연습 3-1]에서 작성했던 것을 토대로 Account 클래스를 작성해보세요.

답안

```
public class Account {0  
    String account_num;      //계좌번호  
    String depositor;        //예금주  
    String date;              //개설일자  
    int balance;              //잔액  
  
    void deposit(int money){  
        this.balance += money;  
        System.out.println("[계좌번호:"+account_num+"] "+money+", 잔액:"+balance);  
    }  
  
    void withdraw(int money){  
        if(this.balance-money<0)  
            System.out.println("잔액이 부족합니다.");  
        else{  
            this.balance-=money;  
            System.out.println("[계좌번호:"+account_num+"] -"+money+", 잔액:"+balance);  
        }  
    }  
  
    void checkBalance(){  
        System.out.println("[계좌번호:"+account_num+"] 잔액:"+balance);  
    }  
}
```

연습 3-3

2015년 8월 31일, 홍길동 씨가 OO은행에서 100,000원으로 계좌를 개설하였습니다. 계좌번호는 1002-03-0001000이었습니다. 홍길동 씨가 개설한 계좌의 객체를 [연습 3-2]에서 작성한 Account 클래스를 사용하여 생성하고, 다음과 같은 실행 결과가 나오게 코드를 작성하세요.

실행 결과

[계좌번호:1002-03-0001000] 잔액:100000

답안

```
public class AccountExample {  
    public static void main(String[] args) {  
        Account account = new Account();  
  
        account.account_num = "1002-03-0001000";  
        account.depositor = "홍길동";  
        account.date = "2015-08-31";  
        account.balance = 100000;  
  
        account.checkBalance();  
    }  
}
```

연습 3-4

[연습 3-2]에서 작성한 Account 클래스에 계좌번호, 예금주, 잔액을 선언과 동시에 초기화할 수 있는 생성자와 디폴트 생성자를 만드세요.

답안

```
public class Account {  
  
    String account_num; //계좌번호  
    String depositor; //예금주  
    String date; //개설일자  
    int balance; //잔액  
  
    Account(String account_num, String depositor, int balance) {  
        super();  
        this.account_num = account_num;  
        this.depositor = depositor;  
        this.balance = balance;  
    }  
  
    Account(){  
    }  
  
    void deposit(int money){  
        this.balance += money;  
        System.out.println("[계좌번호:"+account_num+"] "+"+"+money+", 잔액:"+balance);  
    }  
}
```

```

    }

    void withdraw(int money){
        if(this.balance-money<0)
            System.out.println("잔액이 부족합니다.");
        else{
            this.balance-=money;
            System.out.println("[계좌번호:"+account_num+"] -="+money+", 잔액:"+balance);
        }
    }

    void checkBalance(){
        System.out.println("[계좌번호:"+account_num+"] 잔액:"+balance);
    }
}

```

연습 3-5

다음과 같은 실행 결과가 나오도록 Account 클래스를 사용하여 5개의 계좌 객체를 생성하고, 그 정보를 출력하는 코드를 작성하세요.

TIP 배열과 반복문을 사용하면 코드가 짧아집니다.

실행 결과

```

=====
당일 개설 계좌 명단 =====
계좌번호: 1002-03-0001000, 예금주: 홍길동, 잔액: 100000원
계좌번호: 1002-06-0701421, 예금주: 박채윤, 잔액: 20000000원
계좌번호: 1002-01-9891123, 예금주: 이영희, 잔액: 360433원
계좌번호: 1002-05-0801000, 예금주: 강진철, 잔액: 50000원
계좌번호: 1002-02-0150802, 예금주: 임성수, 잔액: 1000000000원
=====
```

답안

```

public class AccountExample2 {

    public static void main(String[] args) {
        Account acc_arr[] = new Account[5];

        acc_arr[0] = new Account("1002-03-0001000","홍길동",100000);
        acc_arr[1] = new Account("1002-06-0701421","박채윤",20000000);
        acc_arr[2] = new Account("1002-04-0002940","이영희",360433);
    }
}
```

```

acc_arr[3] = new Account("1002-05-0801000","강진철",50000);
acc_arr[4] = new Account("1002-02-0150802","임성수",1000000000);

System.out.println("===== 당일 개설 계좌 명단 =====");
for(int i=0; i<acc_arr.length; i++)
    System.out.println("계좌번호:"+acc_arr[i].account_num+", 예금주:"+acc_
arr[i].depositor+", 잔액:"+acc_arr[i].balance);
    System.out.println("=====");
}
}

```

연습 4-1

김진영 씨는 회사에 입사 후 한도가 3,000만 원까지 가능한 마이너스 통장을 은행에서 개설하였습니다. 다음 코드를 실행했을 때 코드 아래의 실행 결과가 나오도록 [연습 3-2]에서 작성한 Account 클래스를 상속받아 MinusAccount 클래스를 작성해 보세요.

```

public class MinusAccountExample {
    public static void main(String[] args) {
        MinusAccount maccount = new MinusAccount();

        maccount.account_num = "1002-03-0001050";
        maccount.depositor = "김진형";
        maccount.date = "2013-01-04";
        maccount.balance = 100000;

        maccount.withdraw(40000);
        maccount.withdraw(40000);
        maccount.withdraw(40000);
        maccount.withdraw(40000);
    }
}

