차례

서문 xi

1장 자바의 발전 과정 1

1.1 들어가며 1
1.2 자바 연대표 2
1.3 자바는 죽었나? 4
1.4 자바 버전 정책의 변화 6
1.5 자바 버전별 새로운 기능 8
- **1.5.1** 자바 5 9
- **1.5.2** 자바 6 10
- **1.5.3** 자바 7 11
- **1.5.4** 자바 8 12
- **1.5.5** 자바 9 13
- **1.5.6** 자바 10 15
- **1.5.7** 자바 11 16
- **1.5.8** 자바 12 17
- **1.5.9** 자바 13 17
- **1.5.10** 자바 14 18

1.6 자바 버전 명명 규칙 19
1.7 요약 20

2장 인터페이스와 클래스 21

2.1 들어가며 21
2.2 인터페이스 사용 시 문제점 22
2.3 인터페이스의 진화 27
2.4 default, static, private 메서드 32
2.5 클래스와의 차이점과 제약 조건 36
2.6 다중 상속 관계 39
2.7 요약 49

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

3장 함수형 프로그래밍 51

3.1 들어가며 51
3.2 여행 상품 개발 52
3.3 조회 조건 추가 55
3.4 인터페이스로 대응 59
3.5 람다 표현식으로 코드 함축 62
3.6 메서드 참조 64
3.7 요약 66

4장 람다와 함수형 인터페이스 67

4.1 들어가며 67
4.2 람다 표현식이 필요한 이유 68
4.3 람다 표현식 이해하기 72
4.3.1 람다 표현식으로 전환 73
4.3.2 형식 추론 78
4.3.3 람다 표현식과 변수 78

4.4 함수형 인터페이스 기본 79
4.4.1 Consumer 인터페이스 81
4.4.2 Function 인터페이스 83
4.4.3 Predicate 인터페이스 84
4.4.4 Supplier 인터페이스 84
4.4.5 어떻게 쓸 것인가? 85

4.5 함수형 인터페이스 응용 86
4.5.1 기본형 데이터를 위한 인터페이스 86
4.5.2 Operator 인터페이스 90

4.6 메서드 참조 93
4.6.1 메서드 참조란 94
4.6.2 생성자 참조 101

4.7 람다 표현식 조합 103
4.7.1 Consumer 조합 103
4.7.2 Predicate 조합 106
4.7.3 Function 조합 109

4.8 요약 112

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

5장 스트림 API 113

5.1 들어가며 113
5.2 스트림 인터페이스 이해 114
5.2.1 스트림 인터페이스 116
5.2.2 기본형 스트림 인터페이스 119
5.3 스트림 객체 생성 120
5.3.1 스트림 객체 생성 이해 120
5.3.2 스트림 빌더 125
5.4 스트림 연산 이해 129
5.4.1 중간 연산 130
5.4.2 최종 연산 131
5.5 주요 스트림 연산 상세 132
5.5.1 데이터 필터링 132
5.5.2 데이터 정렬 142
5.5.3 데이터 매핑 148
5.5.4 데이터 반복 처리 151
5.5.5 컬렉션으로 변환 154
5.6 기타 스트림 생성 방법 159
5.7 추가 스트림 연산들 161
5.7.1 데이터 평면화 161
5.7.2 데이터 검색 164
5.8 리듀스 연산 166
5.8.1 합계를 구하는 방법들 167
5.8.2 리듀스 연산 이해 170
5.8.3 리듀스 연산 응용 174
5.9 요약 175

6장 병렬 프로그래밍 177

6.1 들어가며 177
6.2 컨커런트 API 179
6.2.1 컨커런트 API 개요 179
6.2.2 컨커런트 API 패키지 180
6.3 Executors 클래스 182

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

6.3.1 Executor 인터페이스 183
6.3.2 ExecutorService 인터페이스 185
6.3.3 ScheduledExecutorService 인터페이스 189
6.3.4 TimeUnit 194

6.4 포크/조인 프레임워크 195
6.5 Future와 CompletableFuture 203
6.6 스트림 병렬 처리 212
6.6.1 스트림 병렬 처리 이해 212
6.6.2 스레드 개수 제어 214
6.6.3 parallel과 sequential 218

6.7 분할 반복 Spliterator 219
6.8 컨커런트 컬렉션 225
6.9 기타 유용한 기능 227
6.9.1 원자적 변수 227
6.9.2 컨커런트 랜덤 숫자 230

6.10 요약 232

7장 파일 I/O(NIO 2.0) 235

7.1 들어가며 235
7.2 개념 이해하기 236
7.3 경로 이해하기 240
7.4 Path를 이용한 경로 관리 241
7.4.1 Path 생성 241
7.4.2 Path 정보 활용 242

7.5 파일 관리 251
7.5.1 Files 클래스 개요 251
7.5.2 파일과 디렉터리 검증 251
7.5.3 파일과 디렉터리 복사 254
7.5.4 파일과 디렉터리 이동 258
7.5.5 파일과 디렉터리 삭제 260
7.5.6 파일 속성 정보 확인 263

7.6 파일 생성, 쓰기, 읽기 266
7.6.1 파일 열기 옵션 266
7.6.2 Files 클래스 이용 267
7.6.3 버퍼 입출력 이용 269

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

7.6.4 스트림 I/O 272
7.6.5 채널과 바이트 버퍼 274
7.6.6 일반 파일과 임시 파일 생성 280

7.7 랜덤 액세스 파일 281
7.8 디렉터리 처리 285
7.8.1 디렉터리 생성 285
7.8.2 디렉터리 목록 조회 287
7.8.3 목록 필터링 288
7.8.4 루트 디렉터리 293

7.9 파일 트리 294
7.9.1 walkFileTree 메서드 295
7.9.2 이전 방식 302
7.9.3 walk와 find 메서드 303

7.10 디렉터리 변경 감지 306
7.11 요약 310

8장 날짜와 시간 313

8.1 들어가며 313
8.2 이전 버전의 문제점 314
8.2.1 멀티 스레드에 취약 314
8.2.2 명명 규칙 317
8.2.3 데이터 불변성 319

8.3 새로운 날짜와 시간 320
8.4 날짜 323
8.5 날짜와 시간 329
8.6 파싱과 포매팅 332
8.7 타임존과 오프셋 343
8.8 날짜와 시간의 차이 347
8.9 Temporal 패키지 351
8.10 과거 버전과의 호환성 360
8.11 요약 361

9장 자바 모듈화 363

9.1 들어가며 363

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

9.2 자바 모듈화 등장 배경 364
9.3 자바 모듈화의 필요성 366
9.3.1 캡슐화의 문제점 366
9.3.2 클래스패스의 문제점 368
9.3.3 한정된 자원 활용 368
9.4 자바 모듈화 이해 369
9.5 자바 모듈 생성 373
9.5.1 모듈 이름 작성 373
9.5.2 모듈 디렉터리 생성 374
9.5.3 자바 코드 작성 375
9.5.4 모듈 기술자 작성 376
9.5.5 모듈 기반 컴파일 및 실행 378
9.6 자바 모듈 의존성과 접근성 382
9.6.1 의존성 정의(requires) 383
9.6.2 접근성 정의(exports) 386
9.6.3 특수한 java.se 모듈 388
9.7 자바 모듈 서비스 390
9.8 링크와 배포 396
9.8.1 jlink의 목표와 활용 396
9.8.2 jlink로 이미지 만들기 398
9.9 요약 402

