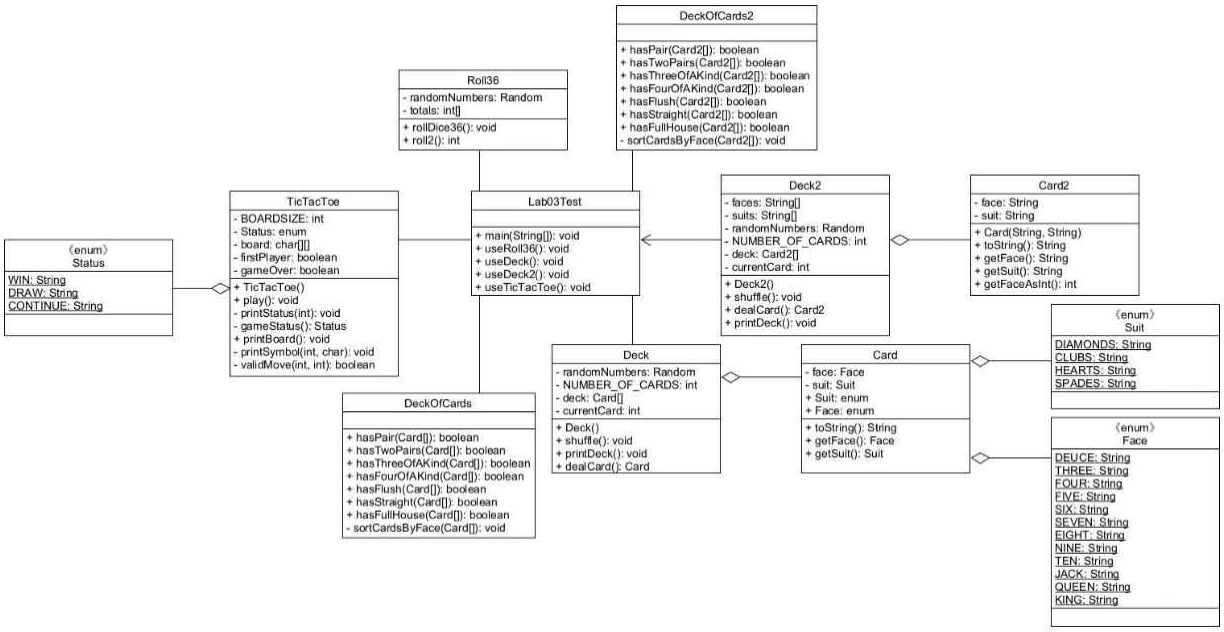
Java Programming (CSE220)

Lab 03

ID: 2020136129 Name: 최수연

Class diagram



Task-1 (Rolling Dice)

Write class Roll36 to simulate the rolling of two dice. The class should use an object of class Random once to roll the first die and again to roll the second die. The sum of the two values should then be calculated. Each die can show an integer value from 1 to 6, so the sum of the values will vary from 2 to 12, with 7 being the most frequent sum, and 2 and 12 the least frequent. Your application should roll the dice 36,000,000 times. Use a one-dimensional array totally the number of times each possible sum appears. Display the results in tabular format.

