Hanbit RealTime 130

누구나 쉽게 배우는 스칼라

빠르고 확실하게 스칼라 핵심 익히기

고락윤 지음

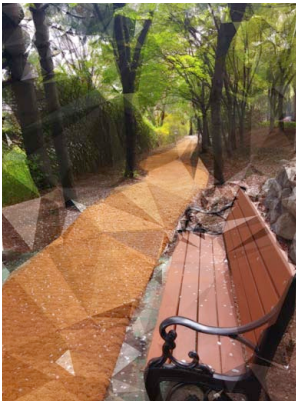
한빛미디어 Hanbit Media, Inc.

누구나 쉽게 배우는 스칼라

빠르고 확실하게 스칼라 핵심 익히기

고락윤 지음

한빛미디어
Hanbit Media, Inc.



표지 사진 **석대진**

이 책의 표지는 석대진 님이 보내주신 풍경사진을 담았습니다.
리얼타임은 독자의 시선을 담은 풍경사진을 책 표지로 보여주고자 합니다.

사진 보내기 **ebookwriter@hanbit.co.kr**

누구나 쉽게 배우는 스칼라 빠르고 확실하게 스칼라 핵심 익히기

전자책 발행 2016년 5월 1일

지은이 고락윤 / **펴낸이** 김태헌
펴낸곳 한빛미디어(주) / **주소** 서울시 마포구 양화로 7길 83 한빛미디어(주) IT출판부
전화 02-325-5544 / **팩스** 02-336-7124
등록 1999년 9월 30일 제10-1779호
ISBN 978-89-6848-822-1 15000 / **정가** 13,000원

총괄 전태호 / **책임편집** 김창수 / **기획·편집** 이상복
디자인 표지/내지 여동일, 조판 방유선
마케팅 박상용, 송경석, 변지영 / **영업** 김형진, 김진불, 조유미 / **제작** 박성우

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 한빛미디어(주)의 홈페이지나 아래 이메일로 알려주십시오.
한빛미디어 홈페이지 www.hanbit.co.kr / **이메일** ask@hanbit.co.kr

Published by HANBIT Media, Inc. Printed in Korea

저자 소개

지은이_ **고락윤** gorakgarak@gorakgarak.com

삼성SDS 소프트웨어 엔지니어. 서울대학교 경영학과를 졸업하고 기술이 가진 무한한 가능성에 매료되어 현재의 길을 선택했다. 오늘도 세상에 기여하는 섹시한 프로그램을 만들어내겠다는 신념으로 공부와 연구를 쉴 틈 없이 반복하고 있다. 스칼라를 이용해 일반인을 위한 금융공학 사이트를 실제 구축 및 운영했으며, 그 밖에도 다양한 프로그래밍 언어를 사용해 웹/모바일 프로젝트를 주도한 바 있다. IT 블로그인 고락가락닷컴(http://gorakgarak.com)을 운영하고 있다.

어렸을 때는 2015년이면 과학이 엄청나게 발전하여 하늘에는 공중을 나는 자동차와 엄청나게 빠른 자기부상열차, 사람같이 행동하는 로보트, 태양열로 발전하는 사이버 의상을 입고 다닐 것이라고 상상하곤 했습니다. 막상 2015년이 되자 그다지 달라진 것은 없고 모바일과 인터넷 세상만 엄청나게 발전하여 사람들이 모두 길거리와 대중교통에서 스마트폰만 쳐다보는 세상이 되었습니다.

이렇게 인터넷이 발전하여 급성장한 상황에는 이에 맞는 새로운 도구가 필요합니다. 어마어마하게 늘어난 접속자와 빅 데이터를 넘어선 거대 데이터를 다루려면, 동시에 빠르게 돌아가는 CPU를 충분히 통제할 수 있는 프로그래밍 구조가 적합합니다.

이러한 면에서 스칼라를 쓰면서 얻을 수 있는 트렌디함과 유용성은 상당히 흥미롭습니다. 멀티코어 CPU에서 끌어쓸 수 있는 성능에 맞춰 프로그래밍의 정교함의 필요성이 점점 두각되는 요즘, 스칼라는 특유의 변경 불가능한 성격 덕분에 스레드 간의 간섭 현상과 여러 프로그래밍 오류를 방지하고 실패에 강한 회복성 좋은 프로그램을 만들어낼 수 있습니다. 또한 스칼라는 다양한 표현식을 이용해 자바의 지루한 코딩을 확연하게 줄어주면서도 자바의 효율성과 성능은 그대로 이용할 수 있습니다.

스칼라는 그 구동 방식 자체가 자바 가상 머신Java Virtual Machine을 기반으로 돌아가며, 자바의 장점을 그대로 지니고 있으면서도 개발자에게 재미와 편리성을 제공해줍니다. 스칼라는 마치 자바의 미래 모습을 미리 맛보는 듯한 느낌을 줍니다. 자바는 사실 보수적인 언어이고 그에 따른 장점이 있는 언어지만, 스칼라는 좀 더 세련되고 젊은 감각으로 자바의 미래를 반영합니다.

그럼에도 현재 스칼라는 유용하지만 접근하기 어려운 언어라는 인식이 있습니다. 워낙 제공하는 표현과 기능이 다양하기 때문에 제대로 배우고 싶으면 외국 사이트를 뒤져가며 하나하나 배워야 하는 실정입니다. 이것이 이 책을 쓰게 된 이유입니다.

이 책은 실제로 필자가 스칼라를 공부하면서 유용했던 점들을 집중적으로 다루었기 때문에 분량이 많지 않더라도 필요한 부분은 모두 담겨 있습니다. 시행착오를 줄이고 최신 트렌드를 반영한 프로그램을 빠르게 만드는 데 도움이 될 것입니다.

스칼라는 단순히 최신 기술일 뿐만 아니라, 프로그래밍에 대해 근본적으로 다시 생각해볼수 있는 기회를 제공합니다. 새롭고 재밌는 세계에 들어온 것을 환영합니다.

