Yahtzee

Team 7

Names and Unity IDs: Sarah Byrd (slbyrd) and Jessica Greene (jmgree22)

NOTE: You will replace all placeholders that are given in <<>>

Introduction 2

Software Requirements 3

Software Design 8

**The Dice Class** 9

**The Scorecard Class** 12

**The Yahtzee Class** 15

Implementation 16

**Dice.java Implementation** 17

**Scorecard.java Implementation** 18

**Yahtzee.java Implementation** 21

Testing 22

# Introduction

This version of the game Yahtzee is for 2 - 4 players. The game will have 5 dice and 13 rounds. After 13 rounds, the winner will be the player with the highest score. In each of the 13 rounds, the player can do two things: roll the dice and then score that roll into a specific category. Each of the categories gets only 1 score. The score is determined by a different rule for each category.

DESCRIPTION OF PLAYER’S TURN:

1. Roll 5 dice together
2. Score the current roll or re-roll any or all of the 5 dice
3. The dice can be rolled up to 3 times -- the first roll (in which all 5 dice must be rolled), and then up to 2 re-rolls of any or all dice
4. The player must score their roll in the category of their choosing and then it is the next player’s turn
5. End of that player’s current turn

Steps 1 - 5 will be repeated until all players have completed 13 rounds.

EXPLANATION OF CATEGORIES AND SCORES IN TABLE BELOW

|  |  |  |
| --- | --- | --- |
| **Category** | **Requirements** | **Points** |
| **Categories 1 - 6** | **At least one die face with the number** | **Sum of all dice that show that number** |
| **Upper Bonus** | **63 or more total points on categories 1 – 6 at the end of the game** | **35 points** |
| **3 of a kind** | **At least 3 of the same value** | **Total all 5 dice** |
| **4 of a kind** | **At least 4 of the same value** | **Total of all 5 dice** |
| **Small straight** | **4 consecutive numbers** | **30 points** |
| **Large straight** | **5 consecutive numbers** | **40 points** |
| **Full House** | **3 of one number, 2 of another (or same) number** | **25 points** |
| **Yahtzee** | **All five die faces the same** | **50 points** |
| **Chance** | **Anything works here** | **Total of all the dice** |

DETAILED SCORE AND CATEGORY EXPLANATION WITH EXAMPLE:

The player can have an upper score or a lower score.

UPPER SCORE:

* If when game is completed the player has 63 or more points from the upper categories, they will then get an added **extra bonus** of 35 points
* For each of these categories the **total** is the sum of the die that show that number. For example, if you have 4 die showing the number 3, you get a score of 12 in the 3’s category

LOWER SCORE:

* Player can score either a set amount (defined by the category)
* Player can score 0 if category requirements aren’t met

# Software Requirements

The application for Yahtzee will keep track of 2 - 4 players’ category scores, upper scores, and lower scores through all 13 rounds of game play, displaying the scores at the beginning of each player’s turn; at the end of 13 rounds, the application will display all the players’ total scores and a message declaring the winner of the game (i.e., the player who has the highest score).

**APPLICATION USERS:**

* 2-4 players

**APPLICATION FEATURES:**

* At the end of the game, all the players total scores should be displayed along with a message declaring the winner.
* At the beginning of each player’s turn: The application must show the player’s current score
* At the end of each game:
  + all the players total scores must be displayed
  + a message declaring the name of the winner

**HOW USER INTERACTS WITH SOFTWARE/APPLICATION**

* Overall instructions in the header for game will be given once at beginning of game
* Players will be prompted to enter number of players (2 players minimum and 4 players maximum)
* Players will be prompted to enter the players’ names
* Players cannot continue until they have entered their player names
* Displays option that allows the players to either play another game or quit at the end of each game
* If the player opts to quit after the 13 rounds, then the application will display a message saying

