TED UNIVERSITY

CMPE 223/242
Programming Homework #1
“Load Delivery”

Author:
Rahymberdi Annyyev

Problem Statement and Code Design

In this HW Assignment, we must consider the Problem in the most general case. For that, we must briefly state the problem itself. We have 4 text files:

1. Destinations
2. Loads
3. FlowDevices
4. Missions

In Destinations, we have a list of Cities from which we will operate. In Loads, we have load packages distributed across all the cities in the Destinations list. In FlowDevices, we have flow devices that are unique to each city and act as a transporter of load packages across the city bases. In Missions, we have lines of commands that should be executed on load packages. They are in the form:

X-Y-Z-a-b-c1,c2…

X – First City.
Y – Middle City.
Z – Final City.
A – Amount of load packages to be delivered to Z base from X.
B - Amount of load packages to be delivered to Z base from Y.
C1, C2, … - Indexes of the load packages from the FlowDevice to be delivered Y base.

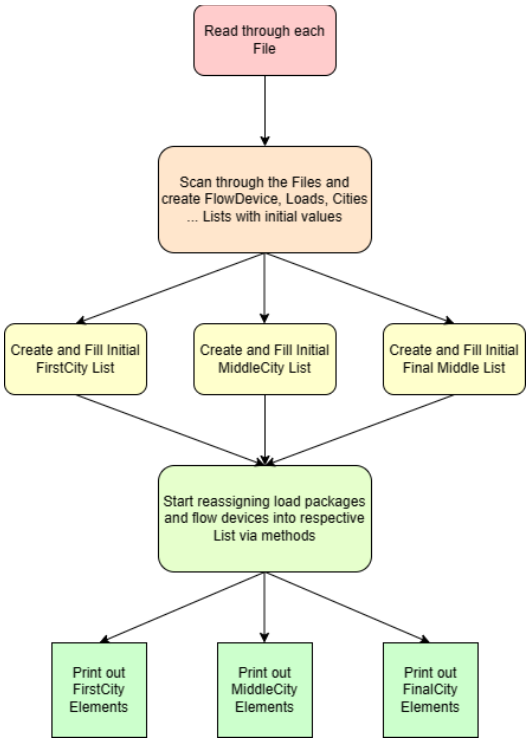
So, we have a case of Load Delivery, and to accomplish this task we have to have some kind of algorithm, and data structures to use. For a complex problem like this, we should break it down into smaller pieces, and create an order of execution.

Pseudo-code:

Let's first consider how we will get the information from the txt files. We create files for each text and assign a scanner through which we can get data. After that, we start to sort out each type of data we get into respective Double-Linked Lists. I used Double-Linked Lists for this Assignment because they can be used as a Stack and a Queue at the same time. All the methods such as pop(), push(), enqueue(), dequeue(), etc.. can all be implemented by DL (Double-Linked) Lists. (The challenge comes with tracking indexes in each of these lists to correctly assign Nodes from one list to another). After we sorted them all out, we can start to execute the mission given to us. We clear the Missions.txt from all signs and replace them with spaces. We read the three first String elements and start to assign each string to 3 lists that are essentially what we will be operating on.
We will have FirstCity, MiddleCity, and FinalCity which will correspond to X, Y, and Z respectively. We will fill them up by "pushing" into their own load packages. It will come in handy in the second step. After filling each one of them, we start to "pop" the load packages we need to assign to FinalCity from FirstCity and MiddleCity. After doing so, we have to read the

last numbers which would tell us which of those load packages should be assigned to MiddleCity. Only after that, we will add at the end Flow Devices into the Lists, picking one of them from FirstCity and assigning it to the FinalCity. After completing these steps, we print our results.

We can also illustrate these steps in this diagram:



Implementation and Functionality

After we briefly discussed our algorithm, let's take a closer look at each of the steps.

