Manjula Nannuri

Day 5_Assignment

Write a class CustomLinkedList that implements a singly linked list with methods for InsertAtBeginning, InsertAtEnd, InsertAtPosition, DeleteNode, UpdateNode, and DisplayAllNodes. Test the class by performing a series of insertions, updates, and deletions.

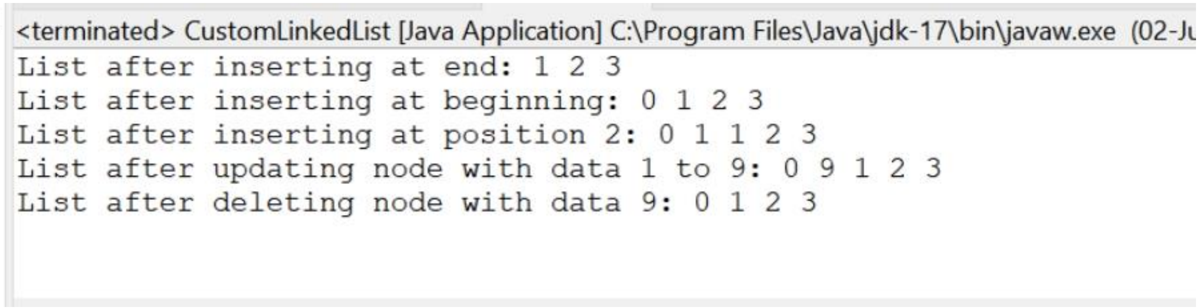
```
package sortings;
public class CustomLinkedList {
class Node {
int data;
Node next;
Node(int data) {
this.data = data;
this.next = null;
}
}
private Node head;
public CustomLinkedList() {
this.head = null;
}
public void insertAtBeginning(int data) {
Node newNode = new Node(data);
newNode.next = head;
head = newNode;
}
public void insertAtEnd(int data) {
Node newNode = new Node(data);
if (head == null) {
head = newNode;
return;
```

```
}
Node temp = head;
while (temp.next != null) {
temp = temp.next;
}
temp.next = newNode;
}
public void insertAtPosition(int data, int position) {
Node newNode = new Node(data);
if (position == 0) {
newNode.next = head;
head = newNode;
return;
}
Node temp = head;
for (int i = 0; i < position - 1 && temp != null; i++) {
temp = temp.next;
}
if (temp == null) {
throw new IllegalArgumentException("Position out of bounds");
}
newNode.next = temp.next;
temp.next = newNode;
}
public void deleteNode(int key) {
Node temp = head, prev = null;
if (temp != null && temp.data == key) {
head = temp.next;
```

```
return;

}

while (temp != null && temp.data != key) {

prev = temp;

temp = temp.next;

}

if (temp == null) {

return;

}

prev.next = temp.next;

}

public void updateNode(int oldData, int newData) {

Node temp = head;

while (temp != null && temp.data != oldData) {

temp = temp.next;

}

if (temp != null) {

temp.data = newData;

}

}

public void displayAllNodes() {

Node temp = head;

while (temp != null) {

System.out.print(temp.data + " ");

temp = temp.next;

}

System.out.println();

}
```

```
public static void main(String[] args) {

CustomLinkedList list = new CustomLinkedList();

list.insertAtEnd(1);

list.insertAtEnd(2);

list.insertAtEnd(3);

System.out.print("List after inserting at end: ");

list.displayAllNodes();

list.insertAtBeginning(0);

System.out.print("List after inserting at beginning: ");

list.displayAllNodes();

list.insertAtPosition(1, 2);

System.out.print("List after inserting at position 2: ");

list.displayAllNodes();

list.updateNode(1, 9);

System.out.print("List after updating node with data 1 to 9: ");

list.displayAllNodes();

list.deleteNode(9);

System.out.print("List after deleting node with data 9: ");

list.displayAllNodes();

}

}
```

OUTPUT:



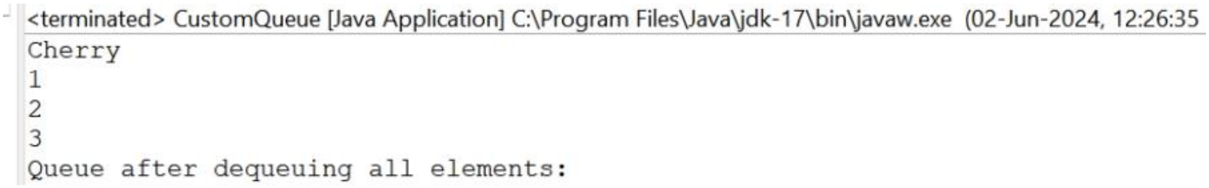
```
<terminated> CustomLinkedList [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (02-Ju
List after inserting at end: 1 2 3
List after inserting at beginning: 0 1 2 3
List after inserting at position 2: 0 1 1 2 3
List after updating node with data 1 to 9: 0 9 1 2 3
List after deleting node with data 9: 0 1 2 3
```

