1 More Practice with Linked Lists

```java
public class SLList {
    private class IntNode {
        public int item;
        public IntNode next;
        public IntNode(int item, IntNode next) {
            this.item = item;
            this.next = next;
        }
    }

    private IntNode first;

    public void addFirst(int x) {
        first = new IntNode(x, first);
    }

    public void insert(int item, int position) {
        if (first == null || position == 0) {
            addFirst(item);
            return;
        }
        IntNode currentNode = first;
        while (position > 1 && currentNode.next != null) {
            position--;
            currentNode = currentNode.next;
        }
        IntNode newNode = new IntNode(item, currentNode.next);
        currentNode.next = newNode;
    }
}
```

(a) Implement SLList.insert which takes in an integer \( x \) and an integer \( \text{position} \). It inserts \( x \) at the given \( \text{position} \). If \( \text{position} \) is after the end of the list, insert the new node at the end.

For example, if the SLList is 5 \( \rightarrow \) 6 \( \rightarrow \) 2, \( \text{insert}(10, 1) \) results in 5 \( \rightarrow \) 10 \( \rightarrow \) 6 \( \rightarrow \) 2 and if the SLList is 5 \( \rightarrow \) 6 \( \rightarrow \) 2, \( \text{insert}(10, 7) \) results in 5 \( \rightarrow \) 6 \( \rightarrow \) 2 \( \rightarrow \) 10. Additionally, for this problem assume that \( \text{position} \) is a non-negative integer.

```java
public void insert(int item, int position) {
    if (first == null || position == 0) {
        addFirst(item);
        return;
    }
    IntNode currentNode = first;
    while (position > 1 && currentNode.next != null) {
        position--;
        currentNode = currentNode.next;
    }
    IntNode newNode = new IntNode(item, currentNode.next);
    currentNode.next = newNode;
}
```

(b) Add another method to SLList that recursively removes all nodes that contain
a certain item. This method takes in an integer \( x \) and destructively changes the list.

For example, if the `SLList` is \( 3 \rightarrow 5 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 5 \), `removeItem(5)` results in \( 3 \rightarrow 4 \rightarrow 6 \).

```java
public void removeItem(int x) {
    first = removeItemHelper(x, first);
}
```

```java
private IntNode removeItemHelper(int x, IntNode current) {
    if (current == null) {
        return null;
    } else if (current.item == x) {
        return removeItemHelper(x, current.next);
    } else {
        current.next = removeItemHelper(x, current.next);
        return current;
    }
}
```

(c) **Extra**: Add another method to the `SLList` class that reverses the elements. Do this using the existing `IntNode` objects (you should not use `new`).

```java
public void reverse() {
    if (first == null || first.next == null) {
        return;
    }

    IntNode ptr = first.next;
    first.next = null;

    while (ptr != null) {
        IntNode temp = ptr.next;
        ptr.next = first;
        first = ptr;
        ptr = temp;
    }
}
```
2 Arrays

(a) Consider a method that inserts an `int` item into an `int[] arr` at the given position. The method should return the resulting array. For example, if `arr = [5, 9, 14, 15]`, `item = 6`, and `position = 2`, then the method should return `[5, 9, 6, 14, 15]`. If `position` is past the end of the array, insert `item` at the end of the array. Assume we will only ever pass in a non-negative `position`.

Is it possible to write a version of this method that returns `void` and changes `arr` in place (i.e., destructively)? *Hint:* These arrays are filled meaning an array containing `n` elements will have length `n`.

No, because arrays have a fixed size, so to add an element, you need to create a new array.

Fill in the below according to the method signature:

```java
public static int[] insert(int[] arr, int item, int position) {
    int[] result = new int[arr.length + 1];
    position = Math.min(arr.length, position);
    for (int i = 0; i < position; i++) {
        result[i] = arr[i];
    }
    result[position] = item;  
    for (int i = position; i < arr.length; i++) {
        result[i + 1] = arr[i];  
    }
    return result;
}
```

(b) Write a non-destructive method `replicate(int[] arr)` that replaces the number at index `i` with `arr[i]` copies of itself. For example, `replicate([3, 2, 1])` would return `[3, 3, 3, 2, 2, 1]`. For this question assume that all elements of the array are positive.

```java
public static int[] replicate(int[] arr) {
    int total = 0;
    for (int item : arr) {
        total += item;
    }
    int[] result = new int[total];
    int i = 0;
    for (int item : arr) {
        for (int counter = 0; counter < item; counter++) {
            result[i] = item;
            i++;
        }
    }
    return result;
}
```