**I have attached below how to create a complete Java Application of its deliverance and its associate point values. I even mentioned my plans and strategies from top bottom and implementing planning plans which is constructed to the best of my knowledge down below. Also link to My Draw.i.o- UML diagram, and Replit.**

[**https://replit.com/@NicholeBerry1/M03-Final-Project-Update-1-Proposal#script.js**](https://replit.com/@NicholeBerry1/M03-Final-Project-Update-1-Proposal#script.js)

[**https://app.diagrams.net/?src=about#G1F8Fl\_lJPumf8ml4UzzD-ZI\_dG174ARWM**](https://app.diagrams.net/?src=about#G1F8Fl_lJPumf8ml4UzzD-ZI_dG174ARWM)

**Blackjack-FinalProject Plan:**

**My Project Proposal for Java Application**

**My Purpose of the App**

**\*The purpose of our Java application is to create a user-friendly and interactive blackjack game for players. The game will implement the standard rules of blackjack, allowing players to place bets, receive cards, make decisions (hit or stand), and compete against a computer-controlled dealer. This application aims to provide an engaging and enjoyable gaming experience for users while incorporating various programming concepts. \***

**Classes Required**

To achieve my project's objectives, I plan to create the following classes:

**Card** Class:

This class represents a playing card with attributes like rank, suit, and value. It will also include methods for retrieving card information and converting it to a string representation.

**Deck** Class:

The **Deck** class will manage the deck of cards used in the game. It will be responsible for creating, shuffling, and dealing cards to players and the dealer.

**Hand** Class:

The **Hand** class will represent a player's or dealer's hand. It will store a list of cards and provide methods to calculate the hand's value, check for blackjack, and determine if the hand is busted.

**Player** Class:

This class will represent the player and include attributes like name, current bankroll, and methods to place bets and interact with the game.

**Dealer** Class:

The **Dealer** class will represent the computer-controlled dealer. It will manage the dealer's actions, such as hitting or standing, and calculate the dealer's hand value.

**BlackjackGame** Class:

This class will serve as the main game engine, coordinating the interactions between the player, dealer, and deck. It will handle the game's flow and outcomes.

**Class Relationships**

**Card** is a parent class to **Deck**. A **Deck** contains multiple **Card** objects.

**Player** and **Dealer** are child classes of the **Hand** class, inheriting hand management features.

**BlackjackGame** will use instances of **Deck**, **Player**, and **Dealer** to control the game flow.

**Fields in Classes**

To meet the minimum requirement of at least 10 fields, the classes will contain the following fields:

**Card** Class:

rank, suit, value

**Deck** Class:

ArrayList of cards

**Hand** Class (both **Player** and **Dealer**):

ArrayList of cards

**Player** Class:

Name, bankroll, bet amount, etc.

**Dealer** Class:

Dealer's name (optional)

**BlackjackGame** Class:

Reference to **Deck**, **Player**, and **Dealer** instances

Game state variables, such as the player's turn and game outcome

**Loop Structures**

To meet the loop requirements, we will use both a **for** loop and a **while** loop in the game logic. The **for** loop may be used for iterating through cards in a hand or deck, and the **while** loop will handle player decision-making and continuous gameplay.

**File Management**

The project will perform file management by saving and loading game states to/from a text-based file. This allows players to continue their game sessions across multiple sessions. I will implement methods for reading and writing game progress to a file.

**Methods**

The project will include at least 10 methods distributed across the classes, handling various aspects of the game. These methods will include functions for shuffling the deck, dealing cards, determining winners, and saving/loading game states. Additionally, I will implement an overloaded method, for example, to handle different player actions based on user input.

**Arrays**

I will use arrays to store and manage collections of cards in the **Deck** and **Hand** classes. ArrayLists may also be employed to allow dynamic sizing of these collections.

**Java Libraries**

Throughout the project, I will import and utilize various Java libraries as needed. Common libraries like **java.util.Scanner** will be used for user input, and others may be incorporated for specific functionality, such as file I/O.

**Exception Handling**

I will implement exception handling to ensure the program can gracefully handle unexpected errors. Common exceptions such as **InputMismatchException** for user input validation and custom exceptions for game-specific issues will be considered.

The Java application will aim to deliver an engaging and interactive blackjack gaming experience while adhering to the specified project requirements. The proposal outlines the key classes, methods, and features to be included in the project. As development progresses, adjustments and enhancements may be made to meet the project's objectives effectively. I look forward to working on this project to the best of my knowledge. Thank you and God Bless. :D

**Key points for implementation:**

**Card Class:**

public class Card {

private String rank;

private String suit;

private int value;

// Constructor

public Card(String rank, String suit, int value) {

this.rank = rank;

this.suit = suit;

this.value = value;

}

// Getter methods

// Other methods for converting to string, etc.

