Java Programming

assignment #: Java App developement

모바일시스템공학과 김태경 (32211203)

목차

2024.06.06.

1. 프로젝트 소개

본 프로젝트는 자바 GUI를 사용하여 블랙잭 게임을 구현한 것이다. 블랙잭은 전 세계적으로 인기 있는 카드 게임으로, 플레이어가 딜러를 상대로 카드를 사용하여 21에 가까운 숫자를 만들기 위해 경쟁한다. 이 보고서는 프로젝트의 목적, 설계, 구현 과정 및 결과를 다룬다.

2. 프로젝트 목표

본 프로젝트의 주요 목적은 자바 GUI 프로그래밍에 대한 이해를 높이고, 객체 지향 프로그래밍(OOP) 원칙을 적용하여 복잡한 게임 로직을 구현하는 것이다. 이를 통해 블랙잭 게임의 전반적인 작동 원리와 프로그래밍 기술을 심화하고자 한다.

최소 목표

블랙잭 게임 한 판 구현: 블랙잭의 기본 규칙을 바탕으로 한 게임 한 판을 구현하여, 플레이어가 딜러와 대결할 수 있는 환경을 구축한다. 이를 통해 기본적인 게임 로직과 사용자 인터페이스를 완성한다.

추가 목표

추가 목표는 기본 게임 구현을 넘어서 게임의 완성도를 높이고 사용자 경험을 향상시키는 데 중점을 둔다. 가능한 한 많은 목표를 구현하는 것을 목표로 한다.

1. 지속적인 게임 플레이와 승패 기록: 게임을 단 한 판만 실행할 수 있는 것이 아니라, 여러 판을 연속적으로 플레이할 수 있는 기능을 추가하고, 각 판의 승패를 기록하고, 플레이어의 승률을 시각적으로 확인할 수 있도록 한다.
2, 베팅 시스템 구현: 게임 내에서 플레이어가 칩을 사용하여 베팅할 수 있는 시스템을 구현한다.
3. 멀티플레이어 모드: 한 명의 플레이어가 아닌 2~3명의 플레이어가 동시에 참여할 수 있는 멀티플레이어 모드를 구현한다.

3. 프로그램 빌드 환경

컴파일 환경: Vs code
프로젝트 언어: Java
파일: App.java / BlackJack/java / file: cards(카드 .png)

4. 프로젝트에 사용된 개념

Black jack에 대해

블랙잭(Blackjack)은 전 세계적으로 인기 있는 카드 게임 중 하나로, 주로 카지노에서 즐겨집니다. 이 게임의 목표는 딜러를 이기기 위해 카드의 총합을 21에 가깝게 만드는 것입니다.

기본 룰 (원본 규칙에서 일부 수정함)

카드 값:
- 숫자 카드(2~10): 해당 숫자만큼의 값.
- 그림 카드(잭, 퀸, 킹): 각 카드의 값은 10.
- 에이스(A): 1 또는 11로 계산할 수 있으며, 플레이어에게 유리하게 계산됩니다.

게임 시작:
- 각 플레이어와 딜러는 두 장의 카드를 받습니다.
- 플레이어의 카드는 모두 공개되지만, 딜러의 카드는 한 장만 공개되고 나머지 한 장은 비공개입니다(홀 카드).

플레이어의 선택:
- 히트(Hit): 추가로 카드를 한 장 더 받음.
- 스테이(Stay): 현재 카드로 턴을 종료

딜러의 플레이:
- 딜러는 총합이 17 이상이 될 때까지 카드를 계속 받습니다.
- 17 이상일 경우 딜러는 더 이상 카드를 받지 않습니다.

승패 결정:
- 플레이어의 카드 총합이 21을 초과하면 버스트(Bust)로 패배.
- 플레이어와 딜러 모두 21을 넘지 않으면, 더 높은 총합을 가진 쪽이 승리.
- 카드 총합이 같으면 무승부(Tie)로 베팅 금액을 다음 판으로 이월시킴.

