```
/**********************************************************************************************
 * Project: Poker - Game Final                                                                *
 * Name: Chris Singleton                                                                      *
 * Date: 12/1/2016                                                                            *
 **********************************************************************************************
 *                                                                                            *
 * Summary (v.1):                                                                             *
 *          1. Create a parent class, four child classes, two enum classes,                   *
 *             and a class that does the work (All in the Library Project).                   *
 *                                                                                            *
 *          2. The four Classes (CardDiamonds, CardHeart, CardClub and CardSpade), are all    *
 *             the same with respect to code and override the parent SuperCard's Suit directly. *
 *                                                                                            *
 *          3. The CarSet Class iterates through each card from 2 to 14 per set and then adds to *
 *             the card to the array index using a while loop that stops after 52 times.     *
 *                                                                                            *
 *          4. Instatiate an object in the Program class project called myDeck and write the  *
 *             iteration out directly to the Console using a writeline.                       *
 *                                                                                            *
 *          5. During the write out process I am using cardsRank from the SuperCard Class and using *
 *             the child classes of each suit (overriding the parent directly) in showing cardSuit. *
 *                                                                                            *
 *             Note: The object myDeck captures each card from cardArray's array index for the each *
 *                   cardsRank and cardSuit while printing 52 for loop iterations out.        *
 *                   (Looping through each index in the cardArray for print out.)             *
 *                                                                                            *
 * Changes (v.2):                                                                             *
 *          6. Not All 52 cards display on the screen all at once anymore (Omitted for loop). *
 *                                                                                            *
 *          7. Super[] CardArray is no longer public. Instead CardArray uses an indexer in CardSet *
 *             to get the CardArray[index].                                                   *
 *                                                                                            *
 *          8. Set and initialize howManyCards to 5 for GetCards and initialized balance to 10. *
 *                                                                                            *
 *          9. Added a method called DisplayHands which  writes out the computer and players cards *
 *             from inside the SuperCard cardArray pointer called index (all five positions). *
 *             Note: The Child classes (CardClub, CardDiamond, CardHeart, and CardSpade overrides *
 *                   the parent class SuperCard that is using abstact which returns _cardSuit.) *
 *                                                                                            *
 *         10. Instantiated a Random Generator Object called rnd and Created a type SuperCard array *
 *             GetCards method that randomly gets 5 cards out of the 52 cards.               *
 *                                                                                            *
 *         11. Created a method called CompareHands which adds each index of the SuperCard array *
 *             for the computer's hand and player's hand using a for loop, then compares who won *
 *             and then decreases or adds to the balance based on the argument that is true.  *
 *                                                                                            *
 * Changes (v.3):                                                                             *
 *         12. Added an Icomparable interface to the SuperCard parent class.                  *
 *                                                                                            *
 *         13. Added methods Array.Sort(computerHand) and Array.Sort(playersHand) to the program console. *
 *                                                                                            *
 *         14. Added a CompareTo method in the SuperCard class that sorts the cards by suit and rank. *
 *                                                                                            *
 * Changes (Final):                                                                           *
 *         15. Add a Get/Set Property of type bool called "inplay" and other Get/Set properties. *
 *                                                                                            *
 *         16. Added Methods Flush, PlayerDrawsOne, ComputerDrawsOne in the Program.cs        *
 *                                                                                            *
 *         17. Incorporated IEquatable Interface to compare bool objects.                     *
 *                                                                                            *
 *         18. Created ResetUsage and GetOneCard Methods.                                     *
 *                                                                                            *
 *         19. Called necessary methods to replace the cards in the hands (computer and player). *
 *                                                                                            *
 *         20. Included arguments for the functionality to work correctly.                   *
 *                                                                                            *
 *         21. Created a while loop to reprompt user of invalid input and handled an exception as necessary.*
 *                                                                                            *
 *         22. Added a class called Welcome with a method called WelcomeMessage.              *
 *                                                                                            *
 *         23. Gave the user a choice to exit or continue after each round with a prompt ('x' to exit). *
 *                                                                                            *
 **********************************************************************************************/
using System;
using CardLibrary;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ChrisSingleton_Poker
{
    class Program
    {
        public static void Main()
        {
            /* Display an introduction to the game while changing the foreground color,
               background color, then initialize the variables for the balance and number of cards
               and window size. */
            // Call the welcome message that lets the user chose how many cards.
            Welcome.WelcomeMessage();
            // Encode fix to see Unicode.
            Console.OutputEncoding = Encoding.Unicode;
            Console.ForegroundColor = ConsoleColor.Red;
            int howManyCards = 0;
            int balance = 10;
            // Change the Window Size.
