**Client**

/\*\*

\*

\* @author Marie Larson

\*@version 3/6/2018

\*/

public class Client {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) {

System.out.println("Begin Lab107");

String names[] = {"Bobbie", "Fred", "Karen", "Linda", "Gale", "Rick", "Sharon", "Frankie", "Vance", "Lori"};

LuckyNumberList list = new LuckyNumberList();

for(int i=0;i<10;i++){

list.addLuckyNumber(new LuckyNumberList(names[i]));

}

Iterator<Position<LuckyNumber>> ListIterator = list.positions().Iterator();

System.out.print("Using ListIterator in lucky number list = ");

while(ListIterator.hasNext()){

LuckyNumber temp = ListIterator.next().getElement();

String even = "odd";

if(temp.getLuckyNumber() % 2 == 0)

even = "even";

String prime = "not prime";

int n = temp.getLuckyNumber();

if(n==2 || n==3 || n==5 || n==7)

prime = "prime";

System.out.printf("\n%10s %6d %10s %10s", temp.getName(), temp.getLuckyNumber(), even, prime);

}

System.out.print("\n\n");

Iterator<Position<LuckyNumber>> primeIterator = list.primePositions().iterator();

System.out.print("Using Prime iterator in lucky number list = ");

while(primeIterator.hasNext()){

LuckyNumber temp = primeIterator.next().getElement();

String even = "odd";

if(temp.getLuckyNumber() % 2 == 0)

even = "even";

String prime = "not prime";

int n = temp.getLuckyNumber();

if(n==2 || n==3 || n==5 || n==7)

prime = "prime";

System.out.printf("\n%10s %6d %10s %10s", temp.getName(), temp.getLuckyNumber(), even, prime);

}

System.out.print("\n\n");

Iterator<Position<LuckyNumber>> evenIterator = list.EvenPosition().iterator();

System.out.print("Using Even iterator in lucky number list = ");

while(evenIterator.hasNext()){

LuckyNumber temp = evenIterator.next().getElement();

String even = "odd";

if(temp.getLuckyNumber() % 2 == 0)

even = "even";

String prime = "not prime";

int n = temp.getLuckyNumber();

if(n==2 || n==3 || n==5 || n==7)

prime = "prime";

System.out.printf("\n%10s %6d %10s %10s", temp.getName(), temp.getLuckyNumber(), even, prime);

}

System.out.print("\n\n"

+ "End Lab107");

}

}

**LuckyNumber**

import java.util.Random;

/\*\*

\*

\* @author marie

\*/

public class LuckyNumber {

private String name;

private int luckyNumber;

public LuckyNumber(String name){

this.name = name;

Random random = new Random();

this.luckyNumber = random.nextInt(9);

}

public String getName(){

return name;

}

public int getLuckyNumber(){

return luckyNumber;

}

@Override

public boolean equals(Object o){

if(!(o instanceof LuckyNumber)){

return false;

}

LuckyNumber IN = (LuckyNumber) o;

return IN.luckyNumber == this.luckyNumber && name.equals(IN.name);

}

@Override

public String toString(){

return getClass().getName() + "@ " + name + ":" + luckyNumber;

}

}

**LuckyNumberList**

import java.util.NoSuchElementException;

/\*\*

\*

\* @author marie

\*/

public class LuckyNumberList {

private LinkedPositionalList<LuckyNumber> luckyList = null;

public LuckyNumberList(){

this.luckyList = new LinkedPositionalList<>();

}

LuckyNumberList(String name) {

throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

LuckyNumberList(String name) {

throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

public void addLuckyNumber(LuckyNumber num){

luckyList.addLast(num);

}

Object positions() {

throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

private class PositionIterator extends Iterator implements Iterator<Position<LuckyNumber>>{

private Position<LuckyNumber> cursor = luckyList.first();

private Position<LuckyNumber> recent = null;

@Override

public boolean hasNext(){

return (cursor != null);

}

@Override

public Position<LuckyNumber> next() throws NoSuchElementException{

if(cursor == null) throw new NoSuchElementException("There isn't anything left");

recent = cursor;

cursor = luckyList.after(cursor);

return recent;

}

@Override

public void remove() throws IllegalStateException{

if(recent == null) {

throw new IllegalStateException("There is nothing to remove");

}

luckyList.remove(recent);

recent = null;

}

}

private class PositionIterable extends Iterable implements Iterable<Position<LuckyNumber>>{

@Override

public Iterator<Position<LuckyNumber>> iterator(){

return new PositionIterator();

}

public Iterable<Position<LuckyNumber>> positions(){

return new PositionIterable();

