Get started Open in app

Mithoon Kumar

111 Followers About Follow

Design Call Centre (OOD)

Mithoon Kumar Jun 5, 2020 · 4 min read

This is my first article on medium. I recently came across this question in an interview. I am writing this article for people who are preparing for a low-level design interview.

Problem: There is a call center in which there is a hierarchy of employees who can attend the call. Freshers should attend the call first, followed by leads and finally the manager. If the fresher wants to escalate the call he/she can escalate it to the leads. Also, attend the call, the caller should wait in the queue. Once an employee is free he/she can take the call.

Let's try to solve the problem

What are the classes we can think of to define in the system

1. **Employee** (`Fresher`, `Lead`, `Manager`)
2. **Customer**
3. **Call**
4. **CallManager**

Here we will have an `Employee` class. `Fresher`, `Lead`, and `Manager` classes will be extending `Employee` class.

Let's see how our `Employee` class will look like. It will have properties like `name`, `free` (either true or false), `rank` (to identify whether it is a fresher, a lead or a manager), and an instance of `CallManager` class. What behaviors we can think about this class? It can receive a call, end a call, and escalate a call. Below is the definition of the employee class.

```
public class Employee {
    private String name;
    private boolean free;
    protected CallManager callManager;
    protected int rank;

    public Employee(String name, boolean free, CallManager
callManager) {
        this.name = name;
        this.free = free;
        this.callManager = callManager;
    }

    public void receiveCall(Call call) {
        this.free = false;
        System.out.println("Call received by employee "+ this.name +
" for customer " + call.getCustomer().getName());
    }
```

```
        this.callManager.callEnded();
    }

    public boolean isFree() {
        return free;
    }

    public void setFree(boolean free) {
        this.free = free;
    }

    public int getRank() {
        return rank;
    }

}
```

The property `rank` is used to know the hierarchy of the employee. Fresher is given a rank of 1, lead is given a rank of 2 and the manager is given a rank of 3.

I can think of defining subclasses `Fresher`, `Manager`, and `Lead` which will be extending `Employee` class. Let's look at the structure of the subclasses. `Fresher` and `Lead` subclasses can escalate a call and the `Manager` class can handle all calls.

```
public class Fresher extends Employee {

    public Fresher(String name, boolean free, CallManager
callManager) {
        super(name, free, callManager);
        this.rank = 1;
    }

    public void escalateCall(Call call) {
        this.setFree(true);
        call.setRank(call.getRank()+1);
        this.callManager.callHandler(call);
    }
}

public class Lead extends Employee {

    public Lead(String name, boolean free, CallManager callManager) {
        super(name, free, callManager);
        this.rank = 2;
    }
```

```
        this.callManager.callHandler(call);
    }
}

public class Manager extends Employee {

    public Manager(String name, boolean free, CallManager
callManager) {
        super(name, free, callManager);
        this.rank = 3;
    }
}
```

We also need to define a `Call` class and a `Customer` class. The `Customer` class will have information about the customer who has made the call. `Call` class will store the information about the call. It will have properties like `rank` and `customer`. Following are the definitions of Call and Customer classes.

```
public class Call {
    private int rank;
    private Customer customer;

    public int getRank() {
        return rank;
    }

    public void setRank(int rank) {
        this.rank = rank;
    }

    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(Customer customer) {
        this.customer = customer;
    }
}

public class Customer {
    private String name;
    private String email;
    private Integer phoneNo;

    public Customer(String name, String email, Integer phoneNo) {
        this.name = name;
        this.email = email;
    public String getName() {
        return name;
    }
    //other getters and setters

}
```

Now we need a `CallManager` class that will actually manage all calls. `CallManager` maintains a list of employees of different hierarchies. Also, it maintains a queue to store the callers on the waiting list. It has methods like callHandler, getFreeEmployee, handleCallFromQueue, and callEnded. `CallHandler` method handles all new calls. When all the employees are busy taking calls, the call is pushed in the queue. Again when an employee becomes free the call manager checks for any call in the queue and assigns the call to some employee.

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;

public class CallManager {
    private static int levels = 3;
    private ArrayList<Employee>[] employeesList;
    private Queue<Call> callQueue;

    public CallManager(int numberOfFreshers, int numberOfLeads) {
        this.employeesList = new ArrayList[levels];
        this.callQueue = new LinkedList<>();
        ArrayList<Employee> freshersList = new ArrayList<>();
        for (int index = 0; index < numberOfFreshers; index++) {
            freshersList.add(new Fresher("Fresher" + index, true,
this));
        }
        this.employeesList[0] = freshersList;
        ArrayList<Employee> leadsList = new ArrayList<>();
        for (int index = 0; index < numberOfLeads; index++) {
            freshersList.add(new Lead("Lead" + index, true, this));
        }
        this.employeesList[1] = leadsList;
        ArrayList<Employee> managersList = new ArrayList<>();
        managersList.add(new Manager("Manager", true, this));
        this.employeesList[2] = managersList;
    }

    public Employee getFreeEmployee(int rank) {
        for (int index = rank-1; index<levels; index++) {
            ArrayList<Employee> employees =
this.employeesList[index];
```

```
                }
            }
            return null;
        }

        public void callHandler(Call call) {
            Employee employee = this.getFreeEmployee(call.getRank());
            if (employee != null) {
                call.setRank(employee.getRank());
                employee.receiveCall(call);
            } else {
                callQueue.add(call);
            }
        }

    public void handleCallFromQueue() {
            if (callQueue.size() > 0) {
                Call call = callQueue.peek();
                int callRank = call.getRank();
                Employee employee = getFreeEmployee(callRank);
                if (employee != null) {
                    callQueue.remove();
                    employee.receiveCall(call);
                }
            }
        }

        public void callEnded() {
            this.handleCallFromQueue();
        }

    }
```

Thanks for reading :)

System Design Interview · Low Level Design · Java · Interview Questions