МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

автоматизованих систем обробки інформації та управління

**КУРСОВА РОБОТА**

З дисципліни «Структури даних та алгоритми»

на тему: Таксопарк. Облік робочого часу та бухгалтерія

Студента 1-го курсу ІС-93 групи

Саміленка Олександра Максимовича

Спеціальності 126 «Інформаційні системи та технології»

Керівник ст. викладач кафедри АСОІУ

Проскура Світлана Леонідівна

Кількість балів:

Національна оцінка

|  |  |  |  |
| --- | --- | --- | --- |
| Члени комісії |  |  |  |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |
|  |  |  |  |
|  | (підпис) |  | (вчене звання, науковий ступінь, прізвище та ініціали) |

Київ – 2020 рік

**Національний технічний університет України «КПІ ім. Сікорського»**

**Кафедра автоматизованих систем обробки інформації і управління**

ДисциплінаСтруктури даних та алгоритми

**Спеціальність 126 «Інформаційні системи та технології»**

Курс 1 Група ІС-93 Семестр 1

**ЗАВДАННЯ**

|  |
| --- |
| на курсову роботу студента |

Саміленка Олександра Максимовича

|  |
| --- |
| Таксопарк. Облік робочого часу та бухгалтерія. |
|  |
|  |

1. Тема проекту

|  |
| --- |
| 1. червня 2020 року. |

1. Строк здачі студентом закінченого проекту(роботи):

|  |
| --- |
| Робочий консольний застосунок |

1. Вихідні дані до проекту(роботи):

|  |
| --- |
| для ведення документації та створення невеликої бази даних всередині компанії. |
|  |
|  |

1. Зміст розрахунково-пояснювальної записки:

|  |
| --- |
| Що це за галузь? Які методи я застосував для реалізації додатку? |

|  |
| --- |
| Яким чином відбувалося тестування програми? |
|  |
|  |

1. Перелік графічного матеріалу(з точним зазначенням обов’язкових

креслень):

|  |
| --- |
| Скріншоти щодо роботи програми, діаграма класів та use case діаграма. |
|  |
|  |
|  |

|  |
| --- |
| 1. березня 2020 року. |

1. Дата видачі завдання

**Зміст**

**Зміст**…………………………………………………………………………………….….4

**Вступ**…………………………………………………………………………………….…5

**РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ**………………………………….…..6

* 1. Таксопарк. Внутрішня структура………………………………………6
  2. Об’єктно-орієнтований аналіз предметної області……………………7
  3. Функціональні вимоги до програми……………………………………8
  4. Use Case діаграма………………………………………………………..9

**РОЗДІЛ 2. ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**…………….…………….10

* 1. Опис логічної структури програми…………………….……………...10

### Стандартні методи…………………….……………….10

* + 1. Користувацькі методи…………………………………12
  1. Діаграм класів програмного забезпечення ………………………..…14

**РОЗДІЛ 3. ІНСТРУКЦІЯ КОРИСТУВАЧА**……………………….………………..15

3.1 Робота з програмою………………………………………….………………..15

3.2 Системні вимоги………………………………………………………………17

**РОЗДІЛ 4. ТЕСТУВАННЯ РОБОТИ ПРОГРАМИ**………………………………..18

4.1 Опис методики тестування програмної системи……………….…………...18

4.2 План тестування програмного забезпечення………………………………..18

4.3 Тестування програмного забезпечення………………………….…………..19

**ВИСНОВКИ**……………………………………………………………….…………….22

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**…………………………………………….23

**ДОДАТКИ** ……………………………………………....................................................24

**ВСТУП**

**Призначення програмного забезпечення.**

Призначенням є зручне регулювання, редагування та отримання інформації про працівників, пасажирів та поїздки у даному таксопарку.

**Мета дослідження –** пришвидшення замовлення таксі, покращення взаємодії менеджера з клієнтами та водіями та удосконалення збереження та обробка даних.

**Об’єкт дослідження:** таксопарк.

**Методи дослідження.**

Для вирішення мети і завдань дослідження застосовувалися:

* сучасні комп’ютерні технології;
* аналіз та узагальнення наукової літератури з проблеми;
* статистичний;
* порівняльний.

**РОЗДІЛ 1**. **Аналіз предметної області**

* 1. **Таксопарк. Внутрішня структура**

Що ж таке таксопарк? Це така велика структура, яка охоплює економічну та туристичну сфери. Її суть полягає у тому, що є деяка компанія зі своєю бухгалтерією, офісом, водіями та парком автомобілей, і усі ці складові усе роблять для того, щоб перевезти людину з одного місця в інше.

В компанії є працівники, а також користувачі, тобто клієнти, які можуть бути новачками, а можуть вже досвітніми пасажирами даних таксі.

Звісно, щоб якісно та зручно делегувати усім таксопарком повинен бути програмний додаток, у якому б бухгалтери мали б змогу дивитись інформацію про поїздки, працівників та пасажирів.

Ось це і є моєю головною задачею – розробити консольний додаток, у якому ви мали б змогу додавати, видаляти, змінювати якісь дані про поїздки, клієнтів чи водіїв.

* 1. **Об’єктно-орієнтований аналіз предметної області**

Дивлячись на структуру таксопарку ми можемо сказати, що там є деякі об’єкти, з якими ми можемо виконувати деякі дії. Тобто, ми маємо працівників, які мають свою з/п, інформацію про робочий час, тощо, у цій предметній області є також користувачі або пасажири, про яких ми також маємо деяку інформацію, а саме: ПІБ, кількість поїздок нашою компанією тощо, і не будемо забувати про самі поїздки, про які також є певні дані так, як хто їде, скільки їде, яка вартість поїздки тощо.

Таким чином, ми маємо велику одну область – таксопарк, яка у свою чергу має такі об’єкти, як пасажири, водії та поїздки.

**1.3 Функціональні вимоги до програми**

Це все чудово, але все ж таки, як я зможу використовувати цей додаток?

Усе дуже просто. Представимо, що у вас поступило замовлення, якась людина викликала таксі. Ви дзвоните їй для того, щоб уточнити усі дані. Після цього створюєте поїзду з вказанням усієї інформації. Тут виникає два шляхи: перший – ви змогли додати нову поїзду так, як ця людина вже зареєстрована в базі, а другий – вам спочатку треба зареєструвати її, і тільки після цього додавати поїздку.

Другий випадок, це, коли до вас у компанію працевлаштувався новий водій, і ви хочете додати його до бази даних. Ви просто вибираєте у додатку «Додати працівника» і вводите його дані. Все! Усе так просто!

Третій, коли вам потрібно видали когось, на приклад, у вас звільнився працівник і вам потрібно видалити його з бази даних. Ви також заходите в додаток, вибираєте видалення працівника і все, інформації про нього вже не має!

Тобто, який функціонал має бути у цьому додатку?

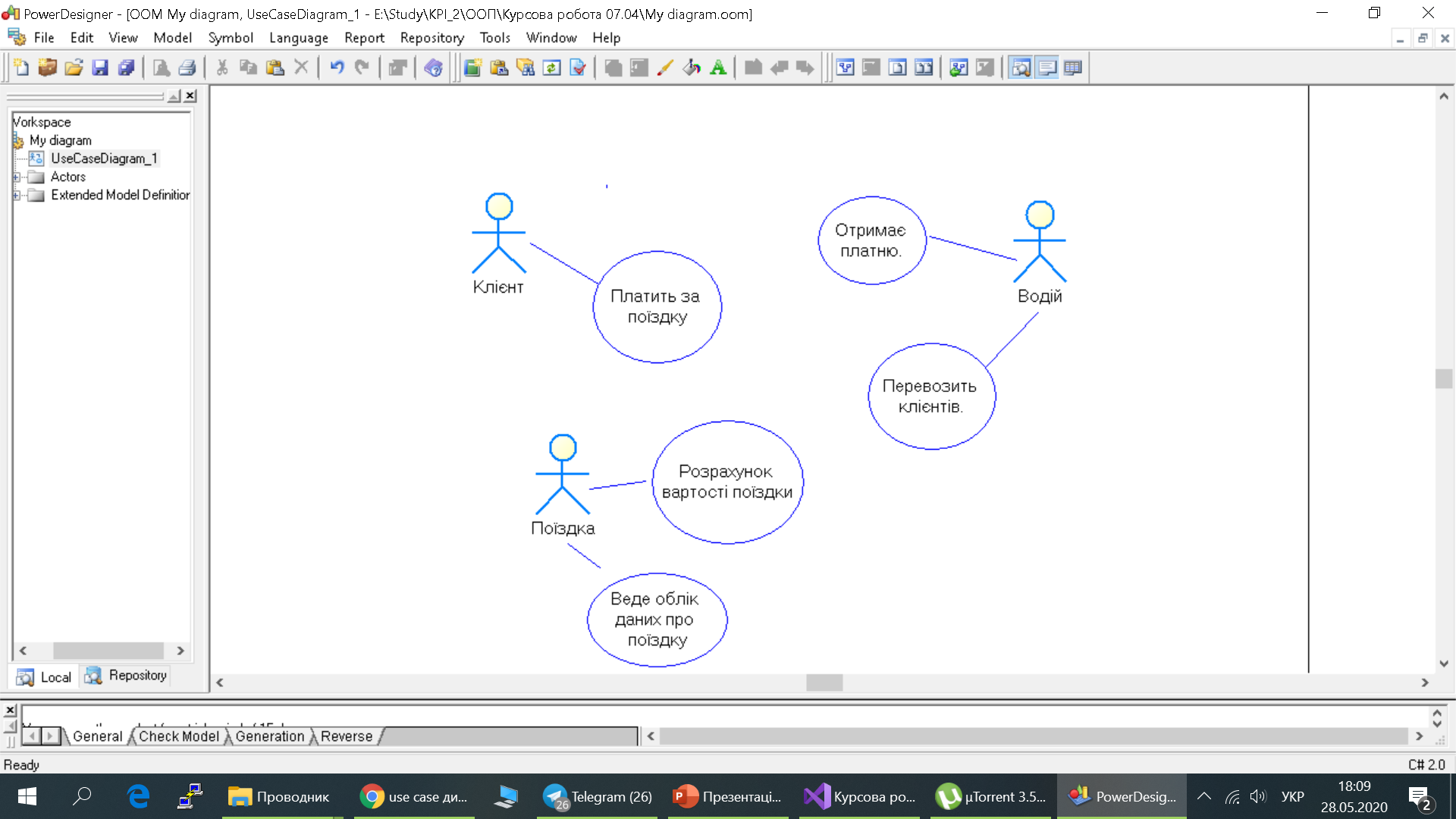
По перше, це додавання працівників, користувачів (клієнтів) та поїздок.

По-друге, це видалення інформації про водіїв, пасажирів та поїздок, які вже закінчилися.

По-третє, це пошук тої чи іншої інформації про окремого працівника, клієнта чи деякої поїздки.

По-четверте, це презентація усієї інформації про водіїв, користувачів чи поїздок, які відбуваються у даний час.

**1.4 Use Case діаграма**



**РОЗДІЛ 2. Опис програмного забезпечення**

**2.1 Опис логічної структури програми**

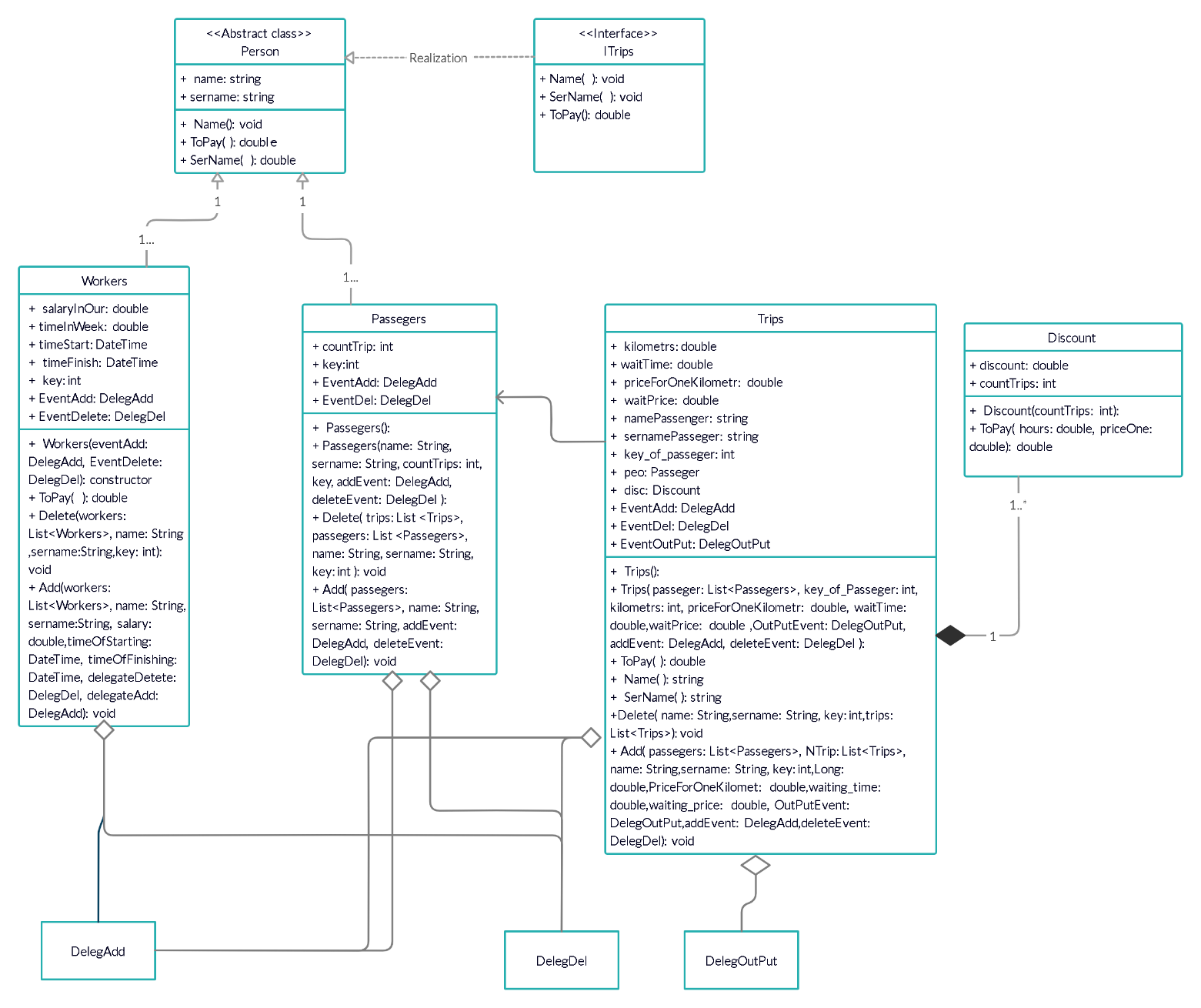
**2.1.1 Стандартні методи**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва методу | Призначення методу | Опис вхідних параметрів | Опис вихідних параметрів |
| 1 | Passegers | Passegers | Конструктор класу | Ім’я, фамілія, кількість поїздок,  Ключ,  Подія видалення, подія додавання. | - |
| 2 | Trips | Trips | Конструктор класу | Об’єкт пасажир, ключ пасажира, кількість кілометрів поїздки,  Ціна за один кілометр, час очікування, ціна за хвилину очікування,  Події додавання, видалення та зміни інформації. | - |
| 3 | Workers | Workers | Конструктор класу | Подія видалення та додавання водія. | - |
| 4 | Discount | Discount | Конструктор класу | Кількість поїздок | - |
| 5 | Discount | ToPay | Знаходження ціни за поїздку | Кількість годин, ціна за одну годину | Ціна поїздки |

**2.1.2 Користувацькі методи**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № п/п | Назва классу | Назва методу | Призначення методу | Опис вхідних параметрів | Опис вихідних параметрів |
| 1 | Passegers | Delete | Видалення даних | Список поїздок, список пасажирів,  Ім’я, фамілія та ключ. | - |
| 2 | Passegers | Add | Додавання даних | Список пасажирів, ім’я, фамілія, подія видалення, подія додавання. | - |
| 3 | Trips | ToPay | Отримання ціни поїздки | - | Ціна поїздки |
| 4 | Trips | Name | Отримання імені | - | Ім’я пасажира |
| 5 | Trips | SerName | Отримані фамілії | - | Фамілія пасажира |
| 6 | Trips | Add | Додавання поїздки | Ім’я, фамілія, ключ, список поїздок, список пасажирів, довжина шляху, ціна за один кілометр, час очікування, ціна за одну хвилину очікування,  подія додавання, видалення та зміни інформації. | - |
| 7 | Trips | Delete | Видалення поїздки | Ім’я, фамілія, ключ, список поїздок | - |
| 8 | Workers | ToPay | Отримання заробітної плати за тиждень | - | Заробітна плата |
| 9 | Workers | Delete | Видалення робітника | Список робітників, ім’я водія та його фамілія, а також ключ. | - |
| 10 | Workers | Add | Додавання робітника | Список робітників, ім’я та фамілія нової людини, заробітня плата за годину, час початку роботи та час закінчення робочого дня,  Подія додавання, подія видалення. | - |

**2.2 Діаграма класів програмного забезпечення**



**РОЗДІЛ 3. Інструкція користувача**

**3.1 Робота з програмою**

Після запуску виконавчого файлу з розширенням \*.exe, відкривається головне вікно програми (Рисунок 2.1).

