**CS 1101 – Introduction to Computer Science**

**Spring 2023**

**Comprehensive Lab 3**

**Pokémon Storage System**

**Due Date Details:**

**Check-In:** Check-in must be done by **April 17th, 2023**.

**Java Program:** Java program is due by **May 1st, 2023.**

**Demo the Code:** You will receive a link to demo to a designated Instructional Team member. You may only demo once you’ve turned in the program.

**Objective:** Students will be able to use their problem-solving skills and implement a solution to the instructed problem using variable declaration, conditional statements, loops, arrays, methods, recursion, linked lists, and user input.

**Background:**

To complete this assignment, you need to have knowledge in the following concepts:

* File reading: Reading the given file with all the Pokémon data.
* Arrays: Two-dimensional array for storing your Pokémon. Traverse arrays, read values, store values.
* Methods: return value (void, boolean, int, int []), pass by reference and pass by value
* Loops: for AND while loops (do..while loops optional)
* Ifs or switch: logical and relational operators (e.g., check if the size of the board is even)
* Linked lists: Used to keep the Pokémon currently in your party and traverse the list in order.
* Recursion: traverse the linked list to display its contents or check if a Pokémon is currently in the party or not.

**Description:**

In Pokémon games, you can keep up to 6 Pokémon with you in your party. Any other Pokémon that you capture, are stored in your storage system. In this system, you can see all your Pokémon, deposit from your party, add to your party, and check statistics. This is what we will be doing for this lab as well.

**Important:** Before you get started, read this entire document. If anything is unclear, do not hesitate to come to your instructor, TA, or IAs for help or clarification. We will be happy to help you.

**Your program should have the following features:**

* + 1. View box.
       - 1. This is where all your Pokémon will be stored, it should display every Pokémon who is not in the party.
         2. Additionally, the users should have the option to filter out the results as there may be too many to see all at once. Users should be able to filter at least by level (given by a range set by the user), or generation.

\*Bonus points for implementing a filter by type. Pokémon can have one or two types, which can make this a bit tricky.

* + 1. View current party.
       - 1. Viewing the party the user currently has in order.

\*Users must always have at least one Pokémon in their party and cannot have more than 6 Pokémon in their party at once.

* + 1. Deposit Pokémon (remove from party).
       - 1. Users should be able to deposit any Pokémon from their party to move back to the box, regardless of position, unless the party has only one Pokémon, in that case, the user should not be able to deposit.
         2. This action should make the deposited Pokémon viewable again the box in their respective position.
    2. Withdraw Pokémon (add to party)
       1. Users should be able to add any Pokémon that is currently in the box to the party, unless the party already contains 6 Pokémon, in that case, the user should not be able to withdraw.
       2. This action should remove the withdrawn Pokémon from the box and display it on the party in the correct order.
    3. View statistics
       1. Users should be able to select any Pokémon (either party or box) and display all its statistics.

**Assumptions/Clarifications**

* When the program first runs, there should already be one Pokémon in the party. There cannot be an empty party at any time.
* You must use the given starter code, as well as the LL.java, LLNode.java (DO NOT MODIFY THIS ONE AT ALL), and pokemonList.txt files, the music file is optional.
* You are not allowed to use any external libraries other than the ones provided in the starter code.
* You cannot modify any of the method’s signature.

**Method implementation**

You should implement and use the following methods.

|  |  |
| --- | --- |
| Name | displayPokemon |
| Parameters needed | String[][] pokemonBox = This 2D array contains your box with all your stored Pokémon. Each individual String array contains all the information of a single Pokémon. The information of these individual arrays will be displayed to the user based on the filter criteria.  String filter = This represents the filter users want to apply to display only certain Pokémon. If the given filter does not match any of the implemented filtering options, it will display all Pokémon from the box. |
| Return value | int. This method will return a 3 that represents that all the pairs of numbers have been found, 0 otherwise.  void – This method will only print out all the Pokémon that match the filtering criteria, so nothing needs to be returned. |
| Description | This is the method used to display all your Pokémon currently in the box. This method will receive the variable where all the Pokémon are stored in the box, and it will also require the user to enter a String that represents the filtering criteria. If the input filter criteria do not match any of the implemented filtering options, the method should display all the Pokémon in the box. |

|  |  |
| --- | --- |
| Name | getFromParty |
| Parameters needed | LLNode headNode = A reference to the head of the party  String getPokemon = A string with the name of the Pokémon you want to get from the party |
| Return value | String[] – The array of String of the desired Pokémon from the input  If the chosen Pokémon is not in the party, the method should return an array of empty Strings. |
| Description | This method will return the Pokémon that matches the name of getPokémon, or an array of empty Strings if the Pokémon is not in the linked list representing the party.  This method DOES NOT remove the Pokémon from the linked list.  \*This method must be done recursively. |
| Example | Current party: Pikachu <- Bulbasaur <- Charmeleon  Input: partyHead (aka, a reference to the node that contains Pikachu, aka a reference to the head of the linked ist), “Bulbasaur” Output: String[] containing all of Bulbasaur’s information Explanation: Since Bulbasaur was in the party, its information is obtainable in the linked list.  Current party: Pikachu <- Bulbasaur <- Charmeleon  Input: partyHead, “Ditto” Output: String[] containing empty Strings Explanation: Since the party does not contain Ditto |

