1장

첫 번째 양파 껍질 벗기기[1]

이 책의 여기 저기에 “양파 껍질을 벗긴다.”, “첫 번째, 두 번째 양파 껍질”과 같은 표현이 등장한다. 나는 학생들에게 양파 껍질을 벗기듯이 학습하라고 이야기한다. 이 무슨 뚱딴지 같은 소리인가?

나는 한 번에 한 가지 지식을 깊이 있게 학습하는 것에 집중하기보다 다양한 분야의 얕은 지식을 학습한 후 일정 수준이 되면 다음 단계의 깊은 지식으로 서서히 깊이를 더해가라는 의미이다. 예를 들어 웹 애플리케이션을 개발하기 위해 이산수학, 자료구조/알고리즘, 네트워크, 데이터베이스, 운영체제, 자바, 서블릿/JSP, HTML/CSS/자바스크립트를 모두 학습한 후에 접근하기보다는 웹 애플리케이션을 개발하기 위한 최소한의 지식을 습득해 일단 만들어 보는 경험을 해 첫 번째 양파 껍질을 벗긴 후 두 번째 양파 껍질을 벗기기 위해 도전하는 방식으로 학습하라는 조언이다. 이 책 또한 완벽하지는 않지만 이 같은 접근 방식으로 설계하고 진행하려는 시도를 하고 있다.

1 이 글은 넥스트에서 함께 학생들을 가르쳤던 정호영 님이 “초보 웹 개발자를 위하여”(https://github.com/honux77/practice/wiki/web-developer)라는 내용으로 작성한 문서를 기반으로 작성하였다.

□□_□.indd 23 2016-09-02 □□ 9:27:27

이 장은 웹 프로그래밍을 처음 시작하는 개발자가 첫 번째 양파 껍질을 벗길 수 있도록 도움을 주기 위한 가이드 문서를 제공하고 있다. 다음과 같은 경험이 있는 독자라면 바로 2장으로 건너 뛰어도 괜찮다.

- 웹 프론트엔드Front End부터 웹 백엔드Back End까지 자바 기반으로 웹 애플리케이션 개발을 한 번이라도 경험해본 개발자.
- HTML/CSS/자바스크립트/자바/JSP/서블릿/웹 서버/데이터베이스에 대한 각각의 역할을 알고 이 기술 기반으로 웹 애플리케이션 개발을 한 번이라도 경험해본 개발자
- 자바 기반 웹 애플리케이션 개발을 위한 통합 개발 환경, git과 같은 버전 관리 시스템 활용 경험이 있는 개발자
- 개발한 웹 애플리케이션을 맥 또는 리눅스 운영체제에 한 번이라도 배포해 본 경험이 있는 개발자

웹 개발자의 길을 걷기로 마음 먹은 친구들에게 환영의 메시지를 전한다. 개발자의 길을 걷는다는 것에 많은 장애물과 어려움이 있겠지만 그보다 더 큰 즐거움이 있다. 아직까지 웹 애플리케이션을 개발한 경험이 없다면 1장의 가이드 문서를 참고해 기본적인 내용을 학습할 것을 추천한다. 1장은 기술적으로 학습할 내용뿐만 아니라 개발할 때 참고할 웹사이트, 학습 방법 등에 대해서도 다루고 있다. 이 가이드 문서에서 제공하는 모든 내용을 한번에 학습하려고 하기보다 현재 자신의 관심사에 초점을 맞춰 학습 계획을 세운 다음 마음의 여유를 가지고 도전해 볼 것을 추천한다.

1.1 대한민국 IT 개발자 직군의 종류

본격적인 내용을 다루기 전에 현재 대한민국(전세계적으로도 비슷하다) IT 산업에서 개발자 직군으로 어떤 종류가 있는지 살펴보도록 하겠다. 대한민국 산업의 형태로 볼

□□_□.indd 24 2016-09-02 □□ 9:27:27

때 과거에는 대다수가 자바를 기반으로 하는 웹 서버 개발자였는데 현재는 다양한 직군이 늘어나고 있다.

- **웹 백엔드 개발자:** 자바, C#, 루비, 파이썬 등의 언어로 서버 쪽의 로직을 개발하는 역할을 한다. 대부분의 경우 데이터베이스도 잘 알아야 한다. 일부 프론트엔드 개발 작업도 담당하는 경우가 일반적이다.
- **웹 프론트엔드 개발자:** HTML/CSS, 자바스크립트를 주로 사용하며 디자이너와 협업을 하는 개발자이다. 센스와 끈기가 필요하다. 최근에는 jQuery, Angular.js, React.js와 같은 라이브러리도 잘 사용해야 하고, node.js를 통해 웹 백엔드 개발까지 가능하다.
- **모바일 앱 개발자:** 글을 쓰는 요즘은 자바 기반의 안드로이드 개발자와 오브젝티브 C(또는 Swift) 기반의 iOS 개발자가 이 범주에 속한다. 안드로이드 개발자 수요가 더 많은 반면 iOS 개발자가 조금 더 멋있어 보인다.
- **기타:** 시스템 프로그래머, 모바일 게임 개발자, 게임 서버 및 게임 클라이언트 개발자가 있는데 이 직군들은 상대적으로 높은 실력에 비해 구직이 어렵고 대우가 좋지 않다. '이 직군이 매력적이다'라고 생각하면 해외 취업을 권하고 싶다.
- **비개발자 직군:** 중요한 비개발자 직군으로 DBA, 시스템 엔지니어, 빅데이터 전문가가 있다. 빅데이터 전문가가 최근 매우 유망하고, 시스템 엔지니어들도 클라우드가 나타난 이후로 과거에 비해 대우가 좋아진 듯하다. DBA는 실력과 연륜에 따라 연봉의 차이가 매우 크게 나는 직업으로 알려져 있다.

1.2 개발자들에게 유용한 웹사이트들

소프트웨어 분야는 정말 빠르게 발전하고 있기 때문에 모든 지식을 알 수 없다. 빠르게 바뀌는 기술의 흐름을 파악하려면 온라인을 통해 지식을 습득하고 다양한 개발자와 소통해야 한다. 또한 모르는 문제는 검색을 통해 해결할 수 있어야 한다. 그런 측

□□_□.indd 25 2016-09-02 □□ 9:27:27

면에서 개발자들이 좋아하는 사이트를 소개해 본다. 이들 사이트에 방문해서 익숙하게 사용할 수 있도록 노력해 본다.

- **google.com**: 모르는 내용에 대한 검색은 구글을 사용한다. 네이버 지식인, 페이스북 그룹, 빙 등의 사용은 추천하지 않는다. 프로그래밍을 시작하는 개발자 중에 검색을 부끄러워 하는 친구들이 있는데 검색은 절대로 부끄러워 할 일이 아니고 개발자가 키워야 할 능력 중의 하나이다.
- **stackoverflow.com**: 개발자들의 지식인 같은 곳이다. 구글에서 프로그래밍 문제를 검색하면 상당부분 이쪽으로 연결이 된다. 질문을 올리고 답변을 달아보는 경험을 해볼 것을 추천한다.
- **github.com**: 소셜 코딩이라고 부르기도 한다. Git의 사용법과 함께 필수적으로 알아야 한다. GitHub 자체의 사용법을 배우는 게 좋다. 참고로 Git과 GitHub은 전혀 다르다. 전자는 소스코드에 대한 버전관리용 도구이고, 후자는 Git을 지원하는 웹 서비스이다.
- **slideshare.net**: 다양한 기술을 예쁘고 쉽게 볼 수 있다. 깊이가 깊지는 않은 경우가 많지만 그래도 매우 유용하다. 새로운 기술이 무엇인지 보고 싶을 때 우선적으로 검색해 볼 것을 권한다.
- **trello.com**: 칸반이라는 툴을 적용할 수 있는 도구이다. 프로젝트를 진행할 때 프로젝트 관리를 위한 협업 도구로 유용하다.
- **페이스북 그룹, 다양한 온라인 커뮤니티**: 생소한 분야인 소프트웨어를 독학으로 학습하는 것은 쉽지 않다. 특히 학습을 지속하는 데 한계가 있다. 이 같은 한계를 극복할 수 있는 길이 다양한 온라인 커뮤니티이다. 온라인 커뮤니티는 항상 열려있다. 문을 두드리면 된다. 최근 개발자 커뮤니티 그룹의 상당수가 페이스북 그룹으로 활동하고 있으니 페이스북을 통해 개발자들과 친구를 맺고 소통할 것을 추천한다.
- **MOOC 사이트들**: 최근에는 다양한 MOOC 사이트들을 통해서 공짜로 공부할 수 있는 곳이 많아졌다. codecademy, codeschool, khanacademy 등을 통해 신규 언어를 직접 실습으로 익히길 권한다.

