Trading System

Chapter1

Trading System 소개

TDD + Mocking Project

[참고] private 에 대한 Unit Test는 생략하자.

✔ private 는 Unit Test 하지 않아도 된다.

- private 는 public에 의해 수행된다.
- public의 Unit Test 수행해도, 대부분의 private은 Test가 가능하다.

✔ private를 Test 하고 싶다면?

- 모두 public으로 변경하여 Test를 수행한다.
- Test가 끝난 함수는 private로 변경한다.

제공되는 API

✔ Nemo증권과 Kiwer증권사에서 제공되는 주식거래 API

- 제공되는 소스코드 링크 : https://github.com/mincoding1/TradingSystem

✔ KiwerStock

- login(ID, PASS)
- buy(종목코드, 수량, 가격)
- sell(종목코드, 수량, 가격)
- currentPrice(종목코드)

✔ NemoStock

- certification(ID, PASS)
- purchasingStock(종목코드, 가격, 수량)
- sellingStock(종목코드, 가격, 수량)
- getMarketPrice (종목코드, 시간)

개발해야 하는 Application - 1

✔ Auto Trading System – 기본 기능

- 증권사 선택 기능 : selectStockBrocker( )
  - 키워 or 네모 증권 선택 가능

- 로그인 기능 : login(id, pass)
- 매수 기능 : buy(종목코드, 가격, 수량)
- 매도 기능 : sell(종목코드, 가격, 수량)
- 현재가 확인 기능 : getPrice(종목코드)

개발해야 하는 Application - 2

✔ Auto Trading System – 핵심 기능

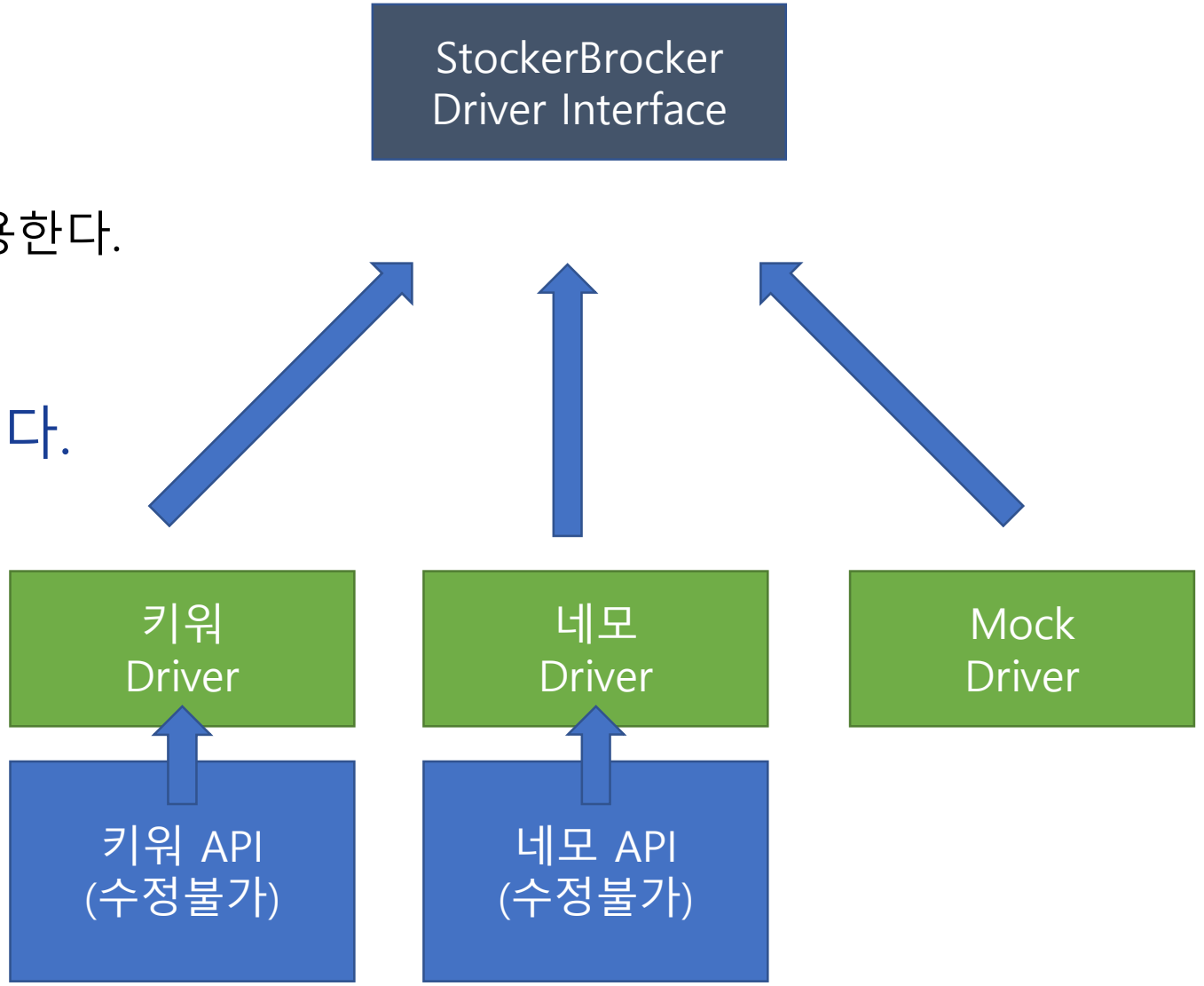
- **기능 1 : buyNiceTiming(종목, 금액)**
  - 200ms 주기로 3회 가격을 읽고, 가격이 올라가는 추세인지 파악한다.
  - 가격이 올라가는 추세라면, 총 금액을 최대한 사용하여 최대 수량만큼 매수한다.
  - 마지막에 읽은 가격으로 매수한다.

