Project title	**PADDLE GAME**
Author(s)	**Borzan Calina Annemary**
Group	**30424**

1. Task Description

Introduction

Paddle Game is an interactive, engaging, and modern take on the classic Pong game. The core objective of this game is to offer users a nostalgic yet refreshing gaming experience, combining the simplicity of traditional Pong with contemporary enhancements. This game aims to cater to both individual players and groups looking for a competitive and fun pastime. The objective is also to create a user-friendly interface that allows players of all ages to easily navigate and enjoy the game. Emphasis is placed on smooth gameplay, responsiveness, and visually appealing graphics and sounds to ensure a delightful gaming experience.This game was implement using Java programming language and can be opened in various applications: BlueJ, IntelijIDEA.

Features

Single Player Mode

- Gameplay Mechanics: In this mode, the player competes against an AI-controlled paddle. The AI paddle automatically moves along the y-axis to intercept and return the ball. The movement speed and response rate of the AI paddle can be tailored to offer different levels of difficulty.

- Player Interaction: The player controls their paddle using keyboard inputs (such as the 'W' and 'S' keys) to move up and down. This mode is designed to test the player's reflexes and ability to adapt to the AI's playing style.

- Scoring and Win Condition: The game keeps track of the player's score. A point is scored when the player successfully gets the ball past the AI paddle. The game ends when either the player or the AI reaches a preset score, typically 10 points.

- User Experience: The single-player mode is ideal for practicing paddle control and improving gameplay skills. It provides a challenging yet engaging experience, with the AI serving as a competent opponent.

Two Player Mode

- Gameplay Mechanics: This mode allows two players to compete against each other on the same computer. Each player controls their own paddle, with one player using the 'W' and 'S' keys, and the other using the Up and Down arrow keys.

- Player Interaction: The players' objective is to hit the ball past their opponent's paddle. This mode requires quick reflexes and strategic movements. The interaction is more unpredictable and dynamic, as each player brings their own style and strategy to the game.

- Scoring and Win Condition: Similar to the single-player mode, the game tracks each player's score. Points are scored when one player fails to return the ball, and the game concludes when one of the players reaches the preset winning score which means 10 points.

- User Experience: The two-player mode fosters a competitive and social gaming experience. It's an opportunity for players to compete with friends or family members, adding an element of personal interaction and rivalry to the game.

General Features Applicable to Both Modes

- Real-Time Score Display: The game continuously displays the current score, keeping players informed about their progress throughout the match.

- Sound Effects: The game includes sound effects for actions like the ball hitting a paddle or scoring a point, enhancing the immersive experience.

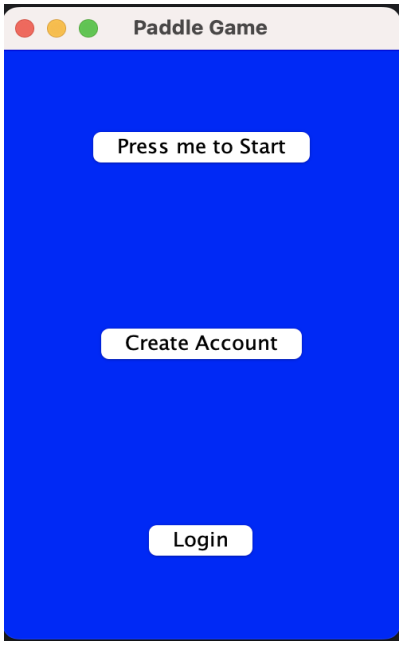
- End-Game Scenario: Once the game reaches its conclusion, a message displays the winner and the final score. Also you have the possibility to restart the game and try again to win.

- History Tab: Once you are logged in you can watch your progress by clicking on history button. There you will be able to see your scores and times.

