Homework 02: Practice Object-oriented Programming

Instruction

1. Load the initial code.
2. Use Eclipse IDE or any IDE you are familiar with for Java.
3. From the given code, save every folder in src to your IDE. Also save grimora.txt and badtext.txt to your IDE project.
4. Use JUnit to test each part of your code. If you don't know how to use it, find out now!
5. Finish the code according to the instructions.
6. Export your project into a jar file called **HW02_2022_2_{ID}** and place it at the root directory of your IDE project.
    - Example name: **HW02_2022_2_6531234521.jar**
    - **Your JAR file must contain all .java codes. If not, you get 0 points.**
7. Attach your jar file in the assignment page on Mycourseville.

1. Problem Statement: Card Game Deck Manager

(This exercise is based on the game Inscryption by Daniel Mullins Games, released in October 2021.)

A strange, mysterious being has captured you and asked you to play a card game with him. In order to plan your cards and decks, you intend to create a deck manager program to keep things organized in order to have a fighting chance against the mysterious dealer.

This is an example card, a Stoat with 1 power and 3 health. It costs 1 blood to play. Each card has the following properties:

1) The card's name
2) The card's Blood Cost[1] (how many other creatures must be sacrificed to summon this card)
3) The card's power (the damage it will deal)
4) The card's health (if it reaches 0, then the creature will die)

[1] If you played the game before, you might know some cards cost Bones to play instead, we will ignore those cards for now. Sigils will also be ignored for now.

On startup, this is how the program will look like. There are 9 options you can choose to do in this program.

```
Kaycee's Deck Manager
==========
Choose Option:
1. View cards
2. Create new card
3. Import cards from text file
4. Delete card
5. View starter decks
6. Create new starter deck
7. Add to starter deck
8. Remove from starter deck
9. Delete starter deck
10. Quit
```

1) View cards: View the cards that are available to you.

```
0) Squirrel (POW: 0, HP: 1)
Blood Cost: 0
Sacrifices must be made.
-----
1) Stoat (POW: 1, HP: 3)
Blood Cost: 1
Bad play.
-----
2) Wolf (POW: 3, HP: 2)
Blood Cost: 2
The proud Wolf. A vicious contender.
-----
3) Grizzly (POW: 4, HP: 6)
Blood Cost: 3
The monstrous Grizzly. Its form speaks enough of its efficacy.
-----
4) Urayuli (POW: 7, HP: 7)
Blood Cost: 4
This level of brutish strength needs no explanation...
==========
```

2) Create new card: Define a new card that you can use to add to your decks.

```
Input card name.
Geck
Input card cost.
0
Input card power.
1
Input card health.
1
Input flavor text.
The unspiring Geck. Perhaps you can find a use for it?
Added new card: Geck (POW: 1, HP: 1)
```

3) Import cards from text file: Adds multiple new cards at a time. Put in the name of a text file that exists in the project folder, and it will add any cards that don't exist yet to the collection.

```
1 Bone Man,0,1,1,Very fragile. It dies in one turn.
2 Grizzly,3,4,6,The monstrous Grizzly. Its form speaks enough of its efficacy.
3 Bee,0,1,1,BUZZBUZZ
```

```
Name your input file (with extension):
grimora.txt
Bone Man (POW: 1, HP: 1) found in grimora.txt. Added to the collection.
Grizzly (POW: 4, HP: 6) already exists! (Not added.)
Bee (POW: 1, HP: 1) found in grimora.txt. Added to the collection.
```

4) Delete card: Remove a card from your list of cards. Note that you cannot delete any cards that are still in any decks.

```
Choose a card to delete.
0) Squirrel (POW: 0, HP: 1)
1) Stoat (POW: 1, HP: 3)
2) Wolf (POW: 3, HP: 2)
3) Grizzly (POW: 4, HP: 6)
4) Urayuli (POW: 7, HP: 7)
5) Geck (POW: 1, HP: 1)
4
Urayuli (POW: 7, HP: 7) has been deleted.
```

```
Choose a card to delete.
0) Squirrel (POW: 0, HP: 1)
1) Stoat (POW: 1, HP: 3)
2) Wolf (POW: 3, HP: 2)
3) Grizzly (POW: 4, HP: 6)
4) Geck (POW: 1, HP: 1)
1
Stoat (POW: 1, HP: 3) still exists in some decks! It cannot be deleted.
```

