A FACTORY MANAGEMENT SYSTEM

BY

ERIC WOULDS

FADI ALMAKAHLEH

JORDAN ANTHONY DROST

ALADE ANOMA EMMANUEL

PROJECT OVERVIEW

The project is a simple factory Management System. A factory management system contains various features to help organize and manage the activities of a company or factory. Features of a factory management system include employee management, payroll management, product management, inventory management, notifications, amongst others.

For the scope and purpose of this project, we chose to focus on three features of the factory management system: employee management, payroll management, and the notification system.

Activities that can be performed on the system include addition and removal of new employees, assignment to existing managers and carpools, updating employee information, and calculating an employee's pay, amongst other things.

DESIGN PATTERNS

Three design patterns have been employed during the implementation of this project. The patterns are composite pattern, decorator pattern, and the Observer pattern. The patterns provide flexibility and are easily extendable for adding new features or modifications of existing features.

OBJECTIVES OF THE PATTERNS

The composite pattern is selected because it allows for the creation and management of a hierarchical structure for the employees of the company. In addition, this provides a multi-level layer categorization for an employee's role and direct reports, such as CEO, managers, drivers, salesperson, to mention but a few.

The composite pattern also lets us identify and treat each employee differently, such as assigning an employee to a carpool.

The decorator pattern is selected to be able to perform payroll-related operations on an employee without directly modifying or manipulating the employee class.

The decorator pattern also helps us keep the Single Responsibility principle by not having the employee class handle both employee and payroll operations.

The observer pattern is selected because it allows the application to provide feedback to the user based on the updates or actions made by the user on the user interface.

IMPLEMENTATION

The employee class is created as a base for all the employees, and on this base class, we have two main types of employees: the carpool team and the managers.

The carpool is responsible for updating and assigning cars/drivers to drive an employee around when needed. The manager class is responsible for creating and removing employees from the system. Thanks to the composite pattern, once an employee is added/removed from a manager, the list of subordinates under the manager is automatically updated.

The payroll class relies on the shared IEmployee interface and calculates all employee payroll-related activities, including the weekly salary, monthly salary, and bonus. It also identifies if the user is a CEO or not when calculating the bonus and uses that to determine the amount of an employee's bonus.

Two observer classes, EmployeeSalaryObserver and EmployeeAddressObserver, were created for each editable field, salary, and address, respectively. Both classes implement the observer interface and are responsible for sending an informative message, including an employee's name, once an observer event happens in the system.

Each system module is connected and linked together via a simple user interface where the data can be accessed. The user interface connects and incorporates the modules to provide a single point of access for the user without going directly into the code.

BENEFITS OF THE SELECTED PATTERN TO THE APPLICATION

The benefits provided by the patterns include:

1. In a single glance, the composite pattern allows one to see an employee level or hierarchy in the system. This includes the role, the sublevel below the employee, the subordinates where applicable.
2. The composite pattern also provides ease for introducing more hierarchy or more positions without modifying the existing hierarchy and positions.
3. The decorator pattern allows the application to effectively extend the employee object to modify the employee class or object as changes made to payroll do not affect the employee class.
4. The decorator pattern also provides flexibility in calculating and using the payroll object.
5. The observer pattern allows the application to have a defined communication given for the user's feedback of the two UI update buttons.
6. The observer pattern also allows defined alerts per update triggered only when an actual update occurs. Random press of the buttons alone does not trigger the observer.

EXTENDABILITY

The program is extendable in different ways. Some of the ways in which the system can be extended includes:

1. More features can be added to the application without breaking current features.
2. The application can be connected to a database or a more permanent form of storage.
3. More employee hierarchies can be added to the employee management, all inheriting from the employee class and none breaking the other classes implementing the employee class.
4. The payroll feature can be extended to calculate more payroll-related functions such as employee pension, severance package, automatic payout, and a few.
5. More observers can be created to observe and notify different stakeholders when an event occurs.
6. The observer behavior can also be modified/extended for improved and more personalized notifications.

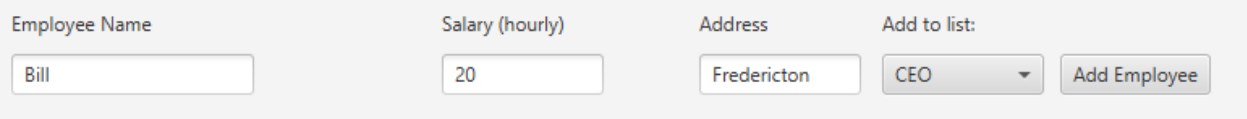
HOW TO RUN

Each row of elements corresponds to a function that can be done.

