```
import java.util.Map;
import java.util.Scanner;
import java.util.InputMismatchException;

/**
 * Yahtzee Game
 *
 * @Nghi Phan
 * @1.0.0
 */
public class Yahtzee
{
    Scanner scanner = new Scanner(System.in);
    Die6 die1 = new Die6();
    Die6 die2 = new Die6();
    Die6 die3 = new Die6();
    Die6 die4 = new Die6();
    Die6 die5 = new Die6();
    private int[] scoreUpper = new int[6];
    private int upperTotal;
    private int[] scoreOfAKind = new int[3];
    private int fullHouse;
    private int NUM_DIE = 6;
    private final int[] straight = new int[2];
    private int lowerTotal;
    private int grandTotal;
    private int Chance = 0;
    private String input;
    private static final int NUM_DICE = 5;
    private static final int NUM_ROUNDS = 13;
    private int round;
    private int score;
    //constructor & rolls all dice for turn one
    public Yahtzee() {
        round = 1;
        score = 0;
        rollAll();
    }

    public static void main(String[] args) {
        Yahtzee yahtzee = new Yahtzee();
        yahtzee.play();
    }

    public void play() {
        int turn = 0;
        int scoreRound = 0;
        boolean rolled = false;
        boolean scoreMarked = false;;
        while(round <= NUM_ROUNDS) {
            System.out.print("\n[ "+toString()+" ]");
            System.out.println("\n{ - 0 - Quit | 4 - Next Turn }");
            System.out.println("{ -- roll - Rolls Specified Dice | rollAll - Rolls All Dice }");
            System.out.println("{ --- 1 - Mark Uppersection | 2 - Mark Lowersection | 3 - Getscores }\n");
            input = scanner.nextLine();
```

```
switch(input) {
    case "rollAll":
        if(!rolled) {
            rollAll();
            turn++;
            System.out.println("Turn:"+turn);
        } else {
            System.out.println("No More Rolls This Round");
        }
        break;

    case "roll":
        if(!rolled) {
            System.out.println("Enter The Die You Wish To Roll Separated By a Space");
            String stringToSplit = scanner.nextLine();
            String[] split = stringToSplit.split(" ");
            int[] array = new int[split.length];
            for(int i = 0; i < array.length; i++) {
                array[i] = Integer.parseInt(split[i]);
            }
            roll(array);
            turn++;
            System.out.println("Turn:"+turn);
        } else {
            System.out.println("Out of Turns");
        }
        break;

        //Done
    case "1":
        if (scoreMarked) {
            System.out.println("(-Score already marked in this round.)");
        } else {
            System.out.print("Select a category (1-6): ");
            int categoryUpper = scanner.nextInt();
            scanner.nextLine();
            System.out.println(scoreUpper(categoryUpper));
            scoreMarked = true;
            scoreRound += scoreUpper(categoryUpper);
        }
        break;

    case "2":
        if (scoreMarked) {
            System.out.println("(-Score already marked in this round.)");
        } else {
            System.out.println("1 - Score Of A Kind");
            System.out.println("2 - Score FullHouse");
            System.out.println("3 - Score Straights");
            System.out.println("4 - Score Chance");
            int cateogoryLower = scanner.nextInt();
            scanner.nextLine();
            switch (cateogoryLower) {
                case 1:
```

```
                System.out.print("Please enter 3 to 5: ");
                int markScoreKind = scanner.nextInt();
                scanner.nextLine();
                switch(markScoreKind) {
                    case 3:
                        System.out.println("3 of a kind: "+scoreOfAKind(3));
                        scoreRound += scoreOfAKind(3);
                        break;
                    case 4:
                        System.out.println("4 of a Kind: "+scoreOfAKind(4));
                        scoreRound += scoreOfAKind(4);
                        break;
                    case 5:
                        System.out.println("Yahtzee: "+scoreOfAKind(5));
                        scoreRound += scoreOfAKind(5);
                        break;
                }
                break;
            case 2:
                System.out.println("Full House: "+fullHouse());
                scoreRound += fullHouse();
                break;
            case 3:
                System.out.print("Please enter 4 or 5: ");
                int markScoreStraight = scanner.nextInt();
                scanner.nextLine();
                switch(markScoreStraight) {
                    case 4:
                        System.out.println("4 Straight: "+straight(4));
                        scoreRound += straight(4);
                        break;
                    case 5:
                        System.out.println("5 Straight: "+straight(5));
                        scoreRound += straight(5);
                        break;
                }
                break;
            case 4:
                System.out.println("Chance: "+Chance());
                scoreRound += Chance();
                break;
        }

        scoreMarked = true;
    }
    break;
    //Done
case "3":
    System.out.println("1 - Upper Scores");
    System.out.println("2 - Lower Scores");
    System.out.println("3 - Totals");
    int type = scanner.nextInt();
    scanner.nextLine();
    switch(type) {
        case 1:
```

```
                    System.out.println(getScoreUpper());
                    break;