“Good-Bye!” or the option to play another game

* Each player will use the same 5 dice during a turn
* During a round, each player will take a turn consisting of the following steps:
  + The player must roll the dice a at least once and no more than 3 times
    - After each roll, the player will decide to score the current roll or re-roll the dice
    - On the first turn the player/user must roll all 5 dice
    - On the second and third rolls, the player/user may re-roll any or all of the 5 dice
    - To re-roll the dice, the player will select the dice they wish to re-roll
  + The player must complete the turn by scoring the current roll in one of the 13 categories
    - The player will select the category they wish to score
    - The player must score each of the 13 categories only once
    - The application will validate that the user’s/player’s category selection is valid
    - If the player’s current roll does not match any of the category requirements for a score the player must select an unfilled category to score as 0
* The application will provide a representation of the current score sheet to the player at their turn so they can decide what they want to try to roll, which category they want to score, and what their final score is.
* Their score sheet will be displayed during the entirety of their turn. At each player’s turn, their own individual score sheet will be displayed in the game.
* The application will display a current scorecard visible to the player at all times.
  + During the entirety of each player’s turn the application will display the following information for the player: player name, roll number, dice values, score sheet, current input prompt
  + After the player scores the roll to complete a turn, the score sheet will switch to the next player’s score sheet
  + At the end of each player’s turn, the displayed player’s information will switch to the next player’s information

**HOW PROGRAM IS STARTED**

csc$ java Yahtzee

<Display game header>

**ERROR HANDLING REQUIREMENTS**

* Player can only input an integer value between 2 and 4 players. If the user enters an invalid number of players, a message will be displayed saying, “Invalid number of players”.

csc$ java Yahtzee

<Display game header>

Number of players (2-4): **1**

**Invalid number of players**

Number of players (2-4): **2**

Player 1 Name:

* Continue prompting until player enters name

csc$ java Yahtzee

<Display game header>

Number of players (2-4): **1**

**Invalid number of players**

Number of players (2-4): **2**

Player 1 Name: **Sari the Programming Queen**

Player 2 Name: **Jessica <3s Coding**

**< Start Player 1 turn >**

* After the first roll, the program should prompt the user to “score” the the current roll or “roll” again. If the user input is not “roll” or “score”, disregarding case, the program should display an error message and reprompt the player to score or roll. If the player chooses to roll again, the program should prompt the user for the values of the dice to re-roll. Players cannot roll dice greater than 3 times per turn.   
    
  If the player elects to score roll 1 or roll 2, or if the player rolls 3 times, the program should prompt the user to score a category and display a menu of category choices (ones, twos, threes, fours, fives, sixes, three of a kind, four of a kind, small straight, large straight, full house, Yahtzee, chance). During scoring, the program should continuously prompt the user to select a category to score from the menu.

The program shall accept and use both upper and lower case letters for the options.

csc$ java Yahtzee

<Display game header>

Number of players (2-4): **1**

**Invalid number of players**

Number of players (2-4): **2**

Player 1 Name: **Sari the Programming Queen**

Player 2 Name: **Jessica <3s Coding**

< Display Player 1 scorecard >

Sari the Programming Queen’s current dice roll: [6] [6] [5] [1] [3]

Do you want to score the current roll or re-roll (Options: score, roll)? **jibberish**

Invalid option

Do you want to score the current roll or re-roll (Options: score, roll)? **123**

Invalid option

Do you want to score the current roll or re-roll (Options: score, roll)? **ROLL**

Enter the value(s) of the dice to re-roll: **1 3 5**

Sari the Programming Queen’s current dice roll: [6] [6] [1] [6] [6]

Do you want to score the current roll or re-roll (Options: score, roll)? **four of a kind**

Invalid option

Do you want to score the current roll or re-roll (Options: score, roll)? **Score**

Enter a category (ones, twos, threes, fours, fives, sixes, three of a kind, four of a kind, small straight, large straight, full house, Yahtzee, chance) to score: **four of a kind**

< Display next player scorecard >

If the user input is an invalid category option, the program shall output an error message and reprompt the player to choose a category from the menu.

csc$ java Yahtzee

<Display game header>

Number of players (2-4): **1**

**Invalid number of players**

Number of players (2-4): **2**

Player 1 Name: **Sari the Programming Queen**

Player 2 Name: **Jessica <3s Coding**

< Display Player 1 scorecard >

Sari the Programming Queen’s current dice roll: [6] [6] [5] [1] [3]

Do you want to score the current roll or re-roll (Options: score, roll)? **jibberish**

