Planning a music festival is a lot of work and requires a great deal of coordination and organization. Often, a planned schedule needs to be changed over time to accommodate for unforeseen situations. Music performers need to be added or removed from the schedule in response to changing factors. You are to implement an interface that will store all of the performances in the form of a LinkedList. LinkedList is efficient in these types of situations because the list can be modified without changing the whole structure of the existing list.

1. Write a fully-documented class named PerformanceNode that contains basic information about a performance in a festival. Required information includes the name of the performance, the name of the lead performer, total participants, duration of the performance (in minutes), and the start time of the performance (in minutes for simplicity). The following is a partial specification and it is up to you to fill in the details:

```java
public class PerformanceNode

  • Constructor for PerformanceNode (you may include a constructor with parameters)
    public PerformanceNode()
  • set/get methods
    include mutator and accessor methods for all data fields
  • setNext
    public void setNext(PerformanceNode node)
  • setPrev
    public void setPrev(PerformanceNode node)
  • getNext
    public PerformanceNode getNext()
  • getPrev
    public PerformanceNode getPrev()
  • toString
    public String toString() // returns a neatly formatted String containing all the data of this PerformanceNode (ignore references)
```

2. Write a fully-documented class named PerformanceList that contains references to the head and tail of a list of PerformanceNode nodes, as well as a cursor representing the current PerformanceNode node. Your PerformanceList must be sorted according to the starting time of the performances. Your class will follow this specification, but you have to fill in the details:

```java
public class PerformanceList
```
• Constructor for PerformanceList
  public PerformanceList()
• addToEnd
  public void addToEnd(PerformanceNode newPerformance)
  Inserts the new performance at the end of the PerformanceList. The current node should now be the newly created node.
  NOTE: This method may still be used even when the list is empty. If this is the case, you must create a new node that will now be the head, tail, and current slide of the list.
• addAfterCurrent
  public void addAfterCurrent(PerformanceNode newPerformance)
  Inserts the new data into the PerformanceList such that the new node directly follows the current node, if it exists. If there is no current node (i.e., the current node is null), simply insert the node at the end of the list. The current node should now be the newly created node.
  NOTE: When adding a new performance into the middle of the list, the starting time of all subsequent performances should be extended by the value of the duration of the newly added performance.
• removeCurrentNode
  public boolean removeCurrentNode()
  Removes the current node, if it exists. The current node should now be the node after the one that was just removed. If there is no node after the one that was just removed, the current node should now be the node before the one that was just removed. If the node removed was the only one in the PerformanceList, the current node should now be null. The return value indicates whether or not any node was removed.
  NOTE: When removing a performance from the middle of the list, the starting time of all subsequent performances should be reduced by the value of the duration of the removed performance. NOTE: Do not destroy the list as you do this.
• displayCurrentPerformance
  public void displayCurrentPerformance()
  Prints the data in the current node
• moveCursorForward
  public boolean moveCursorForward()
  Moves the reference of the current node forward in the list by one position if a node exists after the current one and returns true. If there is no next node, the current node should remain the same and return false. If there is no current node, you should throw an appropriate exception.
  NOTE: This method should move the reference of the current node to the subsequent node. Do not move the node one position forward.
• moveCursorBackward
  public boolean moveCursorBackward()
  Moves the reference of the current node backwards in the list by one position if a node exists before the current one and returns true. If there is no previous node, the current node should remain the same and return false. If
there is no current node, you should throw an appropriate exception.  
**NOTE:** *This method should move the reference of the current node to the previous node. Do not move the node one position backward.*

- **jumpToPosition**
  ```java
  public boolean jumpToPosition(int position)
  ```
  Moves the current node to the given position in the PerformanceList. Assume the first performance in the PerformanceList is position 1. If the given position doesn't exist in the list, leave the current node where it was and return false. Otherwise, set the current node to the correct position and return true.

- **toString**
  ```java
  public String toString()
  ```
  Returns a neatly formatted table of all information for all the scheduled performances.

---

3. Write a fully-documented class named PerformanceScheduler. This class will contain a main method that presents a menu that allows the user to access data from the API, store it in a linked list, and interact with it using the following commands:

- **A** (Add a new performance to the list)
  Prompts the user for information regarding a performance. Create a new PerformanceNode for this performance, and add it to the end of the list.

- **I** (Insert a new performance after the current node)
  Prompts the user for information regarding a performance. Create a new PerformanceNode and add it after the current performance. If there is no current node, insert the new node at the end of the list.

- **R** (Remove the current node from the list)
  If there is a current node, it is removed from the list and the current node is moved to the next appropriate current node. If there is no current node, inform the user of the situation with an appropriate error message.

- **C** (Display the current node)
  Displays the contents of the current node. If there is no current node, display an appropriate error message to the user.

- **D** (Display all nodes)
  Displays the contents of all nodes, in order. If there are no nodes, display an appropriate error message to the user.

- **F** (Move cursor forward)
  Moves the reference of the current node forward by one position if there is another node in the list. If the current slide is the last in the list, simply display an appropriate message to the user and do nothing.
• B (Move cursor backwards)
  Moves the reference of the current node backwards by one position if there is
  a previous node in the list.
  If the current node is the first in the list, simply display an appropriate
  message to the user and do nothing.
• J (Jump to a given position)
  Prompts the user for a position number and moves the current node to that
  number node, if it exists.
  If it does not exist, simply display an error message to the user and do
  nothing.
• Q (Exit the program.)