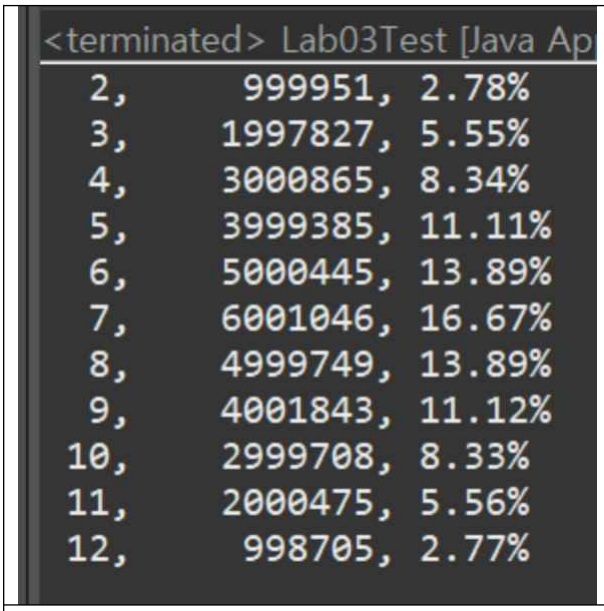
Code with Explanation

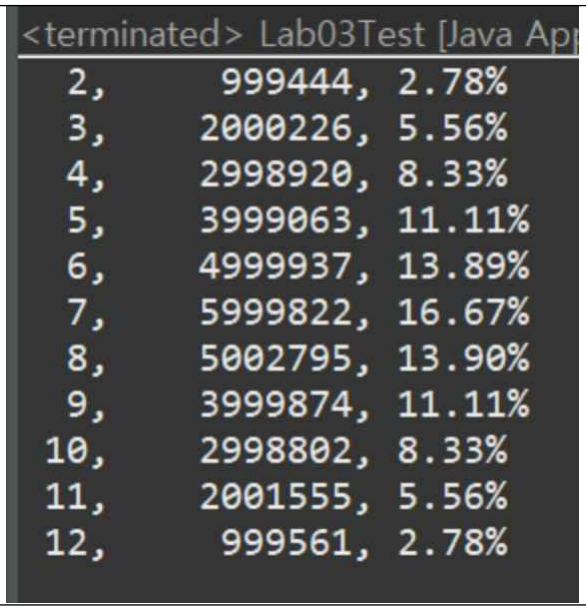
Roll36 class

```
package Lab03;

import java.util.Random;

public class Roll36 {
	// Create an instance of Random Class
	private Random randomNumbers = new Random();
	// Create an array to store the sum of the two values from the two dies
	private int[] totals = new int[13];
```

```
	public void rollDice36() {
		// Initialize values for all arrays to zero
		for (int i = 0; i < totals.length; i++)
			totals[i] = 0;
		// Create an array to store the sum of the two values from the two
dies
		for (int roll = 1; roll < 36000000; roll++) {
			totals[roll2()]++;
		}
		for (int k = 2; k < totals.length; k++) {
			System.out.printf("%3d, %10d, %.2f%% \n", k, totals[k],
((double)totals[k] / 36000000) * 100);
		}
	}

	public int roll2() {
		// Output random dies' value
		int die1 = 1 + randomNumbers.nextInt(6); // The first die
		int die2 = 1 + randomNumbers.nextInt(6); // The second die

		int sum = die1 + die2; // The sum of the two values

		// Show die1, die2, sum of the dies
		//System.out.printf("Player rolled %d + %d = %d%n", die1, die2, sum);

		return sum;
	}
}
```

Results/Output





주사위의 합 / 36000000 굴려서 나온 각 주사위 번호 합의 횟수 / 나오는 비율

The sum of the dice / 3600000000 The number of times each die number is summed up / the percentage that comes out

Task-2(Card Shuffling and Dealing)

1) Develop a class Card, which represents a playing card that has a face (e.g., "Ace", "Deuce", "Three", …, "Jack", "Queen", "King") and a suit (e.g., "Hearts", "Diamonds", "Clubs", "Spades").
2) Develop a DeckOfCards class, which creates a deck of 52 playing cards in which each element is a Card object, and demonstrates card shuffling and dealing capabilities. In addition, include methods DeckOfCards class that determine whether a hand contains
    a) a pair
    b) two pairs
    c) three of a kind (e.g., three jacks)
    d) four of a kind (e.g., four aces)
    e) a flush (i.e., all five cards of the same suit)
    f) a straight (i.e., five cards of consecutive face values)
    g) a full house (i.e., two cards of one face value and three cards of another face value)

Code with Explanation

Card class

```
package Lab03;

public class Card {

	private final Face face;
	private final Suit suit;

	public Card(Face face, Suit suit) {
		this.face = face;
		this.suit = suit;
	}

	public String toString() {
		return face + " of " + suit;
	}

	enum Suit {
		DIAMONDS, CLUBS, HEARTS, SPADES
	}

	enum Face {
		DEUCE, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN,
KING, ACE
	}

	public Face getFace() {
		return face;
	}

	public Suit getSuit() {
		return suit;
	}
}
```

Deck class

```java
package Lab03;

import java.util.Random;

import Lab03.Card.Face;
import Lab03.Card.Suit;

public class Deck {
	private static final Random randomNumbers = new Random(); // random number
generator
	private static final int NUMBER_OF_CARDS = 52; // constant number of cards

	private Card[] deck; // array of Card objects
	private int currentCard; // the index of next Card to be dealt

	public Deck() {
		currentCard = 0;
		deck = new Card[NUMBER_OF_CARDS];
		int i = 0;

		// The way the first conditional statement is simpler, more readable,
and
		// preferred over the second conditional statement
		// Create a deck of cards by traversing all values of the Suit
enumeration and
		// by traversing all values of the Face enumeration for each Suit
value
		for (Suit s : Suit.values()) // array type Suit
			for (Face f : Card.Face.values())
				deck[i++] = new Card(f, s);

		// Access enumeration values for Suit and Face through the index and
create a
		// card decks
		/*
		 * for (int s = 0; s < Suit.values().length; s++) // array type Suit
for (int f
		 * = 0; f < Face.values().length; f++) deck[i++] = new
Card(Face.values()[f],
		 * Suit.values()[s]);
		 */
	}

	public void shuffle() { // only changed type Card2 => Card
		currentCard = 0;
		// for each card, pick another random card and swap them
		for (int first = 0; first < deck.length; first++) {
			int second = randomNumbers.nextInt(NUMBER_OF_CARDS);
			Card temp = deck[first];
			deck[first] = deck[second];
			deck[second] = temp;
		}
	}

	public void printDeck() {
```

```
		for (int i = 0; i < NUMBER_OF_CARDS; i++) {
			if (i % 4 == 0)
				System.out.println();
			System.out.printf(" %-20s", dealCard());
		}
		System.out.println();
	}

	public Card dealCard() {
		// determine whether cards remain to be dealt
		System.out.printf("%2d", currentCard);
		if (currentCard < deck.length)
			return deck[currentCard++]; // return current Card in array
		else
			return null; // return null to indicate that all cards were
dealt
	}

	/*
	 * public static void main(String[] args) { Deck deck = new Deck();
	 * deck.printDeck(); deck.shuffle(); deck.printDeck(); }
	 */

}
```

