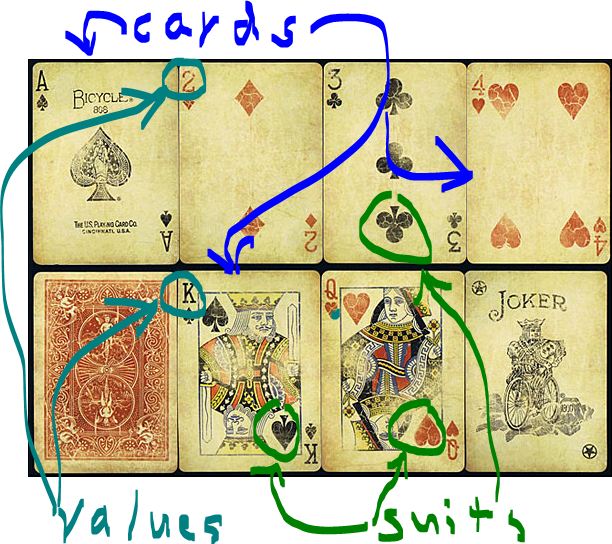
**(M3) Write a Java program: Decks of Cards (4 hrs)**

**Decks of Cards**

**Understand the Class and Problem**

We endeavor to set up some classes that can be used in future programs that involve playing card games with a human, or simulating card games entirely by a computer.  We will start with these two and then adThere are two  basic classes we'll need this week:

* **Card**:  A class like the one presented in the modules, but with a few changes.
* **Hand**:  A class that represents the cards held by a single player.
* **Deck**:  A class that represents the source of the cards for dealing and, as the game progresses, the place from which players can receive new cards (say, as they pick cards "from the deck" or when future hands are to be dealt from the same deck).  Recall this picture, which relates the **Deck** to the various **Hands** that it creates through the process called "dealing".

Here  are eight **cards**, each of which contains both a **value** ('A', '2', '3', ... 'T', 'J', 'Q',' K') and a **suit** (spades ♠, hearts ♥, diamonds ♦, clubs ♣)  


Notice that I am using the char **'T'** to describe the value **10**.  (Ignore the Joker, which we will not need.)

The dealer uses a **Deck** object to deal **Hand** objects to the players.  The dealer may or may not be a player who gets a hand of his own (poker dealers in casinos don't receive a hand, but most other games involve the dealer getting a hand).



***Card***: The **Card** class has two obvious members:  **value** (a **char**)  and **suit** (an **enum**).  But we add a new boolean, **errorFlag**, which can inform a client that a card is in an illegal state. We'll want the usual constructors, mutators, accessors and**toString()** methods for the class.  We only allow standard cards, like **('A', clubs)**,**('9', hearts**) and **('T', diamonds)**, ***no jokers*** or other special cards.

***Hand***:  As you can see, a **Hand** object usually contains several cards, so we'll need an **array of Card** objects (**myArray**) as the principal member of the **Hand** class.  Since each game deals a different number of cards into its players hands, and even within a game the number of cards in a hand will increase or decrease, we must keep track of this with an **int** value (**numCards**).  We'll need constructors, mutators, etc., of course.  We'll also want a way for the hand to receive a card (from the deck or somewhere else), and play a card (to the table or to another player).  These two methods will be called**takeCard()** and **playCard()**, respectively.  Since this class has no information about the game being played, it always puts new cards received by **takeCard()** into the next available location of the array (index position **numCards**) and plays a card via **playCard()** from the highest occupied location (index position **numCards - 1**).  The client game application would somehow prepare this highest position with the correct card to be played before calling **Hand**'s**playCard()** method.  This detail is not our concern.

***Deck:***A **Deck** object is the source of all cards.  It's where the dealer gets cards to deal, and if a player takes an individual card after the deal, he takes it from the **Deck** object.  Naturally, the primary member here is an**array of Card** objects, much like **Hand**.  We'll call this member **cards[]**.  A deck normally consists of a ***single pack*** of cards: 52 cards (four ***suits*** of 13 ***values*** each).  However, some games use ***two, three or more packs***.  If a card game requires two packs, then the deck will consist of two full 52-card packs:  104 cards.  (Many games throw away some cards before beginning.  For example ***Pinochle*** wants all cards with values 8-and-below to be taken out of the deck, but we will not trouble ourselves with this complexity.)  A newly instantiated deck will have a multiple of 52 cards and will contain all the standard cards, so the number of cards in a newly instantiated deck will be 52, 104, 156, ...,  i.e., **numPacks**× 52.

