Shay Snyder

Honors 1260 – Spring 2020

Credit Card List w/ Text Files and Exceptions

Problem: Spring 2020 Honors 1260 Project 4: CreditCardList with Text Files and Exceptions

### List of Inputs, Outputs, and Processing Required

INPUTS

* Allow the user to add a CreditCard object to an ArrayList of CreditCard objects
* Allow the user to create a CreditCard list from a Credit Card text file
* Use a menu system to afford the user a visually appealing interface that makes interfacing with the program and its various methods much easier.

OUTPUTS

* A CreditCard object from a given index **n**
* A list of all CreditCard objects with a particular card holder
* A list of all CreditCard objects that have not expired at the time the program has run
* For the Find and Retrieve methods, display the results to verify they are correct
* Display all credit card information about every credit card in the list
* A text file representing the user’s CreditCardList

PROCESSING

* Take data from a CreditCard text file and create a CreditCardList
* Retrieve the CreditCard currently in position **n** in the list
* Remove a CreditCard from the list
* Find a CreditCard with a particular number in the list
* Find all CreditCard objects that have not expired at the time the program is run
* Sort the CreditCard objects in the list by credit card number
* Sort the CreditCard objects in the list by card holder’s name
* All methods should function properly even if the list is empty or the requested item is not in the list
* Export all CreditCard objects in the CreditCard list to a text file

### Identification of Classes and Their Responsibilities

**Class name: CreditCardListDriver**

Responsibilities:

* Allow the user to add a CreditCard object to an ArrayList of CreditCard objects
* Use a menu system to afford the user a visually appealing interface that makes interfacing with the program and its various methods much easier.
* Remove CreditCard objects from their wallet based on index
* Get a subset of their cards based on card number, card holder, index, or expiration status
* Sort their cards based on card number or card holder
* The ability to view all cards within their wallet

**Class name:** **CreditCardList**

Responsibilities:

* Contain a private attribute representing an ArrayList of CreditCard objects
* Add a collection of CreditCards to the list from a text file
* Add a CreditCard to the list
* Retrieve the CreditCard currently in position **n** in the list
* Export the CreditCardList to a text file.
* Remove a CreditCard from the list
* Find all CreditCard Objects with a particular number from the list
* Find all CreditCard objects that hae not expired at the time the program is run
* Sort the CreditCard objects in the list by credit card number
* Sort the CreditCard objects in the list by the card holder’s number

### UML Class Diagram

**A screenshot of text

Description automatically generated**

**Class:** CreditCardListDriver

**Method:** main(args[] : String) : void

**Desciption:** The main() method in the driver will act as the basis of functionality for project 4, a.k.a every method call will stem from this point

**Class:** CreditCardListDriver

**Method:** intro() : void

**Description:** The intro() method will use JOptionPane to display a promt to the user that introduces them to the program and details its functionality

**Class:** CreditCardListDriver

**Method:** core() : void

**Description:** The core() method will be used to interact with the CreditCardList class. Upon calling this method, the program will determine if the user’s CreditCardList needs to be saved. If their list needs to be saved, it will call the CreditCardList’s save() method. Then the method will continue by allowing the user to create a custom list of credit cards and manipulate the list as they wish. A JOptionPane Dialog Box will be used to allow the user the opportunity to decide how the program will behave. This will be housed in a while loop that will run until the user decides to end the program.

Decision = show JOptionPane dialog box with input, remove, retrieve, and sort options

If (decision = input)

Prompt the user to decide if they would like to add an assortment of cards from a text file or add them individually.

If the user decided to import cards from a list, use the JFileChooser to allow them to select an appropriate file. Then call the CreditCardList arg- constructor with the file path. Everytime this method is called, save the previous CreditCards to a text file first.

If an error is thrown at any point throughout this process, gracefully catch it and tell the user what happened.

If the user decided to import Use JOptionPane to allow the user to enter Cardholder’s name, card number, and expiration date. Then call the CreditCardList addCard() method to add the credit card to the ArrayList

Else if (decision = remove)

Use JOptionPane to allow the user to enter an int that represents the index of the card they would like to remove

Call the CreditCardListDriver removeCard() method

Else if (decision = retrieve)

Call the CreditCardListDriver getCards() method

Else if (decision = sort)

Call the CreditCardListDriver sortCards() method

Else if (decision = save)

Use the JFileChooser to allow the user to export their CreditCards to a specified text file. If an error is thrown at any point throughout this process, gracefully catch it and tell the user what happened.

