**객체지향프로그래밍 2분반**

**하트 게임 구현 보고서**

[#1] 규칙 분석 및 구현 방향 설정

현재 프로젝트에서 구현해야 하는 다양한 규칙에 대해 고려하자.

1. 각 카드가 가진 힘 :

기본적으로 각 카드는 자기가 가진 숫자에 비례하는 힘을 가지나, A를 나타내는 카드 1 은 현재 규칙에서 가장 강한 힘을 가지고 있다. 따라서, 카드의 힘을 score라고 할 때, 각 카드는 자신의 숫자 – 1의 값을 반환하고, A는 13(기존의 가장 큰 숫자) 를 반환하게 하여 순서를 맞춘다.

2. 카드 나눠 가지기:

카드의 수는 플레이어의 수와 어느 정도 반대되는 관계를 가진다. 정확히 말하자면, 1명당 배정되는 카드의 수는 int (52(트럼프 카드의 수) / 플레이어 수) 와 같다. 이때, 4명 이상의 경우 필연적으로 남는 카드가 발생한다. 해당 카드는 원래 카드 덱에서 벗어나 임시 덱에 위치하기로 한다.

이때, 클로버 2의 경우 반드시 필요한 카드이므로, 필요 없는 카드를 제거할 때 해당 카드인 경우는 배제하고 카드를 제거한다.

3. 플레이어의 순서

플레이어의 순서는 클로버 2를 가졌는가 여부에 따라 달라진다. 이때 기존 과제에서 플레이어는 ArrayList을 통해 관리되는데, 클로버 2를 가진 플레이어를 1번째 순서로 만들기 위해 해당 플레이어 이전까지 인덱스 0 에서 플레이어를 제거하고, 해당 플레이어를 리스트에 넣는 동작을 반복한다.

4. 카드의 조건.

각 카드는 (1) 에서 언급한 우열을 가진다. 이때, 각 트릭에서 카드를 가져가게 되는 사람을 설정하기 위해 1. 카드의 모양을 비교하고, 2. 카드의 힘을 비교하여 카드를 가져가게 될 사람의 인덱스를 설정해야 한다.

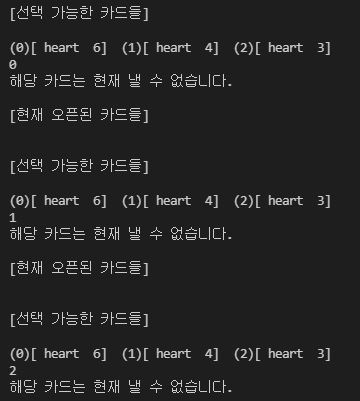
5. 하트는 이전 트릭에서 나온 경우에만 맨 처음 낼 수 있다. 따라서 현재 하트를 낼 수 있는지에 관한 불린 변수를 하나 두고, 이를 통해 카드의 제출 여부를 설정한다.

6. 한 사람이 Q를 4번 모으는 경우를 처리하기 위해 각 플레이어와 연관된 변수를 설정하고, 이 값에 따라 Q에 의한 승리 여부를 반환한다.

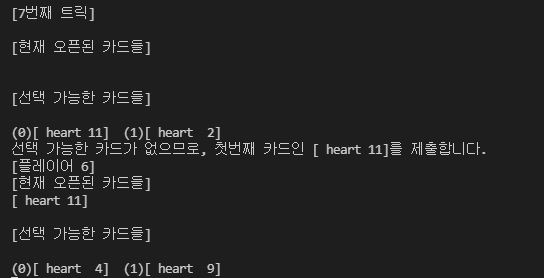
7. 승자가 결정되는 경우는 (6) 혹은 30점 이상의 점수를 보유한 사람이 나타난 경우이다. 따라서 이 두 경우에 대해 검사하는 영역이 필요하다.

위와 같은 경우에 대해 몇 개의 문제가 발생할 수 있다고 생각했는데, 다음과 같다.

8. 5 조건에서 하트 밖에 카드가 없는 경우.



(하트 카드밖에 없어서 아무런 카드도 낼 수 없다.)



(카드가 없다면, 강제로 제출하도록 만든다.)

하트 게임에서 보유한 카드를 내는 것은 자신의 자율로 진행할 수 있다. 따라서, 하트만 남는 상황에 대해 플레이어는 자신의 자율적 선택에 대한 책임을 져야 한다. 따라서 해당 상황에서는 선택 없이 존재하는 카드 중 첫번째 카드를 반드시 제출하도록 한다.