In our code, we will first be greeted by public static variables that will be used throughout the MainClass:

```
1.  import java.io.File;
2.  import java.io.IOException;
3.  import java.util.LinkedList;
4.  import java.util.Scanner;
5.
6.  /*
```

```
7.   * Title: My Main Class
8.   * Author: Rahymberdi Annyyev
9.   * ID: 99597840304
10.  * Section: 03
11.  * Assignment: 1
12.  * Description: This Class contains all methods that creates bulk of the code itself
13.  */
14.
15. public class MainClass {
16.
17.         public static LinkedList<String> DestinationList = new LinkedList<>();
18.         public static LinkedList<String> FlowDevicesList = new LinkedList<>();
19.         public static LinkedList<String> LoadsList = new LinkedList<>();
20.         public static LinkedList<String> MissionList = new LinkedList<>();
21.         public static LinkedList<String> FirstList = new LinkedList<>();
22.         public static LinkedList<String> MiddleList = new LinkedList<>();
23.         public static LinkedList<String> FinalList = new LinkedList<>();
24.         public static int FinalInitialSize;
25.         public static LinkedList<Integer> ArrayOfLastInts = new LinkedList<>();
26.         public static int SizeofFirstBeforeT;
27.         public static int SizeofMiddleBeforeT;
28.         public static int SizeofFinalBeforeT;
29.         public static File dest;
30.         public static File flow;
31.         public static File load;
32.         public static File miss;
```

From lines 17 and 23 we declared some of the LinkedList<String> which are essentially DL Lists. The first 4 (17-20) Lists are the ones in which we push each type of information as destinations, load packages, etc. After that from lines 21 to 23, we declare our First, Middle, and Final Stations. From lines 24 to 28, we have some variables that will be useful and explained later on, so we will have to pass them right now. And the last lines we have our Files variables that will be used as placeholders for any txt files that will be executed in this code.

Let's look now at our main() method:

```
1.  public static void main(String[] args) throws IOException {
2.
3.                  /*
4.                   * We read the Default Case if the user does not want to write new
    addresses for
5.                   * other txt files
6.                   */
7.                  if (TestClass.k == 0) {
8.
9.                          dest = new File("texts\\destinations.txt");
10.                         flow = new File("texts\\flowdevices.txt");
11.                         load = new File("texts\\loads.txt");
12.                         miss = new File("texts\\missions.txt");
13.                 }
14.                 /*
15.                  * Then we create Scanners that will allow us to access the Files and
    their
16.                  * elements
17.                  */
18.                 Scanner DestKey = new Scanner(dest);
19.                 Scanner FlowKey = new Scanner(flow);
20.                 Scanner LoadKey = new Scanner(load);
```

```
                Scanner MissKey = new Scanner(miss);
                /* We Clear the Mission.txt from '-' and ',' characters */
                String MissString = MissKey.nextLine();
                MissString = MissString.replace('-', ' ');
                MissString = MissString.replace(',', ' ');
                Scanner MissStringKey = new Scanner(MissString);
                /* We create the List of Initial Destinations */
                CreateDestList(DestKey);
                System.out.println("Initial List of Destinations: ");
                System.out.println(DestinationList.toString());
                /* We create the List of Initial FlowDevices */
                CreateFlowList(FlowKey);
                System.out.println("Initial List of Flow Devices: ");
                System.out.println(FlowDevicesList.toString());
                /* We create the List of Initial Load Packages */
                CreateLoadList(LoadKey);
                System.out.println("Initial List of Load Packages: ");
                System.out.println(LoadsList.toString());
                /* We create the List of Initial Missions */
                CreateMissList(MissStringKey);
                System.out.println("Mission List: ");
                System.out.println(MissionList.toString());
                System.out.println("");

                /*
                 * We iterate through the MissionList, and we Assign each element either to
                 * FirstCity, Second city and etc.
                 * After we start to assign the amount of the load packages to the "Train" (i.e.
                 * the FlowDevice of The FirstCity)
                 */
                for (int i = 0; i < MissionList.size(); i++) {
                        if (i == 0) {
                                CreateFirst(MissionList.get(i));
                        }
                        if (i == 1) {
                                CreateMiddle(MissionList.get(i));
                        }

                        if (i == 2) {
                                CreateFinal(MissionList.get(i));
                        }

                        if (i == 3) {
                                int k = Integer.parseInt(MissionList.get(i));
                                TakeoutFirst(k);
                        }

                        if (i == 4) {
                                int k = Integer.parseInt(MissionList.get(i));
                                TakeoutMiddle(k);
                        }

                        if (i > 4) {
                                int k = Integer.parseInt(MissionList.get(i));
                                ArrayOfLastInts.add(k);
                        }

                }
                /* We get out the elements from the Train for our SecondCity */
```

```
80.                 LastReshuffle(ArrayOfLastInts);
81.                 /*
82.                  * We fixate the sizes of each of the Lists before pushing the
    Flowdevices into
83.                  * each of the Lists
84.                  */
85.                 SizeofFirstBeforeT = FirstList.size();
86.                 SizeofMiddleBeforeT = MiddleList.size();
87.                 SizeofFinalBeforeT = FinalList.size();
88.                 /* We add FlowDevices into Respective Lists */
89.                 TAdditionFirst(FirstList);
90.                 TAdditionMiddle(MiddleList);
91.                 /* We print each of the Lists at the End */
92.                 FirstToString(FirstList);
93.                 MiddleToString(MiddleList);
94.                 FinalToString(FinalList);
95.                 /* We close each Scanner */
96.                 DestKey.close();
97.                 LoadKey.close();
98.                 FlowKey.close();
99.                 MissKey.close();
100.
101.                        /* The End. */
102.
103.             }
```

