МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНЖЕНЕРНО-ТЕХНОЛОГИЧЕСКАЯ АКАДЕМИЯ

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**Лабораторная работа № 1**

по курсу «Объектно-ориентированное программирование»

**«Классы и объекты в C++»**

**14 ВАРИАНТ**

Выполнили:

студенты гр. КТбо2-1

Неприн М.А.

Проверил:

Тарасов С. А.

**Таганрог 2020**

1. **Вариант задания №14**

Определить классы Автостоянка (Parking) и Автомобиль (Car). Для каждого автомобиля во втором классе задается госномер, марка, цвет и методы Парковаться и Покинуть парковку. Если авто паркуется, то его данные заносятся в массив Автостоянки с соответствующей отметкой. При выезде эта отметка снимается, но данные остаются. Программа должна отвечать на запрос, присутствует ли конкретный автомобиль на парковке, а также по запросу вывести список присутствующих автомобилей.

1. **Спецификация классов**

class Car

{

public:

Car();

Car(std::string carNumber, std::string model, std::string color);

~Car();

bool operator== (const Car &other);

void park(Parking &parking);

void unpark();

void setCarNumber(std::string& carNumber);

std::string getCarNumber() const;

std::string getModel() const;

void setColor(std::string& color);

std::string getColor() const;

Parking\* getCurrentParking() const;

bool isParked();

private:

std::string \_carNumber;

const std::string \_model;

std::string \_color;

Parking \*\_current\_parking;

};

class Parking

{

public:

Parking(int parking\_spaces);

~Parking();

void saveNewCar(Car &car);

void park(Car &car);

void unpark(Car &car);

bool isCarParked(Car &car);

Car\*\* getParkedCars();

Car\*\* getAllCars();

private:

static const int all\_cars\_multiplyer = 100;

const int parking\_spaces;

int \_all\_cars\_iter = 0;

int \_current\_cars\_iter = 0;

Car \*\*\_all\_cars = new Car\*[parking\_spaces \* all\_cars\_multiplyer]{nullptr};

Car \*\*\_current\_cars = new Car\*[parking\_spaces]{nullptr};

};

class Interface

{

public:

Interface();

~Interface();

void printMenuText();

void prompt(); //Приглашение командной строки

void printCarArray(Car\* array);

void printCarArray(Car\*\* array);

void printParkedCars(Parking &parking);

int ReadInt();

private:

int cars\_amount = start\_cars;

Car \*cars = new Car[start\_cars]

{

Car("x235aq23", "Honda", "White"),

Car("a241as77", "Toyota", "White"),

Car("e137rz11", "Ford", "Grey"),

Car("h888qq777", "Kia", "Blue"),

Car("e849dl93", "Peugeot", "Black")

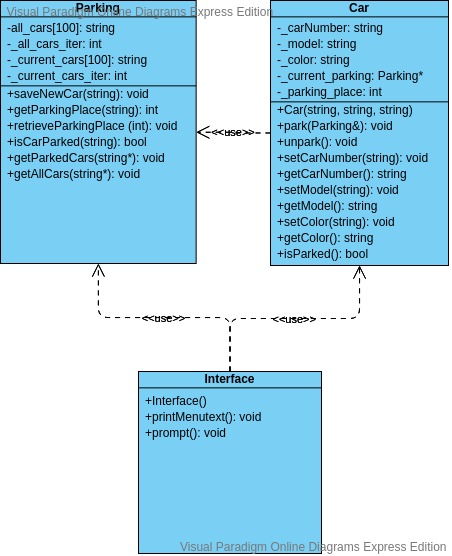
};

};

1. **Используемые математические зависимости и алгоритмы**

В этой лабораторной работе математические зависимости и алгоритмы, например хеширования, не присутствуют.

1. **Диаграммы классов**



1. **Листинг программы**

**main.cpp**

/\*

// ЮФУ ИКТИБ МОП ЭВМ

// Неприн Михаил Андреевич, КТбо2-1

// Лабораторная работ №1, Вариант №14

// 05.09.2020

\*/

#include <iostream>

#include "interface.h"

int main()

{

Interface interface;

return 0;

}

**parking.h**

#ifndef PARKING\_H

#define PARKING\_H

#include "car.h"

#include <string>

class Car;

class Parking

{

public:

Parking(int parking\_spaces);

~Parking();

void saveNewCar(Car &car);

void park(Car &car);

void unpark(Car &car);

bool isCarParked(Car &car);

Car\*\* getParkedCars();

Car\*\* getAllCars();

private:

static const int all\_cars\_multiplyer = 100;

const int parking\_spaces;

int \_all\_cars\_iter = 0;

int \_current\_cars\_iter = 0;

Car \*\*\_all\_cars = new Car\*[parking\_spaces \* all\_cars\_multiplyer]{nullptr};

