Daniel Coy

IT-145 Final Project

April 23, 2017

**Zoo Monitoring System**

**Scenario:** A zoo requires a monitoring systems that allows a user to select between animals and habitats, and specify one of several options within each category. It will then need to read the status of the user’s selection from a file and display the appropriate information on screen. If a status is detected out of range, a dialog box will be used to alert the user. Users will need to be able to return to the original menu to make additional selections.

**Process:** The monitoring system will utilize a loop to continuously seek input from the user for menu selections, until the user inputs the specific command to end the program. It will read input from multiple files, based on the user’s selections, and will display the appropriate data. Sub-menus and user input will be handled by two methods, one for the animal monitoring and one for the habitat monitoring. Each method will use its own loop to continuously prompt the user for menu selections and to display the selected data, until the ‘return to main’ selection is made and the method ends. A class will be created to read input from each file. The class will create a list of the options contained in the file, based on the initial details lines in the files. The class will then create a 2-dimensional array to store the data from the file, which will then be accessed and displayed by a public method based on the user’s input selection. Another class will be used to create a dialog box object for any alert detected in the files, that will be displayed when the alert string is detected while displaying the user’s selected data.

**Pseudocode:**

MAIN Method

DECLARE String userInput

WHILE userInput does not equal EXIT

DISPLAY “Monitor ANIMAL, HABITAT, or EXIT?”

GET userInput

IF userInput equals ANIMAL

CALL Animal Method

ELSE IF userInput equals HABITAT

CALL Habitat Method

ELSE IF userInput equals EXIT

DISPLAY “Exiting Monitoring System”

ELSE userInput is not valid

DISPLAY “Invalid Input”

END WHILE

END MAIN Method

ANIMAL Method

DECLARE INTEGER numAnimals set to 0

DECLARE INTEGER userInput set to 0

OPEN FILE animals.txt

DISPLAY “Enter corresponding number to monitor the following animals or to return to main:”

READ nextLine from file

WHILE nextLine contains “Details”

ITERATE numAnimals

DISPLAY numAnimals + sub-string of nextLine starting at index 11 + newline

READ nextLine from file

ENDWHILE

DISPLAY (numAnimals + 1) + “Return to main menu”

//Create 2d array with the dimensions of the number of animals by 5, which is the number //of lines per each animal status

INITIALIZE String Array animalStatus[numAnimals][5]

//While the file continues to have a next string to read, step through the file 5 lines at a //time and store those lines in the animalStatus string array.

INITIALIZE INTEGER arrayCounter to 0

WHILE animals.txt hasNextString

FOR each of the next 5 lines

STORE nextLine in animalStatus[arrayCounter][i]

GET nextLine

ENDFOR

ITERATE arrayCounter

ENDWHILE

CLOSE FILE animals.txt

GET userInput

//The user’s selection minus 1 will give the row index of the selection in the array

INITIALIZE INTEGER userSelectionIndex set to userInput - 1

WHILE userInput does not equal (numAnimals + 1)

IF userInput is greater than or equal to 1 AND less than or equal to (numAnimal + 1) then input is valid

//Step through each of the 5 columns and display contents, create dialog //box if alert is found

FOR each column in animalStatus[][] in row userSelectionIndex

DISPLAY animalStatus[userSelectionIndex][i] + newline

IF animalStatus[userSelectionIndex][i] contains “\*\*\*\*\*”

CREATE dialog box object

CALL dialog box object display method with current column as parameter

ENDIF

ENDFOR

ELSE input is invalid

DISPLAY “Invalid input”

GET userInput

END WHILE

DISPLAY “Returning to main menu.”

