타입으로 견고하게

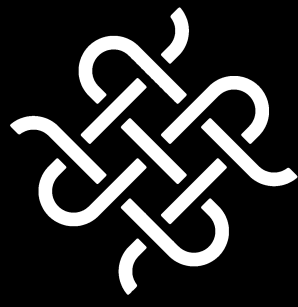
다형성으로 유연하게

타입으로 견고하게 다형성으로 유연하게: 탄탄한 개발을 위한 씨줄과 날줄

전자책 1쇄 발행 2023년 11월 8일 **지은이** 홍재민 **펴낸이** 한기성 **펴낸곳** (주)도서출판인사이트 **편집** 송우일 **등록번호** 제 2002-000049호 **등록일자** 2002년 2월 19일 **주소** 서울특별시 마포구 연남로5길 19-5 **전화** 02-322-5143 **팩스** 02-3143-5579 **블로그** https://blog.insightbook.co.kr **이메일** insight@insightbook.co.kr **ISBN** 978-89-6626-426-1

타입으로 견고하게

다형성으로 유연하게



탄탄한 개발을 위한 씨줄과 날줄

홍재민 지음

인사이트

차례

추천사

"훌륭한 개발자가 되고 싶어요. 어떤 프로그래밍 언어를 배우는 것이 좋을까요? 가장 좋은 언어를 하나 고르라면 무엇을 추천하시겠어요?" 라는 질문을 받습니다. 질문하는 분이 아직 고등학교를 졸업하기 전이라면, 특정 언어를 숙달하는 것보다는 엄밀하게 생각하는 능력과 스스로의 생각을 정확하게 잘 표현하는 훈련이 더 중요하다고 말씀드립니다. 고등학교를 졸업한 분이 질문한다면, 적어도 세 가지 언어로 능숙하게 코딩할 수 있도록 경험을 쌓으시라고 대답합니다. 어떤 문제에도 가장 좋은 단 하나의 언어는 존재하지 않기 때문입니다. 오랜 기간 널리 사용되어 온 C와 C++는 탁월한 프로그램 성능을 제공하고, 자바스크립트는 웬만하면 프로그램이 실행 중에 비정상적으로 멈추지 않으며, 파이썬은 기계 학습 프로그램용으로 매우 널리 사용되고 있습니다. 각자 해결하고자 하는 문제에 제일 적합한 언어를 선택해 사용하면 됩니다.

그렇기는 하지만 프로그래밍 언어의 여러 가지 성질 중 가장 중요한 성질이 무엇이라고 생각하느냐고 물으신다면, 주저하지 않고 '타입 검사'를 선택하겠습니다. 아무리 빠르게 오래 실행되는 프로그램이라고 하더라도 제가 의도한 대로 동작하지 않는 프로그램은 그다지 의미가 없으니까요. 타입 검사는 프로그램이 제대로 동작할지를 프로그램 실행 전에 자동으로 미리 검사해 주는 기능입니다. 프로그램을 실행하려면 타입 검사가 지적하는 모든 문제를 해결해야 해서 귀찮을 수 있지만, 타입 검사를 통과한 후에는 프로그램이 올바르게 동작한다는 것을

보장해 주기 때문에 매우 큰 도움이 되는 기능입니다. 프로그램 작성은 쉽지만 디버깅이 너무나 어려운 자바스크립트나 파이썬과 같은 동적 언어보다는, 프로그램 작성 중에 잔소리를 많이 해서 귀찮기는 해도 올바르게 동작하는 프로그램을 작성하도록 도와주는 정적 언어가 제게는 더 좋은 친구입니다.

이 책은 이러한 타입 검사 기능에 대해 다양한 방식으로 이야기합니다. 타입 검사 기능을 쉬운 우리말로 차근차근 설명하고, 구체적인 코드 예시를 사용해 명확하게 보여주고, 귀여운 처르지와 큐리 박사가 함께 이야기를 이어 갑니다. 특히 중요한 개념에 대한 예시를 자바, C++, C#, 타입스크립트, 고, 코틀린, 러스트, 스칼라, 하스켈, 오캐멀 등 다양한 프로그래밍 언어로 보여 줍니다. 같은 개념을 각 언어가 얼마나 다르게 나타내고 어떤 다른 선택을 했는지 살펴보는 재미가 있습니다.

타입 검사가 약간 낯선 개념일 수도 있지만 마지막까지 큐리 박사와 처르지를 따라가다 보면 타입 시스템의 매력에 흠뻑 빠질 수 있으리라 생각합니다. 이 책의 코딩을 따라 하면서 즐거운 여행을 하길 기대합니다!