- **기능 2 : sellNiceTiming(종목, 수량)**
  - 200ms 주기로 3회 가격을 읽고, 가격이 내려가는 추세인지 파악한다.
  - 가격이 내려가는 추세라면, 사용자가 설정한 수량만큼 주식을 모두 매도한다.
  - 마지막에 읽은 가격으로 매도한다.

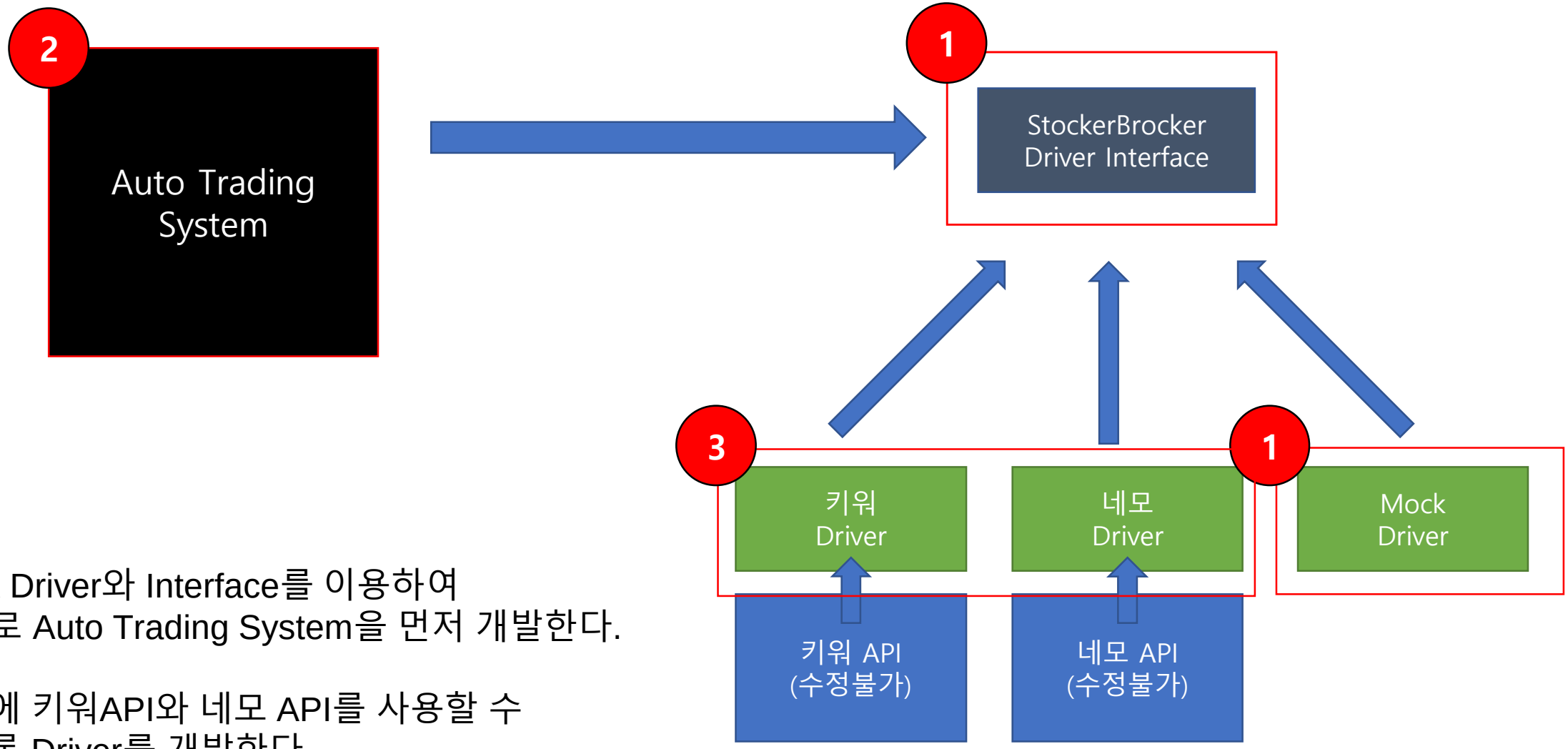
Stock Brocker

✔ OCP로 개발 필요
- 확장성을 고려한 개발
- 공통된 Interface 사용을 위해 Adapter역할을 하는 Driver를 사용한다.

✔ 키워, 네모 API를 사용하여 Unit Test를 하면 비용이 발생한다.
- Mocking으로 TDD 하면 비용이 발생되지 않는다.
- Mock으로 개발 이후, Driver를 추가하여 키워와 네모 API를 사용할 수 있도록 한다.



개발 순서



Mock Driver와 Interface를 이용하여
TDD로 Auto Trading System을 먼저 개발한다.

이후에 키워API와 네모 API를 사용할 수
있도록 Driver를 개발한다.

Chapter2

협업 방법

TDD + Mocking Project

핑퐁 페어프로그래밍

✔ 핑퐁 페어프로그래밍

- 출처 : https://anthonysciamanna.com/2015/04/18/ping-pong-pair-programming.html

Loop {

- 개발자 A : 실패하는 테스트를 작성
- 개발자 B : 통과하기에 충분한 코드만 작성하여, 테스트 통과 후 리팩토링
- 개발자 B : 다음 실패하는 테스트 작성
- 개발자 A : 통과하기에 충분한 코드만 작성하여, 테스트 통과 후 리팩토링

}

Unit Test를 작성한다는 것은
한 명의 개발자가
다른 개발자에게 명확히 할 일을 부여한다는 의미.

진행방법

✔ 팀장님

- 할일(ToDo)를 작성한다.
- 팀원분들이 개발할 명세를 Unit Test로 작성한다.
- 스켈레톤 코드조차 만들지 않는다.
- 메신저에 ToDo를 올린다.

✔ 팀원분들

- 메신저에 올라온, ToDo 중 하나를 맡아 개발한다.

