Jaron and Emre have completed these 4 activitys together in complience with the .pdf file.

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) {

// /\* \*\*\* TO BE IMPLEMENTED IN ACTIVITY 1 \*\*\* \*/

rank = cardRank;

suit = cardSuit;

pointValue = cardPointValue;

}

/\*\*

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

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

\*/

public String suit() {

/\* \*\*\* TO BE IMPLEMENTED IN ACTIVITY 1 \*\*\* \*/

return suit;

}

/\*\*

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

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

\*/

public String rank() {

/\* \*\*\* TO BE IMPLEMENTED IN ACTIVITY 1 \*\*\* \*/

return rank;

}

/\*\*

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

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

\*/

public int pointValue() {

/\* \*\*\* TO BE IMPLEMENTED IN ACTIVITY 1 \*\*\* \*/

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) {

/\* \*\*\* TO BE IMPLEMENTED IN ACTIVITY 1 \*\*\* \*/

return this.suit().equals(otherCard.suit())

&& this.rank().equals(otherCard.rank())

&& this.pointValue() == otherCard.pointValue();

}

/\*\*

\* 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() {

/\* \*\*\* TO BE IMPLEMENTED IN ACTIVITY 1 \*\*\* \*/

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) {

/\* \*\*\* TO BE IMPLEMENTED IN ACTIVITY 1 \*\*\* \*/

Card gooning = new Card("Queen","Spades",12);

Card gooning2 = new Card("Ten","Hearts",10);

Card gooning3 = new Card("Ace","Clubs",1);

System.out.println(gooning.matches(gooning2));

System.out.println(gooning2);

System.out.println(gooning3.rank());

System.out.println(gooning3.suit());

System.out.println(gooning3.pointValue());

}

}

Activity 2:

/\*\*

\* 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) {

//initializes a new Card with the given rank, suit, and point value

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) {

return otherCard.suit().equals(this.suit())

&& otherCard.rank().equals(this.rank())

&& otherCard.pointValue() == this.pointValue();

}

/\*\*

\* 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 + ")";

}

}

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 j = 0; j < ranks.length; j++) {

for (String suitString : suits) {

cards.add(new Card(ranks[j], suitString, values[j]));

}

}

size = cards.size();

shuffle();

}

/\*\*

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

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

\*/

public boolean isEmpty() {

return size == 0;

}

/\*\*

\* 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 (isEmpty()) {

return null;

}

size--;

Card c = cards.get(size);

return c;

}

/\*\*

\* 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"};

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

// Deck deck = new Deck(ranks, suits, pointValues);

String [ ] ranks = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "jack", "queen", "king", "ace"};

String [ ] suits = {"spades", "hearts", "diamonds", "clubs"};

int [ ] pointValues = {2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10, 11};

Deck deck = new Deck(ranks, suits, pointValues);

// System.out.println(deck.deal());

// System.out.println(deck.size());

System.out.println(deck);

}

}

Activity 3:

/\*\*

\* This class provides a convenient way to test shuffling methods.

\*/

public class Shuffler {

/\*\*

\* The number of consecutive shuffle steps to be performed in each call

\* to each sorting procedure.

\*/

private static final int SHUFFLE\_COUNT = 1;

/\*\*

\* Tests shuffling methods.

\* @param args is not used.

\*/

public static void main(String[] args) {

System.out.println("Results of " + SHUFFLE\_COUNT +

" consecutive perfect shuffles:");

int[] values1 = {0, 1, 2, 3};

for (int j = 1; j <= SHUFFLE\_COUNT; j++) {

perfectShuffle(values1);

System.out.print(" " + j + ":");

for (int k = 0; k < values1.length; k++) {

System.out.print(" " + values1[k]);

}

System.out.println();

}

System.out.println();

System.out.println("Results of " + SHUFFLE\_COUNT +

" consecutive efficient selection shuffles:");

int[] values2 = {0, 1, 2, 3};

for (int j = 1; j <= SHUFFLE\_COUNT; j++) {

selectionShuffle(values2);

System.out.print(" " + j + ":");

for (int k = 0; k < values2.length; k++) {

System.out.print(" " + values2[k]);

}

System.out.println();

}

System.out.println();

}

/\*\*

\* Apply a "perfect shuffle" to the argument.

\* The perfect shuffle algorithm splits the deck in half, then interleaves

\* the cards in one half with the cards in the other.

\* @param values is an array of integers simulating cards to be shuffled.