```

실행 결과

```

[계좌번호:1002-03-0001050] -40000, 잔액:60000, 남은 한도:30000
[계좌번호:1002-03-0001050] -40000, 잔액:20000, 남은 한도:30000
[계좌번호:1002-03-0001050] -40000, 잔액:0, 남은 한도:10000
[계좌번호:1002-03-0001050] 출금 실패! 마이너스 통장의 한도가 부족합니다. 남은 한도: 10000

```

답안

```
public class MinusAccount extends Account {  
    int limit = 30000;  
    void withdraw(int money){  
        if(this.balance-money<0){  
            if(this.balance-money<-limit){  
                System.out.println("[계좌번호:"+account_num+"] 출금 실패! 마이너스 통  
장의 한도가 부족합니다. 남은 한도: "+this.limit);  
            }else{  
                int tmp = money-this.balance;  
                this.balance=0;  
                this.limit-=tmp;  
                System.out.println("[계좌번호:"+account_num+"] -"+money+", 잔액:"+  
balance+", 남은 한도:"+this.limit);  
            }  
        }  
        else{  
            this.balance-=money;  
            System.out.println("[계좌번호:"+account_num+"] -"+money+", 잔액:"+balance+",  
남은 한도:"+this.limit);  
        }  
    }  
    void checkBalance(){  
        super.checkBalance();  
        System.out.println("[계좌번호:"+account_num+"] 남은 한도:"+this.limit);  
    }  
}
```

연습 6-1

[연습 3-5]의 실행 결과를 account.txt 파일에 써보세요.

연습 3-5의 실행 결과

```
===== 당일 개설 계좌 명단 =====  
계좌번호: 1002-03-0001000, 예금주: 홍길동, 잔액: 100000원  
계좌번호: 1002-06-0701421, 예금주: 박채윤, 잔액: 20000000원  
계좌번호: 1002-01-9891123, 예금주: 이영희, 잔액: 360433원  
계좌번호: 1002-05-0801000, 예금주: 강진철, 잔액: 50000원  
계좌번호: 1002-02-0150802, 예금주: 임성수, 잔액: 1000000000원  
=====
```

답안

```
class Accountwrite {
    public static void main(String[] args) {

        try {
            Account acc_arr[] = new Account[5];

            acc_arr[0] = new Account("1002-03-0001000","홍길동",100000);
            acc_arr[1] = new Account("1002-06-0701421","박채윤",20000000);
            acc_arr[2] = new Account("1002-04-0002940","이영희",360433);
            acc_arr[3] = new Account("1002-05-0801000","강진철",50000);
            acc_arr[4] = new Account("1002-02-0150802","임성수",1000000000);

            FileWriter fw = new FileWriter("accounts.txt");

            fw.write("===== 당일 개설 계좌 명단 =====\n");

            for (int i = 0; i < acc_arr.length; i++) {
                fw.write("계좌번호:"+acc_arr[i].account_num+", 예금주:"+acc_arr[i].
depositor+", 잔액:"+acc_arr[i].balance+"\n");
            }
            fw.write("===== ===== ===== =====\n");
            fw.close();

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

연습 6-2

[연습 6-1]에서 작성했던 account.txt 파일을 읽는 프로그램을 작성해보세요.

답안

```
class Accountread {
    public static void main(String[] args) {

        try {
            FileReader fr = new FileReader("accounts.txt");

            int i;
```

```

        while ((i = fr.read()) != -1) {
            char c = (char) i;
            System.out.print(c);
        }
        fr.close();
    } catch (IOException e) {
        System.out.println(e);
    }
}
}

```

연습 6-3

실행 결과처럼 '1'을 누르면 계좌를 생성하고, '2'를 누르면 현재 개설된 계좌 정보들을 출력하고, '3'을 누르면 프로그램을 종료하는 은행 시스템을 작성해봅시다.

실행 결과

원하는 서비스의 번호를 입력하세요. 1)계좌 개설, 2)현재 개설된 계좌 정보 보기, 3)시스템 종료

1

예금주:

이지현

예금할 금액:

120040

예금주 이지현 님 앞으로 생성된 계좌 번호는 1002-72-1879206입니다.

원하는 서비스의 번호를 입력하세요. 1)계좌 개설, 2)현재 개설된 계좌 정보 보기, 3)시스템 종료

1

예금주:

임지수

예금할 금액:

65000

예금주 임지수 님 앞으로 생성된 계좌 번호는 1002-59-8189877입니다.

원하는 서비스의 번호를 입력하세요. 1)계좌 개설, 2)현재 개설된 계좌 정보 보기, 3)시스템 종료

2

***** 개설된 계좌 정보 *****

계좌번호:1002-72-1879206, 예금주:이지현, 잔액:120040

계좌번호:1002-59-8189877, 예금주:임지수, 잔액:65000

원하는 서비스의 번호를 입력하세요. 1)계좌 개설, 2)현재 개설된 계좌 정보 보기, 3)시스템 종료

3

답안

