Class: LinkedListStack

This class implements a stack data structure using a linked list. Each element in the stack is represented by a Node.

Inner Class: Node

```
private class Node {
    int data;
    Node next;

    Node(int data) {
        this.data = data;
        this.next = null;
    }
}
```

- **Node**: This inner class represents each node in the linked list.
- int data: Stores the data of the node.
- Node next: Points to the next node in the list.
- **Constructor**: Initializes the node with data and sets the next node to null.

Fields

- Node top: Points to the top node of the stack. Initially, it is null, indicating the stack is empty.

Constructor

```
public LinkedListStack() {
    this.top = null;
}
```

The constructor initializes the stack by setting the top to null.

Methods

1. **Push method**

   ```
   public void push(int data) {
       Node newNode = new Node(data);
       newNode.next = top;
       top = newNode;
   }
   ```

This method adds an element to the top of the stack.

- Creates a new node with the given data.
- Sets the next of the new node to the current top.
- Updates top to point to the new node.

2. **Pop method**

```
public int pop() {
    if (isEmpty()) {
        System.out.println("Stack Underflow");
        return -1;
    }
    int poppedData = top.data;
    top = top.next;
    return poppedData;
}
```

This method removes and returns the top element of the stack.

- Checks if the stack is empty using isEmpty().
- If the stack is empty, prints "Stack Underflow" and returns -1.
- If not empty, stores the data of the top node, updates top to the next node, and returns the stored data.

3. **Peek method**

```
public int peek() {
    if (isEmpty()) {
        System.out.println("Stack is empty");
        return -1;
    }
    return top.data;
}
```

This method returns the top element of the stack without removing it.

- Checks if the stack is empty using isEmpty().
- If empty, prints "Stack is empty" and returns -1.
- If not, returns the data of the top node.

4. **isEmpty method**

```
public boolean isEmpty() {
    return top == null;
}
```

This method checks if the stack is empty.

- Returns true if top is null.
- Otherwise, returns false.

5. **PrintStack method**

```
public void printStack() {
    if (isEmpty()) {
        System.out.println("Stack is empty");
        return;
    }
    Node current = top;
    while (current != null) {
        System.out.print(current.data + " ");
        current = current.next;
    }
    System.out.println();
}
```

This method prints all the elements in the stack from top to bottom.

- Checks if the stack is empty using isEmpty().
- If empty, prints "Stack is empty".
- If not, iterates through the nodes from top to null, printing each node's data.

main Method

```
public static void main(String[] args) {
    LinkedListStack stack = new LinkedListStack();

    // Pushing elements onto the stack
    stack.push(10);
    stack.push(20);
    stack.push(30);
    stack.push(40);
    stack.push(50);

    // Printing the stack elements
    System.out.print("Stack: ");
    stack.printStack();
    // Popping an element from the stack
    System.out.println("Popped Element: " + stack.pop());

    // Peeking at the top element
    System.out.println("Top Element: " + stack.peek());

    // Checking if the stack is empty
    System.out.println("Is Stack Empty? " + stack.isEmpty());

    // Printing the stack elements after pop
    System.out.print("Stack after pop: ");
    stack.printStack();
}
```

This method demonstrates how to use the LinkedListStack class:

- It creates a stack.
- It pushes five elements onto the stack.
- It prints the current stack elements.
- It pops the top element and prints it.
- It peeks at the top element and prints it.
- It checks and prints whether the stack is empty.
- It prints the stack elements after one pop operation.