|  |  |
| --- | --- |
| Name | addToParty |
| Parameters needed | LLNode pkmn = A linked list containing the data of the Pokémon we want to add to the party. |
| Return value | Void |
| Description | This method will add the given Pokémon to the end of the linked list and increase the size counter of the linked list.  If the party contains Pokémon, it means it is full, so no more Pokémon can be added unless one is removed first. |
| Example | Current party: Pikachu <- Bulbasaur <- Charmeleon  Input: “Blastoise” Output:  Current party: Pikachu <- Bulbasaur <- Charmeleon <- Blastoise  Explanation: Blastoise was added to the party  Current party: Pikachu <- Bulbasaur <- Charmeleon <- Blastoise <- Tauros  Input: “Blastoise” Output:  Blastoise is already in the party!  Current party: Pikachu <- Bulbasaur <- Charmeleon <- Blastoise <- Tauros <- Ditto  Input: “Mew” Output:  The party already contains 6 Pokémon, cannot add another Pokémon unless one is removed from the party first. |

|  |  |
| --- | --- |
| Name | removeFromParty |
| Parameters needed | String pkmnName |
| Return value | String [] – this will contain the removed Pokémon’s data.  It will return an array with an empty String if the Pokémon is not found |
| Description | This method will receive the name of a Pokémon to remove from the party. If the Pokémon is found, it will remove it from the linked list that represents the party. If it is not found, it will only return an array containing an empty String |
| Example | Current party: Pikachu <- Bulbasaur <- Charmeleon  Input: “Pikachu” Output:  Current party: Bulbasaur <- Charmeleon  Current party: Pikachu <- Bulbasaur <- Charmeleon <- Blastoise <- Tauros  Input: “Charmeleon” Output:  Current party: Pikachu <- Bulbasaur <- Blastoise <- Tauros  Current party: Pikachu <- Bulbasaur <- Charmeleon <- Blastoise <- Tauros <- Ditto  Input: “Mew” Output:  Mew is not in the party! (Party unmodified)  Current party: Tauros  Input: “Tauros” Output:  Cannot remove if there is only one Pokémon in the party! (Party unmodified) |

|  |  |
| --- | --- |
| Name | displayParty |
| Parameters needed | LLNode headNode = A reference to the head of the party |
| Return value | Void |
| Description | This method will traverse the linked list RECURSIVELY and displaying the Pokémon in order from first to last, displaying their name and their level. |
| Example | Current party (from first to last) = Charmander, Caterpie, Hitmonlee, Golem  Input: reference to the head of the party  Output:  Your current party is:  Charmander (lvl. 5) <- Caterpie (lvl. 3) <- Hitmonlee (lvl. 84) <- Golem (lvl. 80)  Explanation: The method traverses the list and displays the current node’s name and level before moving to the next node. |

|  |  |
| --- | --- |
| Name | inParty |
| Parameters needed | String pokemonName – The name of the Pokémon to be removed |
| Return value | boolean – true if the Pokémon is in the party, false otherwise |
| Description | This method will traverse the linked list and look for the Pokémon that matches the given name. |
| Example | Current party: Pikachu <- Bulbasaur <- Charmeleon <- Blastoise <- Tauros <- Ditto  Input: “Mew” Output: False  Current party: Pikachu <- Bulbasaur <- Charmeleon <- Blastoise <- Tauros <- Ditto  Input: “Tauros” Output: True |

\*\*\* Feel free to create other methods to break down the assignment into smaller sections if you want. Such as a method to display the main menu, a method to read the .txt file and store everything into a 2D array, etc. No extra points will be given for this, but if it helps you, you can do it.

**Sample output:**

Main Menu

Text

Description automatically generated

1. View Box (no filter)

Text

Description automatically generated

Text

Description automatically generated

\*\*\*Pokemon currently in the party are not displayed in View Box

Filter by Type (optional for extra credit)

Text

Description automatically generated

Filter by level range

Text

Description automatically generated

Filter by generation

Text

Description automatically generated

\*\*\* You can handle invalid inputs for the filters however you want, in my case, I made it not display anything if I input a generation that does not match any Pokémon

Text

Description automatically generated

1. View party

Text

Description automatically generated

Another example is where Pikachu is at the front of the party and Sneasel at the back.

