Assignment 3 – Documentation

Contents

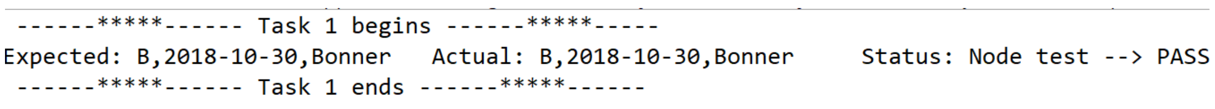
Task 1 – Implementing Node and Edge Classes

Node
- String name - LocalDate dateOB - String suburb protected HashMap<Node, Edge> adj
+ Node(String name, String date, String suburb) + void setName(String name) + void setDate (String date) + void setSuburb(String suburb) + void setHashMap(Node n, Edge e) + String getName() + LocalDate getDate() + String getSuburb() + Node getHashMap(Node n, Edge e) + String toString()

Node Tests

Test ID	Description	Input	Expected Output	Actual Output	Comments
1	Test if node was created	`n = new Node("B", LocalDate.parse("2018-10-30"), "Bonner");`	name = B dateOB = 2018-10-30 suburb = Bonner	`Node [name=B, dateOB=2018-10-30, suburb=Bonner]`	Pass
2	Incorrect node – purposeful error	`n2 = new Node("B", LocalDate.parse("2018000-09-30"), "Ford");`	Throw parsing error with LocalDate	`Text '2018000-09-30' could not be parsed at index 0`	Pass

Test Harness Results:



```
 ------*****------ Task 1 begins ------*****-----
Expected: B,2018-10-30,Bonner   Actual: B,2018-10-30,Bonner    Status: Node test --> PASS
 ------*****------ Task 1 ends ------*****------
```

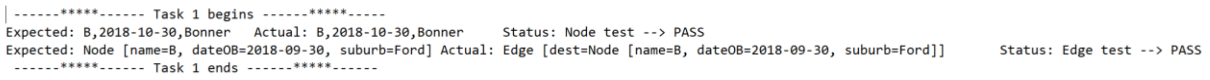
The Node test was passed.

Edge
- Node dest
+ Edge(Node dest) + Node getDest() + void setDest() + String toString()

<u>Edge Tests</u>

Test ID	Description	Input	Expected Output	Actual Output	Comments
1	Test if edge created	`edge = new Edge(n2);`	dest = Node With node details etc.	`Edge [dest=Node [name=B, dateOB=2018-09-30, suburb=Ford]]`	Pass
2	Incorrect Edge – purposeful error	Non-existent edge n3	No node could be found	`null`	Pass

Test Harness Results:



```
------*****------ Task 1 begins ------*****-----
Expected: B,2018-10-30,Bonner   Actual: B,2018-10-30,Bonner     Status: Node test --> PASS
Expected: Node [name=B, dateOB=2018-09-30, suburb=Ford] Actual: Edge [dest=Node [name=B, dateOB=2018-09-30, suburb=Ford]]       Status: Edge test --> PASS
 ------*****------ Task 1 ends ------*****------
```

The Edge test was passed.

Task 2 – Hashing: overring hashCode() and equals() methods in Node Class

Hashing Algorithm approach

The main properties that were considered when creating the hashCode method included the following; deterministic (always have the same result), quick computation, small changes in input change hash value and collision resistant.

From the options presented in class extraction was not utilised as none of the elements have an identifying key that is always unique. Folding was attempted with the date, but it was soon identified that people could share the same birthday from our set of 1000.

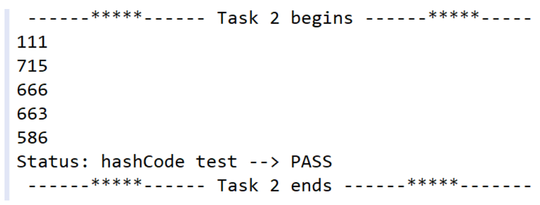
After some research, it was decided to combine certain methods to ensure that collision is avoided wherever possible. It was decided that the outcome of hashing the name and the adding the doB variable would form an effective hashCode with minimal collisions.