Щоб вибрати той, чи інший пункт введіть потрібний номер меню. Для цього натисніть Enter.

Після цього ви можете вибрати потрібне меню(Рисунок 2.1).

І вже у меню можете вибрати, що ви хочете зробити(Рисунок 2.2).

Щоб вийти з внутрішнього меню натисніть клавішу 5 та Enter (Рис. 2.3).

Щоб вийти з програми, знаходячись у головному меню, натисніть клавішу 5 та Enter.(Рис 2.4).

Для прикладу ви захотіли додати нового пасажира.

* У головному меню виберіть 1-й пункт(Рис 2.5).
* У внутрішньому меню виберіть пункт «Add new passeger» (Рис 2.6).
* Після введіть Ім’я, фамілію(Рис 2.7).
* Також вам буде запропоновано підписатися на новини щодо додавання цього пасажира та видалення його. Щоб підписатися на додавання уведіть – “True”, не підписуватись на це “False”. Аналогічно для видалення(Рис 2.8).

Якщо ви хочете видалити, то виберіть пункт «Delete passeger» (Рис 2.9).

Вам будуть надаватися поля, куди ви введете дані пасажира, який хочете видалити, при цьому, щоб ви не запам’ятовували усі дані цього пасажира, угорі консолі ви побачите додаткову таблицю, де показані усі пасажири (Рис 2.10).

Щоб додати або видалити поїздку або водія зробіть аналогічні дії тільки для них, проте будьте уважні, коли вводите дані тому, що інформація може бути відмінною від того, яку я подав вище у правилах. Проте, у вас не буде виникати труднощів при роботі з програмою так, як основні аспекти використання консольного застосунку ми вже розібрали.

Важливо!

Коли ви додаєте поїздку та вибраного пасажира не має в базі даних, ви отримаєте помилку. Якщо це станеться – не панікуйте. Перейдіть в меню пасажира та додайте його.

**3.2 Системні вимоги**

|  |  |  |
| --- | --- | --- |
|  | Мінімальні | Рекомендовані |
| Операційна система | Windows® XP/Windows Vista/Windows 7/ Windows 8/Windows 10 (з останніми обновленнями) | Windows 7/ Windows 8/Windows 10  (з останніми обновленнями) |
| Процесор | Intel® Pentium® ІІІ  1.0 GHz або  AMD Athlon™ 1.0 GHz | Intel® Pentium® D або AMD Athlon™ 64 X2 |
| Оперативна пам'ять | 256 MB RAM (для Windows® XP) / 1 GB RAM (для Windows Vista/Windows 7/  Windows 8/Windows 10) | 2 GB RAM |
| Відеоадаптер | Intel GMA 950 з відеопам'яттю об'ємом не менше 64 МБ (або сумісний аналог) | |
| Дисплей | 800х600 | 1024х768 або краще |
| Прилади введення | Клавіатура, комп’ютерна миша | |
| Додаткове програмне забезпечення | Microsoft .Net Framework 4.5.2 або вище | |

**Розділ 4. Тестування роботи програми**

**4.1 Опис методики тестування програмної системи**

Я обрав таку методику тестування, що на вхід по даються дані у не правильному форматі, або значення яких виходить за межі допустимого, тобто, в програму вводить довжина пройденого шляху -100, то програми виведе exception тому, що довжина не може бути від’ємною. І такого роду перевірку на вхідні значення я роблю для кожного параметра, який вводить користувач.

**4.2 План тестування програмного забезпечення**

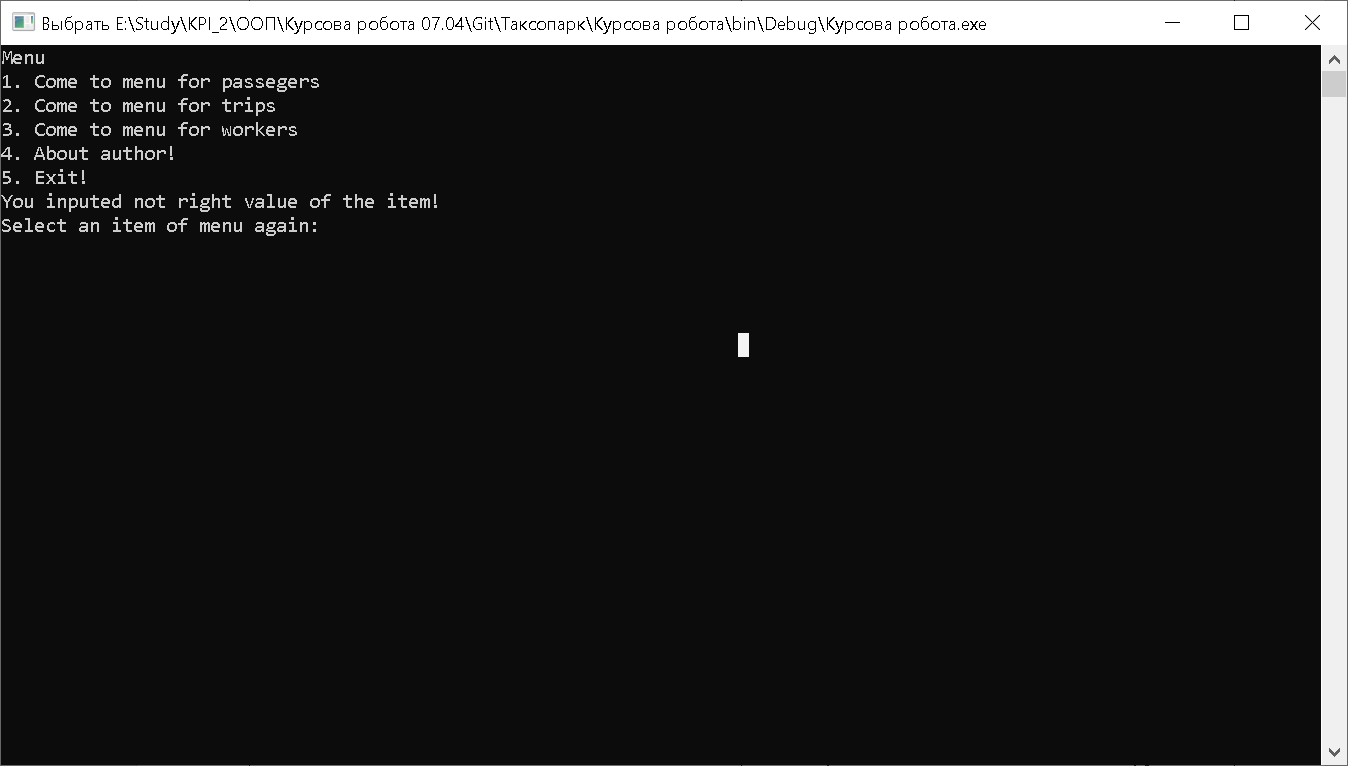
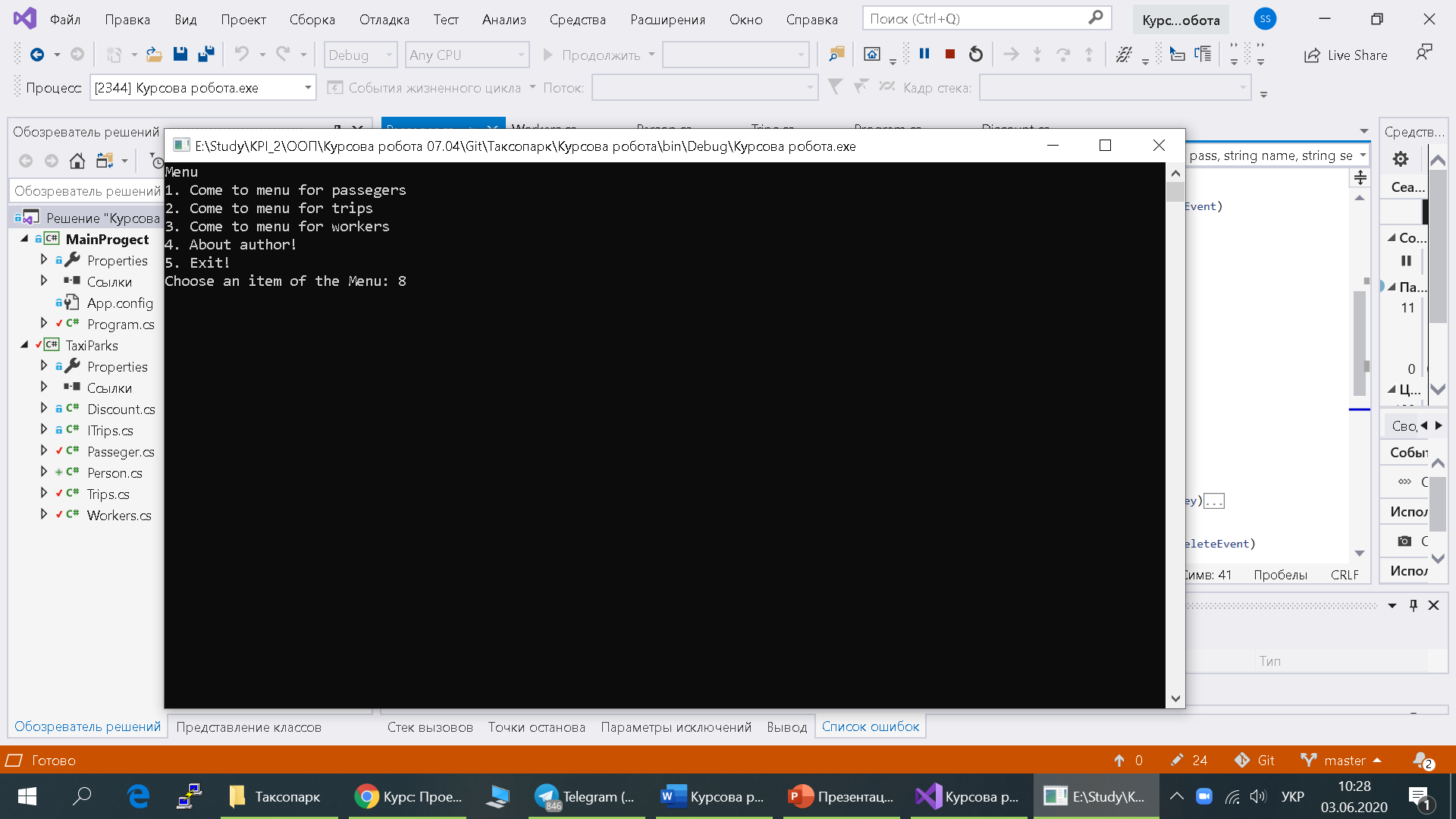
Я обрав декілька методик обробки помилок, а саме: викликання exception та від помилки у консоль.

* 1. Тестування при введенні некоректних символів.
  2. Тестування при введенні замалих значень.
  3. Тестування при введені завеликих значень.
  4. Тестування при введені значень, які є не коректними для роботи з ними у даній частині програмного коду.

**4.3 Тестування програмного забезпечення**

Таблиця 1.1 ‑ Приклад роботи програми при введенні некоректних даних при виборі пункту меню

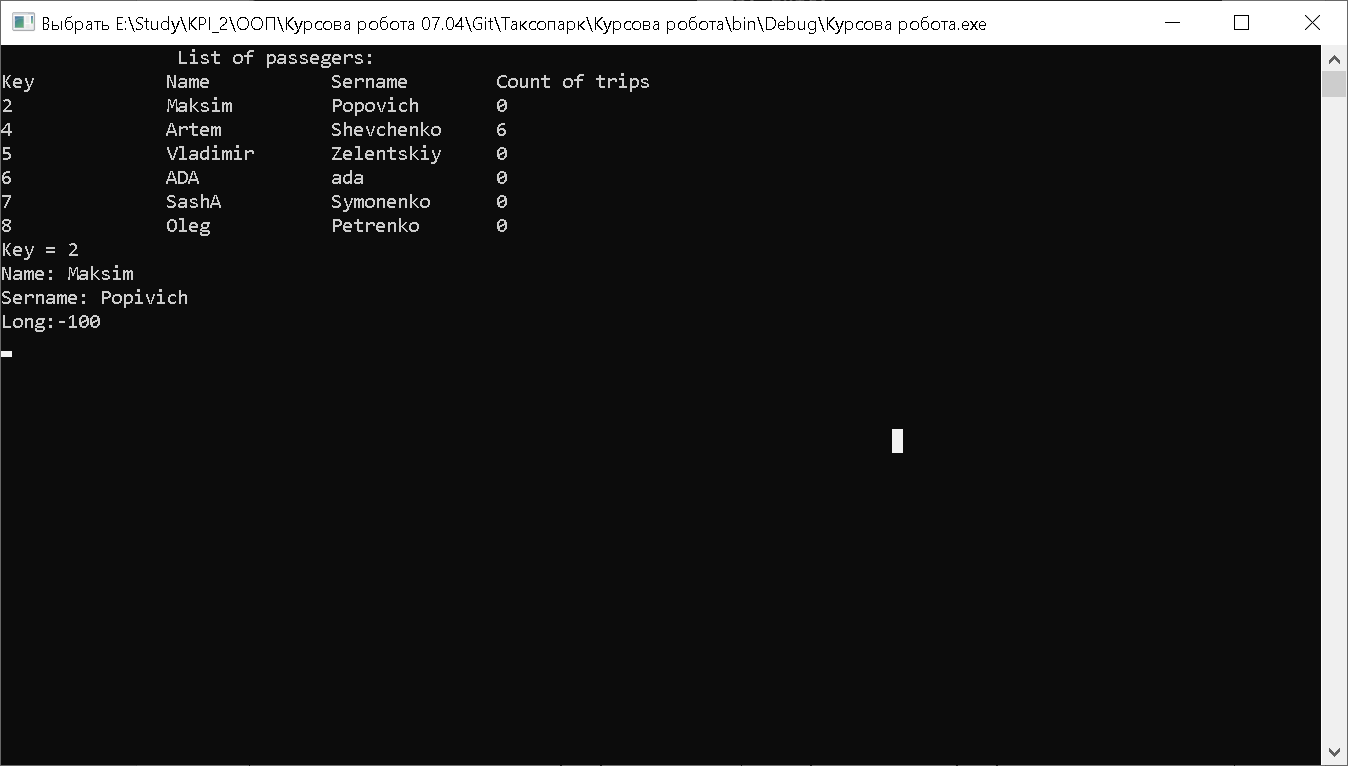
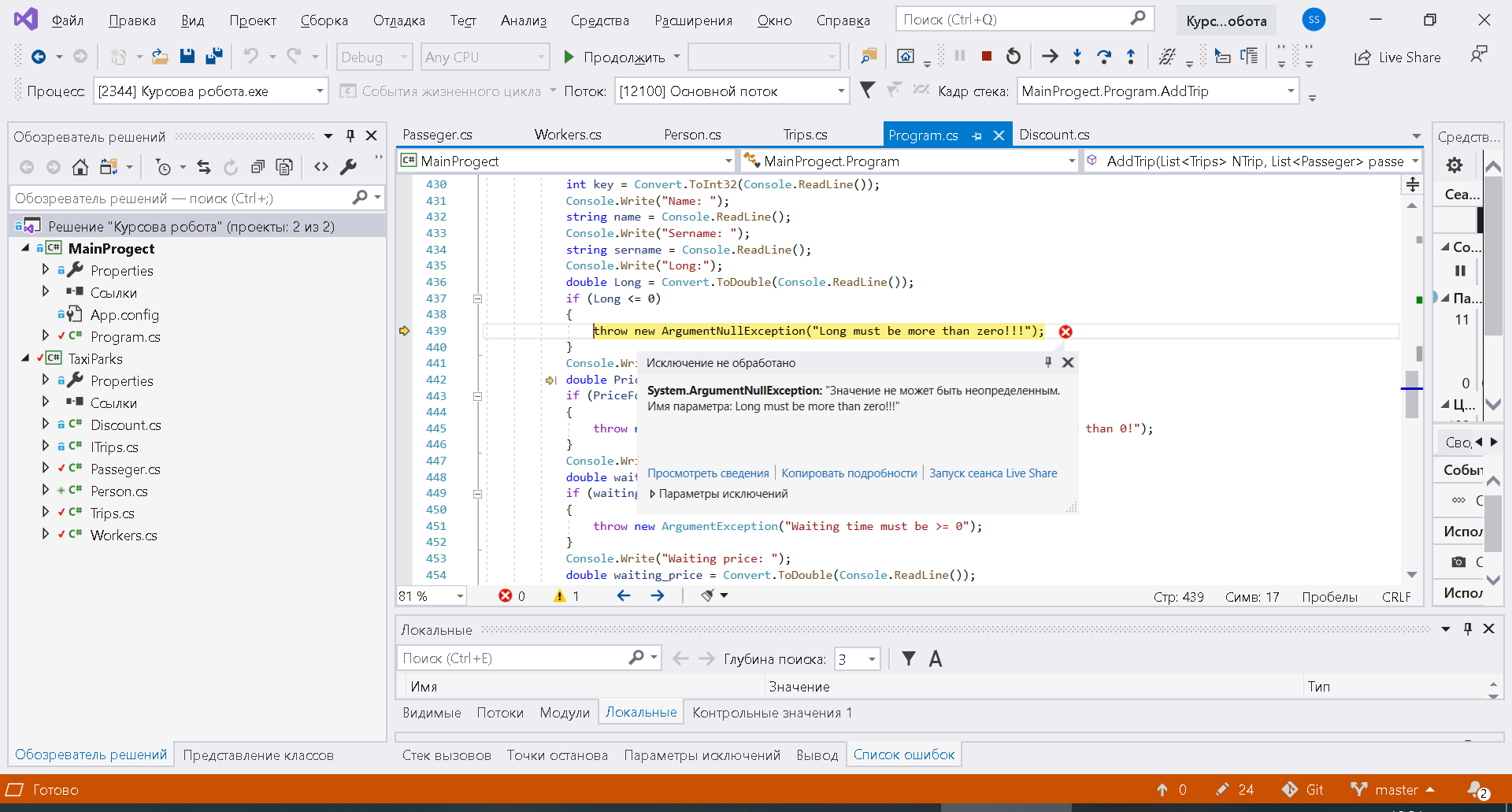
|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення некоректних даних |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | 8 |
| Схема проведення тесту | Перевірка входження у блоці swich |
| Очікуваний результат | Повідомлення про помилку  некоректних даних |
| Стан програми після проведення випробувань | Видано помилку «You inputed not right value of the item!». |