}

Deck Class:

import java.util.ArrayList;

import java.util.Collections;

public class Deck {

private ArrayList<Card> cards;

// Constructor

public Deck() {

this.cards = new ArrayList<>();

initializeDeck();

shuffleDeck();

}

private void initializeDeck() {

// Populate the deck with cards

}

private void shuffleDeck() {

// Shuffle the deck

}

// Other methods for dealing cards, etc.

}

Hand Class:

import java.util.ArrayList;

public class Hand {

protected ArrayList<Card> cards;

// Constructor

public Hand() {

this.cards = new ArrayList<>();

}

// Other methods for calculating hand value, checking for blackjack, etc.

}

**Player Class:**

public class Player extends Hand {

private String name;

private int bankroll;

private int betAmount;

// Constructor

public Player(String name, int initialBankroll) {

this.name = name;

this.bankroll = initialBankroll;

this.betAmount = 0;

}

// Other methods for placing bets, etc.

}

**Dealer Class:**

public class Dealer extends Hand {

// You can add a name for the dealer if needed

// Other methods for dealer's actions, etc.

}

BlackjackGame Class:

public class BlackjackGame {

private Deck deck;

private Player player;

private Dealer dealer;

private boolean playerTurn; // Flag to indicate player's turn

private String gameOutcome; // Variable to store the game outcome

// Constructor

public BlackjackGame() {

this.deck = new Deck();

this.player = new Player("Player", 1000); // You can adjust the initial bankroll

this.dealer = new Dealer();

this.playerTurn = true;

this.gameOutcome = "";

}

// Other methods for game flow, handling player decisions, etc.

}

**\*The actual implementation will involve adding more methods, logic, and details based on project requirements. For running the application, consider creating a Main class with a main method where initialize the BlackjackGame and handle the overall game flow.\***

**Explanation:**

**Card Class:**

• This class represents a playing card and has three private fields: rank, suit, and value.

• The constructor initializes these fields when a Card object is created.

• I would typically have getter methods for accessing these fields and may add other methods as needed, such as converting the card to a string representation.

**Deck Class:**

• The Deck class manages the deck of cards used in the game.

• The constructor initializes an empty ArrayList of cards, then calls initializeDeck to populate the deck and shuffleDeck to shuffle the cards.

• initializeDeck and shuffleDeck are private methods, and you'll need to implement them.

**Hand Class:**

• The Hand class represents a player's or dealer's hand and contains an ArrayList of cards.

• This class doesn't have a specific constructor in this example, but you might add one if needed.

• The cards field is marked as protected to allow access from child classes (Player and Dealer).

**Player Class:**

• The Player class is a child class of Hand and includes additional fields like name, bankroll, and betAmount.

• The constructor initializes these fields when a Player object is created.

• This class inherits the cards field from the Hand class.

**Dealer Class:**

• The Dealer class is also a child class of Hand and can include additional fields or methods specific to the dealer's actions.

BlackjackGame Class:

• The BlackjackGame class is the main game engine and coordinates interactions between the player, dealer, and deck.

• The constructor initializes instances of Deck, Player, and Dealer, and sets initial values for playerTurn and gameOutcome.

• I'll need to add methods to handle the game flow, player decisions, and other aspects of the game.

**\*These classes provide a foundation for Blackjack game, and now start implementing methods and logic for each class. \* Thank you and God Bless! :D**

**My \*UML\* “Blackjack-Final Project Example at the bottom page… Maybe I will make adjustment, if needed, down the road if needs correcting.**

A screenshot of a computer

Description automatically generated

**This UML diagram above,**

* The **BlackjackGame** class is at the top, representing the main game engine. It has associations with the **deck**, **player**, and **dealer** classes.
* The **Deck** class contains a list of c**ard** objects and provides methods for initializing, shuffling, and dealing cards.
* The **Hand** class is a generic class that represents a player's or dealer's hand. It contains methods for calculating hand value, checking for blackjack, and determining whether the hand is busted.
* The **Player** class extends the **Hand** class and has additional fields like **name**, **bankroll**, and **bet** a**mount**. It also includes methods for placing bets and interacting with the game.
* The **Dealer** class can have additional fields and methods specific to the dealer's actions.
* The **Card** class represents a playing card with attributes like **rank**, **suit**, and **value**. It includes getter methods and a **toString** method for converting the card to a string representation.