Workshop 12

19 февраля 2019 г.

Consider the tasks from Workshop 8. Remaster the solution using the OOP approach.

1. Define a class named `Employee` that describes an employee. The fields are:
   a. Name
   b. Date of Birth (in dd.mm.yyyy format)
   c. Array of wages
   d. An auxiliary variable for storing cache of the calculated total wages. Use -1, if no total wage is calculated.
   e. (Additional fields if needed).

   All fields are private. Create all necessary methods to provide a public interface to these fields. Respect constancy.

2. Create methods to calculate a total and an average salary with caching the calculated sum.
3. Overload the `operator<<` for inputting data of an individual employee.
4. Overload the `operator>>` for outputting data of an individual employee. Output wages as a vertical table as in the following example:

1	12.50
2	13.10
3	1.15

5. Create a method for inputting data of several employees using the new `operator<<` approach.
6. Create a method for outputting data of several employees using the new `operator>>` approach.
7. Create a method which sorts an array of employees by name, then by age, then by *average* salary. Use `std::sort method()`. Create a predicate for comparing two employee objects if needed.
8. Create a `main()` method which brings the things together:
   a. inputs a collection of employees;
   b. outputs the initial collection;
   c. creates a copy of the collection of employees, sorts it and outputs the sorted collection.

   Draw a UML Sequence Diagram depicting the call flow of the developed methods and free functions.
9. Create a *static factory method* obtaining the name, the day of birth of an employee and the number of wages (*n*), that creates an employee object with *n*-random wages and returns it.
10. Create a few employees using the method above. Output their details to the standard out.

New for Workshop 12

11. Respect constancy.
12. Create a class called `EmployeeRegister`. The class stores a collection of `Employee`s using `std::map<string, Employee>`. The map is used for fast access to an employee by its name (name duplicates are not allowed).
13. Create a factory method called `EmployeeRegister::createNewEmployee()` which creates and adds a new `Employee` object and initializes it with given parameters: name, age, list of wages (given as a vector of doubles).
14. Create proxy methods for getting information about a collection of employees; e.g., `getEmplNum()` returns a number of employees.
15. Create a method called `findEmployee()` that obtains a name as a parameter and looks up for an employee with the same name. If an employee is found it returns `true` by value and a reference to the employee object (mind the problem of references to temporary objects), otherwise `false`.
    a. Reconsider the method's implementation with a `std::vector<Emloyee>`-based collection. What would be changed in the interface and in the implementation?

–