대상 독자 및 참고사항

새로운 언어를 배우는 데 흥미를 느끼지만, 과거의 유명한 프로그래밍 언어인 C나 자바보다는 조금 더 미래지향적으로 앞을 내다보고 신선한 언어를 배우고 싶은 사람이 주 독자층입니다.

이 책은 '미래 기술'에 대해서 언급하고 있기 때문에 현재 프로그래밍 언어의 구조나 형태를 알아두면 이해가 빠른 경우가 있습니다. 특히 스칼라는 기존의 프로그래밍 언어의 장점과 불편함을 동시에 고찰하여 만들어진 언어이기 때문에 이에 대한 사전 지식이 있으면 물론 배우기가 편할 것입니다.

그러나 무엇보다 중요한 것은 새로운 것에 대한 호기심 있는 태도입니다. 새로운 언어의 모습을 엿보고 프로그램이 설계되고 만들어지는 과정에 대한 호기심만 충분하다면 이 책을 읽을 때 큰 무리는 없을 것입니다. 잘 모르더라도 일단 해보는 것이 중요합니다! 시작이 반입니다. 더 구체적으로 이 책의 대상 독자는 다음과 같습니다.

스칼라를 배우고 싶지만 적당한 국내 도서가 없어 어려움을 겪는 사람

스칼라가 각광을 받고는 있지만, 아직까지 쉽게 스칼라를 배울 수 있는 국내 서적은 없습니다. 너무 두꺼운 책은 입문서로서 부담이 되고, 어쩔 수 없이 웹에서 외국 자료를 찾게 됩니다. 이 책은 독자와 같은 입장에 있던 필자가 쓴 국내 도서로서, 스칼라의 핵심 기능들을 최대한 심플하게 독자에게 전달합니다.

자바에 한계를 느끼는 사람 혹은 미래 자바의 향기를 맡고 싶은 사람

자바는 보수적인 언어입니다. 각종 예외처리, 자료형 선언, 객체 생성 로직은 개발자를 지치게 만들 때가 있습니다. 자바의 버전이 거듭되면서 이러한 부분에 대한 개선이 계속되고는 있지만, 아예 미래로 점프해서 자바의 미래를 보고 싶은 분들은 이 책으로 쉽게 스칼라를 접해보길 권합니다.

복잡하게 꼬여 어디서 난리가 날지 모를 애플리케이션에 질린 사람

스칼라는 동시성 문제에 굉장히 강한 모습을 보입니다. 기존의 언어 방식은 구조적인

설계로 인해 프로그래머가 예상할 수 없는 코드 간의 꼬임을 방지하기 어려웠습니다. 이 책은 예제와 설명을 통해 동시성에 강한 애플리케이션을 만들기 위한 걸음마를 떼는 데에 도움을 줄 것입니다.

마지막으로, 프로그래머로서 특권을 즐기고 싶은 분들

스칼라는 상당히 많은 표현식과 연산자가 존재합니다. 프로그래머들은 새하얀 도화지에 여러 코드를 통해 창의적인 그림을 그려내는 특권을 가진 사람들입니다. 이러한 사람들에게 기존의 딱딱한 언어들이 제공하는 간단한 연산자로 이루어진 길기만 한 코드는 고통입니다. 스칼라에서 제공하는 도화지는 다른 언어가 제공하는 그것과 같지만, 제공하는 크레파스는 몇천 원짜리 10색 크레파스가 아닌 남들이 모두 부러워하는 50색 크레파스입니다. 이 크레파스로 마음대로 도화지를 칠할 수 있을 것입니다.

이 책과 관련하여 궁금한 점은 다음의 저자 사이트 또는 이메일로 문의해주기 바랍니다.

- 블로그: http://gorakgarak.com
- 이메일: gorakgarak@gorakgarak.com

이 책에서 사용한 소스 파일은 다음 주소에서 받을 수 있습니다.

- http://download.hanbit.co.kr/exam/2822

차례

chapter 1

스칼라 준비하기

이 장에서는 스칼라를 컴파일하기 위한 설치를 준비하고 컴파일을 해보는 과정을 거칩니다. 또한 실제로 콘솔창에 글자를 출력하면서 스칼라에 대한 감을 잡아갑니다.

1.1 스칼라는 어떠한 언어인가?

스칼라는 다른 프로그래밍 언어와 비교해서 그 특징이 굉장히 뚜렷한 언어입니다. 분명 자바와 같은 언어도 잘 발전된 언어이지만, 스칼라는 그에 더해 깊은 철학과 다양한 기능이 추가된 언어라고 할 수 있습니다.

코드의 직관성과 신축성

스칼라는 코드가 짧으면서도 직관적입니다. 기존의 자바 프로그램은 필요 없는 코드가 너무 많이 들어가는 경향이 있었습니다. 엄격한 예외 처리나 객체를 만들 때 세터setter와 게터getter, 생성자, 반복문 등이 그렇습니다. 스칼라에서는 이러한 반복적이고 지루한 작업들을 단 몇 줄의 코드로 끝내버릴 수 있습니다. 다음은 많이 쓰는 자바 1.7에서 숫자 리스트 중 5 이상의 숫자만 골라내는 예제 코드입니다.

```
// 위쪽에 랜덤한 여러 숫자가 있는 List 자료형의 list가 있다고 가정합니다.

List<Integer> otherList = new ArrayList<Integer>();
int number = 0;

for(int i = 0 ; i < list.size() ; i++) {
    number = list.get(i);
    if(number >= 5){
        otherList.add(number);
    }
}
```

자바 코드는 하나의 리스트를 만들어 5 이상의 숫자만 골라내는 과정이 상당히 복잡합니다. 스칼라에서 같은 의미를 뜻하는 코드는 다음과 같습니다.

```
val otherList = list.filter(i => i>=5)
```

정말 간단하게 바뀌었습니다. 자바도 버전업을 거듭하면서 비슷한 개념을 조금씩 받아들이고는 있지만, 그 특유의 보수성으로 인해 위와 같이 깔끔한 코드를 작성하기에는 역부족입니다. 스칼라의 미래지향적 코드를 엿볼 수 있는 대목입니다.

