Homework 2 Summary

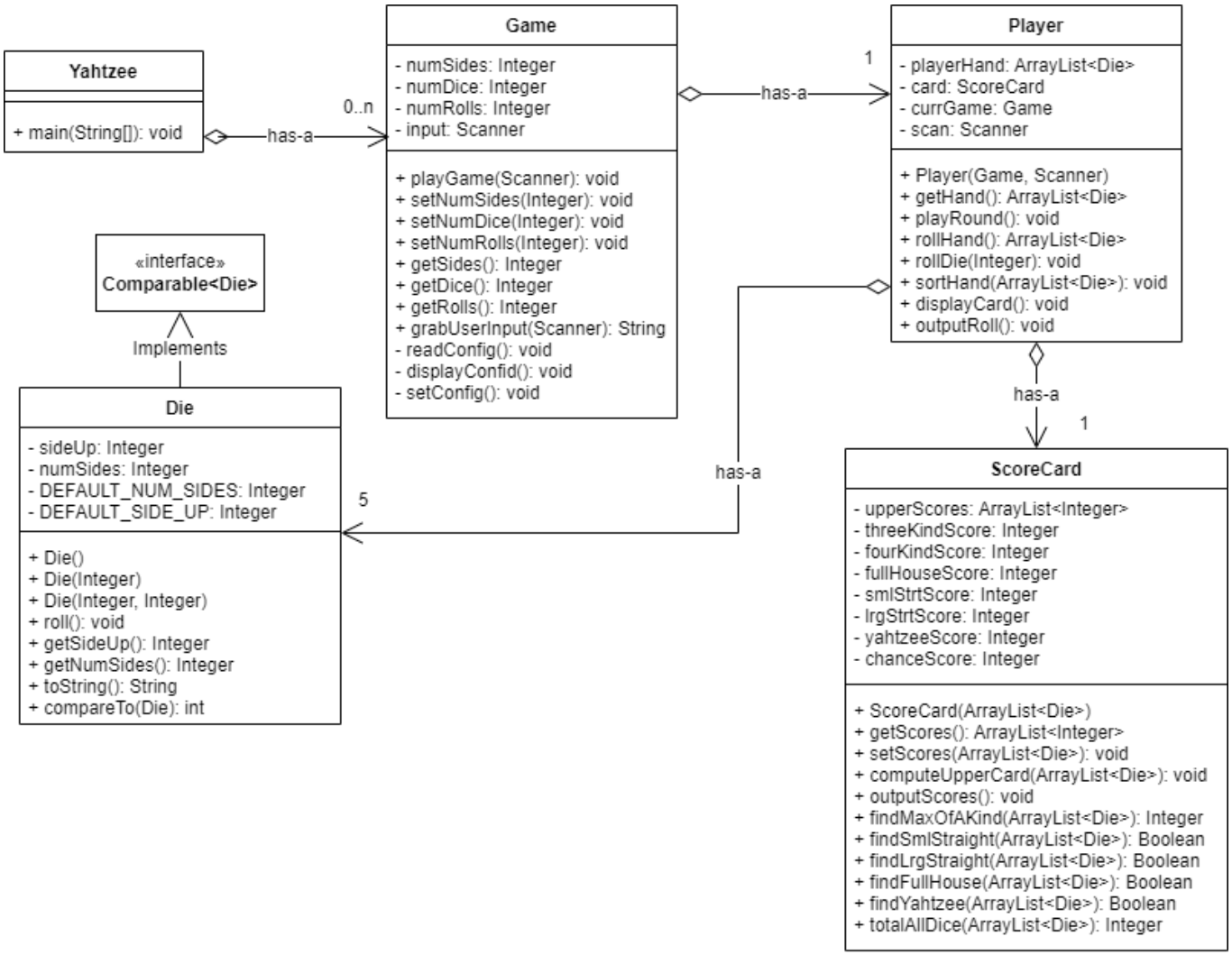
The second version of this Yahtzee project should continue to implement a set of classes that for relationships between the different parts of the board game Yahtzee. Those relationships involve a game to play, a player that can play a round and calculate scores during that round. The status of a player's score and hand should be updated as well. As these elements are all present in the first version of this project, this second version should add something more. Thus, a user should now be able to configure certain game settings like how many dice are used, how many sides those dice have, and even how many times they can re-roll during their turn.