            Console.WindowWidth = 95;
            Console.WindowHeight = 45;
            // Give the user the ability to change how many cards and prompt for invalid entry.
            while (howManyCards < 2 || howManyCards > 8)
            {   try
                {
                    Console.ForegroundColor = ConsoleColor.Magenta;
                    howManyCards = int.Parse(Console.ReadLine());
                    Console.ResetColor();
                }
                catch
                {
                    Console.ForegroundColor = ConsoleColor.Red;
                    Console.Write("Invalid entry! \n");
                    Console.ResetColor();
                }
                if (howManyCards < 2 || howManyCards > 8)
                {
                    Console.ForegroundColor = ConsoleColor.Green;
                    Console.Write("Please try again (integer greater than 1 and less than 8): ");
                    Console.ResetColor();
                }
            }

            /* Continue the loop until balance equals zero.
               Instantiate an Object myDeck to the CardSet type. Then call GetCards(howManyCards)
               method with myDeck into the SuperCard type array of the computer and player.
               Display who won while subtracting the balance from the loser and addit to the winner.
               Finally, show the balance and let them continue until zero balance. */
            while (balance != 0)
            {
                Console.Clear();
                CardSet myDeck = new CardSet(); // Instantiate the object myDeck.
                myDeck.ResetUsage(); // Reset the cards back to false (not in use).
                SuperCard[] computerHand = myDeck.GetCards(howManyCards); // Call: Get the computerHand.
                SuperCard[] playersHand = myDeck.GetCards(howManyCards); // Call: Get the playersHand.
                // Sort the Cards.
                Array.Sort(computerHand);
                Array.Sort(playersHand);
                DisplayHands(computerHand, playersHand); // Call to Display the Computer hand.
                ComputerDrawsOne(computerHand, myDeck); // Call for Computer to Draw One Card.
                PlayerDrawsOne(playersHand, myDeck); // Call for Player to Draw One Card.
                // Sort the Cards.
                Array.Sort(computerHand);
                Array.Sort(playersHand);
                DisplayHands(computerHand, playersHand);// Call to Display the Player's hand.
                bool won = CompareHands(computerHand, playersHand);
                if (!won) // Player did not win.
                {
                    balance--;
                    Console.ForegroundColor = ConsoleColor.Red;
                    Console.WriteLine("\n\nYou lost!");
                    Console.ResetColor();

                    // When you run out of money: Game Over.
                    if (balance == 0)
                    {
                        Console.ForegroundColor = ConsoleColor.Red;
                        Console.WriteLine("\n\n      Your out of money - Game Over! \n\n
                                        ...Please Press any key to exit!");
                        Console.ResetColor();
                        break; // Break's out of the loop.
                    }
                }

                else // Player wins.
                {
                    balance++;
                    Console.ForegroundColor = ConsoleColor.Green;
                    Console.WriteLine("\n\nYou win!");
                    Console.ResetColor();
                }
                // Write out the balance of Cash with a message.
                Console.Write("Your balance is : ");
                Console.ForegroundColor = ConsoleColor.Green;
                Console.Write("{0:C}", balance);
                Console.ResetColor();
                // Give the options to exit, restart or continue.
                Console.Write("\n\nTYPE 'x' TO END THE GAME, OR ANY OTHER KEY TO SEE ANOTHER HAND: ");

                // Option for the player to exit the game.
                if (Console.ReadKey().KeyChar == 'x')
                {
                    Console.ForegroundColor = ConsoleColor.Red;
                    Console.Write("\n\n      Game Over! \n\n              ...Please Press any key to exit!");
                    Console.ResetColor();
                    break; // Break's out of the loop.
                }
                Console.Clear(); // Clear the screen.
            }
            Console.ReadKey(); // Pause the console.
        }
        // Determine if the computer or player has a flush (all cards are one Suit).
        private static bool Flush(SuperCard[] hand)
        {
            bool allSameSuit = false;
            for (int i = 1; i < hand.Length; i++)
            {
                if (hand[i].cardSuit.Equals(hand[0].cardSuit))
                { allSameSuit = true; }
                else
                { allSameSuit = false; break;}
            }
            return allSameSuit;
        }

        /* Compare the hands of the computer and player. The for loop adds each index inside the array
           of the players and then appends them to the variable. Then compares the player score against the
           computer score. Finally, if the player is not greater, then returns false (computer wins).
