DINESHKUMAR A S

Welcome to Linear Data Structure Application

Linear Data Structures:

1. ARRAY

2. STACK

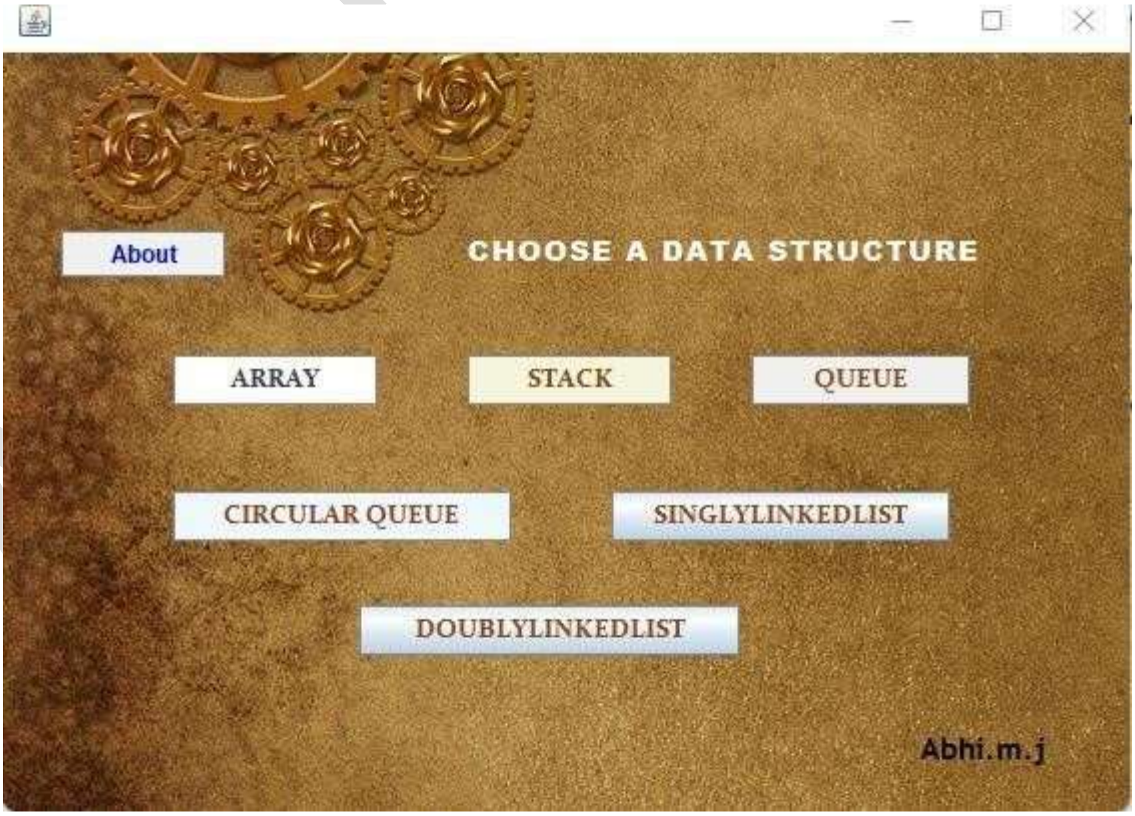
3. QUEUE

4. CIRCULAR QUEUE

5. SINGLY LINKED LIST

6. DOUBLY LINKED LIST

Home Page of Application



About: It will connect to the internet and you can read about Data Structure.

1 . ARRAY:

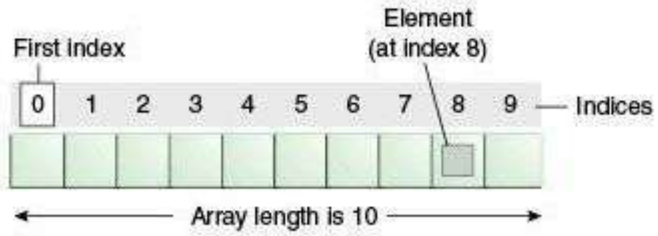
What is Array in Data Structure?

An **array** is a data structure for storing more than one data item that has a similar data type. The items of an array are allocated at adjacent memory locations. These memory locations are called **elements** of that array. The total number of elements in an array is called **length**.

Why do we need arrays?

Here, are some reasons for using arrays in data structure:

Arrays are best for storing multiple values in a single variable . Arrays are better at processing many values easily and quickly.
Sorting and searching the values is easier in arrays.

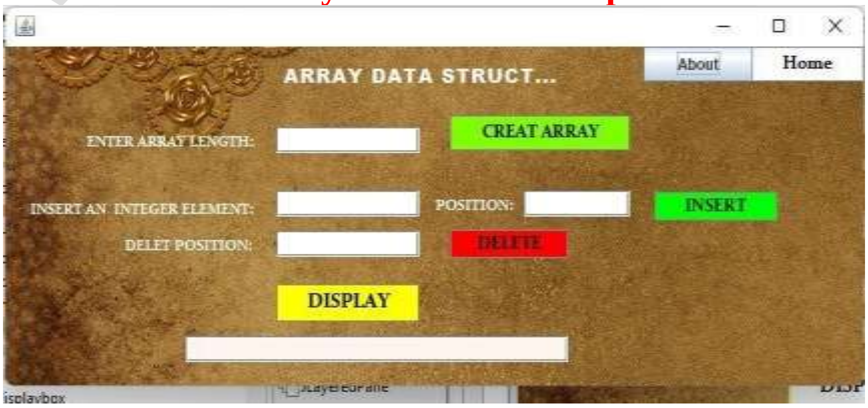


An array of 10 elements

WORK PROCEDURE OF ARRAY:

1. **insert()**

View of Array Data Structure operation



Example of array how it is working.

2. delete()

3. display()

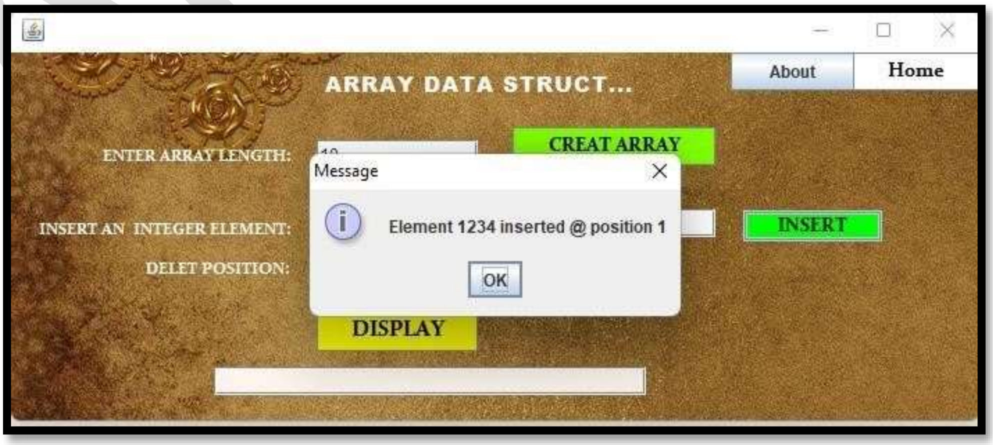
1. insert(): it will insert the elements into array in orderly.

//CODE FOR CREATING ARRAY

//converting string to integer

```
try{

int len =

Integer.valueOf(lengthe.getText());

//storing in an array by creating arr=new



//showing created msg

String message="Array of length "+len+"created";

JOptionPane.showMessageDialog(contentPane,message);

}catch(Exception e)



 e.printStackTrace();
```

View of insert() operation



Element 1234 inserted @ position 1

```
int [len];
```

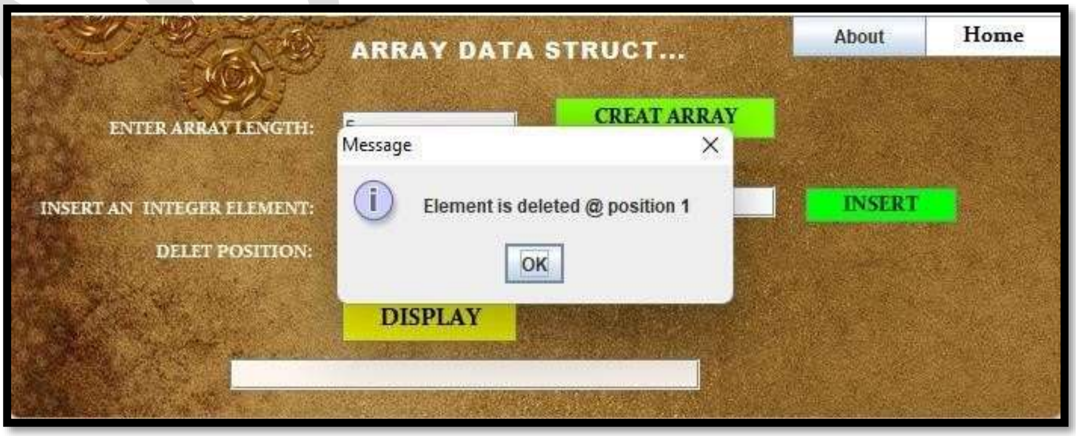
{

}

2. delete( ): it will help to delete an elements in an array ,it can delete at particular index also.

```
//deletion code

try{

int pos=Integer.valueOf(deletposition.getText());

                                        arr[pos]=0;

String message="Element is deleted @position "+pos;

JOptionPane.showMessageDialog(contentPane, message);

deletposition.setText(""); //to make box empty after operation

}

 catch(Exception e)

{

 e.printStackTrace();

}
```

View of delete() operation

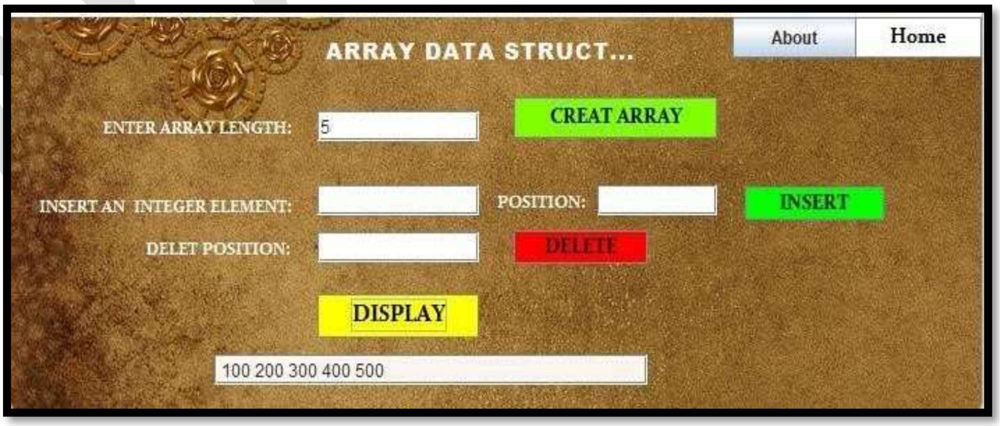


Element is deleted @ position 1

3. ***display():*** ***Displaying the content of an Array in output screen.***

```
//display

try{ String msg=""; for(int
i=0;i<=arr.length-1;i++)

{

msg=msg+" "+arr[i];


}

//to give otput to display box displaybox.setText(msg);


Catch(Exception e)


e.printStackTrace():
```

View of display() operation



ENTER ARRAY LENGTH: 5

100 200 300 400 500

```
}


{
}
```

About: It will connect to the internet and u can read about ARRAY Data Structure.
Home :It will take to you into Home page.