류석영 KAIST 전산학부 학부장

시작하며

요즘 가장 인기 있는 프로그래밍 언어 두 가지를 꼽으라면 단연 파이썬(Python)과 자바스크립트(JavaScript)다. 체감상으로도 그렇고 스택 오버플로와 깃허브 통계를 봐도 그렇다. 이 두 언어는 현역 개발자들이 많이 사용할 뿐 아니라 여러 교육 기관의 프로그래밍 입문용 언어로도 가장 많이 선택된다. 아마 이 책의 독자들 중에도 프로그래밍을 처음 배울 때 파이썬이나 자바스크립트로 시작한 사람이 있을 것이다.

무슨 언어로 프로그래밍을 시작했든 그 언어만 계속 사용하는 경우는 드물다. 대개 다른 언어를 사용할 일이 생긴다. 시스템 프로그래밍 분야에는 여전히 전통의 강자인 C와 C++가 사용되며 최근에 러스트(Rust)도 가세했다. 한편 안드로이드 앱을 개발한다면 자바(Java)나 코틀린(Kotlin)을 사용하기 마련이다. 또, 학계를 비롯한 일부 분야에서는 스칼라(Scala), 오캐멀(OCaml), 하스켈(Haskell) 같은 함수형 언어가 널리 사용된다. 그러다 보니 새로운 언어를 익히고 사용하는 능력 역시 개발자의 중요한 소양 중 하나다.

파이썬이나 자바스크립트로 프로그래밍을 시작한 사람이 다른 언어를 사용할 때 새롭게 접하게 되는 개념 중 하나가 바로 타입 검사(type checking)다. 앞에서 언급한 C, C++, 러스트, 자바, 코틀린, 스칼라, 오캐멀, 하스켈은 모두 타입 검사를 제공하는 언어다. 타입 검사를 한마디로 표현하자면 '내가 작성한 프로그램을 실행해도 문제가 없는지 컴퓨터가 자동으로 검사해 주는 기능'이다. 검사를 통과하지 못하면 프로그램을 아예 실행할 수 없다. 원래부터 타입 검사를 제공하는 언어를 사

용하던 사람에게는 익숙한 일이지만, 파이썬과 자바스크립트만 사용해 봤다면 프로그램을 실행하는 것조차 컴퓨터의 허락을 받아야 한다는 점이 꽤나 불편할지도 모른다.

하지만 슬쩍 봤을 때 불편해 보인다고 무작정 멀리하기에는 타입 검사가 제공하는 이점이 너무나도 매력적이다. 그렇지 않았다면 그 많은 언어가 굳이 타입 검사라는 기능을 제공하지도 않았을 테다. 심지어는 타입 검사를 제공하지 않던 언어에마저 타입 검사를 추가하는 게 최근 추세다. 마이크로소프트는 자바스크립트에 타입 검사를 추가해 타입 스크립트(TypeScript)를 만들었다. 파이썬 역시 타입 검사를 보조하는 기능들을 언어에 지속적으로 추가하고 있다. 그러니 앞으로는 무슨 일을 하든 타입 검사를 제공하는 언어를 사용할 가능성이 높다. 타입 검사를 잘 활용하는 능력이 개발자의 필수 소양 중 하나가 되어 가고 있다.

문제는 타입 검사를 잘 활용하기가 결코 쉬운 일이 아니라는 점이다. 물론 프로그램을 실행하기 전에 매번 자동으로 타입 검사가 진행되니 아무것도 몰라도 타입 검사를 그냥 '사용'할 수는 있다. 하지만 타입 검사를 제대로 '활용'하는 것은 전혀 다른 문제다. 타입 검사를 이해하지 않은 채 그냥 사용하기만 한다면 타입 검사의 장점을 살리기 어렵다. 오히려 타입 검사의 불편한 점에 시달리기 일쑤다.

이런 사태를 피하려면 우선 타입 검사의 장점과 단점을 정확히 알아야 한다. 그렇지 않으면 타입 검사를 사용하면서도 내가 지금 무슨 혜택을 얻고 있으며 무엇 때문에 불편한지 알 수가 없으니 상황을 개선할 방도가 없다. 타입 검사의 장단점을 파악했다면 남은 일은 어떻게 해야 타입 검사의 장점은 키우고 단점은 줄일 수 있는지 배우는 것이다. 그러면 비로소 타입 검사를 제대로 활용할 수 있게 된다.

다시 말해 타입 검사를 이왕 사용할 거라면 제대로 이해하고 써야 한

다. 타입 검사를 제대로 이해했다는 것은 곧 '타입 검사의 장단점은 무엇인가?'와 '타입 검사의 장점은 키우고 단점은 줄이려면 어떻게 해야 하는가?', 이 두 질문에 잘 대답할 수 있다는 말이다. 그리고 이 두 질문에 대한 답은 이 책 속에 있다.

