AYDIN ADNAN MENDERES UNIVERSITY

ENGINEERING FACULTY

COMPUTER ENGINEERING DEPARTMENT

OKTAY CAN SEVİMGİN - **221805001**

CSE 203 Object-Oriented Programming Project

Problem Description

The primary task of the project is to design and implement a software solution for Factory X, which is aimed at calculating the weekly payment amounts for all types of employees. Factory X employs four distinct categories of workers: salaried employees, hourly employees, commission-based employees, and base-plus-commission employees. Each of these employee types has a unique structure for their payment calculation. For instance, salaried employees receive a fixed salary regardless of the hours worked, while hourly employees are paid based on the hours worked, including overtime pay for any hours exceeding 40 per week. Commission employees are compensated with a percentage of their sales, and base-plus-commission employees receive both a base salary and a percentage of their sales. The task is to create a program that will compute these payment amounts accurately based on the employee type and store the resulting data in a file.

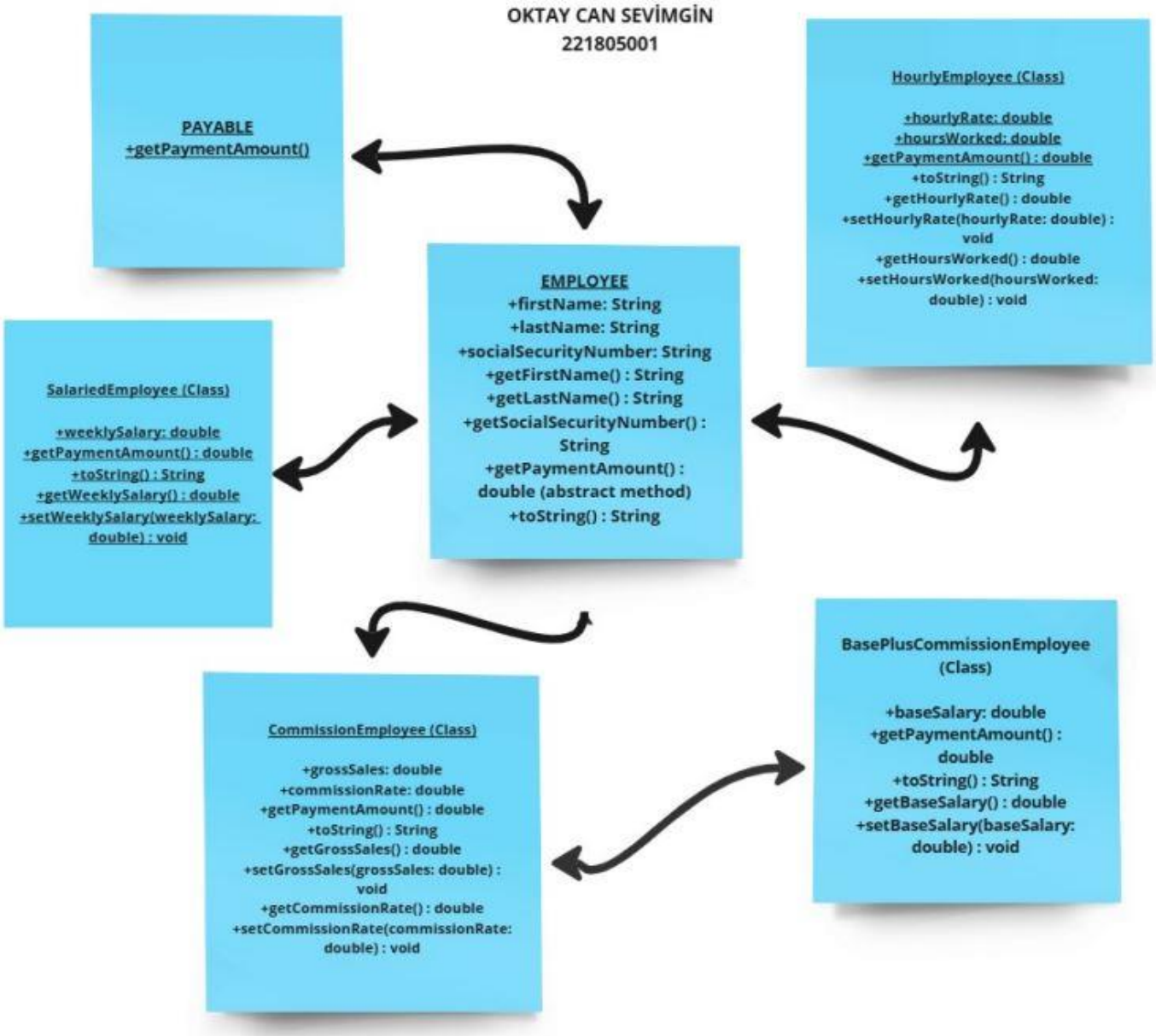
In order to achieve this, the solution follows a well-structured object-oriented design that includes the creation of several classes. The core of the design involves an abstract Employee class that holds common properties such as first name, last name, and social security number (SSN). Subclasses of this Employee class will be created for each type of employee: SalariedEmployee, HourlyEmployee,

CommissionEmployee, and BasePlusCommissionEmployee. Each of these subclasses will implement the getPaymentAmount() method differently to handle the specific calculation logic for each employee type.

