ФГБОУ ВО

Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет) «МГТУ им. Н.Э. Баумана»

Факультет ИУ «Информатика и системы управления»

Кафедра ИУ-3 «Информационные системы и телекоммуникации»

Отчет по лабораторной работе 18

**«ООП»**

По курсу «Информатика»

Выполнил: Еньшин Б. А.

Группа ИУ3-12

Вариант 5

Проверил: Видьманов Д. А.

Москва 2019

**Цель работы:**

Пополнить знания об основах ООП и основных его принципах, научиться использовать их эффективно на практике. Используя полученные знания, реализовать транспортную структуру средствами ООП в языке программирования С++.

**Постановка задания:**

«Вы участвуете в разработке компьютерной игры. В качестве объектов игрового мира выступают средства транспорта различных типов. В следующих четырех пунктах описывается работа с тремя классами, каждый из которых реализует свой тип транспортного средства. Далее дано описание каждого из классов. Требуется сначала реализовать эти классы, затем протестировать их с помощью описанных ниже процедур. Далее требуется выполнить, по крайней мере, одно задание из пяти предлагаемых вариантов и снова протестировать работоспособность программы. Наконец необходимо нарисовать получившуюся в итоге диаграмму классов.»

Дополнительное задание для варианта №1:

«По аналогии с шаблоном сортировки выбором реализуйте шаблон сортировки вставками»

**Ниже приведен код программы lab1\_source.cpp**

#include <iostream>

#include "stdlib.h"

#include <vector>

#include "math.h"

#include <iomanip>

//#include "Vehicle.h"

using namespace std;

const char DefaultVehicleName[] = "Untyped Vehicle";

const char DefaultCoachName[] = "Default Coach";

const char DefaultAutomobileName[] = "Default Automobile";

const char DefaultAeroplaneName[] = "Default Aeroplane";

const double DefaultVehicleSpeed = -1.;

const double DefaultCoachSpeed = 10.;

const double DefaultAutomobileSpeed = 100.;

const double DefaultAeroplaneSpeed = 500.;

const double DefaultTimeToBoot = 0.01;

class Computer

{

public:

Computer() : base\_time\_to\_boot(DefaultTimeToBoot) {};

~Computer() {};

double GetTimeToBoot() { return base\_time\_to\_boot; }

void ComputerUpdate(double new\_\_base\_time\_to\_boot) { base\_time\_to\_boot = new\_\_base\_time\_to\_boot; }

protected:

double base\_time\_to\_boot;

};

class Vehicle

{

public:

Vehicle() : start\_speed(DefaultVehicleSpeed), total\_distance(0), total\_time(0) { SetName(DefaultVehicleName); };

Vehicle(string name, double start\_speed) : start\_speed(start\_speed), total\_distance(0), total\_time(0) { SetName(name); };

virtual ~Vehicle() { };

string GetName() { return name; };

double GetBaseSpeed() { return start\_speed; };

double GetTotalDistance() { return total\_distance; };

double GetTotalTime() { return total\_time; };

virtual bool MakeTrip(double distanceOfTrip)=0;

virtual double GetSpeed() = 0;

protected:

double total\_distance;

double total\_time;

private:

string name;

double start\_speed;

void SetName(string name) { if (name.size() < 50) this->name = name; }

};

class Coach : public Vehicle

{

public:

Coach(void) : Vehicle(DefaultCoachName, DefaultCoachSpeed) {};

Coach(string name, double start\_speed) : Vehicle (name, start\_speed) {};

~Coach() {};

virtual bool MakeTrip(double distance\_of\_trip)

{

if (GetSpeed() <= 0) return false;

if (distance\_of\_trip <= MAX\_DISTANCE)

{

double time\_of\_trip = distance\_of\_trip / GetSpeed();

total\_time += time\_of\_trip; total\_distance += distance\_of\_trip;

return true;

}

else return false;

}

virtual double GetSpeed()

{

return GetBaseSpeed() \* exp(-total\_time / 500.);

}

private:

const double MAX\_DISTANCE = 1000;