Real time Application of an Array:

- Contact lists on mobile phones.
- Arrays are used in online ticket booking portals.

- Pages of book.
  IoT applications use arrays as we know that the number of values in an array will remain constant, and also that the accessing will be faster.
- It is also utilised in speech processing, where each speech signal is represented by an array.
- The viewing screen of any desktop/laptop is also a multidimensional array of pixels.

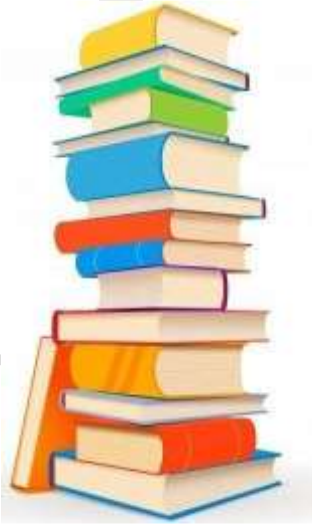
2. STACK

What is stack data structure?

Stack is a linear type of data structure that follows the LIFO (Last-In-First-Out) principle and allows insertion and deletion operations from one end of the stack data structure, that is top. Implementation of the stack can be done by contiguous memory which is an array, and noncontiguous memory which is a linked list. Stack plays a vital role in many applications.

Example:

This example allows you to perform operations from one end only, like when you insert and remove new books from the top of the stack. It means insertion and deletion in the stack data structure can be done only from the top of the stack. You can access only the top of the stack at any given point in time.



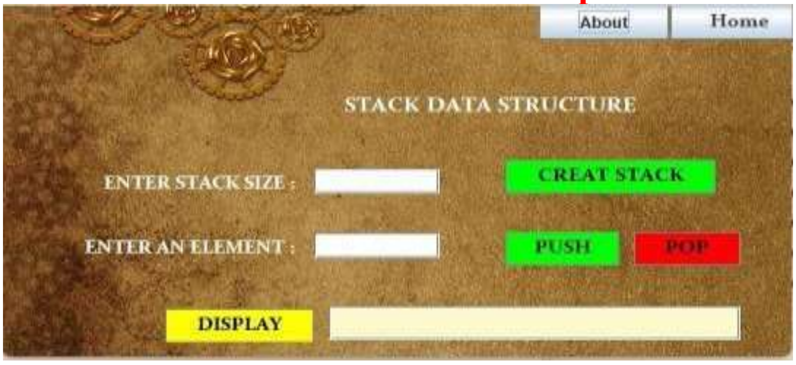
WHY DO WE USE STACK IN DATA STRUCTURE?

Stack is basically a limited access linear data structure which works on the principle of LIFO (Last in first out). Insertion of element is called push operation and deletion of element from the stack is called pop operation. These both operations took place from the same end.

WORK PROCEDURE OF STACK:

1. **push()**
2. **pop()**
3. **display()**

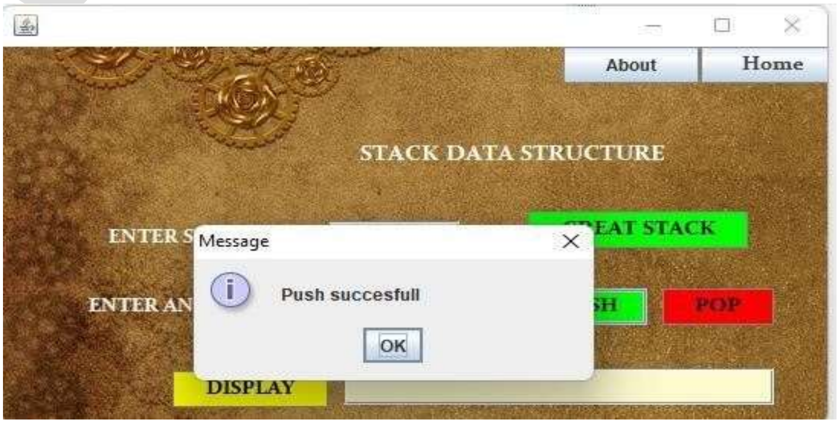
View of Stack Data Structure Operation



1. push(): **it is used for adding new elements at the top of the stack.**

```
//push  try { int elem;
```

```
if(top==size-

1){

JOptionPane.showMessageDialog(contentPane,"push not possible ");





elem=Integer.valueOf(element.getText());

++top; s[top]=elem;

JOptionPane.showMessageDialog(element, "Push succesfull"); element.setText("");


catch(Exception e1)


e1.printStackTrace();}
```



Push succesfull

```
}

Else
```

} }

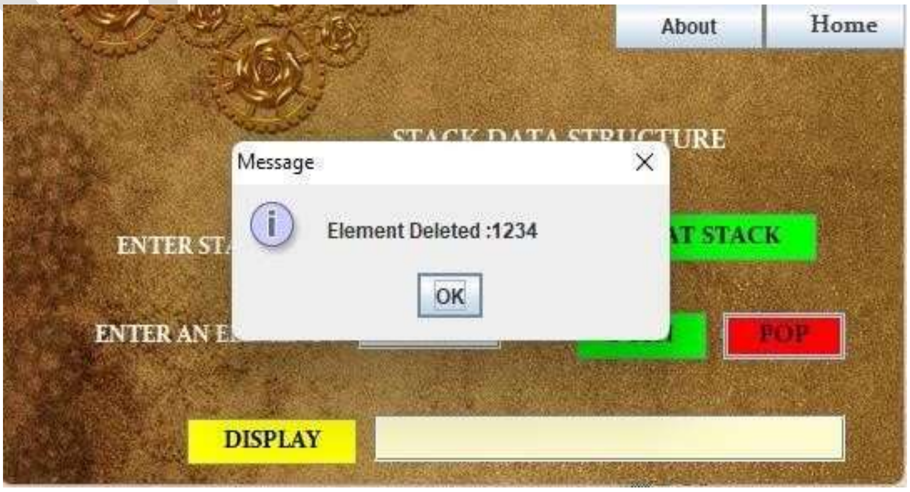
{

View of push() operation

2. pop(): it is used to remove an element from stack.

```
//pop try {



1)

{

JOptionPane.showMessageDialog(contentPane,"pop not possible because now Stack is




String message="Element Deleted :"+s[top];

JOptionPane.showMessageDialog(contentPane, message);




catch(Exception e1)


e1.printStackTrace();}
```

View of pop operation



Message

Element Deleted :1234

OK

```
if(top==empty");
```

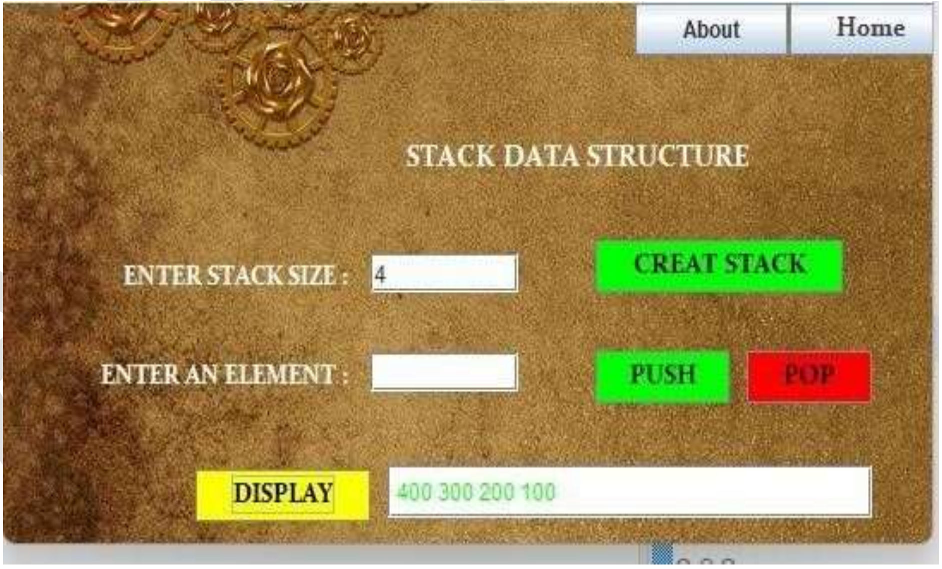
```
} else

{

--top;

}}

{
```

3. display(): **Displaying the content of an Stack in output screen.**

```
//display

String msg="";
```

```
if(top==-1)

{

JOptionPane.showMessageDialog(contentPane, "Boss Stack is Empty ");



for(int i=top;i>=0;i--)


msg =msg+" "+s[i];


display.setText(msg);
```

View of display operation



ENTER STACK SIZE : 4

ENTER AN ELEMENT :

400 300 200 100

```
} else

{


{
}
```

About: It will connect to the internet and u can read about STACK Data Structure. Home :It will take to you into Home page

Real time Application of an Stack:

- CD/DVD stand.
- Stack of books in a book shop.
- Undo and Redo mechanism in text editors.

Call logs, E-mails, and Google photos in any gallery are also stored in form of a stack. YouTube downloads and Notifications are also shown in LIFO format(the latest appears first)

- The history of a web browser is stored in the form of a stack.
-
-

3. QUEUE

What is a queue in data structures?

A queue is an ordered collection of items where the addition of new items happens at one end, called the **“rear,”** and the removal of existing items occurs at the other end, commonly called the **“front.”**

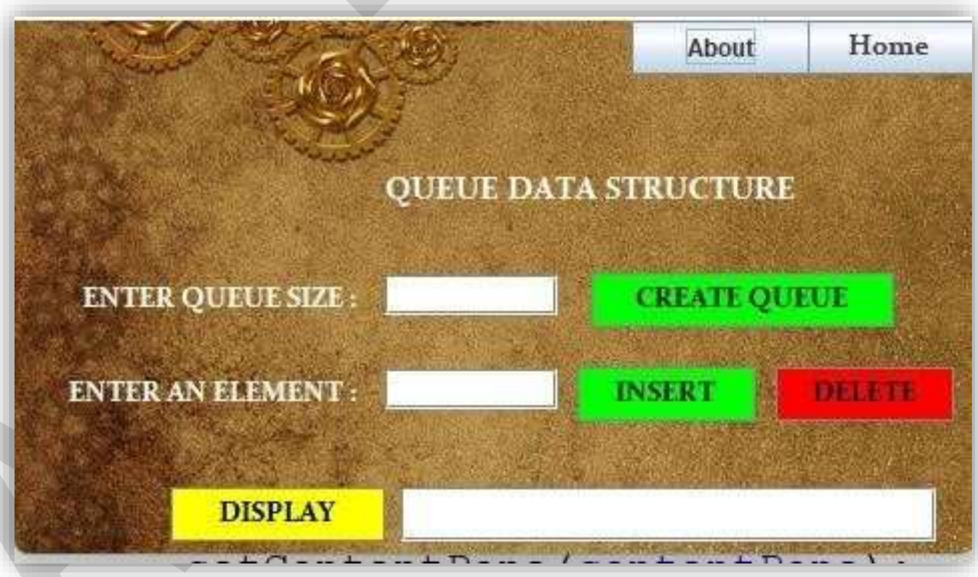
What is the use of queue data structure?

