HW2

name:Shahad almuhizi
id:436201525
section:44126

Problem 1:

1)

```
1 public <T> static void clear(ArrayList<T> l){
2 if (l.empty())
3 return;
4 else{
5 l.findFirst();
6 while(! l.last())
7 l.remove();
8 l.remove();}}
```

for linked list it would be the same except for the header:

```
1 public <T> static void clear(LinkedList<T> l)
```

2)

```
 1 public static <T> void insertAll (List<T> l1,
List<T> l2, int i){
 2 l1.findFirst();
 3 l2.findFirst();
 4 T temp=l2.retrieve();
 5 if (! l1.full() && ! l2.empty()){
 6 for(int j=0; j<i; j++)
 7 l1.findNext();
 8 while(! l2.last()){
 9 l1.insert(temp);
10 l2.findNext();
```

```
11 temp=l2.retrieve();}
12 l1.insert(l2.retrieve());}}
```

3)

```
 1 public static <T> void commonE(List<T> l1, List<T>
l2, List <T> c1){
 2 l1.findFirst();
 3 l2.findFirst();
 4 while (! l1.empty() && ! l1.last()){
 5 while(! l2.last()){
 6 if (l1.retrieve().equals(l2.retrieve()))
 7 c1.insert(l1.retrieve());
 8 l2.findNext();}
 9 if (l1.retrieve().equals(l2.retrieve()))
10 c1.insert(l2.retrieve);
11 l1.findNext();
12 l2.findFirst();}
13 while(! l2.last()){
14 if (l1.retrieve().equals(l2.retrieve()))
15 c1.insert(l1.retrieve());
16 l2.findNext();}
17 if (l1.retrieve().equals(l2.retrieve()))
18 c1.insert(l1.retrieve);}
```

problem 2:

a)

```
 1 public void insert BeforeCurrent(T e) {
 2 Node<T> newNode= new Node<T>(e);
 3 if(head==null)
 4 head=current=new Node<T>(e);
 5 else if(current== head) {
 6 newNode.next= head;
```

```
 7 head= newNode;}
 8 else {
 9 Node<T> previous= head;
10 while(previous.next!= current)
11 previous= previous.next;
12 newNode.next= current;
13 previous.next= newNode;}
14 current= newNode;}
```

b)

```
 1 public removeIth ( int i ){
 2 if (i == 0)
 3 head = head.next;
 4 else {
 5 Node<T> tmp = head;
 6 for (int j=0; j<i; j++)
 7 current=current.next;
 8 while (tmp.next != current)
 9 tmp = tmp.next;
10 tmp.next = current.next;}
11 if (current.next == null)
12 current = head;
13 else
14 current = current.next;}
```

2)

```
1 public void removeEvenElems(){
2 int count=0;
3 for (int j=1; j<size+1; j+=2){
4 nodes[count]=nodes[j];
```

```
5 count++;}
6 size -=count;
7 for(int i=count; i<maxSize; ;i++)
8 nodes[i]=null;}
```

Problem 3:

1)

```
 1 public static<T> boolean
checkListEndsSymmetry(DoubleLinkedList<T> dl,int k){
 2 LinkedList<T> temp=new LinkedList<T>();
 3 int c=0;
 4 while(!dl.last()&&!dl.empty())
 5 dl.findNext();
 6 for(int i=0;i<k;i++){
 7 temp.insert(dl.retrieve());
 8 dl.findPrevious();}
 9 temp.findFirst();
10 dl.findFirst();
11 for(int j=0;j<k;j++)
12 if(dl.retrieve().equals(temp.retrieve())){
13 c++;
14 dl.findNext();
15 temp.findNext();}
16 if(c!=k)
17 return false;
18 else
19 return true;}
```