Clearly, we need an int like **Hand**'s **numCards**, to keep track of how many cards are actually in the **cards[]**array.  To this end, we'll use **topCard** (*not* **numCards**), since a deck typically removes and delivers cards to players from the top-of-the-deck, and this is a convenient variable to use for the number of cards as well as the position of the top of the deck.

There are a few other useful members (**numPacks**, for example).  In addition to the the usual constructors and accessors, we'll want a **dealCard()** to return and remove the card at the top of the deck (which may be received by a client and added to some player's hand), and a **shuffle()** to re-order the cards in a random fashion.  Also, we'll need to restock the deck **(init()**) to the original full condition in preparation for a fresh deal (we would certainly not want to re-instantiate a new deck when we have a perfectly good one available:  garbage collection, done by us or by the operating system, is a resource we do not abuse).

\*\*There will be one main class and three other classes.  Put everything that is needed for testing in one **main()**.  Submit only one .txt file that has all classes and output.

**Phase 1: The Card Class**

**A Public enum Type**

Define the **Suit** enum, **{ clubs, diamonds, hearts, spades }**, *inside* the **Card** class.

**Private Member Data**

Include three members:

char value;

Suit suit;

boolean errorFlag;

**Public Methods**

* **Card(char value, Suit suit)**- The constructor should call the proper mutator(s).  Overload this to cope with a client that wants to instantiate without parameters and use 'A' and 'spades' as the default value and suit when not supplied.  Provide at least two constructors -- no parameters and *all* parameters -- or more if you wish.  Because we have the **errorFlag** member, the constuctor (via the mutator), can set that member when it gets bad data; it does not have to assign default values upon receipt of bad data.  This is a new technique for us.  Again, default card (no parameters passed) is the **('A', spades)**.
* **string toString()** - a *stringizer* that the client can use prior to displaying the card.  It provides a clean representation of the card.  If**errorFlag == true**, it should return correspondingly reasonable reflection of this fact (something like "**[ invalid ]**" rather than a suit and value).
* **boolean set(char value, Suit suit)** - a mutator that accepts the legal values established in the earlier section.  When bad values are passed, **errorFlag** is set to **true** and other values can be left in any state (even partially set). If good values are passed, they are stored and **errorFlag** is set to **false**.  Make use of the private helper, listed below.
* **Accessors for suit and value.**
* **Accessor for errorFlag.**
* **boolean equals(Card card)** - returns true if all the fields (members) are identical and false, otherwise.

**Private Methods**

* **boolean isValid(char value, Suit suit)** - a private helper method that returns **true** or **false**, depending on the legality of the parameters.  Note that, although it may be impossible for **suit** to be illegal (due to its **enum**-ness), we pass it, anyway, in anticipation of possible changes to the type from **enum** to, say, **char** or **int**, someday.  We only need to test **value**, at this time.

Note:  we don't need individual mutators for **value** or **suit** since they would not be needed for this particular class.

**Test of Card class**

Create a main() that is in a class, like Assig3, and then instantiate three cards, two legally, and one illegally. Print all three out and confirm. Then make good card bad by **set()** with an illegal value, and change the initial illegal card to a legal one by setting a legal value.

/\* -------------------------------------------------------

A of spades

\*\* illegal \*\*

J of clubs

\*\* illegal \*\*

Q of spades

J of clubs

------------------------------------------------------- \*/

**Phase 2: The Hand Class**

**Static Class Constants**

Define a public **int** value like **MAX\_CARDS** and set it to something like 50 or 100 so a runaway program can't try to create a monster array.

**Private Member Data**

Card[] myCards;

int numCards;

**Public Methods**

* **Hand()** - a default constructor.
* **void resetHand()** - remove all cards from the hand (in the simplest way).
* **boolean takeCard(Card card)** - adds a card to the next available position in the **myCards** array.  This is an object copy, not a reference copy, since the source of the Card might destroy or change its data after our Hand gets it -- we want our local data to be exactly as it was when we received it.
* **Card playCard()**- returns and removes the card in the top occupied position of the array.
* **String toString()** - a *stringizer* that the client can use prior to displaying the entire hand.
* **Accessor for numCards**.
* **Card inspectCard(int k)** - Accessor for an individual card.  Returns a card with **errorFlag = true** if **k** is bad.