}

private class EvenPositionIterator extends Iterator implements Iterator<Position<LuckyNumber>>{

private Position<LuckyNumber> cursor = luckyList.first();

private Position<LuckyNumber> recent = null;

@Override

public boolean hasNext(){

return (cursor != null);

}

@Override

public Position<LuckyNumber> next() throws NoSuchElementException {

if(recent == null){

while (cursor != null && !isEven(cursor.getElement().getLuckyNumber()))

cursor = luckyList.after(cursor);

}

if(cursor == null) {

throw new NoSuchElementException("no more left");

}

recent = cursor;

cursor = luckyList.after( cursor );

while(cursor != null && !isEven(cursor.getElement().getLuckyNumber()))

cursor = luckyList.after(cursor);

return recent;

}

@Override

public void remove() throws IllegalStateException{

if(recent == null){

throw new IllegalStateException("nothing left");

}

luckyList.remove(recent);

recent = null;

}

private boolean isEven(int luckyNumber) {

throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

}

private class EvenpositionIterable extends Iterable implements Iterable<Position<LuckyNumber>>{

@Override

public Iterator<Position<LuckyNumber>> iterator(){

return new EvenPositionIterator();

}

}

public Iterable<Position<LuckyNumber>> EvenPosition(){

return new EvenpositionIterable();

}

public boolean isPrime(int n){

return n == 2 || n == 3 || n==5 || n==7;

}

private class PrimePositionIterator extends Iterator implements Iterator<Position<LuckyNumber>>{

private Position<LuckyNumber> cursor = luckyList.first();

private Position<LuckyNumber> recent = null;

@Override

public boolean hasNext(){

return (cursor != null);

}

@Override

public Position<LuckyNumber> next() throws NoSuchElementException{

if(recent == null){

while(cursor != null && !isPrime(cursor.getElement().getLuckyNumber()))

cursor = luckyList.after(cursor);

}

if(cursor == null){

throw new NoSuchElementException("There isn't anything left");

}

recent = cursor;

cursor = luckyList.after(cursor);

while(cursor != null && !isPrime(cursor.getElement().getLuckyNumber()))

cursor = luckyList.after(cursor);

return recent;

}

@Override

public void remove() throws IllegalStateException{

if(recent == null) throw new IllegalStateException("There is nothing to remove");

luckyList.remove(recent);

recent = null;

}

}

private class PrimeIterable extends Iterable implements Iterable<Position<LuckyNumber>>{

@Override

public Iterator<Position<LuckyNumber>> iterator(){

return new PrimePositionIterator();

}

}

public Iterable<Position<LuckyNumber>> primePosition(){

return new PrimeIterable();

}

@Override

public String toString(){

String returnString = "";

Iterator<LuckyNumber> listIterator = luckyList.iterator();

while(listIterator.hasNext())

returnString += listIterator.next() + "";

return returnString;

}

}

}

**Position**

/\*\*

\*

\* @author marie

\* @param <E>

\*/

public class Position<E> {

/\*\*

\*

\* @return

\* @throws IllegalStateException

\*/

E getElement() throws IllegalStateException;

}

**PositionalList**

/\*\*

\*

\* @author marie

\*/

public interface PositionalList<E> {

int size();

boolean isEmpty();

Position<E> first();

Position<E> last();

Position<E> before(Position<E> p) throws IllegalArgumentException;

Position<E> after(Position<E> p) throws IllegalArgumentException;

Position<E> addFirst(E e);

Position<E> addLast(E e);

Position<E> addBefore(Position<E> p, E e) throws IllegalArgumentException;

Position<E> addAfter(Position<E> p, E e) throws IllegalArgumentException;

E set (Position<E> p, E e) throws IllegalArgumentException;

E remove(Position<E> p) throws IllegalArgumentException;

}

**LinkedPositionalList**

import java.util.NoSuchElementException;

/\*\*

\*

\* @author marie

\* @param <E>

\*/

public class LinkedPositionalList<E> extends Position implements Position<E> {

//---------------------Nested Node Class-------------------------

private static class Node<E> extends Position implements Position<E>{

private E element;

private Node<E> prev;

private Node<E> next;

public Node(E e, Node<E> p, Node<E> n){

element = e;

prev = p;

next = n;

}

@Override

public E getElement() throws IllegalStateException{

if(next == null)

throw new IllegalStateException("Position invalid");

return element;

}

public Node<E> getPrev(){

return prev;

}

public Node<E> getNext(){

return next;

}

public void setElement(E e){

element = e;

}

public void setPrev(Node<E> p){

prev = p;

