

## CS 2: Introduction to Programming Methods

### Diagnostic: Recursion (due Wednesday, February 1)

Name: \_\_\_\_\_

E-mail: \_\_\_\_\_

#### 0. More Recursion

Write a method `printStrings` which prints all the strings made up of `a`'s and `b`'s of length `n`. For example:

- `printStrings(0) ↪ ""`
- `printStrings(1) ↪ "a", "b"`
- `printStrings(2) ↪ "aa", "ab", "ba", "bb"`

```
public class Recursion {  
    public static void printStrings(int n) {  
        PS(n, "");  
    }  
}
```

→ PRIVATE STATIC VOID PS (int n, String acc)

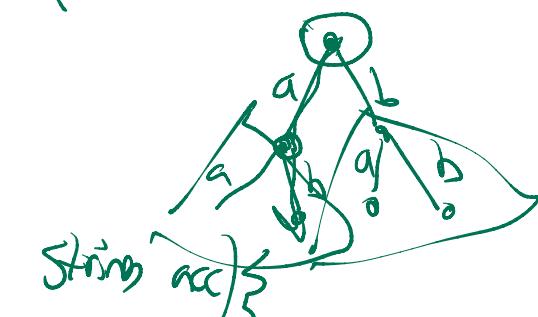
```
    if (n == 0) {  
        printIt(acc)  
    } ELSE {
```

→ PS (n-1, acc + "a") // "with" a ←

→ PS (n-1, acc + "b") // "with" b ←

}

Solve in CID



acc + 'a' + 'b'

for (char c = 'a'; c <= 'b'; c++)

→ acc += c  
 PS(n-1, acc);  
 acc.removeLast();

## CS 2: Introduction to Programming Methods

### Diagnostic: Recursion (due Wednesday, February 1)

Name: \_\_\_\_\_

E-mail: \_\_\_\_\_

#### 0. More Recursion

Write a method printStrings which prints all the strings made up of *a*'s and *b*'s of length *n*. For example:

- printStrings(0) ↪ ""
- printStrings(1) ↪ "a", "b"
- printStrings(2) ↪ "aa", "ab", "ba", "bb"

```
public class Recursion { makeStrings
    public static void printStrings(int n) {
        List<String> list = new ArrayList<>();
        private void List<n, "", list> {
    } return list;
}
```

```
private static void PopulateList(int n, String acc, List<String> l) {
    if (n == 0) {
        l.add(acc);
    } else {
        Popl(n-1, acc + "a", l)
        Popl(n-1, acc + "b", l)
    }
}
```

```
}
```