DeckOfCards class

```
package Lab03;

import java.util.Arrays;

public class DeckOfCards {
	public boolean hasPair(Card[] hand) {
		// Returns true if two cards with the same face exist
		for (int i = 0; i < hand.length - 1; i++)
			for (int j = i + 1; j < hand.length; j++)
				if (hand[i].getFace() == hand[j].getFace())
					return true;
		return false;
	}

	public boolean hasTwoPairs(Card[] hand) {
		// Returns true if two pairs of cards with the same face exist
		int count = 0;
		for (int i = 0; i < hand.length - 1; i++)
			for (int j = i + 1; j < hand.length; j++)
				if (hand[i].getFace() == hand[j].getFace()) {
					count++;
					if (count == 2) return true;
				}
		return false;
	}

	public boolean hasThreeOfAKind(Card[] hand) {
		// Return true if three identical face cards exist
		for (int i = 0; i < hand.length; i++) {
			int count = 0;
```

```
                        for (int j = 0; j < hand.length; j++)
                                if (hand[i].getFace() == hand[j].getFace()) count++;
                        if (count == 3) return true;
                }
                return false;
        }

        public boolean hasFourOfAKind(Card[] hand) {
                //Return true if four identical face cards exist
                for (int i = 0; i < hand.length; i++) {
                        int count = 0;
                        for (int j = 0; j < hand.length; j++)
                                if (hand[i].getFace() == hand[j].getFace()) count++;
                        if (count == 4) return true;
                }
                return false;
        }

        public boolean hasFlush(Card[] hand) {
                // Return true if all cards in your hand are in the same suit
                Card.Suit suit = hand[0].getSuit(); // Based on the first suit
                for (int i = 1; i < hand.length; i++)
                        if (hand[i].getSuit() != suit) return false;
                return true;
        }

        public boolean hasStraight(Card[] hand) {
                // Sort cards by face
                Card[] sortedHand = Arrays.copyOf(hand, hand.length);
                sortCardsByFace(sortedHand);
                sortCardsByFace(hand);

                // Return true if you have five successive faces on your hand
                for (int i = 0; i < hand.length - 1; i++)
                        if (hand[i].getFace().ordinal() != hand[i +
1].getFace().ordinal() - 1)
                                return false;
                return true;
        }

        public boolean hasFullHouse(Card[] hand) {
                // Return true if full house(i.e., two cards of one face value and
three cards of another face value)
                boolean hasThree = false;
                boolean hasTwo = false;

                for (int i = 0; i < hand.length; i++) {
                        int count = 0;
                        for (int j = 0; j < hand.length; j++)
                                if (hand[i].getFace() == hand[j].getFace())count++;
                        if (count == 3) hasThree = true;
                        else if (count == 2) hasTwo = true;
                }
                return hasThree && hasTwo;
        }
```

```
	private void sortCardsByFace(Card[] hand) {
		// Method that sort by face
		for (int i = 0; i < hand.length - 1; i++) {
			for (int j = 0; j < hand.length - i - 1; j++) {
				if (hand[j].getFace().ordinal() > hand[j +
1].getFace().ordinal()) {
					Card temp = hand[j];
					hand[j] = hand[j + 1];
					hand[j + 1] = temp;
				}
			}
		}
	}
}
```

Card2 class

```
package Lab03;

public class Card2 {

	private final String face; // face of card ("Ace", "Deuce", ...)
	private final String suit; // suit of card ("Hearts", "Diamonds", ...)

	// two-argument constructor initializes card's face and suit
	public Card2(String face, String suit) {
		this.face = face;
		this.suit = suit;
	}

	// return String representation of Card
	public String toString() {
		return face + " of " + suit;
	}

	public String getFace() {
		return face;
	}

	public String getSuit() {
		return suit;
	}

	public int getFaceAsInt() {
	  // Map face values to integers
	  switch (face) {
	      case "Ace":
	          return 1;
	      case "Deuce":
	          return 2;
	      case "Three":
	          return 3;
	      case "Four":
	          return 4;
```

```
            case "Five":
                return 5;
            case "Six":
                return 6;
            case "Seven":
                return 7;
            case "Eight":
                return 8;
            case "Nine":
                return 9;
            case "Ten":
                return 10;
            case "Jack":
                return 11;
            case "Queen":
                return 12;
            case "King":
                return 13;
            default:
                // Handle invalid face values
                throw new IllegalArgumentException("Invalid face value: " + face);
        }
    }
}
```