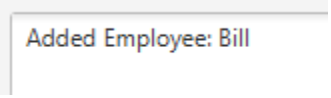
To Add Employee:

Here is the row of elements to add an employee.

To add an employee, fill out each field with the correct information and then select a manager list to add the new employee too.



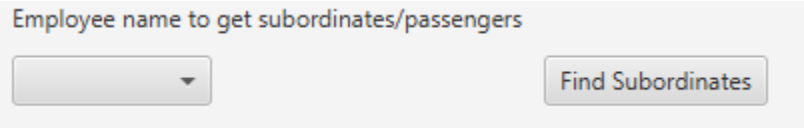
Then click the “Add Employee” Button, if successful you will be greeted by the message below confirming the employee was added.



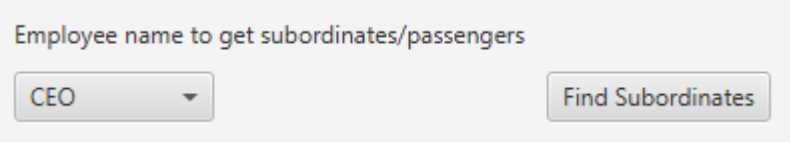
If you don’t fill out all the correct fields, you will be met by a message asking to provide all the information needed.

Get A List of Subordinates/Passengers

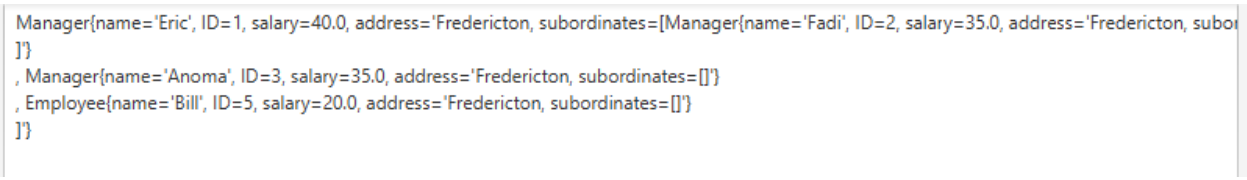
Here is the row of elements for getting a list of subordinates or the list of passengers depending on your selection.



Select an item from the list and hit “Find Subordinates” button to get the list.

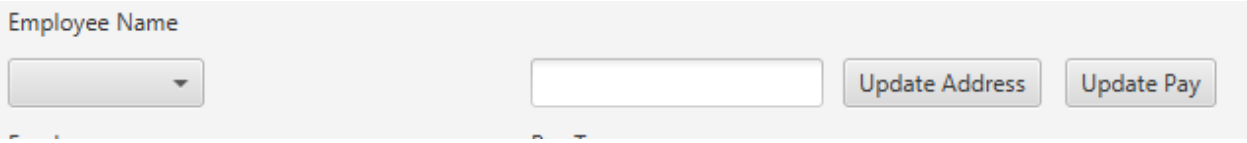


The output should look something like the example below depending on the list you chose.

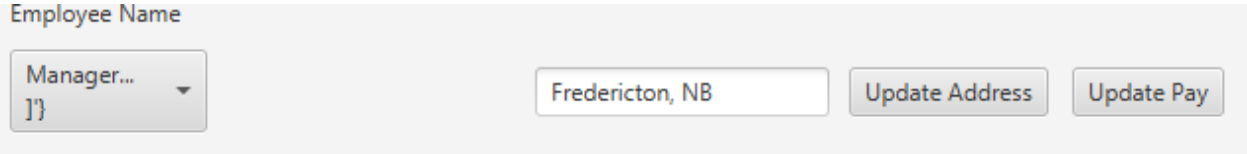


Update Employee Record

Here is the row of elements that are used to update an existing employee’s pay or address.