**Test of Hand class**

Create between two and five explicit **Card** objects and one **Hand** object. Use **takeCard()** on these few cards (resulting in many, unavoidable "duplicates" in the hand)  in a loop to populate the **hand** until the maximum allowable cards is met (use this criterion to end the loop).  Display the **hand** using **toString()**.  Next,  ***play*** each card in a loop, until the **hand** is empty. Display the card played ***as it is played***, and finally, display the (now empty)  **hand**, verifying that no cards remain.  At some point in your program, test **inspectCard()** with both legal and illegal int arguments.

**Example Test Run of Hand Class**

/\* -------------------------------------------------------------------------

Hand full

After deal

Hand = ( 3 of clubs, T of clubs, 9 of hearts, 3 of clubs, T of clubs, 9 of hear

ts, 3 of clubs, T of clubs, 9 of hearts, 3 of clubs, T of clubs, 9 of hearts, 3

of clubs, T of clubs, 9 of hearts, 3 of clubs, T of clubs, 9 of hearts, 3 of clu

bs, T of clubs, 9 of hearts, 3 of clubs, T of clubs, 9 of hearts, 3 of clubs, T

of clubs, 9 of hearts, 3 of clubs, T of clubs, 9 of hearts, 3 of clubs, T of clu

bs, 9 of hearts, 3 of clubs, T of clubs, 9 of hearts, 3 of clubs, T of clubs, 9

of hearts, 3 of clubs, T of clubs, 9 of hearts, 3 of clubs, T of clubs, 9 of hea

rts, 3 of clubs, T of clubs, 9 of hearts, 3 of clubs, T of clubs, 9 of hearts, 3

of clubs, T of clubs, 9 of hearts, 3 of clubs, T of clubs, 9 of hearts, 3 of cl

ubs, T of clubs, 9 of hearts, 3 of clubs, T of clubs, 9 of hearts, 3 of clubs, T

of clubs, 9 of hearts, 3 of clubs, T of clubs, 9 of hearts, 3 of clubs, T of cl

ubs, 9 of hearts, 3 of clubs, T of clubs, 9 of hearts, 3 of clubs, T of clubs, 9

of hearts, 3 of clubs, T of clubs, 9 of hearts, 3 of clubs, T of clubs, 9 of he

arts, 3 of clubs, T of clubs, 9 of hearts, 3 of clubs, T of clubs, 9 of hearts,

3 of clubs, T of clubs, 9 of hearts, 3 of clubs, T of clubs, 9 of hearts, 3 of c

lubs, T of clubs, 9 of hearts, 3 of clubs )

Testing inspectCard()

9 of hearts

\*\* illegal \*\*

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

Playing 9 of hearts

Playing T of clubs

Playing 3 of clubs

After playing all cards

Hand = ( )

----------------------------------------------------------------------- \*/

**Phase 3: The Deck Class**

**Public Static Class Constants**

* Define a public final **int** value like **MAX\_CARDS**, and initialize it to allow a maximum of six packs (6×52 cards).

**Private Static Member Data**

* **Card[] masterPack**

This is a private ***static*** **Card** array, **masterPack[]**, containing exactly 52 card references, which point to all the standard cards.   It will enable us to avoid capriciously and repeatedly declaring the same 52 cards which are needed as the game proceeds.  In other words, once we have, say, a ('6', spades) **Card** constructed and stored (inside this **masterPack[]**), we use that same instance whenever we need it as a source to copy in various places, notably during a***re-initialization*** of the**Deck** object;  it will always be in the **masterPack[]** array for us to copy.

