**Министерство образования и науки Российской Федерации**

**Пермское федеральное государственное бюджетное образовательное учреждение высшего образования «Пермский Национальный Исследовательский Политехнический Университет»**

**Электротехнический факультет**

**Кафедра «Информационные технологии и автоматизированные системы»**

**ОТЧЁТ**

По лабораторной работе №18.11

Вариант №11

Выполнил студент группы РИС-20-2б

Савельев Виталий Антонович

Проверил доцент кафедры ИТАС

Полякова О.А.

Пермь 2021

**Цель работы** –научиться работать с основными парадигмами объектно-ориентированного программирование, в частности, научиться применять инкапсуляцию при разработке объектов в программе.

**Постановка задачи**

Задача – реализовать на языке программирования C++ программу, которая демонстрирует инкапсуляцию в действии. Для этого необходимо реализовать:

1. Определение нового класса в соответствии с вариантом;
2. Главную функцию для демонстрации;
3. Добавить в класс функции ввода данных;

**Исходные данные для варианта №11:**

Класс «Человек»:

* ФИО (Полное имя);
* Заработная плата (доллары в месяц);
* Количество дней в месяце;
* Количество отработанных дней;
* Функция, возвращающая полученную зарплату;

**Анализ задачи**

Для решения задачи были использованы следующие средства:

1. Язык программирования C++ (Microsoft Visual C++)
2. Текстовый редактор Microsoft Visual Studio Code
3. Система контроля версий Git

**Инкапсуляция:**

Инкапсуляция – это механизм системы, который позволяет объединять данные и методы в одном объекте, классе, которые необходимы для работы с ним, и скрыть детали их реализации от внешнего пользователя.

Таким образом, можно точно настроить уровень доступа к компонентам объекта, разделив любой объект на две составляющие:

1. Интерфейс – часть класса, который доступен внешнему пользователю для непосредственного использования класса (автомобиль: интерфейс в нём представлен в виде руля, педалей и кнопок.);
2. Внутренняя реализация – часть класса, который не доступен пользователю. Предполагается, что пользователю совсем не обязательно знать, как работает реализация класса, если она работает безотказно (пользуясь автомобилем далеко не каждый водитель сможет объяснить, как работает двигатель или коробка передач).

**Реализация класса «Человек»:**

Данный класс содержит следующие поля:

1. ФИО (Полное имя);
2. Заработная плата (доллары в месяц);
3. Количество дней в месяце;
4. Количество отработанных дней;

Именно эти поля в целях безопасности будут скрыты для пользователя от прямого доступа к ним. Данный шаг был сделан для того, чтобы пользователь не мог случайно или по злому умыслу во время использования класса присваивать полям некорректные значение такие как:

* Отрицательная зарплата;
* Слишком завышенное или заниженное количество дней месяца;
* Отрицательной количество отработанных дней;

Для того, чтобы пользователь мог получать значения полей и задавать им значения с учётом проверок на корректность введённых данных, необходимо реализовать методы, получающие и задающие значение сокрытых полей.

Скрытые поля имеют модификатор доступа private. В C++ классы и приватные поля объявляются следующим образом:

class Person

{

private:

string \_name = "Janina Kowalski";

float \_salary = 0;

int \_daysWorked = 0;

float \_totalSalary = 0;

int \_daysInMonth = 31;

}

Методы, которые организуют управляемый доступ к полям имеют модификатор доступа public, который объявляется под полями и методами, которые находятся в поле private.

public:

float TotalSalary()

{

return \_totalSalary;

}

float Salary() { return \_salary; }

bool SetSalary(float salary)

{

if (salary > 0)

{

\_salary = salary;

ComputeSalary();

return true;

}

else

{

cout << "Error: invalid salary!!! less than 0." << endl;

return false;

}

}

int DaysWorked() { return \_daysWorked; }

bool SetDaysWorked(int days)

{

if ((days >= 0) && (days <= \_daysInMonth))

{

\_daysWorked = days;

ComputeSalary();

return true;

}

else

{

cout << "Error: invalid days!!! less than 0 or more than days in month." << endl;

return false;

}

}

int DaysInMonth() { return \_daysInMonth; }

bool SetDaysInMonth(int days)

{

if ((days >= 28) && (days <= 31))

{

\_daysInMonth = days;

ComputeSalary();

return true;

}

else

{

cout << "Error: invalid days, out of [28 ; 31]!!!" << endl;

return false;

}

}

string GetName() { return \_name; }

void SetName(string name) { \_name = name; }

Как описано выше, в методах, которые отвечают за присвоение значений полям, производится проверка на корректность значения: в зависимости от статуса возвращается соответствующий флажок типа bool: true – успешный ввод, false – ошибка, введено некорректное значение.

