코드 리뷰체크 리스트.

백명석 & 최범균 강사

체크 그룹

- ☑ 버그/ 장애
- ☑ 기능성
- ☑ 가독성 / 유지보수 용이성
- ☑ 테스트
- ☑ 자료구조
- ☑ 성능
- ☑ 보안
- ☑ 설계

체크 그룹

- ☑ 버그/ 장애
- ☑ 기능성
- ☑ 가독성 / 유지보수 용이성
- ☑ 테스트

버그/장애

- NPE
- Thread Safety
- OOME
- 단위 테스트 없는 중요 로직
- 경계값 테스트
- 외부 URL이나 DB를 n번 호출하는 경우

버그/장애

NPE

equals: code.equals(“”) --> “”.equals(code)

NPE 방지 위해

버그/장애

Thread Safety

멀티쓰레드 환경에 적합한 데이터 구조체를 사용하나 ?

- java.text.XXXFormat
- java.util.LinkedList, java.util.HashMap 등 java.util.concurrent 패키지에 존재하지 않으면서 java.util.Collection을 구현하는 클래스들
- Collections.synchronizedList(), Vector, CopyOnWriteArrayList (변경은 적고 읽기는 많은 경우)
- 모든 output stream
- java.util.Calendar

---

lock을 제대로 사용했나 ?

synchronized, lock, atomic variable 등

버그/장애

Thread Safety

aliasing problem

cache에서 얻은 value object는 immutable해야

경합 발생 가능성이 있나 ?

클래스 상태나 변수의 reassign이 잦은 경우
→ 쓰레드 문제가 생기지 않을까? final로 선언할 수 있지 않을까?

```
public void incrementOrders() {
    orders.setNumberOfOrders(orders.getNumberOfOrders() + 1);
}
```

atomic하지 않으면서 set, get을 함께 하는 경우

버그/장애

OOME

memory leak 가능성이 있나

Static field

- 어플리케이션의 전체 라이프타임과 동일한 라이프타임을 갖음
- 부모 객체가 참조되지 않아도 GC되지 않음

ThreadLocal

- "Sloppy use of thread pools in combination with sloppy use of thread locals can cause unintended object retention, as has been noted in many places"
  - Josh Bloch
- WAS는 thread 재사용을 위해 ThreadPool을 사용
- ThreadLocal#remove를 명시적으로 호출

ClassLoader

- WAS 리스타트 없이 어플리케이션이 재배포/재로딩할 때 발생 가능
- 리스타트시 새로운 클래스 로더가 생성, 이전 클래스 로더는 GC 대상이 됨
  - 하지만 이전 클래스 로더에 대한 모든 참조가 제거되지 않으면
  - ex. WAS 내의 여러 어플리케이션 간에 공유 라이브러리를 사용할 때
- https://bit.ly/3uojD7j / https://bit.ly/3oeRQ7S / https://bit.ly/3tKrsDV

버그/장애

OOME

memory가 지속적으로 증가하는 가능성	• 무한정 증가 가능한 데이터 구조체를 생성하나 ? • 새로운 값이 지속적으로 list나 map 등에 추가되나
connection / stream / session을 close 했나 ?	java7 try-with-resources, finally

resource pool이 제대로 설정되었나 ?	non static inner class	tomcat session-timeout 0

기능성

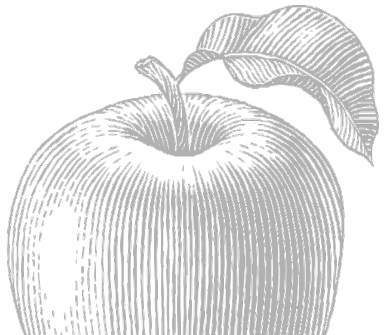
코드가 실제
기대되는 일을 하나 ?

CQS. error code 반환도 위반 exception 처리해야

미묘한 버그

- 검사를 하는데 잘못된 변수를 사용
- `&&` 대신 `ll`, `ll 대신 `l`

선택한 해결책이 요구사항에 적합한가 ?



