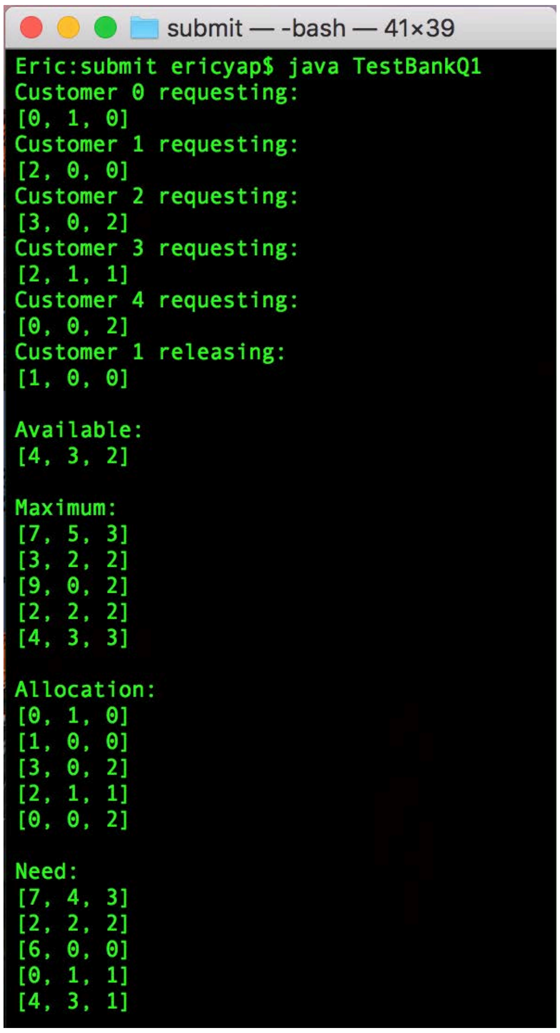
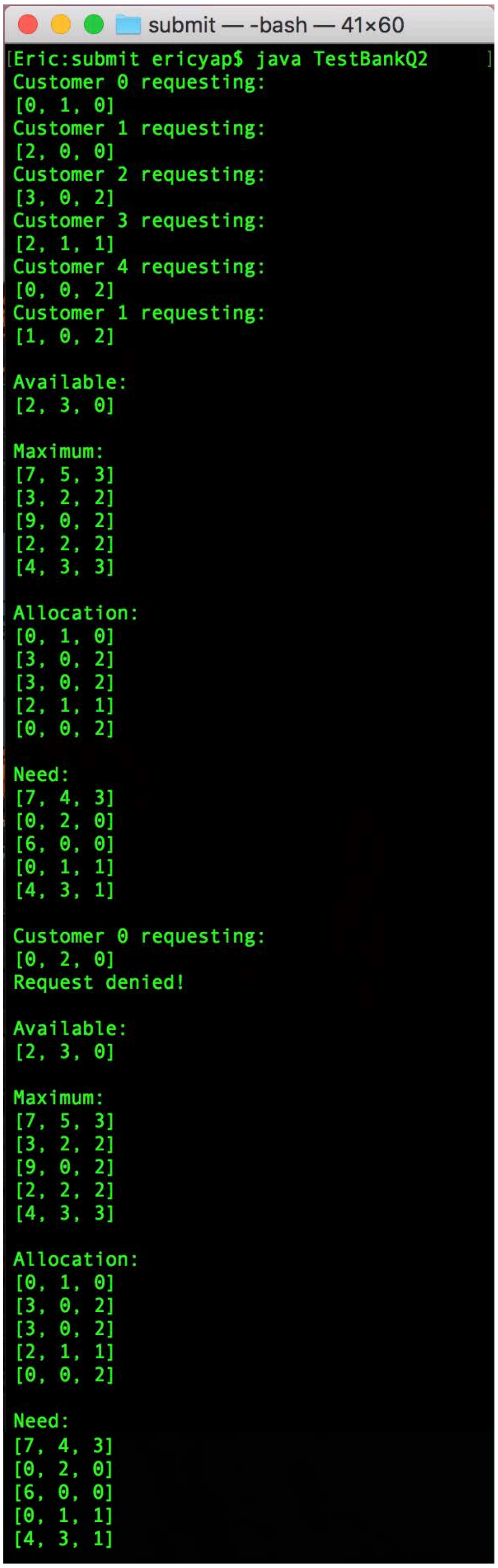
50.005 Computer System Engineering Lab 3 Banker's algorithm
Yap Wei Lok 1002394

