

200명의 베타리더가 검토한 10년 베스트 셀러의 기초편
코딩을 처음 배우는 사람을 위해 세심하게 배려한 책

예약판매용 미리보기

2019.12.18 출시

Java의 정석

누구나 쉽고 빠르게 이해할 수 있도록 잘게 세분화하여 정리
기본원리의 자세한 설명으로 암기보다 이해위주의 학습유도
특히 객체지향개념을 쉬우면서도 자세하고 체계적으로 설명

소스 및 동영상 다운로드 | <http://github.com/castello>
QA게시판 | <http://www.codechobo.com>

기초 편

남궁 성 지음



도우출판

머리말

왜 자바를 배워야 할까요?

자바(Java)는 웹(web)과 모바일(안드로이드)을 비롯한 다양한 분야에서 사용되는 가장 인기 있는 언어이기 때문입니다. 그리고 취업시장 특히 국내에서 자바 개발자를 압도적으로 선호하고 있는 현실입니다. 마지막으로 자바를 통해 컴퓨터 과학 관련 지식과 알고리즘을 배우는데 있어서 다른 언어보다 자바가 유리하기 때문이라고 말씀 드릴 수 있습니다.

자바의 정석은 어떤 책인가요?

10년 넘게 국내 자바 베스트 셀러 자리를 지켜온 책입니다. 책 내용의 우수성은 이미 수많은 독자분들에 의해 검증되었고요. 단순히 자바에 대한 것만이 아니라 프로그래밍을 배우는데 필요한 기본기를 쉽고 빠짐없이 자세하게 설명합니다.(15년 동안 저자가 직접 독자분들의 질문을 빠짐없이 답변). 특히 객체지향개념은 프로그래밍에 있어서 매우 중요한 역할을 하는데, 대부분의 저자들은 쉬워보이는 책을 만들기 위해 객체지향개념에 대한 설명을 소홀히 하고 있습니다. 그러나 자바의 정석은 객체지향개념을 자세하고 원리까지 깊이 있게 설명합니다.

자바의 정석 기초편과 자바의 정석의 차이는?

자바의 정석은 전공자나 프로그래밍을 직업으로 삼으려는 사람을 대상으로 집필한 책이라서 실무에 적용할 수 있는 수준의 실력을 갖추게 하는 것이 목표입니다. 그러나 요즘 코딩 열풍이 불기 시작하면서 프로그래밍을 배우려는 사람들이 많아지고 좀더 쉽게 프로그래밍을 접할 수 있기를 원하는 독자들의 요구가 늘어났습니다. 이에 부응하려면 난이도를 낮춘 입문서가 필요하다고 생각해서 집필한 책이 바로 자바의 정석 기초편입니다. 그렇다고 해서 내용이 부실한 것은 아닙니다. 기본기는 착실히 다져주면서 응용부분에 대한 내용만 줄였을 뿐입니다. 수업시간에 프로그래밍을 어려워하는 학생과 소통하며 난이도를 조정하였고, 이미 200명이 넘는 베타리더들에 의해 검증되고 호평받았습니다.

이 책으로 공부하는 방법을 알려주세요.

1장부터 9장까지는 자바 프로그래밍을 하는데 필수적인 기본적인 내용을 다루었습니다. 별책 부록인 핵심요약 핸드북(pdf)을 한 번 읽어보면 자바에 대한 전체적인 윤곽이 잡힐 것입니다. 앞부분부터 모든 것을 완전히 공부하는 것보다 그림을 그리듯이 전체적인 밀그림을 그려가면서 점차적으로 세부적인 부분을 완성해 가는 것이 좋습니다.

그 다음에는 1장부터 하나하나 자세히 공부해 나갑니다. 처음에는 객체지향개념부분인 6장과 7장이 어렵겠지만 이해되는 만큼만 이해하고 넘어가세요. 이렇게 3~4번 반복한 다음에는 6장과 7장을 집중적으로 5번 정도 반복하세요. 생각보다 시간 많이 안걸립니다. 이정도면 자바에 대한 기초는 확실해집니다. 이제 11장과 14장을 공부하세요. 11장은 이책의 전체적인 수

준을 봤을 때 난이도가 높은 편이기 때문에 처음에는 '어떠한 클래스들이 있고 어떻게 사용하는구나.'라는 정도만 이해하고 반복을 통해 완전히 이해하시기 바랍니다. 나중에 자료구조를 배울 때 많은 도움이 될 겁니다. 12장, 13장, 15장은 필요할 때 공부하셔도 좋습니다.

공부하다 모르는 것 있으면 책 한번 더 읽어보고 그래도 모르겠으면 저에게 질문하세요. 책 관련 질문은 코드초보스터디(<https://cafe.naver.com/javachobostudy>)에서 제가 직접 자세히 답변해드리고 있습니다. 벌써 15년째 해오고 있습니다.

The screenshot shows a Cafe Naver profile page. At the top, there's a banner with the text '전문가와 함께하는 StudyDesk' and '남궁성의 코드초보스터디(java, c언어)'. Below the banner, the URL 'http://cafe.naver.com/javachobostudy' and the establishment date '(Since 2004.02.24)' are displayed. The main content area has tabs for '전체글보기', '필수자비강의', '초보프로그래미에게', '이미지모아보기', and '베스트게시글'. On the left, there's a sidebar with '카페정보' (including 'C언어정석 남궁성' and 'since 2004.02.24.'), '나의활동' (with a count of 146,291), and sections for '업데이트된 글', '즐겨찾는 멤버', '게시판 구독수', and '우리카페멤 수'. The right side shows a list of posts with columns for '제목', '작성자', '작성일', '조회', and ' 좋아요'. The first post is '[꿀독] 자동등업은 여기에 댓글 5개달아주세요. [22265]' by 남궁성 on 2016.09.28. with 8,506 views and 12 likes.

독자분들이 혼자서 공부하실 수 있도록 유튜브(youtube.com)에 무료강좌를 제공하고 있습니다. 계속 업데이트 될 예정이니 꼭 구독해주시고 좋아요도 많이 눌러주세요.



맺음말

이 책이 나오기까지 모든 과정을 함께 해준 주연, 직접 실습해가며 꼼꼼하게 리뷰해준 나의 제자들, 작은 오타까지 싸그리 잡아주신 200여명의 베타리더분들, 옆에서 조언을 아끼지 않으며 응원해준 우리 까팀 형제와 기적 멤버들 모두 고맙습니다. 바쁘다는 핑계로 놀아주지 못한 아이들아 미안하다. 그리고 묵묵히 가정을 안녕히 지켜주는 아내에게 사랑과 감사의 마음을 전합니다.

저자 남궁 성

목차

Chapter 1 자바를 시작하기 전에

01	자바(Java)란?	2
02	자바의 역사	3
03	자바의 특징	4
04	자바 가상 머신(JVM)	6
05	자바 개발도구(JDK) 설치하기	7
06	자바 개발도구(JDK) 설정하기	11
07	자바 API문서 설치하기	15
08	첫 번째 자바 프로그램 작성하기	16
09	자바 프로그램의 실행과정	18
10	이클립스 설치하기	19
11	이클립스로 자바 프로그램 개발하기	23
12	이클립스의 뷰, 퍼스펙티브, 워크스페이스	26
13	이클립스 단축키	28
14	이클립스의 자동 완성 기능	30
15	주석(comment)	32
16	자주 발생하는 에러와 해결방법	34
17	책의 소스와 강의자료 다운로드	36
18	이클립스로 소스파일 가져오기	38
19	이클립스에서 소스파일 내보내기	41

Chapter 2 변수

01	화면에 글자 출력하기 – print()과 println()	46
02	덧셈 뺄셈 계산하기	47
03	변수의 선언과 저장	48
04	변수의 타입	50
05	상수와 리터럴	51
06	리터럴의 타입과 접미사	52
07	문자 리터럴과 문자열 리터럴	53
08	문자열 결합	54
09	두 변수의 값 바꾸기	55

10	기본형과 참조형	56
11	기본형의 종류와 범위	57
12	printf를 이용한 출력	58
13	printf를 이용한 출력 예제	59
14	화면으로부터 입력받기	61
15	정수형의 오버플로우	62
16	부호있는 정수의 오버플로우	64
17	타입 간의 변환방법	66
	연습문제	67

Chapter 3 연산자

01	연산자와 피연산자	70
02	연산자의 종류	71
03	연산자의 우선순위	72
04	연산자의 결합규칙	73
05	증감 연산자 ++과 —	74
06	부호 연산자	76
07	형변환 연산자	77
08	자동 형변환	78
09	사칙 연산자	79
10	산술 변환	80
11	Math.round()로 반올림하기	83
12	나머지 연산자	84
13	비교 연산자	85
14	문자열의 비교	86
15	논리 연산자 && !	87
16	논리 부정 연산자	90
17	조건 연산자	91
18	대입 연산자	93
19	복합 대입 연산자	94
	연습문제	95

Chapter 4 조건문과 반복문

01 if문	98
02 조건식의 다양한 예	99
03 블럭{}	100
04 if–else문	101
05 if–else if문	102
06 if–else if문 예제	103
07 중첩 if문	104
08 중첩 if문 예제	105
09 switch문	106
10 switch문의 제약조건	107
11 switch문의 제약조건 예제	108
12 임의의 정수만들기 Math.random()	109
13 for문	110
14 for문 예제	112
15 중첩 for문	113
16 while문	115
17 while문 예제1	116
18 while문 예제2	117
19 do–while문	118
20 break문	119
21 continue문	120
22 break문과 continue문 예제	121
23 이름 붙은 반복문	122
24 이름 붙은 반복문 예제	123
연습 문제	125

Chapter 5 배열

01 배열이란?	130
02 배열의 선언과 생성	131
03 배열의 인덱스	132

04	배열의 길이(배열이름.length)	133
05	배열의 초기화	134
06	배열의 출력	135
07	배열의 출력 예제	136
08	배열의 활용(1) – 총합과 평균	137
09	배열의 활용(2) – 최대값과 최소값	138
10	배열의 활용(3) – 섞기(shuffle)	139
11	배열의 활용(4) – 로또 번호 만들기	140
12	String 배열의 선언과 생성	141
13	String 배열의 초기화	142
14	String 클래스	143
15	String 클래스의 주요 메서드	144
16	커맨드 라인을 통해 입력받기	145
17	이클립스에서 커マン드라인 매개변수 입력하기	146
18	2차원 배열의 선언	147
19	2차원 배열의 인덱스	148
20	2차원 배열의 초기화	149
21	2차원 배열의 초기화 예제1	150
22	2차원 배열의 초기화 예제2	151
23	2차원 배열의 초기화 예제3	152
24	Arrays로 배열 다루기	153
	연습 문제	154

Chapter 6 객체지향 프로그래밍 I

01	객체지향 언어	160
02	클래스와 객체	161
03	객체의 구성요소 – 속성과 기능	162
04	객체와 인스턴스	163
05	한 파일에 여러 클래스 작성하기	164
06	객체의 생성과 사용	165
07	객체의 생성과 사용 예제	168
08	객체배열	169

09	클래스의 정의(1) – 데이터와 함수의 결합	170
10	클래스의 정의(2) – 사용자 정의 타입	171
11	선언위치에 따른 변수의 종류	173
12	클래스 변수와 인스턴스 변수	174
13	클래스 변수와 인스턴스 변수 예제	175
14	메서드란?	176
15	메서드의 선언부	177
16	메서드의 구현부	178
17	메서드의 호출	179
18	메서드의 실행 흐름	180
19	메서드의 실행 흐름 예제	181
20	return문	182
21	반환값	183
22	호출스택(call stack)	184
23	기본형 매개변수	185
24	참조형 매개변수	186
25	참조형 반환타입	187
26	static 메서드와 인스턴스 메서드	188
27	static 메서드와 인스턴스 메서드 예제	189
28	static을 언제 붙여야 할까?	190
29	메서드 간의 호출과 참조	191
30	오버로딩(overloading)	192
31	오버로딩(overloading) 예제	194
32	생성자(constructor)	195
33	기본 생성자(default constructor)	196
34	매개변수가 있는 생성자	198
35	매개변수가 있는 생성자 예제	199
36	생성자에서 다른 생성자 호출하기 – this()	200
37	객체 자신을 가리키는 참조변수 – this	202
38	변수의 초기화	203
39	멤버변수의 초기화	204
40	멤버변수의 초기화 예제1	205
41	멤버변수의 초기화 예제2	206
	연습 문제	207

Chapter 7 객체지향 프로그래밍 II

01	상속	222
02	상속 예제	224
03	클래스 간의 관계 – 포함관계	225
04	클래스 간의 관계 결정하기	226
05	단일 상속(single inheritance)	227
06	Object클래스 – 모든 클래스의 조상	228
07	오버라이딩(overriding)	229
08	오버라이딩의 조건	230
09	오버로딩 vs. 오버라이딩	231
10	참조변수 super	232
11	super() – 조상의 생성자	233
12	패키지(package)	234
13	패키지의 선언	235
14	클래스 패스(classpath)	236
15	import문	237
16	static import문	238
17	제어자(modifier)	239
18	static – 클래스의, 공통적인	240
19	final – 마지막의, 변경될 수 없는	241
20	abstract – 추상의, 미완성의	242
21	접근 제어자(access modifier)	243
22	캡슐화와 접근 제어자	244
23	다형성(polymorphism)	246
24	참조변수의 형변환	248
25	참조변수의 형변환 예제	249
26	instanceof 연산자	250
27	매개변수의 다형성	251
28	매개변수의 다형성 예제	253
29	여러 종류의 객체를 배열로 다루기	254
30	여러 종류의 객체를 배열로 다루기 예제	255
31	추상 클래스(Abstract class)	257
32	추상 메서드(Abstract method)	258

33	추상클래스의 작성	259
34	추상클래스의 작성 예제	261
35	인터페이스(interface)	263
36	인터페이스의 상속	264
37	인터페이스의 구현	265
38	인터페이스를 이용한 다형성	266
39	인터페이스의 장점	267
40	디폴트 메서드와 static메서드	268
41	디폴트 메서드와 static메서드 예제	269
42	내부 클래스(inner class)	270
43	내부 클래스의 종류와 특징	271
44	내부 클래스의 선언	272
45	내부 클래스의 제어자와 접근성	273
46	내부 클래스의 제어자와 접근성 예제1	274
47	내부 클래스의 제어자와 접근성 예제2	275
48	내부 클래스의 제어자와 접근성 예제3	276
49	내부 클래스의 제어자와 접근성 예제4	277
50	내부 클래스의 제어자와 접근성 예제5	278
51	익명 클래스(anonymous class)	279
52	익명 클래스(anonymous class) 예제	280
	연습 문제	281

Chapter 8 예외처리

01	프로그램 오류	292
02	예외 클래스의 계층구조	293
03	Exception과 RuntimeException	294
04	예외 처리하기 – try-catch문	295
05	try-catch문에서의 흐름	296
06	예외의 발생과 catch블럭	297
07	printStackTrace()와 getMessage()	299
08	멀티 catch블럭	300
09	예외 발생시키기	301

10 checked예외, unchecked예외	302
11 메서드에 예외 선언하기	303
12 메서드에 예외 선언하기 예제1	304
13 메서드에 예외 선언하기 예제2	305
14 finally블럭	306
15 사용자 정의 예외 만들기	307
16 사용자 정의 예외 만들기 예제	308
17 예외 되던지기(exception re-throwing)	310
18 연결된 예외(chained exception)	312
19 연결된 예외(chained exception) 예제	314
연습문제	316

Chapter 9 java.lang패키지와 유용한 클래스

01 Object클래스	324
02 Object클래스의 메서드 – equals()	325
03 equals()의 오버라이딩	326
04 Object클래스의 메서드 – hashCode()	327
05 Object클래스의 메서드 – toString()	328
06 toString()의 오버라이딩	329
07 String클래스	330
08 문자열(String)의 비교	331
09 문자열 리터럴(String리터럴)	332
10 빈 문자열(empty string)	333
11 String클래스의 생성자와 메서드	334
12 join()과 StringJoiner	337
13 문자열과 기본형 간의 변환	338
14 문자열과 기본형 간의 변환 예제	339
15 StringBuffer클래스	340
16 StringBuffer의 생성자	341
17 StringBuffer의 변경	342
18 StringBuffer의 비교	343
19 StringBuffer의 생성자와 메서드	344

20	StringBuffer의 생성자와 메서드 예제	346
21	StringBuilder	347
22	Math클래스	348
23	Math의 메서드	349
24	Math의 메서드 예제	350
25	래퍼(wrapper) 클래스	351
26	래퍼(wrapper) 클래스 예제	352
27	Number클래스	353
28	문자열을 숫자로 변환하기	354
29	문자열을 숫자로 변환하기 예제	355
30	오토박싱 & 언박싱	356
31	오토박싱 & 언박싱 예제	357
	연습문제	358

Chapter 10 날짜와 시간 & 형식화

01	날짜와 시간	366
02	Calendar클래스	367
03	Calendar 예제1	368
04	Calendar 예제2	370
05	Calendar 예제3	371
06	Calendar 예제4	372
07	Calendar 예제5	373
08	Date와 Calendar간의 변환	374
09	형식화 클래스	375
10	DecimalFormat	376
11	DecimalFormat 예제1	377
12	DecimalFormat 예제2	378
13	SimpleDateFormat	379
14	SimpleDateFormat 예제1	380
15	SimpleDateFormat 예제2	381
16	SimpleDateFormat 예제3	382
	연습문제	383

01	컬렉션 프레임워크	388
02	컬렉션 프레임워크의 핵심 인터페이스	389
03	Collection인터페이스	390
04	List인터페이스	391
05	Set인터페이스	392
06	Map인터페이스	393
07	ArrayList	394
08	ArrayList의 메서드	395
09	ArrayList 예제	396
10	ArrayList의 추가와 삭제	398
11	Java API소스보기	399
12	LinkedList	400
13	LinkedList의 추가와 삭제	401
14	ArrayList와 LinkedList의 비교	402
15	Stack과 Queue	403
16	Stack과 Queue의 메서드	404
17	Stack과 Queue 예제	405
18	인터페이스를 구현한 클래스 찾기	406
19	Stack과 Queue의 활용	407
20	Stack과 Queue의 활용 예제1	408
21	Stack과 Queue의 활용 예제2	409
22	Iterator, ListIterator, Enumeration	411
23	Iterator, ListIterator, Enumeration 예제	412
24	Map과 Iterator	413
25	Arrays의 메서드(1) – 복사	414
26	Arrays의 메서드(2) – 채우기, 정렬, 검색	415
27	Arrays의 메서드(3) – 비교와 출력	416
28	Arrays의 메서드(4) – 변환	417
29	Arrays의 메서드 예제	418
30	Comparator와 Comparable	420
31	Comparator와 Comparable 예제	421
32	Integer와 Comparable	422

33	Integer와 Comparable 예제	423
34	HashSet	424
35	HashSet 예제1	425
36	HashSet 예제2	426
37	HashSet 예제3	427
38	HashSet 예제4	428
39	TreeSet	429
40	이진 탐색 트리(binary search tree)	430
41	이진 탐색 트리의 저장과정	431
42	TreeSet의 메서드	432
43	TreeSet 예제1	433
44	TreeSet 예제2	434
45	TreeSet 예제3	435
46	HashMap과 Hashtable	436
47	HashMap의 키(key)와 값(value)	437
48	HashMap의 메서드	438
49	HashMap 예제1	439
50	HashMap 예제2	441
51	HashMap 예제3	442
52	Collections의 메서드 – 동기화	443
53	Collections의 메서드 – 변경불가, 싱글톤	444
54	Collections의 메서드 – 단일 컬렉션	445
55	Collections 예제	446
56	컬렉션 클래스 정리 & 요약	448
	연습 문제	449

Chapter 12 지네릭스, 열거형, 애너테이션

01	지네릭스(Generics)	458
02	타입 변수	459
03	타입 변수에 대입하기	460
04	지네릭스의 용어	461
05	지네릭 타입과 다형성	462

06	지네릭 타입과 다형성 예제	463
07	Iterator<E>	464
08	HashMap<K,V>	465
09	제한된 지네릭 클래스	466
10	제한된 지네릭 클래스 예제	467
11	지네릭스의 제약	468
12	와일드 카드	469
13	와일드 카드 예제	470
14	지네릭 메서드	471
15	지네릭 타입의 형변환	473
16	지네릭 타입의 제거	474
17	열거형(enum)	475
18	열거형의 정의와 사용	476
19	열거형의 조상 – java.lang.Enum	477
20	열거형 예제	478
21	열거형에 멤버 추가하기	479
22	열거형에 멤버 추가하기 예제	480
23	애너테이션이란?	481
24	표준 애너테이션	483
25	@Override	484
26	@Deprecated	485
27	@FunctionalInterface	486
28	@SuppressWarnings	487
29	메타 애너테이션	488
30	@Target	489
31	@Retention	490
32	@Documented, @Inherited	491
33	@Repeatable	492
34	애너테이션 타입 정의하기	493
35	애너테이션의 요소	494
36	모든 애너테이션의 조상	497
37	마커 애너테이션	498
38	애너테이션 요소의 규칙	499
39	애너테이션의 활용 예제	500
	연습문제	502

Chapter 13 쓰레드

01	프로세스(process)와 쓰레드(thread)	506
02	멀티쓰레딩의 장단점	507
03	쓰레드의 구현과 실행	508
04	쓰레드의 구현과 실행 예제	509
05	쓰레드의 실행 – start()	510
06	start()와 run()	511
07	main쓰레드	512
08	싱글쓰레드와 멀티쓰레드	513
09	싱글쓰레드와 멀티쓰레드 예제1	514
10	싱글쓰레드와 멀티쓰레드 예제2	515
11	쓰레드의 I/O블락킹(blocking)	517
12	쓰레드의 I/O블락킹(blocking) 예제1	518
13	쓰레드의 I/O블락킹(blocking) 예제2	519
14	쓰레드의 우선순위	520
15	쓰레드의 우선순위 예제	521
16	쓰레드 그룹(thread group)	523
17	쓰레드 그룹(thread group)의 메서드	524
18	데몬 쓰레드(daemon thread)	525
19	데몬 쓰레드(daemon thread) 예제	526
20	쓰레드의 상태	527
21	쓰레드의 실행제어	528
22	sleep()	529
23	sleep() 예제	530
24	interrupt()	531
25	interrupt() 예제	532
26	suspend(), resume(), stop()	533
27	suspend(), resume(), stop() 예제	534
28	join()과 yield()	535
29	join()과 yield() 예제	536
30	쓰레드의 동기화(synchronization)	537
31	synchronized를 이용한 동기화	538
32	synchronized를 이용한 동기화 예제1	539

33	synchronized를 이용한 동기화 예제2	540
34	wait()과 notify()	541
35	wait()과 notify() 예제1	542
36	wait()과 notify() 예제2	545
연습문제	548

Chapter 14 람다와 스트림

01	람다식(Lambda Expression)	552
02	람다식 작성하기	553
03	람다식의 예	554
04	람다식은 익명 함수? 익명 객체!	555
05	함수형 인터페이스(Functional Interface)	556
06	함수형 인터페이스 타입의 매개변수, 반환 타입	557
07	java.util.function패키지	559
08	java.util.function패키지 예제	561
09	Predicate의 결합	562
10	Predicate의 결합 예제	563
11	컬렉션 프레임워크와 함수형 인터페이스	564
12	컬렉션 프레임워크와 함수형 인터페이스 예제	565
13	메서드 참조	566
14	생성자의 메서드 참조	567
15	스트림(stream)	568
16	스트림의 특징	569
17	스트림 만들기 – 컬렉션	571
18	스트림 만들기 – 배열	572
19	스트림 만들기 – 임의의 수	573
20	스트림 만들기 – 특정 범위의 정수	574
21	스트림 만들기 – 람다식 iterate(), generate()	575
22	스트림 만들기 – 파일과 빈 스트림	576
23	스트림의 연산	577
24	스트림의 연산 – 중간연산	578
25	스트림의 연산 – 최종연산	579

26	스트림의 중간연산 – skip(), limit()	580
27	스트림의 중간연산 – filter(), distinct()	581
28	스트림의 중간연산 – sorted()	582
29	스트림의 중간연산 – Comparator의 메서드	583
30	스트림의 중간연산 – map()	585
31	스트림의 중간연산 – map() 예제	586
32	스트림의 중간연산 – peek()	587
33	스트림의 중간연산 – flatMap()	588
34	스트림의 중간연산 – flatMap() 예제	589
35	Optional<T>	590
36	Optional<T>객체 생성하기	591
37	Optional<T>객체의 값 가져오기	592
38	OptionalInt, OptionalLong, OptionalDouble	593
39	Optional<T> 예제	594
40	스트림의 최종연산 – forEach()	595
41	스트림의 최종연산 – 조건검사	596
42	스트림의 최종연산 – reduce()	597
43	스트림의 최종연산 – reduce()의 이해	598
44	스트림의 최종연산 – reduce() 예제	599
45	collect()와 Collectors	600
46	스트림을 컬렉션, 배열로 변환	601
47	스트림의 통계 – counting(), summingInt()	602
48	스트림을 리듀싱 – reducing()	603
49	스트림을 문자열로 결합 – joining()	604
50	스트림의 그룹화와 분할	605
51	스트림의 분할 – partitioningBy()	606
52	스트림의 분할 – partitioningBy() 예제	608
53	스트림의 그룹화 – groupingBy()	611
54	스트림의 그룹화 – groupingBy() 예제	613
55	스트림의 변환	618
	연습문제	620

01	입출력(I/O)과 스트림(stream)	624
02	바이트 기반 스트림 – InputStream, OutputStream	625
03	보조 스트림	626
04	문자기반 스트림 – Reader, Writer	627
05	바이트 기반 스트림과 문자 기반 스트림의 비교	628
06	InputStream과 OutputStream	629
07	InputStream과 OutputStream 예제1	630
08	InputStream과 OutputStream 예제2	631
09	InputStream과 OutputStream 예제3	632
10	FileInputStream과 FileOutputStream	634
11	FileInputStream과 FileOutputStream 예제1	635
12	FileInputStream과 FileOutputStream 예제2	636
13	FilterInputStream과 FilterOutputStream	637
14	BufferedInputStream	638
15	BufferedOutputStream	639
16	BufferedOutputStream 예제	640
17	SequenceInputStream	642
18	SequenceInputStream 예제	643
19	PrintStream	644
20	문자 기반 스트림 – Reader	645
21	문자 기반 스트림 – Writer	646
22	FileReader와 FileWriter	647
23	StringReader와 StringWriter	649
24	BufferedReader와 BufferedWriter	650
25	InputStreamReader, OutputStreamWriter	651
26	표준 입출력(Standard I/O)	653
27	표준 입출력의 대상변경	654
28	표준 입출력의 대상변경 예제	655
29	File	656
30	File 예제1	657
31	File 예제2	659
32	File 예제3	660

33	File 예제4	661
34	직렬화(serialization)	662
35	ObjectInputStream, ObjectOutputStream	663
36	직렬화가 가능한 클래스 만들기	665
37	직렬화 대상에서 제외시키기 – transient	666
38	직렬화와 역직렬화 예제1	667
39	직렬화와 역직렬화 예제2	668
40	직렬화와 역직렬화 예제3	669
	연습문제	670

Chapter 16 네트워킹

01	네트워킹(networking)이란?	676
02	클라이언트와 서버(client & server)	677
03	IP주소(IP address)	678
04	네트워크 주소와 호스트 주소	679
05	InetAddress클래스	680
06	InetAddress클래스 예제	681
07	URL(Uniform Resource Locator)	682
08	URL클래스	683
09	URL클래스 예제	684
10	URLConnection클래스	685
11	URLConnection클래스 예제1	687
12	URLConnection클래스 예제2	688
13	URLConnection클래스 예제3	689
14	소켓(socket) 프로그래밍	690
15	TCP와 UDP	691
16	TCP소켓 프로그래밍	692
17	Socket과 ServerSocket	693
18	TCP소켓 프로그래밍 예제1	694
19	TCP소켓 프로그래밍 예제2	696
20	UDP 소켓 프로그래밍 – Client	699
21	UDP 소켓 프로그래밍 – Server	700

C H A P T E R

1

자바를 시작하기 전에

getting started with Java

자바는 썬 마이크로시스템즈(Sun Microsystems, Inc. 이하 썬)에서 개발하여 1996년 1월에 공식적으로 발표한 객체지향 프로그래밍 언어이다.

자바의 가장 중요한 특징은 운영체제(Operating System, 플랫폼)에 독립적이라는 것이다. 자바로 작성된 프로그램은 운영체제의 종류에 관계없이 실행이 가능하기 때문에, 운영체제에 따라 프로그램을 전혀 변경하지 않고도 실행이 가능하다.

이러한 장점으로 인해 자바는 다양한 기종의 컴퓨터와 운영체제가 공존하는 인터넷 환경에 적합한 언어로써 인터넷의 발전과 함께 많은 사용자층을 확보할 수 있었다. 또한 객체지향개념과 기존의 다른 프로그래밍언어, 특히 C++의 장점을 채택하는 동시에 잘 사용되지 않는 부분은 과감히 제외시킴으로써 비교적 배우기 쉽고 이해하기 쉬운 간결한 표현이 가능하도록 했다.

자바는 풍부한 클래스 라이브러리(Java API)를 통해 프로그래밍에 필요한 요소들을 기본적으로 제공하기 때문에 자바 프로그래머는 단순히 이 클래스 라이브러리만을 잘 활용해도 강력한 기능의 자바 프로그램을 작성할 수 있다.

지금도 자바는 꾸준히 자바의 성능을 개선하여 새로운 버전을 발표하고 있으며, 모바일(J2ME)이나 대규모 기업환경(J2EE), XML 등의 다양한 최신 기술을 지원함으로써 그 활동 영역을 넓혀 가고 있다.

참고

2010년에 썬이 오라클(oracle)사에 인수되면서 이제 자바는 오라클사의 제품이 되었다.



미리보기용 pdf입니다.

자바의 역사는 1991년에 썬의 엔지니어들에 의해서 고안된 오크(Oak)라는 언어에서부터 시작되었다.

제임스 고슬링과 아서 밴 호프와 같은 썬의 엔지니어들의 원래 목표는 가전제품에 탑재될 소프트웨어를 만드는 것이었다. 처음에는 C++을 확장해서 사용하려 했지만 C++로는 그들의 목적을 이루기에 부족하다는 것을 깨달았다.

그래서 C++의 장점을 도입하고 단점을 보완한 새로운 언어를 개발하기에 이르렀다. Oak는 처음에는 가전제품이나 PDA와 같은 소형기기에 사용될 목적이었으나 여러 종류의 운영체제를 사용하는 컴퓨터들이 통신하는 인터넷이 등장하자 운영체제에 독립적인 Oak가 이에 적합하다고 판단하여 Oak를 인터넷에 적합하도록 그 개발 방향을 바꾸면서 이름을 자바(Java)로 변경하였으며, 자바로 개발한 웹브라우저인 ‘핫 자바(Hot java)’를 발표하고 그 다음 해인 1996년 1월에 자바의 정식 버전을 발표했다.

그 당시만 해도 자바로 작성된 애플릿(Applet)은 정적인 웹페이지에 사운드와 애니메이션 등의 멀티미디어적인 요소들을 제공할 수 있는 유일한 방법이었기 때문에 많은 인기를 얻고 단 기간에 많은 사용자층을 확보할 수 있었다.

그러나 보안상의 이유로 최신 웹브라우저에서 애플릿을 더 이상 지원하지 않게 되었다. 대신 서버 쪽 프로그래밍을 위한 서블릿(Servlet)과 JSP(Java Server Pages)가 더 많이 사용되고 있다. 그리고 구글의 스마트폰 운영체제인 안드로이드에서도 Java를 사용한다.

앞으로는 자바의 원래 목표였던 소규모 가전제품과 대규모 기업환경을 위한 소프트웨어개발 분야에 활발히 사용될 것으로 기대된다.

1. 운영체제에 독립적이다.

기존의 언어는 한 운영체제에 맞게 개발된 프로그램을 다른 종류의 운영체제에 적용하기 위해서는 많은 노력이 필요하였지만, 자바에서는 더 이상 그런 노력을 하지 않아도 된다. 이것은 일종의 에뮬레이터인 자바가상머신(JVM)을 통해서 가능한 것인데, 자바 응용프로그램은 운영체제나 하드웨어가 아닌 JVM하고만 통신하고 JVM이 자바 응용프로그램으로부터 전달 받은 명령을 해당 운영체제가 이해할 수 있도록 변환하여 전달한다. 자바로 작성된 프로그램은 운영체제에 독립적이지만 JVM은 운영체제에 종속적이어서 썬에서는 여러 운영체제에 설치할 수 있는 서로 다른 버전의 JVM을 제공하고 있다.

그래서 자바로 작성된 프로그램은 운영체제와 하드웨어에 관계없이 실행 가능하며 이것을 ‘한번 작성하면, 어디서나 실행된다.(Write once, run anywhere)’고 표현하기도 한다.

2. 객체지향언어이다.

자바는 프로그래밍의 대세로 자리 잡은 객체지향 프로그래밍언어(object-oriented programming language) 중의 하나로 객체지향개념의 특징인 상속, 캡슐화, 다형성이 잘 적용된 순수한 객체지향언어라는 평가를 받고 있다.

3. 비교적 배우기 쉽다.

자바의 연산자와 기본구문은 C++에서, 객체지향관련 구문은 스몰톡(small talk)이라는 객체지향언어에서 가져왔다. 이들 언어의 장점을 취하면서 복잡하고 불필요한 부분은 과감히 제거하여 단순화함으로서 쉽게 배울 수 있으며, 간결하고 이해하기 쉬운 코드를 작성할 수 있도록 하였다. 객체지향언어의 특징인 재사용성과 유지보수의 용이성 등의 많은 장점에도 불구하고 배우기가 어렵기 때문에 많은 사용자층을 확보하지 못했으나 자바의 간결하면서도 명료한 객체지향적 설계는 사용자들이 객체지향개념을 보다 쉽게 이해하고 활용할 수 있도록 하여 객체지향 프로그래밍의 저변확대에 크게 기여했다.

4. 자동 메모리 관리(Garbage Collection)

자바로 작성된 프로그램이 실행되면, 가비지컬렉터(garbage collector)가 자동적으로 메모리를 관리해주기 때문에 프로그래머는 메모리를 따로 관리 하지 않아도 된다. 가비지컬렉터가 없다면 프로그래머가 사용하지 않는 메모리를 체크하고 반환하는 일을 수동적으로 처리해야 할 것이다. 자동으로 메모리를 관리한다는 것이 다소 비효율적인 면도 있지만, 프로그래머가 보다 프로그래밍에 집중할 수 있도록 도와준다.

미리보기용 pdf입니다.

5. 네트워크와 분산처리를 지원한다.

인터넷과 대규모 분산환경을 염두에 둔 깨닭인지 풍부하고 다양한 네트워크 프로그래밍 라이브러리(Java API)를 통해 비교적 짧은 시간에 네트워크 관련 프로그램을 쉽게 개발할 수 있도록 지원한다.

6. 멀티쓰레드를 지원한다.

일반적으로 멀티쓰레드(multi-thread)의 지원은 사용되는 운영체제에 따라 구현방법도 상이하며, 처리 방식도 다르다. 그러나 자바에서 개발되는 멀티쓰레드 프로그램은 시스템과는 관계없이 구현가능하며, 관련된 라이브러리(Java API)가 제공되므로 구현이 쉽다. 그리고 여러 쓰레드에 대한 스케줄링(scheduling)을 자바 인터프리터가 담당하게 된다.

7. 동적 로딩(Dynamic Loading)을 지원한다.

보통 자바로 작성된 애플리케이션은 여러 개의 클래스로 구성되어 있다. 자바는 동적 로딩을 지원하기 때문에 실행 시에 모든 클래스가 로딩되지 않고 필요한 시점에 클래스를 로딩하여 사용할 수 있다는 장점이 있다. 그 외에도 일부 클래스가 변경되어도 전체 애플리케이션을 다시 컴파일하지 않아도 되며, 애플리케이션의 변경사항이 발생해도 비교적 적은 작업만으로도 처리할 수 있는 유연한 애플리케이션을 작성할 수 있다.

JVM은 ‘Java virtual machine’을 줄인 것으로 직역하면 ‘자바를 실행하기 위한 가상 기계’라고 할 수 있다. 가상 기계라는 말이 좀 어색하겠지만 영어권에서는 컴퓨터를 머신(machine)이라고도 부르기 때문에 ‘머신’이라는 용어대신 ‘컴퓨터’를 사용해서 ‘자바를 실행하기 위한 가상 컴퓨터’라고 이해하면 좋을 것이다.

‘가상 기계(virtual machine)’는 소프트웨어로 구현된 하드웨어를 뜻하는 넓은 의미의 용어이며, 컴퓨터의 성능이 향상됨에 따라 점점 더 많은 하드웨어들이 소프트웨어화되어 컴퓨터 속으로 들어오고 있다. 그 예로는 TV와 비디오를 소프트웨어화한 윈도우 미디어 플레이어라던가, 오디오 시스템을 소프트웨어화한 윈앰프(winamp) 등이 있다.

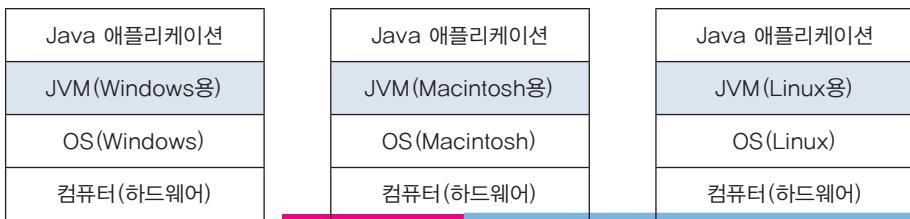
이와 마찬가지로 ‘가상 컴퓨터(virtual computer)’는 실제 컴퓨터(하드웨어)가 아닌 소프트웨어로 구현된 컴퓨터라는 뜻으로 컴퓨터 속의 컴퓨터라고 생각하면 된다.

자바로 작성된 애플리케이션은 모두 이 가상 컴퓨터(JVM)에서만 실행되기 때문에, 자바 애플리케이션이 실행되기 위해서는 반드시 JVM이 필요하다.



일반 애플리케이션의 코드는 OS만 거치고 하드웨어로 전달되는데(오른쪽 그림) Java애플리케이션은 JVM을 한 번 더 거치기 때문에(왼쪽 그림), 그리고 하드웨어에 맞게 완전히 컴파일된 상태가 아니고 실행 시에 해석(interpret)되기 때문에 속도가 느리다는 단점을 가지고 있다. 그러나 요즘엔 바이트코드(컴파일된 자바코드)를 하드웨어의 기계어로 바로 변환해주는 JIT컴파일러와 향상된 최적화 기술이 적용되어서 속도의 격차를 많이 줄였다.