9. 7 상황에서 동점자가 2명 이상 발생하는 경우.

하트 게임의 각 트릭에서 카드는 나중에 낼수록 선택지가 넓어진다고 생각한다. 따라서, 순서상 나중에 낼 수 있는 것으로도 충분히 메리트가 존재한다면, 동점의 상황에서는 메리트가 상대적으로 적은 순서상 앞의 사람이 승리하는 것이 옳다고 가정한다. 따라서, 동점자가 발생하는 경우, 게임이 끝난 트릭에서 순서가 앞에 있는 사람을 승리자로 처리한다.

[#2] 코드에 대한 설명

현재 코드는 이전에 제출했던 과제와 유사한 부분이 많다. 따라서, 현재 레포트에서는 위에서 언급한 조건과 관련된 부분 위주로 설명하며, 과거에 비해 변경된 점이 거의 없는 경우, 코드 설명을 생략한다.

**(01) Card 클래스**

텍스트이(가) 표시된 사진

자동 생성된 설명

현재 프로젝트에서 사용되는 각각의 카드 한 장에 대응되는 클래스이다.

해당 클래스는 모양 shape 및 카드의 숫자 number 을 가진다.

이때 show를 제외한 메서드들은 해당 인스턴스 변수를 위한 get/set 메서드이며, show는 해당 카드를 특정 포맷을 가진 String으로 반환한다.

**(02) CardDeck 클래스**

텍스트이(가) 표시된 사진

자동 생성된 설명

CardDeck 클래스는 위에서 언급한 Card 클래스 객체를 담고, 이를 관리하는 목적을 가진 클래스로, Card 객체를 card\_deck이라는 ArrayList를 이용하여 관리한다.

텍스트, 검은색, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

MakeCardDeck : 52장의 트럼프 카드를 설정하고, 이를 카드 덱에 넣는 역할을 수행한다.

shuffle : 카드 덱을 섞는 역할을 한다.

userGetCard : 유저가 count 장의 카드를 뽑는 동작을 수행한다.

getOneCard: 카드 한 장을 반환한다. 카드가 없다면 null을 반환한다.

showAllCard : 현재 덱 / 제공된 덱에 존재하는 모든 카드를 출력한다.

pushCard : 카드를 덱에 넣는다.

**(03) Player 클래스**

텍스트이(가) 표시된 사진

자동 생성된 설명  
Player 클래스는 카드 게임을 수행하는 플레이어와 관련된 정보를 담는 클래스이다.

각 플레이어는 이름, 나이, 직업(name, age, job) 정보를 가지며, 각자가 가지고 있는 카드의 정보를 ArrayList 타입의 cardDeck을 통해 관리한다.

텍스트, 화면이(가) 표시된 사진

자동 생성된 설명

생성자 및 Init 메서드 : 플레이어의 개인 정보를 설정하고, cardDeck을 초기화한다.

getName : 이름을 전달받는다.

Card\_draw : 덱에서 카드를 한 장 뽑는다. 이 과정이 성공적이면 true를, 아니면 false를 반환한다.

remain\_card : 남은 카드 수를 반환한다.  
텍스트, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

have\_card: 해당 카드를 가지고 있는지 여부를 반환하는 메서드이다. 해당 메서드는 맨 처음 플레이어를 지정하는 클로버 2 카드를 가지고 있는지 검사하기 위해 사용된다.

텍스트, 실내, 스크린샷, 여러개이(가) 표시된 사진

자동 생성된 설명

submitCard: 인덱스에 해당하는 카드를 cardDeck에서 제거하고, 이를 반환한다.

showCards : 가진 카드를 출력한다. 플레이어가 제출할 수 있는 카드를 보여줄 때 사용된다.

showCardInfo : index에 해당하는 카드를 반환한다.

showPlayerInfo : 플레이어 자체와 관련된 정보(이름, 나이, 직업) 을 반환한다.

cardAll: 모든 카드가 func의 조건을 만족하는지 여부를 반환한다. 모든 카드가 하트인지 여부를 알기 위해 사용되는 메서드이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**(04) InputHelper**

텍스트이(가) 표시된 사진

자동 생성된 설명

들어온 변수가 유효한지 여부를 검사하고, 결과를 반환하는 역할을 하는 클래스이다. Scanner을 기반으로 작동하며, 현재 int 및 String 타입에 대해 관리하고 있다.

getString : 단순히 scanner.next() 을 반환한다.

getInt : 들어온 문자열을 try-catch 기반으로 검사, 숫자를 입력했으면 이를 반환한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**(05) CardGame 클래스**