10장 JShell 도구 403

10.1 들어가며 403
10.2 JShell이란? 404
10.3 코드 스니핏 405
10.4 자동 완성 기능 409
10.5 명령어들 410
10.6 코드 스니핏 수정 415
10.6.1 단축 키 415
10.6.2 외부 편집기 418
10.7 외부 코드 활용 419
10.8 요약 421

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

11장 유용한 새 기능들 423

11.1 들어가며 423
11.2 예외 처리의 발전 424
11.2.1 예외 처리 기본 개념 424
11.2.2 try 문의 개선 427
11.2.3 AutoCloseable 인터페이스 433
11.2.4 catch 문의 문제점과 개선 437
11.3 로컬 변수 타입 추론 439
11.4 반응형 스트림 Flow API 442
11.4.1 Subscriber 445
11.4.2 Publisher 449
11.4.3 Processor 453
11.5 HTTP 지원 강화 456
11.5.1 HTTP 클라이언트 458
11.5.2 웹소켓 467
11.6 Optional 클래스 473
11.6.1 Optional 클래스 개념 이해 476
11.6.2 map과 flatMap을 이용한 연결 481
11.6.3 기본형 Optional 클래스 483
11.7 요약 485

부록A 제네릭 487

A.1 들어가며 487
A.2 제네릭 기본 이해 488
A.3 제네릭과 클래스/메서드 설계 490
A.4 JVM에서 제네릭 처리 499
A.5 와일드카드와 타입 제한 502
A.5.1 extends를 이용한 제한 504
A.5.2 super를 이용한 제한 507
A.6 제네릭 제약 조건 508
A.7 다이아몬드 연산자 509

찾아보기 511

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

서문

2020년, 정확히는 5월 23일에 자바는 25번째 생일을 맞이했다. 새로운 기술이 계속해서 등장하고 빠르게 변화해 가는 IT 산업에서 자바가 25년이 지난 지금까지 여전히 개발자들에게 인기 있고 여러 분야에서 다양하게 쓰이는 이유는 무엇일까? 여러 가지 이유가 있겠지만, 변화에 능동적으로 대응하기 위해 지속적으로 새로운 기능을 추가하고 발전시켜 나가고 있기 때문일 것이다.

다음은 자바가 처음 공식적으로 발표된 1996년(베타 버전은 1995년 5월 23일)부터 자바 11이 발표되기까지의 날짜를 정리한 것이다. 버전 6에서 7로 업그레이드되는 데 약 5년의 시간이 걸린 것을 제외하면 대부분 1년에서 3년 간격으로 업그레이드되어 왔고 자바 9 버전 이후부터는 6개월마다 한 번씩 업그레이드가 이루어지고 있다.

- JDK 1.0: 1996년 1월 23일
- JDK 1.1: 1997년 2월 19일
- J2SE 1.2: 1998년 11월 8일
- J2SE 1.3: 2000년 5월 8일
- J2SE 1.4: 2002년 2월 6일
- J2SE 5.0: 2004년 9월 30일
- Java SE 6: 2006년 11월 11일
- Java SE 7: 2011년 7월 7일
- Java SE 8: 2014년 3월 18일
- Java SE 9: 2017년 9월 21일
- Java SE 10: 2018년 3월 20일
- Java SE 11: 2018년 9월 25일

이렇게 잦은 업데이트와 빠른 대응에도 불구하고 왜 발전이 더디다는 얘기가 나온 것일까? 자바 언어가 정체되고 더 이상 발전이 없을 것이라는 얘기는 자바 6과 자바 7 사이에 나왔다. 자바를 주도하던 썬마이크로시스템이 오라클에 합병되면서 신규 버전 발표가 지연되었고, 자바에 대한 우려가 커지던 시기였기 때

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

문이다. 실제로 신규 버전 발표가 계속 연기되는 상황이었으니 누구나 그렇게 추측할 만했다.

그리고 그 이후부터 자바 관련 서적 발표도 줄어들었고 관련 세미나, 커뮤니티 등의 활동도 급격히 줄어들었다(기업 후원이 줄어든 것도 큰 원인이었다). 이로 인해 자바 개발자라면 가만히 있어도 정보가 쏟아져 들어오던 시대에서 스스로 찾아서 새로운 것을 이해하고 시도해야만 새로운 정보를 찾을 수 있는 환경으로 바뀌었다.

이러한 이유에서인지 많은 개발자들의 지식, 관련된 도서나 컬럼, 예제 코드 등이 자바 6이 기준인 경우가 많다. 10년 전의 자바 코드와 현재의 자바 코드가 크게 달라지지 않고 유지되고 있는 것이다. 그 결과 사용하는 개발 환경과 운영 환경의 버전은 꾸준히 업그레이드가 이루어지고 현재 가장 보수적인 금융 기업들도 자바 8 기반으로 빠르게 변경하고 있지만, 여전히 개발하는 코딩 표준과 API는 자바 6을 벗어나지 못하고 있다.

또 최근 개발자들의 인식도 많이 달라져서 언어를 배우고 새로운 API를 응용하는 것보다 도구를 익히고 언어 기반 위에서 돌아가는 프레임워크(대표적으로 스프링)를 익히는 데 더 중점을 두는 경향도 자바의 새로운 기능을 활용하는 데 걸림돌로 작용한다(요즘 현직에서 일하는 개발자 중에는 자신을 자바 개발자라고 소개하지 않고 스프링 개발자라고 소개하는 경우도 있을 정도다).

이러한 시대적 배경과 개발자들의 인식이 달라진 환경에서 개발자들이 언어의 기본 개념, 발전된 기능 그리고 무엇보다 언어가 시대 환경에 맞게 적응해 가는 것을 직접 익히고 느낄 수 있도록 이 책을 기획하였다. 특히 아직도 10년 전 자바 코딩 스타일이 익숙하고 그것만으로도 충분하다고 생각하는 많은 자바 개발자들에게 다시 한번 언어의 매력에 빠지도록 하는 것이 이 책의 목적이다.

이 책의 전제 조건

이 책의 독자는 자바 버전에 상관 없이 자바에 익숙하다고 가정한다. 즉, 이 책은 입문서가 아니며 기존 자바 개발자들이 새로운 자바 기능에 대해 이해하고 활용할 수 있도록 돕는 것을 목표로 한다. 그러므로 최소한 다음 내용을 이해해야 이 책을 읽을 수 있다.

- 자바 설치와 환경 설정
- 자바 컴파일과 실행

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

- 객체 지향에 대한 이해와 자바 언어를 이용한 객체 지향 프로그래밍
- 자바 언어가 제시하는 기본 문법과 각종 예약어들
- 자바 프로그래밍과 관련된 용어들

자바 6 버전 이후 언어적으로나 API적으로 큰 변화가 생긴 부분 위주로 설명할 것이며 이전 기술과 비교하기 위한 경우를 제외하고 위에 언급한 내용들은 이 책에서는 설명하지 않을 것이다. 경우에 따라서는 새로운 기술의 도입 배경을 설명하기 위해 자바 5의 내용을 설명하는 경우도 있다.

