Gebze Technical University
Computer Engineering

CSE 222
2017 Spring

HOMEWORK 3 REPORT

VAKHID BETRAKHMADOV
141044086

Course Assistant: Nur Banu Albayrak

1. System Requirements

JDK 1.8+ installed.

2. Class Diagrams

My Collection hierarchy .pdf

3. Problem Solutions Approach

Question 1:
In order to test and compare performances of all toString methods (toString1, toString2, toString3) I used System class's static method currentTimeMillis(), applying simple formula "startTime – endTime/1000 " in order to get time elapsed in seconds. I measured time it took for 3 different methods to finish 5 times and got following results:

Run1:
toString1: 89.222 s.
toString2: 43.383 s.
toString3: 41.8 s.

Run2:
toString1: 96.529 s.
toString2: 59.376 s.
toString3: 65.777 s.

Run3:
toString1: 85.82 s.
toString2: 46.754 s.
toString3: 43.281 s.

Run4:
toString1: 62.739 s.
toString2: 34.823 s.
toString3: 34.423 s.

Run5:
toString1: 70.524 s.
toString2: 48.901 s.
toString3: 35.869 s.

Run5:
toString1: 59.698 s.
toString2: 36.303 s.
toString3: 36.14 s.

Runtime for all methods differs from run to run. This is quite normal, and might be explained by business of the system at different timepoints, but the common pattern is obvious.
As we can clearly see toString1 method is much slower, then the other two. This is what expected, because toString1 runtime is O(n^2) and toString2 and toString3 are both O(n).
Method toString1 uses indexes and SingleLinkedList's get method. In order to get all the elements in the list we need to traverse the whole list (starting from 0 index and finishing at size()-1 index)

and call get method for each index. Get method itself has O(n) runtime, because it always starts at the first element of the list, and traverses it until it gets to the element with the passed index.
So, all this in total makes runtime of method toString1 O(n^2).
As for the toString2 and toString3, both methods work using iterators (for toString3 method the underlying SingleLinkedList's toString method also uses iterator), which traverse the whole list only once, so runtime for both methods is O(n). Note. Small oscillation is due to the system imperfections.

Question 3:
We have extended AbstractCollection<T> class in MyAbstractCollection<T> and implemented appendAnything method which is actually the same as add(Collection<? extends T> method in Collection<T> interface.
Our appendAnything method, takes one argument of type AbstactCollection<T>, so that we are able to add any collection in Java (type Collection<T> would be better, but …). Any concrete collection extending AbstactCollection<T> must have iterator and size methods implemented and add method overridden, assuming that we can easily iterate through collections and add. Thus, we are able to concatenate two different collections.
I extended MyAbstractCollection<T> class in SingleLinkedList<T> class and tested appending LinkedList<T>, ArrayList<T> and ArrayDeque<T>.

4. Test Cases

Question 1:
The one given in pdf.

Question 3:
All java collections.

5. Running and Results

Question 2:

```
[]
[1]
[2,1]
[3,2,1]
[4,3,2,1]
[5,4,3,2,1]
mans@mans-VirtualBox:~/Desktop/Data Structures/141044086_hw3/Question2$
```

Question 3:

```
a b c d e f g h j m l k
mans@mans-VirtualBox:~/Desktop/Data Structures/141044086_hw3/Question3$
```

Question 4:

```
singleLinkedList before insertion:
size: 0

deletedToString before insertion:
[]
singleLinkedList after 100 were ints inserted:
size: 100
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 8
3 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
deletedToString after 100 were ints inserted:
[]
singleLinkedList after 50 ints were removed:
size: 50
50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76
 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
deletedToString after 50 ints were removed:
[49, 48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38, 37, 36, 35, 34, 33, 32, 31, 30,
 29, 28, 27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10,
 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
singleLinkedList after 100 more ints were inserted:
size: 150
50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76
 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 10
2 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 12
2 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 14
2 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 16
2 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 18
2 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199
deletedToString after 100 more ints were inserted:
[]
mans@mans-VirtualBox:~/Desktop/Data Structures/141044086_hw3/Question4$
```