□□_□.indd 26 2016-09-02 □□ 9:27:27

1.3 처음에 배워야 하는 것들

프로그래밍을 처음 학습하는 학생들을 보면 프로그래밍 언어를 학습하는 데 어려움을 느끼기도 하지만 컴퓨터를 사용하는 것이 아직 익숙하지 않아 어려움을 느낀다. 예를 들어 소프트웨어 설치/삭제, 터미널 사용 등이 익숙하지 않아 어려움을 겪는다. 이에 대한 해결책은 정도가 없다. 일단 많이 사용해보고 새로운 시도를 두려워하지 않아야 한다. 컴퓨터에 무슨 짓을 하더라도 폭발하지 않는다. 너무 조심 조심 다루지 않아도 된다. 사용하다 문제가 생기면 청소하듯이 깨끗하게 밀어버리고 처음부터 다시 시작하면 된다. 소프트웨어가 하드웨어와 다른 점이 그것이다. 언제든지 초기화하고 다시 시작할 수 있다. 겁내지 마라.

- **맥 / 리눅스 사용법:** 개발자 커뮤니티에 초보 개발자들이 가끔 떡밥으로 '맥을 사용하면 개발이 잘 되나요?'라는 글을 던지고 싸움이 일어난다. 답은 yes이다. 맥에서 맥 OS를 사용하는 것 자체가 개발자에게 도움이 된다. 더불어 리눅스를 배워야 한다. 예쁜 맥을 사서 부트캠프 + 윈도우를 사용하는 건 자제했으면 한다. 맥이 비용 측면에서 부담이 되어 구매하기 힘들다면 아마존 웹 서비스AWS와 같은 클라우드 서비스를 통해 무료로 리눅스 서버를 경험할 수 있다.

- **다양한 프로그래밍 언어:** 쉽고 재미있는 프로그래밍 언어를 배우는 것을 추천한다. 개인적으로 파이썬을 선호하는데, 웹 개발자라면 시작 언어로 자바스크립트와 루비, 모바일 개발자라면 파이썬을 배우면 좋을 것 같다. 나중에 실력이 조금 붙으면 반드시 C 언어와 포인터에 대해 배우길 권한다. 그리고 모바일 개발자라면 Objective-C, Swift도 공부하길 추천한다. 마지막으로 내공이 쌓이면 함수형 프로그래밍 언어(Scala, Haskell, Rust 등)를 배우는 게 좋겠다. 웹 개발자에게 기본적으로 추천하는 언어는 HTML, CSS, 자바스크립트, 루비, 파이썬, 자바, 쉘 스크립트이다.

□□_□.indd 27 2016-09-02 □□ 9:27:27

- **내 전문 분야에 대한 방향성을 결정하자:** 웹 개발자라 하더라도 전문 분야는 앞서 본 것처럼 백엔드 개발자와 프론트엔드 개발자로 나뉘게 된다. 사물의 내면이나 돌아가는 원리를 생각해 보는 걸 좋아한다면 백엔드 개발자가 되는 걸 권장한다. 디자인 감각이 있고 꾸미는 걸 좋아하고 반복 작업도 질리지 않고 잘할 수 있다면 프론트엔드 쪽일 가능성이 높다.

웹 애플리케이션 개발에 대한 학습을 시작하는 단계에서는 가능하면 웹 프론트엔드와 웹 백엔드 모두 학습할 것을 추천한다. 이는 웹뿐만 아니라 모바일, 게임 모든 분야에 해당한다. 자신이 프론트엔드와 백엔드를 직접 경험하기 전에 어느 쪽에 성향이 맞는지 정확하게 파악하기 힘들기 때문이다. 또 다른 이유는 같은 작업을 프론트엔드와 백엔드 모두에서 구현할 수 있는데 각 상황에 따라 프론트엔드에서 구현하는 것이 효과적인 경우가 있고, 백엔드에서 구현하는 것이 효과적인 경우가 있다. 이와 같이 내가 백엔드 개발자라 하더라도 프론트엔드에 대해 알고 있어야 효과적인 해결책을 찾을 수 있고, 반대의 경우도 발생한다. 애플리케이션 개발에 대한 참 맛을 느끼려면 프론트엔드부터 백엔드까지 혼자 힘으로 구현해보는 경험을 하는 것도 큰 의미가 있고, 애플리케이션에 대한 전체적인 큰 그림을 그릴 수 있다는 측면도 있다.
웹 애플리케이션 개발에서 프론트엔드는 HTML, CSS, 자바스크립트 학습에 집중하면 된다. 하지만 웹 백엔드의 경우 자바, 루비, 파이썬, 자바스크립트와 같이 다양한 언어 기반으로 개발이 가능하다. 웹 백엔드의 경우 어떤 언어로 시작할 것인지 추천하기 힘들다. 현재 자신이 목표로 하고 있는 지향점에 따라 달라질 수 있기 때문이다. 국내 대부분의 큰 회사들은 웹 백엔드 언어로 자바를 사용하는 경향이고, 스타트업의 경우 파이썬과 루비를 사용하는 경향이 강하다. 이와 관련해 선택하기 어렵다면 학습 비용 측면만 고려해보면 프론트엔드와 같은 언어인 자바스크립트를 웹 백엔드 언어로 시작하는 것도 좋은 선택이 될 수 있다.

□□_□.indd 28 2016-09-02 □□ 9:27:27

이 책이 다루고 있는 자바 기반 웹 개발자가 되는 것을 목표로 학습한다면 공부의 순서는 (1) HTML (2) CSS (3) 자바스크립트 (4) 자바 (5) 자바 웹 프로그래밍 (6) 데이터베이스 순으로 학습할 것을 추천한다.

1.4 일단 시작해 보자

소프트웨어를 만든다는 거창한 계획을 세우기보다 일단 무엇이라도 만드는 작업을 시작해 보자. 굳이 책을 사지 않아도 온라인으로 학습할 수 있는 좋은 컨텐츠가 많다. 일단 온라인으로 시작해보고 관심이 생기고 더 깊이 있는 학습을 하고 싶다면 본격적으로 도전해 보자.
앞으로 추천하는 실습들은 상당 부분 내용이 중복된다. 하지만 코딩이라는 우리에게 필요한 작업은 머리와 손이 함께 배우는 부분이 많다. 반복이 매우 중요하기 때문에 아는 거 또 나왔네?라고 넘어가지 말고 반복해서 학습을 하길 권한다.

- 제일 먼저 시도해 볼 것으로 1시간 정도 투자해서 http://code.org/learn의 hour of code로 놀아 본다. "안나, 엘사와 함께하는 코드"가 재밌는 것 같다. 컴퓨터에서 중요한 기초 개념인 순차, 반복, 조건문의 개념을 배우기 바란다. 프로그래머는 바보같은 일을 하는 천재적인 사람이다. 컴퓨터는 위대한 일을 하는 바보같은 기계이다. 이 둘은 그래서 찰떡궁합! 출처는 기억이 안 난다.
- 칸 아카데미의 컴퓨터 교육(https://www.khanacademy.org/computing/computer-programming): 자바스크립트의 감을 잡게 해준다. 참고로 이 강의는 자바스크립트 분야의 세계 최고의 대가 중 한 분인 John Resig 님이 만들었다. John Resig 님은 유명한 jQuery도 만들었다.

□□_□.indd 29 2016-09-02 □□ 9:27:27

소프트웨어를 학습하는 좋은 방법 중의 하나는 일단 무엇인가 만들어보는 경험을 한 후 이론적인 개념을 학습하고, 다시 다음 단계의 경험을 하고 이론적인 개념을 학습하는 과정을 반복하는 것이라 생각한다. 앞의 온라인 과정이나 또 다른 과정을 통해 따라하기 식으로 무엇인가 만들어 보는 경험을 했다면 다음 단계는 과정 속에 담겨 있는 이론적인 내용, 소프트웨어 업계에서 사용하는 용어들에 친숙해지는 시도를 해 보자.

앞의 경험을 하면서 등장했던 새로운 용어들이 무엇이며, 이 용어들이 왜 등장했으며, 무엇인지, 프로그래밍을 하는 과정 등에 대한 기본적인 내용에 대해 “기초 튼튼 코드 튼튼 다 함께 프로그래밍”(타니지리 카오리 저/타니지리 토요히사 감수/정인식 역, 제이펍/2016년) 책을 통해 학습해볼 것을 추천한다.