User Interaction

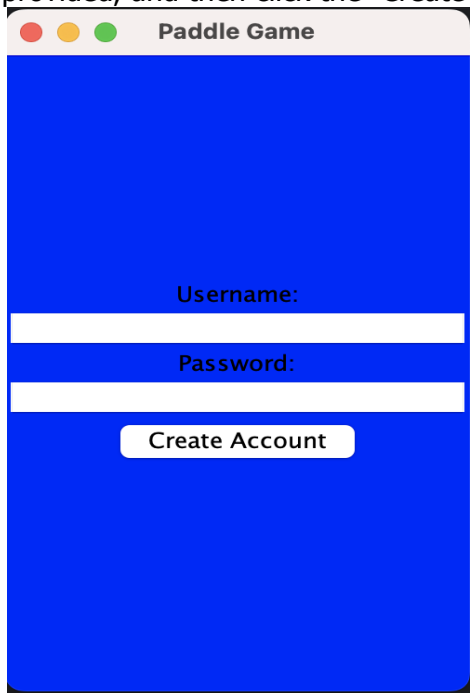
1. Starting the Application

- Description: When you launch the application, you'll be greeted with the Start Screen.
- Actions: As to launch the application you must go in the PongGame Class and compile. On this screen, you have three options: Start the game, create a new account, or log in.



2. Creating an Account

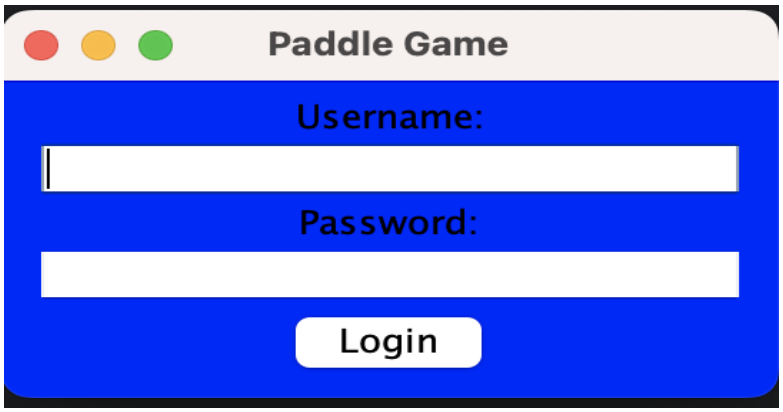
- Description: If you don't have an account, you can create one to save your game history.
- Actions: Click on the "Create Account" button. Enter your desired username and password in the fields provided, and then click the "Create Account" button to register.



3. Logging into Your Account

- Description: For making sure that all your games and progress are being stored.

- Actions: Click on the "Login" button. Enter your username and password, and then click the "Login" button to access your account.



4. Selecting the Game Mode

- Description: After logging in, you'll be directed to the Mode Selection screen.
- Actions: Choose between "Single Player Mode", "Two Player Mode", or "View History".

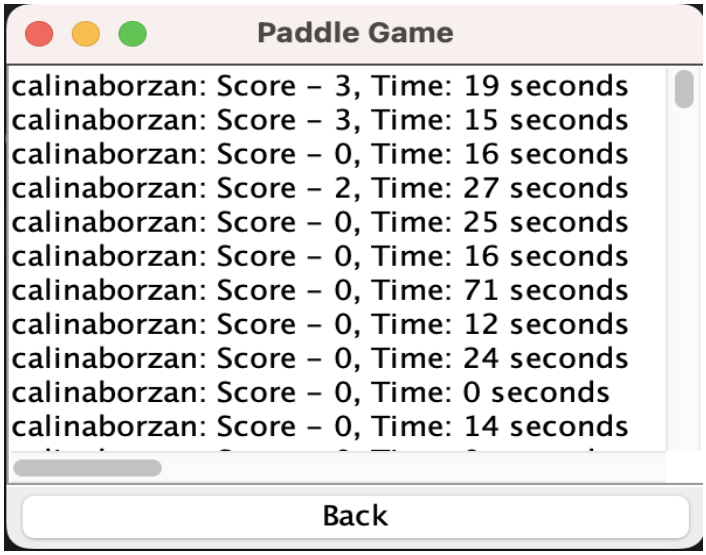
5. Playing the Game

- Description: This is the core part of the application where you play Pong.
  - For Single Player Mode: You will play against an AI-controlled paddle.
  - For Two Player Mode: Two players can play, each controlling a paddle.

