**0. Task Description (russian)**

**Есть компания, у компании могут быть сотрудники. Сотрудник характеризуется именем, датой поступления на работу, базовой ставкой (для простоты, это значение по-умолчанию одинаково для всех видов сотрудников).**

Сотрудники бывают 3 видов - Employee, Manager, Sales. У каждого сотрудника может быть начальник. У каждого сторудника кроме Employee могут быть подчинённые.

Зарплата сотрудника Employee - это базовая ставка плюс 3% за каждый год работы в компании, но не больше 30% суммарной надбавки

Зарплата сотрудника Manager - это базовая ставка плюс 5% за каждый год работы в компании (но не больше 40% суммарной надбавки за стаж работы) плюс 0,5% зарплаты всех подчинённых первого уровня

Зарплата сотрудника Sales - это базовая ставка плюс 1% за каждый год работы в компании (но не больше 35% суммарной надбавки за стаж работы) плюс 0,3% зарплаты всех подчинённых  всех уровней

У сотрудников (кроме Employee) может быть любое количество подчинённых любого вида.

**Требуется**: спроектировать объектную модель, а также реализовать алгоритм расчета зарплаты каждого сотрудника на произвольный момент времени (а также подсчёт суммарной зарплаты всех сотрудников фирмы в целом) (пользовательский интерфейс на выбор, это не существенно для данной задачи).

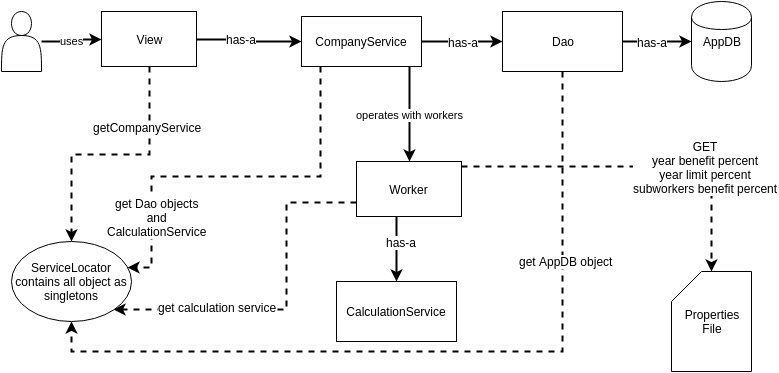
Система должна быть проверена unit-testами (не обязательно полное покрытие, но должны быть показательные тесты для проверки бизнес-логики).

Кроме того, требуется написать (на английском) краткий обзор своего решения тестовой задачи, описав архитектуру, ее плюсы и минусы (что можно улучшить или поменять или еще какие-то соображения для использования решения в реальных целях).

Связь с базами данных не требуется, как не требуется и сохранение данных на диск. Приветствуется минимум зависимостей от разных библиотек, для простоты проверки тестовой работы.

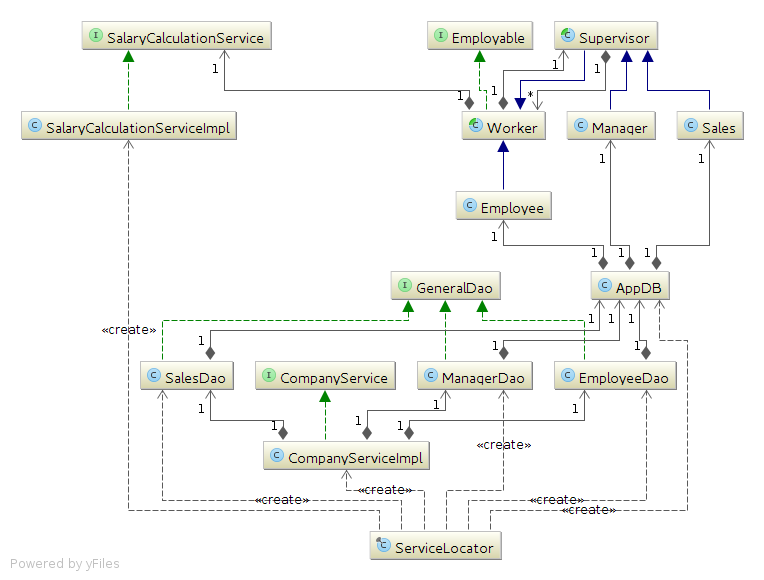
**1.** **Business Logic. Main Service**  
   
 Worker addWorker(Worker newWorker) throws NoSupportedSubTypeException;  
 Worker getById(int id) throws NoWorkerFoundException;  
 List<Worker> searchByName(String name);  
 List<Worker> getAllWorkers();  
 Supervisor addSubworker(int supervisorId, int subworkerId) throws NoWorkerFoundException;  
 String getWorkerType(int workerId) throws NoWorkerFoundException;  
 int calculateMonthSalary(int workerId) throws NoWorkerFoundException;  
 int calculateMonthSalaryAllWorkers();  
 String getTreeView(int workerId) throws NoWorkerFoundException;  
  
