# Introduction 02

# 문제12-1의 해답

(1) 메소드에는 기계적으로 synchronized를 붙여 두는게 「좋을 거야」

### [안전성]

… synchronized를 붙여서 안전성이 떨어지는 일은 없습니다. 하지만 synchronized를 붙였다고 해서 반드시 안전성이 높아지는 것은 아닙니다. 예를 들어 보호해야 할 필드를 public으로 했다면 아무리 메소드에 synchronized를 붙였다 하더라도 안전성이 결여된 프로그램이 되고 맙니다.

# [생존성]

… 기계적으로 synchronized를 붙이면 일반적으로 생존성은 떨어집니다. 예를 들어 데드락이 발생하기 쉬워집니다. 데드락에 대해서는 Chapter 01에서 설명했습니다.

### [수행 능력]

- … 필요 이상으로 synchronized를 붙이면 수행 능력(예를 들어 쓰루풋)이 떨어집니다. 이것은 synchronized 메소드의 호출이 synchronized 메소드가 아닌 메소드의 호출보다 일반적으로 시간이 많이 걸리기 때문입니다.
- (2) synchronized 메소드 안에서 무한 루프하고 있잖아. 프로그램이 "별로"구만

# [안전성]

··· synchronized 메소드 안에서 무한 루프하더라도 그 루프 안에서 실행하고 있는 처리가 올바르다면 안 전성은 떨어지지 않습니다.

## [생존성]

··· synchronized 메소드 안에서 무한 루프하면 그 synchronized 메소드를 실행하려 했던 다른 쓰레드는 모두 블록하고 있기 때문에 생존성이 떨어집니다.

# [수행 능력]

- … 생존성이 떨어지면 수행 능력(예를 들면 쓰루풋)도 떨어집니다.
- (3) 프로그램 실수로 무한 루프에 들어간 쓰레드를 100개나 기동시키고 말았네. 하지만 쓰레드가 단순히 루프하고 있는 것일 뿐이니까 뭐「괜찮겠지」



# [안전성]

… 루프만 하고 있는 쓰레드는 그 수가 많더라도 안전성이 떨어지지 않습니다.

#### [생존성]

… 루프만 하고 있는 쓰레드가 많다고 해서 생존성이 떨어지지는 않습니다. 다만 쓰레드가 시스템이 제공할수 있는 메모리 이상의 메모리를 소비하는 경우에는 필요한 쓰레드가 기동하지 못할 가능성이 있으므로 생존성은 떨어집니다.

#### [수행 능력]

- … 쓰레드를 움직이는데 필요한 메모리, 그리고 CPU 시간이라고 하는 소스를 소비하기 때문에 수행 능력은 떨어집니다.
- (4) 이 서버에는 동시에 한 개의 클라이언트밖에 접속할 수 없나요? 좀 더 「좋은」 서버로 만들어주세요.

# [수행 능력]

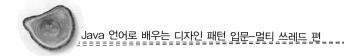
- ··· 여기에서는 동시에 복수의 클라이언트가 접속할 수 있는 서버를 좋은 서버라고 이야기하고 있는 것 같 군요. 즉, 서버의 「용량」이 문제가 되고 있습니다.
- (5) 뭐 이런 「후진」 검색 프로그램이 다 있어! 일단 시작하면 검색을 전부 마칠 때까지 취소를 할 수가 없잖아.

#### [수행 능력]

- … 프로그램의 「응답성」이 문제입니다.
- (6) 이렇게 하면 클래스 A와 클래스 B 양쪽이 쓰레드의 배타제어를 의식하는 게 되어서 「안 좋으려나」.

#### [재사용성]

⋯ 클래스의 「재사용성」이 문제가 되고 있습니다. 이 사람은 클래스 A와 클래스 B 양쪽의 코드가 배타제어를 의식하는 내용으로 되어 있어서 걱정하는 것 같습니다. 기동하는 쓰레드에 대한 코드를 한쪽 클래스에만 가두어 둠으로써 클래스의 의존관계가 적어지고 부품으로서의 재사용성을 높일 수 있는 경우가 있습니다.



# ■ 문제 12-2의 해답

일반적으로는 2배가 되지 않습니다. 예를 들어 다음과 같은 이유를 생각해 볼 수 있습니다.

- :: 이유 1 : 하드웨어의 제약으로 2배의 쓰레드가 동시에 동작한다(병렬로 동작한다)고는 장담할 수 없기 때문이다. 또한 그 경우 쓰레드 교체의 오버헤드가 일어날 가능성이 있다.
- :: 이유 2 : 처리하는 업무를 모든 쓰레드에게 균일하게 분담시키는 것이 가능하다고 장담할 수 없기 때문이다.
- :: 이유 3: 가령 하드웨어의 제약이 없어 2배의 쓰레드가 병렬로 동작했다 하더라도 쓰레드간 배타제어를 하기 위한 오버헤드가 있기 때문이다.