**Java Programming: Lab 5**

**Files and Exceptions**

1. You should complete the Essentials problems in class. Paste your code into this worksheet as directed and answer any questions in italics. At the end of class, save this worksheet and upload it to this lab's dropbox. If you have not finished, upload the code that you have written at the end of class. You should finish the remaining Essential problems outside of class.

2. Outside of class, work on as many Advanced problems as you can. Usually you should expect to spend 2-3 hours on these problems. Paste your code into this worksheet as directed and answer any questions in italics. Save your work and upload this worksheet to this lab's dropbox before the deadline.

***You should always comment all of your code!***

You are encouraged to work with other students, and to give others assistance if they need. And, please ask the instructor if you have any questions.

**Essentials – Work on these in class and upload at the end of class**

**Problem 0:**

Make sure you can get version control set up for a sample project. Please let me know if you have any problems or questions. As a developer, you will be using version control every day at work, so it is well worth your time to learn how to use version control. Git is a very complex and powerful tool, and rather intimidating to learn. So for now, focus on the basics that we covered in class, and then try out more features as you become more comfortable with it.

*Have you got version control set up?*

**MF: Yup yup!**

**Problem 1:**

If you didn’t finish this program from the slides, please do. Use the Buffered classes.

* Write a program that uses the buffered classes to write out your name, your favorite color, and this class's code (2545) to a new file – one item per line.

Now, read in all of the data, line by line, and display it.

package com.mark;  
  
import java.io.BufferedReader;  
import java.io.BufferedWriter;  
import java.io.FileReader;  
import java.io.FileWriter;  
import java.io.IOException;  
  
*/\*\*  
 \* This program writes some lines to a text file and then reads  
 \* the same lines.  
 \*/*public class problem\_1 {  
 public static void main(String[] args) throws IOException {  
 // Creates FileWriter and BufferedWriter objects.  
 FileWriter writer = new FileWriter("lab5\_problem1.txt");  
 BufferedWriter buffWriter = new BufferedWriter(writer);  
 // Writes lines to text file.  
 buffWriter.write("Mark Fox\n");  
 buffWriter.write("green\n");  
 buffWriter.write("2545\n");  
 // Closes object.  
 buffWriter.close();  
 // Creates FileReader and BufferedReader objects.  
 FileReader reader = new FileReader("lab5\_problem1.txt");  
 BufferedReader buffReader = new BufferedReader(reader);  
 // Reads first line of text file.  
 String line = buffReader.readLine();  
 // Continue reading lines until nothing is returned.  
 while (line != null) {  
 // Displays line.  
 System.*out*.println(line);  
 line = buffReader.readLine();  
 }  
 // Closes object.  
 buffReader.close();  
 }  
}

**Problem 2:**

Start wit your recycling truck program from last time.. Add code so that it writes the data as a report to a file. The report file will look something like this example,

House 0 recycled 1 crate

House 1 recycled 2 crates

...

...

Total crates recycled = 29

Houses that recycled the most

House 6: 4 crates

House 7: 4 crates

package com.mark;  
  
import java.io.BufferedWriter;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.Scanner;  
  
/\*  
 \* This program prompts User for values of a street's recycling  
 \* habits and then determines which house sets out the most  
 \* recyclables. These values are written to a text file.  
 \*/  
  
public class problem\_2 {  
 // Creates Scanner object.  
 static Scanner *numberScanner* = new Scanner(System.*in*);  
  
 public static void main(String[] args) throws IOException {  
 // Adds exception handler and includes creating BufferedWriter object.  
 try (BufferedWriter buffWriter = new BufferedWriter(new FileWriter("lab5\_problem2.txt"))) {  
 // Defines static number of houses.  
 int numberOfHouses = 8;  
 // Runs function that prompts User for input.  
 int[] cratesPerHouse = *getRecyclingPerHouse*(numberOfHouses);  
 // Runs function to sum up the User's inputs.  
 int totalCrates = *calculateTotal*(cratesPerHouse);  
 // Runs function to determine the max.  
 int maxCrates = *calculateMax*(cratesPerHouse);  
 // Runs function to determine the min.  
 int minCrates = *calculateMin*(cratesPerHouse);  
 // Runs function to determine the house(s) with the most crates.  
 ArrayList<Integer> housesWithMostRecycling = *calculateHouseWithMostRecycling*(cratesPerHouse, maxCrates);  
 // Loops through array and writes how many crates each house used.  
 for (int x = 0; x < cratesPerHouse.length; x++) {  
 buffWriter.write("House " + x + " recycled " + cratesPerHouse[x] + " crate(s)\n");  
 }  
 // Writes the total number of crates.  
 buffWriter.write("\nTotal crates recycled = " + totalCrates + "\n\n");  
  
