```java
import java.util.Scanner;

public class Streak {

    // Initialising Queue Linked Lists ADT to store cards to carry out the "replay".
    private static final Queue<Card> REPLAY = new Queue<>();
    private static final Queue<Card> REPLAY_SELECTION = new Queue<>();

    // Initialising a List Chain ADT to store the score and names of each player after a completed match.
    private static final ListChain<Scoreboard> SCOREBOARD = new ListChain<>();



    // --- VALIDATION METHODS ---
    // This method is used to check if the input is valid and compare it to the choices given depending on the type of choice that it is and return an integer if it is successful in these checks.
    public static int validateChoice(Scanner input, String type, String[] choiceArray, int handLength) {
        // This will initialise and populate the lowercaseChoiceArray with lowercase values of choiceArray to be used to compare the users choice.
        String[] lowercaseChoiceArray = new String[choiceArray.length];
        if (type.equals("card") || type.equals("replay")) for (int i = 0; i < choiceArray.length; i++) lowercaseChoiceArray[i] = choiceArray[i].toLowerCase();

        // Switch Case is used to print out different messages depending on its type of choice.
        switch (type) {
            case "menu" -> System.out.print("\nEnter choice > ");
            case "game" -> System.out.print("\nHow many cards do you want to play with (5-10) > ");
            case "card" -> System.out.print("\nChoose card to change or X to exit > ");
            case "replay" -> System.out.print("\nSee replay? (y/n) > ");
        }
        String choice = input.nextLine();

        // Else If statement is used to return a value of -1 if the user choices to end the round on a "card" type.
        if (type.equals("card") && (choice.equals("X") || choice.equals("x"))) return -1;

        // For Loop will compare String choice to each element of the choiceArray to find if there is a match.
        for (int i = 0; i < choiceArray.length; i++) {
            // This will return the value of i as the input is a letter.
            if ((type.equals("card") && i < handLength || type.equals("replay")) && (choiceArray[i].equals(choice) || lowercaseChoiceArray[i].equals(choice))) return i;

            // This will convert and return the integer value in the String.
            else if((type.equals("menu") || type.equals("game")) && choiceArray[i].equals(choice)) return Integer.parseInt(choice);
        }

        // If String choice is not able to return a value it will display a message and repeat the method.
        System.out.println("Input does not contain a choice from the list, please try again.");
        return validateChoice(input, type, choiceArray, handLength);
    }

    // This method is used to check if the player name is valid and can be properly formatted in scoreboard.
    public static String validateName(Scanner input, int playerNum){
        System.out.print("\nEnter Player " + playerNum + " name > ");
        String name = input.nextLine();

        // If Statement checks that String name is greater than length 10 or is null, if found a message will be displayed and the method will repeat.
        if (name.length() > 10 || name.equals("")) {
            System.out.println("Please enter a name with 10 characters or less.");
            return validateName(input, playerNum);
        }

        // For Loop will check if String name contains anything other than letters, if found a message will be displayed and the method will repeat.
        for (int i = 0; i < name.length(); i++) {
            if(!Character.isLetter(name.charAt(i))) {
                System.out.println("Please enter a name with only letters and no spaces.");
                return validateName(input, playerNum);
            }
        }

        // Once String name has passed the validation check then it can be returned.
        return name;
    }
```

```java
// --- DISPLAY METHODS ---
// This method is used to display each entry in the queues REPLAY and REPLAY_SELECTION until both queues are empty.
public static void displayReplay(Scanner input, int index, String name) {
    // This will print out the title for the method.
    System.out.println("\nREPLAY");
    System.out.println("------");

    // While Loop is used to determine if there are still elements in the REPLAY queue to dequeue.
    while (!REPLAY.isEmpty()) {
        String[] letters = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J"}; // This is used to print a letter for the corresponding card.
        System.out.println("\n" + name);    // This is used to print out the name of the player before displaying the hand.

        // For Loop is used to print out each dequeued element of the REPLAY queue alongside its corresponding letter, i.e. the first card is card A.
        // The amount of elements it will dequeue is dependent on the index which is just how many cards in the hand there were.
        for (int i = 0; i < index; i++) System.out.println(letters[i] + ": " + REPLAY.dequeue());

        // While Loop is used to determine if there are still elements in the REPLAY_SELECTION queue to dequeue.
        if(!REPLAY_SELECTION.isEmpty()) {
            // This will dequeue and print out the selection of the previous hand and wait for the users input to continue.