구체적인 진행방법

✔ 팀장님

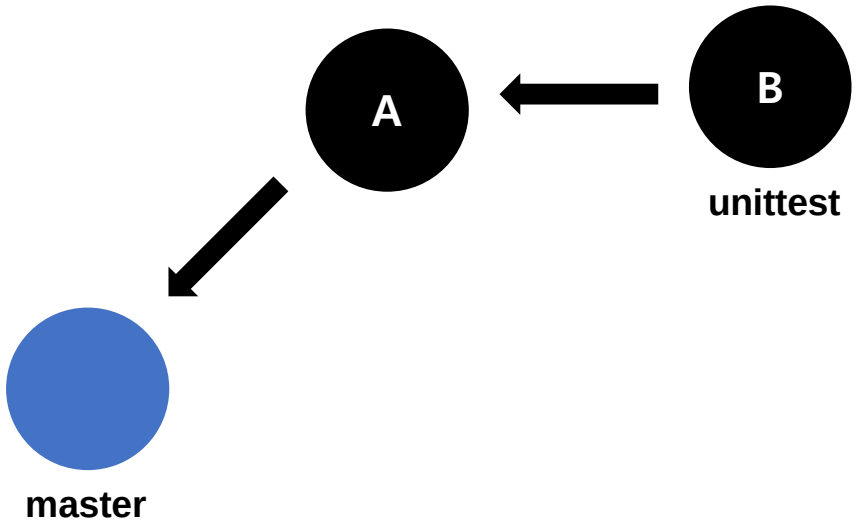
- Unit Test Branch 생성
- 설계 & Unit Test 작성, Unit Test 리팩토링
- 코드리뷰

✔ 팀원분들

- A : Branch 생성, Pass 되도록 개발 + Refactoring, PR 요청
- B : Branch 생성, Pass 되도록 개발 + Refactoring, PR 요청
- C : Branch 생성, Pass 되도록 개발 + Refactoring, PR 요청

핑퐁 페어프로그래밍 - Branch 전략 1

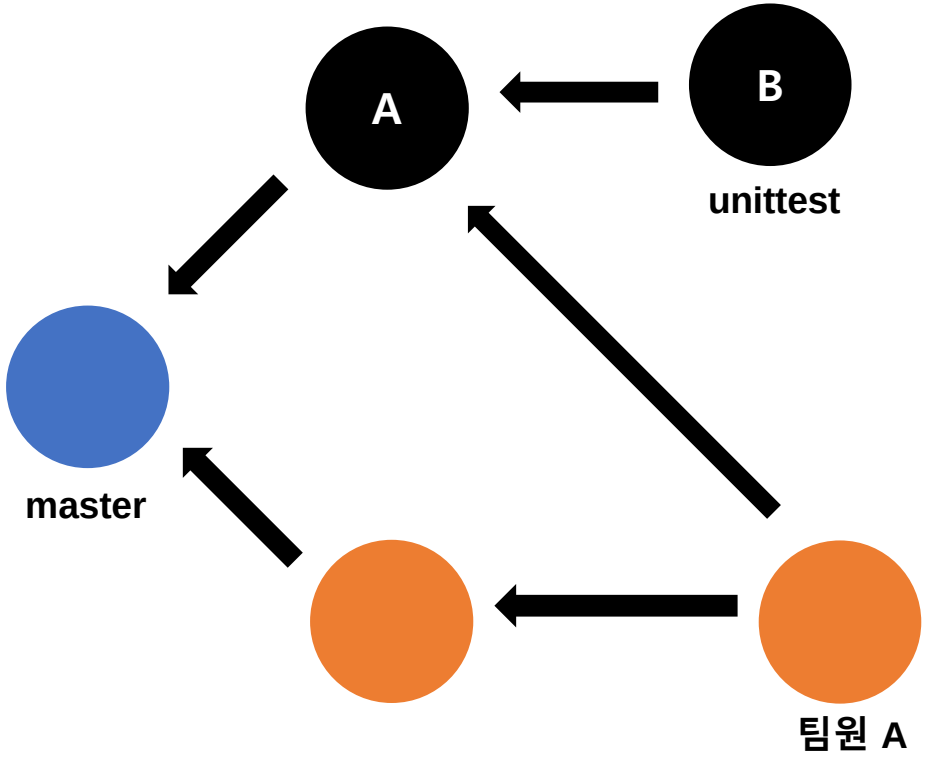
팀장님은 Unittest Branch만 관리합니다.



핑퐁 페어프로그래밍 - Branch 전략 2

팀원 A는 master에서 branch 생성합니다.