**Рис 1.1 Введення 8. Рис 1.2 Отримання помилки.**

Таблиця 1.2 ‑ Приклад роботи програми при введенні некоректної довжини шляху при додаванні нової поїздки

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення некоректних даних |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | -100 |
| Схема проведення тесту | Перевірка даних за допомогою if |
| Очікуваний результат | Повідомлення про помилку  некоректних даних |
| Стан програми після проведення випробувань | Видано помилку «Long must be more than zero!!!». |

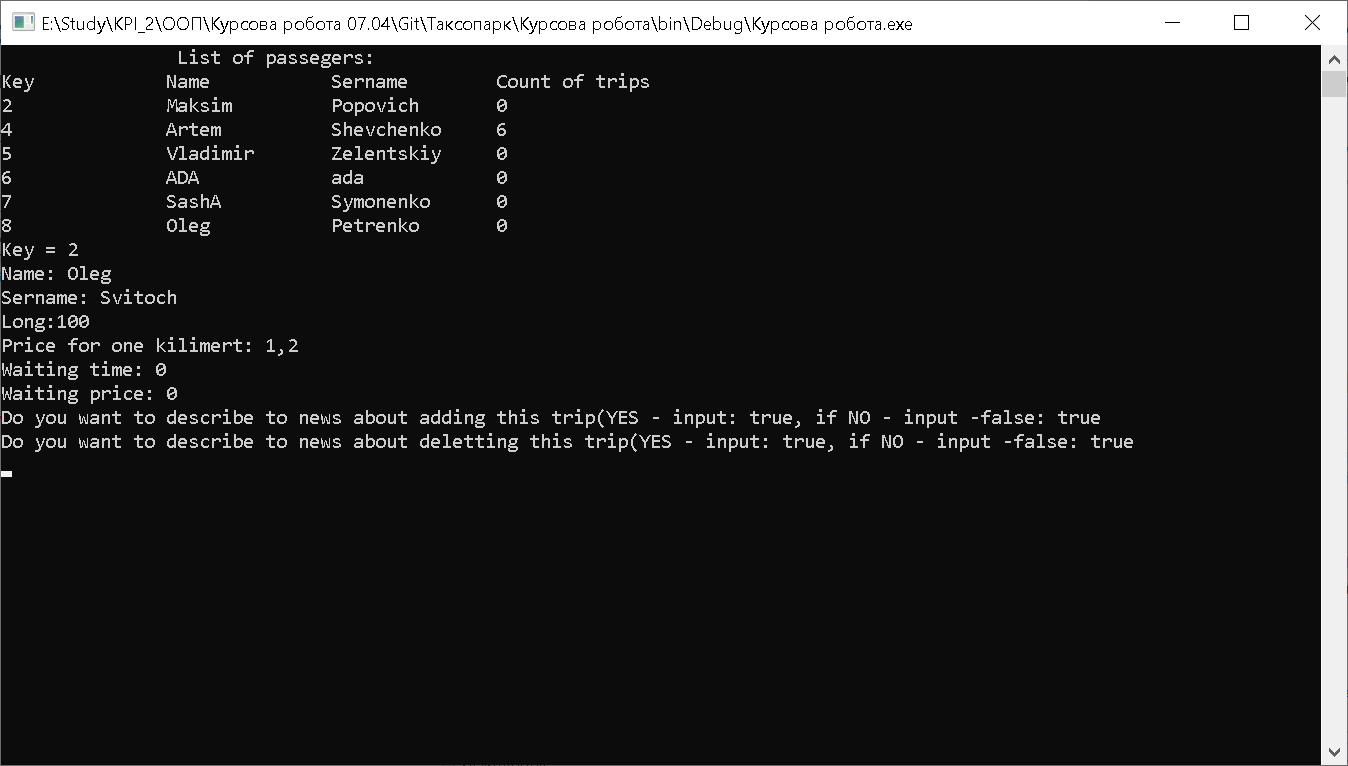
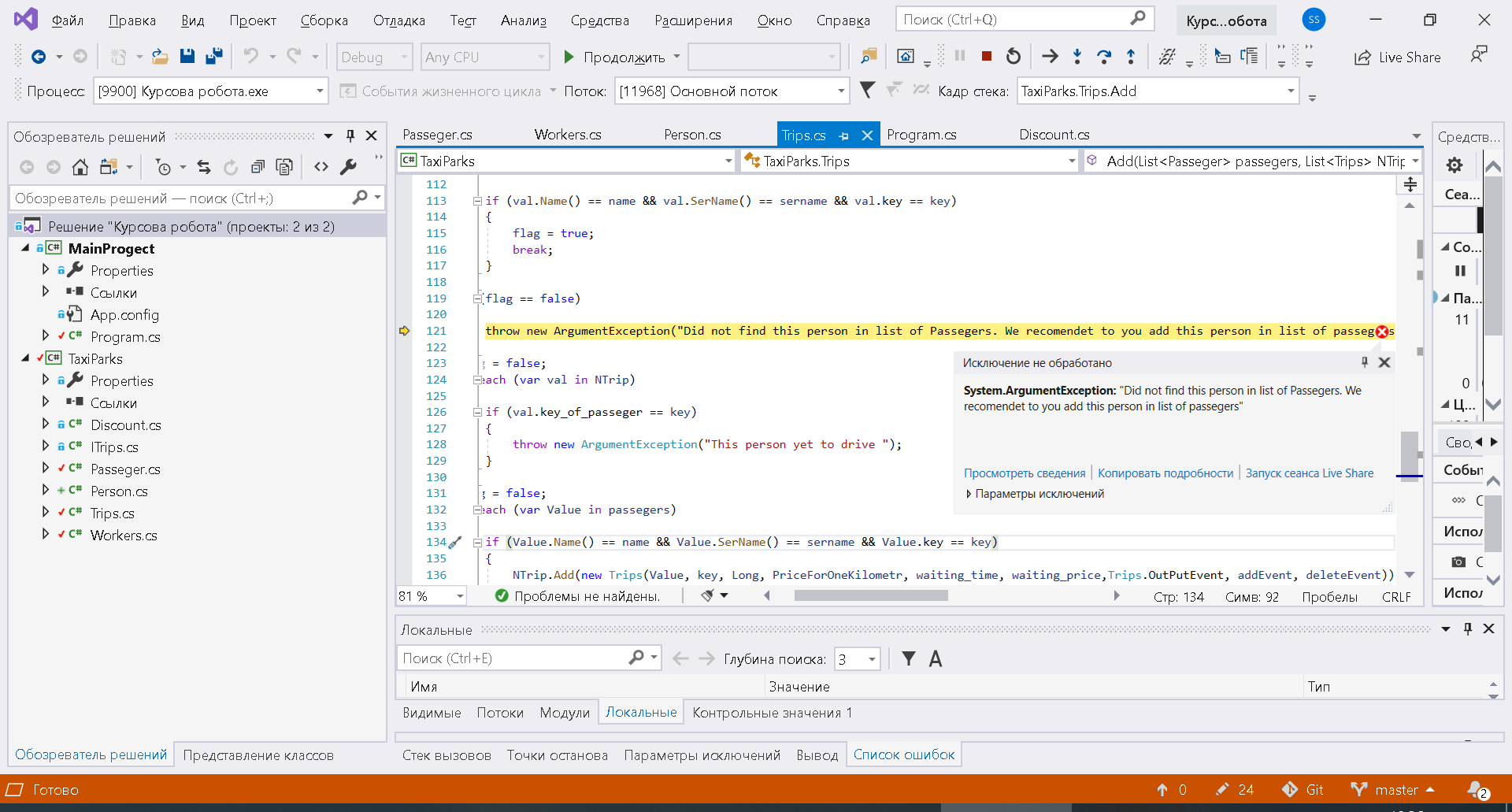


**Рис 1.4 Отримання виключенням.**

**Рис 1.3 Введення некоректного значення довжини.**

Таблиця 1.3 ‑ Приклад роботи програми при введенні некоректних даних (ключа, імені та фамілії пасажира) при додаванні нової поїздки

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення некоректних даних |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | 2 Oleg Svitoch |
| Схема проведення тесту | Перевірка даних за допомогою if |
| Очікуваний результат | Повідомлення про помилку  некоректних даних |
| Стан програми після проведення випробувань | Видано помилку «Did not find this person in list of Passegers. We recomendet to you add this person in list of passegers». |

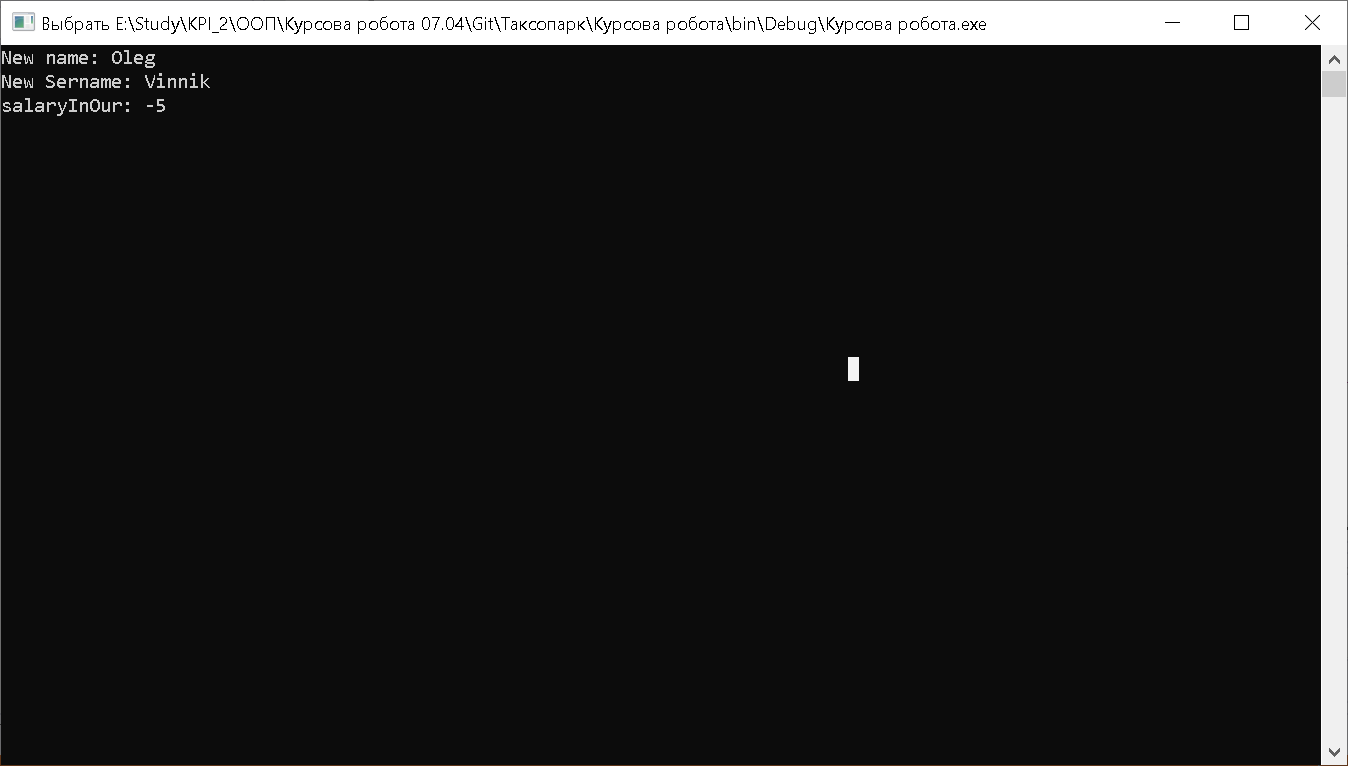
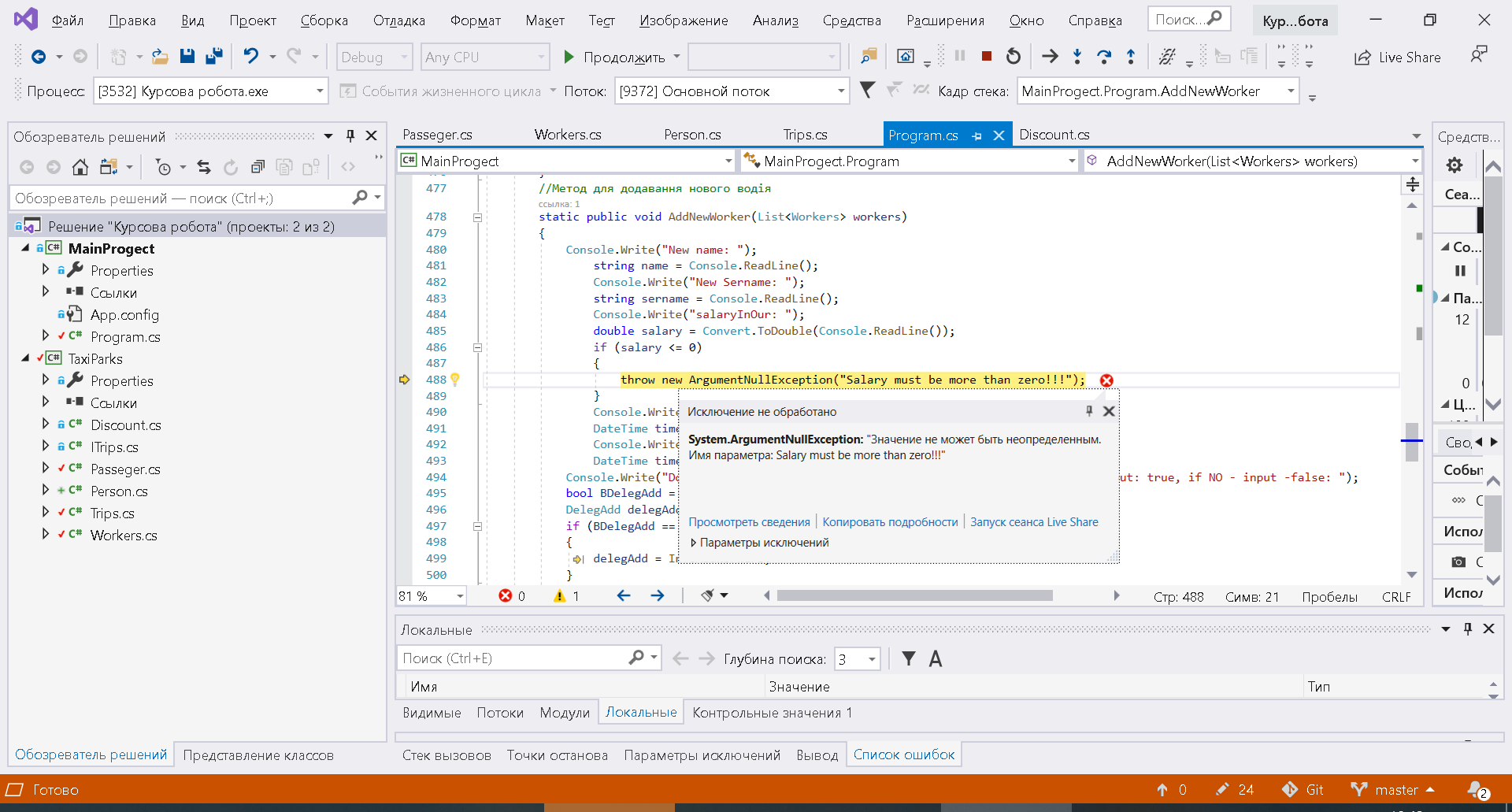


