|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Card Dealer** |  |

|  |
| --- |
| **Your Tasks (Mark these off as you go)** |
| * Review the Card and DeckOfCards classes * Create the CardDealer and CardPanel classes * Declare the dealSize and dealt variables in the CardDealer class * Write the dealCards method in the CardDealer class * Call the dealCards method in the CardDealer class * Have Ms. Pluska check off your Card, DeckOfCards, and CardDealer classes * Complete challenges 1 thru 5 * Have Ms. Pluska check off your challenges 1 thru 3 before you continue * Receive credit for the group portion of this lab * Receive credit for the individual portion of this lab |

* **Review the Card and DeckOfCards classes**

**Card class**

The purpose of the Card class is to create Card objects. To create our cards we will need the following information:

* The face value of the card (e.g., 2, ace, king)
* The suite of the card
* The numeric value of the card (1-13)

Once the card is created, we will need to be able to display the card on the GUI. Additionally, we will need to be able to access the numerical value of the card (1-13).

The completed required Card class is given below.

|  |
| --- |
| **Card Class** |
| public class Card {  private String faceValue, suite;  private int value;  /\*\*  \* Card Constructor  \* @param fv face value of the card, (e.g., king, one, two)  \* @param s suite of the card  \* @param v value of the card (1-13)  \*/  public Card(String fv, String s, int v){  suite = s;  faceValue = fv;  value = v;  }    /\*\*  \* Gets the value of the card  \* @return value of card (1-13)  \*/  public int getValue(){  return value;  }    /\*\*  \* Gets the face value of the card  \* needed to display the card on the GUI  \* @return properly formatted face value  \*/  public String getFaceValue(){    switch(faceValue){  case "one":  return "1";  case "two":  return "2";  case "three":  return "3";  case "four":  return "4";  case "five":  return "5";  case "six":  return "6";  case "seven":  return "7";  case "eight":  return "8";  case "nine":  return "9";  case "ten":  return "10";  case "jack":  return "J";  case "queen":  return "Q";  case "king":  return "K";  case "ace":  return "A";  }  return "";  }    /\*\*  \* Gets the name of the suite  \* needed to display the card on the GUI  \* @return the name with the first letter capatilized  \*/  public String getSuite(){  String tempSuiteArray[] = suite.split(" ");  String firstLetter = tempSuiteArray[2].substring(0,1).toUpperCase();  String everythingElse = tempSuiteArray[2].substring(1);  return firstLetter+everythingElse;  }    public String toString(){  return faceValue+suite;  }  } |

**DeckOfCards class**

The DeckOfCards class creates a deck of 52 card objects. The DeckOfCards methods are described below.

|  |
| --- |
| **DeckOfCards class variables** |
| public class DeckOfCards {    private Card cards[];  private final int DECKSIZE = 52;  public int nextCardIndex = 0;  private String[] suiteNames = {  " of spades " + '\u2660',  " of diamonds " + '\u2666',  " of clubs " + '\u2663',  " of hearts " + '\u2764'  };  private String[] values = {  "ace", //1 or  "two", //2  "three", //3  "four", //4  "five", //4  "six", //5  "seven", //6  "eight", //7  "nine", //8  "ten", //9  "jack", //10  "queen", //11  "king"//12  };    /\*\*  \* Creates a sorted deck of 52 cards and  \* stores them in an array  \*/  public void buildDeck(){    cards = new Card[DECKSIZE];  int cardValueIndex = 0;  for(int s = 0; s < suiteNames.length; s++){  for(int v = 0; v < values.length; v++){  cards[cardValueIndex] = new Card(values[v], suiteNames[s], v);  cardValueIndex++;  }  }  }    /\*\*  \* gets the numeric value of the card  \* @param Card - the Card object we want the value to retreive  \* @return  \*/  public int getValue(Card c){  return c.getValue();  }    /\*\*  \* gets the symbol of the suite of the card  \* @param Card - the Card object we want the suite to retreive  \* @return  \*/  public char getSuite(Card c){    String suite = c.toString();  return suite.charAt(suite.length() - 1);  }    /\*\*  \* Returns the Card at a specified index in the deck  \* @param index - location of card  \* @return  \*/  public Card getCard(int index){  return cards[index];  }    /\*\*  \* Sets the card at a given index to a different Card  \* @param index1 - the location of the card to be set  \* @param c - the Card we want to place at the location  \*/  public void setCard(int index1, Card c){  cards[index1] = c;  }    /\*\*  \* Returns the next Card in the deck  \* @return  \*/  public Card nextCard(){  nextCardIndex++;  return cards[nextCardIndex-1];  }    /\*\*  \* Shows the card at a specified location  \* @param index - the location of the card in the deck  \* @return  \*/  public String showCard(int index){  return cards[index].toString();  }  /\*\*  \*  \* @return the size of the deck  \*/  public int deckSize(){  return DECKSIZE;  }  /\*\*  \*  \* @return an array of card values  \*/  public String[] getCardValues(){  return values;  }      /\*\*  \*  \* @return an array of suite symbols  \*/  public char[] getSuiteValues(){  char suites[] = new char[4];  for(int i = 0; i < suiteNames.length;i++)  suites[i] = suiteNames[i].charAt(suiteNames[i].length() - 1);  return suites;  }    } |

* **Create the CardDealer and CardPanel classes**

The purpose of the CardDealer class is to do the things that CardDealers typically do, shuffle cards, deal cards, etc. To do this, CardDealer will need access to the methods in the DeckOfCards class. When we declare the CardDealer class we will use the keyword “extends” to indicate that the CardDealer is a child of the DeckOfCards parent.