END ANIMAL Method

HABITATS Method

DECLARE INTEGER numHabitats set to 0

DECLARE userInput set to 0

OPEN FILE habitats.txt

DISPLAY “Enter corresponding number to monitor the following habitats or to return to main:”

READ nextLine from file

WHILE nextLine contains “Details”

ITERATE numHabitats

DISPLAY numHabitats + sub-string of nextLine starting at index 11 + newline

READ nextLine from file

ENDWHILE

DISPLAY (numHabitats + 1) + “Return to main menu”

//Create 2d array with the dimensions of the number of habitats by 4, which is the number //of lines per each habitat status

INITIALIZE String Array habitatStatus[numHabitats][4]

//While the file continues to have a next string to read, step through the file 4 lines at a //time and store those lines in the habitatStatus string array.

INITIALIZE INTEGER arrayCounter to 0

WHILE habitats.txt hasNextString

FOR each of the next 4 lines

STORE nextLine in habitatStatus[arrayCounter][i]

GET nextLine

ENDFOR

ITERATE arrayCounter

ENDWHILE

CLOSE FILE habitats.txt

GET userInput

//The user’s selection minus 1 will give the row index of the selection in the array

INITIALIZE INTEGER userSelectionIndex set to userInput - 1

WHILE userInput does not equal (numHabitats + 1)

IF userInput is greater than or equal to 1 AND less than or equal to (numHabitats + 1) then input is valid

//Step through each of the 4 columns and display contents, create dialog //box if alert is found

FOR each column in habitatStatus[][] in row userSelectionIndex

DISPLAY habitatStatus[userSelectionIndex][i] + newline

IF habitatStatus[userSelectionIndex][i] contains “\*\*\*\*\*”

CREATE dialog box object

CALL dialog box object display method with current column as parameter

ENDIF

ENDFOR

ELSE input is invalid

DISPLAY “Invalid input”

GET userInput

END WHILE

DISPLAY “Returning to main menu.”

END HABITATS METHOD

PUBLIC CLASS DialogAlert

DECLARE PRIVATE String alertString

PUBLIC METHOD displayAlert with String inputString as parameter

SET alertString to inputString

REPLACE ‘\*’ chars with ‘’ in alertString

DISPLAY dialog box using JOptionPane class, with alertString as parameter

END CLASS DialogAlert

**Pseudocode Change Log:**

The following changes have been made to the final code, and therefore differ from the pseudocode:

* **Changes to the DialogAlert Class:**
  + Added additional private field, String headerString. This will hold the string that will be displayed in the header of the dialog box.
  + Created default constructor that initializes the private fields.
  + Created overloaded constructor that takes two Strings as parameters. These are then assigned to the private fields to be used to in the creating of the dialog box.
  + Created a setAlertMsg public method that allows the alertString field to be set.
  + Created a setHeaderMsg public method that allows the headerString field to be set.
  + The displayAlert method no longer takes a parameter to set the alert message, as that is now set using the overloaded constructor or the set methods.
* **Changes to Animal and Habitat Monitoring methods:**
  + The file reading and data displaying functionalities were migrated to a new FileReader class, so were removed from the AnimalMonitoring and HabitatMonitoring methods.
  + These methods are now solely focused on displaying the menu, getting the user’s input, and using the FileReader object to display the selection.
  + The methods create a FileReader object, passing the specific file name and data set size as parameters.
  + The FileReader object is then used to display the menu options and user input is retrieved.
  + While the user has not input “RETURN” the methods continually loop through asking for user input and using the FileReader object to display what the user has selected.
  + These changes greatly reduce the complexity of the methods, and allow for quickly adding new methods if additional monitoring options are desired.
* **Created FileReader Class:**
  + Created the FileReader Class to handle all reading from files and displaying of user selections.
  + The class takes the file name, and the number of lines per data set in that particular file. For example, the animals.txt file’s data sets are 5 lines each. This is used to create an array to store the file data, and to step through each line when displaying the user selection.
  + Upon creating a FileReader object, the constructor reads each line with “Details” to create a list of menu options for the user to select from.
  + A private helper method is used to get the number of data sets in the file (e.g. the number of animals entered in to animals.txt), by stepping through each line of the file and counting the number of headers that match the file name. For example, if the file name is habitats.txt, the method counts the number of “habitat – “ headers in the file to determine how many data sets there are.
  + The number of data sets and the length of those data sets is used to create a 2D String array, and nested loops are used to step through the file and store each data set in to a row, with each column being a line of data.
  + A dsiplaySelection public method is used to take the userInput parameter and step through the String array, checking the input against the headers. If they match, a FOR loop is used to step through and display the lines of the data set, creating a dialog box if the alert string (“\*\*\*\*\*”) is detected.
  + An additional public method, getMenuOptions, is used to return the menuOptions string that was created in the constructor.
  + This class will provide much more flexibility in the program by greatly reducing the size and complexity of the monitoring methods. Therefore, if additional monitoring options are added and additional files created, it will be much easier to create a new method for those options. For example, if a Staff Monitoring option was desired and a file containing the status of zoo personnel was created, a new method could be easily added to the program, only needing to copy a current one and make minor changes.