A queue is a linear data structure that stores the elements sequentially. It uses the FIFO approach (First In First Out) for accessing elements. Queues are typically used to manage threads in multithreading and implementing priority queuing systems.

WORK PROCEDURE OF QUEUE:

1. **insert()**
2. **delete()**
3. **display()**

View of queue data structure operation

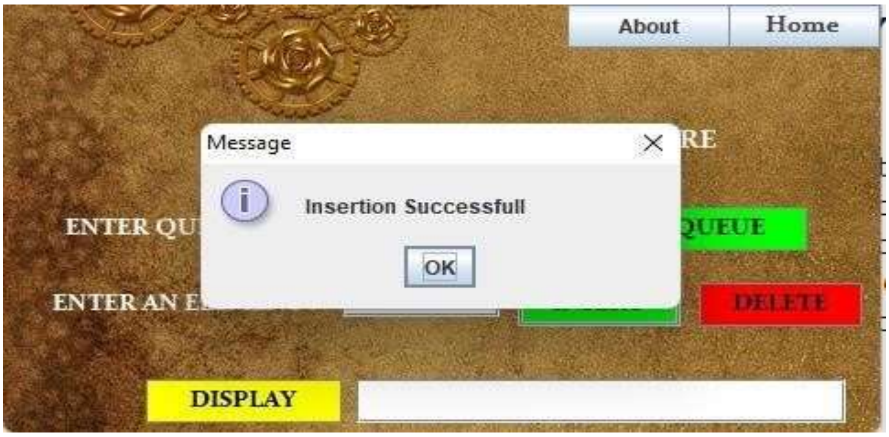


1. insert(): **it will insert a new element in a queue.**

//inserting into Queue if(rear == size-1)//to display a message if queue is full

```
{
JOptionPane.showMessageDialog(contentPane, "Queue is full! Insertion not possible");
elem.setText("");



int elem1 = Integer.valueOf(elem.getText());//asking user for the element
++rear; if(rear==-1)

JOptionPane.showMessageDialog(contentPane, "Queue is not created! Create Queue");

elem.setText("");



sq[rear] = elem1;//inserting element at rear end
JOptionPane.showMessageDialog(contentPane, "Insertion Successful");
elem.setText("");
```

View of insert() operation :



Insertion Successfull

```
} else
{
```

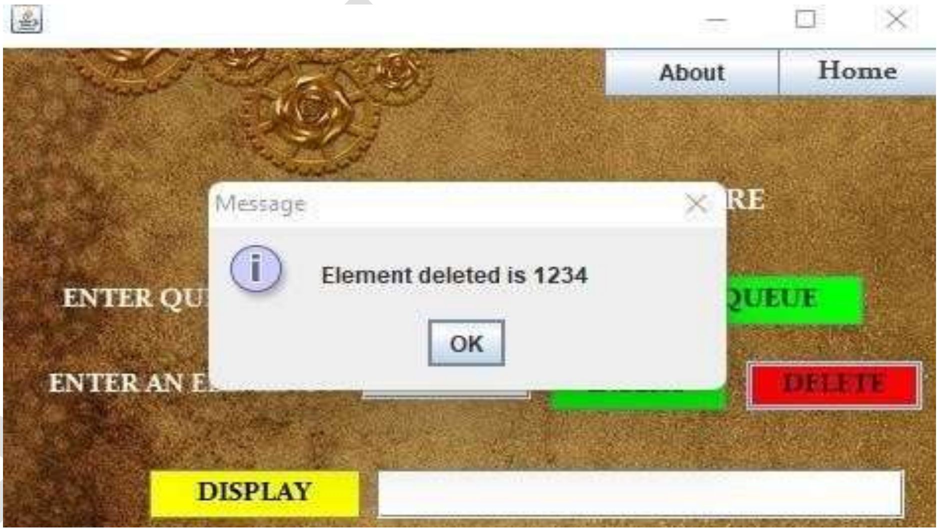
```
{

} else

{
```

2. delete():it will delete an element present front.

```
//deleting  from  queue    if(rear==-1||front
>rear)
```

```
{

JOptionPane.showMessageDialog(contentPane,"Queue is empty! Deletion not

JOptionPane.showMessageDialog(contentPane,"Element deleted is "+sq[front]);
```

View of delete() operation



Message

Element deleted is 1234

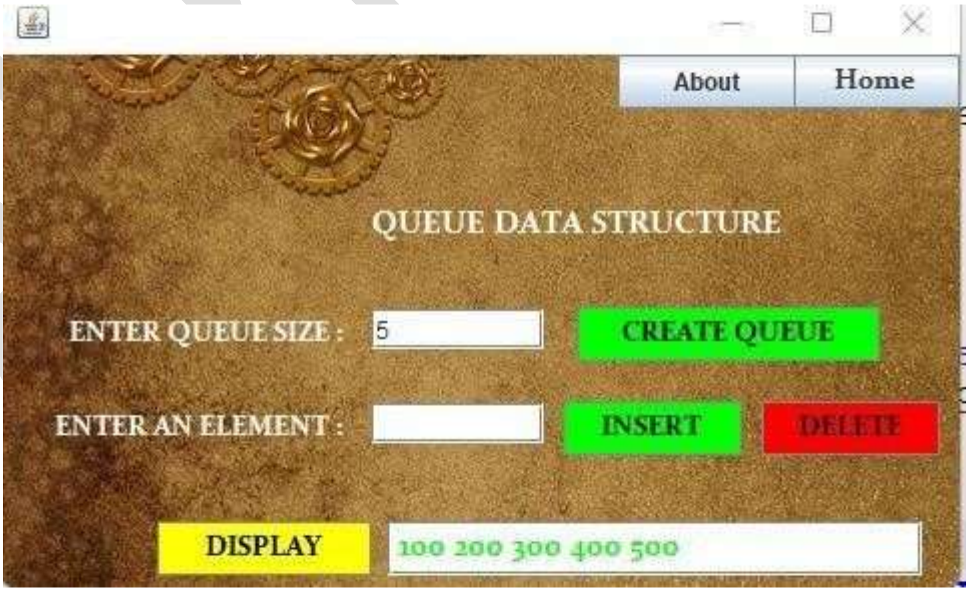
```
possible");

} else

{

++front;

}
```

3. display():It will Display the elements which is present in queue.

```
//display queue elements String msg="";
```

```
if(rear==-1 || front >rear)

{

JOptionPane.showMessageDialog(contentPane,"Queue is empty! Display not
possible"); display.setText("");



for(int i=front;i<=rear;i++)


msg=msg+" "+sq[i];


display.setText(msg);}
```

View of display() operation



ENTER QUEUE SIZE : 5

ENTER AN ELEMENT :

100 200 300 400 500

```
} else

{



{
}
```

About : It will connect to the internet and u can read about Queue Data Structure.

Home :It will take to you into Home page.

Real time Application of an Stack:

-
-

Toilet or Washroom use line :P

- **All the lines similar like above.**
- **Key press sequence in keyboard.**
- **ATM booth line**
- **All the lines similar like above.**

Ticket counter line where people who come first will get his ticket first. Bank line where people who come first will done his transaction first. •

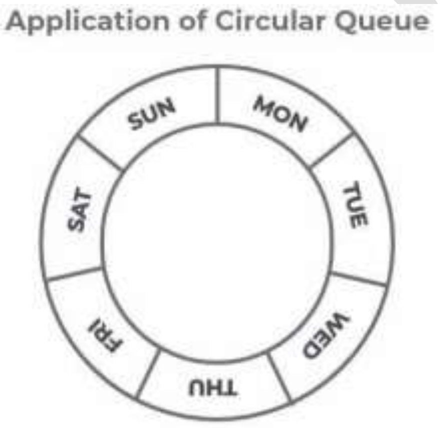
4. CIRCULAR QUEUE:

What is CIRCULAR QUEUE?

A Circular Queue is a special version of queue where the last element of the queue is connected to the first element of the queue forming a circle. The operations are performed based on FIFO (First In First Out) principle. It is also called 'Ring Buffer'**.**

What is the need of a circular queue?

There was one limitation in the array implementation of Queue. If the rear reaches to the end position of the Queue then there might be possibility that some vacant spaces are left in the begi nning which cannot be utilized. So, to overcome such limitations, the concept of the circular queue was introduced.

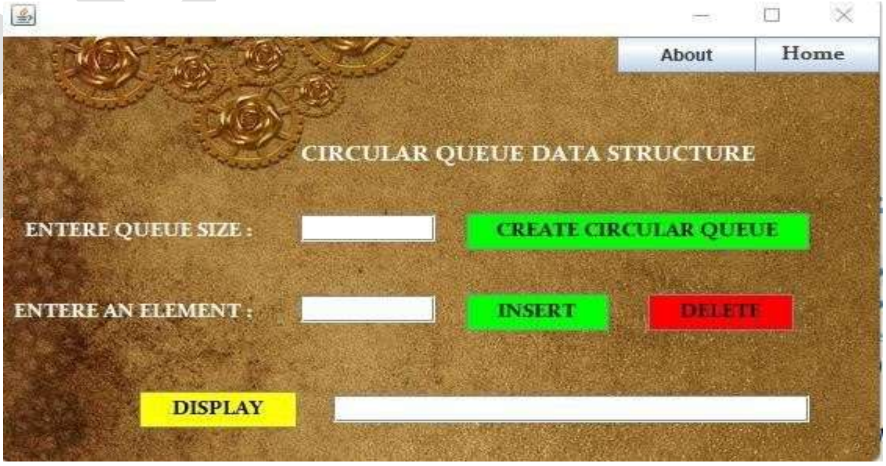


WORK PROCEDURE OF QUEUE:

1.insert()

2.delet()

3.display()



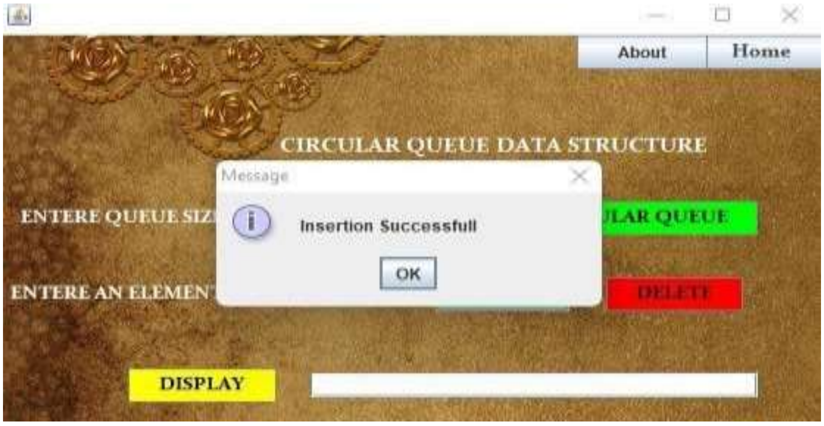
Example of Circular Queue of it will work.

