Problem Set 4: Video Poker

Class Card
In Video Poker each card has its unique value. No two cards can have the same value. A poker card deck has 52 cards. There are four suits: Club, Diamond, Heart, and Spade. Each suit has thirteen individual cards: 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, and Ace. We define the following order over cards:

Club < Diamond < Heart < Spade
2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < T < Jack < Queen < King < Ace

For example:

"4D" is smaller than "JS", whereas "4S" is greater than "KD".

The ICard interface:

```csharp
public interface ICard : IComparable
{
    // Returns the number (2 ... 14) of a Poker card.
    int Number { get; }

    // Returns the suit of a Poker card.
    // enum CardSuit {Club, Diamond, Heart, Spade}
    CardSuit Suit { get; }

    // Returns the two character name of a Poker card.
    string Name { get; }

    // overridden Equals method
    bool Equals( object aOther );

    // overridden GetHashCode method
    int GetHashCode();

    // Returns a string that represents the current card.
    // For example, Suit = CardSuit.Spade, Number = 10 => "TS"
    string ToString();
}
```
The class **Card** should define two public constructors:

- **public Card( int aNumber, CardSuit aSuit ) { ... }**
  
  This constructor takes an integer and a CardSuit argument
  
  ```
  Card KingOfHearts = new Card( 13, CardSuit.Heart);
  ```

- **public Card( string aName ) { ... }**
  
  This constructor takes one Card-string argument
  
  ```
  Card EightOfDiamonds = new Card( "8D" );
  ```

The class **Card** does not define a default constructor.

**Class Deck**

A Poker deck contains 52 different cards. Since the size of the deck never changes, we can use an array of fixed size to store all cards in the deck. Note, however, that you may need an additional array to arrange the cards in a random order.

```csharp
public interface IDeck : IEnumerable
{
    // Arranges all 52 cards in the deck in a random order.
    void Shuffle();

    // Returns a string representation of the deck. That is, a string
    // containing of 52 card names separated by space.
    string Text { get; }

    // See Text.
    string ToString();
}
```

The class **Deck** defines one public constructor that takes no argument. The purpose of the constructor is to instantiate a Poker deck consisting of 52 cards.
Class Tray

The tray is a container that can hold up to 5 cards. A Tray object is used to transfer cards between different parts of the Video Poker application.

```java
public interface ITray {
    // Adds a set of cards to the tray.
    // May throw InvalidOperationException, if tray is full.
    void AddRange( ICard[] aSetOfCards );

    // Adds one card to the tray.
    // May throw InvalidOperationException, if tray is full.
    void Add( ICard aCard );

    // Removes all cards from the tray.
    void Clear();

    // Returns the number of cards currently stored in the tray.
    int Count { get; }

    // Returns the card at index.
    // May throw ArgumentOutOfRangeException.
    ICard this [int index] { get; }
}
```

The class Tray may define one constructor that takes no argument to set up the internal infrastructure to hold up to five cards.
**Class Hand**
The class **Hand** represents a Poker hand. A hand contains exactly 5 cards. The class **Hand** does not only implement an abstraction to represent a Poker hand, but it also partly defines the Poker game logic, since it defines a method to calculate the score of a given hand.

```csharp
public interface IHand : IComparable
{
    // Returns the score of the current hand.
    int Score { get; }

    // Returns a text representation of the score.
    // For example: “Full House”
    string Title { get; }

    // Returns a string representation of the current hand.
    // For example: “3D 3H 3S KC KS”
    string Text { get; }

    // Returns the Card object indexed by a card name.
    // May throw ArgumentOutOfRangeException if the requested card
    // cannot be found.
    ICard this[string i] { get; }

    // Returns the Card object indexed by its position.
    // May throw ArgumentOutOfRangeException.
    ICard this[int i] { get; }

    // Fills aTray with the cards specified in aHoldString.
    void DiscardCards(ITray aTray, string aHoldString);

    // Replaces the cards specified in aHoldString.
    void ReplaceCards(ITray aTray, string aHoldString);

    // See Text.
    string ToString();
}
```

The class **Hand** defines one public constructor:  
```csharp
public Hand(ICard[] aCardSet)
```

So, to create a new **Hand** object one has to put the cards into an array. The constructor always expects 5 cards. If the constructor is called with fewer cards, the an exception will be thrown (e.g., **ArgumentOutOfRangeException**).
Hand Score Table:

<table>
<thead>
<tr>
<th>Score</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No Score</td>
</tr>
<tr>
<td>2</td>
<td>Jacks or Better</td>
</tr>
<tr>
<td>3</td>
<td>Two Pair</td>
</tr>
<tr>
<td>4</td>
<td>Three of a Kind</td>
</tr>
<tr>
<td>5</td>
<td>Straight</td>
</tr>
<tr>
<td>6</td>
<td>Flush</td>
</tr>
<tr>
<td>7</td>
<td>Full House</td>
</tr>
<tr>
<td>8</td>
<td>Four of a Kind</td>
</tr>
<tr>
<td>9</td>
<td>Straight Flush</td>
</tr>
<tr>
<td>10</td>
<td>Royal Flush</td>
</tr>
</tbody>
</table>

A Poker Scenario

At the beginning of each new Poker round the dealer shuffles the deck and deals out 5 cards to each Player (in the case of Video Poker there is only one player). The dealer has to use a tray to transfer the cards to the corresponding player.

