# Staff Management Service

by Vladimir Tyagunov

10.03.2019

## Description

The attached archive contains a Staff Management service solution.

This service need for Calculate a Staff salary. 'Aspose.StaffManagement' project contains interface and implementation for Calculate Staff salary.

## Usage instruction

1. Rebuild a solution

2. Include compiled dll to your project (put dll into your 'Libraries' folder and set reference to this dll in the project references)

3. Use next namespace for access to:

    3.1 Model: 'Aspose.StaffManagement.Model'

    3.2 Calculation Service interface: 'Aspose.StaffManagement.Interfaces'

    3.3 Calculation Service implementation: 'Aspose.StaffManagement.Services'

4. Calculation Service has 2 public methods for use: 'CalculateAllSalaries' and 'CalculateSalary'

## Architecture

Project implements with use the SOLID principles like:

1. Single Responsibility Principle - Calculation service contains methods only for calculate staff salary data, and entire methods contains only calculation functionality without other responsibilities

2. Open/Closed Principle - Model has base staff data class, and for extends this data classes used an Employee, Manager and Sales classes. For calculation service created an interface which contains two methods, for implementation this interface was added a calculation service class, if we need to extends this functionality, we can create a new implementation for this interface or to extends current

3. Liskov Substitution Principle - case with interface and implementations of the calculation service functionality, next we can use calculation service interface, and if we will want to change an algorithm of the calculation, we can add a new calculation service implementation based on this interface, and other part of program will be not changed

4. Dependency Inversion Principle - case with interface and implementation of the calculation service functionality, which was wrote upper

## Advantages and drawbacks

**Advantages:**

1. Solution implements with compliance a SOLID principles

2. It is a flexible code design, which contains unit tests for check main functionality in a specific case

**Drawbacks:**

1. There are no logs. Need to add a logger for log all actions in the methods

2. All functionality only in a one project (Task requirements). Better way to divide an interfaces from an implementations into different projects, and exclude a model in the own project, all layers should be placed in own projects (model, business logic...)

3. For use a service implementation I didn't use a factory, in the real project I'd like to create a factory functionality and use all services from a factory

4. I like to use a #region directives, and if this service could be more in a volume, I'd like to use #region for divide fields, properties, constructors and methods in the different regions

5. For calculate a salary is need to create a rounding service, in real situation it would be more right than use a very long digits after point

6. Values for percentages in the resources should be with '.' only, not ',', because for get a resource value I used a CultureInfo.InvariantCulture, for publish the project better will be found a more good solution