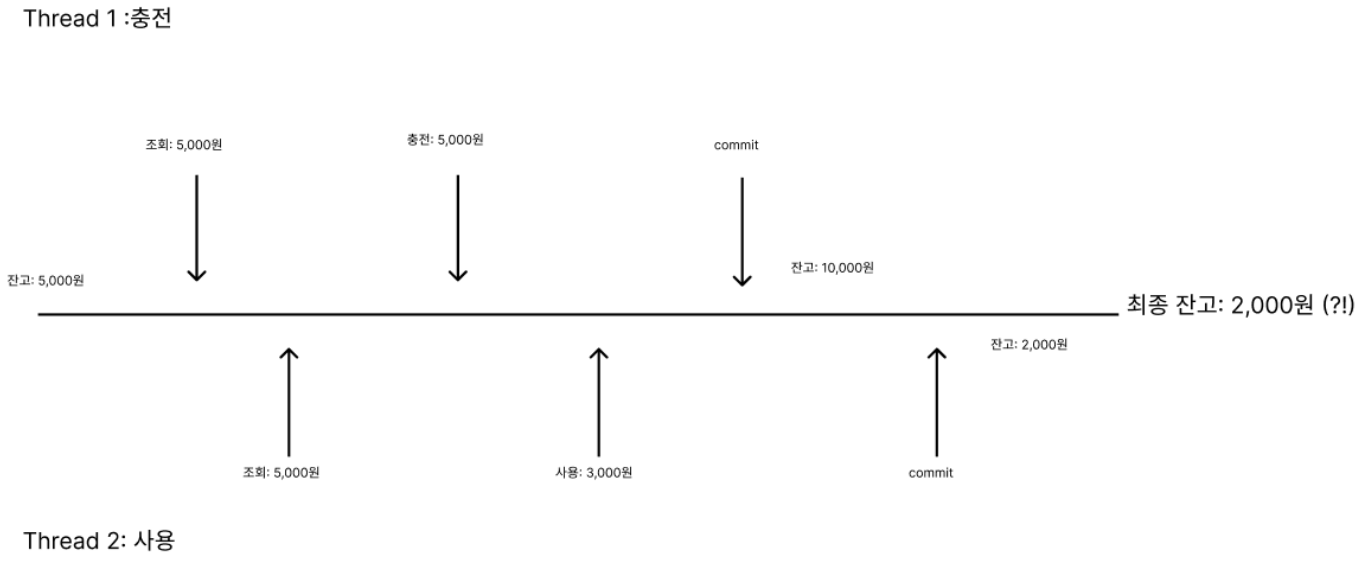
유저의 포인트 잔액(Point) 상세

문제 식별

- 현재 잔고 5,000원인 사용자 포인트에 5,000원 충전 요청과 3,000원 사용 요청이 동시에 발생



- 충전 요청의 기대 잔고 10,000원과는 다른 최종 잔고 2,000원

분석

잔액에 대한 충전과 사용 **트랜잭션이 동시에 수행**되기 때문에 후에 완료된 트랜잭션의 결과만이 데이터베이스에 반영되어 **이전에 수행된 트랙잭션의 결과가 유실**되는 분실 갱신(Lost Update) 발생

- 사용자 포인트 충전/차감 동시성테스트

```
@Test
void 사용자포인트_사용_충전_동시성테스트() throws InterruptedException{

    // given
    UserCommand.Charge chargeCommand = new UserCommand.Charge(testUser.getId(), 500);
    UserCommand.Deduct deductCommand = new UserCommand.Deduct(testUser.getId(), 500);

    List<Runnable> tasks = List.of(
            () -> userService.charge(chargeCommand),
            () -> userService.deduct(deductCommand)
    );


    // when 각 TASK를 100번씩 실행하기 위해 200개의 스레드 생성
    ConcurrentTestExecutor.execute(200, 100, tasks);
    User updatedUser = userRepository.findById(testUser.getId())
            .orElseThrow();

    // then
    System.out.println("최종 포인트: " + updatedUser.point());
    assertEquals(50_000, updatedUser.point());
}
```

