2017

SOURCE CODE

SOURCE CODE

MOUNA GIRI

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;

        }

        }
        }
```

2 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);
}
};
}
}
```

3 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);
}
```

4 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);
}
```

5 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;
}
}
```