추가 규칙
- 베팅규칙: 카드를 받기 전 선납을 원칙으로 한다.
- 배당률: 블랙잭 승리 시 1:1 배당률

프로그램에서 사용되는 주요 개념

클래스와 객체 지향 프로그래밍(OOP)
- Card 클래스: 카드의 값을 나타내며, 각 카드의 값과 타입을 저장
- 클래스: 게임의 전반적인 로직과 GUI를 관리

GUI 프로그래밍
- JFrame: 애플리케이션의 메인 윈도우를 생성
- JPanel: 게임 판과 버튼들을 배치하기 위해 사용
- JButton: 사용자 인터페이스에서 사용자가 클릭할 수 있는 버튼을 만듬
- CardLayout: 여러 패널을 카드처럼 배치하고, 필요에 따라 패널을 전환가능
- mageIcon과 Image: 카드 이미지를 불러와서 화면에 표시

이벤트 처리
- ActionListener 인터페이스: 버튼 클릭 등의 사용자 입력을 처리
- 익명 내부 클래스를 사용하여 이벤트 핸들러를 정의

자료구조
- ArrayList: 카드 덱과 플레이어/딜러의 손패를 저장
- String: 카드의 값과 타입을 나타냄

난수 생성
- Random 클래스: 덱 셔플링 시 카드의 순서를 무작위로 섞기 위해 사용

게임 로직
- 카드 덱 생성 및 셔플: 52장의 카드를 생성하고, 이를 무작위로 섞음
플레이어와 딜러의 손패 관리: 각자 2장의 카드를 받고, 플레이어는 'Hit' 버튼을 눌러 추가 카드를 받음
- 계를 비교하여 승패를 결정
- 베팅 시스템: 플레이어가 베팅 금액을 설정하고, 게임 결과에 따라 베팅 금액을 잃거나 얻음

그래픽스
- paintComponent 메서드: 게임 판에 카드와 텍스트를 그림
- 이미지 로드와 그리기: 카드 이미지를 로드하고, 화면에 표시

5. 프로그램 구성

Card 클래스

: 외부로부터 카드를 경로를 호출하고 카드별 값을 반환

- Card(String value, String type): 카드의 값을 초기화하는 생성자
- toString(): 카드의 값을 문자열로 반환
- getValue(): 카드의 점수를 반환합니다. A는 11점, J/Q/K는 10점으로 처리
- isAce(): 카드가 Ace인지 확인
- getImagePath(): 카드 이미지의 경로를 반환

BlackJack 클래스

: 게임의 진행 중 승패의 기록 등을 위한 전역변수와 GUI기본 프레임, 버튼 호출

- 인스턴스 변수: 게임에 필요한 상태를 저장
- startGame(): 게임을 초기화하고 시작
- buildDeck(): 덱을 생성
- shuffleDeck(): 덱을 셔플
- reducePlayerAce(): 플레이어의 Ace 값을 조정
- reduceDealerAce(): 딜러의 Ace 값을 조정

BlackJack 생성자

: 게임 상태별 버튼의 활성화와 패널의 구성 등을 조정

- GUI 초기화: 프레임 설정, 패널 설정(시작, 카드, 게임 패널 설정)
- 버튼 리스너 설정: ‘star’, ‘exit’, ‘Hit’, ‘Stay’, ‘Retry’, ‘Quit’별 작동 정의