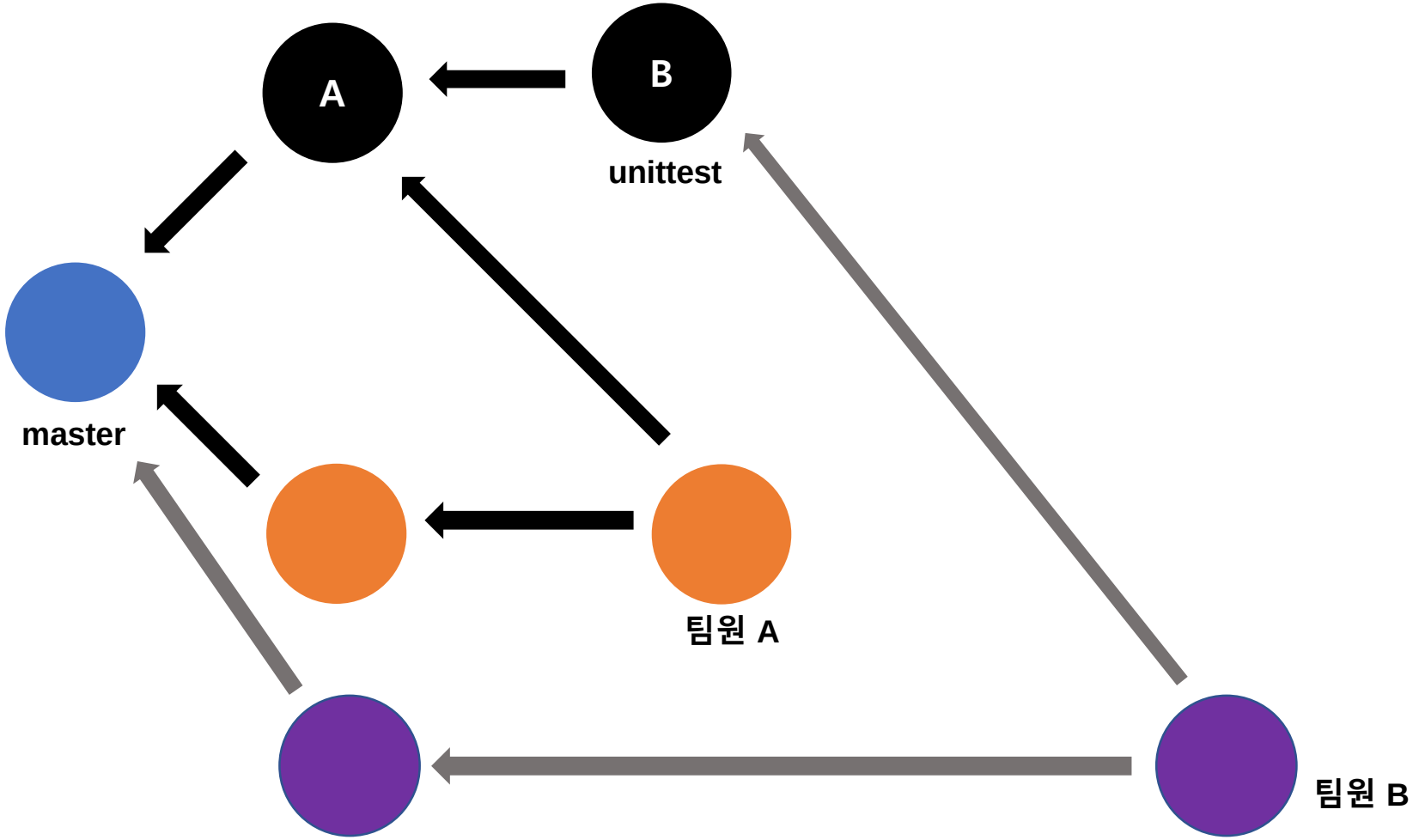
Commit A를 맡기로 합니다. 본인의 Branch 로 A Commit을 Merge합니다.



