Traversing the Linked List 2

Dawn Finney
Question

Suppose we have a linked list of rooms in a building, and the rooms are defined as follows:

```java
public class RoomElement {
    private int area = 0;
    private RoomElement next = null;
    public int getArea() {return this.area;}
    public RoomElement getNext() {return this.next;}
    public void setArea(int a) {this.area = a;}
    public void setNext(RoomElement re) {this.next = re;}
}
```

Write a method totalArea() that returns the total area of all the rooms in a linked list starting from the first one in the list. The method will be called on the starting position.
public int totalArea() {
    int total = this.getArea();
    RoomElement current = this;
    while (current.getNext() != null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}
Testing the Solution:
Setting up the Test List

RoomElement node1 = new RoomElement();
RoomElement node2 = new RoomElement();
RoomElement node3 = new RoomElement();

node1.setArea(1);
node2.setArea(2);
node3.setArea(3);

node1.setNext(node2);
node2.setNext(node3);

System.out.println(node1.totalArea());
Testing the Solution: Running through the Method

```java
public int totalArea() {
    int total = this.getArea();
    RoomElement current = this;
    while (current.getNext() != null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}
```

He initializes `total` to the area of the first node.

```
node 1 -> node 2 -> node 3 -> null
```

```
total = 1
```
Testing the Solution: Running through the Method

```java
public int totalArea() {
    int total = this.getArea();
    RoomElement current = this;
    while (current.getNext() != null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}
```

Just like before he has a variable `current` acting as a pointer to the current node he is considering in the list, but have we not seen this same mistake before? Let us explore further.
public int totalArea() {
    int total = this.getArea();
    RoomElement current = this;
    while (current.getNext() != null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}
Testing the Solution: Running through the Method

```java
public int totalArea() {
    int total = this.getArea();
    RoomElement current = this;
    while (current.getNext() != null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}
```

Here he adds the current node’s data to the running total, but did he not already accounted for the first node’s data already?

current

node 1 1 → node 2 2 → node 3 3 → null

total = 2
Testing the Solution: Running through the Method

```java
public int totalArea() {
    int total = this.getArea();
    RoomElement current = this;
    while (current.getNext() != null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}
```

He finishes looking at the `current` so he changes it to the next node.

`total = 2`
Testing the Solution: Running through the Method

```java
public int totalArea() {
    int total = this.getArea();
    RoomElement current = this;
    while (current.getNext() != null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}
```

Because the expression `current.getNext() != null` evaluates to `true`, the loop continues.
Testing the Solution: Running through the Method

public int totalArea() {
    int total = this.getArea();
    RoomElement current = this;
    while (current.getNext() != null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}

He adds the data from the current node to the running total.
Testing the Solution:
Running through the Method

```java
public int totalArea() {
    int total = this.getArea();
    RoomElement current = this;
    while (current.getNext() != null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}
```

He finishes looking at the `current` so he changes it to the next node.

```
total = 4
```
Testing the Solution:
Running through the Method

public int totalArea() {
    int total = this.getArea();
    RoomElement current = this;
    while (current.getNext() != null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}

Because the expression current.getNext() != null evaluates to false, we do not enter the loop again. Anyone else notice that he missed a node?
Testing the Solution: Running through the Method

```java
public int totalArea() {
    int total = this.getArea();
    RoomElement current = this;
    while (current.getNext() != null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}
```

He returns the total here., in this case, it is 4.
Anyone Else See a Problem Here?

Just using our heads we can tell that the actual answer is supposed to be 6 instead of 4.
If you have not already done so, take a moment to consider what the student did wrong.
What the Student Did Wrong

• The student considered the first node twice and added its data to the total one too many times.

• The student failed to consider the very last node in the list
public int totalArea() {
    int total = 0;
    RoomElement current = this;
    while (current != null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}
Testing the Solution:
Running through the Method

public int totalArea() {
    int total = 0;
    RoomElement current = this;
    while (current!= null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}

We initialize the total to 0.
Testing the Solution: Running through the Method

```java
public int totalArea() {
    int total = 0;
    RoomElement current = this;
    while (current != null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}
```

Just like before we have a variable `current` acting as a pointer to the current node.
Testing the Solution: Running through the Method

public int totalArea() {
    int total = 0;
    RoomElement current = this;
    while (current != null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}
Testing the Solution:  
Running through the Method

```java
public int totalArea() {
    int total = 0;
    RoomElement current = this;
    while (current!= null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}
```

Here we add the current node’s data 1 to the running total.
public int totalArea() {
    int total = 0;
    RoomElement current = this;
    while (current != null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}

We finish looking at the current so we set it equal to the next node.
Testing the Solution: Running through the Method

```java
public int totalArea() {
    int total = 0;
    RoomElement current = this;
    while (current!= null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}
```

Because the expression `current!= null` evaluates to `true`, the loop continues.
Testing the Solution:
Running through the Method

```java
public int totalArea() {
    int total = 0;
    RoomElement current = this;
    while (current != null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}
```

We add the data 2 from the current node to the running total.
Testing the Solution: Running through the Method

```java
public int totalArea() {
    int total = 0;
    RoomElement current = this;
    while (current != null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}
```

We finish looking at the `current` so we change it to the next node.

current

total = 3
Testing the Solution: Running through the Method

public int totalArea() {
    int total = 0;
    RoomElement current = this;
    while (current != null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}

Because the expression `current != null` evaluates to `true`, we continue within the loop.
Testing the Solution: Running through the Method

```java
public int totalArea() {
    int total = 0;
    RoomElement current = this;
    while (current!= null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}
```

We add the data 3 from the current node to the running total.
Testing the Solution: Running through the Method

```java
public int totalArea() {
    int total = 0;
    RoomElement current = this;
    while (current != null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}
```

We finish looking at the current so we change it to the next node.

```
total = 6
```
Testing the Solution: Running through the Method

```java
public int totalArea() {
    int total = 0;
    RoomElement current = this;
    while (current != null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}
```

Because the expression `current != null` evaluates to `false`, we exit the loop.
Testing the Solution:
Running through the Method

public int totalArea() {
    int total = 0;
    RoomElement current = this;
    while (current!= null) {
        total = total + current.getArea();
        current = current.getNext();
    }
    return total;
}

We return the total here., in this case, it is 6.
What You Should Get From This

• There are multiple ways of approaching a question, and there can often be multiple solutions that will yield equally accurate answers.

• Not all linked list methods will fit in a similar mold. Just like in other courses such as physics, chemistry, math, etc, you have consider the current situation (see next slide).
Questions to Ask Yourself When Writing a Linked List

• Do I want to work with the current node or the one after it?
• What should my terminating condition be?
• What do I want to do at each node?
• Do I move to next node after I am done or stay in the same place?
• As there any special cases that might break my code?
  – Do I notify the user that this case has occurred?
  – Do I move on after the case has occurred?
Last Remarks

• Do not be afraid of writing an outline of what you want to do before starting. Many people add comments after everything has been written, right before submission, but comments can also serve as an outline if written before any actual code is written.