5) View starter decks: View decks that you have arranged. Each deck will have a certain number of each card in it.

```
0) Stoat Deck
Stoat (POW: 1, HP: 3) x 1
Wolf (POW: 3, HP: 2) x 2
Squirrel (POW: 0, HP: 1) x 5
Total Cards: 8
-----
1) BEAR DECK
Grizzly (POW: 4, HP: 6) x 3
Squirrel (POW: 0, HP: 1) x 6
Total Cards: 9
==========
```

6) Create new starter deck: Creates a new empty deck that you can place cards in. You cannot use duplicate names.

```
Input deck name.
The Geck
Added new deck: The Geck
==========
```

```
Input deck name.
Stoat Deck
A deck with this name already exists! No new deck added.
==========
```

7) Add to starter deck: Adds cards to a starter deck of your choice.

```
Choose deck to add to.
0) Stoat Deck
Stoat (POW: 1, HP: 3) x 1
Wolf (POW: 3, HP: 2) x 2
Squirrel (POW: 0, HP: 1) x 5
Total Cards: 8
-----
1) BEAR DECK
Grizzly (POW: 4, HP: 6) x 3
Squirrel (POW: 0, HP: 1) x 6
Total Cards: 9
-----
2) The Geck
EMPTY DECK
Total Cards: 0
2
Choose card to add to deck.
0) Squirrel (POW: 0, HP: 1)
1) Stoat (POW: 1, HP: 3)
2) Wolf (POW: 3, HP: 2)
3) Grizzly (POW: 4, HP: 6)
4) Geck (POW: 1, HP: 1)
4
Choose how many to add.
1
Added 1 copy of Geck (POW: 1, HP: 1) to The Geck.
```

8) Remove from starter deck: Removes cards from a starter deck of your choice. You can only remove cards from a deck if there is at least 1 copy of that card in the deck.

```
Choose a deck to remove from.
0) Stoat Deck
Stoat (POW: 1, HP: 3) x 1
Wolf (POW: 3, HP: 2) x 2
Squirrel (POW: 0, HP: 1) x 5
Total Cards: 8
-----
1) BEAR DECK
Grizzly (POW: 4, HP: 6) x 3
Squirrel (POW: 0, HP: 1) x 6
Total Cards: 9
-----
2) The Geck
Geck (POW: 1, HP: 1) x 1
Total Cards: 1
1
Choose card to remove from deck.
0) Squirrel (POW: 0, HP: 1)
1) Stoat (POW: 1, HP: 3)
2) Wolf (POW: 3, HP: 2)
3) Grizzly (POW: 4, HP: 6)
4) Geck (POW: 1, HP: 1)
0
Choose how many to remove.
3
Removed 3 copies of Squirrel (POW: 0, HP: 1) to BEAR DECK.
If you remove more copies than there are in the deck, then it will no longer exist in the deck.
```

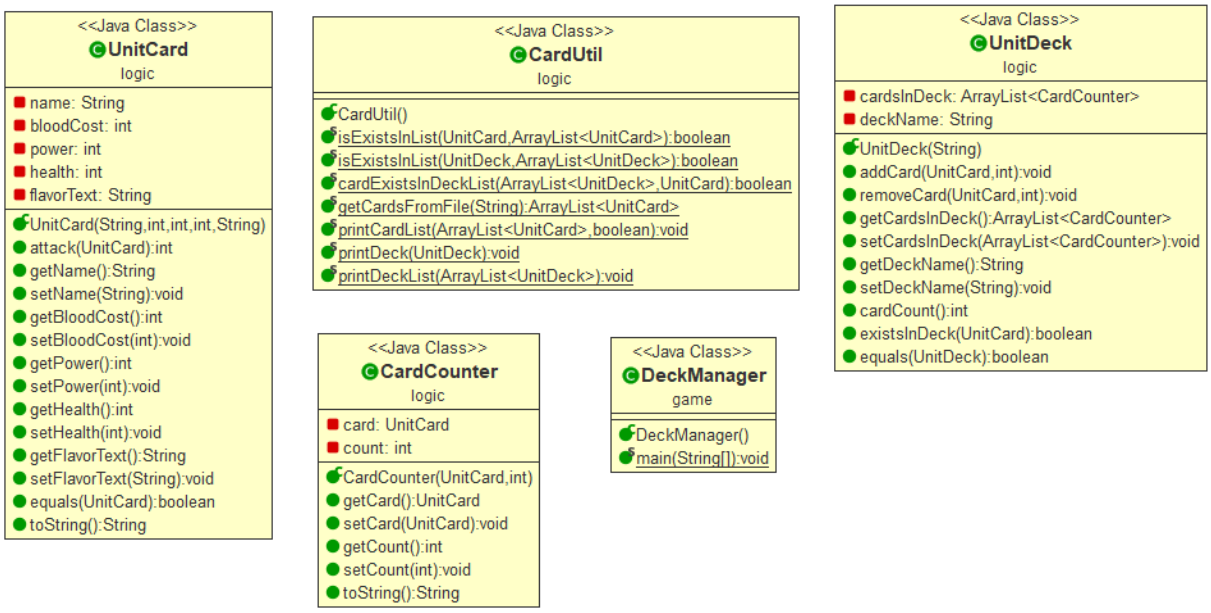
```
Choose a deck to remove from.
0) Stoat Deck
Stoat (POW: 1, HP: 3) x 1
Wolf (POW: 3, HP: 2) x 2
Squirrel (POW: 0, HP: 1) x 5
Total Cards: 8
-----
1) BEAR DECK
Grizzly (POW: 4, HP: 6) x 3
Squirrel (POW: 0, HP: 1) x 3
Total Cards: 6
-----
2) The Geck
Geck (POW: 1, HP: 1) x 1
Total Cards: 1
1
Choose card to remove from deck.
0) Squirrel (POW: 0, HP: 1)
1) Stoat (POW: 1, HP: 3)
2) Wolf (POW: 3, HP: 2)
3) Grizzly (POW: 4, HP: 6)
4) Geck (POW: 1, HP: 1)
2
Choose how many to remove.
3
This card doesn't exist in this deck!
==========
```