풍부한 표현식과 연산자

스칼라는 태생은 자바지만, 표현 방법이 자바보다 훨씬 다양합니다. 물론 풍부한 표현식을 익히는 데 조금의 시간이 걸릴 수는 있지만, 그에 비해 얻을 수 있는 효과와 재미가 큽니다. 예를 들어 메서드를 살펴볼까요? 만약 animal이라는 객체와 그 안에 구현되어 있는 eat() 메서드를 이용해서 매개변수로 fruit를 주고 싶으면 자바에서는 보통 다음과 같이 합니다.

```
animal.eat(fruit);
```

스칼라는 명사 위주의 프로그래밍 언어가 아니라 동사의 형태이므로 이렇게 쓸 수

있습니다.

```
animal eat fruit
```

물론 자바와 똑같은 형식으로 쓰는 것도 가능합니다. 스칼라에서는 상황에 따라 그에 맞게 표현이 가능합니다.

풍부한 표현식이라는 것은 여기서 끝이 아니라 시작입니다. 스칼라에는 다양한 연산자가 존재합니다. 이 연산자들은 모두 하나하나 메서드로 구현되어 있습니다. 예를 들어 List에는 ++라는 연산자이자 메서드가 있습니다. list1와 list2 리스트는 이 연산자로 쉽게 앞뒤로 붙일 수 있습니다. list1 ++ list2의 결과는 직관적으로 예상할 수 있듯이, list1과 list2를 합친 결과입니다. 이렇게 연산자들이 직관적으로 구현되어 있고, 띄어쓰기를 통해 깔끔하게 표현 가능하기 때문에 스칼라의 유연성과 가독성은 상당히 높은 수준이라고 볼 수 있습니다.

동시성에 강한 언어

스칼라는 다른 주류 언어에 없는 큰 장점이 하나 더 있습니다. 바로 동시성 문제를 항상 염두해두고 해결하는 방향으로 언어가 이루어져 있다는 것입니다. 여러 프로그래밍 언어에는 스레드라는 것이 존재합니다. 스레드는 CPU가 동시에 여러 일을 할 수 있게 해주지만, 복잡하게 프로그래밍될 수밖에 없고, 오류가 일관성 없이 테스트할 때마다 다르게 튀어나오기 때문에 개발 및 유지보수 난이도가 굉장히 높은 편입니다. 심지어 완벽한 스레드 프로그램은 존재하지 않는다는 말이 있을 정도입니다.

만약 한 객체에 여러 스레드가 접근한다면 그 객체의 상태가 정확히 어떠한 위치에 있다고 말하기 어려워집니다. 어떠한 값으로 바뀌었을 수도 있고, 그렇지 않을 수도 있습니다. 간단한 스레드 프로그램을 돌려보면 알겠지만, 동시에 숫자를 세는 단순한 스레드 프로그램조차 그 순서가 어떻게 될지는 알 수 없습니다(실행할 때

마다 순서가 다릅니다).

스칼라는 이러한 현상에 대해 굉장히 조심스럽고 신중하게 접근합니다. 스칼라를 공부하다 보면 'immutable'이라는 단어가 굉장히 많이 등장합니다. '변경불가능'이라는 뜻입니다. 스칼라에서는 많은 부분이 변경불가능 속성을 가지게끔 되어 있습니다. 프로그램 상태를 이리저리 바꾸면서 '슈뢰딩거의 고양이'처럼 놔두기보다는 명확하게 나타내게 프로그래밍할 수 있습니다. 기존의 함수형 프로그래밍으로부터 이러한 동시성 문제의 해결법에 대한 장점을 흡수하면서도 객체지향 프로그래밍의 유연함을 동시에 가지고 있어서 강건한 프로그램을 만들 수 있습니다.

스칼라는 이러한 맥락에서 동시성 프로그래밍에서 굉장히 뛰어난 라이브러리 아카Akka를 우군으로 두고 있습니다. 아카는 동시성 프로그래밍에서의 그 뛰어남으로 주목받고 있는 액터 모델을 이용합니다. 액터 모델은 각각의 액터가 서로 간의 메시지를 통해서만 의사소통을 하고, 액터를 이루는 각종 변수나 속성은 서로 공유하지 않습니다. 아카에 대한 전부를 다루는 것은 이 책의 범위를 벗어나지만, 아카를 이용한 웹 소켓 신기술을 책의 후반부에서 언급할 것입니다.

객체지향 + 함수형 언어

자바는 객체가 아닌 기본 자료형이 존재하며, 연산자는 연산자대로 존재합니다. 또한 메서드 자체도 메서드일 뿐이지 그 자체를 객체처럼 취급할 수는 없습니다. 하지만 스칼라에서는 이 모든 것이 객체입니다. 따라서 함수의 매개변수에 함수 객체를 집어넣을 수도 있고, 변수에 함수를 할당할 수도 있습니다. 실제로 스칼라를 다루다 보면 술어predicate 형태의 함수 매개변수가 필요한 경우가 많습니다. 만약 이러한 기능이 지원되지 않는다면 하나의 자료형을 반환하는 함수를 다시 설계하고 이름도 지어야 하지만, 스칼라는 아예 이러한 형태를 매개변수에 직접 넣어줌으로써 개발자의 수고를 덜어주고 코드를 분석하는 시간을 줄여줍니다.

```
Member.members.filter(_.userId === "admin").delete
```

위 코드는 userId가 admin인 회원을 찾아 정보를 삭제하는 코드입니다. 만약 자바로 구현했다면 if 조건절과 List를 도는 긴 코드를 통해 구현해야 할 것입니다. 하지만 스칼라에서는 filter 메서드 안에 Boolean을 반환하는 함수 형태의 매개변수를 넣어줌으로써 단 한 줄로 원하는 결과를 얻어낼 수 있습니다.

맥락을 읽는 언어

스칼라는 프로그래밍의 맥락을 읽을 수 있는 똑똑한 언어입니다. 자바의 경우 하나하나 명시적으로 자료형을 지정하고 매개변수를 써야 하지만, 스칼라는 필요할 때 implicit 예약어keyword를 사용하면 명시적인 표현을 감춰버릴 수 있습니다.

