Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут”

Кафедра АСОІУ

**ЗВІТ**

про виконання комп’ютерного практикуму № 6

з дисципліни

“ООП”

Тема: «Шаблоні класи С++»

|  |  |  |
| --- | --- | --- |
| Прийняв: |  | Виконав: |
| Головченко Максим Миколайович |  | студент 2-го курсу  гр. ІП-51 ФІОТ  Зарічковий Олександр Анатолійович |

Київ – 2016

**ЗМІСТ:**

[1 Мета роботи 3](#_Toc463896091)

[2 Постановка задачі (варіант 7, рівень Б) 4](#_Toc463896092)

[3 Діаграма класів 5](#_Toc463896093)

[4 Код програми 6](#_Toc463896094)

[5 Приклади виконання програми 10](#_Toc463896095)

[6 Висновок 11](#_Toc463896096)

# Мета роботи

Цель работы – изучить особенности шаблонов функций и шаблонов классов С++. Освоить принципы работы шаблонов классов для работы стбазовыми типами и пользовательскими типами.

# Постановка задачі (варіант 7, рівень В)

Работа таксопарка. Имеется таксопарк, в котором есть штат водителей с автомобилями. Диспетчер автопарка принимает телефонные заказы, фиксируя время заказа, адрес и телефон вызова, а также пункт назначения.

После регистрации заказа диспетчер проверяет возможность его выполнения. Если в таксопарке имеются свободные водители, то они назначаются на выполнение заказа, причём выбирается тот водитель. Если все автомобили заняты, то диспетчер связывается с водителями на маршруте, и выясняет, смогут ли они выполнить заказ в установленное время. После этого диспетчер связывается с клиентом, сообщая ему о возможности/невозможности выполнения заказа.

При выполнении заказа водитель оформляет две квитанции на оплату услуг по перевозке. Клиент расплачивается и расписывается в квитанциях. Одна квитанция остаётся у клиента, а вторую водитель передаёт диспетчеру вместе с деньгами при возвращении в таксопарк.

Сформировать коллекцию данных с информацией обо всех рейсах, осуществленных за день.

Задание.

Отобразить информацию по сумме выручки каждого водителя.

# Діаграма класів

Діаграма класів наведена на рисунку 3.1:

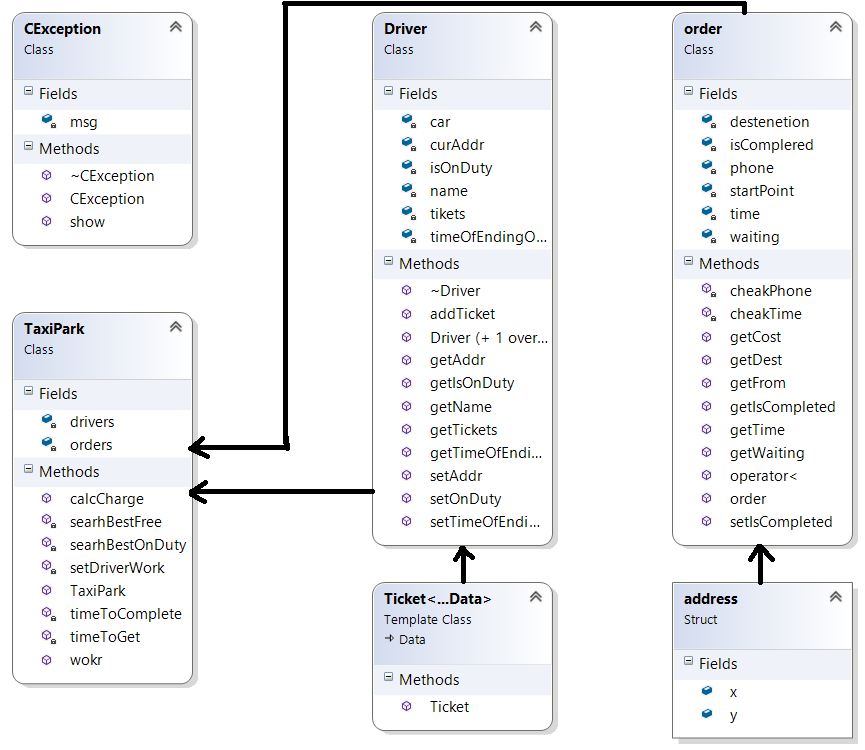


Рисунок 3.1 – Діаграма класів

# Код програми

“main.cpp”