입문 서적이 아니지만 그렇다고 자바 고수만을 위한 책은 아니며 새로운 기술을 소개하고 이것을 잘 활용할 수 있도록 하는 것이 주된 목적이므로 이 목적에 맞게 최대한 쉽고 자세히 그리고 적절한 예제를 통해 설명해 나갈 것이다.

새로운 기술 중 제외하는 것들

자바 6 이후에 업그레이드된 것도 많고 새로 등장한 기능도 많아서 한 권의 책에 그 모든 것을 심도 있게 다루기는 힘들다. 그러므로 이 책에서는 각 버전에서 중점적으로 강조하는 핵심 기능과 개발자들이 알아두면 좋을 흥미로운 내용 위주로 설명할 것이다.

그런 이유로 이 책에서는 다음 내용은 다루지 않는다.

- GUI 기술: Swing, JavaFX 등 GUI 기술은 이 책에서 다루지 않는다.
- 모바일 기술: 안드로이드 등의 모바일 기술에 대해서 언급하지 않는다.
- 엔터프라이즈 기술: 이 책은 Java SE(Standard Edition)를 다루며 Java EE (Enterprise Edition)에 대해서는 언급하지 않는다.
- 보안: 보안이 매우 중요하지만 이 책의 성격과는 맞지 않기 때문에 언급하지 않는다.
- 배포: 배포 역시 다루지 않는다.
- 자바 도구들: JDK에 포함되었거나 없어진 도구들에 대해서 설명하지 않으며 특히 자바 기반의 개발 도구들에 대해서도 설명하지 않는다.

중점적으로 다룰 내용

자바 언어에 혁명적인 변화가 있던 버전은 자바 1.1, 자바 5, 자바 8이다. 나머지 버전들은 주로 개선과 기능 향상에 중점을 두고 있다. 그리고 이 책은 자바 6 이

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

후의 새로운 기능에 중점을 두기 때문에 여러 자바 버전 중 자바 8이 많은 부분을 차지한다.

자바 8은 언어적으로 매우 큰 변화가 있었고 그동안 자바로만 개발한 개발자들이 이해하기 힘든 개념과 문법, API 등이 많이 추가되었다. 때문에 설명은 자바 8의 신기술에 많이 할당하겠지만 각종 예제 및 기술은 최신 LTS(Long Term Support) 버전인 JDK 11을 기준으로 작성할 것이다.

책을 읽기 위한 길잡이

많은 내용을 하나의 책으로 설명하다 보면 항상 지면이 부족함을 느끼며 반대로 자세히 설명하다 보면 내용이 너무 많아져서 책을 읽는 데 피로함을 느낄까 염려스럽다. 이 둘 사이에서 균형을 잡으면서 내용의 줄기를 잡아 나갔고 다음과 같은 원칙을 바탕으로 집필하였다.

- 기본이 되는 내용을 앞 장에 배치하였다.
- 중요한 신기술이라 판단한 내용을 앞 장에 배치하였다.
- 기술적으로 연결되는 내용을 앞뒤로 배치하였다.
- 간단하지만 알아두면 좋은 내용들을 하나의 장으로 묶었다.

람다 표현식과 함수형 프로그래밍에 대해 빠르게 이해하고 싶은 독자는 2장~5장을 읽으면 된다. 특히 2장은 람다 표현식과 함수형 프로그래밍에 대해 직접적으로 설명하고 있는 것은 아니지만 이 기술들이 탄생한 기반을 설명하고 있으니 꼭 먼저 읽고 넘어가길 권한다.

자바의 새로운 API를 익히길 원한다면 7장과 8장을 읽으면 된다. 자바 7에서 새롭게 소개한 파일 NIO와 8에서 크게 개선된 날짜와 시간 API는 기존과 완전히 달라졌다.

자바 9의 새로운 기능을 알고자 한다면 9장과 10장을 읽어야 한다. 특히 자바 9의 핵심인 자바 모듈화에 대해서 설명하고 있다.

아무쪼록 이 책을 통해 자바의 새로운 기능들을 정리하고 즐거운 마음으로 이것들을 습득해 나가면서 새로운 자바의 즐거움에 빠져보길 바란다.

장윤기 (tingle@hanmail.net)

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

1장

P r a c t i c a l M o d e r n **j a v a**

자바의 발전 과정

1.1 들어가며

필자가 자바를 처음 경험한 것이 1996년이었고 어느덧 20년 이상 시간이 흘렀다. 당시에도 이미 많은 프로그래밍 언어가 있었다. 대학에서는 대부분 C와 포트란을 가르쳤으며, 산업 현장에서는 비즈니스 언어로 코볼(COBOL)을 사용했다. 다소 공격적이고 새로운 것을 좋아하는 사람들이 선택하는 언어가 C++이던 시절이다.

필자가 자바를 선택한 이유는 뭔가 새로운 것을 배우고 싶다는 욕망도 컸지만 무엇보다도 인터넷이 발전하면서 인터넷 시대에 적합한 프로그래밍 언어가 자바라고 생각했기 때문이다. 당시 국내에는 관련 서적이 나와 있지 않았고 요즘처럼 인터넷이나 커뮤니티를 통해 정보를 주고받을 수 있던 시절도 아니었다. 궁여지책으로 오로지 제임스 고슬링의 책 《The Java Programming Language》(Addison-Wesley, 1996)를 아마존에서 주문해서 반복해서 읽으면서 배워나갔던 기억이 아직도 생생하다. 자바는 웹브라우저에서 그래픽 출력을 도와주는 애플릿이라는 도구에서 시작해서 서블릿과 JDBC 기술의 도입, 그리고 닷컴 열풍에 힘입어 급속도로 퍼져나갔다. 현재는 다양한 분야에서 가장 널리 사용되는 컴퓨터 프로그래밍 언어로 자리잡았다.

하지만 세월이 흘러 새로운 도전 정신으로 무장한 개발자들이 IT 산업의 주류가 되면서 자바는 낡은 언어, 더는 발전이 없는 언어라는 이미지가 생겼고, “자바는 죽었다”는 소리를 공공연하게 듣기도 했다.

이번 장에서는 다음 내용에 대해서 살펴본다. 이를 통해 자바가 계속 발전하

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

고 변화하고 노력해온 과정과 결과물에 대해 알게 될 것이다.

- 자바 연대표: 자바의 역사를 살펴본다.
- 자바 버전 정책의 변화: 자바 버전에 대한 정책을 알아본다.
- 자바 버전별 새로운 기능: 자바 5 이후 버전에서 새롭게 추가된 기능들에 대해 알아본다.

1.2 자바 연대표

1995년 5월 자바 베타 버전이 썬마이크로시스템즈의 웹사이트를 통해 발표되었을 때 이를 관심있게 지켜본 사람은 별로 없었다. 수많은 컴퓨터 언어가 생겨나고 사라지는 상황에서 하드웨어 업체에서 발표한 개발 언어에 관심을 가질 사람은 많지 않았기 때문이다.