**Рис 1.6 Виведення помилки.**

**Рис 1.5 Введення неіснуючого пасажира.**

Таблиця 1.4 ‑ Приклад роботи програми при введенні некоректних даних (з/п) при додаванні нового водія

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення некоректних даних |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | -5 |
| Схема проведення тесту | Перевірка даних за допомогою if |
| Очікуваний результат | Повідомлення про помилку  некоректних даних |
| Стан програми після проведення випробувань | Видано помилку «Salary must be more than zero!!!». |

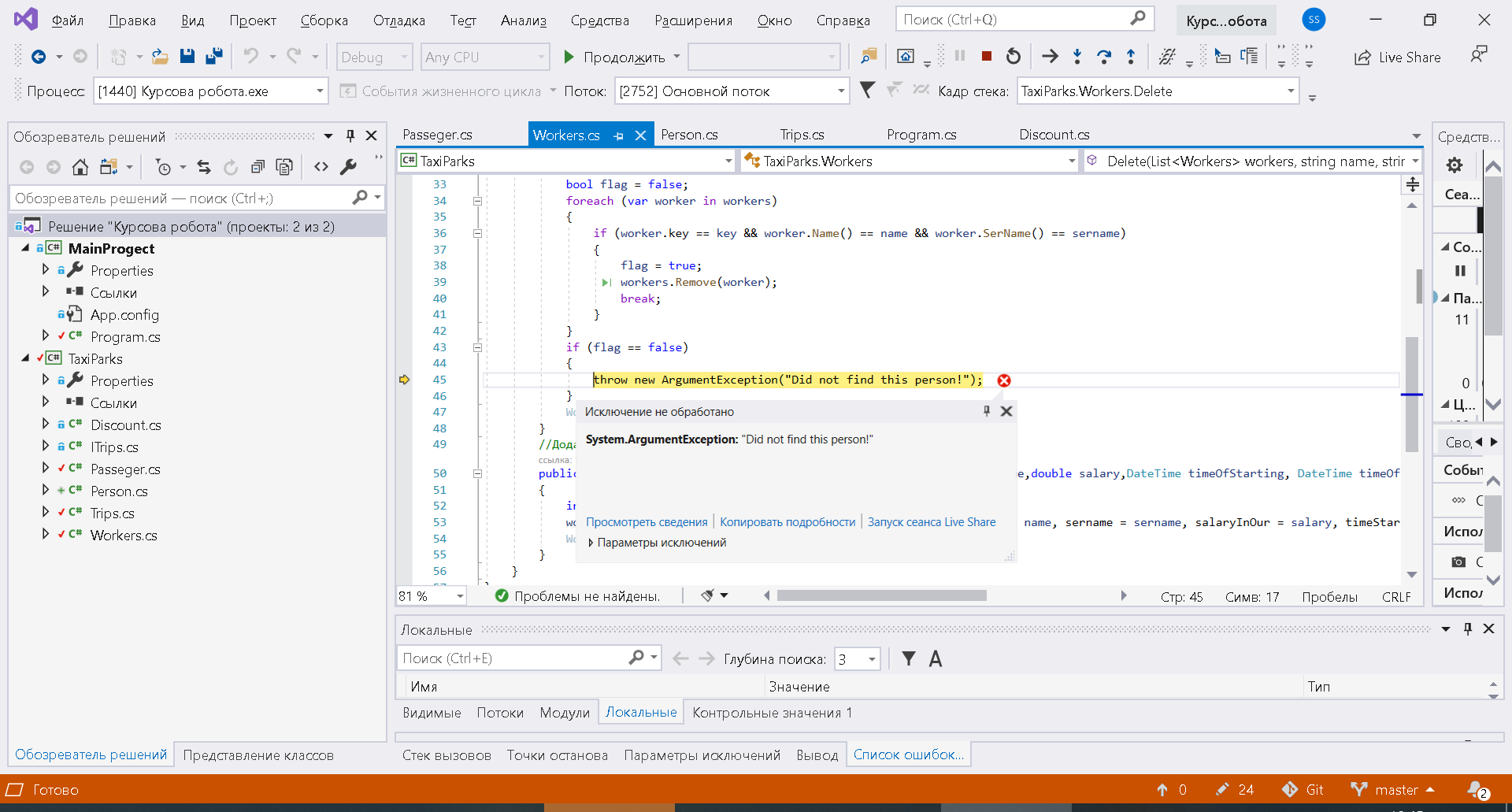
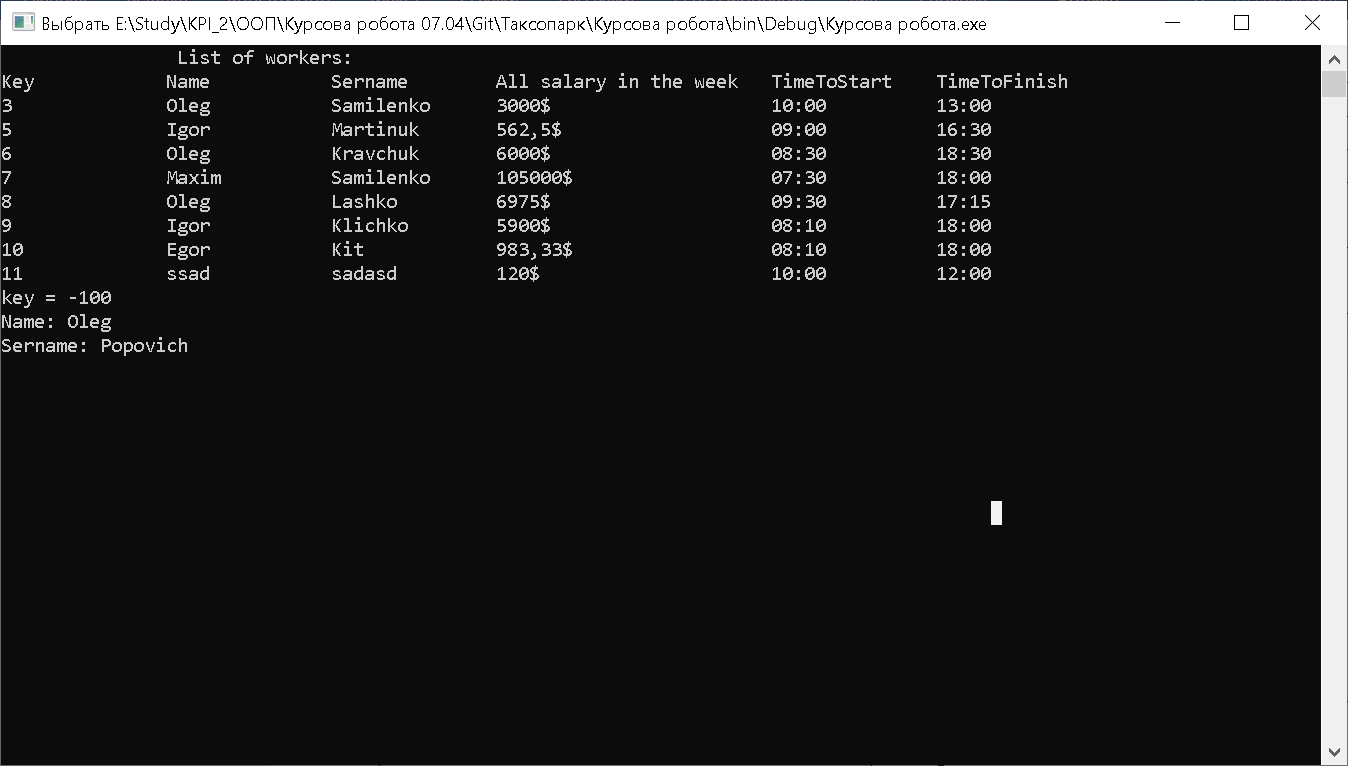


**Рис 1.8 Виведення помилки.**

**Рис 1.7 Введення від’ємного значення заробітної плати водія.**

Таблиця 1.5 ‑ Приклад роботи програми при введенні некоректних даних (ключа, ім'я та фамілії) при видаленні водія

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення некоректних даних |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | -100 Oleg Popovich |
| Схема проведення тесту | Перевірка даних за допомогою if |
| Очікуваний результат | Повідомлення про помилку  некоректних даних |
| Стан програми після проведення випробувань | Видано помилку «Did not find this person!». |

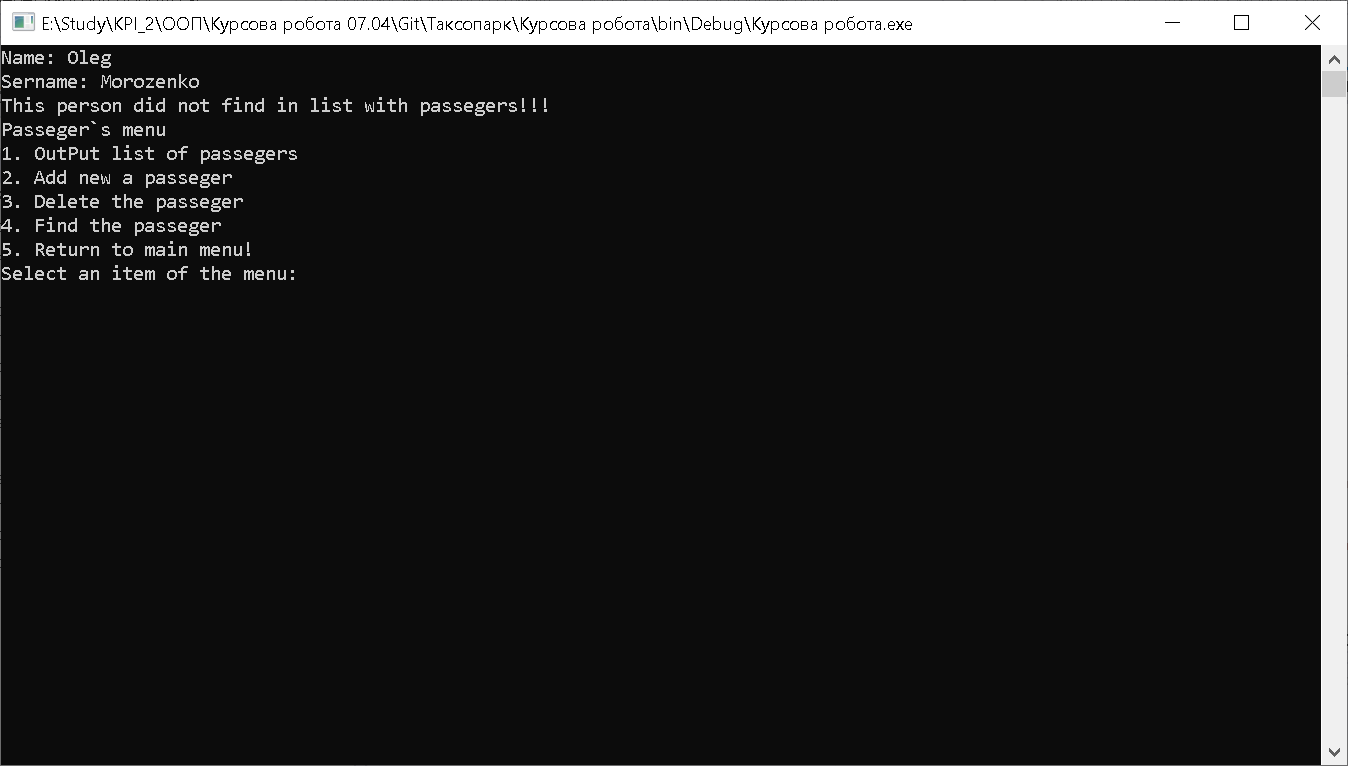


**Рис 1.9 Введення некоректних даних водія.**

**Рис 1.10 Виведення повідомлення про те, що такого водія не було знайдено!**

Таблиця 1.6 ‑ Приклад роботи програми при введенні некоректних даних при пошуку пасажира в базі даних.

|  |  |
| --- | --- |
| Мета тесту | Перевірити можливість введення некоректних даних |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Oleksandr Morozenko |
| Схема проведення тесту | Перевірка даних за допомогою if |
| Очікуваний результат | Повідомлення про помилку  некоректних даних |
| Стан програми після проведення випробувань | Видано помилку «This person did not find in list with passegers!!!». |



**Рис 1.11 Вікно інформування про те, що такого пасажира не було знайдено у базі даних.**

**Висновки**

У своїй науково-дослідницькій роботі я реалізував консольний додаток, який допоможе регулювати таксопарком зручно та швидко.

Проте, маю сказати, що додаток не є доскональним так, як він не працює в онлайн, не має можливості редагування деякої інформації, а тільки видалення та додавання її, і основне це те, що це консольне ПО, яке мало чим схоже на пропозиції інших компаній на ринку додатків, які працюють, як мінімум, на базі

Window forms.

Також я не можу обійти той факт, що це мій перший власноруч створений такої величини проект, і він не ідеальний, навіть, якщо закрити очі на of-line доступ та суто консольний вид. Чому? Тому, що я не має ще того досвіду роботи з додатками, які мають досконально працювати з будь-яким користувачем, чи це буде людина, яка обізнана і вона прочитала «Інструкцію користувача», чи це буде, як це частіше буває, особа, яка не має часу на скучне читання чогось і намагатиметься одразу почати користуватися моїм додатком. То для другої, ПО може здатися складним та незручним, проте все залежить від самої людини, її уміння аналізувати те, що вона бачить перед собою на екрані.

А також, не можу обійти стороною ще й те, що хоча і маю непогані знання з англійської мови, проте деякі користувачі, більш обізнана ніж я, скоріш за все будуть критикувати мою програму за неякісний, як граматично, так і синтаксично виклад помилок, написання пунктів меню тощо. Особливо, моє не ідеальне знання англійської може вплинути на кількість користувачів, рідною мовою яких є англійська. Все ж таки, я покращую свої знання, і наступні мої проекти будь набагато більшої якості ніж це.

**Список використаних джерел**

1. C# 4.0: полное руководство.: Пер. с англ. — М.: ООО "И.Д. Вильямс", 2011. — 1056 с.: ил. — Парал. тит. англ.
2. Copyright © 1998 by McGraw-Hill Companies. All rights reserved. Manufactured in the United States of America. Except as permitted

under the United States Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means,

or stored in a database or retrieval system, without the prior written permission of the publisher.

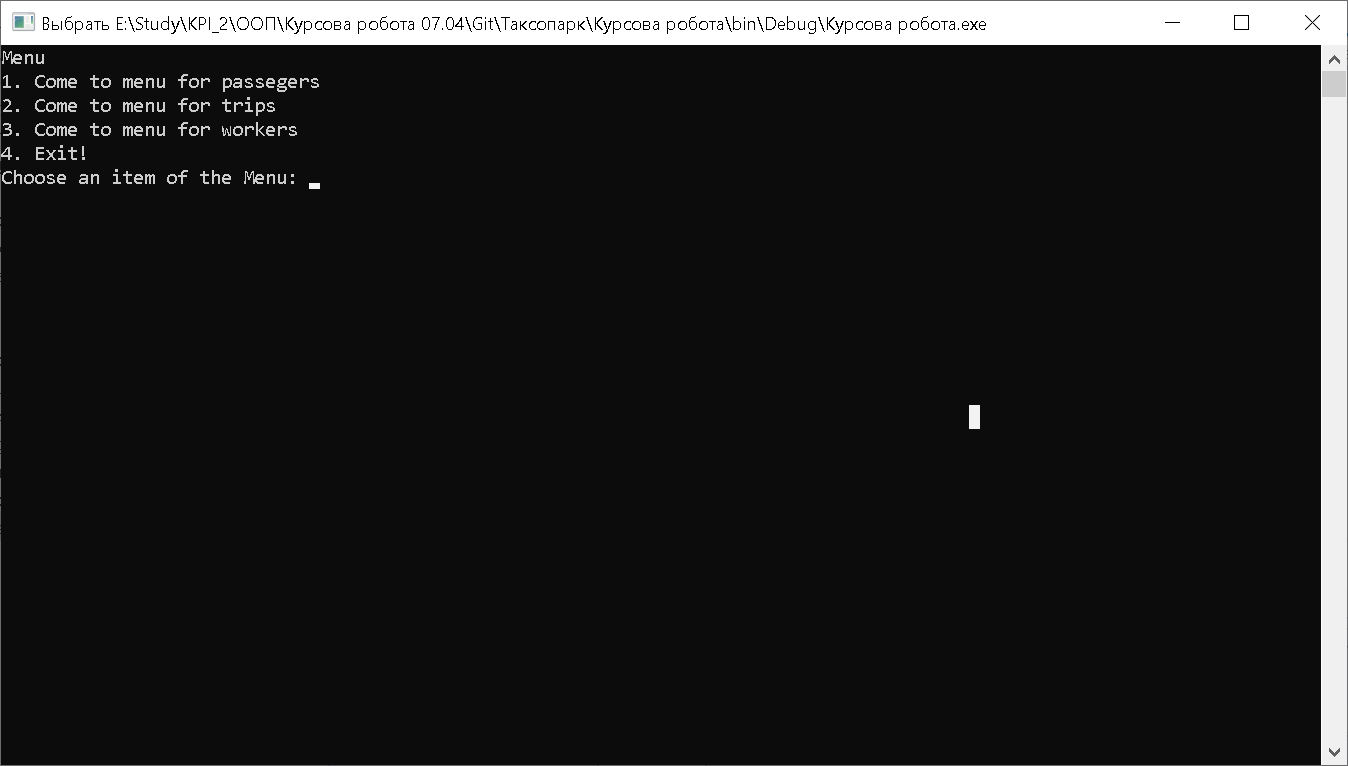
1. C# 5.0 Карманный справочник: Пер. с англ. – М. : ООО «И.Д.Вильямс»,2013. – 288 с.:ил. – Парал. тит. англ.