때로 프로그래밍을 하다 보면 하나의 추가적인 데이터를 보내는 로직을 추가하기 위해 위해 그와 관련된 모든 메서드의 매개변수나 반환 형태를 고쳐야 하는 경우가 있습니다. 자바의 경우에는 하나하나 명시적으로 고쳐주고 디버깅해야 하지만, 스칼라는 필요하다면 알아서 프로그램의 맥락을 읽어서 필요한 곳에 필요한 것을 할당해줍니다.

만약 한국어만을 사용하는 프로그램을 사용자의 국적에 따라 바꿔야 하는 상황이라고 합시다. 따라서 사용자의 브라우저 언어 세팅을 읽어 메서드에 반환 자료형 등을 계속 보내주게끔 바꿔야 한다고 생각해봅시다. 관련 메서드가 많으면 그에 따른 영향도 검사해야 하며 수많은 디버깅을 통해 고쳐야 할 곳이 어마어마하게 많을 것입니다. 만약 암묵적으로 매개변수를 넣어줄 수 있다면 고쳐야 할 곳은 대폭 줄어듭니다. 실제로 그 언어 정보를 받는 곳만 implicit을 선언해줌으로써, 컴파일러가 알아서 언어 정보 매개변수를 찾도록 구현할 수 있습니다.

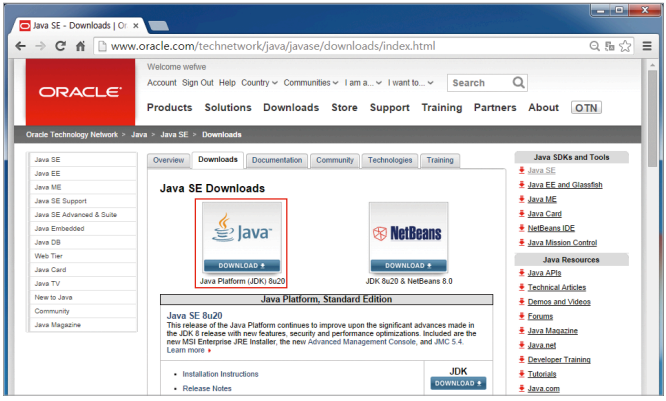
자바와의 뛰어난 연계성

스칼라는 자바와 호환된다는 점이 크나큰 장점입니다. 자바의 대표적인 라이브러리(java.util 패키지 등)는 필요한 경우 언제든지 임포트해서 사용할 수 있으며, 자바로 이루어진 수많은 라이브러리도 물론 이용할 수 있습니다. 또한 스칼라는 자바가 사용하는 JVM(Java Virtual Machine)을 사용합니다. 자바는 상당히 오랫동안 진화를 거듭해왔습니다. 프로그래밍 언어로서 상당히 긴 세월 동안 개량되어왔으며 성능 면에서도 여러 엔터프라이즈급 시스템에 안정적으로 쓰이고 있기 때문에 그 효용성은 인정받고 있다고 할 수 있습니다.

1.2 JDK 설치하기

스칼라는 자바를 기반으로 돌아가는 언어이므로 JDK(Java Development Kit)가 필요합니다. 권장하는 버전은 1.6 이상입니다. JDK는 오라클 자바 공식 사이트에서 다운로드하여 설치할 수 있습니다. URL은 바뀔 수 있으므로 적지 않겠습니다. 구글에서 JDK로 검색하면 최신 버전을 다운로드할 수 있는 페이지로 쉽게 이동할 수 있습니다. 왼쪽의 큰 자바 다운로드 아이콘을 클릭하면 여러 버전을 선택하는 화면으로 이동합니다. 여기서 본인에게 맞는 자바 버전을 다운로드해서 설치합니다.

그림 1-1 자바 다운로드 페이지



chapter 4

클래스 / 객체 / 트레이트

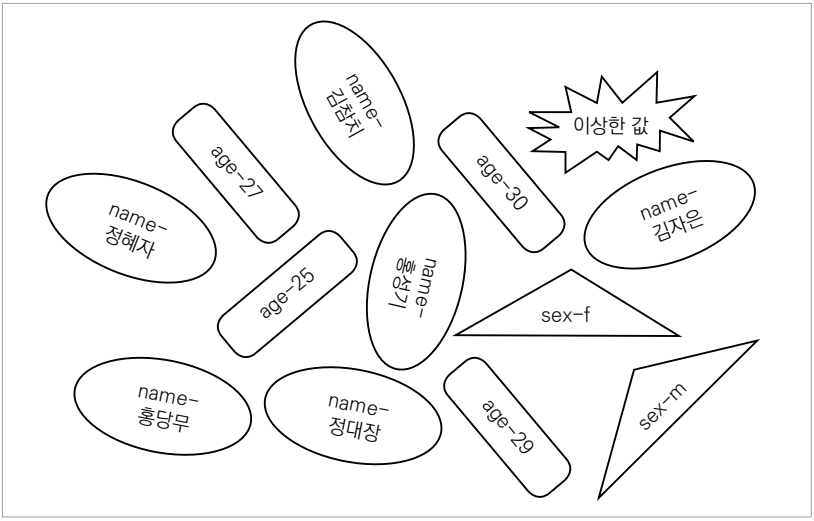
스칼라는 함수형 언어인 동시에 객체지향형 언어입니다. 단순히 객체지향적인 면을 가져다 쓰는 것이 아니라, 오히려 자바보다 훨씬 객체지향적입니다. 자바에서는 연산자와 메서드 자체는 객체라고 할 수 없는 반면, 스칼라에서는 연산자와 메서드를 포함한 모든 것이 객체입니다. 그렇다면 객체지향 측면이 강화된다면 무조건 좋은 것일까요? 이번 장에서는 스칼라의 객체와 이와 관련된 개념들에 대해서 알아봅니다.