현재 클래스들 중 가장 중추의 역할을 수행하는 클래스이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

players : 플레이어를 관리하는 리스트 변수이다.

card\_deck : 현재 트럼프 카드 52장이 생성되는 카드 덱이다.

remain\_card\_deck : 이미 사용된 카드 및 플레이어 수가 5, 6명인 경우 남는 카드를 저장하는 변수로, 새로운 게임마다 새롭게 Card 클래스를 생성하는 대신 해당 카드들을 보관하기 위해 고안된 변수이다.

num\_player : 플레이어의 수에 대한 정보를 가진 변수다.

num\_card : 플레이어 한 명당 가지게 되는 카드 정보를 담은 변수다.

helper : 현재 클래스 내에서 발생하는 모든 input을 관리하기 위해 사용하는 변수로, 현재 클래스에서는 int 형 변수 및 String 객체만이 의미를 가지기 때문에, 현재의 InputHelper은 int와 string에 대응되는 get~ 메서드만을 가진다.

**(05-2) CardGame과 게임의 흐름**

CardGame 클래스는 게임의 흐름과 밀접한 관계가 있다. 따라서, 해당 클래스의 메서드는 게임의 흐름에 따라 설명한다.

main 메서드:

텍스트이(가) 표시된 사진

자동 생성된 설명

게임 객체를 생성하고, PlayGame 메서드를 통해 승자의 인덱스를 받아온다. 해당 인덱스 결과에 기반하여 승자에 대한 문장을 출력한다.

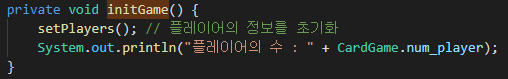
생성자 :

텍스트이(가) 표시된 사진

자동 생성된 설명

본격적인 게임을 시작하기 이전에, 변수의 할당 및 간단한 값의 초기화를 수행한다.

initGame: setPlayer을 통해 플레이어 정보를 초기화한다.



initGame 메서드는 플레이어의 수 및 플레이어와 관련된 정보를 저장하는데, 이 과정에서 플레이어의 수를 나타내는 num\_player 및 각 플레이어가 나눠 가지는 카드의 수를 의미하는 num\_card를 초기화하며, 플레이어와 관련된 정보를 설정하게 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

setPlayerNum : 플레이어의 수를 입력 받고, 이를 통해 num\_player 및 num\_card 클래스 변수를 초기화하는 역할을 한다. 플레이어의 수는 기존에 설명한 InputHelper 클래스를 통해 얻으며, 해당 값이 4 ~ 6 사이(조건에 제시된 플레이어의 수) 인지 검사하여, 해당 값으로 num\_player을 설정한다. num\_card의 경우 52 / num\_player 으로 수행한다.

setPlayers : setPlayerNum을 통해 지정된 플레이어 수에 맞게 플레이어를 players에 추가한다.

위와 같은 방식으로 다양한 값이 초기화된 이후, PlayGame을 통해 게임이 진행된다.

PlayGame: 각 게임은 라운드와 트릭 단위로 진행된다. 트릭은 각 유저가 카드를 한 장씩 내는 사이클을 뜻하고, 라운드는 모든 플레이어가 트릭을 통해 가진 모든 카드를 소모하는 사이클을 뜻한다. 이때 라운드는 승패를 가릴 수 있는 조건(1명이 30점 이상 득점 or Q를 1라운드 안에 4장 모은 사람이 등장) 에 따라 갈리게 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

각 게임을 시작하기 이전에, 플레이어의 점수와 관련된 초기화를 진행해야 한다. 이때, 규칙상 플레이어의 순서는 계속해서 변경된다. 따라서, 플레이어의 점수를 ArrayList로 저장하면, 플레이어의 순서를 바꿀 때마다 player\_score 변수 내의 순서도 바꿔야 하는 불상사가 생기기 때문에, HashMap을 이용하여 플레이어에 따른 점수를 어떤 상황에서도 얻을 수 있도록 설정한다.

play\_score에는 플레이어와 관련된 정보를 담기 위해 이를 K-V 형태로 담는다.

각 게임의 라운드는 while문을 통해 구현된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

한번의 라운드에도 다양한 변수가 필요하다.

round\_player\_score : 한번의 라운드에서 각 유저가 얻게 되는 점수를 나타낸다. 현재 게임에 제시된 조건(각 라운드가 끝날 때 점수를 합산) 을 만족하기 위해 player\_score 이외로 선언한 변수다.