Deck2 class

```
package Lab03;

import java.util.Random;

public class Deck2 {

	private static final String[] faces = { "Ace", "Deuce", "Three", "Four",
"Five", "Six", "Seven", "Eight", "Nine",
			"Ten", "Jack", "Queen", "King" };

	private static final String[] suits = { "Hearts", "Diamonds", "Clubs",
"Spades" };
	private static final Random randomNumbers = new Random(); // random number
generator
	private static final int NUMBER_OF_CARDS = 52; // constant number of cards

	private Card2[] deck; // array of Card objects
	private int currentCard; // the index of next Card to be dealt

	// constructor fills deck of cards
	public Deck2() {
		deck = new Card2[NUMBER_OF_CARDS]; // create array of Card objects
		currentCard = 0; // initialize currentCard

		for (int count = 0; count < deck.length; count++) // populate deck
with Card objects
			deck[count] = new Card2(faces[count % 13], suits[count / 13]);
	}

	// shuffle deck of cards with one-pass algorithm
```

```
	public void shuffle() {
		currentCard = 0; // reinitialize currentCard

		// for each card, pick another random card and swap them
		for (int first = 0; first < deck.length; first++) {
			int second = randomNumbers.nextInt(NUMBER_OF_CARDS);
			Card2 temp = deck[first];
			deck[first] = deck[second];
			deck[second] = temp;
		}
	}

	// deal one card
	public Card2 dealCard() {
		// determine whether cards remain to be dealt
		if (currentCard < deck.length)
			return deck[currentCard++]; // return current Card in array
		else
			return null; // return null to indicate that all cards were
dealt
	}

	public void printDeck() {
		for (int i = 0; i < NUMBER_OF_CARDS; i++) {
			if (i % 4 == 0)
				System.out.println();
			System.out.printf("%-20s", dealCard());
		}
		System.out.println();
	}
}
```

DeckOfCards2 class

```
package Lab03;

import java.util.Arrays;

public class DeckOfCards2 {
    public boolean hasPair(Card2[] hand) {
        // Returns true if two cards with the same face exist
        for (int i = 0; i < hand.length - 1; i++)
            for (int j = i + 1; j < hand.length; j++)
                if (hand[i].getFace().equals(hand[j].getFace()))
                    return true;
        return false;
    }

    public boolean hasTwoPairs(Card2[] hand) {
        // Returns true if two pairs of cards with the same face exist
        int count = 0;
        for (int i = 0; i < hand.length - 1; i++)
            for (int j = i + 1; j < hand.length; j++)
                if (hand[i].getFace().equals(hand[j].getFace())) {
                    count++;
                    if (count == 2)
                        return true;
```

```
                }
        return false;
    }

    public boolean hasThreeOfAKind(Card2[] hand) {
        // Return true if three identical face cards exist
        for (int i = 0; i < hand.length; i++) {
            int count = 0;
            for (int j = 0; j < hand.length; j++)
                if (hand[i].getFace().equals(hand[j].getFace()))
                    count++;
            if (count == 3)
                return true;
        }
        return false;
    }

    public boolean hasFourOfAKind(Card2[] hand) {
        // Return true if four identical face cards exist
        for (int i = 0; i < hand.length; i++) {
            int count = 0;
            for (int j = 0; j < hand.length; j++)
                if (hand[i].getFace().equals(hand[j].getFace()))
                    count++;
            if (count == 4)
                return true;
        }
        return false;
    }

    public boolean hasFlush(Card2[] hand) {
        // Return true if all cards in your hand are in the same suit
        String suit = hand[0].getSuit(); // Based on the first suit
        for (int i = 1; i < hand.length; i++)
            if (!hand[i].getSuit().equals(suit))
                return false;
        return true;
    }

    public boolean hasStraight(Card2[] hand) {
        // Sort cards by face
        Card2[] sortedHand = Arrays.copyOf(hand, hand.length);
        sortCardsByFace(sortedHand);
        sortCardsByFace(hand);

        // Return true if you have five successive faces on your hand
        for (int i = 0; i < hand.length - 1; i++)
            if (hand[i].getFaceAsInt() != hand[i + 1].getFaceAsInt() - 1)
                return false;
        return true;
    }

    public boolean hasFullHouse(Card2[] hand) {
        // Return true if a full house (i.e., two cards of one face value and three
cards of another face value) exists
```

```
        boolean hasThree = false;
        boolean hasTwo = false;

        for (int i = 0; i < hand.length; i++) {
            int count = 0;
            for (int j = 0; j < hand.length; j++)
                if (hand[i].getFace().equals(hand[j].getFace()))
                    count++;
            if (count == 3)
                hasThree = true;
            else if (count == 2)
                hasTwo = true;
        }
        return hasThree && hasTwo;
    }

    private void sortCardsByFace(Card2[] hand) {
        // Method that sorts by face
        Arrays.sort(hand, (a, b) -> a.getFaceAsInt() - b.getFaceAsInt());
    }
}
```

