Report For HW2

Author: Abdullah Mesut Güler

Number: 32815522378

Purpose of the Project: This project is built to have necessary methods in order to create a Doubly Linked List. Along with these methods, also linked list manipulation methods, OpacuraL and SacuraL, are developed.

Documentation for the Software:

- To make this project, 'Apache Netbeans IDE 15' and Java version 'jdk19' are used.
- Firstly the **HW2Interface.java** is created with the following method declerations. These methods will be implemented in LinkedList.java class.

- public void **Insert**(int newElement, int pos) **throws** Exception;
- public int **Delete**(int pos) **throws** Exception;
- public void **LinkReverse**();
- public void **SacuraL**();
- public void **OpacuraL**();
- public void **Output**();
- public void **ReverseOutput**();
- @Override
  public String **toString**();
- public Exception **LinkedListException**();

- To create the nodes in the Doubly Linked List, the DoubleLinkNode.java class is created. This class has 3 fields and constructor.
- **Element:** The data will be kept in the node.
- **Right:** Storing the next nodes reference.
- **Left:** Storing the previous nodes reference.
- With the references stored in **Left** and **Right**, we will be able to move inside the Linked List.
- To create a node using DoubleLinkNode.java class, the constructor below has to be used. The constructor will need you to enter values and references for the node that to be created.
- public **DoubleLinkNode**(int **Element**, DoubleLinkNode **left**, DoubleLinkNode **right**){

```
        this.Element = Element;
        this.left = left;
        this.right = right;
    }
```

- The following doubly linked list methods are created in the LinkedList.java class.
- This class has two private field.
- **DoubleLinkNode** top; : This reference will be pointing to the top of the list.
- **DoubleLinkNode** tail; : This reference will be pointing to the end of the list.

- public void **Insert (**int **newElement,** int **pos) throws** Exception

- This method will be used to insert new nodes to the list. It takes two inputs. First one is the **Element** in the node and the pos value. This method will insert the new node after the pos.
- An important note here is that the first node of the Linked List will have 0th index. Thus if you have a linked list consists of 10 nodes, the last node in the list will have 9th index.
- In the method, pos = 0 considered as a separate case. If pos is entered to be 0, it means it will replace the top node. And also if there isn't a node in the list then it will be the first node. Thus this case is also covered in an if-else construction. Code is as shown below.

```
if(pos == 0){ // adding on top of the stack
      if(top == null){ //initiliazing the stack by adding the first element
        top = new DoubleLinkNode(newElement,null,null);
        tail = top;
      }
      else{ // if there is already an initiliazed stack
        DoubleLinkNode newNode = new DoubleLinkNode(newElement,null,dummy0);
        top = newNode;
        dummy0.left = newNode;
      }
    }
```

- Following else block of the code above is for any entered pos values other than '0'. Here the top reference will be assigned to 'dummy0'. Using 'dummy0', with a for loop the entered pos value will be located in the linked list. Inside the same loop a variable 'cnt' will be incremented at each loop. Then this value will be compared to entered pos value. If pos is smaller than 'cnt' it means that the entered pos is out of the index of the linked list. For this case a exception will be thrown.
- Inserting the new node at the end of the list treated as a seperate case again. Code is as shown below.

```
else{
      int cnt = 0;
      for(int i = 0; i< pos; i++){
        if(!(dummy0.right == null)){
        dummy0 = dummy0.right;
        cnt += 1;
        }
      }
      if(cnt != pos){
        throw new Exception("The entered pos is out of index.");
    }
    else{
        if(dummy0.right == null){ //means updating the tail
            DoubleLinkNode newNode = new DoubleLinkNode(newElement,dummy0,null);
            dummy0.right = newNode;
            tail = newNode;
        }
        else{ // somewhere in the middle
            DoubleLinkNode newNode = new DoubleLinkNode(newElement,dummy0,dummy0.right);
            dummy0.right.left = newNode;
            dummy0.right = newNode;
        }
    }
}
```