**Private Member Data**

Card[] cards;

int topCard;

int numPacks;

**Public Methods**

* **Deck(int numPacks)** - a constructor that populates the arrays and assigns initial values to members.  Overload so that if no parameters are passed, 1 pack is assumed.
* **void init(int numPacks)** - re-populate **cards[]** with the standard 52 × **numPacks** cards.  We should *not* repopulate the static array, **masterPack[]**, since that was done once, in the (first-invoked) constructor and  never changes.
* **void** **shuffle()** - mixes up the cards with the help of the standard random number generator.
* **Card dealCard()**- returns and removes the card in the top occupied position of **cards[]**.
* An **accessor** for the int, **topCard** (no mutator.)
* **Card inspectCard(int k)** - Accessor for an individual card.  Returns a card with **errorFlag = true** if **k** is bad.

**Private Methods**

* **static void allocateMasterPack()** - this is a private method that will be called by the constructor.  However, it has to be done with a very simple twist:  even if many **Deck** objects are constructed in a given program, this static method will not allow itself to be executed more than once.  Since **masterPack[]** is a static, unchanging, entity, it need not be built every time a new **Deck** is instantiated.  So this method needs to be able to ask itself, "Have I been here before?", and if the answer is "yes", it will immediately return without doing anything;  it has already built **masterPack[]** in a previous invocation.

**Test of Class Deck**

Declare a deck containing ***two packs*** of cards. Do not shuffle.  Deal *all* the cards in a loop until the deck is empty (dealt directly to the display/screen, not to any **Hand** objects just yet).  Display each card as it comes off the deck.  Next, reset the deck by initializing it again (to the same two packs).  ***Shuffle****the deck this time*, and re-deal to the screen in a loop again. Notice that the cards are now coming off in a random order.

Repeat this double deal, unshuffled, then shuffled, but this time using a ***single pack*** deck.