위의 오른쪽 그림에서 볼 수 있듯이 일반 애플리케이션은 OS와 바로 맞붙어 있기 때문에 OS종속적이다. 그래서 다른 OS에서 실행시키기 위해서는 애플리케이션을 그 OS에 맞게 변경해야한다. 반면에 Java 애플리케이션은 JVM하고만 상호작용을 하기 때문에 OS와 하드웨어에 독립적이라 다른 OS에서도 프로그램의 변경없이 실행이 가능한 것이다. 단, JVM은 OS에 종속적이기 때문에 해당 OS에서 실행가능한 JVM이 필요하다.



미리보기용 pdf입니다.

자바로 프로그래밍을 하기 위해서는 먼저 JDK(Java Development Kit)를 설치해야 한다. JDK를 설치하면, 자바 가상머신(Java Virtual Machine, JVM)과 자바 클래스 라이브러리(Java API) 외에 자바를 개발하는데 필요한 프로그램들이 설치된다.

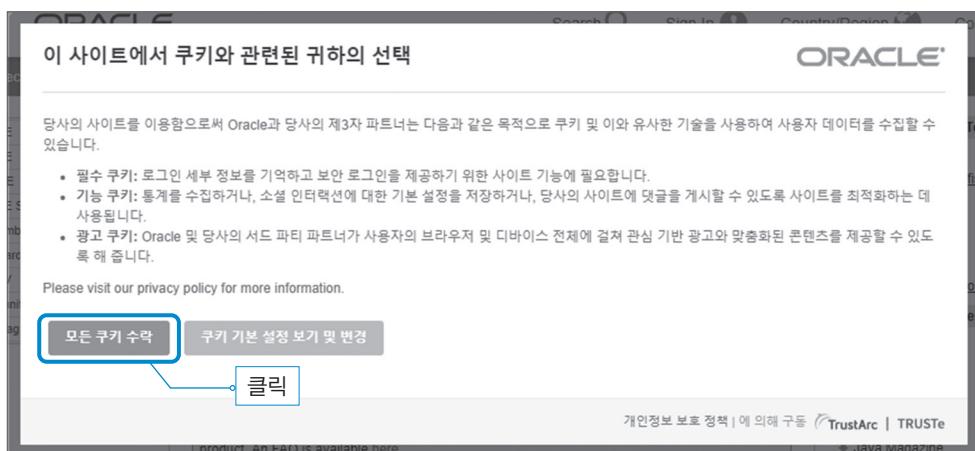
이 책을 학습하기 위해서는 JDK 8.0 이상의 버전이 필요하며 <http://java.sun.com/>에서 다운로드 받을 수 있다. JDK 1.5부터 Java 5이라고 부르기 시작했는데, JDK 1.7은 Java 7, JDK 1.8은 Java 8이라고 부르기도 한다. 앞으로 이 책에서는 Java 8을 JDK 1.8로 하겠다.

먼저 JDK 1.8을 다운로드하자. 구글에서 ‘java 8 download’로 검색하면 아래와 같은 화면이 나타난다.

- 1** 구글(google.com)에서 ‘java 8 download’로 검색 후 결과에서 링크를 클릭하자.



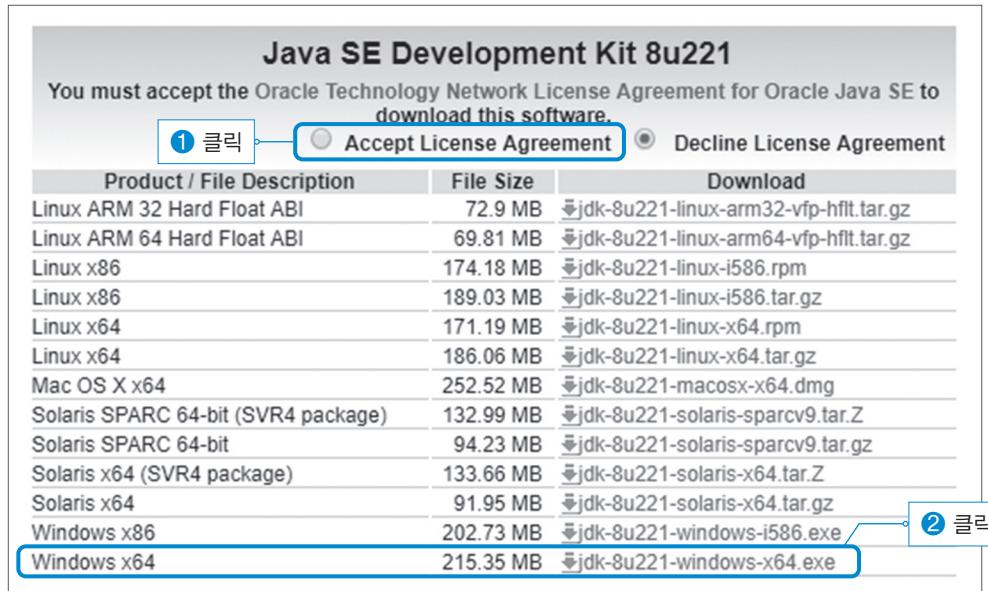
- 2** 아래와 같은 창이 나오면 ‘모든 쿠키 수락’을 클릭하자.



- ③** 아래의 화면에서 ‘Accept License Agreement’를 클릭하고, 64비트 원도우즈 사용자의 경우 ‘jdk-8u221-windows-x64.exe’를 클릭하고, 32비트 원도우즈 사용자는 ‘jdk-8u221-windows-i586.exe’를 클릭한다.

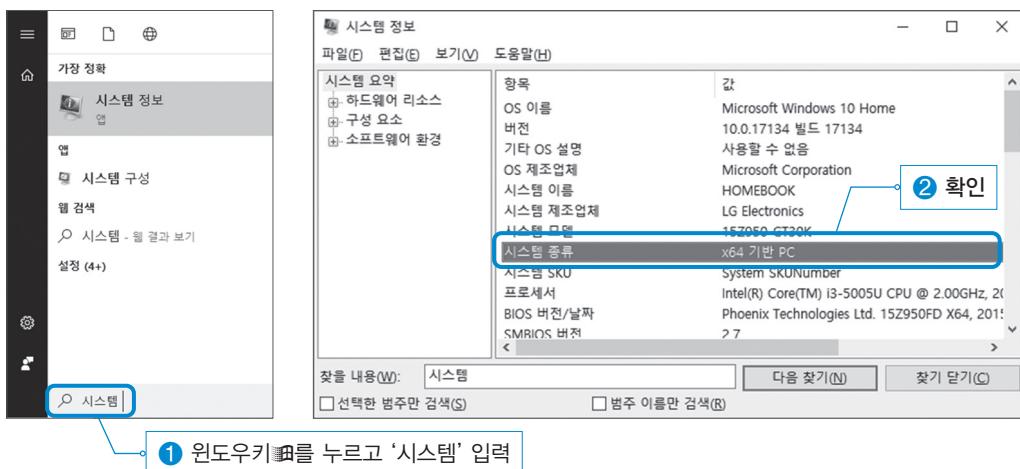
참고

오라클 계정 로그인 창이 나타나면, 로그인을 해야 다운로드가 진행된다. 계정이 없으면 계정을 만들어야 한다.



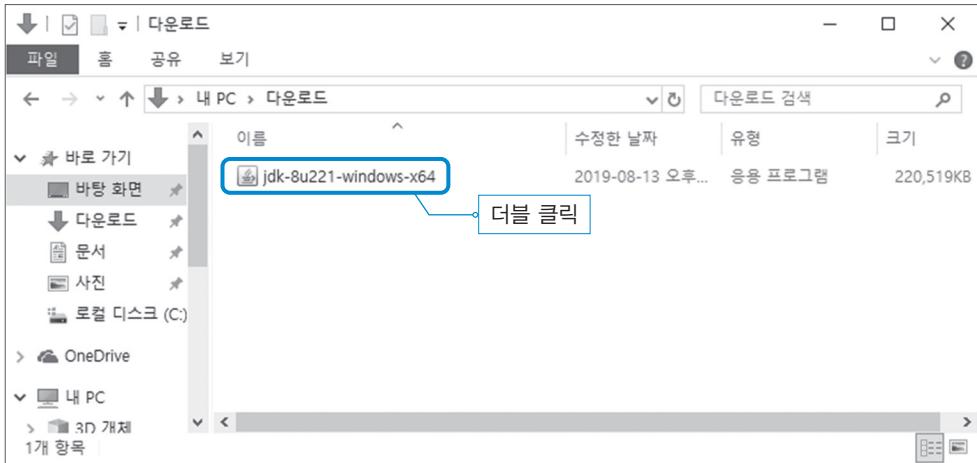
참고

원도우키를 누르고, ‘시스템’을 입력하고 ‘시스템 정보’를 클릭 하면 아래의 우측과 같은 창이 나타난다. 시스템 종류가 ‘x64 기반 PC’이면 64비트 원도우즈이고, 아니면 32비트 원도우즈이다.

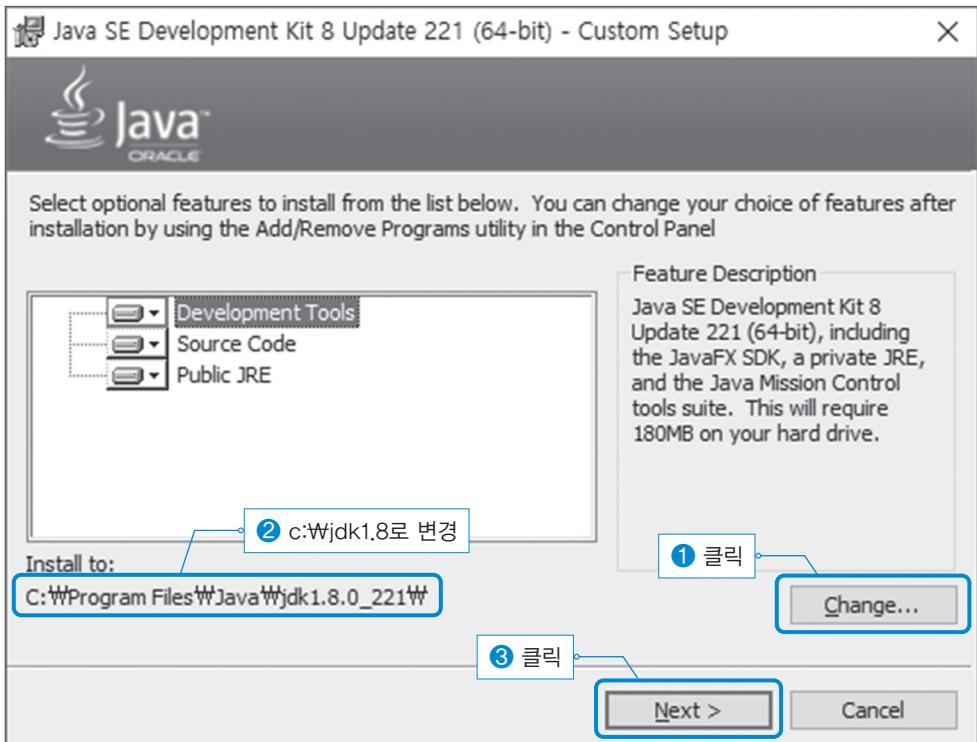


미리보기용 pdf입니다.

4 이전 단계에서 다운로드한 JDK설치파일을 더블 클릭해서 실행한다.



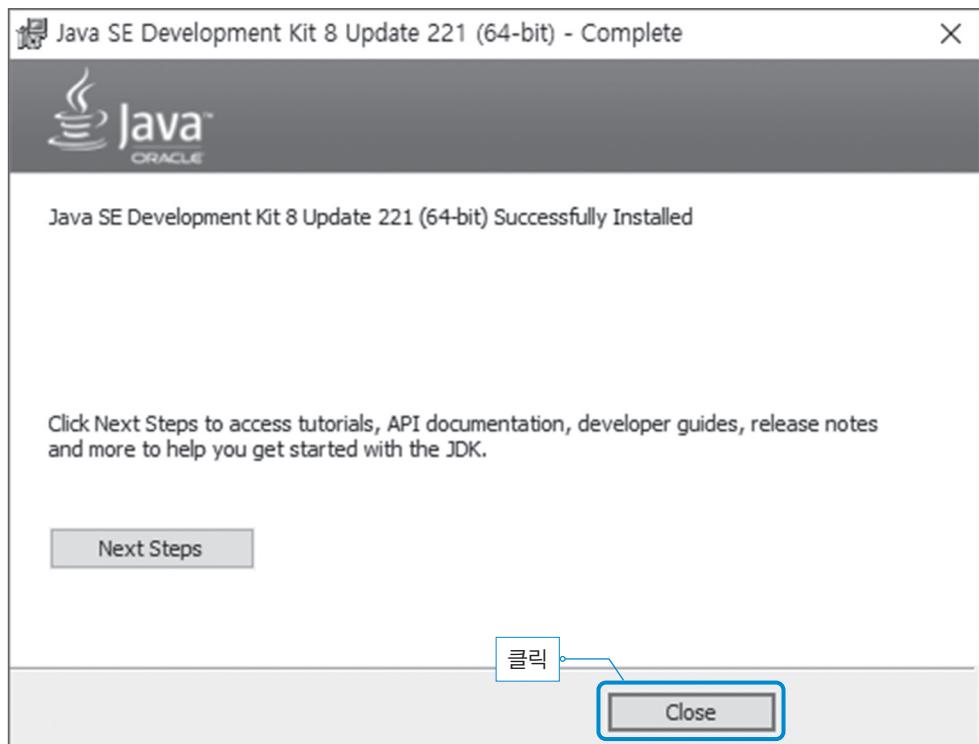
5 첫 화면에서 'Next >'버튼을 클릭하면 아래와 같은 화면이 나타나는데, 'Change...'버튼을 눌러서 설치할 위치를 'C:\jdk1.8'로 변경한다. 그리고 'Next >'버튼을 클릭한다. 이 후로는 특별히 설정할것 없다.



6 아래의 화면에서 '다음(N)>'버튼을 클릭한다.



7 아래와 같은 화면이 나타나면 설치가 잘된 것이다. 'Close'버튼을 누르면 설치가 완료된다.



미리보기용 pdf입니다.

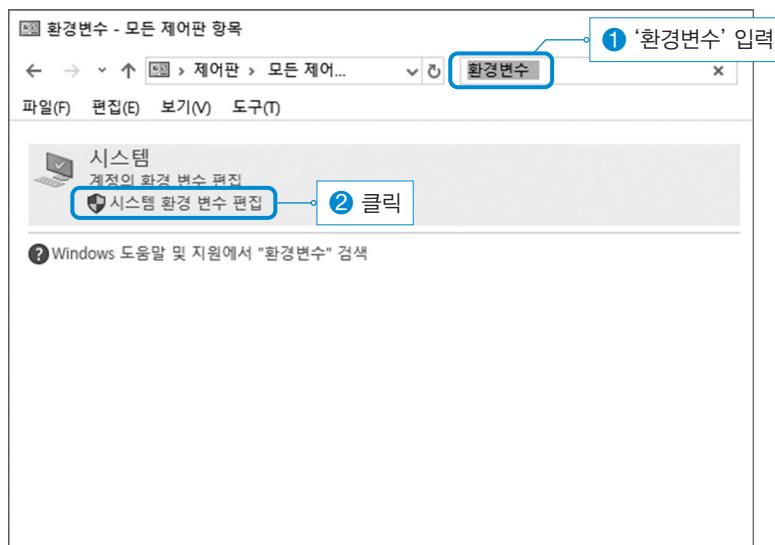
JDK의 설치만으로도 자바로 프로그램을 개발할 준비가 모두 끝났지만, 편의를 위해 JDK의 bin 폴더를 환경변수 path에 등록하는 것이 좋다. 이 폴더에는 자바로 프로그램을 개발하는데 필요한 실행파일들이 들어 있는데, 이 폴더를 path에 등록해 놓으면 실행파일을 실행할 때 일일이 경로를 입력하지 않아도 되어서 편리하다.

(참고) 환경 변수는 윈도우즈에서 사용하는 설정정보가 담겨있는 변수이다.

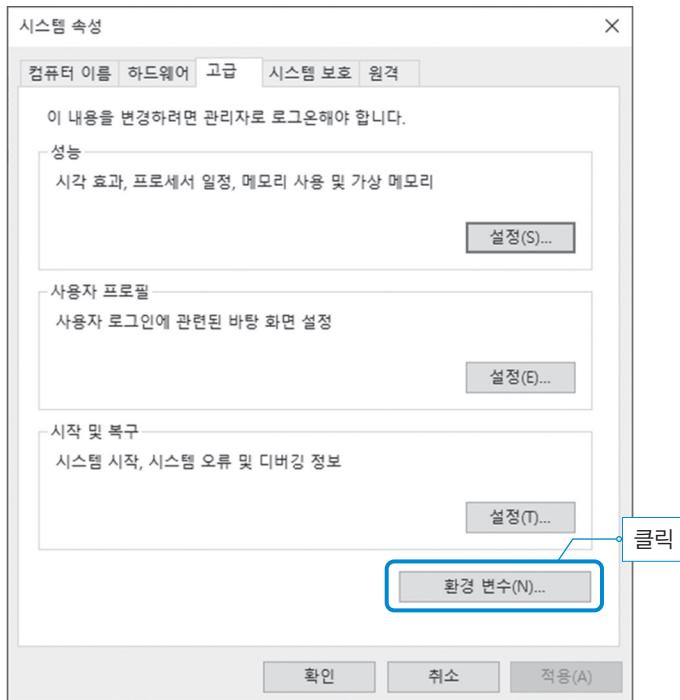
1 원도우키를 누르고 ‘제어판’이라고 입력 후, 엔터키를 누르면 아래의 오른쪽 화면이 나타난다.(원도우키는 키보드의 왼쪽 Alt키의 옆에 있습니다.)



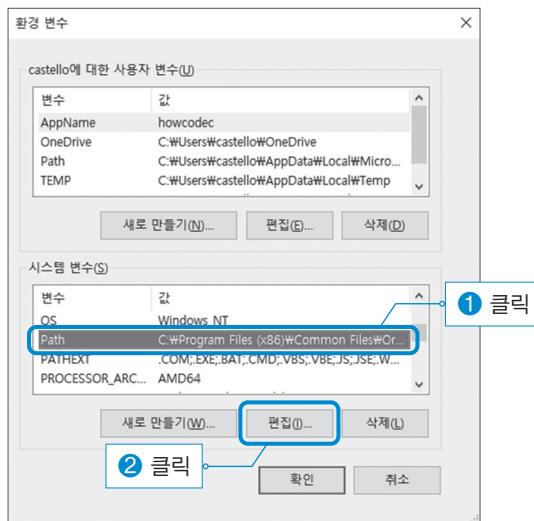
2 화면의 우측 상단에 ‘환경변수’라고 입력하면, 검색결과로 나오는 ‘시스템 환경 변수 편집’을 클릭.



③ 새로 열린 시스템 속성화면에서 '환경 변수(N)...'을 클릭.

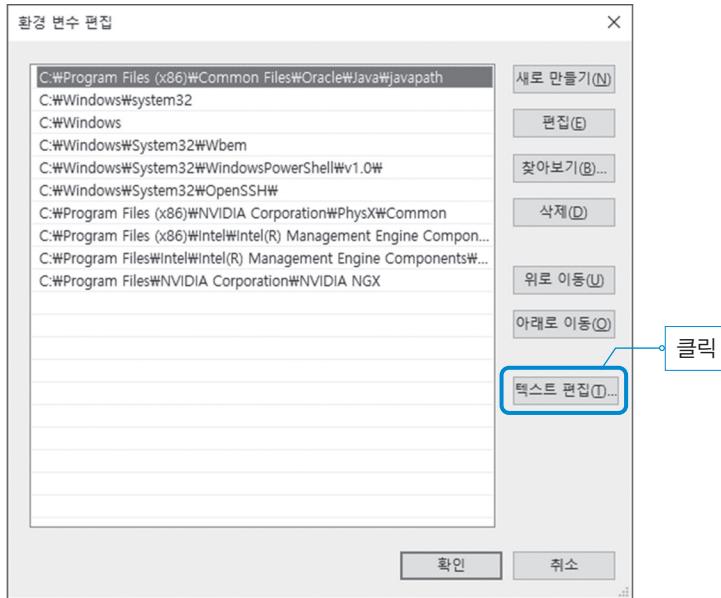


④ 시스템 변수 중에서 'Path'를 선택하고, '편집(I)...'을 클릭

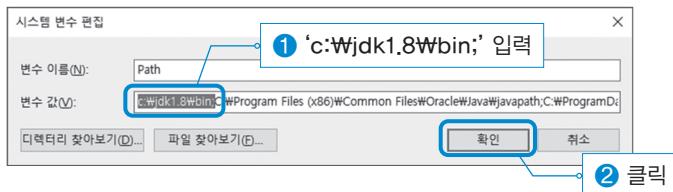


미리보기용 pdf입니다.

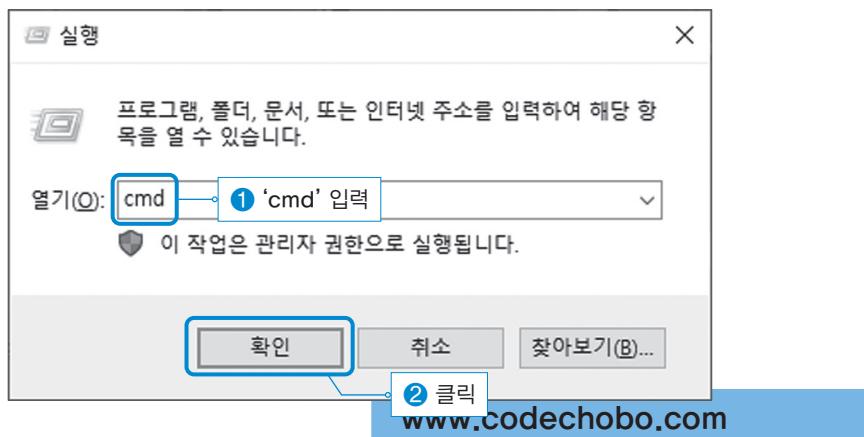
5 새로 열린 화면에서 '텍스트 편집(T)...'을 클릭.(윈도우즈 10이전 버전은 곧바로 다음 단계로)



6 변수 값의 맨 앞에 'c:\jdk1.8\bin;'을 추가하고, '확인'을 클릭한다. 그리고 끝에 ';'를 빼먹지 않도록 주의하자



7 '윈도우키+R'을 눌러서 나타난 실행창에 'cmd'를 입력하고, '확인'을 누른다.



8 먼저 'path'라고 입력하면, 환경변수 path의 값을 확인할 수 있다. 새로 추가한 'C:\jdk1.8\bin;'이 있는지 확인한다.

A screenshot of a Windows Command Prompt window titled '선택 C:\Windows\system32\cmd.exe'. The window shows the command 'path' being run. The output displays the PATH environment variable, which includes 'C:\Program Files (x86)\Intel\iCLS Client;C:\Program Files\Intel\iCLS Client;C:\Windows\system32;C:\Windows;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Program Files\Intel\Management Engine Components\DAL;C:\Program Files (x86)\Intel\Management Engine Components\DAL;C:\Program Files\Intel\Management Engine Components\IPT;C:\Program Files (x86)\Intel\Management Engine Components\IPT;C:\jdk1.8\bin;C:\Users\castello\AppData\Local\Microsoft\Windows Apps;'. A blue box highlights the line 'C:\jdk1.8\bin;', and a callout bubble labeled '① 'path'를 입력하고 Enter키 입력' points to it. Another callout bubble labeled '② 확인' points to the bottom right corner of the window.

9 그 다음 'javac -version'이라고 입력하고 'Enter'키를 눌러서 아래와 같은 화면이 나타나는지 확인한다.

A screenshot of a Windows Command Prompt window titled '관리자: C:\WINDOWS\system32\cmd.exe'. The command 'javac -version' is run, resulting in the output 'javac 1.8.0_221'. A blue box highlights this output, and a callout bubble labeled '① 'javac -version'를 입력하고 Enter키 입력' points to the command line. Another callout bubble labeled '② 확인' points to the bottom right corner of the window.

만일 'javac -version'이라고 입력했을 때, 아래와 같은 화면이 나오면 환경변수 'Path'의 설정이 잘못된 것이다. 환경변수 Path의 값을 다시 확인하여 올바르게 수정하고, 새로운 명령 프롬프트 창()을 열어 올바르게 설정했는데도 아래와 같은 결과가 나온다면, 윈도우즈를 다시 시작해보자.

A screenshot of a Windows Command Prompt window titled '관리자: C:\WINDOWS\system32\cmd.exe'. The command 'javac -version' is run, but the output is an error message: "'javac'은(는) 내부 또는 외부 명령, 실행할 수 있는 프로그램, 또는 배치 파일이 아닙니다.' (The term 'javac' is not recognized as an internal or external command, or as a program or batch file.). A blue box highlights this error message, and a callout bubble labeled '② 확인' points to the bottom right corner of the window.

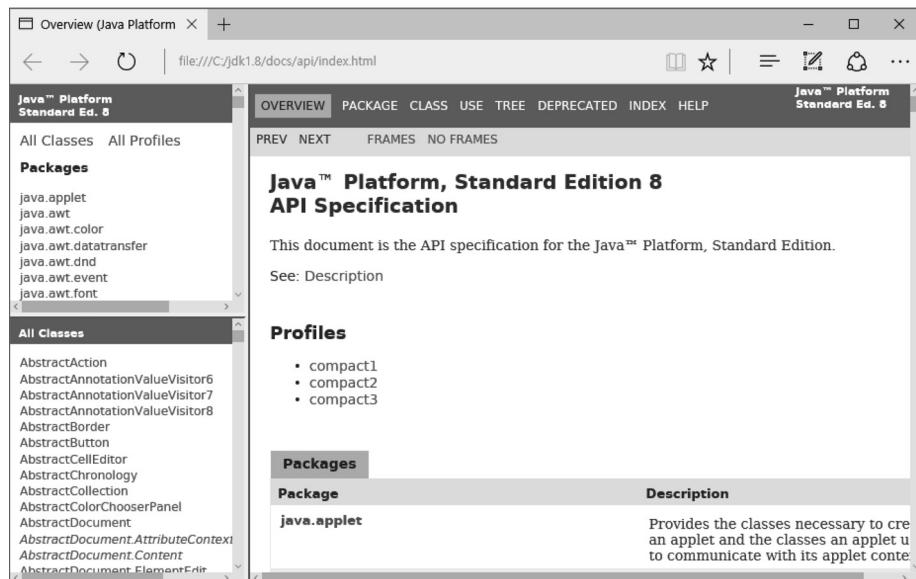
미리보기용 pdf입니다.

자바에서 제공하는 클래스 라이브러리(Java API)를 잘 사용하기 위해서는 Java API문서가 필수적이다. 이 문서에는 클래스 라이브러리의 모든 클래스에 대한 설명이 자세하게 나와 있다. 아마도 자바에서 제공하는 다양하고 방대한 양의 클래스 라이브러리에 감탄하게 될 것이다. 반면에 ‘이 많은 것을 다 공부해야 하나’라는 걱정도 들겠지만, 이 문서에 나오는 모든 클래스를 다 공부할 필요는 없고, 자주 사용되는 것만을 공부한 다음 나머지는 영어사전처럼 필요할 때 찾아서 사용하면 된다.

Java8의 API문서는 웹브라우저로 ‘<https://docs.oracle.com/javase/8/docs/api/>’를 방문하면 볼 수 있다. 그리고 ‘<https://docs.oracle.com/javase/8/docs/>’에는 자바와 관련된 여러 가지 유용한 내용이 수록되어 있으므로 참고하도록 하자.

참 고

Java API문서는 ‘<https://www.oracle.com/technetwork/java/javase/documentation/jdk8-downloads-2133158.html>’에서 다운로드 받을 수 있다.



자바로 프로그램을 개발하려면 JDK이외에 메모장(notepad.exe)이나 에딧플러스(editplus)와 같은 편집기가 필요하다. 요즘은 이클립스(eclipse)나 인텔리제이(IntelliJ)와 같은 뛰어난 기능의 전문 개발 도구들을 주로 사용하지만, 우선 메모장으로 간단한 자바 프로그램을 작성해 보자.

참고

editplus는 <http://www.editplus.com>에 가면 평가판을 무료로 제공한다.

예제
1-1

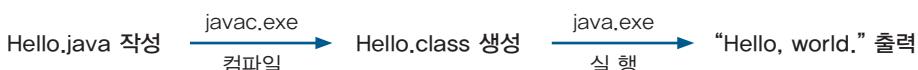
```
class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world."); // 화면에 글자를 출력한다.
    }
}
```

결과
Hello, world.

이 예제는 화면에 ‘Hello, world.’를 출력하는 아주 간단한 프로그램이다. 이 예제를 통해서 화면에 글자를 출력하려면 어떻게 해야 하는지 쉽게 알 수 있을 것이다.

예제1-1을 편집기나 editplus를 이용해서 작성한 다음 ‘Hello.java’로 저장하자. 이 때 클래스의 이름 ‘Hello’가 대소문자까지 정확히 같아야 한다.

이 예제를 실행하려면, 먼저 자바컴파일러(javac.exe)를 사용해서 소스파일(Hello.java)로부터 클래스파일(Hello.class)을 생성해야한다. 그 다음에 자바 인터프리터(java.exe)로 실행한다.



미리보기용 pdf입니다.

자바에서 모든 코드는 반드시 클래스 안에 존재해야 하며, 서로 관련된 코드들을 그룹으로 나누어 별도의 클래스를 구성하게 된다. 그리고 이 클래스들이 모여 하나의 Java 애플리케이션을 이룬다.

클래스를 작성하는 방법은 간단하다. 키워드 ‘class’ 다음에 클래스의 이름을 적고, 클래스의 시작과 끝을 의미하는 괄호{} 안에 원하는 코드를 넣으면 된다.

```
class 클래스이름 {  
    /*  
     * 주석을 제외한 모든 코드는 클래스의 블럭{} 내에 작성해야한다.  
    */  
}
```

참고

나중에 배우게 될 package문과 import문은 예외적으로 클래스의 밖에 작성한다.

아래 코드의 ‘public static void main(String[] args)’는 main메서드의 선언부인데, 프로그램을 실행할 때 ‘java.exe’에 의해 호출될 수 있도록 미리 약속된 부분이므로 항상 똑같이 적어주어야 한다.

참고

‘[]’은 배열을 의미하는 기호로 배열의 타입(type) 또는 배열의 이름 옆에 붙일 수 있다. ‘String[] args’는 String타입의 배열 args를 선언한 것이며, ‘String args[]’와 같이 쓸 수도 있다. 이 둘은 같은 의미이므로 차이가 없다. 자세한 내용은 ‘5장 배열’에서 배우게 될 것이다.

```
class 클래스이름 {  
    public static void main(String[] args) // main메서드의 선언부  
    {  
        // 실행될 문장들을 적는다.  
    }  
}
```

main메서드의 선언부 다음에 나오는 괄호{}는 메서드의 시작과 끝을 의미하며, 이 괄호 사이에 작업할 내용을 작성해 넣으면 된다. Java 애플리케이션은 main메서드의 호출로 시작해서 main메서드의 첫 문장부터 마지막 문장까지 수행을 마치면 종료된다.

모든 클래스가 main메서드를 가지고 있어야 하는 것은 아니지만, 하나의 Java 애플리케이션에는 main메서드를 포함한 클래스가 반드시 하나는 있어야 한다. main메서드는 Java애플리케이션의 시작점이므로 main메서드 없이는 Java 애플리케이션은 실행될 수 없기 때문이다. 작성된 Java애플리케이션을 실행할 때는 ‘java.exe’ 다음에 main메서드를 포함한 클래스의 이름을 적어줘야 한다.

www.codechobo.com

콘솔(명령 프롬프트, cmd.exe)에서 아래와 같이 Java 애플리케이션을 실행시켰을 때

```
c:\jdk1.8\work>java Hello  
                 ↑  
main(String[] args)
```

내부적인 진행순서는 다음과 같다.

1. 프로그램의 실행에 필요한 클래스(*.class파일)를 로드한다.
2. 클래스파일을 검사한다.(파일형식, 악성코드 체크)
3. 지정된 클래스(Hello)에서 main (String[] args)를 호출한다.

main메서드의 첫 줄부터 코드가 실행되기 시작하여 마지막 코드까지 모두 실행되면 프로그램이 종료되고, 프로그램에서 사용했던 자원들은 모두 반환된다.

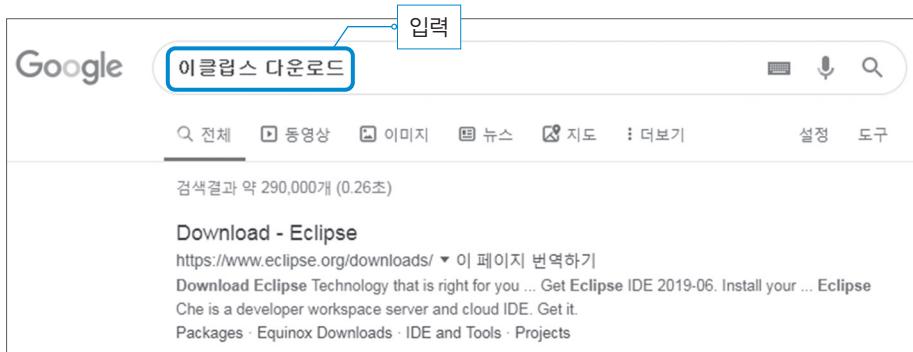
만일 지정된 클래스에 main메서드가 없다면 다음과 같은 에러 메시지가 나타날 것이다.

```
Exception in thread "main" java.lang.NoSuchMethodError: main
```

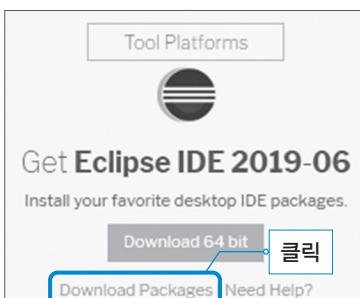
미리보기용 pdf입니다.

메모장과 같은 간단한 편집기로 개발할 수도 있지만, 아무래도 이클립스와 같은 고급 개발 도구가 여러모로 편리하다. 이클립스는 자바 프로그램을 편리하면서도 빠르게 개발할 수 있는 통합 개발 환경(IDE, Integrated Development Environment)을 제공한다. 게다가 무료이므로 자바를 배우는데는 최적의 개발 도구라 할 수 있다. 이제 이클립스를 다운 받아서 설치해 보자.

1 구글에서 '이클립스 다운로드'로 검색하면, 아래와 같은 검색결과가 나타난다. 링크를 클릭하자.



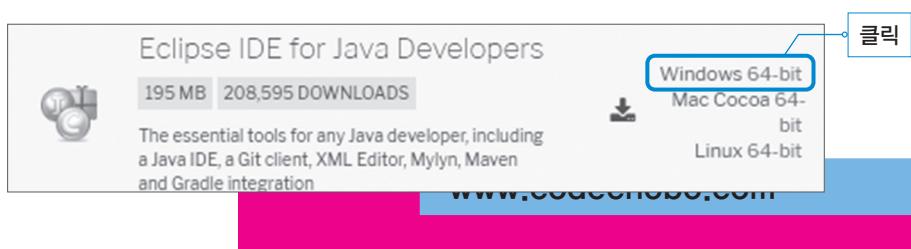
2 이동한 다운로드 페이지에서 'Download Packages' 링크를 클릭한다.



3 이클립스에도 여러가지 종류가 있는데, 아래의 것이 용량도 적고 가벼워서 자바를 공부하는데 적합하다. 윈도우즈의 경우, 'Windows 64-bit'를 클릭하자.

참고

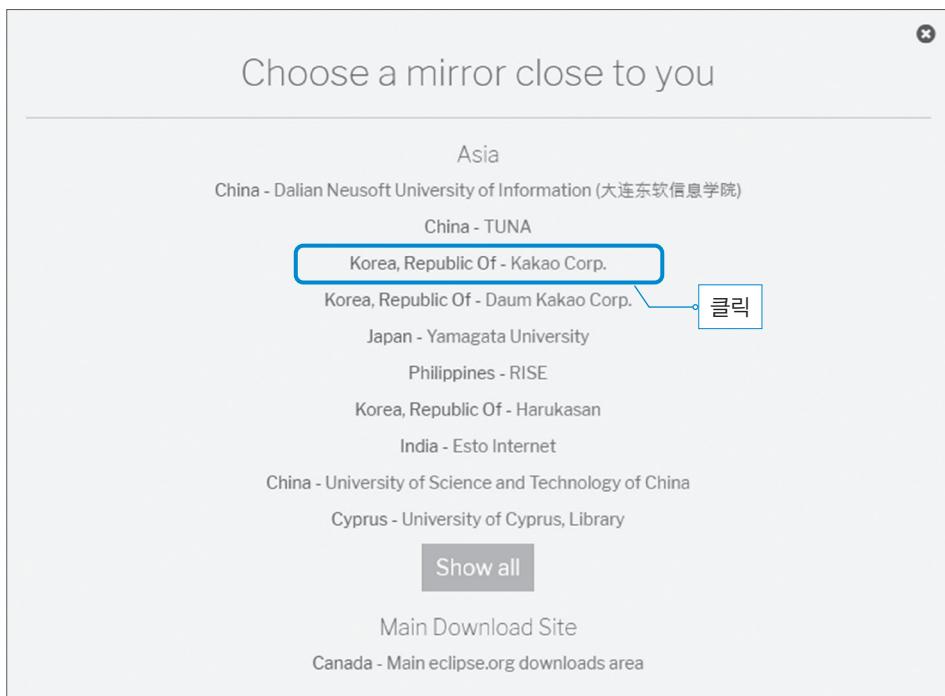
32 비트 윈도우즈의 경우, 구글에서 'eclipse 32 bit download'로 검색하면, 아래의 프로그램에 대한 링크를 찾을 수 있다.



4 아래의 화면에서 'Select Another Mirror'를 클릭한다.



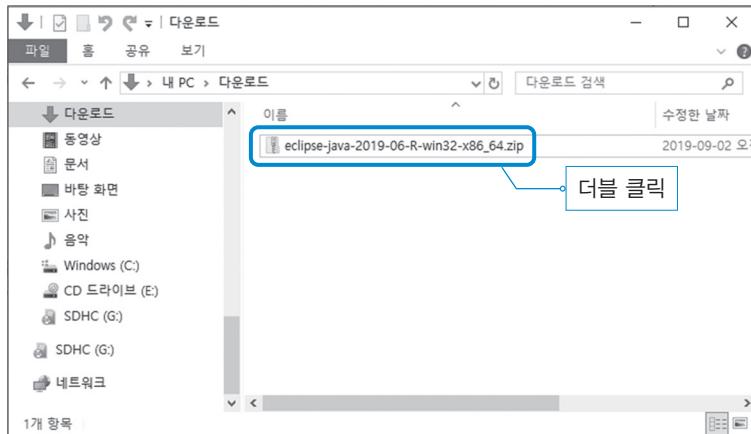
5 그러면 이클립스를 국내 사이트에서 빠르게 다운받을 수 있다. 'Korea Republic Of'로 시작하는 사이트 중에서 하나를 골라 클릭하면 다운로드가 시작된다.



