Yahtzee Program

Team 10

Jaye Norman - sjnorma2

[redacted]

[redacted]

[redacted]

NOTE: Your final documents should **NOT** contain any << >>'s or any comments or suggestions given for each section.

Introduction

The goal of this project is to develop a Java application that allows users to play the popular dice game Yahtzee. Yahtzee involves 2 - 4 players and utilizes a set of five (5) dice. During a player's turn, they have the opportunity to roll the dice a maximum of three (3) times. After the initial roll, the player may choose to keep any number of dice and reroll the rest; a player may have the potential of improving their score for that particular round. The main objective is to strategically obtain as many points as possible for each round. The players will use their dice combinations from each roll to gain points in their scorecard with categories listed below:

Score Category	**Description**	**Points**
Ones	Get as many ones as possible	(# ones) * 1
Twos	Get as many twos as possible	(# twos) * 2
Threes	Get as many threes as possible	(# threes) * 3
Fours	Get as many fours as possible	(# fours) * 4
Fives	Get as many fives as possible	(# fives) * 5
Sixes	Get as many sixes as possible	(# sixes) * 6
3 of a Kind	Get three dice of the same value	Sum of all dice
4 of a Kind	Get four dice of the same value	Sum of all dice
Full House	Get three of a kind and a pair	25
Small Straight	Get four sequential dice	30
Large Straight	Get five sequential dice	40
Chance	Any combination of dice	Sum of all dice
YAHTZEE	Get five dice of the same value	50

Additionally, the numerical categories (highlighted in yellow) allow a player the opportunity of receiving a bonus. If the sum of the highlighted categories is greater than or equal to 63, an additional bonus of 35 points is awarded to the player and is added to their overall score. Each player is allotted 13 turns throughout the game.

The game ends when all players have filled in all categories on their scorecards. At the end, the grand total for each player is calculated, and the player with the highest score wins.

The strategy of the game is to balance and decide between earning high-scoring dice combinations and filling in categories strategically.

Software Requirements

This software will allow the user to play a game of Yahtzee with 2 - 4 people. The user will be able to specify the number of players, and each player will be given a label (Player1, Player2, etc). The game will allow each player to roll five dice at the beginning of their turn. Then, they can roll the dice up to two more times, choosing which dice to roll each time (for example, they can hold 3 of the dice and roll the other 2). The program will allow the user to choose a score category (outlined in the Introduction above) and allot the points. After a user has chosen a score category, they will not be able to use this category again. If the user is unable to gain any points, they can choose any category to assign a score of 0. At the end of the game, all the player's scores are tallied and the player with the highest score wins!

The program will take one command line argument, an integer between 2 and 4 to represent the number of players.

From csc116-651-S24_CE_T10 repository :

The YahtzeeGame program will be compiled as follows:

$ javac -d bin -cp bin src/YahtzeeGame.java

The YahtzeeGame program will be executed as follows:

$ java -cp bin YahtzeeGame.java numberPlayers

$ java -cp bin YahtzeGame.java numberPlayers -t (testing mode)

CLA Error Handling:

- If the number of command line arguments is not one or two, quit program with message "*Usage: java -cp bin YahtzeeGame.java numberPlayers*"
- If the first command line argument is not an int, quit program with message "Number of players must be int"
- If number of players is greater than 4 or less than 2, throw IllegalArgumentException "Invalid number of players"

During gameplay, the user interface will:

1. Display which player's turn it is, and show their current scorecard.
2. Roll all five dice, labeling their results as "Roll 1".
3. Give the user the option to choose a score or roll again.
    a. If the user chooses to score, skip to step 9
4. If the user chooses to roll again, prompt for how many dice to hold and which dice to hold (dice will be labeled 1 - 5)
5. Roll dice specified by user, showing all five dice after rolling and labeling results as "Roll 2"
6. Give the user the option to choose a score or roll again
    a. If the user chooses to score, skip to step 9
7. If the user chooses to roll again, prompt for how many dice to hold and which dice to hold(dice will be labeled 1 - 5)
8. Roll dice specified by user, showing all five dice after rolling and labeling results as "Roll 3"
9. Prompt the user to choose a score option.
10. Display the option that the user chose and how many points will be gained from this.

