Assignment 2

CS 1027
Computer Science Fundamentals II

Due date: October 27 at 11:55 pm

Learning Outcomes

In this assignment, you will get practice with:

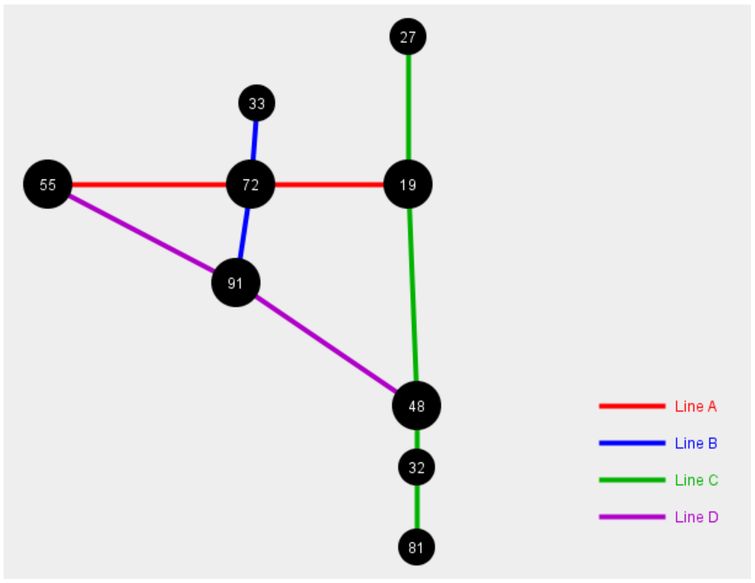
- Working with doubly linked lists
- Inheritance and overriding methods
- Using the instanceof operator
- Casting within an inheritance hierarchy
- Throwing exceptions

Introduction

London has been divided on the subject of building a light rail transit system to complement the city bus system. A rail system would allow passengers to ride around the city much more quickly than they can on the buses. A rail system typically has a few lines (routes) along the main corridors of the city on which the trains would travel back and forth.

In this assignment, you are implementing a computer simulation of a transit system prototype. In this program, a transit system will be formed as a series of Stations on which passengers can get on or off a train and the Stations are connected to each other in a doubly-linked-list type of mechanism. Some Stations, called InterchangeStations. are connected in more than one line. They are the kinds of Stations in which passengers can get off one train and on to another train going in a different direction. A TransitLine (route) typically contains several Stations – some of which are InterchangeStations – that are connected one by one along a specific path.

This program has a visual component to help you see the transit system once you implement the required classes and their methods. An example of a finished system is shown here:



Assignment 2

Provided files

- LTC.java
- LTCTests.java
- Map.java
- StationException.java

The LTCTests.java file will **help** to check if your java classes are implemented correctly. A similar file will be incorporated into Gradescope's auto-grader. **Passing all the tests within LTCTests.java does not necessarily mean that your code is correct.**

Classes to Implement

For this assignment, you must implement three (3) Java classes: **Station**, **InterchangeStation**, and **TransitLine**. Follow the guidelines for each one below.

In all of these classes, you may implement more private (helper) methods, and in some of the files, it is strongly encouraged that you do. However, you may not implement more public methods **except** public static void main(String[] args) for testing purposes (this is allowed and encouraged). You may not add instance variables other than the ones specified in these instructions nor change the variable types or accessibility (i.e. making a variable public when it should be private). Penalties will be applied if you implement additional instance variables or change the variable types or modifiers from what is described here.

Station.java

This class is used to represent a single Station in the transit system. It behaves like a node in a doubly-linked list. That is, a Station will have variables pointing to the previous and next Station so that they are connected in both directions.

The class must have these private instance variables:

- private static int nextID = 0
- private int id
- private int stnNo
- private String name
- private Station prev
- private Station next
- private int x
- private int y

The class must have the following public methods:

- public Station(int stnNo, String stnName, int x, int y) [constructor]

  - Initialize the instance variables stnNo, name, x, and y with the corresponding parameters. Set prev and next to null. Set id to the value of the static variable nextID and then increment nextID (this allows each subsequent Station object to be given a unique ID in increasing order).
- public int getID(), public int getStnNo(), public String getName(), public Station getPrev(), public Station getNext(), public int getX(), public int getY()
  - Getters for each of the instance variables
- public void setPrev(Station stn), public void setNext(Station stn)
  - Setters for the prev and next instance variables
- public String toString()
  - Returns the string containing the stnNo and name, formatted like this:
    Stn: 33 (Oxford)

InterchangeStation.java

This class is used to represent an InterchangeStation, which means a Station on 2 or more transit lines. This kind of station allows passengers to switch from one line to another. This class must be a sub-class of the Station class.