\*/

public static void perfectShuffle(int[] values) {

/\* \*\*\* TO BE IMPLEMENTED IN ACTIVITY 3 \*\*\* \*/

int[]shuffled = new int[values.length];

int k = 0;

for(int i = 0; i<(values.length+1)/2; i++){

shuffled[k] = values[i];

k+=2;

}

k = 1;

for(int c = (values.length + 1)/2; c<values.length; c++){

shuffled[k] = values[c];

k+=2;

}

System.arraycopy(shuffled, 0, values, 0, values.length);

}

/\*\*

\* Apply an "efficient selection shuffle" to the argument.

\* The selection shuffle algorithm conceptually maintains two sequences

\* of cards: the selected cards (initially empty) and the not-yet-selected

\* cards (initially the entire deck). It repeatedly does the following until

\* all cards have been selected: randomly remove a card from those not yet

\* selected and add it to the selected cards.

\* An efficient version of this algorithm makes use of arrays to avoid

\* searching for an as-yet-unselected card.

\* @param values is an array of integers simulating cards to be shuffled.

\*/

public static void selectionShuffle(int[] values) {

/\* \*\*\* TO BE IMPLEMENTED IN ACTIVITY 3 \*\*\* \*/

int lotto = 0;

int[]shuffled = new int[values.length];

for(int i = 0; i<values.length; i++){

lotto = (int)(Math.random()\*values.length)+1;

if(values[lotto-1] != 100){

shuffled[i] = values[lotto-1];

values[lotto-1] = 100;

}

else

i--;

continue;

}

System.arraycopy(shuffled, 0, values, 0, values.length);

}

public static String flip() {

int r = (int) (Math.random() \* 3);

return r < 2 ? "heads ": "tails";

}

public static boolean arePermutations(int[] a, int[] b) {

for (int aV: a) {

boolean foud = false;

for (int bValue: b) {

if (bValue == aV) {

foud = true;

break;

}

}

if (!foud) {

return false;

}

}

return true;

}

}

Activity 4:

/\*\*

\* 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) {

//initializes a new Card with the given rank, suit, and point value

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) {

return otherCard.suit().equals(this.suit())

&& otherCard.rank().equals(this.rank())

&& otherCard.pointValue() == this.pointValue();

}

/\*\*

\* 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 + ")";

}

}

import java.util.Collections;

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 j = 0; j < ranks.length; j++) {

for (String suitString : suits) {

cards.add(new Card(ranks[j], suitString, values[j]));

}

}

size = cards.size();

shuffle();

}

/\*\*

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

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

\*/

public boolean isEmpty() {

return size == 0;

}

/\*\*

\* 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 \*\*\* \*/

Collections.shuffle(cards);

size = cards.size();

}

/\*\*

\* 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 (isEmpty()) {

return null;

}

size--;

Card c = cards.get(size);

return c;

}

/\*\*

\* 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"};

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

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

System.out.println("\*\*\*\* Original Deck Methods \*\*\*\*");

System.out.println(" toString:\n" + d.toString());

System.out.println(" isEmpty: " + d.isEmpty());

System.out.println(" size: " + d.size());

System.out.println();

System.out.println();

System.out.println("\*\*\*\* Deal a Card \*\*\*\*");

System.out.println(" deal: " + d.deal());

System.out.println();

System.out.println();

System.out.println("\*\*\*\* Deck Methods After 1 Card Dealt \*\*\*\*");

System.out.println(" toString:\n" + d.toString());

System.out.println(" isEmpty: " + d.isEmpty());

System.out.println(" size: " + d.size());

System.out.println();

System.out.println();

System.out.println("\*\*\*\* Deal Remaining 5 Cards \*\*\*\*");

for (int i = 0; i < 5; i++) {

System.out.println(" deal: " + d.deal());

}

System.out.println();

System.out.println();

System.out.println("\*\*\*\* Deck Methods After All Cards Dealt \*\*\*\*");

System.out.println(" toString:\n" + d.toString());

System.out.println(" isEmpty: " + d.isEmpty());

System.out.println(" size: " + d.size());

System.out.println();

System.out.println();

System.out.println("\*\*\*\* Deal a Card From Empty Deck \*\*\*\*");

System.out.println(" deal: " + d.deal());

System.out.println();

System.out.println();

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

String[] ranks1 = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "jack", "queen", "king", "ace"};

String[] suits1 = {"clubs", "diamonds", "hearts", "spades"};

int[] pointValues1 = {2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14};

Deck standardDeck = new Deck(ranks1, suits1, pointValues1);

System.out.println("\*\*\*\* Standard Deck of 52 Cards \*\*\*\*");

System.out.println(standardDeck.toString());

System.out.println("\*\*\*\* Shuffle Standard Deck \*\*\*\*");

for (int i = 0; i < 5; i++) {

standardDeck.shuffle();

System.out.println("After shuffle " + (i + 1) + ":\n" + standardDeck.toString());

}

}

}