- Actions: Use the keyboard controls: player1: W & S, player 2: Up & Down, to move your paddle and hit the ball. The objective is to score 10 points by getting the ball past your opponent's paddle. The time is used as to be able to track how fast you managed to win.

6. Viewing Game History

- Description: You can view your past game performances and track your progress during the time.
- Actions: From the Mode Selection screen, click on "View History". This will display your previous game scores and times. The button back takes you back to the previous menu, to choose your mode play.



7. Restarting the game

- Description: When you want to restart the game you just played.
- Actions: Press the space tab and it will be done.

2. Class Discovery

Class-Responsibility-Collaborator (CRC) cards are a tool used in object-oriented programming and design to help in understanding and organizing the roles and interactions of different classes in a system.

GameFrame	
Main application window, switching between different panels.	StartPanel LoginPanel AccountCreationPanel ModeSelectionPanel GamePanel SinglePlayerGamePanel HistoryPanel
Hosting the current user session.	UserAccount

StartPanel	
Display initial options to the user.	GameFrame
Navigate to Login, Account Creation or Start Game	LoginPanel AccountCreationPanel

	ModeSelectionPanel

AccountCreationPanel	
Manage account creation process.	GameFrame UserAccount AccountManager
Validate and save new user data.	UserAccount

LoginPanel	
Handle user login.	GameFrame UserAccount AccountManager
Authenticate user credentials.	UserAccount

ModeSelectionPanel	
Allow user to select game mode or view history.	GameFrame
Navigate to different game modes or history panel.	GamePanel SinglePlayerGamePanel HistoryPanel

GamePanel/SinglePlayerGamePanel	
Control the game's main logic.	Paddle Ball Score SoundPlayer
Render game views and handle user inputs.	GameFrame

HistoryPanel	
Dispaly user's game history.	GameFrame UserAccount
Retrieve and format past game data.	GameRecord

UserAccount	
Store user data and game history.	AccountCreationPanel LoginPanel HistoryPanel

Paddle	
Represent player's paddle in the game.	GamePanel
Handle movement and collision logic.	Ball

SinglePlayerPaddle	
Controls the AI Paddle in Single Player Mode.	GamePanel Ball
Automate movement to follow the ball and handle collision.	Ball
Adjust movement strategy based on game state.	GamePanel

Ball	
Represents the ball in the game.	GamePanel
Handle movement and collision detection.	Paddle Score

Score	
Track and display game scores.	GamePanel
Update scores based on game events.	Ball Paddle