**Example Test Run of Card Class**

/\* ---------------------------------------------------------

K of Spades / Q of Spades / J of Spades / T of Spades / 9 of Spades / 8 of

Spades / 7 of Spades / 6 of Spades / 5 of Spades / 4 of Spades / 3 of Spade

s / 2 of Spades / A of Spades / K of Hearts / Q of Hearts / J of Hearts /

T of Hearts / 9 of Hearts / 8 of Hearts / 7 of Hearts / 6 of Hearts / 5 of

Hearts / 4 of Hearts / 3 of Hearts / 2 of Hearts / A of Hearts / K of Diamo

nds / Q of Diamonds / J of Diamonds / T of Diamonds / 9 of Diamonds / 8 of

Diamonds / 7 of Diamonds / 6 of Diamonds / 5 of Diamonds / 4 of Diamonds /

3 of Diamonds / 2 of Diamonds / A of Diamonds / K of Clubs / Q of Clubs / J

of Clubs / T of Clubs / 9 of Clubs / 8 of Clubs / 7 of Clubs / 6 of Clubs

/ 5 of Clubs / 4 of Clubs / 3 of Clubs / 2 of Clubs / A of Clubs / K of Sp

ades / Q of Spades / J of Spades / T of Spades / 9 of Spades / 8 of Spades

/ 7 of Spades / 6 of Spades / 5 of Spades / 4 of Spades / 3 of Spades / 2

of Spades / A of Spades / K of Hearts / Q of Hearts / J of Hearts / T of He

arts / 9 of Hearts / 8 of Hearts / 7 of Hearts / 6 of Hearts / 5 of Hearts

/ 4 of Hearts / 3 of Hearts / 2 of Hearts / A of Hearts / K of Diamonds /

Q of Diamonds / J of Diamonds / T of Diamonds / 9 of Diamonds / 8 of Diamond

s / 7 of Diamonds / 6 of Diamonds / 5 of Diamonds / 4 of Diamonds / 3 of Di

amonds / 2 of Diamonds / A of Diamonds / K of Clubs / Q of Clubs / J of Clu

bs / T of Clubs / 9 of Clubs / 8 of Clubs / 7 of Clubs / 6 of Clubs / 5 of

Clubs / 4 of Clubs / 3 of Clubs / 2 of Clubs / A of Clubs /

T of Hearts / T of Diamonds / A of Clubs / Q of Spades / 3 of Clubs / 8 of Diamo

nds / T of Clubs / 5 of Clubs / J of Clubs / 3 of Clubs / 3 of Diamonds / 3 of H

earts / 3 of Spades / 2 of Hearts / Q of Clubs / 7 of Spades / A of Spades / Q o

f Hearts / 6 of Spades / 5 of Clubs / K of Spades / 4 of Diamonds / J of Diamond

s / Q of Spades / A of Diamonds / K of Clubs / 4 of Hearts / T of Spades / 9 of

Hearts / 6 of Clubs / A of Hearts / 8 of Clubs / Q of Clubs / 9 of Clubs / T of

Clubs / 6 of Hearts / 6 of Hearts / K of Diamonds / 2 of Clubs / 5 of Diamonds /

Q of Hearts / Q of Diamonds / A of Diamonds / 5 of Diamonds / 2 of Spades / 7 o

f Clubs / 9 of Diamonds / 6 of Diamonds / 7 of Spades / 4 of Clubs / 9 of Diamon

ds / 9 of Spades / A of Spades / 7 of Hearts / 2 of Hearts / J of Clubs / 8 of C

lubs / A of Clubs / 4 of Spades / K of Diamonds / T of Spades / T of Diamonds /

Q of Diamonds / K of Hearts / 4 of Hearts / 6 of Diamonds / J of Hearts / 6 of S

pades / 8 of Spades / 7 of Diamonds / 8 of Diamonds / 2 of Diamonds / 8 of Heart

s / K of Hearts / 5 of Spades / K of Clubs / 4 of Spades / J of Hearts / 9 of Sp

ades / 3 of Hearts / K of Spades / 4 of Clubs / 9 of Hearts / 6 of Clubs / 7 of

Diamonds / A of Hearts / 2 of Spades / J of Spades / J of Spades / T of Hearts /

5 of Spades / 7 of Hearts / J of Diamonds / 5 of Hearts / 3 of Spades / 4 of Di

amonds / 2 of Clubs / 8 of Hearts / 2 of Diamonds / 3 of Diamonds / 7 of Clubs /

8 of Spades / 5 of Hearts / 9 of Clubs /

K of Spades / Q of Spades / J of Spades / T of Spades / 9 of Spades / 8 of

Spades / 7 of Spades / 6 of Spades / 5 of Spades / 4 of Spades / 3 of Spade

s / 2 of Spades / A of Spades / K of Hearts / Q of Hearts / J of Hearts /

T of Hearts / 9 of Hearts / 8 of Hearts / 7 of Hearts / 6 of Hearts / 5 of

Hearts / 4 of Hearts / 3 of Hearts / 2 of Hearts / A of Hearts / K of Diamo

nds / Q of Diamonds / J of Diamonds / T of Diamonds / 9 of Diamonds / 8 of

Diamonds / 7 of Diamonds / 6 of Diamonds / 5 of Diamonds / 4 of Diamonds /

3 of Diamonds / 2 of Diamonds / A of Diamonds / K of Clubs / Q of Clubs / J

of Clubs / T of Clubs / 9 of Clubs / 8 of Clubs / 7 of Clubs / 6 of Clubs

/ 5 of Clubs / 4 of Clubs / 3 of Clubs / 2 of Clubs / A of Clubs /

2 of Hearts / 3 of Diamonds / 3 of Clubs / 4 of Diamonds / Q of Diamonds /

2 of Spades / 3 of Hearts / K of Hearts / 6 of Clubs / K of Spades / 4 of C

lubs / 6 of Spades / 8 of Diamonds / 5 of Clubs / 9 of Diamonds / 5 of Diam

onds / 3 of Spades / K of Clubs / J of Diamonds / A of Diamonds / Q of Spad

es / T of Hearts / J of Hearts / A of Hearts / J of Clubs / 7 of Spades /

6 of Diamonds / 9 of Hearts / 8 of Clubs / K of Diamonds / 7 of Clubs / 7 o

f Diamonds / 8 of Spades / 4 of Spades / 2 of Diamonds / 5 of Hearts / 9 of

Spades / T of Spades / 2 of Clubs / 5 of Spades / 6 of Hearts / Q of Clubs

/ 4 of Hearts / A of Spades / 9 of Clubs / J of Spades / T of Clubs / A o

f Clubs / Q of Hearts / T of Diamonds / 7 of Hearts / 8 of Hearts /

Press any key to continue . . .

--------------------------------------------------------- \*/

**Phase 4: The Deck and Hand Classes**

It is now time to allow your **Deck** class to interact with your **Hand** class.  Don't add anything to the two classes, but do everything in this phase from within your **main()** client.