가독성 / 유지보수 용이성

Formatting / Style

Name
- 실제로 나타내야 하는 것(의도)을 반영하나 ?
- overly short ?

Magic Number

Null을 반환하나 ?

Boolean나 Nullable 파라미터로 분기하나 ?

intellij`s inspections from the command line

https://www.jetbrains.com/help/idea/command-line-code-inspector.html

읽어서 코드가 이해되나 ?
예외 메시지는 이해 가능한가 ?

테스트가 happy paths와 exceptional cases를 모두 다루나

혹시 나중에 사용될지도 모르는 소스를 삭제해야 하는 경우
- comment 처리해서 나중에 쉽게 살리 수 있도록 한다 vs
- 쉽게 언제 뭘 지웠는지가 검색될 수 있도록 commit log를 남기고 지운다
- https://blog.outsider.ne.kr/849

테스트

- ☑ 새로운 / 변경된 코드에 대한 테스트가 존재하나
- ☑ 테스트가 하는 일을 이해할 수 있나 ?

```
@Test
public void testId() throws Exception {
    MyEntity obj = new MyEntity(getDs());
    getDs().save(obj);
    assertEquals(FIRST_ID, obj.getMyLongId());
    obj = new MyEntity(getDs());
    getDs().save(obj);
    assertEquals(SECOND_ID, obj.getMyLongId());
}
```

▲ 이해하기 어려운 테스트

가독성이 좋아진 테스트 ▶

```
@Test
public void shouldIncrementTheEntityIdByOneOnEverySave() throws Exception {
    // when
    MyEntity entity = new MyEntity(getDs());
    getDs().save(entity);

    // then
    assertEquals(1L, entity.getMyLongId());

    // when
    entity = new MyEntity(getDs());
    getDs().save(entity);

    // then
    assertEquals(2L, entity.getMyLongId());
}
```