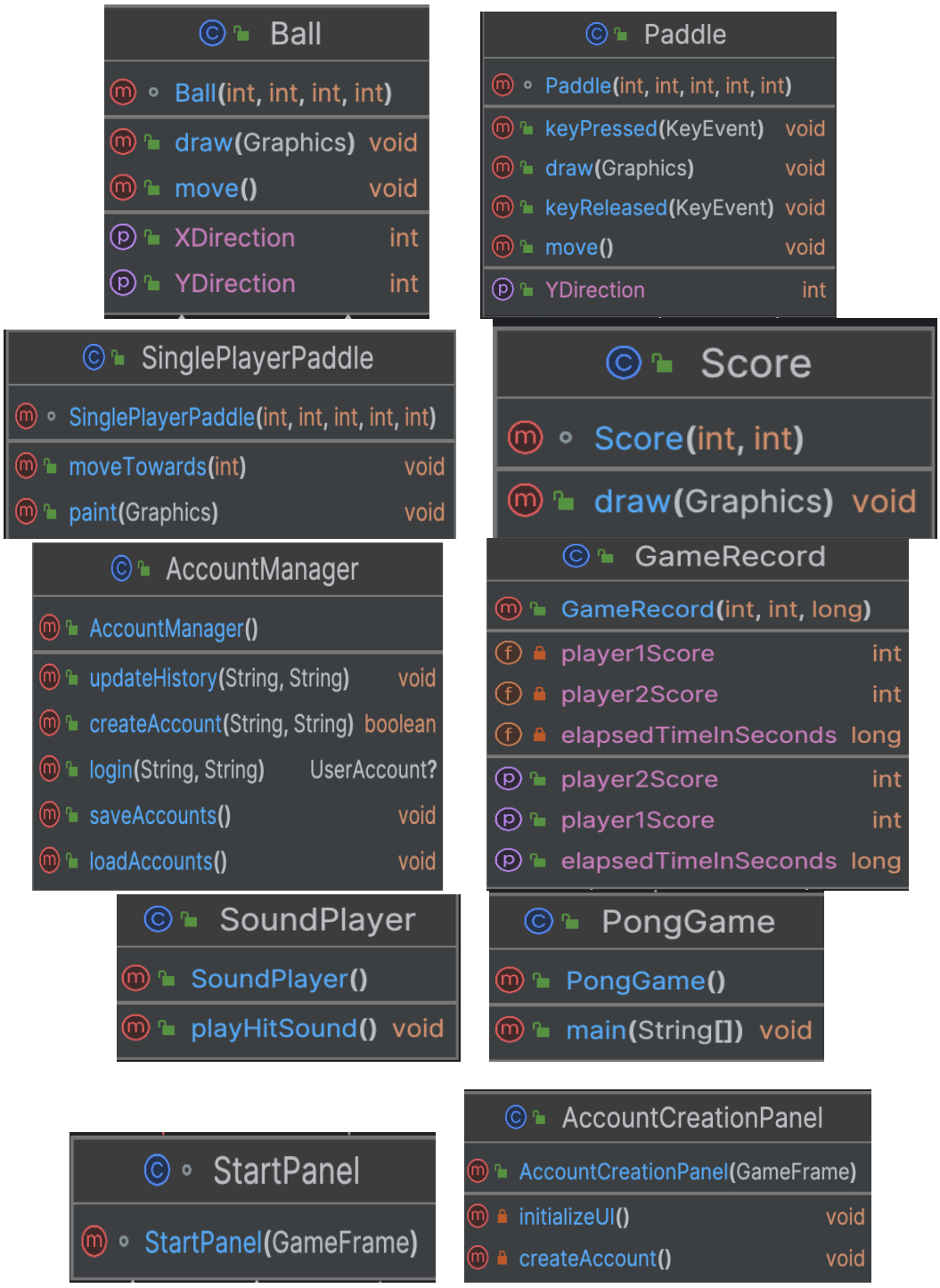
SoundPlayer	
Play sound effects during the game.	GamePanel