 // Determines if multiple houses have the max count.  
 if (housesWithMostRecycling.size() > 1) {  
 // Starts writing for multiple houses.  
 buffWriter.write("Houses that recycled the most:\n");  
  
 // Loops through ArrayList and writes each house along with the max value.  
 for (int x = 0; x < (housesWithMostRecycling.size()); x++) {  
 buffWriter.write("House " + housesWithMostRecycling.get(x) + ": " + maxCrates + " crates\n");  
 }  
 } else {  
 // Writes a single house and its number of crates.  
 buffWriter.write("House that recycled the most:\n");  
 buffWriter.write("House " + housesWithMostRecycling.get(0) + ": " + maxCrates + " crates\n");  
 }  
 // Closes Scanner and Writer objects.  
 *numberScanner*.close();  
 buffWriter.close();  
 }  
 catch (IOException err) {  
 // Displays a simple error message.  
 System.*out*.println("Can't write to file.");  
 }  
 }  
 // Ask user for number of crates for each house. Store in array and return this array.  
 public static int[] getRecyclingPerHouse(int houses) {  
 // Creates empty array the size of provided houses.  
 int[] data = new int[houses];  
 // Loops through array length, prompts User, and collects input.  
 for (int x = 0; x < data.length; x++) {  
 System.*out*.println("How many recycling crates are " +  
 "set out for House #" + x + "?");  
 int input = *numberScanner*.nextInt();  
 // Adds input to array.  
 data[x] = input;  
 }  
 // Returns array.  
 return data;  
 }  
 //Add up all of the numbers in the array and return that  
 public static int calculateTotal(int[] cratesPerHouse) {  
 // Creates accumulator variable.  
 int total = 0;  
 // Loops through array and sums the content.  
 for (int x : cratesPerHouse) {  
 total += x;  
 }  
 // Returns total variable.  
 return total;  
 }  
 //Which is the largest number in the array?  
 public static int calculateMax(int[] cratesPerHouse) {  
 // Defines max variable.  
 int max = 0;  
 // Loops through array and compares item to max variable.  
 // Sets item as new max if greater than current max.  
 for (int x : cratesPerHouse) {  
 if (x > max) {  
 max = x;  
 }  
 }  
 // Returns max variable.  
 return max;  
 }  
 //Which is the smallest number in the array?  
 public static int calculateMin(int[] cratesPerHouse) {  
 // Defines min variable with the first array item.  
 // The min will be at least one of these items.  
 int min = cratesPerHouse[0];  
 // Loops through array and compares item to min variable.  
 // Sets item as new min if less than current min.  
 for (int x : cratesPerHouse) {  
 if (x < min) {  
 min = x;  
 }  
 }  
 // Returns min variable.  
 return min;  
 }  
 //Use the array to figure out which house number - or array element number - has the most recycling  
 public static ArrayList calculateHouseWithMostRecycling(int[] cratesPerHouse, int max) {  
 // Creates new ArrayList object.  
 ArrayList<Integer> housesWithMost = new ArrayList<>();  
 // Loops through provided array and checks if each item  
 // is equal to the provided max value. Matches are added  
 // to the ArrayList.  
 for (int x = 0; x < cratesPerHouse.length; x++) {  
 if (cratesPerHouse[x] == max) {  
 housesWithMost.add(x);  
 }  
 }  
 // return ArrayList.  
 return housesWithMost;  
 }  
}  
//Template used from minneapolis-edu GitHub.