Invalid option

Do you want to score the current roll or re-roll (Options: score, roll)? **123**

Invalid option

Do you want to score the current roll or re-roll (Options: score, roll)? **ROLL**

Enter the value(s) of the dice to re-roll: **1 3 5**

Sari the Programming Queen’s current dice roll: [6] [6] [1] [6] [6]

Do you want to score the current roll or re-roll (Options: score, roll)? **rOlL**

Enter the value(s) of the dice to re-roll: **1**

Sari the Programming Queen’s current dice roll: [6] [6] [6] [6] [6]

Enter a category (ones, twos, threes, fours, fives, sixes, three of a kind, four of a kind, small straight, large straight, full house, Yahtzee, chance) to score: **ROLL**

Invalid category.

Enter a category (ones, twos, threes, fours, fives, sixes, three of a kind, four of a kind, small straight, large straight, full house, Yahtzee, chance) to score: **OMG** **Yahtzee!!!!**

Invalid category.

Enter a category (ones, twos, threes, fours, fives, sixes, three of a kind, four of a kind, small straight, large straight, full house, Yahtzee, chance) to score: **Yahtzee**

< Display next player scorecard >

Players can only enter 1 score per category. If the category is valid but has already been scored, display a message stating “This category has already been scored. Choose a different category.” Reprompt the player to choose a different category.

<Player 2 takes turn to complete Round 1>

< Displays Sari the Programming Queen’s scorecard with Yahtzee category scored >

Sari the Programming Queen’s current dice roll: [5] [5] [5] [5] [5]

Do you want to score the current roll or re-roll (Options: score, roll)? **SCORE**

Enter a category (ones, twos, threes, fours, fives, sixes, three of a kind, four of a kind, small straight, large straight, full house, Yahtzee, chance) to score: **Yahtzee**

This category has already been scored. Choose a different category: **four of a kind**

< Display next player scorecard >

# Software Design

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

<<Complete this section of the formal documentation by planning the classes, methods, and fields that will used in the software. Put the details of the userInterface design choice in this section.

Provide information about each class used in your project and how the classes are related.

* For each class that is **NOT** used to create objects (SEE Project 5 for an example), list:
  + STATIC METHOD HEADERS with javadoc to describe function, inputs, outputs
* For each class that **IS** used to create objects (SEE Project 6 for an example), list:
  + INSTANCE VARIABLES
  + CONSTRUCTOR()
  + INSTANCE METHOD HEADERS with javadoc to describe function, inputs, outputs

*>>*

**The Dice Class**

The Dice class creates an integer array object representing a set of dice and interacts with the Yahtzee class to simulate the dice in the game.

<<Java Class>>

Dice

-dice: int[]

-diceCount: int

-sidesPerDie: int

+Dice (int, int)

+getDiceCount():int

+getDice():int[ ]

+getSidesPerDie():int

+toString():String

+rollAllDice():void

+rollOneDie(int):Boolean

+tallyDice():int[ ]

+findHighestFrequency():int

+sumAllDice():int

Methods

/\*\*

\* Constructs an object representation of dice by initializing an integer array with a

\* length equal to the number of dice, given as a parameter. Each integer in the array

\* represents the value of a die. Sets the number of sides per die to the given value,

\* and sets the initial value of each die to 1.

\* @param numberOfSides number (integer) of sides on each die

\* @param numberOfDice number (integer) of dice in the set

\*/

public Dice (int numberOfSides, int numberOfDice) {

}

/\*\*

\* Returns the number of dice in the set

\* @return number of dice in the set

\*/

public int getDiceCount() {

}

/\*\*

\* Returns the integer array representation of the dice set

\* @return integer array representation of the dice set

\*/

public int[] getDice() {

}

/\*\*

\* Returns the number of sides per die

\* @return number of sides on each die in the set

\*/

public int getSidesPerDie() {

}

/\*\*

\* Returns a String representation of the dice array (e.g., "[1] [2] [3] [4] [5]")

\* @return String representation of the dice array

\*/

public String toString() {

}

/\*\*

\* Simulates the roll of all dice. Assigns a random number between 1 and the number

\* of sides per die in each index of the dice array.

\*/

public void rollAllDice() {

}

/\*\*

\* Simulates the roll of one die in a set of dice by replacing the first occurrence of

\* the given value with a random number between 1 and the number of sides per die.