테스트

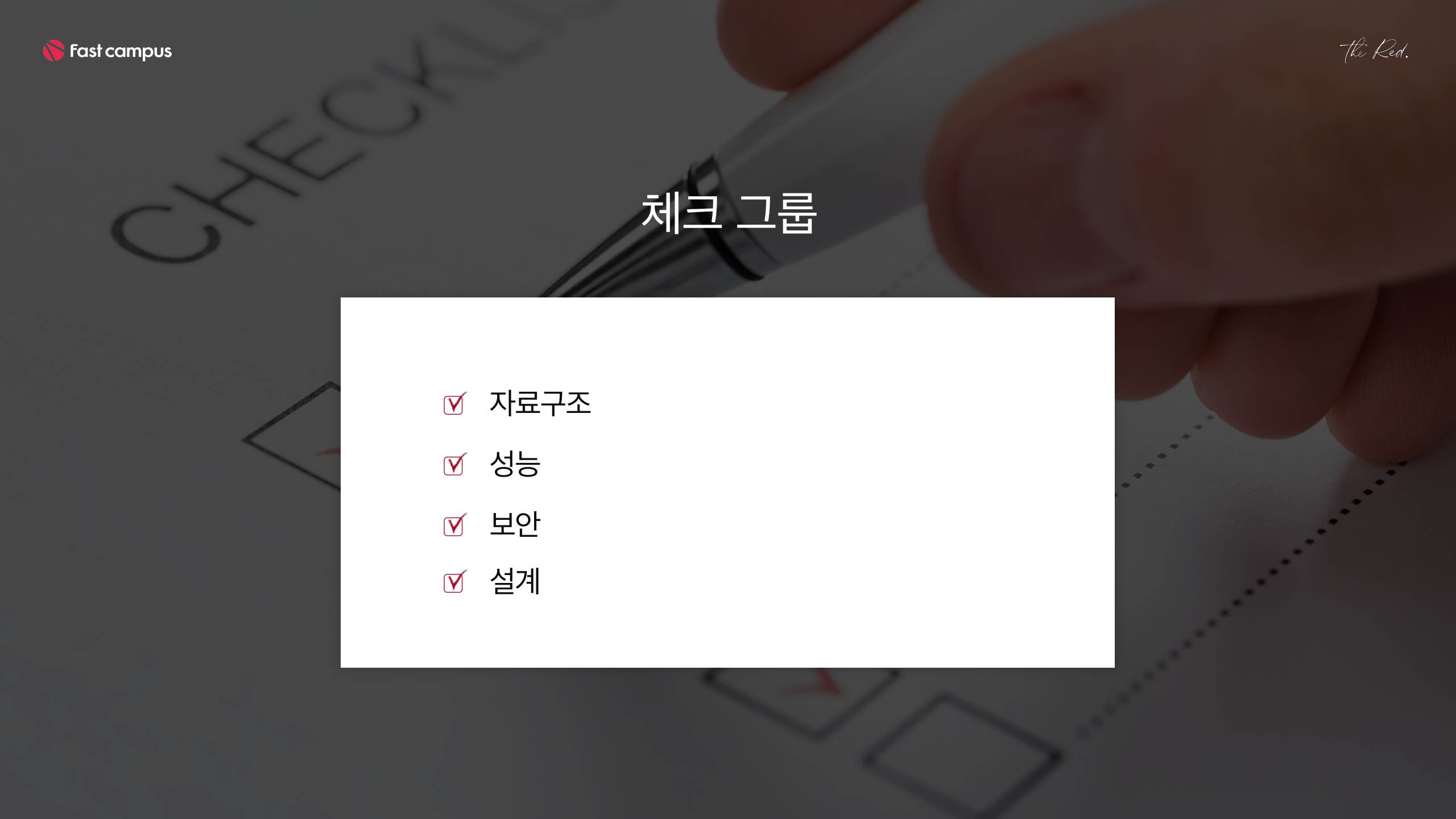
테스트가 요구사항과 매치되나

기존의 테스트들이 커버하지 못하는 테스트를 생각해 낼 수 있나

한계를 문서화하고 있는 테스트가 있나

- 배치가 한번에 1,000개의 항목만 처리 가능한 경우
- 1,000개를 초과하면 예외를 발생시키는 테스트가 존재하나

리뷰어가 테스트를 작성할 수도 있음



체크 그룹

- ☑ 자료구조
- ☑ 성능
- ☑ 보안
- ☑ 설계

자료구조

너무 많은 탐색	list에서 뭔가를 찾기 위해 너무 자주 iterate한다면 map ?
잦은 정렬	결과를 반환하기 전에 항상 정렬한다면 TreeSet?
Map을 전역 상수로 제공	• Map을 외부에 직접 노출 • 원치 않는 키/값이 들어와도 방지 불가(Map에 기능 추가 불가) • get을 하고 null check를 안하면 NPE 가능 • 데이터 구조를 은닉하고 적합한 접근자를 제공할 것을 제안해야
Map의 키로 사용되면	equals, hashCode
Set	equals, hashCode, Comparable

성능

변경 사항이 성능 저하를 유발하지 않나 ?

Prematured Optimization	"We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil." – *Donald Knuth*
외부 호출	• DB, N/W • N/W round trip: app ←→ server
리소스를 효율적/ 효과적으로 사용하나 ?	Lock은 성능 저하 유발 가능
Reflection	사용하지 않는 경우보다 느림
Timeout	

성능

Logging

http://dveamer.github.io/backend/HowToUseSlf4j.html

```
private void bindOneParameter(String userName) {

    // poor performance, poor readability
    logger.debug("Hello " + userName + ".");

    // always good performance, poor readability
    if(logger.isDebugEnabled()) {
        logger.debug("Hello " + userName + ".");
    }

    // always good performance, good readability
    if(logger.isDebugEnabled()) {
        logger.debug("Hello {}.", userName);
    }

    // good performance, best readability - I recommend this.
    logger.debug("Hello {}.", userName);
}
```