Both the name and suburb were hashed by using a for loop to cycle through all the letters of the name and change each value to ascii – before concatenating them all together to form a long variable. The date was hashed by picking positions 2, 4 and 6 (which helps remove any chance of collision with the same date) and adding them all together.

The hashCode then adds the long value (from the name) and the ascii value (from the date) together before finding the modulus with 1201. At the same time, the equals method was also overridden.

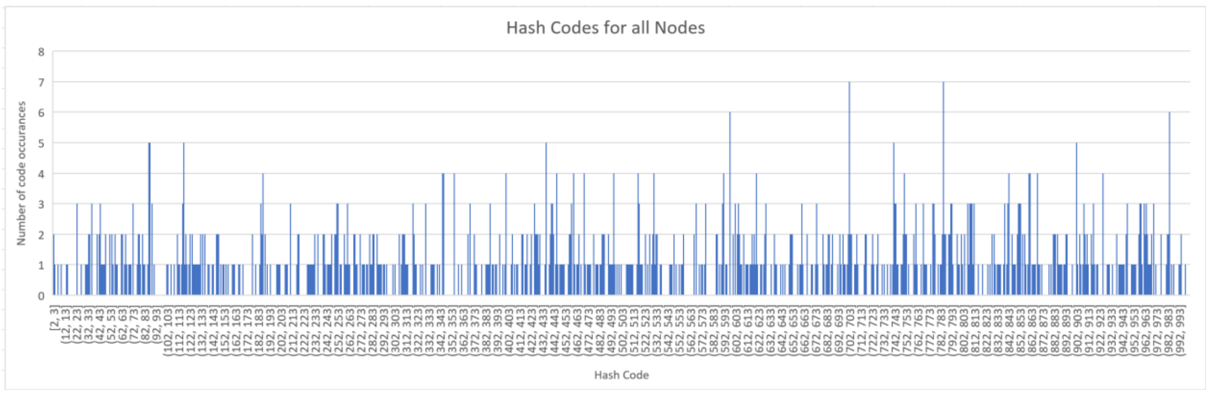
Testing the hashCode() method

5 nodes were trialled with different data sets which can be found in the TestHarness class. These nodes included symbols, numbers and a duplicate node to test the collision of my hashCode. The test passed as seen below:



```
 ------*****------ Task 2 begins ------*****-----
111
715
666
663
586
Status: hashCode test --> PASS
 ------*****------ Task 2 ends ------*****-------
```

The histogram below shows the distribution of my hash code for all nodes within the data set – showing that my hash code is reasonably evenly distributed (with the highest occurrence at 7 for a single hash).



Updated UML:

Node
- String name - LocalDate dateOB - String suburb protected HashMap<Node, Edge> adj
+ Node(String name, String date, String suburb) . . (same as above) . + long nameChange() + int dateChange() + int hashCode() + boolean equals(Object o)

Task 3 – Implementing Graph Class

UML Diagram

Graph
protected ArrayList<Node> nodeList = new ArrayList<>()
+ Graph() + Node addNode(String name, LocalDate dob, String suburb) + void addEdge(Node from, Node to) + void removeEdge(Node A, Node B) + void removeNode(Node node) + Set<Node> getNeighbors(Node node)

Test Cases

I tested each case sequentially (as suggested by the test harness).

