QUIZ

What would you add to complete the following `.addToTail()` method?

```
class Node {
  setNextNode(data) {/* Method that sets the next Node */}
  setPreviousNode(data) {/* Method that sets the previous Node */}
}

class DoublyLinkedList {
  addToTail(data) {
    const newTail = new Node(data);
    const currentTail = this.tail;
    if (currentTail) {
      currentTail.  setNextNode  (  newTail  );

      newTail.  setPreviousNode  (  currentTail  );
    }
    this.tail = newTail;
    if (!this.head) {
      this.head = newTail;
    }
  }
}
```

You got it!

The `.addToHead()` and `.addToTail()` methods in `DoublyLinkedList` are the same as those in the `LinkedList`.

True

False

Yes! They are different because of the added `tail` property in `DoublyLinkedList`.

What would you add to complete the following `.removeHead()` method?

```
class Node {
  getNextNode() {/* Method that returns the next node */}
  setPreviousNode(data) {/* Method that sets the previous node */}
}
class DoublyLinkedList{
  removeHead() {
    const removedHead = this.head;
    if (!removedHead) {
      return;
    }
      this.head = removedHead.getNextNode();
    if (this.head) {
        this.head.setPreviousNode(null);
    }
    if (removedHead ===  this.tail  ) {
        this.removeTail();
    }
    return removedHead.data;
  }
  removeTail() {/* Method that removes the tail of the list */}
}
```

You got it!

What is the difference between the `DoublyLinkedList` and `LinkedList` constructors?

The `DoublyLinkedList` constructor has an added `tail` property.

Yes!

The `DoublyLinkedList` constructor has an added `previous` pointer.

The `DoublyLinkedList` constructor takes a parameter while the `LinkedList` constructor does not.

There is no difference, the constructors are the same.

Given the following code, how would you complete the `.removeByData()` method?

```
class DoublyLinkedList {

  removeByData(data) {
    let nodeToRemove;
    let currentNode = this.head;
    while (currentNode !== null) {
      if (currentNode.data === data) {
        nodeToRemove = currentNode;
        break;
      }
      currentNode = currentNode.getNextNode();
    }
    if (!nodeToRemove) {
      return null;
    }
    if (nodeToRemove === this.head) {
      this.removeHead();
    } else if (nodeToRemove === this.tail) {
      this.removeTail();
    } else {

      const nextNode = nodeToRemove.getNextNode() ;

      const previousNode = nodeToRemove.getPreviousNode() ;

        nextNode .setPreviousNode(previousNode);

        previousNode .setNextNode(nextNode);

    }
    return nodeToRemove;
  }
}
```

Which of the following is NOT true about the JavaScript implementation of a `DoublyLinkedList`?

- There is an added `.removeByData()` method.
- `DoublyLinkedList` has a `tail` property.
- There is an added `.removeTail()` method.
- `DoublyLinkedList` uses a different `Node` class.
- Nodes are no longer in one order since the list has a tail.

👏 Yes! The list is still in order; it starts at the head and ends at the tail. Having a tail just means that you can access the last element more easily.

Why do the `DoublyLinkedList` class and the `LinkedList` class use different `Node` classes?

They use different `Node` classes because `DoublyLinkedList` needs a `Node` class with a `tail` property.

They use different `Node` classes because `DoublyLinkedList` needs a class with an added `previous` pointer and related methods.

👏 Yes! The `DoublyLinkedList` class uses a `Node` class that has an added `previous` pointer and the related setter and getter methods.

They don't use different `Node` classes.

The `.addToHead()` and `.addToTail()` methods in `DoublyLinkedList` are the same as those in the `LinkedList`.

False

👏 Yes! They are different because of the added `tail` property in `DoublyLinkedList`.

True

What would you add to complete the following `.removeTail()` method?

```
class Node {
  setNextNode(data) {/* Method that sets the next node */}
  getPreviousNode() {/* Method that returns the previous node */}
}
class DoublyLinkedList {
  removeHead(){/* Method that removes the head of the list */}
  removeTail() {
    const removedTail = this.tail ;
    if ( !removedTail ) {
      return;
    }
    this.tail = removedTail.getPreviousNode();
    if ( this.tail ) {
      this.tail.setNextNode(null);
    }
    if (removedTail === this.head) {
      this.removeHead();
    }
    return removedTail.data;
  }
}
```

👏 You got it!

What is the difference between the `DoublyLinkedList` and `LinkedList` constructors?

The `DoublyLinkedList` constructor has an added `tail` property.

Yes!

The `DoublyLinkedList` constructor has an added `previous` pointer.

The `DoublyLinkedList` constructor takes a parameter while the `LinkedList` constructor does not.

There is no difference, the constructors are the same.

What would you add to complete the following `.addToHead()` method?

```
class Node {
  setNextNode(data) {/* Method that sets the next Node */}
  setPreviousNode(data) {/* Method that sets the previous Node */}
}

class DoublyLinkedList {
  addToHead(data) {
    const newHead = new Node(data);
    const currentHead = this.head;
    if (currentHead) {
      currentHead.setPreviousNode(newHead);
      newHead.setNextNode(currentHead);
    }
      this.head = newHead;

    if (!this.tail ) {

        this.tail = newHead;

    }
  }
}
```

You got it!

What will the following code output?

```
class DoublyLinkedList {
  constructor() {
    this.head = null;
    this.tail = null;
  }
  addToHead(data) {/* Method that adds a node to the head of the list */}
  addToTail(data) {/* Method that adds a node to the tail of the list */}
  removeHead() {/* Method that removes the head of the list */}
  removeTail() {/* Method that removes the tail of the list */}
  removeByData(data) {/* Method that removes a node that matches the data passed in */}
}

const testList = new DoublyLinkedList();
testList.addToHead(9);
testList.removeTail();
testList.addToTail(8);
testList.addToTail(2);
testList.removeHead();
testList.addToTail(4);
testList.removeByData(9);
testList.removeHead();
console.log(testList.head.data);
```

```
4
```

Yes!