

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ВЛАДИВОСТОКСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ВВГУ»)
ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И АНАЛИЗА ДАННЫХ
КАФЕДРА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И СИСТЕМ

ОТЧЕТ
ПО ФИНАЛЬНОЙ ЛАБОРАТОРНОЙ РАБОТЕ
по дисциплине
«Информатика и программирование»

Студент
гр. БИН-25-02 _____ А.В. Воронин
Ассистент
преподавателя _____ М.В. Водяницкий

Задание

Разработать консольную систему управления автозаправочной станцией на языке Python.

Система должна имитировать реальные процессы работы заправочной станции и предоставлять оператору возможность управления всеми аспектами работы АЗС.

Основные требования к системе:

1. Система учитывает несколько типов топлива: АИ-92, АИ-95, АИ-98 и ДТ.
2. Реализовать систему подземных цистерн для хранения топлива. Каждая цистерна должна иметь следующие характеристики: тип топлива, максимальный объем, текущий уровень, минимальный допустимый уровень, состояние включена или отключена.
3. Реализовать систему заправочных колонок. Всего должно быть 8 колонок, каждая из которых поддерживает несколько типов топлива. Каждый тип топлива на колонке подключен к конкретной цистерне.
4. Схема подключений должна соответствовать следующим правилам: АИ-92 доступен на колонках 1-6 и подключен к цистерне T1, АИ-95 доступен на колонках 1-4 цистерна T2 и 5-8 цистерна T3, АИ-98 доступен на колонках 3-6 и подключен к цистерне T4, ДТ доступен на колонках 3-8 и подключен к цистерне T5.
5. Реализовать меню управления со следующими функциями: обслуживание клиентов продажа топлива, просмотр состояния цистерн, пополнение запасов топлива, просмотр статистики продаж, просмотр истории операций, перекачка топлива между цистернами, управление состоянием цистерн включение или отключение, аварийный режим работы, снятие аварийного режима.
6. При продаже топлива необходимо выполнять следующие проверки: доступность выбранной колонки, состояние цистерны включена или отключена, достаточность топлива в цистерне, корректность введенных данных.
7. Реализовать автоматическое отключение цистерны при падении уровня топлива ниже минимального порога.
8. Включение цистерны после пополнения должно выполняться только вручную через меню управления.
9. Все данные должны сохраняться в файле формата JSON для сохранения состояния между запусками программы.
10. Реализовать ведение статистики продаж: общий доход, количество обслуженных автомобилей, объемы продаж по типам топлива, доход по каждому виду топлива.

11. Реализовать историю операций с временными метками для отслеживания всех действий в системе.

12. Реализовать аварийный режим, который полностью блокирует работу АЗС и фиксирует аварийное событие в истории.

13. Программа должна быть консольной и работать через текстовое меню с вводом команд с клавиатуры пользователем.

14. Код должен быть структурирован, использовать объектно-ориентированный подход и содержать обработку исключений для корректной работы при возможном некорректном пользовательским вводом.

Содержание

1 Выполнение работы	3
1.1 Структура программы	3
1.2 Класс Tank цистерна	3
1.3 Класс Dispenser колонка	4
1.4 Класс GasStation АЗС	5
1.5 Инициализация и сохранение данных.....	5
1.6 Основное меню программы	6
1.7 Функция обслуживания клиентов	7
1.8 Управление цистернами	9
1.9 Система пополнения и перекачки.....	9
1.10 Статистика и история операций	10
1.11 Аварийный режим	10
1.12 Обработка ошибок и исключений	11
1.13 Тестирование работы программы	11

1 Выполнение работы

1.1 Структура программы

Система управления АЗС разработана на языке Python с использованием объектно-ориентированного подхода. Программа состоит из трех основных классов и главного управляющего цикла. Класс Tank моделирует цистерну для хранения топлива, класс Dispenser представляет заправочную колонку, а класс GasStation является центральным управляющим компонентом всей системы. Архитектура программы позволяет эффективно управлять всеми процессами на автозаправочной станции и соответствует требованиям технического задания.

```

1 import json
2 import os
3 from datetime import datetime
4
5 DATA_FILE = "data.json"
6
7 PRICES = {
8     "АИ-92": 50.0,
9     "АИ-95": 58.3,
10    "АИ-98": 65.0,
11    "ДТ": 55.0
12 }
```

Рисунок 1 – Импорт библиотек и настройка констант

Программа использует стандартные библиотеки Python для работы с файлами, JSON форматом и временными метками. Цены на топливо заданы в словаре PRICES, что позволяет легко изменять их при необходимости, что будет целиком и полностью влиять на экономику внутри симулятора.

1.2 Класс Tank цистерна

Класс Tank является одним из главных классов всей программы и представляет собой модель цистерны для хранения топлива. Каждая цистерна имеет уникальный идентификатор и характеристики, соответствующие техническому заданию. Основные характеристики цистерны включают идентификатор цистерны, тип хранимого топлива, максимальный объем цистерны, текущий объем топлива в цистерне, минимальный допустимый уровень топлива, состояние включена или отключена цистерна и флаг блокировки для аварийного режима, который можно включать или выключать, а также соответствующие функции для управления цистерной и топливом внутри неё. Конструктор класса принимает все необходимые параметры для инициализации цистерны. Параметры enabled и blocked имеют значения по умолчанию, что упрощает создание новых объектов, если их не нужно преждевременно блокировать или выключать.

Ключевые методы класса Tank обеспечивают основные операции с цистерной, что и позволяет управлять заправочной станцией.

```

1 class Tank:
2     def __init__(self, id, fuel_type, max_volume,
3                  current_volume, min_level, enabled=True, blocked=False):
4         self.id = id
5         self.fuel_type = fuel_type
6         self.max_volume = max_volume
7         self.current_volume = current_volume
8         self.min_level = min_level
9         self.enabled = enabled
10        self.blocked = blocked
11    def check_min_level(self):
12        if self.current_volume < self.min_level:
13            self.enabled = False
14    def dispense(self, liters):
15        if self.current_volume >= liters:
16            self.current_volume -= liters
17            self.check_min_level()
18            return True
19        return False
20    def refill(self, liters):
21        if self.current_volume + liters <= self.max_volume:
22            self.current_volume += liters
23            return True
24        return False
25
26

```

Рисунок 2 – Конструктор и основные методы класса Tank

Метод `check_min_level` автоматически отключает цистерну при падении уровня топлива ниже минимального порога. Метод `dispense` отвечает за отпуск топлива с проверкой достаточности, а метод `refill` позволяет пополнять запас топлива с контролем переполнения.

```

1     def transfer_to(self, other, liters):
2         if self.fuel_type != other.fuel_type:
3             return False
4         if self.current_volume < liters:
5             return False
6         if other.current_volume + liters > other.max_volume:
7             return False
8
9         self.current_volume -= liters
10        other.current_volume += liters
11        self.check_min_level()
12        return True
13
14    def to_dict(self):
15        return self.__dict__

```

Рисунок 3 – Методы перекачки и сериализации

Метод `transfer_to` обеспечивает перекачку топлива между цистернами одного типа с выполнением всех необходимых проверок. Метод `to_dict` преобразует объект в словарь для сохранения в JSON формате, что обеспечивает сохранение состояния между запусками программы.

1.3 Класс Dispenser колонка

Класс Dispenser представляет заправочную колонку. Каждая колонка имеет словарь подключений, который определяет, к каким цистернам подключены различные типы топ-

лива на данной колонке. Структура словаря подключений: ключ тип топлива, значение идентификатор цистерны.

```

1 class Dispenser:
2     def __init__(self, id, connections):
3         self.id = id
4         self.connections = connections
5
6     def to_dict(self):
7         return self.__dict__

```

Рисунок 4 – Класс Dispenser

Класс Dispenser имеет простую структуру, но играет важную роль в системе. Словарь connections определяет маршрутизацию топлива от колонки к соответствующим цистернам, что позволяет гибко настраивать схему подключений.

1.4 Класс GasStation АЗС

Класс GasStation является центральным компонентом системы и реализует все функции управления АЗС. Он содержит коллекции цистерн и колонок, а также данные статистики и истории операций.

```

1 class GasStation:
2     def __init__(self):
3         self.tanks = {}
4         self.dispensers = {}
5         self.stats = {
6             "income": 0,
7             "cars": 0,
8             "liters": {},
9             "fuel_income": {}
10        }
11        self.history = []
12        self.emergency_mode = False
13        self.load()

```

Рисунок 5 – Конструктор и инициализация GasStation

При инициализации создаются пустые структуры данных для хранения информации о цистернах, колонках, статистике и истории. Метод load вызывается сразу после создания объекта для загрузки сохраненного состояния из файла.