Adding a node test: used the nodeList.size() function to check if nodes were stored correctly.
Adding an edge test: used the getNeighbors() method to check if links had been added to the nodes hashmap
Removing an edge test: used the getNeighbors() method to check if the link had been removed
Removing a node test: used the nodeList.contains() function to see if the node still exists
Finding neighbours test: used the .equals() method to check if one of the sets was empty and if another contained the correct amount of links

```
 ------*****------ Task 3 begins ------*****-----
Status: addNode test --> PASS
Status: addEdge test --> PASS
Status: removeEdge test --> PASS
Status: removeNode test --> PASS
Status: getNeighbors test --> PASS
 ------*****------ Task 3 ends ------*****-------
```

addNode testing –

Test ID	Descriptio n	Input	Expected Output	Actual Output	Comm ents
1	Three nodes are added	`v0 = g.addNode("V0", LocalDate.parse("2010-10-30"), "A");` `v1 = g.addNode("V1", LocalDate.parse("2010-10-30"), "B");` `v2 = g.addNode("V2", LocalDate.parse("2010-10-30"), "C");` `g.nodeList.size() == 3`	Node List size is equal to 3	`3` `Status: addNode test --> PASS`	Pass
2	No nodes added (Empty nodeList)	No nodes created. `System.out.println(g.nodeList.size());`	[] 0	`[]` `0` `Status: addNode test --> FAIL`	Pass

addEdge testing -

Test ID	Description	Input	Expected Output	Actual Output	Comm ents
1	2 edges added to v0	`g.addEdge(v0, v1);` `g.addEdge(v0, v2);` `System.out.println(g.getNeighbors(v0).size());`	2 edges	`2` `Status: addEdge test --> PASS`	Pass
2	No edges	No edges created for node v0. `System.out.println(g.getNeighbors(v0).size());`	0 edges.	`0` `Status: addEdge test --> FAIL`	Pass

removeEdge testing -

Test ID	Description	Input	Expected Output	Actual Output	Comm ents
1	Removing 1 edge from v0	`g.removeEdge(v0, v2);` `System.out.println(g.getNeighbors(v0).size());`	Only 1 edge left (based off edges added in addEdge testing)	`1` `Status: removeEdge test --> PASS`	Pass
2	Trying to remove a null node	`g.removeEdge(v0, v3);`	Test failed – null issue	`Issue: null` `Status: removeEdge test --> FAIL`	Pass

removeNode testing -

Test ID	Description	Input	Expected Output	Actual Output	Comm ents
1	Remove node v0 from graph	`g.removeNode(v0);` `System.out.println(g.nodeList.contains(v0));`	False – no node found	`false` `Status: removeNode test --> PASS`	Pass
2	Try to remove null node	`g.removeNode(v3);`	No node found - null issue	`Node does not exist` `Issue: null` `Status: removeNode test --> FAIL`	Pass

getNeighbors testing -

Test ID	Description	Input	Expected Output	Actual Output	Comm ents
1	Check for empty set and 1 neighbour	`Set<Node> setV = g.getNeighbors(v0);` `Set <Node> setV1 = g.getNeighbors(v1);` `Set<Node> s1 = Collections.emptySet();` `System.out.println(setV.equals(s1));` `System.out.println(setV1.size() == 1);`	Both statements should be true	`true` `true` `Status: getNeighbors test --> PASS`	Pass
2	getNeighbors of null node	`Set<Node> setV = g.getNeighbors(v3);`	No node – null issue	`Issue: null` `Status: getNeighbors test --> FAIL`	Pass

Big O Discussion

The search complexity for a hashtable is O(1) resulting in quick retrieve, store and delete data. When removing the Node (removeNode method) the entire nodeList is not iterated only the entry set also improving efficiency.

Issues:

I encountered an issue with my hashCode() method when attempting to removeEdge(). I originally had a random generator in my hashCode() that picked a position in the date and hashed that number – however, my removeEdge() would then clearly not work because the hashed position cannot be found. With a minor change made to pick a pre-determined position each time, the hashCode() worked and so did my removeEdge() method.

Speed Testing

When testing the speed of the removeNode() method it was discovered it was quicker to run through the entrySet(), creating an iterator and then remove the edge, rather than running through the nodeList and checking if the keySet() of the node contained a single node. A set of 5 tests was run with the average for an entrySet() iterator being 359619 nano seconds and the average for a nodeList for loop being 401744 nano seconds.

