Lab 1:
Poker Face

Due February 6 by the start of lecture.

Overview

In this lab, you will learn how to set up a new .NET Core solution in your editor and then write a simple class that classifies poker hands.

Creating the Solution

Go to BeachBoard's Content: Videos section and follow the link titled **Creating a C# Solution...** for your operating system. Follow the directions in the video with the following changes:

- Instead of **Cecs475.Demo.Library**, name the first library project **Cecs475.Poker.Model**
- **Do not** make a .NET Core application (Console App) project in the **src** folder (what the video calls **Cecs475.Demo.App**)
- Instead of **Cecs475.Demo.Test**, name the final project **Cecs475.Poker.Test**

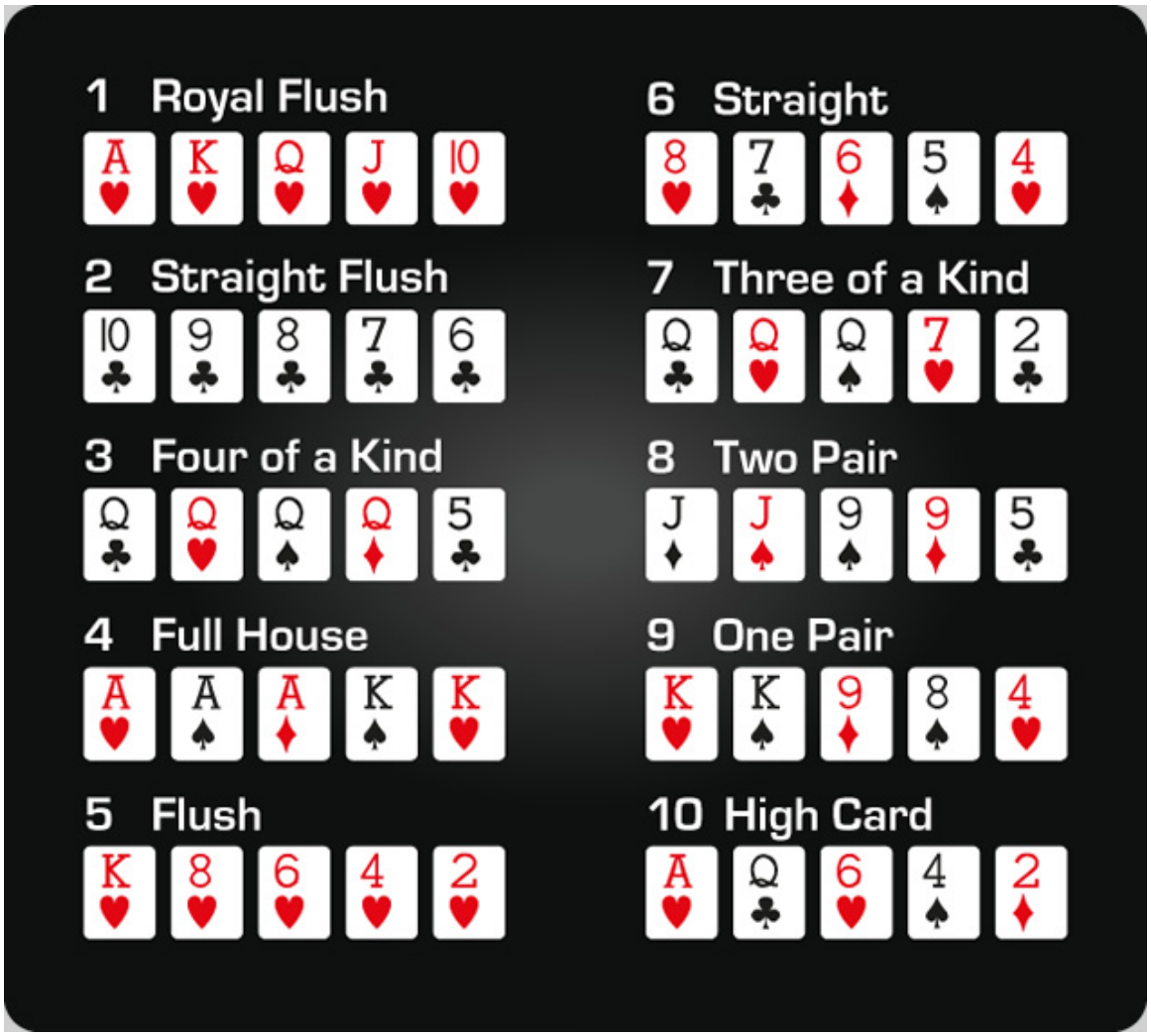
Starter Files

Find the file `Card.cs` from the introductory lectures on GitHub. Copy that file to your `src/Cecs475.Poker.Model` directory, which will cause it to show up in Visual Studio.