4. Supply any exception handling class(es) that you need in addition to the
   classes above.

INPUT FORMAT

• All menu selections should be case-insensitive (e.g. 'q' and 'Q' are the same).
• You may assume that all input will be of the correct type. For example, if the
  input is supposed to be an integer, we will not test your program with a non-
  integer String instead.
• Assume the maximum size of a String is 25 characters long. Assume all time
  durations to be limited to 3 digit values.

SAMPLE INPUT/OUTPUT:

Output in black. Input in red. Comments in green.

A) Add to end
B) Move current node backward
C) Display current node
D) Display all nodes
F) Move current node forward
I) Insert after current node
J) Jump to position
R) Remove current node
Q) Exit

Choose an operation: A

Enter name of performance: Symphony Orchestra
Enter name of lead performer: Jennifer Aylmer
Enter the total participants: 50
Enter the duration of the performance: 30
New performance Symphony Orchestra is added to the end of the list.

Choose an operation: A

Enter name of performance: Wind Ensemble
Enter name of lead performer: Bruce Engel
Enter the total participants: 30
Enter the duration of the performance: 40

New performance Wind Ensemble is added to the end of the list.

Choose an operation: D

Schedule:

<table>
<thead>
<tr>
<th>Current No.</th>
<th>Performance Name</th>
<th>Lead Performer Name</th>
<th>Participants</th>
<th>Duration</th>
<th>Start Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Symphony Orchestra</td>
<td>Jennifer Aylmer</td>
<td>50</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Bruce Engel</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>2 Wind Ensemble</td>
<td></td>
<td></td>
<td>30</td>
<td>40</td>
</tr>
</tbody>
</table>

// The current node the cursor is pointing to on the list is represented by the asterisk (*)

Choose an operation: B

Cursor has been moved backwards.

Choose an operation: C

Current node:

<table>
<thead>
<tr>
<th>Current No.</th>
<th>Performance Name</th>
<th>Lead Performer Name</th>
<th>Participants</th>
<th>Duration</th>
<th>Start Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>1 Symphony Orchestra</td>
<td>Jennifer Aylmer</td>
<td>50</td>
<td>30</td>
<td>0</td>
</tr>
</tbody>
</table>

// The current node the cursor is pointing to on the list is represented by the asterisk (*)
Choose an operation: I

Enter name of performance: Emerson Quartet
Enter name of lead performer: Eugene Drucker
Enter the total participants: 4
Enter the duration of the performance: 30

New performance Emerson Quartet is added after the current performance.

Choose an operation: D

Schedule:

<table>
<thead>
<tr>
<th>Current No.</th>
<th>Performance Name</th>
<th>Lead Performer Name</th>
<th>Participants</th>
<th>Duration</th>
<th>Start Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Symphony Orchestra</td>
<td>Jennifer Aylmer</td>
<td>50</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>*</td>
<td>Emerson Quartet</td>
<td>Eugene Drucker</td>
<td>4</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>Wind Ensemble</td>
<td>Bruce Engel</td>
<td>30</td>
<td>40</td>
<td>60</td>
</tr>
</tbody>
</table>

// The current node the cursor is pointing to on the list is represented by the asterisk (*)

Choose an operation: F

Cursor has been moved forwards.

Choose an operation: F

Cursor is already at the end of the list. Cursor cannot be moved forwards.

Choose an operation: I

Enter name of performance: Piano Solo
Enter name of lead performer: Maksim Mrvica
Enter the total participants: 1
Enter the duration of the performance: 50

New performance Piano Solo is added after the current performance.

Choose an operation: D
<table>
<thead>
<tr>
<th>Current No.</th>
<th>Performance Name</th>
<th>Lead Performer Name</th>
<th>Participants</th>
<th>Duration</th>
<th>Start Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Symphony Orchestra</td>
<td>Jennifer Aylmer</td>
<td>50</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Emerson Quartet</td>
<td>Eugene Drucker</td>
<td>4</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>Wind Ensemble</td>
<td>Bruce Engel</td>
<td>30</td>
<td>40</td>
<td>60</td>
</tr>
<tr>
<td>4</td>
<td>Piano Solo</td>
<td>Maksim Mrvica</td>
<td>1</td>
<td>50</td>
<td>100</td>
</tr>
</tbody>
</table>

// The current node the cursor is pointing to on the list is represented by the asterisk (*)

Choose an operation: J

Enter the position: 1

Cursor has been moved to position 1.

Choose an operation: R

Performance Symphony Orchestra has been removed.

Choose an operation: D

<table>
<thead>
<tr>
<th>Current No.</th>
<th>Performance Name</th>
<th>Lead Performer Name</th>
<th>Participants</th>
<th>Duration</th>
<th>Start Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Emerson Quartet</td>
<td>Eugene Drucker</td>
<td>4</td>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Wind Ensemble</td>
<td>Bruce Engel</td>
<td>30</td>
<td>40</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>Piano Solo</td>
<td>Maksim Mrvica</td>
<td>1</td>
<td>50</td>
<td>70</td>
</tr>
</tbody>
</table>

// The current node the cursor is pointing to on the list is represented by the asterisk (*)

Choose an operation: Q

Program terminating normally...