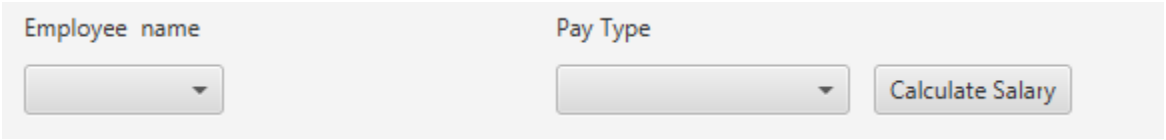
You can update an employees pay or address depending on the input and which button you press



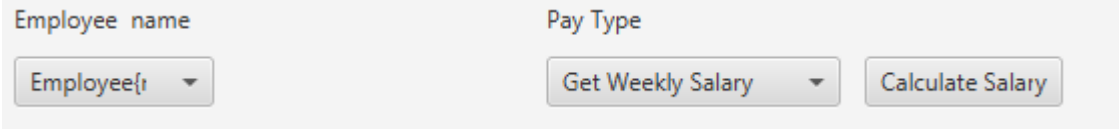
You will be greeted with a message from the observer in the output text area if successful.

Calculate an Employee's Salary

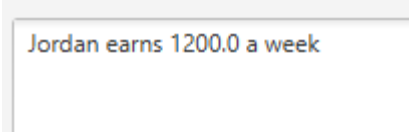
This row of elements is for figuring out an employee's different salaries and this is using the decorator classes.



To get an employee's salary select the employee from the drop-down list and then select what salary you would like to calculate (hourly, weekly, monthly, bonus) and then click "Calculate Salary" button.

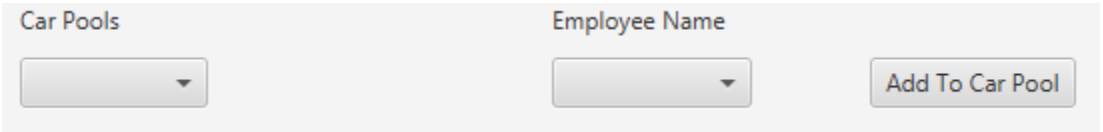


The output will show the employee's pay with the correct calculation depending on your selections

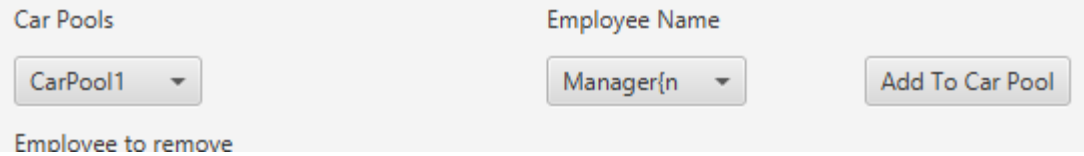


Add An Employee to a Carpool

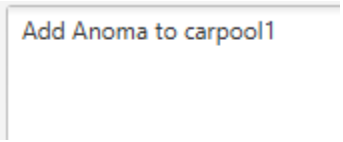
You can also add an existing employee to a carpool, this is the row for adding an employee to a carpool.



Select the carpool you would like to add an employee to and then select an employee before hitting "Add to Car Pool" Button.

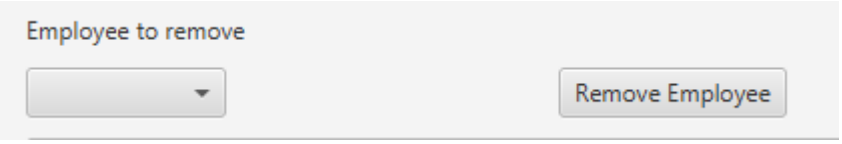


You will see an output similar to below if everything was successful.

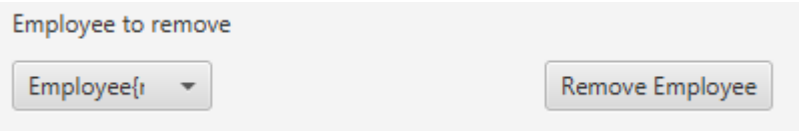


Remove an Employee

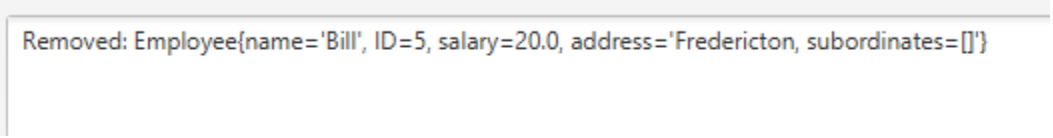
Finally, you can remove any employee other then the Managers. Here is the row of elements for doing so.