};

class Automobile : public Vehicle

{

public:

Automobile(void) : Vehicle(DefaultAutomobileName, DefaultAutomobileSpeed) {};

Automobile(string name, double start\_speed) : Vehicle(name, start\_speed) {};

~Automobile() {};

virtual bool MakeTrip(double distance\_of\_trip)

{

if (GetSpeed() <= 0) return false;

double time\_of\_trip = distance\_of\_trip / GetSpeed();

total\_distance += distance\_of\_trip;

total\_time += time\_of\_trip;

}

virtual double GetSpeed() { return GetBaseSpeed() \* exp(-total\_distance / 500.); }

};

class Aeroplane : public Vehicle, public Computer

{

public:

const double MaxFlyTime = 20000;

Aeroplane(void) :Vehicle(DefaultAeroplaneName, DefaultAeroplaneSpeed),Computer(), time\_since\_last\_repair(0) {};

Aeroplane(string name, double start\_speed) : Vehicle(name, start\_speed), Computer(), time\_since\_last\_repair(0) {};

~Aeroplane() {};

virtual bool MakeTrip(double distance\_of\_trip)

{

double time\_of\_trip = distance\_of\_trip / GetSpeed() + GetTimeToBoot();

if (time\_since\_last\_repair + time\_of\_trip > MaxFlyTime)return false;

time\_since\_last\_repair += time\_of\_trip;

total\_time += time\_of\_trip;

total\_distance += distance\_of\_trip;

return true;

}

virtual double GetSpeed()

{ return GetBaseSpeed(); }

void Repair() { time\_since\_last\_repair = 0; }

double GetTimeSinceLastRepair() { return time\_since\_last\_repair; }

private:

double time\_since\_last\_repair;

};

template <class MyType> void MySwap(MyType& v1, MyType& v2);

template <class ArrayType, class LessFunctionType> int FindMinimumIndex(ArrayType& data\_array, int beginIndex, int endIndex, LessFunctionType LessFunction);

template <class ArrayType, class LessFunctionType> void SelectionSort(ArrayType& data\_array, int beginIndex, int endIndex, LessFunctionType LessFunction);

template <class ArrayType, class LessFunctionType> void InsertionSort(ArrayType& data\_array, int beginIndex, int endIndex, LessFunctionType LessFunction);

bool CompareNames(Vehicle\* lhs, Vehicle\* rhs);

bool CompareBaseSpeeds(Vehicle\* lhs, Vehicle\* rhs);

bool CompareCurrentSpeeds(Vehicle\* lhs, Vehicle\* rhs);

void DisplayVehicles(const vector<Vehicle\*>& vehicles);

void CommitRandomTrips(vector<Vehicle\*>& vehicles);

void main() {

srand(0);

vector<Coach> coaches;

vector<Automobile> automobiles;

vector<Aeroplane> aeroplanes;

vector<Vehicle\*>coach\_pointers;

vector<Vehicle\*>automobile\_pointers;

vector<Vehicle\*>aeroplane\_pointers;

coaches.push\_back(Coach("old coach", 10.));

coaches.push\_back(Coach("common coach", 15.));

coaches.push\_back(Coach("super coach", 20.));

coaches.push\_back(Coach());

automobiles.push\_back(Automobile());

automobiles.push\_back(Automobile("minivan", 110.));

automobiles.push\_back(Automobile("sportscar", 200.));

automobiles.push\_back(Automobile("pickup",140.));

aeroplanes.push\_back(Aeroplane());

aeroplanes.push\_back(Aeroplane("hawk", 700.));

aeroplanes.push\_back(Aeroplane("passenger liner", 600.));

aeroplanes.push\_back(Aeroplane("baggage liner", 550));

vector<Vehicle\*>all\_vehicle\_pointers;

for (int i = 0; i < coaches.size(); i++) { coach\_pointers.push\_back(&coaches[i]); all\_vehicle\_pointers.push\_back(&coaches[i]); }

for (int i = 0; i < automobiles.size(); i++) { automobile\_pointers.push\_back(&automobiles[i]); all\_vehicle\_pointers.push\_back(&automobiles[i]);}

for (int i = 0; i < aeroplanes.size(); i++) { aeroplane\_pointers.push\_back(&aeroplanes[i]); all\_vehicle\_pointers.push\_back(&aeroplanes[i]);}