            System.out.println("\nSelection was " + REPLAY_SELECTION.dequeue());
            System.out.print("(more...) > ");
            input.nextLine();
        }
    }
}

// This method is used to display 5 entries with the highest scores in the list chain SCOREBOARD in ascending order.
public static void displayScoreboard() {
    // This will print out the title for the method.
    System.out.println("\nHIGH SCORES");
    System.out.println("-----------");

    // This will call a method to sort the scores in descending order, remove entries equal to zero and entries after position five.
    sortScoreboard(SCOREBOARD);

    // If Statement is used to determine if the scoreboard is empty, if it is it will print out a message indicating so.
    if(SCOREBOARD.isEmpty()) System.out.println("There are no scores currently held in the scoreboard.");

    // This will print out the top 5 entries in the scoreboard.
    for(int i = 1; i <= SCOREBOARD.getLength(); i++) System.out.print(SCOREBOARD.getEntry(i));
}



// --- SORT METHODS ---
// This is a method that will preform an "Insertion Sort" on the ListChain<Scoreboard> to arrange the scores in ascending order, remove entries after position five and remove entries equal to zero.
public static void sortScoreboard(ListChain<Scoreboard> scoreboard) {
    Scoreboard nextScore;   // This initialises a new Scoreboard called nextScore to store the value of the next score.
    int index;              // This initialises an integer called index to store the position of the current score in the ListChain<Scoreboard>.

    // For Loop is used to sort each element of the ListChain<Scoreboard>.
    for (int i = 2; i < scoreboard.getLength() + 1; i++) {
        nextScore = scoreboard.getEntry(i); // This will give nextScore the value of the next score in the ListChain<Scoreboard>.
        index = i - 1;                      // This will set the value of the index to the position of the current score in the ListChain<Scoreboard>.

        // While Loop is used to shift the score element by one position whilst it hasn't found an element smaller than nextScore.
        while (index >= 1 && scoreboard.getEntry(index).compareTo(nextScore) > 0) {
            scoreboard.replace(index + 1, scoreboard.getEntry(index));
            index--;
        }

        // Once the While Loop has been completed, it has found where the element needs to be inserted and will add it to that position in the ListChain<Scoreboard>.
        scoreboard.replace(index + 1, nextScore);
    }

    // After the scoreboard has been sorted, We will remove all the entries after position 5 as they are no longer needed.
    if(scoreboard.getLength() > 5) for (int i = scoreboard.getLength(); i > 5; i--) scoreboard.remove(i);
}
```

```java
    // This is a method that will preform an "Insertion Sort" to arrange the cards in ascending order.
    public static void sortHand(Card[] hand) {
        Card nextCard;      // This initialises a new Card call nextCard to store the value of the next card.
        int currentCardNum;    // This initialises an integer called currentCardNum to store the value of the current card number in the hand.

        // For Loop is used to sort each element of the hand.
        for (int i = 1; i < hand.length; i++) {
            nextCard = hand[i]; // This will give nextCard the value of the next card in the hand.
            currentCardNum = i - 1;    // This will set the value of currentCardNum to the value of the current card number in the hand.

            // While Loop is used to shift the card element by one position whilst it hasn't found an element smaller than nextCard.
            while (currentCardNum >= 0 && hand[currentCardNum].compareTo(nextCard) > 0) {
                hand[currentCardNum + 1] = hand[currentCardNum];
                currentCardNum--;
            }

            // Once the While Loop has been completed, it has found where the nextCard element needs to be inserted and will add it to that position in the hand.
            hand[currentCardNum + 1] = nextCard;
        }
    }



    // --- PLAY / GAME LOGIC METHODS ---
    // This method will carry out a Single-player game consisting of one round.
    public static void playSingleplayer(Scanner input) {
        String player;
        int score;

        // This is used to call the choice method to determine how many cards will be in the hand.
        String[] digits = {"5", "6", "7", "8", "9", "10"};  // This is used to tell the choice method what choices are available to choose from.
        int numCards = validateChoice(input, "game", digits, 0);

        Deck deck = new Deck();                                 // This initialises a new Deck of cards for the game.