**Class:** CreditCardListDriver

**Method:** outro() : void

**Description:** The outro() method will allow the user to select the final save location of their CreditCardList. Then use JOptionPane to display a prompt to the user that shows our appreciation for their using of the program.

**Class**: CreditCardList Driver

**Method**: addCard()

**Description**: Use JOptionPane to allow the user to enter four strings, card holder name, card number, expiration month, and expiration year. The values will be verified, if they are deemed incorrect, the user will be repeatidly prompted until they enter a correct value. Once all values are correct, the CreditCardList’s addCard() method will be called to add the desired card to the wallet. Once the card is added, return to the main screen.

**Class**: CreditCardListDriver

**Method**: getCardByName()

**Description**: Use JOptionPane to allow the user to enter a String variable representing the card holder name they would like to program to search for. If their input is deemed incorrect, the program will prompt the user to enter another value until a valid name is inputed. Once a valid name is obtained, the program will call the CreditCardList’s getCardByName method() with the result being displayed to the user via JOptionPane.

**Class**: CreditCardListDriver

**Method**: getCardByNumber()

**Description**: Use JOptionPane to allow the user to enter a String variable representing the card number they would like to program to search for. If their input is deemed incorrect, the program will prompt the user to enter another value until a valid number is inputed. Once a valid number is obtained, the program will call the CreditCardList’s getCardByNumber method() with the result being displayed to the user via JOptionPane.

**Class**: CreditCardListDriver

**Method**: getCardExpir()

**Description**: The CreditCardListDriver’s getCardExpir() method will call the CreditCardList’s getCardsByExpir() method which will return a String containing all cards that are expired at the time of program execution. The resulting String will be displayed to the user via JOptionPane.

**Class**: CreditCardListDriver

**Method**: isNumeric()

**Description**: The CreditCardListDriver’s isNumeric() method accepts a String argument. A try-catch block will be used to determine if the string can be parsed to a Long. If the number can be parsed to a Long, return true. If the number cannot be parsed to a Long, return false.

**Class**: CreditCardListDriver()

**Method**: removeCard()

**Description**: Use JOptionPane to allow the user to enter a String variable representing the index of the credit card they would like to remove. If their input is deemed incorrect, the program will prompt the user to enter another index until a valid number is inputed. Once a valid number is obtained, the program will call the CreditCardList’s removeCard method() with a success or fail prompt being displayed to the user via JOptionPane.

**Class:** CreditCardListDriver

**Method:** getCards() : void

**Description:** The getCards() will be called when the user selects to retrieve cards in the core() method. A JOptionPane dialog box will be used to give them the opportunity to get cards by expiration date, index, cardholder name, card number, or all of them.

Decision = show JOptionPane dialog box with expiration, index, name, number, or all options

If (decision = expiration)

Call the CreditCardList getCardsByExpir() method

Show the returned ArrayList of CreditCard objects

Else if (decision = index)

Call the CreditCardListDriver’s getCardByIndex method

Else if (decision = name)

Call the CreditCardListDriver’s getCardsByName Method

Else if (decision = number)

Call the CreditCardListDriver’s getCardsByNumber Method

Else if (decision = number)

Call the CreditCardList getAllCards() method

Show the returned ArrayList of CreditCard objects

**Class:** CreditCardListDriver

**Method:** sortCards() : void

**Description:** The sortCards() method will be called from the core() method in the driver.

Decision = JOptionPane dialog box (sort by number or sort by name)

If (decision = sort by number)

Call the CreditCardList sortCardsByNumber() method

Else if (decision = sort by name)

Call the CreditCardList sortCardsByName() method

**Class**: CreditCardListDriver

**Method**: initializeCreditCardList() : CreditCardList

**Description**: This method allows the user the option to import a CreditCardList from a text file. If they choose to do so, call the CreditcardListDriver’s importPrevious list method. If they choose not to use a text file, the CreditCardList default constructor will be called.