Car \*\*\_current\_cars = new Car\*[parking\_spaces]{nullptr};

};

#endif // PARKING\_H

**parking.cpp**

/\*

// ЮФУ ИКТИБ МОП ЭВМ

// Неприн Михаил Андреевич, КТбо2-1

// Лабораторная работ №1, Вариант №14

// 05.09.2020

\*/

#include "parking.h"

#include <iostream>

#include <stdexcept>

Parking::Parking(int parking\_spaces): parking\_spaces(parking\_spaces)

{};

Parking::~Parking()

{

delete[] \_all\_cars;

delete[] \_current\_cars;

}

void Parking::saveNewCar(Car &car)

{

try{

\_all\_cars[\_all\_cars\_iter] = &car;

} catch(std::range\_error &e) {

throw std::range\_error("Not enough memory to save a new car");

return;

}

++\_all\_cars\_iter;

return;

}

void Parking::park(Car &car)

{

bool hasFound = false;

for (int i = 0; i < \_all\_cars\_iter; ++i) {

if (car == \*\_all\_cars[i]) {

hasFound = true;

break;

}

}

if (!hasFound) {

try

{

try {

while (\_current\_cars[\_current\_cars\_iter] != nullptr){

++\_current\_cars\_iter;

}

} catch (std::range\_error &no\_space) {

throw std::range\_error("No parking space to park this car");

}

\_current\_cars[\_current\_cars\_iter] = &car;

saveNewCar(car);

} //try

catch(std::range\_error &e)

{

throw &e;

}

}

return;

}

void Parking::unpark(Car &car)

{

for (int i = 0; i < parking\_spaces; i++)

{

if (car == \*\_current\_cars[i])

{

\_current\_cars[i] = nullptr;

break;

}

}

return;

}

bool Parking::isCarParked(Car &car)

{

for (int i = 0; i < parking\_spaces; ++i){

if (\*\_current\_cars[i] == car){

return true;

}

}

return false;

}

Car\*\* Parking::getParkedCars()

{

return \_current\_cars;

}

Car\*\* Parking::getAllCars()

{

return \_all\_cars;

}

**car.h**

#ifndef CAR\_H

#define CAR\_H

#include "parking.h"

#include <string>

class Parking;

class Car

{

public:

Car();

Car(std::string carNumber, std::string model, std::string color);

~Car();

bool operator== (const Car &other);

void park(Parking &parking);

void unpark();

void setCarNumber(std::string& carNumber);

std::string getCarNumber() const;

std::string getModel() const;

void setColor(std::string& color);

std::string getColor() const;

Parking\* getCurrentParking() const;

bool isParked();

private:

std::string \_carNumber;

const std::string \_model;

std::string \_color;

Parking \*\_current\_parking;

};

#endif // CAR\_H

**car.cpp**

/\*

// ЮФУ ИКТИБ МОП ЭВМ

// Неприн Михаил Андреевич, КТбо2-1

// Лабораторная работ №1, Вариант №14

// 05.09.2020

\*/

#include "car.h"

Car::Car() : \_carNumber(nullptr), \_model(nullptr), \_color(nullptr)

{};

Car::Car(std::string carNumber, std::string model, std::string color)

: \_carNumber(carNumber), \_model(model), \_color(color)

{};

Car::~Car()

{

\_current\_parking = nullptr;

}

bool Car::operator==(const Car &other)

{

if (this->\_color == other.\_color

&& this->\_carNumber == other.\_carNumber

&& this->\_model == other.\_model)

return true;

return false;

}

void Car::park(Parking &parking)

{

parking.park(\*this);

\_current\_parking = &parking;

return;

}

void Car::unpark()

{

\_current\_parking->unpark(\*this);

\_current\_parking = nullptr;

return;

}

bool Car::isParked()

{

return ((\_current\_parking != nullptr) ? true : false);

}

void Car::setColor(std::string &color)

{

\_color = color;

return;

}

std::string Car::getColor() const

{

return \_color;

}

std::string Car::getModel() const

{

return \_model;

}

Parking\* Car::getCurrentParking() const

{

return \_current\_parking;

}

void Car::setCarNumber(std::string &carNumber)

{

\_carNumber = carNumber;

return;

}

std::string Car::getCarNumber() const

{

return \_carNumber;

}

**interface.h**

#ifndef INTERFACE\_H

#define INTERFACE\_H

#include "parking.h"

#include "car.h"

const int start\_cars = 5;

class Interface

{

public:

Interface();

~Interface();

void printMenuText();

void prompt(); //Приглашение командной строки

void printCarArray(Car\* array);

void printCarArray(Car\*\* array);

void printParkedCars(Parking &parking);

int ReadInt();

private:

int cars\_amount = start\_cars;