이 책은 일차적으로 파이썬이나 자바스크립트처럼 타입 검사가 없는 언어를 주로 사용해 온 사람들을 위한 책이다. 그런 독자들에게 이 책은 타입 검사를 알아 가는 과정에서 좋은 길잡이가 될 것이다. 단, 프로그래밍 경험이 거의 없는 사람에게는 이 책을 권하지 않는다. 최소한 클래스를 정의해 객체를 만들고 사용하는 정도까지는 아는 상태에서 책을 읽을 것을 추천한다.

또한 이 책은 타입 검사를 제공하는 언어를 사용하고 있지만 언어가 제공하는 타입 관련 기능을 능숙하게 사용하지 못한다고 느끼는 사람들을 위한 책이기도 하다. 이런 독자들은 '타입 검사의 장단점은 무엇인가?'에는 이미 어느 정도 스스로 답을 할 수 있을 것이다. 하지만 '타입 검사의 장점은 키우고 단점은 줄이려면 어떻게 해야 하는가?'에 대한 답을 아직 충분히 알지 못하고 있기에 여전히 타입 검사로 인한 불편을 겪을 수 있다. 그렇다면 이 책이 타입을 능숙하게 다룰 수 있는 경지에 도달할 때까지 이끄는 안내자가 될 것이다.

한 가지 주의할 점은, 이 책은 특정한 한 언어의 전문가가 되도록 돕는 책이 아니라는 것이다. 이 책의 목표는 여러 언어가 공통으로 제공하는 타입 관련 기능들을 다룸으로써 독자들이 어느 언어를 사용하든 타입을 능숙히 다루도록 돕는 것이다. 즉, 대다수의 프로그래밍 언어 책과는 다르고, 오히려 자료 구조 책이나 알고리즘 책과 더 비슷하다. 프로그래밍 언어 책들은 대개 한 언어를 정한 뒤 그 언어에 한정된 내용을 구체적으로 다룬다. C 책은 C만 다루고, 자바 책은 자바만 다루는

식이다. C 책을 읽었다고 곧바로 자바까지 능숙히 사용하게 되는 것은 아니다. 그 반면에 자료 구조 책이나 알고리즘 책은 한번 읽으면 그 내용을 어느 언어에나 적용할 수 있다. 정렬 알고리즘을 공부했다면 지금까지 내가 공부한 모든 언어로, 또 앞으로 공부할 모든 언어로 정렬 함수를 구현할 수 있게 된다. 이 책도 마찬가지다. 앞으로 무슨 언어를 사용하게 되든 상관없다. 그 언어가 타입 검사를 제공한다면 이 책에서 다룬 내용이 도움이 될 것이다.

이 책은 다음과 같이 다섯 개 장으로 구성되어 있다.

1. 타입 검사 훑어보기
2. 서브타입에 의한 다형성
3. 매개변수에 의한 다형성
4. 두 다형성의 만남
5. 오버로딩에 의한 다형성

1장에서는 '타입 검사의 장단점은 무엇인가?'에 대한 답을 살펴본다. 그에 더해 타입 검사의 기본 원리를 이해함으로써 타입 검사가 내 프로그램을 퇴짜 놓는 사태에 대응하는 기초적인 전략도 알아본다. 2~5장에서 타입 검사를 통과하는 프로그램을 작성하는 데 도움을 주는 프로그래밍 언어의 각종 기능을 다룸으로써 '타입 검사의 장점은 키우고 단점은 줄이려면 어떻게 해야 하는가?'에 대답한다. 각 장의 내용은 그 앞에 있는 장들의 내용을 아는 것을 전제로 한다. 따라서 자신이 관심 있는 내용을 다루는 부분으로 건너뛰면서 읽기보다는 앞부터 순서대로 읽을 것을 추천한다.