보안

Suppress Warning

```
public int generateNewId() {
    @SuppressWarnings("UnsecureRandomNumberGeneration") // 코드리뷰 XXX에서 논의된 바에 의해
    Random random = new Random();
    return random.nextInt();
}
```

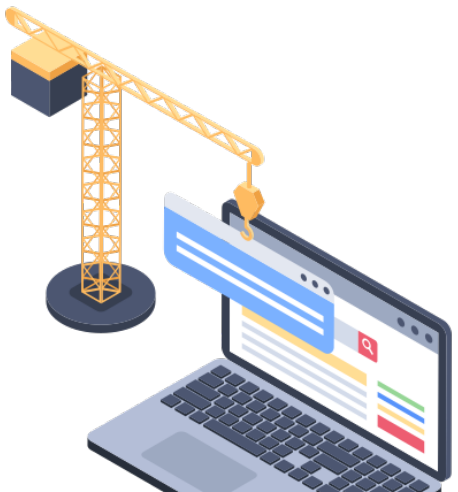
암호화해야 할 데이터가 있는가 ?

DB 접속 암호 등

코드가 로깅 / 감사(auditing) 할 행위를 갖는가 ?

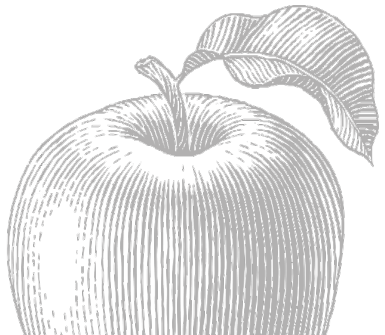
설계

- ☑ SRP(하나의 변경의 원인)를 준수하며 coupling / cohesion을 준수하나
- ☑ 고수준 컴포넌트가 저수준 컴포넌트에게 의존하는지(DIP)
- ☑ 기능 확장을 코드 수정 없이 할 수 있는지(OCP)
- ☑ Feature Envy, Encapsulation, Information Hiding



설계

중복	Locate Duplicates...
오버 엔지니어링	• YAGNI • 재사용 • Premature Abstraction
메타포 역할	• Controller, Service, Dao, Repository • Entity, Value, Factory, Strategy, ...



설계

대칭성 (Symmetry)

- 대칭성이 있는 코드는 비대칭인 코드에 비해 이해하기 쉬움
- Symmetry in code is where the same idea is expressed the same way everywhere it appears in the code.
  (코드의 대칭성은 동일한 아이디어는 코드의 어디서든지 동일한 방식으로 표현되어야 한다는 것임) ← Kent Beck
- 예. "DB에서 마지막으로 갱신된 문서 가져오기"가 코드의 여러 곳에서 나타날 때
  ※ 아래와 같은 경우 대칭성이 깨짐
  - 메소드 이름이 다름
  - 처리 순서가 다름
  - 중요한 차이점이 있음

설계

Composed Method

- 메소드의 모든 오퍼레이션이 같은 수준의 추상화를 갖도록 하는 것

  코드 블록에서 저수준의 할당문과 메소드 호출이 혼합된 경우 할당문을 메소드로 추출하여 추상화
- 이 원칙을 지키면 프로그램을 많은 작은 메소드들로 분리하게 됨
- Refs
  - https://farenda.com/patterns/composed-method-pattern/
  - http://c2.com/ppr/wiki/WikiPagesAboutRefactoring/ComposedMethod.html
  - http://www.informit.com/articles/article.aspx?p=1398607

설계

- ☑ 반복문, 조건문 등에서 2가지 이상의 상태 변경을 하는 경우
- ☑ 파라미터의 수가 3개 이상인 경우 → Parameter Object
- ☑ 불필요한 setter (주로 lombok에 의한)
- ☑ Guard Clause / Early Return으로 복잡도를 줄일 수 있는지 ?
- ☑ Design Pattern