**Error Log:**

* **Animal Method** 4/17/2017 6:00pm: When stepping through the file to read each line using a WHILE (fileScanner.hasNextLine()) loop, program would have a runtime error when the last loop was executed. It was determined that the final fileScanner.nextLine() call was causing the error, because it was trying to read a line that was not there. The solution was to create an IF statement that checked to see if there even was a next line to read, before actually reading it.
* **Animal Method** 4/17/17 7:30pm: Program would not correctly enter the IF statement meant to compare the user’s input to the line in the file with the “Animal – “ header. This is what would trigger the displaying of the animal’s information. The issue was due to directly comparing the two values, even though the user is prompted to input a plural, whereas the “Animal” line is written in the singular (e.g. user enters “lions” but the file lists “Animal – Lion”). This was resolved by checking if the userInput CONTAINS the string from the “Animal” line, after each is converted to lowercase. For example, user inputs “LIONS”, which contains the “Lion” string from the “Animal” line, after each has been converted to lowercase. If user inputs “Lio”, that does not contain “Lion” so does not enter the IF statement. Users inputting additional characters, like “lionsdfg” is handled by a different mechanism and does not enter the IF statement. This portion of the Animal method was later removed when the FileReader class was created. However, the lesson learned from this error was implemented in to that class when comparing the user’s input to the headers in the data sets.
* **FileReader Class:** 4/19/17 7:35pm: While creating the getNumDataSets() method within the class, which returns the number of animals or habitats in the given file, the method was not returning the correct number for the habitats.txt file. Initially, the method just checked each line for whether it contained “animal” or “habitat.” After looking closely at the files, and going step by step through the code with the debugger, it was discovered that one of the lines in the habitats.txt file contains the word “habitat” that is not the start of a new data set. This line was also being counted, when it should not have been. Changing the comparison to be more precise, by also checking for the “ –“ that is in the data set headers, fixed the issue. Being as precise as possible can help prevent unintended side effects and errors while executing a program.
* **FileReader Class:** 4/19/17 8:30pm: While creating the overloaded constructor, and implementing the 2D array which holds the file data, the program kept receiving a runtime error due trying to write to a location outside of the array’s established dimensions. This was caused by iterating the “rows” counter at the end of the outer loop, instead of only when the inner loop executes and steps through all the columns. Ensuring that nested loops iterate their counters at the correct steps is crucial to making them function properly.
* **FilerReader Class:** 4/19/17 9:30pm. A bug similar to the one encountered in the original Animal method, and discussed above, occurred when comparing the userInput to the data set headers to determine if the data set should be displayed (e.g. if user in puts “lions” it needs to be compared to the “Animal – Lions” line from the file to determine if the rest of the lion info should be displayed). A similar resolution was used, checking if the input contained the substring from the line, instead of the other way around.

**Conclusion**

The Zoo Monitoring System functions as intended, allowing the user to make a selection for what they would like to monitor, then using the files to create a list of options, and displaying the appropriate information when given input. The implementation of the FileReader class greatly enhanced the functionality of the program, and allows for the easy addition of more options in the future. The modularity provided by separate classes and methods makes the program and its code much easier to understand, and will allow for easier maintenance and enhancements over time.