Министерство образования и науки Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Алтайский государственный технический университет им. И.И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчёт защищён с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель Троицкий В.С.

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г.

Отчёт

по лабораторной работе № 8

«Статические поля и методы»

по дисциплине «Программирование - 3»

Студент группы ПИ 92 В.М. Шульпов

Преподаватель доцент, к.т. н. Троицкий В.С.

Барнаул 2020

ЗАДАЧА

Модифицировать ваши проекты на С++, C# и Java путем добавления в один из классов как минимум одного статического поля и одного статического метода.

1) Придумать для чего в вашем классе может быть разумным использование статического поля и статического метода;

2) Создать эти статические поля и методы;

3) Продемонстрировать их использование;

!) Git обязателен;

!) В отчете по ЛР должна быть ссылка на Githab.

<https://github.com/ShulV/CSHARP_labs>

<https://github.com/ShulV/CPP_labs>

https://github.com/ShulV/JAVA\_labs

КОД ПРОГРАММЫ

// lab work №7,8 Shulpov Victor PI-92

/\*

Реализовать работу автомобиля на примере динамической структуры.

Поля структуры:

• Название (строка)

• Цена (целое)

• Цвет (строка)

• Количество оборотов двигателя в минуту (целое)

• Скорость (целое)

• Количество бензина (целое)

Функции:

• Инициализация

• Установка параметров автомобиля

• Вывод данных машины

• Запуск двигателя

• Остановка двигателя

• Добавление скорости

• Уменьшить скорость

\*/

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <malloc.h>

#include <conio.h>

#include <string.h>

#define clear(stream) rewind((stream)) //очистка потока

const int CAR\_NUMBERS = 10;

class Car;

int\* allocateArray(int size);

int& getCallNumber();

void setStartPosition(Car\* car);

class Engine

{

public:

void init(int engineRPM, int capacity, int enginePower, int quantityOfCylinders);

void setEngineRPM(int engineRPM);

void setCapacity(int capacity);

void setEnginePower(int enginePower);

void setQuantityOfCylinders(int quantityOfCylinders);

int getEngineRPM();

int getCapacity();

int getEnginePower();

int getQuantityOfCylinders();

private:

int engineRPM; //количество оборотов в минуту

int capacity; //объем см куб

int enginePower; //мощность Л.С.

int quantityOfCylinders; //количество цилиндров

};

class Car

{

private:

int x = 0;

int y = 0;

std::string name;

int price;

std::string color;

int speed;

int benzine;

Engine\* engine;

static int count;

public:

Car();

~Car();

void init(std::string name, int price, std::string color, int speed, int benzine, Engine\* engine);

void readCarData();

void displayDataCar();

void addBenzine(int liters);

void startEngine();

void stopEngine();

void addSpeed(int speed);

void reduceSpeed(int speed);

friend void setStartPosition(Car\* car);

Car operator+(int benzine);//перегрузка оператора +

Car& operator++ ();//префиксная

Car operator++ (int unused);

static int getCount();

};

Car::Car()

{

this->x = 0;

this->y = 0;

this->name = "";

this->price = 0;

this->color = "";

this->speed = 0;

this->benzine = 0;

this->engine = NULL;

if (count != NULL) count++;

else count = 1;

}

Car::~Car()

{

count--;

}

void Car::init(std::string name, int price, std::string color, int speed, int benzine, Engine\* engine)

{

this->name = name;

this->price = price;

this->color = color;

this->benzine = benzine;

this->speed = speed;

this->engine = engine;

printf("Car initialized!\n");

}

void Car::readCarData() {

int number;

std::cout << "ENTER CAR DATA:" << std::endl;

std::cout << "\tname:\t";

std::cin >> this->name;

clear(stdin);

std::cout << "\tprice:\t";

std::cin >> this->price;

clear(stdin);

std::cout << "\tcolor:\t";

std::cin >> this->color;

clear(stdin);

std::cout << "\tengineRPM:\t";

std::cin >> number;

this->engine->setEngineRPM(number);

clear(stdin);

std::cout << "\tcapacity:\t";

std::cin >> number;

this->engine->setCapacity(number);

clear(stdin);

std::cout << "\tengine power:\t";

std::cin >> number;

this->engine->setEnginePower(number);

clear(stdin);

std::cout << "\tquantity of cylinders:\t";

std::cin >> number;

this->engine->setQuantityOfCylinders(number);

clear(stdin);

std::cout << "\tspeed:\t";

std::cin >> this->speed;

clear(stdin);

std::cout << "\tbenzine:\t";

std::cin >> this->benzine;

clear(stdin);

}

void Car::displayDataCar()

