SOURCE CODE

SOURCE CODE

Mouna Giri

2017

1. Coupon\_Inventory\_System.java

package cs401\_Project;

import java.io.File;

import java.io.IOException;

import java.util.LinkedList;

import java.util.Scanner;

public class Coupon\_Inventory\_System {

private static LinkedListArray<Coupon> coupon\_list = new LinkedListArray<Coupon>();

private static Sorted\_LinkedList\_Array<Coupon> sorted\_coupon\_list;

static int search\_count,flag;

public static void main(String[] args) throws Exception{

String readLine;

String[] readLineSplited;

Scanner scan = new Scanner(System.in);

for(;;)

{

System.out.println("---------------------------------------------");

System.out.println("WELCOME TO CS 401 COUPON INVENTORY SYSTEM");

System.out.println("THIS IS THE MENU ");

System.out.println("---------------------------------------------");

System.out.println("1. Purchase Coupons");

System.out.println("2. Search Coupons ");

System.out.println("3. List of all the Coupons");

System.out.println("4. Exit");

System.out.println("---------------------------------------------");

int menu\_entry = scan.nextInt();

if(menu\_entry == 1)

{

System.out.println("\nThis is the section to input the data both manually or through a file\n");

System.out.println("---------------------------------------------------------------------------");

for(;;)

{

System.out.println("Enter the mode of input[FILE, MANUAL or EXIT]");

String input\_entry = scan.next();

if(input\_entry.equalsIgnoreCase("FILE"))

{

System.out.println("ENTER THE PATH OF INPUT FILE");

System.out.println("---------------------------------------------");

String file\_name = scan.next();

File input\_file = new File(file\_name);

int i = 0;

Scanner scan\_file = new Scanner(input\_file);

while(scan\_file.hasNextLine()){

readLine = scan\_file.nextLine();

readLineSplited= readLine.split(" ");

double final\_price = Double.parseDouble(readLineSplited[2]) - (Double.parseDouble(readLineSplited[2]) \* Integer.parseInt(readLineSplited[3]) / 100);

Coupon new\_coupon = new Coupon(readLineSplited[0],readLineSplited[1], Double.parseDouble(readLineSplited[2]), Integer.parseInt(readLineSplited[3]),final\_price ,Integer.parseInt(readLineSplited[4]), readLineSplited[5]);

coupon\_list.add(new\_coupon);

i++;

}

System.out.println("Coupons from the input file has been updated successfully");

System.out.println("----------------------------------------------------------");

}

else if(input\_entry.equalsIgnoreCase("MANUAL"))

{

for(;;){

System.out.println("Enter NEXT for adding new coupon and EXIT to stop adding manually ");

String next\_coupon=scan.next();

if(next\_coupon.equalsIgnoreCase("exit"))

{

break;

}

else if(next\_coupon.equalsIgnoreCase("next"))

{

System.out.println("---------------------------------------------");

System.out.println("Enter the values of the coupons");

System.out.println("Enter the input for Coupon provider ");

String provider\_name =scan.next();

System.out.println("Enter the input for the product name");

String prod\_name = scan.next();

System.out.println("Enter the input for price ");

double prod\_price = scan.nextDouble();

System.out.println("Enter the input for discount");

int prod\_discount = scan.nextInt();

System.out.println("Enter the input for expiration days");

int exp\_days = scan.nextInt();

System.out.println("Enter the status of the coupon");

String status = scan.next();

double final\_price = prod\_price - (prod\_price\*prod\_discount/100);

Coupon new\_coupon = new Coupon(provider\_name, prod\_name, prod\_price, prod\_discount,final\_price, exp\_days, status);

coupon\_list.add(new\_coupon);

System.out.println("The new coupon has been added successfully");

System.out.println("---------------------------------------------");

}

}

}

else if(input\_entry.equalsIgnoreCase("EXIT"))

{

break;

}

else

System.out.println("INVALID USER INPUT - TYPE FILE OR MANUAL OR EXIT BASED ON YOUR CHOICE");

}

}

else if(menu\_entry == 2)

{

System.out.println("This is the section to search for the coupon(s)");

System.out.println("---------------------------------------------");

search\_count=0;

flag=0;

int count\_linear=0;

int count\_BST=0;

int count\_not\_linear=0;

Scanner scan\_search = new Scanner(System.in);

System.out.println("ENTER THE DETAIL OF THE COUPON [COUPON PROVIDER NAME OR COUPON PRODUCT NAME]");