The class must have these private instance variables (in addition to the ones it inherits from Station):

- private Station[] prevs
  - Since this Station is on multiple transit lines, we must hold an array of prevs Stations so that it can have a previous for each line for which it is being used
- private Station[] nexts
  - Since this Station is on multiple transit lines, we must hold an array of nexts Stations so that it can have a next for each line for which it is being used

The class must have the following public methods:

- public InterchangeStation(int stnNo, String stnName, int x, int y) [constructor]
  - Call the super (Station) constructor with the corresponding parameters
  - Initialize prevs and nexts to have 2 spots each (this may have to increase later)
- public Station getPrev(), public Station getNext(), public void setPrev(Station stn), public void getNext(Station stn)
  - These methods are from Station but don't make sense for InterchangeStation since there is more than one prev and/or next station. Therefore, throw a StationException saying "InterchangeStation cannot use ________" with the corresponding method name in the blank in each of these methods.
- public Station getPrev(char lineLetter), public Station getNext(char lineLetter), public void setPrev(Station stn, char lineLetter), public void setNext(Station stn, char lineLetter)
  - These are the getters and setters that will be used instead of the previously explained overridden ones. These ones have a parameter lineLetter which is a character representing the line on which to get/set the previous or next Station.

  - For example, consider an InterchangeStation stn on lines A and C. stn.getNext('A') would return the Station after stn along the A line, and stn.getNext('C') would return the next one on the C line. stn.getNext('B') should return null since this Station is not on the B line.
  - Determine the line "num" by subtracting 65 from the lineLetter (65 is the ASCII code for A, 66 is the code for B, etc.), so this num will be 0 for 'A', 1 for 'B', etc. This will be used as the index in the prevs and nexts arrays.
  - Expand the capacity of the corresponding array (prevs or nexts) to num+1 (if it is out of the bounds of the existing array). For example, if a Station is on Line A and later added to Line C, the original arrays would just be 1 element while it is on Line A alone, but then would have to expand to 3 (num for C is 2 so the arrays need 2+1 = 3 spaces now) when added to Line C.
  - * Please see the example in blue for a clarification on how this works.
- public Station[] getPrevArray(), public Station[] getNextArray()
  - Return the corresponding array
- public String getPrevString(), public String getNextString()
  - Build and return a String containing the prevs / nexts showing the stnNo of the prev or next station for each line and __ (two underscores) for any null connection within the array.
  - Note that the array(s) are only increased when needed in the getters/setters described previously, so some InterchangeStations will have null spots in place of transit lines on which they are not laying, while others will not have a null because the arrays didn't need to increase to contain them.
  - * Please see the example in blue for a clarification on how this works.
- public String toString()
  - Return the String of the same format as the Station class, but after that, add " on Lines: " and list the letters representing the lines on which this InterchangeStation is laying. For example:
    Stn: 48 (Highbury) on Lines: C D

* Example to clarify how the prevs and nexts arrays are supposed to work

Consider three InterchangeStation objects with numbers: 15, 30, and 45 in a transit system that has three lines: A, B, and C.

Stn 15 is on lines A and B, Stn 30 is on lines B and C, and Stn 45 is on lines A and C.

Stn 15's prevs and nexts arrays would be of length 2 since it's not on line C and thus would not have to expand to 3 (even though there is a 3rd line in the system). Thus, Stn 15's getPrevString() might be something like: 74 92. [74 on the A line, 92 on the B line].

Stn 30's prevs and nexts arrays must be of length 3 since it is on line C so they need to contain info about their line C connections in index 2 (3rd element) of those arrays. It is not on line A so the first element will be null. Thus, Stn 30's getPrevString() might be something like: __ 63 28 [nothing on the A line, 63 on the B line, 28 on the C line].

