Be sure to use the given identifiers for the class, instance variables and methods as shown in each of the UML diagrams.

If you want to use C# Property rather than the given Accessor method(eg GetSuit() in Card class) be sure to replace all Accessors with a Property implementation.

Card class information

Declare two enum type within the namespace before the body of this class as follows:

```
public enum Suit { Clubs, Diamonds, Hearts, Spades }

public enum FaceValue {
        Two, Three, Four, Five, Six, Seven, Eight, Nine,
        Ten, Jack, Queen, King, Ace
}
```

Card implements IEquatable<Card>, IComparable<Card>
-faceValue: FaceValue **-suit : Suit**

+Card()
+Card(Suit, FaceValue)
+GetFaceValue(): FaceValue
+GetSuit(): Suit
+Equals(Card): boolean
+CompareTo(Card)

The class heading will be `public class Card : IEquatable<Card>, IComparable<Card> {`

Interfaces are covered in Lecture 10.

IEquatable<T> allows objects to be compared to see if the two objects have the same value. The Equals method must be implemented though you may not actually use it explicitly in your assignment.

IComparable<T> allows objects to be compared, so one can use inbuilt Sort & Search methods on Collections. CompareTo method must be implemented.

Card() – has no body (it initializes a “blank” card)

Hand class information

Hand implements IEnumerable
-cards: List<Card>

+Hand()
+Hand(List<card>)
+GetCount():int
+GetCard(int):Card
+AddCard(Card)
+ContainsCard(Card):boolean
+RemoveCard(Card):boolean
+RemoveCardAt(int)
+SortHand()
+GetEnumerator():IEnumerator

IEnumerable allows code in other classes to use `foreach` with this class, e.g.

```
Hand myHand;
  …
foreach (Card card in myHand) {
                   ...
 }
```

The precise details of IEnumerable are unimportant for this assignment.

Implementation of GetEnumerator:

```
public IEnumerator GetEnumerator() {
    return cards.GetEnumerator();
}
```

GetCard(int):Card	**// returns the card at specified position**
AddCard(Card)	**// adds the card to the hand**
ContainsCard(Card):boolean	**//returns true if card is in the hand**
RemoveCard(Card):boolean	**//removes card from the hand if it is the hand**
RemoveCardAt(int)	**//removes the card at specified position in the hand**

<u>CardPile class information</u>

CardPile
-pile: List<Card>

+CardPile()
+CardPile(boolean)
+AddCard(card)
+GetCount():int
+GetLastCardInPile(): Card
+ShufflePile()
+DealOneCard():Card
+DealCards(int):List<Card>

CardPile() // creates an empty pile (no cards in the pile)

CardPile(boolean) // true creates pile containing a full deck of 52 distinct playing cards
// false creates an empty pile

GetLastCardInPile(): Card // returns the card in the last position of the pile
// but does not remove it from the pile

ShufflePile() // shuffles the pile of cards, implement any publicly available card shuffle
// algorithm or invent your own.

DealOneCard():Card //deals the next card in the pile and removes it from the pile

DealCards(int):List<Card> // deals a number of cards from the pile which are removed
// from the pile.