See code below for nodeList for loop attempt:

```java
public void removeNode2(Node node) {
        if (nodeList.contains(node)) {
                for (Node nodes : nodeList) {
                        if (node.adj.keySet().contains(nodes)) {
                                removeEdge(nodes, node);
                        }
                }
        }
        nodeList.remove(node);
    }
```

Task 4 – Implementing SocialNetwork Class

UML Diagram

SocialNetwork
+ Graph sn
+ SocialNetwork()
+ void processFile()
+ void addFriend(Node x, Node newFriend)
+ void removeFriend(Node x, Node newFriend)
+ List<Node> suggestFriends(Node currentPerson)
+ String getMutualFriends(Node x, Node y)

Selection of the class to read data.txt

A BufferedReader was chosen to wrap around a FileReader as it provides more efficient reading of characters, lines and arrays. The inbuilt readLine() method can be easily used to read a long line of text from the stream which can then be simply split into different sections to fill the Graph.

The processFile() has a big O notation of O(n) resulting in quick retrieving of data without contributing greatly to the overall runtime.

A scanner would be slower as it parses the input data whereas a BufferedReader simply reads the sequence of characters. As only a String is being read, the option of reading int, long or floats are not required (using a Scanner).

Test cases

suggestFriends test:

suggestFriends was tested with a small set of pre-determined nodes and edges so as to confirm the correct ouput.

Test ID	Description	Input	Expected Output	Actual Output	Comments
1	Test for two suggestedFriends	`v0 = driver.sn.addNode("V0", LocalDate.parse("2010-10-30"), "Belco");` `v1 = ….addNode("V1", LocalDate.parse("2010-10-30"), "Maj");` `v2 = ….addNode("V2", LocalDate.parse("2010-10-30"), "Belco");` `v3 = …addNode("V3", LocalDate.parse("2010-10-30"), "Gungahlin");` `v4 = …addNode("V4", LocalDate.parse("2010-10-30"), "Belco");` `….addEdge(v0, v1);` `….addEdge(v1, v2);` `….addEdge(v1, v3);` `….addEdge(v1, v4);` `friendsOffriends = driver.suggestFriends(v0);`	2 suggested friends	`2` `Status: suggestFriends test --> PASS`	Pass
2	Test for failed number of friends	`….addEdge(v0, v1);` `….addEdge(v1, v3);` `friendsOffriends = driver.suggestFriends(v0);`	Fail – does not equal 2 friends (my test case)	`0` `Status: suggestFriends test --> FAIL`	Pass
3	Testing for exception	v5 node = null `friendsOffriends = driver.suggestFriends(v5);`	Issue: null	`Issue: null` `Status: suggestFriends test --> FAIL`	Pass

acutalMutual test:

actualMutual was tested with a small set of pre-determined nodes and edges so as to confirm the correct output.

Test ID	Description	Input	Expected Output	Actual Output	Comments
1	Testing for a single mutual friend	`….addEdge(v0, v1);` `….addEdge(v0, v2);` `….addEdge(v1, v2);` `….addEdge(v1, v3);` `….addEdge(v1, v4);` `String actualMutual = driver.getMutualFriends(v0, v1);` `if (actualMutual.contains("V2")) PASS`	Node V2 to be mutual	`[Node [name=V2, dateOB=2010-10-30, suburb=Belco]] Status: actualMutual test --> PASS`	Pass
2	Testing for a fail with not V2 as a mutual friend	Removed addEdge(v0, v2) from test cases	Empty array []	`[] Status: actualMutual test --> FAIL`	Pass
3	Testing for exception	v5 node = null `actualMutual = driver.getMutualFriends(v0, v5);`	Issue: null	`Issue: null Status: actualMutual test --> FAIL`	Pass

Big O discussion

The bufferedReader choice for processFile() with a Big O notation of O(n) was discussed above. The Big O notation of suggestFriends is also O(n) (with the two for loops) and getMutualFriends has an efficiency of O(1).