#include "Interface.h"

int main(void) {

while (outputMenu());

}

“Interface.h”

#pragma once

#define size 1024

bool outputMenu(void); // Виведення меню

char\* inputStr(char c); // Введення строки

“Interface.cpp”

#include "Interface.h"

#include "TaxiPark.h"

#include <iostream>

#include <cstdlib>

#include "CException.h"

using std::cout;

#define size 1024

bool outputMenu(void) {

system("cls");

cout << "OOP lab work #6\nVariant - 7, Level - C\nTask: Write variadic template class\nWriter: Alexander ALEXKIRNAS Zarichkovyi\n\n";

cout << "Menu:\n1. Start program\n2. Exit\n\nMake your choice ";

try {

char str[size];

gets\_s(str, size);

if (strlen(str) != 1) throw CException("Wrong input format!!!");

switch (str[0]) {

case '1': {

Driver drivers[] = {

Driver("A", "1"),

Driver("B", "2")

};

order orders[] = {

order("00:45",{ 10, 5 },{ 4, 3 }, "+123456789012", 10),

order("00:45",{ 11, 3 },{ 0, 0 }, "+123456789012", 20),

order("01:00",{ 0, 2 },{ 11, 5 }, "+123456789012", 2),

order("12:45",{ 12, 35 },{ 40, 30 }, "+123456789012", 15),

order("10:45",{ 12, 5 },{ 40, 30 }, "+123456789012", 100),

order("10:35",{ 10, 50 },{ 4, 3 }, "+123456789012", 0),

order("00:58",{ 10, 7 },{ 11, 4 }, "+123456789012", 2)

};

TaxiPark park(drivers, sizeof(drivers) / sizeof(Driver), orders, sizeof(orders) / sizeof(order));

park.wokr();

park.calcCharge();

printf("\n");

system("pause");

break;

}

case '2': return false;

default: throw CException("Unexpected input!");

}

return true;

}

catch (CException& excection) {

excection.show();

system("pause");

return true;

}

catch (...) {

std::cout << "Ops. Something go wrong =(\n\n";

system("pause");

return true;

}

}

“TaxiPark.h”

#pragma once

#include "Driver.h"

#include "Order.h"

#include "Tiket.h"

#include <vector>

#include "Order.h"

#include "Driver.h"

using std::vector;

class TaxiPark {

private:

vector <order> orders;

vector <Driver> drivers;

int searhBestFree(int orderIndex); // Searching best driver for curr order

int searhBestOnDuty(int orderIndex); // Searching best driver for curr order

void setDriverWork(int indexOfDriver, int indexOfOrder, short curTime); // Setting driver to complete new order

short timeToGet(int indexOfDriver, int indexOfOrder); // Calculation of time to get to the costumer

short timeToComplete(int indexOfDriver, int indexOfOrder); // Calculation of time to finish order

public:

TaxiPark(Driver\* drivers, int sizeDrivers, order\* orders, int sizeOrders);

void calcCharge(void);

void wokr(void);

};

“TaxiPark.cpp”

#include "TaxiPark.h"

#include <algorithm>

#include <set>

#include <utility>

#include <iostream>

using std::cout;

using std::set;

TaxiPark::TaxiPark(Driver\* drivers, int sizeDrivers, order\* orders, int sizeOrders) {

for (int i = 0; i < sizeDrivers; i++) this->drivers.push\_back(drivers[i]);

for (int i = 0; i < sizeOrders; i++) this->orders.push\_back(orders[i]);

}

void TaxiPark::wokr(void) {

class eventElement {

public:

short time;

bool type;

int index;

bool operator< (const eventElement& obj) const {

if (time != obj.time) return time < obj.time;

if (index != obj.index) return index < obj.index;

if(type != obj.type) return type;

else return false;

}

};

set <eventElement> Events; // type of event (0 - woker is free, 1 - new order)

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

Events.insert({ orders[i].getTime(), true, i });

cout << "Start woking:\n\n";

while (!Events.empty()) {

auto iter = Events.begin();

if (iter->type) {

int index = searhBestFree(iter->index);

if (index != drivers.size()) {

orders[iter->index].setIsCompleted(true);

drivers[index].addTicket(Ticket<Driver, order>(drivers[index], orders[iter->index]));

setDriverWork(index, iter->index, iter->time);

eventElement event = { drivers[index].getTimeOfEndingOrder() , false, index };

Events.insert(event);

cout << iter->time << ": \n\tDriver " << drivers[index].getName() << " start to transit customer\n\tFrom point: " << orders[iter->index].getFrom().x << ", "

