C:\Users\Rich\Documents\NetBeansProjects\Lab07\src\LuckyNumberList.java

```
1
2 import java.util.NoSuchElementException;
3
4 /**
5  *
6  * @author Richelin
7  * @version 03/03/2017
8  */
9 public class LuckyNumberList {
10
11    private LinkedPositionalList<LuckyNumber>  list;   // doubly linked list that will contain objects of LuckyNumber with their positions
12
13    public LuckyNumberList()
14    {
15       list = new LinkedPositionalList();
16    }
17    /**
18     *
19     * @param ln object to be added as last in list
20     */
21    public void addLuckyNumber(LuckyNumber ln)
22    {
23       list.addLast(ln);
24    }
25
26    private class PositionIterator implements Iterator<Position<LuckyNumber>>
27    {
28       private Position<LuckyNumber> cursor = list.first();
29       private Position<LuckyNumber> recent  = null;
30
31       public boolean hasNext()
32       {
33          return (cursor != null);
34       }
35
36       public Position<LuckyNumber> next() throws NoSuchElementException
37       {
38          if(cursor == null) throw new NoSuchElementException(" No More Position");
39
40          recent = cursor;
41          cursor = list.after(cursor);
42          return recent;
43       }
44       /** remove the elements retruned by most recent call to next. */
45       public void remove() throws IllegalStateException
46       {
47          if ( recent == null) throw new IllegalStateException (" Nothing to remove");
48          list.remove(recent);
49          recent = null;
50       }
51
52    }
53    //----------------PositionIterator End----------
54
55
56    // **************Nested PositionIterable Class******
57    private class PositionIterable implements Iterable<Position<LuckyNumber>>
58    {
59       public Iterator<Position<LuckyNumber>> iterator( ){return new PositionIterator(); }
60    }
61    // *************End of Nested PostionIterableClass***************
62
63    /**
64     *
65     * @return an iterable representation of the list's positions.
66     */
67    public Iterable<Position<LuckyNumber>> positions()
68    {
69       return new PositionIterable();       // after this execute, think of cursor = position/address of first luckyNumber object.
70    }
71
72 /* * * * *Customed Iterator for position of Even number ********************/
73
74    //----------------Nested PositionIterator class ----------
75    private class EvenPositionIterator implements Iterator<Position<LuckyNumber>>
76    {
77       private Position<LuckyNumber> cursor = list.first();    // position of first element to report
78       private Position<LuckyNumber> recent = null;            // position of last reported element
79
80       public boolean hasNext() { return (cursor != null); }
81
82       public Position<LuckyNumber> next() throws NoSuchElementException
83       {
84          if(recent == null)         // if at begining position of the list
85          {
86             while(cursor != null&& !isEven(cursor.getElement().getLuckyNumber()))
87                cursor = list.after(cursor);       // R. advance cursor till you find the first position of an object whose element is even
88                                   // R. take care of first call. think of a list had only one object, recent will be null
89          }
90
91          if (cursor == null) throw new NoSuchElementException("Nothing left to see here"); // if at end position of the list
92
```

```
93        /* if not at begining or end of the list , let's say at the a list of two objects(2positions) cursor is pointing at object 2, recent is position1
94          or point at object1.
95          */
96          recent = cursor;              // (if in our example of two objects, cursor will point to object2(position2) of list
97          cursor = list.after(cursor);  // advance cursor one more step (if our example of two objects, cursor will point to trailer(null) of list
98
99          /* now need to advance to next position/node whose element is an even number */
100
101         while (cursor !=null && !isEven(cursor.getElement().getLuckyNumber())) //cursor.getElement() return a node,which is the address of the current object the cursor (a position) is pointing to
102            cursor = list.after(cursor);
103
104         return recent;
105      }
106
107      public void remove() throws IllegalStateException
108      {
109         if (recent == null) throw new IllegalStateException("Nothing to remove");
110         list.remove(recent);    // remove from outer list linKedPositional List
111         recent = null;          // do not allow remove again until next is called
112      }
113   }
114   /************************ End of nested EvenPositionIterator  ***************/
115
116      // ----------nested Class ----------
117   private class EvenPositionIterable implements Iterable<Position<LuckyNumber>>
118   {
119      public Iterator<Position<LuckyNumber>> iterator() { return new EvenPositionIterator();}
120   } //------ end of EvenPositionIterable----------------
121
122   public Iterable<Position<LuckyNumber>> EvenPositions(){
123      return new EvenPositionIterable();
124   }
125
126  /******************** End of All needed for even Iterator********************/
127
128   private class PrimePositionIterator implements Iterator<Position<LuckyNumber>>{
129      private Position<LuckyNumber> cursor = list.first();   // position of the next element to report
130      private Position<LuckyNumber> recent = null;              // position of last reported element
131      /** Tests whether the iterator has a next object. */
132      @Override
133      public boolean hasNext( ) { return ( cursor != null ); }
134      /** Returns the next position in the iterator. */
135      @Override
136      public Position<LuckyNumber> next( ) throws NoSuchElementException {
137         // On the first call to next (i.e. when recent == null) you need to      //<<< new code
138         // advance recent until it is pointing to a vowel elemet.              //<<< new code
139         if ( recent == null )                                          //<<< new code
140         {                                                          //<<< new code
141            while ( cursor != null && !isPrime( cursor.getElement().getLuckyNumber()) )    //<<< new code
142               cursor = list.after( cursor );                          //<<< new code
143         }                                                          //<<< new code
144
145         if ( cursor == null ) throw new NoSuchElementException( "nothing left " );
146         recent = cursor;
147         cursor = list.after( cursor );
148
149         // advance cursor to the next vowel
150
151         while ( cursor != null && !isPrime( cursor.getElement().getLuckyNumber()) )
152            cursor = list.after( cursor );
153
154         return recent;
155      }
156      /** Removes the element returned by most recent call to next. */
157      @Override
158      public void remove( ) throws IllegalStateException {
159         if ( recent == null ) throw new IllegalStateException( "nothing to remove" );
160         list.remove( recent );        // remove from outer list
161         recent = null;             // do not allow remove again until next is called
162      }
163   } //----- end of nested PositionIterator class -----
164
165   //----- nested PositionIterable class -----
166   private class PrimePositionIterable implements Iterable<Position<LuckyNumber>>{
167      @Override
168      public Iterator<Position<LuckyNumber>> iterator( ) { return new PrimePositionIterator( ); }
169   } //----- end of nested PositionIterable class -----
170
171   /** Returns an iterable representation of the list's positions.
172    * @return  */
173   public Iterable<Position<LuckyNumber>> primePositions( ) {
174      return new PrimePositionIterable( );  // create a new instace of the inner class
175   }
176
177
178
179   public boolean isEven(int num)
180   {
181      return (num % 2 == 0);
182   }
183   public boolean isPrime(int num)
184   {
185      if(num == 2 ||num == 3 ){return true;}
186
187      int maxRange = (int) Math.sqrt(num);
```

```
188         for(int i = 2;  i <= maxRange; ++i)
189             if(num % i ==  0)
190                 return false;
191             else if (i == maxRange)
192                 return true;
193
194         return false; // to prevent compiler whining and take care if 1 is passed as argument.
195     }
196 }
```