The program will also need to handle the reading and writing of employee data to a file. This can be either in the form of a text file or using random access files, depending on the requirements. The data will be stored in a file, and the program must be able to read from and write to this file, ensuring that the records are updated whenever an employee is added or modified. To facilitate these operations, the program will include a graphical user interface (GUI) that allows users to interact with the system easily. The GUI will enable users to add new employees, search for existing employees by SSN, update employee records, and clear form fields as needed. These user actions will trigger the relevant methods to update the employee data and file records.

The program's input will be provided by the user through the graphical interface. Users will enter employee details, such as employee type, salary, hourly wage, commission rate, etc., based on the selected employee type. The program will then calculate the employee’s payment amount according to the rules for their specific type and display this information. Additionally, the program will ensure that the employee records in the file are properly updated with any changes, additions, or deletions made through the interface.

UML Diagram


OKTAY CAN SEVİMGİN
221805001
PAYABLE
+getPaymentAmount()
HourlyEmployee (Class)
+hourlyRate: double
+hoursWorked: double
+getPaymentAmount() : double
+toString() : String
+getHourlyRate() : double
+setHourlyRate(hourlyRate: double) : void
+getHoursWorked() : double
+setHoursWorked(hoursWorked: double) : void
EMPLOYEE
+firstName: String
+lastName: String
+socialSecurityNumber: String
+getFirstName() : String
+getLastName() : String
+getSocialSecurityNumber() : String
+getPaymentAmount() : double (abstract method)
+toString() : String
SalariedEmployee (Class)
+weeklySalary: double
+getPaymentAmount() : double
+toString() : String
+getWeeklySalary() : double
+setWeeklySalary(weeklySalary: double) : void
CommissionEmployee (Class)
+grossSales: double
+commissionRate: double
+getPaymentAmount() : double
+toString() : String
+getGrossSales() : double
+setGrossSales(grossSales: double) : void
+getCommissionRate() : double
+setCommissionRate(commissionRate: double) : void
BasePlusCommissionEmployee (Class)
+baseSalary: double
+getPaymentAmount() : double
+toString() : String
+getBaseSalary() : double
+setBaseSalary(baseSalary: double) : void

This UML diagram represents an object-oriented design for different types of employees, each with specific characteristics and payment methods. The `Payable` interface ensures that any class implementing it will have the `getPaymentAmount()` method, which standardizes the way payments are calculated across different employee types. The `Employee` class, which serves as an abstract base class, contains common properties such as the employee's first name, last name, and social security number. It also includes the abstract method `getPaymentAmount()`, which must be implemented by subclasses to calculate payments based on the employee's specific compensation structure