미리보기용 pdf입니다.

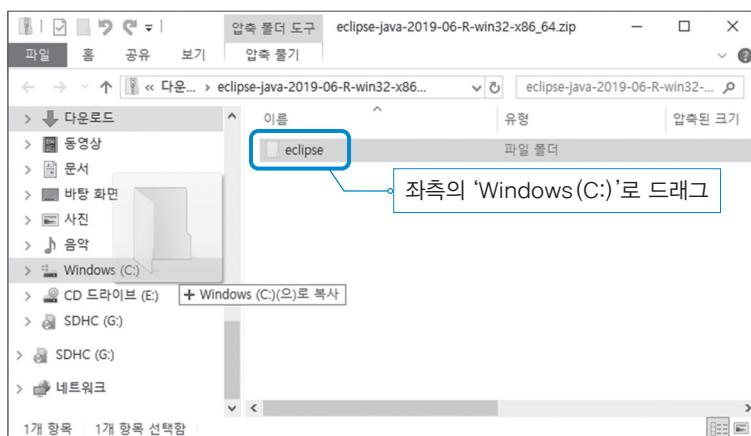
6 다운로드가 끝나고나면 '다운로드' 폴더에서 아래와 같은 파일을 찾을 수 있을 것이다. 이 파일을 더블 클릭한다.

참고 파일이름이 아래의 그림과 약간 다를 수도 있는데 상관없다.

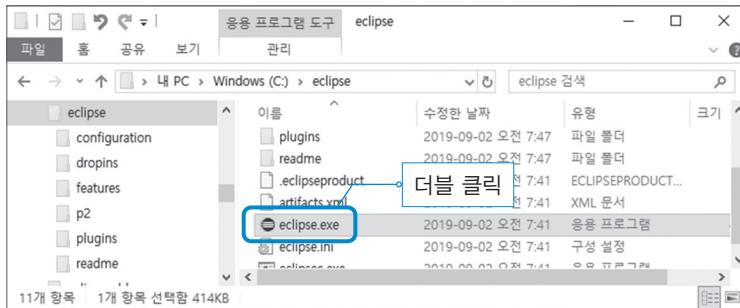


7 그러면 'eclipse'라는 폴더가 나오는데 이 폴더를 'C:\'로 드래그 한다. 이것으로 모든 설치가 끝났다.

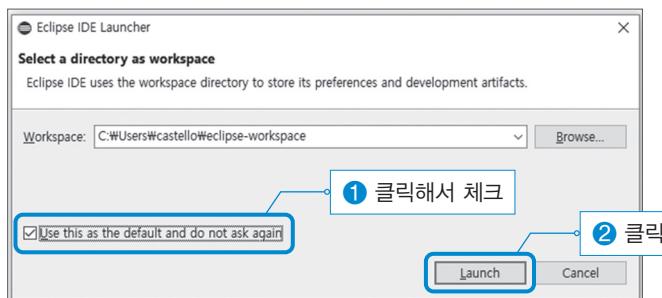
참고 이클립스를 삭제할 때는 eclipse 폴더만 삭제하면 된다.



8 'C:\eclipse' 폴더의 안으로 들어가면 'eclipse.exe' 또는 'eclipse'가 있는데, 이 파일을 더블 클릭하면 이클립스가 실행된다.



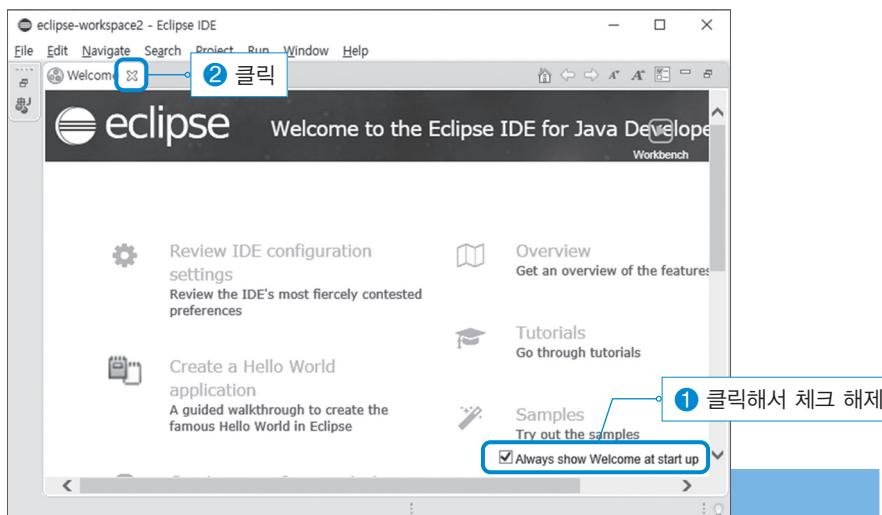
9 이클립스의 로고화면이 나타나고, 잠시 후에 아래와 같은 화면이 나타난다. 폴더의 경로를 확인한 다음, 체크박스를 체크하고 'Launch'버튼을 클릭하자.



10 아래와 같은 화면이 나타나면 이클립스가 잘 실행된 것이다. 우측 하단의 체크박스를 해제하고 'Welcome'창을 닫는다.

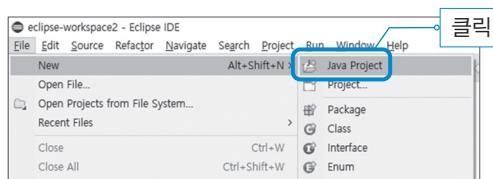
참고

이클립스의 좌측 상단에 현재 작업 중인 워크스페이스의 이름(eclipse-workspace)이 나타난다.

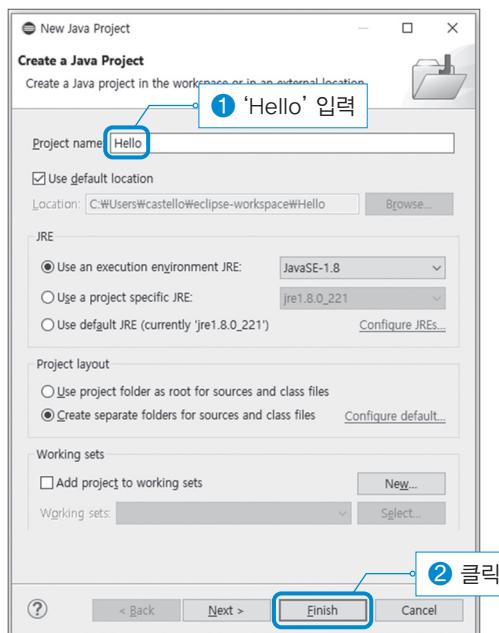


이클립스를 설치했으니 이제 이클립스를 이용해서 자바 애플리케이션을 개발하는 방법에 대해서 배워볼 차례이다. 앞서 작성했던 'Hello, world'를 화면에 출력하는 간단한 자바 애플리케이션을 이클립스로 개발해 보자.

- 1 새로운 프로젝트를 생성하기 위해 메뉴에서 File > New > Java Project를 클릭한다.

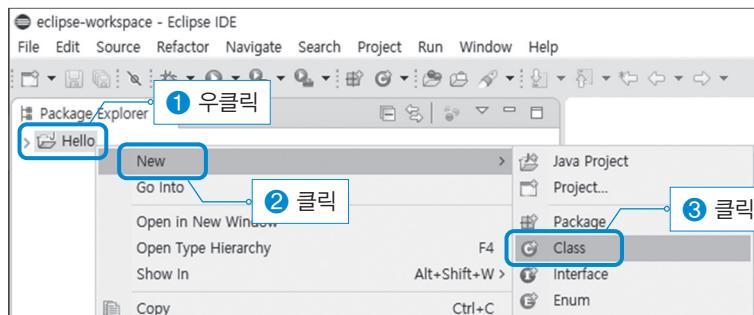


- 2 Project name으로 'Hello'를 입력하고 'Finish'버튼을 누른다.



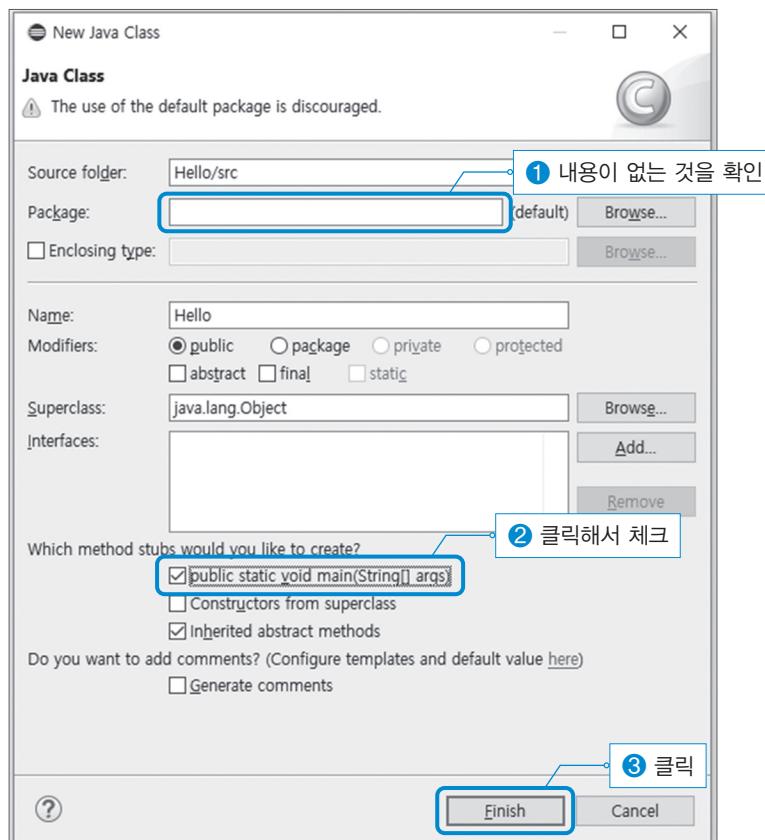
3 패키지 익스플로러(Package Explorer) 아래에 Hello 프로젝트가 생성된 것을 확인하자. 이제 Hello클래스를 새로 추가할 차례이다.

Hello프로젝트 위에서 우클릭하여, 메뉴 New 아래의 Class를 클릭한다.



4 클래스의 이름으로 Hello로 입력하고, 아래의 체크박스를 체크한 후에 Finish버튼을 클릭한다.

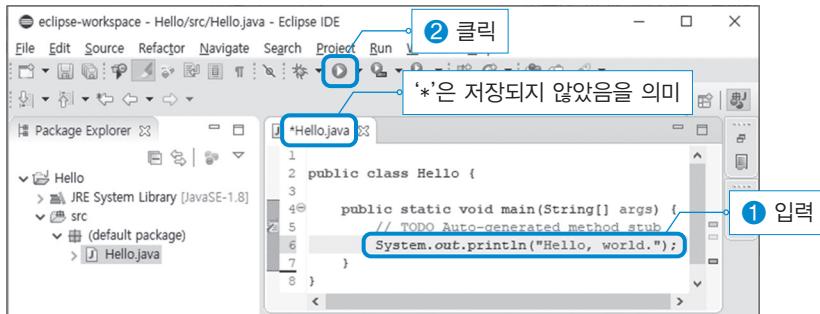
참고 Package란에 자동으로 입력된 내용이 있으면 삭제한다.



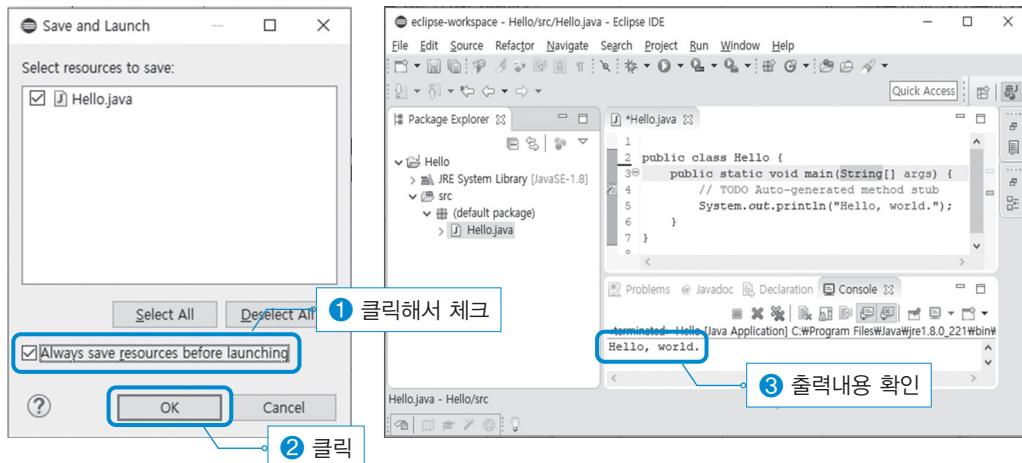
미리보기용 pdf입니다.

5 아래의 화면을 보면, Hello프로젝트에 Hello.java라는 파일이 생성되었고, 우측에는 이 파일의 내용이 자동으로 작성되어 나타난다. 화면과 같이 'System.out.println("Hello, world.");'를 중간에 한 줄 추가하자. 그 다음엔 실행버튼을 누른다.

참고 탭의 파일 이름 왼쪽에 붙은 '*'은 이 파일이 변경된 후에 아직 저장되지 않았다는 것을 의미한다. 저장(ctrl+s) 하면, '*'이 사라진다.



6 이클립스의 로고화면이 나타나고, 잠시 후에 아래와 같은 화면이 나타난다. 체크박스를 체크하고 'OK'버튼을 클릭하자.



이클립스의 화면은 여러 개의 작은 창들로 이루어져 있는데, 이 하나의 창을 뷰(view)라 부르고 이 뷰들로 구성된 화면 전체를 퍼스펙티브(perspective)라고 한다.

이클립스는 약 60여 개의 뷰를 제공하는데, 이 중에서 작업에 필요한 몇 개의 뷰만 선택해서 퍼스펙티브를 구성하는 것이 일반적이다.

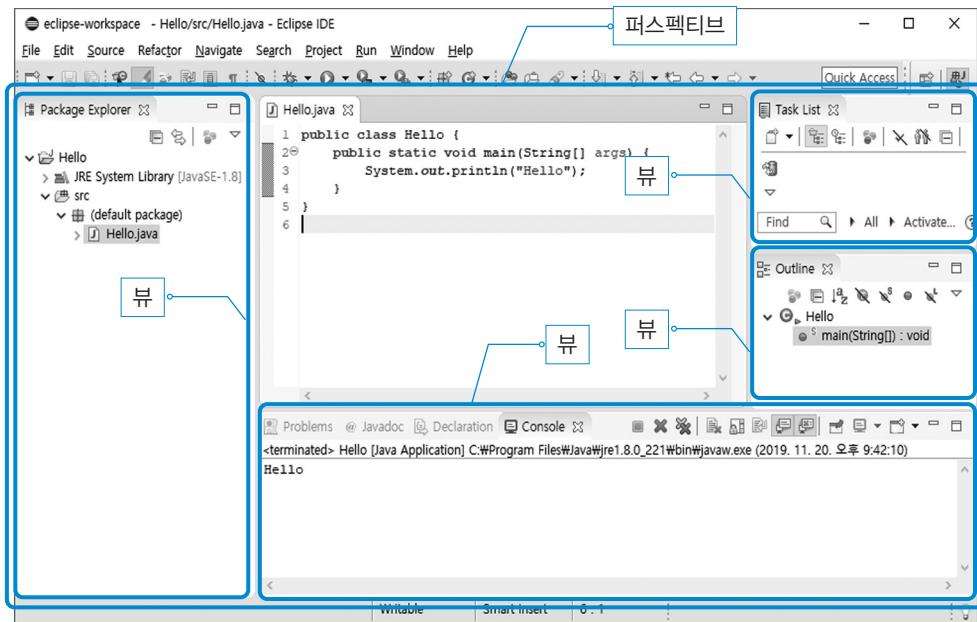
보통 수행할 작업에 따라 사용할 뷰가 달라지기 때문에, 작업이 바뀔 때마다 뷰의 종류와 크기 위치 등을 바꾸는 것은 꽤나 번거로운 일이다.

그래서 현재 퍼스펙티브를 저장했다가 필요할 때 다시 불러서 사용할 수 있는 기능이 제공된다.

참고 퍼스펙티브를 저장할 때는 ‘메뉴 Window > Perspective > Save Perspective As...’, 저장된 퍼스펙티브를 불러올 때는 ‘메뉴 Window > Perspective > Open Perspective’를 클릭하면 된다.

처음에 이클립스를 실행하면, 기본적으로 제공되는 퍼스펙티브 중의 하나인 ‘자바 퍼스펙티브(Java Perspective)’로 화면이 보여진다. 프로그램을 작성하다 실수로 뷰를 닫았는데 뷰의 이름이 기억나지 않는다면, 퍼스펙티브를 원래대로 되돌리던가 ‘자바 퍼스펙티브’를 다시 열면 된다.

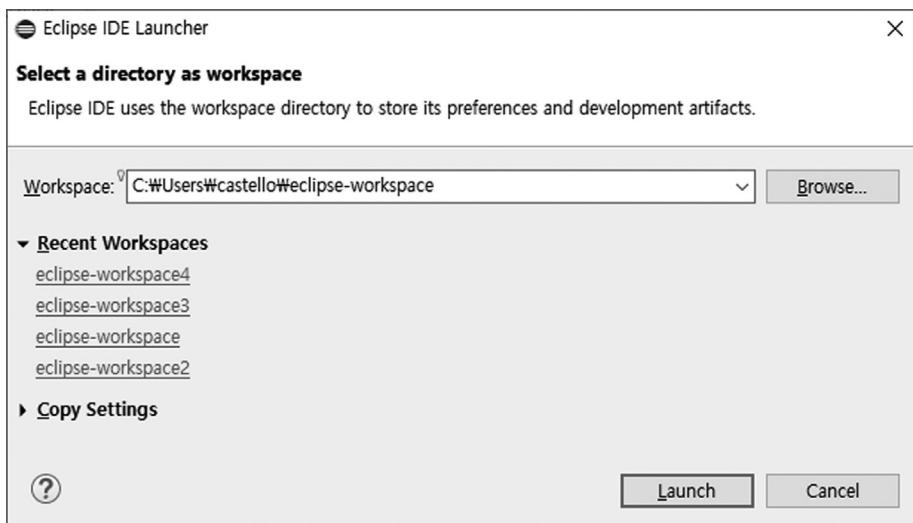
참고 퍼스펙티브를 원래대로 되돌릴 때는 ‘메뉴 Window > Perspective > Reset Perspective...’를 클릭하면 된다.



미리보기용 pdf입니다.

워크스페이스

이클립스에서 작성한 파일이 저장되는 공간을 워크스페이스(workspace)라고 하며, 이클립스를 처음 실행할 때 워크스페이스로 사용할 폴더를 지정하는 화면이 나타난다.



이클립스를 사용하다 보면 가끔 이유없이 오작동하는 경우가 있는데, 이럴 때는 이클립스를 삭제하고 다시 설치하면 해결되곤 한다. 이클립스는 별도의 설치없이 복사만 하면 바로 실행되는 프로그램이라 이클립스를 삭제할 때는 단순히 해당 폴더만 삭제해서 휴지통에 넣으면 된다. 만일 워크스페이스를 이클립스가 설치된 폴더 아래로 지정하면 이클립스를 삭제할 때 워크스페이스까지 같이 삭제하게 된다. 이런 실수를 방지하기 위해서 기본적으로 'C:\Users\사용자아이디\eclipse-workspace'가 워크스페이스의 경로로 제안되는 것이다.

원한다면, 워크스페이스를 다른 곳으로 지정할 수도 있고 새로운 워크스페이스를 추가로 만들 수도 있다. 프로젝트의 수가 너무 많아지거나 성격이 다른 프로젝트를 따로 저장하고자 할 때 새로운 워크스페이스를 만들어서 분리해두면 편리하다.

참고 새로운 워크스페이스를 만들려면, 메뉴 'File > Switch Workspace > Other...'에서 새로운 경로를 지정해준다음, Launch버튼을 누르면 된다.

우리가 프로그램을 작성할 때 이클립스와 같은 개발도구를 사용하는 이유는 편리함과 높은 생산성을 제공하기 때문이다. 이클립스는 프로그램을 더 빠르고 편리하게 개발할 수 있도록 대부분의 기능에 단축키를 제공하며, 본인의 취향에 맞게 다른 단축키를 사용하도록 변경하는 것도 가능하다.

무수히 많은 단축키 중에서도 가장 많이 사용되는 것들을 골라서 아래의 표에 나열하였다. 처음부터 단축키를 모두 외울 필요는 없고, 마우스를 주로 사용하다가 자주 사용하는 기능부터 단축키를 하나둘씩 익혀나가도록 하자.

명령	단축키	명령	단축키
단축키 목록 보기	ctrl + shift + L	단어 완성	단어 일부 입력후, alt + /
저장	ctrl + S	자동 수정(Quick fix)	ctrl + 1
실행	ctrl + F11	같은 단어 표시(형광펜)	alt + shift + O
전체 선택	ctrl + A	행으로 이동	ctrl + L
한 줄 삭제	ctrl + D	최근 수정지점으로 이동	ctrl + Q
다음 단어 삭제	ctrl + delete	소스 탭 간 이동	ctrl + pgup, pgdn
이전 단어 삭제	ctrl + backspace	소스 탭 목록 보기	ctrl + shift + E
단어간 커서 이동	ctrl + ⇌, ⇐	현재 소스 탭 닫기	ctrl + F4
찾기 / 바꾸기	ctrl + F	리소스(파일) 찾기	ctrl + shift + R
검색	ctrl + H	편집 이력 이동	alt + ⇌, ⇐
주석 / 해제	ctrl + /	편집창 폰트 크기	ctrl + +, -
범위 주석 / 해제	ctrl + shift + /, ⌘	속성 보기	alt + Enter
멀티 컬럼 편집	ctrl + A, shift + ↑, ↓	선언 보기	F3
행 이동(여러 행 가능)	alt + ↑, ↓	상속 계층도 보기	클래스 이름 클릭, F4
행 복사(여러 행 가능)	alt + ctrl + shift + ↑, ↓	상속 계층도 보기	ctrl + T
자동 들여쓰기	ctrl + i	경로 보기	alt + shift + B
자동 형식 맞추기	ctrl + shift + F	import문 자동 추가	ctrl + shift + O
자동 완성	ctrl + space	멤버 목록 보기	ctrl + O

참고

멀티 컬럼 편집은 여러 행을 동시에 수정하는 기능. ctrl+A를 누른 후, shift키를 누른 채로 화살표키(↑)로 편집 할 영역을 선택한다.

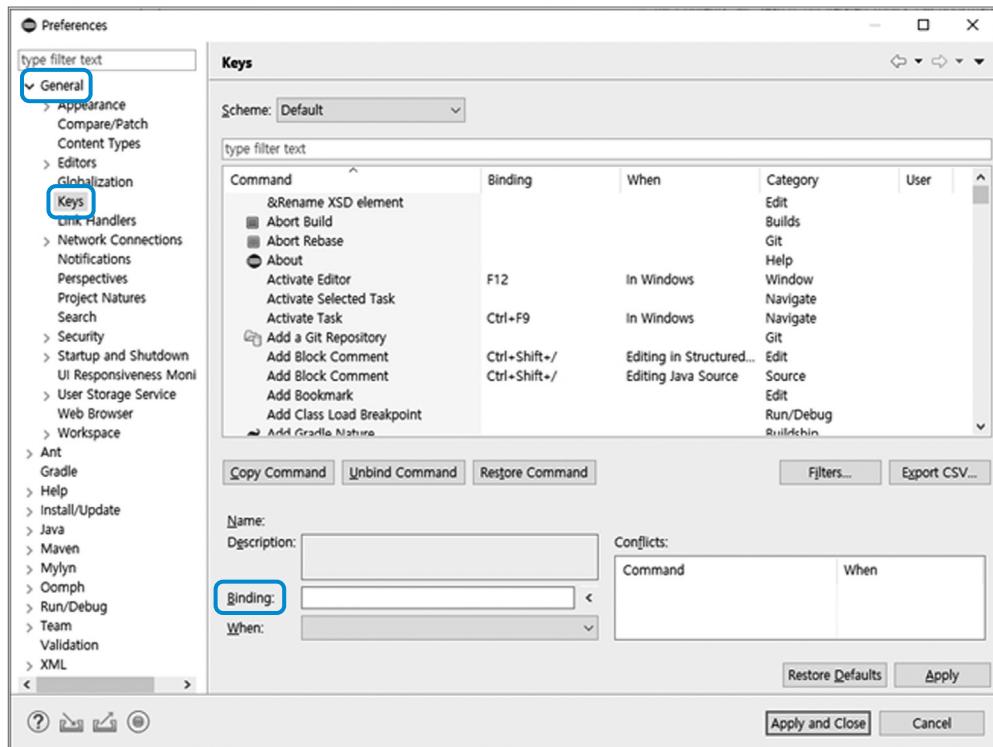
참고

행 복사(copy lines) 단축키는 매우 유용한데, 원도우즈의 단축키와 충돌나면 동작하지 않는다. p.29의 단축 키 설정을 참고하여 충돌나지 않게 변경하자.

미리보기용 pdf입니다.

단축키의 설정 및 변경

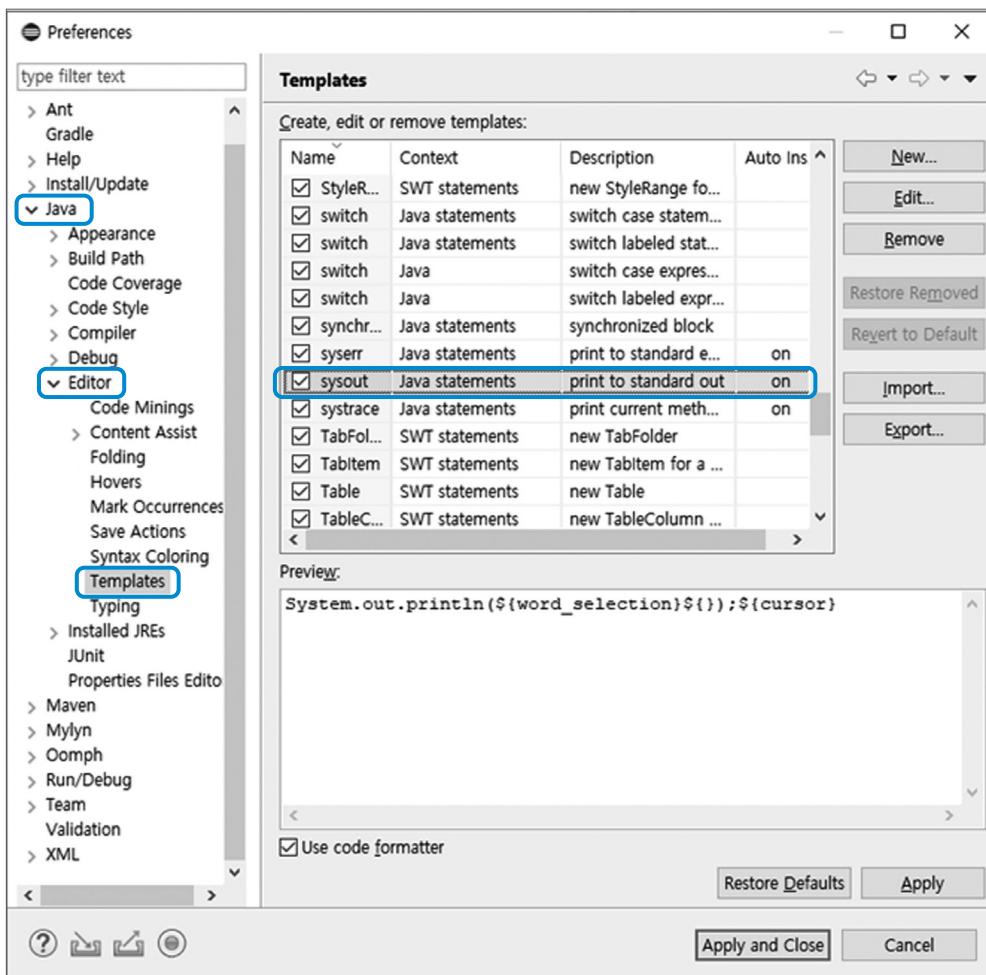
메뉴 Windows > Preferences에서 왼쪽 목록에서 General > Keys를 클릭하면 아래와 같은 화면이 나온다. 예를 들어 행을 복사하는 ‘Copy Lines’의 단축키를 변경하려면, ‘type filter text’라고 적힌 입력란에 ‘copy’라고 입력한 다음에 ‘Copy Lines’를 찾아서 클릭한다. 화면 아래쪽의 ‘Binding’을 클릭하고 원하는 단축키를 누르면 된다.



이클립스의 기능 중에 반드시 알아야 하는 기능이 있다면 바로 ‘자동 완성 기능(Content Assist)’이다. 이 기능은 특정 단어나 문자를 입력한 후에 자동완성 기능 단축키 ‘Ctrl+space’를 누르면, 코드가 자동으로 완성되는 편리한 기능이다. 예를 들어 에디터 창에서 ‘s’를 입력한 다음에 ‘ctrl+space’를 누르면, 이름이 ‘s’로 시작되는 것들의 목록이 나타나고 이 중에서 하나를 선택하면 코드를 쉽게 작성할 수 있다.

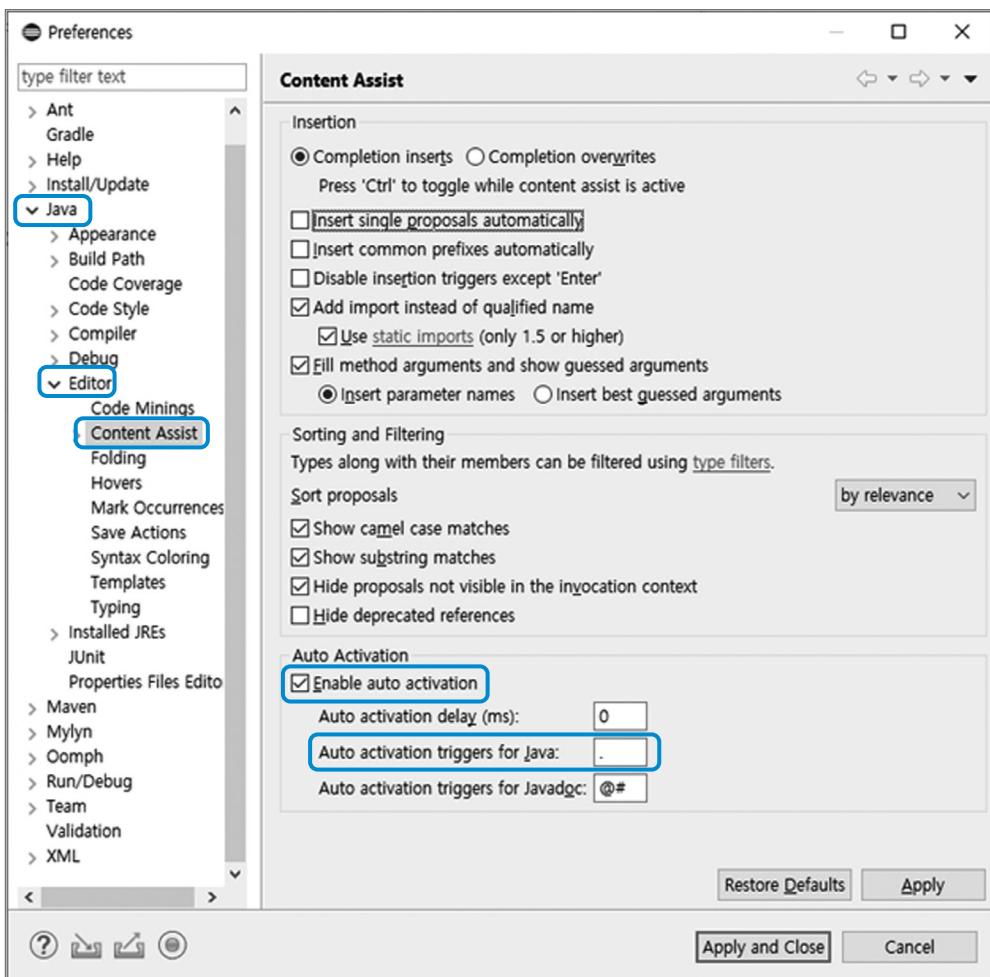
또한 특정 단어를 입력하고 자동완성 단축키(ctrl+space)를 누르면, 지정된 형식으로 자동 완성되게 할 수도 있다. 이 단어를 ‘템플릿(Template)’이라고 하며, ‘sysout’은 템플릿으로 등록되어 있기 때문에 에디터에서 ‘sysout’이라고 입력하고 ‘ctrl+space’를 누르면, ‘System.out.println();’이 자동으로 입력된다.

템플릿의 목록은 메뉴 Window의 아래 Preference의 ‘Java>Editor>Templates’에서 볼 수 있으며, 추가, 삭제 또는 변경이 가능하다.



미리보기용 pdf입니다.

만일 자동 완성(Content Assist) 기능이 동작하지 않는다면, 단축키 설정 화면(p.29)에서 Content Assist의 Binding이 어떤 키조합으로 되어 있는지 확인하자. 그래도 안되면, 아래의 화면(Window 메뉴 아래의 Preference에서 Java>Content Assist)에서 ‘Enable auto activation’이 체크되어 있는지 확인하자.



위 화면 하단의 ‘Auto activation triggers for java’에 ‘.’이 입력되어 있는데, 에디터에서 여기에 입력한 문자를 입력하면, 자동 완성 단축키를 누르지 않아도 저절로 자동 완성 목록이 화면에 나타난다. ‘.’대신에 ‘.abcdefghijklmnopqrstuvwxyz’를 넣어두면, 자동 완성 단축키 (ctrl+space)를 누르지 않아도 키를 누를 때마다 자동 완성 목록이 화면에 나타나므로 타자가 느린 사람도 편리하게 코드를 작성할 수 있다.

작성하는 프로그램의 크기가 커질수록 프로그램을 이해하고 변경하는 일이 점점 어려워진다. 심지어는 자신이 작성한 프로그램도 ‘내가 왜 이렇게 작성했지?’라는 의문이 들기도 하는데, 남이 작성한 코드를 이해한다는 것은 정말 쉬운 일이 아니다.

이러한 어려움을 덜기 위해 사용하는 것이 바로 주석이다. 주석을 이용해서 프로그램 코드에 대한 설명을 적절히 덧붙여 놓으면 프로그램을 이해하는 데 많은 도움이 된다.

그 외에도 주석은 프로그램의 작성자, 작성일시, 버전과 그에 따른 변경이력 등의 정보를 제공할 목적으로 사용된다.

주석을 작성하는 방법은 다음과 같이 두 가지 방법이 있다. ‘/*’와 ‘*/’ 사이에 주석을 넣는 방법과 앞에 ‘//’를 붙이는 방법이 있다.

범위 주석 ‘/*’와 ‘*/’ 사이의 내용은 주석으로 간주된다.

한 줄 주석 ‘//’부터 라인 끝까지의 내용은 주석으로 간주된다.

참고 이 외에도 Java API문서와 같은 형식의 문서를 자동으로 만들 수 있는 주석(** ~ **/)이 있지만 많이 사용되지 않는 않으므로 자세한 설명은 생략하겠다. 이 주석은 javadoc.exe에 의해서 html문서로 자동 변환되며, 보다 자세한 내용은 인터넷에서 ‘javadoc’으로 검색하면 찾을 수 있다.

다음은 주석의 몇 가지 사용 예인데 흰색바탕으로 처리된 부분이 주석이다.

```
/*
Date : 2016. 1. 3
Source : Hello.java
Author : 남궁성
Email : castello@naver.com
*/

class Hello
{
    public static void main(String[] args) /* 프로그램의 시작 */
    {
        System.out.println("Hello, world."); // Hello, world를 출력
    }
}
```

위의 코드는 예제1-1에 주석을 넣은 것인데, 컴파일하는 주석을 무시하고 건너뛰기 때문에 위의 코드를 컴파일한 결과와 예제1-1을 컴파일한 결과는 정확히 일치한다. 따라서 주석이 많다고 해서 프로그램의 성능이 떨어지는 일은 없으니 안심하고 주석을 적극적으로 활용하기 바란다.

미리보기용 pdf입니다.

한 가지 주의해야할 점은 문자열을 의미하는 큰따옴표("") 안에 주석이 있을 때는 주석이 아닌 문자열로 인식된다는 것이다. Hello.java를 아래와 같이 변경하여 실행해보면, 주석의 내용도 같이 출력되는 것을 확인할 수 있을 것이다.

```
class Hello
{
    public static void main(String[] args)
    {
        System.out.println("Hello, /* 이것은 주석 아님 */ world.");
        System.out.println("Hello, world. // 이것도 주석 아님");
    }
}
```

자바로 프로그래밍을 배워나가면서 많은 수의 크고 작은 에러들을 접하게 될 것이다. 대부분의 에러는 작은 실수에서 비롯된 것들이며, 곧 익숙해져서 쉽게 대응할 수 있게 되지만 처음 배울 때는 작은 실수 하나 때문에 많은 시간을 허비하곤 한다.

그래서 자주 발생하는 기본적인 에러와 해결방법을 간단히 정리하였다. 에러가 발생하였을 때 참고하고, 그 외의 에러는 에러메시지의 일부를 인터넷에서 검색해서 찾아보면 해결책을 얻는데 도움이 될 것이다.

1. cannot find symbol 또는 cannot resolve symbol

지정된 변수나 메서드를 찾을 수 없다는 뜻으로 선언되지 않은 변수나 메서드를 사용하거나, 변수 또는 메서드의 이름을 잘못 사용한 경우에 발생한다. 자바에서는 대소문자 구분을 하기 때문에 철자 뿐 만아니라 대소문자의 일치여부도 꼼꼼하게 확인해야한다.

2. ';' expected

세미콜론 ';'이 필요한 곳에 없다는 뜻이다. 자바의 모든 문장의 끝에는 ';'을 붙여주어야 하는데 가끔 이를 잊고 실수하기 쉽다.

3. Exception in thread "main" java.lang.NoSuchMethodError: main

'main메서드를 찾을 수 없다.'는 뜻인데 실제로 클래스 내에 main메서드가 존재하지 않거나 메서드의 선언부 'public static void main(String[] args)'에 오타가 존재하는 경우에 발생한다.