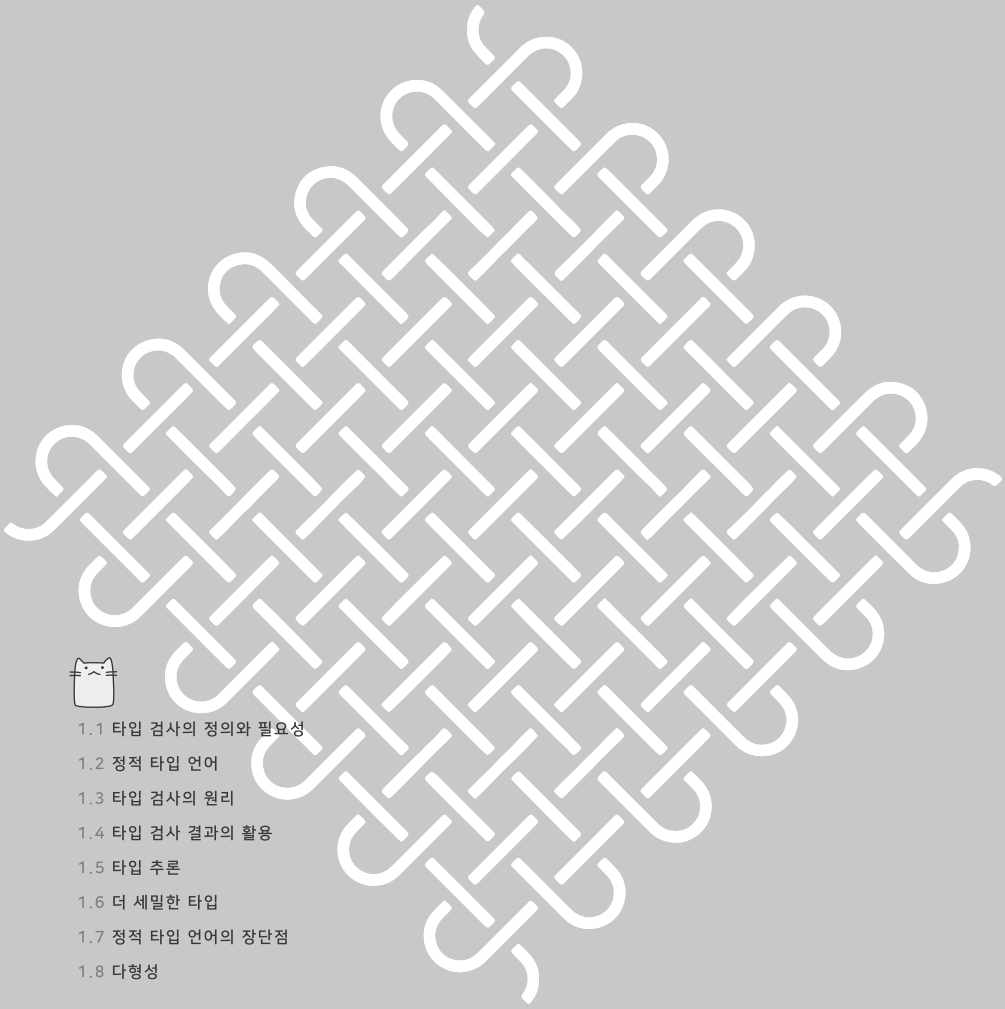
책에서 전문 용어를 적을 때는 대부분 관용적인 방식을 따랐다. 가령 오버라이딩(overriding)과 오버로딩(overloading)은 음차(音借) 형태가 이미 널

리 사용되고 있기에 우리말로 옮기는 대신 음차 형태를 그대로 사용했다. 또한 음차 시 일부 용어는 관용 표기를 유지했다. 다만 우리말로 설명한 자료가 얼마 되지 않는 개념은 음차를 가급적 피하고 우리말로 풀어 썼다. 위치에 민감한 타입 검사(flow-sensitive type checking)와 타입 매개변수 제한(bounded quantification)이 그 예다. 이런 용어는 어차피 검색해도 우리말 자료가 잘 없으니 검색의 용이성보다는 용어를 쉽게 풀어 써 책을 읽기 쉽게 만드는 데 초점을 둔 것이다. 필요한 경우 각 용어에 원래 영어 용어도 병기했으니 영어에 큰 거부감이 없다면 영어로 검색해 더 많은 자료를 찾아보면 도움이 될 것이다.

타 입 으 로 견 고 하 게
다 형 성 으 로 유 연 하 게

1장

타입 검사 훑어보기



1.1 타입 검사의 정의와 필요성

타입 검사는 도대체 무엇이며 왜 필요할까? 타입 검사를 하는 게 좋은가? 불편하지 않나? 왜 사람들이 사용하지? 타입 검사를 공부하기에 앞서 자연스럽게 떠오를 질문이다. 반드시 물어야 할 중요한 질문이기도 하다. 사실 타입 검사는 불편한 게 맞다. 하지만 그 불편을 감수하면서도 사용할 만큼 큰 가치가 있다. 지금부터 타입 검사가 무엇이며 왜 필요한지 알아보자.

