Java DSA assignment

You are working on a music streaming application called "MusicMaster," and one of the essential features you need to implement is managing user playlists. As a Java developer, you are responsible for creating a Playlist class to handle the collection of songs in a playlist. To efficiently manage the playlist, you decide to use a linked list data structure.

The various methods that you have to implement are:

* Insertion: Methods to add a song at the beginning and end of the playlist.
* Deletion: Methods to remove a song from the playlist.
* Playback: Methods to play the current song, play the next song, and play the previous song.
* Display: Method to display the details of the current playlist, including the title, artist, and duration of each song.
* Shuffle: Method to shuffle the playlist.
* Search: Method to search for a song by its title or artist and play it.
* Custom Playlist: Method to create and play a custom playlist based on specific criteria.

Details for the Tasks:

Create a Song Class:

* Define a Song class with properties: ‘title’ to store the song title, ‘artist’ to store the artist’s name, and ‘duration’ to store the duration of the song.
* Implement a constructor to initialize the song details.

Implement the Playlist Class:

* The Playlist class should use a linked list to manage the songs in the playlist.
* Implement a constructor to initialize the linked list.

Insertion Methods:

* Write methods to add a song at the beginning and end of the playlist.
* Ensure that the linked list is updated accordingly.

Deletion Methods:

* Write methods to remove a song from the playlist based on its title or artist.
* Ensure that the linked list is updated accordingly.

Playback Methods:

* Write methods to play the current song, play the next song, and play the previous song.
* Ensure that the current song is updated accordingly.

Display Method:

* Write a method to display the details of all the songs in the playlist, including title, artist, and duration.

Test the Implementation:

* Create a main method to test the functionality of the Playlist class.
* Perform various insertions, deletions, and playback operations to validate the correctness of the implementation.

class node { public Object item; public node next;}public class LinkedListImplementation { public node head; public int size; public LinkedListImplementation() { this.head = null; this.size = 0; } public void insertionAtEnd(String value) { node n1 = new node(); n1.item = value; node current = this.head; if (this.head == null) { this.head = n1; this.head.next = null; this.size = 1; } else { while (current.next != null) { current = current.next; } current.next = n1; n1.next = null; this.size += 1; } } public void insertInMiddle(String value, int position) { node n1 = new node(); n1.item = value; node current = this.head; if (this.head != null && position <= size) { for (int i = 0; i < position; i++) { current = current.next; } n1.next = current.next; current.next = n1; this.size += 1; } else { System.out.println("Position is not in Range"); } } public void insertAtBeginning(String value) { node newnode = new node(); newnode.item = value; newnode.next = head; this.head = newnode; this.size = this.size + 1; } public void printNodes() { if (this.size < 1) System.out.println("There are no nodes in the linked list"); else { node current = this.head; for (int i = 0; i < this.size; i++) { System.out.println("Node " + current.item + " is at location " + i); current = current.next; } } } public static void main(String[] args) { LinkedListImplementation l1 = new LinkedListImplementation(); l1.insertAtBeginning(String.valueOf(20)); l1.insertAtBeginning(String.valueOf(30)); l1.insertAtBeginning(String.valueOf(40)); l1.insertionAtEnd(String.valueOf(400)); l1.insertInMiddle("50", 2); l1.printNodes(); }}

**Min & Max in Binary Search Tree**

public static Node[] findMinMaxInBinarySearchTree(Node current) { if (current == null) { throw new IllegalArgumentException("The tree is empty."); } Node min = current; Node max = current; // Find the leftmost node to get the minimum value while (min.left != null) { min = min.left; } // Find the rightmost node to get the maximum value while (max.right != null) { max = max.right; } return new Node[] { min, max }; }