

블랙잭 게임 규칙을 설명해줄테니 자바 언어로 DP방식으로 짜서 만들어줘
규칙

1. 2~10은 그대로 값을 가지고, A는 10, J는 11, Q는 12, K는 13
2. 덱에서 두장씩 각각 갖고 시작, 한장씩 hit 가능하며 덱 카드 모두 소모할 때 까지 게임을 반복할거야
3. 21이상이 되면 bust가 되고 진 것으로 판정됨
4. 딜러와 플레이어는 각각의 전략을 가지는데 딜러는 총합이 17 이상이 되면 hit를 멈출거야 너는 너만의 승리 전략을 짜서 알려줘(플레이어는 덱의 순서를 다 알고 있다는 특징이 있어)



자바로 DP(동적 프로그래밍) 방식으로 블랙잭 게임을 구현해보겠습니다. 이 예제에서는 딜러와 플레이어가 게임을 하는 시나리오를 다루고, 덱의 순서를 알고 있는 플레이어가 승리할 수 있는 전략을 사용합니다.

블랙잭 게임 규칙을 반영한 자바 코드를 작성하겠습니다. 이 코드는 다음의 구조를 가집니다:

1. 카드의 값 설정
2. 덱 생성 및 섞기
3. 게임 진행 로직
4. 동적 프로그래밍을 이용한 플레이어의 최적 전략 결정

Step 1: 카드의 값 설정

```
java 코드 복사

import java.util.ArrayList;
import java.util.Collections;

public class BlackjackDP {
    static int[] cardValues = new int[14];

    static {
        for (int i = 2; i <= 10; i++) {
            cardValues[i] = i;
        }
        cardValues[11] = 10; // A
        cardValues[12] = 11; // J
        cardValues[13] = 12; // Q
        cardValues[14] = 13; // K
    }

    public static void main(String[] args) {
        ArrayList<Integer> deck = createDeck();
        Collections.shuffle(deck);
        playGame(deck);
    }

    private static ArrayList<Integer> createDeck() {
        ArrayList<Integer> deck = new ArrayList<>();
        for (int i = 1; i <= 13; i++) {
            for (int j = 0; j < 4; j++) {
                deck.add(cardValues[i]);
            }
        }
        return deck;
    }

    private static void playGame(ArrayList<Integer> deck) {
        int dealerScore = 0;
        int playerScore = 0;

        // 딜러와 플레이어가 각각 두 장의 카드를 받음
        dealerScore += deck.remove(0);
        dealerScore += deck.remove(0);
        playerScore += deck.remove(0);
        playerScore += deck.remove(0);

        // 플레이어가 덱의 순서를 알고 있는 상태에서 최적 전략을 수행함
        int optimalPlayerScore = calculateOptimalScore(deck, playerScore);

        // 딜러는 총합이 17 이상이 될 때까지 히트함
        while (dealerScore < 17 && !deck.isEmpty()) {
            dealerScore += deck.remove(0);
        }

        // 결과 출력
        System.out.println("Dealer Score: " + dealerScore);
        System.out.println("Player Optimal Score: " + optimalPlayerScore);
    }
}
```

```

        if (optimalPlayerScore > 21) {
            System.out.println("Player Busted! Dealer Wins.");
        } else if (dealerScore > 21 || optimalPlayerScore > dealerScore) {
            System.out.println("Player Wins!");
        } else {
            System.out.println("Dealer Wins!");
        }
    }

    private static int calculateOptimalScore(ArrayList<Integer> deck, int currentScore) {
        // 동적 프로그래밍 배열 초기화
        int n = deck.size();
        int[][] dp = new int[n + 1][22];

        for (int i = 0; i <= n; i++) {
            for (int j = 0; j <= 21; j++) {
                dp[i][j] = -1;
            }
        }

        return optimalScoreHelper(deck, 0, currentScore, dp);
    }

    private static int optimalScoreHelper(ArrayList<Integer> deck, int index, int currentScore) {
        if (currentScore > 21) {
            return currentScore; // 버스트
        }
        if (index == deck.size()) {
            return currentScore; // 더 이상의 카드 없음
        }
        if (dp[index][currentScore] != -1) {
            return dp[index][currentScore]; // 이미 계산된 값
        }

        // 히트하지 않음
        int scoreWithoutHit = currentScore;

        // 히트함
        int scoreWithHit = optimalScoreHelper(deck, index + 1, currentScore + deck.get(index));

        // 두 가지 경우 중 더 좋은 점수를 선택
        int optimalScore = Math.min(scoreWithHit, 21); // 버스트되지 않은 최적의 점수 선택

        dp[index][currentScore] = optimalScore;
        return optimalScore;
    }
}

```

설명

- 카드 값 설정: 2~10은 그대로, A는 10, J는 11, Q는 12, K는 13으로 설정합니다.
- 덱 생성 및 섞기: 1에서 13까지의 카드 값을 사용하여 덱을 만들고, 각 카드가 4장씩 있습니다. 그 후 덱을 섞습니다.
- 게임 진행 로직: 딜러와 플레이어가 각각 두 장의 카드를 받고, 딜러는 17 이상이 될 때까지 히트합니다. 플레이어는 덱의 순서를 알고 있는 상태에서 최적의 전략을 사용합니다.
- 동적 프로그래밍을 이용한 최적 전략 결정: `calculateOptimalScore` 메서드는 동적 프로그래밍을 사용하여 현재 점수와 남은 카드에 따라 플레이어가 최적의 점수를 계산합니다.

이 코드는 블랙잭 게임을 단순화한 버전으로, 복잡한 게임 규칙과 전략을 모두 포함하지는 않습니다. 실제 블랙잭 게임은 훨씬 더 복잡한 규칙과 전략을 가지고 있습니다.