           Note: This means that even if there is a tie, then computer wins. */
        public static bool CompareHands(SuperCard[] computerHand, SuperCard[] playersHand)
        {   int computerScore = 0;
            int playerScore = 0;
            bool playersFlush = !Flush(computerHand) && Flush(playersHand);
            bool computersFlush = Flush(computerHand);
            for (int i = 0; i < computerHand.Length; i++)
            {   computerScore += (int)computerHand[i].cardsRank;
                playerScore += (int)playersHand[i].cardsRank;
            }
            if (playersFlush)
            {
                Console.ForegroundColor = ConsoleColor.Green;
                Console.WriteLine("Player has a flush!");
                Console.ResetColor();
            }
            else if (computersFlush)
            {
                Console.ForegroundColor = ConsoleColor.Yellow;
                Console.WriteLine("Computer has a flush!");
                Console.ResetColor();
            }
            Console.ResetColor();
            Console.Write("\nCOMPUTER'S SCORE:  ");
            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.Write(computerScore);
            Console.ResetColor();
            Console.Write("     \nPLAYER'S SCORE:    ");
            Console.ForegroundColor = ConsoleColor.Magenta;
            Console.Write(playerScore);
            Console.ResetColor();
            // Sets the condition to see who got a flush.
            if (((playerScore > computerScore) && (!computersFlush)) || playersFlush)
            { return true; }
            else
            { return false; }
        }


           /* Writes computer hand and players hand out to console.
              Displays the hand for the computer and player using two foreach loops
              while calling items (5 cards) inside the array (inside, each card suit and rank). */
           public static void DisplayHands(SuperCard[] computerHand, SuperCard[] playersHand)
           {
               //============ Computer Hand: ===================
               Console.ForegroundColor = ConsoleColor.Yellow;
               Console.WriteLine("\nCOMPUTER'S HAND: ");
               Console.ResetColor();

               foreach (SuperCard item in computerHand)
               {
                   item.Display(); // Display all the computer's cards.
               }

               //============ Player's Hand: ===================
               Console.ForegroundColor = ConsoleColor.Magenta;
               Console.WriteLine("\nPLAYER'S HAND: ");
               Console.ResetColor();

               foreach (SuperCard item in playersHand)
               {
                   item.Display(); // Displays all the player's cards.
               }
           }
        //
        // Note, myDeck is of typeof CardSet, not SuperCard.
        public static void PlayerDrawsOne(SuperCard[] myHand, CardSet myDeck)
        {
            int playerChose = 53; // Flag, Set the player's choice way beyond the range of cards (un-achievable).
            // Set the condition for range that a player can choose with changing a card.
            while (playerChose != 0 && playerChose > myHand.Length)
            {   Console.ForegroundColor = ConsoleColor.Red;
                Console.WriteLine("\nWhich card would you like to replace? ");
                Console.ResetColor();
                Console.ForegroundColor = ConsoleColor.Green;
                Console.Write("Please type 0 for None, or up to how many cards!: ");
                Console.ResetColor();
                try // If the player does not choose a valid entry (try again).
                {
                    Console.ForegroundColor = ConsoleColor.Magenta;
                    playerChose = int.Parse(Console.ReadLine());
                    Console.ResetColor();
                }
                catch
                {
                    Console.ForegroundColor = ConsoleColor.Red;
                    Console.WriteLine("Error, please enter a number! (Inclusive)");
                    Console.ResetColor();
                }
                // if user wants to replace a card.
                if (playerChose != 0 && playerChose <= myHand.Length)
                {
                  myHand[playerChose - 1] = myDeck.GetOneCard(); // Calls GetOneCard moves into the array myHand.
                }
                else if (playerChose == 0)
                { // Do Nothing and continue playing. }
                else
                {
                    Console.ForegroundColor = ConsoleColor.Red;
                    Console.WriteLine("Invalid Choice! Your number should be between 0 and how many cards!");
                    Console.ResetColor();
                }
            }
        }

        /* Find the lowest number in the SuperCard[] computerHand and replace it with a new card.
           Get the smallest card by looping through the length of the array.Compare the smallest card
           to the rest of the cards in the array.Change the card on the computer hand only if the card
           is less than 8 and there is no flush.*/
        public static void ComputerDrawsOne(SuperCard[] computerHand, CardSet myDeck)
        {
            int smallestCard = (int)computerHand[0].cardsRank;
            int computerHandIndex = 0;

            //
            for (int i = 0; i < computerHand.Length; i++)
            {
                // Loop through the length of the array each time finding the smallest card.
                if (smallestCard > (int)computerHand[i].cardsRank)
                {
                    computerHandIndex = i;
                    smallestCard = (int)computerHand[i].cardsRank;
                }
            }

            // Condition that allows you to draw a card (less than 8, no flush).