ISBN 978-5-8459-1820-8(рус.).

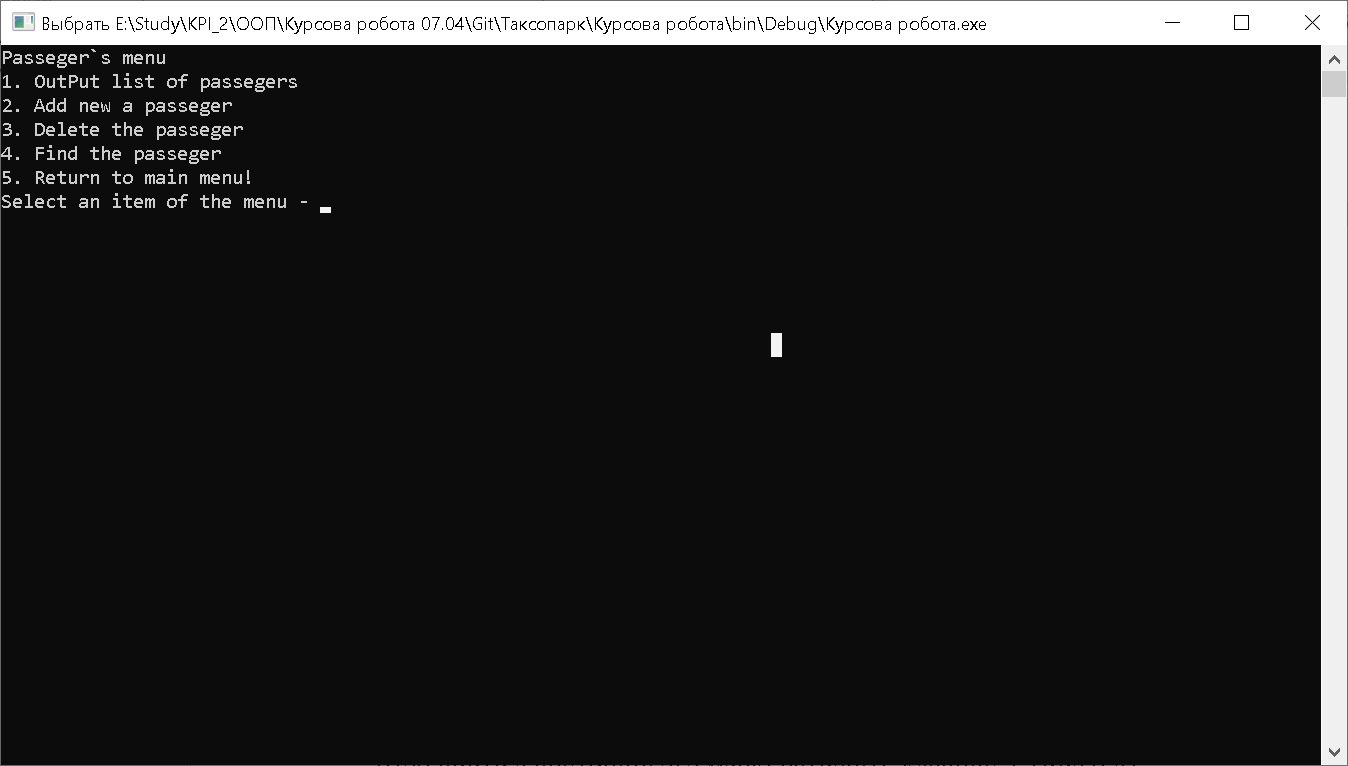
1. <https://metanit.com/sharp/tutorial/>
2. Язык С++. Основы программирования. Издание второе, исправленное и дополненное / Ю. В. Марапулец. — Петропавловск-Камчатский: КамГУ им. Витуса Беринга, 2019. — 158 с.
3. СПб.: Питер; Киев: BHV, 2003. — 512 с. — ISBN 966-552-121-7; ISBN 5-94723-167-0.
4. Pro Git. Scott Chacon, Ben Straub. Version 2.1.21, 2019-07-28.