View of Circular Queue Data Structure

1. insert():it will insert a new element in Circular Queue //inserting into circular queue if(count == size && count !=0)//to display a message if queue is full

{

```
JOptionPane.showMessageDialog(contentPane,"Boss CircularQueue is full! Insertion not possible"); elem.setText("");

else if(count >=0)

int elem1 = Integer.valueOf(elem.getText());//asking user for the element

rear=(rear+1)%size; if(rear==-1)

JOptionPane.showMessageDialog(contentPane,"CircularQueue is not created! Create CircularQueue"); elem.setText("");

cq[rear] = elem1;//inserting element at rear end count++;

JOptionPane.showMessageDialog(contentPane,"Insertion Successfull");

elem.setText("");}}
```



Insertion Successfull

```
}

{
```

```
{



} else
{
```

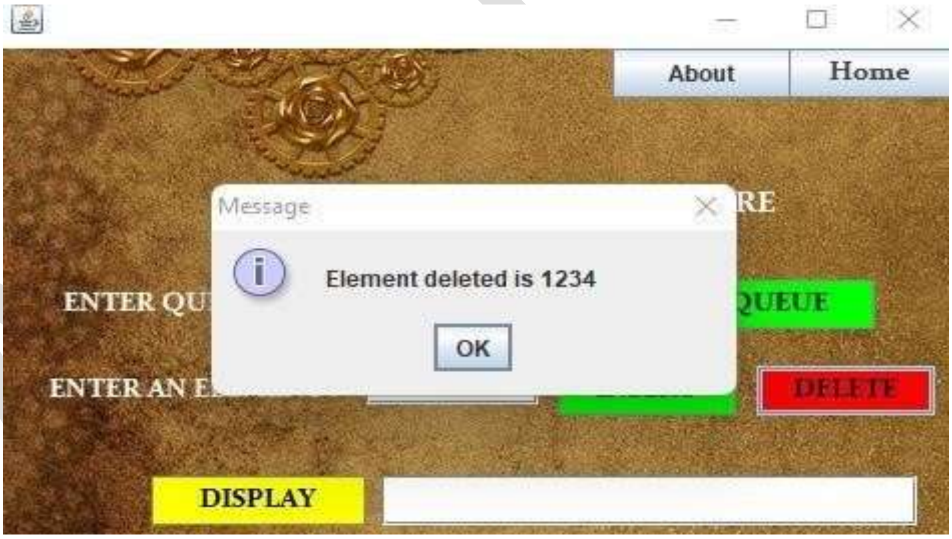
View of insert() operation:

2. delete():it will delete an element at front.

```
//delete
```

```
//delete from circular queue
if(count== 0)
{
display.setText("");
JOptionPane.showMessageDialog(contentPane,"CircularQueue is empty! Deletion not possible");

JOptionPane.showMessageDialog(contentPane,"Element deleted is "+cq[front]);
front=(front+1)%size; count--;
```

View of delete() operation



Message

Element deleted is 1234

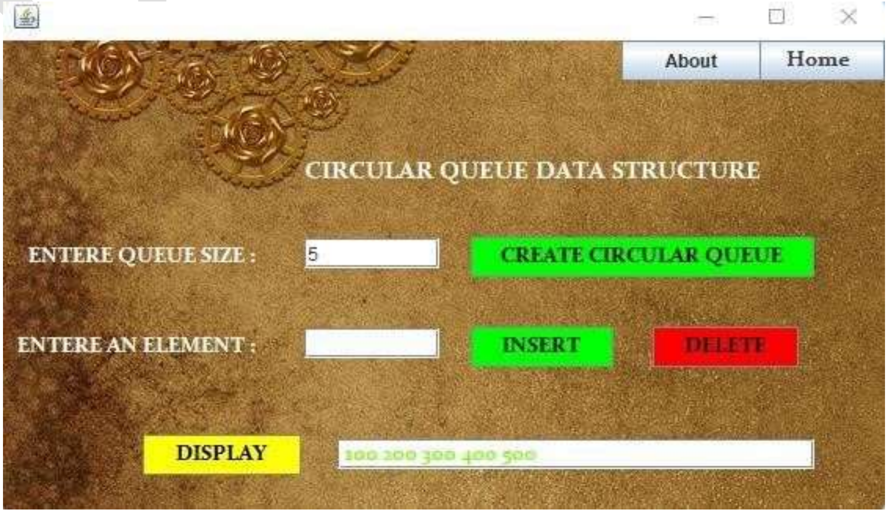
```
} else

{



}
```

3. display():It will Display the elements which is present in queue.

```
//display

//display circular queue contents int
```

```
f1=front;

String msg="";

JOptionPane.showMessageDialog(contentPane,"CircularQueue is empty! Display not possible"); display.setText("");

for(int i=1;i<=count;i++)

msg=msg+" "+cq[f1]; f1=(f1+1)%size;

display.setText(msg);}
```

View of display() operation



ENTERE QUEUE SIZE : 5

ENTERE AN ELEMENT :

100 200 300 400 500

```
if(count == 0)

{
} else

{


{


}
```

About: It will connect to the internet and u can read about Circular queue Data Structure.

Home :It will take to you into Home page.

Real-time Applications of Circular Queue:

- Months in a year: Jan _ Feb _ March _ and so on up to Dec- Jan _ . . .
- Eating: Breakfast _ lunch _ snacks _ dinner _ breakfast _ . . .
- Traffic Light is also a real-time application of circular queue.

5. Singly Linked List:

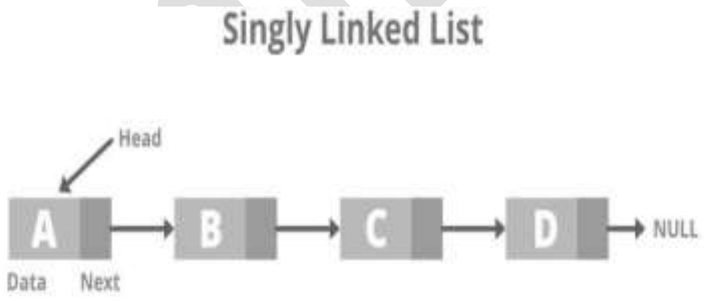
What is Singly Linked List?

A singly linked list is a type of linked list that is unidirectional, that is, it can be traversed in only one direction from head to the last node (tail). Each element in a linked list is called a node. A single node contains data and a pointer to the next node which helps in maintaining the structure of the list.

Why do we need Singly Linked List?

Singly linked list is preferred when we need to save memory and searching is not required as pointer of single index is stored. If we need better performance while searching and memory is not a limitation in this case doubly linked list is more preferred

Example of Singly Linked List how it working.



Work Procedure of Singly Linked List:

1.insert rear()

3.insert at position()

4. delete rear()

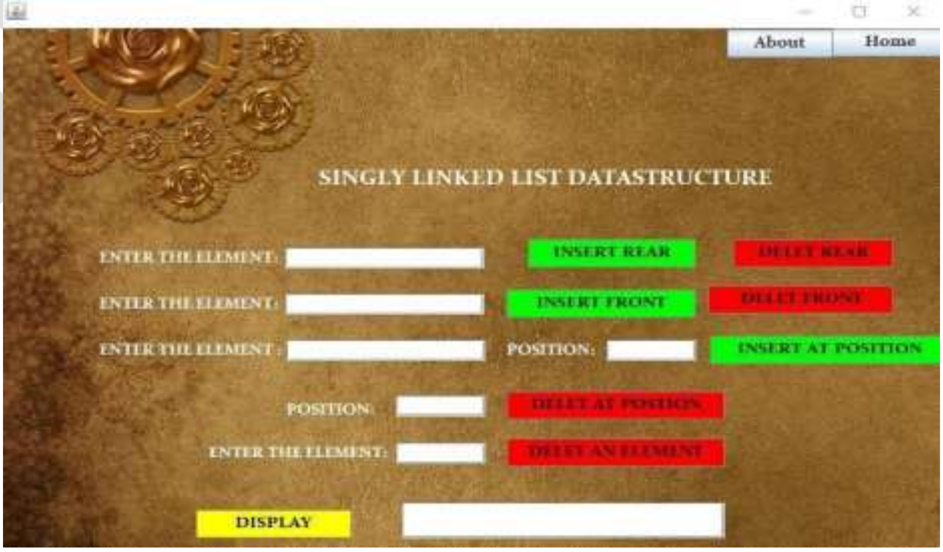
5.delete front()

6.delete at position()

7.delete an element()

8.display()

View of Singly linked List operation



2.insert front()

1. insert rear(): **it will insert an element at the rear end of linked list.**

//INSERT REAR

```
Node temp=null; int elem = Integer.valueOf(element1.getText());//asking
user for the element Node newnode = new Node();//creating a new node
```

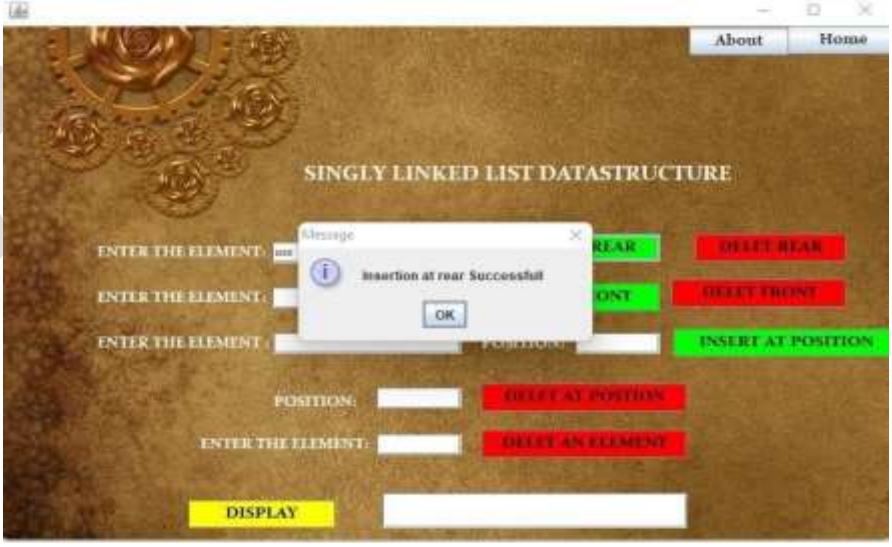
```
first = newnode; }
```

```
temp = first; while(temp.link
```

```
temp = temp.link;}
```

```
temp.link=newnode; }
JOptionPane.showMessageDialog(contentPane,"Insertion at rear Successfull");
element1.setText("")
```

View of insert rear operation.