                case 2:
                    System.out.println(getScoreOfAKind());
                    System.out.println(getFullHouse());
                    System.out.println(getStraight());
                    System.out.println("Chance: "+this.Chance);
                    break;
                case 3:
                    System.out.println(getUpperTotal());
                    System.out.println("Lower Total: "+getLowerTotal());
                    System.out.println("Grand Total: "+getGrandTotal());
                    break;
            }
            break;
            // Done
        case "4":
            if(!rolled) {
                System.out.println("You are required to roll");
            } else {
                round++;
                System.out.println("Round: "+round);
                scoreMarked = false;
                rolled = false;
                turn  = 0;
            }
            break;
    }

    if(scoreRound == 0) {
        scoreMarked = false;
    }

    if(turn >= 3) {
        rolled = true;
    }

    if(input.equals("0")){
        break;
    }
  }
}

//rolls all dice
public void rollAll() {
    Die6[] dice = new Die6[]{die1, die2, die3, die4, die5};
    //iterates over object array and calls roll method
    for (Die6 die : dice) {
        die.roll();
    }
}

//Input an array, if value matches switch cases, rolls that die
public void roll(int[] diceToRoll) {
    for (int i : diceToRoll) {
```

```
            switch (i) {
                case 1:
                    die1.roll();
                    break;
                case 2:
                    die2.roll();
                    break;
                case 3:
                    die3.roll();
                    break;
                case 4:
                    die4.roll();
                    break;
                case 5:
                    die5.roll();
                    break;
            }
        }
    }

    //Returns the number of occurances of a die number
    public String summerize() {
        int[] two = new int[]{ val (1), val (2), val (3), val (4), val (5), val (6) };

        return("1-"+two[0]+"; 2-"+two[1]+"; 3-"+two[2]+"; 4-"+two[3]+"; 5-"+two[4]+"; 6-"+two[5]+";");

    }

    //returns the value of all dice
    public String toString() {
        return("Dice Values: " + die1.value + " " + die2.value + " " + die3.value + " " + die4.value + " " + die5.
value);
    }

    /** Calculates the occurances of a die number
     * Can only be called by summerize method
     */
    private int val(int val) {
        int count = 0;
        int[] dice = new int[]{die1.value,die2.value,die3.value,die4.value,die5.value};
        //iterates over dice array matches with int val and calculates count
        for(int i : dice) {
            if(i == val) {
                count++;
            }
        }
        return count;
    }

    /** stores scores for upper section into scoreUpper[]
     * checks for already indexes of scoreUpper[]
     * checks for ArrayIndexOutOfBoundsException for 1 > score > 6
     */
    public int scoreUpper(int score) {
        int scoreNum = 0;
```

```
        int[] dice = new int[]{die1.value,die2.value,die3.value,die4.value,die5.value};
        try{
            if(score > 6 || score <= 0) {
                throw new ArrayIndexOutOfBoundsException("Please enter 1-6");
            }
            //if there's a stored value in the array stops the calculations
            if (scoreUpper[score-1] != 0) {
                return scoreUpper[score-1];
            }
            //iterates over dice, calculates Uppersection and stores values in int scoreNum, and then stores it i
n the instances array scoreUpper[]
            for(int i: dice) {
                if(i == score) {
                    scoreNum += score;
                    scoreUpper[score-1] = scoreNum;
                }
            }
            //catches ArrayIndexOutOfBoundsException for 1 > score > 6
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: " + e.getMessage());
        }
        return scoreNum;
    }

    /** calculates 3,4, & 5 (yahtzee) of a kind
     * Stores scores into scoreOfAKind[]
     * checks for already initalized scoreOfAKind indexes
     * gives bonus yahtzee score ONCE
     * checks for ArrayIndexOutOfBoundsException for 3 > type > 5
     */
    public int scoreOfAKind(int type) {
        int[] counts = new int[6];
        int score = 0;
        boolean yahtzeeBonus = false;
        int[] dice = new int[]{die1.value,die2.value,die3.value,die4.value,die5.value};
        try{
            if(type > 5 || type < 3) {
                throw new ArrayIndexOutOfBoundsException("Please enter 3-5");
            }
            //checks for already initalized scoreOfAKind[] indexes
            if (scoreOfAKind[type-3] != 0) {
                //checks in cases of a second Yahtzee
                if(type == 5) {
                    if(!yahtzeeBonus) {
                        yahtzeeBonus = true;
                        scoreOfAKind[type-3] += 100;
                        return scoreOfAKind[type-3];
                    }
                }
                //returns already stored value
                return scoreOfAKind[type-3];
            }
            //counts occurances of a die number and stores it in counts[]
            for (int i : dice) {
                counts[i - 1]++;
```

```
        }
        //iterates over counts[]
        for (int i : counts) {
            //if counts matches _ofAKind adds up all dice values and stores it into scores