\* @param dieValue value of the die to roll

\* @return true if a die if re-rolled, false if no dice in the set contain the given value

\*/

public Boolean rollOneDie(int dieValue) {

}

/\*\*

\* Returns an array representing a tally of the number of dice displaying each possible

\* die value.

\* The value is the array index. The tally is an element stored at the value index.

\* This method is based on slide 13 of CSC 116 lecture on Advanced Arrays.

\* @return array of tally data

\*/

public int[] tallyDice() {

}

/\*\*

\* Returns the index of the first occurrence of the maximum value in an integer array.

\* After a simulated roll, this method returns the most frequently occurring

\* die value) in the array and breaks ties by choosing the lower value.

\* This method is based on the findMode method from the CSC 116 CalculateMode assignment.

\* @param tallyArray array containing a tally of values (index = value, element =

\* frequency of the value)

\* @return index of the first occurrence of the maximum value in an integer array

\*/

public int findHighestFrequency() {

}

/\*\*

\* Returns the sum of all values in the dice array

\* @return sum of all dice values

\*/

public int sumAllDice() {

}

**The Scorecard Class**

The Scorecard class interacts with the Yahtzee and Dice classes to score a players rolls and encapsulates all scorecard information for a player.

+DICE\_COUNT: int

+ONE: int

+TWO: int

+THREE: int

+FOUR: int

+FIVE: int

+SIX: int

+FULL\_HOUSE: int

+SMALL\_STRAIGHT: int

+LARGE\_STRAIGHT: int

+YAHZTEE: int

-name: String

-rolls: int

-totalScore:int

-upperScore:int

-lowerScore:int

-upperBonus:int

-ones:int

-twos:int

-threes:int

-fours:int

-fives:int

-sixes:int

-threeOfAKind: int

-fourOfAKind: int

-smallStraight: int

-largeStraight: int

-fullHouse: int

-yahtzee: int

-chance: int

+Scorecard(String playerName)

+getName():String

+getTotalScore(): int

+getUpperScore(): int

+getLowerSocre(): int

+getUpperScoreBonus(): int

+getNumberOfRolls(): int

+setNumberOfRolls(int): void

+setScore(String, Dice): String

+toString(): String

Methods

/\*\*

\* Constructs an object representation of a scorecard for a player.

\* Initializes unscored category fields to -1

\* Initializes score totals to 0.

\*/

public Scorecard(String playerName) {

}

/\*\*

\* Returns a string representation of the name of the player who owns the scorecard

\* @return name of the player

\*/

public String getName() {

}

/\*\*

\* Calculates and returns the player's total score

\* @return player's total score

\*/

public int getTotalScore() {

}

/\*\*

\* Calculates and returns the player's upper score

\* @return player's upper category score

\*/

public int getUpperScore() {

}

/\*\*

\* Calculates and returns the player's lower score

\*/

public int getLowerScore() {

}

/\*\*

\* Calculates and returns the upper score bonus. At the end of the game, if the upper

\* score is greater than or equal to 63, the upper score bonus is 35 points.

\* If the upper score is less than 63, the upper score bonus is 0 points.

\* @return the upper bonus score

\*/

public int getUpperScoreBonus() {

}

/\*\*

\* Returns the number of times a player has rolled the dice during the player's turn

\* @return number of times the player has rolled the dice during the player's turn

\*/

public int getNumberOfRolls() {

}

/\*\*

\* Sets the number of rolls in the rolls private field.

\*

\* @param numberOfRolls passes into the method the number of player's rolls

\*/

public void setNumberOfRolls(int numberOfRolls) {

}

/\*\*

\* Scores a category specified as a parameter

\* @param category String representation of the category to be scored

\* @param dice Dice object

\* @return String stating which category was scored, null if category is invalid or is already scored

\*/

public String setScore(String category, Dice dice) {

}

ADD toString JAVADOC

\* return scorecardString

public String toString() {

}

**The Yahtzee Class**

+SIDES: int

+NUMBER\_OF\_DICE : int

+CATEGORIES: String[ ]

-playerArr: String[ ]

-playerScoreCardArr: Scorecard[ ]

-dice: Dice

-**userInterface(): String**

-s**tartTurn(String): void**

**- rollAgain(String, Scanner): void**

**- scoreRoll(Scorecard player, String category): String**

**Add additional methods here. Describe if necessary.**

# Implementation

<< What programming concepts from the course will you need to implement your design? Briefly explain how each will be used during implementation. Make sure to explain any java class methods you may have used that we did not cover in class (for example: the Scanner class useDelimiter() method), any exceptions thrown, specific output formats, etc.