**Class**: CreditCardListDriver

**Method**: importPreviousList() : CreditCardList

**Description**: The user will be prompted with a JFileChoose.showOpenDialogBox to select which file they would like to use. This path will be passed to the arg constructor in the CreditCardList class.

**Class:** CreditCardList

**Method:** CreditCardList()

**Description:** Default constructor for the CreditCardList class. An empty ArrayList of CreditCard objects is created

**Class:** CreditCardList

**Method:** CreditCard(path : String)

**Description:** Arg-constructor for the CreditCardList class. Create an array list from a text files at the argumented file path

Make sure the fpath is valid

Create a new install of the File and Scanner class.

Loop through each line of the file:

Extract the name, card number, and expiration date and assign them to temp variable

Use the addCard() method

Repeat until all cards have been added

Close the file

**Class:** CreditCardList

**Method:** save()

**Description:** save the entire CreditCardList to a text file at the imported or default file path

If the list was previously imported, use that file path. Otherwise, use the default file path.

Create a new FileWriter object

Create a new PrintWriter object

Loop through every CreditCard in the CreditCardList:

Add the name, number, and expiration date of the card on a unique line

Repeat for every card

saveNeeded = False

**Class:** CreditCardList

**Method:** save(path : String)

**Description:** save the entire CreditCardList to a text file at the argumented location

Make sure the filepath is correct

Create a new FileWriter object

Create a new PrintWriter object

Loop through every CreditCard in the CreditCardList:

Add the name, number, and expiration date of the card on a unique line

Repeat for every card

saveNeeded = False

**Class:** CreditCardList

**Method:** addCard(cardHolderName : String, cardNumber : String, expirationDate : String) : void

**Description:** use the arguments to create a new CreditCard object and put it in the array list

CreditCard card = new CreditCard(name, number, date)

Cards.add(card)

saveNeeded = True

**Class:** CreditCardList

**Method:** removeCard(index : int) : boolean

**Description:** remove the card object at index n

Try Card.remove(n)

If successful, return true

If unsuccessful, return false

saveNeed = True

**Class:** CreditCardList

**Method:** getCardsByExpir() : String

**Description:** Create a new StringBuilder object to hold all cards that are expired

Create a new StringBuilder object

For each item in the copied array list:

Use the isCardExpired() method

If card is expired:

Add the CreditCard object to the StringBuilder object

Return the StringBuilder object as a String

**Class:** CreditCardList

**Method:** getCardsByIndex(index : int) : String

**Description:** return the generated report of a credit card at a given index n within the list

If the card is available at a given index: return Card.generateReport()

If the index is out of range: return “No card is available at index n”

**Class:** CreditCardList

**Method:** getCardsByName(name : String) : String

**Description:** use the argumented String and determine if any of the CreditCard objects in the cards list have the same cardholder name, return those CreditCard objects whose cardholder’s name matches in the form of a StringBuilder object.

Create a new StringBuilder object to hold all matches

For each item in the credit card list:

If (card holder name = desired name):

Add the reditCard object to the newly created StringBuilder object

Return the StringBuilder object as a String

**Class:** CreditCardList

**Method:** getCardsByNumber(number : String) : String

**Description:** use the argumented String and determine if any of the CreditCard objects in the cards list have the same card number, return those CreditCard objects whose card number matches in the form of a new StringBuilder object

Create a new StringBuilder object to hold all matches

For each item in the credit card list:

If (card number = desired number):

Add the CreditCard object to the StringBuilder object

Return the StringBuilder object in the form of a String

**Class:** CreditCardList

**Method:** getAllCards() : ArrayList<CreditCard>

**Description:** return the entire ArrayList

Create a new StringBuilder object to hold a detailed view of all cards

For each item in the credit card list:

Add the CreditCard object to the StringBuilder object

Return the StringBuilder object in the form of a String

**Class:** CreditCardList

**Method:** sortCardsByNumber() : void

**Description:** Use the Selection Sort Algorithm along with the CreditCard object's compareNumber method to sort the cards by number