String coupon\_prod = scan\_search.next();

linear\_search(coupon\_prod);

Coupon cou =new Coupon();

Coupon cou1 =new Coupon();

cou.setProduct\_name(coupon\_prod);

Sorted\_LinkedList\_Array<Coupon> coupon\_list\_sort2 = new Sorted\_LinkedList\_Array<Coupon>(cou1.Coupon\_Product\_Comparator());

for(int p=0;p< coupon\_list.size();p++)

{

coupon\_list\_sort2.add(coupon\_list.get(p));

}

int binary\_count = binarysearch(coupon\_list\_sort2,coupon\_prod,0,coupon\_list.size());

if(flag != 1)

{

System.out.println("NO COUPON COUNT");

System.out.println("The SEARCH COUNT IN BINARY SEARCH IS : "+search\_count);

}

}

else if(menu\_entry == 3)

{

System.out.println("This is the section to list all the coupons based on user's choice");

System.out.println("---------------------------------------------");

Coupon obj = new Coupon();

Scanner scan4 = new Scanner(System.in);

System.out.println("ENTER THE PARAMETER OF COUPON AND IT WILL LISTED ACCORDINGLY [provider, price, final\_ price, discount, expiration, product or status]");

String coupon\_parameter = scan4.next();

if(coupon\_parameter.equalsIgnoreCase("provider"))

{

Sorted\_LinkedList\_Array<Coupon> coupon\_list\_sort1 = new Sorted\_LinkedList\_Array<Coupon>(obj.Coupon\_Provider\_Comparator());

adding\_sorted\_array(coupon\_list\_sort1);

}

else if(coupon\_parameter.equalsIgnoreCase("product"))

{

Sorted\_LinkedList\_Array<Coupon> coupon\_list\_sort2 = new Sorted\_LinkedList\_Array<Coupon>(obj.Coupon\_Product\_Comparator());

adding\_sorted\_array(coupon\_list\_sort2);

}

else if(coupon\_parameter.equalsIgnoreCase("price"))

{

Sorted\_LinkedList\_Array<Coupon> coupon\_list\_sort3 = new Sorted\_LinkedList\_Array<Coupon>(obj.Coupon\_Price\_Comparator());

adding\_sorted\_array(coupon\_list\_sort3);

}

else if(coupon\_parameter.equalsIgnoreCase("discount"))

{

Sorted\_LinkedList\_Array<Coupon> coupon\_list\_sort4 = new Sorted\_LinkedList\_Array<Coupon>(obj.Coupon\_Discount\_Comparator());

adding\_sorted\_array(coupon\_list\_sort4);

}

else if(coupon\_parameter.equalsIgnoreCase("expiration"))

{

Sorted\_LinkedList\_Array<Coupon> coupon\_list\_sort5 = new Sorted\_LinkedList\_Array<Coupon>(obj.Coupon\_Expiration\_Comparator());

adding\_sorted\_array(coupon\_list\_sort5);

}

else if(coupon\_parameter.equalsIgnoreCase("status"))

{

Sorted\_LinkedList\_Array<Coupon> coupon\_list\_sort6 = new Sorted\_LinkedList\_Array<Coupon>(obj.Coupon\_Status\_Comparator());

adding\_sorted\_array(coupon\_list\_sort6);

}

else if(coupon\_parameter.equalsIgnoreCase("final\_Price"))

{

Sorted\_LinkedList\_Array<Coupon> coupon\_list\_sort7 = new Sorted\_LinkedList\_Array<Coupon>(obj.Coupon\_Final\_Price\_Comparator());

adding\_sorted\_array(coupon\_list\_sort7);

}

else

{

System.out.println("INVALID PARAMETER");

}

}

else if(menu\_entry == 4)

{

System.out.println("THANK YOU. EXITING THE PROGRAM");

System.out.println("---------------------------------------------");

break;

}

}

}

public static void linear\_search(String coupon\_entry)

{

LinkedList n = new LinkedList();

int count\_not\_linear=0;

for(int m=0; m< coupon\_list.size();m++){

if(coupon\_list.get(m).getProduct\_name().equalsIgnoreCase(coupon\_entry)

|| coupon\_list.get(m).getCoupon\_provider().equalsIgnoreCase(coupon\_entry)

|| coupon\_list.get(m).getStatus\_coupon().equalsIgnoreCase(coupon\_entry) )