Speed Test

The processFile() method was also implemented using a scanner in order to test efficiency (see code below). A test of five runs of both the buffered reader and scanner found an average of 1250 milliseconds for the BufferedReader compared to an average of 1450 for the Scanner.

The research in conjunction with the time results, prove that there is no requirement to change my code as this is an efficient way of processing a text file.

```
public void processFile2() throws NumberFormatException, Exception {
try {
    File file = new File("data.txt");
    Scanner scanReader = new Scanner(file);
    String[] values = new String[1000];
    int i = 0;
    while (scanReader.hasNextLine()){
        String line = scanReader.nextLine();
        values[i++] = line;
        String[] valueSplit = line.split("\t|\\,");
        sn.addNode(valueSplit[0], LocalDate.parse(valueSplit[1]), valueSplit[2]); }
    for(int j = 1; j < 1000; j++){
        Node n = sn.nodeList.get(j);
        String line = values[j];
        String[] valueSplit = line.split("\t|\\,");
        for (int k = 3; k < valueSplit.length; k++){
         valueSplit[k] = valueSplit[k].replaceAll(" ", "");
            sn.addEdge(n, sn.nodeList.get(Integer.parseInt(valueSplit[k]) - 1));}
    } scanReader.close();}
catch(FileNotFoundException e){ e.printStackTrace(); }}
```

Task 5 – Implementing Birthday Reminder

Update UML Design:

Node
- String name - LocalDate dateOB - String suburb protected HashMap<Node, Edge> adj
+ Node(String name, String date, String suburb) . . (same as above) . + long daystilBirthday(Node n) + int compareTo(Node o)

SocialNetwork
+ Graph sn
+ SocialNetwork() . . (same as above) + String remindBDEvents(Node currentPerson)

<u>Justification of design</u>

Two methods were incorporated into the Node class and one method into the SocialNetwork class (as per assignment direction) to achieve this task.

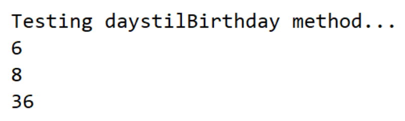
I needed to find a way to weight each Node within the currentPerson hashmap in order to effectively remove (poll) them from the Priority Queue. I did this by creating a new method within Node class called daystilBirthday with accepts a Node object and returns the days from "today" until that date i.e how many days until that day as a long. This method was then utilised when overriding the compareTo method as the you can compare how far away two dates are using an integer result.

The remindBDEvents method was designed to return how far away all currentPerson's friends' birthdays were. This was achieved by adding all the friends of currentPerson to a Priority Queue and then polling each node individually then adding the details of how far away the birthday is (utilising the Period.between(a, b) function) to append all the results to a Stringbuilder which is then returned.

<u>Test Cases (for each method)</u>

daystilBirthday Test:

Three test nodes used with dateOB's that were easy to calculate how far away they were from today's date e.g. 10/06/2000, 12/06/2000 and 10/07/1998. This also confirmed that my code to overwrite the year to either the current year or the following year was working correctly.

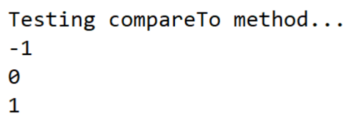


```
Testing daystilBirthday method...
6
8
36
```

(screenshot from test performed on 04/06/2019).

compareTo Test:

The same three test nodes were used to test the compareTo method. It was run so that all three result should be checked including before, equals and after.

