# Blackjack Game

## Overview

This project is a console-based Blackjack game implemented in Java. It includes various classes and packages to handle the game logic, player actions, card management, and data handling.

## Features

- Console-based gameplay
- Multiple players support
- Dealer functionality
- Card shuffling and drawing
- Data persistence using YAML

## Requirements

- Java 8 or higher (I used Java 23)
- SnakeYAML library for YAML data handling (I used version 2.3)

### Installation

1. Download and Install Java from the official Java website.
2. Download the `SnakeYAML.jar` library from the Maven repository: SnakeYAML 2.3

## Usage

### Clone the Repository

#### Using Git

1. Clone the repository:

```
git clone https://github.com/DanyilT/Java.git
```

2. Navigate to the project folder:
    - For Windows:

```
cd Java\BlackJack\src
```

    - For macOS and Linux:

```
cd Java/BlackJack/src
```

#### Downloading the ZIP File

1. Download the ZIP file from the GitHub repository and extract it.
2. Navigate to the project folder in the extracted directory (`Java/BlackJack/src`).

### Run the Program

#### Update the `game_data.yaml` file

If you want to play the game with multiple players, you can update the `game_data.yaml` file in the `data` directory to include the player names.

- The file should look like this:

```yaml
players:
  0:
    name: "Dealer"
  1:
    name: "Player 1"
  2:
    name: "Player 2"
```

- You can add more players by incrementing the player ID and adding their names (Dealer has ID 0).
- Or you can leave it empty, and the game will create a one new player (for single-player).

#### Compile and Run the Program

1. Compile the program:

```
javac Main.java
```

2. Run the program:

```
java Main
```

## Project Structure

### File Structure

- `data/`: Contains the data files for the program.
  - `deck.yaml`: Contains the deck data for the game.
  - `game_data.yaml`: Contains the player data for the game.
- `src/..`: Contains packages and the source code files.

- `UML Diagram/`: Contains the UML diagram for the project.
  - `blackjack_uml.png`: Photo of the UML diagram.
  - `blackjack_uml.puml`: PlantUML file for the UML diagram.
- `README.md`: You are reading it right now.

### Packages

- `cards`: Contains classes related to card management.
- `data`: Handles reading and writing game data.
- `exceptions`: Custom exceptions used in the game.
- `game`: Contains the main game logic and engine.
- `players`: Manages player actions and data.

### Packages.Classes

#### `Main`

- **main()**: Entry point of the application, initializes the `BlackjackGame`.

#### `cards`

**Card Management**: Implemented in the `cards` package.

- **Card**: Represents a single card with a rank and suit.
- **CardRank**: Enum for card ranks.
- **CardSuit**: Enum for card suits.
- **Deck**: Manages a deck of cards, including shuffling and drawing cards.

#### `data`

**Data Handling**: Implemented in the `data` package.

- **DataHandler**: Abstract class for reading and writing data.
- **DeckDataHandler**: Handles deck-specific data operations.
- **GameDataHandler**: Manages player game data.

#### `exceptions`

**Custom Exceptions**: Implemented in the `exceptions` package.

- **InsufficientChipsException**: Thrown when a player tries to bet more chips than they have.
- **InvalidPlayerDataException**: Thrown when player data is invalid.

#### `game`

**Game Logic**: Implemented in the `game` package.

- **BlackjackGame**: Starts the game and initializes the game engine.
- **GameEngine**: Contains the core game loop and logic for determining winners.
- **TableDrawer**: Draws the game table in the console.

### `players`

**Player Actions**: Implemented in the `players` package.

- **Player**: Abstract class representing a player.
- **Dealer**: Extends `Player` and implements dealer-specific actions.
- **UserPlayer**: Extends `Player` and implements user-specific actions.
- **PlayerActions**: Interface for player actions like hit, stand, double down, and split.