{

n.add(coupon\_list.get(m));

n.add(m);

}

else

{

count\_not\_linear=m;

}

}

if(n.size() == 0)

{

System.out.println("NO COUPON FOUND");

System.out.println("THE SEARCH COUNT BY LINEAR SEARCH ALGORITHM IS : " + count\_not\_linear);

}

else

{

System.out.println("COUPON IS FOUND");

for (int a =0; a<n.size();a+=2){

System.out.println("SEARCH COUNT FOR LINEAR SEARCH IS " + n.get(a+1));

System.out.println(" And the coupon is :"+n.get(a));

}

}

}

public static void adding\_sorted\_array(Sorted\_LinkedList\_Array<Coupon> coupon\_list\_sort){

for(int p=0;p< coupon\_list.size();p++)

{

coupon\_list\_sort.add(coupon\_list.get(p));

}

for(int p=0;p< coupon\_list\_sort.size();p++)

{

System.out.println((p+1)+". "+ coupon\_list\_sort.get(p));

}

}

public static int binarysearch(Sorted\_LinkedList\_Array<Coupon> sorted\_list,String target,int first,int last)

{

int midpoint=(first+last)/2;

if(first>last)

return -1;

else if(target.equalsIgnoreCase(sorted\_list.get(midpoint).getProduct\_name()))

{

search\_count++;

flag=1;

System.out.println("The SEARCH COUNT IN BINARY SEARCH IS"+search\_count);

return search\_count;

}

else if(target.compareToIgnoreCase(sorted\_list.get(midpoint).getProduct\_name()) > 0)

{

search\_count++;

binarysearch(sorted\_list,target,midpoint+1,last);

return search\_count;

}

else

{

search\_count++;

binarysearch(sorted\_list,target,first,midpoint-1);

return search\_count;

}

}

}

1. Coupon.java

package cs401\_Project;

import java.util.Comparator;

public class Coupon implements Comparable<Coupon> {

private String coupon\_provider;

private String product\_name;

private double price;

private double final\_price;

private int discount;

private int expiration\_date;

private String status\_coupon;

/\*\*

\* PUBLIC CONSTRUCTOR

\*/

public Coupon() {

coupon\_provider = "";

product\_name = "";

price = 0;

final\_price=0;

discount = 0;

expiration\_date = 0;

status\_coupon = "";

}

/\*\*

\* PARAMETERIZED CONSTRUCTOR

\* @param coupon\_provider

\* @param product\_name

\* @param price

\* @param discount

\* @param final\_price

\* @param expiration\_date

\* @param status\_coupon

\*/

public Coupon(String coupon\_provider, String product\_name, double price,

int discount,double final\_price, int expiration\_date, String status\_coupon) {

this.coupon\_provider = coupon\_provider;

this.product\_name = product\_name;

this.price = price;

this.discount = discount;

this.final\_price = final\_price;

this.expiration\_date = expiration\_date;

this.status\_coupon = status\_coupon;

}

/\*\*

\* @return the coupon\_provider

\*/

public String getCoupon\_provider() {

return coupon\_provider;

}

/\*\*

\* @param coupon\_provider the coupon\_provider to set

\* @throws Exception

\*/

public void setCoupon\_provider(String coupon\_provider) throws Exception {

if(coupon\_provider.length() <= 20)

this.coupon\_provider = coupon\_provider;

else

throw new Exception ("The Coupon Provider name should be less than 20 bytes");

}

/\*\*

\* @return the final\_price

\*/

public double getFinal\_price() {

return final\_price;

}

/\*\*

\* @param final\_price the final\_price to set

\*/

public void setFinal\_price(double final\_price) {

this.final\_price = final\_price;

}

/\*\*

\* @return the product\_name

\*/

public String getProduct\_name() {

return product\_name;

}

/\*\*

\* @param product\_name the product\_name to set

\* @throws Exception

\*/

public void setProduct\_name(String product\_name) throws Exception {

if(product\_name.length() <= 20)

this.product\_name = product\_name;

else

throw new Exception ("The Product name should be less than 20 character");

}

/\*\*

\* @return the price

\*/

public double getPrice() {

return price;

}

/\*\*

\* @param price the price to set

\*/

public void setPrice(double price) {

this.price = price;

}

/\*\*

\* @return the discount

\*/

public int getDiscount() {

return discount;