Game 시스템 함수

: 게임 룰을 기반으로 하여 게임 실행하는 함수

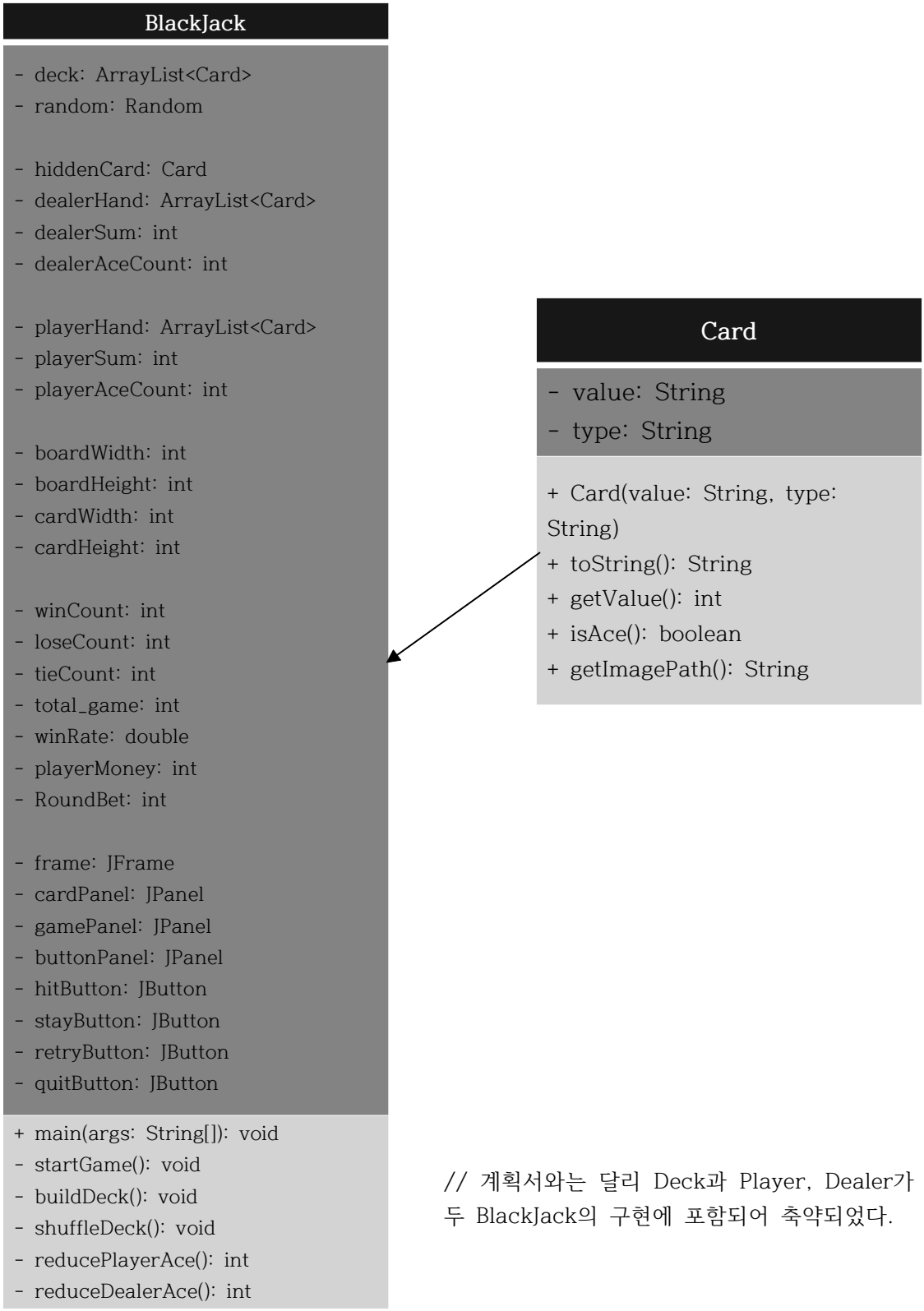
- startGame(): 게임시작, 덱을 빌드&셔플, 딜러와 플레이어의 초기 손패를 설정
- buildDeck(): 카드 덱을 생성
- shuffleDeck(): 덱을 셔플
- reducePlayerAce(): 플레이어의 Ace 값을 조정
- reduceDealerAce(): 딜러의 Ace 값을 조정

Main 메서드

: 프로그램 실행

// 자세한 코드는 실행코드 참고

클래스 다이어그램 구조



// 계획서와는 달리 Deck과 Player, Dealer가 모두 BlackJack의 구현에 포함되어 축약되었다.

6. 문제점 및 해결방법

구현 중 발생한 문제

1. Nan 예외처리, 승률 조정

블랙잭 게임 구현 중 승률 계산에서 0%일 때 NaN이 출력되는 문제는 실수를 정수로 출력하려 시도하여 발생한 것으로, 0일 때의 예외 처리를 통해 수정하였다.