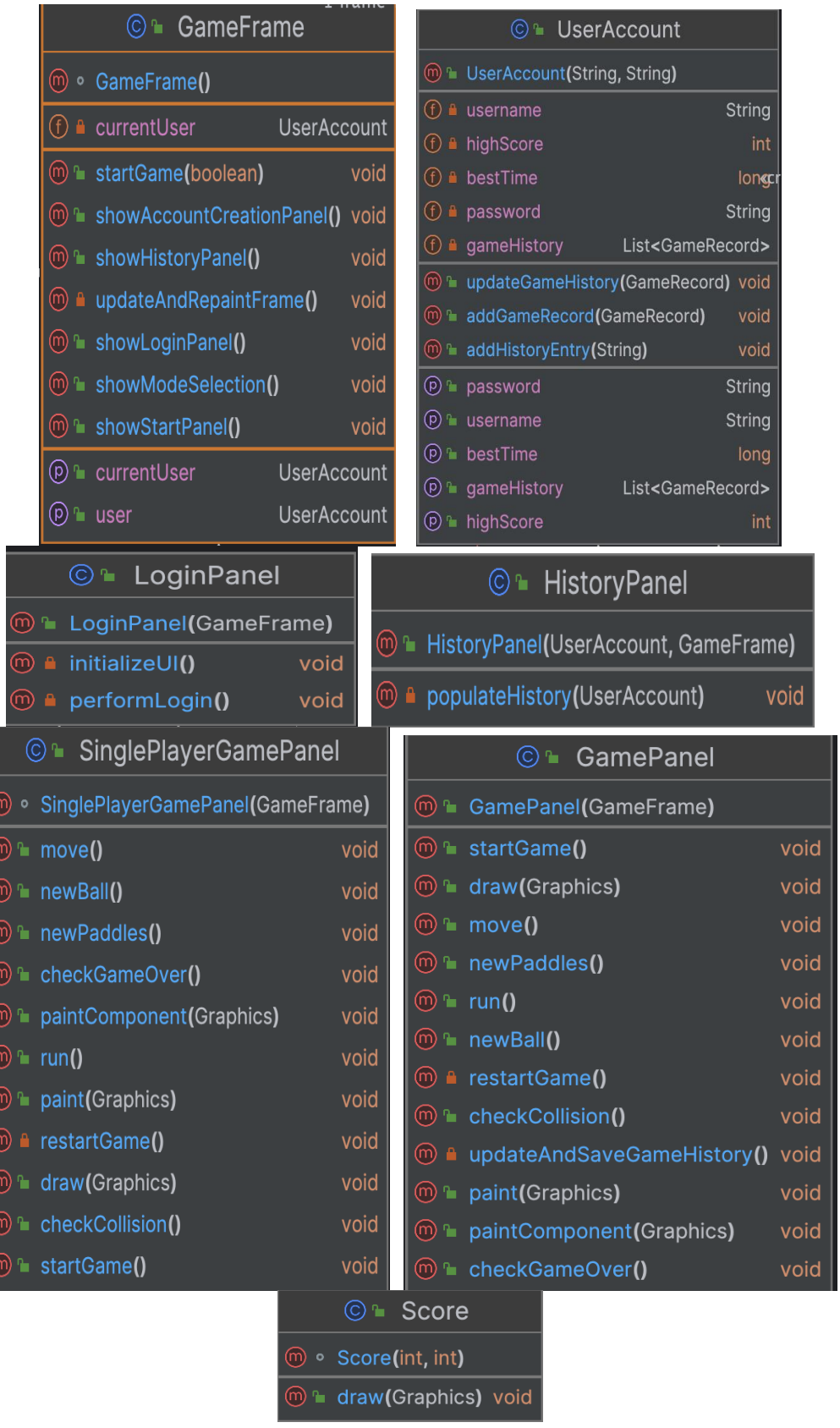
UserAccountManager	
Handle account creation and login processes.	LoginPanel AccountCreationPanel
Persist user account information.	GameRecord UserAccount