우선 버그(bug)에 관한 이야기부터 해야 한다. 버그란 프로그램이 개발자의 의도와 다르게 동작하는 모든 경우를 말한다. 프로그램이 의도대로 동작해야 사용자를 만족시킬 수 있으니 프로그램에 버그가 있으면 고쳐야 한다. 하지만 처음부터 버그가 없는 프로그램을 만들기란 불가능에 가깝다. 개발자는 자신의 프로그램에서 끊임없이 버그를 찾고 고친다. 버그는 개발자에게 가장 큰 적이면서도 절대로 피할 수 없는 존재다.

버그 수정도 중요하고 어렵지만, 그보다 더 중요하고 어려운 일은 버그를 찾는 것이다. 버그를 일단 찾아야 고치든 말든 할 테니 버그를 찾는 것만큼 중요한 일이 없다. 개발 과정에서 버그를 미처 발견하지 못한 채 프로그램을 배포한다면, 사용자가 버그 때문에 피해를 볼 것이다. 버그를 찾는 일은 어렵기까지 하다. 프로그램에 버그가 있는지 없는지조차 모르는데, 수천수만 줄의 코드 속 어딘가에 있을지도 모르는 버그를 찾아 헤매야 한다.

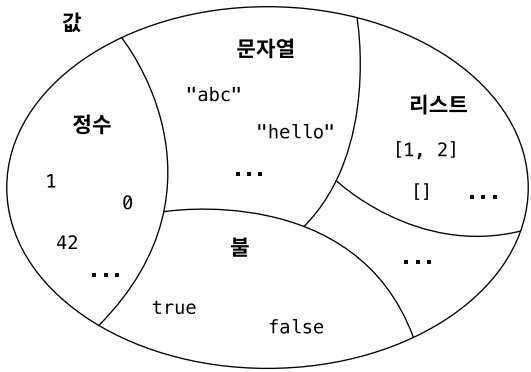
어떻게 해야 버그를 잘 찾을 수 있을까? 지피지기(知彼知己)면 백전불태(百戰不殆)라고(적을 알고 나를 알면 백번 싸워도 위태롭지 않다) 했으

니 먼저 버그라는 적을 잘 이해할 필요가 있다. 버그의 가장 흔한 원인은 타입 오류(type error)다. 타입 오류가 무엇인지 알려면 타입이 무엇인지부터 알아야 한다.

타입은 프로그램에 존재하는 값(value)들을 그 능력에 따라 분류한 것이다. 여기서 값이란 변수(variable)에 저장되거나 함수(function)에서 반환될 수 있는 `1`, `true`, `"abc"` 같은 것들을 말한다. 프로그램에는 다양한 값이 등장하며 제각각 다른 능력을 가지니 값들을 분류하는 행위는 꽤나 자연스럽다. 예를 들어 1 같은 정수는 사칙 연산에 사용할 수 있다. 한편 "abc" 같은 문자열(string)은 정수 n이 주어졌을 때 자신의 n번째 문자(character)가 무엇인지 알려 준다든지, 다른 문자열이 주어졌을 때 그 문자열이 자신의 일부인지 확인한다든지 하는 능력을 가진다. 그 반면에 정수에서 n번째 문자를 받아오거나 문자열을 사칙 연산에 사용하는 것은 불가능하다.

	1	"abc"
사칙 연산에 사용할 수 있는가?	○	×
자신의 n번째 문자를 알려 주는가?	×	○

그러므로 정수는 정수끼리 하나의 타입으로 묶고, 문자열은 문자열끼리 하나의 타입으로 묶을 수 있다. 정수나 문자열 말고도 불(boolean), 리스트(list), 함수 등 다양한 타입이 있다. 각 타입은 능력이 서로 다르다.

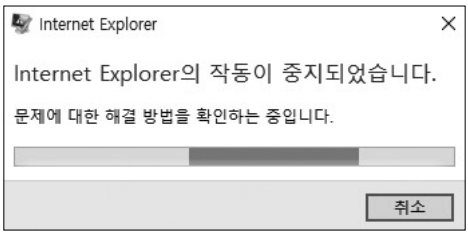


물론 언어마다 존재하는 타입과 각 타입의 능력은 조금씩 다르다. 예를 들어 문자열을 덧셈에 사용할 수 있는 언어도 있고 그렇지 않은 언어도 있다. 하지만 어느 언어에서나 값들을 능력에 따라 여러 타입으로 분류할 수 있다는 사실은 변하지 않는다.