Stn 45's prevs and nexts arrays must also be of length 3 since it is on line C. It is not on line B so the second element will be null. Thus, Stn 45's getPrevString() might be something like: 57 __ 88 [57 on the A line, nothing on the B line, and 88 on the C line].

Note that an InterchangeStation could be at the end of a line, so it may not have a previous or next Station even on the lines that it is laying. For example, suppose Stn 15 is at the end of line B, then it's getNextString() might be something like: 37 __ [37 on the A line, nothing after it on the B line]

Note also that an InterchangeStation can be on ANY number of lines. It should not be limited to just 2 lines.

TransitLine.java

This class is used to represent a line (route) in the transit system. Each line contains some number of Stations (including some InterchangeStations) that are connected in both directions. Each line is labelled with an alphabetical letter, 'A', 'B', 'C', etc.

The class must have these private instance variables:

- private char lineLetter
- private Station firstStn

The class must have the following public methods:

- public TransitLine(char letter, Station first) [constructor]
  - Initialize both instance variables from the given parameters
- public char getLineLetter(), public Station getFirstStn()
  - Getters for the two instance variables
- public boolean hasNext(Station stn)
  - Determine if there is a Station after the given Station stn. If it is an InterchangeStation, then you have to use the lineLetter to check if there is a next Station specifically on the line specified by the letter. Otherwise, just check if there is a next Station. Return true if there is a next Station, or false otherwise.
- public void addStation(Station stn)
  - Start at the firstStn and traverse to the last added Station. Hint: use the hasNext() method above in a loop to continue stepping through the TransitLine until there is no next Station on the line.
  - From the last added Station on this line, add the given Station stn by connecting both the previous and next links so that it is doubly-linked.
- public String toString()

  - Return a String containing the Line letter and the list of the stnNo's of the Stations on the line. It must be formatted like this:
    Line C: 27 19 48 32 81

Marking Notes

Functional Specifications

- Does the program behave according to specifications?
- Does it produce the correct output and pass all tests?
- Are the classes implemented properly?
- Does the code run properly on Gradescope (even if it runs on Eclipse, it is **up to you** to ensure it works on Gradescope to get the test marks)
- Does the program produces compilation or run-time errors on Gradescope?
- Does the program fail to follow the instructions (i.e. changing variable types, etc.)

Non-Functional Specifications

- Are there comments throughout the code (Javadocs or other comments)?
- Are the variables and methods given appropriate, meaningful names?
- Is the code clean and readable with proper indenting and white-space?
- Is the code consistent regarding formatting and naming conventions?
- Submission errors (i.e. missing files, too many files, etc.) will receive a penalty of 5%
- Including a "package" line at the top of a file will receive a penalty of 5%

Remember you must do all the work on your own. Do not copy or even look at the work of another student. All submitted code will be run through similarity-detection software.

Submission (due Thursday, October 27 at 11:55 pm)

Assignments must be submitted to Gradescope, not on OWL. If you are new to this platform, see these instructions on submitting on Gradescope.

Rules

- Please only submit the files specified below.

- Do not attach other files even if they were part of the assignment.
- Do not upload the .class files! Penalties will be applied for this.
- Submit the assignment on time. Late submissions will receive a penalty of 10% per day.
- Forgetting to submit is not a valid excuse for submitting late.
- Submissions must be done through Gradescope. **If your code runs on Eclipse but not on Gradescope, you will NOT get the marks! Make sure it works on Gradescope to get these marks.**
- You are expected to perform additional testing (create your own test harness class to do this) to ensure that your code works for other dice combinations as well. We are providing you with some tests but we may use additional tests that you haven't seen before for marking.
- Assignment files are NOT to be emailed to the instructor(s) or TA(s). They will not be marked if sent by email.
- You may re-submit code if your previous submission was not complete or correct, however, re-submissions after the regular assignment deadline will receive a penalty.

Files to submit

- Station.java
- InterchangeStation.java
- TransitLine.java

Grading Criteria

Total Marks: [20]

Functional Specifications:

[2] Station.java

[3] InterchangeStation.java

[2] TransitLine.java

[10] Passing Tests (some additional, hidden tests will be run on Gradescope)

Non-Functional Specifications:

[1] Meaningful variable names, private instance variables

[1] Code readability and indentation

[1] Code comments