4.1 객체지향 프로그래밍

먼저 자바를 비롯해 거의 모든 프로그래밍 언어가 차용한 객체지향에 대한 개념을 조금 짚고 넘어가겠습니다. 객체지향은 현대 프로그래밍 언어를 주름잡고 있는 대세 중 대세로서, 짧게 말하면 프로그래밍에서 쓰이는 요소들의 공통적인 부분을 묶어놓고 하나의 정체성을 부여하는 프로그래밍 방식입니다.

프로그래밍을 하다 보면 사실 비슷한 변수가 자주 쌍으로 등장할 때가 있습니다. 예를 들어 사용자로부터 회원정보를 받는다고 가정합시다. 회원정보에는 name, age, sex 등과 같은 변수가 포함될 것입니다. 이러한 정보들이 다음 그림과 같이 프로그램으로 왕창 흘러들 것입니다.

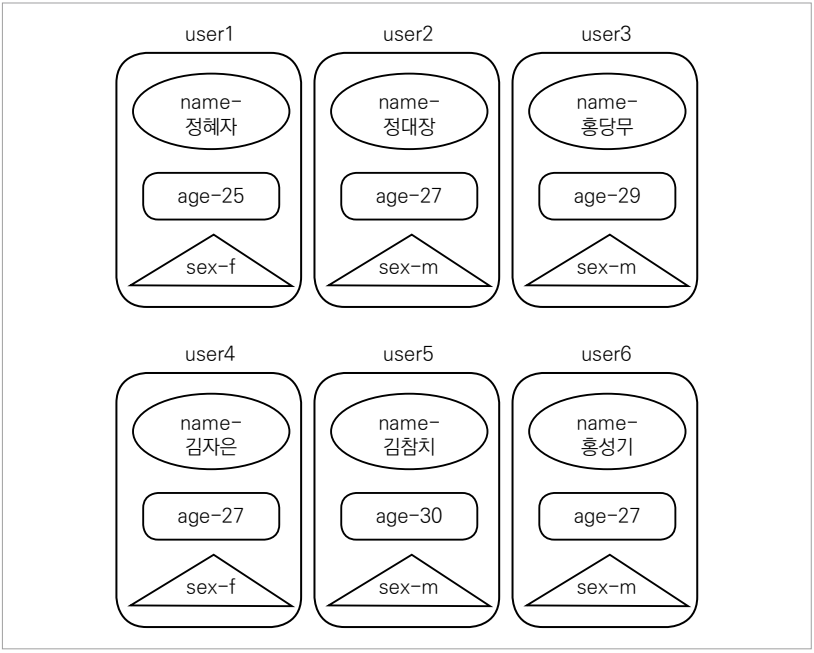
그림 4-1 너저분한 프로그래밍 변수와 값 들



실제 프로그램에는 이보다 더 복잡하고 너저분한 값들이 여기저기에 저장되고 나가게 될 것입니다. 만약 아무런 규칙 없이 필요할 때마다 name과 같은 전역 변수를 생성해서 저장하고, 그 값을 다시 비워서 여기저기 재사용하다 보면 어느 순간 프로그램은 프로그래머의 통제 범위를 넘어가게 될 것입니다. 이 변수가 저기서 쓰이는 것 같기도 하고 아닌 것 같기도 하고, 이걸 바꾸면 저것이 바뀌는 것 같기도 하고 등등 알쏭달쏭한 프로그램이 됩니다. 물론 수백 번의 디버깅을 통해 에러가 나는 상황마다 임기응변을 통해 프로그램을 완성할 수는 있지만, 이러한 프로그래밍은 코드 간 서로 얽혀 있는 것이 너무 많기 때문에 유지보수도 어렵고 재사용은 아예 불가능한 지경에 이를 수 있습니다.

앞의 그림을 잘 보면 무언가 규칙성이 있습니다. 이름을 담는 통인 name이 있고, 나이를 담는 통인 age가 있고, 성별을 담는 sex가 있고, 이상한 에러 값도 존재합니다. 그렇다면 이러한 규칙성을 이용하여 그냥 하나의 큰 통을 만들어버리면 어떠할까요?

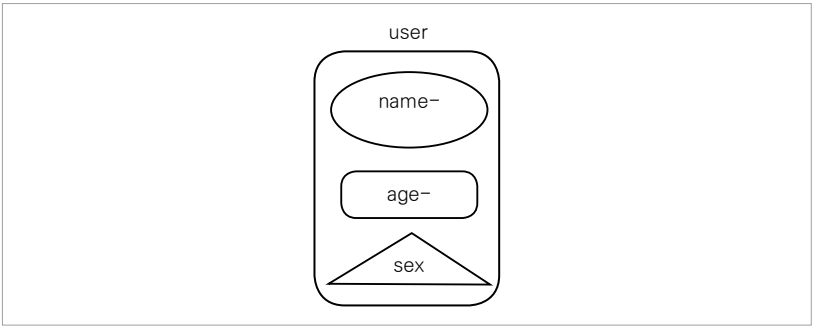
그림 4-2 훨씬 깔끔해진 회원정보



이제 user1부터 uesr6까지 하나씩 정체성이 생겼습니다. 이것이 바로 하나의 객체라고 부를 수 있는 것들입니다. user를 구성하는 공통적 요소인 이름, 나이, 성별을 하나로 묶어 하나의 온전한 인간을 만들어낸 것입니다. 이제 이 큰 통을 들고 다니면서 필요한 user를 불러내 삭제를 하든 수정을 하든 쉽게 할 수 있게 되었습니다. 각각의 객체가 같은 특성을 가지고 있으니 객체 안에 무엇이 들었는지 예상도 가능하고 수정도 쉽게 할 수 있습니다. 또한, 찌꺼기가 생기거나 변수의 재사용으로 인한 코드 간 연관성 문제도 어느 정도 해결되었습니다.

그렇다면 이러한 객체를 하나 생성하고자 할 때는 어떻게 만들어내면 될까요? 이러한 경우에 대비하여 일명 '붕어빵 틀'인 클래스class가 필요합니다.

