Санкт-Петербургский Национальный Исследовательский Университет Информационных технологий, механики и оптики

**Домашнее задание**

**Создание программного обеспечения учёта грузового транспорта**

Выполнил:

Затикян Сергей Амренович

Группа K3121

Проверила:

Казанова Полина Петровна

Санкт-Петербург

2023

**Цель работы:**

Создать программное обеспечение учета грузового транспорта для Автотранспортного отдела логистической компании.

**Задачи:**

1. Добавлять/удалять грузовой транспорт;
2. Просматривать весь доступный транспорт;
3. Просматривать грузовой транспорт по грузоподъемности;
4. Просматривать свободный грузовой транспорт;
5. Вносить заявку на перевоз груза по указанным габаритам;
6. Подобрать и забронировать транспорт;
7. Просматривать занятый грузовой транспорт;
8. Реализовать интерфейс программы;
9. Реализовать возможность сохранения данных в базу данных.

**Анализ предметной области и требований:**

Для более чёткого представления дальнейшей работы было решено рассмотреть уже имеющиеся на рынке аналоги. Узнав, что такие приложения, в основном, являются собственными у каждой транспортной компании, я обратился к своему папе, работающему в этой области.

Именно он подсказал мне, как лучше оформить интерфейс, чтобы не открывать очень много больших окон, отвечающих за каждый раздел. Он предложил сделать программу через маленькие всплывающие окошки, а также разместить все поля ввода в главном меню.

**Ход работы:**

Было принято решение разделить программу на файлы, которых получилось 4: файл main.py, содержащий в себе базовый класс Trucks, файл all\_trucks.py, содержащий в себе класс all\_trucks, файл db\_trucks.py, содержащий в себе класс DataBase и файл tk\_trucks.py, в котором прописан весь графический интерфейс.

В самом начале было решено написать базовый класс Trucks, который бы означал конкретную грузовую машину. При инициализации требуется указать number - номер по счёту в базе данных (указывается автоматически), name – название транспорта, weight – грузоподъёмность транспорта и booking – занятость транспортного средства. Далее в классе прописаны методы, возвращающие параметры объекта, так как все переменные внутри класса инкапсулированы и доступны для обращения только внутри него. Прописаны методы get\_number(), get\_name(), get\_weight(), is\_free() возвращающие соответственно номер, имя, грузоподъёмность и занятость.Также, имеется метод can\_carry(weight), получающий на вход необходимый для перевозки груз. Он производит сравнение с грузоподъёмности автомобиля и возвращает True или False. Последним методом этого класса является book\_car(), который вызывается при бронировании машины. Он меняет значения переменной, отвечающей за занятость машины, и обновляет эти данные в базе данных. Код класса Trucks изображён на рисунке 1.

Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 1 – Класс Trucks

Далее был написан all\_trucks(), который был своеобразной динамической базой данных. Создан он был для того, чтобы постоянно не обращаться к БД, что занимало бы много времени. Именно поэтому, пока работала программа, все изменения вносились сначала в объект этого класса, а уже в конце программы сохранялось в базу данных. При инициализации создаётся список, в котором и будут содержаться объекты транспорта. Также, сразу после этого в список выгружается вся база данных с помощью специальной функции класса DataBase. Первым же реализованным методом был add\_cars(\*args), который добавлял все объекты транспорта, переданные аргументом, в список со всем остальным транспортом. Далее был написан метод get\_all\_cars(), который выводит все имеющиеся в списке атрибуты каждого транспортного средства. Возвращается список из кортежей, где кортеж соответствует одному транспортному средству. В кортеж данные добавляются путём вызова всех get-ов для каждого объекта. Увидеть реализацию можно на рисунке 2.

