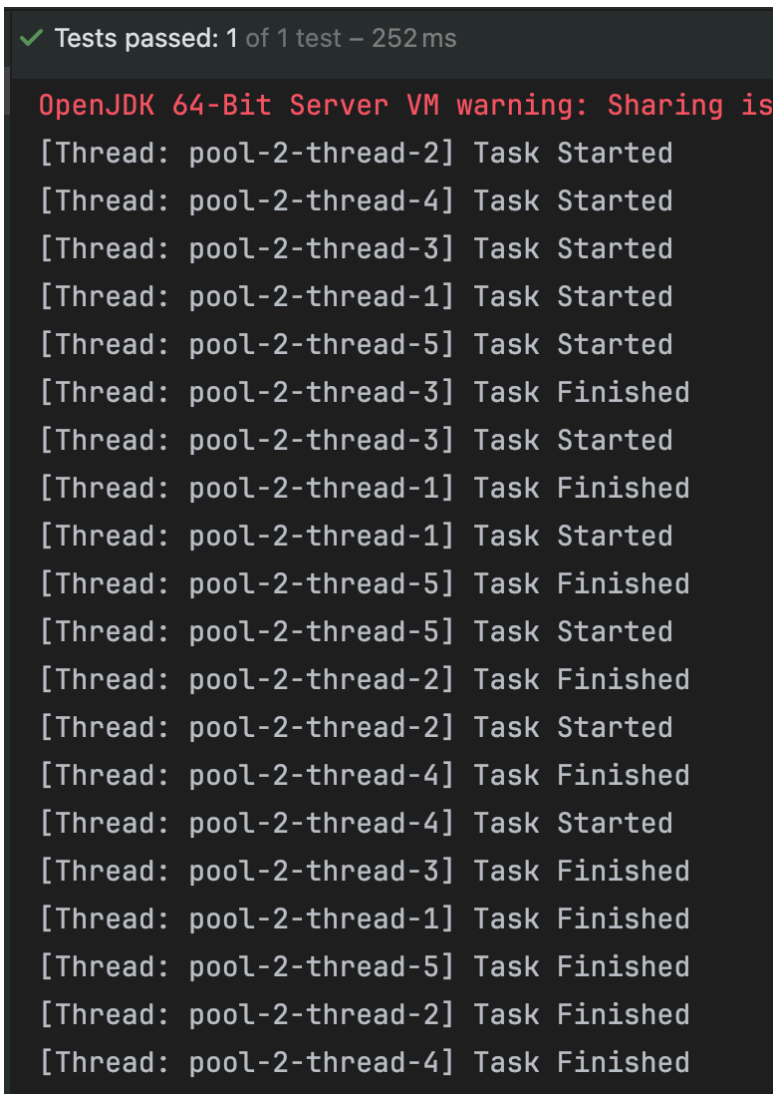
선착순 쿠폰(Coupon) 상세

문제 식별

선착순으로 발급 가능한 쿠폰에 대하여 발급 요청이 동시에 다수 발생할 경우, 잔여 수량 갱신 누락으로 인해 예정된 수량 이상으로 쿠폰이 발급되는 경우 (2.상품 재고 차감과 동일한 상황)
+ 선착순 쿠폰 발행의 경우 동시성에 대한 처리와 동시에
요청이 들어온 순서대로 이후 트랜잭션을 처리해야할 필요가 있음

분석

낙관적 락을 활용하여 재고 감소 수행시 발생할 수 있는 동시성 이슈는 해결되었으나 요청이 순차적으로 진행되고있지않습니다.



해결

낙관적 락(PessimisticLock)을 통하여 데이터베이스 정합성을 확보하였습니다. 추가적으로 애플리케이션 수준에서 요청을 순차적으로 관리하기 위한 별도의 클래스 (`InMemoryCouponIssueQueue`)를 추가하였습니다.

- **InMemoryCouponIssueQueue.java**

```
@Component
public class InMemoryCouponIssueQueue {

    private final BlockingQueue<Command> queue = new LinkedBlockingQueue<>();

        //uuid parameter는 동일 요청에 대한 검증 로깅을 위해 사용되는 임시 파라미터입니다.
    public void enqueue(Long couponId, String uuid) {
        System.out.println("enqueue uuid:  " + uuid + " couponId: " + couponId);
        queue.add(new Command(couponId, uuid));
    }
```

```
public Command dequeue() throws InterruptedException {
    return queue.take();
}

public int size() {
    return queue.size();
}

@Getter
public static class Command {
    Long couponId;
    String uuid;

    public Command(Long couponId, String uuid) {
        this.couponId = couponId;
        this.uuid = uuid;
    }
}
```

- **CouponIssueWorker.java**

```
@Component
@RequiredArgsConstructor
public class CouponIssueWorker {

    private final InMemoryCouponIssueQueue couponIssueQueue;
    private final CouponService couponService;

    @PostConstruct
    public void startWorker() {
        new Thread(() -> {
            System.out.println("[Worker Started]");

            while (true) {
                try {
                    InMemoryCouponIssueQueue.Command command = couponIssueQueue.dequeue();
                    System.out.println("dequeue uuid:  " + command.getUuid() + " couponId: " + command.getCouponId());
                    couponService.deductCouponQuantity(command.getCouponId());
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                    break;
                } catch (Exception e) {
                    System.out.println("[Error] " + e.getMessage());
                }
            }
        }).start();
    }
}
```

- **실행 결과:** enqueue 순서대로 작업이 실행됨

```
✓ Tests passed: 1 of 1 test – 310 ms
OpenJDK 64-Bit Server VM warning: Sha
enqueue uuid:  7fe218 couponId: 1
enqueue uuid:  8eed5a couponId: 1
enqueue uuid:  e87d02 couponId: 1
enqueue uuid:  8bfa3a couponId: 1
enqueue uuid:  71a265 couponId: 1
enqueue uuid:  24188f couponId: 1
enqueue uuid:  a415ad couponId: 1
enqueue uuid:  f4a727 couponId: 1
enqueue uuid:  0ae768 couponId: 1
enqueue uuid:  8dce3a couponId: 1
dequeue uuid:  7fe218 couponId: 1
dequeue uuid:  8eed5a couponId: 1
dequeue uuid:  e87d02 couponId: 1
dequeue uuid:  8bfa3a couponId: 1
dequeue uuid:  71a265 couponId: 1
dequeue uuid:  24188f couponId: 1
dequeue uuid:  a415ad couponId: 1
dequeue uuid:  f4a727 couponId: 1
dequeue uuid:  0ae768 couponId: 1
dequeue uuid:  8dce3a couponId: 1
```

대안

현재 애플리케이션에 구현된 InMemoryCouponIssueQueue의 경우 단일 서버에서는 정상적으로 동작하지만 분산 환경의 경우 각자 다른 InMemoryCouponIssueQueue 객체를 생성하기 때문에 데이터 일관성에 문제가 발생할 수 있습니다. 이 경우 Redis List, Kafka Queue, DB Job Queue 등 분산 환경에서도 데이터 일관성을 보장할 수 있는 기술로의 전환이 필요할 수 있습니다.