9) Delete starter deck: Deletes a deck from your collection.

```
Choose a deck to delete.
0) Stoat Deck
Stoat (POW: 1, HP: 3) x 1
Wolf (POW: 3, HP: 2) x 2
Squirrel (POW: 0, HP: 1) x 5
Total Cards: 8
-----
1) BEAR DECK
Grizzly (POW: 4, HP: 6) x 3
Squirrel (POW: 0, HP: 1) x 3
Total Cards: 6
-----
2) The Geck
Geck (POW: 1, HP: 1) x 1
Total Cards: 1
2
The Geck has been deleted.
```

2. Implementation Details:

The diagram of the program is illustrated below. There are 5 classes: UnitDeck, CardCounter, UnitCard, CardUtil, and DeckManager.



** Note that Access Modifier Notations can be listed below*
** Also note that while other methods have been provided, only methods that you have to implement are listed here*

+ (public)
 # (protected)
- (private)

2.1 Class UnitCard (package logic)

This class represents a card in the collection. Each card has a name, some properties, and a flavor text.

2.1.1 Fields

- String name	The name of this card. It cannot be blank.
- int bloodCost	The blood cost required to summon this card. It cannot be less than 0.
- int power	The card's base power. It cannot be less than 0.
- int health	The card's base health. It cannot be less than 1.
- String flavorText	The card's flavor text.

2.1.2 Constructors

+ UnitCard(String name, int bloodCost, int power, int health, String flavorText)	**/*Fill Code*/** Initialize **all fields**. **Note: You should use setter to set the value of all fields in order to handle negative value cases and special cases.**

2.1.3 Methods

+ void setName(String name)	**/*Fill Code*/** This method will set name of the card. If the String is empty, the method will set the name as "Creature". **Hint: There is a non-static method called isBlank() in the String class. It returns true if a String consists of only whitespace.** **Example:** **String exampleString = " ";** **exampleString.isBlank() will return true.**

+ void setBloodCost(int bloodCost)	**/*Fill Code*/** This method will set the blood cost of the card. If the blood cost is less than 0, it will be set as 0.
+ void setPower(int power)	**/*Fill Code*/** This method will set the power of the card. If the power is less than 0, it will be set as 0.
+ void setHealth(int health)	This method will set the health of the card. If the health is less than 1, it will be set as 1.
+ boolean equals(UnitCard other)	**/*Fill Code*/** This method checks if this card is the same as the other given card. To qualify as the same card, both cards must have the same name. Please note that the type of method parameter is UnitCard. This method is designed for testing only UnitCard against one another.
+ String toString()	Returns the object as a string.
Getter & Setter methods for all other variables	**/*Fill Code*/**

2.2 Class CardCounter (package logic)

This class is an object that is used to count cards in a deck. For each CardCounter object, there is a Card that is being counted, and the number of that card that exists in a deck.