            if(i == 5 && type == 5) {
                scoreOfAKind[type-3] = 50;
                return scoreOfAKind[type-3];
            } else if (i >= type) {
                for (int j : dice) {
                    score += j;
                }
                //interalizes scoreOfAKind[] with values from score, and then returns the values
                scoreOfAKind[type-3] = score;
                return scoreOfAKind[type-3];
            }
        }
        //catches ArrayIndexOutOfBoundsException for 3 > type > 5
    } catch (ArrayIndexOutOfBoundsException g) {
        System.out.println("Error: " + g.getMessage());
    }
    //if there's no 3+ matching die numbers returns 0
    return 0;
}

/**calulates fullHouse in LowerSection
 * fullHouse requires 3 of the same die number and a pair
 */
public int fullHouse() {
    int[] counts = new int[6];
    int[] dice = new int[]{die1.value,die2.value,die3.value,die4.value,die5.value};
    //counts occurances of a die number and stores it in counts[]
    for (int i : dice) {
        counts[i - 1]++;
    }
    //interates over counts[] twice
    for(int j: counts) {
        for(int a: counts) {
            //initalizes fullHouse if conditions are met
            if(j == 3 && a == 2) {
                fullHouse = 25;
                return fullHouse;
            }
        }
    }
    //if conditions are not met return 0
    return 0;
}

/** Calculates for both large and small straights
 * stores values in Instance variable straight[]
 * checks for ArrayIndexOutOfBoundsException 4 > set > 5
 */
public int straight(int set) {
    int[] dice = new int[]{die1.value,die2.value,die3.value,die4.value,die5.value};
    int[] count = new int[6];
```

```
        int NUM = 0;
        try{
            if(set > 5 || set < 4) {
                throw new ArrayIndexOutOfBoundsException("Please enter 4-5");
            }
            //if there's a stored value in the array stops the calculations
            if (straight[set-4] != 0) {
                return straight[set-4];
            }
            //iterates over dice[]
            for(int i : dice){
                //checks for occurances of a die number that's more than 0
                if (count[i-1] == 0) {
                    //initalizes count to die number, ensures there aren't any duplicate die Numbers
                    count[i-1] += i;
                }
            }
            //iterates over count twice
            for(int a: count) {
                if(a != 0) {
                    for(int b: count) {
                        //counts number of sequences, at least 3 is needed, 4 is max
                        if(a+1 == b) {
                            NUM++;
                        }

                    }
                }
            }
            //checks number of sequences
            if(NUM >= set-1) {
                //initalizes straight[] to repective set value
                if(set == 4) {
                    straight[0] = 30;
                    return straight[0];
                } else {
                    straight[1] = 40;
                    return straight[1];
                }
            }
            //catches ArrayIndexOutOfBoundsException for 4 > set > 5
        } catch (ArrayIndexOutOfBoundsException k) {
            System.out.println("Error: " + k.getMessage());
        }
        //if conditions are not met returns 0
        return 0;
    }

    //gets values from scoreOfAKind[] array
    public String getScoreOfAKind() {
        return("3 of a kind: " + scoreOfAKind[0] + "\n4 of a kind: " + scoreOfAKind[1] + "\nYahtzee: " + score
OfAKind[2]);
    }

    //gets values from fullHouse
```

```
    public String getFullHouse() {
        return("FullHouse: " + fullHouse);
    }

    //gets values from straight[] array
    public String getStraight() {
        return("Small Straight: " + straight[0] + "\nLarge Straight: " + straight[1]);
    }

    //Calculates and gets Chance
    public int Chance() {
        int[] dice = new int[]{die1.value,die2.value,die3.value,die4.value,die5.value};
        if(Chance != 0) {
            return Chance;
        }
        for(int i: dice) {
            Chance += i;
        }
        return Chance;
    }

    /** get all upper section scores for scoreUpper[]
     */
    public String getScoreUpper() {
        return ("Score 1: " + scoreUpper[0]+"\nScore 2: " + scoreUpper[1]+"\nScore 3: " + scoreUpper[2]+
            "\nScore 4: " + scoreUpper[3]+"\nScore 5: " + scoreUpper[4]+"\nScore 6: " + scoreUpper[5]);
    }

    /**
     * adds bonus 35 for an Upper score over 63
     * stores total into upperTotal
     * returns all Uppper scores and the total
     */
    public String getUpperTotal() {
        int UpperTotal = scoreUpper[0] + scoreUpper[1]+ scoreUpper[2]+
            scoreUpper[3]+ scoreUpper[4]+ scoreUpper[5];
        //initalizes instances variable
        this.upperTotal = UpperTotal;
        if(upperTotal > 63) {
            upperTotal += 35;
        }

        return ("Upper Total: "+upperTotal);
    }
    //calculates and gets LowerTotal
    public int getLowerTotal(){
        int LowerTotal = scoreOfAKind[0] + scoreOfAKind[1] + scoreOfAKind[2] + fullHouse + straight[0] + st
raight[1] + Chance;
        this.lowerTotal = LowerTotal;
        return lowerTotal;
    }
    //calculates and gets Grand Total
    public int getGrandTotal() {
        grandTotal = upperTotal + lowerTotal;
        return grandTotal;
```

```
    }

}
```