핑퐁 페어프로그래밍 - Branch 전략 3

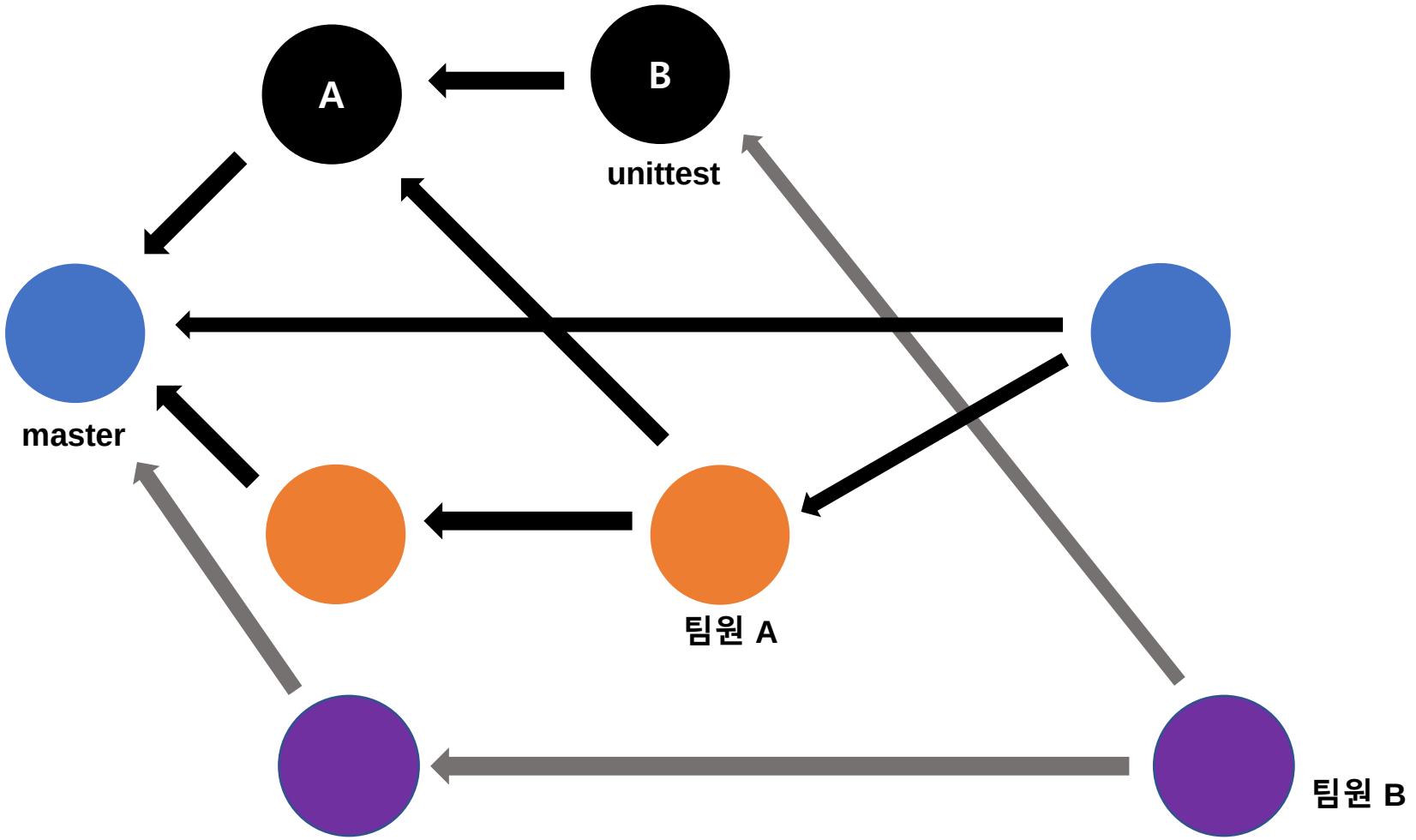
팀원 B는 master에서 branch 생성합니다.

Commit B를 맡기로 합니다. 본인의 Branch 로 B Commit을 Merge합니다.