            if (smallestCard < 8 && Flush(computerHand) == false)
            {
                computerHand[computerHandIndex] = myDeck.GetOneCard();
            }

        }

    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CardLibrary
{
    public static class Welcome
    {
        // Give's the welcome message with choice of how many cards.
        public static void WelcomeMessage()
        {
            Console.OutputEncoding = Encoding.Unicode;
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine();
            for (int i = 0, m = 1; i < 30; i++)

                for (int l = 0; l < new[] { 5, 6, 7, 6, 8, 10, 3, 10, 4, 13, 1, 13, 1, 87, 1, 27, 4, 2
                                     7, 20, 11, 16, 16, 11, 20, 7, 24, 3, 27, 1 }[i]; l++, m++)

            Console.Write((i % 2 > 0 ? "♥♣♦♠"[m % 4] : ' ') + (m % 29 > 0 ? "" : "\n"));
            Console.ResetColor();
            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.BackgroundColor = ConsoleColor.DarkBlue;
            Console.Write("\n\nWelcome to the Poker Game!\nYou have $10.00 to start
                           (*Each round places a bet at $1.00).\n");

            Console.WriteLine("\n        Rules: "+
                                "\n        *Who ever gets a flush without being a tie Wins!"+
                                "\n        *Dealer always wins in a tie, even if both get a flu
                                "\n        *Highest card count always wins (when there is no fl

            //Allow the player to chose how many cards they want dealt.
            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.Write("\nPlease put in the amount of cards you want dealt each time: ");
            Console.ResetColor();
        }
    }
}

using System;
using CardLibrary;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CardLibrary
{
    public class CardSet
    {   // Instantiate an object called rnd at random.
        Random rnd = new Random();
        SuperCard[] cardArray;
        // Indexer with get. Get's the cardArray.
        public SuperCard this[int index]
        {
            get
            {
                return cardArray[index]; // Get's each index randomly out of the 52 indexes for each c
            }
        }
        /* Loop through all 52 cards and add to the cardArray for each suit. */
        public CardSet()
        {
            cardArray = new SuperCard[52];
            int index1 = 0;
            while(index1 < 52)
            {
                  for(int index2 = 2; index2 <= 14; index2++) // Iterates through each card.
                      cardArray[index1++] = new CardClub((Rank)(index2)); // Adds into the index of
                  for (int index2 = 2; index2 <= 14; index2++)
                      cardArray[index1++] = new CardDiamond((Rank)(index2));
                  for (int index2 = 2; index2 <= 14; index2++)
                      cardArray[index1++] = new CardHeart((Rank)(index2));
                  for (int index2 = 2; index2 <= 14; index2++)
                      cardArray[index1++] = new CardSpade((Rank)(index2));
            }
        }

        /* Loop through up to five times while taking out a random card,
           move into the variable temp. Check to see if the temp is not
           in play, then count it as in play, add the card to Cards array
           and then set the temp array as true (cards being used).
           Finally return the Cards. Note: number corresponds to five (Cards). */
        public SuperCard[] GetCards(int number)
        {
            SuperCard[] Cards = new SuperCard[number];
            for (int i = 0; i < number; i++) {
                Cards[i] = GetOneCard();
            }
            return Cards; // Return all five cards in the Card array.
        }
        /*Loop through to get a random card that is not inplay.
          Then take the lowest number card out and replace it with a random card.*/
          // Just returns one random card.
        public SuperCard GetOneCard()
        {   // Loop through 0 - 52 and check if the card is in play. If not, then return the card.
            int temp;
            do
            {
                temp = rnd.Next(0, 52); // Get a random number from 0 to 52.
            }
            while (this[temp].inplay); // Check to see if the random number is in play.
            /* This checks to see if the cards are in play. Set the card inplay equal to true
               (doesn't use it again). Same typeof, return one card.*/
            this[temp].inplay = true; //
            SuperCard oneCard = this[temp];
            return oneCard; // Returns the one card.
        }
        // Resets the cards to all false on inplay inside the card array (Each Round).
        public void ResetUsage()
        {
            for(int i = 0; i< 52; i++)
            {
                cardArray[i].inplay = false;
            }
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace CardLibrary
{
    // IComparable (Compares cards) and IEquatable (compares objects to one another) interfaces.
    public abstract class SuperCard : IComparable<SuperCard>, IEquatable<SuperCard>
    {   // Get/Set the cardsRank, but let the Children override.
        public Rank cardsRank { get; set; }
        public abstract Suit cardSuit { get; }
        public bool inplay { get; set; }
        public string Name { get; set; }
        // Check to see if the cards are equal to one another.
        public  bool Equals(SuperCard otherCard)
        {
            if (this.Name == otherCard.Name)
            { return true; }
            else
            { return false; }
        }
        // Added Method to display from each card class.
        public abstract void Display();
        // Compare the cards. If other is not a valid object reference, this instance is greater.
        public int CompareTo(SuperCard other)
        {
            if (other == null) return 1; // Put all the valid objects at the top of the array.
            if (other.cardSuit < this.cardSuit)
            { return 1; // "I'm bigger that the one you passed in. }
            else if (other.cardSuit == this.cardSuit)
            {
                if (other.cardsRank > this.cardsRank)
                { return -1; }
                return 0;  // Both have the same Suit.