**Problem 3:**

Look for a file called recyclingreport-mainstreet.txt in the Content > Labs tab of D2L. Drag it into the root directory of your project. This file contains data for a much longer street. Can you read it into your program, process the data, and then display the numbers of the house(s) that recycled the most?

package com.mark;  
  
import java.io.BufferedReader;  
import java.io.FileReader;  
import java.io.IOException;  
import java.util.ArrayList;  
  
*/\*\*  
 \* This program reads in lines from a text file, captures specific  
 \* substrings from each line, and determines which houses have  
 \* the most crates.  
 \*/*public class problem\_3 {  
 public static void main(String[] args) {  
 // Adds exception handler with resource.  
 try (BufferedReader buffReader = new BufferedReader(new FileReader("recyclingreport-mainstreet.txt"))){  
 // Creates ArrayList to hold all lines from file.  
 ArrayList<String> allLines = new ArrayList<>();  
 // Reads first line.  
 String line = buffReader.readLine();  
 // Loops through the rest of the lines until null is  
 // returned.  
 while (line != null) {  
 allLines.add(line);  
 line = buffReader.readLine();  
 }  
 // Closes Reader object.  
 buffReader.close();  
 // Creates ArrayList to HOUSE the arrays of houses and crates.  
 ArrayList<int[]> housesAndCrates = new ArrayList<>();  
 // Loops through first ArrayList and splits each line.  
 for (String item : allLines) {  
 String[] temp = item.split(" ");  
 // Grabs the second and fourth substrings and converts  
 // their values to integers.  
 int house = Integer.*parseInt*(temp[1]);  
 int crates = Integer.*parseInt*(temp[3]);  
 // Stores both converted values in a new array.  
 int[] tempArr = new int[2];  
 tempArr[0] = house;  
 tempArr[1] = crates;  
 // Adds array to ArrayList.  
 housesAndCrates.add(tempArr);  
 }  
 // Runs function to determine the max number of crates.  
 int maxCrates = *calculateMax*(housesAndCrates);  
 // Runs function to determine all the houses with the max  
 // number of crates.  
 ArrayList<Integer> housesWithMost = *calculateHouseWithMostRecycling*(housesAndCrates,  
 maxCrates);  
 // Loops through returned ArrayList and adds its values  
 // to a new string variable along with a separator.  
 String houseStr = "";  
 for (int x = 0; x < housesWithMost.size() - 1; x++) {  
 houseStr += housesWithMost.get(x) + ", ";  
 }  
 // Adds the last item without a separator.  
 houseStr += housesWithMost.get(housesWithMost.size() - 1);  
 // Displays the house numbers with the most crates.  
 System.*out*.print("Houses with the most crates: " + houseStr);  
 }  
 // Catch for IO exceptions.  
 catch (IOException err) {  
 // Displays a simple message.  
 System.*out*.println("Can't read file.");  
 }  
 }  
  
 public static int calculateMax(ArrayList<int[]> cratesPerHouse) {  
 // Defines max variable.  
 int max = 0;  
 // Creates temporary integer variable.  
 int temp;  
  
 // Loops through ArrayList and compares second array item to  
 // max variable. Sets item as new max if greater than current max.  
 for (int x = 0; x < cratesPerHouse.size(); x++) {  
 temp = cratesPerHouse.get(x)[1];  
 if (temp > max) {  
 max = temp;  
 }  
 }  
 // Returns max variable.  
 return max;  
 }  
  
 public static ArrayList calculateHouseWithMostRecycling(ArrayList<int[]> cratesPerHouse, int max) {  
 // Creates new ArrayList object.  
 ArrayList<Integer> housesWithMost = new ArrayList<>();  
 // Loops through provided ArrayList and checks if each item  
 // is equal to the provided max value. Matches are added  
 // to the new ArrayList.  
 for (int x = 0; x < cratesPerHouse.size(); x++) {  
 // Houses are index 0; crates are index 1.  
 if (cratesPerHouse.get(x)[1] == max) {  
 housesWithMost.add(cratesPerHouse.get(x)[0]);  
 }  
 }  
 // return ArrayList.  
 return housesWithMost;  
 }  
}

**Problem 4:**

If you haven’t already, add try-catch blocks for IOException to Question 2 and Question 3's programs. .