}

/\*\*

\* @param discount the discount to set

\* @throws Exception

\*/

public void setDiscount(int discount) throws Exception {

if(discount < 80 && discount > 5)

this.discount = discount;

else

throw new Exception ("The discount percent should range between 5 to 80%");

}

/\*\*

\* @return the expiration\_date

\*/

public int getExpiration\_date() {

return expiration\_date;

}

/\*\*

\* @param expiration\_date the expiration\_date to set

\* @throws Exception

\*/

public void setExpiration\_date(int expiration\_date) throws Exception {

if(expiration\_date < 365 && expiration\_date > 0)

this.expiration\_date = expiration\_date;

else

throw new Exception ("The expiration date should range from 0 to 365 days");

}

/\*\*

\* @return the status\_coupon

\*/

public String getStatus\_coupon() {

return status\_coupon;

}

/\*\*

\* @param status\_coupon the status\_coupon to set

\* @throws Exception

\*/

public void setStatus\_coupon(String status\_coupon) throws Exception {

if(status\_coupon.equalsIgnoreCase("UNUSED") || status\_coupon.equalsIgnoreCase("REDEEMED"))

this.status\_coupon = status\_coupon;

else

throw new Exception ("The Product name should be less than 20 character");

}

/\* (non-Javadoc)

\* @see java.lang.Object#toString()

\*/

@Override

public String toString() {

return "Coupon [ coupon\_provider = " + coupon\_provider + ", product\_name = "

+ product\_name + ", price = " + price + ", final\_price = "

+ final\_price + ", discount = " + discount + ", expiration\_date = "

+ expiration\_date + ", status\_coupon = " + status\_coupon + "]";

}

/\* (non-Javadoc)

\* @see java.lang.Object#hashCode()

\*/

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result

+ ((coupon\_provider == null) ? 0 : coupon\_provider.hashCode());

result = prime \* result + discount;

result = prime \* result + expiration\_date;

long temp;

temp = Double.doubleToLongBits(final\_price);

result = prime \* result + (int) (temp ^ (temp >>> 32));

temp = Double.doubleToLongBits(price);

result = prime \* result + (int) (temp ^ (temp >>> 32));

result = prime \* result

+ ((product\_name == null) ? 0 : product\_name.hashCode());

result = prime \* result

+ ((status\_coupon == null) ? 0 : status\_coupon.hashCode());

return result;

}

/\* (non-Javadoc)

\* @see java.lang.Object#equals(java.lang.Object)

\*/

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

Coupon other = (Coupon) obj;

if (coupon\_provider == null) {

if (other.coupon\_provider != null)

return false;

} else if (!coupon\_provider.equals(other.coupon\_provider))

return false;

if (discount != other.discount)

return false;

if (expiration\_date != other.expiration\_date)

return false;

if (Double.doubleToLongBits(final\_price) != Double

.doubleToLongBits(other.final\_price))

return false;

if (Double.doubleToLongBits(price) != Double

.doubleToLongBits(other.price))

return false;

if (product\_name == null) {

if (other.product\_name != null)

return false;

} else if (!product\_name.equals(other.product\_name))

return false;

if (status\_coupon == null) {

if (other.status\_coupon != null)

return false;

} else if (!status\_coupon.equals(other.status\_coupon))

return false;

return true;

}

@Override

public int compareTo(Coupon coupon) {

if (discount < coupon.discount)

return -1;

else if (discount > coupon.discount )

return 1;

else

return 0;

}

public static Comparator<Coupon> Coupon\_Provider\_Comparator()

{

return new Comparator<Coupon>()

{

public int compare(Coupon coupon1, Coupon coupon2)

{

return (coupon1.coupon\_provider.compareTo(coupon2.coupon\_provider));

}

};

}

public static Comparator<Coupon> Coupon\_Product\_Comparator()

{

return new Comparator<Coupon>()

{

public int compare(Coupon coupon1, Coupon coupon2)

{

return (coupon1.product\_name.compareTo(coupon2.product\_name));

}

};

}

public static Comparator<Coupon> Coupon\_Price\_Comparator()

{

return new Comparator<Coupon>()

{

public int compare(Coupon coupon1, Coupon coupon2)

{

return (int) (coupon1.price - coupon2.price);

}

};

}

public static Comparator<Coupon> Coupon\_Discount\_Comparator()

{

return new Comparator<Coupon>()

{

public int compare(Coupon coupon1, Coupon coupon2)

{

return (int) (coupon1.discount - coupon2.discount);