As we can see, our main method is quite huge, so we will go through step by step. Now, on the first line, we can see "`throws IOException`" statement. This is a necessary exception to be implemented because there is a chance that the files we want to read either are empty or non-existent.

Through lines 7 to 13, we assign to each file txt files that are default in order to check if the code either works or not. If k was assigned to 1 from TestClass, then they would be assigned to custom files.

From lines 18 to 43 we make our preparations before shuffling load packages according to Missions.txt. First, we create Scanners that will allow us to read each file. After that, we push into each one of them their respective data types, and then we print them out.

At line 51 we start working with the Mission file. We do so by a for loop because it is important as to which element we are reading right now. Let's look at cases from `(i == 0)` to `(i == 2)`. Because all 3 cases use similar methods, let's look at just one:

```
1.  if (i == 0) {
2.                                  CreateFirst(MissionList.get(i));
3.                          }
```

```
1.  public static void CreateFirst(String x) {
2.                  /*
3.                   * This method created the FirstCity List
```

```
                 * In the first for Loop it gets the name of the First City
                 */
                for (int a = 0; a < DestinationList.size(); a++) {
                        if (DestinationList.get(a).equals(x)) {
                                FirstList.add(DestinationList.get(a));
                                break;
                        }
                }
                /*
                 * In the second for loop it adds all the designated package loads to
   the First
                 * City
                 */
                for (int a = LoadsList.size() - 1; a >= 0; a--) {
                        if (LoadsList.get(a).equals(x)) {
                                FirstList.add(LoadsList.get(a - 1));
                        }
                }
        }
```

In the first for loop, we get the name for our FirstCity List by comparing each Node in DestinationsList and if we find the match, we assign it to our List. In the second for loop, we start to "push" load packages corresponding to the FirstCity name into our List. The same we do for Middle and Final Lists.

