```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Random;
import java.util.Scanner;

public class PD_Mexico {
    // A standard Scanner object to acquire input
    static Scanner sc = new Scanner(System.in);
    // A random object
    static Random rand = new Random();
    // Holds if the amount of allowed rolls has been declared
    static boolean chosenPermittedRolls = false;

    // The initial lives of all players
    static int[] lives = { 6, 6, 6 };
    // Holds all the possible roll values in a descending order of value
    static ArrayList<Integer> order = new ArrayList<Integer>(
            Arrays.asList(21, 66, 55, 44, 33, 22, 11, 65, 64, 63, 62, 61,
54, 53, 52, 51, 43, 42, 41, 32, 31));
    static int currentPlayer = 0;

    // Main method
    public static void main(String[] args) {
        // A small greeting
        System.out.println("====================\nWelcome to
((MEXICO))\n====================");[][]
        // Continues until 2 players have no lives left
        while ((lives[0] > 0 && lives[1] > 0) || (lives[0] > 0 && lives[2]
> 0) || (lives[1] > 0 && lives[2] > 0)) {
            // Holds the roll results
            int[] rollResults = new int[3];
            // Sets the current allowed rolls to 3
            int permittedRolls = 3;
            // Resets the chosen allowed rolls back to false
            chosenPermittedRolls = false;
            // Loops through each move
            for (int player = 0; player < 3; player++) {
```

```
                int realPlayer = player;
                if (player != -1) {
                    // Gets the starting player from the last
currentPlayer
                    realPlayer = (player + currentPlayer) % 3;
                }
                // Only executes if the player is still alive
                if (!(lives[realPlayer] < 1)) {
                    // Set the allowed rolls to the result of the
processing function
                    permittedRolls = getPlayerRolls(realPlayer,
permittedRolls, rollResults);
                    // Print their roll result and end their turn.
                    System.out.printf("+> Player [%d] ended this turn with
a roll of %d!\n", realPlayer + 1,
                            rollResults[realPlayer]);
                    System.out.println();
                }
            }
            // Calculates the currentPlayer of the round
            currentPlayer = calculateLoser(rollResults);
            // Handles if the players didn't get a tie
            if (currentPlayer != -1) {
                // Decreases a life from the currentPlayer
                lives[currentPlayer]--;
                // Shows the players the round's loser and their remaining
lives
                System.out.printf(
                        "===> Player [%d] lost this round with a roll of
%d. They now have %d live(s) left. <===\n\n\n",
                        currentPlayer + 1, rollResults[currentPlayer],
lives[currentPlayer]);
            }
        }
        // When there is only one alive player, print who it is
        printWinner();
    }
```

```
    /**
     * Calculates the result of player rolls in a given turn
     *
     * @param player         The current player
     * @param permittedRolls The number of permitted rolls
     * @param rollResults    The results of the round's rolls
     * @return The number of permitted remaining rolls
     */
    private static int getPlayerRolls(int player, int permittedRolls,
int[] rollResults) {
        // Quits short circuit variable
        boolean quitFlag = false;
        // Loops through the amount of allowed rolls
        for (int roll = 1; roll <= permittedRolls && !quitFlag; roll++) {
            // Gets a random roll result as if we rolled a dice
            int rollResult = getRandomRoll();
            // Prints the roll
            System.out.printf("Player [%d] roll: %d. You have %d roll(s)
left.\n", player + 1, rollResult,
                    permittedRolls - roll);
            // If we haven't reached the limit of our rolls...
            if (roll != permittedRolls) {
                // Asks if the user wants to roll again
                System.out.print("Are you happy with this roll? 1 = YES/0
= NO ");
                // Gets their choice
                int wantQuit = sc.nextInt();
                // If they want to quit...
                if (wantQuit == 1) {
                    // Sets their roll result to their current roll
                    rollResults[player] = rollResult;
                    // If the allowed rolls hasn't been chosen yet...
                    if (!chosenPermittedRolls) {
                        // Sets the allowed rolls to the amount of rolls
used
                        chosenPermittedRolls = true;
```

```
                        permittedRolls = roll;
                    }
                    quitFlag = true;
                }
            } else {
                // If they run out of rolls, sets their roll result to
their current roll
                // without asking
                rollResults[player] = rollResult;
            }
        }
        // Sets allowed rolls to make sure the first player doesn't have
any issues
        if (!chosenPermittedRolls && !quitFlag) {
            chosenPermittedRolls = true;
            permittedRolls = 3;
        }
        // Return the amount of allowed rolls
        return permittedRolls;
    }

    /**
     * Calculates the loser of a round using provided results
     *
     * @param rollResults: The results of each player's roll
     *
     * @return the index of the losing player
     */
    private static int calculateLoser(int[] rollResults) {
        // Calculates the index of a roll in the score list. The lowest
index has the
        // highest score
        int[] index = { order.indexOf(rollResults[0]),
order.indexOf(rollResults[1]), order.indexOf(rollResults[2]) };
        // Sets the currentPlayer to a garbage variable for the start
        int currentPlayer = -1;
        // Gets the worst roll as the max of the indices of the list
```

```
        int losingScore = Math.max(Math.max(index[0], index[1]),
index[2]);
        // List of all the losingPlayers
        ArrayList<Integer> losingPlayers = new ArrayList<Integer>();
        // Loops through all the players
        for (int player = 0; player < 3; player++) {
            // Add a player to the losingPlayers list if they have the
lowest score
            if (index[player] == losingScore) {
                losingPlayers.add(player);
            }
        }
        // Handles the possible cases of losingPlayers
        switch (losingPlayers.size()) {
        // If all the players rolled the same roll
        case 3:
            System.out.println("This round was a draw!\n\n");
            break;
        // If two players rolled the same roll, decide randomly
        case 2:
            System.out.printf(
                    "Player [%d] and player [%d] have tied! The
currentPlayer is decided by a random coin flip...\n",
                    losingPlayers.get(0) + 1, losingPlayers.get(1) + 1);
            currentPlayer = losingPlayers.get(rand.nextInt(2));
            System.out.printf("Player [%d] lost the coin flip!
Unlucky.\n", currentPlayer + 1);
            break;
        // This would mean there is only one player who rolled the lowest
score, so
        // returns them
        default:
            currentPlayer = losingPlayers.get(0);
            break;

        }
```

```
        // Returns the currentPlayer to the caller
        return currentPlayer;
    }

    /**
     * Prints the winner from the amount of lives players have left.
     */
    private static void printWinner() {
        // Loops through the players
        for (int player = 0; player < 3; player++) {
            // Print that a player is alive if they have more than 0 lives
            if (lives[player] > 0) {
                System.out.printf("-==-=->> Player [%d] is the winner!!
Thanks for playing. <<=-=-=-", player + 1);
            }
        }
    }

    /**
     * Produces a random dice roll between 1 and 6
     *
     * @return An integer value from 1 to 6 inclusive
     */
    private static int getRandomRoll() {
        // Get the first roll (1-6)
        int dice1 = rand.nextInt(6) + 1;
        // Get the second roll (1-6)
        int dice2 = rand.nextInt(6) + 1;
        // Calculate the roll result from which roll is higher.
        return dice1 > dice2 ? dice1 * 10 + dice2 : dice2 * 10 + dice1;
    }
}
```