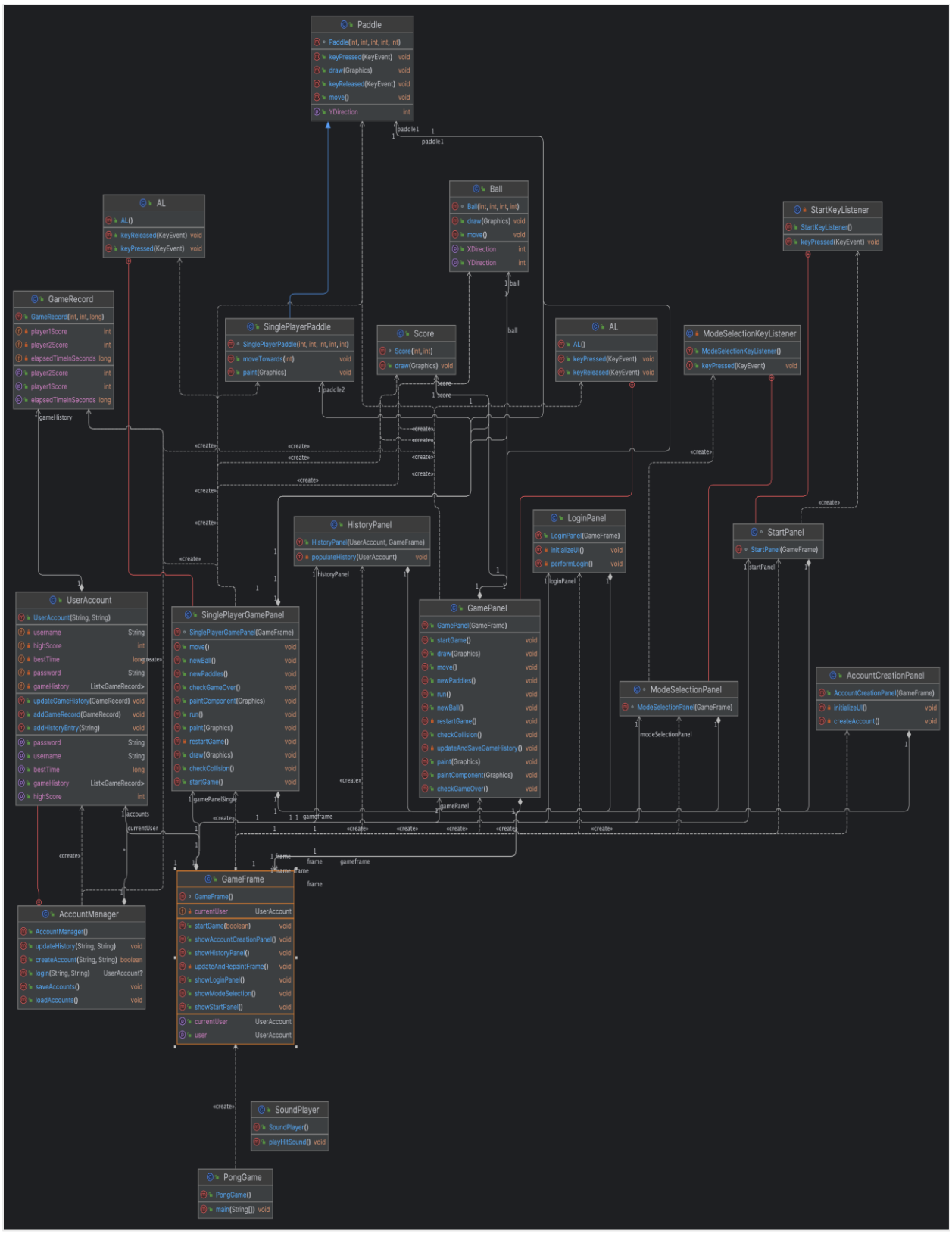
GameRecord	
Store individual game records.	HistoryPanel UserAccount
Provide data for hystorical analysis and display.	UserAccount

3. Class Diagram

A class diagram is a type of static structure diagram within the field of software engineering, specifically within the Unified Modeling Language (UML). It is used to visualize the classes within a system, their attributes, methods, and the relationships between these classes. Class diagrams are a mainstay of object-oriented analysis and design.