2.2.1 Fields

- UnitCard card	The card being counted by this counter.
- int count	The number of cards being counted by this counter. It cannot be less than 0.

2.2.2 Constructors

+ CardCounter(UnitCard card, int count)	**/*Fill Code*/** Initialize **all fields**. **Note: You should use setter to set the value**

	of all fields in order to handle negative value cases and special cases.

2.2.3 Methods

+ void setCount(int count)	**/*Fill Code*/** This method will set the count of the card counter. If the count is less than 0, it will be set as 0.
+ String toString()	Returns the object as a string.
Getter & Setter methods for all other variables	**/*Fill Code*/**

2.3 Class UnitDeck (package logic)

This card represents a deck in your collection. Each deck will have some amount of Card Counter to represent the cards in the deck, and the count of each card in the deck.
/*You need to write this entire class from scratch. */

2.3.1 Fields

- ArrayList<CardCounter> cardsInDeck	The list of Card Counters that counts the number of each card type in the deck.
- String deckName	The name of this deck. It cannot be blank.

2.3.2 Constructors

+ UnitDeck(String deckName)	**/*Fill Code*/** Initialize cardsInDeck as an empty ArrayList. Also, initialize the deck name. Don't forget to use the proper getter/setter to handle special cases.

2.3.3 Methods

+ void addCard(UnitCard newCard, int count)	**/*Fill Code*/**

	This method adds new cards to the deck, equal to the given number. If the given count is not a positive integer, then it does nothing. If this card already exists in the deck, it adds the count to the CardCounter corresponding to the pre-existing card in cardsInDeck. If this card does **NOT** already exist in the deck, then initialize a new CardCounter corresponding to this card with the given count, then add it to cardsInDeck.
+ void removeCard(UnitCard toRemove, int count)	**/*Fill Code*/** This method removes cards from the deck, equal to the given number. If the given count is not a positive integer or if the given card doesn't exist in the deck, then it does nothing. If this card exists in the deck, it decreases the count from the CardCounter of this existing card in cardsInDeck according to the method's count parameter. If the count reaches 0 this way, then the CardCounter must also be removed from cardsInDeck.
+ int cardCount()	**/*Fill Code*/** This method counts the number of cards in this deck. The number of cards is the total sum of each CardCounter's count in cardsInDeck.
+ boolean existsInDeck(UnitCard card)	**/*Fill Code*/** This method checks if the given card exists in this deck. The card exists in the deck if there is a CardCounter with a count of 1 or greater in cardsInDeck.
+ boolean equals(UnitDeck other)	**/*Fill Code*/** This method checks if this deck is the same as

	the other given deck. To qualify as the same deck, both decks must have the same name. Please note that the type of method parameter is UnitDeck. This method is designed for testing only UnitDeck against one another.
Getter & Setter methods for deckName and cardsInDeck.	**/*Fill Code*/** deckName cannot be empty. If the given deckName is empty, then set the deckName as "Untitled Deck". **Hint: There is a non-static method called isBlank() in the String class. It returns true if a String consists of only whitespace.** **Example:** **String exampleString = " ";** **exampleString.isBlank() will return true.**

2.4 Class CardUtil (package logic)

This class contains static utility methods that are useful in one or more classes. Most of these have to do with manipulating cards and decks.

2.4.1 Fields

This class only contains static methods and do not have any fields.

2.4.2 Constructors

This class does not need a constructor.

2.4.3 Methods

+ boolean isExistsInList(UnitCard card, ArrayList<UnitCard> list)	**/*Fill Code*/** This method checks if the given card exists in the given ArrayList. Return true if it does, return false otherwise.
+ boolean isExistsInList(UnitDeck deck, ArrayList<UnitDeck> list)	**/*Fill Code*/** This method checks if the given deck exists in the given ArrayList. Return true if it does, return false otherwise.

+ boolean cardExistsInDeckList(ArrayList<UnitDeck> deckList, UnitCard cardToTest)	**/*Fill Code*/** This method checks if the given deckList contains any CardCounter with the given UnitCard in it. Return true if it does, return false otherwise.
+ ArrayList<UnitCard> getCardsFromFile(String filename)	**/*Fill Code*/** Three lines of this function has been given to you. Replace "//TODO: Fill Code" with the following algorithm: First, create a scanner to read fileToRead. Then, while the scanner still has lines for it to read, get the string of the line. Split it with comma to get an array, and create a new card like this: - The 0th member of the array is the Card Name. - The 1st member of the array is the Blood Cost. - The 2nd member of the array is the Power. - The 3rd member of the array is the Health. - The 4th member of the array is the Flavor Text. Finally, return the list of all cards created this way. Don't forget to close the scanner as well! If the file is not found or if there is error while reading the file (for example, the text file isn't properly formatted), this method should return null instead. Hint: You might need to use try catch or throw exceptions here. IMPORTANT: There is no JUnit test for this method. Your program must be able to read a card file during its actual use.