From `(i == 3)` to `(i == 4)` we have methods which are called `TakeoutFirst(k), TakeoutMiddle(k)` Because they have similar purposes, let us inspect the first method:

```
public static void TakeoutFirst(int k) {
        /* This method deals with load packages which were determined to be taken out
   */
        /* Copies the load packages from the FirstCity List k times */
        for (int s = 1; s <= k; s++) {
                String Temp = FirstList.get(s);
                FinalList.add(Temp);
        }
        /*
         * Deletes the load packages from the FirstCity List that were copied in the
         * first loop
         */
        for (int s = 1; s <= k; s++) {
                FirstList.remove(1);

        }
}
```

In our first loop, we copy each element that we need to transfer to FinalList. In the second loop, we delete all of them. These loops should be done separately because removing instantaneously reduces the size of the FirstList, thus making it hard to get all elements and remove them at the same time. The same is done for MiddleList.

In the case of `(i > 4)` we read the last integers in the MissionList and put them in the `ArrayOfLastInts`. As I mentioned, the most important thing is tracking. After completing the main for loop, we put this Array into the `LastReshuffle()` method. Let's look at this method:

```
1.  public static void LastReshuffle(LinkedList<Integer> x) {
2.
3.                  /*
4.                   * This Method at the end takes out the load packages from the
    FinalCity List
5.                   * and assigns them to the MiddleCity List according to last integers
    at the
6.                   * mission.txt
7.                   */
8.                  /* The first loop copies these packages to MiddleCity List */
9.                  int z = 0;
10.                 LinkedList<String> TemporaryList = new LinkedList<>();
11.                 for (int i = 0; i < x.size(); i++) {
12.                         MiddleList.add(FinalList.get(x.get(z) + FinalInitialSize));
13.                         TemporaryList.add(FinalList.get(x.get(z) + FinalInitialSize));
14.                         z++;
15.                 }
16.                 /* The second loop deletes these packages from FinalCityList */
17.                 for (int p = 0; p < FinalList.size(); p++) {
18.                         for (int y = 0; y < TemporaryList.size(); y++) {
19.                                 if (TemporaryList.get(y).equals(FinalList.get(p))) {
20.                                         FinalList.remove(p);
21.                                 }
22.                         }
23.                 }
24.         }
```

Because it was easier to push all of the Load Packages into the FinalList and only after that to get the specific ones and add them to MiddleList, I created a method that would do the last step. In the first loop, we copy all intended load packages from the Final List to the Middle List. After that, in the second for loop, we delete them from FinalList, thus finalizing every operation about load package delivery.

From lines 85 to 87, we save the sizes of each city list. We will need them during printing out the result of our operations to distinguish load package indexes from FlowDevices. In lines 89 and 90, we add at the end of our city lists FlowDevices. Let's consider the first method because it is more interesting than the second:

```
1.  public static void TAdditionFirst(LinkedList<String> x) {
2.                  /*
3.                   * This method adds FlowDevices to the FirstCity, but checks the first
4.                   * flowdevice that should be assigned to the FinalCityList
5.                   */
6.                  int check = 0;
7.                  for (int q = 1; q < FlowDevicesList.size(); q++) {
8.
9.                          if ((x.get(0).equals(FlowDevicesList.get(q))) && check > 0) {
10.                                 FirstList.add(FlowDevicesList.get(q - 1));
11.                         }
12.
13.                         else if (x.get(0).equals(FlowDevicesList.get(q))) {
14.                                 FinalList.add(FlowDevicesList.get(q - 1));
```

```
                                check++;
                        }

                }
        }
```

In the for loop, we first want to get the FlowDevice from FirstCity and assign it to FinalCity once, otherwise the flow devices will be assigned to FirstCity. The same will be true for MiddleCity but without the initial condition. This ends any operation made on the Lists themselves. Now we will print them out:

```
public static void FirstToString(LinkedList<String> x) {
                for (int i = 0; i < x.size(); i++) {
                        if (i == 0) {
                                System.out.println("City: ");
                                System.out.println(x.get(i));
                                System.out.println("Loads: ");
                        }
                        if (i > 0 && i < SizeofFirstBeforeT) {
                                System.out.println(x.get(i));
                        }

                        if (i >= SizeofFirstBeforeT) {
                                System.out.println("FlowDevices: ");
                                System.out.println(x.get(i));
                        }
                }
                System.out.println("---------------------------------");
        }
```

It is pretty much straightforward, but what needs to be mentioned is those variables we declared come in handy when distinguishing Name, Load packages, and Flow Devices. After printing each List, we close our Scanners, and we end our Program here.