}

};

}

public static Comparator<Coupon> Coupon\_Expiration\_Comparator()

{

return new Comparator<Coupon>()

{

public int compare(Coupon coupon1, Coupon coupon2)

{

return (int) (coupon1.expiration\_date - coupon2.expiration\_date);

}

};

}

public static Comparator<Coupon> Coupon\_Status\_Comparator()

{

return new Comparator<Coupon>()

{

public int compare(Coupon coupon1, Coupon coupon2)

{

return (int) (coupon1.status\_coupon.compareTo(coupon2.status\_coupon));

}

};

}

public static Comparator<Coupon> Coupon\_Final\_Price\_Comparator()

{

return new Comparator<Coupon>()

{

public int compare(Coupon coupon1, Coupon coupon2)

{

return (int) (coupon1.final\_price - coupon2.final\_price);

}

};

}

}

1. CollectionInterface.java

package cs401\_Project;

public interface CollectionInterface<T>

{

/\*\*

\* Determines if this data structure is at its capacity.

\*

\* @return true - if this data structure is at its capacity; false otherwise.

\*/

public boolean is\_full();

/\*\*

\* Determines if this data structure is empty.

\*

\* @return true - if this data structure is empty; false otherwise.

\*/

public boolean is\_empty();

/\*\*

\* Determines the number of elements in this data structure.

\*

\* @return the number of elements currently resident in this

\* data structure.

\*/

public int size();

/\*\*

\* Add a new element.

\*

\* @param e the element to be added.

\*

\* It is expected that classes that extend this interface will

\* provide an order on how the element is added to the collection.

\*/

public boolean add(T e);

/\*\*

\* Remove the specified element.

\*

\* @param i - Index of the element to be removed.

\*

\* @return the element removed, if the element exists on the collection,

\* null otherwise.

\*/

public T remove(int i);

/\*\*

\* Determine if the element is contained in this list.

\*

\* @param e the element to be searched for.

\*

\* @return true - if e was in the list, false otherwise.

\*/

public boolean contains(T e);

}

1. ListInterface.java

package cs401\_Project;

import java.util.\*;

/\*\*

\* @author mounagiri

\*

\* @param <T>

\*/

/\*\*

\* @author mounagiri

\*

\* @param <T>

\*/

public interface ListInterface<T> extends CollectionInterface<T>, Iterable<T>

{

/\*\* A new element is added at the index position

\* @param index

\* @param element

\*/

void add(int index, T element);

/\*\*

\* @param index

\* @param newElement

\* @return T

\*/

T set(int index, T newElement);

/\*\*The element at the index position is returned

\* @param index

\* @return T

\*/

T get(int index);

/\*\* The index of the element is returned

\* @param target

\* @return int

\*/

int indexOf(T target);

/\* The element at the index is removed

\* @param index

\*/

T remove(int index);

}

1. LinkedListArray.java

package cs401\_Project;

import java.util.Iterator;

public class LinkedListArray<T> implements ListInterface<T>{

int num\_elements=0;

T elements[];

int default\_size = 50;

int user\_size=0;

int current\_pointer ;

public LinkedListArray() {

super();

elements = (T[]) new Object[default\_size];

}

public LinkedListArray(int size) {

super();

elements = (T[]) new Object[size];

}

@Override

public boolean is\_full() {

if (num\_elements == default\_size){

return true;

}

return false;

}

@Override

public boolean is\_empty() {

if (num\_elements == 0){

return true;

}

return false;

}

@Override

public int size() {

return num\_elements;

}

@Override

public boolean add(T e) {

elements[num\_elements] = e ;

num\_elements++;

return true;

}

@Override

public boolean contains(T e) {

if (num\_elements > 0) {

for (int i = 0 ; i < num\_elements ; i++ ) {

if(elements[i].equals(e))

return true;

}

}

return false;

}

@Override

public Iterator<T> iterator() {

return new Iterator<T>()

{

private int previousPos = -1;

public boolean hasNext() {

return (previousPos < (size() - 1)) ;

}

public T next()

{

if (!hasNext())

throw new IndexOutOfBoundsException("Illegal invocation of next");

previousPos++;

return elements[previousPos];

}

public void remove()

{

for (int i = previousPos; i <= num\_elements - 2; i++)

elements [i] = elements[i+1];

elements [num\_elements - 1] = null;

num\_elements--;

previousPos--;

} };