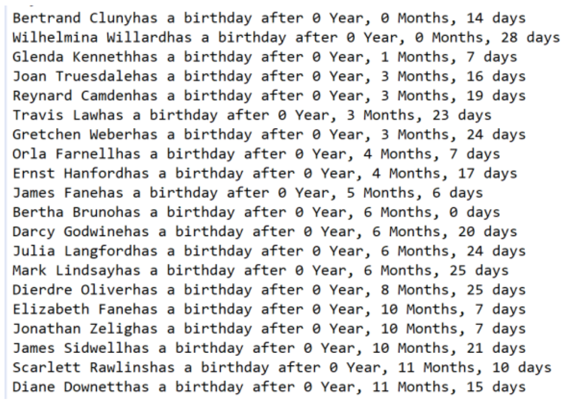


```
Testing compareTo method...
-1
0
1
```

(screenshot from test performed on 04/06/2019).

remindBDEvents Test:

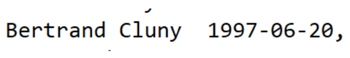
I struggled to work out a way to test the method most efficiently (without doing any manual calculations). However, from evaluation – we can see that people's birthdays are in order from most recent to furthest away. The method also tests if all the friends are being printed as it counts the number of friends (using getNeighbors) and the number of lines printed (numLines). If they are the same number, we know all the friends have been printed e.g. 20 20.



```
Bertrand Clunyhas a birthday after 0 Year, 0 Months, 14 days
Wilhelmina Willardhas a birthday after 0 Year, 0 Months, 28 days
Glenda Kennethhas a birthday after 0 Year, 1 Months, 7 days
Joan Truesdalehas a birthday after 0 Year, 3 Months, 16 days
Reynard Camdenhas a birthday after 0 Year, 3 Months, 19 days
Travis Lawhas a birthday after 0 Year, 3 Months, 23 days
Gretchen Weberhas a birthday after 0 Year, 3 Months, 24 days
Orla Farnellhas a birthday after 0 Year, 4 Months, 7 days
Ernst Hanfordhas a birthday after 0 Year, 4 Months, 17 days
James Fanehas a birthday after 0 Year, 5 Months, 6 days
Bertha Brunohas a birthday after 0 Year, 6 Months, 0 days
Darcy Godwinehas a birthday after 0 Year, 6 Months, 20 days
Julia Langfordhas a birthday after 0 Year, 6 Months, 24 days
Mark Lindsayhas a birthday after 0 Year, 6 Months, 25 days
Dierdre Oliverhas a birthday after 0 Year, 8 Months, 25 days
Elizabeth Fanehas a birthday after 0 Year, 10 Months, 7 days
Jonathan Zelighas a birthday after 0 Year, 10 Months, 7 days
James Sidwellhas a birthday after 0 Year, 10 Months, 21 days
Scarlett Rawlinshas a birthday after 0 Year, 11 Months, 10 days
Diane Downetthas a birthday after 0 Year, 11 Months, 15 days
```

(screenshot from test performed on 06/06/2019).

By manually searching the data.txt file however we can find that Bertrand has a birthday on the 20th of June – which is in 14 days from when the test was conducted. This proves that the remindBDEvents method is working.



```
Bertrand Cluny  1997-06-20,
```

<u>Big O Discussion</u>

From the javaDoc file on Priority Queue's it is discussed that my implementation of the priority queue would provide an O(log(n)) time for the enqueing and dequeing methods (offer, poll, remove() and add). This is highly efficient.

Task 6 – Reflection

I would improve the Social Network application by incorporating a removeAllFriends() method that allows a 'node' to delete all its friends without deleting itself from the Social Network. I would also include an ID that is assigned to that user when they sign up to the Social Network (up to the total number of users – exponentially growing for a real application) that would make accessing the primary key simpler and organising nodes (because it could then be based off when the user signed up to the Social Network).

Continuous testing helped me the most when completing the assignment. General testing cannot be underestimated as it so useful to ensure minor errors are caught from the start and debugged before reaching further tasks and being unable to discover where the issue lies. By designing tests, it often easier to understand what the question is asking you to code and what exceptions must be thrown etc. Designing the tests sometimes led me to redesign my code, but often it was during the test process that bugs were discovered, as the tests detected these errors and they were subsequently corrected.

Overall, I was held back by lack of understanding in hashmap manipulation and my understanding of the questions. It took me a considerable amount of time to understand what each question was asking individually and then what was expected of the entire assignment. Once that was confirmed however, the actual logic application of the assessment was challenging but achievable.