<u>Screenshots</u>



submit — -bash — 41×39

```
Eric:submit ericyap$ java TestBankQ1
Customer 0 requesting:
[0, 1, 0]
Customer 1 requesting:
[2, 0, 0]
Customer 2 requesting:
[3, 0, 2]
Customer 3 requesting:
[2, 1, 1]
Customer 4 requesting:
[0, 0, 2]
Customer 1 releasing:
[1, 0, 0]

Available:
[4, 3, 2]

Maximum:
[7, 5, 3]
[3, 2, 2]
[9, 0, 2]
[2, 2, 2]
[4, 3, 3]

Allocation:
[0, 1, 0]
[1, 0, 0]
[3, 0, 2]
[2, 1, 1]
[0, 0, 2]

Need:
[7, 4, 3]
[2, 2, 2]
[6, 0, 0]
[0, 1, 1]
[4, 3, 1]
```



submit — -bash — 41×60

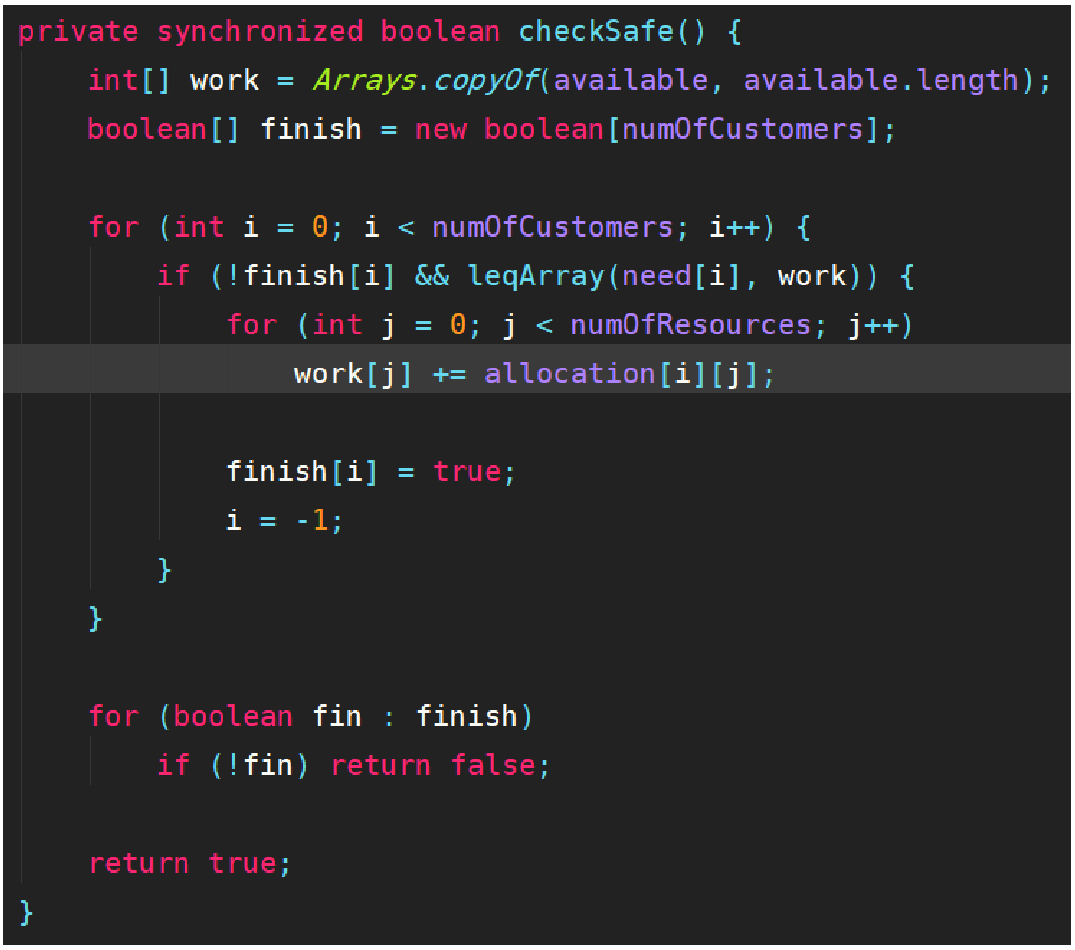
```
[Eric:submit ericyap$ java TestBankQ2          ]
Customer 0 requesting:
[0, 1, 0]
Customer 1 requesting:
[2, 0, 0]
Customer 2 requesting:
[3, 0, 2]
Customer 3 requesting:
[2, 1, 1]
Customer 4 requesting:
[0, 0, 2]
Customer 1 requesting:
[1, 0, 2]

Available:
[2, 3, 0]

Maximum:
[7, 5, 3]
[3, 2, 2]
[9, 0, 2]
[2, 2, 2]
[4, 3, 3]

Allocation:
[0, 1, 0]
[3, 0, 2]
[3, 0, 2]
[2, 1, 1]
[0, 0, 2]

Need:
[7, 4, 3]
[0, 2, 0]
[6, 0, 0]
[0, 1, 1]
[4, 3, 1]

Customer 0 requesting:
[0, 2, 0]
Request denied!

Available:
[2, 3, 0]

Maximum:
[7, 5, 3]
[3, 2, 2]
[9, 0, 2]
[2, 2, 2]
[4, 3, 3]

Allocation:
[0, 1, 0]
[3, 0, 2]
[3, 0, 2]
[2, 1, 1]
[0, 0, 2]

Need:
[7, 4, 3]
[0, 2, 0]
[6, 0, 0]
[0, 1, 1]
[4, 3, 1]
```