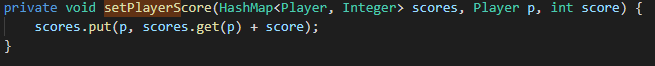
해당 변수는 setPlayerScore 메서드를 통해 초기화할 수 있다.

setPlayerScore : 플레이어가 가진 점수를 초기화하는 메서드이다. 두가지로 오버로딩 되었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

scores를 각 플레이어에 대응되도록 초기화하기 위한 메서드이다.



특정 플레이어의 점수를 설정하기 위한 메서드로, 입력 받은 score을 플레이어의 기존 점수에 더하는 역할을 수행한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이후 initRound 함수를 통해 게임을 초기화하며, 게임 종료 조건 중 하나인 Q 4개를 모으는 조건을 처리하기 위해 Q\_conditions을 설정하고 초기화한다. Q\_conditions는 round\_player\_score 과 동일한 방식으로 초기화한다.

initRound : 각 라운드를 초기화하는 코드이다. 각 플레이어의 순서, 카드 등을 초기화한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

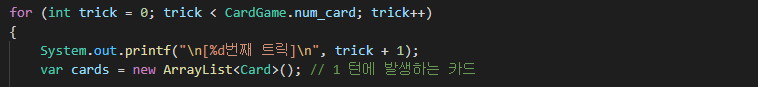
자동 생성된 설명

각 플레이어는 SetCardDeck 메서드를 이용하여 일정한 카드 수만큼 카드를 배정받는다. 이때 플레이어의 수가 5 또는 6 인 경우 필연적으로 버려지는 카드가 발생한다. 해당 카드의 수는 52 – num\_player \* num\_card 이다. 따라서, 해당 수의 카드는 버리는 덱(remain\_card\_deck) 에 보관한다. 이때 유저가 뽑는 카드 안에는 반드시 클로버 2가 포함되어야 한다. 따라서, 클로버 2인 경우, 해당 카드를 카드를 뽑는 덱인 card\_deck에 넣고, 다시 카드를 뽑는 과정을 반복한다. 이 과정을 통해 필요 없는 카드를 미리 덱으로부터 제거한다.

이후 각 플레이어는 userGetCard를 통해 주어진 카드의 수만큼 카드를 가져간다.

즉, 이 과정은 덱의 정리 및 조정 + 카드 분배 과정과 관련되어 있다.

한 라운드와 관련된 변수 설정이 끝나면, 이제 한 트릭에 해당하는 설정이 필요하다.



각 트릭에는 해당 트릭에서 각 플레이어가 선택한 카드를 저장하기 위한 공간이 필요하다. 해당 공간을 ArrayList인 cards로 지정했다.

텍스트, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

이후 한 트릭에서는 여러가지 일을 진행한다.

1. 해당 트릭의 첫번째 사람인지 여부

조건에 따르면, 이전 트릭에서 하트 카드가 나오지 않는다면, 다음 트릭의 첫번째 플레이어는 하트 카드를 낼 수 없다. 해당 조건을 구분하기 위해 (1) 첫번째 사람인지 (index == 0) , (2) 하트를 제출할 수 없는지 (!can\_submit\_\_heart) , (3) 현재 플레이어가 가진 카드가 하트밖에 없는지 (cardAll) 를 검사한다. 이때 세가지 조건을 모두 만족하는 경우는, “이전 트릭에서 하트가 등장하지 않았는데, 현재 첫번째 플레이어가 하트 카드만 가진 경우” 뿐이므로, 우리가 생각하는 조건과 부합한다. 이 상황에 대해서 나는 위에서 언급했듯이 개인에게 책임이 있다고 생각했고, 따라서 첫번째 카드를 강제로 제출하도록 만들었다( p.submitCard(0) ).

위 조건이 아닐 때는, playerChoose을 통해 평범하게 제출할 카드를 입력 받고 input 값에 대해 검사한 후, 조건에 부합하는 경우 원하는 카드를 제출하도록 구현했다.

이때, 한 라운드에 Q 4개를 모으는 상황에 대해 검사하기 위해 추가적인 코드를 생성했다.

card.getNumber() 메서드는 카드의 번호를 알려주는데, Q는 12번에 해당한다. 따라서 현재 뽑은 card의 번호가 12번이라면, 해당 카드는 Q 인 것이다. 이때 52장의 카드 중 중복은 없으므로, 단순히 숫자만 검사하더라도 괜찮다.

이후 Q\_condition.get(p) == 4 조건을 통해 한 사람이 4개의 Q를 모은 경우, 해당 플레이어의 인덱스를 return 하게 하여 승리 조건을 구성했다.