At the end of the game, the user interface will:

1. Show each player's scorecard in succession
2. Announce the rankings, each player and their scores.

Example User Interface during a turn:

```
None
It is Player1's turn!

Player1's scorecard:
Category         Score
Ones             8
Twos             -
Threes           -
Fours            12
Fives            -
Sixes            12
Upper Total      32
Bonus            0
Three of a Kind  -
Four of a Kind   -
Full House       -
Small Straight   30
Large Straight   -
Chance           27
YAHTZEE          -
Total            89


Roll 1:
Die 1       Die 2       Die 3       Die 4       Die 5
+-------+   +-------+   +-------+   +-------+   +-------+
| o   o |   |     o |   | o   o |   | o   o |   | o   o |
|       |   |   o   |   | o   o |   |       |   |       |
| o   o |   | o     |   | o   o |   | o   o |   | o   o |
+-------+   +-------+   +-------+   +-------+   +-------+

Reroll or score? reroll

How many dice to hold? 4

Which dice to hold? 1 2 4 5
```

```
Roll 2:
Die 1       Die 2      Die 3       Die 4      Die 5
+-------+   +-------+   +-------+   +-------+   +-------+
| o   o |   | o   o |   |       |   | o   o |   | o   o |
|       |   |       |   |   o   |   |       |   |       |
| o   o |   | o   o |   |       |   | o   o |   | o   o |
+-------+   +-------+   +-------+   +-------+   +-------+

Choose a score or reroll?
Reroll

How many dice to hold? 4
Which dice would you like to hold? 1 2 4 5

Roll 3:
Die 1       Die 2      Die 3       Die 4      Die 5
+-------+   +-------+   +-------+   +-------+   +-------+
| o   o |   | o   o |   | o   o |   | o   o |   | o   o |
|       |   |       |   |       |   |       |   |       |
| o   o |   | o   o |   | o   o |   | o   o |   | o   o |
+-------+   +-------+   +-------+   +-------+   +-------+


Select a category: YAHTZEE

Player1's scorecard:
Category         Score
Ones             8
Twos             -
Threes           -
Fours            12
Fives            -
Sixes            12
Upper Total      32
Bonus            0
Three of a Kind  -
Four of a Kind   -
Full House       -
Small Straight   30
Large Straight   -
Chance           27
YAHTZEE          50
Total            139
```

User Input Error Handling:

- If the player inputs a dice to be held that is not 1 - 5, output “Dice to hold must be between 0 and 5” and reprompt until valid dice are chosen.
- If the player inputs a number of dice to hold that is not between 0 and 5, output “Number of dice to hold must be between 0 and 5” and reprompt until a valid number is chosen.
- If the player inputs something other than the option to score or reroll, output “Invalid option” and reprompt.
- If the player chooses a score option that does not exist, output “Invalid score category” and reprompt.
- If the player chooses a score option that has already been used, output “This score category has already been used” and reprompt.

Software Design

Remember: You still will not be writing code at this point in the process.

Die class:

```
/** value of the die */
private int value;
/** If the dice is held or not */
private boolean isHold;

/** Creates die object with initial value of -1  and is not held*/
public Die()

/** Returns the value of the die
* @return value of die
 */
public int getValue()

/** Gives the die a random value of 1 - 6 */
public void roll()

/** Returns if the die is held or not
* @return True if the dice is held, false if not
*/
public boolean isHold()

/** Sets the dice isHold to true */
public void hold()

/** Sets the dice isHold to false */
public void unHold()

/** Sets the dice's value to the given int
* (useful for test mode)
* @param userVal value to set die to
*/
Public void setValue(int userVal)

/** Returns string in the format of ASCII die corresponding to value
* @return String representation of value of die
* @throws IllegalArgumentException if value < 1 or > 6
 */
public String toString()
```