그림 4-3 클래스



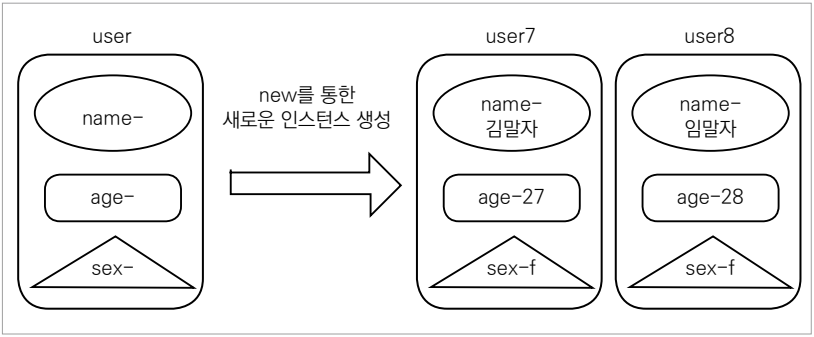
그림에서 볼 수 있듯이, 하나하나의 객체를 생성하기 위한 하나의 틀은 꼭 필요합니다. 이러한 클래스는 class 예약어를 통해 만들어지게 됩니다. 붕어빵처럼 찍어내기 위한 붕어빵 틀입니다. 붕어빵 틀은 붕어빵을 찍어내는 데 사력을 다해야 합니다. 여기서 붕어빵에 해당하는 것은 인스턴스instance라고 합니다. 즉, 클래스를 가지고 일명 인스턴스를 빵빵 찍어내면 클래스는 그 사명을 다하는 것입니다. 스칼라에서는 여기의 회원정보 클래스를 생성하기 위해 다음과 같은 코드가 필요합니다.

```
class User (name: String, age: Int, sex: Char)
```

이러한 클래스를 가지고 얼마든지 인스턴스를 생성해낼 수 있습니다. 예약어 new를 사용하여 다음과 같이 새로운 인스턴스를 찍어낼 수 있습니다.

```
val user7 = new User("김말자", 27, 'f')
val user8 = new User("임말자", 27, 'f')
```

그림 4-4 new를 통한 새로운 인스턴스 생성



객체지향의 기본 개념은 이렇습니다. 클래스라는 붕어빵 틀을 통해 인스턴스라는 붕어빵을 생성해나가면 된다는 것입니다. 공통적인 특성의 모음에 하나의 정체성과 생명을 불어넣는 일이 바로 객체지향 프로그래밍의 핵심입니다.

4.2 클래스와 객체

스칼라에서 객체를 생성하는 방법은 크게 두 가지입니다. 하나는 클래스를 통한 인스턴스화이며, 나머지 하나는 object 예약어를 통해 객체object를 바로 생성하는 것입니다. 쉽게 말하자면 하나는 붕어빵 틀을 이용해 붕어빵을 만들어내는 것이며, 나머지 하나는 처음부터 붕어빵을 수제로 잘 빚어서 이 세상에 유일한 붕어빵을 만들어내는 것입니다. 전자가 복제품이 많을 수 있다는 사실을 제외하면, 둘 모두 실물이며 먹을 수 있습니다.

사실, 클래스 자체가 객체 아닌가라고 생각할 수 있지만, 클래스는 아직 생성되지 않은 객체의 틀에 불과합니다. 추상적인 개념이지 그 자체로 실물이 아닙니다. 보통은 new 예약어를 통해 객체를 생성하게 됩니다.

```
class A // A 클래스는 하나의 틀일 뿐입니다.
val real1 = new A // real1이라는 이름으로 객체가 생성되었습니다.
val real2 = new A // real2라는 이름으로 객체가 또 생성되었습니다.
```

클래스를 만드는 일이 거추장스럽고, 여러 복제품을 만드는 절차가 필요 없을 것 같다면 object 예약어를 통해 바로 객체를 생성할 수 있습니다.

```
object A // A는 그 자체가 객체입니다.
```

객체는 프로그램이 실행될 때 하나의 인스턴스가 만들어지며, 객체를 가지고 다른 인스턴스를 생성할 수는 없습니다. 그냥 생성된 객체를 이용하는 선에서 끝이 나야 합니다.

- 결과적으로 다음의 두 문장으로 클래스와 객체를 정리할 수 있습니다.
- 스칼라 클래스는 new를 통해 계속 인스턴스를 생성할 수 있습니다.
- 스칼라 객체는 싱글턴 객체로서 그 자체가 인스턴스이며 유일합니다.

스칼라의 객체는 싱글턴으로 설계되어 있습니다. 즉, object 이름을 달고 그 안에 여러 메서드를 구현한다면 해당 메서드를 쓰기 위해 굳이 new를 쓸 필요가 없습니다. 자바에서의 대표적 static 메서드 Math.random()처럼, object 이름 뒤에 바로 메서드 이름을 써주어 호출할 수 있습니다. 다만, 자바에서는 한 클래스 안에 static 멤버와 보통의 멤버가 동시에 존재하지만, 스칼라에서는 객체안의 멤버들이 모두 static의 형태를 지닌다고 보면 됩니다. 즉 스칼라에서는 object 예약어를 이용해서 객체를 생성하게 된다면, 바로 object.method() 형태로 호출하는 방식만 있을 뿐입니다.

chapter 6

패턴 매칭

패턴 매칭은 굉장히 엄격하고 제한적으로 쓸 수밖에 없었던 분기문의 훨씬 강력한 변형입니다. 스칼라에서는 자료형에 신경 쓰지 않고 여러 상황에 따라 패턴 매칭을 할 수 있습니다. 이를 이용하면 간결하고 깔끔하게 분기 코드를 처리할 수 있습니다.

6.1 패턴 매칭이란?

패턴 매칭은 특정한 형태의 객체나 값을 받았을 때 그에 따라 코드를 분기해서 실행하기 위해 존재합니다. 스칼라의 패턴 매칭은 아주 예민하고 자세하게 판단 대상이 되는 객체에 대해 판단을 하고 그에 따른 로직을 실행할 수 있습니다.