이후 단순히 카드를 cards에 넣고, can\_submit\_heart를 true로 하여 하트를 낼 수 있게 했다.

텍스트, 실내, 은색이(가) 표시된 사진

자동 생성된 설명

한 트릭 내에서 모든 플레이어가 카드를 한 장씩 제출하게 되면 필연적으로 해당 카드를 모두 가져가게 되는 사람이 설정된다. 여기서는 checkLoser 메서드가 이 역할을 수행한다.

텍스트, 화면이(가) 표시된 사진

자동 생성된 설명

checkLoser은 카드를 입력 받고, 초기 패배자를 맨 처음 카드를 제출한 사람으로 설정한다. 이후 for문을 통해 플레이어들이 제출한 카드를 비교하면서, 모양이 동일 + 점수가 더 큰 경우, 해당 플레이어를 현재 트릭의 패배자로 설정한다. 위와 같은 방식으로 패배자의 인덱스를 얻는다.

이후 해당 플레이어가 얻을 점수를 평가한다. 이때 calculateScore 메서드가 사용된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이 메서드는 규칙에 의거하여 단순히 수를 더하는 코드로 구성된다. 플레이어들이 제출했던 카드들에 대해 하트 모양이면 1점, 스페이드 퀸이면 3점, 스페이드 A 이면 5점을 더해, 해당 스코어를 반환한다.

이렇게 얻은 스코어는 패배자의 player\_score에 setPlayerScore을 통해 추가된다.

텍스트, 실내, 은색이(가) 표시된 사진

자동 생성된 설명

이후 과정에서는 다음 트릭에 하트를 낼 수 있는지를 canSubmitHeart를 통해 검사한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

해당 메서드는 단순히 모든 카드를 검사하고, 모두 하트인지 여부를 반환한다.

이후 setOrder을 통해 현재 패배자에 대한 순서를 재설정하고, remain\_card\_deck에 사용된 카드를 넣는다.

텍스트이(가) 표시된 사진

자동 생성된 설명

한 라운드의 모든 트릭이 종료되면, 해당 게임이 종료될지를 검사하고, 점수를 계산한다.

for문을 통해 각 플레이어를 순회하며, 한 라운드에서 얻게 된 점수를 반영한다.

이 과정에서 해당 유저의 점수가 30점을 넘는지를 조사하고, 게임을 종료하는 상황을 대비하여 승자의 인덱스를 계산한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

현재까지의 점수를 출력한 이후, 게임이 끝난 경우 승자의 인덱스를 반환한다. 아닌 경우, remain\_card\_deck에 담긴 카드를 원래 카드 덱인 card\_deck로 복귀, 다음 라운드를 준비한다.

[#3] 프로그램 실행 결과

여러가지 실행 결과를 보인다.

(1) 이전 트릭에 하트가 나오지 않아 하트를 낼 수 없는 모습.

텍스트이(가) 표시된 사진

자동 생성된 설명

(2) 라운드가 넘어갈 때 점수를 계산하는 모습.

텍스트이(가) 표시된 사진

자동 생성된 설명

(3) 코드 조작을 통해 생성한 Q를 4개 뽑았을 때 승리

해당 경우는 의도적으로 만들기 상당히 어렵다. 따라서, 이 경우는 코드를 임시로 조작하여 실제로 승리로 처리되는지에 대해서 검사했다.

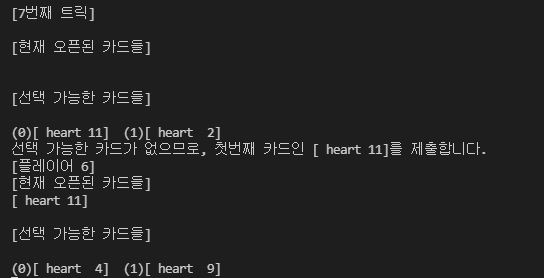
텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

(4) 하트밖에 없는 첫번째 플레이어



강제로 첫 카드가 제출된다.

(5) 일반적인 게임 승리.

텍스트이(가) 표시된 사진

자동 생성된 설명

(6) 값의 유효성 검증

모든 int형에 대한 유효성 검사는 이 프로젝트 내에서 InputHelper의 getInt 메서드에 의해 수행되므로, 하나만 검사해도 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

플레이어 설정의 경우, 잘 수행된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

나이의 설정은 숫자만 허용된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

카드를 뽑는 경우에도 문자 입력 및 잘못된 인덱스 입력을 잘 잡아내고 있다.