**2. How to run**  
  
 1. install maven  
 2. ***git clone https://github.com/presly808/AsposeTask.git***  
 3. **cd ${PROJECT\_DIR}**  
 4. build project ***mvn build*** 5. ***cd target***

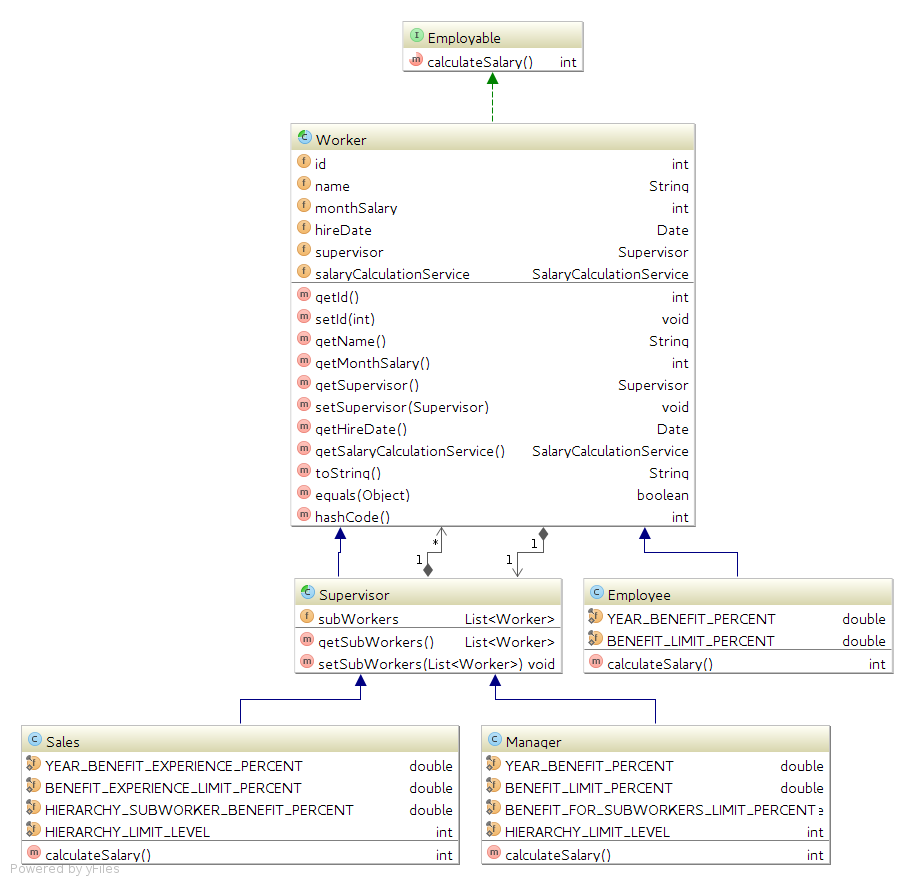
6. run console app ***java -jar CompanyAsposeTask-1.0-jar-with-dependencies.jar***   
  
 You can also run tests only **mvn test**  
  
**3. Project Architecture**  
 **3.1. General App ClassDiagram**



***3.2. ServiceLocator*** holds all singletons as dao, services and others. It takes all objects creation in one place

**3.3. Class Diagram**



**3.4. Class Diagrams of the model package  
**  
  
**3.5 Important class for calculations of year benefit and subworkers hierarchy**  
 *public interface SalaryCalculationService {  
  
 int calculateYearBenefit(int fixedSalary, double yearBenefitPercent,*

*int fullYears, double limitOfYearBenefitPercent);  
  
 /\*\*  
 \* calculate benefits with level limitaion  
 \*  
 \* @param level limit a calculation level, if -1 we consider all hierarchy  
 \* \*/  
 int calculateTeamBenefitByLevel(List<Worker> root, int level);  
 }*

**4. Pluses and minuses of the app**

\* Pluses  
 \* flexible arhcitecture  
 \* configuration via a file  
 \* separated little components  
 \* app is ready to be scaled  
 \* dependency injections using ServiceLocator  
 \* calculation of complex hierarchy is readable  
 \* tool for debugging calculations of tree   
  
\* Minuses  
 \* a lot of classes  
 \* console view  
 \* no persistence db]

**5. Future Issues**   
\* add posibility to calculate salary choosing endDate  
\* replace ServiceLocator by Spring  
\* improve ```com.aspose.company.utils.visualization.PrintUtils``` to get more detailed debug info  
  
**6. Recommendations**  
\* Use ```com.aspose.company.utils.visualization.PrintUtils``` to show hierarchy as a tree  
\* Change benefit percents via ```app.properties``` which is located in the resources  
\* Start point of the program is com.aspose.company.Run