소프트웨어에 대한 개념을 조금 확실하게 잡기 위한 목적으로 하버드 대학교의 CS50 수업(https://cs50.harvard.edu)을 들어보는 것도 추천한다. 기사에 따르면 “정의란 무엇인가?” 강의보다 인기가 많은 수업이었다고 한다. 영어를 잘 못한다고 기죽을 필요는 없다. 한 번에 모든 동영상을 듣기보다 재미삼아 하루에 한편씩 보다보면 나름 재미도 있으며, 우연히 배우는 내용도 생긴다. 특히 우리나라 대학 교육과 비교해 보는 재미도 있다.

1.5 본격적으로 웹 프로그래밍에 도전하기

1.5.1 온라인 강의를 통한 학습

- https://opentutorials.org/course/1688: 생활코딩의 작심 40시간 라이브, 웹 프론트엔드부터 백엔드까지 전체를 경험해 볼 수 있다. 단, 서버측 언어가 자바가 아닌 PHP이지만 웹 애플리케이션 개발의 전체 흐름을 이해할 수 있다.
- https://www.codecademy.com/learn: 웹 애플리케이션 개발과 관련해 필요한 지식을 각 지식별로 학습할 수 있다.

□□_□.indd 30 2016-09-02 □□ 9:27:27

- http://www.w3schools.com/: codecademy와 같이 병행해 학습할 수 있는 곳으로 프론트엔드 지식을 학습할 수 있다.

1.5.2 책을 통한 학습

책은 개인마다 선호하는 스타일이 다르기 때문에 이 책에서 추천하는 책이 반드시 좋은 책은 아니다. 자신의 현재 수준과 맞고, 자신이 선호하는 스타일과 맞는 책이 좋은 책이다. 이 책에서 추천하는 모든 책들은 학습을 하기 위한 참고 자료로 활용했으면 한다. 자신에게 맞는 책을 고르는 것도 능력이고, 연습이 필요하다. 누군가 추천하고, 좋다고 이야기하는 책을 무조건 구매하기보다 자신이 직접 선택할 수 있는 능력을 기를 것을 추천한다.

- "프로가 되기 위한 웹 기술 입문"(고모리 유스케 저/김정환 역, 위키북스/2012): 처음 웹 애플리케이션 개발에서 어려운 점 중의 하나는 기본적으로 사용하는 용어와 기본적인 흐름에 대한 이해가 없기 때문이다. 웹 애플리케이션 개발에 대한 기본적인 용어와 흐름을 이해하는 데 좋다.

- **웹 프론트엔드 학습**[2]
  - "자바스크립트 & 제이쿼리: 인터랙티브 프론트엔드 웹 개발 교과서"(존 두켓 저/장현희 역, 제이펍/2015년): 이 책은 프로그래밍을 처음 시작하는 개발자가 읽어도 괜찮은 책이다. HTML, CSS, 자바스크립트 기초 지식에 대해 잡지 책을 읽는 느낌으로 구성되어 있어 지루함도 적다. 이 책을 통해 HTML, CSS, 자바스크립트에 대한 기초 지식을 쌓은 후 좀 더 깊이 있는 내용을 다루는 책으로 넘어가도 좋겠다.

2 이 3권의 책은 현업에서 프론트엔드 웹 개발을 하고 있는 윤지수, 전용우님이 추천한 책이다. 프론트엔드 웹 개발에 입문하는 독자들에게 추천한다.

□□_□.indd 31 2016-09-02 □□ 9:27:27

- “웹 표준 가이드: HTML5 + CSS3”(존 앨섭 저/김지원, 홍승표 공역, 한빛미디어/2010년): HTML과 CSS에 대해 한 단계 더 깊이 있는 지식을 다룬다.
- “자바스크립트를 말하다 : 가장 간결하면서도 완벽한 자바스크립트 입문서”(악셀 라우슈마이어 저/한선용 역, 한빛미디어/2014년): 자바스크립트에 대해 한 단계 더 깊이 있는 지식을 다룬다.

- **웹 백엔드 학습:** 백엔드 웹 개발로 자바, 파이썬, 루비, 자바스크립트와 같이 많은 언어가 사용되고 있다. 이 책은 다양한 백엔드 개발 언어 중 자바에 대해서만 살펴보도록 하겠다.
  - 국내에 자바 언어를 학습할 수 있는 많은 기본서가 있다. 기본서마다 각자 다른 색깔을 가지고 있기 때문에 자신에게 적합한 책을 골라 학습한다. 인기 있는 책이 무조건 좋은 책은 아니다. 자신의 학습 스타일과 맞는 책이 좋은 책이다. 나는 좀 오래된 책이지만 “자바 프로그래밍”(Jeff Langr 저/권오근 역, 교학사/2005) 책과 같이 무엇인가를 만들어 가면서 관련된 내용을 학습하는 방식을 선호한다. 이 책이 번역 품질이 나쁘지만 책 진행 방식은 마음에 든다.
  - “열혈강의 자바 웹 개발 워크북 – MVC 아키텍처, 마이바티스, 스프링으로 만드는 실무형 개발자 로드맵”(엄진영 저, 이한디지털리/2014): 서블릿, JSP에서부터 시작해 스프링 프레임워크까지 자바 웹 애플리케이션 개발 전반에 대해 따라하기 식으로 구성되어 있다. http://blog.eomjinyoung.com/2014/05/blog-post.html 문서를 보면 각 장별로 동영상 강의도 제공하고 있다. 일단 이 책 한 권으로 자바 기반 웹 애플리케이션 개발에 대한 전반적인 내용을 파악할 수 있다. 하지만 자바 프레임워크를 직접 구현하는 내용도 포함하고 있기 때문에 다소 난이도가 있는 책이다. 따라서 이 책 또한 자신의 색깔과 맞는지 검토할 것을 추천한다. 단순히 따라하기 식의 책이라도 자신의 현재 역량에 맞는 책이 가장 좋은 책이다.

□□_□.indd 32 2016-09-02 □□ 9:27:27

◦ 백엔드 웹 개발자가 반드시 학습해야 할 주제 중의 하나가 데이터베이스이다. 데이터베이스 책으로 추천하고 싶은 책은 "SQL 첫걸음 : 하루 30분 36강으로 배우는 완전 초보의 SQL 따라잡기"(아사이 아츠시 저/박준용 역, 한빛미디어/2015년) 책이다. 데이터베이스에 대한 학습을 시작할 때 유명하지만 엄청 어려운 책을 사 놓은 후 거의 읽지 않은 기억이 난다. 이 책은 SQL문의 작동 원리에 대해 그림을 통해 설명하고 있어 초보 개발자가 읽기에 적합하다.

웹 애플리케이션 개발을 시작할 때 모르는 내용이 많더라도 일단 무엇인가를 만드는 경험을 해볼 것을 추천한다. 모르는 것이 많다는 것이 반드시 나쁜 것은 아니다. 내가 무엇을 모르는 것인지 알 수 있기 때문에 일단 무엇인가를 만드는 경험을 하면서 웹 애플리케이션 개발에 대한 재미를 느껴본 후 모르는 부분에 대해 추가적으로 학습해 나가면 된다. 웹 애플리케이션 개발에 대해 조금씩 눈이 뜨이면 모르는 내용은 점점 더 많아질 것이다. 이와 같은 단계가 정상이다. 알면 알수록 학습할 내용이 더 많아지고, 더 깊이 있게 알고 싶은 욕구가 생긴다. 이 같은 상황이 발생하면 개발자로 잘 성장해 가고 있구나라고 생각하면 된다.

일단 시작하지 않으면 내가 무엇을 모르는지조차 모른다. 내가 모르는 것이 무엇인지에 대해 알고, 다음 단계로 학습할 것이 무엇인지 인식할 수 있다는 것만으로도 한 단계 성장한 것이다. 이 책은 자바 웹 애플리케이션 개발에 대해 시작 단계를 벗어나 한 단계 더 깊이 있는 지식을 학습하고 싶은 개발자를 위한 책이다. 웹 애플리케이션 개발에 대한 기본적인 학습이 끝나고 라이브러리, 프레임워크의 내부가 어떻게 동작하는지 알고 싶은 마음, 내가 만든 코드를 좀 더 깔끔하게 구현하고 싶은 마음, 내가 만든 웹 애플리케이션을 배포하는 과정에 대해 학습하고 싶은 마음이 드는 단계에서 읽으면 학습 효과가 가장 높을 것이다.