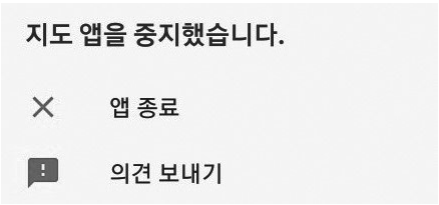
어떤 타입의 값을 그 타입이 갖고 있지 않은 능력이 필요한 자리에 사용한다면 프로그램 실행에 문제가 생긴다. 문자열 타입이 뺄셈이라는 능력을 제공하지 않음에도 "abc" - "d"라는 코드를 실행하는 경우가 한 예다. 타입 오류는 바로 이런 경우를 말한다. 실행 중에 타입 오류가 발생하면 프로그램 실행이 즉시 중단되고 사용자에게 오류 메시지를 출력한다. 오류 메시지의 모양은 무슨 프로그램을 어디서 실행했느냐에 따라 다르지만 프로그램이 갑작스레 멈춘다는 점은 모두 같다.

```
Traceback (most recent call last):
  File "example.py", line 1, in <module>
    "abc" - "d"
TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

명령 줄(command line)에서의 오류 메시지



윈도우에서의 오류 메시지

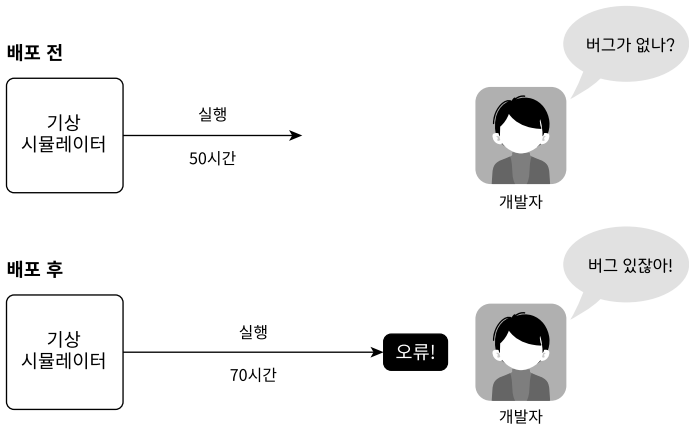


안드로이드에서의 오류 메시지

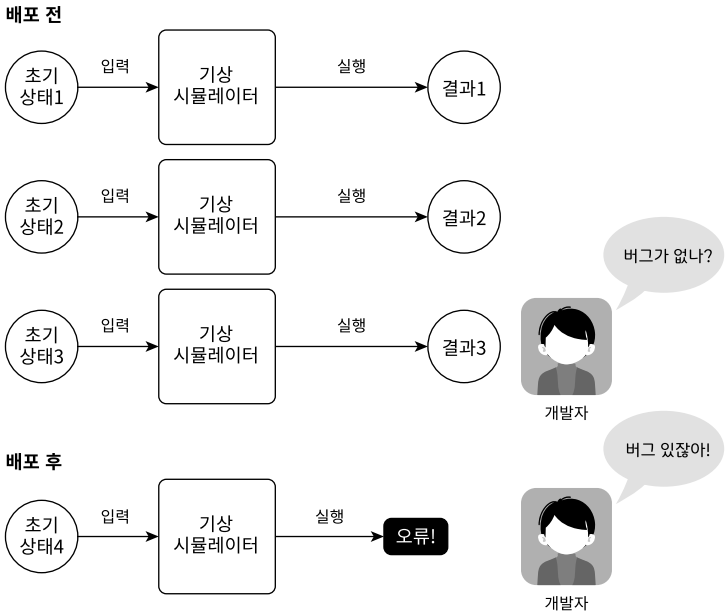
타입의 능력을 벗어나는 일을 시켰으니 프로그램이 어쩔 줄 모르고 그냥 멈춰 버린 것이다. 프로그램이 결과도 출력하지 않고 갑자기 종료되면서 사용자에게 오류 메시지나 내뱉는 일을 개발자가 의도할 리 없다. 그러므로 프로그램이 타입 오류를 일으킨다는 이야기는 곧 그 프로그램에 버그가 있다는 말이다.

버그라는 적을 잘 이해했으니 이제 버그를 찾기 위해 우리가 뭘 할 수 있는지 알아볼 차례다. 버그를 찾는 가장 쉬운 방법은 프로그램을 직접 실행해 보는 것이다. 타입 오류가 발생한다면 프로그램이 실행 중에 오류 메시지를 출력하며 종료될 것이다. 그럼 프로그램에서 버그를 발견한 것이다. 컴퓨터만 있으면 누구나 프로그램을 실행할 수 있으니 정말 쉬운 방법이다.

하지만 동시에 큰 약점도 있다. 실행을 통해 프로그램이 처할 수 있는 모든 상황을 시험해 볼 수는 없다는 것이다. 끝나려면 100시간쯤 걸리는 기상 시뮬레이션 프로그램을 만들었다고 해 보자. 실행해서 타입 오류가 발생하는지 확인하려면 100시간을 기다려야 하는 셈이다. 배포 일정을 맞추기 위해 50시간 안에 버그를 찾아야만 한다면 어떨까? 프로그램을 실행하고 처음 50시간 동안은 타입 오류가 발생하지 않음을 확인할 수 있겠지만, 그다음 50시간 동안도 타입 오류가 없을지는 아무도 모른다.