```
import java.util.Calendar;
import java.util.Scanner;

public class BankSystem {

    public static void main(String[] args) {

        Account acc_arr[] = new Account[100];
        int acc_idx = 0;
        boolean flag = true;

        Scanner sc = new Scanner(System.in);

        while (flag) {
            System.out.println("원하는 서비스의 번호를 입력하세요. 1)계좌 개설, 2)현재
개설된 계좌 정보 보기, 3)시스템 종료");

            int service_num = sc.nextInt();

            switch (service_num) {
                case 1:
                    System.out.println("예금주:");
                    Account acc = new Account();
                    acc.depositor = sc.next();

                    System.out.println("예금할 금액:");
                    acc.balance = sc.nextInt();

                    Calendar cal = Calendar.getInstance();
                    acc.date = cal.get(Calendar.YEAR) + "-" + (cal.get(Calendar.MONTH) + 1) +
"-"+ cal.get(Calendar.DATE);

                    acc.account_num = "1002-"+ (int)((Math.random() * 99) + 1) + "-" +
(int)((Math.random() * 9999999) + 1);

                    System.out.println("예금주 " + acc.depositor + " 님 앞으로 생성된 계좌
번호는 " + acc.account_num + " 입니다.");
                    System.out.println();
                    acc_arr[acc_idx++] = acc;
                    break;
            }
        }
    }
}
```

```
        case 2:
            System.out.println("***** 개설된 계좌 정보 *****");
            for(int i=0; i<acc_idx; i++)
                System.out.println("계좌번호:"+acc_arr[i].account_num+", 예금주:
"+acc_arr[i].depositor+", 잔액:"+acc_arr[i].balance);
            System.out.println("*****");
            System.out.println();
            break;

        case 3:
            flag = false;
            break;

        default:
            break;
    }
}
```

연습 7-1

[연습 3-4]에서 작성한 Account 클래스를 직렬화 클래스로 만들어 보세요.

답안

```
public class Account implements java.io.Serializable{  
  
    String account_num;    //계좌번호  
    String depositor;     //예금주  
    String date;          //개설일자  
    int balance;          //잔액  
  
    Account(String account_num, String depositor, int balance) {  
        super();  
        this.account_num = account_num;  
        this.depositor = depositor;  
        this.balance = balance;  
    }  
  
    Account(){  
    }  
}
```

```

void deposit(int money){
    this.balance += money;
    System.out.println("[계좌번호:"+account_num+"] +" +money+, 잔액:"+balance);
}

void withdraw(int money){
    if(this.balance-money<0)
        System.out.println("잔액이 부족합니다.");
    else{
        this.balance-=money;
        System.out.println("[계좌번호:"+account_num+"] -"+money+, 잔액:"+balance);
    }
}

void checkBalance(){
    System.out.println("[계좌번호:"+account_num+"] 예금주:잔액:"+balance);
}

```

연습 7-2

다음과 같은 정보를 가진 Account 클래스의 객체를 직렬화하여 account2.dat 파일에 저장해보세요.

계좌번호: 1002-47-2881299, 예금주: 김성현, 잔액: 240000원
 계좌번호: 1002-21-5440400, 예금주: 이시원, 잔액: 56000원
 계좌번호: 1002-64-1090347, 예금주: 홍진표, 잔액: 9400000원

답안

```

public class Account2write {

    public static void main(String[] args) {
        try {
            ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream("account2.dat"));

            Account acc1 = new Account("1002-47-2881299", "김성현", 240000);
            out.writeObject(acc1);

            Account acc2 = new Account("1002-21-5440400", "이시원", 56000);
            out.writeObject(acc2);
        }
    }
}

```

```
        Account acc3 = new Account("1002-64-1090347","홍진표",9400000);
        out.writeObject(acc3);

        out.close();

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

연습 7-3

[연습 7-2]에서 작성한 account2.dat 파일을 읽어 Account 클래스의 객체들을 출력해보세요.

출력 예시

```
[계좌번호:1002-47-2881299] 예금주:잔액:240000
[계좌번호:1002-21-5440400] 예금주:잔액:56000
[계좌번호:1002-64-1090347] 예금주:잔액:9400000
더 이상 읽을 데이터가 없습니다
```

답안

```
class Account2read {
    public static void main(String[] args) {

        try {
            ObjectInputStream in = new ObjectInputStream(new
FileInputStream("account2.dat"));
            Account acc;

            while ((acc = (Account) in.readObject()) != null)
                acc.checkBalance();
            in.close();

        } catch (EOFException e) {
            System.out.println("더 이상 읽을 데이터가 없습니다");
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```
