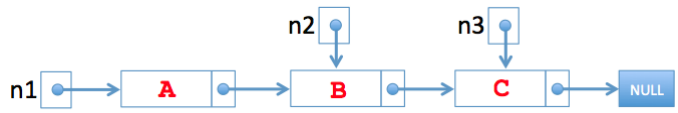
# CST 283 - Practice 9.1.1

1. Given the following definitions:

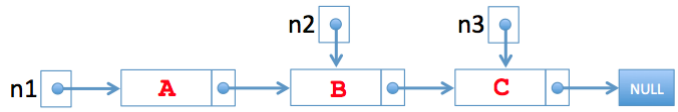
```
private class Node
{
    String value;
    Node next;
    Node(String val, Node n)
    {
        value = val;
        next = n;
    }
}

Node n1, n2, n3;
```

mark or sketch the changes made to the linked structure by the given code:

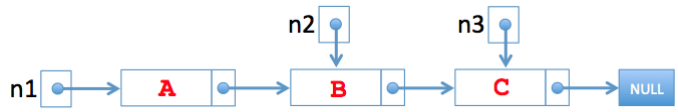
(a) `n1 = n1.next;`



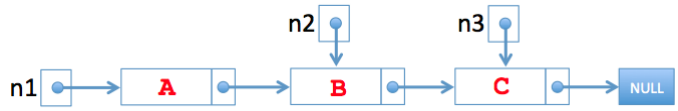

(b) `n2 = n1;`



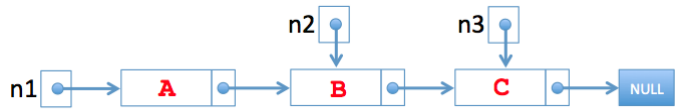

(c) `n3 = n1.next;`



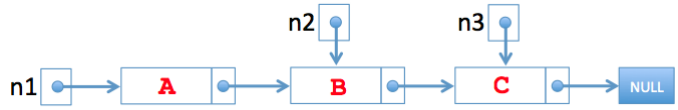

(d) `n1.value = n2.value;`




(e) `n1.value = n2.next.value;`

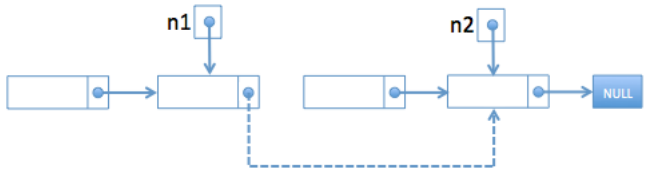


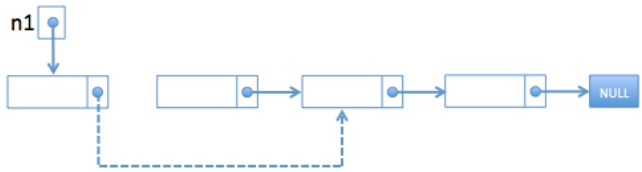

(f) `n3.next = n1;`

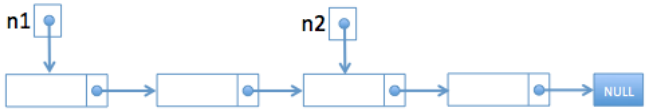
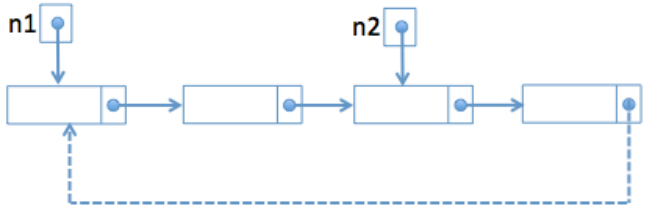


2. Write the Java statement to effect the indicated change from the ***before*** to the ***after***.

| **Before** | **After** |
|---|---|
|  |  |
| | |

| **Before** | **After** |
|---|---|
|  |  |
| | |

| **Before** | **After** |
|---|---|
|  |  |
| | |

| **Before** | **After** |
|---|---|
|  |  |
| | |

**CST 283 - Practice 9.1.2** **Access Code: `t56w`**

---

Access the **`StringStack`** class (from Week 8) and the **`StringQueue`** class (from Week 9) as well as the driver program **`StackQueueDemo.java`** available from the in-class course folder indicated above.

Next, examine the stack and queue operations on strings below. Trace the algorithm and predict the output:

Expected Output

Push **`AB`** onto the stack _____
Push **`XY`** onto the stack
Enqueue **`JK`** to the queue
Pop from the stack and enqueue the value to the queue
Peek at the stack top and write the element to output
Push **`JV`** onto the stack
Push **`PQ`** onto the stack
Enqueue **`SD`** to the queue
Peek at the queue front and write the element to output
Pop the stack and discard
Peek at the stack top and write the element to output
Dequeue from the queue and push onto the stack
Enqueue **`CV`** and **`SA`** to the queue in order
Pop a value from the stack and discard
Until empty, dequeue all elements from queue and push onto stack
Until empty, pop and write the element to output

Finally, implement the code that will perform the operations in the driver to verify your analysis. Alter the existing test driver removing the sequence of stack operations and replacing them with the given sequence.

## CST 283 - Practice 9.1.3

---

Build a project to execute a Week 9 example with source files **`ArrayQueueDemo.java`** and **`StringQueue.java`**.

Add the following new features to the **`StringQueue`** class:

1. A copy constructor:

   **`StringQueue(StringQueue existingQueue)`**

   This constructor should receive a queue object as a parameter and will build the queue as a clone of the object being passed in. An example where this would be invoked could be:

   ```
   StringQueue myQueue = new StringQueue(anotherQueue);
   ```

2. An **`equals(StringQueue existingQueue)`** method that will compare two queues. Return **true** if ...
   - both queues have equal length
   - the elements from the front to rear of the queue match in sequence identically

   Note that the front and rear markers do not have to match for two queues to be defined as equal.

**CST 283 - Practice 9.1.2** **Access Code: `au28`**

**CST 283 - Practice 9.1.3** **Access Code: `re58`**