다행히 시간이 넉넉하다고 하더라도 여전히 문제가 있다. 프로그램에 입력으로 주어질 수 있는 시뮬레이션 초기 상태의 수는 무수히 많다. 그중 몇 개를 골라 실행해 봄으로써 타입 오류가 없음을 확인할 수는 있다. 하지만 시도해 보지 않은 나머지 초기 상태 중에는 프로그램을 타입 오류로 몰고 갈 입력이 있을지도 모른다.

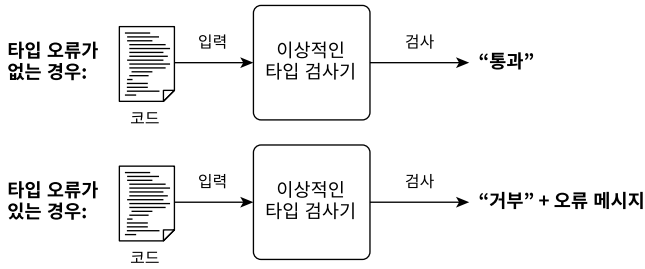


만든 프로그램이 입력을 받지 않고 실행 시간도 짧다면 실행해 보기만 해도 버그를 충분히 찾을 수 있을 것이다. 안타깝게도 대부분의 프로그램은 그렇지 않다. 우리는 더 좋은 방법이 필요하다.

단순히 실행하는 것으로 부족하다면 가지고 있는 코드를 더 적극적으로 활용해야 한다. 프로그램이 무슨 일을 할지는 사실 코드에 다 적혀 있으니 말이다. 그래서 사람들은 프로그램의 코드를 입력받아 그 프로그램이 타입 오류를 일으키는지 자동으로 판단해 주는 프로그램인 타입 검사기(type checker)를 사용한다.

가장 이상적인 타입 검사기는 주어진 프로그램이 타입 오류를 절대 일으키지 않는다면 '통과'라고 출력하고, 타입 오류를 일으키는 경우의 수가 단 하나라도 있다면 '거부'라고 출력한다. 여기에 더해 '거부'라고 출력할 때는 타입 오류가 어디에서 왜 일어날 수 있는지 오류 메시지를

출력한다. 이 오류 메시지는 실행 중에 타입 오류가 발생해서 출력된 오류 메시지와는 다르다. 타입 검사기가 타입 오류가 발생할 가능성을 경고하려고 출력한 것일 뿐, 실제로 타입 오류가 일어난 게 아니다.



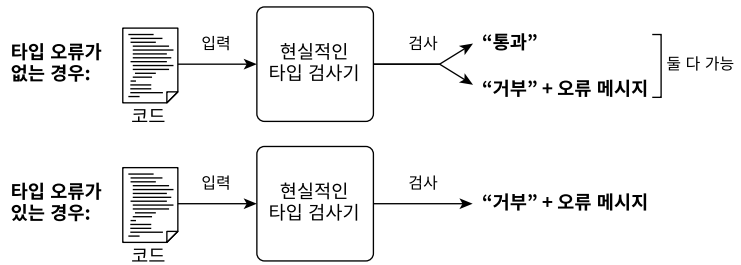
이런 타입 검사기가 있다면 걱정할 게 없다. 프로그램을 만들고 나서 타입 검사기에 코드를 넣기만 하면 된다. 타입 검사기가 '통과'라고 한다면 타입 오류가 없다는 확신이 생기고, '거부'라고 한다면 타입 검사기가 알려 준 내용을 바탕으로 코드를 고치면 된다.

딱 한 가지 문제는 이런 이상적인 타입 검사기가 존재하지 않는다는 것이다. 이런 타입 검사기를 만들기가 매우 어렵다거나, 아직까지 아무도 만드는 방법을 찾지 못했다는 이야기가 아니다. 아무리 기술이 발전하고 대단한 천재가 나와도 이런 타입 검사기를 절대 만들 수 없다는 사실이 논리적으로 이미 증명되어 있다. 이는 무려 1930년대에 컴퓨터의 아버지 튜링(Alan Turing)이 직접 증명한 사실이다.[1]

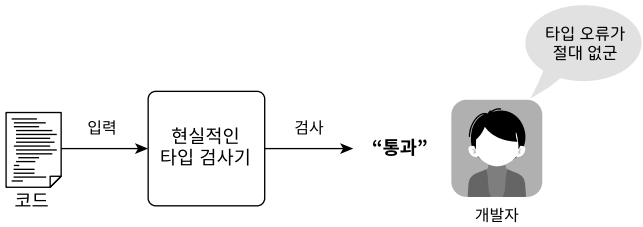