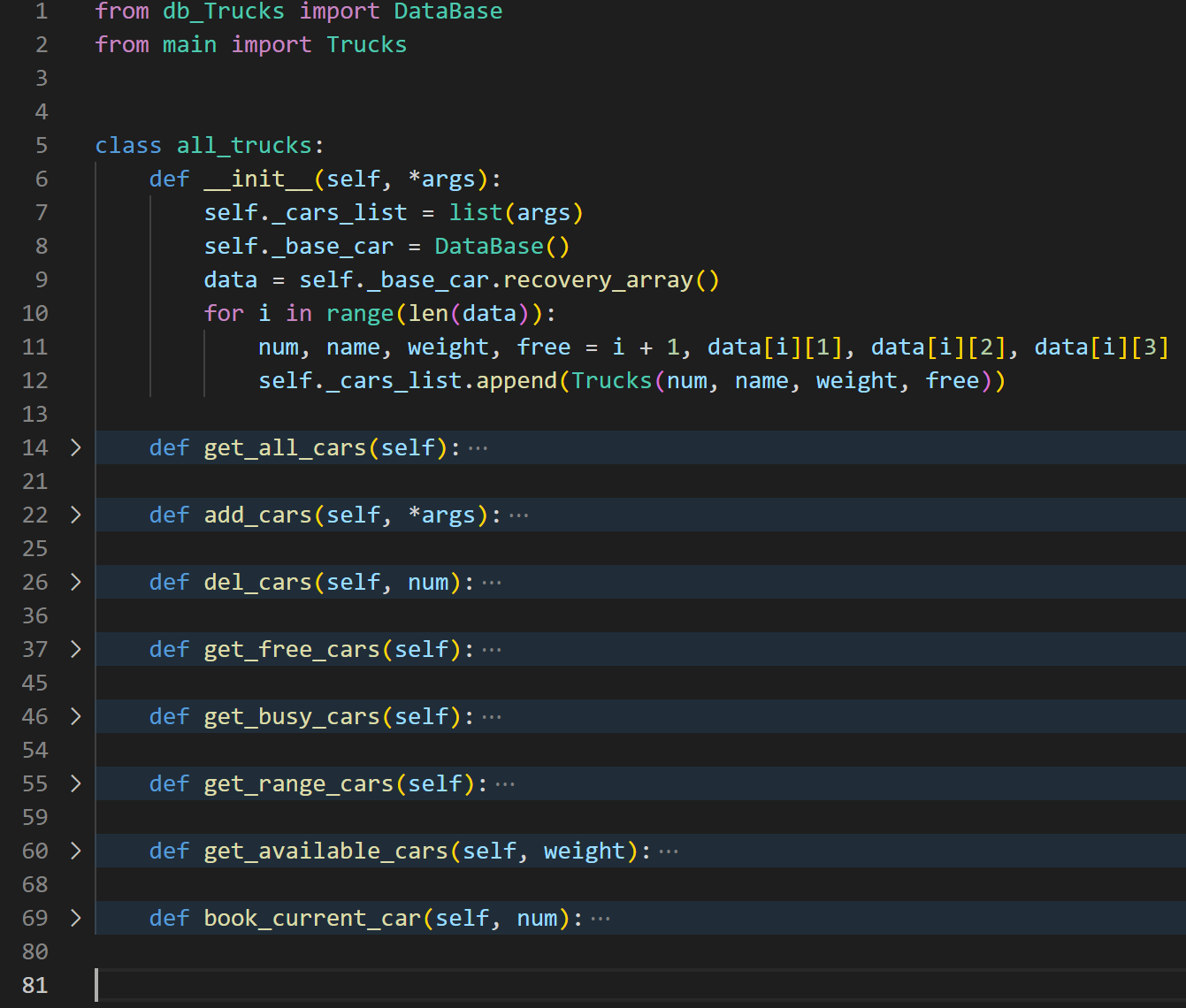


Рисунок 2 – Класс all\_trucks и его основные методы

После этого были написаны всевозможные get-ы, которых требует выданное задание. А именно get\_free\_cars(), get\_busy\_cars(), get\_range\_cars(), get\_avaliable\_cars(weight), которые возвращали все свободные, занятые, отсортированные по грузоподъёмности и доступные для перевозки груза весом weight машины соответственно. Вся логика сбора информации и вывода взята из метода get\_all\_cars(), только лишь добавлялись определённые сортировки и условия. И последними двумя методами являются del\_cars(num) и book\_current\_car(num), которые удаляют транспорт из списка и бронируют машину для перевозки. На вход методам подаётся num – номер машины в базе данных, который можно увидеть при вызове всех машин. Грубо говоря, в обоих методах идёт поиск необходимого номера путём перебора циклом for, если транспорт с таким номером найден, то выполняются необходимые действия (удаление из списка или вызов метода бронирования) и прекращается цикл. Если же нет, то возвращается ошибка. Методы расписаны на рисунке 3 и 4.

