Understand the Class and Problem

We continue to work on the card game effort, now adding the source of all cards for the various players, the **Deck**.

* **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":

Let's deconstruct the meaning of this important class.

***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 **(initializePack()**) 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).

Phase 1: The Deck Class

Private Static Class Constants

Define a private  final **int** value like **MAX\_PACKS** = 6 **, NUM\_CARDS\_PER\_PACK** = 52 , and **MAX\_CARDS\_PER\_DECK** = **MAX\_PACKS** \* **NUM\_CARDS\_PER\_PACK**.  Use them to their full benefit in the class code.

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, one pack is assumed.  This constructor can call a helper,**allocateMasterPack()** (see below), but that helper would only do something the very first time it gets called per program (no need to allocate a static array more than once per program, right?).  It would then use another helper,**initializePack()**, to assign the master pack **Cards** to the various **cards[]** elements.  If initializePack() returns, false, set the number of packs to be 1, then call initializePack() again with 1 as the argument.
* **boolean initializePack(int numPacks)** - re-populate **cards[]** with the standard 52 × **numPacks** cards. (This also gives the client a chance to change the number of packs in the deck in preparation for a new game.)  We should *not* repopulate the static array, **masterPack[]**, since that was done once, in the (first-invoked) constructor and  never changes. The elements of the **cards[]** array can *reference* the **masterPack[]** objects -- that's safe since we will never give the client any of those objects to modify (see **dealCard()**on this issue). If **numPacks** is out-of-range, return **false** without changing the object;  else return **true** and make the change.
* void **shuffle()** - mixes up the cards with the help of the standard random number generator.
* **Card dealCard()**- returns and removes  (*effectively*, not ***physically***) the card in the top occupied position of **cards[]**.  Here we have to return a ***copy*** of the card, not the actual *reference* to the object in the **cards[]**array, since that object is also the object in the **masterPack[]** array, which the client must not be allowed to change.
* An **accessor** for the int, **topCard** (no mutator.)
* **Card inspectCard(int k)** - Accessor for an individual card.  Returns a copy of the card (see admonition for **dealCard()**).  Use a try..catch block when attempting to use k in the cards[] array. If the **IndexOutOfBoundsException**is caught, then return a card with **cardError = true**.

Private Methods

* **static void allocateMasterPack()** - this is a 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.

Recommended 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

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

A of spades / 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 spade

s / 3 of spades / 2 of spades / A of hearts / 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 diamo

nds / K of diamonds / 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 clubs / 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 sp

ades / 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 spades / 2 of spades / A of hearts / K of hearts / Q of hearts / J of he