□□_□.indd 33 2016-09-02 □□ 9:27:27

1.6 학습 방법

우리가 지금까지 학습하는 방식을 보면 이론적인 기초부터 탄탄하게 다진 후 무엇인가를 만드는 경험을 하는 방식으로 학습을 해왔다. 대표적으로 음악과 미술 같은 경우에도 경험을 통해 음악과 미술에 대한 즐거움을 먼저 느끼는 것이 아니라 이론적인 학습에 치중함으로써 음악과 미술에 대한 즐거움을 느끼는 것이 아니라 거부감을 가지도록 만든다.

물론 기초부터 탄탄하게 쌓는 것도 하나의 학습 방법이지만 경험을 통해 즐거움을 느낀 후 이론적인 학습을 하는 것 또한 좋은 학습 방법이다. 하지만 초, 중, 고등학교를 거치면서 기초부터 이론 위주의 학습 습관이 우리 몸에 자연스럽게 베어 있어 소프트웨어 학습 또한 같은 방식으로 접근하는 것이 일반적이다. 하지만 나는 소프트웨어를 학습하는 데 있어 이런 접근방식이 맞는 학생도 있지만 그렇지 않은 학생이 대부분이라고 생각한다. 사람들은 내가 현재 학습하고 있는 지식이 어느 곳에 활용될 것인지 공감이 될 때 깊이 있게 몰입할 수 있으며, 몸으로 체화할 수 있다. 그런데 어느 곳에 활용될 것인지도 모르는 상태에서 전달하는 지식은 사람을 고통스럽게 할 뿐이며, 쓰레기 지식이 될 수도 있다. 이는 소프트웨어뿐만 아니라 다른 분야를 학습하는 데 있어서도 같다.

지금부터라도 이 같은 접근 방식을 깨고 이론적인 지식은 모르더라도 일단 무엇인가를 만들면서 즐거움을 느껴보는 경험을 하면 어떨까? 특히 양파 껍질을 깨는 첫 번째 단계에서 가장 중요한 것이 프로그래밍을 통해 무엇인가를 만들어 보는 즐거움이다. 이런 즐거움을 느낀 후 두 번째 양파 껍질을 벗겨 나가는 단계에서 이론적인 지식을 조금씩 쌓아나가면 된다. 앞의 "일단 시작해 보자" 절에서 제안한 온라인 자료들을 활용해 일단 무엇이라도 만들어 보자. 일단 무엇이라도 시작해 프로그래밍, 컴퓨터에 대한 두려움을 깨야 한다. 프로그래밍, 컴퓨터가 만만해 보이고, 이 만만한 놈을 활용해 내가 원하는 무엇인가를 만들어낼 수 있다는 것에 대한 즐거움과 흥분된 경험을 해봐야 한다. 이 즐거움과 흥분이 있어야 앞으로 두 번째, 세 번째 양파 껍질을 벗

□□_□.indd 34 2016-09-02 □□ 9:27:28

기면서 경험하게 될 힘든 산을 슬기롭게 넘길 수 있다. 처음부터 기초 이론에 집중하면 프로그래밍과 컴퓨터는 거대한 산처럼 느껴져 넘지 못할 산이라 생각하고 넘을 시도조차 하지 않게 된다.
프로그래밍과 컴퓨터는 정말 만만한 놈이다. 누구나 정복할 수 있다. 정복할 수 없는 것은 두려움 때문이다. 첫 번째 양파 껍질을 깨기 위해 가장 중요한 것은 두려움이 자신을 압도하기 전에 즐거움과 자신감이 충만하도록 자신을 탄탄히 하는 것이다.
"코딩을 지탱하는 기술"(니시오 히로카즈 저/김완섭 역, 비제이퍼블릭/2013)이라는 책을 보면 다음과 같이 학습할 것을 추천한다. 나 또한 공감하는 부분이 많아 인용해 본다.

첫 번째 단계, 필요한 부분부터 흡수한다

책이나 자료 전체가 동일한 정도로 중요하다고 말할 수 없다. 목적이 명확하고, 목적 달성을 위해서 어디를 읽어야 할지 알고 있다면 다른 페이지는 신경 쓰지 말고 바로 그곳을 읽도록 한다.
전체 모두 읽지 않은 것이 께름칙한가? 하지만 좌절하고 전혀 읽지 않는 것보다는 낫다. '전부 읽지 않으면'이라는 완벽주의가 배우고자 하는 동기를 짓누르고 있다면, 버려 버리는 것이 낫다. 동기는 매우 중요하다.
이 전략을 사용하기 위해서는 읽고 싶은 부분이 어디인지 대략적으로 전체적인 구조를 파악하고 있어야 한다. 만약 그게 어려우면 다음 전략인 '대략적인 부분을 잡아서 조금씩 상세화한다.'를 시험해보도록 하자.

두 번째 단계, 대략적인 부분을 잡아서 조금씩 상세화한다

책이나 문서에는 목차가 있다. 목차를 보면 전체 구조를 대략적으로 알 수 있다. 그리고 나서 본문을 속독으로 읽어나간다. 자세히 보지 않고 우선은 소제목이나 강조 부분, 그림과 그림 제목 등을 본다.

□□_□.indd 35 2016-09-02 □□ 9:27:28

소스코드를 읽을 때는 우선 디렉토리 구조와 파일명을 본다. 그리고 파일을 속독으로 읽고 거기서 정의하고 있는 함수나 클래스 이름, 자주 호출되는 함수명 등을 본다.
이 방법들에는 '우선 대략적인 구조를 잡고, 조금씩 상세한 정보로 접근한다'는 공통점이 있다. 이것이 기본 원칙이다.
소스코드에는 다른 방식의 독해 방법이 있다. 디버거의 과정을 사용해서, 실행되는 순서나 호출 계층으로 읽는 방법이다. 이 경우도 동일하게 우선은 대략적인 처리 흐름을 따라가고, 조금씩 깊이를 더해서 함수 안의 처리를 따라가는 것이 중요하다.
이 방법으로 읽어도 정보가 한쪽 귀로 들어와서 한쪽 귀로 나가버리는 느낌을 받는 경우가 있다면, 마지막 방법인 '끝에서부터 차례대로 베껴간다'를 시도해보자.

세 번째 단계, 끝에서부터 차례대로 베껴간다
명확히 '하고 싶은 것', '조사하고 싶은 것'이 없이 '대충 읽으면' 읽은 내용이 뇌를 그냥 스쳐 지나갈 뿐이다. 이런 상태에서 어떻게 배울까를 고민한다고 해도, 판단을 위한 지식 자체가 없기 때문에 무의미하다.
그래서 지식의 밑바탕을 만들기 위해서 교과서를 그대로 베껴 쓴다. 이것이 '베끼기'라 불리는 기술이다. 지식이 없는 상태에서 고민하는 것은 무익하기 때문에 우선 아무것도 생각하지 않고 지식을 복사하는 것이다.
이 이상의 방법은 없다. 저자는 시간을 정해서 '25분간 어디까지 베낄 수 있는지' 도전하는 것을 좋아한다. 분량으로 나누는 것도 좋은 방법이다. 중요한 것은 간격을 적절히 해서 목표를 이루었다는 만족감을 얻을 수 있도록 하는 것이다.

대부분의 학습은 첫 번째 단계(필요한 부분부터 흡수한다)만 잘 습관화해도 지치지 않고 학습을 지속할 수 있다. 처음 시작 단계부터 모든 부분을 완벽하게 이해하고 넘어가겠다는 마음을 버리고 현재 상태에서 이해할 수 있는 부분까지만 이해하고 이해하지 못한 부분은 6개월, 1년이 지난 후 다시 도전하겠다는 마음가짐으로 접근하면 좋겠다. 프로그래밍이 0과 1로 나뉘고, 정확하게 정답이 떨어지는 디지털의 세상이

□□_□.indd 36 2016-09-02 □□ 9:27:28

지만 우리 사람은 아날로그적인 성향을 가진다. 자기 자신을 너무 완벽함 속으로 밀어 넣기보다 한 번에 모두 이해하지 못해도 괜찮다는 너그러운 마음으로 자기 자신을 다독이면서 학습할 때 오랜 시간 동안 지속할 수 있다. 지금까지 가지고 있던 자신의 학습 스타일을 프로그래밍을 학습하면서 깰 수 있다면 이는 프로그래밍을 학습하는 것보다 자신의 삶의 틀을 깨는 더 중요한 학습이 될 것이다.