            }
            return -1;  // the one passed it is bigger than I am.
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CardLibrary
{
    public class CardClub: SuperCard
    {
        // Field _cardsuit that has type Suit for the Clubs.
        private Suit _cardsuit = Suit.Club;

        // Constructor CardClub that takes in the enum Rank as cards_Rank.
        public CardClub(Rank cards_Rank)
        {
            this.cardsRank = cards_Rank;
        }

        // Constructor that returns _cardsuit and overrides SuperCard's cardSuit.
        public override Suit cardSuit
        {
            get
            {
                return _cardsuit;
            }
        }

        // Display a club card, change the colors for club cards.
        public override void Display()
        {
            Console.BackgroundColor = ConsoleColor.White;
            Console.ForegroundColor = ConsoleColor.Blue;
            Console.WriteLine("{0} of {1}s ♣", cardsRank, _cardsuit);
            Console.ResetColor();
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CardLibrary
{
    public class CardDiamond : SuperCard
    {
        // Field _cardsuit that has type Suit for the Diamond.
        private Suit _cardsuit = Suit.Diamond;

        // Constructor CardDiamond that takes in the enum Rank as cards_Rank.
        public CardDiamond(Rank cards_rank)
        {
            this.cardsRank = cards_rank;
        }

        // Constructor that returns _cardsuit and overrides SuperCard's cardSuit.
        public override Suit cardSuit
        {
            get
            {
                return _cardsuit;
            }
        }
        // Display a Diamond card.
        public override void Display()
        {
            Console.BackgroundColor = ConsoleColor.White;
            Console.ForegroundColor = ConsoleColor.DarkRed;
            Console.WriteLine("{0} of {1}s ♦", cardsRank, _cardsuit);
            Console.ResetColor();
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CardLibrary
{
    public class CardHeart: SuperCard
    {
        // Field _cardsuit that has type Suit for the Heart.
        private Suit _cardsuit = Suit.Heart;


        // Constructor CardHeart that takes in the enum Rank as cards_Rank.
        public CardHeart(Rank cards_Rank)
        {
            this.cardsRank = cards_Rank;
        }

        // Constructor that returns _cardsuit and overrides SuperCard's cardSuit.
        public override Suit cardSuit
        {
            get {
                return _cardsuit;
                }
        }

        // Display a Heart card.
        public override void Display()
        {
            Console.BackgroundColor = ConsoleColor.White;
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine("{0} of {1}s ♥", cardsRank, _cardsuit);
            Console.ResetColor();
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CardLibrary
{
    public class CardSpade: SuperCard
    {
        // Field _cardsuit that has type Suit for the Spade.
        private Suit _cardsuit = Suit.Spade;

        // Constructor CardSpade that takes in the enum Rank as cards_Rank.
        public CardSpade(Rank cards_Rank)
        {
            this.cardsRank = cards_Rank;
        }

        // Constructor that returns _cardsuit and overrides SuperCard's cardSuit in this class.
        public override Suit cardSuit
        {
            get
            {
                return this._cardsuit;
            }
        }

        // Display a Spade card while changing the colors for Spades.
        public override void Display()
        {
            Console.BackgroundColor = ConsoleColor.White;
            Console.ForegroundColor = ConsoleColor.Black;
            Console.WriteLine("{0} of {1}s ♠", cardsRank, _cardsuit);
            Console.ResetColor();
        }
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace CardLibrary
{   // Enumeration for all cards that will have iteration in the for loops (CardSet).
    public enum Rank
    {
        Deuce = 2,
        Three = 3,
        Four = 4,
        Five = 5,
        Six = 6,
        Seven = 7,
        Eight = 8,
        Nine = 9,
        Ten = 10,
        Jack = 11,
        Queen = 12,
        King = 13,
        Ace = 14
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace CardLibrary
{   /* Enumeration that has all suits through iterations in the for loops.
       Note: Suits have four for loops in the CardSet Class (One for each).*/
    public enum Suit
    {   Club = 1,
        Diamond,
        Heart,
        Spade
    }
}
```