DisplayVehicles(all\_vehicle\_pointers);

InsertionSort(all\_vehicle\_pointers, 0, 11, CompareCurrentSpeeds);

DisplayVehicles(all\_vehicle\_pointers);

CommitRandomTrips(automobile\_pointers);

CommitRandomTrips(coach\_pointers);

CommitRandomTrips(aeroplane\_pointers);

DisplayVehicles(all\_vehicle\_pointers);

}

void DisplayVehicles(const vector<Vehicle\*>& vehicles)

{

cout<<setw(30)<<left<<"NAME"<<setw(16)<<"BASE SPEED"<<setw(16)<<"CURRENT SPEED"<<setw(12)<<"distance"<<setw(12)<<"time"<<endl;

for (int i = 0; i < vehicles.size(); ++i)

{

cout << setw(30) << vehicles[i]->GetName() << setw(16) << vehicles[i]->GetBaseSpeed() <<setw(16)<<vehicles[i]->GetSpeed()<< setw(12) << vehicles[i]->GetTotalDistance() << setw(12) << vehicles[i]->GetTotalTime() << endl;

}

cout << "\n\n";

}

void CommitRandomTrips(vector<Vehicle\*>& vehicles)

{

for (int i = 0; i < vehicles.size(); i++)

{

double rand\_distance = double(rand() % 20001) / 10.;

vehicles[i]->MakeTrip(rand\_distance);

}

}

template <class MyType> void MySwap(MyType& v1, MyType& v2) {

MyType v3 = v1;

v1 = v2;

v2 = v3;

}

template <class ArrayType, class LessFunctionType>

int FindMinimumIndex(ArrayType& data\_array, int beginIndex, int endIndex, LessFunctionType LessFunction)

{

int minimumIndex = beginIndex;

for (int element\_number = beginIndex + 1; element\_number <= endIndex; ++element\_number)

{

if (LessFunction(data\_array[element\_number], data\_array[minimumIndex]))

{ minimumIndex = element\_number; }

} return minimumIndex; }

template <class ArrayType, class LessFunctionType>

void SelectionSort(ArrayType& data\_array, int beginIndex, int endIndex, LessFunctionType LessFunction)

{

for (int element\_number = beginIndex; element\_number < endIndex; ++element\_number)

{

int minimumIndex = FindMinimumIndex(data\_array, element\_number, endIndex, LessFunction);

MySwap(data\_array[minimumIndex], data\_array[element\_number]);

}

}

// вариант 1

template<class ArrayType, class LessFunctionType>

void InsertionSort(ArrayType& data\_array, int beginIndex, int endIndex, LessFunctionType LessFunction)

{

for (int element\_number = beginIndex; element\_number <= endIndex; element\_number++) {

for (int previous\_number = beginIndex; previous\_number<element\_number; previous\_number++) if (LessFunction(data\_array[element\_number], data\_array[previous\_number])) {

MySwap(data\_array[previous\_number], data\_array[element\_number]);

continue;