하지만 1년 뒤인 1996년 "한번 쓰면 어느 곳에서나 동작한다(Write Once, Run Anywhere)"는 슬로건하에 1.0 버전을 발표했고, 당시 급속도로 인터넷이 보급되던 환경적 요인에 다양한 하드웨어와 디바이스를 필요로 하는 수요가 맞물리면서 사람들이 자바에 관심을 가지게 되었다. 처음에는 주로 자바 애플릿(Applet)을 통해 동적인 웹 화면을 구현할 수 있다는 점이 큰 매력이었다. 당시 양대 웹브라우저였던 인터넷 익스플로러와 넷스케이프가 브라우저 내부에 자바 가상 머신을 포함시키면서 자바가 더 널리 퍼질 수 있었다.

하지만 얼마 후 액티브엑스(ActiveX)나 플래시 같은, HTML의 기능을 뛰어넘어 다양한 UI를 개발할 수 있는 기술들이 나오면서 애플릿 기반의 자바 애플리케이션은 더 이상 큰 매력이 없게 되었다.

자바가 지금처럼 다양한 분야에 사용되고 이토록 중요해진 것은 애플릿과 같은 UI 기술보다는 서버에서 동작하는 애플리케이션 기술 때문이었다. 이른바 애플릿에 대응하는 서블릿(Servlet)의 등장은 자바가 또 한 번 도약하는 발판이 되었다. 당시 웹 애플리케이션은 주로 C 언어를 이용해서 CGI 프로그램으로 개발했으나 C에 비해 간단하고 명쾌한 서블릿 문법, 그리고 멀티 스레드 기반의 빠른 서비스 처리, 덜 복잡한 메모리 구조 등으로 인해 자바가 빠르게 퍼져나갔다. 뒤이어 등장한 JSP는 자바가 인터넷 시대에 중요한 개발 환경으로 자리잡는 데 큰 역할을 하였다. 이후 마이크로소프트의 닷넷과 치열하게 경쟁했고, 자바는 자바 엔터프라이즈 에디션 기반의 다양한 기술로 무장하고 닷넷과는 서로 다른

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

독자적인 영역을 만들어 오늘날에 이르렀다.[1]

자바의 이러한 발전과 버전 업그레이드에 따른 주요 기술들을 정리하면 그림 1.1과 같다.

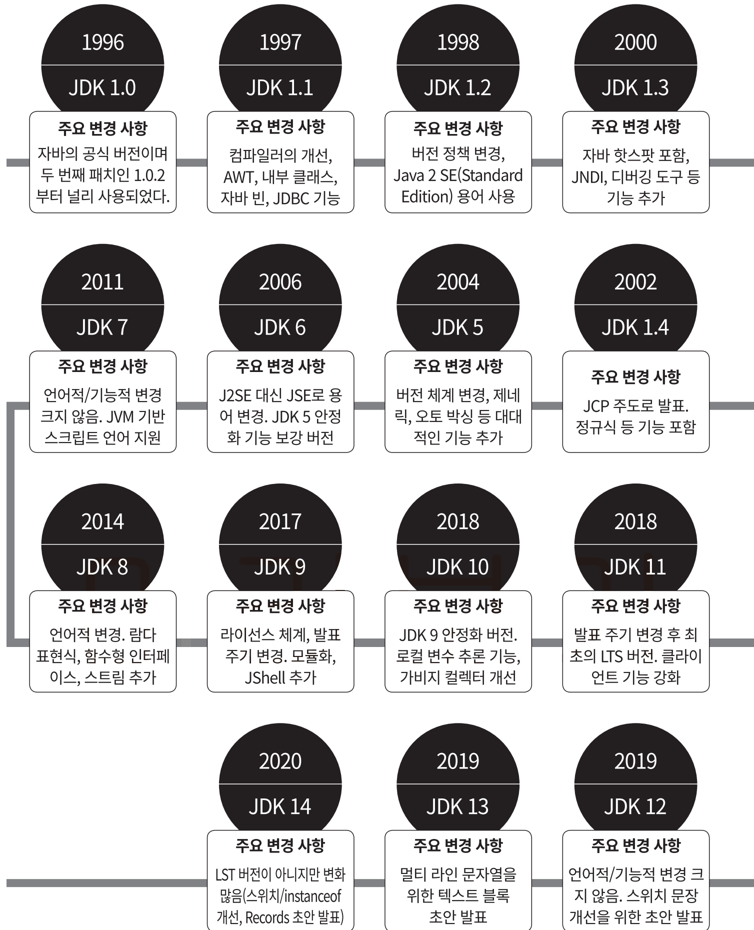

그림 1.1 자바의 역사

그림 1.1의 연대표[2]를 잘 보면 몇 가지 흥미로운 사실을 발견할 수 있다.

첫 번째로 눈여겨볼 것은 1995년 베타 버전이 나온 이후로 이 책을 쓰고 있는 2019년 현재까지 자바는 꾸준히 새로운 버전을 발표하고 있다는 것이다. 자바 언어가 새로운 IT 환경과 변화에 대응하기 위해 계속 노력하고 있다는 사실을

1 90년대 후반에서 2000년대 초반 개발 진영에서의 주된 화제 중 하나가 자바와 닷넷의 비교였다. 많은 프로그래밍 잡지와 인터넷 블로그에서 둘을 비교하는 기사를 많이 썼다. 둘 중 어느 것이 살아남을지에 대한 치열한 논쟁이 있었지만 현재는 각자의 영역에서 자리 잡고 잘 사용되고 있다.

2 *https://en.wikipedia.org/wiki/Java_version_history*를 참고하였다.

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

알 수 있다. 최초로 언어가 세상에 나온 1995년부터 약 25년이 지난 지금까지 꾸준히 노력하고 있다.

두 번째로 추가되는 기능이나 변경되는 내용을 보면 자바라는 언어적인 한계를 넘어서 변화하는 환경에 적응하고자 노력하고 있음을 알 수 있다. 대표적으로 그동안 자바는 객체 지향 언어임을 강조하여 왔고 여기에 반하는 기능이나 기술에 대해서는 다소 배척하는 경향이 있었지만, 자바 8에서 함수형 프로그래밍을 도입함으로써 시대적인 변화와 요청에 대응하였다.

세 번째로 초창기 거의 1년 단위로 업그레이드되던 자바가 2000년대 후반 다소 주춤하였다는 점이다. 버전 6이 발표된 2006년 이후 버전 7이 발표될 때까지 약 5년의 시간이 걸렸다. 바로 이 시기에 IT 업계에서는 자바는 정체되었다는 얘기가 나오기 시작했고 심지어 자바는 죽었다는 말까지 나왔다. 그럼 진정 자바는 죽었나? 아니면 자바는 정체되어서 새로 배우거나 투자할 가치가 없는 언어인가? 다음 절에서 그 내용에 대해 알아보겠다.

1.3 자바는 죽었나?

자바는 죽었다, 혹은 죽어간다는 얘기는 이미 2000년대 초부터 나왔다. 애플릿의 인기가 사라질 때도 나왔고, 닷넷과 ASP가 인터넷 개발 환경의 대안으로 부각될 때도 나왔었다. 그리고 2006년 자바 버전 6의 발표 이후 5년 동안 새로운 버전 발표가 없던 시기에 많은 개발자들이 자바의 정체성에 혼란을 느끼고 고민했던 것도 사실이다. 당시 자바의 소유권을 가지고 있던 썬마이크로시스템즈가 오라클에 인수되면서 위기 의식은 최고조에 이르렀다.

