Phase 1: The CardIdentity Class - a superclass

A Public enum Type

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

Private Member Data

Include two members:

char value;

Suit suit;

Methods listed below:

* **CardIdentity() -**The constructor that will put in the default values 'A' and 'spades'.
* **boolean set(char value, Suit suit)** - the mutator that accepts legal values, meaning that the suit will be in the enum, and the value is A, K, Q, J, T, or 2-9.  When bad values are passed, return **false**and do not store them. If good values are passed, they are stored and **true**is returned.
* **Accessors**for **suit and** **value**
* Note:  we don't need individual mutators for value or suit since they would not be needed for this particular class.

Private Methods

* **boolean static isValid(char value, Suit suit)** - a ***static*** 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.

The Card Class - the subclass of CardIdentity

Private Member Data

Include one member:

  boolean cardError;

Public Methods

* **Card() -**Give cardError the default of False, and call default constructor of the super class.
* **Card(char value, Suit suit)**- The constructor should call super() and then the proper mutator, set().  If the mutator returns false, meaning a bad value, then send the defaults:  'A' and 'spades', and set cardError to true, since there was an error.  If the mutator returns true, then set cardError to false.
* **boolean set(char value, Suit suit) -** override the set() method in the CardIdentity class.  This is where you can set the value of cardError depending on the value returned from the super class.
* No mutator for **cardError**- that would not make sense since it is calculated in the set() method.
* **String toString()** - a *stringizer* that the client can use prior to displaying the card.  It provides a clean representation of the card.  If**cardError== true**, it should return correspondingly reasonable reflection of this fact (something like "**[ invalid ]**" rather than a suit and value).
* **Accessor**for **cardError**
* **boolean equals(Card card)** - returns true if all the fields (members) are identical and false, otherwise.

Recommended test of Card class

Instantiate three cards.  Instantiate two legally and a third illegally. Use one default constructor and the others parameter-taking constructors.  Print all three out and confirm. Then make good card bad by **set()** with an illegal value, and turn a card "good" 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 30 or 50 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 if the parameter is an error-free **Card** object and if there is room in the **Hand** object for another card (according to**MAX\_CARDS**).  It returns **false** if the **Hand** was full, and **true** otherwise, *even if card was an invalid (error-containing) card.*  So, if **card** is invalid but there would have been room for it, the method will return **true**, even though it did not add **card** to the hand. 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 (*effectively*, not *physically*) 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 **cardError = true** if **k** is bad.  k is an index for the array.  k needs to be in the bounds of possible values of the array at that moment in time.  If k is outside of the boundaries of the current values, then k is bad and you need to return a bad card.

Recommended 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 = ( )

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

Your Clients and Runs

Since there are ***two phases***, you'll want to have two different test**main()**s (or else one larger test**main()** that has two unrelated parts, ***phase 1*** followed by ***phase 2***).  So supply both **main()**s and label which is which (***phase 1*** or ***phase 2***) and two runs, or one large run clearly labeling the separation between ***phase 1*** and ***phase 2***.

 Submission and Grading

This assignment will be submitted via file upload in Canvas.  Make sure it is [a .txt file with code and output.](https://foothillcollege.instructure.com/courses/22864/pages/final-pre-submission-checks)

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.

Need more help?

[Check out this tutorialLinks to an external site.](https://www.tutorialspoint.com/java/java_inheritance.htm)

Helpful Canvas Guides

The following Canvas Guides will help you successfully submit this assignment and review your feedback:

[STEM Center for tutoring help](https://foothill.edu/stemcenter/)

[How do I upload a file as an assignment submission in Canvas?  (Links to an external site.)](https://community.canvaslms.com/t5/Student-Guide/How-do-I-upload-a-file-as-an-assignment-submission-in-Canvas/ta-p/274)

[How do I view grades in a current course?](https://community.canvaslms.com/docs/DOC-9540-4212724430)

**Assignment (2)**

| Assignment (2) | |
| --- | --- |
| **标准** | **等级** |
| 此标准已链接至学习结果Followed the spec  Program fits the description completely, all classes and methods have correct functionality |  |
| 此标准已链接至学习结果Logic and Syntax  Program has no issues with logic or syntax errors |  |
| 此标准已链接至学习结果Output  Output is according to spec |  |
| 此标准已链接至学习结果Style issues  All style issues are correct |  |
|  | |

**package** assignment1;

**import** java.util.\*;

**public** **class** Foothill

{

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

{

**double** rent, total, costThisYear;

**int** year;

**final** **double** INITIAL\_RENT = 1700;

rent = INITIAL\_RENT; // our initial rent

total = 0; // keeps track of how much we paid "so far"

// loop for five years

**for** (year = 1; year <= 5; year++)

{

costThisYear = rent \* 12;

total += costThisYear;

// now increase the rent in prep for the next year

rent \*= 1.05;

}

System.***out***.println("\nYou will have paid $" + total + " after five years\n");

}

}

**public** **enum** Suit {

***clubs***,

***diamonds***,

***hearts***,

***spades***

}

**class** CardIdentity { // superclass

**private** **char** value;

**private** Suit suit;

**public** CardIdentity() {

value = 'A';

suit = Suit.***spades***;

}

**boolean** set(**char** value, Suit suit) {

**if** (!isValid(value, suit)) {

**return** **false**;

}

**this**.value = value;

**this**.suit = suit;

**return** **true**;

}

Suit getSuit() {

**return** suit;

}

**char** getValue() {

**return** value;

}

**private** **boolean** **static** isValid(**char** value, Suit suit) {

**boolean** checkVal = **false**;

**boolean** checkSuit = **false**;

**if** (value <= '9' && value >= '2') {

checkVal = **true**;

} **else** **if** (value == 'A' || value == 'K' || value == 'Q'

|| value == 'J' || value == 'T') {

checkVal = **true**;

}

**for** (Suit s : Suit.values()) {

**if** (suit.equals(s)) {

checkSuit = **true**;

}

}

**return** checkVal && checkSuit;

}

}

**class** Card **extends** CardIdentity { // subclass

**private** **boolean** cardError;

**public** Card() {

cardError = **false**;

**this**.CardIdentity();

}

**public** Card(**char** value, Suit suit) {

}

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

// Create a myCar object

Car myCar = **new** Car();

// Call the honk() method (from the Vehicle class) on the myCar object

myCar.honk();

// Display the value of the brand attribute (from the Vehicle class) and the value of the modelName from the Car class

System.***out***.println(myCar.brand + " " + myCar.modelName);

}

}