"코딩을 지탱하는 기술" 책은 위 학습 방법과 관련한 내용 이외에도 프로그래밍 언어가 어떻게 변화 발전해 왔는지에 대해 흥미롭게 풀어내고 있기 때문에 읽어볼 것을 추천한다.

학습에 도움이 될 만한 책을 추천하면서 이 장을 마친다.

- "습관의 힘"(찰스 두히그 저/강주헌 역, 갤리온/2012): 삶에 있어 가장 중요하면서도 힘든 일이 좋은 습관을 만드는 것이다. 배움 또한 좋은 습관을 만들 때 가능하다.
- "이너 게임–배우면 즐겁게 일하는 법"(티머시 골웨이 저/최명돈 역,오즈컨설팅/2006): 배움에 대하여 다른 관점을 느낄 수 있는 책이다. 특히 다른 사람과 비교하고 경쟁하는 것에 집중하지 않고 자기 자신에게 집중함으로써 몰입하는 방법에 대해 다루고 있다.
- "몰입의 즐거움"(미하이 칙센트미하이 저/이희재 역, 해냄/2007): 행복한 삶을 살기 위해 몰입하는 것이 얼마나 중요한 것인지에 대해 다루고 있는 책이다. 몰입하는 것이 왜 어려운지에 대해서도 다룬다. 개발자에게 있어 몰입은 특히 즐겁고도 재미있는 경험이다.

많은 독자들이 첫 번째 양파 껍질을 벗기고 두 번째 양파 껍질을 벗기기 위한 도전을 했으면 좋겠다. 두 번째 양파 껍질을 벗기는 도전에 이 책이 조금이나마 도움이 되었으면 한다.

□□_□.indd 37 2016-09-02 □□ 9:27:28

두 번째 양파 껍질 벗기기

1장에서 다룬 첫 번째 양파 껍질의 목표는 프로그래밍이 무엇인지에 대한 경험을 하고, 웹 애플리케이션을 프론트엔드부터 백엔드까지 개발하는 경험을 하면서 웹 애플리케이션을 구성하는 요소, 역할을 이해하는 데 집중했다. 웹 애플리케이션 개발에 필요한 최소한의 내용을 학습하는 단계이다.

프로그래밍에 도전하는 사람들을 보면 첫 번째 양파 껍질은 소프트웨어에 대한 흥미를 가지면서 나름 재미있는 경험을 한다. 초급자가 학습할 컨텐츠도 많고 따라하기식의 쉽고 재미있는 컨텐츠가 많기 때문이기도 하다. 하지만 첫 번째 단계에서 두 번째 단계로 도전하는 사람보다 포기하는 사람이 더 많다. 따라서 두 번째 양파 껍질에 도전한다는 것만으로도 한 단계의 어려움을 극복한 것이기 때문에 스스로를 대견하게 생각해도 괜찮다.

하지만 첫 번째 양파 껍질을 벗겨냈다고 안심하기에는 이르다. 두 번째 양파 껍질은 첫 번째 양파 껍질보다 훨씬 더 두껍고, 알아야 할 지식도 폭발적으로 늘어나는 경향이 있기 때문이다[3]. 이 두 번째 양파 껍질 단계에서 더 많은 도전자들이 중도 포기하거나, 현재 상태에 만족하고 다음 단계로 성장하지 못한다. 이 두 번째 양파 껍질을 벗는 순간 소프트웨어 개발과 관련한 대략적인 그림도 그릴 수 있으며, 앞으로 무엇을 학습해야 할 것인지에 대한 통찰도 얻을 수 있다. 쉽지 않은 과정이 될 것이다. 두 번째 양파 껍질을 벗는 험난한 여정에 이 책이 작으나마 도움이 되었으면 한다.

이 책의 두 번째 양파 껍질은 1장에서 다룬 첫 번째 양파 껍질과 자연스럽게 연결되는 단계는 아니다. 이 책의 첫 번째 양파 껍질과 두 번째 양파 껍질 사이에는 다음과 같은 단계가 추가되어야 좀 더 자연스럽다. 이 단계는 1장에서 다룬 지식을 기반으로 하여 웹 프론트엔드부터 백엔드까지의 개발에 깊이를 더한 후 일정 기간 동안 웹 애

3 https://www.vikingcodeschool.com/posts/why-learning-to-code-is-so-damn-hard 문서에서 “Phase II: The Cliff of Confusion”을 보면 시작 단계를 벗어나 다음 단계로 도약할 때 학습해야 할 지식의 양이 급격하게 증가하는 것을 알 수 있다.

□□_□.indd 38 2016-09-02 □□ 9:27:28

플리케이션을 개발한 현장 경험을 의미한다. 취업을 통해 경험하든, 스스로 독학을 통해 경험하든 몇 개의 웹 애플리케이션을 자바 기반으로 개발한 경험을 쌓은 후 이 책의 두 번째 양파 껍질에 도전할 때 학습 효과를 높일 수 있다. 이 같은 이유 때문에 이 책을 읽는 가장 적합한 대상 독자를 최소 1년 이상의 현장 경험을 가진 자바 웹 개발자로 정했다.

어쩌면 이 책의 두 번째 양파 껍질은 세 번째 양파 껍질로 표현하고, 두 번째 양파 껍질을 첫 번째 양파 껍질에서 경험한 것에 이론적인 지식을 더하고, 반복적인 웹 애플리케이션 개발을 통해 개발 경험을 쌓는 단계로 보는 것이 더 적합하다. 하지만 이 책은 현장 경험을 쌓는 두 번째 단계를 고려하지 않고, 학습 로드맵 측면에서 학습할 지식 단계만을 고려해 두 번째 양파 껍질로 표현했다.

2장부터 시작되는 이 책의 두 번째 양파 껍질은 첫 번째 양파 껍질에서 한 단계 더 깊이 있는 경험을 하는 것에 초점이 맞추어져 있다. 두 번째 양파 껍질의 학습 목표는 다음과 같다.

- 첫 번째 단계에서 경험한 지식(특히 자바)에 더해 좀 더 깔끔한 코드를 구현하는 코딩 관례, 사용법에 대해 경험한다. 공통 라이브러리와 프레임워크를 직접 구현해 봄으로써 자바를 더 깊이 있게 사용하는 경험, 객체지향 설계와 개발, 리팩토링 경험을 한다.
- 개발자가 학습해야 할 지식은 순수하게 애플리케이션 개발만으로 국한되지 않는다. 개발자는 자신이 개발한 소스코드를 효과적으로 빌드, 배포하고 운영할 수 있어야 한다. 이에 대한 학습이 선행되지 않으면 주객이 전도되어 애플리케이션 개발에 투자하는 시간보다 빌드, 배포, 운영에 투자하는 시간이 더 많아질 수도 있다. 따라서 개발한 소스코드를 개발 서버, 실 서버에 배포하는 경험을 한다. 단순히 배포하는 경험만 하는 것이 아니라 단순, 반복적으로 발생하는 빌드, 배포 과정을 쉘 스크립트를 활용해 자동화하는 과정을 경험한다.

□□_□.indd 39 2016-09-02 □□ 9:27:28

- 두 번째 양파 껍질부터 웹 애플리케이션을 지탱하고 있는 기술, 보안, 성능에 대해서도 서서히 눈을 떠야 하는 시기이다. 내가 개발한 애플리케이션이 컴퓨터 내부에서 어떻게 동작하는지, 웹 브라우저와 웹 서버가 HTTP를 통해 어떻게 데이터를 주고 받는지, 안전한 웹 애플리케이션을 개발하기 위해 알아야 하는 지식에 대한 학습을 시작할 수 있는 동기부여를 할 수 있도록 한다.

위 세 가지 목표가 두 번째 양파 껍질의 핵심 목표이다. 이 책 한 권을 통해 위 세 가지 목표를 모두 달성할 수 없다. 이 책은 이 세 가지 목표 각각에 대해 일부 내용을 포함하고 있다. 따라서 두 번째 양파 껍질을 극복하기 위해 이 책과 같이 읽었으면 하는 책은 다음과 같다.

