클린 코드 Chapter 2: 의미 있는 이름

프로그래밍에서 이름 짓기는 중요하다. 좋은 이름은 나중에 코드를 읽는 이에게 적절한 맥락을 제공해주고 코드의 의미를 명료하게 만든다. 나쁜 이름은 읽는 이로 하여금 코드가 어떤 의도를 가지고 쓰였는지 추측하게 만들고, 코드의 의미를 혼동하게 만든다. 클린 코드의 저자는 좋은 이름을 짓기 위한 몇 가지 원칙들을 다음과 같이 설명한다.

1. 의도를 분명히 밝혀라.

나쁜 이름을 짓고 주석을 달아 부연 설명하지 말고 처음부터 좋은 이름을 지어라. 수량을 나타내는 변수명일 경우에는 단위를 달아 변수의 의미를 명확히 해라. 코드를 이해하기 위한 맥락을 이름으로 제공해라.

2. 그릇된 정보를 피하라.

특별한 의미를 가진 단어를 변수명에 사용하여 읽는 이를 혼동하게 하지 않는다. 헷갈리기 쉬운 한 글자 이름은 피하고, 다른 객체, 자료형, 변수, 함수는 확실하게 다른 이름을 써서 혼동을 방지한다. 반대로 비슷한 종류의 대상들은 비슷한 이름을 써서 비슷한 일을 할 것임을 예측 가능하게 한다.

3. 의미 있게 구분하라.

군더더기 말, 혹은 불용어를 추가하여 이름 중복 문제를 손쉽게 해결하려 들지 않는다. 독자에게 의도를 잘 전달하는 이름 중 가장 간결한 이름을 사용한다. 같은 타입을 가진 함수 인수가 여러 개라 해서 숫자로만 구분 가능한 이름, 예를 들어 arg1, arg2, …과 같은 이름을 짓지 않는다.

4. 발음하기 쉬운 이름을 사용하라.

단어를 과도하게 축약하여 발음을 어렵게 만들지 않는다. 과도하게 생략된 이름은 읽는 이가 의미를 추측하기 어렵게 할 뿐만 아니라, 차후 의사소통에 지장을 준다.

5. 검색하기 쉬운 이름을 사용하라.

변수, 함수, 객체가 하는 일이 무엇인지 알아보기 위해 소스 코드를 검색하는 일은 흔하다. 과도하게 짧거나 서술적이지 못한 이름, 예를 들어 한 글자 알파벳이나 숫자 등으로 검색을 방해하지 않는다. 상수는 적절한 이름을 붙여 맥락을 제공하고 차후 값이 바뀔 때 모든 사용처를 찾아 일일이 확인 후 바꿔주는 수고를 예방한다.

6. 인코딩을 피하라.

헝가리안 표기법, 멤버 변수 접두사, 인터페이스 접두사 등을 변수 이름에 넣지 않는다. 최근의 IDE는 함수 혹은 변수의 선언부로 바로 이동하거나 들춰보는 기능이 있어 헝가리안 표기법은 더 이상 의미가 없으며, 새로운 타입이 생긴다면 이에 대응하는 새로운 접두사를 또 만들어야 하는 수고가 생겨난다. 게다가 변수의 타입이 바뀐다면 일반적으로 작성된 코드는 논리적 구현만 손보면 그만이지만, 이름에 타입 등을 인코딩한 코드는 이름까지 바꿔주어야 한다. 이 과정을 잊어버린다면 바로 그릇된 정보를 제공하는 이름이 된다.

7. 한 개념에 한 단어를 사용하되 말장난은 하지 마라.

일관적으로 이름을 지어 읽는 이가 헷갈리게 하지 않는다. 무언가를 덧붙이는 기능을 하는 메서드가 여러 개 있다면 append, 네트워크에서 무언가를 받아오는 기능을 하는 메서드가 여러 개 있다면 fetch 등과 같이 비슷한 개념은 같은 단어로 묶는다. 단, 이는 어디까지나 비슷하거나 같은 개념을 가진 대상에만 적용되며, 다른 속성을 가진 대상에 억지로 위를 적용하려 하지 않는다. 어떤 대상의 맨 끝에 무언가를 덧붙이는 게 아닌 중간에 삽입하는 메서드라면 append는 좋지 않은 변수 이름이다.

8. 해법 영역과 문제 영역에서 가져온 이름을 사용하라.

여러 자료구조, 알고리즘, 디자인 패턴 등의 해법 영역과 코드가 해결하고자 하는 문제 영역에서 이름을 가져오는 것은 좋다. 읽는 이 또한 프로그래머이며 해당 문제에 대한 지식이 어느 정도 있을 것이라고 가정할 수 있기 때문이다.

9. 의미 있는 맥락을 추가하되 불필요한 맥락은 피하라.

이름의 의미가 불분명하다면 우선 해당 변수, 함수, 클래스를 상위 클래스, 함수, 이름 공간에 넣는 것을 고려해본다. lookup은 혼자 쓰이기엔 너무 광범위한 이름이지만 TranslationEnvironment.lookup은 무언가를 번역하기 위한 어떤 환경 내에서 (아마 인자로 주어질 무언가를) 찾는 기능을 하는 함수라는 것을 추측하기 쉽다. 모든 시도가 실패한다면 접두사를 추가하되, 코드 재사용성을 저해하는 무의미하거나 불필요한 맥락은 피한다.

10. 읽는 이가 자신과 같은 배경지식을 가지고 있을 것이라 가정하지 말라.

자신만 아는 유머, 자신만 아는 불분명한 변수 이름 등은 피한다. 읽는 이가 프로그래머에게 일반적인 전산 지식과 현재 프로그램이 해결하고자 하는 문제에 대한 지식 이외를 알 것이라고 가정하여서는 안된다.

위 원칙에 비추어서 내가 과거 작성했던 코드를 다시 한 번 살펴보았다. 가장 먼저 눈에 띈 것은 cnthp라는 함수 이름이다. 보자마자 의미를 추측하기는 꽤 힘든 이름이다. cnt가 가끔씩 count의 약어로 쓰인다는 것을 알고 있다면 뭔가를 세는 함수 이름이라는 것은 추측할 수 있겠다. 이 이름이 의도한 원래 발음은 countHeap이었고, 함수의 시그니처는 cntst (env: env) : int이다. 이 함수는 인자로 주어진 환경 내에 총 몇개의 힙 영역 변수가 있는지를 돌려주는 보조 함수였다. 솔직히 말하자면, 내가 작성한 코드임에도 불구하고 이 이름을 처음 맞닥뜨렸을 때 무엇을 하는 함수인지 추측해내는데 꽤 오랜 시간이 걸렸다. 만약 이 코드를 다시 짠다면 먼저 이 함수를 Environment 클래스의 메서드로 바꾸고, 이름 또한 하는 일을 더 명확히 파악할 수 있도록 countHeapVariable과 같이 바꾸는 편이 좋을 것이다.

클린 코드 Chapter 3: 함수

함수는 소스 코드의 가장 기본적인 단위로, 좋은 함수를 짜는 것은 좋은 코드를 짜는 것의 첫걸음이다.