/\*\* Chapter No. 15 - Exercise No. 1

File Name: Suitor.java

Programmer: Broderick Higby

Date Last Modified: November 17, 2015

\*Algorithm:

- Create a linked list class

- Create instance variables: head of type root, node, count of type int, & number of type node

- Use next and previous to reference other nodes

- default constructors

- Linked List class will run through various methods, find, contains, isEmpty

addToStart, Delete, outputList, clear, and getWinner

- When class gets to beginning it'll set the head to null

Suitor:

- Prompt for number of suitors waiting to marry - numOfSuitors

- Create nodes of Linked List based on numOfSuitors

- Run getWinner(number of suitors) to find winning position - posToMarry

- Output winning position

\*/

import java.util.Scanner;

public class Suitor

{

public static void main(String[] args)

{

LinkedList list = new LinkedList();

Scanner keyboard = new Scanner(System.in);

int numOfSuitors, posToMarry;

//INPUT - Get current number of suitors

System.out.print("Please enter the number of suitors: ");

numOfSuitors = keyboard.nextInt();

//PROCESSING - Add nodes for calculation

for(int i = numOfSuitors; i > 0; i--)

list.addToStart(i);

posToMarry = list.getWinner(numOfSuitors);

//OUTPUT

System.out.println();

System.out.println("Stand in position "+ posToMarry + " to marry the "+

"beautiful princess.\n\n");

}

}

public class LinkedList

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//NODE INNER CLASS

private class Node

{

private int suitor;

private Node link;

public Node()

{

suitor = 0;

link = null;

}

public Node(int newSuitor, Node linkValue)

{

suitor = newSuitor;

link = linkValue;

}

}//END OF INNER CLASS

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

private Node head;

public LinkedList()

{

head = null;

}

public void addToStart(int newSuitor)

{

head = new Node(newSuitor, head);

}

public boolean deleteHeadNode()

{

if (head!=null)

{

head = head.link;

return true;

}

else

return false;

}

public int size()

{

int count = 0;

Node position = head;

while (position != null)

{

count++;

position = position.link;

}

return count;

}

public boolean contains(int num)

{

return (find(num) != null);

}

private Node find(int target)

{

Node position = head;

int suitorAtPosition;

while(position!=null)

{

suitorAtPosition = position.suitor;

if(suitorAtPosition==target)

return position;

position = position.link;

}

return null;

}

public void outputList()

{

Node position = head;

while (position != null)

{

System.out.println(position.suitor);

position = position.link;

}

}

public boolean isEmpty()

{

return (head == null);

}

public void clear()

{

head = null;

}

public int getWinner(int n)

{

Node current = head;

Node previous = null;

while(n != 1)//Runs until winner remaining

{

for(int i = 1; i < 3; i++)//Moves to third suitor

{

//sets previous to current reference value

previous = current;

//sets current to link value (reference) of next node

current = current.link;

//Returns to beginning if end of list reached

if(current == null)

{

current = head;

}

}

//Removed current reference value from list

previous.link = current.link;

current = current.link;

//Returns to beginning if end of list reached

if(current == null)

{

current = head;

}

n--;

}

return current.suitor;

}

}