2)

```
1 public void bubbleSort(DoubleLinkedList<Integer> l){
2 l.findFirst();
3 int size=0;
4 int count=0;
5 if(l.empty())
6 return;
7 while(! l.last()){
8 size++;
9 l.findNext();}
10 size++;
11 l.findFirst();
12 for(int i=0;i<n;i++){
13 l.findFirst();
14 for(int j=0;j<n-i-1;j++){
15 Integer tmp=l.retrieve();
16 l.findNext();
17 if(tmp<l.retrieve()){
18 l.findPrevious();
19 Integer tmp2=l.retrieve();
20 l.remove();
21 l.insert(tmp2);
22 count++;}
23 if(count==0)
24 return;}}}
```

problem 4:

```
1 public boolean full(){
2 return size==maxSize;}
```

```
public void insert(T val){
if (! full()){
data[size++]=new Node<T> (val);
current++;
return;}
else{
T[] arrayA;
T[] arrayB;
int size2=size;
if(full()){
int maxsize2=size*2;
int curr2=-1;
arrayA=(T[]) new Object[maxsize2];
 for(int i=0;i<size2 ;i++){
 arrayA[i]=data[current];
 curr2++;}
 arrayA[size2++]=new Node<T> (val);}
 if(size<maxSize*minRatio){
 int maxsize3=maxSize/2;
 size2=size;
 curr2=-1;
 arrayB=(T[]) new Object[maxsize3];
 for(int i=0;i<size2;i++){
 arrayB[i]=data[current];
 curr2++; }
 arrayB[size2++]=val; }
 for(int i=0;i<maxSize;i++){
 findFirst();
 remove();}}}
```

```
1 public void remove(){
2 for (int i=current; i<size; i++)
3 data[i] = data[i+1];
4 size--;
5 if (size == 0)
6 current=-1;
7 else if (current==size)
8 current=0;}
```

Problem 5:

1)

Elements: The elements are of generic type <Type>

Structure: the elements are linearly arranged. List where all nodes are connected to form a circle. There is no null at the end of it.

Domain: the number of elements in the list is bounded therefore the domain is finite. Type name of elements in the domain: List

Operations: We assume all operations operate on a list L.

1. **Method** FindNext ( )

   requires: list L is not empty, Cur is not last. **input:** none

   results: the element following the current element is made the current element.

   output: none.

2. Method Retrieve(Type e)

   requires: list L is not empty. **input**: none

   results: current element is copied into e. **output**: element e.

3. **Method** Update (Type e).

   requires: list L is not empty. **input**: e.

   results: the element e is copied into the current node.

   output: none.

4. **Method** Insert (Type e).

   requires: list L is not full. **input**: e.

   results: a new node containing element e is created and inserted after the current element in the list. The new element e is made the current element. If the list is empty e is also made the current element. **output**: none.

5. **Method** Remove ( )

   requires: list L is not empty. **input**: none

   results: the current element is removed from the list. If the resulting list is empty current is set to NULL. If successor of the deleted element exists it is made the new current element. **output**: none.

6. **Method** Full (boolean flag)

input: none. **returns**: if the number of elements in L has reached the maximum number allowed then flag is set to true otherwise false. **output**: flag.

7. **Method** Empty (boolean flag).

input: none. **results**: if the number of elements in L is zero, then flag is set to true otherwise false. **Output**: flag.

2)

```
public interface CircularList<T>{
public void findNext( );
public T retrieve( );
public void update(T e);
public void insert(T e);
public void remove( );
public boolean full( );
public bolean empty( );
}
```

3)

```
1 public void print(CircularList<String> l){
2 linkedList<String> l2= new linkedList<String>();
3 while(! l.empty()){
4 System.out.println(l.retrieve()+" ");
5 l2.insert((l.retrieve());
6 l.remove();}
7 l2.findFirst();
```

```
 8 while(! l2.empty()){
 9 l.insert((l2.retrieve());
10 l2.remove();}
11 l.findNext();}
```

4)

```
public class LinkedCircularList<T> implements CircularList<T> {
   private Node<T> current;

   public LinkedList () {
   current = null;}

public void findNext () {
                current = current.next;
   }
   public T retrieve () {
                return current.data;
   }
   public void update (T val) {
                current.data = val;
   }


public boolean empty () {
                return head == null;
   }
public boolean full () {
                return false;
   }



public void insert (T val) {
```

```
                Node<T> tmp;
                if (empty()) {
                                current = new Node<T> (val);
                        current.next=current;
                }
                else {
                                tmp = current.next;
                                current.next = new Node<T> (val);
                                current = current.next;
                                current.next = tmp;
                }
    }


public void remove () {
                if (current.next == current) {
                                current = null;
                }
                else {
                                Node<T> tmp = current;

                                while (tmp.next != current)
                                                tmp = tmp.next;

                                tmp.next = current.next;
                }

                                }   }
```