*Replace the Java code for those programs with the version with exceptions:*

**Done.**

**Problem 5:**

***Question:*** *Java doesn't make you deal with possible NullPointerException, or ArrayIndexOutOfBoundsException. So, why does Java insist that you deal with IOException in some way? Java's checked vs. unchecked exceptions are unusual and questions on this subject are common in Java job interviews!*

**NullPointerException and ArrayIndexOutOfBoundsException, among others, are implicitly imported into each Java program. These are available to be used when necessary; however, other exceptions are not included and need to be caught if one of them is thrown.**

**Problem 6:**

This code crashes. Can you **add exception handling** to catch the error and print a message that the LinkedList is empty?

LinkedList languages = new LinkedList();

languages.push("Python");

languages.push("Java");

languages.push("C#");

while (true) {

String oneLanguage = (String)languages.pop();

System.out.println(oneLanguage);

}

System.out.println("The list is empty")

***Question:*** *what data structure is LinkedList behaving as in this code? Hint: what are push() and pop() used with?*

**This LinkedList is behaving like a Stack.**

package com.mark;  
import java.util.LinkedList;  
import java.util.NoSuchElementException;  
  
*/\*\*  
 \* This program is corrected to include exception handling.  
 \*/*public class problem\_6 {  
 public static void main(String[] args) {  
 // Exception handler  
 try {  
 LinkedList languages = new LinkedList();  
 languages.push("Python");  
 languages.push("Java");  
 languages.push("C#");  
  
 while (true) {  
 String oneLanguage = (String)languages.pop();  
 System.*out*.println(oneLanguage);  
 }  
 }  
 // Catches error thrown.  
 catch (NoSuchElementException err) {  
 System.*out*.println("The list is empty");  
 }  
 }  
}

**Problem 7:**

Instead of waiting for an exception and catching, it is usually better to try and prevent problems.

Can you think of a better way to stop this code crashing? Modify this program so that it runs correctly without using a try-catch block.

package com.mark;  
import java.util.LinkedList;  
import java.util.NoSuchElementException;  
  
*/\*\*  
 \* This program is corrected and avoids using exception handling.  
 \*/*public class problem\_7 {  
 public static void main(String[] args) {  
 LinkedList languages = new LinkedList();  
 languages.push("Python");  
 languages.push("Java");  
 languages.push("C#");  
 // Changed while's condition to check the list's size.  
 while (languages.size() > 0) {  
 String oneLanguage = (String)languages.pop();  
 System.*out*.println(oneLanguage);  
 }  
 System.*out*.println("The list is empty");  
 }  
}

**Advanced**

Finish any incomplete Essentials problems first.

**Problem 1:**

Write a program that creates a sales report for a coffee shop. The coffee shop will use this at the end of every day to calculate sales, expenses, and profit.

The coffee shop sells 12 different drinks. The name of each drink, the price the shop charges the customer, and how much it costs to make each drink, are saved in the file coffee.txt, which you'll find in the Assignments section. The data is in the format

name;cost to make;price charged

As in this example,

cappuccino;1.56;3.50

So the cappuccino drink costs the coffee shop $1.56 to make, and they charge the customer $3.50.

Your program should read this data from coffee.txt, and **store it in some kind of data structure**.

Next, your program should ask the user how many of each drink were sold today. So, it might ask "How many cappuccino drinks were sold today?" and the user can type in a number. It will need to ask for each drink individually. You'll need to store this data somehow.

You should **add input validation** so that if the user enters "ten" or "10.4" or -45 then your program will respond appropriately.  You should also deal with any file-related exceptions properly.

Once you have gathered all the data, generate a report that will be written out to a new file called sales-report.txt. For each drink, record the number of drinks sold, the total that it cost to make those drinks (expenses), and the total amount (revenue) spent by customers on that drink. So, for example, if the coffee shop sold 10 cappuccinos today, you'll write a line that says

Cappuccino: Sold 10, Expenses $15.60, Revenue $35.00, Profit $19.40

Your output file should have a similar line for each of the drinks.

At the bottom of the file, write the total expenses, total revenue, and total profit for all drinks.

**You should use try-with-resources exception handling** for both file reading, and file writing.

Test and comment your code.

package com.mark;  
  
import java.io.\*;  
import java.text.DateFormat;  
import java.text.SimpleDateFormat;  
import java.util.ArrayList;  
import java.util.Date;  
import java.util.HashMap;  
import java.util.Scanner;  
  
*/\*\*  
 \* This program generates a sales report after receiving input from  
 \* the User. Prices and costs are located in a separate file and the  
 \* report is written to a new file.  
 \*/*public class adv\_problem\_1 {  
 public static void main(String[] args) {  
 // Exception handler with a new BufferedReader inside it.  
 try (BufferedReader buffReader = new BufferedReader(new FileReader("coffee.txt"))) {  
 // Retrieving the prices and costs from file.  
 ArrayList<String[]> all\_drinks = *readDrinkInfo*(buffReader);  
 // Retrieving user inputs.  
 HashMap<String, Integer> salesData = *askSales*(all\_drinks);  
 // Null returns indicate incorrect inputs.  
 if (salesData != null) {  
 // Runs function to create report file.  
 *makeSalesReport*(all\_drinks,salesData);  
 }  
 }  
 catch (IOException err) {  
 // The file could not be found and/or read.  
 System.*out*.println("Unable to read/find file.");  
 }  
 }  
  
 public static ArrayList<String[]> readDrinkInfo(BufferedReader buffReader) {  
 // Exception handler  
 try {  
 // Creates ArrayList and reads first line of file.  
 ArrayList<String[]> all\_drinks = new ArrayList<String[]>();  
 String drink\_line = buffReader.readLine();  
 // Continues reading lines and adding its data to ArrayList  
 // until there are no more lines.  
 while (drink\_line != null) {  
 String[] drink\_info = drink\_line.split(";");  
 all\_drinks.add(drink\_info);  
 drink\_line = buffReader.readLine();  
 }  
 // Closes reader and returns ArrayList.  
 buffReader.close();  
 return all\_drinks;  
 }  
 catch (IOException err) {  
 // The file could not be read.  
 System.*out*.println("There is a problem reading the file.");  
 return null;  
 }  
 }

public static HashMap askSales(ArrayList<String[]> all\_drinks) {  
 // Creates Scanner object and HashMap.  
 Scanner numScanner = new Scanner(System.*in*);  
 HashMap<String, Integer> salesData = new HashMap<>();  
 // Loops through ArrayList and prompts User for input.  
 for (int i = 0; i < all\_drinks.size(); i++) {  
 // Index 0 should contain the drink name.  
 System.*out*.println("How many " + all\_drinks.get(i)[0] + " drinks were sold today?");  
 // Captures input.  
 String sales\_query = numScanner.nextLine();  
 int sales\_num = 0;  
 // Checks if the input was not an integer.  
 if (sales\_query.matches("-?\\d+(\\d+)?")) {  
 // Converts input to integer and adds to HashMap.  
 sales\_num = Integer.*parseInt*(sales\_query);  
 salesData.put(all\_drinks.get(i)[0], sales\_num);  
 }  
 else {  
 // User entered bad information.  
 System.*out*.println("Invalid input attempted.");  
 return null;  
 }  
 }  
 // Closes Scanner and returns HashMap.  
 numScanner.close();  
 return salesData;  
 }  
  
 public static void makeSalesReport(ArrayList<String[]> all\_drinks, HashMap salesData) {  
 // Exception handler with new BufferedWriter created inside it.  
 try (BufferedWriter buffWriter = new BufferedWriter(new FileWriter("sales-report.txt"))) {  
  
 // Determines current date, formats it accordingly, and writes  
 // to file as first line.  
 Date today = new Date();  
 DateFormat formatDate = new SimpleDateFormat("MMMM dd, yyyy");  
 String todayStr = formatDate.format(today);  
 buffWriter.write(String.*format*("Sales Report for %s:\n", todayStr));  
  
 // Defines accumulators.  
 double totalExpenses = 0;  
 double totalRevenue = 0;  
 double totalProfit = 0;  
  
 // Loops through ArrayList, captures values, and performs  
 // calculations.  
 for (Integer i = 0; i < all\_drinks.size(); i++) {  
 String drink = all\_drinks.get(i)[0];  
 int sold = (int) salesData.get(drink);  
 double costPer = Double.*parseDouble*(all\_drinks.get(i)[1]);  
 double cost = costPer \* sold;  
 double pricePer = Double.*parseDouble*(all\_drinks.get(i)[2]);  
 double earned = pricePer \* sold;  
 double profit = earned - cost;  
  