패턴 매칭을 이해하기 위해서는 간단한 자바의 분기 코드를 먼저 살펴봅시다. 예를 들어 자바에서는 다음과 같이 switch 문을 통해 코드를 분기할 수 있습니다.

```
switch (a) {
  case 1 :  // a가 1일 때 실행되는 로직
  case 2 :  // a가 2일 때 실행되는 로직
  default : // 기본적으로 실행되는 로직
}
```

즉, a에 대한 값을 판단해서 그에 따른 로직을 실행하는 것입니다. 사실 switch의

모든 로직은 조건문으로 다시 구성할 수 있습니다. if else를 이용해서 구성하는 조건문은 분명 굉장히 중요한 기능이지만, 가끔은 중첩되는 괄호 등으로 가독성이 심하게 떨어질 수 있습니다. 그래도 if else 조건문이 많이 쓰이는 이유는 switch 문이 모든 조건문을 대체할 수는 없기 때문입니다. switch 문은 상당히 제약적입니다. 범위에 대한 판단도 불가능하고, 정수나 문자열이 아니면 그에 대한 판단이 불가능합니다.

스칼라의 패턴 매칭은 자바의 switch 분기문에서 훨씬 업그레이드된 버전이라고 할 수 있습니다. 패턴 매칭은 크게 두 가지 면에서 편의를 제공합니다.

먼저 패턴 매칭은 정수나 문자열 판단에 더해, 객체와 객체의 구조를 파악해서 그에 따른 로직을 실행할 수 있습니다. 게다가 문자열의 규칙을 판단하는 정규식도 이용할 수 있습니다. 때로 프로그래밍을 하다 보면, 변수가 어떠한 객체인지에 따라 코드를 다르게 실행할 필요가 있습니다. 혹은 문자열을 입력받아 분석한 후 규칙에 따라 코드를 다르게 실행해야 할 때도 있을 것입니다. 패턴 매칭은 이러한 경우에서 굉장히 깔끔하고 직관적으로 코딩을 할 수 있게 도와줍니다. 만약 패턴 매칭을 적용하지 않는다면 복잡한 instanceof 예약어와 if else 조건문, 그리고 성가신 예외처리 때문에 코드가 몇 배는 길어질 것입니다.

즉 패턴 매칭은, 흔한 switch 문보다 훨씬 진보된 개념입니다. 객체를 판별해내는 데다가, 그 객체를 이루고 있는 변수들을 바로 분기문에서 이용한다는 것은 코드를 줄이고 프로그램을 좀 더 직관적으로 만드는 데 일등공신으로 활약할 수 있습니다.

객체에 대한 판단이 왜 중요한 것일까요? 하나의 대표적인 예로 JSON 같은 데이터 집합을 받을 때를 꼽을 수 있습니다. JSON은 데이터를 구조적으로 담고 있는 하나의 텍스트 뭉치입니다. 이 텍스트 뭉치는 객체 형태로 변환이 필요할 경우가 굉장히 자주 있는데, 인터넷에 돌아다니는 JSON 데이터 구조란 것이 중간중간 구

멍(?)이 나서 전송될 때도 있고, 값이 일부만 있는 경우도 있을 수도 있습니다. 우리는 이러한 데이터의 유형에 따라서 해당 데이터를 버릴지, 아니면 특정한 메서드를 실행해야 할지, 아니면 경고 메시지라도 띄워야 할지 판단해야 합니다. 또한 가끔은 그 객체를 이루고 있는 요소들을 분기문에서 바로 이용할 필요도 있을 수 있습니다. 이러한 로직을 만약 자바로 구현하게 된다면 복잡한 if 문, getClass() 메서드, instanceof 예약어, 객체 안을 탐색하는 for 등으로 코드를 점철해야 할 뿐더러 골치 아픈 디버깅이 기다리고 있을 것입니다.

스칼라의 패턴 매칭은 객체 형태를 보고 코드를 분기하기 때문에 작업이 훨씬 빠르고 직관적입니다. 그저 간단히 switch 문을 대체하여 사용할 수도 있지만, 패턴 매칭을 잘 이해하고 쓴다면 단순한 분기문 이상의 효율을 낼 수 있습니다.

6.2 기본 자료형 패턴 매칭

간단하게 기본 자료형 객체들을 패턴 매칭 해보며 문법을 익혀보겠습니다. 간단한 스칼라 패턴 매칭의 구조는 다음과 같습니다.

```
obj match {
  case x => println("x")
  case y => println("y")
  case _ => println("etc")
}
```

기본적인 패턴 매칭 구문은 이렇게 아주 쉽습니다. obj가 x일 때는 case x => 뒤에 있는 명령을 실행하고, y일 때는 case y => 뒤에 있는 명령을 실행하는 구문입니다. x나 y에 일치하지 않는 나머지 모든 경우에는 case _ => 뒤에 있는 명령을 실행합니다. 언더바는 들어올 수 있는 모든 값을 뜻하는 와일드카드입니다. 언더바를 이용한 기본값 처리가 없을 경우, 일치하지 않는 값을 넣으면 런타임 에러가 나므로 반드시 쓰는 편이 좋습니다.

다음과 같이 패턴 매칭을 함수로 만들어 사용할 수도 있습니다.

예제 6.1 패턴 매칭 함수 예제

```
object Ex_6_1 {
  def main(args: Array[String]): Unit = {
    println(matchFunction(100))
    println(matchFunction("hundred"))
    println(matchFunction(1000))
    println(matchFunction(1000.5))
    println(matchFunction("thousand"))
  }

  def matchFunction(input: Any): Any = input match {
    case 100 => "hundred"
    case "hundred" => 100
    case etcNumber: Int => "입력값은 100이 아닌 Int형 정수입니다."
    case _ => "기타"
  }
}
```

결과 6.1

```
hundred
100
입력값은 100이 아닌 Int형 정수입니다.
기타
기타
```

matchFunction이라는 함수에 match 예약어를 사용해 패턴 매칭 함수로 만들었습니다. 함수 선언부를 보면 자료형에 Any를 썼습니다. 이는 아무 자료형이나 올 수 있다는 뜻입니다. 함수의 반환 값 역시 Any이기 때문에 무슨 값이든 반환할 수 있습니다. 다른 언어들처럼 귀찮은 캐스팅도 필요 없습니다.

