Lab

Exercise

Complete the helper method `smallest`.

It finds the smallest integer in an integer array *recursively*.

You may assume that input array has at least one element in it

You must **not** use any loops or regular expressions.

Test cases:

`smallest`([10, 5, 7, 9]) → 5

For example:

Test	Result

`int[] arr1 = {10, 5, 7, 9};`
`System.out.println(smallest(arr1));`
	5

```java
// Lab Exercise #11.1 Recursive Smallest Integer
// Appear in FINAL EXAM 2019
public static int smallest(int[] array) {
    return smallest(array, 0);
}

private static int smallest(int[] array, int start) {

    // base case
     if (start == array.length - 1) {
            return array[start];
    }

    // recursive step
    int smallestInRest = smallest(array, start + 1);
    return Math.min(array[start], smallestInRest);


}
```

challenge

Complete the helper method `replaceOddZeroHelper`.

It replaces *all* odd numbers with zero in the input integer list *recursively*.

You must **not** use any loops or regular expressions.

The imported libraries are Arrays and List.

Test cases:

`replaceOddZero`([1, 2, 3, 4, 5]) → [0, 2, 0, 4, 0]

For example:

Test	Result

`List<Integer> list = Arrays.asList(1, 2, 3, 4, 5);`
`replaceOddZero(list);`
`System.out.println(list);`
	`[0, 2, 0, 4, 0]`

```java
// Lab 11 Challenge Problem
public static void replaceOddZero(List<Integer> list) {
    replaceOddZeroHelper(list, 0);
}

```

```java
private static void replaceOddZeroHelper(List<Integer> list, int start) {
    // Base case
    if  (start == list.size()) {
        return;
    }
    // Recursive step
    int current = list.get(start);
    if (current % 2 != 0) {
        list.set(start, 0);
    }
    replaceOddZeroHelper(list, start + 1);
}
```

Exercise

11.1

Complete the method `numXY`.

It finds the number of times the string "XY" appears in the input string *recursively*.

You must **not** use any loops or regular expressions.

Test cases:

`numXY`("AAXYAA") → 1

`numXY`("AXYBXYAA") → 2

For example:

Test	Result
`System.out.println(numXY("AAXYAA"));`	1
`System.out.println(numXY("AXYBXYAA"));`	2

```java
// Exercise #11.1
public static int numXY(String input) {
    // base case: length < 2 cannot contain "XY"
    if (input.length() < 2) {
        return 0;
    }

    // if the first two chars are "XY", count 1 and continue from index 1
    if (input.substring(0, 2).equals("XY")) {
        return 1 + numXY(input.substring(1));
    }

    // otherwise skip the first character
    return numXY(input.substring(1));
}
```

11.2

Complete the method remDup.

It reduces all adjacent same characters that appear in the input string to a single character *recursively*.

You must **not** use any loops or regular expressions.

Test cases:
remDup("hello") → "helo"
remDup("abbbcd") → "abcd"

For example:

Test	Result
`System.out.println(remDup("hello"));`	`helo`
`System.out.println(remDup("abbbcd"));`	`abcd`

```java
// Exercise #11.2
public static String remDup(String input) {
    // base case: length 0 or 1 → no duplicates possible
    if (input.length() <= 1) {
        return input;
    }

    char first = input.charAt(0);
    char second = input.charAt(1);

    if (first == second) {
        // skip the duplicate character
        return remDup(input.substring(1));
    } else {
        // keep the first character and continue
        return first + remDup(input.substring(1));
    }
}
```

11.3

Complete the method `sepStar`.

It separates all identical adjacent characters that appear in the input string from each other by "*" *recursively*.

You must **not** use any loops or regular expressions.

Test cases:

`sepStar`("hello") → "hel*lo"

`sepStar`("uuvxxyzzz") → "u*uvx*xyz*z*z"

For example:

Test	Result
`System.out.println(sepStar("hello"));`	`hel*lo`
`System.out.println(sepStar("uuvxxyzzz"));`	`u*uvx*xyz*z*z`

```java
// Exercise #11.3
public static String sepStar(String input) {
    // base case: length 0 or 1 → no adjacent characters → return as is
    if (input.length() <= 1) {
        return input;
    }

    // check first two characters
    char first = input.charAt(0);
    char second = input.charAt(1);

    if (first == second) {
        // insert "*" between them, and recurse on the rest starting from second
        return first + "*" + sepStar(input.substring(1));
    } else {
        // no "*" needed, keep first and recurse on the rest
        return first + sepStar(input.substring(1));
    }
}
```

CW1

Complete the following Java method `checkAllLowercase` that given a string contained *only letters*, returns **true** if the string is composed of all lowercase letters, and **false** otherwise, using ***recursion***.

You must use recursion;
and must **not** use any loops or regular expressions, in which case it will be graded 0 marks.

Test cases:
String str1 = "abcd";
System.out.println(checkAllLowercase(str1)); → true
String str2 = "xYz";
System.out.println(checkAllLowercase(str2)); → false

For example:

Test	Result

`String str1 = "abcd";`
`System.out.println(checkAllLowercase(str1));`
	`true`

`String str2 = "xYz";`
`System.out.println(checkAllLowercase(str2));`
	`false`

```java
// CW1 #11.1 Check All Lowercase
// From FINAL EXAM CPT111 2324
public static boolean checkAllLowercase(String input) {
    // base case: empty string is considered valid
    if (input.length() == 0) {
        return true;
    }

    // check first character
    char ch = input.charAt(0);
    if (ch < 'a' || ch > 'z') {
        return false;
    }

    // recursive step: check the rest
    return checkAllLowercase(input.substring(1));
}
```