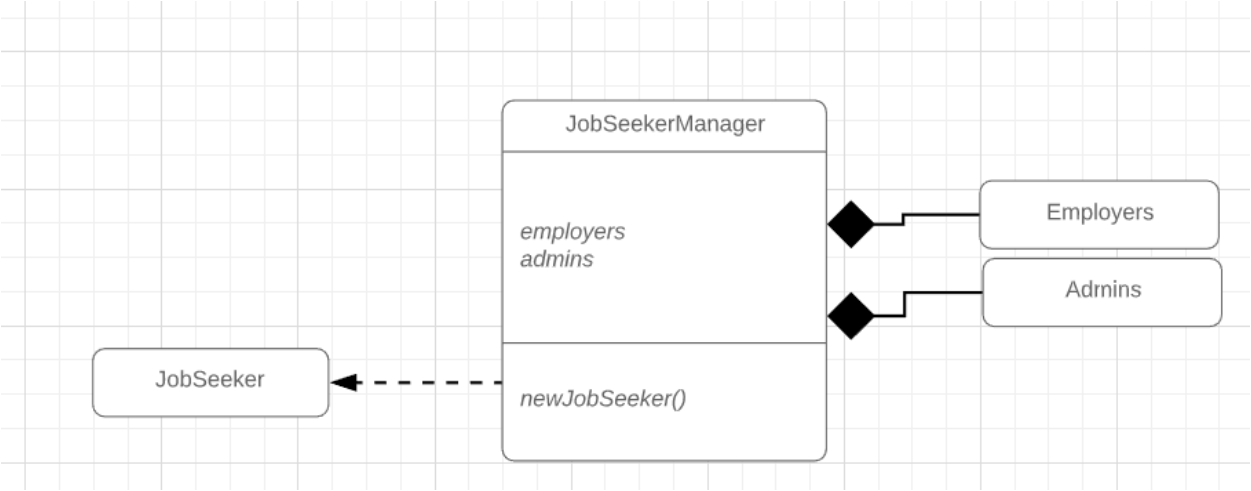
Design Patterns Exercise 1

In a Job Recruitment application, the following highlights the segment of code responsible for allowing Employers to respond to when new JobSeekers join the platform.

Code Base:https://replit.com/@Snickdx/DP-Exercise-1



Discuss any potential issues with the current design with regards to extensibility, modularity or scalability.
In order to introduce new entities that must be notified about new JobSeekers the JobSeeker Manager must be modified to cater for the new entity. This violates open close and affects the extendability of the system.

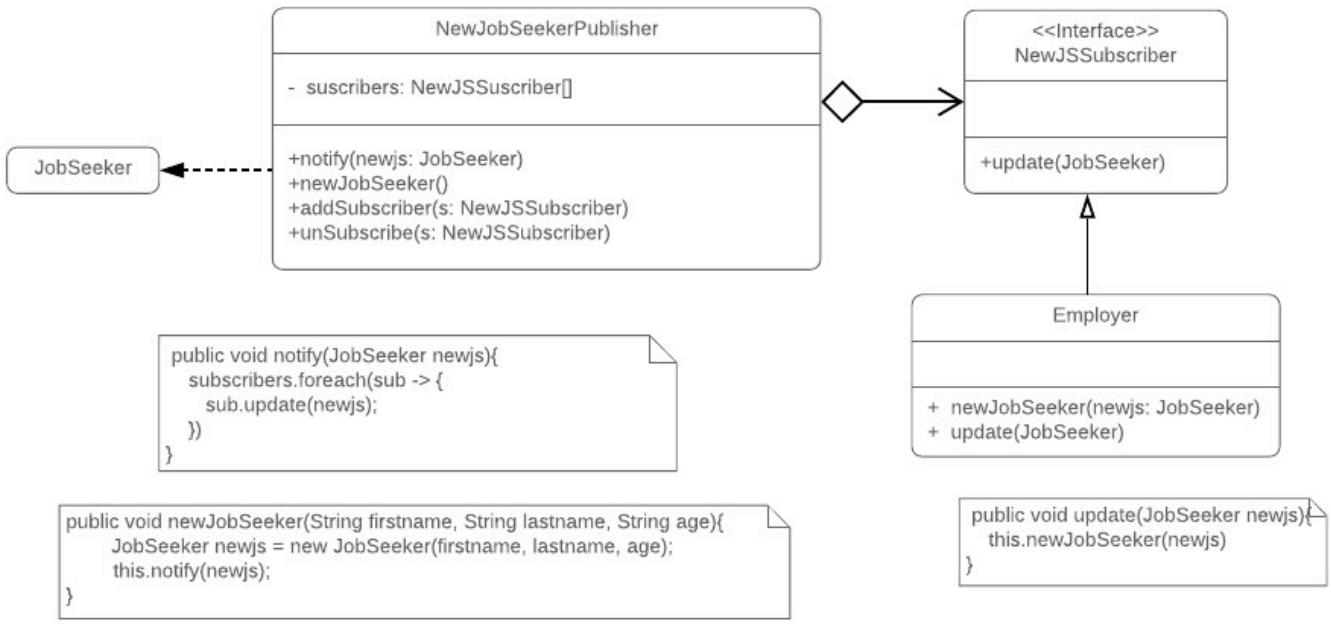
If the JobSeeker manager is allowed to grow in order to cater for new entities, then JobSeeker Manager is at risk of becoming a monolith violating SRP.

The larger the class, the more likely for it to house unused functionality when it is being reused. This goes against the concept of modularity.

Observer/Subscriber: Employers, Admins
Observable/Publisher: JobSeeker Manager
How observers are updated : JobSeekerManager.newJobSeeker() => Admin/Employee.newJobSeeker()

Suggest a refactor to this design via application of a design pattern, provide a UML diagram and code samples of how employers would get updates in your new design.

Solution: Apply the observer pattern so more subscribers can be easily added without affecting the code which dispatches events to them.



Employers would receive updates via the update method of the NewJSSubscriber interface which is invoked in the notify method of the NewJobSeekerPublisher. If other subscribers are added to the application they can subscribe to the NewJobSeekerPublisher.

Solution Code: https://replit.com/@Snickdx/DP-Exercise-1-Solution