Lab:ADeckofCards

Generated by Doxygen 1.8.4

Tue Jan 21 2014 12:46:54

Contents

1 Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Cloneable

student_classes.Deck **4**

Comparable

student_classes.Card **2**

Iterable

student_classes.Deck **4**

student_classes.Numerals **6**

student_classes.Suits **7**

Comparator

2 Class Index

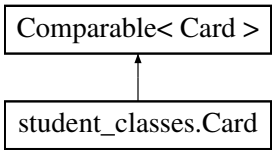
2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

3 Class Documentation

3.1 student_classes.Card Class Reference

Inheritance diagram for student_classes.Card:



Public Member Functions

- Card (Suits aSuit, Numerals aNumeral)
- Card (Card aCard)
- Suits get_suit ()
- Numerals get_numeral ()
- String toString ()
- int compareTo (Card otherCard)
- boolean equals (Object other)

3.1.1 Detailed Description

A `Card` object is a product of two enumerations: `Suits` and `Numerals`, where `Suits` contain spades, hearts, diamonds, and clubs; `Numerals` contain deuce (2), through Jack, Queen, King,and Ace (high). which brings the number of numerals to 13. Thus, we have 52 possible possible cards (products) $4 \times 13 = 52$.

Author

CS Dept., UMD.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 student_classes.Card.Card (Suits *aSuit,* Numerals *aNumeral*)

The most likely constructor that clients will use. Notice, this class does not support (expose) a default constructor—after all, what would be the default suit and default numeral for such a card?

Parameters

aSuit	[in]
aNumeral	[in]

3.1.2.2 student_classes.Card.Card (Card *aCard*)

The copy constructor is required (used by) the `clone()` method defined on the `Deck` class.

Parameters

aCard	[in]

3.1.3 Member Function Documentation

3.1.3.1 int student_classes.Card.compareTo (Card *otherCard*)

This method *compares only the* `Numeral`s of the two `Card` objects. The `compare()` method (qv) implemented on the `Deck` method implements a *more complete* notion of comparison, i.e., one that takes the `Suit` into account as well.

3.1.3.2 boolean student_classes.Card.equals (Object *other*)

Override must satisfy the requirement that `equals` returns `true` in the case where `compareTo` returns 0.

3.1.3.3 Numerals student_classes.Card.get_numeral ( )

Default read accessor that returns the `Numeral` belonging to `Card` objects.

Returns

this Card's Rank (Numeral)

3.1.3.4 Suits student_classes.Card.get_suit ( )

Default read accessor that returns the `Suit` belonging to `Card` objects.

Returns

this Card's Suit

3.1.3.5 String student_classes.Card.toString ( )

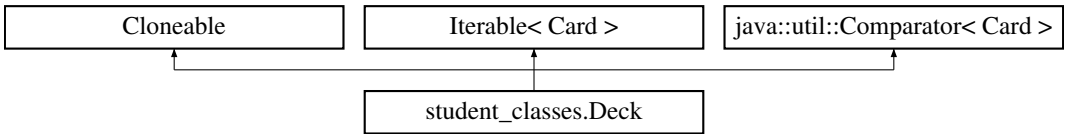
You don't need to override this method, but I strongly suggest doing so ....

The documentation for this class was generated from the following file:

- student_classes/Card.java

3.2 student_classes.Deck Class Reference

Inheritance diagram for student_classes.Deck:



Classes

- class **_Iterator**

Public Member Functions

- Deck ()
- Deck clone ()
- Iterator< Card > iterator ()
- int compare (Card card1, Card card2)
- int size ()
- void shuffle ()
- void sort ()
- String toString ()
- boolean equals (Object other)

3.2.1 Detailed Description

A `Deck` is `Cloneable`, meaning that we can make independent copies of `Deck` objects. In addition, the `Deck` also must allow clients to *iteratively* operate over `Card` objects.

Author

UMD CS Dept.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 student_classes.Deck.Deck ( )

Returns a "sorted" deck of 52 cards. Note, this Deck must be sorted according to the logic embodied by your `compare` method that you defined on `Card`.

3.2.3 Member Function Documentation

3.2.3.1 Deck student_classes.Deck.clone ( )

Essentially, this creates a clone of the existing deck by performing a *deep copy*, meaning that `Card` objects are also copied (the motivation for requiring you to do this is pedagogical...for reasons that you should be able to state, such deep copying isn't really necessary for this particular set of objects).

3.2.3.2 int student_classes.Deck.compare (Card *card1*, Card *card2*)

Implements a *two-faceted* comparison predicate: Facet one dispatches on the `Suit` of the `Card` with the following order (from least to greatest):

$$\{\text{clubs}, \text{diamonds}, \text{hearts}, \text{spades}\}$$

The second facet compares `Numerals`,

$$\{\text{deuce}, \text{three}, \ldots, \text{ace}\}$$

Thus the smallest `Numeral`, say `deuce` of spades is greater than any `ace` of a lower ranking suit, such as `hearts`.

Another way of visualizing this, passing the `compare` method to a standard sorting algorithm would result in a `Deck` sorted in its original order (i.e., in the order in which the constructor for the `Deck` class would create).

3.2.3.3 boolean student_classes.Deck.equals (Object *other*)

Two `Deck`s are equal iff their corresponding `Card`s are `equal`, using the `Deck` object's `compare` method. (Why?)

3.2.3.4 Iterator<Card> student_classes.Deck.iterator ( )

Returns the standard `Iterator<Deck>`.

3.2.3.5 void student_classes.Deck.shuffle ( )

Delegates to the Java `Collections shuffle()` method. Note: this method changes the internal representation of the `Card`s.

3.2.3.6 int student_classes.Deck.size ( )

Returns the number of `Card`s in this `Deck`.

Returns

the number of Cards in this Deck

3.2.3.7 void student_classes.Deck.sort ( )

Delegates to the Java `Collections sort` method and the `Card`'s `compare` method in order to put the `Deck` in ascending order. Note: calling this method modifies the internal order of the Deck.

3.2.3.8 String student_classes.Deck.toString ( )

You don't need to override this method, but I think it is helpful

The documentation for this class was generated from the following file:

- student_classes/Deck.java

3.3 student_classes.Numerals Enum Reference

Public Attributes

- **deuce**
- **three**
- **four**
- **five**
- **six**

- **seven**
- **eight**
- **nine**
- **ten**
- **jack**
- **queen**
- **king**
- **ace**

3.3.1 Detailed Description

`public enum` (enumeration) comprising thirteen *ranks*, starting with deuce(2), through Ace.

Author

UMD CS Department

The documentation for this enum was generated from the following file:

- student_classes/Numerals.java

3.4 student_classes.Suits Enum Reference

Public Attributes

- **clubs**
- **diamonds**
- **hearts**
- **spades**

3.4.1 Detailed Description

`public enum` (enumeration) that provides the standard 4 suits, ordered by the standard rules of Bridge (i.e., clubs (low), through spades (high).

Author

UMD CS Dept.

The documentation for this enum was generated from the following file:

- student_classes/Suits.java

Index