첫 번째 책은 "이펙티브 자바, 2판"(조슈아 블로크 저/이병준 역, 인사이트/2014년)이다. 외국어를 배우려면 언어의 구조를 알아야 하고(문법), 사물의 이름들을 알아야 하며(단어), 일상적인 필요를 표현하는 관례와 효과적 전달 방법을 알아야 한다(용례). 학교에서는 보통 앞 두 가지만 가르치는 경우가 많다. 프로그래밍 언어에 있어서도 상황은 비슷해서 대부분의 책이 앞의 두 가지 주제에 집중하고 있다[4]. 자바와 관련해 기본적인 문법과 단어를 학습했다면 다음 단계로 코딩 관례와 효과적인 사용법을 학습하기 위한 목적으로 이 책을 읽었으면 한다. 이 책은 자바 개발자가 반드시 읽어야할 필독서 중의 하나이다.

두 번째로 추천하고 싶은 책은 빌드, 배포, 운영과 관련해 알아야 할 도구들에 대한 전반적인 학습을 할 수 있는 "성공으로 이끄는 팀 개발 실천 기술"(이케다 타카후미, 후지쿠라 카즈아키,이노우에 후미아키 공저/김완섭 역, 제이펍/2014년) 책이다. 이 책은 개발한 애플리케이션을 효과적으로 배포하고, 운영하기 위해 알아야 할 도구들과 각 도구들 간의 효과적인 활용 방법에 대해 다루고 있다. 각 도구들에 대해 깊이 있게 다루기보다는 다양한 도구들을 활용해 어떻게 효과적인 개발 환경을 구축할 수

4 이 내용은 "이펙티브 자바, 2판"(조슈아 블로크 저/이병준 역, 인사이트/2014년) 책의 헌사에서 발췌한 내용이다.

□□_□.indd 40 2016-09-02 □□ 9:27:28

있는지에 대한 큰 그림을 그릴 수 있도록 해준다. 이 책을 추천하는 이유는 이 책이 제안하는 개발 환경 구축을 주도하라는 것이 아니다. 현재 자신의 개발 환경과 비교해보고 부족하다고 생각하는 부분이 있으면 하나씩 시도해 봤으면 하는 바람 때문이다. 개발자가 애플리케이션 개발 업무에 더 집중하기 위해서는 프로그래밍과 관련한 학습과 더불어 개발 문화와 개발 환경을 개선하는 시도를 지속적으로 해야 한다.

웹 애플리케이션 개발자가 반드시 학습해야 할 주제 중의 하나가 HTTP이다. 물론 다른 중요한 지식들도 많지만 웹의 근간을 지탱하는 HTTP는 좀 더 깊이 있게 학습했으면 하는 바람으로 세 번째로 추천하고 싶은 책은 "HTTP & Network : 그림으로 배우는 책으로 학습"(우에노 센 저/이병억 역, 영진닷컴/2015)이다.

마지막으로 소개하고 싶은 책은 "IT 인프라 구조 : 그림으로 공부하는"(야마자키 야스시, 미나와 요시코, 아제카츠 요헤이, 사토 타카히코 공저/오다 케이지 감수/김완섭 역, 제이펍/2015년)이다. 내가 이 책을 소개하는 이유는 대부분의 개발자가 프로그램이 동작하는 시스템, 운영체제에 대한 관심이 높지 않기 때문이다. 시스템과 운영체제를 이해하는 것이 생각보다 쉽지 않고 몰라도 소프트웨어 개발이 가능하기 때문이다. 나 또한 그렇게 살아왔다. 하지만 경험이 쌓이고, 한 단계 더 깊이 있는 지식을 학습하다보면 항상 막히는 부분은 시스템, 운영체제와 관련한 부분이었다. 그런데 이 책을 보면서 깊이 있는 지식은 아니라도 시스템과 운영체제 뿐만 아니라 서버 아키텍처까지 전반적인 내용에 대해 이해할 수 있겠다는 생각이 들었다. 또한 이 책은 우리가 흔히 사용하는 많은 이론들에 대해 정리하고, 이 이론들이 어떻게 활용되고 있는지에 대해 설명하고 있다. 그런데 이 이론이 시스템, 운영체제 뿐만 아니라 웹 애플리케이션 개발자가 한 단계 성장하는 데 반드시 알아야 할 내용을 포함하고 있어 필독서로 추천한다.

□□_□.indd 41 2016-09-02 □□ 9:27:28

두 번째 양파 껍질을 벗기 위해 최소한 이 책들이 담고 있는 내용은 학습할 것을 추천한다. 이 몇 권의 책으로 모두 소화하기 힘들다. 두 번째 양파 껍질 단계에서 읽었으면 하는 다른 책들은 각 장을 진행하면서 관련된 내용의 책들을 추천할 계획이다. 각 장에서 추천하는 책까지 모두 학습하기 힘들다면 최소한 위에서 추천한 4권의 책만이라도 학습할 것을 추천한다. 위 4권의 책을 통해 시야를 넓힌 후 더 깊이 있게 학습할 주제를 직접 찾아가는 것도 좋은 선택이다.

□□_□.indd 42 2016-09-02 □□ 9:27:28

2장

문자열 계산기 구현을 통한 테스트와 리팩토링

두 번째 양파 껍질을 벗기기 위한 첫 번째 과정으로 추천하는 학습은 테스트와 리팩토링이다. 테스트와 리팩토링은 개발자가 갖추어야 할 중요한 역량이다. 리팩토링의 즐거움을 한 번 맛보면 프로그래밍하는 즐거움과 재미를 느낄 수 있다.

이 장은 테스트와 리팩토링을 학습하는 것과 더불어 자바 개발 환경에 익숙하지 않은 개발자가 자바 개발 환경을 익히고, 이 책의 실습 진행 방식을 경험하는 것을 목표로 한다.

2.1 main() 메소드를 활용한 테스트의 문제점

소스코드를 구현한 후 정상적으로 동작하는지 확인하는 일반적인 방법은 `main()` 메소드를 활용해 우리가 의도한 결과 값이 정상적으로 출력되는지를 콘솔을 통해 확인하는 것이 일반적이다.

□□_□.indd 43 2016-09-02 □□ 9:27:28

이 과정을 살펴보기 위해 덧셈add, 뺄셈subtract, 곱셈multiply, 나눗셈divide를 구현하는 간단한 사칙연산 계산기 구현 코드를 보면 다음과 같다.

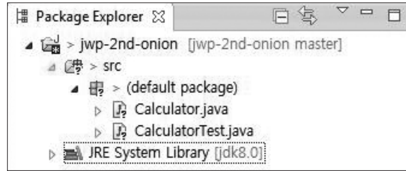
```
public class Calculator {
    int add(int i, int j) {
        return i + j;
    }

    int subtract(int i, int j) {
        return i - j;
    }

    int multiply(int i, int j) {
        return i * j;
    }

    int divide(int i, int j) {
        return i / j;
    }

    public static void main(String[] args) {
        Calculator cal = new Calculator();
        System.out.println(cal.add(3, 4));
        System.out.println(cal.subtract(5, 4));
        System.out.println(cal.multiply(2, 6));
        System.out.println(cal.divide(8, 4));
    }
}
```

계산기 코드는 실제로 서비스를 담당하는 프로덕션 코드production code와 이 프로덕션 코드가 정상적으로 동작하는지 확인하기 위한 `main()`으로 나뉜다. 일반적으로 `main()`은 프로그래밍을 실행하기 위한 목적과 프로덕션 코드가 정상적으로 동작하는지 확인하는 테스트 목적으로 나뉜다. 이 책에서는 `main()`의 목적을 테스트로 생각해 테스트 코드로 부르도록 하겠다.

□□_□.indd 44 2016-09-02 □□ 9:27:28

위 계산기 코드의 첫 번째 문제점은 프로덕션 코드와 테스트 코드(main() 메소드)가 같은 클래스에 위치하고 있다는 것이다. 테스트 코드의 경우 테스트 단계에서만 필요하기 때문에 굳이 서비스하는 시점에 같이 배포할 필요가 없다.

