Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

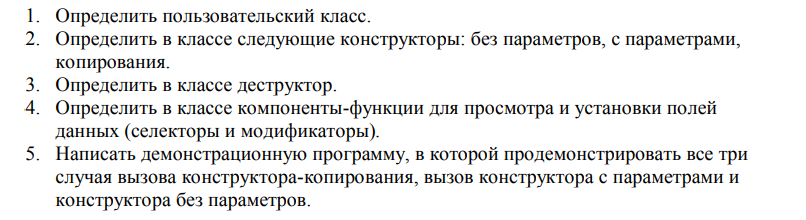
**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: Лабораторная работа по классам №2

Выполнил работу  
Студент группы РИС-22-1Б  
Ишемцева Мария Андреевна  
Проверил  
Доцент кафедры ИТАС  
Полякова Ольга Андреевна

Постановка задачи:



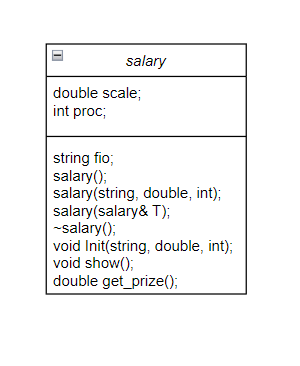


Анализ задачи:

1. Пользовательский класс exam содержит все поля и методы в области видимости public. Поля – double scale(зарплата), int proc(процент премии), string fio;(ФИО человека). Описаны 3 конструктора – без параметров, с параметрами, конструктор копирования. Также описан деструктор.
2. Для каждого поля есть геттер GetNONE(), выводящий поле структуры на экран и возвращающий значение этого поля. Аналогично есть сеттеры SetNONE(PARAMETR), принимающий в себя значение для поля, внутри сеттера параметр присваивается полю. Аналогичную операцию проводят конструкторы. Без параметров – заполняет заранее написанными данными, с параметрами – заполняет переданными данными(аналогично сеттерам), копирования(копирует все поля для своего обьекта у конструктора без параметров(в моем случае). Также реализован метод show(), вызывающийся для каждого обьекта после конструкторов.
3. Ответы на вопросы:

* Конструктор необходим для инициализации полей класса.
* 3. Без параметров, с параметрами, конструктор копирования.
* Для завершения работы с конструктором класса после выхода из области видимости ф-ии. Если конструктор описан явно, то необходимо и деструктор описать явно.
* Без параметров – для инициализации полей по умолчанию(заранее заданными значениями). С параметрами – для инициализации полей переданными в конструктор значениями. Копирования – для инициализации полей с помощью копирования их из другого обьекта.
* Конструктор копирования вызывается в случае, когда нужно заполнить n обьектов одинаковыми значениями полей.
* Инициализация полей переданными параметрами, инициализация полей по умолчанию, инициализация полей с помощью копирования, перегрузка конструкторов, конструктор списка по умолчанию, конструктор списка инициализаторов.
* Не могут иметь аргументов; Не возвращают значение (или void ); Не может быть объявлен как const , volatile или static . Однако они могут вызываться для уничтожения объектов, объявленных как const , volatile или static ; Может быть объявлен как virtual . Используя виртуальные деструкторы, можно уничтожать объекты, не зная их тип — правильный деструктор для объекта вызывается с помощью механизма виртуальных функций. Обратите внимание, что для абстрактных классов деструкторы также могут объявляться как чисто виртуальные функции; не описаны явно – автоматически вызовутся.
* Private, Public, Protected;
* Адрес обьекта класса;
* Внутри класса можно не передать параметры, вне – обязательно. Использование шаблонов ф-ий порой обязательно.
* Конструкторы не возвращают значений.
* Конструктор и деструктор.
* Аналогично, деструктор не возвращает значений.
* Сеттер.
* Никакой.
* Конструктор с параметрами.
* Конструктор с параметрами для первого обьекта и конструктор копирования для второго.
* Аналогично предыдущему, только сначала второй обьект вызовет конструктор без параметров.
* С параметрами.
* p.set\_name(“name”);

UML диаграмма:



Код:

Заголовочный файл

#pragma once

#include <iostream>

using namespace std;

class salary {

double scale;

int proc;

public:

string fio;

salary();

salary(string, double, int);

salary(salary& T);

~salary();

void Init(string, double, int);

void show();

double get\_prize();

};

Файл с описанием методов

#include <iostream>

#include "salary.h"

using namespace std;

salary::salary() {

fio = "";

scale = 0;

proc = 0;

cout << "Конструктор без параметров для " << this << endl;

}

salary::salary(string f, double s, int p) {

fio = f;

scale = s;

proc = p;

cout << "Коснтруктор с параметрами для - " << this << endl;

}

salary::salary(salary& T) {

fio = T.fio;

scale = T.scale;

proc = T.proc;

cout << "Конструктор копирования для - " << this << endl;

}

salary::~salary() {

cout << "Дуструктор для - " << this << endl;

}

void salary::Init(string foi, double sale, int poc) {

fio = foi;

scale = sale;

proc = poc;

}

void salary::show() {

cout << "ФИО - " << fio;

cout << "\nОклад - " << scale;

cout << "\nПроцент - " << proc;

cout << endl;

}

double salary::get\_prize() {

return proc \* 0.01 \* scale;

}

Основной файл

#include <iostream>

#include <string>

#include <windows.h>

#include "salary.h"

using namespace std;

void sal(salary S) {

S.show();

}

salary make\_prize() {

string s;

double sca;

int pro;

cout << "Введите ФИО: ";

getline(cin, s);

cout << "Введите оклад: ";

cin >> sca;

cout << "Введите, какой процент от оклада - премия: ";

cin >> pro;

salary Sa(s, sca, pro);

return Sa;

}

int maasasin() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

salary s1;

s1.show();

salary s2("Ivanov Ivan Ivanovich", 10000, 50);

s2.show();

cout << "Премия = " << s2.get\_prize() << endl;

salary s3 = s2;

s3.Init("Petrov Petr Petrovich", 20000, 20);

sal(s3);

cout << "Премия = " << s3.get\_prize() << endl;

s1 = make\_prize();

s1.show();

cout << "Премия = " << s1.get\_prize() << endl;

return 0;

}

Вывод:

