Object Oriented Programming

Summer Semester 2020

Homework Assignment 4.

July 3, 2020

Contents

Chapter 1

Introduction

1. Please pay attention to the due date. I urge you to start working on the homework right away and not wait until the last minute.
2. You should provide comments for your code so it will be completely clear what you are trying to achieve. WARNING! Lack of comments might lead to points reduction.
3. You cannot use multiple inheritance in this assignment.
4. Please note the following few points which may lead to points reduction during the submission check:
   (a) Avoid using *magic numbers*. For example: "if (i>17)", 17 is a magic number. If 17 is representing, for example, number of shoes, then instead you should write: "if (i>shoesNumber)".
   (b) Try to avoid code duplication as much as possible.
   (c) You should not globally enable *std* namespace usage or any other namespaces, e.g. *using namespace std;*.

1.1 Overview

The purpose of this assignment is to learn you the basic concepts of inheritance in C++, make you work with polymorphic dependencies between classes. Given a description of the system which you need to develop, your goal is to determine dependencies between different classes derive hierarchy of similar classes.

1.2 Details

1.2.1 Employee

This is an abstract class which encapsulate general company worker entity, for each employee in the company there are exist next properties:

1. **description** - description string with employee job title.
2. **id** - number identifying employee ID, id numbers should be in the increasing order (1, 2, 3, ...) and you cannot reuse ID of employee which has been removed.

In addition to properties here basic methods which should be provided with the **Employee** class:

1. **Constructor** - constructor with following parameters:
    a. **Description** - employee job title.

   Employee ID should be automatically generated.
2. **clone** - method which capable to perform deep copying of object instance according to his run-time type (dynamic binding).
3. **getID** - returns employee ID number.
4. **getDescription** - returns employee job description.
5. **setDescription** - allows to change employee job description:
    a. **description** - job title.

The last thing, you should take care to implement **operator<<** to be able to print employee according to his run-time type (dynamic binding).

NOTE: No need to re-implement **operator<<** for derived classes.

1.2.2 DeveloperEmployee

Class which represents worker from company R&D department and encapsulates all his capabilities and properties. Properties:

1. **project** - string which represents the employee project name.

Methods:

1. **Constructor** - constructor with following parameters:
    a. **Description** - string with employee job title.
    b. **Project** - string which represents the employee project name.
2. **setProject** - allows to change project name for given developer employee.

    While printing developer employee on the screen you should preserve the following string format:

    "Developer ID = <id>, project = <project_name>"

    in case no project has been defined you should output:

    "Developer ID = <id>, project = ???"

1.2.3 TesterEmployee

Class which represents worker from company Q&A department and encapsulates all his capabilities and properties. Properties:

1. **level** - string which represents employee skills level.

Methods:

1. **Constructor** - constructor with following parameters:
    a. **Description** - string with employee job title.
    b. **level** - string includes information about experience level of QA employee
2. **setLevel** - allows to change employee QA level.

    While printing tester employee on the screen you should preserve the following string format:

    "Tester ID = <id>, level = <project_name>"

    in case no level has been defined you should output:

    " Tester ID = <id>, level = ???"

1.2.4 IllegalArguments

Class which will be thrown as an exception in case one of the methods will receive illegal argument.

1.2.5 Company

Class which will keep data structure to persist objects of type **Employee**. You are required to extend functionality of Company class to make test programs attached to compile without warnings or errors.

You should provide next capabilities/methods to the class:

1. **Inserting new employee** - insert deep copy of an employee given as a parameter. In case employee with given ID already exists, function do nothing. In case of illegal parameter function will throw an IllegalArgument exception.
2. **Delete an employee** - remove employee with given ID number, if employee with given ID doesn't exist in the company, function will do nothing.
3. **Finding employee** - will look up for an employee in the company having his ID number, creates a deep copy and returns pointer to it.
4. **Check employee existence** - will return true in case employee with given ID exists in the company and false otherwise.

The last thing, you should take care to implement **operator<<** to be able to print information about company and list of its employees.

While printing company information on the screen you should preserve following string format:

"Company employees:"

" <employee_1_details>"

" <employee_2_details>"

....

" <employee_N_details>"

while *<employee_i_details>* is the output of the i'th employee in the company list.

in case there is no employees in the company, the output should be:

"Company is empty"

1.2.6 Extending System Capabilities

Printable

Class which represents entity responsible to print object description. Methods:

- **printDescription** - prints class instance description, if not description defined print "???"

You can assume, that no instance of **Printable** could be created.

PrintableDeveloper

Has all same functionality as **DeveloperEmployee** class, and in addition an ability to print its description. You should pay attention, that instance of this class could be added to company as active employee.

PrintableTester

Has all same functionality as **TesterEmployee** class, and in addition an ability to print it's description. You should pay attention, that instance of this class could be added to company as active employee.

1.3 Evaluation

Homework exercise provided with the following example program file and corresponding output:

1. main1.cpp
2. main2.cpp
3. output1.txt
4. output2.txt

For every $1 \leq x \leq 2$, you should be able to compile your code with "main***x***.cpp" file and receive correct output, which placed in "output***x***.txt" file.

Make sure you keep the C++ syntax convention and submit the files exactly as described in "Oop – Cpp – Conventions and Requirements" file in Moodle.

GOOD LUCK! ☺