        Card[] hand = new Card[numCards];                       // This initialises a new hand of cards with the amount of cards being determined by the player input.
        // For Loop is used to deal a card from the deck for each element of the hand.
        for (int i = 0; i < numCards; i++) {
            hand[i] = deck.deal();
            // THIS IS A DEBUG MESSAGE
            // System.out.println("\n" + player + " was dealt with the card " + hand[i]);
        }
        // THIS IS A DEBUG MESSAGE
        // System.out.println("\n" + player + "'s Hand:");
        // for (Card card : hand) System.out.println(card);

        // This will ask for the users name and carry out a single round to determine the score.
        player = validateName(input, 1);
        score = playRound(input, hand, deck, player, true);

        Scoreboard scoreboardEntry = new Scoreboard(player, score);
        SCOREBOARD.add(scoreboardEntry);

        // This is used to carry out a "replay" of the game if the user chooses too.
        String[] letters = {"Y", "N"}; // This is used to tell the choice method what choices are available to choose from and to print it.
        if (validateChoice(input, "replay", letters, 0) == 0) displayReplay(input, numCards, player);

        // This will clear both the queues if the player chooses not to see the "replay".
        else {
            REPLAY.clear();
            REPLAY_SELECTION.clear();
            // THIS IS A DEBUG MESSAGE
            // System.out.println("\nIs the REPLAY queue empty? " + REPLAY.isEmpty());
            // System.out.println("\nIs the REPLAY_SELECTION queue empty? " + REPLAY.isEmpty());
        }
    }
```

```java
// This method will carry out a 2-Player game consisting of three rounds and displays a total match score.
public static void play2Player(Scanner input) {
    String player1, player2;        // This is used to initialise a String for both players names to be stored in.
    int score1 = 0, score2 = 0;     // This is used to initialise an integer for both players scores to be stored in.
    int matchScore1 = 0, matchScore2 = 0;

    // This is used to call the choice method to determine how many cards will be in the hand.
    String[] digits = {"5", "6", "7", "8", "9", "10"};  // This is used to tell the choice method what choices are available to choose from.
    int numCards = validateChoice(input, "game", digits, 0);

    // This calls for the name method in order to determine the name of both players.
    player1 = validateName(input, 1);
    player2 = validateName(input, 2);

    // For Loop is used to carry out 3 rounds.
    for (int i = 0; i < 3; i++) {
        Deck deck = new Deck();                                         // This initialises a new Deck of cards for the round.
        Card[] hand1 = new Card[numCards], hand2 = new Card[numCards];  // This initialises a new hand of cards with the amount of cards being determined by the player input.

        // For Loop is used to deal a card from the deck for each element of both players hands respectively.
        for (int j = 0; j < numCards; j++) {
            hand1[j] = deck.deal();
            hand2[j] = deck.deal();
            // THIS IS A DEBUG MESSAGE
            // System.out.println("\n" + player1 + " was dealt with the card " + hand1[j]);
            // System.out.println(player2 + " was dealt with the card " + hand2[j]);
        }

        // THIS IS A DEBUG MESSAGE
        // System.out.println("\n" + player1 + "'s Hand:");
        // for (Card card : hand1) System.out.println(card);
        // System.out.println("\n" + player1 + "'s Hand:");
        // for (Card card : hand2) System.out.println(card);

        // This will print out the current round number.
        System.out.println("\n************");
        System.out.println("Round " + (i + 1) + " of 3");
        System.out.println("************");

        // This will play out a round for both players creating values for the current round scores and add it to the final score.
        score1 = playRound(input, hand1, deck, player1, false);
        score2 = playRound(input, hand2, deck, player2, false);
        matchScore1 += score1;
        matchScore2 += score2;

        // This will mark the end of the current round and print out the statistics for both players round scores.
        System.out.println("\nEND OF ROUND " + (i + 1));
        System.out.println(player1 + " " + score1 + " " + player2 + " " + score2);

        Scoreboard scoreboardEntry1 = new Scoreboard(player1, score1);
        Scoreboard scoreboardEntry2 = new Scoreboard(player2, score2);
        SCOREBOARD.add(scoreboardEntry1);
        SCOREBOARD.add(scoreboardEntry2);

    }
    // This will mark the end of the match and print out the statistics for both players final match scores.
    System.out.println("\nMATCH SCORE");
    System.out.println(player1 + " " + matchScore1 + " " + player2 + " " + matchScore2);
}
```

```java
// This method is used to play out a single round and return the final score for that round.
public static int playRound(Scanner input, Card[] hand, Deck deck, String name, boolean isSingleplayer) {
    String[] letters = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J"}; // This is used to tell the choice method what choices are available to choose from and to print it alongside the cards available in the hand.
    int score = 0;  // This initialises the score as 0 outside the For Loop so that it can be return as the final round score.
    sortHand(hand); // This calls for the sort method to sort the first hand in ascending order.

    // If Statement is used to save the first hand to the queue if it is a Single-Player game.
    if (isSingleplayer) for (Card card : hand) REPLAY.enqueue(card);

    // Main For Loop is used in order to allow the user to exchange a card up to the number of cards in the hand.
    for (int i = 0; i < hand.length; i++) {

        // This will print out the players name and display the current hand.
        System.out.println("\n" + name);
        for (int j = 0; j < hand.length; j++) System.out.println(letters[j] + " : " + hand[j]);

        // This calls for the score method in order to determine the highest streak plus the bonuses in the current hand and prints it out so the user can see.
        score = getScore(hand);
        System.out.println("\nMax Streak Value is " + score);

        // This calls the choice method to give the user the option to exchange a card from the list or finish the round.
        System.out.print("\n" + (i + 1) + " of " + hand.length);
        int cardChoice = validateChoice(input, "card", letters, hand.length);
        if (cardChoice == -1) break;

        // If it is a Single-player game, the card the user has selected will be saved to the queue so that it can be displayed during the replay.
        if(isSingleplayer) REPLAY_SELECTION.enqueue(hand[cardChoice]);

        // This will deal a new card based on the selection, sort the hand and update the score.
        hand[cardChoice] = deck.deal();
        sortHand(hand);

        // If Statement is used to save the current hand after the card selection to the queue if it is a Single-Player game.
        if(isSingleplayer) for (Card card : hand) REPLAY.enqueue(card);
        score = getScore(hand);
    }

    // This is used to print out the final score for this round.
    System.out.println("\n******************");
    System.out.println("Streak Value is " + score);
    System.out.println("******************");

    // Once the For Loop has completed or broken, the final score for the round has been found and will then return this value.
    return score;
}
```

```java
// --- GET / SCORE CALCULATION METHODS ---
// This method is used to find the score given an array of cards, it will find the highest streak and add any bonuses.
public static int getScore(Card[] cards) {
    int highestStreak = 0, highestBonus = 0;   // This will initialise two integers in order to keep track of the last highest streak found alongside its bonus.

    // For Loop is used in order to find the length of each streak in the card array.
    for (int currentPos = 0; currentPos < cards.length - 1; currentPos++) {
        int currentStreak = getStreak(cards, currentPos);   // This will call the getStreak method to find the length of the streak starting on the card given.

        // If Statement is used to determine if the current streak is greater than or equal to the last highest streak found.
        if (currentStreak >= highestStreak && currentStreak != 1) {
            // THIS IS DEBUGGING MESSAGE.
            // System.out.println("This streak is greater than or equal to the last streak found, calculating bonus.");

            int currentBonus = getBonus(cards, currentPos, currentStreak);  // This will call the getBonus method to find the amount of bonuses to award.

            // If Statement is used set new values to the highestStreak and highestBonus if the currentStreak is greater than the highestStreak.
            if (currentStreak > highestStreak) {
                highestStreak = currentStreak;
                highestBonus = currentBonus;
            }

            // Else If Statement is needed when the currentStreak and highestStreak are equal, it will determine which streak has the highest value by comparing bonus points.
            else if ((currentStreak + currentBonus) > (highestStreak + highestBonus)) highestBonus = currentBonus;
        }
        // This will set the current position to scan for a streak to be the card after the last streak found.
        currentPos = currentPos + currentStreak - 1;
    }
    // Once the For Loop has completed, the highest streak will have been found it will then return this value plus its corresponding bonus.
    return highestStreak + highestBonus;
}

// This method is used to find the total length of the streak given a Card array and the position of the first card in the streak.
public static int getStreak(Card[] cards, int startingPos) {
    int nextPos = startingPos + 1;  // This initialises the value of nextPos to used to find whether the card in the nextPos is a streak.
    int streak = 1;                 // This initialises the streak to 1 as there is currently only one card in the streak.