arts / 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 diamonds /

K of diamonds / Q of diamonds / J of diamonds / T of diamonds / 9 of diamond

s / 8 of diamonds / 7 of diamonds / 6 of diamonds / 5 of diamonds / 4 of di

amonds / 3 of diamonds / 2 of diamonds / A of clubs / K of clubs / Q of clu

bs / 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 /

7 of spades / 6 of diamonds / 2 of hearts / 7 of clubs / 3 of clubs / 7 of diamo

nds / 2 of clubs / J of spades / 4 of diamonds / 2 of spades / J of spades / A o

f spades / 3 of spades / 2 of diamonds / 2 of clubs / 7 of hearts / A of spades

/ A of clubs / K of clubs / 9 of hearts / T of spades / 6 of clubs / Q of clubs

/ 8 of hearts / Q of spades / J of hearts / 5 of hearts / T of hearts / 5 of spa

des / Q of diamonds / 7 of clubs / K of hearts / K of hearts / 5 of spades / T o

f hearts / 3 of diamonds / 7 of spades / 8 of diamonds / A of clubs / 8 of heart

s / 6 of spades / 7 of hearts / Q of clubs / K of spades / 6 of hearts / 5 of cl

ubs / A of hearts / Q of diamonds / 9 of spades / Q of hearts / 2 of diamonds /

9 of spades / 5 of clubs / 4 of spades / 2 of spades / 5 of diamonds / 3 of spad

es / 9 of clubs / 6 of diamonds / J of hearts / T of diamonds / A of diamonds /

A of diamonds / 6 of clubs / 4 of hearts / K of spades / 4 of clubs / 3 of heart

s / 9 of diamonds / T of spades / T of clubs / 8 of spades / T of diamonds / 6 o

f spades / 8 of spades / K of diamonds / K of diamonds / 9 of diamonds / 4 of cl

ubs / Q of spades / 8 of clubs / K of clubs / 8 of clubs / 5 of hearts / 4 of di

amonds / 6 of hearts / 4 of hearts / T of clubs / 7 of diamonds / J of diamonds

/ 8 of diamonds / 2 of hearts / 9 of clubs / 3 of clubs / 9 of hearts / J of clu

bs / 3 of hearts / J of diamonds / 3 of diamonds / 5 of diamonds / A of hearts /

4 of spades / Q of hearts / J of clubs /

A of spades / 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 spade

s / 3 of spades / 2 of spades / A of hearts / 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 diamo

nds / K of diamonds / 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 clubs / 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 /

6 of diamonds / A of spades / 6 of spades / K of spades / 9 of clubs / Q of

spades / J of diamonds / Q of diamonds / 8 of spades / 9 of spades / T of

clubs / 5 of clubs / T of diamonds / A of diamonds / 2 of clubs / 5 of diam

onds / T of spades / J of spades / 9 of diamonds / A of hearts / 8 of heart

s / 8 of clubs / 2 of spades / 3 of diamonds / A of clubs / J of clubs / J

of hearts / 5 of hearts / 9 of hearts / 4 of clubs / 5 of spades / K of di

amonds / Q of hearts / 6 of hearts / 4 of diamonds / 2 of hearts / Q of clu

bs / 3 of clubs / K of hearts / 7 of spades / 4 of spades / 7 of diamonds /

7 of clubs / K of clubs / 3 of spades / 8 of diamonds / 3 of hearts / T o

f hearts / 6 of clubs / 4 of hearts / 7 of hearts / 2 of diamonds /

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

Phase 2: The Deck and Hand Classes

For your second test client, 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 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, since there should not be that much to do in **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 = ( A of spades, 8 of spades, 2 of spades, 9 of hearts, 3 of hearts, T of

diamonds, 4 of diamonds, J of clubs, 5 of clubs )

Hand = ( K of spades, 7 of spades, A of hearts, 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 diamonds, 8 o

f 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 clubs, 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 )

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 )

Here are our hands, from SHUFFLED deck:

Hand = ( J of diamonds, 9 of hearts, Q of spades, 5 of clubs, 8 of spades, 4 of

spades, T of diamonds, Q of clubs, K of clubs )

Hand = ( 4 of diamonds, 3 of clubs, T of spades, 5 of diamonds, 7 of diamonds,

8 of hearts, A of diamonds, 6 of diamonds, 3 of diamonds )

Hand = ( 7 of clubs, 6 of spades, K of diamonds, 6 of clubs, 8 of clubs, 5 of s

pades, 8 of diamonds, 4 of hearts, 4 of clubs )

Hand = ( T of clubs, 2 of diamonds, 6 of hearts, 2 of hearts, 9 of clubs, A of

clubs, K of hearts, A of spades, A of hearts )

Hand = ( 7 of spades, J of clubs, 9 of spades, 3 of spades, J of hearts, 5 of h

earts, K of spades, 3 of hearts )

Hand = ( 9 of diamonds, J of spades, 2 of spades, 2 of clubs, Q of hearts, T of

hearts, Q of diamonds, 7 of hearts )

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

You will be graded, in part, 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.

Comments

Create a header of comments at the top of the program with your name and describing what the program does.

Put in some general comments at the top of each class as well.

 Submission and Grading

This assignment will be submitted via file upload in Canvas.

I will grade the submission according to the rubric posted here.

To view the rubric, select the gear icon. If you're using a mobile device, the rubrics can be found in the course navigation menu.

The grading rubric can be found right below this assignment.