The player receiving the filled tray has to construct a fresh Hand object based of the received cards. Since the class Hand only provides a public constructor with the following signature: public Hand( ICard[] aCardSet ), the cards in the tray have to be taken out of it and put in an array without changing the order. The constructor stores the cards in the order they have been dealt (i.e., the external visible order of the cards in the Hand object has to remain the same at all times).

Now, the player can chose to discard none, some, or all of these cards in an attempt to improve the hand’s score. That is, the player specifies a hold string consisting of the positions of the cards (1 ... 5) and calls the Hand’s method DiscardCards. This method will fill the tray with the selected cards. The player uses the tray to request replacement cards from the dealer.

Once the dealer receives the tray, the dealer will replace the discarded cards with new cards and return the tray to the player. The player will use the cards in the tray to substitute the discarded cards.

The class Hand provides two methods to replace cards: DiscardCards and ReplaceCards. Both methods have two arguments: a Tray object and a hold string. The method DiscardCards fills the tray with the cards to be discarded, whereas the method ReplaceCards exchanges the discarded cards.
Calculating the Score of a Hand

In order to calculate the score of a hand, the cards of the hand have to be sorted first. This operation must preserve the external view of the hand. That is, if we have the following hand:

\[
\begin{align*}
5H & & KD & & 5C & & 2S & & AH \\
\end{align*}
\]

then a proper internally sorted hand is

\[
\begin{align*}
2S & & 5C & & 5H & & KD & & AH \\
\end{align*}
\]

The cards in the hand should be stored in an array. The .NET Array class has a method

```csharp
public static void Sort(Array array)
```

that sorts the array using QuickSort on elements that implement the `IComparable` interface.

In order to calculate the score, use an integer array, which contains the card numbers:

```csharp
int[] SortedHand
```

For example, using the above card set, the array `SortedHand` contains \{2, 5, 5, 13, 14\}.

To facilitate the score calculation you may also define the following auxiliary methods:

- `bool IsStraight(ICard[] SortedCards) { ... }`
  returns true if `SortedHand` is a straight (e.g. “3D 4S 5C 6H 7H”)
- `bool IsStraightToAce(ICard[] SortedCards) { ... }`
  returns true if `SortedHand` is a straight to the ace (e.g. “TD JS QC KH AH”)
- `bool IsStraightFromAce(ICard[] SortedCards) { ... }`
  returns true if `SortedHand` is a straight from the ace (e.g. “2S 3C 4H 5H AD”)
- `bool IsFlush(ICard[] SortedCards) { ... }`
  returns true if `SortedHand` is a flush (e.g. “3H 5H 7H TH QH”).
Score calculation is described by the following Pseudo code:

```plaintext
if ( IsStraightToAce && IsFlush ) then Score = 10;
if ( IsStraight && IsFlush ) then Score = 9;
if ( (first four elements in SortedHand are equal) ||
    (last four elements in SortedHand are equal) ) then Score = 8;
if ( ((first three elements in SortedHand are equal) &&
    (last two elements in SortedHand are equal)) ||
    ((first two elements in SortedHand are equal) &&
    (last three elements in SortedHand are equal)) ) then Score = 7;
if ( IsFlush ) then Score = 6;
if ( IsStraight ) then Score = 5;
if ( (first three elements in SortedHand are equal) ||
    (second, third, and fourth element in SortedHand are equal) ||
    (last three elements in SortedHand are equal) ) then Score = 4;
if ( ((first and second element in SortedHand are equal) &&
    (third and fourth element in SortedHand are equal)) ||
    ((first and second element in SortedHand are equal) &&
    (fourth and fifth element in SortedHand are equal)) ||
    ((second and third element in SortedHand are equal) &&
    (fourth and fifth element in SortedHand are equal)) ) then Score = 3;
if ( ((first and second element in SortedHand are equal) &&
    (first element in SortedHand > 10)) ||
    ((second and third element in SortedHand are equal) &&
    (second element in SortedHand > 10)) ||
    ((third and fourth element in SortedHand are equal) &&
    (third element in SortedHand > 10)) ||
    ((fourth and fifth element in SortedHand are equal) &&
    (fourth element in SortedHand > 10)) ) then Score = 2;
else Score = 0;
```
Hand Ranking

To decide, which poker hand is better, we define the following order over hands:

- Royal Flush:
  If two royal flushes tie, use suit.