 // Adds values to appropriate accumulators.  
 totalExpenses += cost;  
 totalRevenue += earned;  
 totalProfit += profit;  
  
 // Creates strings of values and adds formatting to them.  
 // I was unable to get formats added to all variables with  
 // a single builder so this was plan B.  
 String drinkStr = drink + ":";  
 String soldStr = "Sold " + sold + ",";  
 String costStr = String.*format*("Expenses $%.2f,", cost);  
 String earnedStr = String.*format*("Revenue $%.2f,", earned);  
 String profitStr = String.*format*("Profit $%.2f", profit);  
 buffWriter.write(String.*format*("%-14s %-2s %-14s %-14s %s\n",  
 drinkStr, soldStr, costStr, earnedStr, profitStr));  
 }  
 // Writes the totals to the end of the file.  
 buffWriter.write(String.*format*("\nTotal Expenses: $%.2f\n", totalExpenses));  
 buffWriter.write(String.*format*("Total Revenue: $%.2f\n", totalRevenue));  
 buffWriter.write(String.*format*("Total Profits: $%.2f", totalProfit));  
  
 // Closes writer and displays success message.  
 buffWriter.close();  
 System.*out*.println("Report successfully created.");  
 }  
 catch (IOException err) {  
 // There was a problem writing to the file.  
 System.*out*.println("Unable to write to file.");  
 }  
 }  
}  
  
//helpful sites:  
//http://stackoverflow.com/questions/1102891/how-to-check-if-a-string-is-numeric-in-java

**Problem 2:**

All of the code so far has assumed that we are working with a file in the root directory of your project. But, a file could be anywhere on your system, so Java needs to support reading files anywhere on your computer (and even on remote computers). Perhaps your program needs several data files, and it would be nice to organize them into one /data directory in your project directory.

Create a /data/ directory in your project directory. Write a program that creates and stores a test file in this directory.

FileWriter can take a filename as an argument. The filename can only be a file in your current directory. So you can’t say

new FileWriter fw = new FileWriter(“/data/testing.txt”);

But, FileWriter can also take a File object as an argument, and a File object can store a path to a file elsewhere on your system (or even on a remote system).

Here’s how to create a File object;

File f = new File(“/mydirectory/mysubdirectory/textfile.txt”);

Assuming these directories exist, you can create and write that file in this location. The location is relative to the current directory. I’ll leave it to you to put these pieces together.

Notice that one file system uses forward slashes, and one uses backslashes, so you'll need to use the correct \ or / for your computer. Mac is also case-sensitive.

[Note: You can also use absolute pathnames, for example

C:\program files\java\textfile.txt on Windows, or

/users/admin/Documents/textfile.txt on a Unix-based system like Linux or Mac,

but if you do that, your program may not work on anyone else's computer! ]

Make sure you catch and handle any exceptions.

*Paste your Java program here:*

**Problem 3:** Java is architecture neutral and the same code is supposed to run on PC and Mac and Linux, and anything else a JVM can be install on. Until we got to files, it does, and we'd like to maintain that as we work with files.

How can your code know what operating system it is running on, and specify files with backslash or forward slashes?

This does need a little work on the part of the programmer to help code run correctly on different platforms. How would this help?

<http://docs.oracle.com/javase/tutorial/essential/environment/sysprop.html>

Can you re-write your code above using system properties, so it will run on Windows and Mac?

Test your program on a Windows and Mac computer. There are Macs and PCs in the hallway outside T3050.

*Paste your Java program here:*

**Problem 4:**

Write a program that writes the OS name, OS version, OS architecture, and user account name to a file, without hard coding any of these things. Test your program on a Windows and Mac computer. There are Macs and PC computers in the hallway outside T3050.

*Paste your Java program here:*

*And, please paste a* ***screenshot*** *of your program's output on each platform here:*

*Mac: Command+Shift+Control+4, drag the crosshairs over the desired area of screen to copy a screenshot to the clipboard. Return to word, press Command+V to paste the screenshot.*

*PC: in Word's menu, there's an Insert screenshot option, look for the Screen Clipping option.*