그래서 약간의 타협이 필요하다. 여전히 타입 오류를 놓치고 싶지는 않다. 그러니 우리의 현실적인 타입 검사기도 주어진 프로그램이 타입

1 정확히 말하자면, 튜링이 증명한 것은 주어진 프로그램이 언젠가 정지하는지, 아니면 무한히 실행되는지 판단하는 프로그램이 없다는 사실이다. 하지만 타입 오류가 있는지 판단하는 타입 검사기가 없다는 사실도 같은 방식으로 증명할 수 있으니, 사실상 튜링이 증명했다고 봐도 무방하다.

오류를 일으키는 경우의 수가 단 하나라도 있다면 '거부'라고 출력해야 한다. 여기까지는 이상적인 타입 검사기와 똑같다. 그 대신 실제로는 타입 오류가 없는데도 '거부'라고 출력하는 경우가 있다. 다시 말해 주어진 프로그램이 타입 오류를 절대 일으키지 않는다면, 올바르게 '통과'를 출력할 때도 있지만 간혹 가다 '거부'라고 출력하는 경우도 있는 것이다. 이게 바로 항상 올바르게 '통과'를 출력할 이상적인 타입 검사기와의 차이다.

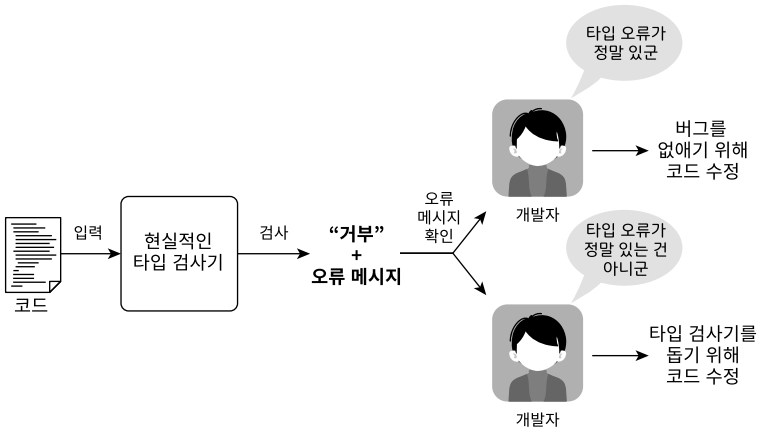


이상적인 타입 검사기만은 못해도 현실적인 타입 검사기 역시 충분히 쓸 만하다. 타입 검사기가 '통과'라고 한다면 타입 오류가 없다고 여전히 확신할 수 있다.



이는 타입 검사기의 가장 중요한 성질로, 흔히 타입 안전성(type safety)이라 부른다. 타입 검사기가 '통과'라고 했다면 프로그램을 타입 오류 없이 안전하게 실행할 수 있다는 말이다.

문제는 타입 검사기가 '거부'라고 한 경우다. 정말 타입 오류가 있어서 '거부'라고 했을 수도 있지만, 사실은 타입 오류가 없는데도 '거부'라고 했을 수도 있다. 이는 타입 검사기를 사용하는 데 있어서 가장 큰 불편이다. 그나마 다행인 점은 타입 검사기가 '거부'라고 출력할 때는 타입 오류가 어디에서 왜 일어날 수 있는지도 함께 알려 준다는 것이다. 대부분의 경우, 개발자가 이 정보를 바탕으로 타입 오류가 있는지 없는지 쉽게 판단할 수 있다. 타입 오류가 정말 있다면 타입 오류를 없애기 위해 코드를 고치면 되고, 타입 오류가 사실은 없다면 타입 검사기가 그 사실을 올바르게 알아낼 수 있도록 코드를 살짝 바꾸면 된다.



이처럼 타입 검사는 작성한 프로그램에서 버그를 자동으로 찾아 준다는 가치를 지닌다. 그 대가로 불편을 약간 감내해야 하지만 말이다. 바로 나는 제대로 짰는데도 타입 검사기가 '거부'라고 잘못 알려 주는 불편함이다. 하지만 불편보다는 가치가 훨씬 크다. 버그는 아주 많고 사람의 힘만으로 모든 버그를 찾기란 매우 어렵다. 그 반면에 '거부'라고 잘못 말하는 경우는 적고 그런 경우에도 오류 메시지를 참고하면 타입