<< orders[iter->index].getFrom().y << "\n\tTo: " << orders[iter->index].getDest().x << ", " << orders[iter->index].getDest().y << "\n"

<< "\tTime of complete: " << event.time << "\n" << "\tOrder #" << iter->index + 1 << "\n\n";

}

else {

index = searhBestOnDuty(iter->index);

if (index == drivers.size()) {

orders[iter->index].setIsCompleted(false);

cout << iter->time << ": \n\tOrder #" << iter->index + 1 << " FAILD! No free drivers!\n\n";

}

else {

eventElement event = \*iter;

event.time = drivers[index].getTimeOfEndingOrder();

Events.insert(event);

}

}

}

else {

drivers[iter->index].setOnDuty();

cout << iter->time << ": \n\tDriver " << drivers[iter->index].getName() << " ready to transit new customer!\n"

<< "\tStandby in point: " << drivers[iter->index].getAddr().x << ", " << drivers[iter->index].getAddr().y << "\n\n";

}

Events.erase(iter);

}

}

int TaxiPark::searhBestFree(int orderIndex) {

int index = drivers.size();

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

if (!drivers[i].getIsOnDuty() && timeToGet(i, orderIndex) <= orders[orderIndex].getWaiting())

if(index == size || timeToGet(i, orderIndex) < timeToGet(index, orderIndex)) index = i;

return index;

}

int TaxiPark::searhBestOnDuty(int orderIndex) {

int index = drivers.size();

if (index == drivers.size())

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

if (drivers[i].getIsOnDuty() && drivers[i].getTimeOfEndingOrder() + timeToGet(i, orderIndex) <= orders[orderIndex].getWaiting() + orders[orderIndex].getTime())

if(index == size || drivers[i].getTimeOfEndingOrder() + timeToGet(i, orderIndex) < drivers[index].getTimeOfEndingOrder() + timeToGet(index, orderIndex)) index = i;

return index;

}

void TaxiPark::setDriverWork(int indexOfDriver, int indexOfOrder, short curTime) {

drivers[indexOfDriver].setTimeOfEndingOrder(curTime + timeToComplete(indexOfDriver, indexOfOrder));

drivers[indexOfDriver].setAddr(orders[indexOfOrder].getDest());

drivers[indexOfDriver].setOnDuty();

}

short TaxiPark::timeToGet(int indexOfDriver, int indexOfOrder) {

int deltaX = abs(drivers[indexOfDriver].getAddr().x - orders[indexOfOrder].getFrom().x);

int deltaY = abs(drivers[indexOfDriver].getAddr().y- orders[indexOfOrder].getFrom().y);

return deltaX + deltaY;

}

short TaxiPark::timeToComplete(int indexOfDriver, int indexOfOrder) {

int deltaX = abs(orders[indexOfOrder].getDest().x - orders[indexOfOrder].getFrom().x);

int deltaY = abs(orders[indexOfOrder].getDest().y - orders[indexOfOrder].getFrom().y);

return timeToGet(indexOfDriver, indexOfOrder) + deltaX + deltaY;

}

void TaxiPark::calcCharge(void) {

for (int i = 0, size = drivers.size(); i < size; i++) {

cout << "Driver " << drivers[i].getName() << " charged: ";

int charge = 0;

const vector<Ticket <Driver, order>>& tikets = drivers[i].getTickets();

for (int z = 0; z < tikets.size(); z++)

charge += tikets[z].getCost();

cout << charge << "$\n";

}

}

“CException.h”

#pragma once

class CException {

private:

char\* msg;

public:

CException(char\* message);

~CException();

void show(void);

};

“ CException.cpp”

#include "CException.h"

#include <cstdio>

#include <string.h>

CException::CException(char\* message) {

msg = new char[strlen(message) + 1];

strcpy(msg, message);

}

CException::~CException() {

delete[] msg;

msg = nullptr;

}

void CException::show(void) {

printf("%s\n", msg);

}

“Address.h”

#pragma once

struct address {

int x, y; // Coordinates of point

};

“Driver.h”

#pragma once

#include "Tiket.h"

#include "Address.h"

#include "Order.h"

#include <vector>

using std::vector;

class Driver {

private:

char\* name;

char\* car;

vector <Ticket <Driver, order>> tikets;

address curAddr; // Current address

bool isOnDuty;

short timeOfEndingOrder;

public:

Driver(char\* name, char\* car);