Message

Insertion at rear Successfull

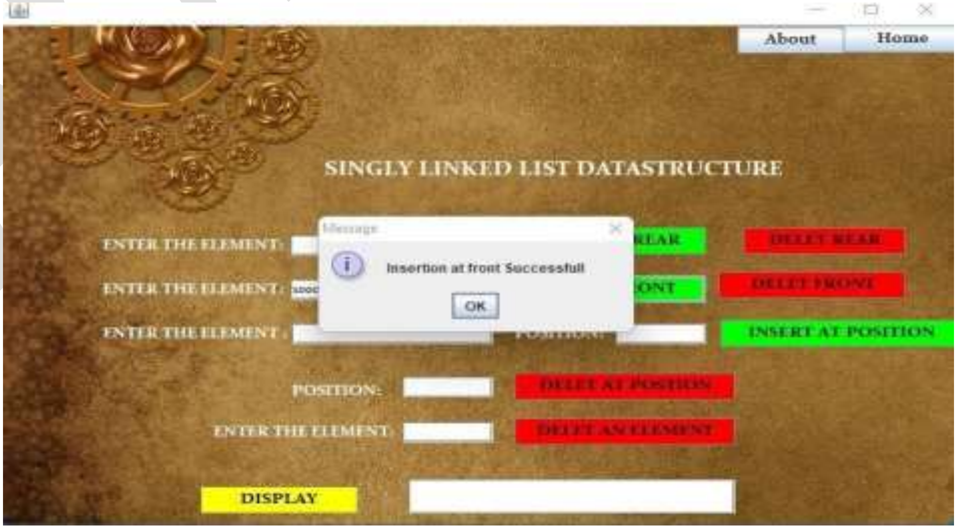
```
newnode.data = elem;//inserting the element into new node's data part
newnode.link = null; if(first == null)//insertion logic
```

```
{

 else {

!= null) {
```

2. insert front():it will insert an element at front in linked list.

```
//INSERT FRONT

int elem = Integer.valueOf(element2.getText());//asking user for the element

Node newnode = new Node();//creating a new node newnode.data =

first = newnode;

newnode.link = first; first

= newnode;

JOptionPane.showMessageDialog(contentPane,"Insertion at front Successfull");

element2.setText("");
```

View of insert front operation



Insertion at front Successfull

```
elem;//inserting the element into new node's data part newnode.link = null;

if(first == null)//insertion logic

{


} else

{



}
```

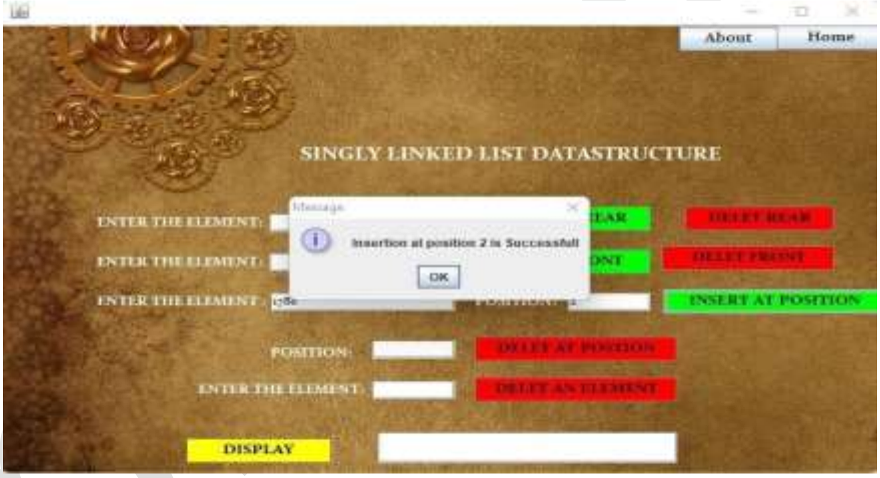
3. insert at position():it will insert an element at particular position.

```
//insert at position

Node temp=null; int
```

```
count = 1;

int elem = Integer.valueOf(element3.getText());//asking user for the element

Node newnode = new Node();//creating a new node newnode.data =

elem;//inserting the element into new node's data part newnode.link = null;

int pos = Integer.valueOf(pos1.getText()); if(first == null)//insertion logic


JOptionPane.showMessageDialog(contentPane,"Linked List doesn't exist!");

element3.setText(""); pos1.setText("");





newnode.link = first; first


JOptionPane.showMessageDialog(contentPane,"Insertion at position "+pos+" is Successfull");

element3.setText(""); pos1.setText("");


{ temp = first;

while(temp.link != null)

{
{




} else if(pos ==

1)

{
```

```
= newnode;



} else




count++; if(count

== pos) {

newnode.link = temp.link; temp.link

= newnode;

JOptionPane.showMessageDialog(contentPane,"Insertion at position "+pos+" is Successfull");

element3.setText(""); pos1.setText(""); return;
```

```
}

temp = temp.link;


JOptionPane.showMessageDialog(contentPane,"Invalid position");

element3.setText(""); pos1.setText("");
```



SINGLY LINKED LIST DATASTRUCTURE

Message

Insertion at position 2 is Successfull

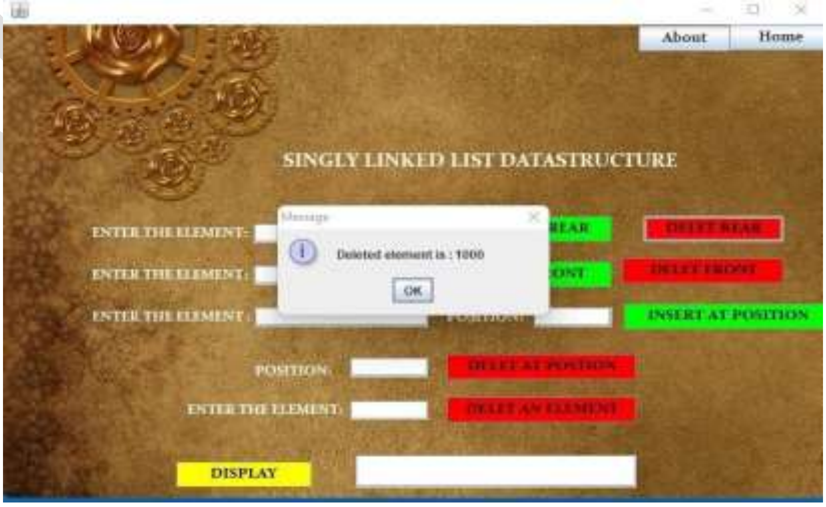
4. delete rear():it will delete an element at rear end

```
//DELETE REAR
                        Node temp;

}



}
```

View of insert at position operation

```
                    if(first == null)//logic to delete node at the rear end
                    {
doesn't  JOptionPane.showMessageDialog(contentPane,"LinkedList exist!");}

                    else if(first.link == null)
                    {
                        JOptionPane.showMessageDialog(contentPane,"Deleted
element is : "+first.data);
                           first = null;
                           }
                           else
                           temp = first;
                           while(temp.link.link != null)
                           {
                           temp = temp.link;
                           }
                           JOptionPane.showMessageDialog(contentPane,"Deleted
element is : "+temp.link.data);
                           temp.link = null;
```

View of Delete rear operation



Message

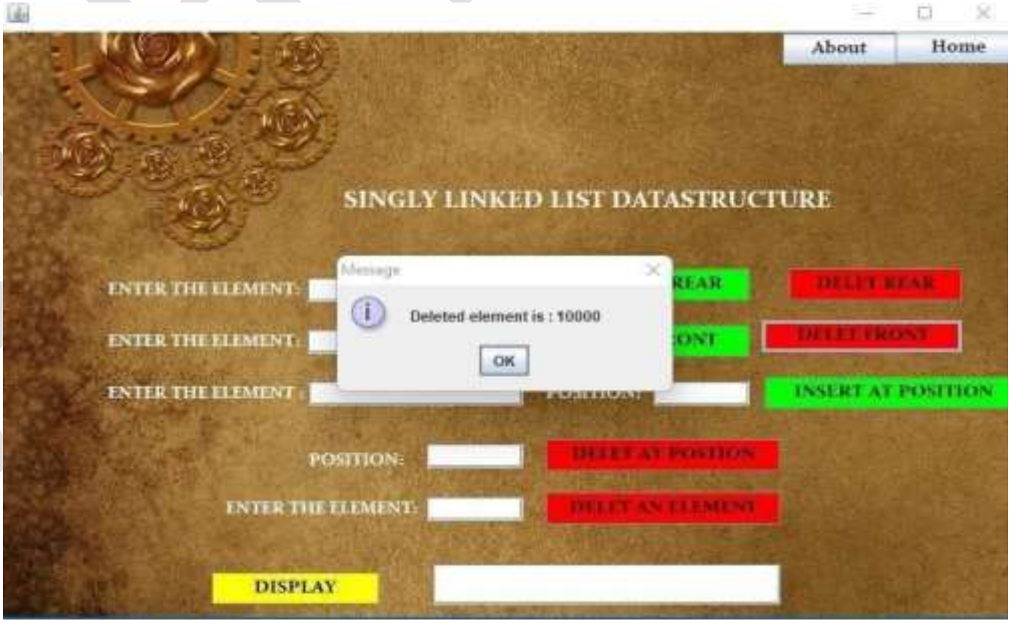
Deleted element is : 1000

OK

5. delete front():it will delete an element at front end.

```
//DELETE FRONT
if(first == null)//logic to delete node at the rear  end
```

```
{

JOptionPane.showMessageDialog(contentPane,"LinkedList doesn't exist!");


else if(first.link == null)


JOptionPane.showMessageDialog(contentPane,"Deleted element is : "+first.data); first



JOptionPane.showMessageDialog(contentPane,"Deleted element is : "+first.data); first
= first.link;
```



Deleted element is : 10000

```
}


{

= null;
```

```
} else {



}
```

View of Delete front operation

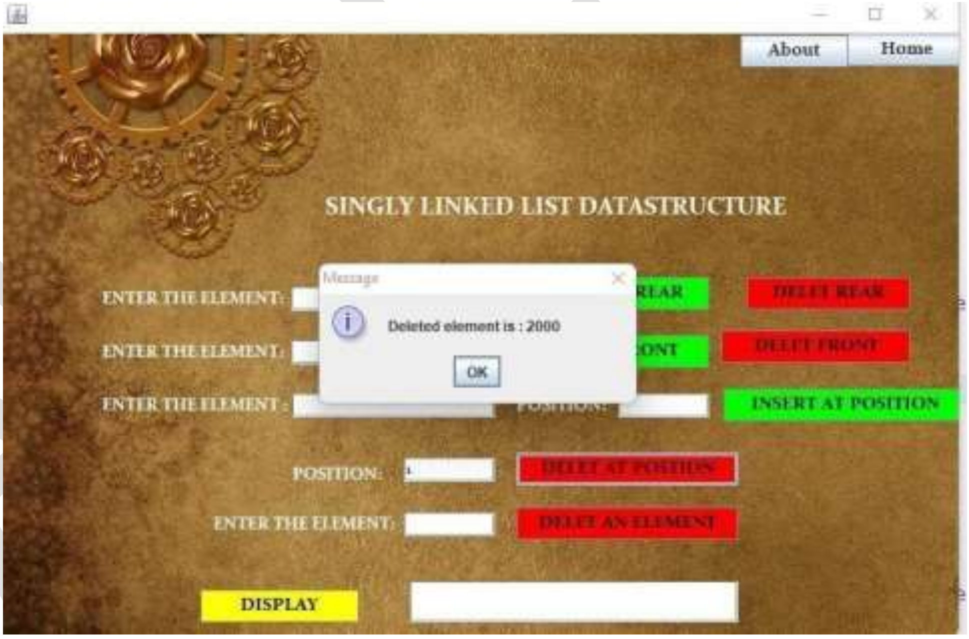
6. delete at position():it will delete an element at particular position.

```
//DELETE AT POSITION  Node temp=null;  int count = 1;  int pos =

Integer.valueOf(pos2.getText()); if(first == null)//Logic to delete an element at a position

{
```

```
JOptionPane.showMessageDialog(contentPane,"Linked List doesn't exist!"); pos2.setText("");



else if(pos == 1 && first.link == null)


JOptionPane.showMessageDialog(contentPane,"Deleted element is : "+first.data); first=null;

pos2.setText(""); return;


else if(pos == 1 && first.link != null)

{ temp=first;

JOptionPane.showMessageDialog(contentPane,"Deleted element is : "+temp.data); first=temp.link;

pos2.setText(""); temp=null;  return;


{ temp=first; while(temp.link.link != null)

{ count++;  if(count




JOptionPane.showMessageDialog(contentPane,"Deleted element is : "+temp.link.data);
temp.link=temp.link.link; pos2.setText(""); return;

} temp=temp.link;
return;

}



{





}
```

```
} else



== pos)

{




if(temp.link.link == null)

{ count++;  if(count

== pos)

{

JOptionPane.showMessageDialog(contentPane,"Deleted element is : "+temp.link.data);

temp.link=null; pos2.setText(""); return;
}

}

} count++;  if(temp.link.link == null &&

count==pos)



JOptionPane.showMessageDialog(contentPane,"Deleted element is : "+temp.link.data);
temp.link=null; return;




JOptionPane.showMessageDialog(contentPane,"Invalid position");  pos2.setText("");
```