1.5 Инициализация и сохранение данных

Программа использует файл data.json для хранения состояния АЗС между запусками. Данные сохраняются в структурированном формате JSON, что обеспечивает удобство чтения и редактирования. Метод load класса GasStation загружает данные из файла при инициализации, а метод save сохраняет текущее состояние системы. Этот подход соответствует требованию технического задания о сохранении состояния между запусками программы. Механизм сохранения и загрузки данных реализован с учетом особенностей работы автозаправочной станции. Система должна сохранять не только текущие объе-

мы топлива в цистернах, но и всю историю операций, статистику продаж, а также состояние оборудования включение или отключение цистерн, аварийный режим. Использование формата JSON позволяет легко читать и анализировать сохраненные данные без необходимости запуска самой программы, что полезно для аудита и отладки.

```

1 {
2     "tanks": [
3         {
4             "id": "T1",
5             "fuel_type": "АИ-92",
6             "max_volume": 20000,
7             "current_volume": 15000,
8             "min_level": 1000,
9             "enabled": true,
10            "blocked": false
11        }
12    ],
13    "dispensers": [
14        {
15            "id": 1,
16            "connections": {
17                АИ"-92": "T1",
18                АИ"-95": "T2",
19                АИ"-98": "T4",
20                ДТ"": "T5"
21            }
22        }
23    ],
24    "stats": {
25        "income": 0,
26        "cars": 0,
27        "liters": {
28            АИ"-92": 0,
29            АИ"-95": 0,
30            АИ"-98": 0,
31            ДТ"": 0
32        }
33    },
34    "history": [],
35    "emergency_mode": false
36 }
```

Рисунок 6 – Пример структуры данных JSON

Файл данных содержит всю необходимую информацию для восстановления состояния системы. Структура файла хорошо организована и легко читаема, что упрощает отладку и модификацию данных при необходимости.

1.6 Основное меню программы

Главный цикл программы предоставляет пользователю меню с девятью основными функциями управления АЗС. Меню реализовано с использованием цикла while True, что обеспечивает непрерывную работу программы до явного выхода пользователя. Каждый пункт меню вызывает соответствующий метод класса GasStation, что обеспечивает модульность и удобство поддержки кода.

Архитектура главного меню построена по принципу единой точки входа для всех операций управления станцией. Пользовательский интерфейс представляет собой текстовое меню с последовательной нумерацией опций от 1 до 9, где каждая цифра соответствует

конкретной функции. Нулевой пункт (0) зарезервирован для выхода из программы. Такой подход к организации интерфейса соответствует стандартам консольных приложений и интуитивно понятен пользователям.

```

1 station = GasStation()
2
3 while True:
4     print("\n==== СИСТЕМА УПРАВЛЕНИЯ АЗС ===")
5     print("1) Обслужить клиента")
6     print("2) Состояние цистерн")
7     print("3) Пополнение")
8     print("4) Статистика")
9     print("5) История")
10    print("6) Перекачка")
11    print("7) Управление цистернами")
12    print("8) EMERGENCY")
13    print("9) Снять аварийный режим")
14    print("0) Выход")
15
16    choice = input("> ")

```

Рисунок 7 – Главный цикл программы

Меню организовано интуитивно понятно и предоставляет доступ ко всем необходимым функциям управления АЗС. Пользователь может легко выбирать нужные операции с помощью цифрового ввода.

```

1 if choice == "1":
2     station.serve_client()
3 elif choice == "2":
4     station.show_tanks()
5 elif choice == "3":
6     station.refill()
7 elif choice == "4":
8     station.show_stats()
9 elif choice == "5":
10    station.show_history()
11 elif choice == "6":
12    station.transfer()
13 elif choice == "7":
14    station.manage_tanks()
15 elif choice == "8":
16    station.emergency()
17 elif choice == "9":
18    station.exit_emergency()
19 elif choice == "0":
20     break

```

Рисунок 8 – Обработка выбора меню

Каждый выбор пользователя обрабатывается соответствующей функцией, что обеспечивает четкое разделение ответственности между различными модулями программы. Такой подход упрощает добавление новых функций и модификацию существующих.

