Use Cases and Object-Oriented Modeling

Overview:

The purpose of this activity is to come up with the initial design of your team project. As you continue working on this software, the design will change, but the work done during this activity will become the foundation of your project design.

At the end of the activity you are asked to write skeleton code for your classes. This code will become part of your “first draft of project design” submission.

Learning objectives:

As you work on this activity you will:

- Apply the object-oriented design skills you learned in class
- Develop an understanding of difficulties and benefits of team work.

Team Number: 5

A. Define the use cases of your application. What can the user do?

Use Case 1: User chooses to start a new game

1. User types in username for each player.
2. Red chips are given to one player.
3. Black chips are given to the second player.

Use Case 2: User chooses a location for a chip

1. Players rotate turns - when it is a player’s turn, the user must choose a column to place a single chip.
2. Chip then falls to the lowest open space in the column that the user chose.
3. The board is checked for a winning combination of chips after the chip is placed

Use Case 3: User chooses whether to play another game or not

1. After the game is ended, a window pops up asking the user if they want to play again or end the game.

B. Define (in plain English) what happens in each use case (be specific)

1. In the first use case, the user decides to start the game. Users must enter two usernames, one for each player. The players then begin game play by receiving a certain number of chips that they can use throughout the game.

2. In the second use case, the players must make a decision on each turn whether to place a chip in one column or another. Chips get placed automatically in the lowest open space in a column. After the chip is placed, the board is checked for a winning combination of chips.

3. In the third use case, a window pops up and asks the player if they want to play again or end the game.

C. Identify Classes and Behaviors

Identify nouns (potential classes) in your use case descriptions

Column
Space
Chip
Player
Game

Identify verbs (potential behaviors/class methods) in your use case descriptions

Place
Play

Map verbs to nouns

Player places the chip in the space on the column
Player plays the game

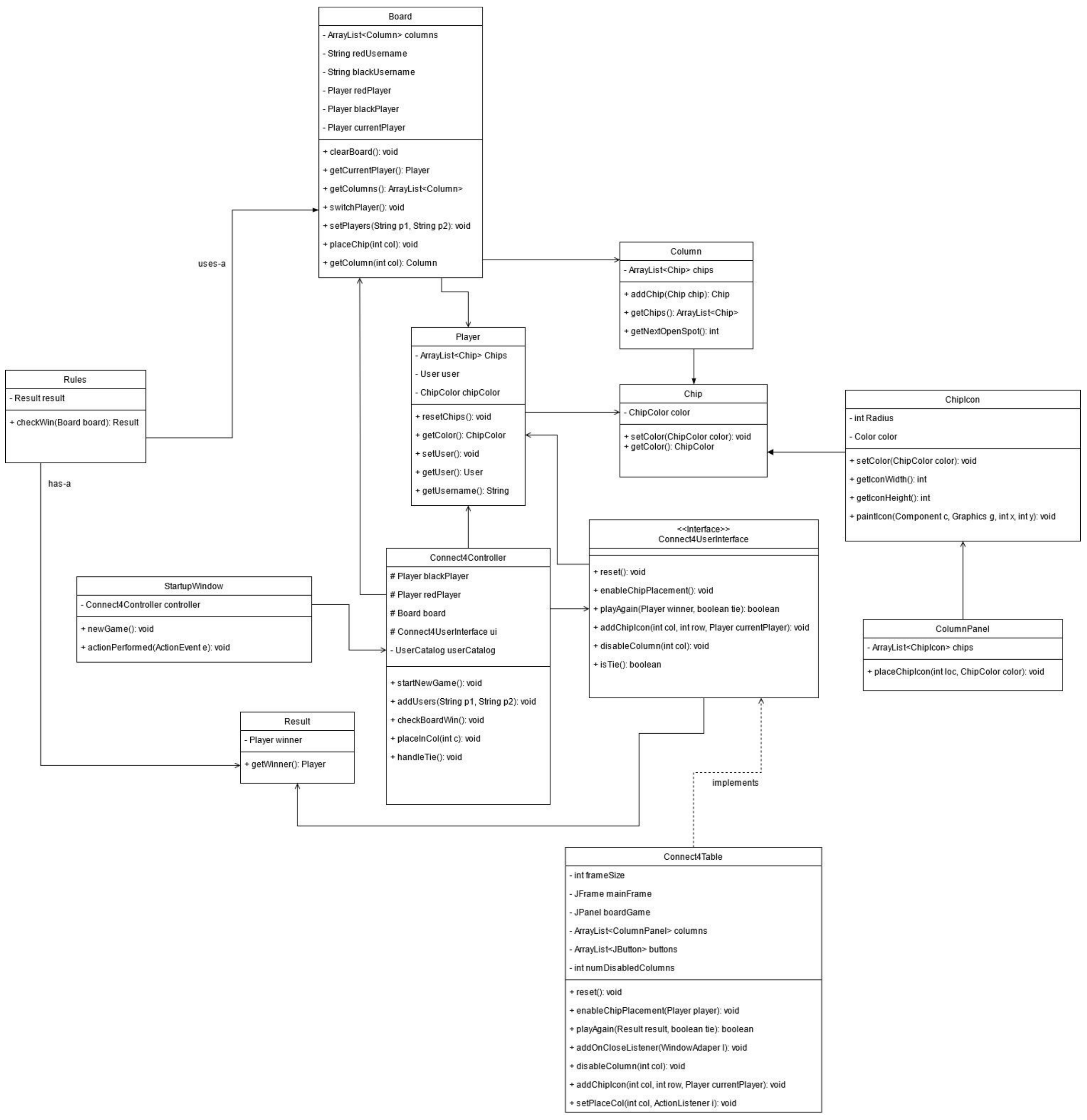
D. Re-write the use cases in terms of your classes and their behaviors

1. In the first use case, the user decides to start the game. Users must enter two usernames, creating multiple player objects. The players then begin game play by receiving a certain number of chip objects that they can use throughout the game.
2. In the second use case, the players must make a decision on each turn whether to place a chip in one column or another. Columns are objects that store information about how many and what color chips are in the column. Chips get placed automatically in the lowest open space in a column, based on the column’s own memory of what has been placed previously.

E. Sketch your Model

Create skeletons/stubs for the classes used in section D. A skeleton class defines a class name and all public methods of the class. However, the methods are not implemented. If a given method needs to return a value, just return some dummy value for now. Add print statements to each method, printing the name of the class and the name of the method.

Create a Driver class that uses your class skeletons to demonstrate the implementation of each use case. Label the start of each use case in the comments. For an example of this, consider the Driver.java of our first Black Jack implementation (you can find it in the BlackJack directory of your git repos).



Board
- ArrayList<Column> columns
- String redUsername
- String blackUsername
- Player redPlayer
- Player blackPlayer
- Player currentPlayer
+ clearBoard(): void
+ getCurrentPlayer(): Player
+ getColumns(): ArrayList<Column>
+ switchPlayer(): void
+ setPlayers(String p1, String p2): void
+ placeChip(int col): void
+ getColumn(int col): Column
uses-a
Column
- ArrayList<Chip> chips
+ addChip(Chip chip): Chip
+ getChips(): ArrayList<Chip>
+ getNextOpenSpot(): int
Player
- ArrayList<Chip> Chips
- User user
- ChipColor chipColor
+ resetChips(): void
+ getColor(): ChipColor
+ setUser(): void
+ getUser(): User
+ getUsername(): String
Rules
- Result result
+ checkWin(Board board): Result
has-a
Chip
- ChipColor color
+ setColor(ChipColor color): void
+ getColor(): ChipColor
ChipIcon
- int Radius
- Color color
+ setColor(ChipColor color): void
+ getIconWidth(): int
+ getIconHeight(): int
+ paintIcon(Component c, Graphics g, int x, int y): void
<<Interface>>
Connect4UserInterface
+ reset(): void
+ enableChipPlacement(): void
+ playAgain(Player winner, boolean tie): boolean
+ addChipIcon(int col, int row, Player currentPlayer): void
+ disableColumn(int col): void
+ isTie(): boolean
StartupWindow
- Connect4Controller controller
+ newGame(): void
+ actionPerformed(ActionEvent e): void
Connect4Controller
# Player blackPlayer
# Player redPlayer
# Board board
# Connect4UserInterface ui
- UserCatalog userCatalog
+ startNewGame(): void
+ addUsers(String p1, String p2): void
+ checkBoardWin(): void
+ placeInCol(int c): void
+ handleTie(): void
ColumnPanel
- ArrayList<ChipIcon> chips
+ placeChipIcon(int loc, ChipColor color): void
Result
- Player winner
+ getWinner(): Player
implements
Connect4Table
- int frameSize
- JFrame mainFrame
- JPanel boardGame
- ArrayList<ColumnPanel> columns
- ArrayList<JButton> buttons
- int numDisabledColumns
+ reset(): void
+ enableChipPlacement(Player player): void
+ playAgain(Result result, boolean tie): boolean
+ addOnCloseListener(WindowAdaper l): void
+ disableColumn(int col): void
+ addChipIcon(int col, int row, Player currentPlayer): void
+ setPlaceCol(int col, ActionListener i): void