See the implementation section in our projects to get idea on what to add to this section).

>>

**Programming Concepts**

* There are three interacting classes — Dice, Scorecard, and Yahtzee — used to construct/roll dice for the game, score rolls and keep track of player scores, and interface with the user, respectively.
* The Dice class creates an encapsulated object representation of a set of dice.
  + The Arrays class is used to create the representation of the set of dice and to create an array representation of the tally for each possible die value.
  + The Random class is used to simulate rolling dice.
* The Scorecard class creates an encapsulated object representation of a Yahtzee scorecard.
  + Nested if/then statements containing both mathematical and String comparisons and for loops are used in the scoring logic.
  + Array methods (e.g. toString) are used in the scoring logic.
  + printf is used to output the scorecard in a neat, readable, tabulated format
* The Yahtzee class
  + A console Scanner is used to interact with the users.
  + For loops are used to control the logic for player turns and rounds of game play.
  + If/then logic is used to validate user input
  + Sari, add any other notes here.

**Dice.java Implementation**

<< What programming concepts from the course will you need to implement your design? Briefly explain how each will be used during implementation. Make sure to explain any java class methods you may have used that we did not cover in class (for example: the Scanner class useDelimiter() method), any exceptions thrown, specific output formats, etc.

See the implementation section in our projects to get idea on what to add to this section).

>>

Instance Fields

Declare the following private instance fields:

* An array of integers that represents of a set of dice. Each element in the array is a single die.
* An integer that represents the number of dice in the set. This is the length of the dice array.
* An integer that represents the number of sides per die. This is the maximum value of each integer in the array.

Methods

The following is the list of methods to complete.

* public Dice (int numberOfSides, int numberOfDice)
* public int[] getDice()
* public int getDiceCount()
* public int getSidesPerDie()
* public String toString()
* public void rollAllDice()
* public Boolean rollOneDie(int dieValue)
* public int[] tallyDice()
* public int findHighestFrequency()
* public int sumAllDice()

**Scorecard.java Implementation**

<< What programming concepts from the course will you need to implement your design? Briefly explain how each will be used during implementation. Make sure to explain any java class methods you may have used that we did not cover in class (for example: the Scanner class useDelimiter() method), any exceptions thrown, specific output formats, etc.

See the implementation section in our projects to get idea on what to add to this section).

>>

Instance Variables

Class Constants

Include and use the following class constants in the Scorecard class:

/\*\* Number of dice in game \*/

public static final int DICE\_COUNT = 5;

/\*\* Number of points to add for each die with a 1 face value when scoring Ones \*/

public static final int ONE = 1;

/\*\* Number of points to add for each die with a 2 face value when scoring Twos \*/

public static final int TWO = 2;

/\*\* Number of points to add for each die with a 3 face value when scoring Threes \*/

public static final int THREE = 3;

/\*\* Number of points to add for each die with a 4 face value when scoring Fours \*/

public static final int FOUR = 4;

/\*\* Number of points to add for each die with a 5 on the face when scoring Fives \*/

public static final int FIVE = 5;

/\*\* Number of points to add for each die with a 6 on the face when scoring Sixes \*/

public static final int SIX = 6;

/\*\* Number of points to add for each die with a 6 on the face when scoring Sixes \*/

public static final int FULL\_HOUSE = 25;

/\*\* Number of points scored for a small straight \*/

public static final int SMALL\_STRAIGHT = 30;

/\*\* Number of points scored for a large straight \*/

public static final int LARGE\_STRAIGHT = 40;

/\*\* Number of points scored for a Yahtzee \*/

public static final int YAHZTEE = 50;