+ void printCardList(ArrayList<UnitCard> cardList, boolean verbose)	Prints a list of cards as formatted string. If verbose is set to true, it prints the flavor text as well.
+ void printDeck(UnitDeck ud)	Prints the list of cards in the deck as a formatted string list of CardCounters.
+ static void printDeckList(ArrayList<UnitDeck> deckList)	Prints all the decks in the given list as formatted string.

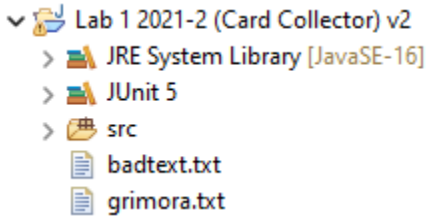
2.5 Class DeckManager (package game)

This is the class where you can run the deck manager from.

3. Test Scenario

(User input is in green.)

Since other methods can be tested in the JUnit, we will only display how to test the card import function here. In order to test the import function, we have provided two text files. The file "grimora.txt" contains correctly formatted string, while "badtext.txt" contains incorrectly formatted string. We will also attempt to read "nonexist.txt", which does not exist in the project folder.



This is the current state of the project, with the two mentioned text files. Note that "nonexist.txt" does not exist.

The file "grimora.txt" looks like this.

```
1 Bone Man,0,1,1,Very fragile. It dies in one turn.
2 Grizzly,3,4,6,The monstrous Grizzly. Its form speaks enough of its efficacy.
3 Bee,0,1,1,BUZZBUZZ
```

Note that if this file is imported, Bone Man and Bee should be added since they don't already exist, and Grizzly should not be added since it already exists in the program's initial state.

The file "badtext.txt" looks like this.

```
1 Head Chef,1,1,1,He brings us the food. We should be thankful.
2 Ham,Cheese,Croissant,Bacon,Egg
3 Pasta,Sauce,Spinach,Cheese,Ricotta
4 Pasta,Sauce,Cheese,Ricotta
5 Pasta,Sauce,Cheese
6 Spicy Chicken,Lettuce,Bacon,Tomatoes,Pickles,Top Bun
7 Corn Husk,Cornmeal,Spicy Chicken,Wrap
```

Since this would result in an error, importing this file should not add any new cards at all. Head Chef should not exist in your collection after importing cards from this text file.

```
Kaycee's Deck Manager
==========
Choose Option:
1. View cards
2. Create new card
3. Import cards from text file
4. Delete card
5. View starter decks
6. Create new starter deck
7. Add to starter deck
8. Remove from starter deck
9. Delete starter deck
10. Quit
1
0) Squirrel (POW: 0, HP: 1)
Blood Cost: 0
Sacrifices must be made.
-----
1) Stoat (POW: 1, HP: 3)
Blood Cost: 1
Bad play.
-----
2) Wolf (POW: 3, HP: 2)
Blood Cost: 2
The proud Wolf. A vicious contender.
-----
3) Grizzly (POW: 4, HP: 6)
Blood Cost: 3
The monstrous Grizzly. Its form speaks enough of its efficacy.
-----
4) Urayuli (POW: 7, HP: 7)
Blood Cost: 4
This level of brutish strength needs no explanation...
==========
```

