COSC 2100 – Assignment 4

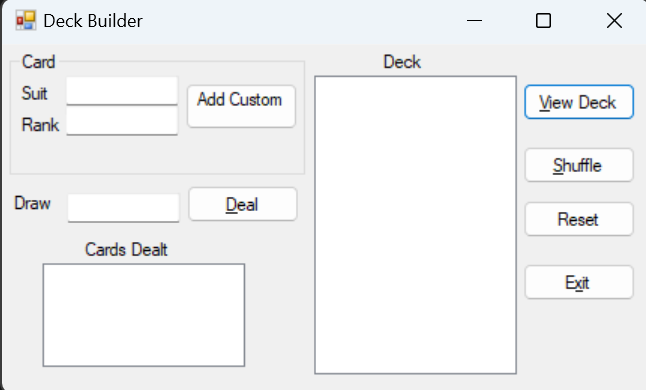
This assignment is an individual assignment. Submit your work via DC Connect by the due date provided.

# Client Requirements

Create a **simple deck of cards** application using C#.

This assignment focuses on creating a deck of cards program, incorporating inheritance, and working with collections to manage a deck. The assignment requires you to build a program that models the behavior of a deck of playing cards, shuffling, dealing, and customizing the deck with additional cards.

# Approximate Design



# Instructions from the UX (User Experience) Department

* The form should not allow minimizing or maximizing the screen
* The form’s title bar text should say something meaningful
* The form should have its AcceptButton (ViewDeck) and CancelButton (Reset) set appropriately
* The form should open in the centre of the screen
* Where possible, all interactive controls (including TextBoxes) should have access keys (hotkeys) set up.
* All controls should have their tab indexes set in a useful way
* All controls that the user interacts with should have a tooltip that explains each control’s purpose

# General Operation

* An user can exit by:
  + Clicking the Exit button with a mouse cursor
  + Using an access key to activate the Exit button
  + Tabbing to the Exit button and tapping Enter on the keyboard
* An user can reset the application by:
  + Clicking the Reset button with a mouse cursor
  + Pressing the Esc key on the keyboard
  + Using an access key to activate the Reset button
  + Tabbing to the Reset button and tapping Enter on the keyboard
* When the Add Custom button is used:
  + The user can enter data by:
  + Clicking the Add button with a mouse cursor
  + Pressing the Enter key on the keyboard
  + Using an access key to activate the Enter button
  + If the entered data is invalid, validation messages should be displayed in a MessageBox or error tooltip.
  + If the entered data is valid, all input fields should be cleared, and the new cards data should be added to the ListView control for the deck, along with any prior existing cards.
* When the Deal button is used:
  + The Draw input will pull x cards from the deck
  + The ListView control for the cards dealt will display the pulled cards
* When the View Deck button is used:
  + Will display all cards in the deck in the ListView control for the deck.
* When the Shuffle button is used:
  + Will randomise the deck of cards.
  + Will display all cards in the deck in the ListView control for the deck in their new random order.
* When the Exit button is used:
  + Ends the application.
  + All input controls will clear or be set to their default state.

# Instructions from the Business Analyst/Tech Lead

* For input to be considered valid for a custom card:
  + A Suit must be entered; this field cannot be blank or filled only with spaces.
  + A Rank must be entered; this field cannot be blank or filled only with spaces.
* For input to be considered valid to draw from the deck:
  + An amount > 0 and < the total amount of cards remaining in the deck must be entered; this field cannot be blank or filled only with spaces.
* Validation should take place in a function that takes the input fields as parameters and returns True if all of these fields are valid and False if any of these fields are invalid.
* For each field that is invalid when the function is called, an error message specific to that field should be displayed in a MessageBox.
* There must be a function (method) to reset the form's input fields to their default state. This function will be triggered when either the Exit button is pressed or when the Reset button is pressed.
* A class must be defined representing a Card, a Deck, a StandardDeck and a CustomDeck; code reusability is something to consider.
* The class must be defined in a separate file (likely Card.cs, Deck.cs, StandardDeck,cs and CustomDeck.cs).
* Card Information:
  + Suit (e.g., Hearts, Diamonds, Clubs, Spades)
  + Rank (e.g., Ace, 2, 3, ..., King)
* Deck Information:
  + A deck should consist of multiple cards with various suits and ranks.
* Card Class:
  + Create a base Card class with properties for Suit and Rank.
  + Override the ToString() method to display card information.
* Deck Class:
  + Implement a Deck class to manage a collection of cards.
  + Initialize the deck with a full set of 52 cards.
* StandardDeck Class (Inherits from Deck):
  + Create a StandardDeck class that inherits from the base Deck class.
  + In the StandardDeck, initialize the deck with standard playing cards.
* CustomDeck Class (Inherits from Deck):
  + Create a CustomDeck class that inherits from the base Deck class.
  + The CustomDeck allows adding custom cards to the deck.
* Functionality:
  + Implement methods for shuffling the deck, displaying the deck, and dealing cards.
* Methods: There will only be one method for the Card class
  + Override the ToString() method to display card information
* Ensure that your design incorporates suitable exception handling.

# Development Tips

# Additional Considerations

* The program must be adequately documented:
  + methods, functions and event handlers should all have block comments
  + calculations, decisions and iteration should be explained with brief comments
  + there should be a header at the top of each human-generated code files, including your name, the last modified date, and a description
* Adhere to an approved style guide and ensure your variable names, form controls, and other elements are properly cased and adequately descriptive.

# Assessment

Each student’s work will be submitted individually as a compressed folder containing the entire solution. The instructor will assess Your completed work using the COSC2100 Assignment 4 Rubric available on DC Connect. Reasonably detailed feedback will be provided. For the Specific Requirements portion of the rubric, the most important criteria are:

• A Windows Form created in .NET 6 or .NET 7

• Customers can be entered into the list

• Selecting a customer in the list populates the input fields

• A class (in a separate code file) has been implemented

• A GetServiceData () or equivalent function has been created as part of the class

• A minimum of one-third in the Functionality section of the rubric.