이 에러의 해결방법은 main메서드가 클래스에 정의되어 있는지 확인하고, 정의되어 있다면 main메서드의 선언부에 오타가 없는지 확인한다. 자바는 대소문자를 구별하므로 대소문자의 일치여부까지 정확히 확인해야한다.

참고 args는 매개변수의 이름이므로 args 대신 argv나 arg와 같이 다른 이름을 사용할 수 있다.

4. Exception in thread "main" java.lang.NoClassDefFoundError: Hello

'Hello라는 클래스를 찾을 수 없다.'는 뜻이다. 클래스 'Hello'의 철자, 특히 대소문자를 확인해보고 이상이 없으면 클래스파일 (*.class)이 생성되었는지 확인한다.

예를 들어 'Hello.java'가 정상적으로 컴파일 되었다면 클래스파일 'Hello.class'가 있어야한다. 클래스파일이 존재하는데도 동일한 메시지가 반복해서 나타난다면 클래스패스(classpath)의 설정이 바르게 되었는지 다시 확인해보자.

미리보기용 pdf입니다.

5. illegal start of expression

직역하면 문장(또는 수식, expression)의 앞부분이 문법에 맞지 않는다는 의미인데, 간단히 말해서 문장에 문법적 오류가 있다는 뜻이다. 괄호‘(’나 ‘{’를 열고서 닫지 않거나, 수식이나 if문, for문 등에 문법적 오류가 있을 때 또는 public이나 static과 같은 키워드를 잘못 사용한 경우에도 발생한다. 에러가 발생한 곳이 문법적으로 옳은지 확인하라.

6. class, interface, or enum expected

이 메시지의 의미는 ‘키워드 class나 interface 또는 enum이 없다.’이지만, 보통 괄호‘(’ 또는 ‘}’의 개수가 일치하지 않는 경우에 발생한다. 열린괄호‘(’와 닫힌괄호‘}’의 개수가 같은지 확인하자.

마지막으로 한 가지 더 얘기하고 싶은 것은 에러가 발생했을 때, 어떻게 해결할 것인가에 대한 방법이다. 아주 간단하고 당연한 내용이라서 다소 실망스럽게 느껴질지도 모르지만, 막상 실제 에러가 발생했을 때 아래의 순서대로 처리해보면 도움이 될 것이다.

1. 에러 메시지를 잘 읽고 해당 부분의 코드를 살펴본다.
이상이 없으면 해당 코드의 주위(윗줄과 아래 줄)도 함께 살펴본다.
2. 그래도 이상이 없으면 에러 메시지는 잊어버리고 기본적인 부분을 재확인한다.
대부분의 에러는 사소한 것인 경우가 많다.
3. 의심이 가는 부분을 주석처리하거나 따로 떼어내서 테스트 한다.

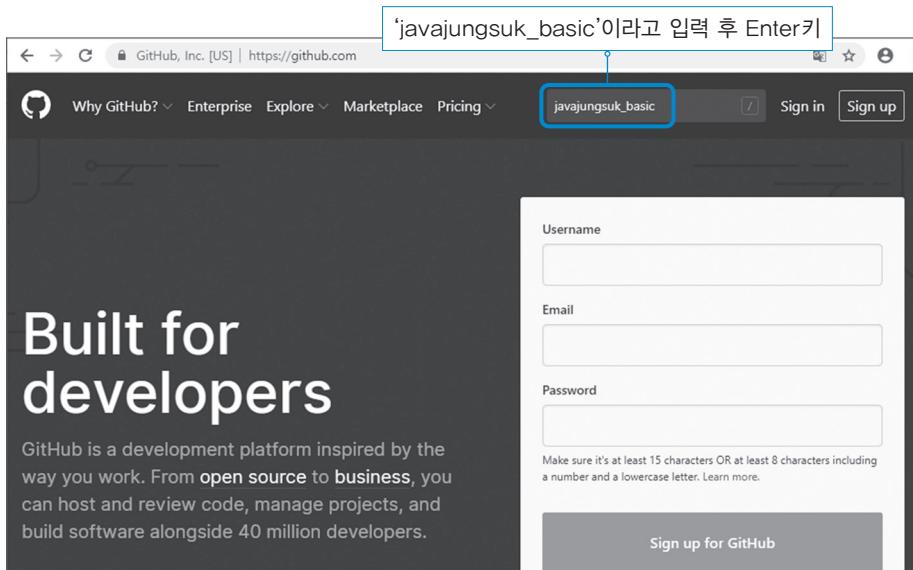
에러 메시지가 실제 에러와는 관계없는 내용일 때도 있지만, 대부분의 경우 에러 메시지만 잘 이해해도 문제가 해결되는 경우가 많으므로 에러 해결을 위해서 제일 먼저 해야 할 일은 에러 메시지를 잘 읽는 것임을 명심하자.

이 책에 소개하는 모든 예제와 학습 자료는 깃헙(github.com)에 올려놓았으며, 깃헙에서 이 자료들을 다운로드받는 방법에 대해서 단계별로 자세히 설명하겠다.

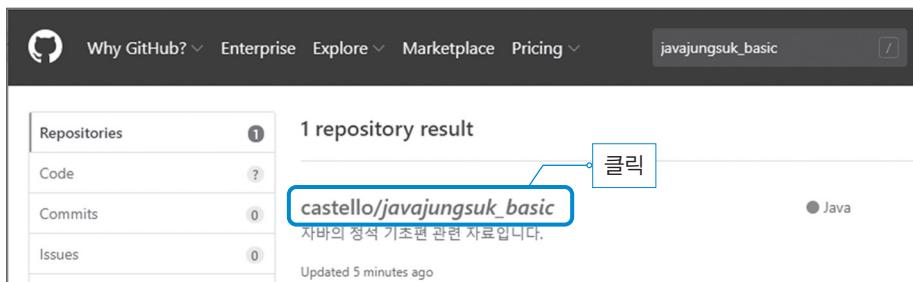
참고

동영상강좌는 유튜브(youtube.com)에서 '자바의정석기초'라고 검색하면 찾을 수 있다.

- 깃헙(github.com)을 방문해서, 사이트 상단의 검색창에 'javajungsuk_basic'이라고 입력해서 검색한다.

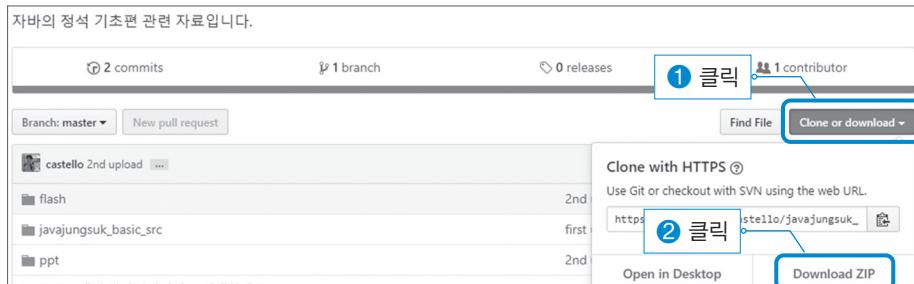


- 아래와 같은 검색결과가 나오면, 'castello/javajungsuk_basic'를 클릭한다.

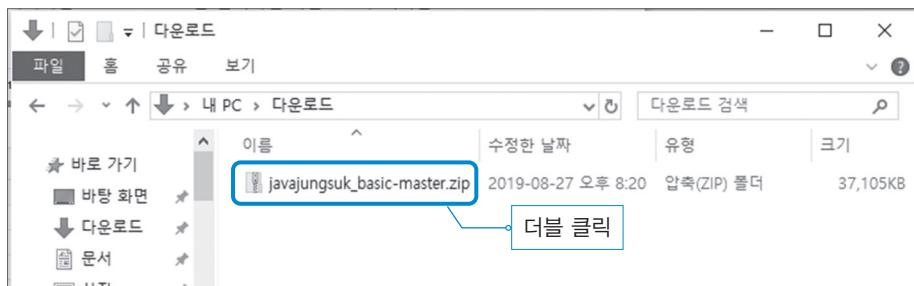


미리보기용 pdf입니다.

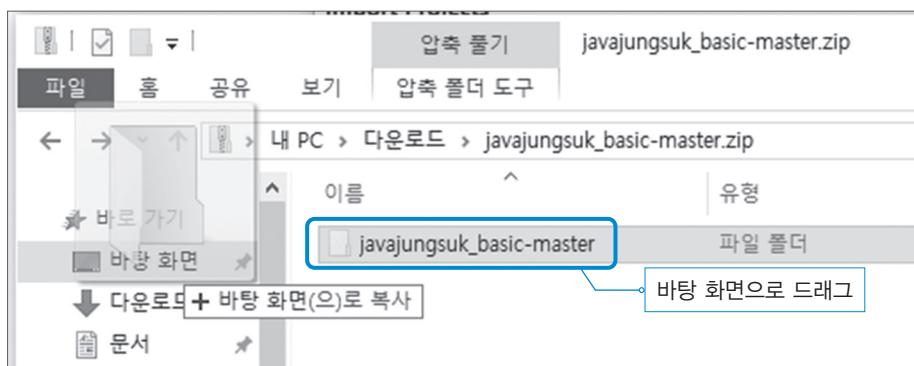
- ③ 화면 우측 중간의 'Clone or download'버튼을 누르면, 'Download ZIP'버튼이 나타난다. 이 버튼을 클릭하면 다운로드가 시작된다.



- ④ 다운로드가 완료되면, 다운로드 폴더에 'javajungsuk_basic-master.zip'파일을 찾을 수 있다. 이 파일을 더블 클릭한다.



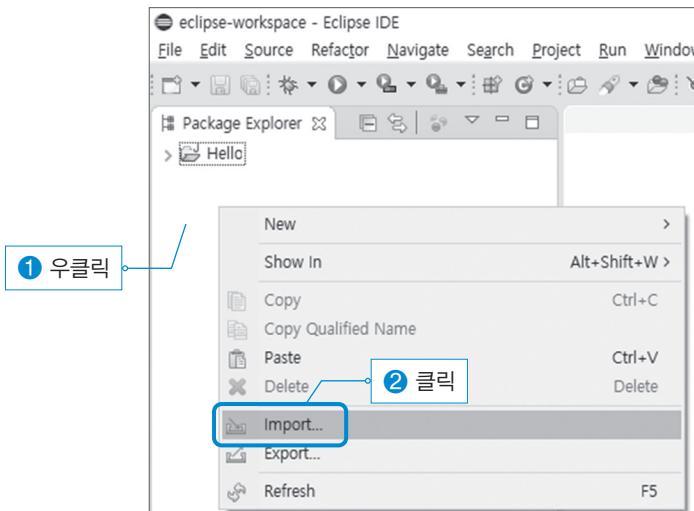
그러면 아래와 같이 'javajungsuk_basic-master'라는 폴더가 나오는데 이 폴더를 바탕화면으로 드래그 한다.



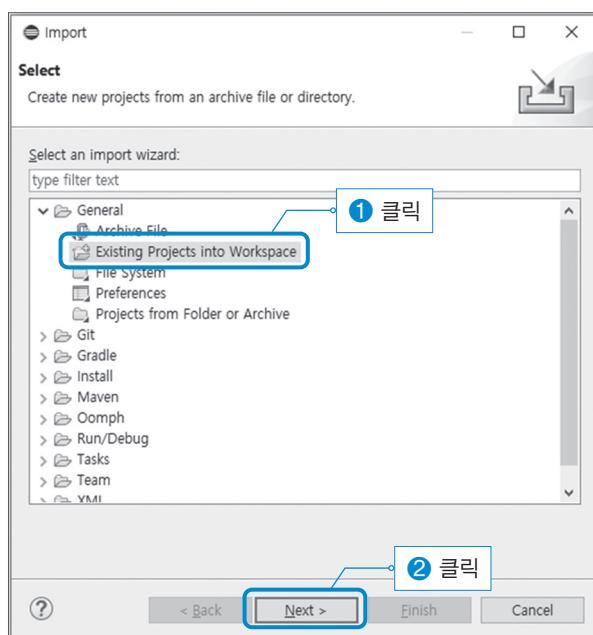
18 이클립스로 소스파일 가져오기

이제 앞서 깃헙에서 다운로드 받은 파일을 이클립스로 가져오는 방법에 대해서 알아볼 것이다. 일단 이클립스를 실행하자.

- 1 패키지 익스플로러(Package Explorer)의 빈 공간에서 우클릭한 다음, 'import...'를 클릭한다.

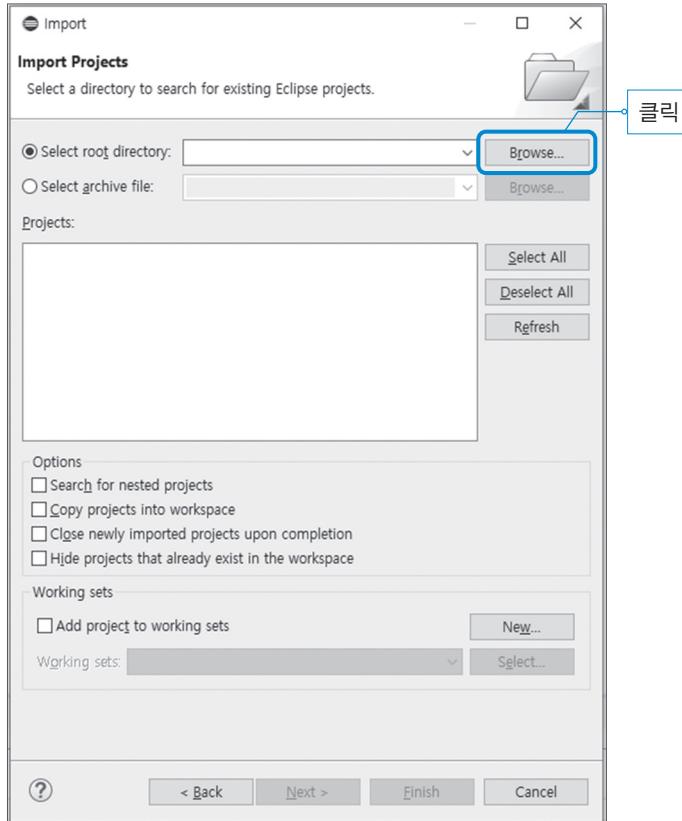


- 2 아래의 화면이 나타나면, 'General' 항목 아래의 'Existing Projects into Workspace'를 클릭하고 'Next >' 버튼을 누른다.

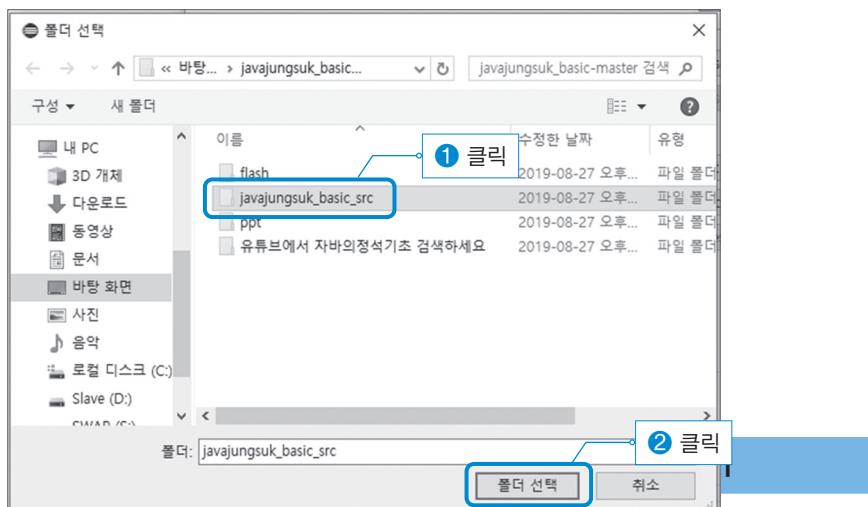


미리보기용 pdf입니다.

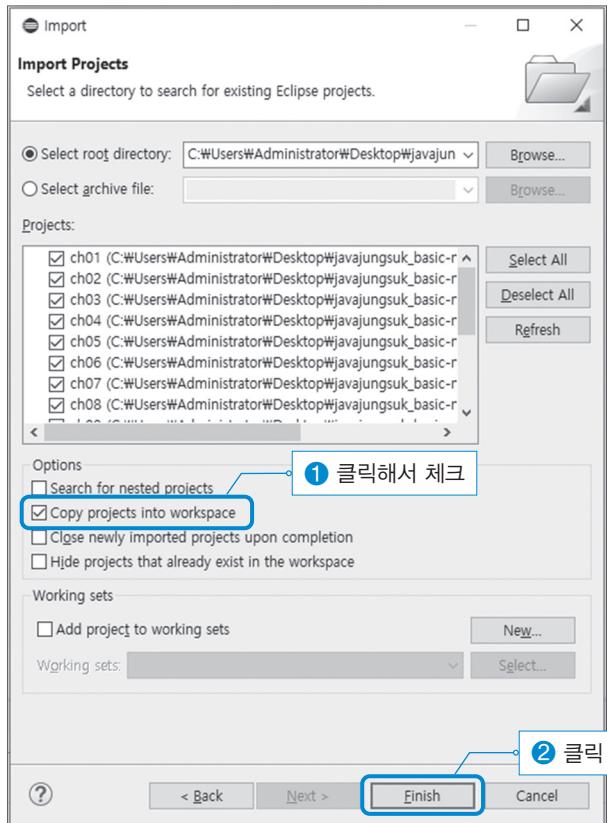
③ 이클립스로 import할 프로젝트가 담긴 폴더를 지정하기 위해 아래의 화면에서 'Browse...'버튼을 누른다.



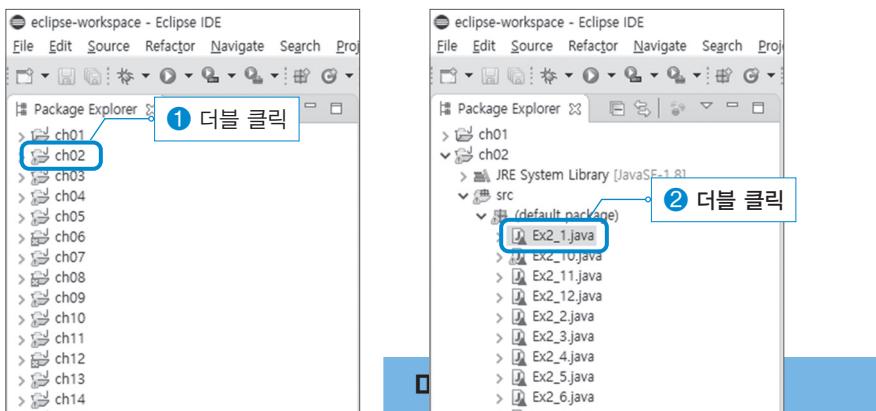
④ 바탕화면에 있는 'javajungsuk_basic-master'폴더 안의 'javajungsuk_basic_src'폴더를 클릭하고, '폴더 선택'버튼을 누른다.



5 지정된 프로젝트를 이클립스의 워크스페이스로 복사하기 위해 'Copy projects into workspace'를 체크하고 'Finish'버튼을 누른다.

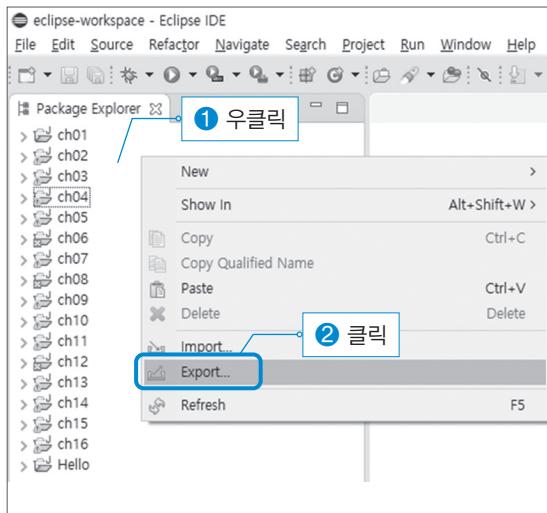


6 패키지 익스플로러에서 아래의 왼쪽 그림과 같이 보여야 한다. 만일 2장 첫번째 예제의 소스파일을 보려면, ch02 프로젝트를 더블 클릭해서, src폴더안의 디폴트 패키지(default package)에 있는 Ex2_1.java를 더블 클릭하면 된다.

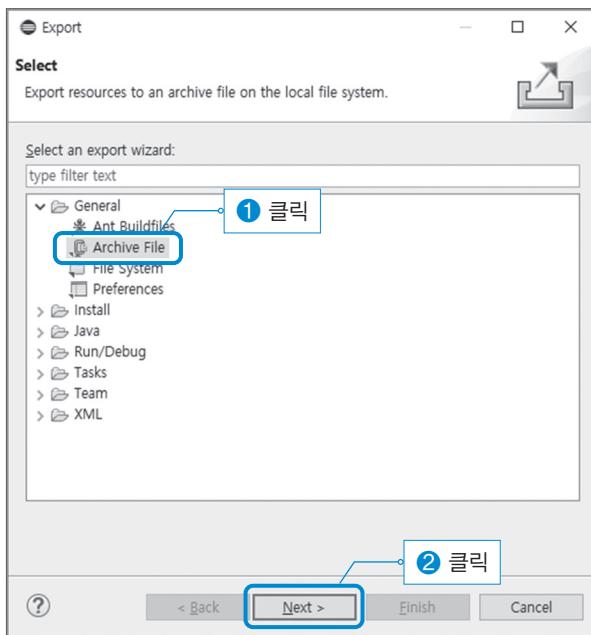


이클립스에서 작성한 프로젝트를 다른 사람에게 전달해야 할 때가 있다. 그럴 때는 전과 반대로 특정 프로젝트를 압축 파일로 익스포트(export)할 수 있다. 이 압축 파일을 옮긴 다음에 앞서 배운 것처럼 이클립스에서 임포트(import)하면 된다.

- 1** 패키지 익스플로러(Package Explorer)의 빈 공간에서 우클릭한 다음, 'Export...'를 클릭한다.



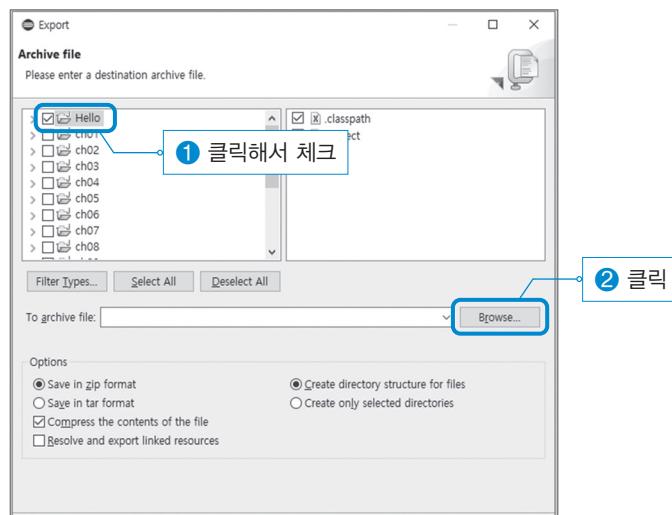
- 2** 아래의 화면이 나타나면, 'General' 항목 아래의 'Archive File'을 클릭하고 'Next >' 버튼을 누른다.



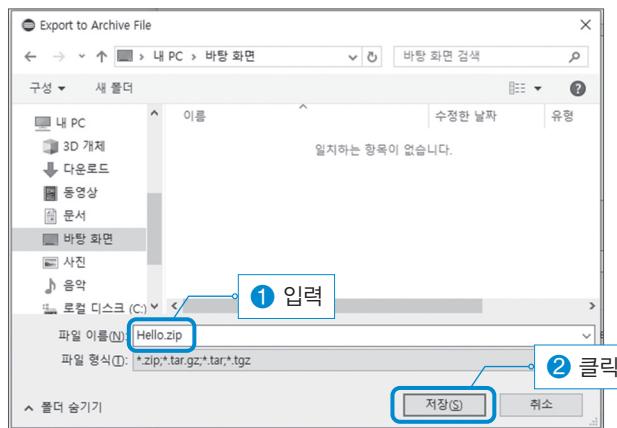
- ③ 아래 화면의 좌측에서 익스포트(export)할 프로젝트를 골라서 체크한 다음에 ‘Browse...’버튼을 클릭한다.

참고

‘Select All’버튼을 클릭하면 모든 프로젝트가 체크되고 ‘Deselect All’버튼을 클릭하면 모든 선택이 해제 된다.

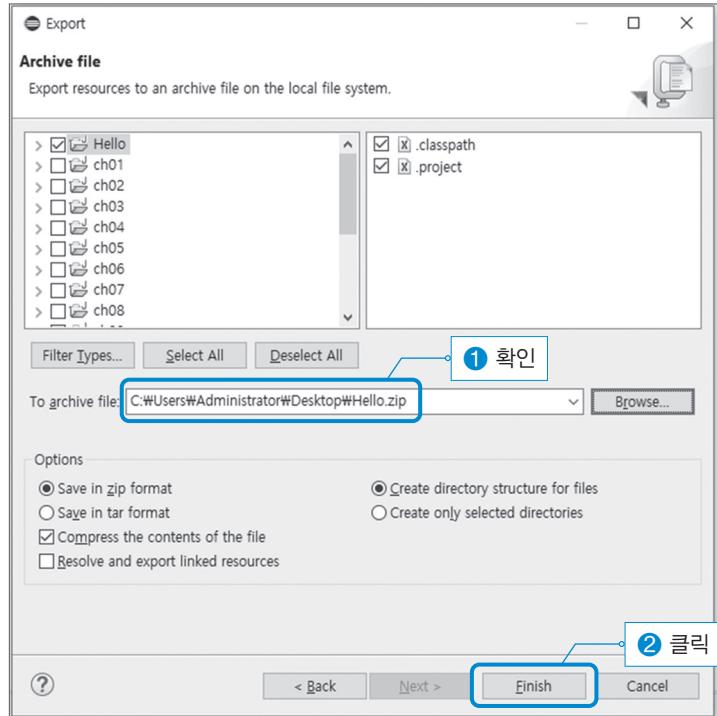


- ④ 아래와 같은 화면이 나타나면, 익스포트한 프로젝트를 저장할 위치와 파일이름을 지정하고 ‘저장’ 버튼을 누른다.



미리보기용 pdf입니다.

5 프로젝트가 익스포트될 파일의 경로와 이름을 확인한 후에 'Finish'버튼을 누른다. 아래 화면의 경우, 바탕화면에 'Hello.zip'이라는 파일 이름으로 저장될 것이다.



MEMO

C H A P T E R

2

변수

Variable



화면에 글자를 출력할 때는 `System.out.print()`을 사용한다. 괄호() 안에 출력하고자 하는 내용을 넣으면 된다.

```
System.out.print("Hello, world"); // 화면에 Hello, world를 출력
System.out.print(3+5);           // 화면에 8을 출력
System.out.print("3+5");         // 화면에 3+5를 출력
```

위의 코드에서 알 수 있듯이 괄호() 안에 숫자를 넣으면 계산된 결과가 출력되지만 큰따옴표 “” 안에 넣은 내용은 글자로 간주되어 계산되지 않고 있는 그대로 출력된다.

`System.out.print()` 외에도 `System.out.println()`이 있는데, 이 둘의 차이는 아래와 같다.

System.out.print() 괄호 안의 내용을 출력하고 줄바꿈을 하지 않는다.

System.out.println() 괄호 안의 내용을 출력하고 줄바꿈을 한다.

줄바꿈을 하지 않으면, 이전에 출력된 내용 바로 뒤에 이어서 출력된다.

다음의 예제들을 실행해서 배운 내용을 직접 확인해 보자.



자바는 대소문자를 구분한다. System을 system으로 입력하지 않도록 주의하자.

예제

2-1

```
class Ex2_1 {
    public static void main(String args[]) {
        System.out.println("Hello, world"); // 화면에 Hello, world를 출력하고 줄바꿈 한다.
        System.out.print("Hello");          // 화면에 Hello를 출력하고 줄바꿈 안한다.
        System.out.println("World");        // 화면에 World를 출력하고 줄바꿈 한다.
    }
}
```

결과
Hello, world
HelloWorld

예제

2-2

```
class Ex2_2 {
    public static void main(String args[]) {
        System.out.println("Hello, world"); // 화면에 Hello, world가 출력된다.
        System.out.print("3+5=");           // 화면에 3+5=를 출력하고 줄바꿈 안한다.
        System.out.println(3+5);            // 화면에 8이 출력된다.
    }
}
```

결과
Hello, world
3+5=8

미리보기용 pdf입니다.

사칙연산(+, -, *, /)이 포함된 식(式, expression)의 결과를 화면에 출력하려면, 앞서 배운 것과 같이 팔호 안에 식을 넣기만 하면 된다.

```
System.out.println(5+3);      // 5+3의 결과인 8이 화면에 출력된다.
```

위의 문장이 수행되는 과정은 다음과 같다.

```
System.out.println(5+3);      // 팔호 안의 식을 계산한다.  
→ System.out.println(8);    // 식이 계산 결과로 바뀌어 8이 화면에 출력된다.
```

덧셈(+) 외에도 뺄셈(-), 곱셈(*), 나눗셈(/)과 같은 연산자(operator)가 있으며, 자바는 이 외에도 다양한 종류의 연산자를 제공한다. 한 번에 다 소개하기보다 자주 사용되는 것들을 중심으로 조금씩 소개할 것이다.

예제
2-3

```
class Ex2_3 {  
    public static void main(String args[]) {  
        System.out.println(5+3);      // 화면에 5+3의 결과인 8이 출력된다.  
        System.out.println(5-3);      // 화면에 5-3의 결과인 2가 출력된다.  
        System.out.println(5*3);      // 화면에 5*3의 결과인 15가 출력된다.  
        System.out.println(5/3);      // 화면에 5/3의 결과인 1이 출력된다.  
    }  
}
```

결과	8
	2
	15
	1

실행결과를 보면 '5/3'의 결과가 왜 1인지 의아할 것이다. 이에 대해서는 곧 자세히 설명할 것 이니 지금은 정수 나누기 정수의 결과가 정수라는 정도만 기억해두자.

프로그래밍을 하다 보면 값을 저장해 둘 공간이 필요한데, 그 공간을 변수(variable)라 한다.

변수란? 하나의 값을 저장할 수 있는 저장공간

저장공간, 즉 변수가 필요하다면 먼저 변수를 선언해야 한다. 변수를 선언하는 방법은 다음과 같다.

`변수타입 변수이름; // 변수를 선언하는 방법`

변수의 타입은 변수에 저장할 값이 어떤 것인가에 따라 달라지며, 변수의 이름은 저장공간이 서로 구별될 수 있어야 하기 때문에 필요하다. 예를 들어 정수(integer)를 저장할 공간이 필요하다면 다음과 같이 변수를 선언한다.

`int x; // 정수(integer)를 저장하기 위한 변수 x를 선언`

위의 문장이 수행되면, x라는 이름의 변수(저장공간)가 생기며, 그림으로 그리면 다음과 같다.



그리고 이 변수에 값을 저장할 때는 다음과 같이 한다.

`x = 5; // 변수 x에 5를 저장`



수학에서는 '='가 같음을 의미하지만, 자바에서는 오른쪽의 값을 왼쪽에 저장하라는 의미의 '대입 연산자(assignment operator)'이다. 혼동하지 않도록 주의하자.

`x = 3; // 변수 x에 3을 저장. 기존의 값을 지워진다.`



변수는 오직 하나의 값만 저장할 수 있기 때문에, 이미 값이 저장된 변수에 새로운 값을 저장하면 기존의 값을 지워지고 새로 저장된 값만 남는다.

변수의 선언과 대입을 아래의 오른쪽 코드와 같이 한 줄로 간단히 할 수도 있다

`int x; // 변수의 선언
x = 5; // 변수에 대입`



`int x = 5;`

미리보기용 pdf입니다.

예제
2-4

```
class Ex2_4 {  
    public static void main(String args[]) {  
        int x = 5; // int x;와 x = 5;를 이처럼 한 줄로 합칠 수 있다.  
        System.out.println(x); // 화면에 x의 값인 5가 출력된다.  
  
        x = 10; // 변수 x에 10을 저장. 기존에 저장되어 있던 5는 지워짐.  
        System.out.println(x); // 화면에 x의 값인 10이 출력된다.  
    }  
}
```

결과
5
10

아래의 코드는 예제2-3의 일부인데, 5와 3 대신 다른 숫자의 계산결과를 얻으려면 매번 숫자를 다 바꿔줘야한다.

```
System.out.println(5+3); // 화면에 5+3의 결과인 8이 출력된다.  
System.out.println(5 - 3); // 화면에 5 - 3의 결과인 2가 출력된다.  
System.out.println(5 * 3); // 화면에 5 * 3의 결과인 15가 출력된다.  
System.out.println(5 / 3); // 화면에 5 / 3의 결과인 1이 출력된다.
```

그러나 변수를 이용하면 각 변수에 다른 값만 저장하고 나머지 부분은 바꾸지 않아도 된다.

```
int x = 5; // 변수에 다른 값을 저장하기만 하면 된다.  
int y = 3; // 변수에 다른 값을 저장하기만 하면 된다.  
System.out.println(x+y);  
System.out.println(x-y);  
System.out.println(x*y);  
System.out.println(x/y);
```

x, y의 값이 바뀌어도 변경하지 않아도 된다.

변수를 사용하지 않았을 때 보다 한결 편리하다. 이것만으로도 변수가 왜 필요한지 충분히 이해할 수 있을 것이다.

예제
2-5

```
class Ex2_5 {  
    public static void main(String args[]) {  
        int x = 10;  
        int y = 5;  
        System.out.println(x+y);  
        System.out.println(x-y);  
        System.out.println(x*y);  
        System.out.println(x/y);  
    }  
}
```

결과
15
5
50
2

변수를 선언할 때, 변수에 저장할 값의 종류에 따라 변수의 타입을 선택해야한다. 변수의 타입은 참조형과 8개의 기본형이 있는데, 일단 자주 쓰이는 타입만 소개한다.

분류	변수의 타입	설명
숫자	int long	정수(integer)를 저장하기 위한 타입(20억이 넘을 땐 long)
	float double	실수(floating-point number)를 저장하기 위한 타입 (float는 오차없이 7자리, double은 15자리)
문자	char	문자(character)를 저장하기 위한 타입
	String	여러 문자(문자열, string)를 저장하기 위한 타입

이 중에서도 아래 4개의 타입만 알아도 프로그래밍을 배우는데 큰 지장이 없다. 각 타입의 변수를 선언한 예는 다음과 같다.

```
int x = 100;           // 정수(integer)를 저장할 변수의 타입은 int로 한다.
double pi = 3.14;      // 실수를 저장할 변수의 타입은 double로 한다.
char ch = 'a';         // 문자(1개)를 저장할 변수의 타입은 char로 한다.
String str = "abc";    // 여러 문자(0~n개)를 저장할 변수의 타입은 String으로 한다.
```

이처럼 변수를 선언할 때 변수의 타입은 변수에 저장할 값의 종류에 맞는 것을 선택해야 한다. 달라도 허용되는 경우가 있지만, 나중에 자세히 설명할 것이다.

예제

2-6

```
class Ex2_6 {
    public static void main(String args[]) {
        int     x = 100;
        double pi = 3.14;
        char   ch = 'a';
        String str = "abc";

        System.out.println(x);
        System.out.println(pi);
        System.out.println(ch);
        System.out.println(str);
    }
}
```

결과	100
	3.14
	a
	abc

미리보기용 pdf입니다.

'상수(constant)'는 변수와 마찬가지로 '값을 저장할 수 있는 공간'이지만, 변수와 달리 한번 값을 저장하면 다른 값으로 변경할 수 없다. 상수를 선언하는 방법은 변수와 동일하며, 단지 변수의 타입 앞에 키워드 'final'을 붙여주기만 하면 된다.

```
final int MAX_SPEED = 10;
```

일단 상수에 값이 저장된 후에는 상수의 값을 변경하는 것이 허용되지 않는다.

```
final int MAX_VALUE; // 정수형 상수 MAX_VALUE를 선언
MAX_VALUE = 100; // OK. 상수에 처음으로 값 저장
MAX_VALUE = 200; // 예외. 상수에 저장된 값을 변경할 수 없음.
```

상수의 이름은 모두 대문자로 하는 것이 관례이며, 여러 단어로 이루어져 있는 경우 '_'로 구분한다.

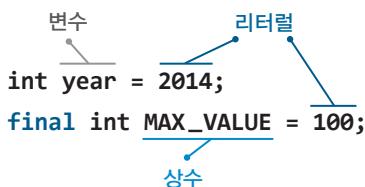
리터럴(literal)

원래 12, 123, 3.14, 'A'와 같은 값들이 '상수'인데, 프로그래밍에서는 상수를 '값을 한 번 저장하면 변경할 수 없는 저장공간'으로 정의하였기 때문에 이와 구분하기 위해 상수를 다른 이름으로 불러야만 했다. 그래서 상수 대신 리터럴이라는 용어를 사용한다. 많은 사람들이 리터럴이라는 용어를 어려워하는데, 리터럴은 단지 우리가 기존에 알고 있던 '상수'의 다른 이름일 뿐이다.

변수(variable) 하나의 값을 저장하기 위한 공간

상수(constant) 값을 한번만 저장할 수 있는 공간

리터럴(literal) 그 자체로 값을 의미하는 것



변수에 타입이 있는 것처럼 리터럴에도 타입이 있다. 변수의 타입은 저장될 ‘값의 타입(리터럴의 타입)’에 의해 결정되므로, 만일 리터럴에 타입이 없다면 변수의 타입도 필요없을 것이다.

종류	리터럴	접미사
논리형	false, true	없음
정수형	123, 0b0101, 077, 0xFF, 100L	L
실수형	3.14, 3.0e8, 1.4f, 0x1.0p-1	f, d
문자형	'A', '1', '\n'	없음
문자열	"ABC", "123", "A", "true"	없음

정수형과 실수형에는 여러 타입이 존재하므로, 리터럴에 접미사를 붙여서 타입을 구분한다. 정수형의 경우, long타입의 리터럴에 접미사 ‘l’ 또는 ‘L’을 붙이고, 접미사가 없으면 int타입의 리터럴이다. byte와 short타입의 리터럴은 별도로 존재하지 않으며 byte와 short타입의 변수에 값을 저장할 때는 int타입의 리터럴을 사용한다.