Флажок возвращается не просто так, он будет необходим для того, чтобы осуществлять ввод с клавиатуры: в случае неудачного ввода данных, система попросит повторить ввод.

void ReadPersonData()

{

cout << "Input the person's salary:" << endl;

int value;

bool isInputSuccessful;

do

{

cout << "Input the person's salary: ";

cin >> value;

isInputSuccessful = SetSalary(value);

}

while (!isInputSuccessful);

do

{

cout << "Input days in the month: " << endl;

cin >> value;

isInputSuccessful = SetDaysInMonth(value);

}

while (!isInputSuccessful);

do

{

cout << "Input worked days: " << endl;

cin >> value;

isInputSuccessful = SetDaysWorked(value);

}

while (!isInputSuccessful);

}

Также, управляемый ввод данных через специальные методы необходим для использования в конструкторе класса с параметрами, так как и в конструкторе пользователь может случайно ввести некорректные данные.

Person(string name, float salary, int daysInMonth, int daysWorked)

{

SetName(name);

SetSalary(salary);

SetDaysInMonth(daysInMonth);

SetDaysWorked(daysWorked);

}

Так как конструктор без параметров не был указан явно, он будет создан компилятором автоматически, при этом приватные поля будут инициализированы (см. код выше).

Наконец, для вывода данных о человеке на консоль, был реализован метод PrintPersonData, который структурированно выводит все данные о человеке.

void PrintPersonData()

{

cout << "Person's data:" << endl;

cout << " - Name: " << \_name << ";\n";

cout << " - Salary: " << \_salary << ";\n";

cout << " - Days in the month: " << \_daysInMonth << ";\n";

cout << " - Worked days: " << \_daysWorked << ";\n";

cout << " >> Total salary: " << \_totalSalary << ";\n\n";

}

**Реализация главной функции main:**

Задача главной функции – это демонстрация работы инкапсуляции в программе.

int main()

{

**//конструктор с параметрами**

Person firstPerson("Amelie", 25000.17, 30, 21);

**//вывод данных на экран**

firstPerson.PrintPersonData();

**//ввод данных с клавиатуры**

firstPerson.ReadPersonData();

**//вывод данных на экран**

firstPerson.PrintPersonData();

**//попытка ввести некорректные данные: вывод сообщений об ошибке**

firstPerson.SetSalary(-1);

firstPerson.SetDaysWorked(100);

firstPerson.SetDaysInMonth(13);

**//вывод данных на экран**

firstPerson.PrintPersonData();

**//Функция возвращает 0 – успешное выполнение программы**

return 0;

}

Ответы на вопросы:

1. **Что такое класс?**

Класс – это абстрактный тип данных, сущность, которую задаёт программист. Каждый класс представляет собой модель некоторого реального объекта в виде набора данных и функций для работы с ним.

1. **Что такое объект (экземпляр) класса?**

Экземпляр класса, это сама переменная, имеющая тип данных некоторого объекта. Работа с классами происходит именно через экземпляры класса, количество которых может быть любым.

1. **Как называются поля класса?**

Поля класса называют через два варианта: поле или атрибут. Данные определения являются равнозначными.

1. **Как называются функции класса?**

Функции класса называют методами. По смыслу, функция и метод – это одно и тоже.

1. **Для чего используются спецификаторы доступа?**

Разные спецификаторы доступа используются для настройки уровня доступа элементов класса для пользователей и наследников.

1. **Для чего используется спецификатор public?**

Модификатор доступа public необходим для описания общего доступа элементов класса для пользователя и наследника, предоставив полный доступ и к ним. По смыслу, модификатор public позволяет сформировать интерфейс класса, рычаги управления классом.

1. **Для чего используется спецификатор private?**

Модификатор доступа private необходим для сокрытия деталей реализации класса, знание которого не требуется от пользователя. Также необходимо скрывать те поля, значения которых могут быть введены некорректно: например, отрицательный возраст человека. Модификатор private позволяет сберечь поле от прямого доступа незадачливого пользователя.

1. **Если описание класса начинается со спецификатора class, то какой спецификатор доступа будет использоваться по умолчанию?**

По умолчанию в классах, которые начинаются со спецификатора class, для всех элементов будет задан модификатор доступа **private**.

1. **Если описание класса начинается со спецификатора struct, то какой спецификатор доступа будет использоваться по умолчанию?**

По умолчанию в классах, которые начинаются со спецификатора class, для всех элементов будет задан модификатор доступа **public**.

1. **Какой спецификатор доступа должен использоваться при описании интерфейса класса? Почему?**

Как было описано выше, интерфейс является публичным/доступным для пользователя, так как он представляет собой рычаги управлением класса: руль, педали, кнопки, тачпад, мышь, клавиатура, экран. Поэтому при описании интерфейса необходимо использовать спецификатор доступа **public**.

1. **Каким образом можно изменить значения атрибутов экземпляра класса?**

Поскольку атрибуты/поля принято объявлять с модификатором доступа private. Соответственно, необходимо реализовать методы, которые могут присвоить значение полю, пройдя через необходимые проверки.

