Assignment 7

NOTE: exercises with prefix * are not mandatory**

Topics: Exception Handling, Collections, Iterators, and Recursion

1. *** Read about collections at: *http://docs.oracle.com/javase/tutorial/collections/index.html*

2. A linked list is a sequence of objects connected by reference links. Usually, to access a linked list one needs the reference to its first node. Each subsequence element in the list is therefore accessed via the link stored in the previous node. By convention the link reference to the last node in the list is set to null (see Figure 1). New objects are stored in the list in a dynamic way i.e. the application has to create a node and insert it in the list while the other existing list elements are not moved (only the reference links are changed!).
   Your task is to implement your own linked list (you may not use the *LinkedList* class!). Your linked list must hold integer values. Any node in the data structure can be reached by starting at root node and repeatedly following references to the next node.

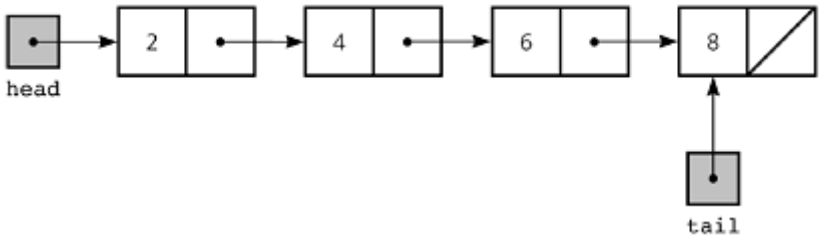

Figure 1: example of a linked list of integer values

   Design and implement a class *MyIntegerLinkedList* that stores integer numbers. Your implementation should have (at least) the following methods:

   void insertHead(int element) → it inserts an element in front of the list
   void insertTail(int element) → it inserts an element at the end of the list
   void removeHead() → it removes the first element from the list
   void removeTail() → it removes the last element from the list
   void printLinkedList() → it prints out all the elements of the list

3. Write your own exception class *MyIntegerLinkedListException* that deals with exceptions thrown when there is an attempt of removing either the first element or the last element in an empty list. Notice that you also have to reflect the changes to accommodate to such kind of exceptions in the definition and implementation of the following two methods in question 2:
   void removeHead()
   void removeTail()

4. Implement an *Iterator* that traverses your class *MyIntegerLinkedList*; you are free to decide how traversing occurs but obviously you need to implement the methods defined in the interface *Iterator*

5. ***Write a *Main* class where you test your implementation of the linked list. More specifically:

a. create a linked list
b. add a few elements to the list and then print out its elements
c. remove the last element from the list and then print out the list
d. remove all elements from the list and then print out the list
e. remove one more element from the list (the list should be empty!)
f. add a few elements to the list and print them out

6. Implement a *Labyrinth* class and a recursive method that finds the path from the entrance to the exit of the labyrinth. You are free to either create a method that builds a random labyrinth (this is likely going to be one of the constructors) or to hardcode the topology of the labyrinth (in this latter case, you are likely to use a 2D array initialization inside the class). Implement also a *displayLabyrinth()* method that displays the labyrinth. Before searching for an exit, call the *displayLabyrinth()* to get a view of the initial setting, which could be something like:

```
                00000000
                01111100
                00100100
Entrance →      11100100
                01100111  ←Exit
                00000000
```

0s indicate locations where you cannot go to. 1s indicate accessible locations.
Once your method has found the way out, you should display again the labyrinth with the path highlighted with e.g. X characters like

```
                00000000
                01xxxx 00
                00x00x00
Entrance →      xxx00x00
                01100xxx  ←Exit
                00000000
```

Suggestion: make use of the *enum* type to keep track of which location is taken and which is free and other possible state of each single location in the labyrinth