**Додатки**

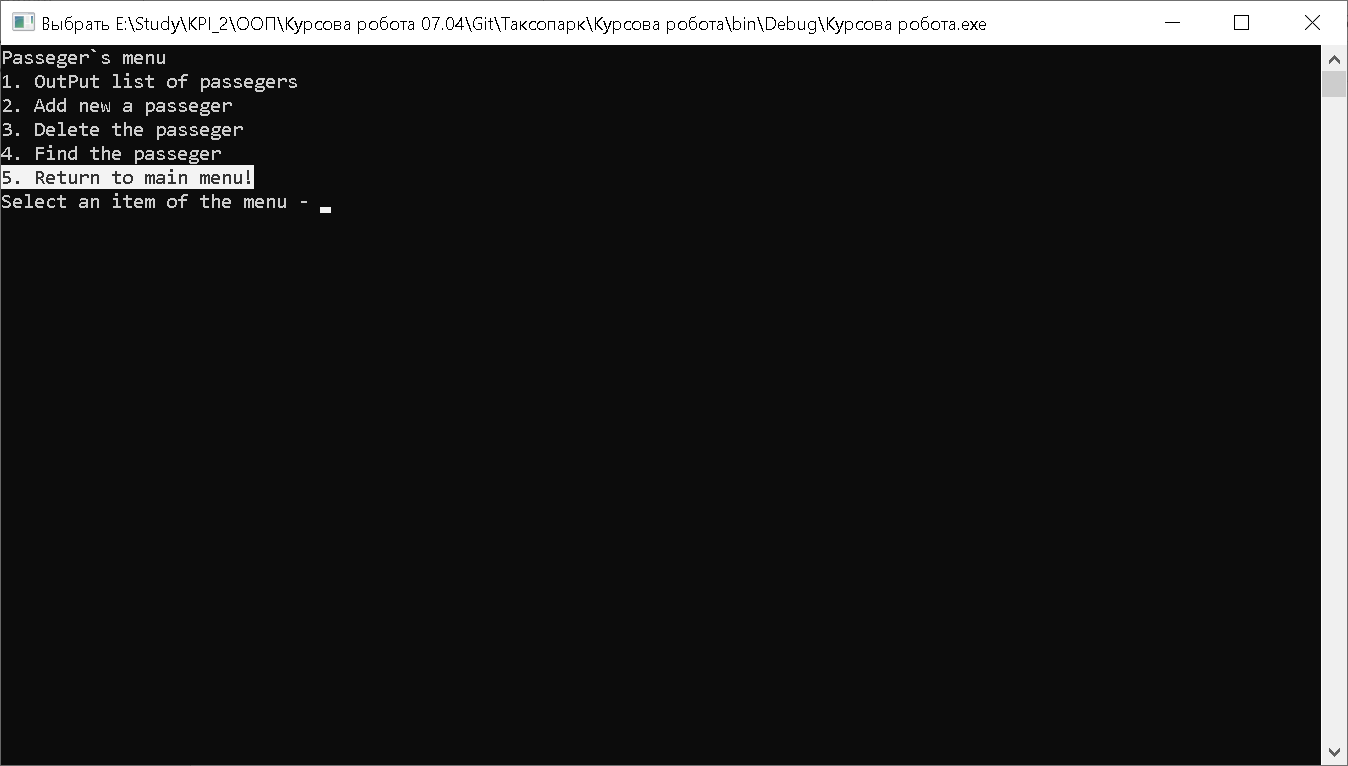
**Рис 2.1 Вікно програми**



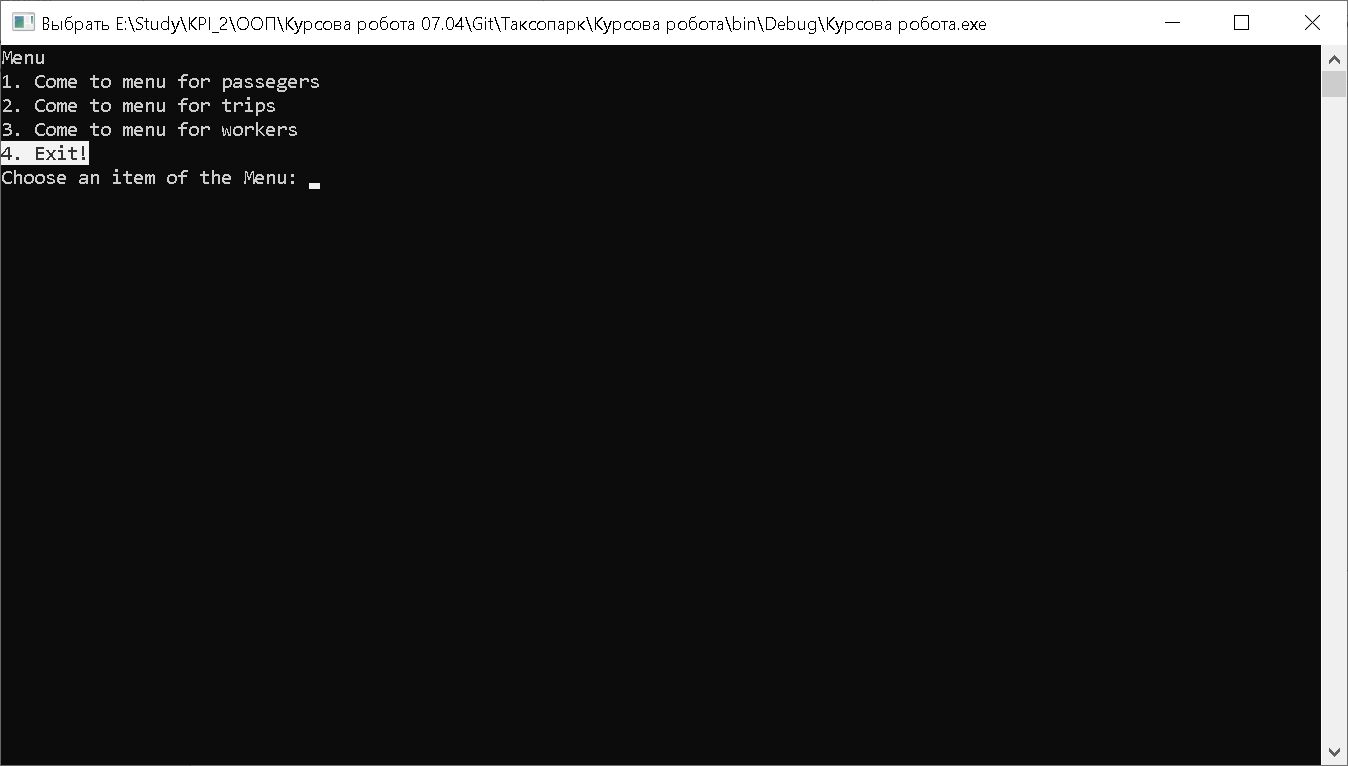
**Рис 2.2 Вікно під меню(Пасажири)**



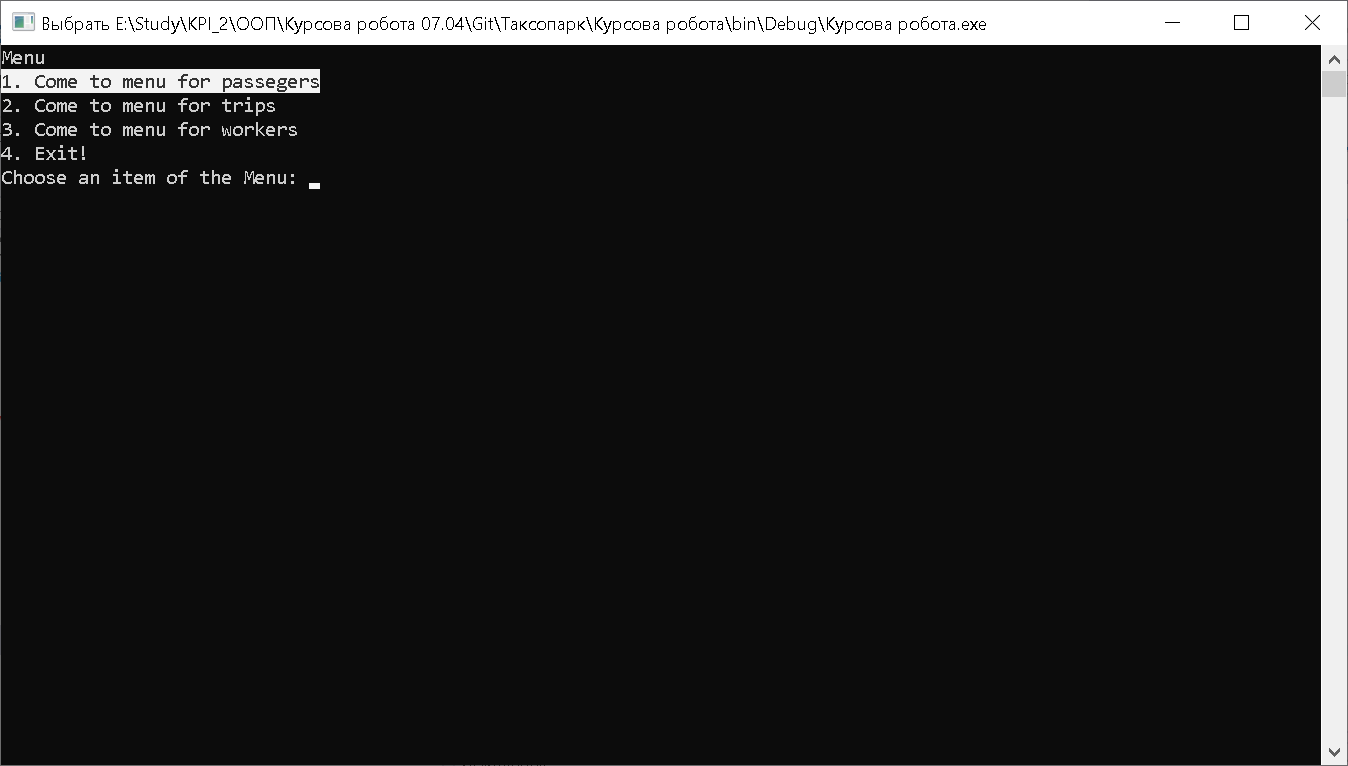
**Рис 2.3**



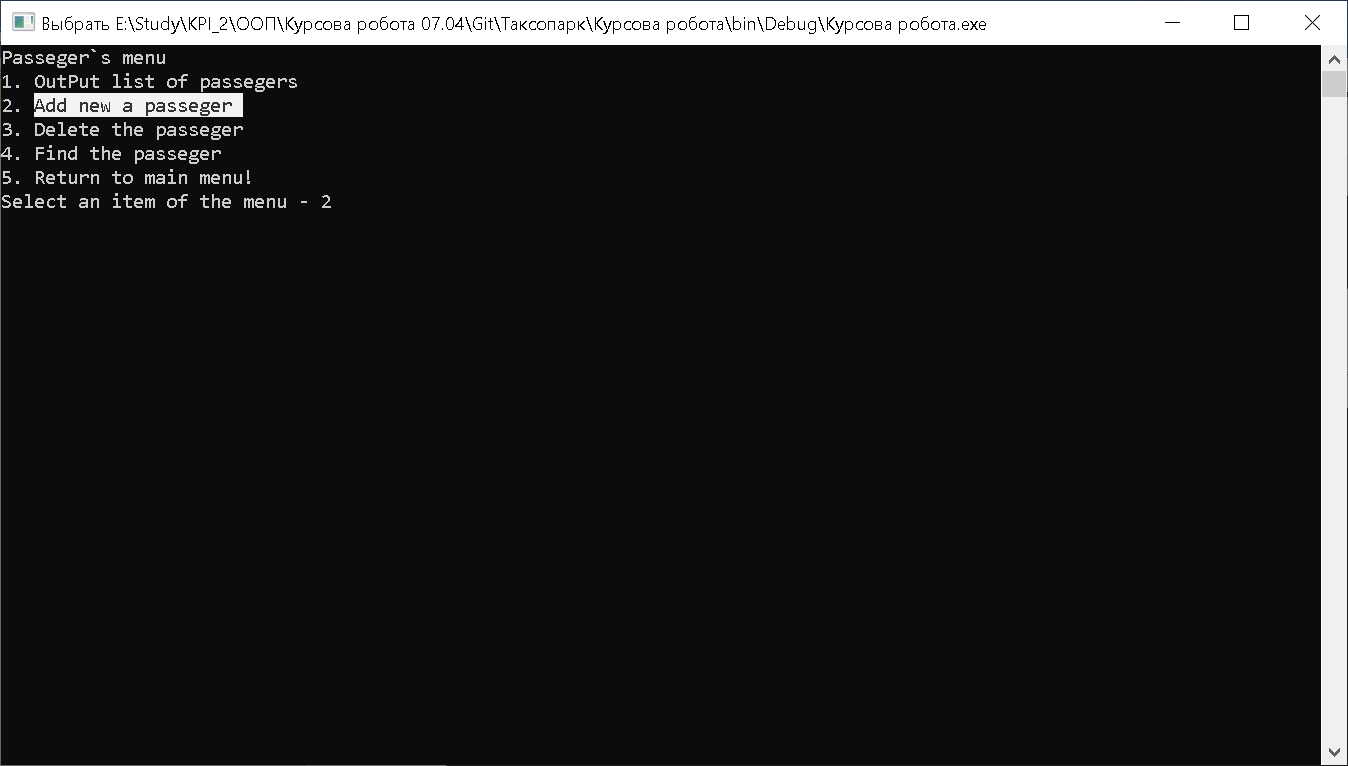
**Рис 2.4**



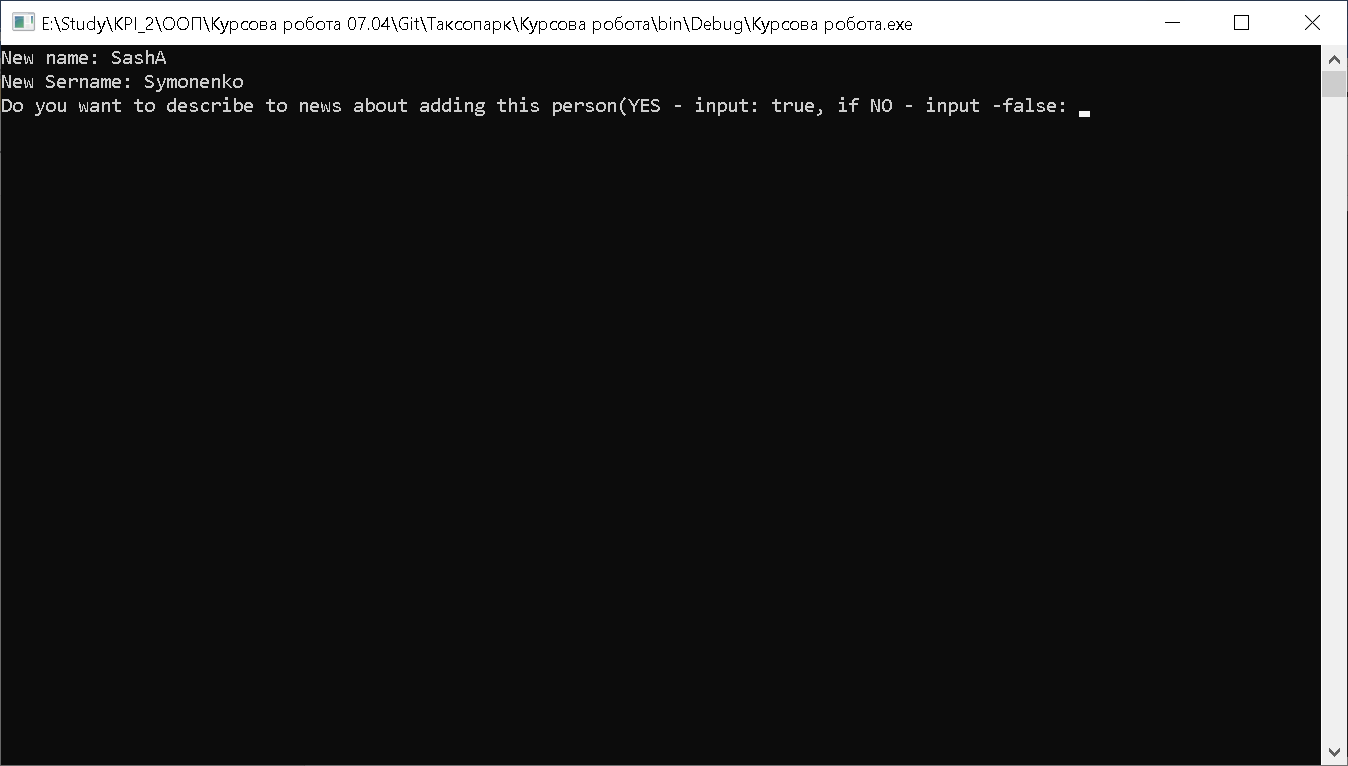
**Рис 2.5**



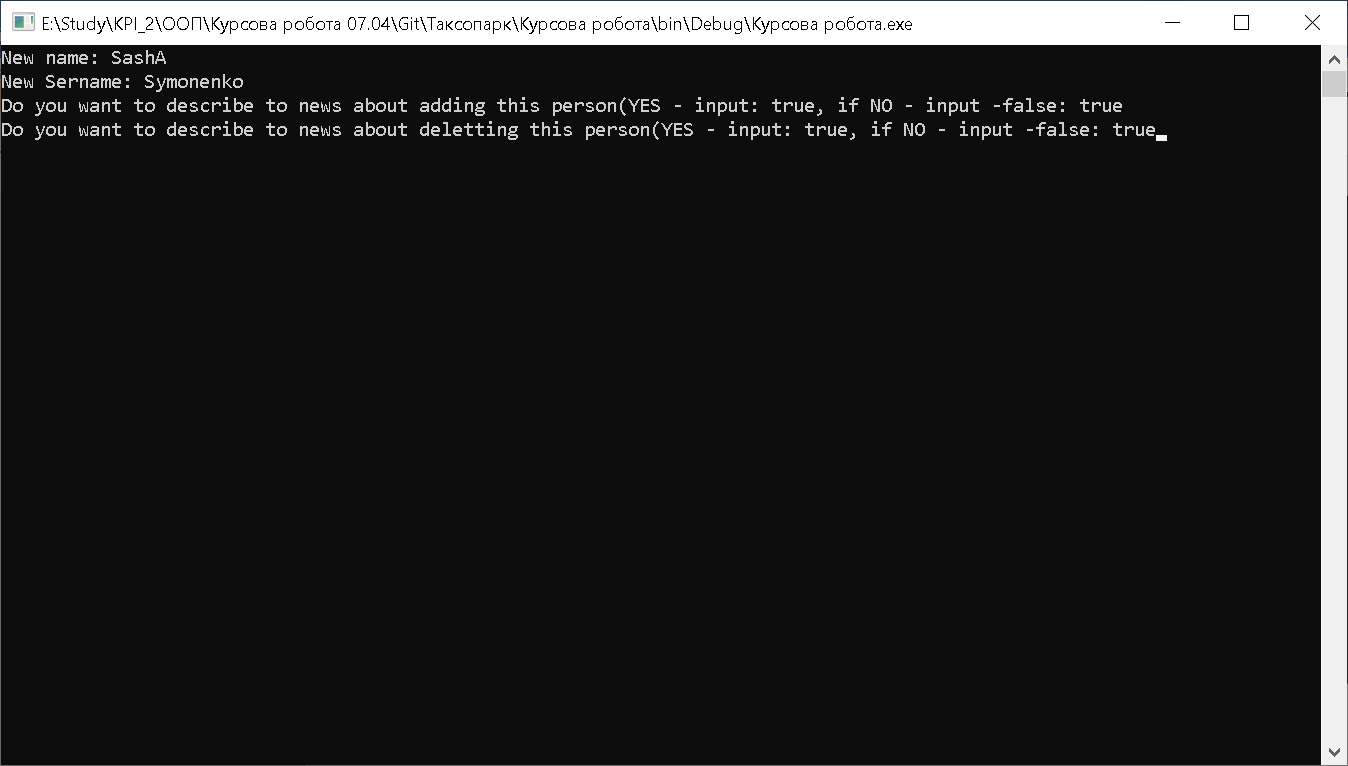
**Рис 2.6**



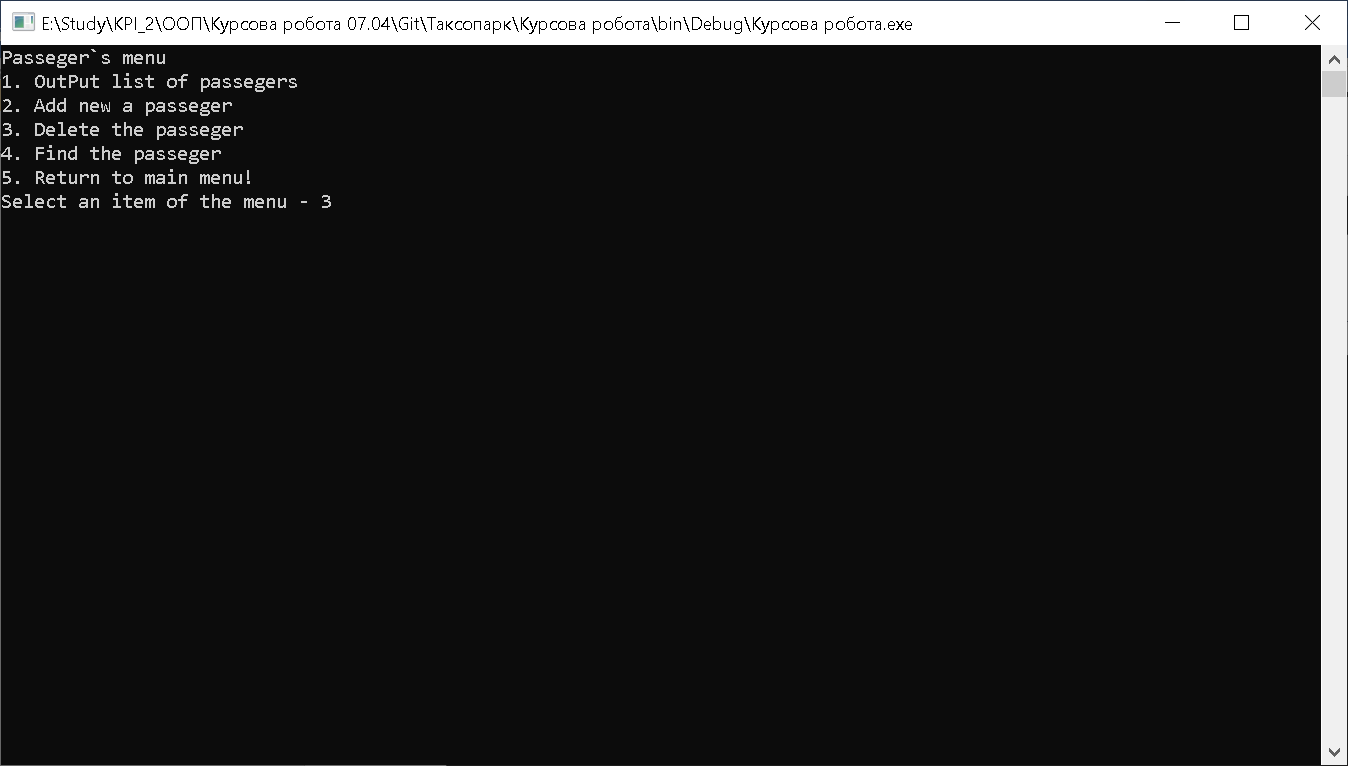
**Рис 2.7**

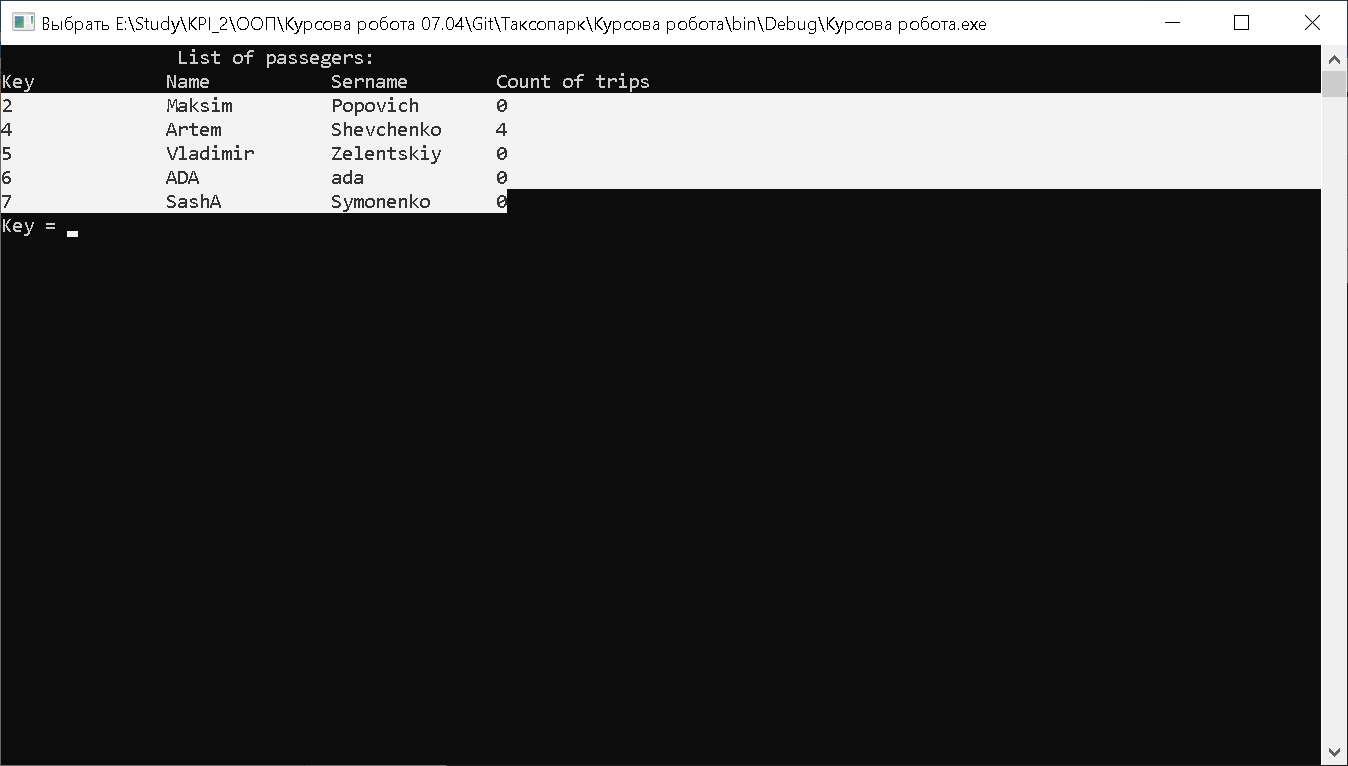


**Рис 2.8**



**Рис 2.9**



**Рис 2.10**

**Код програми, написаної на мові C#:**

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using TaxiParks;

using System.IO;

namespace MainProgect

{

class Program

{

// Головна програма

static void Main(string[] args)

{

//Input list of workers

var listOfWorkers = new List<Workers>();

InputWorkers(ref listOfWorkers);

//Make list of passeger

var listOfPassegers = new List<Passeger>();

InputPasseger(ref listOfPassegers);

//List of trips

var listOfNowTrips = new List<Trips>();

InputNowListOfTrips(ref listOfNowTrips, listOfPassegers);

var park = new List<List<TaxiPark>>();

Menu();

short c;

Console.Write("Choose an item of the Menu: ");

c = short.Parse(Console.ReadLine());

while (true)

{

switch (c)

{

case 1: Console.Clear(); Passeger\_Doing\_Menu(listOfPassegers, listOfNowTrips); Menu(); Console.Write("Choose an item of the Menu: "); c = short.Parse(Console.ReadLine()); break;

case 2: Console.Clear(); Trips\_Doing\_Menu(listOfPassegers, listOfNowTrips); Menu(); Console.Write("Choose an item of the Menu: "); c = short.Parse(Console.ReadLine()); break;

case 3: Console.Clear(); Workers\_Doing\_Menu(listOfWorkers); Menu(); Console.Write("Choose an item of the Menu: "); c = short.Parse(Console.ReadLine()); break;

case 4: Console.Clear(); AboutMe(); Menu(); Console.Write("Choose an item of the Menu: "); c = short.Parse(Console.ReadLine()); break;

case 5: Console.Clear(); Environment.Exit(0); break;

default:

Console.Clear();

Menu();

Console.WriteLine("You inputed not right value of the item!");

Console.Write("Select an item of menu again: ");

c = short.Parse(Console.ReadLine());

break;

}

}

Console.ReadKey();

}

//Метод для виводу інформацію про робітників

public static void WorkersInformationOfList(List<Workers> kind)

{

Console.WriteLine("\t\tList of workers:");

Console.WriteLine("Key".PadRight(15)+"Name".PadRight(15) + "Sername".PadRight(15) + "All salary in the week".PadRight(25) + "TimeToStart".PadRight(15) + "TimeToFinish".PadRight(15));

foreach (var Obj in kind)