* The outer loop’s purpose is to keep track of the last index within the user's CreditCardList. As the loop progresses, this value willbe decremented to account for the action of swapping the CreditCard object whose card number has the max numerical value to the last position.
* The inner loop is used to sort through the entire list to find the maxiumum value. As the algorithm progresses, the indecies available to this loop will shorten as the maxiumum values will swapped to the end of the ArrayList and no longer need to be accessed.
* Use the CreditCardList's compareNumber method to determine if the value at index i is greater than the value in the last position of the list (n), swap the CreditCard object at i w/ the CreditCardObject at n. As this method iterates over the entire list, the maxiumum will be determined since every value is compared with the last CreditCard object in the list.
* Use the Collections.swap() method to swap the CreditCard with a greater number at index i to the CreditCard with a lesser value at index n

saveNeeded = true

**Class:** CreditCardList

**Method:** sortCardsByName() : void

**Description:** Use the Selection Sort Algorithm along with the CreditCard object's compareName method to sort the cards by the name of their holder

* The outer loop’s purpose is to keep track of the last index within the user's CreditCardList. As the loop progresses, this value will be decremented to account for the action of swapping the CreditCard objects whose card holder has the max alphabetical value to the last position.
* The inner loop is used to sort through the entire list to find the maxiumum alphabetical value. As the algorithm progresses, the indicies available to this loop will shorten as the maxiumum values will swapped to the end of the ArrayList and no longer need to be accessed
* Use the CreditCardList's compareName method to determine if the alphabetical value at index i is greater than the alphabetical value in the last position of the list (n), swap the CreditCard object at i w/ the CreditCardObject at n. As this method iterates over the entire list, the maxiumum will be determined since every value is compared with the last CreditCard object in the list
* Use the Collections.swap() method to swap the CreditCard with a greater number at index i to the CreditCard with a lesser value at index n

saveNeed = true

**Class**: CreditCardList

**Method**: getImported() : boolean

**Description**: return the boolean attribute specifiying whether the CreditCardList was imported

**Class:** CreditCardList

**Method:** getSaveNeeded() : boolean

**Description:** return the boolean attribute specifying whether the CreditCardList needs to be saved

**Class:** CreditCardList

**Method:** deleteTempFile() : void

**Description:** if a temporary file was used during runtime, remove that file from the computer

**Class**: CreditCard

**Method**: toString() : String

**Description**: return a String attribute containing the card holder name, card number, and expiration will a pipe delimiter. This variable will be used to export the card to a text file.

### Test Cases

This information will be initially entered into the program before . Methods and expected outputs will be listed below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Cardholder’s Name** | **Card Number** | **Expiration Date** | **Expected Output** |
| John Doe | 378282246310005 | 01/2020 | Expired |
| John Cena | 371449635398431 | 02/2020 | Expired |
| Katherine Booher | 378734493671000 | 04/2019 | Expired |
| Justin Timberlake | 9610591081018250 | 12/2021 | Invalid |
| Krikor Faddey | 6011111111111117 | 04/2023 | Valid |
| Dmittrii Sasha | 6011000990139424 | 05/2022 | Valid |
| Leonty Natali | 5425233430109903 | 04/2023 | Valid |
| Polina Lazar | 4917484589897107 | 05/2025 | Valid |

**Test Case #1 (Testing the addCard() method)**

Input the following card using the addCard() method

|  |  |  |  |
| --- | --- | --- | --- |
| **Cardholder’s Name** | **Card Number** | **Expiration Date** | **Expected Output** |
| Bob the Builder | 2222 4053 4324 8877 | 04/2023 | Valid |

Expected output when calling the getAllCards() method:

|  |  |  |  |
| --- | --- | --- | --- |
| **Cardholder’s Name** | **Card Number** | **Expiration Date** | **Expected Output** |
| John Doe | 378282246310005 | 01/2020 | Expired |
| John Cena | 371449635398431 | 02/2020 | Expired |
| Katherine Booher | 378734493671000 | 04/2019 | Expired |
| Justin Timberlake | 9610591081018250 | 12/2021 | Invalid |
| Krikor Faddey | 6011111111111117 | 04/2023 | Valid |
| Dmittrii Sasha | 6011000990139424 | 05/2022 | Valid |
| Leonty Natali | 5425233430109903 | 04/2023 | Valid |
| Polina Lazar | 4917484589897107 | 05/2025 | Valid |
| Bob the Builder | 2222 4053 4324 8877 | 04/2023 | Valid |

**Test Case #2 (Test the removeCard() method)**

Expected output of Cards.remove(0):

|  |  |  |  |
| --- | --- | --- | --- |
| **Cardholder’s Name** | **Card Number** | **Expiration Date** | **Expected Output** |
| John Cena | 371449635398431 | 02/2020 | Expired |
| Katherine Booher | 378734493671000 | 04/2019 | Expired |
| Justin Timberlake | 9610591081018250 | 12/2021 | Invalid |
| Krikor Faddey | 6011111111111117 | 04/2023 | Valid |
| Dmittrii Sasha | 6011000990139424 | 05/2022 | Valid |
| Leonty Natali | 5425233430109903 | 04/2023 | Valid |
| Polina Lazar | 4917484589897107 | 05/2025 | Valid |

**Test Case #3 (Test the getCardsByExpir() method)**

Expected result of Cards.getCardsByExpir()

|  |  |  |  |
| --- | --- | --- | --- |
| **Cardholder’s Name** | **Card Number** | **Expiration Date** | **Expected Output** |
| John Doe | 378282246310005 | 01/2020 | Expired |
| John Cena | 371449635398431 | 02/2020 | Expired |
| Katherine Booher | 378734493671000 | 04/2019 | Expired |

**Test Case #4 (Test getCardByIndex)**

Expected result of Cards.getCardByIndex(0):

|  |  |  |  |
| --- | --- | --- | --- |
| **Cardholder’s Name** | **Card Number** | **Expiration Date** | **Expected Output** |
| John Doe | 378282246310005 | 01/2020 | Expired |

**Test Case #5 (Test getCardsByName() method)**

Expected result of Cards.getCardsByName(“johndoe”)

|  |  |  |  |
| --- | --- | --- | --- |
| **Cardholder’s Name** | **Card Number** | **Expiration Date** | **Expected Output** |
| John Doe | 378282246310005 | 01/2020 | Expired |

**Test Case #6 (Test getCardsByNumber() method)**

Expected result of Cards.getCardByNumber(“6011000990139424”)

|  |  |  |  |
| --- | --- | --- | --- |
| **Cardholder’s Name** | **Card Number** | **Expiration Date** | **Expected Output** |
| Dmittrii Sasha | 6011000990139424 | 05/2022 | Valid |

**Test Case #7 (Test getAllCards() method)**

Expected results of Cards.getAllCards():

|  |  |  |  |
| --- | --- | --- | --- |
| **Cardholder’s Name** | **Card Number** | **Expiration Date** | **Expected Output** |
| John Doe | 378282246310005 | 01/2020 | Expired |
| John Cena | 371449635398431 | 02/2020 | Expired |
| Katherine Booher | 378734493671000 | 04/2019 | Expired |
| Justin Timberlake | 9610591081018250 | 12/2021 | Invalid |
| Krikor Faddey | 6011111111111117 | 04/2023 | Valid |
| Dmittrii Sasha | 6011000990139424 | 05/2022 | Valid |
| Leonty Natali | 5425233430109903 | 04/2023 | Valid |
| Polina Lazar | 4917484589897107 | 05/2025 | Valid |

**Test Case #8 (Test sortCardsByNumber() method)**

Input: Cards.sortCardsByNumber()

Expected results:

|  |  |  |  |
| --- | --- | --- | --- |
| **Cardholder’s Name** | **Card Number** | **Expiration Date** | **Expected Output** |
| John Cena | 371449635398431 | 02/2020 | Expired |
| John Doe | 378282246310005 | 01/2020 | Expired |
| Katherine Booher | 378734493671000 | 04/2019 | Expired |
| Polina Lazar | 4917484589897107 | 05/2025 | Valid |
| Leonty Natali | 5425233430109903 | 04/2023 | Valid |
| Dmittrii Sasha | 6011000990139424 | 05/2022 | Valid |
| Krikor Faddey | 6011111111111117 | 04/2023 | Valid |
| Justin Timberlake | 9610591081018250 | 12/2021 | Invalid |

**Test case #9 (Test sortCardsByName() method)**

Input: Cards.sortCardsByName()