}

public void setNext(Node<E> n){

next = n;

}

}

//----------------------End Nested Node Class----------------------

//--------------------Begin Nested PositionIterator class--------------------

private class PositionIterator extends Iterator implements Iterator<Position<E>>{

private Position<E> cursor = first();

private Position<E> recent = null;

@Override

public Position<E> hasNext(){

return (cursor != null);

}

@Override

public Position<E> next() throws NoSuchElementException{

if(cursor == null) throw new NoSuchElementException("There isn't anything else");

LinkedPositionalList.this.remove(recent);

recent = null;

}

}

//-------------------End Nested PositionIterator class-----------------

//--------------------Nested PositionIterable Class-----------------------

private class PositionIterable extends Iterable implements Iterable<Position<E>>{

@Override

public Iterator<Position<E>> iterator() {

return new PositionIterator();

}

}

//---------------------End Nested PositionIterable Class----------------------

public Iterable<Position<E>> positions(){

return new PositionIterable();

}

//-------------------Begin Nested ElementIterator class-------------------

private class ElementIterator extends Iterator implements Iterator<E>{

Iterator<Position<E>> pIterator = new PositionIterator();

@Override

public boolean hasNext(){

return pIterator.hasNext();

}

@Override

public E next(){

return pIterator.next().getElement();

}

@Override

public void remove(){

pIterator.remove();

}

}

public Iterator<E> iterator(){

return new ElementIterator();

}

private Node<E> head;

private Node<E> tail;

private int size = 0;

public LinkedPositionalList(){

head = new Node<> (null,null,null);

tail = new Node<> (null, head, null);

head.setNext(tail);

}

private Node<E> validate(Position<E> p) throws IllegalArgumentException{

if(!(p instanceof Node)) throw new IllegalArgumentException("p invalid");

Node<E> node = (Node<E>) p;

if(node.getNext() == null)

throw new IllegalArgumentException("p isn't in the list anymore");

return node;

}

private Position<E> position(Node<E> node){

if(node == head || node == tail)

return null;

return node;

}

public int size(){

return size;

}

public boolean isEmpty(){

return (size == 0);

}

public Position<E> first(){

return position(head.getNext());

}

public Position<E> last(){

return position(tail.getPrev());

}

public Position<E> before(Position<E> p) throws IllegalArgumentException{

Node<E> node = validate(p);

return position(node.getPrev());

}

public Position<E> after(Position<E> p) throws IllegalArgumentException{

Node<E> node = validate(p);

return position(node.getNext());

}

private Position<E> addBetween(E e, Node<E> pred, Node<E> var){

Node<E> newest = new Node<>(e, pred, var);

pred.setNext(newest);

var.setPrev(newest);

size++;

return newest;

}

public Position<E> addFirst(E e){

return addBetween(e, head, head.getNext());

}

public Position<E> addLast(E e){

return addBetween(e, tail.getPrev(), tail);

}

public Position<E> addBefore(Position<E> p, E e) throws IllegalArgumentException{

Node<E> node = validate(p);

return addBetween(e, node.getPrev(), node);

}

public Position<E> addAfter(Position<E> p, E e) throws IllegalArgumentException{

Node<E> node = validate(p);

return addBetween(e, node, node.getNext());

}

public E set(Position<E> p, E e) throws IllegalArgumentException{

Node<E> node = validate(p);

E answer = node.getElement();

node.setElement(e);

return answer;

}

public E remove(Position<E> p) throws IllegalArgumentException{

Node<E> node = validate(p);

Node<E> predecessor = node.getPrev();

Node<E> successor = node.getNext();

predecessor.setNext(successor);

successor.setPrev(predecessor);

size--;

E answer = node.getElement();

node.setElement(null);

node.setNext(null);

node.setPrev(null);

return answer;

}

}

**Iterator**

/\*\*

\*

\* @author marie

\* @param <E>

\*/

public class Iterator<E> {

boolean hasNext();

E next();

void remove() throws IllegalStateException;

}

**Iterable**

/\*\*

\*

\* @author marie

\* @param <E>

\*/

public class Iterable<E> {

Iterator<E> iterator();

}

**Output:**

run:

Begin Lab107

Exception in thread "main" java.lang.ExceptionInInitializerError

at Client.main(Client.java:15)

Caused by: java.lang.RuntimeException: Uncompilable source code - constructor LuckyNumberList(java.lang.String) is already defined in class LuckyNumberList

at LuckyNumberList.<clinit>(LuckyNumberList.java:17)

... 1 more

C:\Users\marie\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1

BUILD FAILED (total time: 1 second)