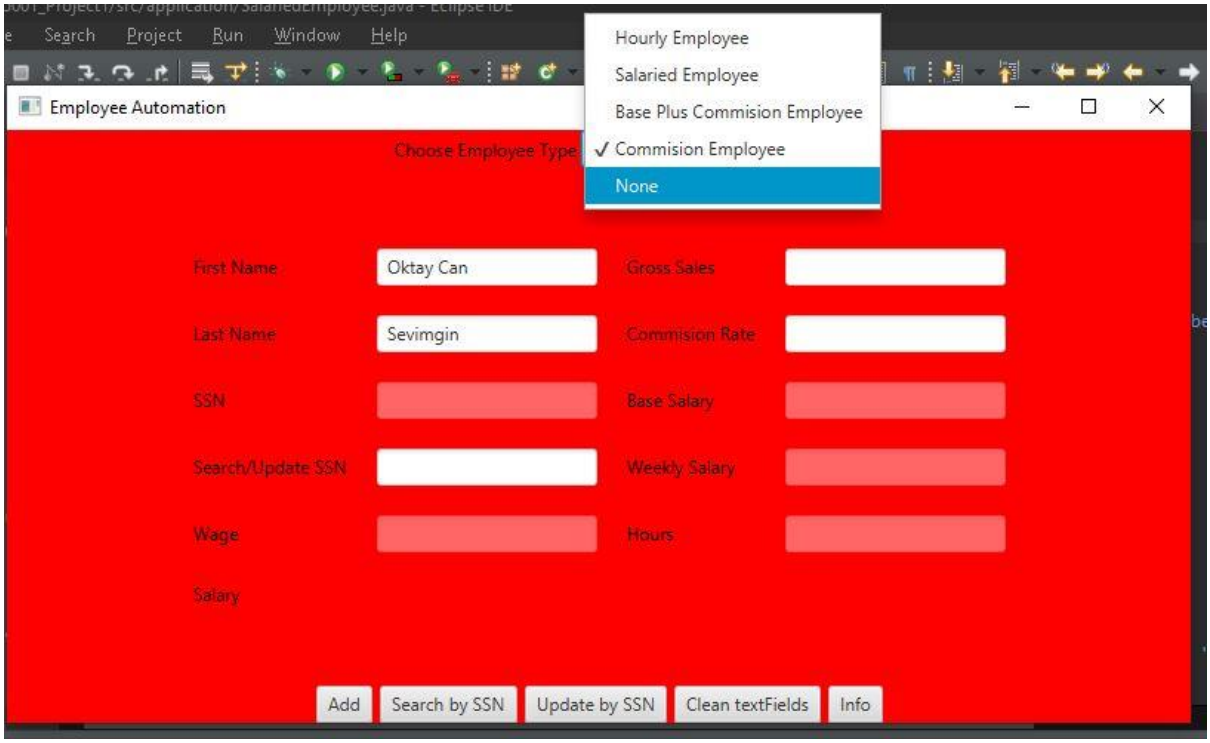
The `SalariedEmployee` subclass extends `Employee` and is designed for employees who receive a fixed weekly salary. It adds a `weeklySalary` property and implements the `getPaymentAmount()` method to return this salary. Similarly, the `HourlyEmployee` subclass represents employees paid hourly. It has properties for `hourlyRate` and `hoursWorked`, and its `getPaymentAmount()` method calculates the payment based on the hourly rate and hours worked.

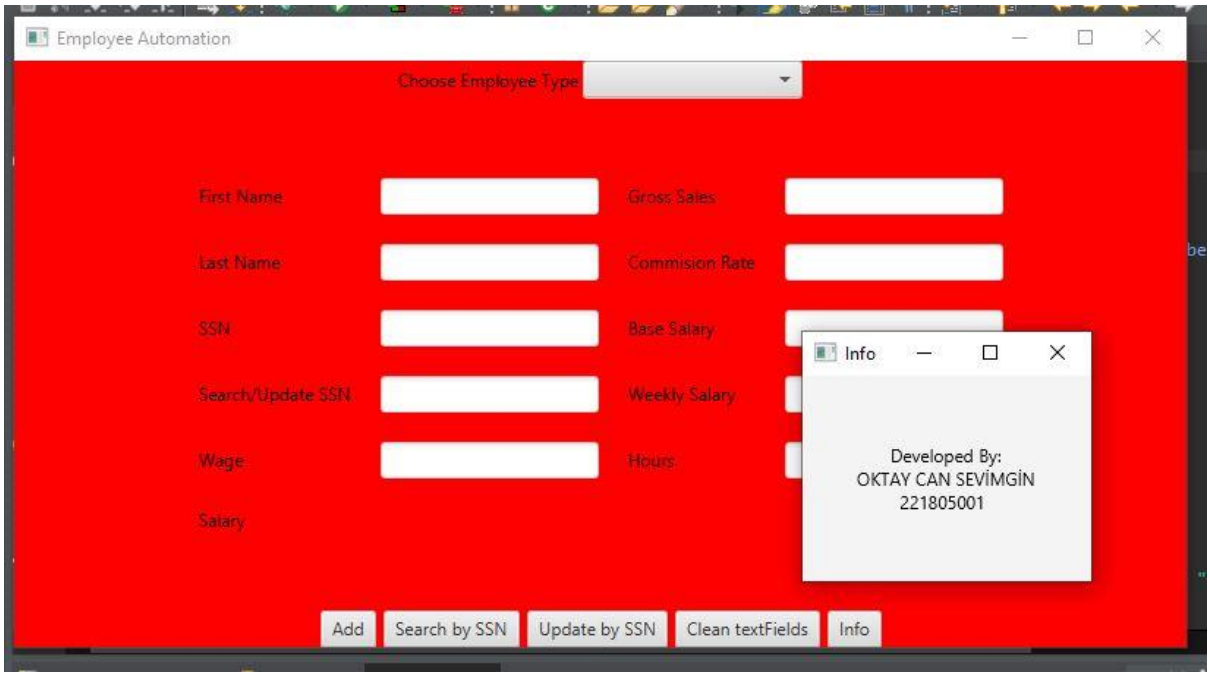
The `CommissionEmployee` subclass is designed for employees who earn commissions based on sales. It includes properties for `grossSales` and `commissionRate`, with the `getPaymentAmount()` method calculating the payment based on these values. The `BasePlusCommissionEmployee` subclass extends `CommissionEmployee` and adds a base salary to the commission payment, with its own `getPaymentAmount()` method that incorporates both the base salary and commission.

