로또 미션

[OOP] 원시값 포장 - 3

내용

- LottoNumber 객체와 Money 객체를 도출하여 모든 원시값과 문자열을 포장한다. 규칙을 지킴
- LottoNumber와 관련된 비즈니스 로직을 Lotto에서 분리할 수 있었음
- Money의 경우 구입 금액을 입력받는 부분에서 따로 분리할 수 있었음

[JDK] ENUM - 2

내용

- ENUM 클래스를 활용하여 로또 당첨금을 구분함
- 자료구조 EnumMap을 활용하여 당첨 번호와 금액을 저장해주었음

[JDK] Stream API - 4

내용

- 로또 번호를 생성할 때 중복 방지를 위해 Stream을 사용함
- 로또 랭크를 반환하기 위해 Stream을 사용하여 맞춘 번호에 따라 LottoRank를 반환해줌
- 그 외에도 여러 부분에서 Stream을 사용한 결과 가독성과 편리성이 뛰어나다는 것을 느낌
- Stream을 접한지 얼마 되지 않아 아직은 내가 원하는데로 사용하기 어려워 더 공부해야겠다고 생각함

[TDD] 테스트 주도 개발 - 5

내용

- View를 제외한 모든 부분을 TDD 형식으로 개발함
- View 또한 테스트 주도 개발을 적용할 수 있을까? 라는 생각이 들었음

- 코드를 리팩토링 하거나 변경하고 난 후 정상 코드인지 확인할 때 미리 테스트 코드를 만들어 놓으니 확인이 쉬워졌음

[OOP] Exeception 예외 처리 - 2

내용

- LottoNumber와 LottoPrice를 상위 클래스인 LottoExeception으로부터 받아 예외처리를 해주었음

[자료구조] EnumMap - 2

내용

- LottoResult에서 각 LottoRank 와 당첨 횟수를 연결시켜주기 위해 EnumMap을 사용하여 당첨금과 횟수를 묶어줌

[Object] 일급 컬렉션 - 5

내용

- LottoGroup을 외부에서 수정되지 않도록 일급컬렉션으로 감싸주었다.
- 일급 컬렉션의 특징 중 불변성을 유지하기 위해 add메소드를 삭제하고 생성자를 통해 초기화를 시켜주었다.
- 수동 구매를 구현하는 과정에서 로또 자동, 수동의 개수를 인스턴스 변수로 추가시켜주었는데 일급 컬렉션을 유지시켜주기 위해 삭제하였다.

[JDK] Stream API - 3

내용

- Stream을 사용하면서 다양한 메소드 활용이 아직 어렵다는 것을 느꼈다.
- 이번 로또 번호를 수동으로 생성하는 과정에서 map을 이용하여 int로 변경해주는 것을 원했는데, 아직 익숙치 못한 Stream 때문에 map을 사용하여 바로 원하는 결과를 얻을 수 있음에도 mapToInt, boxed(IntStream을 일반 Stream으로 변환)를 사용하여 잘못 활용하고 있어 Stream에 대한 이해가 부족하다는 것을 느꼈다.

[Pattern] MVC 모델 - 3

내용

- MVC 모델은 Model, View, Controller로 나뉘는데 간단하게 소개하자면 Model과 View는 서로 직접적으로 접근하지 말고, 중간 매개로 Controller를 사용하여 구현하는 모델입니다.
- 이번 로또를 구현하면서 MVC 모델을 적용하려고 애썼지만 피드백을 받으면서 아직 부족하다는 사실을 느꼈다.
- 이번 로또 미션을 진행하면서 InputView에서 LottoGenerate 도메인 모델을 불러 쓰거나, 일관되지 않게 InputView를 사용하여 입력받는 등 도메인과 뷰 그리고 컨트롤러를 명확히 분리하지 않았다. 이러한 부분을 리뷰어의 피드백을 통해 내 코드에 적용하며 MVC 모델을 조금 더 잘 이해할 수 있었다.

블랙잭 미션

[자료구조] LinkedHashMap - 1

내용

- 게임 결과를 저장하고 출력할 때 기존 HashMap 특성 상 순서가 지켜지지 않아 순서가 지켜지는 LinkedHashMap을 사용하였다.

[JDK] Stream API - 3

내용

- 카드 덱에서 52장의 카드를 생성할 때 문양과 숫자를 선택하는데 Stream 내부에 Stream을 한번 더 사용함으로써 짝을 맞추어 주었다. 이를 하기 위해서는 flatMap 메서드를 사용해야 했는데 평면화에 대해 알 수 있었다.
- indent를 지키고 가독성을 위해 점수 계산, 카드 생성 등을 Stream API를 이용하여 구현하였다.

[OOP] 상속 - 3

내용

- Player와 Dealer 객체에서 중복되는 코드를 상속을 이용하여 구현하였다.

[JAVA] Enum - 2

내용

- 카드 문양과 숫자 그리고 게임 결과(승, 무, 패)를 Enum을 적용하여 구현하였다.
- 게임 결과 Enum은 내부적으로 게임 결과를 판단하는 메서드를 Override하여 구현해보았다.

[OOP] 일급 컬렉션 - 1

내용

- Card와 Player를 일급 컬렉션으로 구현하였다.

[TDD] 테스트 주도 개발 - 3

내용

- 테스트 코드를 먼저 작성하고 코드를 만들어가는 방식으로 개발하였다.
- Enum이나 결과를 출력하는 클래스 등 테스트 코드를 먼저 작성하기 어려운 부분들을 TDD 방식으로 구현하지 못하여 아쉬웠다.
- TDD 방식은 앞서 해본 미션으로 코드를 바꾸더라도 바로 테스트하기 쉽다는 장점을 깨달았기 때문에 이번 블랙잭 미션에서도 장점을 잘 활용했다.

[Design Pattern] 상태 패턴 - 3

내용

- 카드를 뽑는 결과에 따라 상태 패턴을 이용하여 Blackjack, Bust 등 플레이어 상태를 정할 수 있도록 구현함.
- 디자인 패턴을 처음 적용해보는 것이라 다소 어려움이 있었지만 경험해볼 수 있어서 좋았음.

체스 미션

[Design Pattern] 상태 패턴 - 3

내용

- 체스 게임의 진행 상황을 상태 패턴을 활용하여 구현해보는 연습을 하였다.
- 상태는 시작 전, 진행 중, 종료 상태를 가지고 있다.
- 아직 많이 익숙하지 않아 어려움을 겪었지만, 경험해본다는 의의를 두고 구현하였다.

[OOP] 상속 - 3

내용

- 게임에 사용되는 체스 말들을 Piece 객체를 상속 받도록 구현하였다.

[JAVA] Enum - 2

내용

- 보드의 행렬을 나타내는 Column, Row와 흑백 진영을 나타내는 Side를 Enum으로 구현

[TDD] 테스트 주도 개발 - 5

내용

- 항상 미션을 하면서 TDD를 적용하려고 노력했지만 이번 체스 게임에서 난이도 때문인지 TDD를 잘 적용하지 못하였다. 그러나 테스트 코드는 만족스럽게 작성했던 것 같다.
- 이전부터 느낀 점인데 구현 난이도가 올라갈 수록 TDD를 적용하는데 어려움이 느껴지는 것 같다. 조금 더 연습이 필요하다고 느꼈다.

[JDK] Stream API - 3

내용

- for문 또는 if문 보다 stream을 최대한 자주 쓰려고 노력하고 있다.
- Winner 반환, Score 계산 등 많은 부분에서 쓰였다.

[Exception] 예외 처리 - 2

내용

- 체스게임에서 나올 수 있는 다양한 상황에 대해 새로 ChessException을 만들어 예외 처리를 해주었다.
- ChessException에 RuntimeException을 상속받아 최상위로 두고 나머지 - Exception이 ChessException을 상속하는 방식으로 처리하였다.

[OOP] Interface - 2

내용

- 명령에 대한 정보를 가지고 있는 Command를 interface를 이용하여 start, move, end, status 체스 게임에서 사용 되는 명령어들을 구현하였다.

JWP 체스

[JAVA] Optional

내용

- DB 조회 시 없는 결과를 반환할 때 빈 값을 반환한다. 이 때 Optional을 이용하여 null 값을 처리해주었다.
- Optional에 빈 값이 들어갈 경우 orElseThrow를 사용하여 예외를 반환시켜주었다.

[데이터] DTO

내용

- 데이터를 보낼 때 DTO를 만들어 보내고자 하는 데이터를 감싸주었다.

[Spring] ResponseEntity

내용

- Controller에서 보낼 경우 ResponseEntity를 이용하여 HTTP 상태 코드와 함께 데이터를 보냈다.

[DB] jdbcTemplate

내용

- jdbcTemplate를 사용하여 DB에 접근해서 정보를 변경 및 추가, 삭제했다.
- query, queryForList, update 등 jdbcTemplate에서 제공하는 다양한 메서드를 이용해서 쿼리를 보냈다.

[Spring] Spring

내용

- GetMapping, PostMapping을 이용하여 데이터를 매핑해주었다.