Results/Output

Deck class

```
 0 DEUCE of DIAMONDS     1 THREE of DIAMONDS     2 FOUR of DIAMONDS      3 FIVE of DIAMONDS
 4 SIX of DIAMONDS       5 SEVEN of DIAMONDS     6 EIGHT of DIAMONDS     7 NINE of DIAMONDS
 8 TEN of DIAMONDS       9 JACK of DIAMONDS     10 QUEEN of DIAMONDS    11 KING of DIAMONDS
12 ACE of DIAMONDS      13 DEUCE of CLUBS       14 THREE of CLUBS       15 FOUR of CLUBS
16 FIVE of CLUBS        17 SIX of CLUBS         18 SEVEN of CLUBS       19 EIGHT of CLUBS
20 NINE of CLUBS        21 TEN of CLUBS         22 JACK of CLUBS        23 QUEEN of CLUBS
24 KING of CLUBS        25 ACE of CLUBS         26 DEUCE of HEARTS      27 THREE of HEARTS
28 FOUR of HEARTS       29 FIVE of HEARTS       30 SIX of HEARTS        31 SEVEN of HEARTS
32 EIGHT of HEARTS      33 NINE of HEARTS       34 TEN of HEARTS        35 JACK of HEARTS
36 QUEEN of HEARTS      37 KING of HEARTS       38 ACE of HEARTS        39 DEUCE of SPADES
40 THREE of SPADES      41 FOUR of SPADES       42 FIVE of SPADES       43 SIX of SPADES
44 SEVEN of SPADES      45 EIGHT of SPADES      46 NINE of SPADES       47 TEN of SPADES
48 JACK of SPADES       49 QUEEN of SPADES      50 KING of SPADES       51 ACE of SPADES

 0 EIGHT of CLUBS
 1 JACK of HEARTS
 2 DEUCE of CLUBS
 3 DEUCE of HEARTS
 4 EIGHT of DIAMONDS

Has Pair: true
Has Two Pairs: true
Has Three of a Kind: false
Has Four of a Kind: false
Has Flush: false
Has Straight: false
Has Full House: false

 5 QUEEN of SPADES       6 QUEEN of CLUBS        7 JACK of CLUBS         8 KING of SPADES
 9 ACE of CLUBS         10 QUEEN of DIAMONDS    11 ACE of HEARTS        12 THREE of HEARTS
13 EIGHT of HEARTS      14 SIX of DIAMONDS      15 FIVE of HEARTS       16 ACE of SPADES
17 NINE of DIAMONDS     18 SIX of CLUBS         19 NINE of HEARTS       20 ACE of DIAMONDS
21 FIVE of DIAMONDS     22 SEVEN of DIAMONDS    23 EIGHT of SPADES      24 KING of HEARTS
25 SEVEN of CLUBS       26 JACK of DIAMONDS     27 KING of DIAMONDS     28 FOUR of HEARTS
29 FOUR of CLUBS        30 NINE of SPADES       31 NINE of CLUBS        32 DEUCE of DIAMONDS
33 FOUR of DIAMONDS     34 FIVE of SPADES       35 THREE of DIAMONDS    36 THREE of SPADES
37 TEN of DIAMONDS      38 SIX of HEARTS        39 SEVEN of SPADES      40 KING of CLUBS
41 SIX of SPADES        42 FOUR of SPADES       43 FIVE of CLUBS        44 THREE of CLUBS
45 SEVEN of HEARTS      46 TEN of HEARTS        47 DEUCE of SPADES      48 QUEEN of HEARTS
49 TEN of CLUBS         50 JACK of SPADES       51 TEN of SPADES        52 null
52 null                 52 null                 52 null                 52 null
```

1. 섞기 전 52장의 카드를 보여줌 2. 섞은 후 섞은 카드 52장 중 5장을 뽑음 3. 이 중에서 2)-a)~g)에 해당하는 카드 쌍이 있는지 확인 4. 5장을 뽑고 난 후 나머지 카드 보여줌	1. Show 52 cards before mixing 2. 5 cards out of 52 mixed cards 3. Determine if there is a pair of cards corresponding to 2)-a) to g) 4. After 5 cards, show the rest of the cards

Deck2 class

```
Ace of Hearts       Deuce of Hearts     Three of Hearts     Four of Hearts
Five of Hearts      Six of Hearts       Seven of Hearts     Eight of Hearts
Nine of Hearts      Ten of Hearts       Jack of Hearts      Queen of Hearts
King of Hearts      Ace of Diamonds     Deuce of Diamonds   Three of Diamonds
Four of Diamonds    Five of Diamonds    Six of Diamonds     Seven of Diamonds
Eight of Diamonds   Nine of Diamonds    Ten of Diamonds     Jack of Diamonds
Queen of Diamonds   King of Diamonds    Ace of Clubs        Deuce of Clubs
Three of Clubs      Four of Clubs       Five of Clubs       Six of Clubs
Seven of Clubs      Eight of Clubs      Nine of Clubs       Ten of Clubs
Jack of Clubs       Queen of Clubs      King of Clubs       Ace of Spades
Deuce of Spades     Three of Spades     Four of Spades      Five of Spades
Six of Spades       Seven of Spades     Eight of Spades     Nine of Spades
Ten of Spades       Jack of Spades      Queen of Spades     King of Spades

King of Spades
Five of Clubs
Ten of Clubs
Ten of Diamonds
King of Clubs

Has Pair: true
Has Two Pairs: true
Has Three of a Kind: false
Has Four of a Kind: false
Has Flush: false
Has Straight: false
Has Full House: false

Six of Hearts       Six of Spades       Jack of Spades      Queen of Hearts
Nine of Spades      Three of Clubs      King of Diamonds    Ace of Clubs
Ace of Spades       Jack of Clubs       Six of Clubs        Queen of Diamonds
Deuce of Clubs      Ace of Diamonds     Deuce of Hearts     Six of Diamonds
Queen of Clubs      Nine of Hearts      Three of Diamonds   Four of Diamonds
Queen of Spades     Deuce of Spades     Four of Hearts      King of Hearts
Nine of Clubs       Seven of Clubs      Five of Spades      Eight of Clubs
Three of Hearts     Ace of Hearts       Ten of Hearts       Jack of Hearts
Four of Spades      Eight of Diamonds   Eight of Spades     Deuce of Diamonds
Five of Hearts      Seven of Spades     Eight of Hearts     Jack of Diamonds
Seven of Hearts     Three of Spades     Four of Clubs       Seven of Diamonds
Five of Diamonds    Nine of Diamonds    Ten of Spades       null
null                null                null                null
```