10진수 외에도 2, 8, 16진수로 표현된 리터럴을 변수에 저장할 수 있으며, 16진수라는 것을 표시하기 위해 리터럴 앞에 접두사 ‘0x’ 또는 ‘0X’를, 8진수의 경우에는 ‘0’을 붙인다.

```
int octNum = 010;           // 8진수 10, 10진수로 8
int hexNum = 0x10;          // 16진수 10, 10진수로 16
```

그리고 JDK1.7부터 정수형 리터럴의 중간에 구분자 ‘_’를 넣을 수 있게 되어서 큰 숫자를 편하게 읽을 수 있게 되었다.

```
long big = 100_000_000_000L;      // long big = 100000000000L;
long hex = 0xFFFF_FFFF_FFFF_FFFF; // long hex = 0xFFFFFFFFFFFFFFFL;
```

실수형에서는 float타입의 리터럴에 접미사 ‘f’ 또는 ‘F’를 붙이고, double타입의 리터럴에는 접미사 ‘d’ 또는 ‘D’를 붙인다.

```
float pi = 3.14f;             // 접미사 f 대신 F를 사용해도 된다. 생략불가
double rate = 1.618d;         // 접미사 d 대신 D를 사용해도 된다. 생략가능
```

실수형 리터럴에는 접미사를 붙여서 타입을 구분하며, float타입 리터럴에는 ‘f’를, double타입 리터럴에는 ‘d’를 붙인다. 정수형에서는 int가 기본 자료형인 것처럼 실수형에서는 double이 기본 자료형이라서 접미사 ‘d’는 생략이 가능하다. 접미사 f와 L 두 개는 꼭 기억하자.

미리보기용 pdf입니다.

'A'와 같이 작은따옴표로 문자 하나를 감싼 것을 '문자 리터럴'이라고 한다. 두 문자 이상은 큰 따옴표로 감싸야 하며 '문자열 리터럴'이라고 한다.

참고 문자열은 '문자의 연속된 나열'이라는 뜻이며, 영어로 'string'이라고 한다.

```
char ch = 'J';      // char ch = 'Java'; 이렇게 할 수 없다.  
String name = "Java"; // 변수 name에 문자열 리터럴 "Java"를 저장
```

char타입의 변수는 단 하나의 문자만 저장할 수 있으므로, 여러 문자(문자열)를 저장하기 위해서는 String타입을 사용해야 한다.

문자열 리터럴은 “” 안에 아무런 문자도 넣지 않는 것을 허용하며, 이를 빈 문자열(empty string)이라고 한다. 그러나 문자 리터럴은 반드시 “” 안에 하나의 문자가 있어야 한다.

```
String str = "";    // OK. 내용이 없는 빈 문자열  
char ch = '';     // 예리. '' 안에 반드시 하나의 문자가 필요  
char ch = ' ';   // OK. 공백 문자(blank)로 변수 ch를 초기화
```

원래 String은 클래스이므로 아래와 같이 객체를 생성하는 연산자 new를 사용해야 하지만 특별히 이와 같은 표현도 허용한다.

```
String name = new String("Java"); // String객체를 생성  
String name = "Java"; // 위의 문장을 간단히. 둘의 차이점은 9장에서 자세히 설명
```

숫자 뿐만 아니라 아래와 같이 두 문자열을 합칠 때도 덧셈(+)을 사용할 수 있다.

```
String name = "Ja" + "va";
String str = name + 8.0;
```

덧셈 연산자(+)는 피연산자가 모두 숫자일 때는 두 수를 더하지만, 피연산자 중 어느 한 쪽이 String이면 나머지 한 쪽을 먼저 String으로 변환한 다음 두 String을 결합한다.

어떤 타입의 변수도 문자열과 덧셈연산을 수행하면 그 결과가 문자열이 되는 것이다.

문자열 + any type	→	문자열 + 문자열	→	문자열
any type + 문자열	→	문자열 + 문자열	→	문자열

예를 들어 `7 + "7"`을 계산할 때 7이 String이 아니므로, 먼저 7을 String으로 변환한 다음 `"7" + "7"`을 수행하여 `"77"`을 결과로 얻는다. 다음은 문자열 결합의 몇 가지 예를 보여준다.

```
7 + " " → "7" + " " → "7 "
" " + 7 → " " + "7" → " 7"

7 + "7" → "7" + "7" → "77"

7 + 7 + "" → 14 + "" → "14" + "" → "14"
"" + 7 + 7 → "7" + 7 → "7" + "7" → "77"
```

덧셈 연산자는 왼쪽에서 오른쪽의 방향으로 연산을 수행하기 때문에 결합순서에 따라 결과가 달라진다는 것에 주의하자. 그리고 숫자 7을 문자열 `"7"`로 변환할 때는 아무런 내용도 없는 빈 문자열("")을 더해주면 된다는 것도 알아두자.

예제
2-7

```
class Ex2_7 {
    public static void main(String[] args) {
        String name = "Ja" + "va";
        String str = name + 8.0;

        System.out.println(name);
        System.out.println(str);
        System.out.println(7 + " ");
        System.out.println(" " + 7);
        System.out.println(7 + "");
        System.out.println("") + 7;
        System.out.println("") + "";
        System.out.println(7 + 7 + "");
        System.out.println("") + 7 + 7;
    }
}
```

결과	Java
	Java
	Java8.0
	7
	7
	7
	7
	14
	77

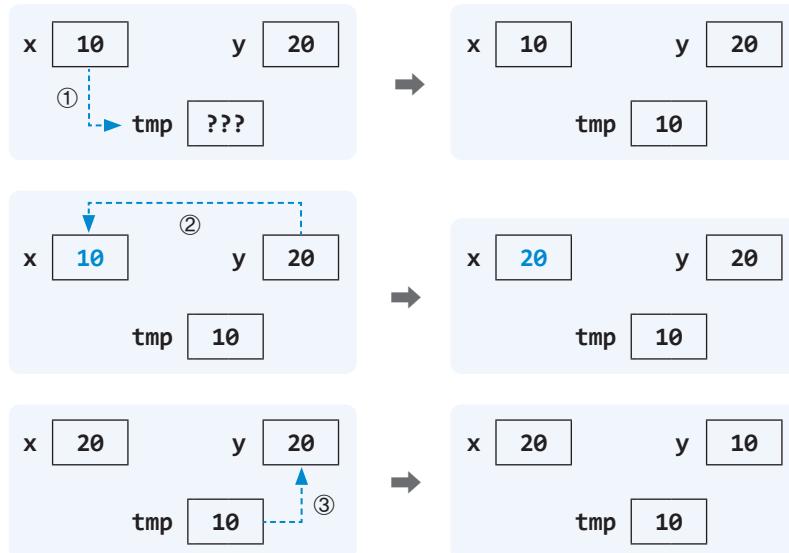
미리보기창 참조합니다.

두 변수 x와 y에 저장된 값을 바꾸려면 어떻게 해야 할까?

```
int x = 10;
int y = 20;
```

단순히 x의 값을 y에 저장하고, y의 값을 x에 저장해서는 원하는 결과를 얻을 수 없다. 두 컵에 담긴 내용물을 바꾸려면 빈 컵이 필요한 것처럼, 값을 임시로 저장할 변수가 하나 더 필요하다.

```
int tmp; // 임시로 값을 저장하기 위한 변수(빈 컵 역할)
tmp = x; // ① x의 값을 tmp에 저장
x = y; // ② y의 값을 x에 저장
y = tmp; // ③ tmp에 저장된 값을 y에 저장
```



예제

2-8

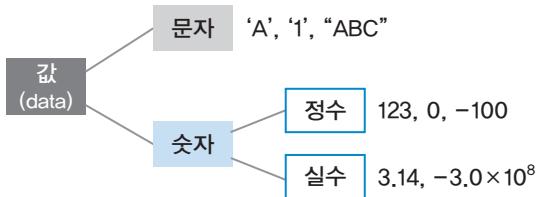
```
class Ex2_8 {
    public static void main(String args[]) {
        int x = 10, y = 5; // int x = 10; int y = 5;를 한 줄로
        System.out.println("x="+x);
        System.out.println("y="+y);

        int tmp = x; // 1. x의 값을 tmp에 저장
        x = y; // 2. y의 값을 x에 저장
        y = tmp; // 3. tmp에 저장된 값을 y에 저장
        System.out.println("x="+x);
        System.out.println("y="+y);
    }
}
```

결과	x=10 y=5 x=5 y=10
----	----------------------------

10 기본형과 참조형

우리가 주로 사용하는 값(data)의 종류(type)는 크게 ‘문자와 숫자’로 나눌 수 있으며, 숫자는 다시 ‘정수와 실수’로 나눌 수 있다.



이러한 값(data)의 종류(type)에 따라 값이 저장될 공간의 크기와 저장 형식을 정의한 것이 자료형(data type)이다. 자료형에는 문자형(char), 정수형(byte, short, int, long), 실수형(float, double) 등이 있으며, 변수를 선언할 때는 저장하려는 값의 특성을 고려하여 가장 알맞은 자료형을 변수의 타입으로 선택하면 된다.

기본형과 참조형

자료형은 크게 ‘기본형’과 ‘참조형’ 두 가지로 나눌 수 있는데, 기본형 변수는 실제 값(data)을 저장하는 반면, 참조형 변수는 어떤 값이 저장되어 있는 주소(memory address)를 값으로 갖는다. 자바는 C언어와 달리 참조형 변수 간의 연산을 할 수 없으므로 실제 연산에 사용되는 것은 모두 기본형 변수이다.

참고

메모리에는 1 byte 단위로 일련번호가 붙어 있는데, 이 번호를 ‘메모리 주소(memory address)’ 또는 간단히 ‘주소’라고 한다. 객체의 주소는 객체가 저장된 메모리 주소를 뜻한다.

기본형(primitive type)

논리형(boolean), 문자형(char), 정수형(byte, short, int, long), 실수형(float, double)
계산을 위한 실제 값을 저장한다. 모두 8개

참조형(reference type)

객체의 주소를 저장한다. 8개의 기본형을 제외한 나머지 타입.

Q. 자료형(data type)과 타입(type)의 차이가 뭔가요?

A. 기본형은 저장할 값(data)의 종류에 따라 구분되므로 기본형의 종류를 얘기할 때는 ‘자료형(data type)’이라는 용어를 씁니다. 그러나 참조형은 항상 ‘객체의 주소(4 byte 정수)’를 저장하므로 값(data)이 아닌, 객체의 종류에 의해 구분되므로 참조형 변수의 종류를 구분할 때는 ‘타입(type)’이라는 용어를 사용합니다. ‘타입(type)’이 ‘자료형(data type)’을 포함하는 보다 넓은 의미의 용어이므로 굳이 구분하지 않아도 됩니다.

미리보기용 pdf입니다.

기본형에는 모두 8개의 타입(자료형)이 있으며, 크게 논리형, 문자형, 정수형, 실수형으로 구분된다. 정수형 중에는 int가 기본이고, 실수형에서는 double이 기본이다.

종류 \ 크기	1 byte	2 byte	4 byte	8 byte
논리형	boolean			
문자형		char		
정수형	byte	short	int	long
실수형			float	double

기본 자료형의 종류와 크기는 반드시 외워야 하며, 아래의 문장들이 도움이 될 것이다.

- ▶ boolean은 true와 false 두 가지 값만 표현할 수 있으면 되므로 가장 작은 크기인 1 byte.
- ▶ char은 자바에서 유니코드(2 byte 문자체계)를 사용하므로 2 byte.
- ▶ byte는 크기가 1 byte라서 byte.
- ▶ int(4 byte)를 기준으로 짧아서 short(2 byte), 길어서 long(8 byte). (short ↔ long)
- ▶ float는 실수값을 부동소수점(floating-point)방식으로 저장하기 때문에 float.
- ▶ double은 float보다 **두 배의 크기**(8 byte)를 갖기 때문에 double.

그리고 각 타입의 변수가 저장할 수 있는 값의 범위는 다음과 같다.

자료형	저장 가능한 값의 범위	크기	
		bit	byte
boolean	false, true	8	1
char	'\u0000' ~ '\uffff' (0~2 ¹⁶ -1, 0~65535)	16	2
byte	-128 ~ 127 (-2 ⁷ ~2 ⁷ -1)	8	1
short	-32,768 ~ 32,767 (-2 ¹⁵ ~2 ¹⁵ -1)	16	2
int	-2,147,483,648 ~ 2,147,483,647 (-2 ³¹ ~2 ³¹ -1, 약 ±20억)	32	4
long	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807 (-2 ⁶³ ~2 ⁶³ -1)	64	8
float	1.4E-45 ~ 3.4E38 (1.4×10 ⁻⁴⁵ ~3.4×10 ³⁸)	32	4
double	4.9E-324 ~ 1.8E308 (4.9×10 ⁻³²⁴ ~1.8×10 ³⁰⁸)	64	8

참고

float와 double은 양의 범위만 적은 것이다. 음의 범위는 양의 범위에 음수 부호(-)를 붙이면 된다.

각 자료형이 가질 수 있는 값의 범위를 정확히 외울 필요는 없고, 정수형(byte, short, int, long)의 경우 $-2^{n-1} \sim 2^{n-1}-1$ (n은 bit수)이라는 정도만 기억하고 있으면 된다.

예를 들어 int형의 경우 32 bit(4 byte)이므로 $-2^{31} \sim 2^{31}-1$ 의 범위를 갖는다.

$$2^{10} = 1024 \approx 10^3 \text{이므로, } 2^{31} = 2^{10} \times 2^{10} \times 2^{10} \times 2 = 1024 \times 1024 \times 1024 \times 2 \approx 2 \times 10^9$$

그래서 int타입의 변수는 대략 10자리 수(약 ±20억)의 값을 저장할 수 있다는 것을 알 수 있다. 7~9자리의 수를 계산할 때는 넉넉하게 long타입(약 19자리)으로 변수를 선언하는 것이 좋다.

www.codechobo.com

12 printf를 이용한 출력

지금까지 화면 출력에 사용해온 `println()`은 사용하기 편하지만 변수의 값을 그대로 출력하므로, 값을 변환하지 않고는 다른 형식으로 출력할 수 없다. 같은 값이라도 다른 형식으로 출력하고 싶을 때. 예를 들어 소수점 둘째자리까지만 출력하거나 정수를 16진수나 8진수로 출력할 때 `printf()`를 사용하면 된다.

`printf()`는 ‘지시자(specifier)’를 통해 변수의 값을 여러 가지 형식으로 변환하여 출력하는 기능을 가지고 있다. ‘지시자’는 값을 어떻게 출력할 것인지를 지시해주는 역할을 한다. 정수형 변수에 저장된 값을 10진 정수로 출력할 때는 지시자 ‘%d’를 사용하며, 변수의 값을 지정된 형식으로 변환해서 지시자 대신 넣는다. 예를 들어 int타입의 변수 `age`의 값이 14일 때, `printf()`는 지시자 ‘%d’ 대신 14를 넣어서 출력한다.

```
System.out.printf("age:%d", age);
→ System.out.printf("age:%d", 14);
→ System.out.printf("age:14");      // "age:14"가 화면에 출력된다.
```

만일 출력하려는 값이 2개라면, 지시자도 2개를 사용해야 하며 출력될 값과 지시자의 순서는 일치해야 한다. 물론 3개 이상의 값도 지시자를 지정해서 출력할 수 있으며 개수의 제한은 없다.

```
System.out.printf("age:%d year:%d", age, year);
→ System.out.printf("age:%d year:%d", 14, 2019);
          ↑           ↑
          ↓           ↓
"age:14 year:2019"이 화면에 출력된다.
```

`println()`과 달리 `printf()`는 출력 후 줄바꿈을 하지 않는다. 줄바꿈을 하려면 지시자 ‘%n’을 따로 넣어줘야 한다.

참고 ‘%n’ 대신 ‘\n’을 사용해도 되지만, OS마다 줄바꿈 문자가 다를 수 있기 때문에 ‘%n’을 사용하는 것이 더 안전하다.

```
System.out.printf("age:%d", age);    // 출력 후 줄바꿈을 하지 않는다.
System.out.printf("age:%d\n", age);   // 출력 후 줄바꿈을 한다.
```

`printf()`의 지시자 중에서 자주 사용되는 것만 뽑아보면 다음과 같다.

지시자	설명
%d	10진(decimal) 정수의 형식으로 출력
%x	16진(hexa-decimal) 정수의 형식으로 출력
%f	부동 소수점(floating-point)의 형식으로 출력
%c	문자(character)로 출력
%s	문자열(string)로 출력

미리보기용 pdf입니다.

예제
2-9

```
class Ex2_9 {
    public static void main(String[] args) {
        String url = "www.codechobo.com";
        float f1 = .10f;      // 0.10, 1.0e-1
        float f2 = 1e1f;      // 10.0, 1.0e1, 1.0e+1
        float f3 = 3.14e3f;
        double d = 1.23456789;
        System.out.printf("f1=%f, %e, %g%n", f1, f1, f1);
        System.out.printf("f2=%f, %e, %g%n", f2, f2, f2);
        System.out.printf("f3=%f, %e, %g%n", f3, f3, f3);
        System.out.printf("d=%f%n", d);
        System.out.printf("d=%14.10f%n", d); // 전체 14자리 중 소수점 10자리
        System.out.printf("[12345678901234567890]%n");
        System.out.printf("[%s]%n", url);
        System.out.printf("[%20s]%n", url);
        System.out.printf("[%-.20s]%n", url); // 왼쪽 정렬
        System.out.printf("%.8s%n", url); // 왼쪽에서 8글자만 출력
    }
}
```

결과

```
f1=0.100000, 1.000000e-01, 0.100000
f2=10.000000, 1.000000e+01, 10.0000
f3=3140.000000, 3.140000e+03, 3140.00
d=1.234568 ← 마지막 자리 반올림됨
d= 1.2345678900
[12345678901234567890]
[www.codechobo.com]
[ www.codechobo.com]
[www.codechobo.com ]
[www.code]
```

실수형 값의 출력에 사용되는 지시자는 '%f', '%e', '%g'가 있는데, '%f'가 주로 쓰이고 '%e'는 지수형태로 출력할 때, '%g'는 값을 간략하게 표현할 때 사용한다.

'%f'는 기본적으로 소수점 아래 6자리까지만 출력하기 때문에 소수점 아래 7자리에서 반올림한다. 그래서 1.23456789가 1.234568로 출력되었다. 그리고 다음과 같이 전체 자리수와 소수점 아래의 자리수를 지정할 수도 있다.

%전체자리.소수점아래자리f

```
System.out.printf("d=%14.10f%n", d); // 전체 14자리 중 소수점 아래 10자리
```



소수점도 한자리를 차지하며, 소수점 아래의 빈자리는 0으로 채우고 정수의 빈자리는 공백으로 채워서 전체 자리수를 맞춘다.

참고

지시자를 '%014.10'으로 지정했다면, 양쪽 빈자리를 모두 0으로 채웠을 것이다.

지시자 '%s'에도 숫자를 추가하면 원하는 만큼의 출력공간을 확보하거나 문자열의 일부만 출력할 수 있다.

```
System.out.printf("[%s]%n", url); // 문자열의 길이만큼 출력공간을 확보  
System.out.printf("[%20s]%n", url); // 최소 20글자 출력공간 확보.(우측정렬)  
System.out.printf("[% -20s]%n", url); // 최소 20글자 출력공간 확보.(좌측정렬)  
System.out.printf("%.8s%n", url); // 왼쪽에서 8글자만 출력
```

지정된 숫자보다 문자열의 길이가 작으면 빈자리는 공백으로 출력된다. 공백이 있는 경우 기본적으로 우측 끝에 문자열을 붙이지만, ‘-’를 붙이면 좌측 끝에 붙인다. 그리고 ‘.’을 붙이면 문자열의 일부만 출력할 수 있다. 숫자를 직접 바꿔가면서 다양하게 테스트 해보자.

미리보기용 pdf입니다.

화면으로부터 입력받는 방법은 아직 배우지 않은 것들이 있지만, 본인이 직접 입력을 하면 자칫 지루할 수 있는 내용이 좀 더 재미있어지지 않을까하는 생각에서 미리 소개하게 되었다.

나중에 자세히 배울 테니 지금은 이해하기보다는 가져다 사용하는 정도로만 활용해주었으면 한다. 먼저 아래의 한 문장을 추가해주자.

```
import java.util.Scanner; // Scanner클래스를 사용하기 위해 추가
```

그 다음엔 Scanner클래스의 객체를 생성한다.

```
Scanner scanner = new Scanner(System.in); // Scanner클래스의 객체를 생성
```

그리고 nextLine()이라는 메서드를 호출하면, 입력대기 상태에 있다가 입력을 마치고 ‘엔터 키(Enter)’를 누르면 입력한 내용이 문자열로 반환된다.

```
String input = scanner.nextLine(); // 입력받은 내용을 input에 저장  
int num = Integer.parseInt(input); // 입력받은 내용을 int타입의 값으로 변환
```

만일 입력받은 문자열을 숫자로 변환하려면, Integer.parseInt()라는 메서드를 이용해야 한다. 이 메서드는 문자열을 int타입의 정수로 변환한다.

사실 Scanner클래스에는 nextInt()나 nextFloat()와 같이 변환없이 숫자로 바로 입력받을 수 있는 메서드들이 있고, 이 메서드들을 사용하면 문자열을 숫자로 변환하는 수고는 하지 않아도 된다.

```
int num = scanner.nextInt(); // 정수를 입력받아서 변수 num에 저장
```

예제

2-10

```
import java.util.Scanner; // Scanner를 사용하기 위해 추가

class Ex2_10 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("두자리 정수를 하나 입력해주세요.>");
        String input = scanner.nextLine();
        int num = Integer.parseInt(input); // 입력받은 문자열을 숫자로 변환

        System.out.println("입력내용 :" + input);
        System.out.printf("num=%d\n", num);
    }
}
```

결과

두자리 정수를 하나 입력해주세요.>22
입력내용 :22
num=22

실행결과의 22는 이클립스의 콘솔(Console)에 키보드로 입력한 것이며, 22대신 원하는 숫자를 직접 입력해보자. 만일 입력내용에 문자 또는 기호(특히 공백)가 있으면 오류가 발생한다.

www.codechobo.com

15 정수형의 오버플로우

만일 4 bit 2진수의 최대값인 ‘1111’에 1을 더하면 어떤 결과를 얻을까? 4 bit의 범위를 넘어서는 값이 되기 때문에 에러가 발생할까?

$$\begin{array}{r}
 \boxed{1} \quad \boxed{1} \quad \boxed{1} \quad \boxed{1} \\
 +) \quad \boxed{0} \quad \boxed{0} \quad \boxed{0} \quad \boxed{1} \\
 \hline
 \boxed{?} \quad \boxed{?} \quad \boxed{?} \quad \boxed{?}
 \end{array}$$

원래 2진수 ‘1111’에 1을 더하면 ‘10000’이 되지만, 4 bit로는 4자리의 2진수만 저장할 수 있기 때문에 ‘0000’이 된다. 즉, 5자리의 2진수 ‘10000’중에서 하위 4 bit만 저장하게 되는 것이다. 이처럼 연산과정에서 해당 타입이 표현할 수 있는 값의 범위를 넘어서는 것을 오버플로우(overflow)라고 한다. 오버플로우가 발생했다고 해서 에러가 발생하는 것은 아니다. 다만 예상했던 결과를 얻지 못할 뿐이다. 애초부터 오버플로우가 발생하지 않게 충분한 크기의 타입을 선택해서 사용해야 한다.

10진수	2진수
$ \begin{array}{r} \boxed{9} \quad \boxed{9} \quad \boxed{9} \quad \boxed{9} \\ +) \qquad \qquad \qquad 1 \\ \hline \boxed{1} \quad \boxed{0} \quad \boxed{0} \quad \boxed{0} \end{array} $	$ \begin{array}{r} \boxed{1} \quad \boxed{1} \quad \boxed{1} \quad \boxed{1} \\ +) \qquad \qquad \qquad 1 \\ \hline \boxed{1} \quad \boxed{0} \quad \boxed{0} \quad \boxed{0} \end{array} $
← 저장할 공간이 없어서 1은 버려짐 →	

오버플로우는 ‘자동차 주행표시기(odometer)’나, ‘계수기(counter)’ 등 우리의 일상생활에서도 발견할 수 있는데, 네 자리 계수기라면 ‘0000’부터 ‘9999’까지 밖에 표현하지 못하므로 최대값인 ‘9999’ 다음의 숫자는 ‘0000’이 될 것이다. 원래는 10000이 되어야하는데 다섯 자리는 표현할 수 없어서 맨 앞의 1은 버려지기 때문이다.

그러면 이번엔 반대로 최소값인 ‘0000’에서 1을 감소시키면 어떤 결과를 얻을까? 0에서 1을 뺄 수 없으므로 ‘0000’ 앞에 저장되지 않은 1이 있다고 가정하고 뺄셈을 한다. 결과는 아래와 같이 네 자리로 표현할 수 있는 최대값이 된다.

10진수	2진수
$ \begin{array}{r} \boxed{1} \quad \boxed{0} \quad \boxed{0} \quad \boxed{0} \\ -) \qquad \qquad \qquad 1 \\ \hline \boxed{9} \quad \boxed{9} \quad \boxed{9} \quad \boxed{9} \end{array} $	$ \begin{array}{r} \boxed{1} \quad \boxed{0} \quad \boxed{0} \quad \boxed{0} \\ -) \qquad \qquad \qquad 1 \\ \hline \boxed{1} \quad \boxed{1} \quad \boxed{1} \quad \boxed{1} \end{array} $
← 저장되지 않은 1이 있다고 가정 →	

미리보기용 pdf입니다.

이는 마치 계수기를 거꾸로 돌리는 것과 같다. ‘0000’에서 정방향으로 돌리면 ‘0001’이 되지만 역방향으로 돌리면 ‘9999’가 되는 것이다.

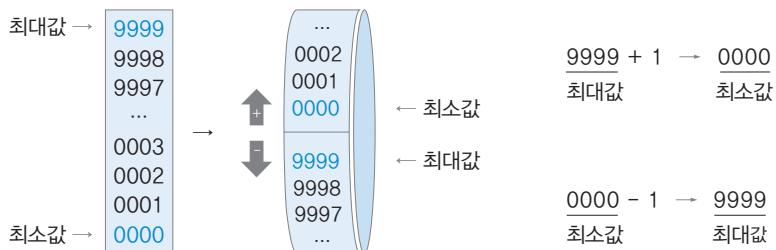
참고 TV의 채널을 증가시키다가 마지막 채널에서 채널을 더 증가시키면 첫 번째 채널로 이동하고, 첫번째 채널에서 채널을 감소시키면 마지막 채널로 이동하는 것과 유사하다.

그래서 정수형 타입이 표현할 수 있는 최대값에 1을 더하면 최소값이 되고, 최소값에서 1을 빼면 최대값이 된다.

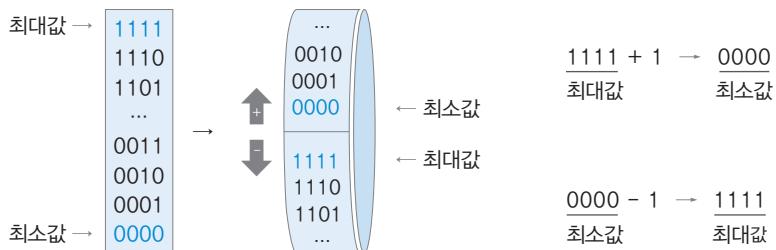
$$\text{최대값} + 1 \rightarrow \text{최소값}$$

$$\text{최소값} - 1 \rightarrow \text{최대값}$$

아래 그림과 같이 최소값과 최대값을 이어 놓았다고 생각하면 오버플로우의 결과를 더 이해하기 쉽다.



위의 그림을 2진수로 바꾸면 다음과 같다.



4 bit 2진수의 최소값인 ‘0000’부터 시작해서 1씩 계속 증가하다 최대값인 ‘1111’을 넘으면 다시 ‘0000’이 된다. 그래서 값을 1씩 무한히 증가시켜도 ‘0000’과 ‘1111’의 범위를 벗어나지 않게 된다.

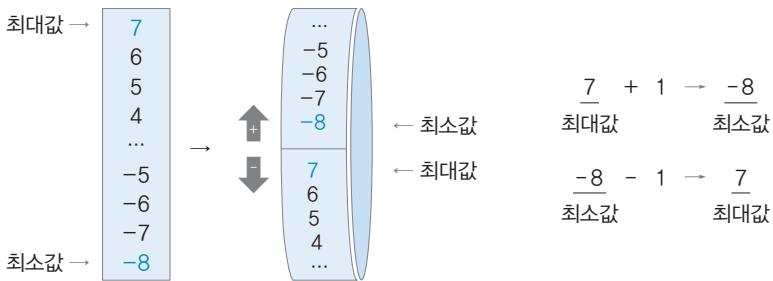
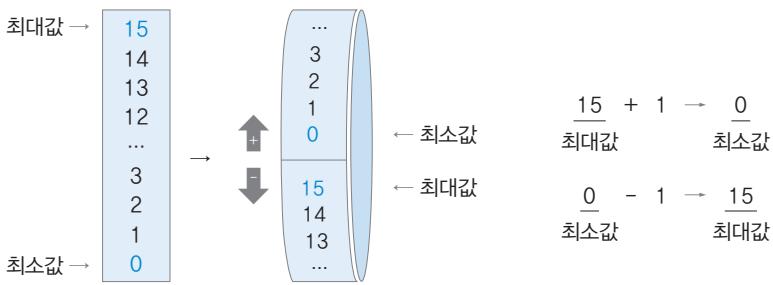
부호없는 정수와 부호있는 정수는 표현범위 즉, 최대값과 최소값이 다르기 때문에 오버플로우가 발생하는 시점이 다르다. 부호없는 정수는 2진수로 '0000'이 될 때 오버플로우가 발생하고, 부호있는 정수는 부호비트가 0에서 1이 될 때 오버플로우가 발생한다.

부호없는 10진수	2진수	부호있는 10진수
...
7	0111	7 ← 최대값
8	1000	-8 ← 최소값
9	1001	-7
10	1010	-6
11	1011	-5
12	1100	-4
13	1101	-3
14	1110	-2
최대값 → 15	1111	-1
최소값 → 0	0000	0
...

오버플로우 발생 {

} 오버플로우 발생

부호없는 정수(4 bit)의 경우 표현범위가 '0~15'이므로 이 값이 계속 반복되고, 부호있는 정수(4 bit)의 경우 표현범위가 '-8~7'이므로 이 값이 무한히 반복된다.



미리보기용 pdf입니다.

예제
2-11

```
class Ex2_11 {
    public static void main(String[] args) {
        short sMin = -32768, sMax = 32767;
        char cMin = 0, cMax = 65535;

        System.out.println("sMin = " + sMin);
        System.out.println("sMin-1= " + (short)(sMin-1));
        System.out.println("sMax = " + sMax);
        System.out.println("sMax+1= " + (short)(sMax+1));
        System.out.println("cMin = " + (int)cMin);
        System.out.println("cMin-1= " + (int)--cMin);
        System.out.println("cMax = " + (int)cMax);
        System.out.println("cMax+1= " + (int)++cMax);
    }
}
```

결과

sMin	= -32768
sMin-1=	32767
sMax	= 32767
sMax+1=	-32768
cMin	= 0
cMin-1=	65535
cMax	= 65535
cMax+1=	0

short타입과 char타입의 최대값과 최소값에 1을 더하거나 빼 결과를 출력하였다. 실행결과를 좀더 이해하기 쉽게 정리하면 다음과 같다.

sMin - 1	→	sMax	// 최소값 - 1 → 최대값
-32768		32767	
sMax + 1	→	sMin	// 최대값 + 1 → 최소값
32767		-32768	
cMin - 1	→	cMax	// 최소값 - 1 → 최대값
0		65535	
cMax + 1	→	cMin	// 최대값 + 1 → 최소값
65535		0	

최소값에서 1을 빼면 최대값이 되고, 최대값에 1을 더하면 최소값이 된다는 것을 알 수 있다.

개수	부호	char(부호X)	2진수(16 bit)	short(부호O)	부호
65536개 (2 ¹⁶ 개)	0 (1개)	최소값 → 0	0000000000000000	0	0 (1개)
	양수 (2 ¹⁶ -1개, 65535개)	1	0000000000000001	1	양수 (2 ¹⁵ -1개, 32767개)
		
		32766	0111111111111110	32766	
		32767	0111111111111111	32767 ← 최대값	
		32768	1000000000000000	-32768 ← 최소값	
		32769	1000000000000001	-32767	
		
		65534	1111111111111110	-2	
	최대값 → 65535	1111111111111111	-1	음수 (2 ¹⁵ 개, 32768개)	

타입 간의 변환은 프로그램에서 자주 사용되므로 반드시 정리해서 알아둘 필요가 있다.

- 숫자를 문자로 변환 – 숫자에 '0'을 더한다.

`(char)(3 + '0') → '3'`

- 문자를 숫자로 변환 – 문자에서 '0'을 뺀다.

`'3' - '0' → 3`

- 숫자를 문자열로 변환 – 숫자에 빈 문자열("")을 더한다.

`3 + "" → "3"`

- 문자열을 숫자로 변환 – `Integer.parseInt()` 또는 `Double.parseDouble()`을 사용한다.

`Integer.parseInt("3") → 3`

`Double.parseDouble("3.14") → 3.14`

- 문자열을 문자로 변환 – `charAt(0)`을 사용한다.

`"3".charAt(0) → '3'`

- 문자를 문자열로 변환 – 빈 문자열("")을 더한다.

`'3' + "" → "3"`

예제

2-12

```
class Ex2_12 {
    public static void main(String args[]) {
        String str = "3";

        System.out.println(str.charAt(0) - '0');
        System.out.println('3' - '0' + 1);
        System.out.println(Integer.parseInt("3") + 1);
        System.out.println("3" + 1);
        System.out.println((char)(3 + '0'));
    }
}
```

결과	3
	4
	4
	31
	3

미리보기용 pdf입니다.

연습문제

2-1 다음 표의 빈칸에 8개의 기본형(primitive type)을 알맞은 자리에 넣으시오.

종류 \ 크기	1 byte	2 byte	4 byte	8 byte
논리형				
문자형				
정수형				
실수형				

2-2 다음 중 키워드가 아닌 것은?(모두 고르시오)

- ① if ② True ③ NULL ④ Class ⑤ System

2-3 char타입(2 byte)의 변수에 저장될 수 있는 정수 값의 범위는? (10진수로 적으시오)

2-4 다음 중 변수를 잘못 초기화 한 것은? (모두 고르시오)

- ① byte b = 256;
② char c = '';
③ char answer = 'no';
④ float f = 3.14
⑤ double d = 1.4e3f;

2-5 다음의 문장에서 리터럴, 변수, 상수, 키워드를 적으시오.

```
int i = 100;
long l = 100L;
final float PI = 3.14f;
```

- 리터럴 :

- 키워드 :

- 변수 :

- 상수 :

2-6 다음 중 기본형(primitive type)이 아닌 것은?

- ① int
- ② Byte
- ③ double
- ④ boolean

2-7 다음 문장들의 출력결과를 적으세요. 오류가 있는 문장의 경우, 괄호 안에 '오류'라고 적으시오.

- ① System.out.println("1" + "2") → ()
- ② System.out.println(true + "") → ()
- ③ System.out.println('A' + 'B') → ()
- ④ System.out.println('1' + 2) → ()
- ⑤ System.out.println('1' + '2') → ()
- ⑥ System.out.println('J' + "ava") → ()
- ⑦ System.out.println(true + null) → ()

2-8 아래는 변수 x, y, z의 값을 서로 바꾸는 예제이다. 결과와 같이 출력되도록 (1)에 알맞은 코드를 넣으시오.

```
public class Exercise2_8 {
    public static void main(String[] args) {
        int x = 1;
        int y = 2;
        int z = 3;

        /*
         * (1) 알맞은 코드를 넣어 완성하시오.
         */
        System.out.println("x=" + x);
        System.out.println("y=" + y);
        System.out.println("z=" + z);
    }
}
```

결과	x=2 y=3 z=1
----	-------------------

C H A P T E R

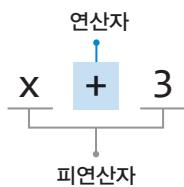
3

연산자

operator

연산자는 ‘연산을 수행하는 기호’를 말한다. 예를 들어 ‘+’기호는 덧셈 연산을 수행하며, ‘덧셈 연산자’라고 한다. 자바에서는 사칙연산(+, -, *, /)을 비롯해서 다양한 연산자를 제공한다. 연산자가 연산을 수행하려면 반드시 연산의 대상이 있어야하는데, 이것을 ‘피연산자(operand)’라고 한다.

다음과 같이 ‘ $x + 3$ ’이라는 식(式)이 있을 때, ‘+’는 두 피연산자를 더해서 그 결과를 반환하는 덧셈 연산자이고, 변수 x 와 상수 3은 이 연산자의 피연산자이다.



이처럼 덧셈 연산자 ‘+’는 두 값을 더한 결과를 반환하므로, 두 개의 피연산자를 필요로 한다. 연산자는 피연산자로 연산을 수행하고 나면 항상 결과값을 반환한다. 예를 들어 x 의 값이 5일 때, 덧셈 연산 ‘ $x + 3$ ’의 결과값은 8이 된다.

연산자와 피연산자를 조합하여 계산하고자 하는 바를 표현한 것을 ‘식(式, expression)’이라고 한다. 그리고 식을 계산하여 결과를 얻는 것을 ‘식을 평가(evaluation)한다’고 한다. 하나의 식을 평가(계산)하면, 단 하나의 결과를 얻는다. 만일 x 의 값이 5라면, 아래의 식을 평가한 결과는 23이 된다.

```

    4 * x + 3
    → 4 * 5 + 3
    → 23
  
```

식이 평가되어 23이라는 결과를 얻었지만, 이 값이 어디에도 쓰이지 않고 사라지기 때문에 이 식은 아무런 의미가 없다. 그래서 아래와 같이 대입 연산자 '='를 사용해서 변수와 같이 값을 저장할 수 있는 공간에 결과를 저장해야 한다.

```

y = 4 * x + 3;           // x의 값이 5라면, y의 값은 23이 된다.
System.out.println(y);   // y의 값인 23이 화면에 출력된다.
  
```

그 다음에 변수 y 에 저장된 값을 다른 곳에 사용하거나 화면에 출력함으로써 의미있는 결과를 얻을 수 있다. 만일 식의 평가결과를 출력하기만 원할 뿐, 이 값을 다른 곳에 사용하지 않을 것이면 아래처럼 변수에 저장하지 않고 `println`메서드의 팔호() 안에 직접 식을 써도 된다.

```

System.out.println(4 * x + 3); // x의 값이 5라고 가정하면
→ System.out.println(23);
  
```

미리보기용 pdf입니다.

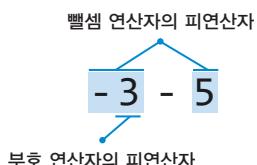
배워야 할 연산자의 개수가 많아서 부담스러울 수 있는데, 기능이 비슷한 것들끼리 묶어놓고 보면 몇 종류 안 된다.