이 문제를 해결하기 위한 첫 번째 단계로 프로덕션 코드(Calculator 클래스)와 테스트 코드(CalculatorTest)를 분리할 수 있다.




```
public class CalculatorTest {
    public static void main(String[] args) {
        Calculator cal = new Calculator();
        System.out.println(cal.add(9, 3));
        System.out.println(cal.subtract(9, 3));
        System.out.println(cal.multiply(9, 3));
        System.out.println(cal.divide(9, 3));
    }
}
```

테스트를 담당하는 별도의 클래스를 추가했지만 main() 메소드 하나에서 프로덕션 코드의 여러 메소드를 동시에 테스트하고 있다. 이는 프로덕션 코드의 복잡도가 증가하면 증가할수록, main() 메소드의 복잡도도 증가하고, 결과적으로 main() 메소드를 유지하는 데 부담이 된다. 이 같은 문제를 해결하기 위해 다음과 같이 테스트 코드를 각 메소드별로 분리할 수도 있다.

```
public class CalculatorTest {
    public static void main(String[] args) {
        Calculator cal = new Calculator();
        add(cal);
        subtract(cal);
```


□□_□.indd 45 2016-09-02 □□ 9:27:28

```
        multiply(cal);
        divide(cal);
    }

    private static void divide(Calculator cal) {
        System.out.println(cal.divide(9, 3));
    }

    private static void multiply(Calculator cal) {
        System.out.println(cal.multiply(9, 3));
    }

    private static void subtract(Calculator cal) {
        System.out.println(cal.subtract(9, 3));
    }

    private static void add(Calculator cal) {
        System.out.println(cal.add(9, 3));
    }
}
```

하지만 이 또한 최종적인 해결책이 될 수 없다. 그 이유는 개발자가 프로그래밍하는 과정을 살펴보면 된다. 우리는 프로그래밍을 할 때 한 번에 메소드 하나의 구현에 집중한다. 클래스가 가지고 있는 모든 메소드에 관심이 있는 것이 아니라 현재 내가 구현하고 있는 메소드에만 집중하고 싶다. 하지만 위 테스트 코드는 `Calculator` 클래스가 가지고 있는 모든 메소드를 테스트할 수밖에 없다. 그렇다고 테스트하는 메소드만 남기고 다른 메소드를 주석처리하는 것 또한 불합리한 작업이다.

`main()` 메소드를 활용한 위 테스트가 안고 있는 다른 문제점은 테스트 결과를 매번 콘솔에 출력되는 값을 통해 수동으로 확인해야 한다는 것이다. 위와 같이 로직이 간단한 경우에는 결과 값을 쉽게 예측할 수 있다. 하지만 로직의 복잡도가 높은 경우, 구현을 완료한 후 한 달이 지난 시점이라고 생각해 보자. 이 때 프로덕션 코드의 복잡한 로직을 머릿속으로 계산해 결과 값이 정상적으로 출력되는지 일일이 확인해야 하는 번거로움이 있다.

□□_□.indd 46 2016-09-02 □□ 9:27:28

`main()` 메소드를 활용한 테스트의 이 같은 문제점을 해결하기 위해 등장한 라이브러리가 JUnit이다. JUnit은 내가 관심을 가지는 메소드에 대한 테스트만 가능하다. 또한 로직을 실행한 후의 결과 값 확인을 프로그래밍을 통해 자동화하는 것이 가능하다. JUnit을 활용해 문제를 해결하는 방법에 대해 살펴보자.

2.2 JUnit을 활용해 main() 메소드 문제점 극복

JUnit(http://junit.org)은 단위 테스트 프레임워크 중 하나이다. JUnit은 앞 절에서 언급한 것처럼 `main()` 메소드의 한계를 해결해 줄 수 있는 도구이다. 자바 진영에 많은 라이브러리들이 있지만 JUnit 만큼 사용하기 쉽고 학습 비용이 낮은 라이브러리는 많지 않으니 큰 부담 없이 접근해도 된다.

책보다 동영상을 통해 학습하는 것이 익숙한 독자는 다음 동영상을 통해 이클립스 개발 환경하에서 프로젝트 추가, 실행, 이클립스 활용 팁, JUnit 사용법을 학습할 수 있다.

https://youtu.be/vrUGCv80xqI 이클립스 활용, JUnit 3 버전 사용방법
https://youtu.be/tyZMdwT3rIY JUnit 4 버전 사용방법

동영상에는 이 절에서 다루는 모든 내용을 포함하고 있다.

2.2.1 한 번에 메소드 하나에만 집중

프로젝트에 JUnit 라이브러리를 추가한 후 `main()` 메소드로 구현한 `CalculatorTest`를 삭제하고 새로운 `CalculatorTest`를 추가한다. JUnit을 사용하려면 라이브러리를 추가해야 한다. JUnit 라이브러리를 추가하는 방법을 모르는 독자는 앞의 동영상을 참고해 추가하고 다음 과정을 실습하기 바란다.

□□_□.indd 47 2016-09-02 □□ 9:27:28

JUnit은 다음 예제 소스와 같이 테스트 메소드에 `@Test` 애노테이션annotation을 추가한다.

```
import org.junit.Test;

public class CalculatorTest {
    @Test
    public void add() {
        Calculator cal = new Calculator();
        System.out.println(cal.add(6, 3));
    }
}
```

위와 같이 `@Test` 애노테이션을 추가한 후 이클립스 메뉴에서 Run 〉 Run As 〉 JUnit Test를 실행하면 `add()` 메소드를 실행할 수 있다. 다음 단계로 뺄셈subtract에 대한 테스트 메소드도 다음과 같이 추가할 수 있다.

```
import org.junit.Test;

public class CalculatorTest {
    @Test
    public void add() {
        Calculator cal = new Calculator();
        System.out.println(cal.add(6, 3));
    }

    @Test
    public void subtract() {
        Calculator cal = new Calculator();
        System.out.println(cal.subtract(6, 3));
    }
}
```

위와 같이 JUnit 기반으로 테스트 코드를 구현하면 `CalculatorTest` 클래스가 가지는 전체 메소드를 한번에 실행할 수도 있으며, `add()`, `subtract()` 메소드 각각을 실

□□_□.indd 48 2016-09-02 □□ 9:27:28

행할 수도 있다. 이와 같이 각각의 테스트 메소드를 독립적으로 실행할 수 있기 때문에 현재 내가 구현하고 있는 프로덕션 코드의 메소드만 실행해 볼 수 있다. 즉, 다른 메소드에 영향을 받지 않기 때문에 내가 현재 구현하고 있는 프로덕션 코드에 집중할 수 있는 효과를 얻을 수 있다.

2.2.2 결과 값을 눈이 아닌 프로그램을 통해 자동화

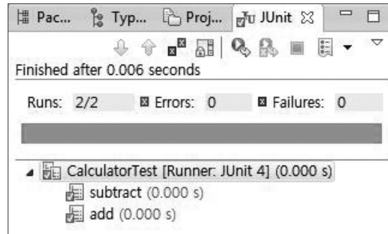
`main()` 메소드의 두 번째 문제점은 실행 결과를 눈으로 직접 확인해야 한다는 것이다. JUnit은 이 같은 문제점을 극복하기 위해 `assertEquals()` 메소드를 제공한다. 위 `CalculatorTest` 클래스에 `assertEquals()` 메소드를 적용하면 다음과 같다.

```
import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class CalculatorTest {
    @Test
    public void add() {
        Calculator cal = new Calculator();
        assertEquals(9, cal.add(6, 3));
    }

    @Test
    public void subtract() {
        Calculator cal = new Calculator();
        assertEquals(3, cal.subtract(6, 3));
    }
}
```

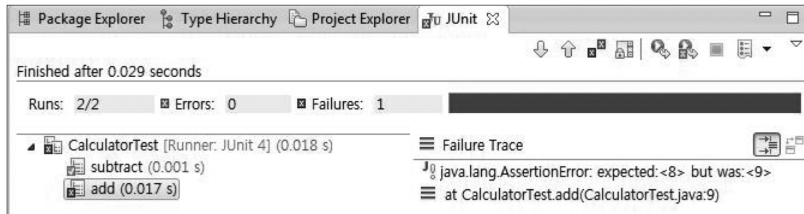
`assertEquals`는 `static` 메소드라 `import static`으로 메소드를 `import`한 후 위와 같이 구현할 수 있다. `assertEquals()` 메소드의 첫 번째 인자는 기대하는 결과 값(expected)이고, 두 번째 인자는 프로덕션 코드의 메소드를 실행한 결과 값(actual)이다. `assertEquals()` 메소드는 `int`, `long`, `String` 등 다양한 데이터 타입 지원이

□□_□.indd 49 2016-09-02 □□ 9:27:28

가능하다. 위와 같이 구현한 후 JUnit Test를 실행하면 다음과 같은 결과 화면을 확인할 수 있다.



앞에서 구현한 `CalculatorTest` 클래스가 가지는 두 개의 테스트가 모두 성공할 경우 초록색 바가 뜨면서 테스트가 성공했음을 알려준다. 만약 `add()` 메소드의 `assertEquals()` 메소드를 "`assertEquals(8, cal.add(6, 3));`"와 같이 기대하는 값을 9에서 8로 변경한 후 실행하면 다음과 같이 테스트가 실패하는 결과 화면이 나타난다.