1.7 Функция обслуживания клиентов

Функция `serve_client` реализует процесс продажи топлива клиенту. Алгоритм работы включает несколько этапов проверки и подтверждения. Этапы обслуживания клиента включают выбор номера колонки от 1 до 8, выбор типа топлива из доступных на

выбранной колонке, проверку состояния подключенной цистерны, ввод количества литров с проверкой корректности, расчет стоимости на основе фиксированных цен, подтверждение операции пользователем и обновление данных с сохранением в файл.

```

1 def serve_client(self):
2     if self.emergency_mode:
3         print("АЗС в аварийном режиме!")
4         return
5
6     try:
7         col = int(input("Выберите колонку (1-8): "))
8         dispenser = self.dispensers[col]
9     except:
10        print("Ошибка!")
11        return
12
13     fuels = list(dispenser.connections.keys())
14     for i, f in enumerate(fuels, 1):
15         print(f"{i}) {f}")
16
17     try:
18         choice = int(input("Выберите топливо: "))
19         fuel = fuels[choice - 1]
20     except:
21         print("Ошибка!")
22         return

```

Рисунок 9 – Выбор колонки и типа топлива

В начале функции проверяется, не находится ли АЗС в аварийном режиме. Затем пользователь выбирает колонку и тип топлива. Оба выбора защищены обработкой исключений, что предотвращает сбой программы при некорректном вводе.

```

1     tank = self.tanks[dispenser.connections[fuel]]
2
3     if not tank.enabled or tank.blocked:
4         print("Цистерна недоступна!")
5         return
6
7     try:
8         liters = float(input("Введите литры: "))
9     except:
10        print("Ошибка!")
11        return
12
13     if liters <= 0:
14         print("Некорректное количество!")
15         return

```

Рисунок 10 – Проверка доступности и ввод количества

Процесс определения цистерны представляет собой ключевой этап в цепочке обслуживания клиента. Система использует словарь connections объекта Dispenser, который содержит сопоставление типов топлива с идентификаторами цистерн. При запросе конкретного типа топлива программа выполняет поиск в словаре connections по ключу, соответствующему выбранному топливу, и получает идентификатор цистерны. Получение объекта цистерны происходит через обращение к словарю tanks класса GasStation по найденному идентификатору. Этот словарь содержит все цистерны станции, обеспечивая быстрый доступ к нужному объекту. Система гарантирует, что идентификатор, хранящий-

ся в connections, всегда соответствует существующей цистерне, что проверяется на этапе загрузки конфигурации и при любых изменениях схемы подключений.

```

1  if not tank.dispense(liters):
2      print("Недостаточно топлива!")
3      return
4
5      cost = liters * PRICES[fuel]
6      print(f"К оплате: {cost:.2f} ₽")
7      confirm = input("Подтвердить? (y/n): ")
8
9      if confirm.lower() != "y":
10         tank.current_volume += liters
11         return
12
13     self.stats["income"] += cost
14     self.stats["cars"] += 1
15     self.stats["liters"][fuel] += liters
16     self.stats["fuel_income"][fuel] += cost
17
18     self.history.append(f"{datetime.now()} - Продажа {fuel} {liters} л")
19     self.save()
20     print("Успешно!")

```

Рисунок 11 – Расчет и подтверждение операции

Финальная часть функции выполняет расчет стоимости, запрашивает подтверждение у пользователя и обновляет все данные системы. При отказе от операции топливо возвращается в цистерну. При подтверждении обновляются статистика, история и сохраняется новое состояние системы.

В функции реализована обработка всех возможных ошибок: некорректный ввод, недоступность цистерны, недостаток топлива. При отмене операции производится возврат топлива в цистерну. Это обеспечивает надежность и безопасность операций продажи топлива.

1.8 Управление цистернами

Функция show_tanks отображает текущее состояние всех цистерн в удобном табличном формате. Для каждой цистерны выводится следующая информация: идентификатор цистерны, тип топлива, текущий и максимальный объем, статус ВКЛ или ВЫКЛ.

```

1 def show_tanks(self):
2     for t in self.tanks.values():
3         status = "ВКЛ" if t.enabled else "ВЫКЛ"
4         print(f"{t.id} | {t.fuel_type} | {t.current_volume}/{t.max_volume} | {status}")

```

Рисунок 12 – Метод отображения состояния цистерн

Функция manage_tanks позволяет оператору вручную включать или отключать цистерны. Включение возможно только при достаточном уровне топлива, что соответствует требованиям безопасности. Это обеспечивает контроль над работой оборудования и предотвращает попытки использования пустых или неисправных цистерн.