Car \*cars = new Car[start\_cars]

{

Car("x235aq23", "Honda", "White"),

Car("a241as77", "Toyota", "White"),

Car("e137rz11", "Ford", "Grey"),

Car("h888qq777", "Kia", "Blue"),

Car("e849dl93", "Peugeot", "Black")

};

};

#endif // INTERFACE\_H

**interface.cpp**

/\*

// ЮФУ ИКТИБ МОП ЭВМ

// Неприн Михаил Андреевич, КТбо2-1

// Лабораторная работ №1, Вариант №14

// 05.09.2020

\*/

#include "interface.h"

#include <iostream>

using namespace std;

Interface::Interface()

{

const int parking\_size = 20;

bool running = true;

Parking parking(parking\_size);

printMenuText();

while (running)

{

int input;

input = ReadInt();

switch (input) {

case 0: {

running = false;

break;

}

case 1: {

int x = 0;

bool doNotPark=false;

do{

doNotPark=false;

cout << "Введите номер машины, которую нужно поставить на парковку.\n0 - чтобы отменить действие\n";

printCarArray(cars);

x = ReadInt();

if (x > this->cars\_amount)

{

doNotPark = true;

cout << "Машин всего " << this->cars\_amount << endl;

}

else if (x < 0)

{

doNotPark = true;

cout << "Неверный ввод\n";

}

else if (x == 0){

doNotPark = true;

break;

}

else if (cars[x-1].isParked())

{

doNotPark = true;

cout << "Эта машина уже припаркована\n";

}

}while (doNotPark);

if (!doNotPark)

cars[x-1].park(parking);

cout << endl << "Припаркованные машины:" << endl;

printParkedCars(parking);

cout << endl;

printMenuText();

break;

}

case 2: {

int x = 0;

bool doNotPark=false;

do{

doNotPark = false;

cout << "Введите номер машины, которую нужно убрать с парковки.\n0 - чтобы отменить действие\n";

printCarArray(cars);

x = ReadInt();

if (x > this->cars\_amount)

{

doNotPark = true;

cout << "Машин всего " << this->cars\_amount << endl;

}

else if (x < 0)

{

doNotPark = true;

cout << "Неверный ввод\n";

}

else if (x == 0) {

doNotPark = true;

break;

}

else if (!cars[x-1].isParked())

{

doNotPark = true;

cout << "Эта машина не на парковке\n";

}

}while (doNotPark);

if (!doNotPark)

cars[x-1].unpark();

cout << endl << "Припаркованные машины:" << endl;

printParkedCars(parking);

cout << endl;

printMenuText();

break;

}

case 3: {

cout << "Припаркованные машины:" << endl;

printParkedCars(parking);

printMenuText();

break;

}

case 4: {

cout << "Все машины:" << endl;

printCarArray(cars);

printMenuText();

break;

}

case 5: {

Car\*\* all\_cars = parking.getAllCars();

cout << "Машины зафиксированные на парковке:" << endl;

printCarArray(all\_cars);

break;

}

case 6: printMenuText(); break;

}

} // while (running)

}

Interface::~Interface()

{

delete[] cars;

}

void Interface::printMenuText() {

std::cout << "\nВведите 1, чтобы поставить на парковку машину.\nВведите 2, чтобы убрать машину с парковки.\n"

"Введите 2, чтобы добавить новый автомобиль\n"

"Введите 3, чтобы вывести список всех припаркованных автомобилей.\n"

"Введите 4, чтобы вывести список всех автомобилей\n"

"Введите 5, чтобы вывести список всех автомобилей, зафиксированных на парковке\n"

"Введите 6, чтобы повторить этот текст\n"

"Введите 0, чтобы выйти.\n\n";

return;

}

void Interface::prompt() {

std::cout << "$ ";

return;

}

void Interface::printCarArray(Car\* array)

{

for (int i = 0; i < this->cars\_amount ; ++i)

{

cout << "(" << array[i].getCarNumber() << ", " << array[i].getModel() << ", " << array[i].getColor() << ")" << endl;

}

}

void Interface::printCarArray(Car\*\* array)

{

for (size\_t i = 0; array[i] != nullptr; ++i)

{

cout << "(" << array[i]->getCarNumber() << ", " << array[i]->getModel() << ", " << array[i]->getColor() << ")" << endl;

}

}

void Interface::printParkedCars(Parking &parking)

{

Car \*\*parked\_cars;

parked\_cars = parking.getParkedCars();

printCarArray(parked\_cars);

}

int Interface::ReadInt()

{

int a = 0;

prompt();

cin >> a;

if (cin.fail())

{

do

{

prompt();

cin.clear();

cin.ignore(INT64\_MAX, '\n');

cin >> a;

}while (cin.fail());

}

return a;

}