Expected Results:

|  |  |  |  |
| --- | --- | --- | --- |
| **Cardholder’s Name** | **Card Number** | **Expiration Date** | **Expected Output** |
| Dmittrii Sasha | 6011000990139424 | 05/2022 | Valid |
| John Cena | 371449635398431 | 02/2020 | Expired |
| John Doe | 378282246310005 | 01/2020 | Expired |
| Justin Timberlake | 9610591081018250 | 12/2021 | Invalid |
| Katherine Booher | 378734493671000 | 04/2019 | Expired |
| Krikor Faddey | 6011111111111117 | 04/2023 | Valid |
| Leonty Natali | 5425233430109903 | 04/2023 | Valid |
| Polina Lazar | 4917484589897107 | 05/2025 | Valid |

**Test Case #10**

Attempt to add a credit card via the addCreditCard() method but input null values.

Expected results: No error should be thrown, just don’t add the card

|  |  |  |  |
| --- | --- | --- | --- |
| **Cardholder’s Name** | **Card Number** | **Expiration Date** | **Expected Output** |
| John Doe | 378282246310005 | 01/2020 | Expired |
| John Cena | 371449635398431 | 02/2020 | Expired |
| Katherine Booher | 378734493671000 | 04/2019 | Expired |
| Justin Timberlake | 9610591081018250 | 12/2021 | Invalid |
| Krikor Faddey | 6011111111111117 | 04/2023 | Valid |
| Dmittrii Sasha | 6011000990139424 | 05/2022 | Valid |
| Leonty Natali | 5425233430109903 | 04/2023 | Valid |
| Polina Lazar | 4917484589897107 | 05/2025 | Valid |

**Test Case #11**

Attempt to remove a credit card that isn’t in the list using removeCard() method

Expected results: No error should be thrown, just don’t remove anything and display a prompt

**Test Case #12**

Attempt to retrieve a card from an index that isn’t in the list

Expected results: No error should be thrown, just don’t retrieve a card and display a prompt saying that no card was able to be accessed

**Test Case #13**

Attempt to retrieve a card by name that isn’t in the list

Expected results: No error should be thrown, just don’t retrieve a card and display a prompt saying that no card was able to be accessed

**Test Case #14**

Attempt to retrieve a card by number that isn’t in the list

Expected results: No error should be thrown, just don’t retrieve a card and display a prompt saying that no card was able to be accessed

**Test Case #15**

When no cards are in the list and the user tries to getAllCards(), sortCardsByNumber(), or sortCardsByName(), No error should be thrown, just don’t retrieve a card and display a prompt saying that no card was able to be accessed

**Test Case #16**

When the selected file path is invalid, prompt the user of the mishap and then return to the main screen

**Test Case #17**

Importing from the following text file (representing correct data) should result in the following CreditCardList

Text File:

**A screenshot of a cell phone

Description automatically generated**

CreditCardList:

|  |  |  |  |
| --- | --- | --- | --- |
| **Cardholder’s Name** | **Card Number** | **Expiration Date** | **Expected Output** |
| John Doe | 378282246310005 | 01/2020 | Expired |
| John Cena | 371449635398431 | 02/2020 | Expired |
| Katherine Booher | 378734493671000 | 04/2019 | Expired |
| Justin Timberlake | 9610591081018250 | 12/2021 | Invalid |
| Krikor Faddey | 6011111111111117 | 04/2023 | Valid |
| Dmittrii Sasha | 6011000990139424 | 05/2022 | Valid |
| Leonty Natali | 5425233430109903 | 04/2023 | Valid |
| Polina Lazar | 4917484589897107 | 05/2025 | Valid |

**Test Case #18**

Importing from the following text file (representing invalid delimiters) should result in an empty CreditCardList

A screenshot of a cell phone

Description automatically generated

**Test Case #19**

Importing from the following text file (representing the lack of delimiters) should result in an empty CreditCardList

A screenshot of a cell phone

Description automatically generated

**Test Case** **#20**

Importing the following text file (represents a missing or invalid value in the card holder name portion of a row) should result in the following CreditCardList

A screenshot of a cell phone

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| **Cardholder’s Name** | **Card Number** | **Expiration Date** | **Expected Output** |
| John Cena | 371449635398431 | 02/2020 | Expired |
| Katherine Booher | 378734493671000 | 04/2019 | Expired |
| Justin Timberlake | 9610591081018250 | 12/2021 | Invalid |
| Krikor Faddey | 6011111111111117 | 04/2023 | Valid |
| Dmittrii Sasha | 6011000990139424 | 05/2022 | Valid |
| Leonty Natali | 5425233430109903 | 04/2023 | Valid |
| Polina Lazar | 4917484589897107 | 05/2025 | Valid |