Driver(const Driver& obj);

~Driver(void);

void addTicket(const Ticket<Driver, order>&& ticket);

const vector <Ticket<Driver, order>>& getTickets(void);

bool getIsOnDuty(void);

void setOnDuty(void);

void setAddr(address);

address getAddr(void);

short getTimeOfEndingOrder(void);

void setTimeOfEndingOrder(short time);

const char\* getName(void);

};

“Driver.cpp”

#include "Driver.h"

#include "CException.h"

#include <string.h>

Driver::Driver(char\* name, char\* car) {

if (strlen(name) == 0) throw CException("Empty name of driver");

if(strlen(car) == 0) throw CException("Empty car name for driver");

this->name = new char[strlen(name) + 1];

strcpy(this->name, name);

this->car = new char[strlen(car) + 1];

strcpy(this->car, car);

isOnDuty = false;

curAddr.x = curAddr.y = 0;

timeOfEndingOrder = 0;

}

Driver::Driver(const Driver& obj) {

this->name = new char[strlen(obj.name) + 1];

strcpy(this->name, obj.name);

this->car = new char[strlen(obj.car) + 1];

strcpy(this->car, obj.car);

this->curAddr = obj.curAddr;

isOnDuty = false;

this->timeOfEndingOrder = 0;

}

Driver::~Driver(void) {

delete[] name;

name = nullptr;

delete[] car;

car = nullptr;

}

void Driver::addTicket(const Ticket<Driver, order>&& ticket) {

tikets.push\_back(ticket);

}

const vector <Ticket<Driver, order>>& Driver::getTickets(void) {

return tikets;

}

bool Driver::getIsOnDuty(void) {

return isOnDuty;

}

void Driver::setOnDuty(void) {

isOnDuty = !isOnDuty;

}

void Driver::setAddr(address addr) {

this->curAddr = addr;

}

short Driver::getTimeOfEndingOrder(void) {

return timeOfEndingOrder;

}

void Driver::setTimeOfEndingOrder(short time) {

timeOfEndingOrder = time;

}

address Driver::getAddr(void) {

return curAddr;

}

const char\* Driver::getName(void) {

return name;

}

“Order.h”

#pragma once

#include "Address.h"

#define phoneSize 13

class order {

private:

short time; // Time of order in format time = hours \* 60 + minutes

address destenetion; // Point of destenetion

char phone[phoneSize]; // Phone number in international format

address startPoint; // Position of client

short waiting; // Time of client waiting

bool isComplered; // Is completed oreder

bool cheakPhone(char\* phone); // Cheacking phone format

bool cheakTime(char\* time); // Cheaking time

public:

bool operator< (const order& obj) const;

order(char\* timeOfOrder, address dest, address from, char\* phone, short waiting);

bool getIsCompleted(void);

void setIsCompleted(bool value);

int getCost(void) const;

short getTime(void) const;

short getWaiting(void) const;

address getDest(void) const;

address getFrom(void) const;

};

“order.cpp”

#include "Order.h"

#include "CException.h"

#include <string.h>

#include <cstdlib>

#include <ctype.h>

#define timeLenth 5

#define coefOfCost 5

order::order(char\* timeOfOrder, address dest, address from, char\* phone, short waiting) {

if (!cheakTime(timeOfOrder)) throw CException("Wrong time input format");

else time = atoi(timeOfOrder) \* 60 + atoi(timeOfOrder + 3);

destenetion = dest;

startPoint = from;

if(!cheakPhone(phone)) throw CException("Wrong phone input format");

else strcpy(this->phone, phone);

this->waiting = waiting;

}

bool order::getIsCompleted(void) {

return isComplered;

}

void order::setIsCompleted(bool value) {

isComplered = value;

}

bool order::cheakPhone(char\* phone) {

if (strlen(phone) != phoneSize || \*phone != '+') return false;

for (int i = 1; i < phoneSize; i++) if (!isdigit(phone[i])) return false;

return true;

}

int order::getCost(void) const {

return (abs(startPoint.x - destenetion.x) + abs(startPoint.y - destenetion.y)) \* coefOfCost;

}

bool order::cheakTime(char\* timeOfOrder) {

if (strlen(timeOfOrder) != timeLenth || timeOfOrder[2] != ':') return false;

if (!isdigit(timeOfOrder[0]) || timeOfOrder[0] > '2') return false;

if (!isdigit(timeOfOrder[1]) || !isdigit(timeOfOrder[3]) || !isdigit(timeOfOrder[4])) return false;

return true;