}

}

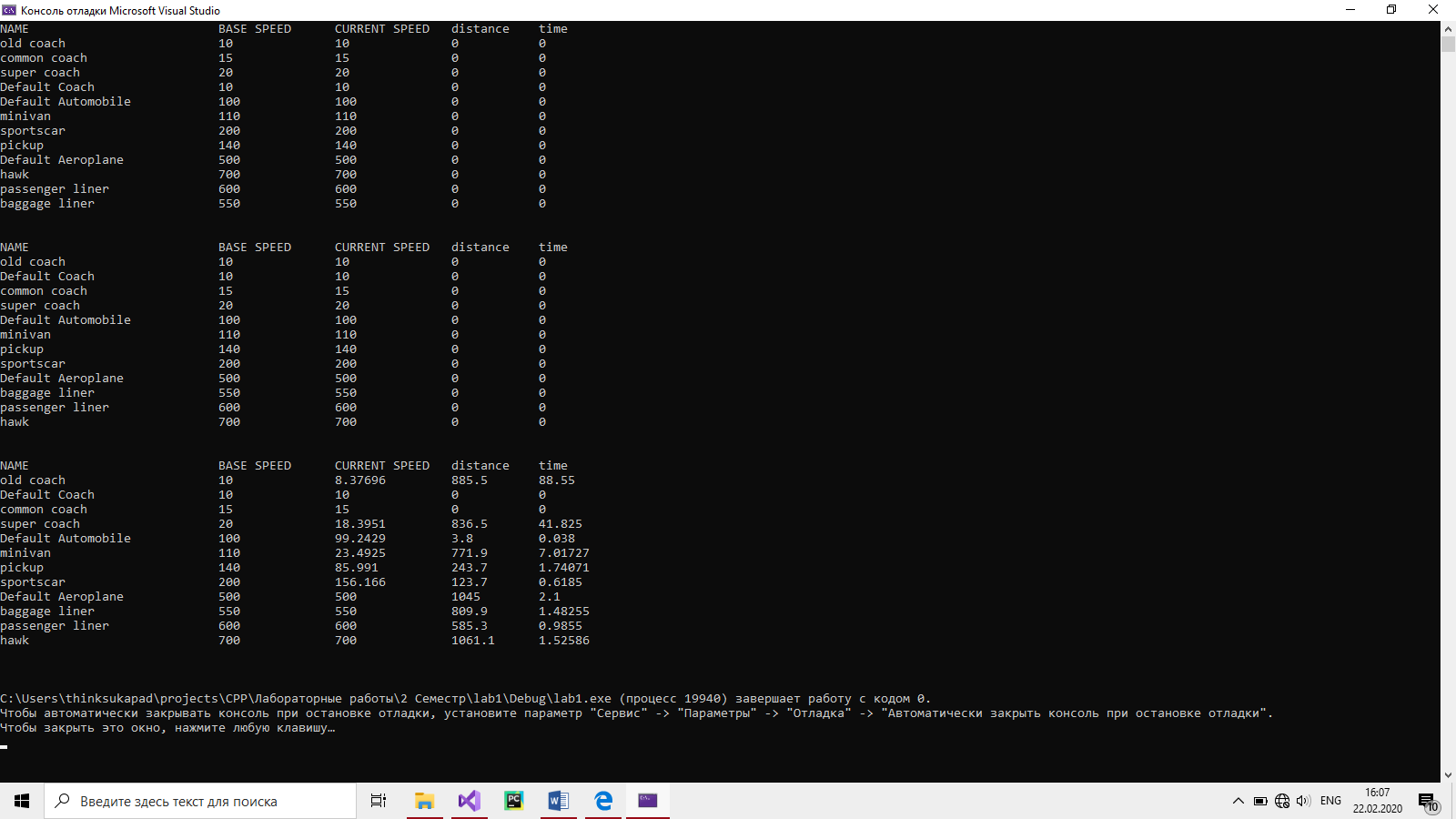
}

bool CompareNames(Vehicle\* lhs, Vehicle\* rhs) { return lhs->GetName() < rhs->GetName(); }

bool CompareBaseSpeeds(Vehicle\* lhs, Vehicle\* rhs) { return lhs->GetBaseSpeed() < rhs->GetBaseSpeed(); }

bool CompareCurrentSpeeds(Vehicle\* lhs, Vehicle\* rhs) { return lhs->GetSpeed() < rhs->GetSpeed(); }

**Скриншот консоли вывода программы**



Вывод:

В ходе данной лабораторной работы мы закрепили свои знания ООП и таких его основных принципах, как наследование, полиморфизм и инкапсуляция.

Полученные и подкрепленные знания мы подтвердили написанием программы на С++, работающей с разными видами транспорта путем наследования из базового класса «Транспорт».