Documentation

Design Objectives and Approaches

Design objective is to make a payroll system for the employees. Some employees work by the hour, some are paid a flat salary, and some salaried employees are also paid commission on sales. Employees should be able to select method of payment. And some employees belong to Union and will have deductions accordingly. Requirements were clearly mentioned in the problem statement.

My approach for the design was to make classes independent of each other as much as possible. **Decoupling** of objects was my first goal. This can be seen from employee class. I could have directly implemented various types of employees on the basis of their payments, but that choice would have been concrete and was not futuristic. I also tried my best towards **single responsibility design**, and most of the functions designed perform single task only. **Encapsulation** is taken seriously and all the classes have their data members private and methods public. **Polymorphism** can be seen in case of Constructors using which objects can be instantiated based on what arguments are passed.

Design Role and Responsibilities

Though I have not implemented roles and responsibilities in the code due to time constraint, but I had designed 3 types of roles for the system based on **scope design**. An employee can post TimeCards, SalesReceipts of itself based on he/she is a hourly paid or commission based employee. Manager of the employee has all the privilages of the employee and can post Union deductions for the employee. Admin has all the privilages of the employee and Manager and has the responsibility of running payroll everyday, changing details of the employees etc.

Design Choices and Preferences

1. Giving all the privilages to the admin, or design a scoped system based on scope design.
2. Each function should have single responsibility or it may be allowed to perform more than one task.
3. Keeping data members encapsulated or they may be allowed to be seen outside the class.
4. **Design towards abstraction or concrete design**. Concrete design would have harmed the flexibility and future modifications could have been very costly.
5. Objects should be decoupled as much as possible. This improves code flexibility.

Future Design Improvements

UnionMembership class have been implemented concretely. I should have made a contract first for Union and then implemented the contract for various types of Unions. This would have made the code flexible for future changes.

Design Challenges, Experiments and Alternative Design

Major challenge I faced was to connect to database since this was the first I was connecting code to database. Lots of time got wasted in this. Experimentation with designs was done in the beginning and in my first attempt I designed a very lighweight solution which performs all the use cases. But that design was very concrete and even for small modifications, either lots of code changes were there or it was impossible to adapt to change without modifying whole of the code. I build my design by separating different layers of code.

Design Improvements over Existing Design

Scope design can further be improved and group scope can be added to the system. Database design can be improved and it can be made more efficient. Security is not implemented in the design and can be added in the future.