To meet the goals of this version of Yahtzee each class had to be updated. The game class now handles changing the game configuration in addition to running a game loop. The player class still handles playing a round, but now the current game configuration influences the player's hand and score so there was a need to instantiate a game object within player for the use of game-specific methods and fields. The scorecard class also requires the passing in of the current game configuration as the upper card adjusts to the given number of sides. However, most of this scorecard class has remained identical to the first version.

While it is interesting to cover any one scorecard test as each calculation of a Yahtzee-specific scorecard line has a distinct test, these functions all depend on a method found in the player class that organizes a player's hand. This being a test of the implemented bubble sort algorithm that sorts a player's hand from smallest to largest. Said method is called right before a newly rolled hand is called into the scorecard object for review. The "testSorting()" method, found within the "PlayerTest" harness, sorts a hand of 6-sided dice with consecutive sides up from 1-5. Because the test passes, it means that the sorting algorithm used within Yahtzee properly sorts a hand such that the scoring methods within the scorecard can iterate and compute correct scores.

As mentioned before, a major issue I encountered was generalizing the methods in the scorecard class to fit die of any number of sides and hands of any size though, luckily, this was not a distinct requirement for this second version of the Yahtzee project. Nevertheless, this can be named an issue because if I fail to minorly abstract these scoring methods, games that involve hands and dice of larger sizes will fail to compute scores correctly. To address this issue, the configuration settings chosen by a user and stored at the game class level have been passed all the way into the scorecard through the hand ArrayList object. Therefore, these integer values can be used as terminators in loops and conditions in logical expressions that meet the current configuration of Yahtzee.

Given more time, I would spend more time refactoring the scoring methods for each line of a Yahtzee scorecard. Because the user can now change certain aspects of Yahtzee, the score calculations should adapt to these new conditions. The full house score, for instance, where 2 of one side and three of another qualifies for a hand with 5 dice, the same is not true for hands of larger size. Though I attempted to make this change in implementation, I could not devote enough time to fully and correctly updating these methods for the submission of this second version of Yahtzee.


Yahtzee
+ main(String[]): void
has-a
0..n
Game
- numSides: Integer
- numDice: Integer
- numRolls: Integer
- input: Scanner
+ playGame(Scanner): void
+ setNumSides(Integer): void
+ setNumDice(Integer): void
+ setNumRolls(Integer): void
+ getSides(): Integer
+ getDice(): Integer
+ getRolls(): Integer
+ grabUserInput(Scanner): String
- readConfig(): void
- displayConfid(): void
- setConfig(): void
has-a
1
Player
- playerHand: ArrayList<Die>
- card: ScoreCard
- currGame: Game
- scan: Scanner
+ Player(Game, Scanner)
+ getHand(): ArrayList<Die>
+ playRound(): void
+ rollHand(): ArrayList<Die>
+ rollDie(Integer): void
+ sortHand(ArrayList<Die>): void
+ displayCard(): void
+ outputRoll(): void
«interface»
Comparable<Die>
Implements
Die
- sideUp: Integer
- numSides: Integer
- DEFAULT_NUM_SIDES: Integer
- DEFAULT_SIDE_UP: Integer
+ Die()
+ Die(Integer)
+ Die(Integer, Integer)
+ roll(): void
+ getSideUp(): Integer
+ getNumSides(): Integer
+ toString(): String
+ compareTo(Die): int
5
has-a
has-a
1
ScoreCard
- upperScores: ArrayList<Integer>
- threeKindScore: Integer
- fourKindScore: Integer
- fullHouseScore: Integer
- smlStrtScore: Integer
- lrgStrtScore: Integer
- yahtzeeScore: Integer
- chanceScore: Integer
+ ScoreCard(ArrayList<Die>)
+ getScores(): ArrayList<Integer>
+ setScores(ArrayList<Die>): void
+ computeUpperCard(ArrayList<Die>): void
+ outputScores(): void
+ findMaxOfAKind(ArrayList<Die>): Integer
+ findSmlStraight(ArrayList<Die>): Boolean
+ findLrgStraight(ArrayList<Die>): Boolean
+ findFullHouse(ArrayList<Die>): Boolean
+ findYahtzee(ArrayList<Die>): Boolean
+ totalAllDice(ArrayList<Die>): Integer