Ask the user (interactively) to select the number of players (a number from 1 to 10).  That's one question, one numeric answer, and no further user-interaction.  Once you have validated a legal value, ***instantiate*** a single-pack **Deck** object *without shuffling*, ***deal*** a deck into that many **Hand** objects, dealing all cards until the deck is empty.  Since the number of players chosen by the user may not divide evenly into 52, the number of cards dealt into the various hands might differ, but only by, at most, one.  Display all the hands after the deal.

***Reset*** the objects to their initial state, but this time ***shuffle*** the deck before a second deal (same # of players).

To be clear, dealing to hands means dealing a*single card to each hand*, until all hands have one card, then repeating to give all hands a second card, etc., until the cards are gone, and each hand has (nearly) the same number of cards.  It does *not*mean dealing *x* cards to one hand, then *x* to the next hand, etc.  This is very important.

\*\*You don't need any more classes than the ones we've already created.  Put everything in one **main()**.

**Example of One of Possibly Many Test Runs of Deck + Card Classes**

--------------- run #2 ----------------------------------

How many hands? (1 - 10, please): 6

Here are our hands, from unshuffled deck:

Hand = ( K of Spades, 7 of Spades, A of Spades, 8 of Hearts, 2 of Hearts, 9 of

Diamonds, 3 of Diamonds, T of Clubs, 4 of Clubs )

Hand = ( Q of Spades, 6 of Spades, K of Hearts, 7 of Hearts, A of Hearts, 8 of

Diamonds, 2 of Diamonds, 9 of Clubs, 3 of Clubs )

Hand = ( J of Spades, 5 of Spades, Q of Hearts, 6 of Hearts, K of Diamonds, 7 o

f Diamonds, A of Diamonds, 8 of Clubs, 2 of Clubs )

Hand = ( T of Spades, 4 of Spades, J of Hearts, 5 of Hearts, Q of Diamonds, 6 o

f Diamonds, K of Clubs, 7 of Clubs, A of Clubs )

Hand = ( 9 of Spades, 3 of Spades, T of Hearts, 4 of Hearts, J of Diamonds, 5 o

f Diamonds, Q of Clubs, 6 of Clubs )

Hand = ( 8 of Spades, 2 of Spades, 9 of Hearts, 3 of Hearts, T of Diamonds, 4 o

f Diamonds, J of Clubs, 5 of Clubs )

Here are our hands, from SHUFFLED deck:

Hand = ( 9 of Clubs, Q of Spades, 8 of Hearts, Q of Hearts, 3 of Hearts, 9 of S

pades, K of Hearts, 8 of Spades, T of Diamonds )

Hand = ( T of Clubs, 5 of Spades, 3 of Clubs, A of Diamonds, K of Clubs, 5 of H

earts, J of Diamonds, 7 of Diamonds, 2 of Spades )

Hand = ( 4 of Spades, 2 of Clubs, Q of Diamonds, 8 of Clubs, 4 of Hearts, 2 of

Hearts, 3 of Spades, 2 of Diamonds, J of Clubs )

Hand = ( 9 of Hearts, 4 of Diamonds, T of Hearts, 4 of Clubs, 5 of Diamonds, 7

of Clubs, A of Clubs, 7 of Spades, A of Hearts )

Hand = ( J of Spades, 6 of Diamonds, 9 of Diamonds, 5 of Clubs, 6 of Spades, J

of Hearts, 8 of Diamonds, K of Diamonds )

Hand = ( Q of Clubs, T of Spades, 3 of Diamonds, 6 of Hearts, 6 of Clubs, 7 of

Hearts, K of Spades, A of Spades )

Press any key to continue . . .

--------------------------------------------------------- \*/

For this part, you will be graded on how efficiently you put together these two classes.  Use what you know about arrays, loops, the methods available in the **Deck** and **Hand** classes -- even testing user input for valid in-range response --  to give a clean, short and completely tested client that proves that your **Deck** can feed the number of **Hand**s requested by the user.  There is some amount of creativity and variability allowed in this part, and any two correct solutions will look very different.  You can implement this in any way that interprets the instructions.  Yet, I can and will deduct when I see basic programming concepts misused, deduction amounts commensurate with the type of infraction.