2) Develop a CustomQueue class with methods for Enqueue, Dequeue, Peek, and IsEmpty. Show how your queue can handle different data types by enqueuing strings and integers, then dequeuing and displaying them to confirm FIFO order.

```
package sortings;
import java.util.NoSuchElementException;
public class CustomQueue <T> {
private Node<T> front;
private Node<T> rear;
private static class Node<T> {
private T data;
private Node<T> next;
public Node(T data) {
this.data = data;
}
}
public CustomQueue() {
this.front = null;
this.rear = null;
}
public void Enqueue(T data) {
Node<T> newNode = new Node<>(data);
if (rear != null) {
rear.next = newNode;
}
rear = newNode;
if (front == null) {
front = newNode;
}
}
public T Dequeue() {
if (IsEmpty()) {
throw new NoSuchElementException("Queue is empty");
}
T data = front.data;
front = front.next;
if (front == null) {
rear = null;
}
return data;
}
public T Peek() {
if (IsEmpty()) {
throw new NoSuchElementException("Queue is empty");
}
return front.data;
}
public boolean IsEmpty() {
return front == null;
}
public void DisplayQueue() {
Node<T> current = front;
while (current != null) {
System.out.print(current.data + " ");
current = current.next;
}
```

```
System.out.println();

}

public static void main(String[] args) {

CustomQueue<Object> queue = new CustomQueue<>();

queue.Enqueue("Apple");

queue.Enqueue("Banana");

queue.Enqueue("Cherry");

queue.Enqueue(1);

queue.Enqueue(2);

queue.Enqueue(3);

System.out.println("Queue after enqueuing elements:");

queue.DisplayQueue();

System.out.println("Dequeuing elements from the queue:");

while (!queue.IsEmpty()) {

System.out.println(queue.Dequeue());

}

System.out.println("Queue after dequeuing all elements:");

queue.DisplayQueue();

}

}
```

OUTPUT:



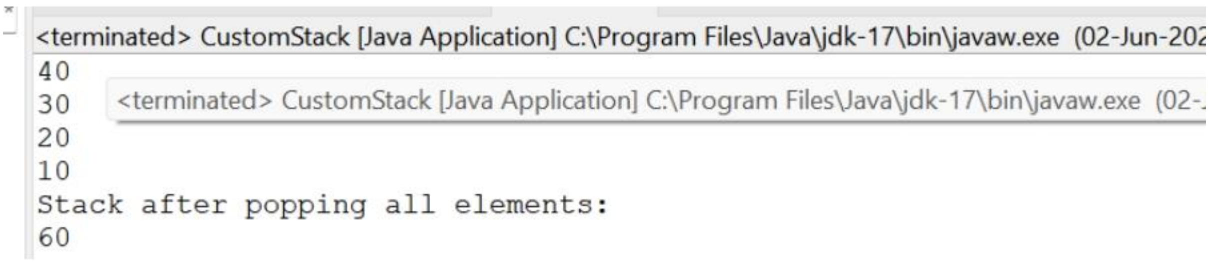
```
<terminated> CustomQueue [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe  (02-Jun-2024, 12:26:35
Cherry
1
2
3
Queue after dequeuing all elements:
```

Create a CustomStack class with operations Push, Pop, Peek, and IsEmpty. Demonstrate its LIFO behavior by pushing integers onto the stack, then popping and displaying them until the stack is empty.

```
package sortings;
import java.util.EmptyStackException;
public class CustomStack {
private Node top;
private static class Node {
private int data;
private Node next;
public Node(int data) {
this.data = data;
}
}
public CustomStack() {
this.top = null;
}
public void Push(int data) {
Node newNode = new Node(data);
newNode.next = top;
top = newNode;
}
public int Pop() {
if (IsEmpty()) {
throw new EmptyStackException();
}
int data = top.data;
top = top.next;
return data;
}
public int Peek() {
```

```
if (IsEmpty()) {
throw new EmptyStackException();
}
return top.data;
}
public boolean IsEmpty() {
return top == null;
}
public void DisplayStack() {
Node current = top;
while (current != null) {
System.out.print(current.data + " ");
current = current.next;
}
System.out.println();
}
public static void main(String[] args) {
CustomStack stack = new CustomStack();
stack.Push(10);
stack.Push(20);
stack.Push(30);
stack.Push(40);
stack.Push(50);
System.out.println("Stack after pushing elements:");
stack.DisplayStack();
System.out.println("Popping elements from the stack:");
while (!stack.IsEmpty()) {
System.out.println(stack.Pop());
}

stack.Push(60);

System.out.println("Stack after popping all elements:");

stack.DisplayStack();

}

}
```

OUTPUT:



<terminated> CustomStack [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (02-Jun-202

<terminated> CustomStack [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (02-

```
40
30
20
10
Stack after popping all elements:
60
```