실행 결과 화면에 빨간 바가 뜨면서 테스트가 실패했음을 알려준다. 실패한 테스트(add)를 선택하면 테스트가 실패한 원인을 알려준다.

이와 같이 JUnit의 `assertEquals()` 메소드를 활용하면 지금까지 수동으로 확인했던 실행 결과를 자동화하는 것이 가능하다. JUnit의 `Assert` 클래스는 `assertEquals()` 메소드 이외에도 결과 값이 true/false 인지를 확인할 수 있는 `assertTrue()`, `assertFalse()` 메소드, 결과 값이 null 유무를 판단할 수 있는 `assertNull()`,

□□_□.indd 50 2016-09-02 □□ 9:27:28

assertNotNull() 메소드, 배열 값이 같은지를 검증하는 assertArrayEquals() 메소드를 제공하니 Assert 클래스 Javadoc 문서를 참고하기 바란다[1].

2.2.3 테스트 코드 중복 제거

개발자가 가져야 할 좋은 습관 중의 하나는 중복 코드를 제거하는 것이다. 중복 코드는 프로그래밍의 가장 큰 적 중의 하나이다. 테스트 코드 또한 많은 중복이 발생한다. 앞에서 구현한 CalculatorTest 클래스 또한 Calculator 인스턴스를 생성하는 부분에 중복이 발생한다. 이 중복을 다음과 같이 제거할 수 있다.

```
import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class CalculatorTest {
    private Calculator cal = new Calculator();

    @Test
    public void add() {
        assertEquals(9, cal.add(6, 3));
    }

    @Test
    public void subtract() {
        assertEquals(3, cal.subtract(6, 3));
    }
}
```

이와 같이 중복을 제거하는 것은 자바 문법에 아무런 문제도 없다. 하지만 JUnit은 테스트를 실행하기 위한 초기화 작업을 위와 같이 구현하는 것을 추천하지 않는다.

1 최근에는 JUnit의 Assert를 사용하기보다 테스트의 의도를 더 쉽게 파악할 수 있는 기능을 제공하는 AssertJ(http://joel-costigliola.github.io/assertj/)도 많이 사용한다.

□□_□.indd 51 2016-09-02 □□ 9:27:28

JUnit은 위 구현을 `@Before` 애노테이션을 활용해 다음과 같이 구현할 것을 추천한다.

```
import static org.junit.Assert.assertEquals;

import org.junit.Before;
import org.junit.Test;

public class CalculatorTest {
    private Calculator cal;

    @Before
    public void setup() {
        cal = new Calculator();
    }

    @Test
    public void add() {
        assertEquals(9, cal.add(6, 3));
    }

    @Test
    public void subtract() {
        assertEquals(3, cal.subtract(6, 3));
    }
}
```

앞의 구현 방식보다 코딩량도 더 많아 구현 비용도 더 큰데 왜 이 같은 방식으로 구현할 것을 추천할까? `@Before` 애노테이션을 사용하지 않고 필드field로 구현하는 방법은 `CalculatorTest`의 인스턴스가 생성될 때 `Calculator` 인스턴스가 생성되어 모든 테스트 메소드가 재사용하는 방식으로 동작한다. 이와 같이 `Calculator` 인스턴스가 재사용될 경우 `add()` 테스트 메소드를 실행할 때 `Calculator`의 상태 값이 변경되어 다음 테스트 메소드인 `subtract()` 테스트 메소드를 실행할 때 영향을 미칠 수 있기 때문이다. 이와 같이 테스트 메소드 간에 영향을 미칠 경우 테스트 실행 순서나 `Calculator` 상태 값에 따라 테스트가 성공하거나 실패할 수 있기 때문이다. 즉,

□□_□.indd 52 2016-09-02 □□ 9:27:28

@Before 애노테이션을 사용해 테스트 메소드 간에 영향을 미치지 않으면서 독립적으로 테스트 메소드를 실행하기 위함이다. @Before 애노테이션을 사용해 Calculator 인스턴스를 생성하면 각 테스트가 실행될 때마다 인스턴스가 매번 다시 생성되는 방식으로 동작한다.

JUnit은 @Before 애노테이션을 제공해 초기화 작업을 하듯이 @After 애노테이션을 제공한다. @After 애노테이션은 메소드 실행이 끝난 후 실행됨으로써 후처리 작업을 담당한다. @Before, @After 애노테이션 실행 순서를 확인하기 위해 다음과 같이 구현한 코드를 실행해 보자.

```
import static org.junit.Assert.assertEquals;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class CalculatorTest {
    private Calculator cal;

    @Before
    public void setup() {
        cal = new Calculator();
        System.out.println("before");
    }

    @Test
    public void add() {
        assertEquals(9, cal.add(6, 3));
        System.out.println("add");
    }

    @Test
    public void subtract() {
        assertEquals(3, cal.subtract(6, 3));
        System.out.println("subtract");
    }
```


□□_□.indd 53 2016-09-02 □□ 9:27:28

```
    @After
    public void teardown() {
        System.out.println("teardown");
    }
}
```

실행결과는 다음과 같다.

```
before
subtract
teardown
before
add
teardown
```

실행 결과를 보면 각 테스트 메소드가 실행될 때 매번 `@Before`, `@After` 애노테이션으로 설정한 메소드가 실행되면서 초기화와 후처리 작업을 하는 것을 확인할 수 있다. 이와 같이 매번 초기화, 후처리 작업을 통해 각 테스트 간에 영향을 미치지 않으면서 독립적인 실행이 가능하도록 지원한다.

JUnit에 대한 기본적인 사용법은 여기까지 끝이다. 지금 단계는 이 정도만의 학습으로도 충분하다. 지금까지 학습한 내용을 바탕으로 다음 절의 문자열 계산기를 JUnit으로 구현해 보자.

2.3 문자열 계산기 요구사항 및 실습

2.3.1 요구사항

문자열 계산기의 요구사항은 전달하는 문자를 구분자로 분리한 후 각 숫자의 합을 구해 반환해야 한다.

□□_□.indd 54 2016-09-02 □□ 9:27:28

- 쉼표(,) 또는 콜론(:)을 구분자로 가지는 문자열을 전달하는 경우 구분자를 기준으로 분리한 각 숫자의 합을 반환한다.
  (예 " " => 0, "1,2" => 3, "1,2,3" => 6, "1,2:3" => 6)
- 앞의 기본 구분자(쉼표, 콜론) 외에 커스텀 구분자를 지정할 수 있다. 커스텀 구분자는 문자열 앞부분의 "//"와 "\n" 사이에 위치하는 문자를 커스텀 구분자로 사용한다. 예를 들어 "//;\n1;2;3"과 같이 값을 입력할 경우 커스텀 구분자는 세미콜론(;)이며, 결과 값은 6이 반환되어야 한다.
- 문자열 계산기에 음수를 전달하는 경우 RuntimeException으로 예외 처리해야 한다.

위 요구사항만 보면 그리 복잡하지 않다. 간단하다고 곧바로 구현을 시작하지 말고, 요구사항을 더 작은 단위로 나눠 테스트할 경우의 수를 분리해 본다. 곧바로 구현을 시작하기보다 구현을 시작하기 전에 작은 단위로 나누는 연습을 하는 것이 개발자의 역량을 키우기 위한 좋은 연습이다.

http://docs.oracle.com/javase/8/docs/api/java.lang 패키지의 String 클래스를 보면 String 클래스가 지원하는 수많은 메소드가 있다. String 클래스의 메소드를 활용하면 이 문제를 좀 더 쉽게 해결할 수 있다.

다음 절에 각 구현 단계 및 힌트가 있지만 가능하면 힌트를 보지 말고 직접 구현해 볼 것을 추천한다.

2.3.2 요구사항 분리 및 각 단계별 힌트

프로덕션 코드를 구현할 StringCalculator 클래스와 테스트 코드를 구현할 StringCalculatorTest 클래스를 생성한다. 테스트 클래스 이름은 프로덕션 클래스 이름에 Test 접미사를 붙이는 것이 관례이다. 프로덕션 코드와 테스트 코드를 구현하기 위해 클래스를 분리할 뿐만 아니라 최초 소스코드를 관리하는 디렉토리까지 분리한다. 이렇게 분리한 디렉토리 구조는 다음과 같다.

□□_□.indd 55 2016-09-02 □□ 9:27:28