또 오라클 인수 후 처음으로 발표한 버전 7에 대한 실망감은 이러한 의구심을 더욱 증폭시켰다. 5년 만에 발표된 버전치고는 큰 변화가 없었기 때문이다.

그래서 과거 코볼이 그랬던 것처럼 자바가 죽지는 않겠지만 결국 서서히 그 영향력을 잃어갈 것이고, 자바를 대체할 많은 언어와 기술들이 탄생할 것이라는 분석이 쏟아졌다. 때마침 자바스크립트와 같은 스크립트 기반 언어가 유행하고, 빅데이터의 등장과 유행에 따라 자바보다 더 오랜 역사를 가진 파이썬이 대세 언어로 관심을 받기 시작하였다. 새로 출판되는 책들도 자바보다는 스크립트 기반의 언어, 혹은 파이썬을 이용한 머신 러닝이나 빅데이터 처리 등에 초점을 맞추고 있다.

그렇다면 정말 자바는 더 이상 관심을 가지고 배워 나갈 필요가 없는 언어일

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

까? 필자는 동의하지 않는다. 이유는 다음과 같다.

여전히 세계에서 가장 많은 개발자가 사용하고 있는 언어다

사용률과 개발자의 수가 과거에 비해서 서서히 줄어들고 있는 것은 사실이지만 자바는 여전히 세계에서 가장 많은 개발자가 선택하고 있는 언어다. 아직도 많은 시스템과 서비스가 자바를 기반으로 하고 있으며, 이는 개발된 라이브러리와 코드가 가장 많다는 의미이기도 하다.

여전히 발전하고 있다

이 부분이 굉장히 중요하다. 과거 쇠퇴해가거나 사라진 언어들을 보면 잠깐 인기를 얻은 이후에 추가적인 버전 업그레이드나 혁신을 위한 노력이 이어지지 않았다. 하지만 자바는 매번 신규 버전이 나올 때마다 새로운 패러다임을 흡수하려고 노력하였다. 자바 5와 자바 8의 혁신이 여기에 해당한다.

또한 자바 언어의 단점이라고 지적 받는 부분, 즉 다른 언어에 비해 형식을 중요시하고 상대적으로 코딩량이 많으며, 불필요한 코드를 반복적으로 작성해야 하는 문제를 해결하기 위해 오랜 기간 유지해 온 언어적인 원칙을 버리고 람다 표현식과 함수형 인터페이스, 메서드 참조 등을 수용하였다.

그동안 생산된 많은 코드와 라이브러리가 있다

자바는 IT의 대중화와 폭발적인 성장을 이끈 닷컴 시대의 핵심 기술이다. 때문에 자바 언어에 기반한 많은 라이브러리, 프레임워크, 코드들이 무료로(!) 유통되고 있으며 지속적으로 버전 업그레이드가 이루어지고 있다. 이는 매우 중요한 포인트다. 개발자가 언어만으로 모든 것을 만들어낸다는 것은 불가능에 가깝다. 그래서 추가적인 라이브러리와 프레임워크를 사용하게 되는데 현재 자바만큼 많은 무료 도구와 코드를 활용할 수 있는 언어는 없다.

내가 직접 큰 노력을 하지 않아도 계속 버전이 업그레이드되고 기능이 추가되면서 내가 만든 소프트웨어도 같이 업그레이드되는 효과를 얻을 수 있다. 검증받은 라이브러리와 프레임워크를 이용해서 견고한 소프트웨어를 만들어낼 수 있다는 뜻이다. 결과적으로 소프트웨어의 생산성도 높아진다.

이 외에도 여러 가지 이유가 있다. 필자의 동료 중에는 자바를 좋아하고 메인 프로그래밍 언어로 사용하는 이유가 스프링 프레임워크와 인텔리제이 때문이라고 얘기하는 사람도 있다. 자바 개발 환경은 상당히 탄탄하기 때문에 여전히 각광 받고 있다.

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

1.4 자바 버전 정책의 변화

자바는 역사가 오래된 만큼 버전 발표 정책과 규칙에도 조금씩 변화가 있었으나 개발자나 사용자가 느끼기에는 큰 차이가 없었다. 이러한 경향은 자바 8까지 지속되었다. 몇 가지 변화 중 하나는 버전 1.4까지 사용하던 1.x 형태가 아니라 5, 6, 7과 같이 버전 번호 부여 방식을 바꾼 것이다(하지만 JDK를 설치하거나 java -version으로 버전 정보를 확인해 보면 여전히 앞에 1.x가 붙어 있는 것을 확인할 수 있다). 또한 버전 1.2부터는 엔터프라이즈 버전과 마이크로 버전이 추가되었다. 이를 구분하기 위해 J2SE(Java 2 Standard Edition)라는 이름을 부여했지만 자바 버전 6부터 JSE(Java Standard Edition)라는 이름을 사용하고 있다.

전통적으로 소프트웨어의 버전 업그레이드는 2~3년 정도의 긴 기간 동안 혁신적인 개념과 기능을 추가하고 안정화를 거친 후 발표하는 것이 일반적이었다. 하지만 최근에는 짧은 주기로 계속해서 업그레이드를 제공해서 소프트웨어의 기능을 빠르게 보강하는 방식이 추세이다. 예를 들어 마이크로소프트는 윈도우 10이 마지막 버전이며, 6개월마다 한 번씩 메이저 패치를 제공한다고 한다. 이와 유사하게 리눅스 진영의 유명 배포판인 우분투 역시 6개월 단위로 새로운 버전을 제공하고 있으며, 유틸리티 소프트웨어 및 애플리케이션 소프트웨어 역시 이와 유사한 버전 업그레이드 정책을 적용하고 있다.

자바 역시 이러한 추세에 맞춰 몇 년에 한 번 대규모로 업그레이드하기보다 잦은 업그레이드를 통해 개발자의 부담을 줄이고 자바의 신기술을 반영해 나갈 수 있도록 하겠다는 것이다.

6개월에 한 번씩 신규 버전 발표

오라클은 6개월에 한 번씩 신규 자바 버전을 발표하겠다고 밝혔다.[3] 발표하는 특정 월을 명시적으로 선언하지는 않았지만 현재까지의 패턴을 보면 4월과 10월에 한 번씩 버전 업그레이드를 하고 있다.

이와 관련해서 많은 우려도 있다. 과연 자바같이 널리, 그리고 다양하게 사용되는 개발 플랫폼이 6개월마다 버전 업그레이드를 할 수 있을지에 대한 우려도 있고, 새로운 기능 하나를 추가하기 위해서는 JCP(Java Community Process)를 통해 제안을 하고 투표를 하고 테스트해서 최종 적용까지 굉장히 오랜 시간이 걸리는데 과연 6개월마다 버전을 업그레이드하는 것이 적합한지 의문을 제기하

3 *https://blogs.oracle.com/java-platform-group/update-and-faq-on-the-java-se-release-cadence*

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

기도 한다.

또한 소프트웨어를 운영하고 관리해야 하는 엔지니어 입장에서는 기반 기술인 자바 버전의 업그레이드를 굉장히 두려워한다. 그동안 하위 버전에 대한 호환성을 굉장히 강조했지만, 실제로 마이너 패치만 적용해도 소프트웨어에 예상치 못한 문제가 생기는 경우가 자주 있었다. 때문에 메이저 버전을 6개월마다 업그레이드한다는 것은 큰 부담으로 다가온다.