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 3 – Все методы

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 4 – Все методы

Последним классом в программе будет DataBase, который является набором стандартных для моей программы методов с запросами к базе данных. Например, первые два метода являются противоположностями друг другу, add\_to\_db(car\_id, name, weight, free) и delete\_row(num), где на вход первой подаются номер в БД, имя машины, грузоподъёмность и занятость, а второй – номер в БД. Первый метод добавляет машину в базу данных, а второй- удаляет её оттуда. Далее были написаны оставшиеся два метода. Первый – update\_free(num), где num – номер машины в БД. Данный метод обновляет значение занятости машины в БД при её бронировании. Второй – recovery\_array(). Данный метод собирает всю информацию из базы данных и возвращает её в виде списка из кортежей. Это необходимо в самом начале программы, чтобы восстановить главный список с транспортом из базы данных. Все методы класса можно увидеть на рисунке 6

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 6 – Класс DataBase

Для реализации графического интерфейса была выбрана библиотека tkinter в силу своей достаточно мощной базы и простоты для начинающего. Для красивого интерфейса я также добавил сторонние библиотеки с темами для tkinter. Все элементы интерфейса были размещены на одном окне, что упрощало использование программы

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Рисунок 7 – Главное окошко программы

Во все поля ввода введены ограничения как по типу вводимого (нельзя в цифровые поля вводить буквы), так и по длине и логическому смыслу (нельзя в грузоподъёмность ввести 9 триллионов). В случае возникновения подобных ситуаций выводится диалоговое окно, сообщающее об ошибке, как, например, на рисунке 8.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 8 – Диалоговое окно с ошибкой ввода

Приведём пример диалогового окна, которое показывает какой-либо список автотранспорта. Увидеть его можно на рисунке 9.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 9 – Диалоговое окно, показывающее весь транспорт

Чтобы связать объекты и методы классов с графическим интерфейсом, были написаны функции-посредники в файле main, которые являются лишь оболочкой для вызова метода. В данных функциях помимо самого вызова есть ещё проверки и структурирование данных для верной работы. Но в особом внимании они не нуждаются в силу своей простоты и примитивности.

Приведём пример работы подбора транспорта по грузоподъёмности на рисунке 10. То есть нам необходимо ввести лишь желаемую грузоподъёмность, и программа покажет нам список машин, которые и свободны, и могут везти такой груз.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 10 – Подбор транспорта по грузоподъёмности

Таким образом, удалось создать приложение с обширным функционалом, которое действительно имеет практическое применение. Логику конечного варианта системы можно увидеть на рисунке 11. Там изображена диаграмма классов моего приложения.

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

Рисунок 11 – Диаграмма классов

Особенностью приложения также является использование базы данных. Я использовал базу данных SQLite3. Построение архитектуры базы данных я проводил в специальном прикладном приложении SQLiteStudio, которое помогло не только грамотно и быстро составить схему БД, но и комфортно работать с базой данных в последующем. Детальное описание базы данных вы можете увидеть на рисунке 12.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 12 – Схема базы данных

**Вывод:**

Для реализации программы использовал БД, ООП, графический дизайн. Все это помогает сделать программу удобной, взаимодействие с ней сделать понятным пользователю. Реализовано добавление/удаление транспорта, просмотр всего доступного транспорта, просмотр по грузоподъемности, просмотр свободного, внос заявки на перевоз по габаритам, подбор и бронь, просмотр занятого. Создан интерфейс, данные сохраняются в базу данных.