예제를 더 자세히 보겠습니다. 먼저 첫 번째와 두 번째 case를 보면, 정수 100이 들어왔을 때 "hundred" 문자열을 반환하고, "hundred"라는 문자열이 들어오

면 정수 100을 반환합니다. 세 번째 case는 Int 자료형을 지정했으므로, 100을 제외한 수 중에서도 정수가 들어왔을 경우만 해당 코드가 실행됩니다. 예를 들어 100.5 같은 소수가 들어오면 결과에서 볼 수 있듯 기본값으로 처리됩니다.

자바는 switch 문에서 여러 자료형을 나눠 판단할 수 없습니다. 만약 위 예제와 같은 함수를 구현하려면 조건문과 캐스팅 등을 통해 구현해야 하는데 상당히 귀찮은 작업이 아닐 수 없습니다. 스칼라의 패턴 매칭은 이렇게 다양한 자료형에 굉장히 강합니다.

6.3 객체 패턴 매칭

패턴 매칭에 대해 '고작 이것뿐이야?'라고 의문을 가질 수도 있는데, 스칼라에서는 '케이스 클래스'를 이용해 객체 또한 패턴 매칭이 가능합니다.

케이스 클래스는 일반 클래스와는 달리 패턴 매칭에 적합한 클래스입니다. 일반 클래스 앞에 case 예약어만 넣어 선언하면 됩니다.

```
case class Person(name: String, age: Int, rank: String)
```

케이스 클래스는 패턴 매칭을 위해 사용할 수 있는 클래스입니다. 보통의 클래스보다 많은 기능을 가지고 있기 때문에 굳이 패턴 매칭을 이용하지 않더라도 case class 예약어로 선언한 다음 자유롭게 써도 무방합니다.

그렇다면 객체를 판별할 때 그 안에 있는 변수들의 값이 같다는 것을 내부적으로 어떻게 판단을 하는 걸까요? 사실, 동일한 값이 들어 있다 하더라도 new를 두 번 이상 호출해서 생성한 객체는 내부적으로는 메모리 주소가 다릅니다. 이 때문에 스칼라는 패턴 매칭에 사용되는 객체가 겉보기에 동일한 경우, 즉 객체가 가지고 있는 멤버 변수의 값이 같을 경우 객체가 같다고 판별해줍니다.

한빛 리얼타임

한빛 리얼타임은 IT 개발자를 위한 전자책입니다.

요즘 IT 업계에는 하루가 멀다 하고 수많은 기술이 나타나고 사라져 갑니다. 인터넷을 아무리 뒤져도 조금이나마 정리된 정보를 찾기도 쉽지 않습니다. 또한, 잘 정리되어 책으로 나오기까지는 오랜 시간이 걸립니다. 어떻게 하면 조금이라도 더 유용한 정보를 빠르게 얻을 수 있을까요? 남보다 조금 더 빨리 경험하고 습득한 지식을 공유하고 발전시켜 나갈 방법으로 전자책은 어떨까요? 세상에는 수많은 종이책과 그 종이책을 그대로 옮긴 전자책이 이미 많습니다. 하지만 전자책에는 전자책에 적합한 콘텐츠와 전자책의 특성을 살린 형식이 있다고 생각합니다.

한빛이 지금 생각하고 추구하는, 개발자를 위한 리얼타임 전자책은 이렇습니다.

1 eBook First – 빠르게 변화하는 IT 기술에 대해 핵심적인 정보를 신속하게 제공합니다

500페이지 가까운 분량의 잘 정리된 도서(종이책)가 아니라, 핵심적인 내용을 빠르게 전달하기 위해 조금은 거칠지만 100페이지 내외의 전자책 전용으로 개발한 서비스입니다. 독자에게는 새로운 정보를 빨리 얻을 기회가 되고, 자신이 먼저 경험한 지식과 정보를 책으로 펴내고 싶지만 너무 바빠서 엄두를 못 내는 선배, 전문가, 고수 분에게는 좀 더 쉽게 집필할 수 있는 기회가 될 수 있으리라 생각합니다. 또한, 새로운 정보와 지식을 빠르게 전달하기 위해 O'Reilly의 전자책 번역 서비스도 하고 있습니다.

2 무료로 업데이트되는 전자책 전용 서비스입니다

종이책으로는 기술의 변화 속도를 따라잡기가 쉽지 않습니다. 책이 일정 분량 이상으로 집필되고 정리되어 나오는 동안 기술은 이미 변해 있습니다. 전자책으로 출간된 이후에도 버전 업을 통해 중요한 기술적 변화가 있거나 저자(역자)와 독자가 소통하면서 보완하여 발전된 노하우가 정리되면 구매하신 분께 무료로 업데이트해 드립니다.

3 독자의 편의를 위해 DRM-Free로 제공합니다

구매한 전자책을 다양한 IT 기기에서 자유롭게 활용할 수 있도록 DRM-Free PDF 포맷으로 제공합니다. 이는 독자 여러분과 한빛이 생각하고 추구하는 전자책을 만들어 나가기 위해 독자 여러분이 언제 어디서 어떤 기기를 사용하더라도 편리하게 전자책을 볼 수 있도록 하기 위함입니다.

4 전자책 환경을 고려한 최적의 형태와 디자인에 담고자 노력했습니다

종이책을 그대로 옮겨 놓아 가독성이 떨어지고 읽기 어려운 전자책이 아니라, 전자책의 환경에 가능한 한 최적화하여 쾌적한 경험을 드리고자 합니다. 링크 등의 기능을 적극적으로 이용할 수 있음은 물론이고 글자 크기나 행간, 여백 등을 전자책에 가장 최적화된 형태로 새롭게 디자인하였습니다.

앞으로도 독자 여러분의 충고에 귀 기울이며 지속해서 발전시켜 나가겠습니다.