이러한 우려에 대해 오라클 측은 버전이 높아지더라도 실제로 플랫폼적인 변화는 과거 패치 버전 수준 정도의 영향만 있을 뿐 앞자리 숫자가 바뀌는 정도의 큰 변화는 아니라는 입장이다. 그리고 과거처럼 3년에서 5년 간격으로 버전을 발표하면 변동의 폭이 너무 커져서 자바 버전 업그레이드를 주저하게 된다는 의견도 상당히 많았다. 그래서 버전 업그레이드 주기를 짧게 해서 영향을 최소화하겠다는 것이다.

버전에 대한 명명 규칙 변경안 제시

자바 버전을 6개월마다 업그레이드하기로 결정한 이후 오라클에서는 자바의 버전 명명 규칙을 숫자를 하나씩 늘려나가는 형태에서 연도와 월을 조합하는 방식으로 변경할 것을 제안하였다. 예를 들어 2018년 4월에 발표하면 자바 18.04 버전으로, 2018년 10월에 발표하면 자바 18.10 버전으로 명명하는 것이다.

하지만 이러한 명명 규칙 변경 안은 채택되지 않았고, 지금처럼 계속 숫자를 늘려가는 형태를 유지하기로 하였다. 하지만, 잦은 버전 발표로 숫자를 계속 올리면 오히려 버전에 대한 식별성이 떨어질 수 있고 버전의 숫자가 너무 커질 수 있어서 연도와 월을 조합하는 방식이 더 합리적이라는 의견도 많다.[4]

장기 지원 버전 구분

자바는 버전을 한 번 발표하면 다음 버전이 발표될 때까지 1년에서 5년 정도 해당 버전의 버그에 대한 패치 서비스를 제공했다. 패치는 오라클 홈페이지를 통해서 무료로 제공하다가 일정 시간이 지나면 홈페이지를 통한 패치 제공을 중단하고, 별도로 계약한 상용 고객에 한해서 패치를 지원했다. 하지만 이번 발표 주기의 변경과 함께 버그 패치 및 보안 패치에 대한 서비스 주기도 6개월로 변경되었다. 즉, 버전이 발표된 이후 6개월 동안만 버그 패치 및 보안 패치를 지원한다

4 필자의 개인적인 의견은 연도와 월의 조합이 훨씬 의미 있다고 생각한다. 지금이야 9, 10, 11, 12 형태의 버전이 익숙하고 해당 버전에 어떤 것들이 추가되었는지 기억할 수 있지만 계속해서 6개월마다 변화가 있다면 해당 버전 체계가 더 혼란스러울 것이다.

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

는 의미이다. 그리고 특정 버전에 대해서는 LTS(Long Term Support) 버전으로 발표해서 장기간 해당 패치 서비스를 제공한다고 한다.[5]

자바 9부터 버전 정책이 변경된 상태에서 발표된 LTS 버전은 11이다. 그리고 11 버전에 대한 패치 서비스를 3년간 제공한 후 다음 LTS 버전은 17이 된다. 아마도 특별한 사정이 없는 한 17 이후의 LTS 버전은 3년 후인 23이 될 예정이다.

버전 발표 주기의 변화와 함께 라이선스 정책도 변경되었다. 많이 사용하는 오라클 JDK의 버전 11부터는 상업적인 목적으로 사용할 경우 반드시 라이선스를 구매해야 한다. 개인적인 혹은 비상업적인 개발의 목적으로는 여전히 무료로 사용할 수 있지만, 상업성이 조금이라도 포함되어 있다면 상당히 고가의 비용을 지불해야 하기 때문에 오라클 JDK가 반드시 필요하지 않다면 오픈 JDK나 다른 기업에서 제공하는 JDK 사용을 검토해 볼 필요도 있다.

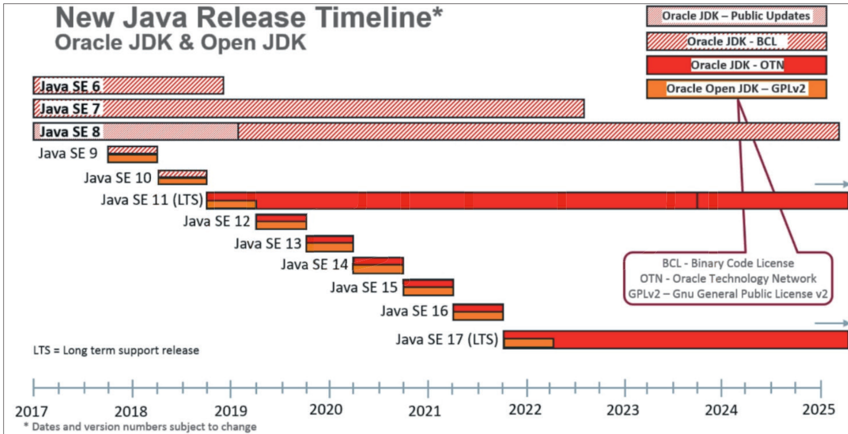

그림 1.2 새로운 자바 발표 모델

1.5 자바 버전별 새로운 기능

자바를 잘 사용하기 위해서는 새로운 기능을 이해하고 이를 활용하는 것이 중요하다. 이번 절에서는 각 버전별로 추가된 중요한 기능들을 정리해 보자. 1.0 버전부터 정리하는 것은 무의미하기에 자바 5부터 이 책을 쓰는 시점의 최신 버전인 자바 14까지 내용을 정리하였다.

5 *https://www.oracle.com/technetwork/java/java-se-support-roadmap.html*

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

1.5.1 자바 5

자바 5부터 자바의 버전 명명 규칙을 1.x에서 5 형태로 변경하였다. 하지만 여전히 자바 개발 도구나 자바 가상 머신을 다운로드 받아서 설치해보면 1.5 형태의 디렉터리가 생성되고 내부적으로 버전을 확인해도 1.5 형태의 번호 체계를 사용하고 있다. 하지만 공식적인 버전은 5이기 때문에 이 책에서는 5라고 표기할 것이다.

자바 5는 이전 버전과 비교해서 언어적으로 큰 변화가 있었다. 그래서 많은 서적과 인터넷에 있는 샘플 소스 코드, 그리고 실제 개발된 소프트웨어 들이 버전 5를 기준으로 만들어졌고 이로 인해 아직까지도 많은 개발자들이 버전 5의 기술에 맞게 훈련되었다.

자바 5에서 새롭게 추가되거나 변경된 내용들을 정리하면 표 1.1과 같다.