1. **Каким образом можно получить значения атрибутов экземпляра класса?**

Как было описано выше, при помощи специальных методов, которые имеют доступ к полю и извлекают из него данные.

1. **Класс описан следующим образом:**

struct Student

{

string name;

int group;

.........

}

**Объект класса определён следующим образом:**

Student\* s = new Student;

**Как можно обратиться к полю name объекта s?**

Никаких проблем при непосредственном обращении к полю name не произойдёт, так как по умолчанию в структурах все элементы имеют модификатор доступа public. Соответственно, можно обратиться напрямую:

cout << s->name;

1. Вопрос повторяется
2. **Класс описан следующим образом:**

class Student

{

string name;

int group;

.........

}

**Объект класса определён следующим образом:**

Student\* s = new Student;

**Как можно обратиться к полю name объекта s?**

Ответ: никак. По умолчанию у всех элементов класса стоит модификатор доступа private. Соответственно, пользователь не имеет доступа к полям класса.

1. **Класс описан следующим образом:**

class Student

{

string name;

int group;

.........

public:

}

**Объект класса определён следующим образом:**

Student\* s = new Student;

**Как можно обратиться к полю name объекта s?**

Ответ: никак. По умолчанию у всех элементов класса стоит модификатор доступа private. Соответственно, пользователь не имеет доступа к полям класса.

1. **Класс описан следующим образом:**

class Student

{

public:

char\* name;

int group;

.........

}

**Объект класса определён следующим образом:**

Student\* s = new Student;

**Как можно обратиться к полю name объекта s?**

В данном случае установлен модификатор доступа public. Соответственно, никаких проблем в доступе нет и можно без проблем обратиться к полю напрямую.

cout << s->name;

аоеи

**Полный исходный код программы на языке программирования C++:**

#include <iostream>

using namespace std;

class Person

{

private:

string \_name = "Janina Kowalski";

float \_salary = 0;

int \_daysWorked = 0;

float \_totalSalary = 0;

int \_daysInMonth = 31;

void ComputeSalary()

{

\_totalSalary = \_salary \* \_daysWorked / \_daysInMonth;

}

public:

float TotalSalary()

{

return \_totalSalary;

}

float Salary() { return \_salary; }

bool SetSalary(float salary)

{

if (salary > 0)

{

\_salary = salary;

ComputeSalary();

return true;

}

else

{

cout << "Error: invalid salary!!! less than 0." << endl;

return false;

}

}

int DaysWorked() { return \_daysWorked; }

bool SetDaysWorked(int days)

{

if ((days >= 0) && (days <= \_daysInMonth))

{

\_daysWorked = days;

ComputeSalary();

return true;

}

else

{

cout << "Error: invalid days!!! less than 0 or more than days in month." << endl;

return false;

}

}

int DaysInMonth() { return \_daysInMonth; }

bool SetDaysInMonth(int days)

{

if ((days >= 28) && (days <= 31))

{

\_daysInMonth = days;

ComputeSalary();

return true;

}

else

{

cout << "Error: invalid days, out of [28 ; 31]!!!" << endl;

return false;

}

}

string GetName() { return \_name; }

void SetName(string name) { \_name = name; }

void ReadPersonData()

{

cout << "Input the person's salary:" << endl;

int value;

bool isInputSuccessful;

do

{

cout << "Input the person's salary: ";

cin >> value;

isInputSuccessful = SetSalary(value);

}

while (!isInputSuccessful);

do

{

cout << "Input days in the month: " << endl;

cin >> value;

isInputSuccessful = SetDaysInMonth(value);

}

while (!isInputSuccessful);

do

{

cout << "Input worked days: " << endl;

cin >> value;

isInputSuccessful = SetDaysWorked(value);

}

while (!isInputSuccessful);

}

void PrintPersonData()

{

cout << "Person's data:" << endl;

cout << " - Name: " << \_name << ";\n";

cout << " - Salary: " << \_salary << ";\n";

cout << " - Days in the month: " << \_daysInMonth << ";\n";

cout << " - Worked days: " << \_daysWorked << ";\n";

cout << " >> Total salary: " << \_totalSalary << ";\n\n";

}

Person(string name, float salary, int daysInMonth, int daysWorked)

{

SetName(name);

SetSalary(salary);

SetDaysInMonth(daysInMonth);

SetDaysWorked(daysWorked);

}

};

int main()

{

Person firstPerson("Amelie", 25000.17, 30, 21);

firstPerson.PrintPersonData();

firstPerson.ReadPersonData();

firstPerson.PrintPersonData();

firstPerson.SetSalary(-1);

firstPerson.SetDaysWorked(100);

firstPerson.SetDaysInMonth(13);

firstPerson.PrintPersonData();

return 0;

}

**Скриншот с успешным выполнением программы:**