Task-3: (Tic-Tac-Toe)

Create a class TicTacToe that will enable you to write a program to play Tic-Tac-Toe. The class contains a private 3-by-3 two-dimensional array. Use an **enum** type to represent the value in each cell of the array. The **enum's** constants should be named X, O and EMPTY (for a position that does not contain an X or an O). The constructor should initialize the board elements to EMPTY. Allow two human players. Wherever the first player moves, place an X in the specified square, and place an O wherever the second player moves. Each move must be to an empty square. After each move, determine whether the game has been won and whether it's a draw.

Code with Explanation

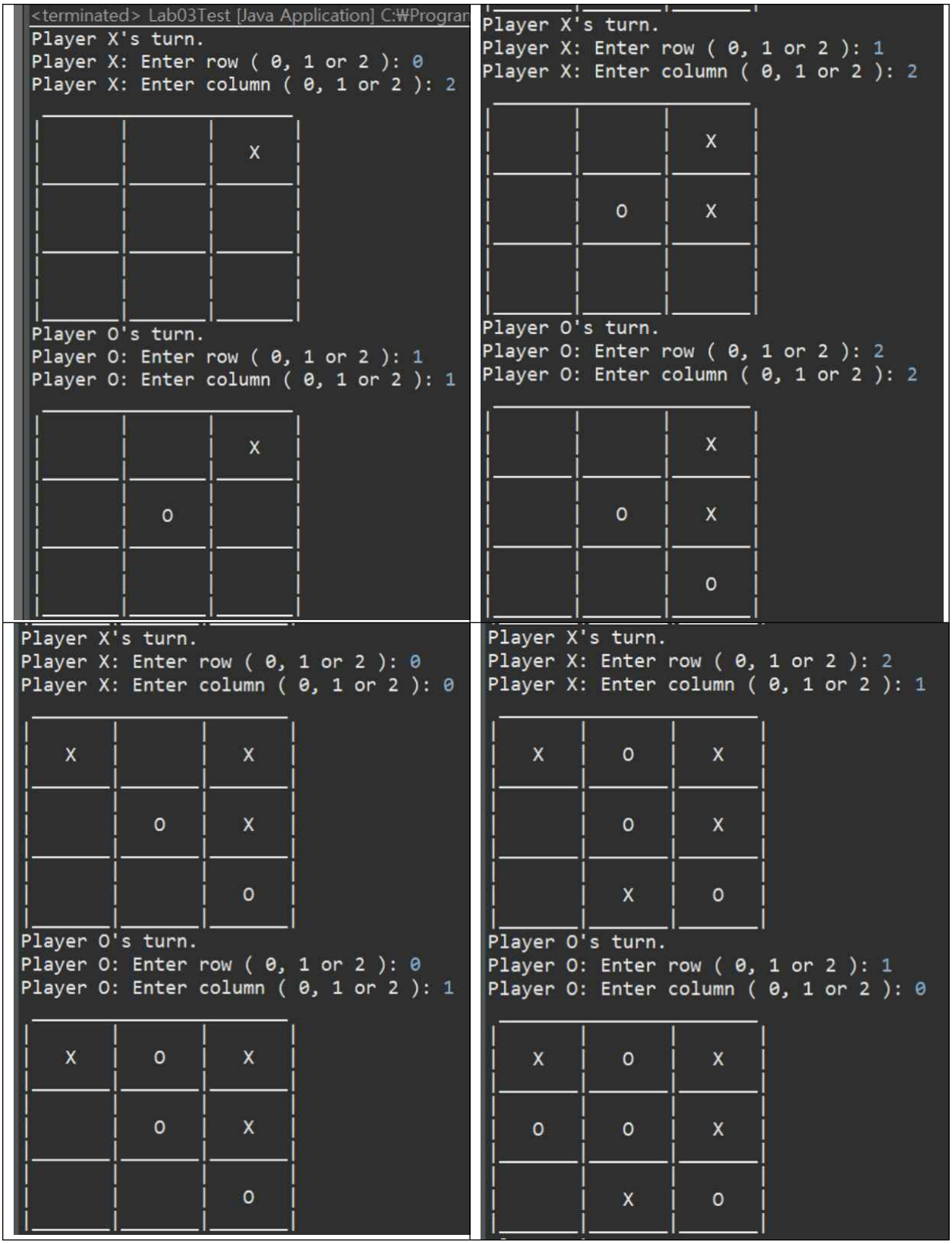
TicTacToe class

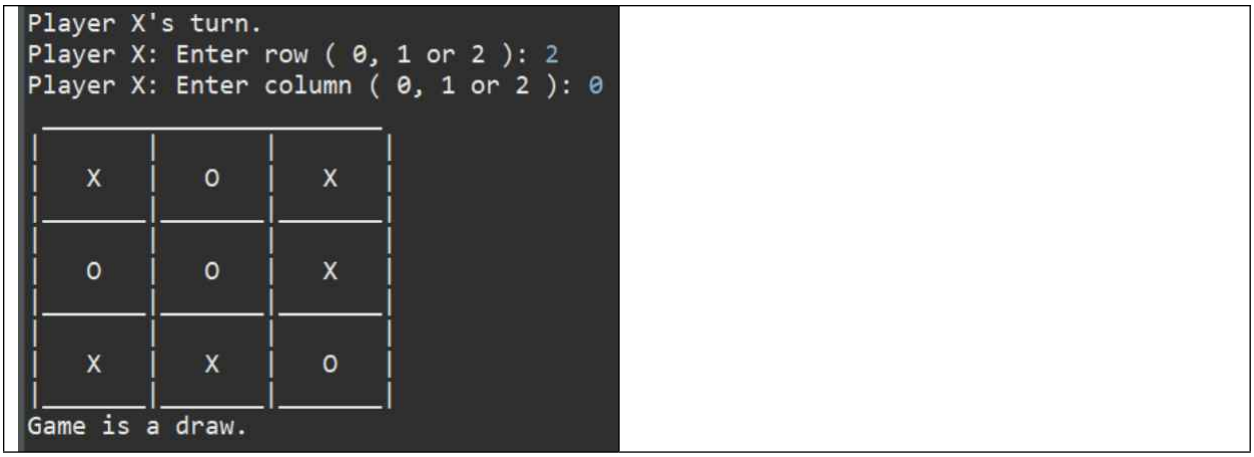
```
package Lab03;

import java.util.Scanner;

public class TicTacToe {
	private final int BOARDSIZE = 3; // size of the board

	private enum Status {
		WIN, DRAW, CONTINUE
	}; // game states

	private char[][] board; // board representation
	private boolean firstPlayer; // whether it's player 1's move
	private boolean gameOver; // whether game is over

	// Constructor
	public TicTacToe() {
		board = new char[BOARDSIZE][BOARDSIZE];
		firstPlayer = true;
		gameOver = false;
	} // end Constructor

	// start game
	public void play() {
		Scanner input = new Scanner(System.in);
		int row; // row for next move
		int column; // column for next move

		System.out.println("Player X's turn.");

		while (!gameOver) {
			char player = (firstPlayer ? 'X' : 'O');

			// player's turn
			do {
				System.out.printf("Player %c: Enter row ( 0, 1 or 2 ): 
", player);
				row = input.nextInt();
				System.out.printf("Player %c: Enter column ( 0, 1 or 
2 ): ", player);
```

```
                        column = input.nextInt();
                } while (!validMove(row, column));

                board[row][column] = player;

                firstPlayer = !firstPlayer;

                printBoard();
                printStatus(player);
            } // end while
            input.close();
        } // end method play

        // show game status in status bar
        private void printStatus(int player) {
            Status status = gameStatus();

            // check game status
            switch (status) {
            case WIN:
                System.out.printf("Player %c wins.", player);
                gameOver = true;
                break;
            case DRAW:
                System.out.println("Game is a draw.");
                gameOver = true;
                break;
            case CONTINUE:
                if (player == 'X')
                    System.out.println("Player O's turn.");
                else
                    System.out.println("Player X's turn.");
                break;
            } // end switch
        } // end method printStatus

        // get game status
        private Status gameStatus() {
            int a;

            // check for a win on diagonals
            if (board[0][0] != 0 && board[0][0] == board[1][1] && board[0][0] ==
board[2][2])
                return Status.WIN;
            else if (board[2][0] != 0 && board[2][0] == board[1][1] &&
board[2][0] == board[0][2])
                return Status.WIN;

            // check for win in rows
            for (a = 0; a < 3; a++)
                if (board[a][0] != 0 && board[a][0] == board[a][1] &&
board[a][0] == board[a][2])
                    return Status.WIN;

            // check for win in columns
```

```
            for (a = 0; a < 3; a++)
                  if (board[0][a] != 0 && board[0][a] == board[1][a] &&
board[0][a] == board[2][a])
                        return Status.WIN;

            // check for a completed game
            for (int r = 0; r < 3; r++)
                  for (int c = 0; c < 3; c++)
                        if (board[r][c] == 0)
                              return Status.CONTINUE; // game is not finished

            return Status.DRAW; // game is a draw
      } // end method gameStatus

      // display board
      public void printBoard() {
            System.out.println(" _____________________ ");

            for (int row = 0; row < BOARDSIZE; row++) {
                  System.out.println("|       |       |       |");

                  for (int column = 0; column < BOARDSIZE; column++)
                        printSymbol(column, board[row][column]);

                  System.out.println("|_______|_______|_______|");
            } // end for
      } // end method printBoard

      // print moves
      private void printSymbol(int column, char value) {
            System.out.printf("|   %c   ", value);

            if (column == 2)
                  System.out.println("|");
      } // end method printSymbol

      // validate move
      private boolean validMove(int row, int column) {
            return row >= 0 && row < 3 && column >= 0 && column < 3 &&
board[row][column] == 0;
      } // end method validMove
}
```