{

std::cout << "\t" << "Car data" << std::endl;

std::cout << "\t\tName:\t" << this->name << std::endl;

std::cout << "\t\tPrice:\t" << this->price << std::endl;

std::cout << "\t\tColor:\t" << this->color << std::endl;

std::cout << "\t\tEngineRPM:\t" << this->engine->getEngineRPM() << std::endl;

std::cout << "\t\tCapacity:\t" << this->engine->getCapacity() << std::endl;

std::cout << "\t\tEngine Power:\t" << this->engine->getEnginePower() << std::endl;

std::cout << "\t\tQuanity of cylinders:\t" << this->engine->getQuantityOfCylinders() << std::endl;

std::cout << "\t\tBenzine:\t" << this->benzine << std::endl;

std::cout << "\t\tSpeed:\t" << this->speed << std::endl;

}

void Car::addBenzine(int liters)

{

std::cout << liters << "lit. benzine added!" << std::endl;

this->benzine += liters;

}

void Car::startEngine()

{

if (this->benzine > 0) {

this->engine->setEngineRPM(800);

std::cout << "Engine started!" << std::endl;

}

else {

std::cout << "No benzine. Engine didn't start!" << std::endl;

}

}

void Car::stopEngine()

{

if (this->engine->getEngineRPM() > 0) {

this->engine->setEngineRPM(0);

std::cout << "Engine stopped!" << std::endl;

}

else {

std::cout << "Engine stopped already!" << std::endl;

}

}

void Car::addSpeed(int speed)

{

if (this->engine->getEngineRPM() > 0) {

this->speed += speed;

std::cout << "Car speeded up!" << std::endl;

}

else {

std::cout << "Engine isn't starting. Car didn't speed up!" << std::endl;

}

}

void Car::reduceSpeed(int speed)

{

if (this->speed > 0) {

this->speed -= speed;

std::cout << "Car speeded down!" << std::endl;

}

else {

std::cout << "Car is parking. Car didn't speed down!" << std::endl;

}

}

Car Car::operator+(int benzine)

{

Car new\_car = \*this;

new\_car.benzine = this->benzine + benzine;

return new\_car;

}

Car& Car::operator++()

{

this->benzine++;

return \*this;

}

Car Car::operator++(int unused)

{

Car car = \*this;

++this->benzine;

return car;

}

int Car::getCount()

{

return count;

}

int Car::count = 0; // определение статической переменной-члена класса

void Engine::init(int engineRPM, int capacity, int enginePower, int quantityOfCylinders)

{

this->engineRPM = engineRPM;

this->capacity = capacity;

this->enginePower = enginePower;

this->quantityOfCylinders = quantityOfCylinders;

}

void Engine::setEngineRPM(int engineRPM)

{

this->engineRPM = engineRPM;

}

void Engine::setCapacity(int capacity)

{

this->capacity = capacity;

}

void Engine::setEnginePower(int enginePower)

{

this->enginePower = enginePower;

}

void Engine::setQuantityOfCylinders(int quantityOfCylinders)

{

this->quantityOfCylinders = quantityOfCylinders;

}

int Engine::getEngineRPM()

{

return this->engineRPM;

}

int Engine::getCapacity()

{

return this->capacity;

}

int Engine::getEnginePower()

{

return this->enginePower;

}

int Engine::getQuantityOfCylinders()

{

return this->quantityOfCylinders;

}

int main()

{

setlocale(LC\_ALL, "Russian");

int choice=1;

while (choice!=0){

std::cout << "\n\nВведите 1 - ПОКАЗАТЬ 4 ЛАБУ\n" << "Введите 2 - ПОКАЗАТЬ 7 ЛАБУ\n" << "Введите 3 - ПОКАЗАТЬ 8 ЛАБУ\n" << "Введите 0 - ВЫХОД\n" << "ваш выбор: ";

std::cin >> choice;

std::cout << std::endl;

if (choice == 0) { break; }

if (choice == 1) {

//////////////////////////////////////STATIC OBJECT////////

std::cout << "\n\nSTATIC OBJECT\n\n";

Engine\* bmw\_engine = new Engine;

bmw\_engine->init(0, 4395, 625, 8);

Car bmw\_x6;

bmw\_x6.init("BMW\_X6", 3500000, "black", 0, 0, bmw\_engine); //инициализируем поля объекта

bmw\_x6.displayDataCar();

//bmw\_x6.readCarData();

bmw\_x6.displayDataCar();

bmw\_x6.startEngine(); //пытаемся завести двигатель

bmw\_x6.displayDataCar();

bmw\_x6.addBenzine(10); //добавляем бензин

bmw\_x6.displayDataCar();

bmw\_x6.startEngine(); //снова пытаемся завести двигатель

bmw\_x6.displayDataCar();

for (int i = 0; i < 4; i++) {

bmw\_x6.addSpeed(i \* 5); //добавляем скорость

bmw\_x6.displayDataCar();