종류	연산자	설명
산술 연산자	+ - * / % << >>	사칙 연산과 나머지 연산(%)
비교 연산자	> < >= <= == !=	크고 작음과 같고 다른을 비교
논리 연산자	&& ! & ^ ~	'그리고(AND)'와 '또는(OR)'으로 조건을 연결
대입 연산자	=	우변의 값을 좌변에 저장
기 타	(type) ?: instanceof	형변환 연산자, 삼항 연산자, instanceof연산자

피연산자의 개수로 연산자를 분류하기도 하는데, 피연산자의 개수가 하나면 ‘단항 연산자’, 두 개면 ‘이항 연산자’, 세 개면 ‘삼항 연산자’라고 부른다. 대부분의 연산자는 ‘이항 연산자’이다.



위의 식에는 두 개의 연산자가 포함되어 있는데, 둘 다 같은 기호‘-’로 나타내지만 염연히 다른 연산자이다. 왼쪽의 것은 ‘부호 연산자’이고, 오른쪽의 것은 ‘뺄셈 연산자’이다. 이처럼 서로 다른 연산자의 기호가 같은 경우도 있는데, 이럴 때는 피연산자의 개수로 구분이 가능하다.

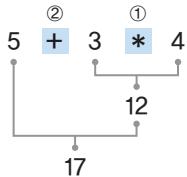


‘부호 연산자’는 단항 연산자로 피연산자가 ‘3’ 한 개뿐이지만, ‘뺄셈 연산자’는 이항 연산자로 피연산자가 ‘-3’과 ‘5’ 두 개이다.

이처럼 연산자를 기능별, 피연산자의 개수별로 나누어 분류하는 것은 곧이어 배우게 될 ‘연산자의 우선순위’ 때문이기도 하다. 연산자마다 우선순위가 다르지만, 같은 종류의 연산자들은 우선순위가 비슷하기 때문에 각 종류별로 우선순위를 외우면 기억하기 더 쉽다.

03 연산자의 우선순위

식에 사용된 연산자가 둘 이상인 경우, 연산자의 우선순위에 의해서 연산순서가 결정된다. 곱셈과 나눗셈(*, /)은 덧셈과 뺄셈(+, -)보다 우선순위가 높다는 것은 이미 수학에서 배워서 알고 있을 것이다. 그래서 아래의 식은 ‘ $3 * 4$ ’가 먼저 계산된 다음, 그 결과에 5를 더해서 17을 결과로 얻는다.

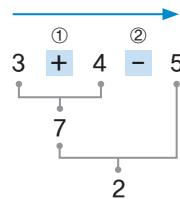


이처럼 연산자의 우선순위는 대부분 상식적인 선에서 해결된다. 아래의 표를 통해 이 사실을 한번 확인해 보자.

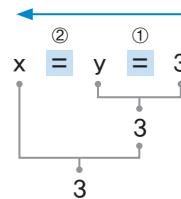
식	설명
$-x + 3$	단항 연산자가 이항 연산자보다 우선순위가 높다. 그래서 x 의 부호를 바꾼 다음 덧셈이 수행된다. 여기서 ‘-’는 뺄셈 연산자가 아니라 부호 연산자이다.
$x + 3 * y$	곱셈과 나눗셈이 덧셈과 뺄셈보다 우선순위가 높다. 그래서 ‘ $3 * y$ ’가 먼저 계산된다.
$x + 3 > y - 2$	비교 연산자(>)보다 산술 연산자 ‘+’와 ‘-’가 먼저 수행된다. 그래서 ‘ $x + 3$ ’과 ‘ $y - 2$ ’가 먼저 계산된 다음에 ‘>’가 수행된다.
$x > 3 \&& x < 5$	논리 연산자 ‘&&’보다 비교 연산자가 먼저 수행된다. 그래서 ‘ $x > 3$ ’와 ‘ $x < 5$ ’가 먼저 계산된 다음에 ‘&&’가 수행된다. 식의 의미는 ‘ x 가 3보다 크고 5보다 작다’이다.
<code>result = x + y * 3;</code>	대입 연산자는 연산자 중에서 제일 우선순위가 낮다. 그래서 우변의 최종 연산결과가 변수 <code>result</code> 에 저장된다.

미리보기용 pdf입니다.

하나의 식에 같은 우선순위의 연산자들이 여러 개 있는 경우, 어떤 순서로 연산을 수행할까? 우선순위가 같다고 해서 아무거나 먼저 처리하는 것은 아니고 나름대로의 규칙을 가지고 있는데, 그 규칙을 ‘연산자의 결합규칙’이라고 한다.



(a) 연산자의 결합규칙이 왼쪽에서 오른쪽인 경우



(b) 연산자의 결합규칙이 오른쪽에서 왼쪽인 경우

위 그림의 (a)에서 수식 ‘ $3 + 4 - 5$ ’는 덧셈연산자‘+’의 결합방향이 왼쪽에서 오른쪽이므로 수식의 왼쪽에 있는 ‘ $3 + 4$ ’를 먼저 계산하고, 그 다음에 ‘ $3 + 4$ ’의 연산결과인 7과 5의 뺄셈을 수행한다. (b)에서 대입 연산자는 연산자의 결합규칙이 오른쪽에서 왼쪽이므로 수식 ‘ $x = y = 3$ ’에서 오른쪽의 대입연산자부터 처리한다.

따라서 ‘ $y = 3$ ’이 먼저 수행되어서 y 에 3이 저장되고 그 다음에 ‘ $x = 3$ ’이 수행되어 x 에도 3이 저장된다.

$$\begin{aligned} &x = y = 3 \\ \rightarrow &x = 3 \\ \rightarrow &3 \end{aligned}$$

앞서 모든 연산자는 연산결과를 갖는다고 했는데, 대입 연산자도 예외는 아니다. 대입 연산자는 우변의 값을 좌변에 저장하고, 저장된 값을 연산결과로 반환한다. 그래서 ‘ $y = 3$ ’의 연산결과가 3이 되는 것이다.

1. 산술 > 비교 > 논리 > 대입. 대입은 제일 마지막에 수행된다.
2. 단항(1) > 이항(2) > 삼항(3). 단항 연산자의 우선순위가 이항 연산자보다 높다.
3. 단항 연산자와 대입 연산자를 제외한 모든 연산의 진행방향은 왼쪽에서 오른쪽이다.

예제

3-1

```
class Ex3_1 {
    public static void main(String[] args) {
        int x, y;

        x = y = 3; // y에 3이 저장된 후에, x에 3이 저장된다.
        System.out.println("x=" + x);
        System.out.println("y=" + y);
    }
}
```

결과	x=3 y=3
----	------------

증감 연산자는 피연산자에 저장된 값을 1 증가 또는 감소시킨다. 증감 연산자의 피연산자로 정수와 실수가 모두 가능하지만, 상수는 값을 변경할 수 없으므로 가능하지 않다.

참고

증감 연산자는 일반 산술 변환에 의한 자동 형변환이 발생하지 않으며, 연산결과의 타입은 피연산자의 타입과 같다.

증가 연산자(++) 피연산자의 값을 1 증가시킨다.

감소 연산자(--) 피연산자의 값을 1 감소시킨다.

일반적으로 단항 연산자는 피연산자의 왼쪽에 위치하지만, 증가 연산자 ‘++’와 감소 연산자 ‘--’는 양쪽 모두 가능하다. 피연산자의 왼쪽에 위치하면 ‘전위형(prefix)’, 오른쪽에 위치하면 ‘후위형(postfix)’이라고 한다.

타입	설명	사용예
전위형	값이 참조되기 전에 증가시킨다.	j = ++i;
후위형	값이 참조된 후에 증가시킨다.	j = i++;

그러나 ‘++i;’와 ‘i++;’처럼 증감연산자가 수식이나 메서드 호출에 포함되지 않고 독립적인 하나의 문장으로 쓰인 경우에는 전위형과 후위형의 차이가 없다.

```
++i; // 전위형. i의 값을 1 증가시킨다.  
i++; // 후위형. 위의 문장과 차이가 없다.
```

예제

3-2 class Ex3_2 {

```
    public static void main(String args[]) {  
        int i=5, j=0;  
  
        j = i++;  
        System.out.println("j=i++; 실행 후, i=" + i + ", j=" + j);
```

```
        i=5;          // 결과를 비교하기 위해, i와 j의 값을 다시 5와 0으로 변경  
        j=0;
```

```
        j = ++i;  
        System.out.println("j=++i; 실행 후, i=" + i + ", j=" + j);
```

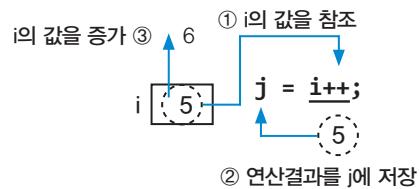
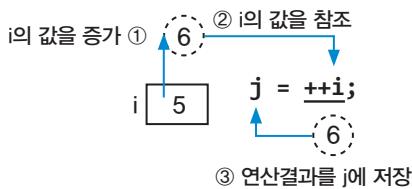
}

결과
j=i++; 실행 후, i=6, j=5
j=++i; 실행 후, i=6, j=6

미리보기용 pdf입니다.

실행결과를 보면 i의 값은 두 경우 모두 1이 증가되어 6이 되지만, j의 값은 그렇지 않다.

식을 계산하기 위해서는 식에 포함된 변수의 값을 읽어 와야 하는데, 전위형은 변수(피연산자)의 값을 먼저 증가시킨 후에 변수의 값을 읽어오는 반면, 후위형은 변수의 값을 먼저 읽어온 후에 값을 증가시킨다.



증감 연산자가 포함된 식을 이해하기 어려울 때는 다음과 같이 증감 연산자를 따로 빼어내면 이해하기가 쉬워진다. 전위형의 경우 증감 연산자를 식의 이전으로,

`j = ++i; // 전위형`

`++i; // 증가 후에
j = i; // 참조하여 대입`

후위형의 경우 증감 연산자를 식의 이후로 빼어내면 된다.

`j = i++; // 후위형`

`j = i; // 참조하여 대입 후에
i++; // 증가`

다음은 메서드 호출에 증감 연산자가 사용된 예이다.

예제
3-3

```
class Ex3_3 {  
    public static void main(String args[]) {  
        int i=5, j=5;  
        System.out.println(i++); // i의 값을 출력 후, 1 증가  
        System.out.println(++j); // j의 값을 1 증가 후, 출력  
        System.out.println("i = " + i + ", j = " +j);  
    }  
}
```

결과
5
6
i = 6, j = 6

`System.out.println(i++);
System.out.println(++j);`

`System.out.println(i);
i++;
++j;
System.out.println(j);`

부호 연산자‘-’는 피연산자의 부호를 반대로 변경한 결과를 반환한다. 피연산자가 음수면 양수, 양수면 음수가 연산의 결과가 된다. 부호 연산자‘+’는 하는 일이 없으며, 쓰이는 경우도 거의 없다. 부호 연산자‘-’가 있으니까 형식적으로 ‘+’를 추가해 놓은 것뿐이다.

부호 연산자는 boolean형과 char형을 제외한 기본형에만 사용할 수 있다.

참고

부호 연산자는 덧셈, 뺄셈 연산자와 같은 기호를 쓰지만 다른 연산자이다. 기호는 같아도 피연산자의 개수가 달라서 구별이 가능하다.

예제

3-4

```
class Ex3_4 {
    public static void main(String[] args) {
        int i = -10;
        i = +i;
        System.out.println(i);

        i = -10;
        i = -i;
        System.out.println(i);
    }
}
```

결과	-10 10
----	-----------

미리보기용 pdf입니다.

프로그램을 작성하다 보면 같은 타입뿐만 아니라 서로 다른 타입 간의 연산을 수행해야 하는 경우도 있다. 이럴 때는 연산을 수행하기 전에 타입을 일치시켜야 하는데, 변수나 리터럴의 타입을 다른 타입으로 변환하는 것을 ‘형변환(casting)’이라고 한다.

형변환이란, 변수 또는 상수의 타입을 다른 타입으로 변환하는 것

형변환 방법은 아주 간단하다. 형변환하고자 하는 변수나 리터럴의 앞에 변환하고자 하는 타입을 괄호와 함께 붙여주기만 하면 된다.

(타입) 피연산자

여기에서 사용되는 괄호()는 ‘캐스트 연산자’ 또는 ‘형변환 연산자’라고 하며, 형변환을 ‘캐스팅(casting)’이라고도 한다. 예를 들어 다음과 같은 코드가 있을 때,

```
double d = 85.4;
int score = (int)d;
```

두 번째 줄의 연산과정을 단계별로 살펴보면 다음과 같다.

```
int score = (int)d; → int score = (int)85.4; → int score = 85;
```

이 과정에서 알 수 있듯이, 형변환 연산자는 그저 피연산자의 값을 읽어서 지정된 타입으로 형변환하고 그 결과를 반환할 뿐이다. 그래서 피연산자인 변수 d의 값은 형변환 후에도 아무런 변화가 없다.

변환	수식	결과
int → char	(char)65	'A'
char → int	(int)'A'	65
float → int	(int)1.6f	1
int → float	(float)10	10.0f

▲ 표 3-1 형변환의 다양한 예시

예제
3-5

```
class Ex3_5 {
    public static void main(String[] args) {
        double d = 85.4;
        int score = (int) d;
        System.out.println("score=" + score);
        System.out.println("d=" + d);
    }
}
```

결과
score=85
d=85.4 ← 형변환 후에도 피연산자에는 아무런 변화가 없다.

서로 다른 타입 간의 대입이나 연산을 할 때, 먼저 형변환으로 타입을 일치시키는 것이 원칙이다. 하지만, 경우에 따라 편의상의 이유로 형변환을 생략할 수 있다. 그렇다고 해서 형변환이 이루어지지 않는 것은 아니고, 컴파일러가 생략된 형변환을 자동적으로 추가해준다.

```
float f = 1234; // float f = (float)1234;에서 (float)가 생략됨
```

위의 문장에서 우변은 int타입의 상수이고, 이 값을 저장하려는 변수의 타입은 float이다. 서로 타입이 달라서 형변환이 필요하지만 편의상 생략하였다. float타입의 변수는 1234라는 값을 저장하는데 아무런 문제가 없기 때문이다.

그러나 다음과 같이 변수가 저장할 수 있는 값의 범위보다 더 큰 값을 저장하려는 경우에 형변환을 생략하면 에러가 발생한다.

```
byte b = 1000; // 예리. byte타입의 범위(-128 ~ 127)를 벗어난 값의 대입
```

에러 메시지는 ‘incompatible types: possible lossy conversion from int to byte’인데, 앞서 배운 것과 같이 큰 타입에서 작은 타입으로의 형변환은 값 손실이 발생할 수 있다는 뜻이다.

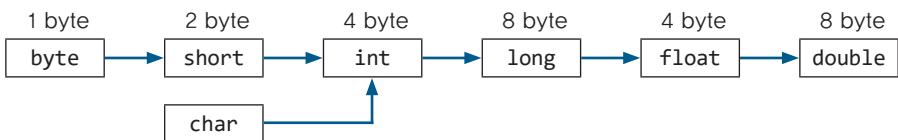
그러나 다음과 같이 명시적으로 형변환 해줬을 경우, 형변환이 프로그래머의 실수가 아닌 의도적인 것으로 간주하고 컴파일러는 에러를 발생시키지 않는다.

```
byte b = (byte)1000; // OK. 그러나 값 손실이 발생해서 변수 b에는 -24가 저장됨.
```

형변환을 하는 이유는 주로 서로 다른 두 타입을 일치시키기 위해서인데, 형변환을 생략하면 컴파일러가 알아서 자동적으로 형변환을 한다.

“기존의 값을 최대한 보존할 수 있는 타입으로 자동 형변환된다.”

그래서 표현범위가 좁은 타입에서 넓은 타입으로 형변환하는 경우에는 값 손실이 없으므로 두 타입 중에서 표현범위가 더 넓은 쪽으로 형변환된다.



위 그림은 형변환이 가능한 7개의 기본형을 왼쪽부터 오른쪽으로 표현할 수 있는 값의 범위가 작은 것부터 큰 것의 순서로 나열한 것이다.

화살표 방향으로의 변환, 즉 왼쪽에서 오른쪽으로의 변환은 형변환 연산자를 사용하지 않아도 자동 형변환이 되며, 그 반대 방향으로의 변환은 반드시 형변환 연산자를 써야 한다.

미리보기용 pdf입니다!

사칙 연산자, 덧셈(+), 뺄셈(-), 곱셈(*), 나눗셈(/)은 아마도 프로그래밍에 가장 많이 사용되는 연산자들 일 것이다. 여러분들이 이미 알고 있는 것처럼 곱셈(*), 나눗셈(/), 나머지(%) 연산자가 덧셈(+) 뺄셈(-) 연산자보다 우선순위가 높으므로 먼저 처리된다.

그리고 피연산자가 정수형인 경우, 나누는 수로 0을 사용할 수 없다. 만일 0으로 나눈다면, 실행 시에 에러가 발생할 것이다.

예제

3-6

```
class Ex3_6 {
    public static void main(String args[]) {
        int a = 10;
        int b = 4;

        System.out.printf("%d + %d = %d\n", a, b, a + b);
        System.out.printf("%d - %d = %d\n", a, b, a - b);
        System.out.printf("%d * %d = %d\n", a, b, a * b);
        System.out.printf("%d / %d = %d\n", a, b, a / b);
        System.out.printf("%d / %f = %f\n", a, (float)b, a / (float)b);
    }
}
```

결과

10 + 4 = 14
10 - 4 = 6
10 * 4 = 40
10 / 4 = 2
10 / 4.000000 = 2.500000

두 변수 a와 b에 각각 10과 4를 저장하여 사칙연산을 수행하고 그 결과를 출력하는 예제이다. 한 가지 눈여겨볼 것은 10을 4로 나눈 결과가 2.5가 아닌 2라는 것이다.

int int int
10 / 4 → 2 // 소수점 이하는 버려진다.

나누기 연산자의 두 피연산자가 모두 int타입인 경우, 연산결과 역시 int타입이다. 그래서 실제 연산결과는 2.5일지라도 int타입의 값인 2를 결과로 얻는다. int타입은 소수점을 저장하지 못하므로 정수만 남고 소수점 이하는 버려지기 때문이다. 이 때, 반올림이 발생하지 않는다는 점에 주의하자.

그래서 올바른 연산결과를 얻기 위해서는 두 피연산자 중 어느 한 쪽을 실수형으로 형변환해야 한다. 그래야만 다른 한 쪽도 같이 실수형으로 자동 형변환되어 결국 실수형의 값을 결과로 얻는다.

int float float float float
10 / 4.0f → 10.0f / 4.0f → 2.5f

위의 연산과정을 보면, 두 피연산자의 타입이 일치하지 않으므로 int타입보다 범위가 넓은 float타입으로 일치시킨 후에 연산을 수행하는 것을 알 수 있다. 이제 float타입과 float타입의 연산이므로 연산결과 역시 float타입이다.

www.codechobo.com

이항 연산자는 두 피연산자의 타입이 일치해야 연산이 가능하므로, 피연산자의 타입이 서로 다르다면 연산 전에 형변환 연산자로 타입을 일치시켜야 한다.

이처럼 연산 전에 피연산자 타입의 일치를 위해 자동 형변환되는 것을 ‘산술 변환’ 또는 ‘일반 산술 변환’이라 하며, 이 변환은 이항 연산에서만 아니라 단항 연산에서도 일어난다. ‘산술 변환’의 규칙은 다음과 같다.

- ① 두 피연산자의 타입을 같게 일치시킨다.(보다 큰 타입으로 일치)

```
long + int → long + long → long
float + int → float + float → float
double + float → double + double → double
```

- ② 피연산자의 타입이 int보다 작은 타입이면 int로 변환된다.

```
byte + short → int + int → int
char + short → int + int → int
```

첫 번째 규칙은 앞서 자동 형변환에서 배운 것처럼 피연산자의 값손실을 최소화하기 위한 것이고, 두 번째 규칙은 int보다 작은 타입, 예를 들면 char나 short의 표현범위가 좁아서 연산 중에 오버플로우(overflow)가 발생할 가능성이 높기 때문에 있는 것이다.

여기서 한 가지 주목해야 할 점은 연산결과의 타입인데, 연산결과의 타입은 피연산자의 타입과 일치한다. 예를 들어 int와 int의 나눗셈 연산결과는 int이다.

```
int / int → int
5 / 2 → 2
```

그래서 아래의 식 ‘5 나누기 2’의 결과가 2.5가 아닌 2이다. 위의 식에서 2.5라는 실수를 결과로 얻으려면, 피연산자 중 어느 한 쪽을 float와 같은 실수형으로 형변환해야 한다. 그러면, 다른 한 쪽은 일반 산술 변환의 첫 번째 규칙에 의해 자동적으로 형변환되어 두 피연산자 모두 실수형이 되고, 연산결과 역시 실수형의 값을 얻을 수 있다.

```
int / (float)int → int / float → float / float → float
5 / (float)2 → 5 / 2.0f → 5.0f / 2.0f → 2.5f
```

결국 산술 변환이란, 그저 연산 직전에 발생하는 자동 형변환일 뿐이다. 어렵게 생각하지 말자.

예제
3-7

```
class Ex3_7 {
    public static void main(String[] args) {
        System.out.println(5/2);
        System.out.println(5/(float)2); // (float)5/2의 결과도 동일
    }
}
```

결과	2
	2.5

마지막으로 java입니다.

아래의 코드를 컴파일하면 에러가 발생한다.

```
byte a = 10;  
byte b = 20;  
byte c = a + b; •—————  
System.out.println(c);
```

컴파일 에러가 발생한다.
명시적으로 형변환이 필요하다.

a와 b는 모두 int형보다 작은 byte형이기 때문에 연산자 '+'는 이 두 개의 피연산자들의 자료형을 int형으로 변환한 다음 연산(덧셈)을 수행한다.

그래서 'a + b'의 연산결과는 byte형이 아닌 int형(4 byte)인 것이다. 4 byte의 값을 1 byte의 변수에 형변환없이 저장하려고 했기 때문에 에러가 발생하는 것이다.

크기가 작은 자료형의 변수를 큰 자료형의 변수에 저장할 때는 자동으로 형변환(type conversion, casting)되지만, 반대로 큰 자료형의 값을 작은 자료형의 변수에 저장하려면 명시적으로 형변환 연산자를 사용해서 변환해주어야 한다.

위의 코드에서 3번째 줄 'byte c = a + b;'를 'byte c = (byte)(a + b);'와 같이 변경해야 컴파일에러가 발생하지 않는다.

예제
3-8

```
class Ex3_8 {  
    public static void main(String[] args) {  
        byte a = 10;  
        byte b = 30;  
        byte c = (byte)(a * b);  
        System.out.println(c);  
    }  
}
```

결과 44

이 예제를 실행하면 44가 화면에 출력된다. '10 * 30'의 결과는 300이지만, 형변환(캐스팅, casting)에서 배운 것처럼, 큰 자료형에서 작은 자료형으로 변환하면 데이터의 손실이 발생하므로 값이 바뀔 수 있다. 300은 byte형의 범위를 넘기 때문에 byte형으로 변환하면 데이터 손실이 발생하여 결국 44가 byte형 변수 c에 저장된다.

변환	2진수	10진수	값손실
byte ↓ int	 0000000000000000000000000000001010	10 10	없음
int ↓ byte	 000000000000000000000000100101100	300 44	있음

예제

3-9

```
class Ex3_9 {
    public static void main(String args[]) {
        int a = 1_000_000;      // 1,000,000  1백만
        int b = 2_000_000;      // 2,000,000  2백만

        long c = a * b;        // a * b = 2,000,000,000,000 ?

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

결과 -1454759936

식 ‘ $a * b$ ’의 결과 값을 담는 변수 c 의 자료형이 long타입(8 byte)이기 때문에 2×10^{12} 을 저장하기에 충분하므로 ‘2000000000000’이 출력될 것 같지만, 결과는 전혀 다른 값이 출력된다. 그 이유는 int타입과 int타입의 연산결과는 int타입이기 때문이다. ‘ $a * b$ ’의 결과가 이미 int 타입의 값(-1454759936)이므로 long형으로 자동 형변환되어도 같은 변하지 않는다.

```
long c = a * b;
→ long c = 1000000 * 2000000;
→ long c = -1454759936;
```

올바른 결과를 얻으려면 아래와 같이 변수 a 또는 b 의 타입을 ‘long’으로 형변환해야 한다.

```
long c = (long)a * b;
→ long c = (long)1000000 * 2000000;
→ long c = 1000000L * 2000000;
→ long c = 1000000L * 2000000L;
→ long c = 2000000000000L;
```

int int int
1000000 * 1000000 → -727379968 오버플로우 발생!!!

int long long long long
1000000 * 1000000L → 1000000L * 1000000L → 1000000000000L

예제

3-10

```
class Ex3_10 {
    public static void main(String args[]) {
        long a = 1_000_000 * 1_000_000;
        long b = 1_000_000 * 1_000_000L;

        System.out.println("a="+a);
        System.out.println("b="+b);
    }
}
```

결과 a=-727379968
b=1000000000000

반올림을 하려면 Math.round()를 사용하면 된다. 이 메서드는 소수점 첫째 자리에서 반올림한 결과를 정수로 반환한다.

```
long result = Math.round(4.52); // result에 5가 저장된다.
```

만일 소수점 첫째 자리가 아닌 다른 자리에서 반올림을 하려면 10의 n제곱으로 적절히 곱하고 나누어야 한다.

예제

3-11

```
class Ex3_11 {
    public static void main(String args[]) {
        double pi = 3.141592;
        double shortPi = Math.round(pi * 1000) / 1000.0;
        System.out.println(shortPi);
    }
}
```

결과 3.142

이 예제의 결과는 pi의 값을 소수점 넷째 자리인 5에서 반올림을 해서 3.142가 출력되었다. round메서드는 매개변수로 받은 값을 소수점 첫째 자리에서 반올림을 하니까 Math.round(3141.592)의 결과는 3142이다.

```
Math.round(pi * 1000) / 1000.0
→ Math.round(3.141592 * 1000) / 1000.0
→ Math.round(3141.592) / 1000.0
→ 3142 / 1000.0
→ 3.142
```

위의 과정에서 1000.0이 아니라 1000으로 나누었으면 결과는 3.142가 아닌 3이 된다. 그 이유는 int와 int의 나눗셈 결과는 int이기 때문이다.

int	int	int
3142	/ 1000	→ 3

int	double	double	double	double
3142	/ 1000.0	→ 3142.0	/ 1000.0	→ 3.142

12 나머지 연산자

나머지 연산자는 왼쪽의 피연산자를 오른쪽 피연산자로 나누고 난 나머지 값을 결과로 반환한다. 나눗셈에서처럼 나누는 수(오른쪽 피연산자)로 0을 사용할 수 없고, 피연산자로 정수만 허용한다. 나머지 연산자는 주로 짹수, 홀수 또는 배수 검사 등에 주로 사용된다.

예제
3-12

```
class Ex3_12 {
    public static void main(String args[]) {
        int x = 10;
        int y = 8;

        System.out.printf("%d를 %d로 나누면, %n", x, y);
        System.out.printf("몫은 %d이고, 나머지는 %d입니다.%n", x / y, x % y);
    }
}
```

결과 10을 8로 나누면,
몫은 1이고, 나머지는 2입니다.

나눗셈 연산자와 나머지 연산자를 이용해서 몫과 나머지를 구하는 예제이다. 간단한 예제라서 따로 설명하지 않아도 이해하는데 어려움이 없을 것이다.

예제
3-13

```
class Ex3_13 {
    public static void main(String[] args) {
        System.out.println(-10%8);
        System.out.println(10%-8);
        System.out.println(-10%-8);
    }
}
```

결과 -2
2
-2

나머지 연산자(%)는 나누는 수로 음수도 허용한다. 그러나 부호는 무시되므로 결과는 음수의 절대값으로 나눈 나머지와 결과가 같다.

```
System.out.println(10 % 8); // 10을 8로 나눈 나머지 2가 출력된다.
System.out.println(10 %-8); // 위와 같은 결과를 얻는다.
```

그냥 피연산자의 부호를 모두 무시하고, 나머지 연산을 한 결과에 왼쪽 피연산자(나눠지는 수)의 부호를 붙이면 된다.

미리보기용 pdf입니다.

비교 연산자는 두 피연산자를 비교하는 데 사용되는 연산자다. 주로 조건문과 반복문의 조건식에 사용되며, 연산결과는 오직 true와 false 둘 중의 하나이다.

비교 연산자 역시 이항 연산자이므로 비교하는 피연산자의 타입이 서로 다를 경우에는 자료형의 범위가 큰 쪽으로 자동 형변환하여 피연산자의 타입을 일치시킨 후에 비교한다는 점에 주의하자.

대소비교 연산자 < > <= >=

두 피연산자의 값의 크기를 비교하는 연산자이다. 참이면 true를, 거짓이면 false를 결과로 반환한다. 기본형 중에서는 boolean을 제외한 나머지 자료형에 다 사용할 수 있지만 참조형에는 사용할 수 없다.

비교연산자	연산결과
>	좌변 값이 크면 , true 아니면 false
<	좌변 값이 작으면 , true 아니면 false
>=	좌변 값이 크거나 같으면 , true 아니면 false
<=	좌변 값이 작거나 같으면 , true 아니면 false

▲ 표 3-2 대소비교 연산자의 종류와 연산결과

등가비교 연산자 == !=

두 피연산자의 값이 같은지 또는 다른지를 비교하는 연산자이다. 대소비교 연산자(<, >, <=, >=)와는 달리, 모든 자료형(기본형, 참조형)에 사용할 수 있다. 기본형의 경우 변수에 저장되어 있는 값이 같은지를 알 수 있고, 참조형의 경우 객체의 주소값을 저장하기 때문에 두 개의 피연산자(참조변수)가 같은 객체를 가리키고 있는지(주소값이 같은지)를 알 수 있다.

기본형과 참조형은 서로 형변환이 가능하지 않기 때문에 등가비교 연산자(==, !=)로 기본형과 참조형을 비교할 수는 없다.

비교연산자	연산결과
==	두 값이 같으면 , true 아니면 false
!=	두 값이 다르면 , true 아니면 false

▲ 표 3-3 등가비교 연산자의 종류와 연산결과

비교 연산자는 수학기호와 유사한 기호와 의미를 가지고 있으므로 이해하는데 별 어려움이 없을 것이다. 한 가지 다른 점은 ‘두 값이 같다’는 의미로 ‘=’가 아닌 ‘==’를 사용한다는 것인데, ‘=’는 이미 배운 것과 같이 변수에 값을 저장할 때 사용하는 ‘대입 연산자’이기 때문에 ‘==’로 두 값이 같은지 비교하는 연산자를 표현한다.

주의

‘==’와 같이 두 개의 기호로 이루어진 연산자는 ‘=>’와 같이 기호의 순서를 바꾸거나 ‘>=’와 같이 중간에 공백이 들어가서는 안 된다.

14 문자열의 비교

두 문자열을 비교할 때는, 비교 연산자 ‘==’ 대신 equals()라는 메서드를 사용해야 한다. 비교 연산자는 두 문자열이 완전히 같은 것인지 비교할 뿐이므로, 문자열의 내용이 같은지 비교하기 위해서는 equals()를 사용하는 것이다. equals()는 비교하는 두 문자열이 같으면 true를, 다르면 false를 반환한다.

```
String str = new String("abc");

// equals()는 두 문자열의 내용이 같으면 true, 다르면 false를 결과로 반환
boolean result = str.equals("abc"); // 내용이 같으므로 result에 true가 저장됨
```

원래 String은 클래스이므로, 아래와 같이 new를 사용해서 객체를 생성해야 한다.

```
String str = new String("abc"); // String클래스의 객체를 생성
String str = "abc";           // 위의 문장을 간단히 표현
```

그러나 특별히 String만 new를 사용하지 않고, 위와 같이 간단히 쓸 수 있게 허용한다.

예제
3-14

```
class Ex3_14 {
    public static void main(String[] args) {
        String str1 = "abc";
        String str2 = new String("abc");
        System.out.printf("\"abc\" == \"abc\" ? %b%n", "abc" == "abc");
        System.out.printf("str1 == \"abc\" ? %b%n", str1 == "abc");
        System.out.printf("str2 == \"abc\" ? %b%n", str2 == "abc");
        System.out.printf("str1.equals(\"abc\") ? %b%n", str1.equals("abc"));
        System.out.printf("str2.equals(\"abc\") ? %b%n", str2.equals("abc"));
        System.out.printf("str2.equals(\"ABC\") ? %b%n", str2.equals("ABC"));
        System.out.printf("str2.equalsIgnoreCase(\"ABC\") ? %b%n",
                         str2.equalsIgnoreCase("ABC"));
    }
}
```

결과	"abc" == "abc" ? true str1 == "abc" ? true str2 == "abc" ? false str1.equals("abc") ? true str2.equals("abc") ? true str2.equals("ABC") ? false str2.equalsIgnoreCase("ABC") ? true
----	---

str2와 “abc”의 내용이 같은데도 ‘==’로 비교하면, false를 결과로 얻는다. 내용은 같지만 서로 다른 객체라서 그렇다. 그러나 equals()는 객체가 달라도 내용이 같으면 true를 반환한다. 그래서 문자열을 비교할 때는 항상 equals()를 사용해야 한다는 것을 기억하자.

만일 대소문자를 구별하지 않고 비교하고 싶으면, equals() 대신 equalsIgnoreCase()를 사용하면 된다.

미리보기용 pdf입니다.

'x가 4보다 작다'라는 조건은 비교 연산자를 써서 ' $x < 4$ '와 같이 표현할 수 있다. 그러면, x가 4보다 작거나 또는 10보다 크다'와 같이 두 개의 조건이 결합된 경우는 어떻게 표현해야 할까? 이 때 사용하는 것이 '논리 연산자'이다. 논리 연산자는 둘 이상의 조건을 '그리고(AND)'나 '또는(OR)'으로 연결하여 하나의 식으로 표현할 수 있게 해준다.

논리 연산자 '&&'는 우리말로 '그리고(AND)'에 해당하며, 두 피연산자가 모두 true일 때만 true를 결과로 얻는다. '||'는 '또는(OR)'에 해당하며, 두 피연산자 중 어느 한 쪽만 true이어도 true를 결과로 얻는다. 그리고 논리 연산자는 피연산자로 boolean형 또는 boolean형 값을 결과로 하는 조건식만을 허용한다.

|| (OR결합) 피연산자 중 어느 한 쪽이 true이면 true를 결과로 얻는다.

&& (AND결합) 피연산자 양쪽 모두 true이어야 true를 결과로 얻는다.

논리 연산자의 피연산자가 '참(true)인 경우'와 '거짓(false)인 경우'의 연산결과를 표로 나타내면 다음과 같다.

x	y	$x \parallel y$	$x \&\& y$
true	true	true	true
true	false	true	false
false	true	true	false
false	false	false	false

이제 자주 사용될만한 몇 가지 예를 통해서 논리 연산자가 실제로 어떻게 사용되고, 주의해야 할 점은 어떤 것들이 있는지 살펴보자.

① x는 10보다 크고, 20보다 작다.

' $x > 10$ '와 ' $x < 20$ '가 '그리고(and)'로 연결된 조건이므로 다음과 같이 쓸 수 있다.

$x > 10 \&\& x < 20$

' $x > 10$ '는 ' $10 < x$ '와 같으므로 다음과 같이 쓸 수도 있다. 보통은 변수를 왼쪽에 쓰지만 이런 경우 가독성측면에서 보면 아래의 식이 더 나을 수 있다.

$10 < x \&\& x < 20$

그렇다고 해서 위의 식에서 논리연산자를 생략하고 ' $10 < x < 20$ '과 같이 표현하는 것은 허용되지 않는다.

www.codechobo.com

② i는 2의 배수 또는 3의 배수이다.

어떤 수가 2의 배수라는 얘기는 2로 나누었을 때 나머지가 0이라는 뜻이다. 그래서 나머지 연산의 결과가 0인지 확인하면 된다. ‘또는’으로 두 조건이 연결되었으므로 논리 연산자 ‘||’ (OR)를 사용해야 한다.

```
i%2==0 || i%3==0
```

i의 값이 8일 때, 위의 식은 다음과 같은 과정으로 연산된다.

```
i%2==0 || i%3==0  
→ 8%2==0 || 8%3==0  
→ 0==0 || 2==0  
→ true || false  
→ true
```

③ i는 2의 배수 또는 3의 배수지만 6의 배수는 아니다.

이전 조건에 6의 배수를 제외하는 조건이 더 붙었다. 6의 배수가 아니어야 한다는 조건은 ‘i%6!=0’이고, 이 조건을 ‘&&(AND)’로 연결해야 한다.

```
( i%2==0 || i%3==0 ) && i%6!=0
```

위의 식에 괄호를 사용한 이유는 ‘&&’가 ‘||’보다 우선순위가 높기 때문이다. 만일 괄호를 사용하지 않으면 ‘&&’를 먼저 연산한다. 다음의 두 식은 동일하다.

```
i%2==0 || i%3==0 && i%6!=0  
i%2==0 || (i%3==0 && i%6!=0)
```

이처럼 하나의 식에 ‘&&’와 ‘||’가 같이 포함된 경우, ‘&&’가 먼저 연산되어야 하는 경우라도 괄호를 사용해서 우선순위를 명확히 해주는 것이 좋다.

④ 문자 ch는 숫자('0'~'9')이다.

사용자로부터 입력된 문자가 숫자('0'~'9')인지 확인하는 식은 다음과 같이 쓸 수 있다.

```
'0' <= ch && ch <= '9'
```

유니코드에서 문자 ‘0’부터 ‘9’까지 연속적으로 배치되어 있기 때문에 가능한 식이다. 문자 ‘0’부터 ‘9’까지 유니코드는 10진수로 다음과 같다.

문자	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
문자코드	48	49	50	51	52	53	54	55	56	57

그래서 ch의 값이 '5'인 경우 위의 식은 다음과 같은 과정으로 연산된다.

```
'0' <= ch && ch <= '9'  
→ '0' <= '5' && '5' <= '9'  
→ 48 <= 53 && 53 <= 57  
→ true && true  
→ true
```

⑤ 문자 ch는 대문자 또는 소문자이다.

④의 경우와 마찬가지로 문자 'a'부터 'z'까지, 그리고 'A'부터 'Z'까지도 연속적으로 배치되어 있으므로 문자 ch가 대문자 또는 소문자인지 확인하는 식은 다음과 같이 쓸 수 있다.