View of Delete at position operation



Message

Deleted element is : 2000

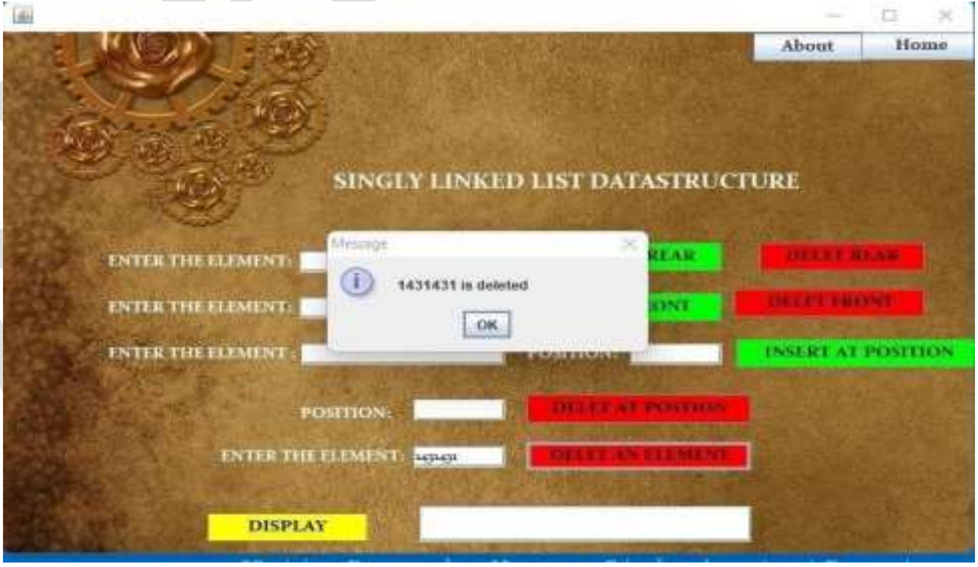
```
{



}

}
```

7. delete an element():it will delete a particular element.

```
//DELETE AN ELEMENT

Node temp = null;  int elem =

Integer.valueOf(element4.getText());  if(first

== null)//Logic to delete an element


System.out.println("Linked List doesn't exist!"); element4.setText("");

return; }  else if(first.data == elem && first.link == null)


JOptionPane.showMessageDialog(contentPane,first.data+" is deleted");

element4.setText(""); first=null; return; }  else if(first.data == elem &&

first.link != null) { temp=first;

JOptionPane.showMessageDialog(contentPane,first.data+" is deleted");

element4.setText(""); first=temp.link; temp=null;  return;


else if(first.data != elem)  { temp=first; if(temp.link.link != null){ while(temp.link.link !=



if(temp.link.data == elem)



{
```

```
{








}

null)

{


{

JOptionPane.showMessageDialog(contentPane,temp.link.da ta+" is deleted");

element4.setText("");  temp.link=temp.link.link; return;

} temp = temp.link;  if(temp.link.link==null)

{


if(elem==temp.link.data)

{ JOptionPane.showMessageDialog(contentPane,temp.link.da ta+" is deleted");

element4.setText(""); temp.link=null; return;

}
} }}
if(temp.link.link == null && temp.link.data == elem)

JOptionPane.showMessageDialog(contentPane,temp.link.da ta+" is deleted");
element4.setText(""); temp.link=null; return;



JOptionPane.showMessageDialog(contentPane,"Element not present!");
element4.setText("");
```

View of Delete a particular element operation



SINGLY LINKED LIST DATASTRUCTURE

Message

1431431 is deleted

OK

ENTER THE ELEMENT: 1431431

```
{



}
}
```

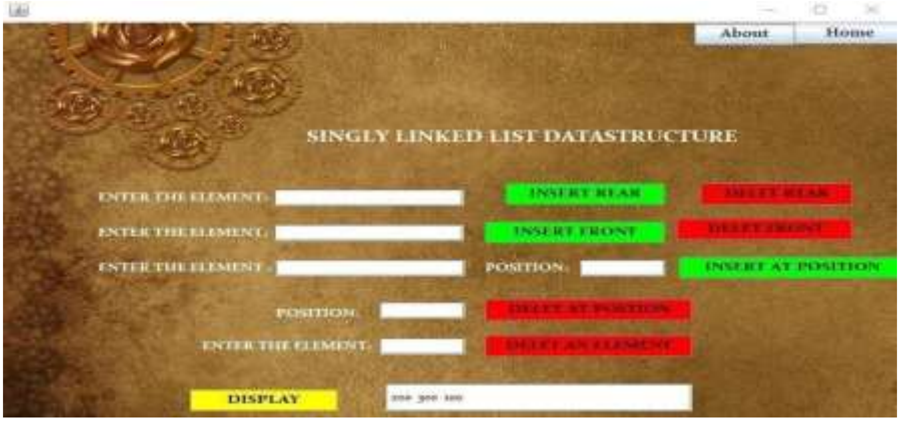
8. display():it will display the contents of Singly Linked List.

//DISPLAY Node

temp; String msg

```
= "";

if(first == null)


JOptionPane.showMessageDialog(contentPane,"LinkedList doesn't exist!");

display.setText("");


else if(first.link == null)


msg=msg+" "+first.data;

} else  { temp = first;  while(temp != null)


msg=msg+" "+temp.data+" "; temp = temp.link;


System.out.println();


display.setText(msg);
```

View of display() operation



```
{
```

}

{

{

}

}

About: It will connect to the internet and u can read about “Singly Linked List” Data Structure.

Home :It will take to you into Home page.

Applications of linked list in the real world:

- Image viewer – Previous and next images are linked and can be accessed by the next and previous buttons.
- Previous and next page in a web browser – We can access the previous and next URL searched in a web browser by pressing the back and next buttons since they are linked as a linked list.
- Music Player – Songs in the music player are linked to the previous and next songs. So you can play songs either from starting or ending of the list.

6. Doubly Linked List:

WHAT IS DOUBLY LINKED LIST?

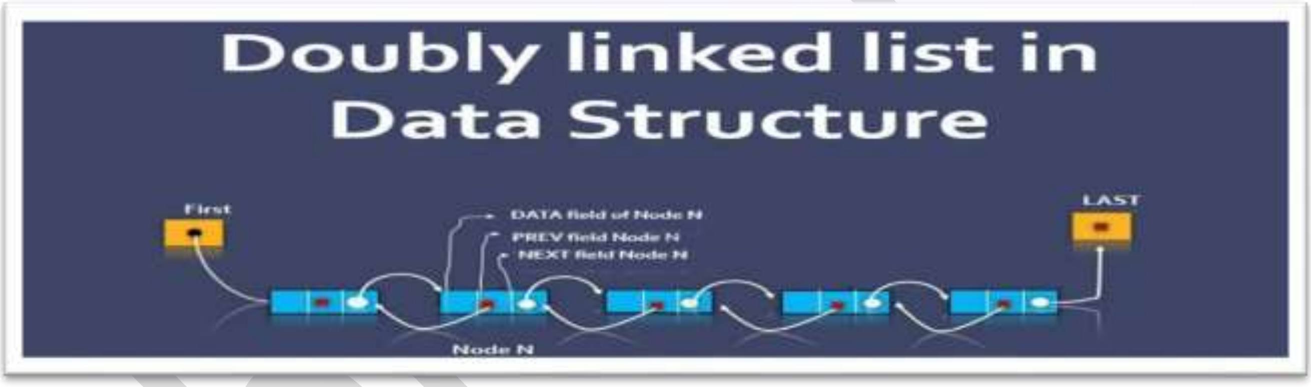
Doubly Linked List is a variation of Linked list in which navigation is possible in both

are the important terms to understand the concept of doubly linked list. Link – Each link

WHY DO WE NEED DOUBLY LINKED LIST?

The most common reason to use a doubly linked list is because it is easier to implement than a singly linked list. While the code for the doubly linked implementation is a little longer than for the singly linked version, it tends to be a bit more "obvious " in its intention, and so easier to implement and debug .

Example of Doubly Linked List how it ? how it will work?



Work procedure of Doubly Linked List.

1.Iinsert rear()

2.insert front()

3.insert at position()

4. delete rear()

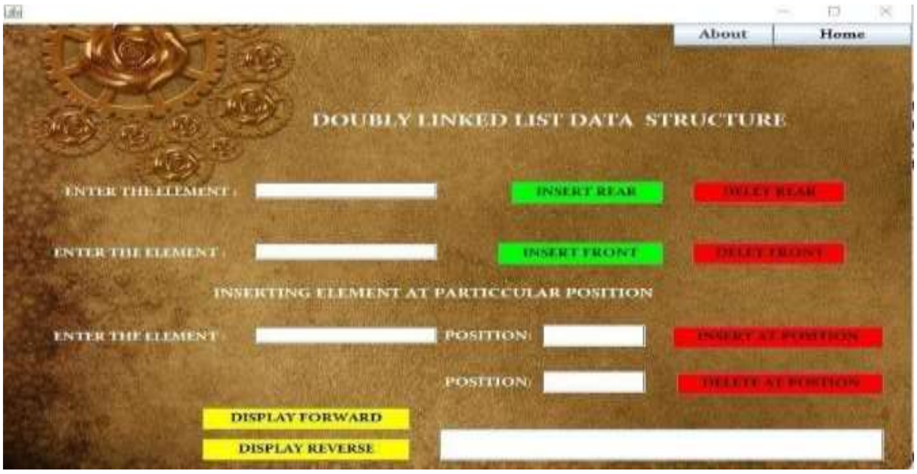
5.delete front()

6.delete at position()

7.display forward()

8.display reverse()

View of Doubly Linked List Operation



ways, either forward and backward easily as compared to Single Linked List. Following

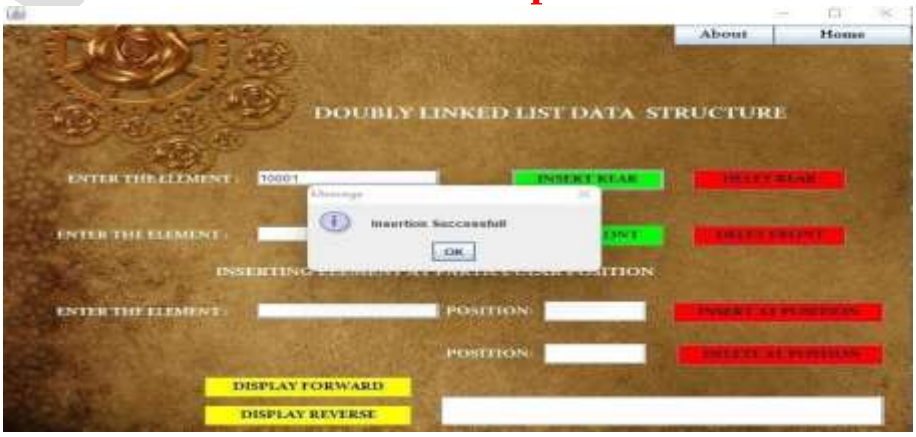
of a linked list can store a data called an element.