- public int **Delete**(int **pos**) **throws** Exception
- With the **Delete** the node at the entered pos will be deleted from the list and the deleted nodes element will be returned to the user.
- The pos = 0 case is treated seperately again. In the if(pos == 0) block the second nodes reference in the list will be assigned to top then this nodes **left** field will assigned to be null. Code is as shown below.

```
if(pos == 0){
        content = top.Element;
        top = dummy0.right;
        top.left = null;
    }
```

- For any other pos value the else block will work. Similar to the **Insert** method, the top reference will be assigned to dummy0. Then in a for loop dummy will be iterated the entered pos node. Again at each loop cnt will be incremented. Then this cnt will be compared to the entered pos. If they're not equal then it will throw an exception and it will return -1.
- If the entered pos is valid then the following else block work. Inside else block, if the case is deleting the tail it will treated seperately. If not the following else block will delete the node which is somewhere in the middle of the list. Code is as shown below.

```
else{
    for(int i = 0; i < pos ; i++){
          if(!(dummy0.right == null)){
              dummy0 = dummy0.right;
              cnt++;
          }
      }
      if(cnt != pos){
          content = -1;
          throw new Exception("The entered pos is out of index. Returning -1");
        }
      else{
          if(dummy0.right == null){ //means deleting the tail
              content = tail.Element;
              tail = dummy0.left;
              tail.right = null;
          }
          else{ // deleting from somewhere in the middle
              content = dummy0.Element;
              dummy0.right.left = dummy0.left;
              dummy0.left.right = dummy0.right;
              dummy0.left = null;
              dummy0.right = null;
          }
        }
  }
```

- public void **LinkReverse**()

- This method is for reversing the order of the nodes in the list.
- Top will be assigned to dummy0 and tail will be assigned to dummy1. Inside while loop these dummys will iterated to each other to meet in the middle. At each loop dummy0.Element and dummy1.Element will be exchanged.
- If there are even number of nodes in the list, the last Exchange operation will be performed in the “ **if(dummy0.right == dummy1)** ”block. But if there are odd number of nodes in the list then the last Exchange operation will be performed in “**if(dummy1.left == dummy0.right)**“ block. The middle block will remained as the same. In each loop

"if(dummy0.right == dummy1)" will be checked. If it is true it will break from while loop. Code is as shown below.

```
while(!(dummy1.left == dummy0.right)){
      if(dummy0.right == dummy1){
         break;
      }
      tmp0 = dummy1.Element;
      dummy1.Element = dummy0.Element;
      dummy0.Element = tmp0;
      dummy1 = dummy1.left;
      dummy0 = dummy0.right;
    }
    if(dummy1.left == dummy0.right){
      tmp0 = dummy1.Element;
      dummy1.Element = dummy0.Element;
      dummy0.Element = tmp0;
    }

    if(dummy0.right == dummy1){
      tmp0 = dummy1.Element;
      dummy1.Element = dummy0.Element;
      dummy0.Element = tmp0;
    }
```

- public void **SacuraL**()

- This method will count consecutive nodes with the same **Element.** Then counted number of nodes will be inserted in the list.
- **Eg.** If the list is 1 1 1 3 2 2 2 4 4, the output of SacuraL() will look like, 1 3 3 1 2 3 4 2
- Top will be assigned to dummy0 and top.right will be assigned to dummy1. Then buffer node is created with the **Element** = 999.
- Inside the while loop will continue till dummy1 == null. At each iteration dummy1.Element will be compared to dummy0.Element. If they're equal then it will enter the else block. Inside the if block cnt will be incremented and dummy1 will be iteretad to next node. This way consecutive nodes with the same **Element** will be counted.
- If they're not equal then it will enter the if block. Inside the if block a new node with **Element**=(cnt+1) will be created. Created nodes will be connected to dummy0 and right is

connected to dummy1. Then dummy1 will be assigned to dummy0 and dummy1 will iterated to the next node.