}

short order::getTime(void) const {

return time;

}

short order::getWaiting(void) const {

return waiting;

}

address order::getDest(void) const {

return destenetion;

}

address order::getFrom(void) const {

return startPoint;

}

bool order::operator< (const order& obj) const {

return this->time < obj.time;

}

“Ticket.h”

#pragma once

template <class ... Data>

class Ticket : public Data... {

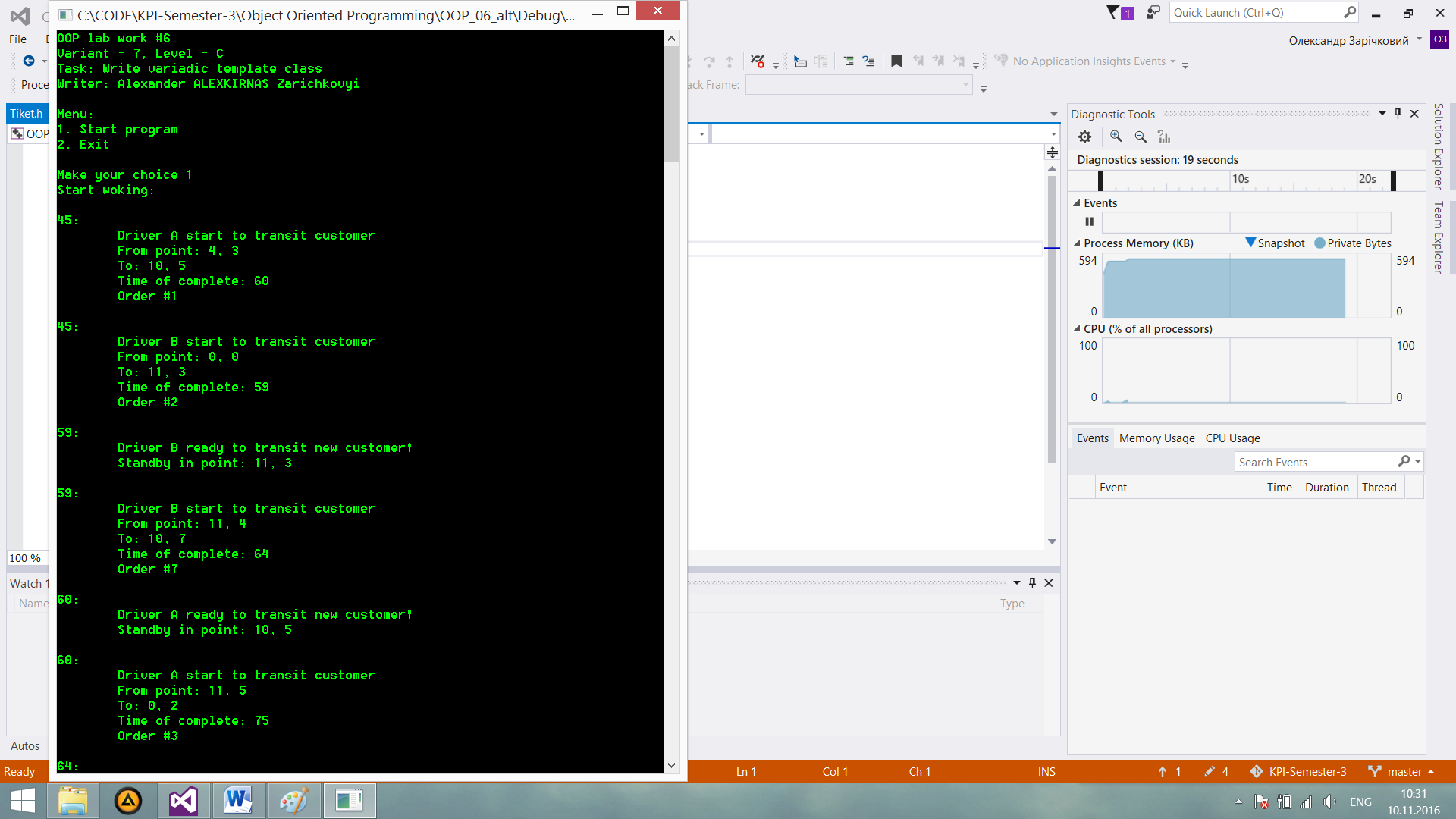
public:

Ticket(const Data& ... data) : Data(data)... {}

};