DiceSet class:

```
/** A set of five die objects for the Yahtzee game */
private Die[] dice;

/** Constructs the dice array with five Die[] objects */
```

```
public DiceSet()

/** Returns the value of a die at a given index
* @param index Index of die to getValue
* @return value of die at index
* @throws IllegalArgumentException if index is not 0 - length of dice
*/
public int getValue(int index)

/** Returns if a die is held at a given index
* @param index Index of die to get isHold
* @return true if dice at index is held, false if not
* @throws IllegalArgumentException if index is not 0 - length of dice
*/
public boolean isHold(int index)

/** Holds a die at a given index */
* @param index Index of die to hold
* @throws IllegalArgumentException if index is not 0 - length of dice
*/
public void hold(int index)

/** Unholds a die at a given index */
* @param index Index of die to hold
* @throws IllegalArgumentException if index is not 0 - length of dice
*/
public void unhold(int index)

/** Rolls a die at a given index */
* @param index Index of die to roll
* @throws IllegalArgumentException if index is not 0 - length of dice
*/
public void roll(int index)

/** Returns the number of dice in
* the DiceSet
* @return number of dice
*/
public int getLength()

 /** Sets the value of the die to the given value
* (useful for testing mode)
* @param userVal value to set die to
* @param index Index of die to set
* @throws IllegalArgumentException if index is not 0 - length of dice
*/
public void setValue(int userVal, int index)

/** Returns all five die, labeled and with ascii art corresponding to their value
```

```
* @return string representation of all five die's values
*/
public String toString()
```

Scorecard Class

```
/** Upper categories current scores */
private int[] upperCategories;

/** total of all the upper categories */
private int totalUpper;

/** value of bonus for upper categories */
private int bonus;

/** Lower categories current scores */
private int[] lowerCategories;

/** Total score, including upper category bonus */
private int totalScore;

/** Creates a scorecard object, setting all categories to -1 */
public Scorecard()

/* If the ones score is chosen, multiplies the number of ones in the dice
* times 1 to get score for category, then sets the 0 index of upperCategories to
* the score
* @param dice DiceSet to be scored
* @return score of ones category
*/
public int ones(DiceSet dice)

/* If the twos score is chosen, multiplies the number of twos in the dice
* times 2 to get score for category, then sets the 1 index of upperCategories to
* the score
* @param dice DiceSet to be scored
* @return score of twos category
*/
public int twos(DiceSet dice)

/* If the threes score is chosen, multiplies the number of threes in the dice
* times 3 to get score for category, then sets the 2 index of upperCategories to
* the score
* @param dice DiceSet to be scored
* @return score of threes category
*/
public int threes(DiceSet dice)

/* If the fours score is chosen, multiplies the number of fours in the dice
* times 4 to get score for category, then sets the 3 index of upperCategories to
```

```
* the score
* @param dice DiceSet to be scored
* @return score of fours category
*/
public int fours(DiceSet dice)

/* If the fives score is chosen, multiplies the number of fives in the dice
* times 5 to get score for category, then sets the 4 index of upperCategories to
* the score
* @param dice DiceSet to be scored
* @return score of fives category
*/
public int fives(DiceSet dice)

/* If the sixes score is chosen, multiplies the number of sixes in the dice
* times 6 to get score for category, then sets the 5 index of upperCategories to
* the score
* @param dice DiceSet to be scored
* @return score of sixes category
*/
public int sixes(DiceSet dice)

/** Returns the sum of the ones - sixes categories and sets 6 index of
* upperCategories to sum
* @return Sum of upper categories
*/
public int upperSum()

/** If the upperSum is 63 or more, set the bonus variable to 35, otherwise
* set to 0 and set 7 index of upperCategories to bonus
* @param totalUpper Total value of the upper categories
*/
public void bonus()

/** If the three of a kind choice is chosen, return the total of all five dice to
* the 0 index of lowerCategories, unless there are not 3 of a kind, then
* return 0 to 0 index
* @param dice DiceSet to be scored
* @return score of threeOfAKind category
*/
public int threeOfAKind(Diceset dice)

/** If the four of a kind choice is chosen, return the total of all five dice to
* the 1 index of lowerCategories, unless there are not four of a kind, then
* return 0 to 1 index
* @param dice DiceSet to be scored
* @return score of the Four of a Kind category
*/
public int fourOfAKind(DiceSet dice)
```

```
/** If the full house choice is chosen, return 25 to the
* the 2 index of lowerCategories, unless two dice do not
* have the same value and the other three do not have
* the same value, then return 0 to the 2 index
@param dice DiceSet to be scored
@return score of Full House Category
*/
public int fullHouse(DiceSet dice)

/** If the small straight choice is chosen, return 30 to the
* the 3 index of lowerCategories, unless there are not
* four sequential dice, then return 0 to the 3 index
@param dice DiceSet to be scored
@return score of Small Straight Category
*/
public int smallStraight(DiceSet dice)

/** If the large straight choice is chosen, return 40 to the
* the 4 index of lowerCategories, unless the dice are not sequential
* then return 0 to the 4 index
@param dice DiceSet to be scored
@return score of LargeStraight Category
*/
public int largeStraight(DiceSet dice)

/** If the chance choice is chosen, return the sum of the dice
* to the 5 index of lowerCategories
@param dice DiceSet to be scored
@return score of chance Category
*/
public int chance(DiceSet dice)

/** If the yahtzee choice is chosen, return 50 to the
* the 6 index of lowerCategories, unless all dice are not the same
* value, then return 0
@param dice DiceSet to be scored
@return score of yahtzee Category
*/
public int yahtzee(DiceSet dice)


/** Returns the total score, which includes the totalUpper, bonus
* and sum of the lower categories
* @return total score
*/
public int getTotalScore()

/** Returns a full scorecard in string form, with categories not yet used
```

```
* having a dash
* @return String representation of scorecard
*/
public String toString()

/** Determines if a scorecard is full, returning true
* if full and false otherwise
* @return true if full
*/
public boolean isFull()

/** Determines if a user input option has already
* been used on the scorecard before
* @param option User-input string category
* @return true if used, false otherwise
*/
public boolean optionHasBeenUsed(String option)
```