1.9 Система пополнения и перекачки

Функция `refill` имитирует прибытие бензовоза и пополнение запасов топлива. Оператор выбирает цистерну и указывает количество литров для пополнения. Система проверяет, чтобы новый объем не превышал максимальную вместимость цистерны.

```

1 def refill(self):
2     for t in self.tanks.values():
3         print(f"{t.id} - {t.fuel_type}")
4
5     tank_id = input("Выберите цистерну: ")
6     if tank_id not in self.tanks:
7         return
8
9     try:
10         liters = float(input("Введите литры: "))
11     except:
12         return
13
14     if self.tanks[tank_id].refill(liters):
15         self.history.append(f"datetime.now() - Пополнение {tank_id} {liters} л")
16         self.save()
17         print("Пополнение выполнено.")
18     else:
19         print("Переполнение!")

```

Рисунок 13 – Метод пополнения топлива

Функция `transfer` позволяет перекачивать топливо между цистернами одного типа. Это полезно для балансировки запасов или подготовки к обслуживанию. Проверяются следующие условия: цистерны должны содержать один тип топлива, в исходной цистерне должно быть достаточно топлива, целевая цистерна должна иметь достаточную вместимость.

1.10 Статистика и история операций

Система ведет подробную статистику продаж, которая включает общий доход от продаж, количество обслуженных автомобилей, объемы проданного топлива по типам, доход по каждому виду топлива. История операций сохраняет все значимые события с временными метками: продажи топлива клиентам, пополнение запасов, перекачка между цистернами, изменение состояния цистерн, аварийные события.

```

1 def show_stats(self):
2     print(f"Доход: {self.stats['income']:.2f} ₽")
3     print(f"Машин: {self.stats['cars']} ")
4     for fuel in PRICES:
5         print(f"{fuel}: {self.stats['liters'][fuel]} л | {self.stats['fuel_income'][fuel]:.2f} ₽")

```

Рисунок 14 – Метод отображения статистики

История хранится в виде списка строк с датой и описанием события, что позволяет отслеживать все действия в хронологическом порядке.

1.11 Аварийный режим

Аварийный режим реализован для обеспечения безопасности на АЗС. При активации этого режима все цистерны блокируются, продажа топлива прекращается, событие фиксируется в истории и система переходит в заблокированное состояние.

```

1 def emergency(self):
2     confirm = input("Подтвердить аварийный режим? (y/n): ")
3     if confirm.lower() == "y":
4         self.emergency_mode = True
5         for t in self.tanks.values():
6             t.blocked = True
7             t.enabled = False
8         self.history.append(f"{datetime.now()} - АВАРИЯ")
9         self.save()
10        print("АЗС остановлена!")

```

Рисунок 15 – Метод активации аварийного режима

Выход из аварийного режима требует ручного подтверждения оператора. После снятия аварийного режима цистерны остаются отключенными и требуют ручного включения, что соответствует требованиям безопасности. Такая двухэтапная система обеспечивает дополнительный уровень защиты от случайных или ошибочных действий.

1.12 Обработка ошибок и исключений

Во всех функциях ввода реализована обработка исключений для предотвращения сбоев программы при некорректном вводе пользователя. Используются блоки try-except для преобразования строк в числа и проверки диапазонов значений. Основные проверки включают корректность преобразования строк в числовые типы, проверку диапазонов номеров колонок и количества литров, проверку существования выбранных элементов и проверку достаточности ресурсов.

Такой подход обеспечивает устойчивость программы к ошибкам ввода и предотвращает ее аварийное завершение. Пользователь получает понятные сообщения об ошибках и возможность исправить ввод без потери данных или необходимости перезапуска программы.

1.13 Тестирование работы программы

Для проверки корректности работы программы было выполнено тестирование всех функций системы. Тестирование включало проверку продажи топлива с различными комбинациями колонок и типов топлива, проверку автоматического отключения цистерн при низком уровне, тестирование пополнения запасов с проверкой переполнения, проверку перекачки топлива между цистернами, тестирование аварийного режима и восстановления работы, проверку сохранения и загрузки данных из файла.