또한, 승률이 평균 20%대로 낮았던 문제는 초기 프로그램이 WIN과 LOSE만 처리하여 BUST와 TIE 상황을 고려하지 않은 것에서 비롯되었고, 블랙잭의 공식 룰을 다시 숙지하고, BUST와 TIE 등을 추가로 구현하여 게임 규칙에 맞게 프로그램을 재작성했다. 이를 통해 승률 계산이 정확해지고, 승률이 지나치게 낮은 문제도 해결되었다.

재작성한 코드(이전버전은 없음)

```
if (playerSum > 21)
{
    message = "Bust You Lose";
    loseCount++;
    RoundBet = 0;
}
else if ((dealerSum < 21) && (playerSum < dealerSum))
{
    message = "You Lose!";
    loseCount++;
    RoundBet = 0;
}
else if ((dealerSum < 21) && (playerSum > dealerSum))
{
    message = "You Win!";
    winCount++;
    playerMoney += RoundBet*2;
    RoundBet = 0;
}
else if ((dealerSum > 21) && (playerSum < 21))
{
    message = "You Win!";
    winCount++;
    playerMoney += RoundBet*2;
    RoundBet = 0;
}
else if (playerSum == dealerSum)
{
    message = "Tie!";
    tieCount++;
}
```

2. 처음에는 베팅 시스템을 구현하면서 플레이어가 마지막에 Hit를 누를 때 베팅 금액을 정하도록 했습니다. 하지만 추가 카드를 받을 때마다 계속 베팅하는 것이 이상하다고 생각하여 베팅 룰을 다시 확인했습니다. 보통 블랙잭에서는 처음에 카드를 받기 전에 베팅하는 것이 원칙이므로, 이를 반영하여 시스템을 수정했습니다. 이는 블랙잭 룰에 대한 이해 없이 구현하여 생긴 문제였다.

3. Quit할 경우에 win, lose, tie 및 승률과 플레이어 머니 등이 초기화 되지 않았기에, QuitButton이 실행될 경우 다음과 같이 각각의 값을 초기화 해주어 문제를 해결하였다.

// 이외에 별도의 프로그래밍 중 문제가 되는 부분은 없었습니다.

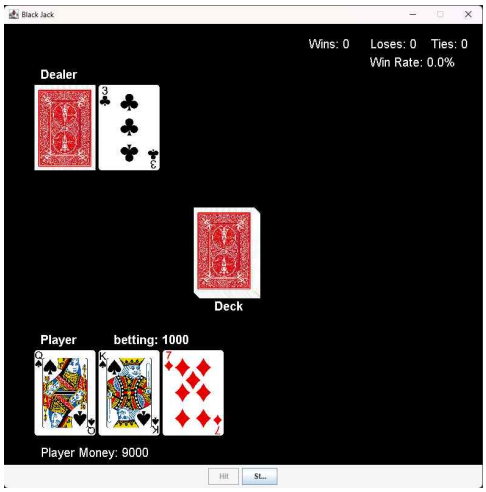
```
// Quit button action listener
quitButton.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        CardLayout cl = (CardLayout) (cardPanel.getLayout());
        cl.show(cardPanel, name:"start");
        retryButton.setVisible(aFlag:false);
        quitButton.setVisible(aFlag:false);
        winCount = 0;
        loseCount = 0;
        tieCount = 0;
        winRate = 0;
        total_game = 0;
        playerMoney = 10000;
    }
});
```

