// Josh Martin

// Activity 1

/\*\*

\* Card.java

\*

\* <code>Card</code> represents a playing card.

\*/

**public** **class** Card {

/\*\*

\* String value that holds the suit of the card

\*/

**private** String suit;

/\*\*

\* String value that holds the rank of the card

\*/

**private** String rank;

/\*\*

\* int value that holds the point value.

\*/

**private** **int** pointValue;

/\*\*

\* Creates a new <code>Card</code> instance.

\*

\* **@param** cardRank a <code>String</code> value

\* containing the rank of the card

\* **@param** cardSuit a <code>String</code> value

\* containing the suit of the card

\* **@param** cardPointValue an <code>int</code> value

\* containing the point value of the card

\*/

**public** Card(String cardRank, String cardSuit, **int** cardPointValue) {

rank = cardRank;

suit = cardSuit;

pointValue = cardPointValue;

}

/\*\*

\* Accesses this <code>Card's</code> suit.

\* **@return** this <code>Card's</code> suit.

\*/

**public** String suit() {

**return** suit;

}

/\*\*

\* Accesses this <code>Card's</code> rank.

\* **@return** this <code>Card's</code> rank.

\*/

**public** String rank() {

**return** rank;

}

/\*\*

\* Accesses this <code>Card's</code> point value.

\* **@return** this <code>Card's</code> point value.

\*/

**public** **int** pointValue() {

**return** pointValue;

}

/\*\* Compare this card with the argument.

\* **@param** otherCard the other card to compare to this

\* **@return** true if the rank, suit, and point value of this card

\* are equal to those of the argument;

\* false otherwise.

\*/

**public** **boolean** matches(Card otherCard) {

**if** (**this**.suit.equals(otherCard.suit()) && **this**.rank.equals(otherCard.rank()) && (**this**.pointValue == otherCard.pointValue()))

**return** **true**;

**return** **false**;

}

/\*\*

\* Converts the rank, suit, and point value into a string in the format

\* "[Rank] of [Suit] (point value = [PointValue])".

\* This provides a useful way of printing the contents

\* of a <code>Deck</code> in an easily readable format or performing

\* other similar functions.

\*

\* **@return** a <code>String</code> containing the rank, suit,

\* and point value of the card.

\*/

@Override

**public** String toString() {

**return** rank() + " of " + suit() + " (point value = " + pointValue() + ")";

}

}

/\*\*

\* This is a class that tests the Card class.

\*/

**public** **class** CardTester {

/\*\*

\* The main method in this class checks the Card operations for consistency.

\* **@param** args is not used.

\*/

**public** **static** **void** main(String[] args) {

Card c1 = **new** Card("Ace" ,"spades", 1 );

Card c2 = **new** Card("Queen" ,"hearts", 5 );

Card c3 = **new** Card("King" ,"diamonds ", 10);

System.out.println(c1);

System.out.println(c2);

System.out.println(c3);

}

}

//Josh Martin

// Activity 2

**import** java.util.List;

**import** java.util.ArrayList;

/\*\*

\* The Deck class represents a shuffled deck of cards.

\* It provides several operations including

\* initialize, shuffle, deal, and check if empty.

\*/

**public** **class** Deck {

/\*\*

\* cards contains all the cards in the deck.

\*/

**private** List<Card> cards;

/\*\*

\* size is the number of not-yet-dealt cards.

\* Cards are dealt from the top (highest index) down.

\* The next card to be dealt is at size - 1.

\*/

**private** **int** size;

/\*\*

\* Creates a new <code>Deck</code> instance.<BR>

\* It pairs each element of ranks with each element of suits,

\* and produces one of the corresponding card.

\* **@param** ranks is an array containing all of the card ranks.

\* **@param** suits is an array containing all of the card suits.

\* **@param** values is an array containing all of the card point values.

\*/

**public** Deck(String[] ranks, String[] suits, **int**[] values) {

cards = **new** ArrayList<Card>();

**for** (**int** i= 0; i < ranks.length; i++)

cards.add(**new** Card(ranks[i],suits[i], values[i]));

size = cards.size();

}

/\*\*

\* Determines if this deck is empty (no undealt cards).

\* **@return** true if this deck is empty, false otherwise.

\*/

**public** **boolean** isEmpty() {

**if** (size() < 0)

**return** **false**;

**return** **true**;

}

/\*\*

\* Accesses the number of undealt cards in this deck.

\* **@return** the number of undealt cards in this deck.

\*/

**public** **int** size() {

**return** size;

}

/\*\*

\* Randomly permute the given collection of cards

\* and reset the size to represent the entire deck.

\*/

**public** **void** shuffle() {

/\* \*\*\* TO BE IMPLEMENTED IN ACTIVITY 4 \*\*\* \*/

}

/\*\*

\* Deals a card from this deck.

\* **@return** the card just dealt, or null if all the cards have been

\* previously dealt.

\*/

**public** Card deal() {

**if** ( size < 0)

**return** **null**;

**else**{

Card deal = cards.get(size);

size--;

**return** deal;

}

}

/\*\*

\* Generates and returns a string representation of this deck.

\* **@return** a string representation of this deck.

\*/

@Override

**public** String toString() {

String rtn = "size = " + size + "\nUndealt cards: \n";

**for** (**int** k = size - 1; k >= 0; k--) {

rtn = rtn + cards.get(k);

**if** (k != 0) {

rtn = rtn + ", ";

}

**if** ((size - k) % 2 == 0) {

// Insert carriage returns so entire deck is visible on console.

rtn = rtn + "\n";

}

}

rtn = rtn + "\nDealt cards: \n";

**for** (**int** k = cards.size() - 1; k >= size; k--) {

rtn = rtn + cards.get(k);

**if** (k != size) {

rtn = rtn + ", ";

}

**if** ((k - cards.size()) % 2 == 0) {

// Insert carriage returns so entire deck is visible on console.

rtn = rtn + "\n";

}

}

rtn = rtn + "\n";

**return** rtn;

}

}

/\*\*

\* This is a class that tests the Deck class.

\*/

**public** **class** DeckTester {

/\*\*

\* The main method in this class checks the Deck operations for consistency.

\* **@param** args is not used.

\*/

**public** **static** **void** main(String[] args) {

String[] ranks = {"jack", "queen", "king"};

String[] suits = {"blue", "red", "black"};

**int**[] pointValues = {11, 12, 13};

Deck d = **new** Deck(ranks, suits, pointValues);

System.***out***.print(d);

}

}

1. Deck inherits code from card by using an arraylist with the type of card.
2. 2 cards
3. Help
4. Yes it does because it has to make the right card