Ball
Ball(int, int, int, int)
draw(Graphics) void
move() void
XDirection int
YDirection int
Paddle
Paddle(int, int, int, int, int)
keyPressed(KeyEvent) void
draw(Graphics) void
keyReleased(KeyEvent) void
move() void
YDirection int
SinglePlayerPaddle
SinglePlayerPaddle(int, int, int, int, int)
moveTowards(int) void
paint(Graphics) void
Score
Score(int, int)
draw(Graphics) void
AccountManager
AccountManager()
updateHistory(String, String) void
createAccount(String, String) boolean
login(String, String) UserAccount?
saveAccounts() void
loadAccounts() void
GameRecord
GameRecord(int, int, long)
player1Score int
player2Score int
elapsedTimeInSeconds long
player2Score int
player1Score int
elapsedTimeInSeconds long
SoundPlayer
SoundPlayer()
playHitSound() void
PongGame
PongGame()
main(String[]) void
StartPanel
StartPanel(GameFrame)
AccountCreationPanel
AccountCreationPanel(GameFrame)
initializeUI() void
createAccount() void


GameFrame
GameFrame()
currentUser UserAccount
startGame(boolean) void
showAccountCreationPanel() void
showHistoryPanel() void
updateAndRepaintFrame() void
showLoginPanel() void
showModeSelection() void
showStartPanel() void
currentUser UserAccount
user UserAccount
UserAccount
UserAccount(String, String)
username String
highScore int
bestTime long
password String
gameHistory List<GameRecord>
updateGameHistory(GameRecord) void
addGameRecord(GameRecord) void
addHistoryEntry(String) void
password String
username String
bestTime long
gameHistory List<GameRecord>
highScore int
LoginPanel
LoginPanel(GameFrame)
initializeUI() void
performLogin() void
HistoryPanel
HistoryPanel(UserAccount, GameFrame)
populateHistory(UserAccount) void
SinglePlayerGamePanel
SinglePlayerGamePanel(GameFrame)
move() void
newBall() void
newPaddles() void
checkGameOver() void
paintComponent(Graphics) void
run() void
paint(Graphics) void
restartGame() void
draw(Graphics) void
checkCollision() void
startGame() void
GamePanel
GamePanel(GameFrame)
startGame() void
draw(Graphics) void
move() void
newPaddles() void
run() void
newBall() void
restartGame() void
checkCollision() void
updateAndSaveGameHistory() void
paint(Graphics) void
paintComponent(Graphics) void
checkGameOver() void
Score
Score(int, int)
draw(Graphics) void


Paddle
Paddle(int, int, int, int, int)
keyPressed(KeyEvent) void
draw(Graphics) void
keyReleased(KeyEvent) void
move() void
YDirection int
AL
AL()
keyReleased(KeyEvent) void
keyPressed(KeyEvent) void
Ball
Ball(int, int, int, int)
draw(Graphics) void
move() void
XDirection int
YDirection int
StartKeyListener
StartKeyListener()
keyPressed(KeyEvent) void
GameRecord
GameRecord(int, int, long)
player1Score int
player2Score int
elapsedTimeInSeconds long
SinglePlayerPaddle
SinglePlayerPaddle(int, int, int, int, int)
moveTowards(int) void
paint(Graphics) void
Score
Score(int, int)
draw(Graphics) void
AL
AL()
keyPressed(KeyEvent) void
keyReleased(KeyEvent) void
ModeSelectionKeyListener
ModeSelectionKeyListener()
keyPressed(KeyEvent) void
HistoryPanel
HistoryPanel(UserAccount, GameFrame)
populateHistory(UserAccount) void
LoginPanel
LoginPanel(GameFrame)
initializeUi() void
performLogin() void
StartPanel
StartPanel(GameFrame)
UserAccount
UserAccount(String, String)
username String
highScore int
bestTime long
password String
gameHistory List<GameRecord>
updateGameHistory(GameRecord) void
addGameRecord(GameRecord) void
addHistoryEntry(String) void
SinglePlayerGamePanel
SinglePlayerGamePanel(GameFrame)
move() void
newBall() void
newPaddles() void
checkGameOver() void
paintComponent(Graphics) void
run() void
paint(Graphics) void
restartGame() void
draw(Graphics) void
checkCollision() void
startGame() void
GamePanel
GamePanel(GameFrame)
startGame() void
draw(Graphics) void
move() void
newPaddles() void
run() void
newBall() void
restartGame() void
checkCollision() void
updateAndSaveGameHistory() void
paint(Graphics) void
paintComponent(Graphics) void
checkGameOver() void
ModeSelectionPanel
ModeSelectionPanel(GameFrame)
AccountCreationPanel
AccountCreationPanel(GameFrame)
initializeUi() void
createAccount() void
AccountManager
AccountManager()
updateHistory(String, String) void
createAccount(String, String) boolean
login(String, String) UserAccount?
saveAccounts() void
loadAccounts() void
GameFrame
GameFrame()
currentUser UserAccount
startGame(boolean) void
showAccountCreationPanel() void
showHistoryPanel() void
updateAndRepaintFrame() void
showLoginPanel() void
showModeSelection() void
showStartPanel() void
currentUser UserAccount
user UserAccount
SoundPlayer
SoundPlayer()
playHitSound() void
PongGame
PongGame()
main(String[]) void
«create»
paddle1
paddle2
ball
score
gameHistory
historyPanel
loginPanel
startPanel
gamePanel
gameFrame
modeSelectionPanel
accounts
currentUser
frame