1. insert rear():it will insert an element at rear end.

//INSERT REAR

```
Node temp=null; int elem =

Integer.valueOf(element1.getText());//asking user for the element

Node newnode = new Node();//creating a new node newnode.data = elem;//inserting the
element into new node's data part  newnode.prelink = null;  newnode.nextlink = null;
if(first == null)//insertion logic

{ first = newnode;

JOptionPane.showMessageDialog(contentPane,"Insertion

Successfull"); element1.setText(""); } else

{ temp = first; while(temp.nextlink != null)


temp = temp.nextlink;


temp.nextlink=newnode;

newnode.prelink=temp;

JOptionPane.showMessageDialog(contentPane,"Insertion

Successfull"); element1.setText(""); }
```

View of insert rear operation



ENTER THE ELEMENT : 10001

Message

Insertion Successfull

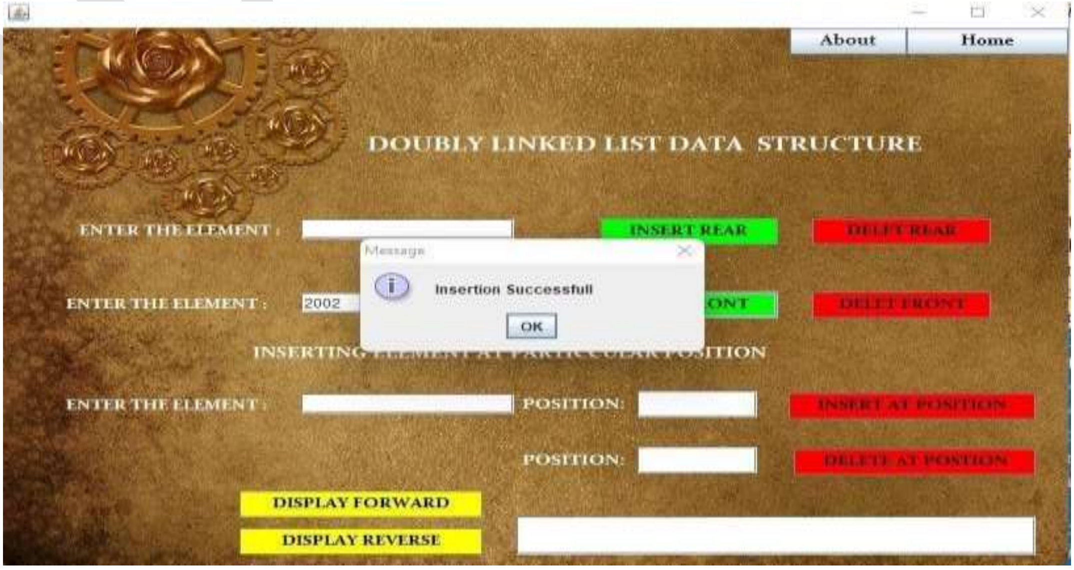
```
{
}
```

2. insert front():it will insert an element at front end.

```
//INSERT FRONT int elem =

Integer.valueOf(element2.getText());//asking user for the element

Node newnode = new Node();//creating a new node newnode.data = elem;//inserting the
```

```
element into new node's data part  newnode.nextlink = null;  newnode.prelink = null;

Successfull"); element2.setText("");

newnode.nextlink = first;  first.prelink =
newnode; first = newnode;
JOptionPane.showMessageDialog(conten
tPane,"Insertion

Successfull"); element2.setText("");
```

View of insert front operation



2002

Insertion Successfull

```
if(first == null)//insertion logic
{ first = newnode; JOptionPane.showMessageDialog(contentPane,"Insertion
} else

{









}
```

3. insert at position():it will insert an element at a particular position.

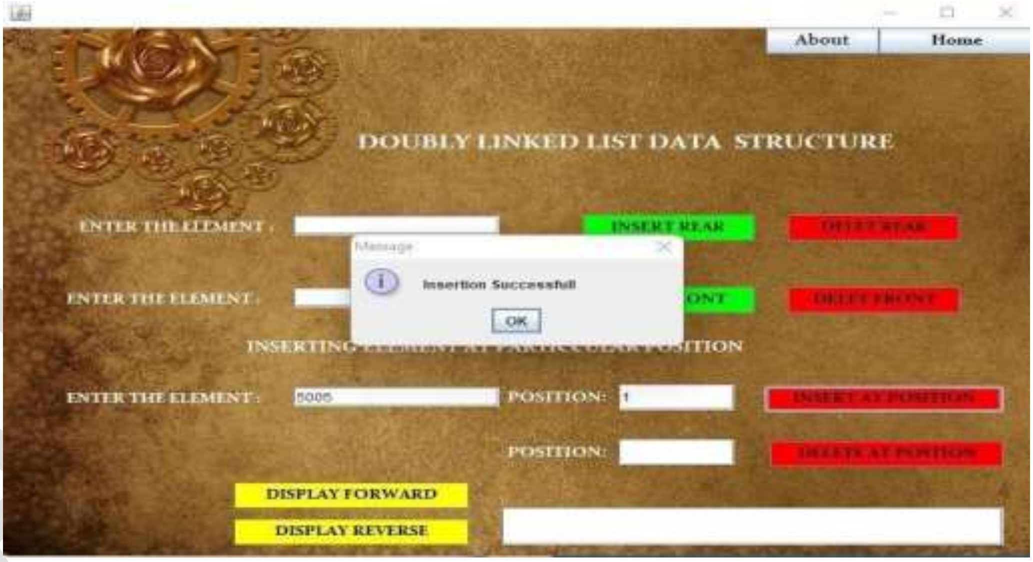
```
//INSERT AT POSITION  Node

temp=null;
```

```
int count = 1; int elem =

Integer.valueOf(element3.getText());//asking user for the element

Node newnode = new Node();//creating a new node newnode.data = elem;//inserting the

element into new node's data part  newnode.nextlink = null;  newnode.prelink = null;

int pos = Integer.valueOf(pos1.getText()); if(first == null)//insertion logic


JOptionPane.showMessageDialog(contentPane,"Doubly

Linked List doesn't exist!"); element3.setText("");

pos1.setText("");

} else if(pos == 1)


newnode.nextlink = first;  first.prelink=newnode;

first = newnode;

JOptionPane.showMessageDialog(contentPane,"Insertion

Successfull"); element3.setText("");  pos1.setText("");


{ temp = first; while(temp.nextlink != null)

{ count++;  if(count

== pos)
{
```

```
{

} else

{

newnode.nextlink=temp.nextlink;  temp.nextlink.prelink=newnode;
temp.nextlink=newnode;  newnode.prelink=temp;
```

```
JOptionPane.showMessageDialog(contentPane,"Insertion Successfull"); element3.setText("");  pos1.setText("");



temp=temp.nextlink;


JOptionPane.showMessageDialog(contentPane,"Invalid position");
element3.setText(""); pos1.setText("");
```

View of insert at particular position.



Insertion Successfull

ENTER THE ELEMENT : 5005

POSITION: 1

```
return;
}


}
```

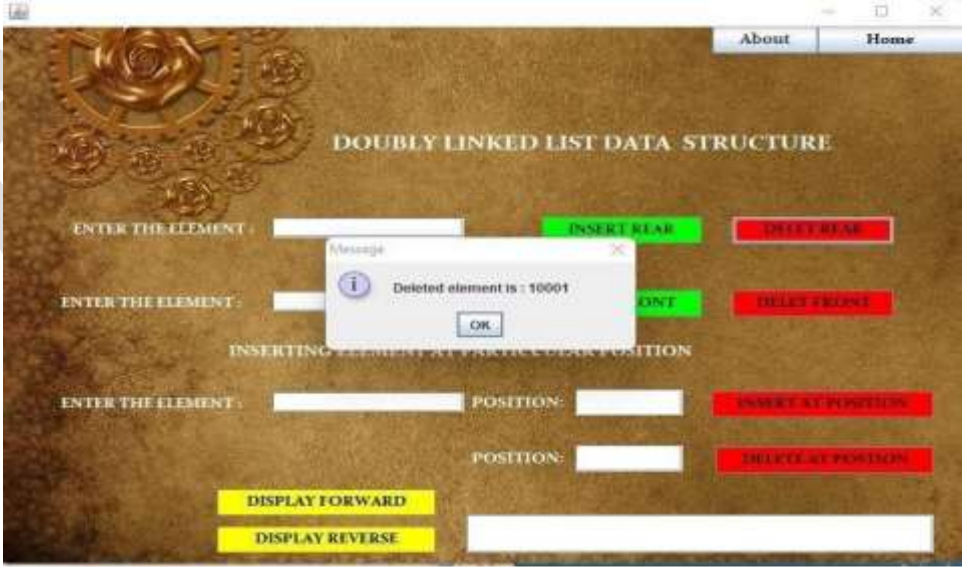
```
}
```

4. **delete rear():** it will delete an element at rear end.

```
//DELETE REAR Node temp;

if(first == null)//logic to delete node at the rear  end

{
```

```
JOptionPane.showMessageDialog(contentPane,"DoublyLinke dList doesn't exist!");


else if(first.nextlink == null) {

JOptionPane.showMessageDialog(contentPane,"Deleted element is : "+first.data); first


} else { temp = first;

while(temp.nextlink.nextlink != null)


temp = temp.nextlink;


JOptionPane.showMessageDialog(contentPane,"Deleted element is :
"+temp.nextlink.data); temp.nextlink = null;      }
```

View of Delete rear operation.