YahtzeeGame Class:

```
/** Array of players with names */
public String playerList;

/** Array of scorecards corresponding to amount of players */
public Scorecard[] scorecards;

/** String representing the current player's name*/
public static String currentPlayer = "";

/** Current player's index in parallel arrays */
public static int currentPlayerIndex = -1;

/** Outputs user interface's text to the console, calls the methods
* to control the game, checks for user input errors and makes the
* program robust
* @param args Command line arguments
* @throws IllegalArgumentException invalid number of players
*/
public static void main(String[] args)

/** Rolls the dice in the diceset that are not held
*and returns the result using the toString method in
* the DiceSet class
* @param dice DiceSet to be rolled
* @param testing True if in testing mode
* @return String representation of all five dice results
*/
public static String rollDice(DiceSet dice, boolean testing)
```

```
/** Holds the die that the user gives, noting that
* the user will see die 1 - 5 while the indexes are 0 - 4
* @param userDieIndex int index of die to be held
* @param dice Array of dice to act on
* @throws IllegalArgumentException if userDieIndex is not 1 - 5
*/
public static void holdDice(int userDieIndex, DiceSet dice)

/** Prints the scorecard of a player by finding the index of
* the playerID and calling the toString method from the scorecard
* in that index of the scorecards array
* @param playerID String ID of player
* @throws IllegalArgumentException if playerID is not Player1 - Player4
*/
public static void printScorecard(String playerID)

/** Readies the dice for the nextTurn by setting hold to false
* @param dice DiceSet to be reset
* @param currentPlayerIndex index of player
* @param playerList array of num of players
* @return int index of current player in the array
*/
public static void nextTurn(DiceSet dice, int currentPlayerIndex, String[] playerList)



/** Checks if all scorecards are filled
* @param scorecards array of scorecards
* @return true if all scorecards are filled and game should end, otherwise false
*/
public static boolean isGameOver(Scorecard[] scorecards)


/** Determines the winner by seeing the totalScore amount and then printing
* the winner's name from playerList and their score
* @param scorecards Scorecards of all players
* @param playerList List of all players
* @throw IllegalArgumentException if Scorecards length is not equal to playerList length
* @throw IllegalArgumentException if scorecards or playerList length < 1 or > 4
*/
public static void nameWinner(Scorecard[] scorecards, String[] playerList)

/**
* Asks player if they would like to reroll a die
* @param dice specific dice object
* @param console response from user
*/
public static void reroll(DiceSet dice, Scanner console)
```

```
/**
* outputs the result of the user selecting a category
* to go toward their overall score
* @param dice dice object
* @param scorecards array of players scorecards
* @param currentPlayerIndex current index of player
* @param console option from user
*/
public static void score(DiceSet dice, Scorecard[] scorecards,
int currentPlayerIndex, Scanner console)
```