}

for (int i = 0; i < 4; i++) {

bmw\_x6.reduceSpeed(i \* 5); //убавляем скорость

bmw\_x6.displayDataCar();

}

bmw\_x6.stopEngine(); //останавливаем двигатель

bmw\_x6.displayDataCar();

std::cout << "\n\nPress key to continue!\n\n";

\_getch();

std::system("cls");

////////////////////////////////////DYNAMIC OBJECT/////

std::cout << "\n\nDYNAMIC OBJECT\n\n";

Car\* audi\_a7 = new Car;

Engine\* audi\_engine = new Engine;

audi\_engine->init(0, 2995, 340, 6);

audi\_a7->init("audi", 2000000, "blue", 0, 0, audi\_engine); //инициализируем поля объекта

audi\_a7->displayDataCar();

//audi\_a7->readCarData();

audi\_a7->displayDataCar();

audi\_a7->startEngine(); //пытаемся завести двигатель

audi\_a7->displayDataCar();

audi\_a7->addBenzine(10); //добавляем бензин

audi\_a7->displayDataCar();

audi\_a7->startEngine(); //снова пытаемся завести двигатель

audi\_a7->displayDataCar();

for (int i = 0; i < 4; i++) {

audi\_a7->addSpeed(i \* 5); //добавляем скорость

audi\_a7->displayDataCar();

}

for (int i = 0; i < 4; i++) {

audi\_a7->reduceSpeed(i \* 5); //убавляем скорость

audi\_a7->displayDataCar();

}

audi\_a7->stopEngine(); //останавливаем двигатель

audi\_a7->displayDataCar();

delete audi\_a7;

///////////////////////////////////DYNAMIC OBJECTS ARRAY///

Car\* dynamic\_obj\_cars;

dynamic\_obj\_cars = new Car[5];

delete[] dynamic\_obj\_cars;

///////////////////////////////////MALLOC ///

Car\* dynamic\_obj\_car2;

dynamic\_obj\_car2 = (Car\*)malloc(CAR\_NUMBERS \* sizeof(Car));

free(dynamic\_obj\_car2);

//////////////////////////////////CALLOC /////

Car\* dynamic\_obj\_car3;

dynamic\_obj\_car3 = (Car\*)calloc(CAR\_NUMBERS, sizeof(Car));

free(dynamic\_obj\_car3);

//////////////////////////////////REALOC - change current allocated memory

Car\* dynamic\_obj\_car4 = (Car\*)calloc(2, sizeof(Car));

Car\* tmp;

if (dynamic\_obj\_car4 != NULL)

{

tmp = (Car\*)realloc(dynamic\_obj\_car4, CAR\_NUMBERS \* sizeof(Car));

if (tmp != NULL)

{

dynamic\_obj\_car4 = tmp;

}

free(dynamic\_obj\_car4);

}

}

if (choice == 2) {

//возврат значения из метода через указатель(\*)

int\* myArray = allocateArray(20);

// делаем что-нибудь с array

delete[] myArray;

//возврат значения из метода через ссылку(&)

std::cout << "Количество вызовов функции: " << getCallNumber() << "\n";

//перегрузка

Engine\* bmw\_engine = new Engine;

bmw\_engine->init(0, 4395, 625, 8);

Car bmw\_x6;

bmw\_x6.init("BMW\_X6", 3500000, "black", 0, 10, bmw\_engine); //инициализируем поля объекта

bmw\_x6.displayDataCar();

bmw\_x6 = bmw\_x6 + 10;//перегрузка оператора + (добавляем бензин)

bmw\_x6.displayDataCar();

bmw\_x6 = ++bmw\_x6; //(добавляем бензин)

bmw\_x6.displayDataCar();

bmw\_x6 = bmw\_x6++; //(добавляем бензин)

bmw\_x6.displayDataCar();

}

if (choice == 3) {

std::cout << "count =" << Car::getCount() << "\n";

Car car1;

std::cout << "После создания одного статического объекта Car\ncount =" << Car::getCount() << "\n";

Car \*car2 = new Car();

std::cout << "После создания одного динамического объекта Car\ncount =" << Car::getCount() << "\n";

Car car[5];

std::cout << "После создания массива из 5 статических объектов Car\ncount =" << Car::getCount() << "\n";

}

}

std::cout << "Вы вышли\n";

}

void setStartPosition(Car\* car)

{

car->x = 0;

car->y = 0;

}

int& getCallNumber()

// Возвращаем ссылку на индекс элемента массива

{

static int number;

if (number == NULL) number = 1;

else number++;

return number;

}

int\* allocateArray(int size)

//функция выделения памяти (возврат значения по адресу(указателю))

{

return new int[size];

}

ТЕСТЫ

