**Chapter 7**

Ex. 7.1 – The busiest time of day is 18.

Ex 7.2 – Write a declaration for an array variable people that is used to refer to an array of Person objects.

Person[ ] people;

Ex 7.3 –Write a declaration for a variable vacant that could be used to refer to an array of Boolean values.

boolean[ ] vacant;

Ex. 7.4 – The variable hourCounts is used a total of 5 times in the LogAnalyzer class.

Ex. 7.5 – What is wrong with the following array declarations

[] int counts; - int must be positioned in front of the [] as the standard format for an array is data type [] variable.

Boolean [5000] occupied; - 5000 is the issue here. In C++, this would define the size of the array which must be done. But in Java the size of an array is defined when the array is created by a new operator.

Ex. 7.6 – Written assignments for double[] readings, String[]urls, and TicketMachine[] machines:

double[] readings = new Array[60];

String[] urls = new Array[90];

TicketMachine[] machine = new Array[5];

Ex. 7.7 – How many string objects are created by the following declaration?

String[] labels = new String[20];

This creates no string objects. When we create an array we create a fixed sized collection that can hold a defined number of objects.

Ex. 7.8 – What is wrong with the following array creation?

Double[] prices = new double(90);

The issue is that parentheses are used instead of brackets.

Ex. 7.9 – Cehck to see what happens if the for loops’s condition is incorrectly written using the <= operator in printHourCounts.

When the miswritten we get an out of bounds error and this is because we tried to count beyond the array. Since indexes stat at 0 it tries cover 25 objects in a collection that only has 24.

Ex. 7.10 – rewrite the body of printHourlyCounts so that the for loop is replaced by an equivalent while loop.

Done and saved.

Ex. 7.11 – Correct all errors in the printGreater method.

public void printGreater(double [] marks, double mean){

for(index = 0; index < marks.lengtj; index++){

if(marks[index] > mean){

System.out.println(marks[index]);

}

}

}

Ex. 7.12 – Modify the LogAnalyzer class so that it has a constructor that can take the name of the log file to be analyzed. Have this constructor pass the file name to the constructor of LogFileReader class. Use LogdileCreator class to create a file with “random” log entries, and analyze the data.

Ex. 7.13– Complete the numberOfAccesses method to count the total number of accesses recorded in the log file.

Done and saved (in LogAnalyzer class)

Ex. 7.14– See 13

Ex. 7.15– Add a method busiestHour to LogAnalyzer that returns the busiest hour.

Done and saved.

Ex. 7.16 – Add a method slowestHour to LogAnalyzer that returns the busiest hour.

Done and saved.

Ex. 7.17 – Which hour is returned by your busiest hour method if more than one our has the biggest count?

Because of how the if statements condition is written (hourCounts[hour]> busiest hour) the method will return the first number of that value and none of the others. This is because busiest hour doesn’t change in the case of a tie, to be changed a new value must be more than it.

Ex. 7.18 – Add a method to LogAnalyzer that returns finds which two-hour period was busiest.

Done and saved

Ex. 7.19– Add accessors an allow for analysis to be done over seven-day cycle

Will attempt

Ex. 7.20– Will attempt

Ex. 7.21– Read through the LabClass class in the lab-class project

Done

Ex. 7.22– Student objects in the lab-class project could ne stored in a fixed sized array but as it is written they are currently in a dynamic sized collection. The biggest adavantage of a fixed size array is an increase in performance but because of what we do with the collection a dynamically sized one will be better. this is because it allows students to be added and removed.

Ex. 7.23– Rewrite the listAllFiles method in the musicOrganizer class from the music-organizer-v3 by using a for loop rather than a for each loop.

Done and saved

Ex. 7.24– Done

Ex. 7.25– Yes if I repeat the process the same pattern emerges.

Ex. 7.26– there are two versions of the fill method. One version is uased to assign all the elements of an array with the variable val. The second dos the same but over a user defined range of indexes. The reset method uses this method to reset every cell to 0.

Ex. 7.27– Alter the constructor of Automaton so that more than one cell starts as 1. Do different patterns emerge?

Yes, a different pattern emerges if the initial state is different. This pattern depends on the starting state of the automaton.

Ex. 7.28– Rewrite the two if else statements in the loop of the update method of Automaton so that assignments to left and right use conditional operators.

Done I think, original coded commented out and saved just in case.

Ex. 7.29– Use state instead of nextState in update. What are the changes?

When written this way the image updates after every iteration instead of updating after the whole array has been travesed.

Ex. 7.30– Can we avoid these isssues? Find a way and write it. Which version do you prefer?

Can be fixed but I don’t really understand what is going on here. So I prefer the original. I will revisit this time permitting.

Ex. 7.31– re-write the update codeNextState to automaton class that takes three values, left, right, and center, and returns the calculation of the value of the next state.

Ex. 7.33– how many different sets of unique rules do you think there are to calculate a cell’s next 0 or 1 state given the three binary values left, center, and right?

There is a finite number of possibilities because there are only two real outcomes 1 or zero. So my best guess without doing any math is there are hundreds of possibilities.

Upon reading further the book states there is 256 possible combinations but for truly unique sets there are only 88.

Ex. 7.34– Implement the code given in the book

Done and saved

Ex. 7.35– Experiment with different initialization patters of the look-up table in automaton-v4.

Done

Ex. 7.36 – Set up a state table based on an additional integer parameter passed to the constructor of automaton.

Done in automaton-v4 exercises.

Ex. 7.37 – Play with the brain project and get a feel for its methods.

Done

Ex. 7.38 – Modify the brain project to follow the same simpler rules of the Game of Life.

Done and saved

Ex. 7.39 – Modify the environment class in your version of the Game of Lie so that the environment is not toroidal. What differences do you observe in behavior?

Done and saved

The biggest difference in behavior is instead of passing from one side of the screen to the other once hitting the wall it progresses until there aren’t enough alive to keep it moving on.

Ex. 7.40 – 39 but have it bounce/wander once impacting a side

Done and Saved, it runs and does something but I am not certain it is right though

Ex. 7.41 – Rewrite the following code using the static arraycopy method of the System Class.

Int[] copy = new int [original.length];

For(int I =0; I < original.length; i++0){

System.arraycopy(original, 0, copy, 0 , original.length);

}

Ex. 7.42 – Describe the following static method in the java.util.Arrays class: asList, binarySearch, fill, and sort.

The asList method returns a List and uses the array as storage

The binarySearch method is used to search for a specified key in the array using a binary algorithm.

The fill method is used to assign an int value to all elements in an array or a range of indexes.

The sort method is used to sort the elements of an integer array into ascending order.

Ex. 7.43 – Try to write example code that use the previous methods.

Done and Saved

Ex. 7.44 –Write statements to copy all 2D integer array, original into a new array called copy.

Int [] [] copy = new int[original.length][];

for( int x = 0; x < copy.length; x ++){

copy[x] = new int [length];

}

Ex. 7.45 –