핑퐁 페어프로그래밍 - Branch 전략 4

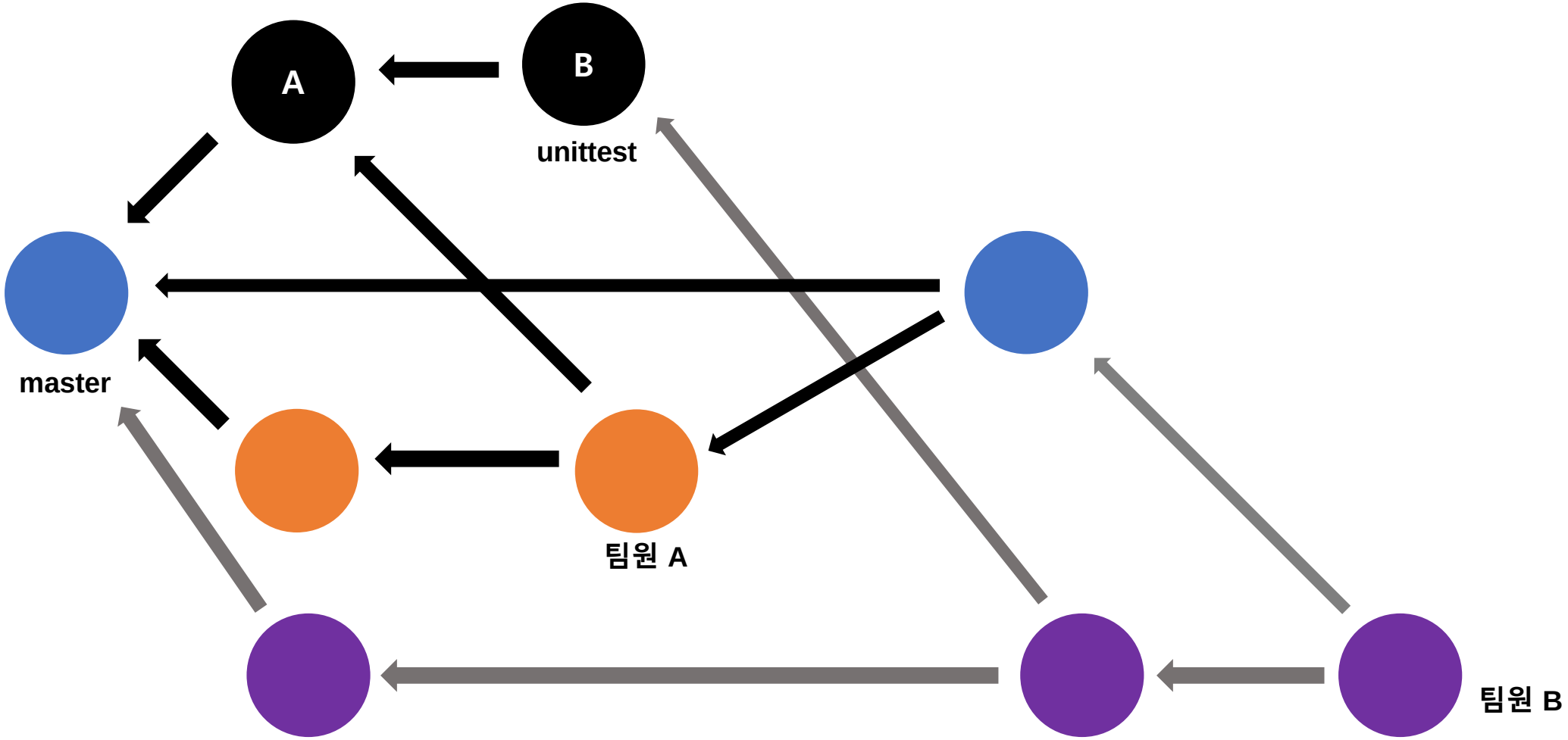
팀원 A의 개발이 끝나고, master에 PR합니다.