**Test Case #21**

Importing the following text file (represents a missing value in the card number section of a row) should result in the following CreditCardList

A screenshot of a cell phone

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| **Cardholder’s Name** | **Card Number** | **Expiration Date** | **Expected Output** |
| John Cena | 371449635398431 | 02/2020 | Expired |
| Katherine Booher | 378734493671000 | 04/2019 | Expired |
| Justin Timberlake | 9610591081018250 | 12/2021 | Invalid |
| Krikor Faddey | 6011111111111117 | 04/2023 | Valid |
| Dmittrii Sasha | 6011000990139424 | 05/2022 | Valid |
| Leonty Natali | 5425233430109903 | 04/2023 | Valid |
| Polina Lazar | 4917484589897107 | 05/2025 | Valid |

**Test Case #22**

Importing the following text file (represents a missing or invalid value in the expiration data portion of a row) should result in the following CreditCardList

A screenshot of a cell phone

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| **Cardholder’s Name** | **Card Number** | **Expiration Date** | **Expected Output** |
| John Cena | 371449635398431 | 02/2020 | Expired |
| Katherine Booher | 378734493671000 | 04/2019 | Expired |
| Justin Timberlake | 9610591081018250 | 12/2021 | Invalid |
| Krikor Faddey | 6011111111111117 | 04/2023 | Valid |
| Dmittrii Sasha | 6011000990139424 | 05/2022 | Valid |
| Leonty Natali | 5425233430109903 | 04/2023 | Valid |
| Polina Lazar | 4917484589897107 | 05/2025 | Valid |

**Test Case #23**

Importing an empty text files should result in an empty CreditCardList

**Test Case #24**

Importing the following text file (which represents an invalid entry in the card number position) should return the following CreditCardList

A screenshot of a cell phone

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| **Cardholder’s Name** | **Card Number** | **Expiration Date** | **Expected Output** |
| John Cena | 371449635398431 | 02/2020 | Expired |
| Katherine Booher | 378734493671000 | 04/2019 | Expired |
| Justin Timberlake | 9610591081018250 | 12/2021 | Invalid |
| Krikor Faddey | 6011111111111117 | 04/2023 | Valid |
| Dmittrii Sasha | 6011000990139424 | 05/2022 | Valid |
| Leonty Natali | 5425233430109903 | 04/2023 | Valid |
| Polina Lazar | 4917484589897107 | 05/2025 | Valid |

**Test Case #25**

Importing the following text file (which represents and invalid expiration date) should return the following CreditCardList

A screenshot of a cell phone

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| **Cardholder’s Name** | **Card Number** | **Expiration Date** | **Expected Output** |
| John Cena | 371449635398431 | 02/2020 | Expired |
| Katherine Booher | 378734493671000 | 04/2019 | Expired |
| Justin Timberlake | 9610591081018250 | 12/2021 | Invalid |
| Krikor Faddey | 6011111111111117 | 04/2023 | Valid |
| Dmittrii Sasha | 6011000990139424 | 05/2022 | Valid |
| Leonty Natali | 5425233430109903 | 04/2023 | Valid |
| Polina Lazar | 4917484589897107 | 05/2025 | Valid |

### Changes to Previous Files

**Class**: CreditCard.java

**Method**: getIsCardExpired()

**Description**: this method is a getter that returns the boolean isCardExpired to the runtime environment

**Class**: CreditCard.java

**Method**: compareNumber()

**Description**: return the integer value when using the compareTo() method between this CreditCard object's cardHolderName to the argumented CreditCard object or String

**Class**: CreditCard.java

**Method**: compareName()

**Description**: return the integer value of the compareTo() method when comparing card numbers to an argumented String or another CreditCard object

* Removed the public setters and replace them with an arg constructor to prevent data manipulation after card initialization

**Class:**  CreditCard.java

**Method**: toString()

**Description:** return a String attribute containing the card holder name, card number, and expiration will a pipe delimiter. This variable will be used to export the card to a text file.