**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

Ex. 7.20– To long didn’t attempt

Ex. 7.21–

Ex. 7.22–

Ex. 7.23–

Ex. 7.24–

Ex. 7.25–

Ex. 7.26–

Ex. 7.27–

Ex. 7.28–

Ex. 7.29–

Ex. 7.30–

Ex. 7.31–

Ex. 7.32–

Ex. 7.33–

Ex. 7.34–

Ex. 7.35–

Ex. 7.36 –

Ex. 7.37 –

Ex. 7.38 –

Ex. 7.39 –

Ex. 7.40 –

Ex. 7.41 –

Ex. 7.42 –

Ex. 7.43 –

Ex. 7.44 –

Ex. 7.45 –