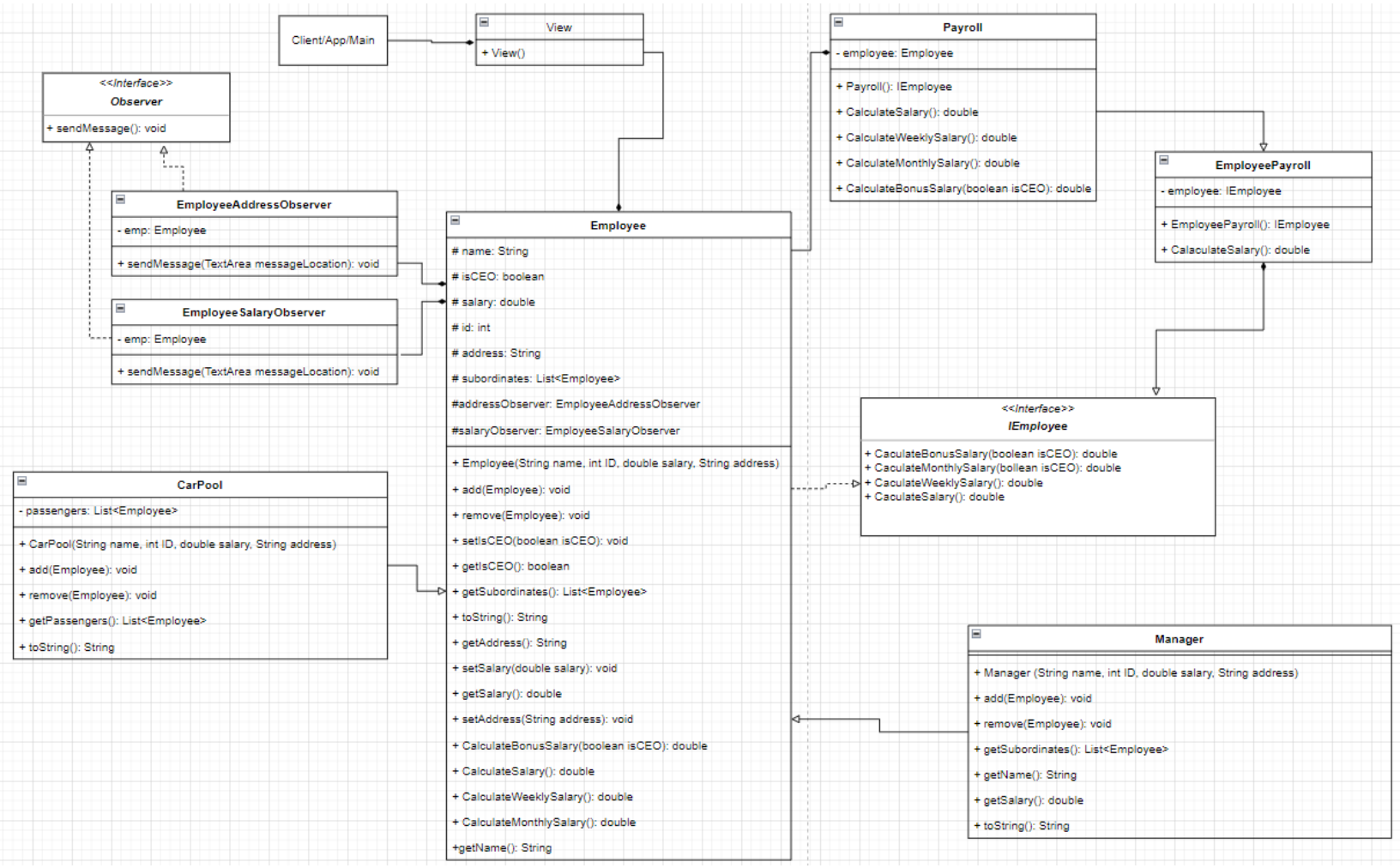
Select an employee from the employee list and then click the “Remove Employee” button.



Here is a message for the output after removing an employee.



UML DIAGRAM



MEMBERS CONTRIBUTION

For this project we all did an equal share of the work and therefore the contribution percentage for each member of the group was 25%.