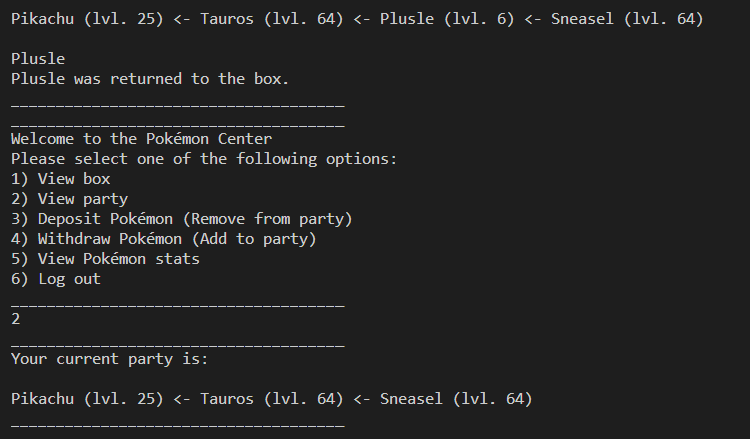
Text

Description automatically generated

1. Deposit Pokémon (remove from party)

Text

Description automatically generated



\*\*\* You cannot make a deposit if you only have one Pokémon in your party

Text

Description automatically generated

1. Withdraw (Add to party)

Text

Description automatically generated

Cannot have more than 6 Pokémon in your party

Text

Description automatically generated

Cannot add someone who is already in your party

Text

Description automatically generated

1. View Pokémon stats

Text

Description automatically generated with medium confidence

1. Log out

Text

Description automatically generated

**Tips:**

* **Be sure to watch out for syntax errors, especially where it concerns opening and closing parentheses and braces.**
* **Comment where your loops start and end so you can more easily keep track of them.**
* **Careful with your use of scanners:**
  + **Know what your scanner’s purpose is.**
  + **Be on the lookout if cursor does not go to next line when reading user input.**
* **Be aware of when to use “==” vs. “.equals()”**
* **Trace the execution of your linked list methods to know they are doing what they are supposed to.**
* **It’s easier to handle invalid input if you keep all your user input as the same type and then convert to whatever you need.** 
  + **Remember you can use Integer.parseInt(string) to convert a string into an int.**

**Deliverables:** You are expected to submit one file in Blackboard:

1. CL3\_YourLastName.java --- the java file of your program.
2. CL3Report\_YourLastName.pdf – your lab report

**Grading Criteria:**

|  |  |
| --- | --- |
| **Checklist** | **Score** |
| **The program has proper documentation. “Certificate of Acknowledge” comment is included.**  /\* Your Full Name  [CS1101] Comprehensive Lab 2  This work is to be done individually. It is not permitted to.  share, reproduce, or alter any part of this assignment for any  purpose. Students are not permitted from sharing code, uploading  this assignment online in any form, or viewing/receiving/  modifying code written from anyone else. This assignment is part.  of an academic course at The University of Texas at El Paso and  a grade will be assigned for the work produced individually by  the student./\* | 5 pts |
| **Check-In**   * 1. [**5 pts**] Students **must** have an idea of a solution to CL3.   2. [**5 pts**] Students **must** have, at minimum, a working menu and the file read and stored in a 2D String array. The user must be able to view all Pokémon with no filters. | 10 pts |
| **Comments explaining the different sections of your code** | 10 pts |
| **Implementation**   * 1. [**10 pts**] The program uses conditional, loops, arrays, and methods.   2. [**10 pts**] The program implements the provided methods.   3. [**10 pts**] The methods getFromParty and displayParty are implemented recursively.   4. [**10 pts**] The program handles all of the specified situations like not being able to add more than 6 Pokémon to the party or not being able to withdraw a Pokémon that is already in the party.   5. [**5 pts**] The file containing all of the Pokémon is stored appropriately within a 2D string array and used appropriately when modifying it or displaying its contents. | 45 pts |
| **Variable Names and Types**   * 1. [**5 pts**] Correct naming conventions (class name is camel case but first letter is   capitalized). Variables are camel case, but the first letter is in lowercase.   * 1. [**5 pts**] Variables should have meaningful variable names | 10 pts |
| **Program output follows the correct format and style as the samples.** | 5 pts |
| **The deliverables follow proper name and format** | 5 pts |
| **The program is indented properly** | 10 pts |
| **Total** | 100 pts |
| **Bonus**   1. Implement a filter by type - User must be able to input one or two types, if only input type 1 is valid and matches either type of the Pokémon, display it. If both input types are valid, both must match (order does not matter) to display the Pokémon) - 10 pts 2. When displaying the statistics, the stat name and the value are aligned, regardless of the Pokémon as in the example above. – 5 pts 3. The program handles invalid inputs anytime the program asks for a Pokémon name. – 5 pts |  |

Late submission: [**-10**] points for every 24 hours after the deadline.

No report: [-10] points

\*\*\*\*Your code must compile and run to have a demo

\*\*\*\* You must pass your demo to get a grade. If you fail your demo, you must reschedule.

If you need any clarification, please ask your TA/IA for further details.