    // If Statement is used to find if the first card is a streak with the card in the nextPos.
    if (cards[startingPos].isStreak(cards[nextPos])) {
        // THIS IS A DEBUGGING MESSAGE.
        // System.out.println("\nFirst card in the streak is " + cards[startingPos]);

        // For Loop will increment the value of currentPos while it is less than the length of the Card array given - 1 to find the total length of the streak.
        for (int currentPos = startingPos; currentPos < cards.length - 1; currentPos++) {
            nextPos = currentPos + 1;   // This sets the value of nextPos to the position after the currentPos

            // If the card in the currentPos is not a streak with the card in the nextPos the For Loop will be broken.
            if (!cards[currentPos].isStreak(cards[nextPos])) break;

            // THIS IS A DEBUGGING MESSAGE.
            // System.out.println("Next card in the streak is " + cards[nextPos]);
            streak++;
        }
        // THIS IS A DEBUGGING MESSAGE.
        // System.out.println("It has a total streak length of " + streak);
    }
    return streak;
}
```

```java
    // This method will return the total bonus of a streak given the Card array, the starting position of the streak and the length of the streak.
    public static int getBonus(Card[] cards, int startingPos, int streakLength) {
        boolean colourBonus = true, suitBonus = true;   // This initialises the currentBonus and suitBonus to true as they have not yet been found.
        int endingPos = startingPos + streakLength - 1; // This initialises the value of the endingPos to be used in the For Loop.
        int bonus = 0;                                  // This initialises the value of bonus to zero as no bonuses have been found yet.

        // For Loop will increment the value of currentPos while it is less than the endingPos to find if a bonus is applicable to the streak.
        for (int currentPos = startingPos; currentPos < endingPos; currentPos++) {
            int nextPos = currentPos + 1;   // This initialises the value of nextPos to used to find whether the card in the nextPos has a bonus.

            // If the card in the currentPos is not a bonus with the card in the nextPos it will set its bonus boolean to false.
            if (!cards[currentPos].isSuitBonus(cards[nextPos])) suitBonus = false;
            if (!cards[currentPos].isColourBonus(cards[nextPos])) {
                colourBonus = false;
                break;  // This will break the For Loop as the streak is not of the same colour and is therefore not of the same suit.
            }
        }

        // If Statements are used to apply the bonuses if they are both found to be true after the For Loop has been completed.
        if (colourBonus) {
            // THIS IS A DEBUGGING MESSAGE.
            // System.out.println("All the cards in this streak are of the same colour. (" + cards[startingPos].getColour() + ")");
            bonus++;
        }
        if (suitBonus) {
            // THIS IS A DEBUGGING MESSAGE.
            // System.out.println("All the cards in this streak are of the same suit. (" + cards[startingPos].getSuit() + ")");
            bonus++;
        }
        return bonus;
    }



    // --- MAIN / MENU METHOD ---
    // The main method is used to initialise the scanner and display the menu on a loop until the user
    public static void main(String[] args){
        Scanner input = new Scanner(System.in); // Initialises the scanner to scan user input throughout the different methods.
        boolean menu = true;                    // Initialises a boolean value to be used to control if the menu should repeat.

        // This will constantly display the menu method until the user chooses to exit the program.
        while(menu) {
            System.out.println("\n        STREAK        ");
            System.out.println("----------------------");
            System.out.println("1. Single player game");
            System.out.println("2. 2-player game");
            System.out.println("3. View Hi-Score Table");
            System.out.println("9. Exit");

            String[] digits = {"1", "2", "3", "9"}; // This is used to tell the choice method what choices are available to choose from.
            // Switch Case will call the validateChoice method to determine and validate the users choice and carry out the case that corresponds with prior input.
            switch (validateChoice(input, "menu", digits, 0)) {
                case 1 -> playSingleplayer(input);            // This will call the playGame method to carry out a Single-Player game.
                case 2 -> play2Player(input);                 // This will call the playGame method to carry out a 2-Player game.
                case 3 -> displayScoreboard();                // This will call the displayScoreboard method to print out the top 5 scores in the SCOREBOARD.
                case 9 -> menu = false;                       // This will set menu to false so that it will no longer display the menu.
            }
        }
        System.out.println("\n+++++ GOODBYE +++++");
    }
}
```