{

Console.WriteLine(Obj.key.ToString().PadRight(15)+Obj.Name().PadRight(15) + Obj.SerName().PadRight(15) + Math.Round(Obj.ToPay(), 2).ToString() + "$".PadRight(25 - Math.Round(Obj.ToPay(), 2).ToString().Length) + Obj.timeStart.ToShortTimeString().PadRight(15) + Obj.timeFinish.ToShortTimeString().PadRight(15));

}

}

//Метод для виводу інформацію про пасажирів

public static void PassegersInformation(List<Passeger> passeger)

{

Console.WriteLine($"\t\tList of passegers:");

Console.WriteLine("Key".PadRight(15) + "Name".PadRight(15) + "Sername".PadRight(15) + "Count of trips".PadRight(15));

foreach (var Obj in passeger)

{

Console.WriteLine(Obj.key.ToString().PadRight(15) + Obj.Name().PadRight(15) + Obj.SerName().PadRight(15) + Obj.countTrips.ToString().PadRight(15));

}

}

//Метод для виводу інформацію про поїздки

public static void TripsInformation(List<Trips> listOfNowTrips)

{

Console.WriteLine("\t\tList of trips");

Console.WriteLine("Key".PadRight(15)+$"Name".PadRight(15) + "Sername".PadRight(15) + "Price".PadRight(15) + "Long".PadRight(15) + "TimeInWaiting".PadRight(15));

foreach (var Obj in listOfNowTrips)

{

Console.WriteLine(Obj.key\_of\_passeger.ToString().PadRight(15)+Obj.Name().PadRight(15) + Obj.SerName().PadRight(15) + Math.Round(Obj.ToPay(), 2).ToString() + "$".PadRight(15 - Math.Round(Obj.ToPay(), 2).ToString().Length) + Obj.kilometrs.ToString().PadRight(15) + Obj.waitTime.ToString() + "(hour.)".PadRight(15 - Obj.waitTime.ToString().Length));

}

}

//Метод для пошуку пасажирів

public static void FindPasseger(List<Passeger> passeger)

{

Console.Write("Name: ");

string name = Console.ReadLine();

Console.Write("Sername: ");

string sername = Console.ReadLine();

bool flag = false;

foreach (var value in passeger)

{

if (value.Name() == name && value.SerName() == sername)

{

if (flag == false)

{

Console.WriteLine("Key".PadRight(15) + "Name".PadRight(15) + "Sername".PadRight(15) + "Count of trips".PadRight(15));

}

flag = true;

Console.WriteLine(value.key.ToString().PadRight(15) + value.Name().PadRight(15) + value.SerName().PadRight(15) + value.countTrips.ToString().PadRight(15));

}

}

if (flag == false)

{

Console.WriteLine("This person did not find in list with passegers!!!");

}

}

//Метод для пошуку Поїздок

public static void FindTrips(List<Trips> trips)

{

Console.Write("Name: ");

string name = Console.ReadLine();

Console.Write("Sername: ");

string sername = Console.ReadLine();

bool flag = false;

foreach (var value in trips)

{

if (value.Name() == name && value.SerName() == sername)

{

if (flag == false)

{

Console.WriteLine("Key".PadRight(15) + "Name".PadRight(15) + "Sername".PadRight(15) + "Long".PadRight(15) + "Full- Price".PadRight(15));

}

flag = true;

Console.WriteLine(value.key\_of\_passeger.ToString().PadRight(15) + value.Name().PadRight(15) + value.SerName().PadRight(15) + value.kilometrs.ToString().PadRight(15) + Math.Round(value.ToPay(), 2).ToString() + "$".PadRight(15 - Math.Round(value.ToPay(), 2).ToString().Length));

}

}

if (flag == false)

{

Console.WriteLine("This person did not find in list with trips!!!");

}

}

//Метод для пошуку водіїв

public static void FindWorker(List<Workers> workers)

{

Console.Write("Name: ");

string name = Console.ReadLine();

Console.Write("Sername: ");

string sername = Console.ReadLine();

bool flag = false;

foreach (var value in workers)

{

if (value.Name() == name && value.SerName() == sername)

{

if (flag == false)

{

Console.WriteLine("Key".PadRight(15) + "Name".PadRight(15) + "Sername".PadRight(15) + "Salary".PadRight(15));

}

flag = true;

Console.WriteLine(value.key.ToString().PadRight(15) + value.Name().PadRight(15) + value.SerName().PadRight(15) + value.ToPay().ToString().PadRight(15));

}

}

if (flag == false)

{

Console.WriteLine("This persone did not find in list with workers!!!");

}

}

//Метод для того, щоб зчитувати дані про водіїв з файлу

public static void InputWorkers(ref List<Workers> listWorkers)

{

string pathToWorkers = @"ListOfWorkers.txt";

FileInfo fileInf = new FileInfo(pathToWorkers);

StreamReader inWork = new StreamReader(pathToWorkers);

string wors = inWork.ReadToEnd();

int files\_count = 0;

int lenght = Convert.ToInt32(fileInf.Length);

while (files\_count < lenght)

{

DelegDel delegDel;

DelegAdd delegAdd;

delegAdd = InfoAddWorkers;

delegDel = InfoPopWorkers;

string name = "\0", sername = "\0", dateofStart = "\0", dateofFinish = "\0";

double salary = 0;

string s = "";

char c = ' ';

int key = 0;

c = wors[files\_count];

files\_count += 1;

s += c;

int k\_space = 1;

while (c != '\n' && files\_count < lenght)

{

c = wors[files\_count];

files\_count += 1;

if (c == ' ' || files\_count == lenght || c == '\n')

{

if (files\_count == lenght)

{

s += c;

}

switch (k\_space)

{

case 1: key = Convert.ToInt32(s); break;

case 2: name = s; break;

case 3: sername = s; break;

case 4: salary = Convert.ToDouble(s); break;

case 5: dateofStart = s; break;

case 6: dateofFinish = s; break;

}

k\_space += 1;

s = "";

}

else

{

s += c;

}

}

listWorkers.Add(new Workers(delegAdd,delegDel) { key = key, name = name, sername = sername, salaryInOur = salary, timeStart = DateTime.Parse(dateofStart), timeFinish = DateTime.Parse(dateofFinish) });

}

inWork.Close();

}

//Метод для того, щоб зчитувати дані про пасажирів з файлу

public static void InputPasseger(ref List<Passeger> listOfPassegers)

{

DelegDel delegDel;

DelegAdd delegAdd;

delegAdd = InfoAddPassegers;

delegDel = InfoPopPassegers;

string pathToPassegers = @"ListOfPassegers.txt";

FileInfo fileInf = new FileInfo(pathToPassegers);

StreamReader inWork = new StreamReader(pathToPassegers);

string wors = inWork.ReadToEnd();

int files\_count = 0;

int lenght = Convert.ToInt32(fileInf.Length);

while (files\_count < lenght)

{

string name = "\0", sername = "\0";

int count\_of\_trips = 0;

string s = "";

char c = ' ';

int key = 0;

c = wors[files\_count];

files\_count += 1;

s += c;

int k\_space = 1;

while (c != '\n' && files\_count < lenght)

{

c = wors[files\_count];

files\_count += 1;

if (c == ' ' || files\_count == lenght || c == '\n')

{

if (files\_count == lenght)

{

s += c;

}

switch (k\_space)

{

case 1: key = Convert.ToInt32(s); break;

case 2: name = s; break;

case 3: sername = s; break;

case 4: count\_of\_trips = Convert.ToInt32(s); break;

}

k\_space += 1;

s = "";

}

else

{

s += c;

}

}

listOfPassegers.Add(new Passeger(name, sername, count\_of\_trips, key,delegAdd,delegDel));

}

inWork.Close();

}

//Метод для того, щоб зчитувати діні про поїздки з файлу

public static void InputNowListOfTrips(ref List<Trips> ArrgOfTrips, List<Passeger> passegers)

{

DelegDel delegDel;

DelegAdd delegAdd;

delegAdd = InfoAddTrips;

delegDel = InfoPopTrips;

DelegOutPut OutPutTOFileDeleg = OutPutListOfTripsInTheFile;

string pathNowTrips = @"ListOfTrips.txt";

FileInfo fileInf = new FileInfo(pathNowTrips);

StreamReader inTrip = new StreamReader(pathNowTrips);

string trees = inTrip.ReadToEnd();

int files\_count = 0;

int lenght = Convert.ToInt32(fileInf.Length);

while (files\_count < lenght)

{

int key = 0;

double Long = 0;

double TripNowPrice = 0;

double waiting\_time = 0;

double waiting\_price = 0;

string s = "";

char c = ' ';

c = trees[files\_count];

files\_count += 1;

s += c;

int k\_space = 1;

while (c != '\n' && files\_count < lenght)

{

c = trees[files\_count];

files\_count += 1;

if (c == ' ' || files\_count == lenght || c == '\n')

{

if (files\_count == lenght)

{

s += c;

}

switch (k\_space)

{

case 1: key = Convert.ToInt32(s); break;

case 2: Long = Convert.ToDouble(s); break;

case 3: TripNowPrice = Convert.ToDouble(s); break;

case 4: waiting\_time = Convert.ToDouble(s); break;

case 5: waiting\_price = Convert.ToDouble(s); break;

}

k\_space += 1;

s = "";

}

else

{

s += c;

}

}

for (int i = 0; i < passegers.Count(); i++)

{

if (passegers[i].key == key)

{

ArrgOfTrips.Add(new Trips(passegers[i], key, Long, TripNowPrice, waiting\_time, waiting\_price,OutPutTOFileDeleg,delegAdd,delegDel));

break;

}

}

}

inTrip.Close();

}

//Вивід даних про робітників у файл

public static void OutPutWorkersInFile(List<Workers> workers)

{

string pathToPassegers = @"ListOfWorkers.txt";

StreamWriter inID = new StreamWriter(pathToPassegers);

foreach (var Val in workers)

{

inID.WriteLine(Val.key.ToString() + " " + Val.Name() + " " + Val.SerName() + " " + Val.salaryInOur.ToString() + " " + Val.timeStart.ToShortTimeString() + " " + Val.timeFinish.ToShortTimeString());

}

inID.Close();

}

//Вивід даних про пасажирів у файл

public static void OutPutPassegersInFile(List<Passeger> passegers)

{

string pathToPassegers = @"ListOfPassegers.txt";

StreamWriter inWork = new StreamWriter(pathToPassegers);

foreach (var Val in passegers)

{

inWork.WriteLine(Val.key.ToString() + " " + Val.Name() + " " + Val.SerName() + " " + Val.countTrips);

}

inWork.Close();

}

//Вивід даних про поїздки у файл

public static void OutPutListOfTripsInTheFile(List<Trips> trips)

{

string pathToPassegers = @"ListOfTrips.txt";

StreamWriter ToTrips = new StreamWriter(pathToPassegers);

for (int i = 0; i < trips.Count(); i++)

{

if (i != trips.Count() - 1)

ToTrips.WriteLine(trips[i].key\_of\_passeger.ToString() + " " + trips[i].kilometrs.ToString() + " " + trips[i].priceForOneKilometr.ToString() + " " + trips[i].waitTime.ToString() + " " + trips[i].waitPrice.ToString());

else

{

ToTrips.Write(trips[i].key\_of\_passeger.ToString() + " " + trips[i].kilometrs.ToString() + " " + trips[i].priceForOneKilometr.ToString() + " " + trips[i].waitTime.ToString() + " " + trips[i].waitPrice.ToString());

}

}

ToTrips.Close();

}

//Метод для видалення поїздки

static public void deleteTrip(List<Trips> trips)

{

Console.Write("Key = ");

int key = Convert.ToInt32(Console.ReadLine());

Console.Write("Name: ");

string name = Console.ReadLine();

Console.Write("Sername: ");

string sername = Console.ReadLine();

Trips.Delete(name, sername, key, trips);

OutPutListOfTripsInTheFile(trips);

}

//Метод для видалення робітника

static public void deleteWorker(List<Workers> workers)

{

Console.Write("key = ");

int key = Convert.ToInt32(Console.ReadLine());

Console.Write("Name: ");

string name = Console.ReadLine();

Console.Write("Sername: ");

string sername = Console.ReadLine();

Workers.Delete(workers, name, sername, key);

OutPutWorkersInFile(workers);

}

//Метод для видалення пасажира

static public void deletePasseger(List<Passeger> passegers, List<Trips> trips)

{

Console.Write("Key = ");

int key = Convert.ToInt32(Console.ReadLine());

Console.Write("Name: ");

string name = Console.ReadLine();

Console.Write("Sername: ");

string sername = Console.ReadLine();

Passeger.Delete(trips, passegers, name, sername,key);

OutPutPassegersInFile(passegers);

OutPutListOfTripsInTheFile(trips);

}

//Метод для додавання нового пасажира

static public void AddNewPasseger(List<Passeger> pass)

{

Console.Write("New name: ");

string name = Console.ReadLine();

Console.Write("New Sername: ");

string sername = Console.ReadLine();

Console.Write("Do you want to describe to news about adding this person(YES - input: true, if NO - input -false: ");

bool BDelegAdd = bool.Parse(Console.ReadLine());

DelegAdd delegAdd = null;

if (BDelegAdd==true)

{

delegAdd = InfoAddPassegers;

}

Console.Write("Do you want to describe to news about deletting this person(YES - input: true, if NO - input -false: ");

bool BDelegDel = bool.Parse(Console.ReadLine());

DelegDel delegDel = null;

if (BDelegDel == true)

{

delegDel = InfoPopPassegers;

}

Passeger.Add(pass, name, sername,delegAdd,delegDel);

OutPutPassegersInFile(pass);

}

//Метод для додавання нової поїздки

static public void AddTrip(List<Trips> NTrip, List<Passeger> passegers)

{

DelegOutPut OutPutToFile = OutPutListOfTripsInTheFile;

Console.Write("Key = ");

int key = Convert.ToInt32(Console.ReadLine());

Console.Write("Name: ");

string name = Console.ReadLine();

Console.Write("Sername: ");

string sername = Console.ReadLine();

Console.Write("Long:");

double Long = Convert.ToDouble(Console.ReadLine());

if (Long <= 0)

{

throw new ArgumentNullException("Long must be more than zero!!!");

}

Console.Write("Price for one kilimert: ");

double PriceForOneKilometr = Convert.ToDouble(Console.ReadLine());

if (PriceForOneKilometr <= 0)

{

throw new ArithmeticException("Price for a one kilometrs should be more than 0!");

}

Console.Write("Waiting time: ");

double waiting\_time = Convert.ToDouble(Console.ReadLine());

if (waiting\_time < 0)

{

throw new ArgumentException("Waiting time must be >= 0");

}

Console.Write("Waiting price: ");

double waiting\_price = Convert.ToDouble(Console.ReadLine());

if (waiting\_price < 0)

{

throw new ArgumentException("Price for a waiting should be more than zero or is equal with zero!!!");

}

Console.Write("Do you want to describe to news about adding this trip(YES - input: true, if NO - input -false: ");

bool BDelegAdd = bool.Parse(Console.ReadLine());

DelegAdd delegAdd = null;

if (BDelegAdd == true)

{

delegAdd = InfoAddTrips;

}

Console.Write("Do you want to describe to news about deletting this trip(YES - input: true, if NO - input -false: ");

bool BDelegDel = bool.Parse(Console.ReadLine());

DelegDel delegDel = null;

if (BDelegDel == true)

{

delegDel = InfoPopTrips;

}

Trips.Add(passegers, NTrip, name, sername, key, Long, PriceForOneKilometr, waiting\_time, waiting\_price,OutPutToFile,delegAdd,delegDel);

OutPutListOfTripsInTheFile(NTrip);

OutPutPassegersInFile(passegers);

}

//Метод для додавання нового водія

static public void AddNewWorker(List<Workers> workers)

{

Console.Write("New name: ");

string name = Console.ReadLine();

Console.Write("New Sername: ");

string sername = Console.ReadLine();

Console.Write("salaryInOur: ");

double salary = Convert.ToDouble(Console.ReadLine());

if (salary <= 0)

{

throw new ArgumentNullException("Salary must be more than zero!!!");

}

Console.Write("Date of starting: ");

DateTime timeOfStarting = DateTime.Parse(Console.ReadLine());

Console.Write("Date of finishing: ");

DateTime timeOfFinishing = DateTime.Parse(Console.ReadLine());

Console.Write("Do you want to describe to news about adding this person(YES - input: true, if NO - input -false: ");

bool BDelegAdd = bool.Parse(Console.ReadLine());

DelegAdd delegAdd = null;

if (BDelegAdd == true)

{

delegAdd = InfoAddWorkers;

}

Console.Write("Do you want to describe to news about deletting this person(YES - input: true, if NO - input -false: ");

bool BDelegDel = bool.Parse(Console.ReadLine());

DelegDel delegDel = null;

if (BDelegDel == true)

{

delegDel = InfoPopWorkers;

}

Workers.Add(workers, name, sername, salary, timeOfStarting, timeOfFinishing,delegDel,delegAdd);

OutPutWorkersInFile(workers);

}

//Метод для виводу інформації додавання Робітників

static public void InfoAddWorkers()