Overall, this design follows principles of inheritance and polymorphism, where the `Employee` class serves as a base for different employee types, each implementing the `getPaymentAmount()` method according to their specific pay structure. This approach allows for easy extension if new employee types need to be added in the future, providing flexibility and maintainability. The use of the `Payable` interface also ensures that all employee types can be treated uniformly when calculating payments.

OKTAY CAN SEVİMGİN -
221805001

PROJECT SCREENSHOTS





Testing Strategy

Unit Testing of Classes:

Each class, such as `Employee`, `SalariedEmployee`, `HourlyEmployee`, `CommissionEmployee`, and `BasePlusCommissionEmployee`, should be tested independently to ensure that the methods, especially `getPaymentAmount()` and `toString()`, return the correct results based on the input provided.
For the `Employee` class, since it is abstract, test that subclasses correctly implement the `getPaymentAmount()` method and handle any edge cases like negative or invalid inputs (e.g., negative salary or hours).
Test each subclass's `getPaymentAmount()` method to verify that it calculates the correct weekly payment based on employee-specific rules (fixed salary for salaried employees, hourly wage for hourly employees, commission for commission-based employees, etc.).

Integration Testing:

Once individual components are tested, the next step is to ensure that all classes interact correctly. This involves testing the communication between the `Employee` subclasses and the `Payable` interface, checking that the `getPaymentAmount()` method is correctly overridden and that it can handle all input values as expected.
The GUI's interaction with the backend logic (calculating and displaying payment amounts) should also be tested. Ensure that when an employee is added, updated, or searched by SSN through the GUI, the program correctly manipulates the employee records and calculates the payment amounts.

File I/O Testing:

Test the file operations (read, write, and update) by adding, updating, and searching employee records stored in a file (e.g., text file or random access file). After performing these operations, verify that the employee records are correctly saved and retrieved from the file.
Test the program’s ability to handle invalid file data or non-existent files. For example, if an employee record is added and the program crashes or the file is corrupted, the program should be able to recover from such errors and maintain integrity.

GUI Testing:

Test the GUI's usability and functionality. Ensure that each action (add, update, search, clear text fields) behaves as expected when interacting with the form fields. Validate that only the relevant input fields are enabled based on the employee type selected in the dropdown menu.
Verify that entering incorrect data (e.g., a negative salary or hours) is appropriately handled, either by showing a warning or by preventing the submission of invalid data.
Ensure that the "Read Only" SSN field cannot be edited by the user but is still displayed when performing a search or update.

Boundary and Edge Case Testing:

Test edge cases for various employee types. For example:
For `HourlyEmployee`, test a scenario where an employee works exactly 40 hours and another where they work overtime (more than 40 hours).
For `CommissionEmployee`, verify that the commission rate is calculated correctly for different gross sales values.
For `BasePlusCommissionEmployee`, check the combined effect of base salary and commission.
Test cases where no data is available in the file or when all records are deleted. Ensure the program behaves as expected in such cases (e.g., showing a message when no records are found).

Final Testing with Multiple Employees:

Simulate a scenario where multiple employees of different types are added, updated, and searched. This ensures the program can handle multiple records, maintain correct payment calculations, and update files accurately.

Test Data

SalariedEmployee: Test with a fixed weekly salary, ensuring that the result is correct regardless of input for hours worked.
HourlyEmployee: Test with different hourly rates and hours worked, including overtime scenarios where more than 40 hours are worked in a week.
CommissionEmployee: Test with various gross sales and commission rates to ensure commission calculations are accurate.

BasePlusCommissionEmployee: Test with base salaries, commission rates, and gross sales to verify that both the base salary and commission are properly combined in the payment calculation.

Expected Outcomes

The correct weekly payment amounts should be calculated for all employee types.
Employee records should be correctly added, updated, and saved in the file.
Invalid input or edge cases should be appropriately handled with error messages or validation prompts.
The GUI should display the correct data and allow for easy interaction with the system.

Conclusion

By thoroughly testing the individual components, integrating them, and validating the program's ability to handle various input scenarios, the program will be fully validated for correct functionality. Testing ensures that the program performs as expected, is user-friendly, and can handle real-world use cases.