The purpose of the CardPanel is display our cards in the GUI. In the The CardPanel class is where you will create your CardDealer object and ultimately perform your C

Obtain two sheets of paper

On the first sheet of paper, write “CardDealer class” at the top of the page

On the second sheet of paper, write “CardPanel class” at the top of the page

The CardDealer class will use methods in the DeckOfCards class. To access these methods we will use the keyword “extends” to indicate that the CardDealer is a child of the DeckOfCards parent.

* Declare the CardCounter class using the appropriate signature

On the second sheet of paper,

* Write “CardPanel class” at the top of the page
* Declare the “CardPanel” class using the appropriate signature.

Your papers should like the example below,

|  |  |
| --- | --- |
| **Sheet 1** | **Sheet 2** |
| CardCounter class  public class CardCounter extends CardDealer{  //Leave lots of space here  } | CardPanel class  public class CardPanel{  //Leave lots of space here  } |

* **Write the main method in the CardDealer class**

Now that our Card and DeckOfCard classes are built, we can start dealing cards! Locate your CardDealer class and write a main method like shown below,

public static void main(String args[]){

}

Recall that we only want one deck of cards and that each deck contains 52 card objects. The static methods in the DeckOfCards class prevent us from confusing our current deck of cards, with a different deck of cards. To create a new deck of cards, we simply call the method in the main method of our CardDealer class,

DeckOfCards.buildDeck();

To see a particular card in our deck, we simply call the appropriate method. For example, the following code would show the value of the card at index 24.

System.out.println(DeckOfCards.showCard(24));

* **Declare the dealSize and dealt variables in the CardDealer class**

To play cards we need to know how many cards each play gets (dealSize), we also need to know the identies of the cards that have been dealt. To keep track of this information, declare the folllowing variables at the top of the CardDealer class,

private static final int DEALSIZE = 5;

private static Card[] dealt = new Card[DEALSIZE];

* **Write a the dealCards method in the CardDealer class**

Below the main method we will now write a new method. But, because this method will be used in the main method (which is static), it must also be designated as static. To get started, write the following,

|  |  |
| --- | --- |
| **dealCards method** | **Key terms defined** |
| public static Card[] dealCards(){    //leave some space here    return dealt;  } | Static – required because it will be accessed in a static method  Card[] - this method will return an array of dealt cards  dealt – the array of cards that will be returned |

To deal our cards will require that we populate the dealt array with the next card in the deck until it is full. This can be done with the code below. Add this code to the dealCards method you just wrote.

for(int i = 0; i < DEALSIZE; i++){

dealt[i] = DeckOfCards.nextCard();

}

Your final dealCards method should look as follows,

|  |
| --- |
| **Completed dealCards method** |
| public static Card[] dealCards(){    for(int i = 0; i < DEALSIZE; i++){  dealt[i] = DeckOfCards.nextCard();  }    return dealt;  } |

* **Call the dealCards method in the CardDealer class**

To deal your cards, simply return to your main method in the CardDealer class and write the following. This will deal a hand of cards.

dealCards();

* **Have Ms. Pluska check off your Card, DeckOfCards, and CardDealer classes before you continue**



Before you continue have Ms. Pluska check off your Card, DeckOfCards, and CardDealer classes

Do not continue until you have Ms. Pluska’s (or her designated TA’s) signature \_\_\_\_\_\_\_\_\_\_\_\_

* **Complete Challenges 1 thru 3**

Challenge 1

Write a method called swapCards that swaps the values of two cards in the deck. The swapCards method should have the following signature,

public static void swapCards(Card a, Card b, int cardAIndex, int cardBIndex)

Challenge 2

The buildDeck method builds a sorted deck. The dealCards method deals the required cards. But, card dealers do not deal sorted cards. Your challenge is to write a method that shuffles the cards.

In the CardDealer class write the shuffleCards method, this method will have the following signature,

public static void shuffleCards()

Use the swapCards method you wrote in challenge 1 in this method. When you are done, call this method in the main method.

Challenge 3

Write a method in the CardDealer that finds the highest card in a shuffled hand and returns the card

* **Have Ms. Pluska check off challenges 1 thru 3**



Before you continue have Ms. Pluska check off challenges 1 thru 3.

Do not continue until you have Ms. Pluska’s (or her designated TA’s) signature \_\_\_\_\_\_\_\_\_\_\_\_

* **Receive Credit for the group portion of this lab**

Make sure indicate the names of all group members, then submit this lab to the needs to be graded folder to receive credit for the group portion of this lab.

* **Receive Credit for the individual portion of this lab**

Implement challenges 1 thru 3 on your computer. Show Ms. Pluska the completed challenges to receive credit for the individual portion of this lab.