**Problem Description**

A business startup specialized in software development with team members from different countries need an efficient way of handling jobs entrusted to them. Nonetheless, they have been facing difficulties managing the various jobs given to them because of a difficulty in maintaining effective communication to enhance their workflow. Thus, they decided to code a software that would allow them to easily track their different tasks and progress, and to know who exactly is doing what and when.

To address this issue, the team, of which we are part, decided to code a task management system called Master, which is a representation of the hierarchical breakdown of the company.

The Managing Director**:** He has an admin account that allows him to overview everything in the system, admit and remove team members from platforms and suggest improvements for the various team members.

The Team Leads**:** The team leads will receive project descriptions and assign tasks to the various team members under them. They will be able to view the completion state of each team member’s task and assign them to new tasks. They will report to the managing director after completing each task

Team Members:They will receive tasks from their team leads and will be able to receive and send help requests from other team members on their team and would have the liberty to accept or reject help requests. They will mark their tasks as done when completed. Over time, team members will be able to create sub teams of which they will be heads to handle volunteers and interns in the company.

**Detailed Description of solution architecture**

The program makes use of various data structures to ensure the smooth implementation of the system. These data structures include; trees, arraylists and sets. These specific data structures were chosen due to their suitability to the tasks they were needed to perform.

Our solution allows users to construct the hierarchy of a company based on the employee and team objects that we have constructed. Users will be able to edit and display this hierarchy as a function of our program. In order to achieve this functionality we made use of the tree data structure. This is because trees are structures that store data in a hierarchical order. They also provide moderate access/search(quicker then linked list) and moderate insertion and deletion (quicker than arrays). This made them a suitable structure to implement our hierarchy functionality.

The program also contains a number of lists which hold the names of employees, teams and other data. These lists need to be accessed in efficient time to facilitate quick display of information to the user. In order to achieve that we used the ArrayList structure which features constant time element access and this makes them a suitable data structure to support this functionality.

We implemented our own tree with a node subclass to handle the creation of the company hierarchy. This enabled us to build each node of the tree as a sort of tree on its own with a list of children which could be empty in the case where the node is a single team member rather than a tree.

Text

Description automatically generated

Text

Description automatically generated

Lastly, the use of sets was determined by the need to maintain a list which depended that needed to have only distinct elements. The set interface was therefore used, and the implementation we decided on was the LinkedHashSet. This is because, this set implementation allows the order of elements to be maintained and this feature is beneficial to our program since we are keeping track of the order in which these various objects are created. The LinkedHashSet is an ordered version of HashSet that maintains a doubly-linked List across all elements. When the iteration order is needed to be maintained this class is used.

Text

Description automatically generated

**UML Diagram**

|  |
| --- |
| Employee |
| - name: String  - employeeCode: String  - codeGenerator: static Integer  - codeGeneratorHelp: static Integer  - tenure: Integer  - noTasksCompleted: Integer  - currTask: String  - qualification[ ]: String  - position: String  - allTeams[ ]: String  - currTeams[ ]: String |
| + Employee(name)  + Employee(name, employeeCode, tenure, noTasksComplemented)  + getEmployeeName(): String  + changeEmployeeName(name): void  + getEmployeeCode(): String  + getTenure(): Integer  + getNoTasksCompleted(): Integer  + incrementNoTasksCompleted(): void  + getCurrTask(): String  + assignTask(newTask): Boolean  + getQualifications(): String[]  + addQualifications(newQualis): void  + removeQualifications(position): void  + getPostion(): String  + changePosition(newPosition): void  + getCurrTeams(): String[ ]  + getAllItems(): String[ ]  + assignToTeam(team): void  + removeFromTeam(team): void  + getCurrNoTeams(): Integer  + toString(): String  + printFullDetails(): void  + main(args): void |

|  |
| --- |
| Team |
| - teamName: String  - teamCode: String  - department: Team  - noTeamMembers: Integer  - description: String  - teamLead: Employee  - teamMembers[ ]: Team  - subTeams[ ]: Team  - isDepartment: Boolean  - codeGenerator: String  - codeGeneratorHelp: Integer |
| + Team(teamName)  + Team(teamName, department, description, teamLead)  + getTeamName(): String  + changeTeamName(): void  + getTeamCode(): String  + getDepartment(): Team  + changeDepartment(): void  + getNoTeamMembers(): Integer  + getTeamDesc(): String  + getTeamLead(): Employee  + changeTeamLead(newTeamLead): void  + getTeamMembers(): Employee[ ]  + addTeamMember(newTeamMember): Boolean  + isInTeam(reqEmployeeCode): Boolean  + removeFromTeam(codeToRemove): Boolean  + getSubTeams(): Team[ ]  + addSubTeam(teamToAdd): Boolean  + isSubTeamInTeam(reqTeamCode): Boolean  + removeSubTeamFromTeam(codeToRemove): Boolean  + getActiveTasks(): String[ ]  + makeDepartment(): void  + toString(): String  + printFullDetails(): void  + main(args): void |

Results

The code was able to model a company architecture and allow for assignment of tasks to different team members. Moreover, the program allowed the user to display the company hierarchy and the information of each team into webpages with a clear hierarchical display when they needed to. This covers up for the display limitation involved with printing lots of data in the command line.

Thus, our users can use our program to easily represent teams and tasks. Our program can be extended for use in genealogy and historical representations to get a creative display of a family’s or country’s descendancy and the major events that each family member or pioneer of the nation accomplished represented as tasks to ease education.

Data Used

Text

Description automatically generated

We created a company – named Facebook – with three departments, namely Software Engineering, Social Media and Human Resource. These departments were then assigned subteams, Feature Development and Feature Shipping were assigned to Software Engineering and Brand Awareness was assigned to Social Media. These subteams were assigned different team members who are employees of this company. When the company hierarchy structure was displayed the result looked like this:

Shape

Description automatically generated with low confidence

Conclusion

Our program, therefore, confirms that a tree is an efficient data structure for creating non-linear hierarchies such as companies and for managing the flow of information from one part of the tree to another. For further improvements, we consider adding a functionality to allow employees to accept or decline assigned tasks and to allow collaboration between different teams and team members. Also, we consider making the platform web based to serve as an online collaboration site for projects of various types.