***Company library specification***

**Task definition**

Let’s we have some company (corporation). Company can have some collaborators. Collaborator has the following properties: name, hiring date and a base rate (for simplicity by default this value is the same for all types of collaborators).

There are three types of collaborators: Employee, Manager, Sales. Each collaborator can have a chief. Each collaborator except Employee can have subordinates.

Salary of Employee is calculated as: <Base Rate> plus 3% for each working year in a company but not more than 30% of total increment.

Salary of Manager is calculated as: <Base Rate> plus 5% for each working year in a company (but not more than 30% of total increment) plus 0.5% of total salary of the 1-st level subordinates.

Salary of Sales is calculated as: <Base Rate> plus 1% for each working year in a company (but not more than 35% of total increment) plus 0.3% of total salary of all levels subordinates.

Each collaborator (except Employee) can have any count of subordinates of any type.

It is required to design library of classes for this Company’s model. Also it is required to implement salary calculation algorithm for each type of collaborator for defined moment of time (and also implement salary calculation algorithm for the whole company). Implementation should be verified by unit tests (full coverage is not necessary but significant tests for business logic are required).

In addition it is necessary briefly describe test task solution, architecture, advantages and disadvantages (what can be enhanced or modified, or any other considerations for using of solution for real purposes).

Only “boost” and “googletest” (framework for unit testing) 3-rd party libraries can be used. Databases are not required, data model is transient.

**Folders structure**

* company – source code of the “company” library project: C++ static library company.lib.
* company\_test – project with unit tests of the “company” library.
* doc – folder with documentation.
* output – folder with outputs of the company and company\_test projects.
* company\_lib.sln – Visual Studio 2013 solution with projects “company” and “company\_test”.

All projects were created in Visual Studio 2013.

Sources of the Google Test were downloaded from GitHub:

<https://github.com/google/googletest/tree/master/googletest>

**Company library description**

Interface of the library is located in the folder: “company/include”.

Only STL library and features of C++ 11 standard were used for implementation.

As a coding standard JSF-AV rules were used:

<http://www.stroustrup.com/JSF-AV-rules.pdf>

Company library contains the following classes: Collaborator, Employee, Manager, Sales and Company.

* Collaborator – abstract class of collaborator.

Collaborator has name, hiring date and base salary. Collaborator keeps pointer to chief and has collection of subordinates.

Collaborator has a pure virtual function of salary calculation which takes as a parameter date and time for which salary should be calculated.

* Employee – class derived from Collaborator. Employee can’t have subordinates. Salary of employee is calculated as: <Base Rate> plus 3% for each working year in a company but not more than 30% of total increment.
* Manager – class derived from Collaborator. Manager can have any subordinates. Salary of manager is calculated as: <Base Rate> plus 5% for each working year in a company (but not more than 30% of total increment) plus 0.5% of total salary of the 1-st level subordinates.
* Sales - class derived from Collaborator. Sales can have any subordinates. Salary of sales is calculated as: <Base Rate> plus 1% for each working year in a company (but not more than 35% of total increment) plus 0.3% of total salary of all levels subordinates.
* Company – class which keeps all collaborators and can calculate total salary. It has operations of hiring and firing collaborators.

Such architecture is well suited for the task conditions because it uses polymorphism for separating different algorithms of salary calculation for each type of collaborator. So we can keep collection of collaborators inside Company class as pointers to the abstract class Collaborator and calculate salary for each collaborator calling function with uniform prototype.

For using this architecture in real project we need to work with Database. It is necessary to add ID property to each collaborator. Currently I use STL vector container to store pointers to collaborators but probably it would be better to store collaborators to map with ID as key. It will allow to quickly querying collaborators by ID.

My current implementation doesn’t check the absence of loops in hierarchy of collaborators, this check should be added to the function which adds new subordinate.

Also it is possible to move logic of salary calculation into separate class. As a variant instead of creating a subclass for each type of collaborator we can use “Collaborator type” property. I also added this property to my implementation for easily distinguishing collaborators by their type.

Class Collaborator should be slightly refactored because it contains some static functions which are not related to it (functions of working with date and time). These functions should be added to some class Common\_utils.

**Unit tests for company library**

Project with unit tests of company library is called company\_test. It contains \*.cpp file for each class of “company” library (for example collaborator\_class\_test.cpp for class Collaborator). Tests don’t cover all possible combinations but they cover most parts of business logic. Also I implemented some additional functionality which wasn’t defined in task and this functionality wasn’t completely covered by unit tests.