CTEC2909 Practical 3 – More Linked Lists

Use the LinkedLists project that was used for previous practicals.

1. Open LinkedList.java in the src/lib folder, and add the following method:

```
public void insertBefore(Node<E> newNode, Node<E> after){
    Node<E> current = head;
    Node<E> prev = null;
    // Loop through the list to find the place to insert.
    while(current != after){
           prev = current;
           current = current.next();
    }
    // prev now points to the node before the place to insert.

    // Set the new node to point to after, and prev to point to the new node.
    newNode.setNext(after);
    prev.setNext(newNode);
}
```

This method inserts a node *before* a given node. The node to insert is **newNode** and the node after the place where it should be inserted is called **after**. Look carefully at the code. The first part is for finding the node before the node **after**. First, a node **current** is set to the head and a node **prev** is set to null. Then, a while loop runs. Each time the while loop runs, it changes **prev** to be the same as **current**, and **current** to be the next node. In this way, at each step, **prev** is referring to one node before **current**. When the while loop ends, **current** is the same as **after** and **prev** is one before it.

Now, **prev** is the node before **after**. We want to make **prev** point to the new node and the new node point to **after**. That's what the last two lines do.

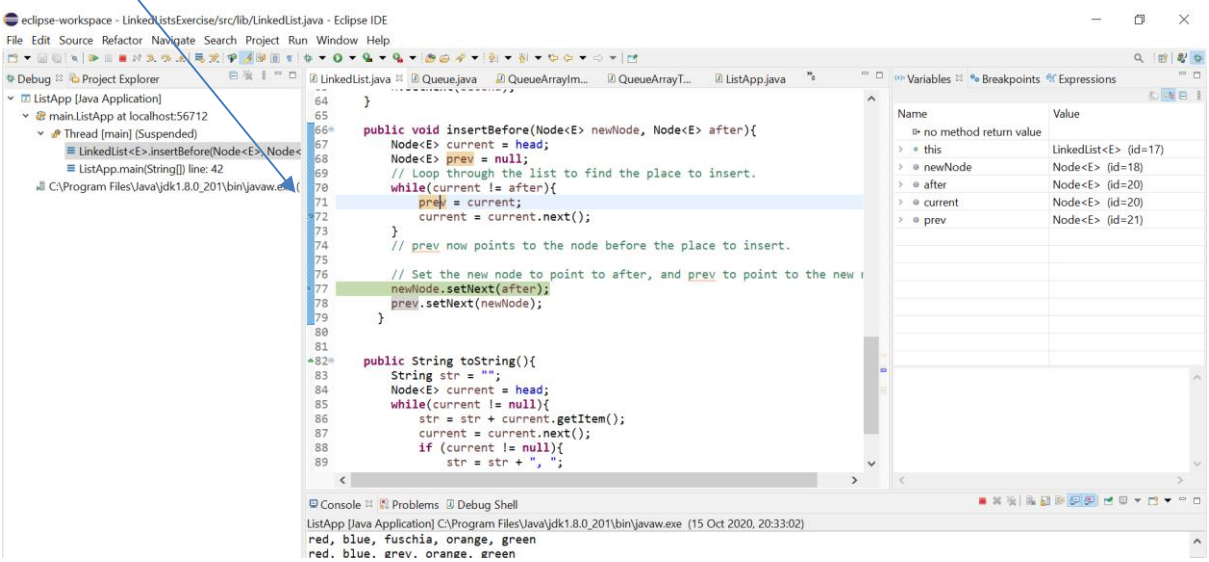
Try using the method by adding lines such as the following into ListApp.java (change **n2** to be some node that exists in your list):

```
Node<String> purple = new Node<>("purple", null);
list.insertBefore(purple, n2);
System.out.println(list);
```

Make sure that you understand how the while loop works. If you like, you can add System.out.println statements inside the while loop so that when it runs you can see the values of current and prev at each step, e.g.

```
while(current != after){
       prev = current;
       current = current.next();
       System.out.println("Current: " + current.getItem());
       System.out.println("Prev: " + prev.getItem());
}
```

Alternatively, you could use the Debug mode in Eclipse to see what is happening. Double-click the vertical blue bar next to the code so that a dot appears next to the line you clicked on.



This is a breakpoint. Put one next to one of the lines inside the while loop. Then select the menu item Run->Debug. Select ok if you are asked about switching to the Debug window. The program executes and stops at the breakpoint. Hold your mouse over the variables and you can see their values. You might need to step over to the next line by selecting Run->Step over (or F6). Keep stepping over and look at the values of the variables at each iteration of the while loop.

2. The **`insertBefore`** method has some problems. Try using the method with a null reference for the **`after`** parameter, e.g.

   ```
   list.insertBefore(purple, null);
   ```

   This will incorrectly add a node to the end of a list. The reason is that the while loop will stop when **`current`** is equal to **`after`**, so it will stop when **`current`** is null. This is when **`prev`** is the last node.

   Fix the code so that it does not attempt to do anything if **`after`** is null. To do this, add the line `if (after != null){` as follows:

```
public void insertBefore(Node<E> newNode, Node<E> after){
  if (after != null){
    Node<E> current = head;
    Node<E> prev = null;
    // Loop through the list to find the place to insert.
    while(current != after){
      prev = current;
      current = current.next();
    }
    // prev now points to the node before the place to insert.

    // Set the new node to point to after, and prev to point to the new node.
    newNode.setNext(after);
```

```
        prev.setNext(newNode);
    }
}
```

Try using the **insertBefore** with a null reference for the **after** parameter again and see what happens.

3. Now try using **insertBefore** with a null reference for the **newNode** parameter. You will get a NullPointerException. Why is this? When it attempts to call **setNext** on **newNode**, it cannot do it, as **newNode** is null. Fix the code in a similar way to Question 2, so that it will not do anything if **newNode** is null.

4. What happens if the **after** node is not in the list at all? Try it and see. Create another node that is not in the list and give that as the parameter **after**. You will get a NullPointerException. This is because the while loop's condition will never be satisfied. It will keep on looping until it reaches the end of the list, and then attempt to set **current** to the next node, even after **current** is null.

Fix the code to stop it from doing anything if the node is not in the list. This time, the while loop needs to run until it reaches the end of the list. If it still hasn't found the node by then, then it shouldn't do anything more and should end the while loop.

5. Now, what happens if you try to add a node before the first node in the list (the head)? You will get a NullPointerException. Try it and see. This is because when **after** is the head, after the while loop ends, **prev** will be still null, so it cannot call **setNext** on **prev**.

Fix the code so that it handles this case separately. This time, it should still insert the node, but it will need to make the new node the head and will not use **prev**.

6. Write a method **void changeNodeBefore(Node<E> after, E e)** that changes the item of the node *before* the given node **after**. Hint: You will need to get to the previous node by using a while loop in a similar way to **insertBefore**. Remember to watch out for the case where **after** is the head node.

7. Write a method **void deleteHead()** that deletes the head of the list. It should then make the next node in the list the new head.