7. 실행결과

시작화면

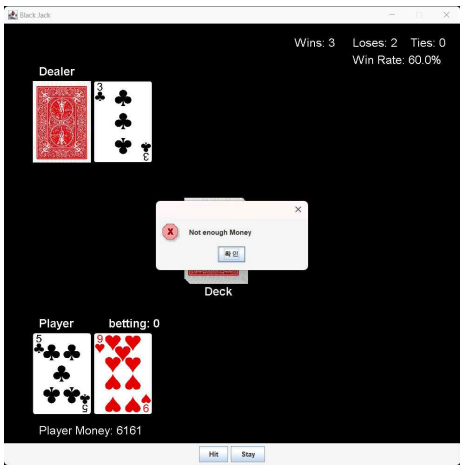
Start & Betting

Hit

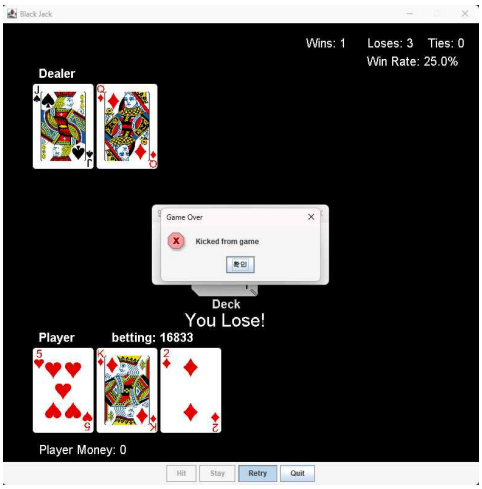


Stay & Win

Not enough money



Game over



8. 느낀점

프로젝트를 진행하면서 꾸준히 하지 못하고 띄엄띄엄 하다보니 프로그램에 대한 이해를 위해 다시 파악하느라 시간을 소비하는 경우가 많았는데, 카드와 덱, 플레이어와 딜러 등 게임의 주요 요소들을 객체로 정의하고, 각 객체의 역할과 책임을 명확히 구분하여 주석으로 설명을 자세히 달아 두었더니, 코드의 가독성과 유지보수성을 높일 수 있었다.

Java의 Swing 라이브러리를 사용하여 사용자 인터페이스를 설계하고, 다양한 이벤트를 처리하는 방법을 학습했습니다. 게임 시작, 카드 히트, 스테이, 다시 시작 등의 기능을 버튼과 이벤트 리스너를 통해 구현하며 사용자와의 상호작용을 효과적으로 처리할 수 있었습니다.

블랙잭 게임의 로직을 구현하던 중, 플레이어가 너무 자주 패배하는 문제가 발생하여 원활한 게임 플레이가 어려운 상황이 자주 발생하였다. 초기에는 실행 중 오류로 인해 이러한 문제가 발생한다고 생각했는데, 문제의 원인은 블랙잭 게임 룰을 잘못 이해하여 생기는 문제였다. 이 경험을 통해 프로그램의 꼼꼼한 사전 설계의 중요성을 다시 한 번 깨달았다.

추가구현으로 선택한 배팅 시스템은 플레이어의 돈을 관리하고, 배팅 금액에 따라 승패에 따른 돈의 변동을 처리하는 기능을 추가하여 게임의 현실감을 더했다. 추가 목표에서의 멀티플레이어 모드는 구현하지 않았는데, 한 화면에서 다수가 플레이하기에는 적절하지 못한 환경이고, 네트워크를 연결해서 구현하더라도 딜러와 개개인이 일대일로 대결하는 게임이라 큰 의미가 없는 목표라고 생각하여 구현하지 않았다.

GUI를 통한 사용자와의 상호작용 설계능력을 향상시킬 수 있었고, 복잡한 게임 로직을 단계별로 구현하고 디버깅하는 과정에서 문제 해결 능력과 논리적 사고력을 키울 수 있었다. 앞으로도 이러한 경험을 바탕으로 더 복잡하고 흥미로운 프로젝트에 도전해보고 싶다.

9. 참고자료

- 트럼프 카드 png 출처 : https://www.pngwing.com/ko/search?q=%ED%8A%B8%EB%9F%BC%ED%94%84+%EC%B9%B4%EB%93%9C
- 자바 GUI 기초 : https://yooniron.tistory.com/12
- 블랙잭 룰 및 구현 : https://velog.io/@mandarine_punch/My-Toy-%EB%B8%94%EB%9E%99-%EC%9E%AD-Java