The screenshot on the left shows the output for q1 while the one on the right displays the output for q2.

On the second result, customer 0 requested [0, 2, 0] but was denied because no safe state can be found. Therefore, the available resources, maximum demanded resources, allocation and needed resources remain the same after the request.

Analysis

Let n be the number of customers and m be the number of resources.



```java
private synchronized boolean checkSafe() {
    int[] work = Arrays.copyOf(available, available.length);
    boolean[] finish = new boolean[numOfCustomers];

    for (int i = 0; i < numOfCustomers; i++) {
        if (!finish[i] && leqArray(need[i], work)) {
            for (int j = 0; j < numOfResources; j++)
                work[j] += allocation[i][j];

            finish[i] = true;
            i = -1;
        }
    }

    for (boolean fin : finish)
        if (!fin) return false;

    return true;
}
```

Looking at the checkSafe function, there is a nested for loop in the function, the outer loop being $O(n)$ since it goes through the $numOfCustomers$ and the inner loop being $O(m)$, going from 0 to $numOfResources - 1$. The leqArray function also has $O(m)$ complexity since it loops through the two arrays which are $numOfResources$ long to compare the array elements. The complexity so far is $O(n * 2m)$ or $O(n * m)$ to be short.

However, every time the execution goes into the <if> statement, the counting integer i gets reset to -1. This means that in a worst-case scenario, the outer loop will go through $2 * numOfCustomers$. Therefore, the whole control flow has $O(n * n * m)$ complexity.

There is also another loop that checks for the finish status of all customers, so the complexity is $O(n * n * m + m)$ to be more specific, but $O(n * n * m)$ suffice to describe the function's complexity. The checkSafe function is also embedded in the requestResources function but is not nested inside any loop so it does not change the overall complexity of the Banker's algorithm.

Overall complexity: $O(n * n * m)$