**Analyse & Refactor a class**

|  |  |
| --- | --- |
| **Before** | **After** |
|  |  |
| Rationale:  I split the Employee class into two subclasses: Operator and Admin. Operator and Admin classes have much in common, and due to this reason I kept the original Employee class but made it abstract. The primary difference between Operator and Admin is the set of actions each one is authorized two. According to the Use Case diagram, the Operator is allowed to look-up, check-in, check-out, and view the reports. The Admin is allowed to register a new Customer, edit Customer’s profile, look-up a Customer, and view the reports. To facilitate the distinction between the roles each employee is entitled to (only Admin can register, only Operator can check-in/out), I propose to assign a set of permitted roles (based on the use-cases) to each type of employee. The method isRolePermited() returns a Boolean, indicating the access privileges of a particular employee. In my opinion, the Role class might be represented by the enum type. | |
|  |  |
| Rationale:  To implement the repository pattern, I propose to spin-off CustomerRepository, EmployeeRepository, and VisitRepository from the Pool class. This would realize O-O concept of separation of concerns by limiting the scope of the Pool class to only one thing – being an aggregator of repositories. Each repository is responsible for management of a collection of similar objects (CRUD operations) and for their storage.  Furthermore, I suggest to make the storage-related methods (load() and save()) private. This encapsulation would decouple the business logic from the database access logic and simplify further testing and maintenance. | |