```
('a' <= ch && ch <= 'z') || ('A' <= ch && ch <= 'Z')
```

예제

3-15

```
import java.util.Scanner; // Scanner클래스를 사용하기 위해 추가  
  
class Ex3_15 {  
    public static void main(String args[]) {  
        Scanner scanner = new Scanner(System.in);  
        char ch = ' ';  
  
        System.out.printf("문자를 하나 입력하세요.>");  
  
        String input = scanner.nextLine();  
        ch = input.charAt(0);  
  
        if('0' <= ch && ch <= '9') {  
            System.out.printf("입력하신 문자는 숫자입니다.%n");  
        }  
  
        if(('a' <= ch && ch <= 'z') || ('A' <= ch && ch <= 'Z')) {  
            System.out.printf("입력하신 문자는 영문자입니다.%n");  
        }  
    } // main  
}
```

결과 1 문자를 하나 입력하세요.>7
결과 2 문자를 하나 입력하세요.>a
입력하신 문자는 숫자입니다.
입력하신 문자는 영문자입니다.

이 예제는 사용자로부터 하나의 문자를 입력받아서 숫자인지 영문자인지 확인한다. 조건문 if는 팔호() 안의 연산결과가 참인 경우 블럭{} 내의 문장을 수행한다. 그래서 아래의 코드는 '0' <= ch && ch <= '9'가 참일 때, 화면에 '입력하신 문자는 숫자입니다.'라고 출력한다.

```
if('0' <= ch && ch <= '9') {  
    System.out.printf("입력하신 문자는 숫자입니다.%n");  
}
```

www.codechobo.com

16 논리 부정 연산자

이 연산자는 피연산자가 true이면 false를, false면 true를 결과로 반환한다. 간단히 말해서, true와 false를 반대로 바꾸는 것이다.

x	!x
true	false
false	true

어떤 값에 논리 부정 연산자 ‘!’를 반복적으로 적용하면, 참과 거짓이 차례대로 반복된다. 이 연산자의 이러한 성질을 이용하면, 한번 누르면 켜지고, 다시 한 번 누르면 꺼지는 TV의 전원 버튼과 같은 ‘토글 버튼(toggle button)’을 논리적으로 구현할 수 있다.

`false(거짓, off) → ! true(참, on) → ! false(거짓, off) → ! true(참, on) → ...`

논리 부정 연산자 ‘!’가 주로 사용되는 곳은 조건문과 반복문의 조건식이며, 이 연산자를 잘 사용하면 조건식이 보다 이해하기 쉬워진다. 예를 들어 ‘문자 ch는 소문자가 아니다’라는 조건을 아래의 왼쪽과 같이 쓰기보다 오른쪽과 같이 논리 부정 연산자 ‘!’를 사용하는 쪽이 알기 쉽다.

`ch < 'a' || ch > 'z'`



`!('a' <= ch && ch <= 'z')`

위와 같이 논리 부정 연산자 ‘!’를 적절히 사용해서 보다 이해하기 쉬운 식이 되도록 노력하자.

예제
3-16

```
class Ex3_16 {
    public static void main(String[] args) {
        boolean b = true;
        char ch = 'C';

        System.out.printf("b=%b%n", b);
        System.out.printf("!b=%b%n", !b);
        System.out.printf("!!b=%b%n", !!b);
        System.out.printf("!!!b=%b%n", !!!b);
        System.out.println();

        System.out.printf("ch=%c%n", ch);
        System.out.printf("ch < 'a' || ch > 'z'=%b%n", ch < 'a' || ch > 'z');
        System.out.printf("!(a'<=ch && ch<='z')=%b%n", !(a'<= ch && ch<='z'));
        System.out.printf(" 'a'<=ch && ch<='z'=%b%n", 'a'<=ch && ch<='z');
    } // main의 끝
}
```

결과
b=true
!b=false
!!b=true
!!!b=false

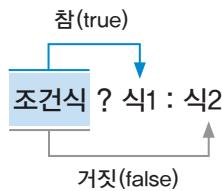
ch=C
ch < 'a' || ch > 'z'=true
!(a'<=ch && ch<='z')=true
'a'<=ch && ch<='z' =false

식 ‘!!b’가 평가되는 과정은 아래와 같다. 단항 연산자는 결합 방향이 오른쪽에서 왼쪽이므로 피연산자와 가까운 것부터 먼저 연산된다.

`!!b → !true → !false → true`

미리보기용 pdf입니다.

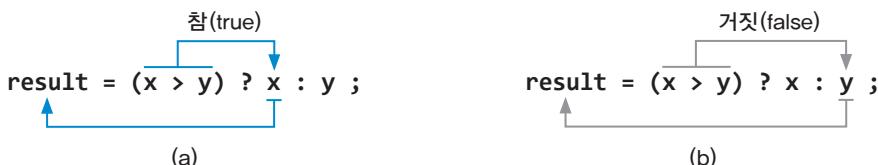
조건 연산자는 조건식, 식1, 식2 모두 세 개의 피연산자를 필요로 하는 삼항 연산자이며, 삼항 연산자는 조건 연산자 하나뿐이다.



조건 연산자는 첫 번째 피연산자인 조건식의 평가결과에 따라 다른 결과를 반환한다. 조건식의 평가결과가 true이면 식1이, false이면 식2가 연산결과가 된다. 가독성을 높이기 위해 조건식을 괄호()로 둘러싸는 경우가 많지만 필수는 아니다.

```
result = (x > y) ? x : y ; // 괄호 생략 가능
```

위의 문장에서 식 ' $x > y$ '의 결과가 true이면, 변수 result에는 x의 값이 저장되고, false이면 y의 값이 저장된다.



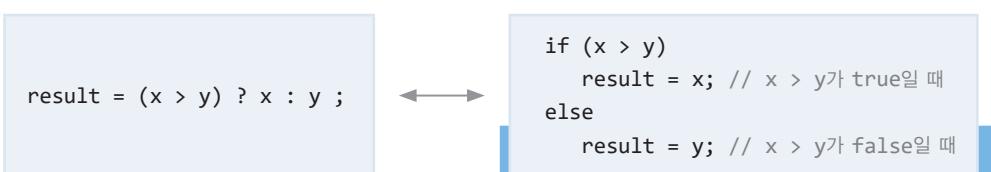
만일 x의 값이 5, y의 값이 3이라면, 이 식은 다음과 같은 과정으로 계산된다.

```

result = (x > y) ? x : y ;
→ result = (5 > 3) ? 5 : 3 ;
→ result = (true) ? 5 : 3 ;     조건식이 true(참)이므로 연산결과는 5
→ result = 5;

```

조건 연산자는 조건문인 if문으로 바꿔 쓸 수 있으며, if문 대신 조건 연산자를 사용하면 코드를 보다 간단히 할 수 있다. 아래 왼쪽의 조건 연산자가 쓰인 문장을 if문으로 바꾸면 오른쪽과 같다.



그리고 조건 연산자의 식1과 식2, 이 두 피연산자의 타입이 다른 경우, 이항 연산자처럼 산술 변환이 발생한다.

```
x = x + (mod < 0.5 ? 0 : 0.5) 0과 0.5의 타입이 다르다.  
→ x = x + (mod < 0.5 ? 0.0 : 0.5) 0이 0.0으로 자동 형변환되었다.
```

위의 식에서 조건 연산자의 피연산자 0과 0.5의 타입이 다르므로, 자동 형변환이 일어나서 double타입으로 통일되고 연산결과 역시 double타입이 된다.

예제

3-17

```
class Ex3_17 {  
    public static void main(String args[]) {  
        int x, y, z;  
        int absX, absY, absZ;  
        char signX, signY, signZ;  
  
        x = 10;  
        y = -5;  
        z = 0;  
  
        absX = x >= 0 ? x : -x; // x의 값이 음수이면, 양수로 만든다.  
        absY = y >= 0 ? y : -y;  
        absZ = z >= 0 ? z : -z;  
        signX = x > 0 ? '+' : (x==0 ? ' ' : '-'); // 조건 연산자를 중첩  
        signY = y > 0 ? '+' : (y==0 ? ' ' : '-');  
        signZ = z > 0 ? '+' : (z==0 ? ' ' : '-');  
  
        System.out.printf("x=%c%d%n", signX, absX);  
        System.out.printf("y=%c%d%n", signY, absY);  
        System.out.printf("z=%c%d%n", signZ, absZ);  
    }  
}
```

결과
x=+10
y=-5
z= 0

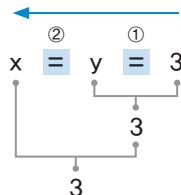
조건 연산자를 이용해서 변수의 절대값을 구한 후, 부호를 붙여 출력하는 예제이다. 간단해서 이해하는데 별 어려움은 없을 것이다.

미리보기용 pdf입니다.

대입 연산자는 변수와 같은 저장공간에 값 또는 수식의 연산결과를 저장하는데 사용된다. 이 연산자는 오른쪽 피연산자의 값(식이라면 평가값)을 왼쪽 피연산자에 저장한다. 그리고 저장된 값을 연산결과로 반환한다. 예를 들어, 아래의 문장은 변수 `x`에 3을 저장하고, 연산결과인 3을 화면에 출력한다.

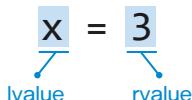
```
System.out.println(x = 3);      변수 x에 3이 저장되고
→ System.out.println(3);        연산결과인 3이 출력된다.
```

대입 연산자는 연산자들 중에서 가장 낮은 우선순위를 가지고 있기 때문에 식에서 제일 나중에 수행된다. 그리고 앞서 배운 것처럼 연산 진행 방향이 오른쪽에서 왼쪽이기 때문에 '`x=y=3;`'에서 '`y=3`'이 먼저 수행되고 그 다음에 '`x=y`'가 수행된다.



lvalue와 rvalue

대입 연산자의 왼쪽 피연산자를 ‘lvalue(left value)’이라 하고, 오른쪽 피연산자를 ‘rvalue(right value)’라고 한다.



대입 연산자의 rvalue는 변수뿐만 아니라 식이나 상수 등이 모두 가능한 반면, lvalue는 반드시 변수처럼 값을 변경할 수 있는 것이어야 한다. 그래서 리터럴이나 상수같이 값을 저장할 수 없는 것들은 lvalue가 될 수 없다.

```
int i = 0;           // 에러. lvalue가 값을 저장할 수 있는 공간이 아니다.
3 = i + 3;          // 에러. lvalue의 연산결과가 리터럴(i+3 → 0+3 → 3)
i + 3 = i;          // 변수 앞에 키워드 final을 붙이면 상수가 된다.

final int MAX = 3;  // 변수 앞에 키워드 final을 붙이면 상수가 된다.
MAX = 10;           // 에러. 상수(MAX)에 새로운 값을 저장할 수 없다.
```

앞서 배운 것과 같이 변수 앞에 키워드 ‘final’을 붙이면 상수가 된다. 상수는 한 번 저장된 값은 바꿀 수 없다.

www.codechobo.com

대입 연산자는 다른 연산자(op)와 결합하여 ‘op=’와 같은 방식으로 사용될 수 있다. 예를 들면, ‘`i = i + 3`’은 ‘`i += 3`’과 같이 표현될 수 있다. 그리고 결합된 두 연산자는 반드시 공백없이 붙여 써야 한다.

op=	=
<code>i += 3;</code>	<code>i = i + 3;</code>
<code>i -= 3;</code>	<code>i = i - 3;</code>
<code>i *= 3;</code>	<code>i = i * 3;</code>
<code>i /= 3;</code>	<code>i = i / 3;</code>
<code>i %= 3;</code>	<code>i = i % 3;</code>
<code>i <= 3;</code>	<code>i = i << 3;</code>
<code>i >= 3;</code>	<code>i = i >> 3;</code>
<code>i &= 3;</code>	<code>i = i & 3;</code>
<code>i ^= 3;</code>	<code>i = i ^ 3;</code>
<code>i = 3;</code>	<code>i = i 3;</code>
<code>i *= 10 + j;</code>	<code>i = i * (10 + j);</code>

위 표의 왼쪽은 복합 연산자의 사용 예이고, 오른쪽은 대입 연산자를 이용한 왼쪽과 동일한 의미의 식이다. 복합 연산자가 잘 익숙해지지 않는다면, 오른쪽과 같은 형태의 식을 사용하다가 점차 왼쪽의 형태로 바꿔가도록 하자.

한 가지 주의할 점은 위 표의 마지막 줄처럼, 대입 연산자의 우변이 둘 이상의 항으로 이루어져 있는 경우이다. ‘`i *= 10 + j;`’를 ‘`i = i * 10 + j;`’와 같은 것으로 오해하지 않도록 하자.

미리보기용 pdf입니다.

연습문제

3-1 다음 중 형변환을 생략할 수 있는 것은? (모두 고르시오)

```
byte b = 10;
char ch = 'A';
int i = 100;
long l = 1000L;
```

- ① `b = (byte)i;`
- ② `ch = (char)b;`
- ③ `short s = (short)ch;`
- ④ `float f = (float)l;`
- ⑤ `i = (int)ch;`

3-2 다음 연산의 결과를 적으시오.

```
class Exercise3_2 {
    public static void main(String[] args) {
        int x = 2;
        int y = 5;
        char c = 'A'; // 'A'의 문자코드는 65

        System.out.println(1 + x << 33);
        System.out.println(y >= 5 || x < 0 && x > 2);
        System.out.println(y += 10 - x++);
        System.out.println(x += 2);
        System.out.println(!(‘A’ <= c && c <= ‘Z’));
        System.out.println(‘C’ - c);
        System.out.println(‘5’ - ‘0’);
        System.out.println(c + 1);
        System.out.println(++c);
        System.out.println(c++);
        System.out.println(c);
    }
}
```

3-3 아래는 변수 num의 값 중에서 백의 자리 이하를 버리는 코드이다. 만일 변수 num의 값이 '456'이라면 '400'이 되고, '111'이라면 '100'이 된다. (1)에 알맞은 코드를 넣으시오.

```
class Exercise3_3 {
    public static void main(String[] args) {
        int num = 456;
        System.out.println( /* (1) */ );
    }
}
```

결과
400

3-4 아래의 코드는 사과를 담는데 필요한 바구니(버켓)의 수를 구하는 코드이다. 만일 사과의 수가 123개이고 하나의 바구니에는 10개의 사과를 담을 수 있다면, 13개의 바구니가 필요할 것이다. (1)에 알맞은 코드를 넣으시오.

```
class Exercise3_4 {
    public static void main(String[] args) {
        int numOfApples = 123; // 사과의 개수
        int sizeOfBucket = 10; // 바구니의 크기(바구니에 담을 수 있는 사과의 개수)
        int numOfBucket = ( /* (1) */ ); // 모든 사과를 담는데 필요한 바구니의 수

        System.out.println("필요한 바구니의 수 :" + numOfBucket);
    }
}
```

결과 필요한 바구니의 수 :13

3-5 아래는 변수 num의 값에 따라 ‘양수’, ‘음수’, ‘0’을 출력하는 코드이다. 삼항 연산자를 이용해서 (1)에 알맞은 코드를 넣으시오.

(Hint) 삼항 연산자를 두 번 사용하라.

```
class Exercise3_5 {
    public static void main(String[] args) {
        int num = 10;
        System.out.println( /* (1) */ );
    }
}
```

결과 양수

3-6 아래는 화씨(Fahrenheit)를 섭씨(Celcius)로 변환하는 코드이다. 변환공식이 ‘ $C = \frac{5}{9} \times (F - 32)$ ’라고 할 때, (1)에 알맞은 코드를 넣으시오. 단, 변환 결과값은 소수점 셋째자리에서 반올림해야 한다.(Math.round())를 사용하지 않고 처리할 것)

```
class Exercise3_6 {
    public static void main(String[] args) {
        int fahrenheit = 100;
        float celcius = ( /* (1) */ );

        System.out.println("Fahrenheit:" + fahrenheit);
        System.out.println("Celcius:" + celcius);
    }
}
```

결과 Fahrenheit:100
Celcius:37.78

C H A P T E R

4

조건문과 반복문

if, switch, for, while statement



지금까지는 코드의 실행 흐름이 무조건 위에서 아래로 한 문장씩 순차적으로 진행되었지만 때로는 조건에 따라 문장을 건너뛰고, 때로는 같은 문장을 반복해서 수행해야 할 때가 있다. 이처럼 프로그램의 흐름(flow)을 바꾸는 역할을 하는 문장들을 ‘제어문(control statement)’이라고 한다. 제어문에는 ‘조건문과 반복문’이 있는데, 조건문은 조건에 따라 다른 문장이 수행되도록 하고, 반복문은 특정 문장들을 반복해서 수행한다.

if문은 가장 기본적인 조건문이며, 다음과 같이 ‘조건식’과 ‘괄호{}’로 이루어져 있다. ‘if’의 뜻이 ‘만일 ~이라면...’이므로 ‘만일(if) 조건식이 참(true)이면 괄호{} 안의 문장들을 수행하라.’라는 의미로 이해하면 된다.

```
if (조건식) {
    // 조건식이 참(true)일 때 수행될 문장들을 적는다.
}
```

만일 다음과 같은 if문이 있을 때, 조건식 ‘score > 60’이 참(true)이면 괄호{} 안의 문장이 수행되어 화면에 “합격입니다.”라고 출력되고 거짓(false)이면, if문 다음의 문장으로 넘어간다.

```
if (score > 60) {
    System.out.println("합격입니다.");
}
```

위 if문의 조건식이 평가되는 과정을 단계별로 살펴보면 다음과 같다. 변수 score의 값을 80 으로 가정하였다.

```
score > 60
→ 80 > 60
→ true           조건식이 참(true)이므로 괄호{} 안의 문장이 실행된다.
```

위 조건식의 결과는 ‘true’이므로 if문 괄호{} 안의 문장이 실행된다. 만일 조건식의 결과가 ‘false’이면, 괄호{} 안의 문장은 수행되지 않을 것이다.

**예제
4-1**

```
class Ex4_1 {
    public static void main(String args[]) {
        int score = 80;

        if (score > 60) {
            System.out.println("합격입니다.");
        }
    }
}
```

결과	합격입니다.
----	--------

if문에 사용되는 조건식은 일반적으로 비교 연산자와 논리 연산자로 구성된다. 이미 연산자를 배울 때 살펴보았지만, 복습도 할겸 기본적인 것들 몇 개를 골라서 표로 정리했다.

조건식	조건식이 참일 조건
<code>90 <= x && x <= 100</code>	정수 x가 90이상 100이하일 때
<code>x < 0 x > 100</code>	정수 x가 0보다 작거나 100보다 클 때
<code>x%3==0 && x%2!=0</code>	정수 x가 3의 배수지만, 2의 배수는 아닐 때
<code>ch=='y' ch=='Y'</code>	문자 ch가 'y' 또는 'Y'일 때
<code>ch==' ' ch=='\t' ch=='\n'</code>	문자 ch가 공백이거나 탭 또는 개행 문자일 때
<code>'A' <= ch && ch <= 'Z'</code>	문자 ch가 대문자일 때
<code>'a' <= ch && ch <= 'z'</code>	문자 ch가 소문자일 때
<code>'0' <= ch && ch <= '9'</code>	문자 ch가 숫자일 때
<code>str.equals("yes")</code>	문자열 str의 내용이 "yes"일 때(대소문자 구분)
<code>str.equalsIgnoreCase("yes")</code>	문자열 str의 내용이 "yes"일 때(대소문자 구분안함)

조건식을 작성할 때 실수하기 쉬운 것이, 등가비교 연산자 '==' 대신 대입 연산자 '='를 사용하는 것이다. 예를 들어 'x가 0일 때 참'인 조건식은 'x==0'인데, 아래와 같이 실수로 'x=0'이라고 적는 경우가 있다.

```
if (x=0) { ... }      x에 0이 저장되고, 결과는 0이 된다.  
→ if (0) { ... }      결과가 true 또는 false가 아니므로 에러가 발생한다.
```

자바에서 조건식의 결과는 반드시 true 또는 false이어야 한다는 것을 잊지 말자.

예제
4-2

```
class Ex4_2 {
    public static void main(String[] args) {
        int x = 0;
        System.out.printf("x=%d 일 때, 참인 것은%n", x);

        if(x==0) System.out.println("x==0");
        if(x!=0) System.out.println("x!=0");
        if(!(x==0)) System.out.println("!(x==0)");
        if(!(x!=0)) System.out.println("!(x!=0)");

        x = 1;
        System.out.printf("x=%d 일 때, 참인 것은%n", x);

        if(x==0) System.out.println("x==0");
        if(x!=0) System.out.println("x!=0");
        if(!(x==0)) System.out.println("!(x==0)");
        if(!(x!=0)) System.out.println("!(x!=0)");
    }
}
```

결과	x=0 일 때, 참인 것은 x==0 !(x!=0) x=1 일 때, 참인 것은 x!=0 !(x==0)
----	--

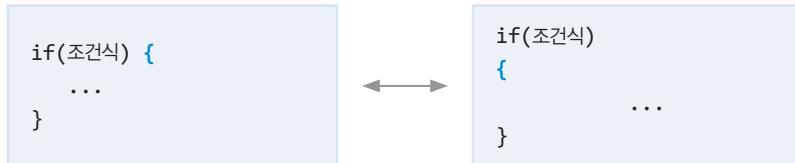
괄호{}를 이용해서 여러 문장을 하나의 단위로 묶을 수 있는데, 이것을 ‘블럭(block)’이라고 한다. 블럭은 ‘{’로 시작해서 ‘}’로 끝나는데, ‘}’ 다음에 문장의 끝을 의미하는 ‘;’을 붙이지 않는다는 것에 주의하자.

블럭 내의 문장들은 탭(tab)으로 들여쓰기(indentation)를 해서 블럭 안에 속한 문장이라는 것을 알기 쉽게 해주는 것이 좋다. 탭(tab)은 키보드의 맨 왼쪽에 있다.

```
if(score > 60)
    블럭의 시작 → {
        System.out.println("합격입니다.");
        블럭의 끝 → }
```

• 탭(tab)에 의한 들여쓰기

블럭의 시작을 의미하는 ‘{’의 위치는 아래와 같이 두 가지 스타일이 있는데, 각 스타일마다 장단점이 있으므로 본인의 취향에 맞는 것으로 선택해서 사용하면 된다. 왼쪽의 스타일은 라인의 수가 짧아진다는 장점이, 오른쪽의 스타일은 블럭의 시작과 끝을 찾기 쉽다는 장점이 있다.



블럭 안에는 보통 여러 문장을 넣지만, 한 문장만 넣거나 아무런 문장도 넣지 않을 수 있다. 만일 블럭 내의 문장이 하나뿐일 때는 아래와 같이 괄호{}를 생략할 수 있다.

```
if(score > 60)
    System.out.println("합격입니다.");
```

또는 아래와 같이 한 줄로 쓸 수도 있다.

```
if(score > 60) System.out.println("합격입니다.");
```

이처럼 블럭 내의 문장이 하나뿐인 경우 괄호{}를 생략할 수 있지만 가능하면 생략하지 않고 사용하는 것이 바람직하다. 나중에 새로운 문장이 추가되면 괄호{}로 문장들을 감싸 주어야 하는데, 이 때 괄호{}를 추가하는 것을 잊기 쉽기 때문이다.

```
if (score > 60)
    System.out.println("합격입니다."); // 문장1 if문에 속한 문장
    System.out.println("축하드립니다."); // 문장2 if문에 속한 문장이 아님
                                            // 미리보기용 pdf입니다.
```

if문의 변형인 if-else문의 구조는 다음과 같다. if문에 ‘else블럭’이 더 추가되었다. ‘else’의 뜻이 ‘그 밖의 다른’이므로 조건식의 결과가 참이 아닐 때, 즉 거짓일 때 else블럭의 문장을 수행하라는 뜻이다.

```
if (조건식) {
    // 조건식이 참(true)일 때 수행될 문장들을 적는다.
} else {
    // 조건식이 거짓(false)일 때 수행될 문장들을 적는다.
}
```

조건식의 결과에 따라 이 두 개의 블럭{} 중 어느 한 블럭{}의 내용이 수행되고 전체 if문을 벗어나게 된다. 두 블럭{}의 내용이 모두 수행되거나, 모두 수행되지 않는 경우는 있을 수 없다. 아래 왼쪽의 두 개의 if문을 if-else문으로 바꾸면 오른쪽과 같다.

<pre>if(input==0) { System.out.println("0입니다."); } if(input!=0) { System.out.println("0이 아닙니다."); }</pre>	<pre>if(input==0) { System.out.println("0입니다."); } else { System.out.println("0이 아닙니다."); }</pre>
--	--

왼쪽 코드의 두 조건식은 어느 한 쪽이 참이면 다른 한 쪽이 거짓인 상반된 관계에 있기 때문에 오른쪽과 같이 if-else문으로 바꿀 수 있는 것이지, 두 개의 if문을 항상 if-else문으로 바꿀 수 있는 것은 아니다.

그리고 왼쪽의 코드는 두 개의 조건식을 계산해야 하지만, if-else문을 사용한 오른쪽의 코드는 하나의 조건식만 계산하면 되므로 더 효율적이고 간단하다.

예제**4-3**

```
import java.util.Scanner; // Scanner클래스를 사용하기 위해 추가

class Ex4_3 {
    public static void main(String[] args) {
        System.out.print("숫자를 하나 입력하세요.>");
        Scanner scanner = new Scanner(System.in);
        int input = scanner.nextInt(); // 화면을 통해 입력받은 숫자를 input에 저장

        if(input==0) {
            System.out.println("입력하신 숫자는 0입니다.");
        } else { // input!=0인 경우
            System.out.println("입력하신 숫자는 0이 아닙니다.");
        }
    } // main의 끝
}
```

결과
1 숫자를 하나 입력하세요.>5
입력하신 숫자는 0이 아닙니다.

결과
2 숫자를 하나 입력하세요.>0
입력하신 숫자는 0입니다.

05 if–else if문

if–else문은 두 가지 경우 중 하나가 수행되는 구조인데, 처리해야 할 경우의 수가 셋 이상인 경우에는 어떻게 해야 할까? 그럴 때는 한 문장에 여러 개의 조건식을 쓸 수 있는 ‘if–else if’ 문을 사용하면 된다.

```
if (조건식1) {
    // 조건식1의 연산결과가 참일 때 수행될 문장들을 적는다.
} else if (조건식2) {
    // 조건식2의 연산결과가 참일 때 수행될 문장들을 적는다.
} else if (조건식3) {           // 여러 개의 else if를 사용할 수 있다.
    // 조건식3의 연산결과가 참일 때 수행될 문장들을 적는다.
} else { // 마지막은 보통 else블럭으로 끝나며, else블럭은 생략가능하다.
    // 위의 어느 조건식도 만족하지 않을 때 수행될 문장들을 적는다.
}
```

첫 번째 조건식부터 순서대로 평가해서 결과가 참인 조건식을 만나면, 해당 블럭{}만 수행하고 ‘if–else if’ 문 전체를 벗어난다. 만일 결과가 참인 조건식이 하나도 없으면, 마지막에 있는 else블럭의 문장들이 수행된다. 그리고 else블럭은 생략이 가능하다. else블럭이 생략되었을 때는 if–else if문의 어떤 블럭도 수행되지 않을 수 있다.

예를 들어 다음과 같은 if–else if문이 있을 때, 변수 score의 값이 85라면, 다음의 과정으로 처리된다.

```
거짓
if(score >=90)
{
    grade = 'A';
}
else if(score >=80)
{
    grade = 'B';
}
else if(score >=70)
{
    grade = 'C';
}
else {
    grade = 'D';
}
```

① 결과가 참인 조건식을 만날 때까지 첫 번째 조건식부터 순서대로 평가한다.

(첫 번째 조건식은 거짓이므로, 두 번째 조건식으로 넘어간다.)

② 참인 조건식을 만나면, 해당 블럭{}의 문장을 수행한다.

③ if–else if문 전체를 빠져나온다.

미리보기용 pdf입니다.

예제

4-4

```

import java.util.Scanner;

class Ex4_4 {
    public static void main(String[] args) {
        int score = 0; // 점수를 저장하기 위한 변수
        char grade = ' '; // 학점을 저장하기 위한 변수. 공백으로 초기화한다.

        System.out.print("점수를 입력하세요.>");
        Scanner scanner = new Scanner(System.in);
        score = scanner.nextInt(); // 화면을 통해 입력받은 숫자를 score에 저장

        if (score >= 90) { // score가 90점 보다 같거나 크면 A학점
            grade = 'A';
        } else if (score >= 80) { // score가 80점 보다 같거나 크면 B학점
            grade = 'B';
        } else if (score >= 70) { // score가 70점 보다 같거나 크면 C학점
            grade = 'C';
        } else { // 나머지는 D학점
            grade = 'D';
        }
        System.out.println("당신의 학점은 " + grade + "입니다.");
    }
}

```

결과 1
점수를 입력하세요.>70
당신의 학점은 C입니다.

결과 2
점수를 입력하세요.>63
당신의 학점은 D입니다.

점수를 입력하면, 그에 해당하는 학점을 출력하는 간단한 예제이다. 여기서 한 가지 눈여겨봐야 할 것은 두 번째와 세 번째 조건식이다.

```

if (score >= 90) {
    grade = 'A';
} else if (80 <= score && score < 90) { // 80 ≤ score < 90
    grade = 'B';
} else if (70 <= score && score < 80) { // 70 ≤ score < 80
    grade = 'C';
} else { // score < 70
    grade = 'D';
}

```

점수가 90점 미만이고, 80점 이상인 사람에게 ‘B’학점을 주는 조건이라면, 위의 코드에서처럼 조건식이 ‘`80 <= score && score < 90`’이 되어야 한다. 그런데도 두 번째 조건식을 ‘`score >= 80`’이라고 쓸 수 있는 것은 첫 번째 조건식인 ‘`score >= 90`’이 거짓이기 때문이다.

‘`score >= 90`’이 거짓이라는 것은 ‘`score < 90`’이 참이라는 뜻이므로 두 번째 조건식에서 ‘`score < 90`’이라는 조건을 중복해서 확인할 필요가 없다.

```

if (score >= 90) {
    grade = 'A';
} else if (80 <= score && score < 90 ) {
    grade = 'B';
} else if (70 <= score && score < 80 ) {
    grade = 'C';
} else {
    grade = 'D';
}

```



www.coc

```

if (score >= 90) {
    grade = 'A';
} else if (score >=80) {
    grade = 'B';
} else if (score >=70) {
    grade = 'C';
} else {
    grade = 'D';
}

```

if문의 블럭 내에 또 다른 if문을 포함시키는 것이 가능한데 이것을 중첩 if문이라고 부르며 중첩의 횟수에는 거의 제한이 없다.

```
if (조건식1) {
    // 조건식1의 연산결과가 true일 때 수행될 문장들을 적는다.
    if (조건식2) {
        // 조건식1과 조건식2가 모두 true일 때 수행될 문장들
    } else {
        // 조건식1이 true이고, 조건식2가 false일 때 수행되는 문장들
    }
} else {
    // 조건식1이 false일 때 수행되는 문장들
}
```

위와 같이 내부의 if문은 외부의 if문보다 안쪽으로 들여쓰기를 해서 두 if문의 범위가 명확히 구분될 수 있도록 작성해야 한다.

중첩 if문에서는 괄호{}의 생략에 더욱 조심해야 한다. 바깥쪽의 if문과 안쪽의 if문이 서로 엉켜서 if문과 else블럭의 관계가 의도한 바와 다르게 형성될 수도 있기 때문이다.

```
if (num >= 0)
    if (num != 0)
        sign = '+';
    else
        sign = '-';
```

```
if (num >= 0) {
    if (num != 0) {
        sign = '+';
    } else {
        sign = '-';
    }
}
```



왼쪽의 코드는 언뜻 보기에도 else블럭이 바깥쪽의 if문에 속한 것처럼 보이지만, 괄호가 생략되었을 때 else블럭은 가까운 if문에 속한 것으로 간주되기 때문에 실제로는 오른쪽과 같이 안쪽 if문의 else블럭이 되어버린다. 이제 else블럭은 어떤 경우에도 수행될 수 없다. 아래와 같이 괄호{}를 넣어서 if블럭과 else블럭의 관계를 확실히 해주는 것이 좋다.

```
if (num >= 0) {
    if (num != 0)
        sign = '+';
} else {
    sign = '-';
}
```

미리보기용 pdf입니다.

예제

4-5

```

import java.util.Scanner;

class Ex4_5 {
    public static void main(String[] args) {
        int score = 0;
        char grade = ' ', opt = '0';

        System.out.print("점수를 입력해주세요.>");

        Scanner scanner = new Scanner(System.in);
        score = scanner.nextInt(); // 화면을 통해 입력받은 점수를 score에 저장

        System.out.printf("당신의 점수는 %d입니다.%n", score);

        if (score >= 90) {           // score가 90점 보다 같거나 크면 A학점(grade)
            grade = 'A';
            if (score >= 98) {       // 90점 이상 중에서도 98점 이상은 A+
                opt = '+';
            } else if (score < 94) { // 90점 이상 94점 미만은 A-
                opt = '-';
            }
        } else if (score >= 80){     // score가 80점 보다 같거나 크면 B학점(grade)
            grade = 'B';
            if (score >= 88) {
                opt = '+';
            } else if (score < 84) {
                opt = '-';
            }
        } else {                     // 나머지는 C학점(grade)
            grade = 'C';
        }
        System.out.printf("당신의 학점은 %c%c입니다.%n", grade, opt);
    }
}

```

결과 1 점수를 입력해주세요.>81
당신의 점수는 81입니다.
당신의 학점은 B-입니다.

결과 2 점수를 입력해주세요.>85
당신의 점수는 85입니다.
당신의 학점은 B0입니다.

결과 3 점수를 입력해주세요.>100
당신의 점수는 100입니다.
당신의 학점은 A+입니다.

위 예제는 모두 3개의 if문으로 이루어져 있으며 if문 안에 또 다른 2개의 if문을 포함하고 있는 모습을 하고 있다. 제일 바깥쪽에 있는 if문에서 점수에 따라 학점(grade)을 결정하고, 내부의 if문에서는 학점을 더 세부적으로 나누어서 평가를 하고 그 결과를 출력한다.

외부 if문의 조건식에 의해 한번 걸러졌기 때문에 내부 if문의 조건식은 더 간단해 질 수 있다.

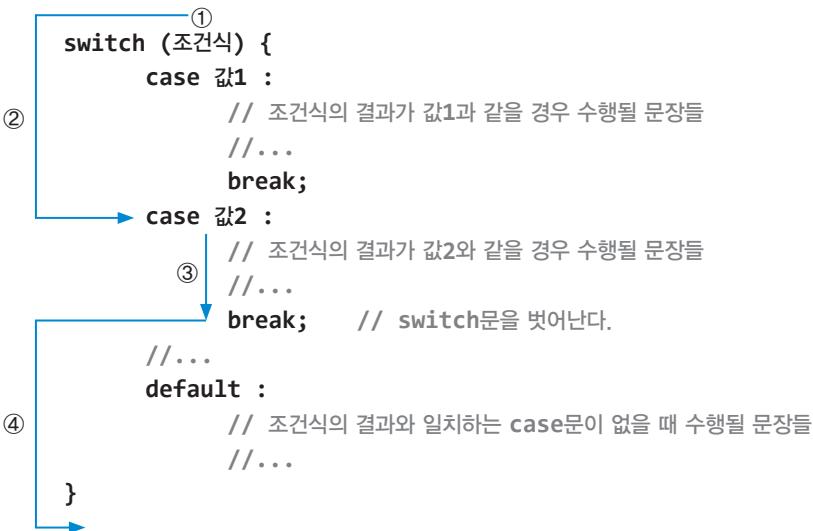
www.codechobo.com

if문은 조건식의 결과가 참과 거짓, 두 가지 밖에 없기 때문에 경우의 수가 많아질수록 else-if를 계속 추가해야 하므로 조건식이 많아져서 복잡해지고, 여러 개의 조건식을 계산해야 하므로 처리시간도 많이 걸린다.

이러한 if문과 달리 switch문은 단 하나의 조건식으로 많은 경우의 수를 처리할 수 있고, 표현도 간결하므로 알아보기 쉽다. 그래서 처리할 경우의 수가 많은 경우에는 if문 보다 switch문으로 작성하는 것이 좋다. 다만 switch문은 제약조건이 있기 때문에, 경우의 수가 많아도 어쩔 수 없이 if문으로 작성해야 하는 경우가 있다.

switch문은 조건식을 먼저 계산한 다음, 그 결과와 일치하는 case문으로 이동한다. 이동한 case문 아래에 있는 문장들을 수행하며, break문을 만나면 전체 switch문을 빠져나가게 된다.

- ① 조건식을 계산한다.
- ② 조건식의 결과와 일치하는 case문으로 이동한다.
- ③ 이후의 문장들을 수행한다.
- ④ break문이나 switch문의 끝을 만나면 switch문 전체를 빠져나간다.



만일 조건식의 결과와 일치하는 case문이 하나도 없는 경우에는 default문으로 이동한다. default문은 if문의 else블럭과 같은 역할을 한다고 보면 이해가 쉬울 것이다. default문의 위치는 어디라도 상관없으나 보통 마지막에 놓기 때문에 break문을 쓰지 않아도 된다.

switch문에서 break문은 각 case문의 영역을 구분하는 역할을 하는데, 만일 break문을 생략하면 case문 사이의 구분이 없어지므로 다른 break문을 만나거나 switch문 블럭{}의 끝을 만날 때까지 나오는 모든 문장들을 수행한다. 이러한 이유로 각 case문의 마지막에 break문을 빼먹는 실수를 하지 않도록 주의해야 한다.

미리보기용 pdf입니다.

switch문의 조건식은 결과값이 반드시 정수이어야 하며, 이 값과 일치하는 case문으로 이동하기 때문에 case문의 값 역시 정수이어야 한다. 그리고 중복되지 않아야 한다. 같은 값의 case문이 여러 개이면, 어디로 이동해야 할지 알 수 없기 때문이다.

게다가 case문의 값은 반드시 상수이어야 한다. 변수나 실수는 case문의 값으로 사용할 수 없다.

switch문의 제약조건

1. switch문의 조건식 결과는 정수 또는 문자열이어야 한다.
2. case문의 값은 정수 상수(문자 포함), 문자열만 가능하며, 중복되지 않아야 한다.

case문의 몇 가지 예를 아래의 코드에 적어보았다.