{

Console.WriteLine("Worker was add!");

}

//Метод для виводу інформації додавання Поїздок

static public void InfoAddTrips()

{

Console.WriteLine("Trip was add!");

}

//Метод для вивіду ітформації додавання Пасажирів

static public void InfoAddPassegers()

{

Console.WriteLine("Passeger was add!");

}

//Метод для виводу інформації видалення Робітників

static public void InfoPopWorkers()

{

Console.WriteLine("Worker was delete!");

}

//Метод для виводу інформації видалення Поїздок

static public void InfoPopTrips()

{

Console.WriteLine("Trip was delete!");

}

//Метод для виводу інформації видалення Пасажирів

static public void InfoPopPassegers()

{

Console.WriteLine("Passeger was delete!");

}

//Вивід у консоль головного меню

static public void Menu()

{

Console.WriteLine("Menu");

Console.WriteLine("1. Come to menu for passegers");

Console.WriteLine("2. Come to menu for trips");

Console.WriteLine("3. Come to menu for workers");

Console.WriteLine("4. About author! ");

Console.WriteLine("5. Exit!");

}

//Вивід інформацію про автора курсової роботи

static public void AboutMe()

{

Console.WriteLine("Sername: Samilenko\nName: Oleksandr");

Console.WriteLine("First course");

Console.WriteLine("Group of: IS-93");

}

//Метод для реалізації роботи з меню для пасажирів

static public void Passeger\_Doing\_Menu(List<Passeger> listOfPassegers, List<Trips> listOfNowTrips )

{

Passeger\_Menu();

short c;

Console.Write("Select an item of the menu - ");

c = short.Parse(Console.ReadLine());

bool flag = true;

while (flag==true)

{

switch (c)

{

case 1: Console.Clear(); PassegersInformation(listOfPassegers); Passeger\_Menu(); Console.Write("Select an item of the menu: "); c = short.Parse(Console.ReadLine()); break;

case 2: Console.Clear(); AddNewPasseger(listOfPassegers); Passeger\_Menu(); Console.Write("Select an item of the menu: "); c = short.Parse(Console.ReadLine()); break;

case 3: Console.Clear(); PassegersInformation(listOfPassegers); deletePasseger(listOfPassegers, listOfNowTrips); Passeger\_Menu(); Console.Write("Select an item of the menu: "); c = short.Parse(Console.ReadLine()); break;

case 4: Console.Clear(); FindPasseger(listOfPassegers); Passeger\_Menu(); Console.Write("Select an item of the menu: "); c = short.Parse(Console.ReadLine()); break;

case 5: Console.Clear(); flag = false; break;

default:

Console.Clear();

Menu();

Console.WriteLine("You inputed not right value of the item!");

Console.Write("Select an item of menu again: ");

c = short.Parse(Console.ReadLine());

break;

}

}

}

//Метод для реалізації роботи з меню поїздок

static public void Trips\_Doing\_Menu(List<Passeger> listOfPassegers, List<Trips> listOfNowTrips)

{

Trips\_Menu();

short c;

Console.Write("Select an item of the menu - ");

c = short.Parse(Console.ReadLine());

bool flag = true;

while (flag)

{

switch (c)

{

case 1: Console.Clear(); TripsInformation(listOfNowTrips); Trips\_Menu(); Console.Write("Select an item of the menu: "); c = short.Parse(Console.ReadLine()); break;

case 2: Console.Clear(); PassegersInformation(listOfPassegers); AddTrip(listOfNowTrips,listOfPassegers); Trips\_Menu(); Console.Write("Select an item of the menu: "); c = short.Parse(Console.ReadLine()); break;

case 3: Console.Clear(); TripsInformation(listOfNowTrips); deleteTrip(listOfNowTrips); Trips\_Menu(); Console.Write("Select an item of the menu: "); c = short.Parse(Console.ReadLine()); break;

case 4: Console.Clear(); FindTrips(listOfNowTrips); Trips\_Menu(); Console.Write("Select an item of the menu: "); c = short.Parse(Console.ReadLine()); break;

case 5: Console.Clear(); flag = false; break;

default:

Console.Clear();

Menu();

Console.WriteLine("You inputed not right value of the item!");

Console.Write("Select an item of menu again: ");

c = short.Parse(Console.ReadLine());

break;

}

}

}

//Метод для реалізації роботи з меню поїздок

static public void Workers\_Doing\_Menu(List<Workers> listOfWorkers)

{

Workers\_Menu();

short c;

Console.Write("Select an item of the menu - ");

c = short.Parse(Console.ReadLine());

bool flag = true;

while (flag)

{

switch (c)

{

case 1: Console.Clear(); WorkersInformationOfList(listOfWorkers); Workers\_Menu(); Console.Write("Select an item of the menu: "); c = short.Parse(Console.ReadLine()); break;

case 2: Console.Clear(); AddNewWorker(listOfWorkers); Workers\_Menu(); Console.Write("Select an item of the menu: "); c = short.Parse(Console.ReadLine()); break;

case 3: Console.Clear(); WorkersInformationOfList(listOfWorkers); deleteWorker(listOfWorkers); Workers\_Menu(); Console.Write("Select an item of the menu: "); c = short.Parse(Console.ReadLine()); break;

case 4: Console.Clear(); FindWorker(listOfWorkers); Workers\_Menu(); Console.Write("Select an item of the menu: "); c = short.Parse(Console.ReadLine()); break;

case 5: Console.Clear(); flag = false; break;

default:

Console.Clear();

Menu();

Console.WriteLine("You inputed not right value of the item!");

Console.Write("Select an item of menu again: ");

c = short.Parse(Console.ReadLine());

break;

}

}

}

//Вивід меню пасажирів

static public void Passeger\_Menu()

{

Console.WriteLine("Passeger`s menu");

Console.WriteLine("1. OutPut list of passegers");

Console.WriteLine("2. Add new a passeger");

Console.WriteLine("3. Delete the passeger");

Console.WriteLine("4. Find the passeger");

Console.WriteLine("5. Return to main menu!");

}

//Вивід меню для поїздок

static public void Trips\_Menu()

{

Console.WriteLine("Trip`s menu");

Console.WriteLine("1. OutPut list of trips");

Console.WriteLine("2. Add new a trip");

Console.WriteLine("3. Delete the trip");

Console.WriteLine("4. Find the trip");

Console.WriteLine("5. Return to main menu!");

}

//Вивід меню для робітників - водіїв

static public void Workers\_Menu()

{

Console.WriteLine("Worker`s menu");

Console.WriteLine("1. OutPut list of workers");

Console.WriteLine("2. Add new a new worker");

Console.WriteLine("3. Delete the worker");

Console.WriteLine("4. Find the worker");

Console.WriteLine("5. Return to main menu!");

}

}

}

**ITrips.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TaxiParks

{

public interface ITrips//Інтерфейс

{

double ToPay();//Метод ціни

string Name();//Ім'я

string SerName();//Фамілії

}

}

**Passegers.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace TaxiParks

{

public delegate void DelegAdd();//Делегат додавання нового об'єкта

public delegate void DelegDel();//Делегат видалення старого об'єкта

public class Passeger: Person

{

public static event DelegAdd EventAdd;//Подія додавання пасажира

public static event DelegDel EventDel;//Подія видалення пасажира

public int countTrips=0;//Значення кількості поїздок

public int key { get; set; }//Значення ключа пасажира

//Конструктор без параметрів

public Passeger(){}

//Конструктор

public Passeger(string name,string sername,int countTrips,int key, DelegAdd addEvent, DelegDel deleteEvent)

{

if(name.Length==0 || name==null||sername.Length==0||sername==null)

{

throw new ArgumentNullException("String of name has a lenght with is equal to null!");

}

if(countTrips<0)

{

throw new ArgumentException("count of trips this passeger must be more or equal zero");

}

this.name = name;

this.sername = sername;

this.countTrips = countTrips;

this.key = key;

EventAdd = (addEvent != null) ? addEvent : null;

EventDel = (deleteEvent != null) ? deleteEvent : null;

}

//Метод для видалення пасажира

static public void Delete(List<Trips> trips,List<Passeger> passegers,string name,string sername,int key)

{

bool flag = false;

foreach (var trip in trips)

{

if (trip.key\_of\_passeger == key && trip.Name() == name && trip.SerName() == sername)

{

trips.Remove(trip);

break;

}

}

foreach (var passeger in passegers)

{

if (passeger.Name() == name && passeger.SerName() == sername && passeger.key == key)

{

flag = true;

passegers.Remove(passeger);

break;

}

}

if (flag == false)

{

throw new ArgumentException("Did not find this person!");

}

for (int i = 0; i < passegers.Count(); i++)

{

passegers[0].key = i;

}

foreach (var passeger in passegers)

{

for (int i = 0; i < trips.Count(); i++)

{

if (passeger.Name() == trips[i].Name() && passeger.SerName() == trips[i].SerName())

{

trips[i].key\_of\_passeger = passeger.key;

}

}

}

Passeger.EventDel?.Invoke();

}

//Додавання пасажира

public static void Add(List<Passeger> pass, string name,string sername, DelegAdd addEvent, DelegDel deleteEvent)

{

int countofList = pass[pass.Count() - 1].key + 1;

pass.Add(new Passeger(name, sername, 0, countofList, addEvent, deleteEvent));

Passeger.EventAdd?.Invoke();

}

}

}

**TaxiPark.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TaxiParks

{

public abstract class Person:ITrips

{

public string name { private get; set; }//Властивість імені

public string sername { private get; set; }//Властивість фамілії

//Повернення ім'я

public virtual string Name()

{

return name;

}

//Повернення фамілії

public virtual string SerName()

{

return sername;

}

//Повернення ціни/вартості

public virtual double ToPay()

{

return 0;

}

}

}

**Trips.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TaxiParks

{

public delegate void DelegOutPut(List<Trips> NTrip);// Делегат виведення інформації при зміні даних списку поїздок

public class Trips :

{

public static event DelegOutPut OutPutEvent;//Подія зміни даних поїздки

public static event DelegAdd EventAdd;//Подія додавання поїздки

public static event DelegDel EventDel;//Подія видалення поїздки

public double kilometrs { get; set; }//Кількість кілометрів за поїздку

public double waitTime { get; set; }//час очікування

public double priceForOneKilometr { get; set; }//ціна за один кілометр

public double waitPrice{ get; set; }//Ціна за одну хвилину очікуання пасажира

public string namePassenger { private get; set; }//ім'я пасажира

public string sernamePassenger { private get; set; }//фамілія пасажира

public int key\_of\_passeger { get; set; }//ключ пасажира

public Passeger peo;//об'єк пасажир

public Discount disc;//об'єкт знижки

//Конструктор з параметрами

public Trips(DelegOutPut OutPutEvent,DelegAdd addEvent,DelegDel deleteEvent)

{

if (OutPutEvent!= null)

{

Trips.OutPutEvent += OutPutEvent;

}

EventAdd = (addEvent != null) ? addEvent : null;

EventDel = (deleteEvent != null) ? deleteEvent : null;

}

//Конструктор без параметрів

public Trips() { }

//Конструктор з параметрами

public Trips(Passeger passeger,int key\_of\_Passeger, double kilometrs, double priceForOneKilometr,double waitTime,double waitPrice, DelegOutPut OutPutEvent, DelegAdd addEvent, DelegDel deleteEvent)

{

if(kilometrs<=0)

{

throw new ArgumentNullException("Hours must be more than zero!!!");

}

if(priceForOneKilometr<=0)

{

throw new ArgumentException("Price for one kilometr must be more than zero!");

}

if(waitTime<0)

{

throw new ArgumentNullException("wait TIME should be more or equal zero");

}

if(waitTime>0 && waitPrice<=0)

{

throw new ArithmeticException("Waitng price must be more than zero, when waiting time > 0");

}

this.kilometrs = kilometrs;

this.waitTime = waitTime;

this.priceForOneKilometr = priceForOneKilometr;

this.waitPrice = waitPrice;

this.waitTime = waitTime;

this.key\_of\_passeger = key\_of\_Passeger;

passeger.countTrips += 1;

peo = passeger;

namePassenger = peo.Name();

sernamePassenger = peo.SerName();

disc = new Discount(peo.countTrips);

if(OutPutEvent!=null)

{

Trips.OutPutEvent += OutPutEvent;

}

EventAdd += (addEvent != null) ? addEvent : null;

EventDel += (deleteEvent != null) ? deleteEvent : null;

}

//Перевизначення методу повернення вартості

public override double ToPay()

{

return (disc.ToPay(kilometrs, priceForOneKilometr) + waitTime \* waitPrice);

}

//Перевизначення методу повернення імені

public override string Name()

{

return namePassenger;

}

//Перевизначення методу повернення фамілії

public override string SerName()

{

return sernamePassenger;

}

//Видалення поїздки

static public void Delete(string name,string sername,int key,List<Trips> trips)

{

bool flag = false;

foreach (var trip in trips)

{

if (trip.Name() == name && trip.SerName() == sername && trip.key\_of\_passeger == key)

{

flag = true;

trips.Remove(trip);

break;

}

}

if (flag == false)

{

throw new ArgumentException("Did not find this person!");

}

Trips.EventDel?.Invoke();

}

//Додавання поїздки

static public void Add(List<Passeger> passegers, List<Trips> NTrip, string name,string sername,int key,double Long, double PriceForOneKilometr,double waiting\_time,double waiting\_price, DelegOutPut OutPutEvent, DelegAdd addEvent, DelegDel deleteEvent)

{

bool flag = false;

foreach (var val in passegers)

{

if (val.Name() == name && val.SerName() == sername && val.key == key)

{

flag = true;

break;

}

}

if (flag == false)

{

throw new ArgumentException("Did not find this person in list of Passegers. We recomendet to you add this person in list of passegers");

}

flag = false;

foreach (var val in NTrip)

{

if (val.key\_of\_passeger == key)

{

throw new ArgumentException("This person yet to drive ");

}

}

flag = false;

foreach (var Value in passegers)

{

if (Value.Name() == name && Value.SerName() == sername && Value.key == key)

{

NTrip.Add(new Trips(Value, key, Long, PriceForOneKilometr, waiting\_time, waiting\_price,Trips.OutPutEvent, addEvent, deleteEvent));

if(OutPutEvent!=null)

{

Trips.OutPutEvent = OutPutEvent;

Trips.OutPutEvent?.Invoke(NTrip);

}

flag = true;

break;

}

}

if (flag == false)

{

throw new ArgumentException("Warning: This passeger did not find. You can to try this: add new passeger.");

}

Trips.EventAdd?.Invoke();

}

}

}

**Workers.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace TaxiParks

{

public class Workers : Person

{

public static event DelegAdd EventAdd;//Подія додавання елемента

public static event DelegDel EventDelete;//Подія видалення елемента

public double salaryInOur { get; set; }//заробітня плата за годину

public double timeInWeek { get; set;}//кількість роботи за тиждень

public DateTime timeStart{ get; set; }//час початку роботи

public DateTime timeFinish { get; set; }//значення часу закінчення роботи

public int key { get; set; }//значення ключа

//Конструктор з параметрами

public Workers(DelegAdd eventAdd,DelegDel eventDelete)

{

Workers.EventDelete = (eventDelete!=null) ? eventDelete : null;

Workers.EventAdd = (eventAdd != null) ? eventAdd : null;

}

//Перевизначення методу повернення вартості, у даному випадку заробітної плати працівника (людини)

public override double ToPay()

{

return 5\* salaryInOur\*((Convert.ToDouble(timeFinish.Hour) + Convert.ToDouble(timeFinish.Minute) / 60) - (Convert.ToDouble(timeStart.Hour) + Convert.ToDouble(timeStart.Minute) / 60));

}

//Видалення старого робітника (водія)

public static void Delete(List<Workers> workers,string name,string sername,int key)

{

bool flag = false;

foreach (var worker in workers)

{

if (worker.key == key && worker.Name() == name && worker.SerName() == sername)

{

flag = true;

workers.Remove(worker);

break;

}

}

if (flag == false)

{

throw new ArgumentException("Did not find this person!");

}

Workers.EventDelete?.Invoke();

}

//Додавання нового водія(робітника)

public static void Add(List<Workers> workers,string name,string sername,double salary,DateTime timeOfStarting, DateTime timeOfFinishing,DelegDel delF,DelegAdd addF)

{

int countofList = workers[workers.Count() - 1].key;

workers.Add(new Workers(addF,delF) { key = countofList + 1, name = name, sername = sername, salaryInOur = salary, timeStart = timeOfStarting, timeFinish = timeOfFinishing });

Workers.EventAdd?.Invoke();

}

}

}

**Discount.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TaxiParks

{

public class Discount

{

public double discount{ get; set; }//Властивіть скидки

public int countTrips;//Кількість поїздок

public Discount(int countTrips)//Конструктор, на вхід якого задається параметр - кількості поїздок

{

this.countTrips = countTrips;

}

public double ToPay(double hours,double priceOne)//Метод, який повертає вартість повної поїздки

{

discount = countTrips % priceOne+9;

return (priceOne \* hours \* (1 - discount/89));

}

}

}