항목	내용
제네릭	자바 5의 가장 중요한 신규 기능이다. 제네릭을 이용해서 기존에 컬렉션 프레임워크를 이용했을 때 발생할 수 있는 ClassCastException을 컴파일 시간에 검증할 수 있다. 이러한 컴파일 검증 기능뿐만 아니라 코드에 대한 데이터를 명확하게 해서 가독성을 높이는 결과도 가져왔다.
for 루프 개선	이전까지는 루프를 실행하기 위한 인덱스 변수를 생성해서 기계적으로 값을 늘리거나 Iterator 인터페이스를 이용하였지만, for each 구문을 사용하면 그러한 번거로움 없이 for 루프를 구현할 수 있다. for each를 사용하기 위해서는 데이터 타입이 명확해야 하는 제약이 있어서 제네릭을 사용하거나 형이 명확한 배열을 이용해야 한다.
컨커런트 API	병렬 프로그래밍 혹은 멀티 스레드 프로그래밍을 위해서는 자바 스레드 구조에 대한 해박한 지식을 기반으로 저수준의 프로그래밍을 해야 한다. 하지만 컨커런트 API를 이용하면 비교적 손쉽게 병렬 프로그래밍을 구현할 수 있으며 스레드의 라이프사이클을 관리할 수 있다.
어노테이션	어노테이션을 이용하면 간단한 선언만으로도 메서드나 변수의 행동을 정의할 수 있다. 특히 많이 사용되고 있는 스프링 프레임워크는 어노테이션을 이용한 대표적인 예이다. 이외에도 다양한 개발 프레임워크와 테스트 도구 등에서 어노테이션을 이용해서 개발을 지원하고 있다.
Enum	자바 개발자들은 데이터 구조를 좀 더 손쉽게 정의하고 사용할 수 있는 열거형(Enum) 기능을 원했고 자바 5에서 추가되었다. 이를 이용해 클래스, 인터페이스, 열거형으로 소스 코드를 작성할 수 있게 되었다.
vararg	메서드의 입력 파라미터를 유연하게 가져갈 수 있도록 기능이 추가되었다. 이전까지는 원하는 파라미터 개수만큼 메서드에 모두 기술해야 했지만 vararg 기능이 추가된 이후 데이터 타입만 선언하면 복수의 파라미터를 전달할 수 있게 되었다.

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

오토 박싱/언박싱	자바는 객체 지향 언어를 지향하고 있어서 모든 데이터를 객체화하지만, 예외적으로 기본형인 int, long, float, double 등을 제공하고 있다. 하지만 기본형 타입을 컬렉션에서 활용하거나 제네릭으로 선언해서 사용하기 위해서는 이와 연관된 래퍼 클래스인 Integer, Double 등을 별도로 선언해서 만들어야 했다. 자바 5에서는 오토 박싱/언박싱 기능을 통해 개발자가 기본형 데이터를 래퍼 클래스로 직접 변환하지 않아도 언어 차원에서 자동 변환이 가능하도록 보강되었다. 단, 오토 박싱/언박싱은 성능상 비용이 들어가는 작업이므로 사용 시 주의해야 한다.

표 1.1 자바 5 업그레이드 내용

이외에도 개선된 기능과 추가된 기능이 많지만 대표적인 것 위주로 정리했다. 자바 5는 가장 성공적인 업그레이드로 평가 받고 있다. 언어적인 개선이 이루어졌고 새로 소개된 기능도 많을 뿐 아니라 그 모든 것이 자바 개발자들에게 큰 사랑을 받았다. 그 결과 새로운 기능들이 개발 코드에 빠르게 반영되었다.

1.5.2 자바 6

자바 6에서 새롭게 추가되거나 변경된 내용들을 정리하면 표 1.2와 같다.

항목	내용
G1 가비지 콜렉션	가상 머신의 성능에 가장 크게 영향을 주는 것 중 하나가 가비지 컬렉션이다. G1 가비지 컬렉터는 정확히는 버전 6의 중간 버전부터 추가된 기능으로 기존에는 힙 메모리 영역을 영과 올드 영역으로 분할해서 관리했지만 버전 6부터는 해당 영역 구분을 없애고 1MB 크기의 영역('리전(region)'이라고 부른다)으로 구분해서 메모리를 관리한다. Full GC 발생을 최소화하기 위함이다. 자바 7부터 기본 가비지 컬렉터로 사용한다.
데스크톱 API	데스크톱 API는 자바를 이용해서 UI를 개발할 때 운영체제의 기본 설정값을 기반으로 애플리케이션을 실행하고 출력 등을 편리하게 개발할 수 있도록 해준다. 이 기능을 이용해서 html 파일을 열면 OS에 기본값으로 설정한 브라우저가 실행된다. 예를 들어 리눅스에서는 파이어폭스, 윈도우에서는 엣지가 실행된다.
자바 컴파일러 API	자바 코드에서 직접 자바 컴파일러를 호출할 수 있는 API를 제공한다. 이를 통해 자바 애플리케이션에서 자바 소스 코드를 컴파일하고 실행할 수 있다.

표 1.2 자바 6 업그레이드 내용

자바 6은 자바 5의 안정화 버전이라고 보면 된다. 표 1.2를 보면 새로운 기능은 거의 없고 주로 가비지 컬렉션이나 동기화, 자바 가상 머신의 성능 향상에 중점을 두었다. 이외에도 몇 개의 클래스에 메서드가 추가되고 기능적으로 향상되었지만 특별히 중요한 내용은 없다.

자바 버전 5와 6 사이의 변화는 미미한 수준이다. 버전 6이 발표된 이후에 자

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

바가 더 이상 발전 가능성이 없다거나 정체되었다거나 곧 유행에서 멀어지는 언어가 될 것이라는 얘기가 여기저기서 흘러나오기 시작하였다.

1.5.3 자바 7

자바 6 이후 5년 만에 발표된 자바 7은 개발자들의 기대가 큰 만큼 실망도 큰 버전이었다. 5년이라는 긴 세월 동안 프로그래밍 환경이나 유형이 많이 변화했고, 특히 빠르게 개발하고 배포할 수 있는 스크립트 언어가 대두되면서 이러한 시장 상황에 맞게 자바도 많은 변화가 있을 것이라고 예상했기 때문이다. 하지만 5년이나 지나 발표한 버전치고는 새로운 기능이 많지 않았고, 기존에 제공하는 API를 개선하고 변경하는 정도에 그쳤다. 때문에 개발자들 사이에서 다소 실망스럽다는 의견이 많았다.

자바 7에서 새롭게 추가되거나 변경된 내용을 정리하면 표 1.3과 같다.

항목	내용
File NIO 2.0	자바 7에서 가장 관심 있게 살펴볼 만한 기능으로, 파일 처리를 위한 새로운 기능을 제공한다. 기존에는 다양한 운영체제에서 파일을 처리할 때 예상하지 못한 에러가 발생하거나 기능이 부족한 점 등이 문제로 거론되었다. File NIO는 기능이 풍부하고 처리 속도도 개선되었다. 다만 기존에 사용하던 `java.io.File` 클래스와 개념이 완전히 달라서 새로 공부해야 하는 부담감이 있다.
포크/조인 프레임워크	자바 5에서 처음 등장한 컨커런트 API에 포크/조인 기능이 추가되었다. 이 기능을 이용하면 기존에 멀티 스레드를 생성하는 것에 더해서 라이프 사이클을 관리하고 모니터링할 수 있다.
다이아몬드 연산자	제네릭의 선언 방법이 개선되었다. 변수 선언 시 제네릭의 타입을 이미 선언하였다면 객체 생성 시에는 제네릭 타입을 다시 기입할 필요 없이 다이아몬드 연산자만 기술하면 된다.[6]
try-with-resource	자바로 개발한 애플리케이션에서 메모리 누수가 발생하는 중요한 원인 중 하나가 자원을 사용한 이후 종료시키지 않는 것이다. 이러한 작업은 굉장히 단순하고 반복적으로 이루어지지만 세심하게 작업하지 않으면 치명적인 오류가 생길 수 있다. 자바 7에서는 자원의 종료를 언어 차원에서 보장하고 있으며, 이 기능을 이용할 경우 안정적인 자원 관리뿐만 아니라 소스 코드 작성량도 현저히 줄어든다.
예외 처리	예외 처리를 위해 `try catch` 문장을 작성할 때 경우에 따라서 많은 `catch` 문장을 작성해야 한다. 심지어 `catch` 절 안에 처리하는 로직이 같더라도 반복적인 작업이 발생한다. 자바 7에서는 하나의 `catch` 절에 여러 개의 `Exception`을 처리할 수 있도록 개선했다.