핑퐁 페어프로그래밍 - Branch 전략 5

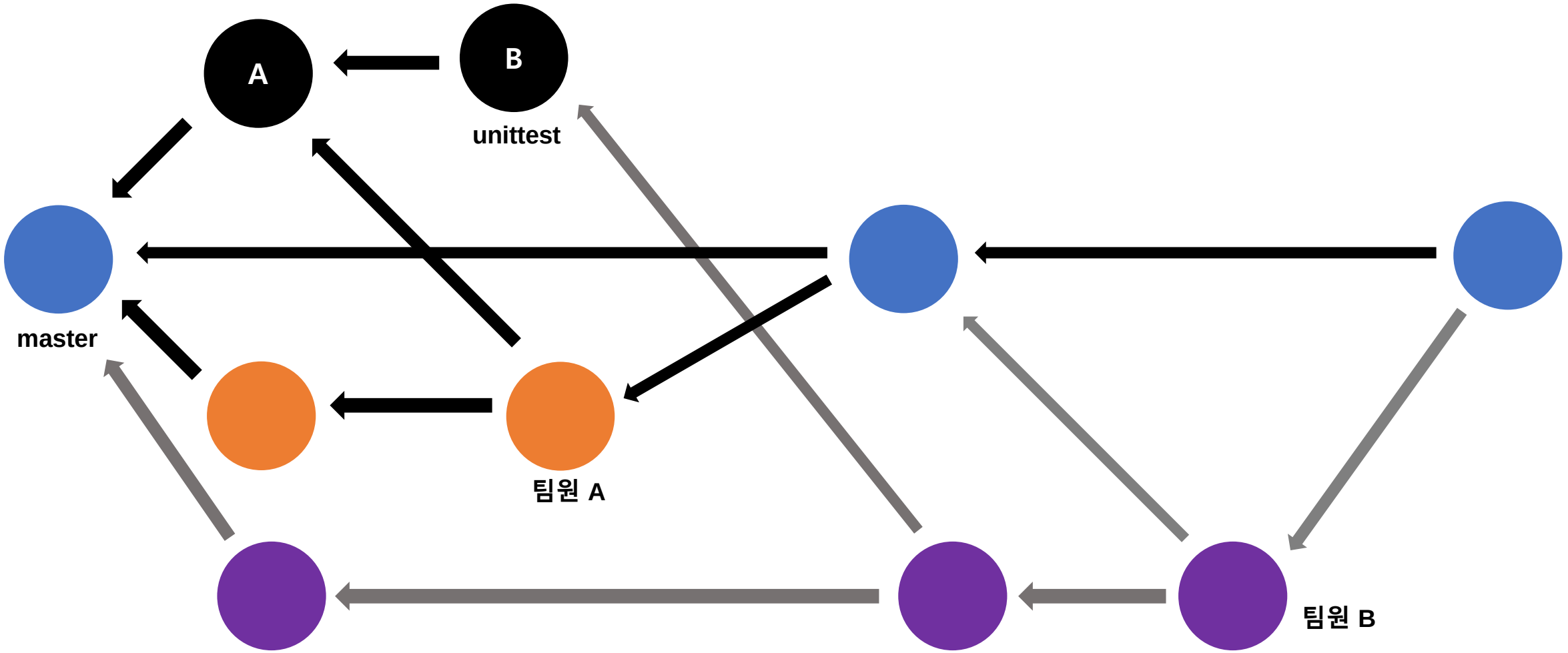
팀원 B의 개발이 끝나고, master에 PR 하기 전

충돌 해결을 Local에서 해결하기 위해 master를 Pull 이후 Merge 합니다.



핑퐁 페어프로그래밍 - Branch 전략 6

충돌 해결이 되었으며, 이제 master에 PR합니다.



Q&A

Q. 명세에 표현이 안된 세부사항은 어떻게 진행할까요?

A. 팀장님이 재량 것 진행해주시면 됩니다.

Q. Kiwer, Nemo Driver 도 UnitTest를 만들어야 할까요?

A. 네, 만들어주세요.

실제로는 비용이 발생되므로 UnitTest를 만들면 안되는데
팀장님과 팀원분 역할이 나누어져 있으므로,
AutoTradingSystem 요구사항을 만족한 후
UnitTest를 추가하여 개발해주세요.

Q. 반환값이 없는 Kiwer, Nemo Driver는 어떻게 검증하나요?

A. 콘솔출력을 검증하는 방법을 써주세요. (Trivia KATA 참조)

Chapter3

끝으로

TDD RIP 사건을 통해, TDD 더 살펴보기

[참고] TDD의 장단점 토론

✔ **TDD is dead** - Ruby on Rails 개발자
(David Heinemeier Hansson)

- https://www.moreagile.net/2014/05/IsTDDdead1.html
- TDD는 Test에 대해 눈뜨게 해주는 좋은 역할이었다.
- Unit Test 에만 중요성이 너무 높게 치중 되어있다.
  느리지만 더 많은 모듈을 한꺼번에 테스트하는 System Test 도 중요하다.

✔ **그 이후, 켄트백, 마틴파울러와의 미팅**

- https://jinson.tistory.com/271

감사합니다.