IT – 145 Foundations of Application Development

Final Project

Process Documentation

R. Hutton

21 April 2019

**Problem Statement / Scenario**

The activities within the zoo need to be monitored daily. Such a system requires identifying the animal type, status, health, and habitat of each animal. The purpose of this is to efficiently manage the zoo systems to react accordingly to various issues that occur within a zoo. Therefore, as an IT person assigned to the Midwest Regional Zoo, I plan to develop a program to allow the user to check the status of each animal, habitat or exit the program.

The program requires the use of a minimum of two classes, and two methods within the main class to support the execution of the main method. Also, the program must successfully read two files, *animals.txt* and *habitat.txt*, and extract data from each file to report that specific information based on the user selection. If a line in the file reads a concerning message, the program will take that line from the file and pass it to a dialog box to alert the user.

**Overall Process**

After receiving eye-opening feedback on my pseudocode (explained in the next section), I had to reevaluate my development and understanding of the project. Throughout Week 7, I reviewed the instructor’s lesson video discussing the final project on multiple occasions to really absorb the information. I also sat in on an IT-145 peer tutoring later in Week 7 as a Java refresher and to discuss any topics on this course’s final project.

Once I had a detailed understanding of the project, I began to think about how I envisioned the program to execute. I first started with planning out the execution of the main function - the main function would prompt the user, then call functions to other parts of the program and read the file. The function called in main depended on the user selection, thus a branch statement (switch statement) needing to be created. Within each function, I needed to open and read the file to run the remainder of the program.

To save time in developing the project, I followed the plan I laid out and started to code the main function. In the switch statement, I identified, by name only, the functions I wanted to call in the main function before coding them. I commented them out once I was complete with the main function. Next, I developed one of the functions called in main (launchAnimals()), incrementally, to perfect that function with the goal of applying the same logic to the second function (launchHabitat()), which would follow the same pattern. As I was coding each function, I created each class to supplement the function and incorporated the classes in the functions. Once I debugged, tested, and ran that portion of the function and there were no errors, I coded the remainder of the program.

**Pseudocode**

My original pseudocode from Module 5 was the following (displayed in single-space for concise clarity):

PROMPT the user to choose to monitor an “animal”, “habitat” or “exit” program

RECEIVE input selection option from user

READ in file according to the user input in step 2

PROMPT user to specify detailed selection from Step 1

RECEIVE input selection from user

DISPLAY the monitoring system by finding appropriate section in the file

* 1. IF user selection matches item in file,
     1. DISPLAY appropriate data
  2. Otherwise if data contains “\*\*\*\*\*”
     1. PARSE data from asterisks to capture appropriate message
     2. DISPLAY alert message to user showing something is wrong
  3. Otherwise, throw exception due to failure to read file

RETURN to the main menu by accepting a user input

LOOP entire program until user exits the program.

This pseudocode lacks enough detail to accurately run the program per the project requirements, which was the main feedback I received back from the class and instructor. I also realized that the pseudocode above could only work in one method because it did not specify how other methods or classes would be utilized. Therefore, I incorporated the use of methods and classes into the pseudocode to add more detail and describe how the files will be read, the use of loops, branch statements and throwing exceptions:

PROMPT the user to examine (A)nimal, (H)abitat, or (Q)uit

RECEIVE input selection option from user

If user selects ‘A’ or ‘a’, THEN call function launchAnimals()

OTHERWISE IF user selects ‘H’ or ‘h’, THEN call function launchHabitat()

OTHERWISE IF user selects ‘Q’ or ‘q’, THEN exit program.

OTHERWISE, DISPLAY “Invalid input”

IF FUNCTION launchAnimals():

DECLARE ReadAnimal objects, subMenu, and animalSpec, and open “animals.txt” file.

LOOP through the animals.txt file menu and parse data to read “Details on” menu

PROMPT user to select animal to monitor

ACCEPT user input

LOOP through the remainder of the animals.txt file and search for specified animals

DISPLAY animal information

IF any line in the file contains “\*\*\*\*\*”, launch dialog box

CLOSE both animals.txt files

IF FUNCTION launchHabitat():

DECLARE ReadHabitat objects, subMenu, and habitatSpec, and open “habitats.txt” file.

LOOP through the habitats.txt file menu and parse data to read “Details on” menu

PROMPT user to select habitat to monitor

ACCEPT user input

LOOP through the remainder of the habitats.txt file and search for specified habitats

DISPLAY habitat information

IF any line in the file contains “\*\*\*\*\*”, launch dialog box

CLOSE both habitat.txt files

DEFININING ReadAnimal Class:

IMPORT FileInputStream, IOException, and Scanner classes.

DEFINE class constructors to open animals.txt file into private variable, animalFile.

THROW exception to each constructor

PASS animalFile into a new Scanner object.

DEFINE methods in ReadAnimal class:

closeFile(), setInput(Scanner input), getInput(), readLine(), and hasInput()

DEFININING ReadHabitat Class:

IMPORT FileInputStream, IOException, and Scanner classes.

DEFINE class constructors to open animals.txt file into private variable, habitatFile.

THROW exception to each constructor

PASS habitatFile into a new Scanner object.

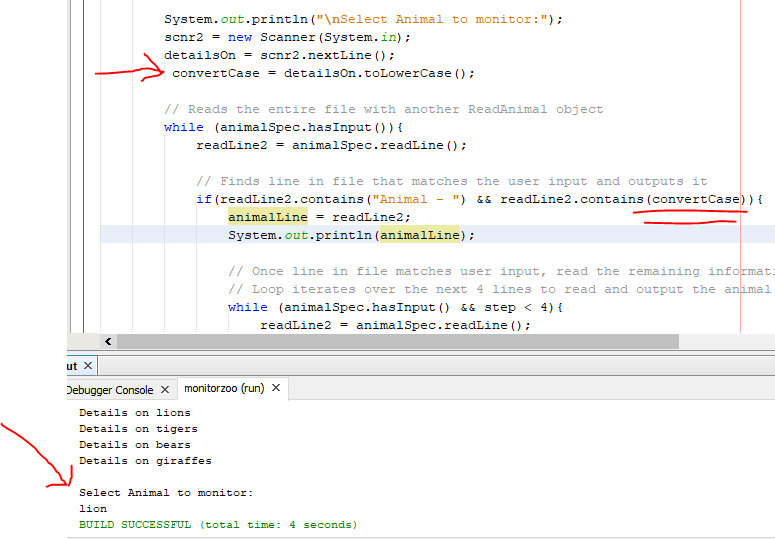
DEFINE methods in ReadHabitat class:

closeFile(), setInput(Scanner input), getInput(), readLine(), and hasInput()

**Methods and Classes**

As mentioned before, I utilized two primary methods within the main class and two additional classes. I designed the program to have the ability to open their respective files during each object declaration of the class. For example, the ReadAnimal class, opens the animals.txt file within the ReadAnimal constructor. This is the same process with the ReadHabitat class. Both classes are declared in launchAnimals() and launchHabitat() methods. Methods from each class that are called in the main methods include: closeFile(), readLine(), and hasInput(). These methods are self-explanatory. The main methods, launchAnimal() and launchHabitat() then reads the opened file via while loops, and extract and display data depending on the user input.

**Error Documentation**



The primary error I received involved a logic error that did not display the output as intended in the project requirements. This occurred in the launchAnimal() method. The intention was to have the line read in the file on the animal detail match with the user’s input regardless of case sensitivity. Thus, I called the String function toLowerCase() and assigned that to convertCase to pass that to readLine.contains(convertCase). I did not attempt this on the launchHabitat() method as to not add more repetitive errors.

**Solution Documentation**

To solve the logic error I received, I traced through the lines of code within the launchAnimals() method with a series of System.out.println(…) to display what data in the file is being passed to the call to readLine. This showed me that the variables were collecting input when read from the file. Lastly, I commented out the lines that declared the convertCase string and changed the line, readLine.contains(convertCase) to readLine.contains(detailsOn). The variable detailsOn is the variable that accepts user input when the user types in the animal they wish to view. This seemed to fix the error and the program ran as intended.