# Приклади виконання програми

Приклад виконання програми наведений на рисунку 5.1:



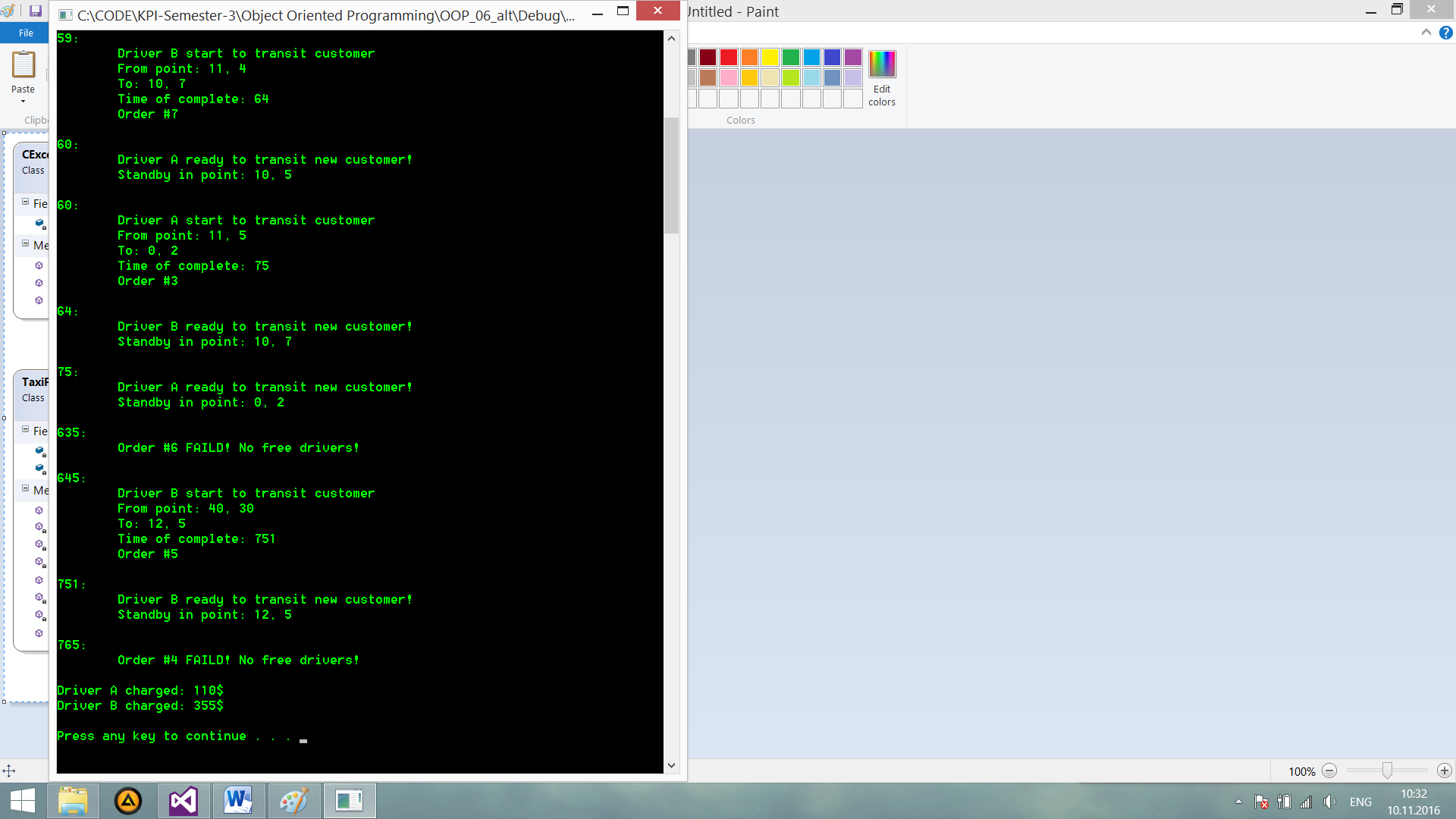


Рисунок 5.1 – Приклад виконання програми

# Висновок

Отже, дана програма ємулює роботу таксопарку та створює множну квитанцій, які є шаблонним класом зі зміною кількістю параметрів. Добавлено обробники виключних ситуацій у всіх потенційно небезпечних місяцях програми. Оскільки дана програма реалізована за модульним принципом і кожен модуль програми створювався й налагоджувався автономно, то дана програма буде видавати очікуваний результат на всіх наборах вхідних даних.