Results/Output




Player X's turn.
Player X: Enter row (0, 1 or 2): 2
Player X: Enter column (0, 1 or 2): 0
X O X
O O X
X X O
Game is a draw.

Code with Explanation

Lab03Test class

```
package Lab03;

public class Lab03Test {

	public static void main(String[] args) {
		// useTicTacToe();
		// useDeck();
		// useDeck2();
		// useCard2();
		// useRoll36();
	}

	public static void useRoll36() {
		Roll36 r = new Roll36();
		r.rollDice36();
	}

	public static void useDeck() {
		DeckOfCards deck = new DeckOfCards();
		Deck d = new Deck();

		d.printDeck();
		d.shuffle();

		// 5 cards out of 52 mixed cards
		System.out.println();
		Card[] hand = new Card[5];
	  for (int i = 0; i < 5; i++) {
	      hand[i] = d.dealCard();
	      System.out.println(" " + hand[i]);
	  }
		System.out.println();

		// Check 5 cards per method
		boolean hasPair = deck.hasPair(hand);
		boolean hasTwoPairs = deck.hasTwoPairs(hand);
		boolean hasThreeOfAKind = deck.hasThreeOfAKind(hand);
		boolean hasFourOfAKind = deck.hasFourOfAKind(hand);
		boolean hasFlush = deck.hasFlush(hand);
		boolean hasStraight = deck.hasStraight(hand);
		boolean hasFullHouse = deck.hasFullHouse(hand);

		System.out.println("Has Pair: " + hasPair);
		System.out.println("Has Two Pairs: " + hasTwoPairs);
		System.out.println("Has Three of a Kind: " + hasThreeOfAKind);
		System.out.println("Has Four of a Kind: " + hasFourOfAKind);
		System.out.println("Has Flush: " + hasFlush);
		System.out.println("Has Straight: " + hasStraight);
		System.out.println("Has Full House: " + hasFullHouse);
		d.printDeck();
	}
```

```
	public static void useDeck2() {
		DeckOfCards2 deck2 = new DeckOfCards2();
		Deck2 d2 = new Deck2();
		d2.printDeck();
		d2.shuffle();

		// 5 cards out of 52 mixed cards
		System.out.println();
		Card2[] hand = new Card2[5];
	 for (int i = 0; i < 5; i++) {
		hand[i] = d2.dealCard();
		System.out.println(hand[i]);
	 }
		System.out.println();

		// Check 5 cards per method
		boolean hasPair = deck2.hasPair(hand);
		boolean hasTwoPairs = deck2.hasTwoPairs(hand);
		boolean hasThreeOfAKind = deck2.hasThreeOfAKind(hand);
		boolean hasFourOfAKind = deck2.hasFourOfAKind(hand);
		boolean hasFlush = deck2.hasFlush(hand);
		boolean hasStraight = deck2.hasStraight(hand);
		boolean hasFullHouse = deck2.hasFullHouse(hand);

		System.out.println("Has Pair: " + hasPair);
		System.out.println("Has Two Pairs: " + hasTwoPairs);
		System.out.println("Has Three of a Kind: " + hasThreeOfAKind);
		System.out.println("Has Four of a Kind: " + hasFourOfAKind);
		System.out.println("Has Flush: " + hasFlush);
		System.out.println("Has Straight: " + hasStraight);
		System.out.println("Has Full House: " + hasFullHouse);
		d2.printDeck();
	}

	/*
	 * public static void useCard2() { Card2 c = new Card2("face", "suit");
	 * System.out.println(c); }
	 */

	public static void useTicTacToe() {
		TicTacToe ticTacToe = new TicTacToe();
		ticTacToe.play();
	}
}
```

Conclusion

이번 과제를 통해 클래스 간의 Association 과 Aggregation에 대해서 잘 이해할 수 있게 되었다. 카드 게임 같은 경우에는 카드 게임을 해본 적이 없어서 룰을 이해 해야 해서 생각보다 어려웠다. 그리고 UML다이어그램을 만들 때, enum도 따로 추가해서 넣는 것 인줄 이번에 처음 배우게 되었다. 이번에는 UML이 저번주보다 더 복잡해진 느낌이 들었다.

Through this task, we can better understand the association and aggregation between classes. In the case of card games, it was more difficult than I thought because I had never played a card game before, so I had to understand the rules. And when I made a UML diagram, I learned for the first time that I was adding enum separately. This time, UML felt more complicated than last week.