- After while loop the buffer node will be released from the list. The buffer node is used because without it at end of the list, when dummy1 == null, while loop stops. This causes the last node of the list to be left not processed. To avoid that the buffer node is created. Code is as shown below.

```
DoubleLinkNode bufferNode = new DoubleLinkNode(999,tail,null);
    tail.right = bufferNode;
    tail = bufferNode;
    while(!(dummy1 == null)){
      if(!(dummy1.Element==dummy0.Element)){

        DoubleLinkNode newNode = new DoubleLinkNode(cnt+1,dummy0,dummy1);
        dummy0.right = newNode;
        dummy1.left = newNode;
        dummy0 = dummy1;
        dummy1 = dummy1.right;
        cnt = 0;
      }
      else{
        cnt++;
        dummy1 = dummy1.right;
      }
    }
    // lose the buffer
    tail = bufferNode.left;
    tail.right = null;
    bufferNode.left = null;
```

- public void **OpacuraL**()
- **OpacuraL** is for reversing the affects of SacucaL.
- **Eg.** If the list is " 1 2 3 2 4 5", then the output of **OpacuraL** will be " 1 1 3 3 4 4 4 4 4".
- Top will be assigned to dummy0 and top.right will be assigned to dummy1. Similar to **SacuraL**, a buffer node will be created for the same reasons.
- The while loop will contiune until dummy0 == null, and at each loop if(dummy1 == null) will be checked. If true it will break from the loop.

- Then it will check if(dummy1.Element == 1). If it's true then dummy1 will be iterated 2 nodes to the right for its new position. if(dummy0.right.right == null) will be checked, if true it means we are at the end of the list. And the last node will be just eliminated and the tail will be updated. If not true, then it means we are somewhere in the middle of the list. Then dummy0 will simply be connected to next tuple. And dummy0 will be updated for the next tuple.
- In the following else block, in which case if(dummy1.Element == 1) is not true, a new node will be created with the Element of dummy0. Then dummy0 and new node will be connected to each other. At this point newNode.right is null. Then dummy1.Element will be decremented. Then dummy0 will be iterated to created new node.
- After the while loop, the buffer node is released. This buffer node was created for the same reasons as **SacuraL.** Code is as shown below.

```
// create buffer

        DoubleLinkNode bufferNode = new DoubleLinkNode(999,tail,null);

        tail.right = bufferNode;

        tail = bufferNode;

        while(!(dummy0 == null)){

           if(dummy1 == null) break;

           if(dummy1.Element == 1){

              dummy1 = dummy1.right.right;

              if(dummy0.right.right == null){

                 dummy0.right = null;

                 tail.left = null;

                 tail = dummy0;

                 dummy0 = null;

              }

              else{

                 dummy0.right.right.left = dummy0;

                 dummy0.right = dummy0.right.right;

                 dummy0 = dummy0.right;

              }

           }

           else{

              if(dummy1.Element > 1){

                 DoubleLinkNode newNode = new DoubleLinkNode(dummy0.Element,dummy0,dummy1);
```

```
            dummy1.left = newNode;

            dummy0.right = newNode;

            dummy1.Element--;

            dummy0 = dummy0.right;

        }

    }

}

// lose the buffer

tail = bufferNode.left;

tail.right = null;

bufferNode.left = null;
```

- public void **Output**()
- This method will print out the list starting from the top node to the tail node.
- Top will be assigned to the dummy0. Inside a while loop, till dummy0 == null, it will print the Element of each node. Then dummy0 will be iterated to right node.
- public void **ReverseOutput**()
- Identical operation is performed but this time tail is assigned to the dummy0. This way the list will be printed out in the reverse order, starting from tail to the top node.
- public String **toString**()
- Ordinary toString method. It will simply convert the created linked list object to a string in which each character is Element of the node.
- public Exception **LinkedListException**()
- It will simply throw an exception for linked list boundary issues.

END OF THE REPORT FOR HW2