Deleted element is : 10001

```
}



= null;
```

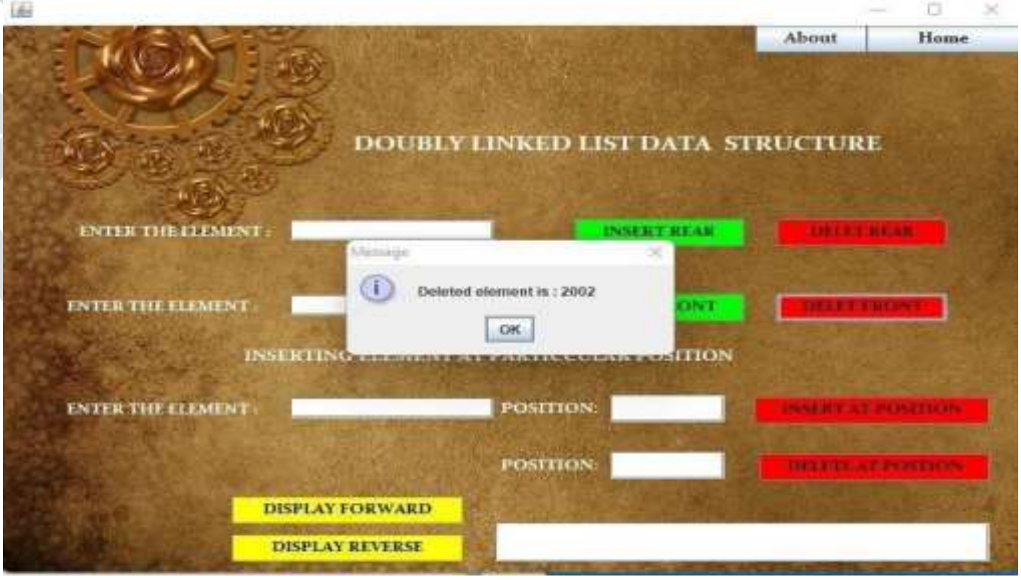
```
{

}
```

5. delete front():it will delete an element an front end.

```
//DELETE FRONT

if(first == null)//logic to delete node at the rear   end
```

```
{

JOptionPane.showMessageDialog(contentPane,"DoublyLinke dList doesn't exist!");


else if(first.nextlink == null)


JOptionPane.showMessageDialog(contentPane,"Deleted element is : "+first.data);





JOptionPane.showMessageDialog(contentPane,"Deleted element is : "+first.data);

first = first.nextlink; first.prelink=null;

}
```

View of delete an element at front.



Deleted element is : 2002

```
}


{
```

```
first = null;   } else

{
```

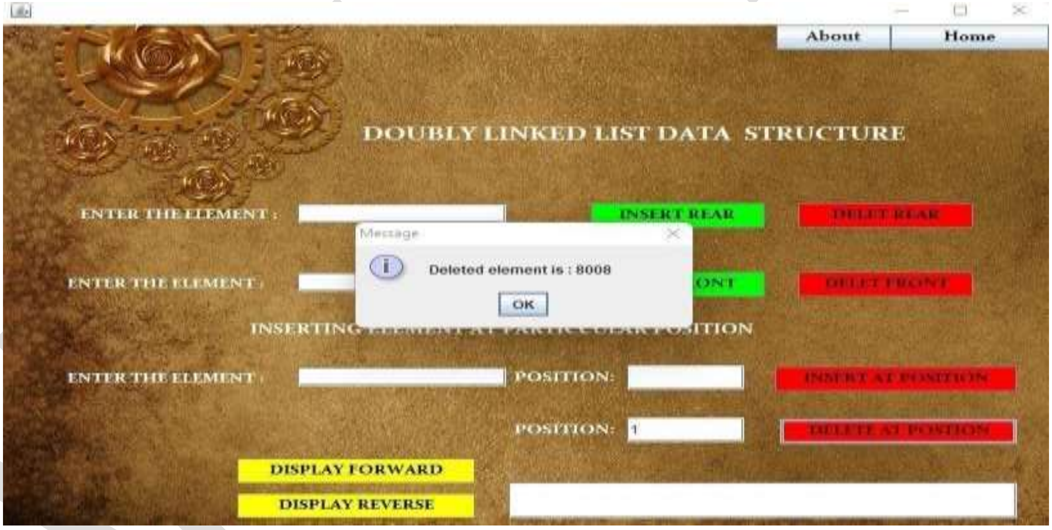
6. delete at position():it will delete an element at particular position.

```
//DELETE AT POSITION  Node

temp=null;

int count = 1;

int pos = Integer.valueOf(pos2.getText()); if(first == null)//Logic to delete an element at a position


JOptionPane.showMessageDialog(contentPane,"DoublyLinke d List doesn't exist!");

pos2.setText(""); return; }  else if(pos == 1 && first.nextlink == null)


JOptionPane.showMessageDialog(contentPane,"Deleted element is : "+first.data);

pos2.setText(""); first=null;  return;


else if(pos == 1 && first.nextlink != null)

{ temp=first;

JOptionPane.showMessageDialog(contentPane,"Deleted element is : "+temp.data);

pos2.setText("");  first=temp.nextlink;  first.prelink=null; temp=null;  return;


{ temp=first;  while(temp.nextlink!=null)

{ temp=temp.nextlink; count++;

if(temp.nextlink!=null && pos == count)

{
```

```
{

}

} else

{
JOptionPane.showMessageDialog(contentPane,"Deleted element is : "+temp.data);
pos2.setText("");

temp.prelink.nextlink=temp.nextlink;  temp.nextlink.prelink=temp.prelink;
return;
}
```

```
if(temp.nextlink==null && pos==count)

{

JOptionPane.showMessageDialog(contentPane,"Deleted element is : "+temp.data);
pos2.setText("");  temp.prelink.nextlink=null; return;





JOptionPane.showMessageDialog(contentPane,"Invalid position");  pos2.setText("");
```

View of delete at particular position operation.



Deleted element is : 8008

POSITION: 1

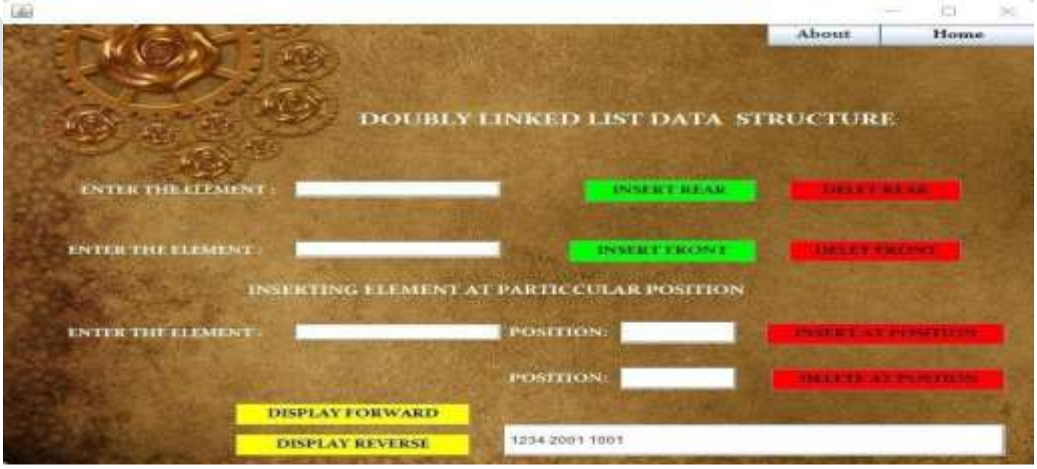
```
}

}

}
```

7. display forward():it will display the content of Doubly Linked list in forward .

```
//DISPLAY FRONT

Node temp;
```

```
String msg=""; if(first == null)

{

JOptionPane.showMessageDialog(contentPane,"DoublyLinke dList doesn't exist! Display not possible"); display.setText("");



else if(first.nextlink == null)



msg=msg+" "+first.data;

} else  { temp = first; while(temp != null)



msg=msg+" "+temp.data;

temp = temp.nextlink;



} display.setText(msg);
```

View of Display forward operation.



1234 2001 1001

```
}



{
```

{

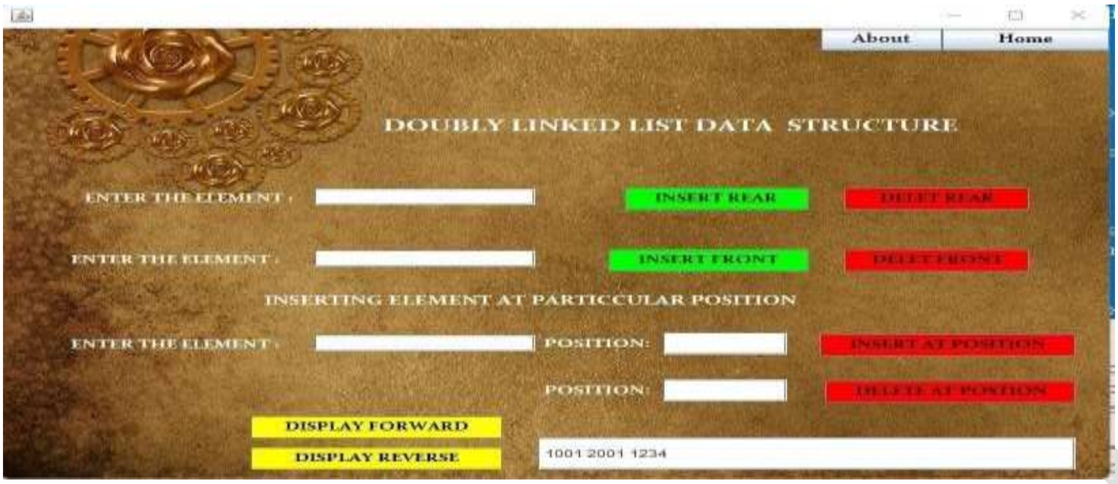
}

8. display reverse():it will Display the contents of Doubly Linked list in Reverse order.

```
//DISPLAY REVERSE Node temp;

String msg=""; if(first == null)
```

```
{

JOptionPane.showMessageDialog(contentPane,"DoublyLinke dList doesn't exist!
Display not possible"); display.setText("");



else if(first.nextlink == null)



msg=msg+" "+first.data;





{ temp = first; while(temp.nextlink != null)



temp = temp.nextlink;



while(temp != null)



msg=msg+" "+temp.data;  temp

= temp.prelink;

}

} display.setText(msg);
}



{
}  else

{

}

{
```

View of Display Reverse operation.



```
1001 2001 1234
```

About: It will connect to the internet and u can read about "Doubly Linked List" Data Structure.

Home :It will take to you into Home page.

Real Time Applications of Doubly Linked List.

- It is used in the navigation systems where front and back navigation is required.
- It is used by the browser to implement backward and forward navigation of visited web pages that is a back and forward button.
- It is also used to represent a classic game deck of cards.
- It is also used by various applications to implement undo and redo functionality.

What are the most common operations performed in linear data structures?

The common possible operations that can be performed in all linear data structures include traversing, insertion, deletion, modification, search operation, and sort operation.

These operations are recognized by different names in different data structures. For example, the

to as enqueue and dequeue operations in

Conclusion:

- In linear data structure, data elements are ordered in a sequential order, with each element connected to the previous and next element.
- Arrays, Linked List, Stack, and Queue are the different types of linear data structures.
- Array elements store in a contiguous memory location but Linked list elements can be stored anywhere in the memory.
- Stack follows Last in First out and Queue follows First in First out.
- Both insert and delete operations inside stack and queue take O(1) time.
- In linear data structures memory is not efficiently utilized as compared to the non-linear data structures.

Future work:

- Handling Exceptions
- Right now project is implemented on primitive data structures in future object type data structure has to be implemented.

insertion and deletion operations are known as Push and Pop operations in Stack, whereas they are referred

Queue. There can be some other operations as well such as merging and the empty operation to check if the data structure is empty or not.