Implementation

This program leverages many concepts covered in the course. Concepts from basic syntax and utilizing the Java API to optimize our codebase via object oriented programming (OOP) are used.

OOP allows us to create well-organized code by utilizing objects within dedicated classes. Constructors and public/private member variables are implemented to enable manipulation of data in a both clear and secure way. Each object has its own unique behavior which promotes independence and reduces reliance on other objects.

OOP offers many benefits such as code reusability, collaboration and modularity - all of which are essential aspects of software engineering. Software engineers leverage open-source code from various online sources, including developer hubs such as GitHub, DockerHub, GitLab, to even a random Reddit thread from eight years ago at 2am, depending on the project's complexity and necessary functions.

We will utilize many concepts covered in labs, such as exceptions. Exceptions allow the program to define user limitations which ensures the stability of the program and prevents possible data corruption. Command line arguments are also implemented, allowing the user to enter parameters from the command line while calling the program, simplifying the user interface, and collecting data vital to the running of the program.

Yahtzee classes and their implementation:

- *Dice(){}* - Utilizes constructors and OOP, allowing object manipulation of a single die.
    - Uses constructor to hold initial values of a dice
    - Uses String concatenation to provide formatted dice to the user

- *DiceSet(){}:* - Enhances program stability via exception handling and OOP for object manipulation.
    - Uses 2 dimensional String array to provide formatted dice set to the user
    - Uses constructor to create an array of multiple objects

- *Scorecard(){}:* Manipulation of dice to add up score depending on the scorecard categories
    - Uses arrays to store and retrieve player scores
    - Uses switch statements to determine the number of each dice with a certain value to determine the score options
    - Uses String concatenation to provide a formatted scorecard

- *YahtzeeGame(){}* - User Interface, the game functionalities
    - Provides in depth game of Yahtzee, following flow control.
    - Uses command line arguments to determine the number of players and the testing mode
    - Uses character and string methods and logic to provide a robust program
    - Uses do-while loop to execute game one time before requiring user input to continue or quit
    - Uses parallel arrays to organize Player's IDs and scorecards

Testing

- System Testing
  The System testing of the Yahtzee program should be performed according to the SystemTestPlan_CE.pdf document. List any test files required to run these tests. Discuss any special testing scenario requirements (for example, test arrays with prespecified values to be used instead of random numbers while in test mode, command line arguments to run in test mode, etc).
    - For some test cases, the files in the test-files directory are required (fullGameTest, fullGameTestOutput, playAgainTest, playAgainTestOutput)
    - The use of the flag “-t” after the number of players in the command line arguments activates testing mode, which makes the first dice rolled have a value of 1, second 2, etc.(note: this does mean some score categories are impossible to achieve, so white-box testing using random dice rolls are also included on the System Test Plan for completeness)
- Unit Testing - no unit test files or documentation are required this semester, so this can be omitted.

Team Reflection

- Challenges faced/Lessons learned
    - What did the team learn during the project development process?
    - What problems arose and how did you solve them?

In addition to the above team reflection that must be included in this document each team member must also complete an individual reflection (not in this document).

The following information should be included in each individual's peer review form of themselves.

- What did you like/dislike about the exercise?
- Any suggested changes/improvements?
- Add any comments/thoughts you care to share.