- Straight Flush:
  When two straights tie, the higher straight wins. If two straights have the same value, use suit.

- Four of a Kind:
  If there are two or more hands that qualify, the hand with the higher-ranked Four of a Kind wins.

- Full House:
  When there are two full houses the tie is broken by the Three of a kind.

- Flush:
  When flushes tie, follow the rule for High Card.

- Straight:
  When two straights tie, the higher straight wins. If two straights have the same value, then High Card breaks ties.

- Three of a Kind:
  The highest-ranking Three of a Kind wins.

- Two Pair:
  The highest-ranking pair wins ties. High Card breaks ties between the sets of Two Pair.

- Jacks or Better:
  The highest-ranking pair wins. High Card breaks ties.

- High Card (not used in Video Poker):
  The highest-ranking card wins. This rule has a unique solution due to problem statement.
Class Dealer & Player

The class Dealer implements the Poker dealer logic. However, the dealer is event-based, that is, the player and dealer communicate using events. The following diagram illustrates the interaction:

The necessary definitions:

// Dealer notifies player to pick up cards.
public delegate void CardsAvailableHandler( ITray aTray );

// Player notifies dealer to start a new round.
public delegate void StartRoundHandler();

// Player notifies dealer to replace cards in tray.
public delegate void RequestCardsHandler( ITray aTray );

// Player notifies dealer to stop round.
public delegate void EndRoundHandler( ITray aTray );

public interface IDealer
{
    // Event to signal that new cards are available to the player.
    event CardsAvailableHandler OnCardsAvailable;

    // Event handler for the player’s OnRequestCards event.
    void RequestCards( ITray aTray );

    // Event handler for the player’s OnStartRound event.
    void StartRound();

    // Event handler for the player’s OnEndRound event.
    void EndRound( ITray aTray );

    // Register a specific player.
    void RegisterPlayer( IPlayer aPlayer );
}
// Player notifies console that the hand has changed.
public delegate void DataReadyHandler();

public interface IPlayer
{
    // Event to signal that the dealer should start a new round.
    event StartRoundHandler OnStartRound;

    // Event to signal that the dealer should deal new cards.
    event RequestCardsHandler OnRequestCards;

    // Event to signal that the dealer should end the current round.
    event EndRoundHandler OnEndRound;

    // Event to signal that the console to display the player’s hand.
    event DataReadyHandler OnDataReadyStart;

    // Event to signal that the console to display the player’s hand.
    event DataReadyHandler OnDataReadyDraw;

    // Returns the current hand of the player.
    IHand Hand { get; }

    // Event handler for the dealer’s OnCardsAvailable event.
    void CardsAvailable( ITray aTray );

    // Initiate a new game. This method will fire the OnStartRound
    // event.
    void NewGame();

    // Discard and replace cards. This method will fire the
    // OnRequestCards event.
    void Draw( string aHoldString );

    // End current game. This method will fire the OnEndRound
    // event.
    void EndGame();
}

Problem 1
Create an empty Visual Studio .NET solution named VideoPoker. Add a new class library project PokerGeneral.dll to the solution that contains all the above-mentioned classes and interfaces.

Problem 2
Add a new Windows application project WinVideoPoker to the solution that uses the PokerGeneral assembly (i.e., the DLL). Create a class Game and a class Player that implement a Video Poker console application.

The Main Form must also implement two event handlers, one for the OnDataReadyStart event and one for the OnDataReadyDraw event, which are part of the game logic.
Game sequence:

- **Start:**

Game started using the Player's NewGame method, which triggers the OnDataReadyStart event (handler must enable Draw button):

- Cards-to-hold selected and Draw button pressed. Draw click event handler invokes the Player's Draw method using an appropriate hold string. This will trigger the Players OnDataReadyDraw event.
To manage the bitmaps of the card it is recommended to use ImageList objects:

Also, set the BackColor to MidnightBlue, the ForeColor to Yellow, and the Font size of the label in the center and to two game control buttons to 14pts.

**Submission deadline:** Monday, October 17, 2004, 4:10 p.m.

**Submission procedure:** on paper in class (.cs files only) and electronically using the turnin-hw4 script, which is located in ~/cs430x/public/bin. Please use the printout of the submission confirmation email as cover page and check the problems that you have solved.

In order to submit your homework solutions, go (using your CS UNIX account) into the directory that contains your solution (i.e., C#-source files and all related project files). In that directory run the command “~/cs430x/public/bin/turnin-hw4”. After a successful submission, your will receive a confirmation email. Before the due date, you can resubmit your solutions as often as you like.

On the department’s Windows XP systems you can use the command **csc** to compile C#-programs. However, it is recommended to use Visual Studio .NET, because most assignments require some GUI work.