Writing the Poker Hand Model

Create a new file in the Model project called `PokerHand.cs`. Complete this class to the following specification:

1. Namespace: `Cecs475.Poker.Cards`.
2. Class name: `PokerHand`.
3. Create a public enum inside `PokerHand` named `HandType`. It must have 10 members representing the 10 types of poker hands, named `HighCard`, `Pair`, `ThreeOfKind`, `FourOfKind`, `TwoPair`, `Flush`, `FullHouse`, `Straight`, `StraightFlush`, and `RoyalFlush`. The enum members **must** be arranged in increasing order, such that the worst hand type is first in the enum and the best hand type is last. Here is a picture:



4. Properties:
   (a) an auto property of type `List<Card>` representing the "hand" of cards. This property should be **read-only**, and should be **sorted in increasing card value order**. (The opposite order as shown in the image above.)
   (b) an auto property of type `HandType`, representing what "kind" of hand is represented. This property should be **read-only**.
5. A constructor:
   (a) takes a list of cards and a `HandType`.
   (b) initializes the `HandType` property to equal the given parameter.
   (c) intitializes the `List<Card>` property to be a **copy** of the given parameter list, and then sorts the property as described above.
6. Implements the `IComparable<PokerHand>` interface, and the method `public int CompareTo(PokerHand other)`.
   (a) The primary comparison between two poker hands is based on their `HandType`.
   (b) If the two `HandTypes` are equal, then the comparison is made by examining the hands' cards in **decreasing** order. Comparing cards in equivalent positions one at a time, the larger hand is determined by the first card position that is different between the two hands. For example, if

the High Card hand from the image is compared against a hand of **"Ace of Spades, Queen of Diamonds, 5 of Hearts, 4 of Clubs, 2 of Clubs"**, then the hand in the image is "larger"; in decreasing order, both hands have an Ace, both have a Queen, but the second hand has a 5 whereas the first has a 6, so the first wins.

Writing the Poker Hand Classifier

Create a new file in the Model project called `PokerHandClassifier.cs`. Complete this class to the following specification:

1. Namespace: `Cecs475.Poker.Cards`.
2. Class name: `PokerHandClassifier`.
3. A single method:
   (a) a static method `ClassifyHand`, taking an `IEnumerable<Card>` parameter and returning a `PokerHand` object representing the type of hand represented. After examining the cards, constructs a `PokerHand` object (see the constructor you wrote earlier) and returns that object.

   This method **cannot modify** the list that is passed as a parameter, but it may **create a copy** of that list. (You may find it helpful to sort the list of cards in order to classify the hand.)

Determining a hand's type:

Determining a poker hand's type is fairly simple (assuming you know what the different types of hands are) if you make sure to sort the cards in the hand first. It should then be simple enough to count how many pairs, threes, and fours you have, and if you have a flush or straight. If you are clever, you will note that some hand types are mutually exclusive with others (can you have both a pair and a straight?) and structure your code to take advantage of that.

Validating Your Code

Making sure your code is correct is easy for this assignment:

1. Download the file `PokerHandTests.cs` from BeachBoard's Content: Labs section. Move the file to your `test/Cecs475.Poker.Test` directory.
2. Make sure your **Cecs475.Poker.Test** project has a reference to your **Cecs475.Poker.Core** project, as appropriate to your IDE.
3. Delete whatever `.cs` file was originally added to your project by your IDE (either `Program.cs` or `Tests.cs`).
4. Build the solution and make sure the tests are discovered by the IDE. Run the tests to see if your code is correct and fix any issues that come up.
   (a) Visual Studio Code may have trouble finding the tests. If it does, you can run them on a command line; see the Deliverables section below for how.

Deliverables

Turn in the following when the lab is due:

1. A printed copy of your `PokerHand.cs` and `PokerHandClassifier.cs` code, printed from your IDE whenever possible.
2. A screenshot showing the test results for `PokerHandTests.cs`, either from your IDE, or by running `dotnet test test/cecs475.Poker.Test` from a command prompt at the project's root directory.