Instance Fields

Declare the following private instance fields

* A string that stores the name of player
* An integer that stores the number of times a player has rolled in each turn
* An integer that stores the player’s total score.
* An integer that stores the player’s lower score.
* An integer that stores the player’s upper bonus score.
* An integer that stoers the player’s ones category score.
* An integer that stores the player’s twos category score.
* An integer that stores the player’s threes category score.
* An integer that stores the player’s fours category score.
* An integer that stores the player’s fives category score.
* An integer that stores the player’s sixes category score.
* An integer that stores the player’s three of a kind category score.
* An integer that stores the player’s four of a kind category score.
* An integer that stores the player’s small straight category score.
* An integer that stores the player’s large straight category score.
* An integer that stores the player’s full house category score.
* An integer that stores the player’s yahtzee category score.
* An integer that stores the player’s chance category score.

Methods

The following is the list of methods to complete.

* public Scorecard(String playerName)
* public String getName()
* public int getTotalScore()
* public int getUpperScore()
* public int getLowerScore()
* public int getUpperScoreBonus()
* public int getNumberOfRolls()
* public void setNumberOfRolls(int numberOfRolls)
* public String setScore(String category, Dice dice)
* public String toString()

**Yahtzee.java Implementation**

<< What programming concepts from the course will you need to implement your design? Briefly explain how each will be used during implementation. Make sure to explain any java class methods you may have used that we did not cover in class (for example: the Scanner class useDelimiter() method), any exceptions thrown, specific output formats, etc.

See the implementation section in our projects to get idea on what to add to this section).

>>

**Methods**

* **public static String userInterface() – This method interacts with the user using a console Scanner, and the method controls the logic of the game by completing the following tasks:** 
  + **Prints a head with game instructions.**
  + **Prompts user for the number of players and player names. Players cannot continue until they have all entered names. Calls Scorecard to construct a scorecard for each player.**
  + **Initiates 13 rounds of game play. During each round, each player takes a turn.** 
    - **Starts each player's turn by calling startTurn(String player)**
    - **Prompts player to score the a roll or roll again. The player may roll any or all dice two additional times after the starting their turn with an initial roll of all dice.**
    - **If the player chooses to roll again, calls the rollAgain(String player, Scanner console) method**
    - **If the user chooses to score a roll, the program should display a menu of category choices and prompt for a category to score. If the user enters an invalid category, the program should display "Invalid Category" and prompt continuously until a valid category is entered.**
    - **If the program is unable to score the roll in the category selected by the user, print a message that says "This category has already been scored. Choose a different category." Then, continuously prompt until a scoreable category is selected.**
    - **After a player successfully scores a roll, move on to the next player's turn.**
  + **After 13 rounds of game play, displays the total score for each player and a message declaring the winner. Prompts the user to quit or play again.**
* **public static void startTurn(String player) - This method starts a player's turn by displaying the player's scorecard and rolling all of the dice for the first roll. Increments the roll counter and displays a String representation of the dice roll.**
* **public static void rollAgain(String player, Scanner console) – This method uses the console Scanner to read in a String containing the values of the dice that the given player would like to roll again. If the player enters an invalid value (value < 1, value > number of sides on the dice, or value != any of the value of any dice in the currently roll), print an error message and continuously prompt until the user gives a valid value. If the dice values are valid, the method passes each value to the Dice rollOneDie() method as a parameter. Then increments the roll counter in the scorecard and prints a String representation of the dice roll using the Dice toString() method.**
* **public static String scoreRoll(Scorecard player, String category) - Attempt to score the category using the Scorecard methods and returns the result of the score attempt**
* **Add additional methods here. Describe if necessary.**

# Testing

<<Complete this section of the formal documentation by creating a list of your test programs along with descriptions of what they are testing.

* BlackBox Testing

The Black Box testing of the <<PROGRAM NAME>> program was performed according to the

BlackBoxTestPlan.docx document. List any test files required to run these tests. Discuss any special testing scenario requirements (for example, seeds needed for random number generators, command line arguments to run in test mode, etc).

* WhiteBox Testing (see project 6 for documentation example)

The following test program(s) (one for each class) and test file(s) were used to unit test the <<PROGRAM NAME>> program.

The following methods were tested (list for each class). Can include equivalence classes and boundary values tested for each method.

>>