- **실행 결과**
  동일한 금액(500원)의 사용과 충전을 각 100회씩 반복하여 최종 금액은 50,000원으로 초기 금액과 동일해야하나 동시성 이슈 발생으로 예상과는 다른 잔고액 조회

```
org.opentest4j.AssertionFailedError:
Expected :50000
Actual   :54500
```

해결

잔액(Point)의 경우 소유한 사용자(User)만이 접근 가능한 상황이기 때문에 경합상황이 빈번하지 않을 것으로 판단되어 **낙관적 락 (@Version 및 컬럼 추가) 적용**

- Point.java

```
public class Point {

    public static final Integer MINIMUM_CHARGE_AMOUNT = 100;
    public static final Integer MAXIMUM_BALANCE = 100_000_000;

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "point_id")
    private Long id;

    @OneToOne(fetch = FetchType.EAGER, cascade = CascadeType.ALL, orphanRemoval = true)
    @JoinColumn(name = "user_id", foreignKey = @ForeignKey(ConstraintMode.NO_CONSTRAINT))
    private User user;

    @Column(name = "pointAmount", nullable = false)
    private Integer pointAmount;

    @Version
    private Integer version; //← @Version 추가

        //....

}
```

- 실제 update query

```
Hibernate: update point set point_amount=?,user_id=?,version=? where point_id=? and version=?
```

- 낙관적 락 적용 후 **동시성 이슈** 발생 시

`Exception in thread "pool-2-thread-1" org.springframework.orm.ObjectOptimisticLockingFailureException`

```
Exception in thread "pool-2-thread-2" org.springframework.orm.ObjectOptimisticLockingFailureException Create breakpoint : Row was updated or deleted
    at org.springframework.orm.jpa.vendor.HibernateJpaDialect.convertHibernateAccessException(HibernateJpaDialect.java:325)
    at org.springframework.orm.jpa.vendor.HibernateJpaDialect.translateExceptionIfPossible(HibernateJpaDialect.java:244)
    at org.springframework.orm.jpa.JpaTransactionManager.doCommit(JpaTransactionManager.java:566)
    at org.springframework.transaction.support.AbstractPlatformTransactionManager.processCommit(AbstractPlatformTransactionManager.java:795)
    at org.springframework.transaction.support.AbstractPlatformTransactionManager.commit(AbstractPlatformTransactionManager.java:758)
    at org.springframework.transaction.interceptor.TransactionAspectSupport.commitTransactionAfterReturning(TransactionAspectSupport.java:698)
    at org.springframework.transaction.interceptor.TransactionAspectSupport.invokeWithinTransaction(TransactionAspectSupport.java:416)
    at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:119)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:184)
    at org.springframework.aop.framework.CglibAopProxy$DynamicAdvisedInterceptor.intercept(CglibAopProxy.java:727)
    at kr.hhplus.be.server.domain.user.UserService$$SpringCGLIB$$0.deduct(<generated>)
    at kr.hhplus.be.server.concurrency.point.PointConcurrencyTest.lambda$사용자포인트_사용_충전_동시성테스트$1(PointConcurrencyTest.java:43)
    at kr.hhplus.be.server.concurrency.support.ConcurrentTestExecutor.lambda$execute$0(ConcurrentTestExecutor.java:18) <3 internal lines>
```

> !! 낙관적 락(Optimistic Lock)의 경우 실패한 트랜잭션에 대한 재시도를 자동 처리하고있지 않아 애플리케이션 수준에서의 트랜잭션 retry로직의 추가가 필요합니다.
> 만약 JPA를 사용하고 있다면, 포인트 업데이트를 위한 객체를 영속석 컨텍스트에서 조회하여 앞선 트랜잭션이 반영되지 않은 이전의 값을 읽어와 또다른 값의 불일치 문제가 발생할 수 있어 주의가 필요합니다.

대안

사용자의 행위의 경우 인증 사용자 세션 개수의 제한을 둘 경우 동시성 이슈를 사전에 방지할 수 있으므로, 동시에 인증 가능한 사용자의 수를 1명으로 제한하는 정책을 추가하는 방식으로 해당 이슈를 해결할 수 있습니다.