```
public static void main(String[] args) {  
    int num, result;  
    final int ONE = 1;  
    ...  
    switch(result) {  
        case '1':           // OK. 문자 리터럴(정수 49와 동일)  
        case ONE:          // OK. 정수 상수  
        case "YES":         // OK. 문자열 리터럴. JDK 1.7부터 허용  
        case num:           // 예리. 변수는 불가  
        case 1.0:           // 예리. 실수도 불가  
        ...  
    }  
}
```

문자 '1'은 정수 49와 동등하므로 문제가 없고, ONE은 정수가 아닌 것처럼 보이지만, 'final'이 붙은 정수 상수이므로 case문의 값으로 적합하다. 그러나 변수나 실수 리터럴은 case문의 값으로 적합하지 않다.

11 switch문의 제약조건 예제

예제

4-6

```

import java.util.Scanner;

class Ex4_6 {
    public static void main(String[] args) {
        System.out.print("현재 월을 입력하세요.>");
        Scanner scanner = new Scanner(System.in);
        int month = scanner.nextInt(); // 화면을 통해 입력받은 숫자를 month에 저장

        switch(month) {
            case 3:
            case 4:
            case 5:
                System.out.println("현재의 계절은 봄입니다.");
                break;
            case 6: case 7: case 8:
                System.out.println("현재의 계절은 여름입니다.");
                break;
            case 9: case 10: case 11:
                System.out.println("현재의 계절은 가을입니다.");
                break;
            default:
                // case 12: case 1: case 2:
                System.out.println("현재의 계절은 겨울입니다.");
        }
    } // main의 끝
}

```

결과

현재 월을 입력하세요.>3
현재의 계절은 봄입니다.

현재 몇 월인지 입력받아서 해당하는 계절을 출력하는 예제이다. 간단한 예제이므로 별로 설명할 것은 없다. case문은 한 줄에 하나씩 쓰던, 한 줄에 붙여서 쓰던 상관없다.

<pre> case 3: case 4: case 5: System.out.println("현재의 계절은 ..."); break; </pre>		<pre> case 3: case 4: case 5: System.out.println("현재의 계절은 ..."); break; </pre>
--	--	--

그리고 예제의 switch문을 if문으로 변경하면 다음과 같다.

```

if(month==3 || month==4 || month==5) {
    System.out.println("현재의 계절은 봄입니다.");
} else if(month==6 || month==7 || month==8) {
    System.out.println("현재의 계절은 여름입니다.");
} else if(month==9 || month==10 || month==11) {
    System.out.println("현재의 계절은 가을입니다.");
} else { // if(month==12 || month==1 || month==2)
    System.out.println("현재의 계절은 겨울입니다.");
}

```

난수(임의의 수)를 얻기 위해서는 Math.random()을 사용해야 하는데, 이 메서드는 0.0과 1.0사이의 범위에 속하는 하나의 double값을 반환한다. 0.0은 범위에 포함되고 1.0은 포함되지 않는다.

```
0.0 <= Math.random() < 1.0
```

만일 1과 3 사이의 정수를 구하기를 원한다면, 다음과 같은 과정으로 난수를 구하는 식을 얻을 수 있다.

① 각 변에 3을 곱한다.

```
0.0 * 3 <= Math.random() * 3 < 1.0 * 3  
0.0 <= Math.random() * 3 < 3.0
```

② 각 변을 int형으로 변환한다.

```
(int)0.0 <= (int)(Math.random() * 3) < (int)3.0  
0 <= (int)(Math.random() * 3) < 3
```

③ 각 변에 1을 더한다.

```
0 + 1 <= (int)(Math.random() * 3) + 1 < 3 + 1  
1 <= (int)(Math.random() * 3) + 1 < 4
```

자, 이제는 1과 3사이의 정수 중 하나를 얻을 수 있다. 1은 포함되고 4는 포함되지 않는다.

예제

4-7

```
class Ex4_7 {  
    public static void main(String args[]) {  
        int num = 0;  
  
        // 팰로{} 안의 내용을 5번 반복한다.  
        for (int i = 1; i <= 5; i++) {  
            num = (int) (Math.random() * 6) + 1;  
            System.out.println(num);  
        }  
    }  
}
```

결과	6
	1
	1
	5
	2

Math.random()을 사용했기 때문에 실행할 때마다 실행결과가 달라진다. 반복문 for를 이용해서 1과 6사이의 임의의 수를 얻어 출력하는 일을 5번 반복한다. 아직 반복문을 배우지 않았지만 이 예제를 이해하는데 어려움은 없을 것이다.

www.codechobo.com

반복문은 어떤 작업이 반복적으로 수행되도록 할 때 사용되며, 반복문의 종류로는 for문과 while문, 그리고 while문의 변형인 do-while문이 있다.

for문과 while문은 구조와 기능이 유사하여 어느 경우에나 서로 변환이 가능하며, 반복 횟수를 알고 있을 때는 for문을, 그렇지 않을 때는 while문을 사용한다. 아래의 for문은 블럭{} 내의 문장을 5번 반복한다. 즉, “I can do it.”이라는 문장이 5번 출력된다.

```

    1부터   5까지   1씩 증가
    ↓       ↓       ↓
for(int i=1;i<=5;i++) {
    System.out.println("I can do it.");
}

```

변수 i에 1을 저장한 다음, 매 반복마다 i의 값을 1씩 증가시킨다. 그러다가 i의 값이 5를 넘으면 조건식 ‘*i<=5*’가 거짓이 되어 반복을 마치게 된다. i의 값이 1부터 5까지 1씩 증가하니까 모두 5번 반복한다. 만일 10번 반복하기를 원한다면, 그저 5를 10으로 바꾸기만 하면 된다.

for문의 구조와 수행순서

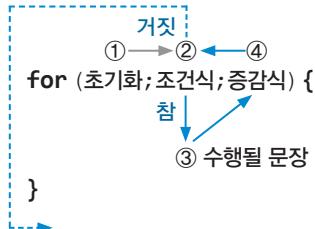
for문은 아래와 같이 ‘초기화’, ‘조건식’, ‘증감식’, ‘블럭{}’, 모두 4부분으로 이루어져 있으며, 조건식이 참인 동안 블럭{} 내의 문장들을 반복하다 거짓이 되면 반복문을 벗어난다.

```

for (초기화; 조건식; 증감식) {
    // 조건식이 참(true)인 동안 수행될 문장들을 적는다.
}

```

제일 먼저 ‘①초기화’가 수행되고, 그 이후부터는 조건식이 참인 동안 ‘②조건식 → ③수행될 문장 → ④증감식’의 순서로 계속 반복된다. 그러다가 조건식이 거짓이 되면, for문 전체를 빠져나가게 된다.



예제 4-8

```

class Ex4_8 {
    public static void main(String args[]) {
        for (int i = 1; i <= 3; i++) { // 괄호{}안의 문장을 3번 반복
            System.out.println("Hello");
        }
    }
}

```

결과
Hello
Hello
Hello

초기화

반복문에 사용될 변수를 초기화하는 부분이며 처음에 한번만 수행된다. 보통 변수 하나로 for 문을 제어하지만, 둘 이상의 변수가 필요할 때는 아래와 같이 콤마','를 구분자로 변수를 초기화하면 된다. 단, 두 변수의 타입은 같아야 한다.

```
for(int i=1;i<=10;i++) { ... }      // 변수 i의 값을 1로 초기화 한다.  
for(int i=1,j=0;i<=10;i++) { ... } // int타입의 변수 i와 j를 선언하고 초기화
```

조건식

조건식의 값이 참(true)이면 반복을 계속하고, 거짓(false)이면 반복을 중단하고 for문을 벗어난다. for의 뜻이 ‘~하는 동안’이므로 조건식이 ‘참인 동안’ 반복을 계속한다고 생각하면 쉽다.

```
for(int i=1;i<=10;i++) { ... } // 'i<=10'가 참인 동안 블럭{}안의 문장들을 반복
```

조건식을 잘못 작성하면 블럭{} 내의 문장이 한 번도 수행되지 않거나 영원히 반복되는 무한 반복에 빠지기 쉬우므로 주의해야 한다.

증감식

반복문을 제어하는 변수의 값을 증가 또는 감소시키는 식이다. 매 반복마다 변수의 값이 증감식에 의해서 점진적으로 변하다가 결국 조건식이 거짓이 되어 for문을 벗어나게 된다. 변수의 값을 1씩 증가시키는 연산자 ‘++’이 증감식에 주로 사용되지만, 다음과 같이 다양한 연산자들로 증감식을 작성할 수도 있다.

```
for(int i=1;i<=10;i++) { ... } // 1부터 10까지 1씩 증가  
for(int i=10;i>=1;i--) { ... } // 10부터 1까지 1씩 감소  
for(int i=1;i<=10;i+=2) { ... } // 1부터 10까지 2씩 증가  
for(int i=1;i<=10;i*=3) { ... } // 1부터 10까지 3배씩 증가
```

증감식도 쉼표','를 이용해서 두 문장 이상을 하나로 연결해서 쓸 수 있다.

```
for(int i=1, j=10;i<=10;i++, j--) { ... } // i는 1부터 10까지 1씩 증가하고  
// j는 10부터 1까지 1씩 감소한다.
```

지금까지 살펴본 이 세 가지 요소는 필요하지 않으면 생략할 수 있으며, 심지어 모두 생략하는 것도 가능하다.

```
for(;;) { ... } // 초기화, 조건식, 증감식 모두 생략. 조건식은 참이 된다.
```

조건식이 생략된 경우, 참(true)으로 간주되어 무한 반복문이 된다. 대신 블럭{} 안에 if문을 넣어서 특정 조건을 만족하면 for문을 빠져 나오게 해야 한다.

www.codechobo.com

14 for문 예제

예제

4-9

```
class Ex4_9 {
    public static void main(String[] args) {
        for(int i=1;i<=5;i++)
            System.out.println(i); // i의 값을 출력한다.

        for(int i=1;i<=5;i++)
            System.out.print(i); // print()를 쓰면 가로로 출력된다.

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

결과	1
	2
	3
	4
	5
	12345

(참고) 반복하려는 문장이 단 하나일 때는 괄호{}를 생략할 수 있다.

1부터 5까지 세로로 한번, 가로로 한번 출력하는 간단한 예제이다. 아래의 표를 보면 i의 값이 변화함에 따라 조건식의 결과가 어떻게 되는지 알 수 있다.

i	i <= 5
1	1 <= 5 → true 참
2	2 <= 5 → true 참
3	3 <= 5 → true 참
4	4 <= 5 → true 참
5	5 <= 5 → true 참
6	6 <= 5 → false 거짓, 반복종료

사실 i의 값은 1부터 6까지 변하지만, i값이 6일 때 조건식이 '6<=5'가 되고, 이 식의 결과는 거짓(false)이므로 for문을 벗어나기 때문에 6은 출력되지 않는다.

예제

4-10

```
class Ex4_10 {
    public static void main(String[] args) {
        int sum = 0; // 합계를 저장하기 위한 변수.

        for(int i=1; i <= 5; i++) {
            sum += i; // sum = sum + i;
            System.out.printf("1부터 %2d 까지의 합: %2d\n", i, sum);
        } // main의 끝
    }
}
```

결과	1부터 1 까지의 합: 1
	1부터 2 까지의 합: 3
	1부터 3 까지의 합: 6
	1부터 4 까지의 합: 10
	1부터 5 까지의 합: 15

1부터 10까지의 합을 구하는 예제이다. 변수 i를 1부터 10까지 변화시키면서 i를 sum에 계속 더해서 누적시킨다. 그 과정을 출력했으므로 어렵지 않게 이해할 수 있을 것이다.

i	sum = sum + i
1	1 = 0 + 1
2	3 = 1 + 2
3	6 = 3 + 3
4	10 = 6 + 4
5	15 = 10 + 5

미리보기용 pdf입니다.

for문 안에 또 다른 for문을 포함시키는 것이 가능하며, 중첩 for문이라고 한다. 중첩 횟수는 거의 제한이 없다. 중첩 for문을 설명하는데 별찍기 만큼 좋은 것은 없다.

만일 다음과 같이 5행 10열의 별 '*'을 찍으려면 어떻게 해야 할까?

```
*****
*****
*****
*****
*****
```

가장 간단한 방법은 다음과 같이 한 줄씩 5번 출력하는 것이다.

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

그러나 우리는 for문을 배웠으니, 다음과 같이 간단히 할 수 있다.

```
for(int i=1;i<=5;i++) {
    System.out.println("*****"); // 10개의 별을 출력한다.
}
```

'System.out.println("*****")' 역시 반복적인 일을 하는 문장이니 for문으로 바꿀 수 있다. 이 문장을 for문으로 바꾸면 다음과 같다.

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

```
for(int j=1;j<=10;j++) {
    System.out.print("*");
}
System.out.println();
```

왼쪽의 문장 대신 오른쪽의 for문을 이전의 for문에 넣으면 다음과 같이 두 개의 for문이 중첩된 형태가 된다.

```
for(int i=1;i<=5;i++) {
    System.out.println("*****");
}
```

```
for(int i=1;i<=5;i++) {
    for(int j=1;j<=10;j++) {
        System.out.print("*");
    }
    System.out.println();
}
```

이번엔 다음과 같은 삼각형 모양의 별을 출력해보자.

```
*  
**  
***  
****  
*****
```

앞서 배운 바와 같이 가로로 출력하려면, `println`메서드 대신 `print`메서드로 출력하면 된다.
아래의 `for`문은 ‘*****’을 출력하고 줄 바꿈을 한다.

```
for(int j=1;j<=5;j++) {  
    System.out.print("*"); // *****을 출력한다.  
}  
System.out.println(); // 줄 바꿈을 한다.
```

따라서 다음과 같이 코드를 작성하면, 우리가 원하는 결과를 얻을 수 있다.

```
for(int j=1;j<=1;j++){System.out.print("*");} System.out.println(); // *  
for(int j=1;j<=2;j++){System.out.print("*");} System.out.println(); // **  
for(int j=1;j<=3;j++){System.out.print("*");} System.out.println(); // ***  
for(int j=1;j<=4;j++){System.out.print("*");} System.out.println(); // ****  
for(int j=1;j<=5;j++){System.out.print("*");} System.out.println(); // *****
```

위 문장들을 잘 보면 조건식의 숫자만 변할 뿐 나머지는 같다. 똑같은 내용이 반복되는데 반복문으로 간단히 처리할 방법이 없을까? 이럴 때는 한 문장의 조건식에 숫자 대신 변수 `i`를 넣고, 이 문장을 `i`의 값이 1부터 5까지 증가하는 `for`문 안에 넣으면 된다.

```
for(int i=1;i<=5;i++) {  
    for(int j=1;j<=i;j++) { System.out.print("*");} System.out.println();  
}
```

예제

4-11

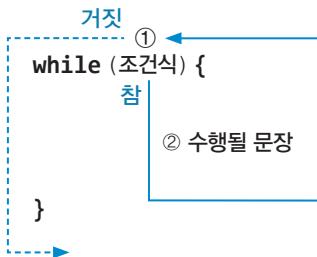
```
import java.util.*;  
  
class Ex4_11 {  
    public static void main(String[] args) {  
  
        for(int i=1;i<=5;i++) {  
            for(int j=1;j<=i;j++) {  
                System.out.print("*");  
            }  
            System.out.println();  
        } // main의 끝  
    }
```

결과
*
**

for문에 비해 while문은 구조가 간단하다. if문처럼 조건식과 블럭{}만으로 이루어져 있다. 다만 if문과 달리 while문은 조건식이 ‘참(true)인 동안’, 즉 조건식이 거짓이 될 때까지 블럭{} 내의 문장을 반복한다.

```
while (조건식) {
    // 조건식의 연산결과가 참(true)인 동안, 반복될 문장들을 적는다.
}
```

while문은 먼저 조건식을 평가해서 조건식이 거짓이면 문장 전체를 벗어나고, 참이면 블럭{} 내의 문장을 수행하고 다시 조건식으로 돌아간다. 조건식이 거짓이 될 때까지 이 과정이 계속 반복된다.



- ① 조건식이 참(true)이면 블럭{} 안으로 들어가고, 거짓(false)이면 while문을 벗어난다.
- ② 블럭{}의 문장을 수행하고 다시 조건식으로 돌아간다.

for문과 while문의 비교

1부터 10까지의 정수를 순서대로 출력하는 for문을 while문으로 변경하면 아래 오른쪽과 같다.

```
// 초기화, 조건식, 증감식
for(int i=1;i<=10;i++) {
    System.out.println(i);
}
```

```
int i=1; // 초기화

while(i<=10) { // 조건식
    System.out.println(i);
    i++; // 증감식
}
```

위의 두 코드는 완전히 동일하다. for문은 초기화, 조건식, 증감식을 한 곳에 모아 놓은 것일 뿐, while문과 다르지 않다. 그래서 for문과 while문은 항상 서로 변환이 가능하다.

그래도 이 경우 while문 보다 for문이 더 간결하고 알아보기 쉽다. 만일 초기화이나 증감식이 필요하지 않은 경우라면, while문이 더 적합할 것이다. 앞으로 소개할 예제들은 for문보다 while문이 더 적합하다고 판단된 것들이다.

www.codechobo.com

예제

4-12

```
class Ex4_12 {
    public static void main(String[] args) {
        int i = 5;

        while(i--!=0) {
            System.out.println(i + " - I can do it.");
        }
    } // main의 끝
}
```

결과
4 - I can do it.
3 - I can do it.
2 - I can do it.
1 - I can do it.
0 - I can do it.

변수 i의 값만큼 블럭{}을 반복하는 예제이다. i의 값이 5이므로 'I can do it.'이 모두 5번 (4~0) 출력되었다. while문의 조건식 'i--!=0'은 i의 값이 0이 아닌 동안만 참이 되고, i의 값이 매 반복마다 1씩 감소하다 0이 되면 조건식은 거짓이 되어 while문을 벗어난다.

'i--'가 후위형이므로 조건식이 평가된 후에 i의 값이 감소된다는 점에 주의하자. 예를 들어, i의 값이 1일 때는 조건식이 참으로 평가된 후에 i의 값이 1 감소되어 0이 된다. 그래서 실행 결과에서 i의 값이 5~1이 아닌 4~0으로 출력된 것이다.

아직 이해가 잘 안 간다면 아래 오른쪽과 같이 조건식에서 감소 연산자 '--'를 분리해보자. 좀 더 이해하기 쉬운 코드가 될 것이다.

```
while(i--!=0) {
    System.out.println(i);
}
```



```
while(i!=0) {
    i--;
    System.out.println(i);
}
```

다음은 1부터 몇까지 더해야 100을 넘지 않는지 알아내는 예제이다.

예제

4-13

```
class Ex4_13 {
    public static void main(String[] args) {
        int sum = 0;
        int i = 0;
        // i를 1씩 증가시켜서 sum에 계속 더해나간다.
        while (sum <= 100) {
            System.out.printf("%d - %d%n", i, sum);
            sum += ++i;
        }
    } // main의 끝
}
```

결과
0 - 0
1 - 1
2 - 3
3 - 6
4 - 10
5 - 15
6 - 21
7 - 28
8 - 36
9 - 45
10 - 55
11 - 66
12 - 78
13 - 91

미리보기용 pdf입니다.

예제

4-14

```

import java.util.*;
class Ex4_14 {
    public static void main(String[] args) {
        int num = 0, sum = 0;
        System.out.print("숫자를 입력하세요.(예:12345)>");
        Scanner scanner = new Scanner(System.in);
        String tmp = scanner.nextLine(); // 화면을 통해 입력받은 내용을 tmp에 저장
        num = Integer.parseInt(tmp); // 입력받은 문자열(tmp)을 숫자로 변환

        while(num!=0) {
            // num을 10으로 나눈 나머지를 sum에 더함
            sum += num%10; // sum = sum + num%10;
            System.out.printf("sum=%3d num=%d\n", sum, num);

            num /= 10; // num = num / 10; num을 10으로 나눈 값을 다시 num에 저장
        }
        System.out.println("각 자리수의 합:"+sum);
    }
}

```

결과

숫자를 입력하세요.(예:12345)>12345
 sum= 5 num=12345
 sum= 9 num=1234
 sum= 12 num=123
 sum= 14 num=12
 sum= 15 num=1
 각 자리수의 합:15

사용자로부터 숫자를 입력받고, 이 숫자의 각 자리의 합을 구하는 예제이다. 실행 결과에서 알 수 있듯이 12345를 입력하면, 결과는 15(1+2+3+4+5=15)이다.

어떤 수를 10으로 나머지 연산하면 마지막 자리를 얻을 수 있다. 그리고 10으로 나누면 마지막 한자리가 제거된다.

$$\begin{aligned} 12345 \% 10 &\rightarrow 5 \\ 12345 / 10 &\rightarrow 1234 \end{aligned}$$

그래서 입력 받은 숫자 num을 0이 될 때까지 반복해서 10으로 나눠가면서, 10으로 나머지 연산을 하면 num의 모든 자리를 얻을 수 있다. 이 과정을 단계별로 살펴보면 다음과 같다.

num	num%10	sum = sum + num%10 (sum+=num%10)	num = num / 10 (num/=10)
12345	5	5 = 0 + 5	1234 = 12345 / 10
1234	4	9 = 5 + 4	123 = 1234 / 10
123	3	12 = 9 + 3	12 = 123 / 10
12	2	14 = 12 + 2	1 = 12 / 10
1	1	15 = 14 + 1	0 = 1 / 10
0	-	-	-

num의 값은 'num/=10'에 의해 한자리씩 줄어들다가 0이 되면, while문의 조건식이 거짓이 되어 반복을 멈춘다.

19 do-while문

do-while문은 while문의 변형으로 기본적인 구조는 while문과 같으나 조건식과 블럭{}의 순서를 바꿔놓은 것이다. 그래서 while문과 반대로 블럭{}을 먼저 수행한 후에 조건식을 평가한다. while문은 조건식의 결과에 따라 블럭{}이 한 번도 수행되지 않을 수 있지만, do-while문은 최소한 한번은 수행될 것을 보장한다.

```
do {
    // 조건식의 연산결과가 참일 때 수행될 문장들을 적는다.(처음 한 번은 무조건 실행)
} while (조건식); ← 끝에 ';'을 잊지 않도록 주의
```

그리 많이 쓰이지는 않지만, 다음의 예제처럼 반복적으로 사용자의 입력을 받아서 처리할 때 유용하다.

예제

4-15

```
import java.util.*;
class Ex4_15 {
    public static void main(String[] args) {
        int input = 0, answer = 0;

        answer = (int)(Math.random() * 100) + 1; // 1~100 사이의 임의의 수를 저장
        Scanner scanner = new Scanner(System.in);

        do {
            System.out.print("1과 100사이의 정수를 입력하세요.>");
            input = scanner.nextInt();

            if(input > answer) {
                System.out.println("더 작은 수로 다시 시도해보세요.");
            } else if(input < answer) {
                System.out.println("더 큰 수로 다시 시도해보세요.");
            }
        } while(input!=answer);

        System.out.println("정답입니다.");
    }
}
```

결과

1과 100사이의 정수를 입력하세요.>50
 더 작은 값으로 다시 시도해보세요.
 1과 100사이의 정수를 입력하세요.>25
 더 작은 값으로 다시 시도해보세요.
 1과 100사이의 정수를 입력하세요.>12
 더 큰 값으로 다시 시도해보세요.
 1과 100사이의 정수를 입력하세요.>21
 정답입니다.

Math.random()을 이용해서 1과 100 사이의 임의의 수를 변수 answer에 저장하고, 이 값을 맞출 때까지 반복하는 예제이다. 사용자 입력인 input이 변수 answer의 값과 다른 동안 반복 하다가 두 값이 같으면 반복을 벗어난다.

미리보기용 pdf입니다.

앞서 switch문에서 break문에 대해 배웠던 것을 기억할 것이다. 반복문에서도 break문을 사용할 수 있는데, switch문에서 그랬던 것처럼, break문은 자신이 포함된 가장 가까운 반복문을 벗어난다. 주로 if문과 함께 사용되어 특정 조건을 만족할 때 반복문을 벗어나게 한다.

예제
4-16

```
class Ex4_16 {
    public static void main(String[] args) {
        int sum = 0;
        int i = 0;

        while(true) {
            if(sum > 100)
                • break;
            ++i;
            sum += i;
        } // end of while
        System.out.println("i=" + i);
        System.out.println("sum=" + sum);
    }
}
```

break문이 수행되면 이 부분은 실행되지 않고 while문을 완전히 벗어난다.

결과
i=14
sum=105

숫자를 1부터 계속 더해 나가서 몇까지 더하면 합이 100을 넘는지 알아내는 예제이다. i의 값을 1부터 1씩 계속 증가시켜가며 더해서 sum에 저장한다. sum의 값이 100을 넘으면 if문의 조건식이 참이므로 break문이 수행되어 자신이 속한 반복문을 즉시 벗어난다.

이처럼 무한 반복문에는 조건문과 break문이 항상 같이 사용된다. 그렇지 않으면 무한히 반복되기 때문에 프로그램이 종료되지 않을 것이다.

참고 sum += i;와 ++i; 두 문장을 sum += ++i;과 같이 한 문장으로 줄여 쓸 수 있다.

continue문은 반복문 내에서만 사용될 수 있으며, 반복이 진행되는 도중에 continue문을 만나면 반복문의 끝으로 이동하여 다음 반복으로 넘어간다. for문의 경우 중감식으로 이동하며, while문과 do-while문의 경우 조건식으로 이동한다.

continue문은 반복문 전체를 벗어나지 않고 다음 반복을 계속 수행한다는 점이 break문과 다르다. 주로 if문과 함께 사용되어 특정 조건을 만족하는 경우에 continue문 이후의 문장들을 수행하지 않고 다음 반복으로 넘어가서 계속 진행하도록 한다.

전체 반복 중에 특정조건을 만족하는 경우를 제외하고자 할 때 유용하다.

예제

4-17

```
class Ex4_17 {
    public static void main(String[] args) {
        for(int i=0;i <= 10;i++) {
            if (i%3==0)
                • continue; •
                System.out.println(i);
            }
        }
    }
```

조건식이 참이 되어 continue문이 수행되면
블럭의 끝으로 이동한다.
break문과 달리 반복문을 벗어나지 않는다.

결과	1
	2
	4
	5
	7
	8
	10

1과 10사이의 숫자를 출력하되 그 중에서 3의 배수인 것은 제외하도록 하였다. i의 값이 3의 배수인 경우, if문의 조건식 'i%3==0'은 참이 되어 continue문에 의해 반복문의 블럭 끝'}'으로 이동된다. 즉, continue문과 반복문 블럭의 끝'}' 사이의 문장들을 건너뛰고 반복을 이어가는 것이다.

미리보기용 pdf입니다.

예제

4-18

```

import java.util.*;

class Ex4_18 {
    public static void main(String[] args) {
        int menu = 0;
        int num = 0;

        Scanner scanner = new Scanner(System.in);

        while(true) {
            System.out.println("(1) square");
            System.out.println("(2) square root");
            System.out.println("(3) log");
            System.out.print("원하는 메뉴(1~3)를 선택하세요.(종료:0)>");

            String tmp = scanner.nextLine(); // 화면에서 입력받은 내용을 tmp에 저장
            menu = Integer.parseInt(tmp); // 입력받은 문자열(tmp)을 숫자로 변환

            if(menu==0) {
                System.out.println("프로그램을 종료합니다.");
                break;
            } else if (!(1 <= menu && menu <= 3)) {
                System.out.println("메뉴를 잘못 선택하셨습니다.(종료는 0)");
                continue;
            }

            System.out.println("선택하신 메뉴는 "+ menu +"번입니다.");
        }
    } // main의 끝
}

```

결과

```

(1) square
(2) square root
(3) log
원하는 메뉴(1~3)를 선택하세요.(종료:0)>4
메뉴를 잘못 선택하셨습니다.(종료는 0)
(1) square
(2) square root
(3) log
원하는 메뉴(1~3)를 선택하세요.(종료:0)>1
선택하신 메뉴는 1번입니다.
(1) square
(2) square root
(3) log
원하는 메뉴(1~3)를 선택하세요.(종료:0)>0
프로그램을 종료합니다.

```

메뉴를 보여주고 선택하게 하는 예제이다. 메뉴를 잘못 선택한 경우, continue문으로 다시 메뉴를 보여주고, 종료(0)를 선택한 경우 break문으로 반복을 벗어나 프로그램이 종료되도록 했다. 이 예제는 메뉴를 보여주고 선택하는 것을 반복하는 것 외에 별다른 기능이 없지만, 곧 이 예제를 좀 더 쓸 만한 것으로 발전시킨 예제를 소개할 것이다.

23 이름 붙은 반복문

break문은 근접한 단 하나의 반복문만 벗어날 수 있기 때문에, 여러 개의 반복문이 중첩된 경우에는 break문으로 중첩 반복문을 완전히 벗어날 수 없다. 이때는 중첩 반복문 앞에 이름을 붙이고 break문과 continue문에 이름을 지정해 줌으로써 하나 이상의 반복문을 벗어나거나 반복을 건너뛸 수 있다.

예제
4-19

```
class Ex4_19
{
    public static void main(String[] args)
    {
        // for문에 Loop1이라는 이름을 붙였다.
        Loop1 : for(int i=2;i <=9;i++) {
            for(int j=1;j <=9;j++) {
                if(j==5)
                    • break Loop1;
                // • break;
                // continue Loop1; •
                // continue; •
                System.out.println(i+"*"+ j +"="+ i*j);
            } // end of for i
            System.out.println();
        } // end of Loop1
    }
}
```

결과	2*1=2 2*2=4 2*3=6 2*4=8
----	----------------------------------

구구단을 출력하는 예제이다. 제일 바깥에 있는 for문에 Loop1이라는 이름을 붙였다. 그리고 j가 5일 때 break문을 수행하도록 했다. 반복문의 이름이 지정되지 않은 break문은 자신이 속한 하나의 반복문만 벗어날 수 있지만, 지금처럼 반복문에 이름을 붙여 주고 break문에 반복문 이름을 지정해주면 하나 이상의 반복문도 벗어날 수 있다.

j가 5일 때 반복문 Loop1을 벗어나도록 했으므로 2단의 4번째 줄까지 밖에 출력되지 않았다. 만일 반복문의 이름이 지정되지 않은 break문이었다면 2단부터 9단까지 모두 네 줄씩 출력되었을 것이다.

예제에서는 ‘break Loop1;’ 아래의 세 문장들을 주석처리하였다. 이 네 문장(2개의 break문과 2개의 continue문) 중의 하나를 선택하고 선택한 문장을 제외한 나머지는 주석처리한 다음, 어떤 결과를 얻을지 예측하고 실행한 후에 예측한 결과와 비교해보자.

참고

continue Loop1;과 같은 문장을 쓸 일은 거의 없을 테니 무시해도 좋다.

미리보기용 pdf입니다.

예제

4-20

```

import java.util.*;

class Ex4_20 {
    public static void main(String[] args) {
        int menu = 0, num = 0;
        Scanner scanner = new Scanner(System.in);

        outer: // while문에 outer라는 이름을 붙인다.
        while(true) {
            System.out.println("(1) square");
            System.out.println("(2) square root");
            System.out.println("(3) log");
            System.out.print("원하는 메뉴(1~3)를 선택하세요.(종료:0)>");

            String tmp = scanner.nextLine(); // 화면에서 입력받은 내용을 tmp에 저장
            menu = Integer.parseInt(tmp); // 입력받은 문자열(tmp)을 숫자로 변환

            if(menu==0) {
                System.out.println("프로그램을 종료합니다.");
                break;
            } else if (!(1<= menu && menu <= 3)) {
                System.out.println("메뉴를 잘못 선택하셨습니다.(종료는 0)");
                continue;
            }

            for(;;) {
                System.out.print("계산할 값을 입력하세요.(계산 종료:0, 전체 종료:99)>");
                tmp = scanner.nextLine(); // 화면에서 입력받은 내용을 tmp에 저장
                num = Integer.parseInt(tmp); // 입력받은 문자열(tmp)을 숫자로 변환

                if(num==0)
                    break; // 계산 종료. for문을 벗어난다.

                if(num==99)
                    break outer; // 전체 종료. for문과 while문을 모두 벗어난다.

                switch(menu) {
                    case 1:
                        System.out.println("result="+ num*num);
                        break;
                    case 2:
                        System.out.println("result="+ Math.sqrt(num));
                        break;
                    case 3:
                        System.out.println("result="+ Math.log(num));
                        break;
                }
            } // for(;;)
        } // while의 끝
    } // main의 끝
}

```

```

결과
(1) square
(2) square root
(3) log
원하는 메뉴(1~3)를 선택하세요.(종료:0)>1
계산할 값을 입력하세요.(계산 종료:0, 전체 종료:99)>2
result=4
계산할 값을 입력하세요.(계산 종료:0, 전체 종료:99)>3
result=9
계산할 값을 입력하세요.(계산 종료:0, 전체 종료:99)>0
(1) square
(2) square root
(3) log
원하는 메뉴(1~3)를 선택하세요.(종료:0)>2
계산할 값을 입력하세요.(종료:0, 전체종료:99)>4
result=2.0
계산할 값을 입력하세요.(종료:0, 전체종료:99)>99

```

이 예제는 예제4-18를 발전시킨 것으로 메뉴를 선택하면 해당 연산을 반복적으로 수행할 수 있게 for문을 추가하였다. 이 예제를 실행해서 다양하게 테스트한 후에 분석하면 더 이해하기 쉬울 것이다.

아래와 같이 반복문만 떼어놓고 보면, 무한 반복문인 while문 안에 또 다른 무한 반복문인 for문이 중첩된 구조라는 것을 알 수 있다. while문은 메뉴를 반복해서 선택할 수 있게 해주고, for문은 선택된 메뉴의 작업을 반복해서 할 수 있게 해준다.

```

outer:
while(true) { // 무한 반복문
    ...
    for(;;) { // 무한 반복문
        ...
        if(num==0) // 계산 종료. for문을 벗어난다.
            • break;
        if(num==99) // 전체 종료. for문과 while문 모두 벗어난다.
            • break outer;
        ...
    } // for(;;)
} // while(true)

```

선택된 메뉴에서 0을 입력하면 break문으로 for문을 벗어나서 다른 메뉴를 선택할 수 있게 되고, 99를 입력하면 'break outer;'에 의해 for문과 while문 모두를 벗어나 프로그램이 종료된다.

미리보기용 pdf입니다.

연습문제

4-1 다음의 문장들을 조건식으로 표현하라.

- ① int형 변수 x가 10보다 크고 20보다 작을 때 true인 조건식
- ② char형 변수 ch가 공백이나 탭이 아닐 때 true인 조건식
- ③ char형 변수 ch가 'x' 또는 'X'일 때 true인 조건식
- ④ char형 변수 ch가 숫자('0'~'9')일 때 true인 조건식
- ⑤ char형 변수 ch가 영문자(대문자 또는 소문자)일 때 true인 조건식
- ⑥ int형 변수 year가 400으로 나눠떨어지거나 또는 4로 나눠떨어지고 100으로 나눠떨어지지 않을 때 true인 조건식
- ⑦ boolean형 변수 powerOn가 false일 때 true인 조건식
- ⑧ 문자열 참조변수 str0 "yes"일 때 true인 조건식

4-2 1부터 20까지의 정수 중에서 2 또는 3의 배수가 아닌 수의 총합을 구하시오.

4-3 $1 + (1+2) + (1+2+3) + (1+2+3+4) + \dots + (1+2+3+\dots+10)$ 의 결과를 계산하시오.

4-4 $1 + (-2) + 3 + (-4) + \dots$ 과 같은 식으로 계속 더해나갔을 때, 몇까지 더해야 총합이 100 이상이 되는지 구하시오.

4-5 다음의 for문을 while문으로 변경하시오.

```
public class Exercise4_5 {
    public static void main(String[] args) {
        for (int i = 0; i <= 10; i++) {
            for (int j = 0; j <= i; j++)
                System.out.print("*");
            System.out.println();
        }
    } // end of main
} // end of class
```

4-6 두 개의 주사위를 던졌을 때, 눈의 합이 6이 되는 모든 경우의 수를 출력하는 프로그램을 작성하시오.**4-7** 숫자로 이루어진 문자열 str이 있을 때, 각 자리의 합을 더한 결과를 출력하는 코드를 완성하라. 만일 문자열이 "12345"라면, '1+2+3+4+5'의 결과인 15를 출력이 출력되어야 한다. (1)에 알맞은 코드를 넣으시오.

(Hint) String클래스의 charAt(int i)을 사용

```
class Exercise4_7 {
    public static void main(String[] args) {
        String str = "12345";
        int sum = 0;

        for (int i = 0; i < str.length(); i++) {
            /*
                (1) 알맞은 코드를 넣어 완성하시오.
            */
        }

        System.out.println("sum=" + sum);
    }
}
```

결과 15

4-8 `Math.random()`을 이용해서 1부터 6 사이의 임의의 정수를 변수 `value`에 저장하는 코드를 완성하라. (1)에 알맞은 코드를 넣으시오.

```
class Exercise4_8 {
    public static void main(String[] args) {
        int value = ( /* (1) */ );
        System.out.println("value:"+value);
    }
}
```

4-9 `int` 타입의 변수 `num`이 있을 때, 각 자리의 합을 더한 결과를 출력하는 코드를 완성하라. 만일 변수 `num`의 값이 12345라면, '1+2+3+4+5'의 결과인 15를 출력하라. (1)에 알맞은 코드를 넣으시오.

(주의) 문자열로 변환하지 말고 숫자로만 처리해야 한다.

```
class Exercise4_9 {
    public static void main(String[] args) {
        int num = 12345;
        int sum = 0;

        /*
            (2) 알맞은 코드를 넣어 완성하시오.
        */

        System.out.println("sum="+sum);
    }
}
```

결과 15

4-10 다음은 숫자맞히기 게임을 작성한 것이다. 1과 100 사이의 값을 반복적으로 입력해서 컴퓨터가 생각한 값을 맞히면 게임이 끝난다. 사용자가 값을 입력하면, 컴퓨터는 자신이 생각한 값과 비교해서 결과를 알려준다. 사용자가 컴퓨터가 생각한 숫자를 맞히면 게임이 끝나고 몇 번 만에 숫자를 맞혔는지 알려준다. (1)~(2)에 알맞은 코드를 넣어 프로그램을 완성하시오.

```
class Exercise4_10
{
    public static void main(String[] args)
    {
        // 1~100사이의 임의의 값을 얻어서 answer에 저장한다.
        int answer = /* (1) */ ;
        int input = 0; // 사용자입력을 저장할 공간
        int count = 0; // 시도횟수를 세기위한 변수

        // 화면으로 부터 사용자입력을 받기 위해서 Scanner클래스 사용
        java.util.Scanner s = new java.util.Scanner(System.in);

        do {
            count++;
            System.out.print("1과 100사이의 값을 입력하세요 :");
            input = s.nextInt(); // 입력받은 값을 변수 input에 저장한다.

            /*
             * (2) 알맞은 코드를 넣어 완성하시오.
             */
        } while(true); // 무한반복문
    } // end of main
} // end of class
```

결과 1과 100사이의 값을 입력하세요 :50
더 큰 수를 입력하세요.
1과 100사이의 값을 입력하세요 :75
더 큰 수를 입력하세요.
1과 100사이의 값을 입력하세요 :87
더 작은 수를 입력하세요.
1과 100사이의 값을 입력하세요 :80
더 작은 수를 입력하세요.
1과 100사이의 값을 입력하세요 :77
더 작은 수를 입력하세요.
1과 100사이의 값을 입력하세요 :76
맞혔습니다.
시도횟수는 6번입니다.