}

@Override

public void add(int index, T element) {

if (element != null) {

elements[index] = element ;

num\_elements++;

}

}

@Override

public T set(int index, T newElement)

{

if ((index < 0) || (index >= size()))

throw new IndexOutOfBoundsException("Illegal index of " + index +" passed to ABList set method.\n");

T hold = elements[index];

elements[index] = newElement;

num\_elements++;

return hold;

}

@Override

public T get(int index) {

T element\_value= elements[index];

if(element\_value != null)

{

return element\_value;

}

return null;

}

@Override

public int indexOf(T target) {

if (num\_elements > 0) {

for (int i = 0 ; i < num\_elements ; i++ ) {

if(elements[i] == target)

return i;

}

}

return -1;

}

@Override

public T remove(int index) {

T elem=null;

T current\_pointer = elements[index] ;

for (int i = index + 1 ; i < num\_elements ; i++) {

elements[index] = elements[i] ;

index++ ;

}

num\_elements-- ;

return elem;

}

}

6. Sorted\_LinkedList\_Array.java

package cs401\_Project;

import java.util.Comparator;

import java.util.Iterator;

public class Sorted\_LinkedList\_Array<T> implements ListInterface<T>{

int num\_elements=0;

T elements[];

int size = 50;

protected Comparator<T> comp\_obj;

protected boolean found;

protected int loc\_value;

int current\_pointer ;

public Sorted\_LinkedList\_Array()

{

elements = (T[]) new Object[size];

comp\_obj = new Comparator<T>()

{

public int compare(T element1, T element2)

{

return ((Comparable<T>)element1).compareTo(element2);

}

};

}

public Sorted\_LinkedList\_Array(Comparator<T> comp\_obj)

{

elements = (T[]) new Object[size];

this.comp\_obj = comp\_obj;

}

public void add(int index, T element)

{

throw new UnsupportedOperationException("Unsupported index-based add method");

}

public T set(int index, T newElement)

{

throw new UnsupportedOperationException("Unsupported index-based set method");

}

@Override

public boolean is\_full() {

if (num\_elements == size ){

return true;

}

return false;

}

@Override

public boolean is\_empty() {

if (num\_elements == 0){

return true;

}

return false;

}

@Override

public int size() {

return num\_elements;

}

protected void Find\_recursive(T target, int first, int last)

{

int result;

if (first > last)

{

found = false;

result = comp\_obj.compare(target,elements[loc\_value]);

if (result > 0)

loc\_value++;

}

else

{

loc\_value = (first + last) / 2;

result = comp\_obj.compare(target,elements[loc\_value]);

if (result == 0)

found = true;

else

if (result > 0)

Find\_recursive(target, loc\_value + 1, last);

else

Find\_recursive(target, first, loc\_value - 1);

}

}

public boolean add(T element)

{

loc\_value = 0;

found = false;

if (!is\_empty())

Find\_recursive(element, 0, num\_elements - 1);

for (int m = num\_elements; m > loc\_value; m--)

elements[m] =elements[m - 1];

elements[loc\_value] = element;

num\_elements++;

return true;

}

@Override

public boolean contains(T e) {

if (num\_elements > 0) {

for (int i = 0 ; i < num\_elements ; i++ ) {

if(elements[i].equals(e))

return true;

}

}

return false;

}

@Override

public Iterator<T> iterator() {

return new Iterator<T>()

{

private int previousPos = -1;

public boolean hasNext() {

return (previousPos < (size() - 1)) ;

}

public T next()

{

if (!hasNext())

throw new IndexOutOfBoundsException("Illegal invocation of next " +" in LBList iterator.\n");

previousPos++;

return elements[previousPos];

}

public void remove()

{

for (int i = previousPos; i <= num\_elements - 2; i++)

elements [i] = elements[i+1];

elements [num\_elements - 1] = null;

num\_elements--;

previousPos--;

} };

}

@Override

public T get(int index) {

T element\_value= elements[index];

return element\_value;

}

@Override

public int indexOf(T target) {

if (num\_elements > 0) {

for (int i = 0 ; i < num\_elements ; i++ ) {

if(elements[i] == target)

return i;

}

}

return -1;

}

@Override

public T remove(int index) {

T elem=null;

T current\_pointer = elements[index] ;

for (int i = index + 1 ; i < num\_elements ; i++) {

elements[index] = elements[i] ;

index++ ;

}

num\_elements-- ;

return elem;

}

}