Here, we test the initial state of the card collection. **Note that the Grizzly exists so it should not be added when imported.**

```
Choose Option:
1. View cards
2. Create new card
3. Import cards from text file
4. Delete card
5. View starter decks
6. Create new starter deck
7. Add to starter deck
8. Remove from starter deck
9. Delete starter deck
10. Quit
3
Name your input file (with extension):
grimora.txt
Bone Man (POW: 1, HP: 1) found in grimora.txt. Added to the collection.
Grizzly (POW: 4, HP: 6) already exists! (Not added.)
Bee (POW: 1, HP: 1) found in grimora.txt. Added to the collection.
Choose Option:
1. View cards
2. Create new card
3. Import cards from text file
4. Delete card
5. View starter decks
6. Create new starter deck
7. Add to starter deck
8. Remove from starter deck
9. Delete starter deck
10. Quit
1
0) Squirrel (POW: 0, HP: 1)
Blood Cost: 0
Sacrifices must be made.
-----
1) Stoat (POW: 1, HP: 3)
Blood Cost: 1
Bad play.
-----
2) Wolf (POW: 3, HP: 2)
Blood Cost: 2
The proud Wolf. A vicious contender.
-----
3) Grizzly (POW: 4, HP: 6)
Blood Cost: 3
The monstrous Grizzly. Its form speaks enough of its efficacy.
-----
4) Urayuli (POW: 7, HP: 7)
Blood Cost: 4
This level of brutish strength needs no explanation...
-----
5) Bone Man (POW: 1, HP: 1)
Blood Cost: 0
Very fragile. It dies in one turn.
-----
6) Bee (POW: 1, HP: 1)
Blood Cost: 0
BUZZBUZZ
```

This is what happens if you import a valid text file. This step tests if your function is able to properly read imported text files. **Note that the Grizzly does not appear twice.**

```
Choose Option:
1. View cards
2. Create new card
3. Import cards from text file
4. Delete card
5. View starter decks
6. Create new starter deck
7. Add to starter deck
8. Remove from starter deck
9. Delete starter deck
10. Quit
3
Name your input file (with extension):
nonexist.txt
Cannot find file!
File error! No new card added.
```

This is what happens if you import a text file that does not exist. This step tests if your function can handle non-existent files. **No new cards should be added.**

```
Choose Option:
1. View cards
2. Create new card
3. Import cards from text file
4. Delete card
5. View starter decks
6. Create new starter deck
7. Add to starter deck
8. Remove from starter deck
9. Delete starter deck
10. Quit
3
Name your input file (with extension):
badtext.txt
File contains string with incorrect format!
File error! No new card added.
Choose Option:
1. View cards
2. Create new card
3. Import cards from text file
4. Delete card
5. View starter decks
6. Create new starter deck
7. Add to starter deck
8. Remove from starter deck
9. Delete starter deck
10. Quit
1
0) Squirrel (POW: 0, HP: 1)
Blood Cost: 0
Sacrifices must be made.
-----
1) Stoat (POW: 1, HP: 3)
Blood Cost: 1
Bad play.
-----
2) Wolf (POW: 3, HP: 2)
Blood Cost: 2
The proud Wolf. A vicious contender.
-----
3) Grizzly (POW: 4, HP: 6)
Blood Cost: 3
The monstrous Grizzly. Its form speaks enough of its efficacy.
-----
4) Urayuli (POW: 7, HP: 7)
Blood Cost: 4
This level of brutish strength needs no explanation...
-----
5) Bone Man (POW: 1, HP: 1)
Blood Cost: 0
Very fragile. It dies in one turn.
-----
6) Bee (POW: 1, HP: 1)
Blood Cost: 0
BUZZBUZZ
```

The following card should not exist:

```
7) Head Chef (POW: 1, HP: 1)
Blood Cost: 1
He brings us the food. We should be thankful.
```

This is what happens if you import a text file with bad string format. This step tests if your function can handle files with incorrect format. **Note that no new cards are added at all, even if the first line in the text file is valid.**

4. Criteria

4.1 JUnit

UnitCardTest

- testConstructor (2)
- testBadConstructor (2)
- testSetName (1)
- testSetBloodCost (1)
- testSetPower (1)
- testSetHealth (1)
- testEquals (1)

CardCounterTest

- testConstructor (2)
- testBadConstructor (2)
- testSetCount (1)

UnitDeckTest

- testConstructor (2)
- testBadConstructor (2)
- testSetName (1)
- testAddCardNonExist (1)
- testAddCardExist (1)
- testRemoveNonExistingCard (1)
- testRemoveExistingCard (1)
- testCardCount (1)
- testExistsInDeck (1)
- testEquals (1)

CardUtilTest

- testIsExistsInListCard (1)
- testIsExistsInListDeck (1)
- testCardExistsInDeckList (1)

4.2 Exception (getCardsFromFile Functionality)

File With Valid String Format

- New cards are added (1)
- Cards that already exist aren’t added (1)

File With Invalid String Format

- The program shows error message and can continue running (1)
- No new cards are added (1)

File That Doesn’t Exist

- The program shows error message and can continue running (1)
- No new cards are added (1)