표 1.3 자바 7 업그레이드 내용

6 다이아몬드 연산자는 A.7 "다이아몬드 연산자"에서 자세히 설명한다.

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

대표적으로 완전히 새로운 방식으로 파일을 처리하는 File NIO는 시간을 투자해서 공부할 만한 값어치가 있는 기능이다. 예외 처리 측면에서 `try-with-resource`와 `catch` 구문이 개선되었고 다이아몬드 연산자 등을 통해 코딩의 재미를 느낄 수 있게 되었다.

1.5.4 자바 8

자바가 업그레이드될 때마다 새로운 패키지와 클래스, API가 발표되었지만 배우고 익히는 데 큰 어려움이 없었다. 하지만 자바 8에서 새로 제안한 기술들은 아주 끈기를 갖고 고심해가면서 배우지 않으면 쉽게 익힐 수 없을 정도로 변화의 수준이 파격적이다.

혁신을 추구하는 개발자들에게는 자바의 이러한 변화가 매우 큰 지지와 격려를 받았지만, 그렇지 않은 대다수의 개발자들에게는 새로운 것을 배워야 한다는 부담을 주었다. 심지어 새로운 기능으로 개발된 코드가 자바 개발자들이 해석하고 이해할 수 없을 정도로 문법적으로도 변화가 크다 보니 같은 소프트웨어를 개발하는 개발자 사이에서도 동일한 표준과 코딩 규칙을 만드는 데 애를 먹었다.

그 중에서도 특히 람다와 함수형 프로그래밍을 적용하고 싶지만 해당 코드를 제대로 이해하지 못하고, 이로 인해 잘못된 코딩을 하는 결과로 이어져서 다시 이전 버전으로 되돌리는 현상도 발생했다. 또한 자바 8 이상의 가상 머신을 사용하더라도, 개발하는 코드는 그 이전 버전을 기준으로 하고 있는 경우도 많다.

자바 8에서 새롭게 추가되거나 변경된 내용을 정리하면 표 1.4와 같다.

항목	내용
람다 표현식	자바 언어 역사상 가장 큰 변화가 자바 8의 람다 채용이다. 별도의 익명 클래스를 만들어서 선언하던 방식을 람다를 통해 대폭 간소화할 수 있으며, 함수형 프로그래밍, 스트림 API 그리고 컬렉션 프레임워크의 개선 등에 영향을 주었다. 영향이 큰 만큼 개발자들의 저항이 가장 심한 신기술이기도 하다.
함수형 인터페이스	람다와 더불어 자바 8의 큰 변화 중 하나이다. 함수형 인터페이스는 람다 표현식을 사용할 때 만들어야 하는 하나의 메서드를 가진 인터페이스 생성을 줄여준다. 또한 람다 표현식을 위한 인터페이스 사용 시 표준 가이드 및 의사 소통 용어로도 사용한다.

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/

메서드 참조	기존에는 값과 객체 참조만을 메서드의 인수로 전달할 수 있었지만 자바 8부터는 특정 메서드를 메서드의 인수로 전달할 수 있게 되었다. 궁극적으로는 복잡한 람다 표현식을 메서드 참조로 간략하게 만들 수 있고 재사용성도 높일 수 있으며 람다에 익숙하지 않은 개발자도 소스 코드를 쉽게 해석할 수 있다.
스트림 API	람다 표현식, 함수형 인터페이스 그리고 메서드 참조를 이용한 최종 산출물이 바로 스트림 API이다. 앞서 설명한 3가지 기술이 스트림 API를 위해 만들어진 것이라는 얘기가 있을 정도로 스트림 API는 혁신적인 기능을 제공한다. 스트림 API를 이용하면 기존 컬렉션 프레임워크를 이용할 때보다 간결하게 코드를 작성할 수 있으며 병렬 처리, 스트림 파이프라인 등을 통해 하나의 문장으로 다양한 데이터 처리 기능을 구현할 수 있다.
날짜와 시간 API	자바 8에서 새롭게 선보이는 날짜와 시간 API는 기존 `Date`와 `Calendar` 클래스의 기능 부족과 비표준적인 명명 규칙, 그리고 일관되지 못한 속성값 등의 문제를 해결하기 위해 새롭게 추가되었다. 명명 규칙을 새로 정의하고 각 클래스 간의 역할을 분명히 분배하여 개발자들의 혼란을 최소화하였으며 멀티 스레드 환경에서의 안전성을 보장하고 있다.
인터페이스 개선	자바 8 이전의 인터페이스에는 구현 내용이 없는 `public` 메서드 명세서만 작성이 가능하고 인터페이스를 구현한 클래스에서 상세 내용을 정의해야 했다. 때문에 인터페이스에 메서드를 추가하면 이를 구현한 모든 클래스에 컴파일 에러가 발생했다. 특히 컬렉션 프레임워크와 같이 인터페이스를 기반으로 동작하는 프레임워크는 기능 추가나 개선을 어렵게 하는 가장 대표적인 요인이었다. 자바 8에서는 `default` 키워드를 이용해서 인터페이스에 메서드를 추가하고 직접 내용을 정의할 수 있어서 인터페이스의 메서드 추가로 인한 문제점을 해소할 수 있다.
Optional	자바를 개발하면서 `null` 값 때문에 고민해 보지 않은 개발자가 없을 정도로 `null`은 굉장히 중요한 부분이다. 자바 8에서는 `null` 값을 확인하고 관리할 수 있는 새로운 기능을 제공한다.
CompletableFuture 기능	자바 8에서 멀티 스레드 프로그래밍 시에 중요한 기능 중 하나가 `CompletableFuture`이다. `CompletableFuture`는 기존 `Future` 인터페이스에서 제공하는 기능을 개선하였다.

표 1.4 자바 8 업그레이드 내용

자바 8 이후의 기능 역시 람다 표현식과 스트림 API를 기반으로 계속 발전하고 있다. 이 기술들을 익히지 않으면 새롭게 추가되는 기능 역시 이해하기 어려우므로 반드시 이해하고 익숙해질 필요가 있다.

1.5.5 자바 9

자바 9는 자바 8이 소개된 이후 약 3년만에 발표된 버전으로 자바 8에서만큼 언어적인 혁신이나 변화가 크지는 않지만 자바 개발자들과 커뮤니티에서 요구한 기능들이 많이 채용되었다. 대표적으로 자바 모듈화와 REPL(Read-Eval-Print-

미리보기용 PDF입니다. 구매: http://ebook.insightbook.co.kr/