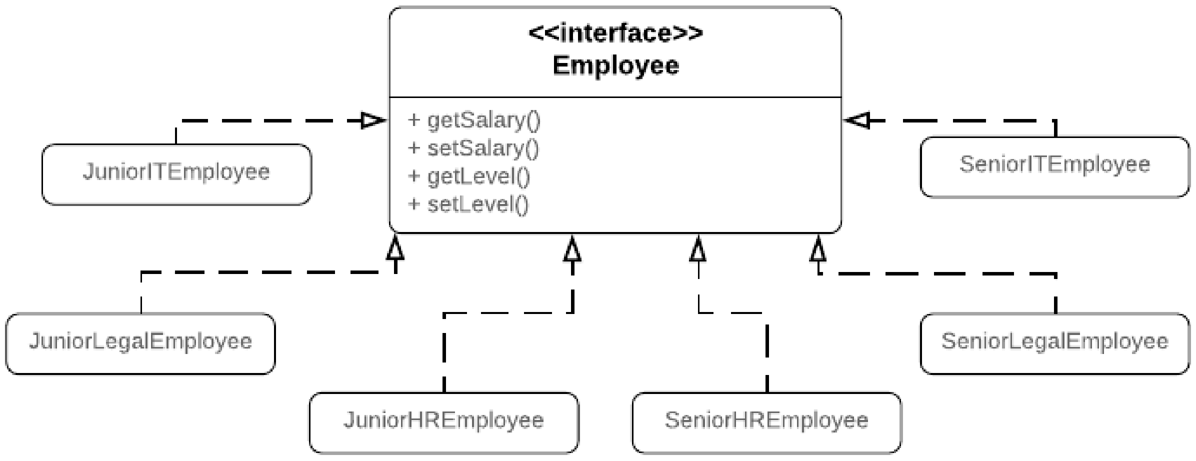
DP Exercise 3



Consider Figure 1 that describes several derived classes of the Employee interface. Each derived class must represent both department and level dimensions of an employee.

1. Identify any flaws in this design citing any solid principle violations
2. Identify a structural design pattern to address this issue.
3. Apply your identified design pattern with the use of a UML diagram.
4. If a new method "Manage()" is to be introduced to any instance of a senior-level employee, in which class should it be defined in your new design?
5. Fork and refactor the codebase to implement your redesign

Codebase: https://replit.com/@Snickdx/DP-Exercise-3

DP Solution 3

1. The number of classes to be greeted grows steadily with the introduction to new employee levels/departments. This makes the codebase harder to manage as functionality is extended.

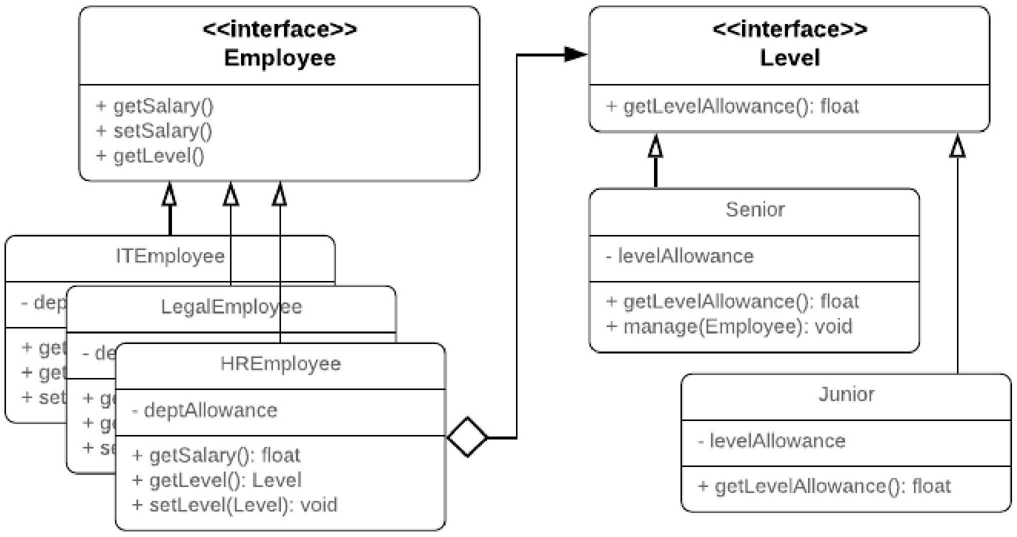
   Lack of abstraction, there may be shared functionality among all employees of a dept but it must be repeated in multiple classes related to that dept. Similarly for level.

   An argument can be made for SRP single class modeling both level and dept.

   The implementation of setLevel would not be straightforward as it would require rebuilding an object of one class from another.

```
//Eg
Employee bob = new JuniorEmployee(); //JuniorEmployee object
bob = bob.setLevel("senior")
// should return a SeniorEmployee object with all of the appropriate
state of old object
```

2. The Bridge Design pattern should be used to divide the class across the department and level dimensions.



3.
4. The manage() method need only introduced in the senior class

Solution: https://replit.com/@Snickdx/DP-Exercise-3-Solution#Employee.java