Testing

To test my code, I created a TestClass that will allow us to put in any Txt files and data and check if it correctly executes the missions. It looks like this:

```
import java.io.File;
import java.util.Scanner;

/*This Class acts as a user-interface through which you can assign texts files and run
the missions*/

public class TestClass {

  public static int k;

  public static void main(String[] args) {
    Scanner keyboard = new Scanner(System.in);
    System.out.println("Welcome to the Load Package Managment System!");
```

```
13.     System.out.println("Rules of usage: ");
14.     System.out.println("1. When the program asks for the path to the File, write down
    the Absolute Path.");
15.     System.out.println("2. Files should not be null.");
16.     System.out.println(
17.         "3. In Mission File, last values of integers should not go out of boundry of
    the FlowDevice Train (i.e. the Package Loader List)");
18.
19.     System.out.println("Would you like to write your own paths or want to use defaults
    one?");
20.     System.out.println("Write 0 for default, 1 for your own paths");
21.     k = keyboard.nextInt();
22.
23.     if (k == 1) {
24.       System.out.println("Write down the path to Destination.txt");
25.       String destinationpath = keyboard.next();
26.       System.out.println("Write down the path to Loads.txt");
27.       String loadspath = keyboard.next();
28.       System.out.println("Write down the path to FlowDevices.txt");
29.       String flowdevicespath = keyboard.next();
30.       System.out.println("Write down the path to Mission.txt");
31.       String missionpath = keyboard.next();
32.
33.       MainClass.dest = new File(destinationpath);
34.       MainClass.load = new File(loadspath);
35.       MainClass.flow = new File(flowdevicespath);
36.       MainClass.miss = new File(missionpath);
37.     }
38.
39.     keyboard.close();
40.
41.     MainClass.MainRunner();
42.   }
43. }
```

The only thing to be explained is that if we write for k = 1, then we will have to write the Absolute Paths to the files mentioned. If we write 0, then the default case will be executed.

Final Assessments

As to mention the error that I got in my code is that it cannot read multiple lines in mission.txt because it will restart the Lists that have been changed previously, and will delete all the progress. I tried to figure out a way out of this, but I couldn't. I did not use ChatGPT in any way during the design and implementation of this code. So, I will point out this issue as an honest drawback of my program. Also, the choice of DL Lists as a main data structure is for the efficiency of the code. To efficiently use DL Lists, I had to create multiple tracking index variables that would help with this task. It is complex, but it allowed me to do this Assignment using only DLs. As for Credits, I will list sources down below.

Credits

1) GeeksforGeeks. (n.d.). Program to extract words from a given string. GeeksforGeeks. https://www.geeksforgeeks.org/program-extract-words-given-string/
2) GeeksforGeeks. (n.d.). How to Convert a String to a Path in Java. GeeksforGeeks. https://www.geeksforgeeks.org/how-to-convert-a-string-to-a-path-in-java/
3) CodingwithJohn. (Year, Month Day). Title of video [Video]. YouTube. https://www.youtube.com/watch?v=ScUJx4aWRi0&ab_channel=CodingwithJohn
4) Coding with John. (2021, April 26). Java File Input/Output - It's Way Easier Than You Think [Video]. YouTube. https://www.youtube.com/watch?v=ScUJx4aWRi0
5) Alex Lee. (2018, December 13). Files in Java - Read text file easily #38 [Video]. YouTube. https://www.youtube.com/watch?v=lHFlAYaNfdo