Hrishikesh AJ
140736

FKPayrollDesign

11th May 2020

Design Objectives and Approaches

1. Focus towards contracts : The project requirements laid out could be modelled into contracts which would be carried out by concrete classes. Creating interfaces for like ‘PayableHourly’ and ‘PayableFlat’ which provided specific features for the employees to implement.
2. Design towards extensibility : Even though the project requirements covered quite a few test cases, however in a real world scenario of managing employee payroll depicted in the project, so many new requirements may arise. The objective of my design was to make it easy to add new features as and when the requirement comes in. This meant loose coupling among classes had to be ensured and entities had to be created in such a way that they can service similar and dissimilar clients. One way this is implemented is the Employee class implementing the Union interface, and HourlyPay implementing PayalbleHourly and MonthlyPay implementing PayableFlat.

Design Role and Responsibilities

1. Employee class to represent a generic employee. Every employee is an instance of this type.
2. PayableHourly to describe the contracts for those employees who are paid by the hour.
3. PayableFlat to describe the contracts for those employees who are paid each month.
4. Union to describe the functionality of all employees part of the employee union.
5. SalesReceipt to implement salaried employees who can post a sales receipt and receive commission based on their sales.
6. TimeCard to provide structured of time cards which can be posted by employees and used to generate payroll.

Design Choices and Preferences with Reasoning

1. Composition over inheritance : First few commits contains HourlyEmployee and MonthlyEmployee inheriting from the Employee class. But I later realised that this was a

bad design choice as it does not support subtraction of class members. Later, the functions to support the payment for employees was modelled as a composition of the classes ‘HourlyPay’ and ‘MonthlyPay’, composed in the Employee class which holds the Hourly pay details and Monthly Pay details of each employee. This way, an employee can change from one payment type to the other also in the future.

2. Focus on contracts: The project requirements laid out could be modelled into contracts which would be carried out by concrete classes. Creating interfaces for like ‘PayableHourly’ and ‘PayableFlat’ which provided specific features for the employees to implement.

Future Design Improvements

1. Introduce Generic classes : Employee should be made a generic class to handle the different ways that an employee can be paid and the different ways that he receives those payments. Many functionalities like generatePayroll(), postSalesReciept() and recievePaymentThroughBank() etc would be similar in all employees, and would be a good application of Generics.
2. Introduce functionalities in contracts for calculating payroll. Currently, payroll is generated in a similar way for both the type of employee payables. However, their requirements are different. To handle those requirements uniquely and to improve the efficiency of generating the payroll, new functionalities have to be modelled into contracts.
3. EmplyessMain class implementing all functionalities is causing tight coupling and single-point-of-failure. Further division of features into specific modules can help decrease coupling, and improve code maintainability.