4.Purpose

I chose to create this game because I think it is not only a versatile and enjoyable gaming experience that appeals to a wide array of audiences, but it is also as a tool for enhancing hand-eye coordination, reaction time, and strategic thinking. The idea came to me as an experience from my own life and an unpleasant moment of it. I tended to play a lot of ping pong with my boyfriend who, after he broke his hand, was unable to play anymore. But now, with this game I created, we can still play together, so as to not lose our dexterity. So, this game is also beneficial in this way.

5.Integration of Object-Oriented Programming Principles in Paddle Game Application

The Paddle game application developed in Java exemplifies several key principles of Object-Oriented Programming (OOP), contributing to a modular, maintainable, and scalable codebase. Below, I detail how these OOP principles are employed in the application:

- Encapsulation

Implementation: The Ball class in the Paddle game encapsulates the attributes and behaviors associated with the pong ball. This class combines properties such as xVelocity and yVelocity with methods like move() and draw(), encapsulating the state and functionalities pertinent to the ball's movement and rendering.
Advantages: This encapsulation ensures that the ball's internal state is safeguarded against external interference and manipulation, promoting data integrity and security. It also enhances the readability and maintainability of the code by localizing related functionalities within a single, coherent unit.

- Inheritance

Implementation: In the Paddle Game's architecture, the SinglePlayerPaddle class is a prime example of inheritance, as it extends from the base Paddle class. The Paddle class provides the basic structure and properties of a paddle, such as its dimensions and position on the game panel. The SinglePlayerPaddle inherits these fundamental attributes and methods from the Paddle class.

Advantages: While inheriting common features, SinglePlayerPaddle can also implement additional methods or modify inherited ones to cater to the specific requirements of single-player mode. For instance, it might include AI-driven movement logic to control the paddle's motion automatically.

- Polymorphism

Implementation: Polymorphism is primarily exhibited through method overriding within the game's classes. During the game, objects of Paddle and its subclasses are often stored in a collection (like an array or list). When iterating over this collection to update the game state

(like moving the paddles), the move() method is called on each object. Depending on the actual class of the object (whether it's a Paddle or a SinglePlayerPaddle), the appropriate move() method is executed - demonstrating polymorphism.
Advantages: Polymorphism allows for flexibility and dynamic behavior in the application. It enables the program to decide at runtime which specific method implementation to execute, depending on the object's class. This dynamic method dispatch is instrumental in creating a more flexible and extendable system.

- Conclusion

The incorporation of these Object-Oriented Programming principles significantly contributes to the robustness and coherence of the Pong game application. By adhering to OOP tenets, the application achieves a high level of modularity, scalability, and maintainability, pivotal for efficient software development and future enhancements.